Command Line Interface and Data Management Systems (for Big Data)

Thejas Raju Ravi Govind Raju

1 Introduction

A database management system (DBMS) refers to the framework that is responsible for the storage, retrieval and performing of modifications to a computing system. This system is mainly used for creating and managing a database. A DBMS helps in storing and maintaining data in a secure manner. It is equipped with tools that allow implementation of various operations such as insert, update, delete, etc. which aid in management of a database system.

A database management system is characterised by its ability to store any type of data and supporting atomicity, consistency, isolation, and durability properties. This ensures that data is not lost during transactions. The DBMS also has a provision for maintaining backups and recovery which are essential in the event of system failure or crash.

The popular types of Database Management Systems are Relational Database Management System (RDBMS) and NoSQL Database Management Systems. The relational model uses table based databases and each row in a table contains unique data. However, it has limited scalability and complexity increases for data which is difficult to be represented as tables.

On the other hand, a NoSQL model uses collections and nodes instead of a relational schema. It provides for easy scalability and has relatively lesser initial and running costs as compared to RDBMS model.

A Command Line Interface (CLI) is an interface that takes textual input to perform some fixed procedures. A CLI lacks the flexibility of a graphical user interface, however, the working of CLI is relatively simple and straightforward with the results being represented visually to the user.

Big Data refers to a very large set of data. It is characterised by its volume or size of the data, velocity or speeds at which its being generated, variety or the types of data being generated, and variability or the flow of data across systems.

Big data can be structured and unstructured. RDBMS models such as MySQL can be used for structured data which has clearly defined data and types. NoSQL can be used to store unstructured data where there is no clear indication of the type of data being stored.

In this project, a Command Line Interface is being used to implement database management system for big data. Bash is used to perform the pre-processing of the dataset which contains excerpts of novels. Another database containing employee

information which consists of 4 million records is stored in a structured manner using MySQL and also stored in collections within MongoDB which is a NoSQL system. The performance of these two database management systems is analysed and interpreted.

2 Requirements

The system is expected to be capable of handling a large volume of data and perform analysis and extract meaningful information from the data.

For performing Big Data jobs in Bash, the system must be capable of downloading files which contain data for analysis. There must be provisions to perform procedures such as pre-processing of the data which is an essential step in data analysis as it helps in the removal of noise or outliers, across the whole dataset. The system should be able to interpret and represent the computations across the dataset to reveal patterns present in the dataset.

The system is also required to test the performance of two database systems. The system is expected to be able to load the large dataset and execute the same across the local server. The system must be capable of making changes to the SQL file to make modifications to names of database. This file must then be parsed and executed to create a relational database system. The system is expected to be able to represent the database for visual interpretation of the user.

The relational database containing the Big Data is to be represented in a NoSQL schema as well. MongoDB is used for this and the system should be able to handle the conversion of a MySQL database tables into a MongoDB collection. These collections are expected to be populated with the data present in the MySQL tables.

On successful creation on two database systems, their respective performance is to be evaluated against each other to make observations with respect to the time taken to perform a specific transactions.

3 Design

The system is designed to run on hardware which consists of x86_64 architecture. There are four CPUs present and the OS used is Ubuntu which in combination is capable of handling large volumes of data. The Bash shell consists of tools which facilitate various Big Data jobs and routine system utilities such as downloading files. The files to be analysed are downloaded using the ‘wget’ command which is helps in downloading files from location specified as the parameter.

The file to be processed needs to be pre-processed before analysis. This step includes removal of noise such as punctuations. This is achieved by ‘tr’ command which is an abbreviation for ‘translate’. The ‘tr’ command is specified with option –d followed by pattern to be deleted from the file. Punctuations are specified within the pattern to be deleted from the file. Every occurrence of a punctuation symbol is removed from the file on execution.

Stop-words are also removed as they do not aid in analysis of the file. The ‘grep’ command is used to load a file containing stop-words to be removed, and performs

pattern matching to remove every occurrence of a stop-word from the file being analysed. The analysis to find pattern such as most frequent words is performed using a pipeline of 'sort' and 'uniq' commands. This helps in drawing insights about the file being analysed.

The system is capable of downloading large volumes of data using the 'git' command which takes the location of files to be downloaded as its parameter. These files are downloaded and stored in the current working directory. Modifications to the SQL files are performed using the stream editor 'sed' which is used to find a pattern string present in the file and replaced in-place with a specified string.

The SQL file is parsed using the 'source' command to create the databases specified within the file. The content of the database can be viewed using the database manipulation commands such as 'show' used to display databases or tables present and 'select' used to display content specific to one or more tables.

The relational database containing Big Data are migrated to a NoSQL schema with the help of a script that executes MySQL command 'show tables' in combination with the 'awk' command which is used to slice the column containing the table names. These table names are passed to a mongo command 'createCollection()' which takes the table names as parameter and creates collections corresponding to the MySQL tables.

The MongoDB collections are populated using a script which reads the contents of each table present in the relational database using the 'select' command. The bash 'tr' command is used to insert tabs between the returned results and is stored in a CSV file. This CSV file is imported to MongoDB using the 'mongoimport' command and the collection corresponding to the table in MySQL is populated.

The performance of the relational database model and the NoSQL model are evaluated by executing a script that measure the time taken to perform a specified transaction on each database. This time returned provides a guideline to measure the speed of execution of transaction on a database and draw conclusions from them.

4 Challenges Faced

4.1 Task-1 Big Data Jobs in Bash

The pre-processing of data to remove punctuations can be performed using the stream editor 'sed' to find a pattern (punctuation symbol) and deletes it. The 'tr' command can also be used to delete a character when a matching character is found in the target file. The decision to favour 'tr' command over 'sed' was taken as the 'tr' command performs character-based transformations whereas the 'sed' performs string-based transformations.

The challenge of including all punctuation symbols was solved using the '[:punct:]' option to search and delete all punctuation symbols.

To list the twenty most frequent characters, it was necessary to find all the words present in the file and count how many times each of these words occurred. In order to achieve this, the file containing data without punctuation was parsed to find all words present in the file and was sorted alphabetically. To

count repeated words, the ‘uniq’ command was used with –c option. The result was again sorted in descending order using the ‘sort –nr’ command. The twenty most frequent words were extracted using the ‘head -20’ command.

The results of the above query presented that stop-words were occurring in most frequency. As stop-words seldom convey meaning, their removal was essential to calculate the frequency of words that actually convey meaningful information. The file containing list of stop-words to be removed was passed as an argument to the ‘grep’ command which when pipelined with the data containing stop-words, performed string matching and deleted the stop-words from the file. The frequency of the words in data which is free from stop-words was calculated in a similar manner as the previous word-frequency calculation.

4.2 Task-2 Performance Testing of the Two Database Systems

The GitHub repository was downloaded using the ‘git clone’ functionality. The challenge of parsing the SQL file to find name of database and performing a change was met by examining the existing name of the database. This existing name was passed to the stream editor ‘sed’ which used existing name and new name as the parameters. The ‘sed’ used string matching to find and replace the each occurrence of the database name. To make changes in more than one places, the option ‘g’ was used to specify global replacement across the file.

The creation of collections in MongoDB corresponding to the tables present in the MySQL database required passing the names of each table to the createCollection() command. The challenge of obtaining the table names was solved using a Bash script where a MySQL command ‘show tables’ was executed to find all tables present in the database and the result was sliced using the ‘awk’ command to return only the table-names. To create all collections, these table names were passed iteratively such that each iteration created a single instance of a collection which corresponded to the table in MySQL schema.

To populate the collections within MongoDB, the challenge lied in reading each record present in each table, parsing the same and storing it in a corresponding collection. A Bash script was used in a similarly as before to store all table names within a variable. The challenge of reading each record in each table could be overcome in two methods. One, using ‘awk’ to iteratively read each record within each column and store it in the corresponding collection. As this could result in a poor performance, the second method was adopted where the content of each table present in the MySQL database was queried using the ‘select’ command and tabs were added between the results of the query using the ‘tr’ command. This was saved in a CSV file which was then uploaded to the MongoDB database using the ‘mongoimport’ command to import the CSV file and populate each corresponding collection.

The performance was measured by observing the time taken for the execution of a query of equal complexity on each database system.

5 Conclusion

This project demonstrates the handling of Big Data in Bash. The Bash commands were utilised to perform pre-processing of the files to eliminate any noise or unnecessary data. The ability of Bash to handle files of large volumes can be noticed which further shows its robust nature. The flexibility to fulfil various kinds of processes that need to be applied across a large volume of data. The tools not only support pre-processing but also aid in the analysis of the files and draw meaningful insights from them. Hence, Bash must be considered for crude Big Data jobs due to its flexibility and robustness.

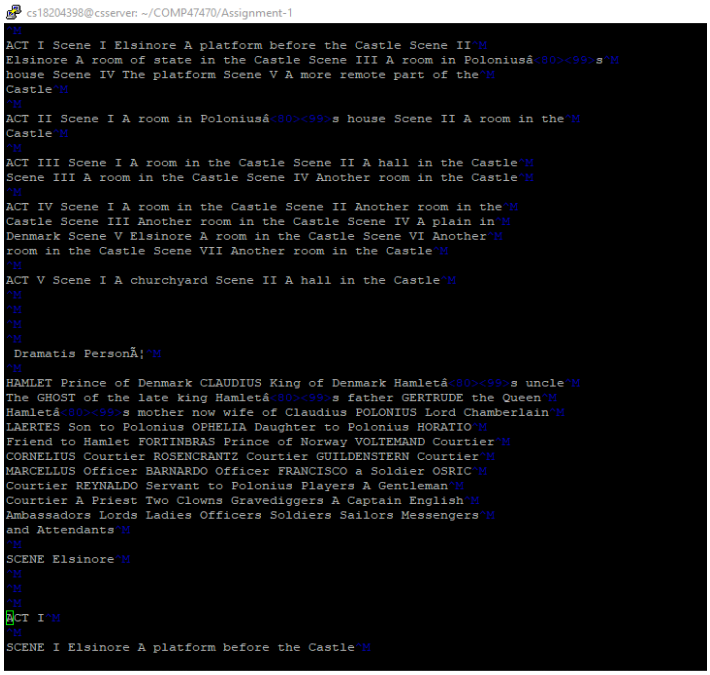
The project also helps in establishing the performance of database systems. By comparing the time taken to execute queries of equal complexities across the two databases, the speed of execution of queries can be observed and it can be seen that the NoSQL model has higher execution time for complex queries whereas both the relational and NoSQL models take similar time to execute simple queries. NoSQL model must be chosen for unstructured data where scaling is required however, for faster computations and look-ups, a relational model must be preferred.

Appendix

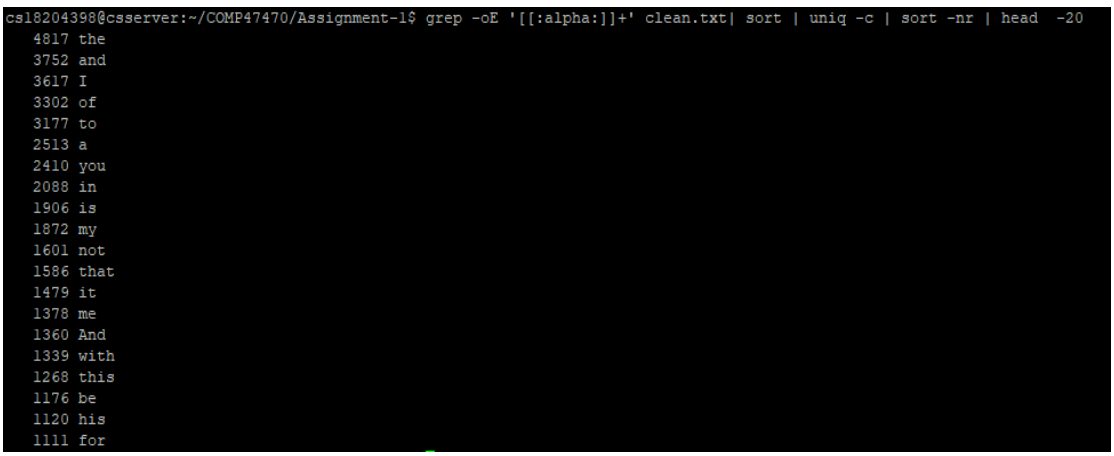
Commands used for execution in the project with respective outputs are represented.

Big Data Jobs in Bash:

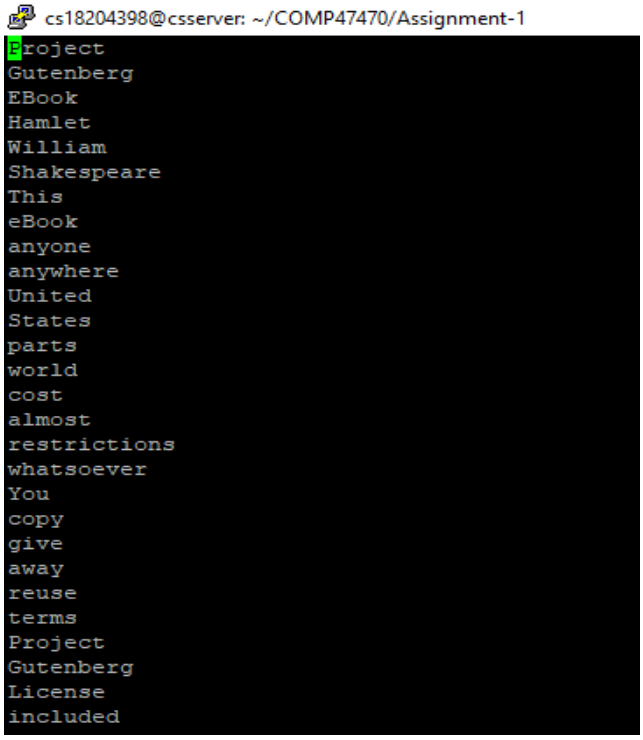
1) Parsing text file to remove punctuation and saving a new version of the file.
   Bash Command: cat 1524-0.txt 1112.txt 2267.txt 2253.txt 1513.txt 1120.txt | tr -d '[:punct:]' > clean.txt
   Output: #New text file named clean.txt containing data combined from all parameter files with punctuations removed.



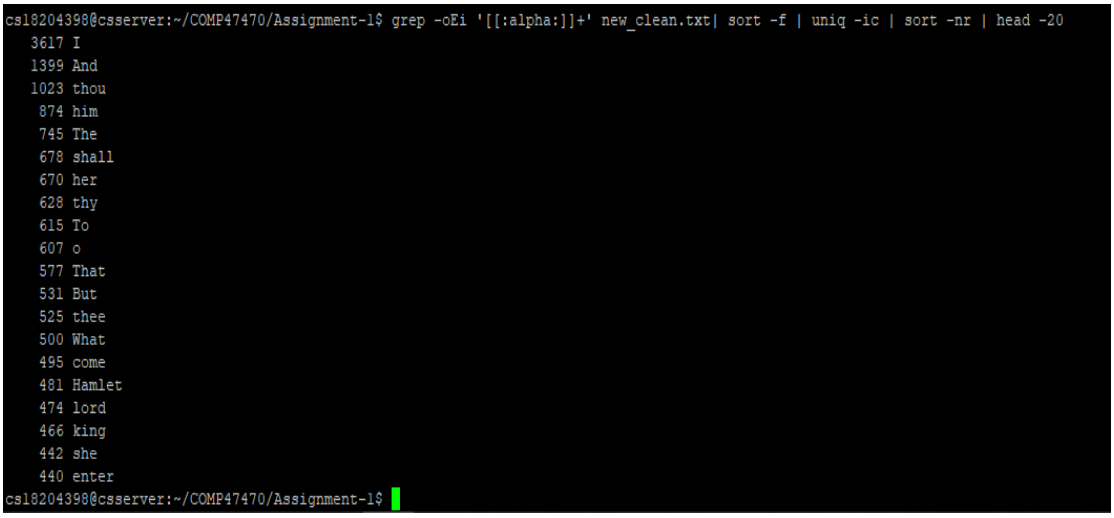
2) #Listing twenty most frequent words from file containing data with punctuations removed.
   Bash Command: grep -oE '[[:alpha:]]+' clean.txt| sort | uniq -c | sort -nr | head -20
   Output: #List of twenty words occurring most frequently in the text file. It can be observed that there are mostly stop words present in the list.



3) # Removing stop words from file using a list of stop words.
Bash Command: grep -oE '[[:alpha:]]+' clean.txt | grep -vwFf stop-words.txt > new_clean.txt
Output: #The list being used is contained within the file named 'stop-words.txt' and the new file which is free from stop-words is saved as new_clean.txt



4) # Listing twenty most frequent words from file containing data with stop-words removed.
Bash Command: grep -oEi '[[:alpha:]]+' new_clean.txt| sort -f | uniq -ic | sort -nr | head -20
Output:



Performance Testing of Two Database Systems:

Relational Database Model-

1) #Downloading the GitHub Repository:
   Command: git clone https://github.com/datacharmer/test_db
2) #Modifying the name of employee.sql file to change name of database.
   Commands: sed -i 's/EXISTS employees/EXISTS <name>/g' employees.sql
   sed -i 's/USE employees/USE <name>/g' employees.sql
3) #Loading the database in MySQL
   Commands: Linux Server: mysql -h localhost -u <name> -p (Password)
   MySQL Server: source employees.sql
4) #Checking the database
   Command: use <name>;
   show tables;
   Output:

NoSQL Model-

1) # Bash script to read structure of the MySQL database and creating MongoDB collection for each table present in the MySQL server.

   Script:

   #!/bin/bash

   line=$(mysql -h localhost -u <name> -p<password> cs18204398 -e "show tables from cs18204398;" | awk 'FNR>=2{print $1}')

   for i in $line
   do

   mongo cs18204398_as1 --eval 'db.createCollection("""${i}""");'

   done

   echo "MongoDB Collections created successfully"

Output:

```
cs18204398@csserver:~/COMP47470/Assignment-1/task-2/test_db$ sh script_1.sh
mysql: [Warning] Using a password on the command line interface can be insecure.
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
[object Object]
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
[object Object]
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
[object Object]
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
[object Object]
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
[object Object]
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
[object Object]
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
[object Object]
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
[object Object]
MongoDB Collections created successfully
cs18204398@csserver:~/COMP47470/Assignment-1/task-2/test_db$
```

2) # Populating the collections is MongoDB using a bash script.

Script:

```
#!/bin/bash

line=$(mysql -h localhost -u <name> -p<password> cs18204398 -e "show tables from cs18204398;" | awk 'FNR>=2{print $1}')

for i in $line
do

line1=$(mysql -h localhost -u <name> -p<password> cs18204398 -e "use cs18204398; select * from $i" )

echo "$line1"| tr '\t' ','> output.csv

mongoimport -d cs18204398_as1 -c $i --type csv --file output.csv –headerline

rm output.csv

done

echo "The Collections have been populated"
```

Excerpt of Output:

```
mysql: [Warning] Using a password on the command line interface can be insecure.
connected to: 127.0.0.1
2019-03-06T20:58:24.118+0000            Progress: 4630675/17718407      26%
2019-03-06T20:58:24.118+0000                    117900  39300/second
2019-03-06T20:58:27.008+0000            Progress: 7285866/17718407      41%
2019-03-06T20:58:27.008+0000                    184200  30700/second
2019-03-06T20:58:30.006+0000            Progress: 9657538/17718407      54%
2019-03-06T20:58:30.006+0000                    243100  27011/second
2019-03-06T20:58:33.186+0000            Progress: 12004534/17718407     67%
2019-03-06T20:58:33.186+0000                    301400  25116/second
2019-03-06T20:58:36.135+0000            Progress: 14456915/17718407     81%
2019-03-06T20:58:36.135+0000                    362300  24153/second
2019-03-06T20:58:39.166+0000            Progress: 16989363/17718407     95%
2019-03-06T20:58:39.166+0000                    425200  23622/second
2019-03-06T20:58:40.061+0000 check 9 443309
2019-03-06T20:58:43.134+0000 imported 443308 objects
The Collections have been populated
cs18204398@csserver:~/COMP47470/Assignment-1/task-2/test_db$
```

Comparing the performance of both the models:

1) *Simple MySQL query:*

#Displays the contents of table ‘titles’ whose employee number is 10014.

```
select * from titles
where emp_no=10014;
```

Output:

```
mysql> select * from titles
    -> where emp_no=10014;
+--------+----------+------------+------------+
| emp_no | title    | from_date  | to_date    |
+--------+----------+------------+------------+
|  10014 | Engineer | 1993-12-29 | 9999-01-01 |
+--------+----------+------------+------------+
1 row in set (0.04 sec)
```

Simple MongoDB query:

#Displays the contents of collection ‘titles’ whose employee number is 10014.

```
db.titles.find({"emp_no":10014});
```

Output:

```
> db.titles.find({"emp_no":10014});
{ "_id" : ObjectId("5c7b22021e11df7c5b398565"), "emp_no" : 10014, "title" : "Engineer", "from_date" : "1993-12-29", "to_date" : "9999-01-01" }
>
```

Complex MySQL query:

#Displays the total of salary column of the salaries table paid to all employees.

```
select sum(salary) "TotalAmount"
from salaries;
```

Output:

```
mysql> select sum(salary) "TotalAmount"
    -> from salaries;
+--------------+
| TotalAmount  |
+--------------+
| 181480757419 |
+--------------+
1 row in set (1.34 sec)
```

Complex MongoDB query:

#Displays the total of salary column of the salaries collection paid to all employees.

db.salaries.aggregate({ $group: { _id : null, sum : { $sum: "$salary" } } });

Output:

```
> db.salaries.aggregate( { $group: { _id : null, sum : { $sum: "$salary" } } });
{ "_id" : null, "sum" : NumberLong("181480757419") }
>
```

2) # Bash script to compare the performance of both the models.

Script:

```
#!/bin/bash
START_TIME1=$(date +%s)
mysql -h localhost -u <name> -p<password> cs18204398 -e "use cs18204398;
select * from titles where emp_no=10014;"
END_TIME1=$(date +%s)
START_TIME2=$(date +%s)
mongo cs18204398_as1 --eval 'db.titles.find({"emp_no":10014});'
END_TIME2=$(date +%s)
START_TIME3=$(date +%s)
mysql -h localhost -u <name> -p<password> cs18204398 -e 'select sum(salary)
"TotalAmount" from salaries'
END_TIME3=$(date +%s)
START_TIME4=$(date +%s)
mongo cs18204398_as1 --eval 'db.salaries.aggregate( { $group: { _id : null,
Amount : { $sum : "$salary" } } });'
END_TIME4=$(date +%s)
echo "Time taken to execute simple MySQL query:$(($END_TIME1-
$START_TIME1))"
echo "Time taken to execute simple MongoDB query:$(($END_TIME2-
$START_TIME2))"
echo "Time taken to execute complex MySQL query:$(($END_TIME3-
$START_TIME3))"
echo "Time taken to execute complex MongoDB query:$(($END_TIME4-
$START_TIME4))"
```

Output:

```
cs18204398@csserver:~/COMP47470/Assignment-1/task-2/test_db$ sh script_3.sh
mysql: [Warning] Using a password on the command line interface can be insecure.
+--------+----------+------------+------------+
| emp_no | title    | from_date  | to_date    |
+--------+----------+------------+------------+
|  10014 | Engineer | 1993-12-29 | 9999-01-01 |
+--------+----------+------------+------------+
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
DBQuery: cs18204398_as1.titles -> { "emp_no" : 10014 }
mysql: [Warning] Using a password on the command line interface can be insecure.
+--------------+
| TotalAmount  |
+--------------+
| 181480757419 |
+--------------+
MongoDB shell version: 2.6.10
connecting to: cs18204398_as1
[object Object]
Time taken to execute simple MySQL query:0
Time taken to execute simple MongoDB query:0
Time taken to execute complex MySQL query:2
Time taken to execute complex MongoDB query:4
cs18204398@csserver:~/COMP47470/Assignment-1/task-2/test_db$
```

3) It can be seen that the time taken to execute the queries is almost the same when a simple query is executed across both the relational and NoSQL model. However, the NoSQL model tends to perform at a slower pace than the relational model when a complex query is executed. This may be due to the unstructured way in which data is stored in a NoSQL model which makes the lookup and computation relatively more time intensive than the structured relational model.

4) This may not be the best way to compare performance of the two data management systems as different runs tend to provide a varying estimate of the time taken. This may be due to the execution process of the system on which the queries are being executed. The processor might execute the queries differently at each run which leads to varying estimates. Taking the average of time taken across a number of runs would provide a better estimate of performance of both the database management systems.

6 References

https://www.stackoverflow.com

https://www.datamation.com/big-data/structured-vs-unstructured-data.html

https://www.sas.com/en_ie/insights/big-data/what-is-big-data.html

https://searchsqlserver.techtarget.com/definition/database-management-system

https://docs.mongodb.com/manual/reference/sql-comparison/