Evaluation and Analysis of NID Database of Synthetically Generated Data with SQL and NoSQL Database Management Systems

Prepared By: Noshin Islam(1521733042) , Muiz Ahmed Khan (1521282642), SM Asif Bin Eshaq (1521479642)

Submitted to: Dr. Abu Sayed Md. Latiful Hoque

Submission Date: 18/07/19

1. Introduction:

Traditional database systems are based on the relational model for storing data. They were named SQL database because Structured Query Language is used to query them. This is also known as relational database. In the last few years the non-relational databases have dramatically risen which is known as NoSQL database. These databases are different from the traditional SQL databases. Such databases have existed since the late 1960s, but did not obtain the "NoSQL" moniker until a surge of popularity in the early 21st century because of its simplicity and speed.

With the increase of internet and availability of cheap storage, huge amount of structured, semi-structured and unstructured data are generating day by day and stored for various work purpose. These data are referred as “Big Data". To process such huge amount data requires speed, flexible database schemas, and distributed architectures which do not fit into the traditional relational databases. NoSQL database became popular because the can satisfy these requirements. There are several commercial and open-source implementation of NoSQL databases (such as MongoDB, HBase etc)

In general, when the dataset is huge and has not any specific structure and needs a huge amount of storage, choice between SQL and NoSQL databases, NoSQL database engines are a good choice. But it can’t be said that NoSQL databases will perform better than the SQL databases. The following is a brief overview of the differences between SQL and different types of NoSQL database in terms of speed, space measurement.

The focus of our paper is to evaluate a comparison between SQL and NoSQL databases based on Bangladeshi NID card. Our approach will be making a SQL database and analysis its performance and measure its speed. With the same data we will make key-value, document and column oriented database in NoSQL database and do the same work as we worked on the relational database

2. Methodology

In this work, we evaluated and analyzed a synthetic version of our National Identification Card (NID) database with different Database Management Systems (DBMS). We divided our work into different phases such as i) generating NID like synthetic database with random attributes, ii) import the database into different DBMS, iii) Evaluate their performance in terms of their query transaction time, allocated space and other attributes.

2.1 Generate Synthetic Database

This kind of database or data-set is not available in the context of Bangladesh. We needed to generate a synthetic database with random attributes. For our work, we considered a few attributes like a person's NID, name (first-name, middle name, last name), father's name (first-name, middle name, last name), mother's name (first-name, middle name, last name), present address (house number, street, ward number, area, thana (Police Station), city, district, division) for people living in cities, present address (ward number, village, thana (Police Station), district, division) for people living in villages, permanent address (house number, street, ward number, area, thana, city, district, division) for people living in cities, permanent address (ward number, village, thana (Police Station), district, division) for people living in villages, date of birth, profession, mobile, email if any, marital status, gender, place of birth, Passport number if any, TIN number if any, height, weight, emergency contact person (name as above, address as above, mobile and email), income, asset, tax, educational qualification (SSC, HSC, graduation, post graduation, PhD).

2.1.1 Database Design

Because of huge size of the database, design of the database kept simple with only one table named 'Person', containing over 160 million rows and 71 columns (for each attributes mentioned above).

2.1.2 Data Generator Algorithm

First, a list of Bangladeshi people's names were collected from the internet. All the names were fragmented to single names. Then different tags were assigned to each of them based on whether the name is a male or female name (gender), whether it is a Muslim, or other religion's name and whether it is a first, middle or last name (Positioning). Tags were used to generate more realistic names randomly. Total number of 350,000 random unique names (2 and 3 word length) were generated from roughly 800 single names. Their gender, religion were also generated based on their names. Similarly, three other names, mother's name, father's name, emergency contact person's name were generated for each person. Attributes like NID number, Passport number, phone number, Tax Identification Number (TIN) were randomly generated as sequence of characters or numbers. Profession was generated based on a person's age from a specified list in the program with corresponding income range. Random number between the income ranges assigned as their income per month and income tax was generated based on 30% of their income. Permanent addresses and present addresses have many fields such as house number, street number, ward number, thana, district, division. A sample list for each of the fields were given and then randomly picked from them. For educational qualification S.S.C result, school (where a person passed S.S.C from), H.S.C result, college (where a person passed H.S.C from), under graduation result, university, post graduation result, university were generated. Other personal information like marital status, height, weight also generated randomly. After generating unique names with random attributes, same names were duplicated to generate more random persons with different attributes. Not every person has all the attributes such as phone number, email address, educational qualification etc. So, some attributes were left blank for people who simply don't have the attribute.

2.1.3 MySQL: Phpmyadmin

We used Phpmyadmin as our administrative tool to manipulate MySQL database for uploading and manipulating our data. It is a portable web application written in php and much faster than any other mysql tool because the queries can be run in a server. We had to change the php script for updating the default "**memory_limit**", "**upload_max_filesize**", "**max_file_uploads**", "**post_max_size**".

2.1.4 NoSQL: MongoDB

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is a multiuser system that likes to use as much memory as it can, it is much better installed on a server. We installed MongoDB community edition in our 8GB RAM laptop and used the Mongo Shell command prompt to load data in the MongoDB server installed with command query i.e : "**mongoimport –db People –collection Person –type csv –file c:\datasets\5000000_data1.csv –headerline"**

2.1.4 Sample Database

A row from the database is given below where all the column names and their corresponding values are shown: First name: "Rawshan", Middle name: "Hossain", Last name: "Shah", NID: "7033484034", Fathers First Name: "Litu", Fathers Middle Name: "Hossain", Fathers Last Name: "Shah", Mothers First Name: "Nafisa", Mothers Middle Name: "Umme", Mothers Last Name: "Sawdagor", Sex: "male", Religion: "muslim", Marital Status: "Devorced", Height: "5 ft. 11 inches", Weight: "109 lbs", Profession: "Minister", Income:"70000", Income Tax: "21000.0", TIN No: "436833432258", Assets: "1 car(s) 1 flat(s) ", Date of Birth: "4/1/1971", Passport No: "DX6599864", Email Address: "", Mobile No: "01613266412", Prsnt addr house no: "4", Prsnt addr street no: "12", Prsnt addr word no: "32", Prsnt addr village: "", Prsnt addr thana: "Banani", Prsnt addr city: "Dhaka", Prsnt addr district: "Dhaka", Prsnt addr division: "Dhaka", Prmntn addr house no: null, Prmntn addr street no: " ", Prmntn addr word no: "47", Prmntn addr village: "Mistry Para", Prmntn addr thana: "Jaldhaka", Prmntn addr city: " ", Prmntn addr district: "Thakurgaon", Prmntn addr division: "Rangpur", SSC Passed From: "Ranpur Zilla School", SSC Result: "5.0", HSC Passed From:, "St. Joseph College", HSC Result: "3.88", Graduated From: "Rajshahi University", Undergrad Result: "3.35", Post Graduated From: "", Postgrad Result: "", Doctorate From: "", Place of Birth: "Rangpur", EC First Name: "Roman", EC Middle Name: "Hossain", EC Last Name: "Ahmed", EC Prsnt addr house no: " ", EC Prsnt addr street no: " ", EC Prsnt addr word no: "63", EC Prsnt addr village: "Panchanan Barma", EC Prsnt addr thana: "Jaldhaka", EC Prsnt addr city: " ", EC Prsnt addr division: "Rangpur", EC Prmntn addr house no: " ", EC

Prmntn addr street no: " ", EC Prmntn addr word no: "38", EC Prmntn addr village: "Trishal", EC Prmntn addr thana: "Gaforgaon", EC Prmntn addr city: " ", EC Prmntn addr district: "Netrokona", EC Prmntn addr division: "Mymensingh", EC Mobile No: "01559778909", EC Email Address: "roman.ahmed@gmail.com"

3 Result & Analysis

Initially we tested with 500k data, measured their performance with size, query execution time. We took a few sample queries like single attribute query, multiple attribute queries, ranging queries. Then, we kept adding data(ranging between 500k to 10M) to the database and measured performance every time. We used Xampp for MySQL and Mongo DB Shell command for Mongo DB to visualize, make queries and measure their performance.

3.1 Sample Queries

3.1.1 MySQL

i) SELECT * from ‘person’ where marital status = ‘single’

ii) SELECT * FROM ‘person’ WHERE religion = ‘muslim’ and sex = ’male’ and profession = ‘Student’

iii) SELECT * FROM ‘person’ WHERE religion = ‘muslim’ and sex = ‘male’

and profession = ‘Student’ and ssc result = 5 and hsc result = 5

and prsnt addr city = \Dhaka"

iv) SELECT * FROM ‘person’ WHERE income < 15000

v) SELECT * FROM ‘person’ WHERE income = 15000

vi) SELECT * FROM nid_data_1 WHERE religion = 'muslim' and sex = 'male' and profession = 'Student' and ssc_result = 5 and hsc_result = 5 and prsnt_addr_city = 'Dhaka' AND income < 15000

3.1.2 Mongo DB

i) db.Person.find(“Marital Status”: “Single”).explain(“executionStats”)

ii) db.Person.find(Religion: “muslim", Sex: “male", Profession: “Student”)

.explain(“executionStats”)

iii) db.Person.find(Religion: “muslim", Sex: “male", Profession: “Student”, Prsnt_addr_city: “Dhaka”, HSC_Result: 5, SSC_Result: 5) .explain(“executionStats”)

iv) db.Person.find(“Income": {$lt: 15000}).explain(“executionStats”)

v) db.Person.find(“Income": 15000).explain(“executionStats”)

vi) db.Person.find(Religion: “muslim", Sex: “male", Profession: “Student”, Prsnt_addr_city: “Dhaka”, HSC_Result: 5, SSC_Result: 5, “Income": { $lt: 15000}).explain(“executionStats”)

3.2 Performance

3.2.1 MySQL (SQL)

DATA	Estimated Import Time	Database Size	Query 1	Query 2	Query 3	Query 4	Query 5	Query 6
1. 500k	59.21s	274 MB	0.0059s (1,24,646 resutls)	0.0080s(1,16,882 results)	0.5725s (156 results)	0.0061s (2,03,918 results)	3.426s (13 results)	0.5760s (156 results)
2. 1M	54.78s	550 MB	0.00325s (2,49,494 results)	0.0070s(2,33,005 results)	0.6300s (304 results)	0.0055s (4,07,755 results)	6.6136s (23 results)	0.5609s (304 results)
3. 2M	54.97s	995.1MB	0.0089s (4,99,110 results)	0.0072s (4,65,307 results)	0.7700s (637 results)	0.0034s (8,14,123 results)	9.1738s (38 results)	0.8357s (637 results)
4. 3M	128s	1.5 GB	0.0059s (7,49,958 results)	0.0072s (6,97,703 results)	0.78s (930 results)	0.0057s (1221461 results)	8.6255s (64 results)	0.7696s (930 total)
5. 4M	107s	2.04GB	0.0058s (999958 results)	0.0089s (931528 results)	0.74745s (1247 results)	0.0061s (1628734 results)	8.4236s (86 results)	0.73s (1247 results)
6. 5M	180s	2.46GB	0.0075s (1249324 results)	0.0080s (1164487 total)	0.7622s (1584 results)	0.0065s (2935556 results)	8.5005s (102 results)	0.73s (1584 results)
7. 7M	342s	3.75GB	0.0059s (17483738 results)	0.0073S (1629843 results)	0.7684s (2189 results)	0.00605s (284915 results)	8.9866s (148 results)	0.7811s (2189 results)
8. 10M	572s	5.04GB	0.0054s (2497398 results)	0.0075s (2328535 results)	0.8460s (3088 results)	0.0074s (40071193 results)	8.6654s (216 results)	0.7426s (3088 results)
9. 13M	580s	6.65GB	0.0060s (3246587 results)	0.0068s (3026541 results)	0.6682s (4044 results)	0.6682s (5291691 results)	7.0061s (270 results)	8.707s (4078 results)

10. 15M	305.7s	8.16GB	0.0068s (3749067 5 results)	0.0084s(3490675 results)	0.7791s (4621 results)	0.0063s (9104648 results)	8.7047s (305 results)	0.7623s (4621 results)
11. 18M	500.2s	9.3 GB	0.0041s (4564594 results)	0.0075s (413141 3 results)	0.5649s (5487 results)	0.0064s (7437324 results)	6.4594s (377 results)	0.6277s (5487 results)
12. 21M	528.79 63s	11.17GB	0.0110s (5258939 results)	0.0084s (465468 5 results)	5.3110s (6177 results)	0.0065s (8564768 results)	9.5775s (438 results)	0.4833s (6177 results)
13. 24M	768.33 6s	12.78GB	0.0413s (6009411 results)	0.0190s (535196 8 results)	2.5117s (7071 results)	0.0220s (9784794 results)	21.4313s (494 results)	1.7190s (7071 results)
14. 27M	1143.2 487s	14.6GB	0.0222s (6740241 results)	0.0074 (593409 5 results)	0.5775s (7813 results)	0.0768s(1 0974393 results)	8.4687s (564 results)	0.6657s (7813 results)
15. 30M	1309.3s	16.3GB	0.0069s (7696383 results)	0.0076s (665567 8 results)	0.5111s (8777 results)	0.0073s (1252894 7 results)	25.2644s (653 results)	8.5145 s (8777 results)
16. 33M	570.3s	18.14GB	0.0069s (8444834 results)	0.0108s (735407 1 results)	0.5348s (9790 results)	0.0060s (1374901 7 results)	6.6264s (722 results)	8.7498s (9790 results)
17. 40M	476.15 12s	24.6GB	0.0073s (9195060 results)	0.0065s (805139 9 results)	0.5073s (10740 results)	0.0062s (1496766 9 results)	7.4211s (783 results)	0.5855s(10 740 results)
18. 50M	2055.1 2s	34.7GB	0.0232s (1231891 0 results)	0.0409s (109623 87 results)	0.4674s (14645 results)	0.0073s (2005281 0 results)	10.4563s (200528 10 results)	6.7309s (1010 results)

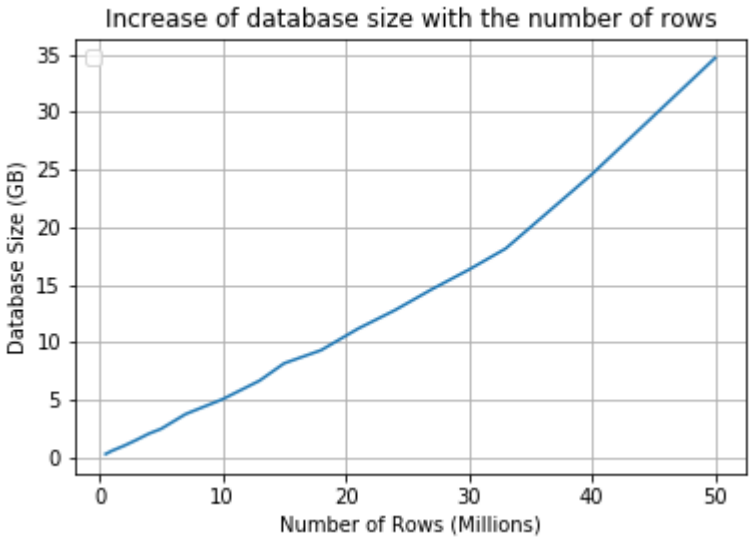

Figure 1 Increase of database size with increasing number of data

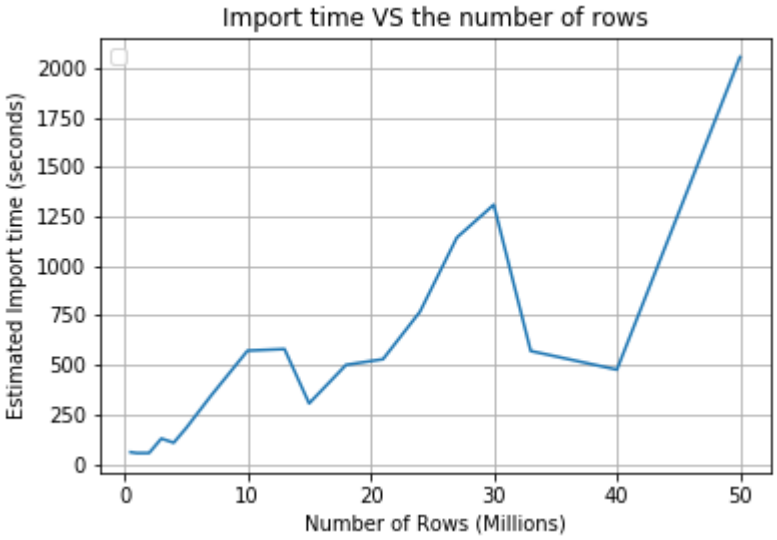

Figure 2 Data Import time vs Number of rows(data) uploaded

Figure 3 Query Execution time with increasing number of rows

3.2.1 MongoDB (Mongo Shell Command Prompt)

MongoDB works best if we keep individual documents to a few kilobytes in size, treating them more like rows in a wide SQL table. Large documents will cause several performance problems which happened in our case because our database structure is rigid and well structured.

DATA	Estimated Import Time	Disk Space use	Query 1	Query 2	Query 3	Query 4	Query 5	Query 6
1. 5o0k	1.30min	200 MB	1.06s, (124646 results)	1.22s (116882 results)	1.3s, (1384 results)	0.926 s, (203918 results)	0.856ms, (13 results)	1.122s, (1384 results)
2. 1M	1.45min	1GB	1.295s (249494 results)	1.211s (233005 results)	2.25s (304 results)	1.23s (407755 results)	1.211s (23 results)	2.254s (304 results)
3. 2M	1.9min	1.8 GB	11.8s(499 110 results)	26.82s (465307 results)	18.36s (637 results)	21.70s(8141 23 results)	28.651s (38 results)	20.8s (637 results)
4. 3M	1.5min	2.6GB	43.8s (749958 results)	45.5s (697703 results)	51.95s (930 results)	54.57s (1221455 results)	49.7s(64 results)	44.13s (930 results)
5. 4M	2.1min	3.4GB	978s (999958 results)	58.1s (931528 results)	78.9s (1247 results)	74.94s (1628734 results	71.8s (86 results)	53.58s (1247 results)
6. 5M	1.7min	4.3GB	640.9s (1249324 results)	163.71s (116448 7 results)	87.59s (1584 results)	108.78s (2035555 results)	91.95s (102 results)	107.21s (1584 results)
7. 7M	1.5min	6.0GB	340.57s (1748373 8 results)	151.54s (162984 3 results)	142.82s (2189 results)	115.6s (284915 results)	115.62s (148 results)	119.8s (2189 results)
8. 10M	3.6min	8.6GB	202.65s (2497398 results)	193.25s (232853 5 results)	194.9s(3088 results)	153.43s (40071193 results)	187.21s (216 results)	194.5s (3088 results)
9. 13M	4.2min	11.2GB	515.03s (3246587 results)	208.04s (302654 1 results)	219.4s (4044 results)	361.4s (5291691 results)	280.4s (270 results)	249.5s (4078 results)
10. 15M	3.9min	12.9GB	239.19s (3749067 5 results)	400.84s (349067 5 results)	313.69s (4621 results)	244.7s (9104648 results)	230.26 (305 results)	247.22s (4621 results)
11. 18M	4.5min	15.5GB	449.2s (4564594 results)	338.49s (413141 3 results)	293.5s (5487 results)	363.3s (7437324 results)	312.8s (377 results)	394.5s (5487 results)
12. 21M	3.6min	18.1GB/29.1 GB uncompressed	348.531s (5258939 results)	350.3s (465468 5 results)	316.5s (6177 results)	394.42s (8564768results)	452.196s (438 results)	403.89s (6177 results)
13. 24M	4.9min	20.6GB/46.6GB	589.5s (6009411 results)	577.5s (535196 8 results)	484.6s (7071 results)	456.5s (9784794 results)	445.31s (494 results)	1131.5s (7071 results)

14. 27M	4.5min	23.3GB/52.6GB	551.5s (6740241 results)	520.5s (5934095 results)	516.58s (7813 results)	601.9s (10974393 results)	581.5s (564 results)	538.36s (7813 results)
15. 30M	4.8min	25.9GB/58.6GB	602.9s (7696383 results)	587.4s (6655678 results)	562.06s (8777 results)	534.6s (12528947 results)	514.23s (653 results)	680.9s s (8777 results)
16. 33M	4.9min	28.9GB/62.7GB	672.6s (8444834 results)	712.5s (7354071 results)	643.6s (9790 results)	617.5s (13749017 results)	801.7s (722 results)	630.3s (9790 results)
17.40M	6.2min	34.5GB/77.9GB	735.8s (9195060 results)	815.25s (8051399 results)	780,2s (10740 results)	667.5s (14967669 results)	672.2s (783 results)	743.7s (10740 results)
18. 50M	10.2min	43.1GB/97.4GB	1000.8s (12318910 results)	987.9s (10962387 results)	1018.5s (14645 results)	987.6s (20052810 results)	1021.76s (20052810 results)	1004.1s (1010 results)

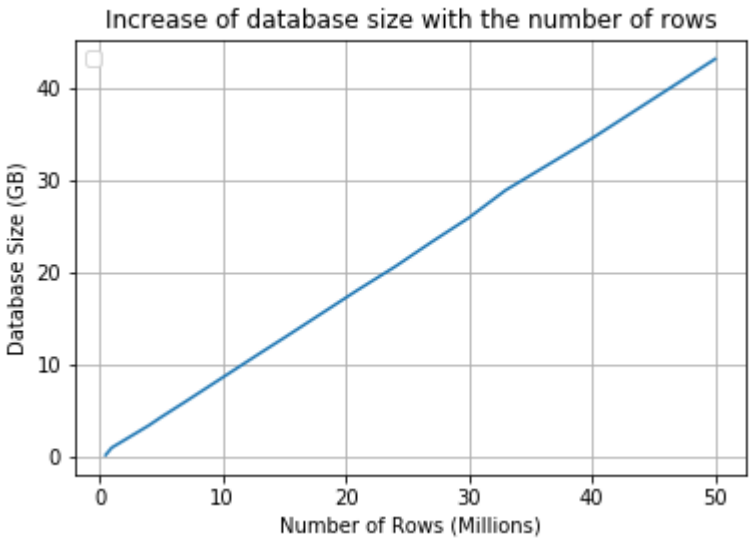

Figure 4 Increase of database size with number of rows

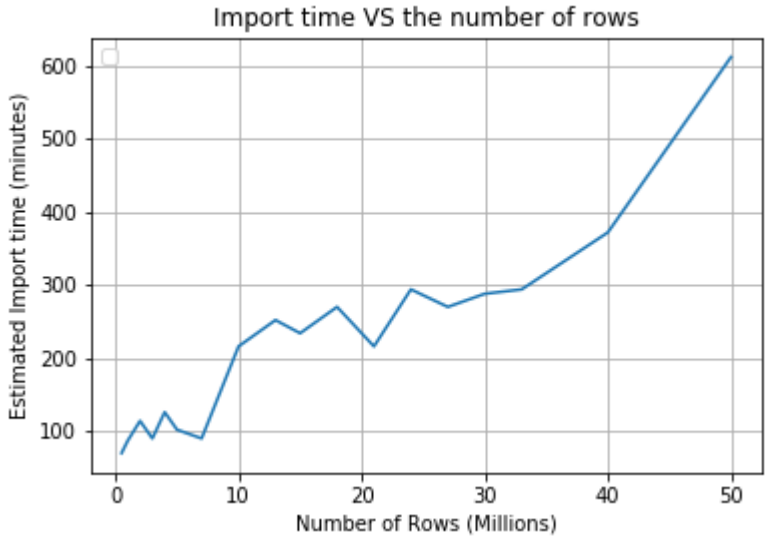

Figure 5 Import time vs the number of rows

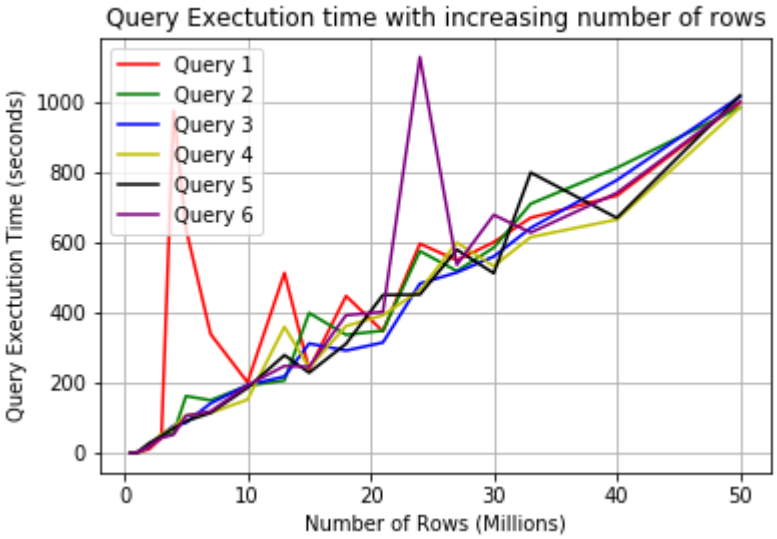

Figure 6 Query Execution time with increasing number of rows

3.3 Discussion

NoSQL is a good choice for those cases experiencing rapid growth with no clear schema definitions. NoSQL offers much more flexibility than a relational database and is a solid option for companies who must analyze large quantities of data or whose data structures they manage are variable. But in our case, a NID database management system will have a predefined structure where a large data will be increased over-time within this rigid structure. SQL is thus a good choice for any cases that will benefit from a predefined structure and set schemas, particularly if they require multi-row transactions.

MongoDB's WiredTiger storage engine compresses data and indexes by default, so the database storage size on disk (which includes both collection & index data) is typically smaller than the sum of the uncompressed document sizes and index sizes reported in collection stats.

The ratio of storage to uncompressed data size will vary depending on factors such as how compressible your data is, the number and type of indexes you create, whether you have deleted a significant number of documents (creating free space which is available for reuse), and any configuration changes from the default server or collection options.

In our study, we have a total of about 97.4 GB of data & indexes in this database after uploading 50million data which uses 43.1GBB of disk storage(compressed), whereas using phpmyadmin mysql database of NID data has a size of 34.7GB with 50 million data.

In our query execution time analysis, we found that MySQL worked better for our case with a large database with structured schema. The displayed query execution time in phpmyadmin is how long the query took to run on the server - it's accurate and comes from the MySQL engine itself. Unfortunately, the results have to then be sent over the web to your browser to be displayed, which takes a lot longer. We recorded the actual execution time, database size and import time for each data increase scenario.

In short, if we were to have a unstable schema which changes overtime, MongoDB would have performed better, but as we have a fixed schema of data generated as csv files, MySQL works better in Query Performance and occupies less storage than MongoDB

4 Conclusion

MySQL is highly organized for its flexibility, high performance, reliable data protection, and ease of managing data. Proper data indexing can resolve your issue with performance, facilitate interaction and ensure robustness. NID Database system need a well structered pre defined schema design and complex joint operations for data manipulation, which is why MySQL works better in performance than MongoDB.