Performance Analysis of Modern Databases

Simmi Bagga[1], Dr. Anil Sharma[2]

[1]Research Scholar, Lovely Professional University, Phagwara, (Punjab), India

[2]Professor, Lovely Professional University, (School of Computer Application) Phagwara, (Punjab), India

[1]*simmibagga12@gmail.com*, [2]anil.19656@lpu.co.in

Abstract

The use of data analysis in decision and strategy making has recently become an important aspect of the business world. A lot of sources are configured to generate this enormous quantity of data and store it into local and remote databases. Due to the variety and abundance in the sources of this data, the nature of this data is very heterogeneous. The traditional Relational Database approach was designed to work on independent mainframes with homogeneity in data but nowadays the aim is to get data distributed through the nodes. This wide variety of scalable data is no longer manageable by relational approach completely; hence the need of unstructured database arises. The performance of unstructured database proved to be a major achievement apart from the liberty in the data structures. In this research paper, different types of unstructured database are studied. The features exhibited by each of them are listed. The data models used for various data stores for a single application are discussed using a case study of the supply chain. Finally similar queries are performed on each of these databases to compare their performance.

Keywords: *Data Modeling, Querying, Document datastore, Graph datastore, Relational Database, NoSQL*

Declarations

Funding: Not Applicable
Conflicts of interest/Competing interests: Not Applicable
Availability of data and material: Not Applicable
Code availability: Not Applicable
Author’s contributions: Not Applicable

1. Introduction

Data is regarded as the foundation of all the megatrends that are happening. With the advent of rising database applications in the previous decade, many tech giants started developing a database for their extending needs to handle big data. They built an agile, scalable, and flexible database that could adapt to changing requirements of the applications [1]. These kinds of requirements were not fulfilled by the relational database because of their rigid schema and inefficient scalability. Hence, NoSQL came into the market and was rapidly deployed globally. NoSQL database proved to be very efficient in storing large volumes of data with little or no structure. Also, cloud computing platforms made extensive use of MySQL’s schema less model [1] [2].
A sudden rise in the popularity of NoSQL was a consequence of the features of the NoSQL database. Rather than following ACID (Atomicity, Consistency, Isolation, and Durability) properties, NoSQL was meant to give BASE (Basically Available, Soft state, and Eventually Consistent) features to users [1]. This model proved to be useful because the information collected through the web and other sources could not be forced into structured tables. NoSQL database allowed users to enter data freely while supporting fast

querying mechanisms. However, these databases were application-specific and couldn't be generic enough to meet all user demands at once [2] [3]. Based on their data storage, the NoSQL database is divided into four categories: Key-Value datastores, Document-based datastores, Column based datastores, and Graph Oriented datastores. Key-Value Datastores store the data in the form of key-value pairs where each key identifies a unique record. It is similar to the hash table which provides quick access to the data. Document-based Datastores store data in the form of documents and each document acts as a record. Different documents may have different format i.e. they may contain different numbers of fields. Column Oriented Datastore stores data in tables by columns rather than rows. It is used where random read and write is frequently required. Graph Oriented Datastore is suitable where data contains highly interconnected relationships. Data is stored in the form of a graph which contains nodes and edges.

2. Data Modeling and Querying

Data Modeling is the method of constructing a data model that is a conceptual depiction of data objects, attributes, and relationships between various data entities. A data model is a way to represent what data is required to be stored and how data should be organized in the database for its effective usage. It is a bridge between data in the real world and the physical representation of data in the database. This is because data models are easily understood by layman because of their relevance in physical world [5]. There are mainly three types of data models: Physical Model, Logical Model, and Conceptual Models. The physical model describes the actual implementation of the internal schema of the database. It defines how the data should be structured in a specific database. The logical model is the bridge between database design and user requirements. It tells about the types of entities with their data attributes and the relationships with their properties. The conceptual model expresses what data should comprise the system. The purpose of this model is to define the scope of data and business rules that validate data in the system [1] [7].

Querying means to get useful information from the database. The whole database is not required at any particular time, only a portion of data fulfills the requirement of user. A database has no use if particular information cannot be retrieved from database efficiently. Querying is a method of fetching useful information from database efficiently. The performance of querying is measured in terms of time, space and accuracy of result. Queries can be designed to perform various tasks like selecting data from database by applying some filters, aggregating data, update or deleting data etc.
As it is clear from the above discussion, the conceptual model is designed after which the construction of logical and physical models takes place. In this research, physical modeling will be discussed in detail. Before going further, it should be clear why the physical models differ in various databases. For different types of databases, entities take different forms based on the data storage mechanisms of the database being talked about. In a relational database, tables are the most highlighted objects, in which data of similar kind and rigid structure can be stored in the form of rows/records. Modeling in RDBMS will define how the fields of each table are related to the fields of other table. Similarly in a Document Oriented database like MongoDB, the primary entity is a collection in which data can be stored in the form of documents. Collections can have dependencies which need to be included in the data model. A graph-based database however has a different way of managing these dependencies through relationships between nodes. Unlike, other datastore where the relationships come into picture while querying using joins, etc., in the graph-oriented database, relationships form an important part of the data structure and are defined as data entities rather than being mere queries. This type of representation makes it very clear to understand and derive conclusions from data and also to debug the model. In the coming section, there is a detailed discussion about the modeling and querying of data in each database along with a case study that implements the same dataset in three databases: MySQL, MongoDB, and Neo4j.

3.0. Case Study

For this research, a supply chain dataset has been taken from the source [17]. This dataset provides enough resources to have a full querying experience of the involved databases. Each record has details about shipment id, origin state-city, destination state-city, NAICS number, SCTG number, mode of transport, quarter number, shipment value, shipment weight, shipment distance routed and GC (great circle), temperature control, export Boolean and country, hazardous material and weight factor.
Shipment id serves as the primary key for this table. The origin and destination areas are each represented by a set of three fields: the state id, the city id, and a string combination of these two. This string can be matched to the complete address of the industry in the “cityarea” table. Here, NAICS number stores the id for North American Industry Classification System and tells what kind of industries are involved in the shipment. SCTG stores the id for Standard Classification for Transported Goods and tells what kinds of goods were transported in the shipment. The mode field contains id for the mode of transport used for the transportation. All this information is in numeric form and there are separate files that map the numbers for NAICS, SCTG, modes of transport and state-city codes to their respective descriptions. Thus the text descriptions can be retrieved from these files by using joins operations. This data can be incorporated into a single CSV (comma separated variables) file so that there is no need of these joins, but on the cost of size and redundancy of the database; hence this practice is depreciated. Thus, the fields’ NAICS, SCTG, modes of transport and state-city codes will act as foreign keys for shipment table and primary keys for their respective tables.

3.1 Relational Data Model

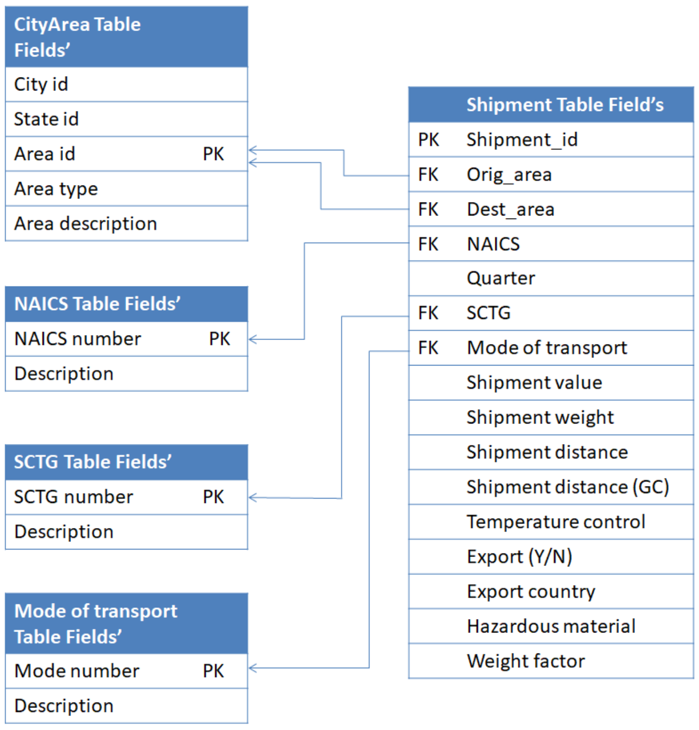
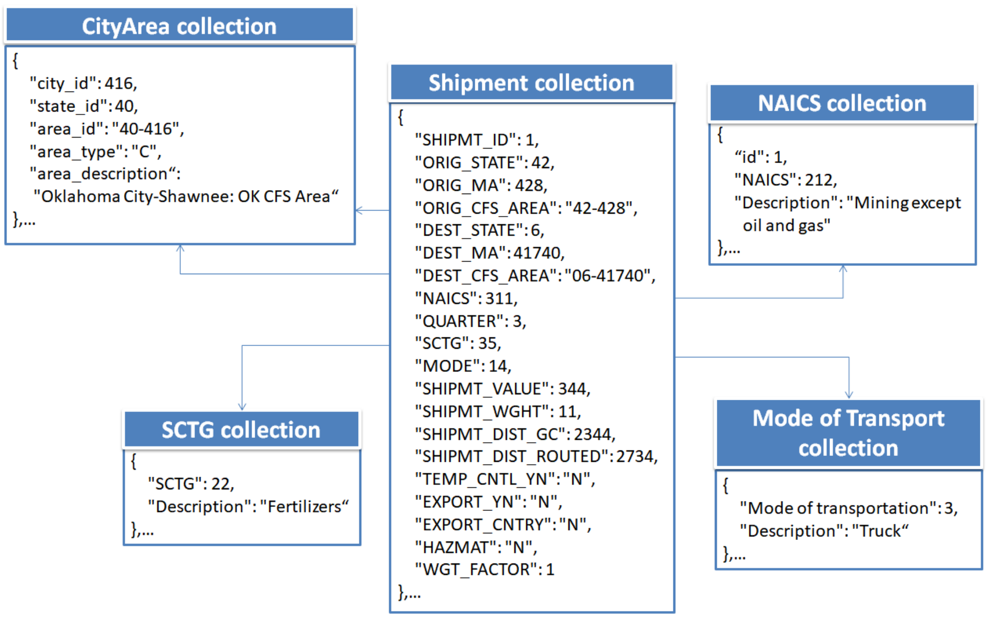


Fig.1 Data Modeling in Relational Databases

As mentioned in earlier section 2.0, before proceeding with queries, a competent data model is necessary for any system’s effective and efficient working. Starting with relational database like MySQL, ER model representation of the system will be an apt model. Not only it ensures that there is no redundancy, and its associated disadvantages, but also it will provide consistency of data while performing any update. Following is the relational model that can be used for implementing this data in MySQL.

3.2 Document Oriented Data Model

A document database stores unstructured data implying that there need not be any fixed schema of the data. JSON and XML format of data allows this flexibility inside the documents of this database where we can insert arrays and even nested documents inside the documents. Hence, this opens the possibilities for many new kinds of data models out of which the best has to be chosen. This choice can vary from user to user because there can be wide variety of application requirements. For instance, there can be a data model where all the NAICS will be stored in the main collection each having a shipments field. The shipments field further gets an array of documents pertaining to that NAICS. All the information about the shipments will be stored in the array of documents as new array entries.



```
{
  "city_id": 416,
  "state_id": 40,
  "area_id": "40-416",
  "area_type": "C",
  "area_description“:
   "Oklahoma City-Shawnee: OK CFS Area“
},...
```

```
{
  "SHIPMT_ID": 1,
  "ORIG_STATE": 42,
  "ORIG_MA": 428,
  "ORIG_CFS_AREA": "42-428",
  "DEST_STATE": 6,
  "DEST_MA": 41740,
  "DEST_CFS_AREA": "06-41740",
  "NAICS": 311,
  "QUARTER": 3,
  "SCTG": 35,
  "MODE": 14,
  "SHIPMT_VALUE": 344,
  "SHIPMT_WGHT": 11,
  "SHIPMT_DIST_GC": 2344,
  "SHIPMT_DIST_ROUTED": 2734,
  "TEMP_CNTL_YN": "N",
  "EXPORT_YN": "N",
  "EXPORT_CNTRY": "N",
  "HAZMAT": "N",
  "WGT_FACTOR": 1
},...
```

```
{
  “id": 1,
  "NAICS": 212,
  "Description": "Mining except
   oil and gas"
},...
```

```
{
  "SCTG": 22,
  "Description": "Fertilizers“
},...
```

```
{
  "Mode of transportation": 3,
  "Description": "Truck“
},...
```

Fig.2 Data Modeling in Document Oriented Database

This data model is a poor choice because whenever there is a shipment from one NAICS to another, there are chances of duplicate array entries in both the documents. Modeling of this type is usually promoted when the array contains a small number of documents. Querying in this model will be more time consuming because array operations will be performed for each document and then combined for the overall document. Similarly, the choice of any other entity like “SCTG”, “mode of transport” or “cityarea” as the major collection and others included inside these documents as arrays of documents will be highly depreciated. So, a model similar to the relational model must be used. Shipments collection is created that stores the details of each shipment as a document inside the collection. Various other entities like NAICS, SCTG, mode of transport and locations are used as fields in the documents of these collections. Also, other information like quarter number, shipment value, shipment weight, shipment distance routed and GC (great circle), temperature control, export Boolean and country, hazardous material and weight factor are included as other fields in the documents of Shipments collection. NAICS, SCTG, mode of transport and location description can be derived from the other collections that map their integer value to string descriptions. Following is a data model for the document-oriented database that can be used in applications such as MongoDB.

3.3 Graph data model

Graph oriented database allow extensive use of the relations between the data. They are designed to store entities in the form of nodes and relationships such that a given entity set has a unique label. These nodes are related to each other by relationships of required labels. Both nodes and relationships can have a list of attributes associated to them that can be used to describe them. This network structure can be very efficiently used to represent the real world data which itself has many kinds of relations among the entities. Because of the graphical storage, very fast traversal from one node to another is possible whenever there is a relationship between the nodes. This database not only ensures the storage of unstructured data in which entities of a given entity set can have varied number and kinds of attributes but also provides the most useful ACID properties that were present in relational database. Moreover, the use of cypher query language makes it enormously easy to perform the CRUD operations in this database. Ad-hoc relationships can be quickly created and destroyed in this format.
In this research, the database used is of supply chain which itself is a network of transport of goods among various cities. Hence, the best database that can be used to model this is a graph oriented database. Here, all the locations are implemented as nodes and each shipment establishes a new relationship between two nodes. The data associated to the location in the “cityarea” table is stored as properties of the nodes and the details of the shipment are stored as the properties of each shipment relation. This model is represented below:

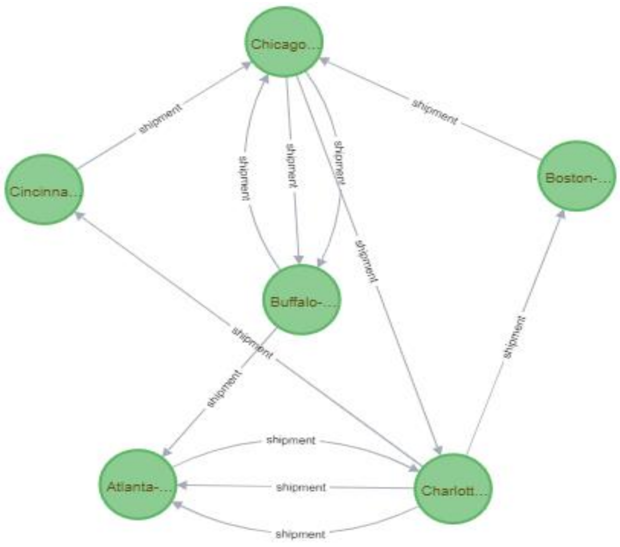


Fig.3 Data Modeling in Graph Oriented Database

3.4 Querying

The further analysis of the performance of these databases can be carried out by querying these. Hence, a set of query questions were prepared and solved for each database. Here, MySQL, MongoDB and Neo4j were used as relational, document oriented and graph oriented database respectively. Data was loaded into each of these database as per the suitable data models mentioned in the previous section. MySQL and Neo4j import data from CSV files while for MongoDB, dataset was transformed to JSON format. After importing data, the queries were executed one after another in the same sequence for each database and the execution time of each query was noted for comparison purposes. The queries are divided into categories based on which of the CRUD action do they perform. The following subsections illustrate each of these categories.

3.4.1 CREATE Operations

Database was created by importing the data from their respective files. In MySQL, first the schema is created and then data is loaded into that schema. On the other hand in MongoDB, data can be directly brought in from a json file using mongoimport application. In Neo4j, first the Location nodes are created and then shipments relation is established among them. The following table enlists the commands used with database name.

DataBase	Query	Exection Time
MySQL	create table supply_chain(SHIPMT_ID int, ORIG_STATE int, ORIG_MA int, ORIG_CFS_AREA varchar(15), DEST_STATE int, DEST_MA int, DEST_CFS_AREA varchar(15), NAICS int, QUARTER int, SCTG int, MODE int, SHIPMT_VALUE float(5), SHIPMT_WGHT float(5), SHIPMT_DIST_GC int, SHIPMT_DIST_ROUTED int, TEMP_CNTL_YN char, EXPORT_YN char, EXPORT_CNTRY char, HAZMAT char, WGT_FACTOR float(5)); load data local infile "export.csv" into table newsupply fields terminated by ',' lines terminated by '\n';	3077 records: 1.014 50000 records: 4.912 1000000 records: 55.605
MongoDB	mongoimport export.json -d research -c supply_chain --jsonArray –drop	3077 records: 0.557 50000 records: 2.516 1000000 records: 17.183
Neo4j	load csv with headers from "file:///cityarea.csv" as row merge (l:Location{city:row.city_id,state: row.state_id,area: row.area_id,type: row.area_type, name: row.area_name}) load csv with headers from "file:///export.csv" as row match (s:Location{area:row.ORIG_CFS_AREA}) match (d:Location{area:row.DEST_CFS_AREA}) merge (s)-[: shipment{id: row.SHIPMT_ID, naics: row.NAICS, quarter: row.QUARTER, sctg: row.SCTG, mode: row.MODE, value: row.SHIPMT_VALUE, weight: row.SHIPMT_WGHT, distgc: row.SHIPMT_DIST_GC, distrouted: row.SHIPMT_DIST_ROUTED, tempcontrol: row.TEMP_CNTL_YN, exportyn: row.EXPORT_YN, exportcountry: row.EXPORT_CNTRY, hazmat: row.HAZMAT, weightfactor: row.WGT_FACTOR }] -> (d);	3077 records: 2.843 50000 records: 36.588

Table 1: Create and Load Operation

3.4.2 READ Operations

Read Operations are one of the most used operations in runtime of most applications. They are used to fetch data from the database based on certain conditions. Following read queries were performed. These queries are listed in the increasing order of their complexity. Also, their execution time only includes the fetching time and not the displaying time for better results.

Q1: **Show all the shipments (3077 entries)**

DataBase	Query	Execution Time
MySQL	Select * from supply_chain	0.077
MongoDB	db.shipments.find().explain("executionStats");	0.002
Neo4j	match (n)-[r]-(m) return (r);	0.025

Table 2: Show all the Shipments

Q2: **Find all shipments done from state code=1, to state code=1 in quarter 1**

DataBase	Query	Execution Time
MySQL	select * from supply_chain where orig_state = 1 and dest_state = 1 and quarter=1;	0.005
MongoDB	db.shipments.find({"ORIG_STATE":1,"DEST_STATE":1,"QUARTER":1}).explain("executionStats");	0.002
Neo4j	match (n:Location{state:"1"})-[r{quarter:"1"}]-(m:Location{state:"1"}) return (r)	0.005

Table 3: Find all shipments done from state with code=1, to state with code=1 in quarter 1

Q3: **States having total shipments of naics="4248" in descending order**

DataBase	Query	Execution Time
MySQL	select orig_state,sum(shipmt_wt) as total_shipment from supply_chain where naics=4248 group by orig_state order by total_shipment desc;	0.009
MongoDB	db.shipments.aggregate([{$match: {NAICS: 4248}},{$group: {_id: "$ORIG_STATE", total_weight: {$sum: "$SHIPMT_WGHT"}}}]);	0.002
Neo4j	match (n)-[r:shipment{naics: "4248"}]->(m) return n.state, sum(r.weight) as total_weight order by total_weight desc;	0.017

Table 4: States having total shipments of naics="4248" in descending order

Q4: **Find all shipments done from state with code=1**

DataBase	Query	Execution Time
MySQL	select * from supply_chain where orig_state = 1;	0.096
MongoDB	db.shipments.find({"ORIG_STATE":1}).explain("executionStats");	0.004
Neo4j	match (n{state:"1"})-[r:shipment]-(m) return (r);	0.024

Table 5: Find all shipments done from state with code=1

Q5: **Count Shipments of value more than 50000$**

DataBase	Query	Execution Time
MySQL	select * from supply_chain where shipmt_val > 50000;	0.067
MongoDB	db.shipments.find({SHIPMT_VALUE: {$gt : 50000}}).explain("executionStats");	0.018
Neo4j	match (n:Location)-[r]-(m:Location) where toInteger(r.value) > 50000 return (r)	0.069

Table 6: Count Shipments of value more than 50000$

Q6: **Top 10 industrial areas with maximum valued exports**

DataBase	Query	Execution Time
MySQL	select cityarea.area_desc,naics.description,sum(shipmt_val) as total_shipment from supply_chain inner join cityarea on cityarea.area_id=supply_chain.orig_area inner join naics on naics.naics_no=supply_chain.naics where exportyn='y' group by orig_area,naics order by total_shipment desc limit 10;	0.214
MongoDB	db.shipments.aggregate([{$match: {"EXPORT_YN":"Y"}},{$group: {_id: {"NAICS": "$NAICS", "ORIG_CFS_AREA":"$ORIG_CFS_AREA"}, total_value: {$sum: "$SHIPMT_VALUE"}}},{$sort:{total_value: -1}},{$limit: 10},{$lookup: { "from": "cityarea", "localField": "_id.ORIG_CFS_AREA", "foreignField": "area_id", "as": "area"}},{$lookup: { "from": "naics", "localField": "_id.NAICS", "foreignField": "NAICS", "as": "naics_desc"}},{$project:{ "AREA": "$area.area_description", "NAICS": "$naics_desc.Description", total_value: 1, _id: 0 }}]);	0.006
Neo4j	match (n:Location)<-[r:shipment{exportyn: "Y"}]-(m:Location) return n.name, r.naics, sum(r.value) as total_value order by total_value desc limit 10;	0.057

Table 7: Top 10 industrial areas with maximum valued exports

3.4.3 UPDATE Operation

Update Operations are used to change some properties of the data. In case of relational database, only data values could be changes while in non-relational database, even new fields could be added. Following update query was applied on sample dataset.

Update naics of all entries that have naics to 314 to 313 (Textile Mills to Textile Product Mills)

DataBase	Query	Execution Time
MySQL	update supply_chain set naics=314 where naics=313;	0.203
MongoDB	db.shipments.updateMany({ "NAICS": 313}, {$set: {"NAICS": 314}}).	0.003
Neo4j	match (n:Location)<-[r:shipment{naics:"313"}]-(m:Location) set r.naics="314";	0.107

Table 8: Update Operation

3.4.4 DELETE Operation

Delete Operations are used to remove records from a database. These records may be filtered based on some conditions. Following delete query was implemented.

Delete shipments of naics=314

DataBase	Query	Execution Time
MySQL	delete from supply_chain where naics=314;	0.169
MongoDB	db.shipments.deleteMany({"NAICS":314});	0.092
Neo4j	match (n:Location)<-[r:shipment{naics:"314"}]-(m:Location) delete (r);	0.059

Table 9: Delete Operation

4.0 Performance Analysis

Performance analysis of these datastores can be done based on their query processing time. There are also other measures involved like disk space occupied, complexity of query designing etc. Below is the graphical resentation of the query execution times.

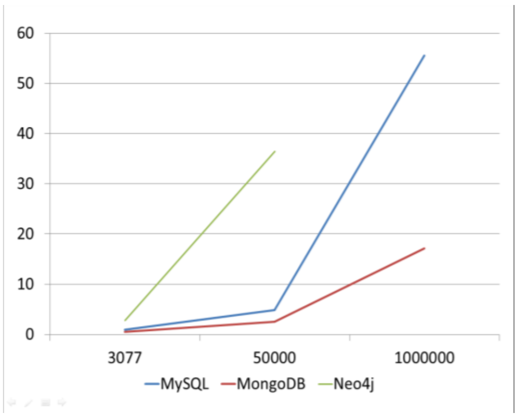


Fig.4 Create Query Analysis

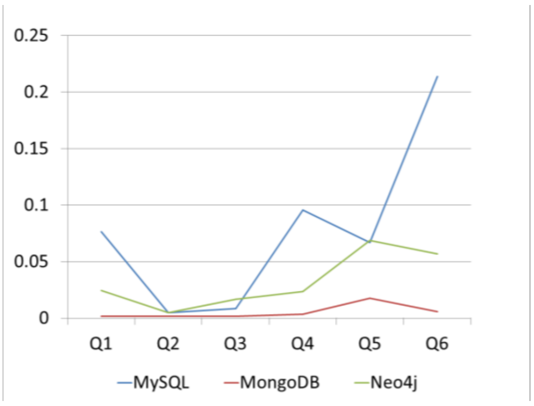


Fig.5 Read Query Analysis

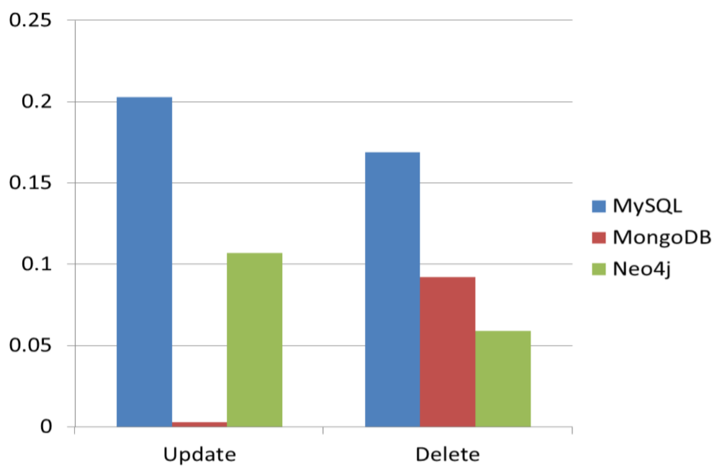


Fig.4 Update and Delete Query Analysis

5. Findings

The performance analysis of databases gives the following findings:

1. Fig. 4 represents the comparison of database creation time as a function of database size for each datastore. This figure clearly demonstrates that MongoDB takes the smallest time to insert values to its database. MongoDB uses WiredTiger Framework to store its data values to disk. Moreover it converts data into BSON format from JSON before storing it onto disk which accelerates the speed of insertion and other operations. MySQL's insertion time is a steep function of data size. Data loading in MySQL is divided into two steps i.e. schema creation and inserting values into that schema. The former step takes equal time while the execution time of later step varies with data size. On the other hand, Neo4j has various mechanisms to import data. The one used here was to use LOAD command. This command is unable to import very

large datasets but even for smaller datasets, it takes longer durations to insert than other databases.

2. A similar kind of performance is seen in READ queries. MongoDB comes out to be the fastest with MySQL and Neo4j following it. In some queries, Neo4j performs fast as compared to MySQL because of its graphical data storage and more suitable data model.
3. It is further seen that MongoDB performs very quick updates while MySQL and Neo4j take greater time.
4. However, the DELETE queries are quickest in Neo4j because it is easier to disallocate memory assigned to a node in a graph.
5. Although Neo4j comes out to be slower database but its data representation is very useful while drawing relationships between the data. This feature is highly appreciated for analysis but in case of application development, where front end is designed separately by developers, such features don't matter. Also, there are a lot of queries that can be easily executed in Neo4j. The queries such as mapping the path from one firm to another are an example of such queries.
6. There is also a large variety in disk space occupied by these databases. Neo4j uses a lot of space for installing its functionalities in the database folder while MySQL and MongoDB just store data in respective files. In this study, Neo4j occupied 32MB, MySQL occupied 1.58MB while MongoDB used up only 0.264 MB of disk space.
7. Another important parameter is the query language. MongoDB although performs a lot of functions very quick, but has a complex query language in comparison with SQL used in RDBMS and CQL(Cypher Query Language) used Neo4j. CQL is a very easy query language with capabilities to perform graphical queries.

6.0 Conclusion

From this research, it is clear that different database can be used to model same data in different manners. The query execution time is a very general measure of database performance. Different database use different methods to accomplish the solutions to different queries. Some of them outperform the others in some cases but may be completely unknown to some other functionality. The choice for best database can never give a unanimous result because it solely depends upon the requirements of the users. In this research, since data taken belongs to a network structure, graph data stores provides best representation of the results and covers a lot more useful queries' execution. However, MongoDB proves to be the fast than others. This indicates that the choice for best depends upon what kind of queries are to be implemented on the database. Hence, a thorough analysis of the data entities, relationships and future requirements from the database is to be made while choosing a proper data model and a suitable database to implement that model.

References

1. Abramova V et al (2014) Experimental Evaluation Of Nosql Databases. International Journal of Database Management Systems. https://doi.org/10.5121/ijdms.2014.6301
2. Arora R, Aggarwal R (2013) Modeling and querying data in MongoDB. International Journal of Scientific and Engineering Research, Volume 4, Issue 7.
3. Chauhan A (2019) A Review on Various Aspects of MongoDb Databases. International Journal of Engineering Research & Technology, Volume 08, Issue 05.
4. Damodaran D et al (2016) Performance Evaluation Of MySQL And MongoDB Databases. International Journal on Cybernetics & Informatics. https://doi.org/10.5121/ijci.2016.5241

5. Gessert F, Wingerath W et al (2016) NoSQL database systems: a Survey and decision guidance. Computer Sci Res Dev, © Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/s00450-016-0334-3
6. Gu Y, Shen S, Wang J, Kim J (2015) Application of NoSQL database MongoDB. IEEE International Conference on Consumer Electronics. https://doi.org/10.1109/ICCE-TW.2015.7216831.
7. Gupta A, Tyagi S et al (2017) NoSQL Databases: Critical Analysis and Comparison. IEEE. https://doi.org/10.1109/IC3TSN.2017.8284494
8. Gyorodi C et al (2015) A Comparative Study: MongoDB vs. MySQL. 13th International Conference on Engineering of Modern Electric Systems. https://doi.org/10.1109/EMES.2015.7158433
9. Kaur K, and Rani R (2013) Modeling and querying data in NoSQL databases. International Conference on IEEE on Big Data, Silicon Valley, CA, USA. https://doi.org/10.1109/BigData.2013.6691765
10. Klein J, Gorton I et al (2015) Performance Evaluation of NoSQL Databases: A Case Study. Association for Computing Machinery. https://doi.org/10.1145/2694730.2694731
11. Miller J, (2013) Graph database applications and concepts with Neo4j. 23rd-24th Southern Association for Information Systems Conference, Atlanta, GA, USA.
12. Priyanka, AmitPal (2016) A Review of NoSQL Databases: Types and Comparison with Relational Database. International Journal of Engineering Science and Computing. https://doi.org/10.4010/2016.1226
13. Ryan D L et al (2018) Evaluation Criteria For Selecting NoSQL Databases In A Single-Box Environment. International Journal of Database Management Systems. https://doi.org/10.5121/ijdms.2018.10401
14. Sanobar K, Mane V (2013) SQL Support over MongoDB using Metadata. International Journal of Scientific and Research Publications, Vol. 3, No.10, pp. 1-5
15. Zhao G, Huang W et al (2013) Modeling MongoDB with Relational Model. Fourth International Conference on Emerging Intelligent Data and Web Technologies. https://doi.org/10.1109/EIDWT.2013.25