WONDWESSEN HAILE ADDAL

A COMPARATIVE ANALYSIS OF NON-RELATIONAL DATABASES FOR WEB

NEU 2019

A COMPARATIVE ANALYSIS OF RELATIONAL AND NON-RELATIONAL DATABASES FOR WEB APPLICATION

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES OF NEAR EAST UNIVERSITY

By
WONDWESSEN HAILE ADDAL

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Software Engineering

NICOSIA, 2019

A COMPARATIVE ANALYSIS OF RELATIONAL AND NON-RELATIONAL DATABASES FOR WEB APPLICATION

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES OF NEAR EAST UNIVERSITY

By
WONDWESSEN HAILE ADDAL

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Software Engineering

NICOSIA, 2019

Wondwessen Haile Addal: A COMPARATIVE ANALYSIS OF RELATIONAL AND NON-RELATIONAL DATABASES FOR WEB APPLICATION

Approval of Director of Graduate School of Applied Sciences

Prof. Dr. Nadire ÇAVUŞ

We certify this thesis is satisfactory for the award of the degree of Masters of Science in Software Engineering

Examining Committee in Charge:

Assist. Prof. Dr. Kaan UYAR	Committee Chairman, Computer Engineering Department, NEU
Assist. Prof. Dr. Erkut İnan İŞERİ	Committee Member, Electrical and Electronic Engineering Department, NEU
Assist. Prof. Dr. Ümit İLHAN	Supervisor, Committee Member, Computer Engineering Department, NEU

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Wondwessen Haile
Signature:
Date:

ACKNOWLEDGEMENT

I am very grateful to my advisor, Assist. Prof. Dr. Umit Ilhan for his continuous follow up, advisory and encouragement in planning and implementation of my master's thesis, without his encouragement and motivation finalization in time wouldn't have been a reality.

The moral and financial support from my sister Agernesh Terefe, specially during periods of my acclimatization in Northern Cyprus was also of big value for her gratefulness that has helped me to be successful in my stay in near east university.

The special memory of my Dad and Mama, seeing your strength is also with considerable place in my life as memorizing your parenthood gives me special vigor.

ABSTRACT

For the past several years, the data storage technology has been developed to store various types information with different specification. Manly relational database has been the default choice for data storage, especially for commercial applications. But now many other database technologies are emerged with high performance, scalability, security, speed to retrieve data in different form. Because of having many database applications, it is difficult task to choose the appropriate database technology for enterprise applications. This paper aims to provide theoretical and the extensive experimental comparisons on four databases technologies MYSQL, PostgreSQL, MongoDB and Cassandra. Due to those database technologies are currently used in different enterprise system, this research provides a possible much investigation and report faster and more detailed analysis based on the concrete examination and test cases. The test cases for research includes several key elements such as the nature of data modeling, data, scalability and speed. We can measure and analyze the performance of databases based on CRUD (create, read, update and delete) operations. Moreover, the study includes the difference among those databases regarding with their popularity and community support. At last, the performance of the databases explained based on CRUD operations. This factor helps web application architects and designers to choose the suitable database software for their commercial applications.

Keywords: SQL; NoSQL; MYSQL, PostgreSQL; MongoDB; CASSANDRA; Database

ÖZET

Geçtiğimiz birkaç yıl boyunca, veri depolama teknolojisi, farklı özelliklere sahip çeşitli tiplerdeki bilgileri depolamak için geliştirilmiştir. Manly ilişkisel veritabanı, özellikle ticari uygulamalar için ve veri depolama için varsayılan seçenek olmuştur. Ancak şimdi birçok başka veritabanı teknolojisi, yüksek performans, ölçeklenebilirlik, güvenlik ve farklı formlarda veri alma hızı ile ortaya çıkmıştır. Birçok veritabanı uygulamasından dolayı, kurumsal uygulamalar için uygun veritabanı teknolojisini seçmek zor bir iştir. Bu makale, dört veritabanı teknolojisi MYSQL, PostgreSQL, MongoDB ve MongoDB ile ilgili teorik ve kapsamlı deneysel karşılaştırmalar sunmayı amaçlamaktadır. Çünkü bu veritabanı teknolojileri şu anda ertelenmiş kurumsal sistemde kullanıldığından, bu araştırma olası bir incelemeyi mümkün kılıyor ve somut incelemeye ve test durumlarına dayanarak daha hızlı ve daha ayrıntılı analizler sunuyor. Araştırma için test senaryoları, veri modellemenin doğası, veri, ölçeklenebilirlik ve hız gibi temel unsurları içerir. Veritabanlarının performansını CRUD (oluşturma, okuma, güncelleme ve silme) işlemlerine dayanarak ölçebilir ve analiz edebiliriz. Ayrıca, çalışma, popülariteleri ve topluluk desteği ile ilgili olarak bu veritabanları arasındaki farkı içermektedir. her birinin artılarını ve eksilerini belirleyebiliriz. Son olarak, veritabanlarının performansı CRUD işlemlerine dayanarak açıklanmıştır. Bu faktör web uygulama mimarlarının ve tasarımcılarının ticari uygulamaları için uygun veritabanı yazılımını tercih etmelerine yardımcı olur.

Anahtar Kelimeler: SQL; NoSQL; MYSQL, PostgreSQL; MongoDB; CASSANDRA; Veritabanı

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACID:	Atomicity, Consistency, Isolation, and Durability
BSD:	Berkeley Software Distribution
BSON:	Binary Structure Object Notation
CAP:	Consistency, Availability and Partition Tolerance
CPU:	Central Processing Unit
CRUD:	Create, Read, Update, Delete
DB:	Database
DBMS:	Database Management System
GPA:	General Public License
IT:	Information Technology
JSON:	JavaScript Object Notation
NOSQL:	Not Only SQL
PC:	Personal Computer
RAM:	Random Access Memory
RDBMS:	Relational Database Management System
SQL:	Structured Query Language
SSTABLE:	Static and Sorted Table
UUID:	Universally Unique Identifier
WAL:	Write Ahead Log

CHAPTER 1
INTRODUCTION

The Database is a kind of data collections. It is the backbone of many software systems and choosing the appropriate technology needs an elaborate process. Though when using the word database, we mean to the entire database system, the term basically mentions only to the collection and data. The system which stores Big data, transactions, complications or any other feature of the database is the Database Management System (DBMS). What follows is an enlightenment of the two database types which will be contrasted in this thesis. The focus of this thesis is to explore Relational and non-relational database systems

Relational database model has been serving the web community from 1970 up to now with a higher degree of consistency and functionality. This is because of their powerful futures in terms of data modeling, rich query capability. However, in the late 90s, the data that is to be manipulated in various web applications are constantly changed and becomes more sophisticated to manage in the traditional way. especially, the relational database system has limitations when it approaches big data, non-structured and large amount of data. These leads the web community and software architectures to focus on the new way of data handling system with higher consistency and scalability compared with the traditional database system. Not only SQL or NoSQL is a new conventional database technology that has developed in the recent past as another alternative solution to relational databases.

The relational database is widely used technology in the history of the web industry that has been developed for the last 30 years(Mohamed, et al., 2014). Lots relational databases have come and gone, and presently, there are only a few possibilities to choose from. On the other hand, advanced database technologies such as NoSQL technology provides masses of options through various classifications including key-value, document-oriented, wide-column and graph databases. Each database is designed for particular use cases and has its own strengths and weaknesses.

In this thesis, we explore these two major types of databases systems, relational databases (SQL) and non-relational databases (NoSQL) to investigate their features, benefits, and weaknesses based on the data model and query capability. Selecting the appropriate database for web applications is a very long-term decision that can have severe effects if the business logic cannot be supported easily by the selected database system.

This investigation basically helps the software architectures and developers to look across the possibility to choose the right database system based on analyzing the advantage and disadvantages of those databases before starting their business.

1.1 Motivation

Development of applications is now continuously shifting. Every year software developers discover a new application that tries to make our lives more convenient. And the development of the software industry is mainly depending on the design and architectures of software to provide a solution for the given problem domain. The quality of the software system can be measured from different parameters, parallel to those parameters we have to consider the right database technologies to maximize the performance of the software products, so providing quality products for the community is directly related to the process of selecting appropriate database technology. Choosing the right database technology for the right application is a time-consuming activity.

In addition, the size of the data is continuously increasing. The huge volume of data is getting collected evaluated, and processed in different business enterprises today, and it becomes required to collect several kinds of structured, semi-structured and unstructured data and its use grow into an integral part and this shows the size of the data richness to applications, widely known as big data.

In order to provide quality products to the community and to address these issues related to a large volume of data, the scalability and data processing capability of database technologies needs to be improved. currently, there are various types of database, such as Relational databases, and widely accepted NoSQL databases. This day, there are a lot of computation among several databases, and each database has own advantages and disadvantages.

RDBMS have been viewed as a standard for web applications compared with other storage technologies. they represent and hold data in tables and rows. They're grounded on an outlet of algebraic set theory which is acknowledged as relational algebra. Although the current software products are consuming large volumes of data with varying structures, they are straggling to meet the demands of modern applications. Relational databases use Structured Querying Language (SQL), which is mostly used in different applications mainly involves the management of several. Meanwhile, A non-relational database is a modern storage system that stores and represent complex data in a flexible structure. this makes them more acceptable for advanced web applications than relational databases systems. additionally, the data model of NoSQL databases is highly enhanced to manipulate non-structured data. NoSQL databases like MongoDB represent data in collections of JSON documents.

1.2 Research Objectives

The objective of this study is to compare SQL based and NoSQL based databases in three parameters. The research will present the positive and negative impacts of all selected database technologies on web applications. The assessment of these two databases are based on data model and query performance of each database technologies.

The tree parameters to accomplish in this study are:

❖ To study the nature of those databases by assessing various literature including

- Programming language support, documentation and community support
- Popularity among IT people

❖ To investigate the data model of each database using relatively similar data.

❖ To Contrast the performance of each database by writing queries on a single node

1.3 Scope of The Research

This research assesses non-relational and relational databases from the perspective of data model and performance use cases. The analysis depends on how the given data should be represented, manipulated, structured on those databases and Examining the query performance of each database based on read and write query operations. Based on the result of examination we present the advantage and disadvantage of those technologies. These are important points for analyzing the characteristics of data, which is to be supported by databases that help system architecture and developers to choose the right database system for their web applications

1.4 Thesis Structure and Outline

The first two chapters of the study provide the introductory, purpose of the research and literature review part. The remaining part of the research describes the main features and capabilities of the investigated Databases with data modeling and query performance test.

Chapter 2 Basic Overviews and Literature Review: This chapter focused on reviewing materials related to the study, such as related journals, websites, and books etc.

Chapter 3 Research methodology: This chapter States the research methodologies used to achieve this research including selecting the databases required to perform the experiment, the key construct identification, the data model along with choosing databases technologies.

Chapter 4 Investigated Databases Comparison: This chapter states data model of the selected databases based on comparatively related data

Chapter 5 Implementation: This chapter emphasizes on experimental testing on each database based on CRUD operations with evaluation of the results of the experiment

Chapter 6 Evaluation: Assesses the difference among the selected databases based on the information from chapter 4 and 5

Chapter 7 Conclusion: summary of the entire research

CHAPTER 2

BASIC OVERVIEWS AND LITERATURE REVIEW

There have been enormous researches done to compare and assess existing relational and non-relational databases. experiments targeting on a variety of aspects such as performance, scalability, transaction model. Performance and scalability are the two most significant features to compare database technologies. Therefore, it has been covered thoroughly in many researches work (Tudorica & Bucur, 2011).Few scholars have also enlisted the advantage and disadvantages of both SQL and NoSQL databases. This chapter will focus on the basic concepts of both relational and non-relational databases from the perspectives of different related scholars and literature articles. This articles and related concepts will help the readers to get the right track to know how the remaining activities of research will going on.

The SQL and NoSQL databases are evaluated to measure the performance of the two categories of the databases. The evaluation was done by testing MySQL database from RDBMS category and contrast with MongoDB of the NoSQL database. Based on the authors implementation, they realize that MongoDB is more efficient than MySQL.(Gyorodi, et al., 2015)

Similarly, the performance of relational and non-relational databases was examined by writing large queries and calculating the scalability and efficiency of the selected databases. In this scholar, the authors come up with the challenges, results and the summery from performance test of two NoSQL databases and one relational database systems. They compare 4 databases Cassandra, MongoDB, Couch base Server, and MS SQL Server, based on performance and scalability benchmarks to measure the efficiency of those databases when behaving bulky write requests from real-world organization data. The experiment was held on assessing database efficiency by execution large queries over mostly realistic data (Lourenço, et al., 2018). Based on the result of the test they provide conclusions for real-world system.it is one of the most visible ways of characterizing the performance of databases. Furthermore, other authors examine the performance of NoSQL and SQL databases based on key-value stores, This article contrasts key-value store implementations on NoSQL and SQL databases. Though NoSQL databases are mostly enhanced for key-value stores, SQL databases are not optimized like NoSQL databases. The author's presence, not all NoSQL databases perform better than the SQL databases They compare read, write, delete, and instantiate operations on the key-value storage. They detect that even within non-relational databases there is a wide variation in the performance of these operations. This article also presents the selection of NoSQL databases provides an experimental setup to measure the performance of each database based on the operations. Experimental outcomes measure the timing of these operations and they summarize their findings of how the databases stack up against each other (Li & Manoharan, 2013).

As well as, the performance of MongoDB and MySQL database is examined based on insertion and retrieval operations using a web/android application to explore load balancing (Patil, Hanni, Tejeshwar, & Patil, 2017). This article answers how the data obtained from the various input and output sources should be treated in the several steps to prevent data loss, various strategies were implemented corresponding to prevent the data losses, for implementation purpose the authors used NoSQL and MongoDB databases. Mongo DB is a cross-platform, document-oriented database that provides, high performance and easy scalability confirming effective data management with its noticeable feature of auto-sharding. This paper examines the time taken by the system to read/ insert the data into a MySQL database and MongoDB. The retrieval time or total time is taken by MySQL database to validate user by fetching data is more than a total time taken by MongoDB. Another similar research was performed by Jain and Upadhyay to analyze the transition from relational to non-relational databases (Jain & Upadhyay, 2017). In this paper, the authors explain the modeling about the changes from SQL to NoSQL database also carrying out its pros and cons. They conclude that a NoSQL database is faster and better than the SQL database in many ways, ranging from speed to flexibility. This is the reasons many enterprise organizations are ever-changing their projects so as to use MongoDB instead of traditional SQL database NoSQL is also called the future of data economy. They also suggest some drawbacks of NoSQL databases from the perspective of data security.

2.1 CAP Theorem

Almost nineteen years ago, in 2000, Eric Brewer announced the idea that there is a vital trade-off between consistency, availability, and partition tolerance. This trade-off, which has become recognized as the CAP Theorem, has been broadly discussed ever since.

CAP Theorem is very vital in the Big Data world, especially when we want to make trade-offs between the three, primarily based on our special use case

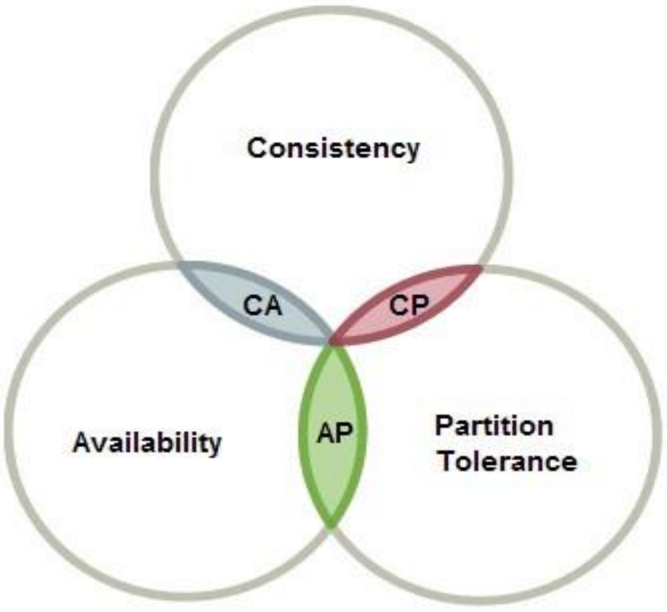


Figure 2.1: CAP theorem(Syed Sadat Nazrul, 2018)

The cap theorem is a tool that helps system designers to make them attentive on the changes while designing distributed web systems. many distributed systems are directly influenced by CAP theorem. It made designers attentive of extensive range of tradeoffs to consider while designing distributed web systems. A web service is executed by a set of servers, possibly distributed over a set of various data centers in different geographical area. Clients make requests of the service and the server respond back to the clients

According to Eric Brewer, in any networked distributed-data system there is an essential trade-off between consistency, availability, and partition tolerance (Brewer, 2003). The theorem states that networked distributed-data systems can only guarantee/strongly support two of the following three parameters:

Consistency - A guarantee that every server in a distributed system returns the right, most recent, similar, and successful write. Consistency refers to every client having the identical view of the data. There are many types of consistency models. Consistency in CAP refers to linearizability or sequential consistency, a very strong form of consistency.

Availability - The second requirement of the CAP Theorem is that the service assures availability. Every server (non-failing) returns a response for all read and write requests in a reasonable amount of time. a fast response is relevant than a slow response, but for the purpose of CAP, it turns out that even needing a subsequent response is sufficient to create problems. The key point here is every non-failing server in the networked environment. To be accessible, every sever must be able to respond in a reasonable amount of time. In real world implementation, certainly, a response that is sufficiently late is just as bad as a response that never occurs

Partition Tolerant - The third requirement of the CAP theorem is that the service be partition tolerant. This characteristic can be seen when communication among servers is not reliable, and the servers may be portioned into numerous clusters that cannot communicate with each other. The system remains on providing a service and upholds its consistency guarantees in spite of network partitions. Network partitions are common in real system. Distributed systems assuring partition tolerance can graciously recover from partitions when the partition heals.

2.2 ACID Properties

ACID properties are a vital idea for databases. The abbreviation stands for Atomicity, consistency, Isolation, and Durability. explanation of ACID properties for distributed database system is that it is a set of characteristics that assures the reliability of database transactions. transaction is the most important unit of a program that allows the system to share data across the globe. it may contain several low-level tasks. Without those ACID properties, everyday transaction such as buy and sale products would be problematic and the potential for wrongness would be massive while using computer systems (Douglas K Barry, 2019).

Atomicity

The phrase "all or nothing" briefly defines the first ACID property of atomicity. Atomicity refers to the ability of the distributed database system to assure that either all of the responsibilities of a transaction are performed or none of them are done. Atomicity states that database transaction modification must track based on “all or nothing” rules. When modification happens to a database, either entire or none of the modification would be presented to anyone. These atomicity properties are very important role in almost all real-world business enterprises.

Consistency

The Consistency property guarantees that the database system keeps in a consistent condition, whether the transaction is well performed or letdown and both before the start of the transaction and after the transaction is ended. Consistency ensures that the change in the value of one instance are consistent with the change in other values in the similar instance. consistency limitation is a predicate on data which serves as a precondition, post-condition, and transformation condition on any transaction.

Isolation

The isolation portion of the ACID Properties refers to the requirement that other set of activates cannot access or see the data in a midway state during a transaction. Isolation property can assist to make concurrency of database. Basically, this property is required when there are concurrent transactions. Concurrent transactions are transactions that happen at the same time.

Durability

Durability property refers that once a transaction is done, its effects are guaranteed to continue even in situations where the system experiences frequent failures. This means if the transaction is committed once, the operation can't be undone and survive from system failure.

2.3 Database Scalability

Scalability of the database can be defined as the ability to increase the computer resources to store huge amount of works. It refers to the system's ability to handle a growing in load by increasing its potential to achieve more total work in the same elapsed time when resources are added. A system is said to be scalable if it can promote increasing workload and data when extra resources are added.it also assures the capability of the system to scaling up and down as per requirement Additionally, it makes the database system to grow to a huge size to support more transactions and operations as the volume of the enterprise business and customer amount increases(Tony Branson, 2016). Figure 2.2 shows how vertical and horizontal scaling works
There are two types of database scalability:

1. Vertical Scaling or Scale-up

In a database world vertical scaling is the process of adding more physical resources to an existing server for improving the performance. the performance of the database server is directly related to the physical resources (memory, storage and CPU) Vertical scaling has been a standard approach of scaling for relational database management system that are designed on a single-server type model.

2. Horizontal Scaling or Scale-Out

Horizontal scaling, is the process of adding many hardware to a system. which means adding new servers to an existing system. When there are more servers with less RAM and processors, it is called horizontal scaling. It increases the performance of the database system by networked many more computers together. Scaling out works based on partitioning the data   and spreading the load on multiple RAM and processors

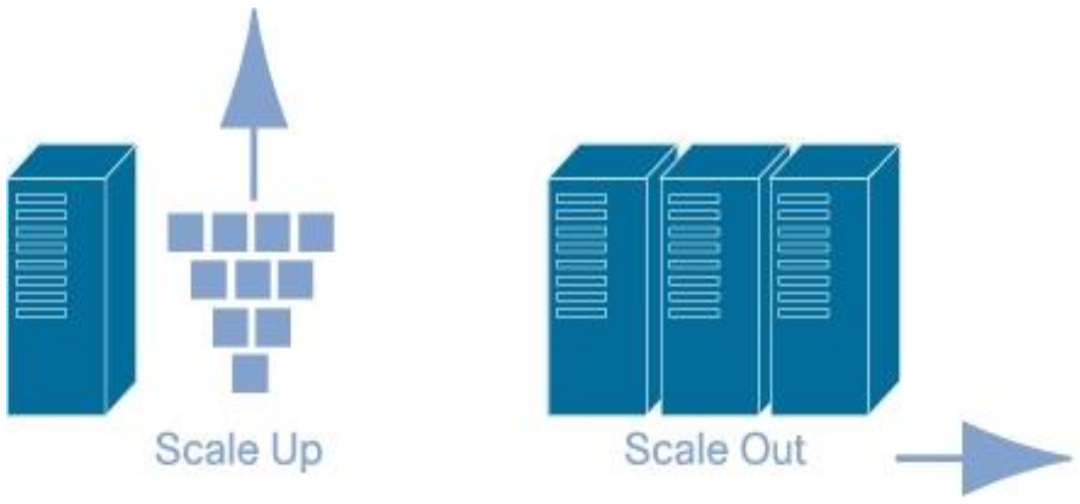


Figure 2.2: Vertical and horizontal scaling example (Georgi Georgiev, 2016)

2.4 Sharding

Sharding is a database architectural pattern which associated to horizontal partitioning. it is very important idea when the system needs very high scalability and absolute availability. In practice, the term is frequently used to mention to any database partitioning that is meant to make a very huge database more manageable. Figure 2.2. shows how data is shared from central cluster to four smaller cluster. This makes the database system more manageable and cost-effective. The central thought behind sharding is based on the concept that as the size of a database and the number of transactions per unit of time made on the database grown up linearly, the response time for querying the database increases exponentially.

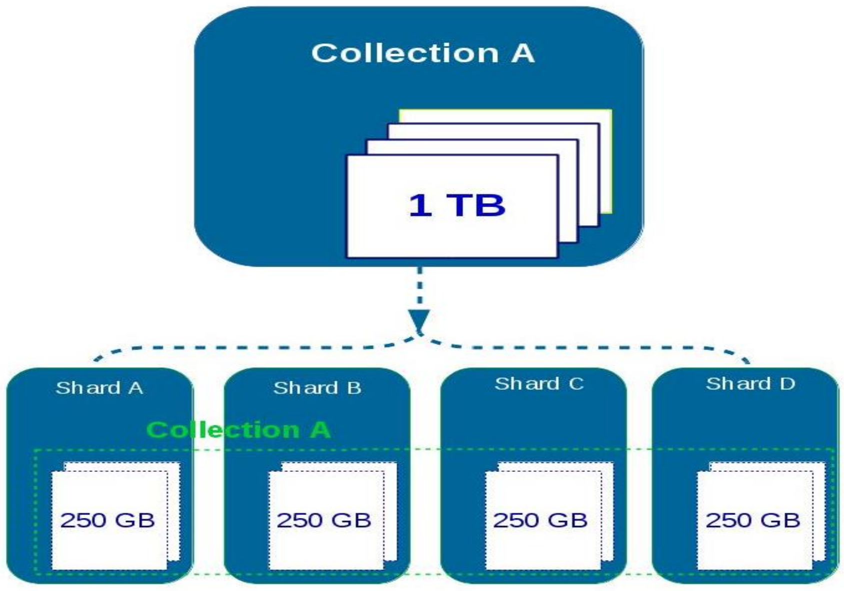


Figure 2.3: A Sharding Example (Eugen Hoble, 2016)

2.5 Data Replication

Data Replication is the process of copying data from central database to one or more databases. It is useful in improving the availability of applications and data. It is simply storing the coped data from one database to other databases. Depending on the replication type the data would be distributed across the server and it allows the users to share the same data without any inconsistency(Arts, 2013). Figure 2.4 shows how replication usually works on.

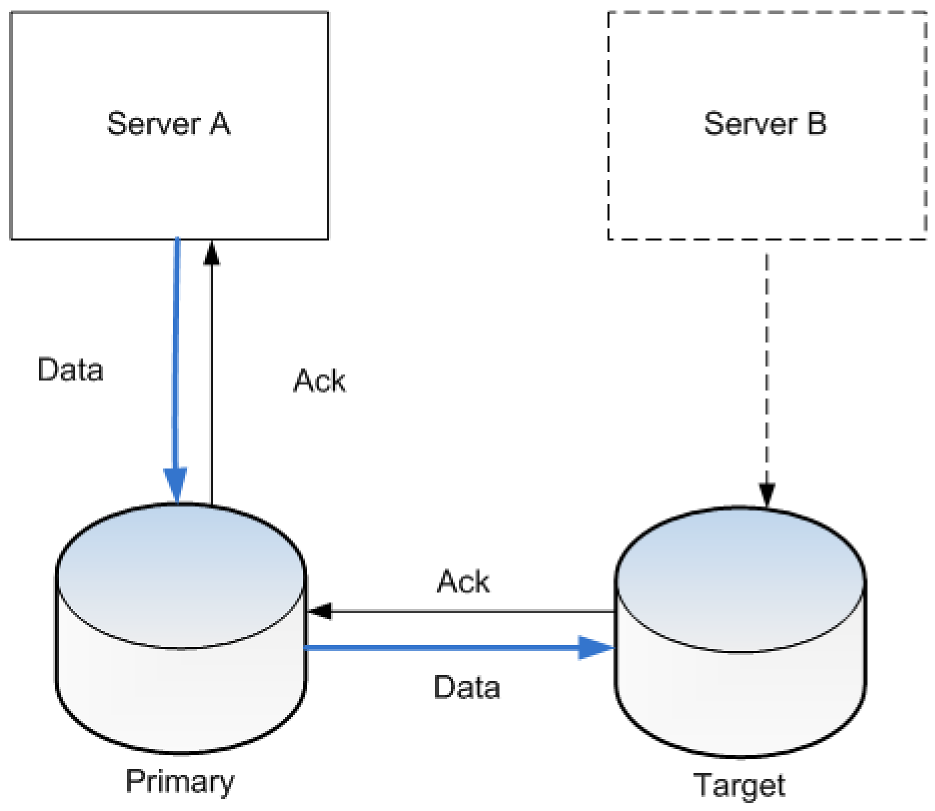


Figure 2.4: A replication Example

CHAPTER 3
METHODOLOGY

The goal of this chapter is describing the steps followed to complete this thesis. The leading step is describing the research methods and conditions which are required to compare the databases. The second step is picking the databases technology that helps us to explore throughout the study and the reasons for selecting it. Then the key ideas that would be necessary to explore each data storage technologies are identified. Lastly, we will select data model which will be used by the selected databases to compare some queries from the web.

3.1 Methods

there are many possible ways to compare those database technologies, the first possibility is exploring the data models of each technology and evaluate based on their data model. The second possibility is by practical experimental approach including to measure the speed and latency based on CRUD operations. finally, the research will come up with the comparison of those databases from the perspectives of speed, programming language support, popularity among various enterprises and IT people.

3.2 General Concept of SQL And NoSQL Database

Fundamentally, before selecting the particular databases from the two cariologies, there are prerequisite to be viewed and analyzed about the difference between SQL and NoSQL databases from the perspectives of IT People and organizations to choose the right database for research.

SQL Databases: the concept of relational database is hangs on relationship between entity's or objects and the data are represented and stored in structured query languages. This property makes them the most preferable to store structured data due to their nature of organizing elements and build relationship among entities (Luke P. Issac, 2014).

NoSQL Databases: An exciting feature of NoSQL database is using a dynamic schema and scalability. in NoSQL database cases, the data is stored and retrieved in the form of documents. It is also the most preferable databases for storing and retrieving big data including structured, semi-structured and unstructured data easily(Luke P. Issac, 2014). Data can be represented flexibly in a different structure like it can be in the form document, graph, column, or Key Value. Table 3.1 shows the difference between relational and non-relational database from various perspectives

Table 3.1: Overview of SQL vs. NoSQL database

	SQL	**NoSQL**
Data storage	Store data in the form of table, data represented in a relational model, with rows and columns. Rows keep unique information about one entity or objects and columns are all the separate data points	The term NoSQL include the mass of databases, each with diverse data storage models. The predominant ones are: document, columnar, key-value and graph.
Schemas and Flexibility	Each record follows to static schema, this means the columns have to be absolute and inaccessible before data entry and each row must keep data for each column. This can be modified; however, it involves altering the entire database and going down.	Schemas are dynamic and regularly data are represented in document mode which means doesn't have to hold data for each column and records can be added easily.
Scalability	In almost all circumstances SQL databases are vertically scalable. this states that we can load data on a single server through increasing the size the hardware resources. in essence, it is feasible to scale RDMS throughout many nodes, but this is a difficult and time-consuming process.	Scaling is horizontal, which means adding extra servers on NoSQL database. This many server can be affordable hardware, making it a lot extra cost-effective than vertical scaling.
ACID Compliancy (Atomicity, Consistency, Isolation, Durability)	The vast majority of relational databases allows ACID properties.	Varies between technologies, but many NoSQL solutions sacrifice ACID properties for availability and scalability.

3.2.1 Popularity of SQL and NoSQL

According to world DB-engine ranking, SQL based databases have been used by many enterprises for decades without competitors, but now days NoSQL is rapidly getting approaches to SQL with advanced storage technologies such as MongoDB, and Cassandra.

and various enterprises are taking decisions to change from relational database to NoSQL based databases. figure 3.1 shows popularity of both SQL and NoSQL

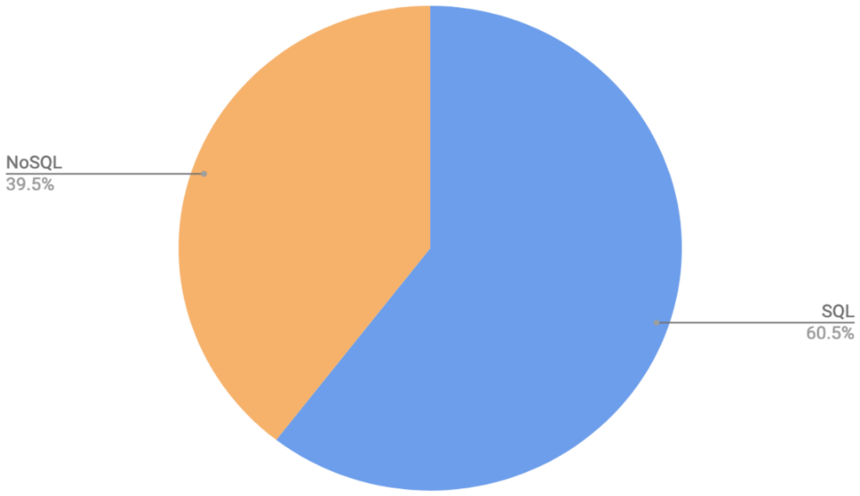


Figure 3.1: Popularity of SQL and NoSQL database(solid IT, 2019)

3.3 Databases Selection

There are many approaches to categorize database systems. A basic classification is based on the database data model, schema, scalability and community support. this research targets on the following 4 database technologies from relational and non-relational categories. The reason to choose those database technologies from many types of database system is based on the popularity of the databases from the viewpoints of software developers, engineers, software architects, dev teams, and IT leaders and grades of the DB-engines ranking. figure 3.2 show popular database technologies in 2019.

- MySQL
- PostgreSQL
- MongoDB
- Cassandra

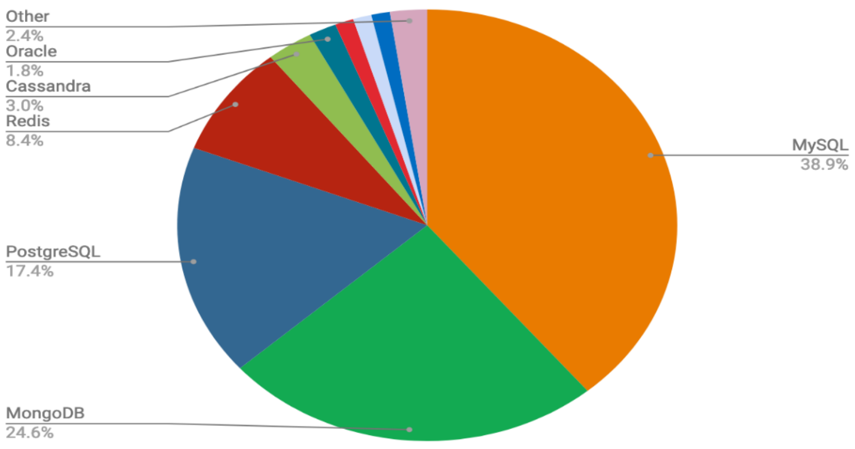


Figure 3.2: popularity of database(solid IT, 2019)

PostgreSQL and MySQL

PostgreSQL and MySQL are two of the most frequently used relational databases technologies. both stores data in tables, these tables are organized into rows and columns to store the data. MySQL is a well-known large-scale relational database. it is a vertically scalable database that helps the database system to manipulate the data by adding more physical resource to the system.

it's an ASCII text file system that powers a huge variety of applications and websites in different sectors It is very flexible that making it a widespread choice for multiple applications. Some additional features regarding the accessibility of detailed security features, ACID properties, and ease of access to support.

PostgreSQL is also one of the most available open source databases and designed by the PostgreSQL Global development group, since it is one of the non-relational database categories, it is also vertically scalable like MySQL databases Both PostgreSQL and MySQL support many operating system such as window, Unix and Linux versions, and supports a number of programming languages

MongoDB and Cassandra

In NoSQL database category we found MongoDB and Cassandra databases more flexible than another non-relational database. MongoDB is classified under a document-based database whereas Cassandra is classified under column-based databases due to the nature of the structure. These technologies are mostly distributed and schema free database systems and holds the data in the form of documents instead of tables.

MongoDB stores data using JSON-like documents that supports multiple data types with multiple data structures. It uses a document like structured query language to manipulate data. Since it is schema-free, it allows developers to create documents without having to create a structure for the document first.

Cassandra is one of the distributed types of database system that stores data in columns instead of storing data in rows. Generally, this database is designed to store data as parts of columns of data. While this indicates that it is the reverse of a relational databases such as MySQL and PostgreSQL.

This popularity and the ability to handle loads of applications makes them used by different companies. for instance, MongoDB has been used by Google, Facebook, Cisco, eBay and Forbes where we came to Cassandra it has been used by Facebook, IBM, Instagram, Spotify, Netflix, And really anymore. In general, non-relational databases are will become the first alternatives of standard database system in the future due to their ability of scalability and their distributive nature

3.4 Criteria for Database Comparison

The comparison criteria for selected databases will be based on the following significant concepts.

- Data model
- Performance evaluation
- Scalability
- Programming language support and popularity among IT people.
- Available resources such as documentation and community

the performance test for all databases is done on the CRUD test benchmarks. CRUD testing is a black box testing. CRUD is an abbreviation for Create, Read, Update, Delete. CRUD testing is one of the testing procedures that allowed us to view the performance of a given database management system. This chapter shows the performance test of all selected databases based on testing benchmarks figure 5.1 shows How the general overviews the crud operations.

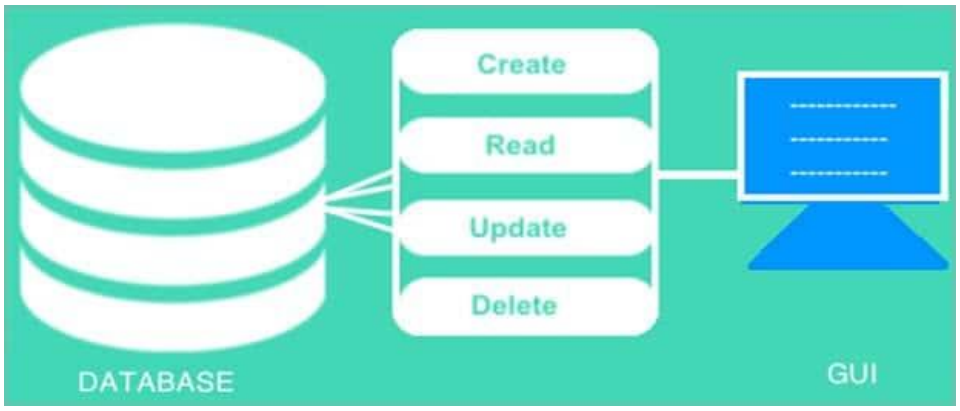


Figure 3.3: Show an overview of CRUD operations (Software Testing Help, 2019)

CRUD describes the basic functionality of database systems from the viewpoint of users. It is consisting of the following operations

- Create: refers to create any new transaction
- Read: describes reading or viewing any transaction.
- Update: modifying the data in the database.
- Delete: refers to removing a certain data from the databases.

3.5 Data Sets

for evaluating the selected databases, we use real-world data which is generated from the web-based customer information system that has seven column sections to evaluate the performance of all databases. the evaluation process is done by executing CRUD (create read, update, delete) operation multiple times to get the average execution time taken by the databases. the size of the data is divided into different sections to examine the result of the execution we used. the size of the data is ranging from 1000 number of records to 100000 number of records. including 1000,5000,10000, 20000, 40000,60000,80000, 100000 records respectively.

3.6 Resources

for experimental part of this study we use:

- ✓ PHP programming language:
- ✓ Apache Web Server, developed by apache foundation.
- ✓ The UNIX timestamp to calculate the speed of the selected query on each database. This helps to know How long did it take the criteria query script to run from start to finish; based on the timestamp we could measure the speed CRUD (create, read, update and delete) operations.
- ✓ For comparisons purpose, we use currently available versions of the database.
- ✓ For testing there is multiple sources of data, however, we use customer information data sets which is the same for all databases.

3.7 Process

This part describes how the study will proceed to compare these 4 database technologies. the leading step is exploring each technologies, MySQL, PostgreSQL MongoDB and Cassandra, from the perspective of data model and query languages as the whole structures, before we go to the practical section. then after the data model of each database we will proceed on the experimental part of the research. finally, the research will explore those storage systems from the outlook of programming language support, popularity among IT people and organizations. This is to offers a detailed description and summary of the selected technologies.

CHAPTER 4
INVESTIGATED DATABASES COMPARISON

This section starts with an outline of the most imperative ideas from the database system and after that dives into the data model and basic terminologies of MySQL, PostgreSQL, Cassandra, MongoDB databases

4.1 MYSQL

MySQL is the leading and the most optimized open source database management tool among all other relational database management systems. it is written in C and C++ and owned by Oracle Corporation (Kofler & Kramer, 2005).

The project of MySQL was begun in 1979. A significant number of the world's biggest and quickest developing companies including Facebook, Google, Adobe, Lucent, and Zappos depend on MySQL to spare time and cash controlling their high-volume Websites, business-basic frameworks, and bundled applications. As new and diverse requisites and needs rose with the web service, MySQL turned into the default for web architectures. From that point forward, the execution and versatility, unwavering quality, and convenience of the world's most well-known open source database, qualities that settle MySQL the first for web applications. due to the service of the web continues developing with driving web properties, for example, Facebook and Google are spearheading better approaches to control the massive amount of data, MySQL is additionally advancing to settled on the heading position on the web industry

4.1.1 Data model

4.1.1.1 MySQL architecture

MySQL is designed based on a client-server framework. There is a database server (MySQL) and subjectively numerous clients (application programs), which connect with the server; that allows them to inquiry information and spare changes. The clients can keep running on the same PC. MySQL architecture is very different from other database servers, and its design qualities make it valuable for a wide scope of purposes just as making it a poor choice for other relational database management systems. MySQL isn't perfect, however, it is sufficiently adaptable to function admirably in extremely requesting conditions, for example, web applications. In the meantime, MySQL can control inserted applications, information distribution centers, content ordering and conveyance programming, profoundly accessible repetitive frameworks, online transaction processing (OLTP), and considerably more. Figure 4.1 shows a logical view of MySQL's architecture.

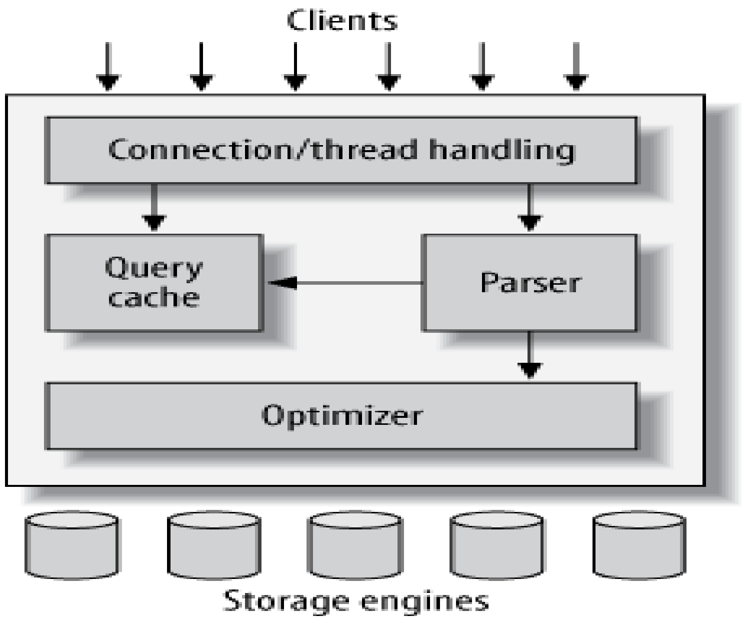


Figure 4.1: Shows a logical view of MySQL's architecture (Safari Books Online, 2019)

.

The top layer is a collection of administrative issues regarding the connection between clients with the server which is not unique to MySQL. it deals on the authentication security and focused on how the second layer contains significant activities of MySQL. Including quite a bit of MySQL's minds are here it comprises the code for query parsing caching searching optimization reserving and all built-in functions e.g. dates times math and encryption any usefulness gave crosswise over storage engine lives at this dimension: handling methodology triggers and perspective. to handle the connection since MySQL is it is a networked-based system. The third layer is responsible for data holding and retrieving data which is stored in the database.

4.1.1.2 Availability

Replication in MySQL is done on the bases of master-slave configurations and master-master configuration, which makes duplicate data distributed from one database to multiple storages slaves. It helps to reduce the workloads of the main server master-slave it is normally done to distribute read access on various servers for availability and also it can be utilized for different purposes it also empowers data accessibility by allowing clients to access the shared data from different slaves.

The master-slave configuration is always work in one direction so the master slave is the only node that is responsible for data write and read operation but the other nodes are dedicated for only reading purpose. Distributing the data on different servers help to improve the performance of the system and data is always available if one data center fails the other will continue to operate the activities. The communication between the master server and the slaves are managed by load manager and finally announce the clients about the current state of the servers. Figure 4.2 shows MySQL replication works.

\

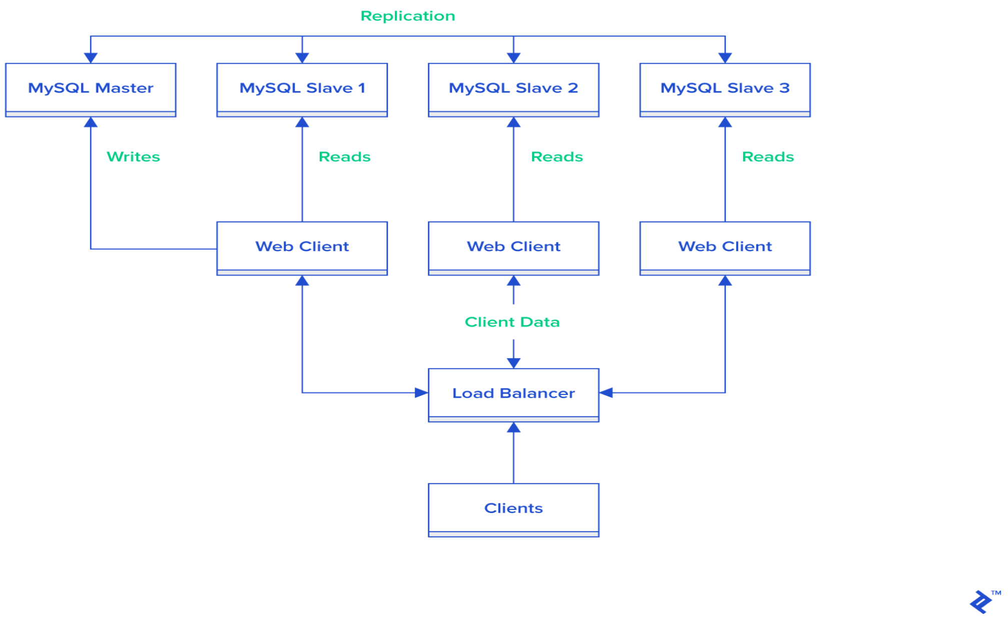


Figure 4.2: MySQL replication example

4.1.1.3 Scalability

MySQL support vertical scaling. Scaling in MySQL is directly related to sharding of data over multiple servers. sharding is commonly used by many databases to share data across the connected servers. and it additionally encourages dynamic failover. Singular shards are comprised of a copy set comprising of no less than two nodes. MySQL uses auto-sharding techniques to guarantee automatic recovery with no single point failure. MySQL shares the read and write workloads over shards in the cluster, enabling every shard to process a subpart of sharded cluster activity.in this case, the workloads of read and write operation are scaling out horizontally in the cluster.

The rows from some random table are straightforwardly part into different sections. For each section, there will be a data center that stores the majority of its data and handles all read and write on that data. Every datum server likewise has a pal and together they structure a hub gathering; the pal holds an auxiliary duplicate of the section just as its very own essential piece.

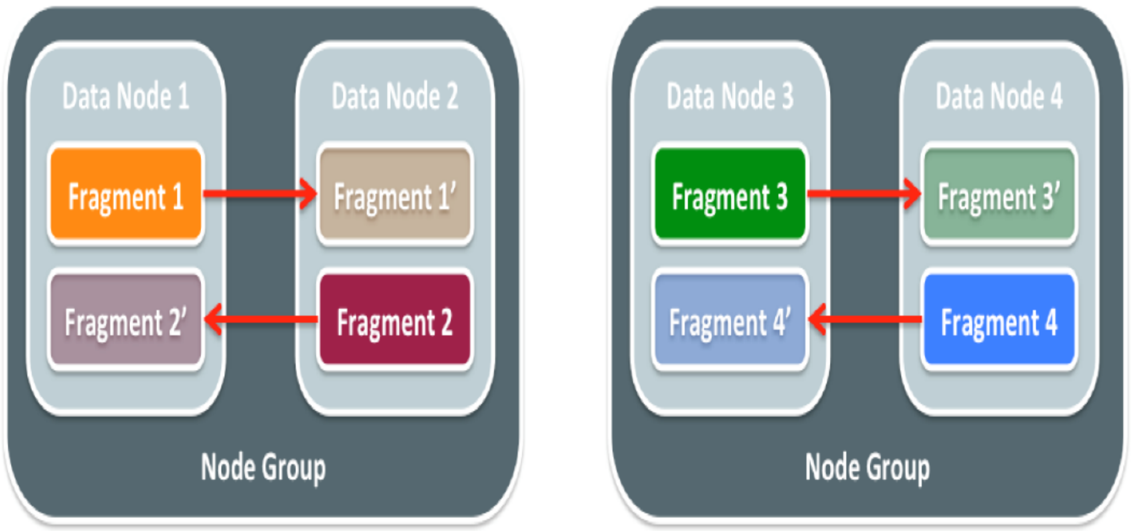


Figure 4.3: MySQL scalability

4.1.2 Supported languages and Platforms

MySQL database supports many structural and object-oriented programming language. Since it is designed for server-client architecture. it supports all web server programming languages and cloud-oriented design and platforms. It also supports many old and new operating systems. programming languages such as Ada, C, C#, C++, Delphi, Java, JavaScript (Node.js), Objective-C, OCaml, Perl, PHP, Python Ruby are supported by MySQL server. FreeBSD, Linux, OS X, Solaris, Windows are some of the operating systems which are supported by MySQL

4.1.2 Documentation, community and support

MySQL community edition is the most downloadable edition of the world's most well-known open source servers. it is accessible under the General Public License (GPA) global permit and is encouraged by an immense and dynamic network of open source designers(Oracle, 2019). The documentation provides the whole bunch of different sorts of executions, operations, drivers, and commands, it also provides consulting and training on new releases and new futures to developers and designers. it provides two licensing futures. open source and commercial license. since MySQL is an open source venture, the complete source code is freely accessible and maintainable regularly. The Open source project is free for all without cost. it is also free for locally used and it provides developers to fix or to implement some futures which are not yet implemented in MySQL or they can request help to the community. commercial license of MySQL requests a payment fee for some futures. Documentation of MySQL server is a broadly acceptable and preferable for developers than the documentation in MongoDB and Cassandra.

4.2 PostgreSQL

PostgreSQL is one of the well-known full featured open source relational database management system. It is written in C programming language and created by PostgreSQL Global Development Group, different organizations and many individual funders(Edition, 2006). It is sometimes categorized as an object-oriented database system that stores data in the form of tables. it is a super implementation of an object-oriented RDBMS which is completely included and allowed to use freely. It can deal with outstanding burdens running from single-machine applications to Web administrations or information warehousing with numerous simultaneous clients and supports many operating systems including macOS, UNIX, FreeBSD, OpenBSD, window. PostgreSQL is the most preferable programs to deal with an extensive amount of data and supporting various datatypes. It is ranked as the 4th most used database system among the entire relational database management system based on the DB-Engines Ranking analysis(solid IT, 2019).

4.2.1 Data model

4.2.1.1 Architecture

In the database world, PostgreSQL works based on client/server architecture like MySQL relational database system(Edition, 2006). its architecture session comprises of three main procedures (programs)

1. **server process:** it is also called Postgres program which deals with the files, acknowledges a connection with the database from users' applications and executes activities in the interest of the users.

2. **client's processor:** it is front-end applications which need to perform operations. Users can be various forms including applications, another server, tools that get to the database to display pages. Some users' applications are incorporated with the PostgreSQL

3. **shard memory:** users' applications send a request to the server. the read and write operation will be processed in the shard memory for performance issues before it processed to or from disk files. Rather, it cushions them in a mutual memory zone which is recognized as the shard memory. figure 4.4 shows how those three components work together in the PostgreSQL database.

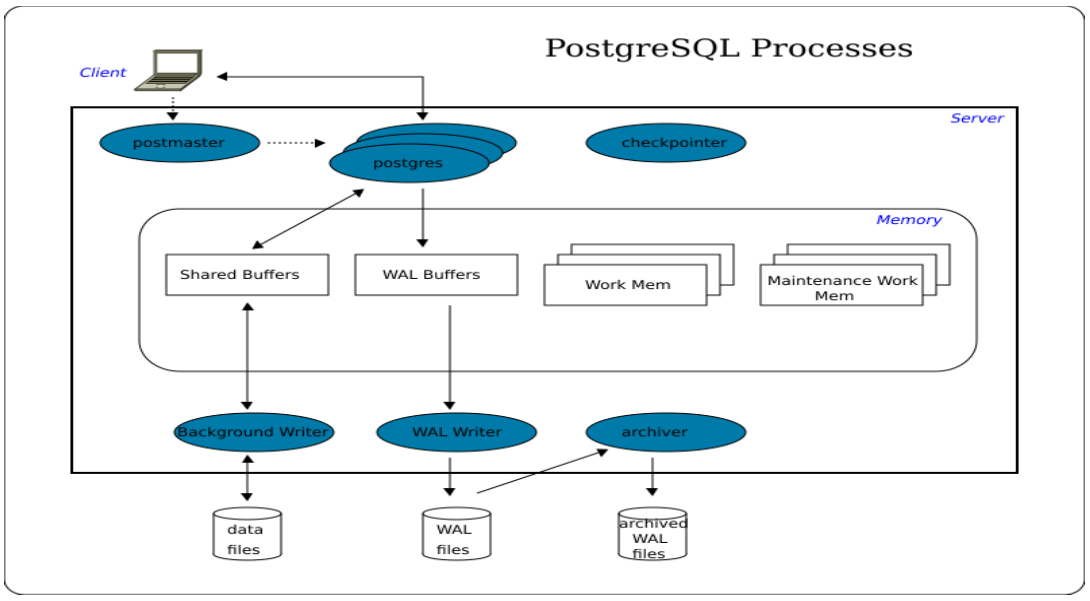


Figure 4.4: Shows architectures of the PostgreSQL database

4.2.1.2 Availability

PostgreSQL supports higher read execution and high-accessibility, by means of a component known as streaming replication. this could be achieved by copying records from the primary database server to other multiple data servers (slave nodes) which would then be able to be utilized as read-only servers. to achieve scalability for reading operations. Rather than using a different framework to implement replication, PostgreSQL usually uses the Write Ahead log to copy the written file from primary node to slave nodes. Write ahead log (WAL) is a sequential instruction that holds the transaction change records for guarantee of atomicity and durability of PostgreSQL. figure 4.5 show the communication between primary node and replica nodes. The primary node is responsible for holding almost all operations including with their schema changes (read, write, delete, and update). The replica node receives instar action from WAL in primary nodes to copy the files throughout the replica nodes and those replica nodes are responsible for read only operations.

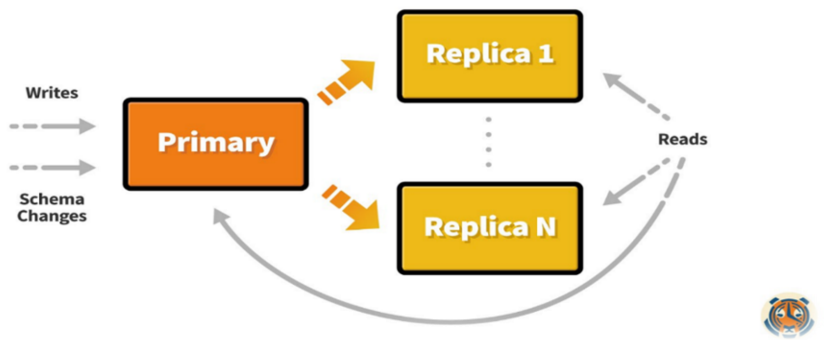


Figure 4.5: Shows replication in PostgreSQL (Timescale, 2018)

4.2.1.3 Scalability

since PostgreSQL is implemented based on a client-server architecture, it supports vertical scaling by adding more hardware resources on the single server to maximize performance issues. PostgreSQL achieved higher scalability on read operation after the release of PostgreSQL version 9.6, it supports parallel processing to achieve higher queries speed (performance) on a single to get a lot quicker while CPUs are expanding in cores instead of raw speed.

4.2.2 Supported languages and Platforms

PostgreSQL supports for many structural and object-oriented programming language. Like another relational database system, it is also designed for server-client architecture. it supports for many web server programming languages and cloud-oriented platforms, operating systems, and web API., programming languages such as C, C++, Delphi, Java info, JavaScript (Node.js), Perl, PHP, Python are supported by PostgreSQL. FreeBSD, HP-UX Linux, NetBSD, OpenBSD, OS X, Solaris, Unix, Windows are among the operating systems which are supported by this server.

4.2.2 Documentation, community and support

PostgreSQL is one of the most supported servers by huge web community. it provides different procedures to assist and consult developers by providing more than twelve mailing services for updating users about the new futures and developments(PostgreSQL, 2019) .

Trainings and consultations are mostly given by active communities around the world with different languages.

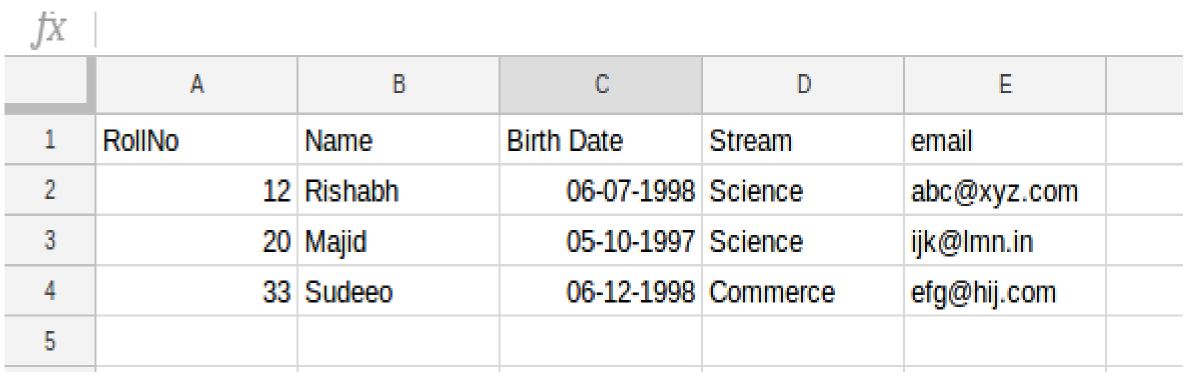
It is free, open source and accessible under the PostgreSQL license global permit and supported by many developers(PostgreSQL, 2019b). its documentation is poor compared with the documentation of MySQL, it provides the current version information such us upgraded futures that are available now and report the progress of those new versions. since PostgreSQL is an open source project, the complete source code is freely accessible and maintainable regularly. The Open source project is free for all without cost. it is also free for locally and web development

Common terminologies of relational database

Although MySQL and PostgreSQL have different behaviors on some parts of data model properties, they have also supported relatively the same data models on the following terminologies.

Database: it is the basic elements in all relational database management system that comprises tables with their rows and fields

Table: the data in a relational database management system is stored in the tables. tables have columns and rows. this makes the relational database system easy to access the data. Figure 4.6 shows how data represent in columns and rows inside the table. Every single row is associated with unique records and the columns represent the single entity values



	A	B	C	D	E
1	RollNo	Name	Birth Date	Stream	email
2	12	Rishabh	06-07-1998	Science	abc@xyz.com
3	20	Majid	05-10-1997	Science	ijk@lmn.in
4	33	Sudeeo	06-12-1998	Commerce	efg@hij.com
5					

Figure 4.6: Relational table example

SQL query language: the acronym SQL stands for Structured Query Language; it is the standard language in all relational database management system so both MySQL and

PostgreSQL support SQL as a query language for data manipulations. SQL is associated with the SELECT, UPDATE, INSERT, and DELETE commands to retrieve data based on the requirements from the data model.

Schema and relationship

Relational database systems are always relying on the relationship between objects or tables. It governs the relationship between 2 or more tables. There are three types of relationship in RDBMS.

- one to one relationship: it refers the association between only two objects
- if one entity associated with the many objects the relationship is called one to many
- If many objects associated with many objects in different group, there relations are called many to many

Figure 4.7 shows the relationship between 3 entity in website forum webpages a single post is associated with many comments and many tag lists

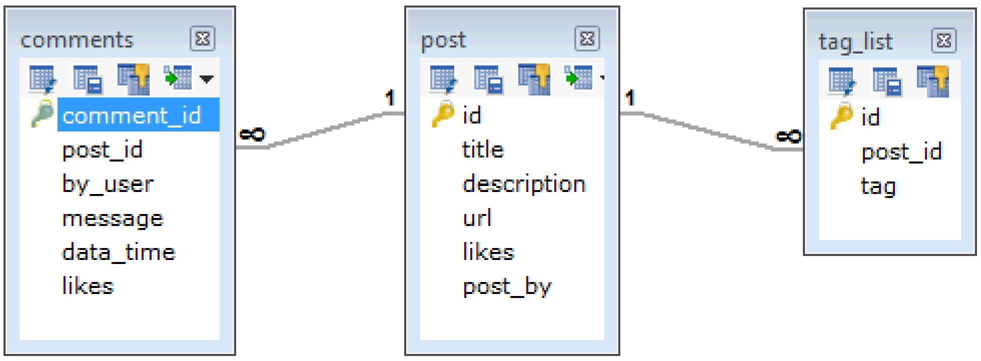


Figure 4.7: Relationship between tables relational databases example

4.3 Cassandra

Apache Cassandra is an open source distributed database system (Hewitt, 2016). It is intended to deal with a lot of information spread crosswise over numerous servers while providing high availability and performance. it was initially created at Facebook in 2008 to control Facebook's in-box search highlight. In the wake of being underway at Facebook for some time, apache Cassandra was settled as an open-source project on Google Code in July of 2008. In Spring of 2009, it was acknowledged to the Apache Establishment as a hatchery project. In February of 2010, it turned into a top-level Apache venture. As of the season of this announcement, the latest release of Apache Cassandra is the 1.2 version. Cassandra has made some amazing progress since the main significant discharge after its advancement to a top-level Apache project. It has grabbed support for Hadoop, content pursuit joining through Solr, CQL, zero-downtime upgrades, virtual nodes. Cassandra is still in consistent overwhelming advancement, and new futures are continually being included and tested (Hewitt, 2016).

Cassandra currently widely accessed by many enterprises, and is usage developing constantly. Business and social network Organizations like Netflix, eBay, Twitter, Reddit, and Ooyala all use Cassandra to control bits of their design, and it is basic to the everyday tasks of those company. To date, the biggest freely realized Cassandra group by machine tally has over 300TB of information traversing 400 machines. Due to Cassandra's capacity to deal with high-volume information, it functions admirably for large number of applications. This implies it's appropriate to taking care of activities from the fast universe of promoting innovation progressively to the high-volume universe of big data analysis and everything in the middle.

4.3.1 Data model

4.3.1.1 Architecture

The data model of Cassandra is fundamentally unique compared to what we typically find in a RDBMS. This part gives a review of how Cassandra handles its data.

Cluster: Cassandra database is distributed on more than a few machines that work together. The outlying compartment is known as the Cluster. For failures taking care of, each node contains a replica, and if there should arise an occurrence of a failure, the copy undertakes responsibility in another node. Cassandra masterminds the nodes in a cluster, in a ring design, and doles out data to them.

Keyspace: A cluster is a compartment for keyspaces ordinarily a single keyspace. A keyspace is the outlying compartment for data in Cassandra, matching nearly to a relational database. Like relational database, a keyspace has a name and attributes that characterize keyspace-wide conduct. The fundamental properties of a Keyspace in Cassandra are listed next. Figure 4.8 shows keyspace architecture in Cassandra.

Replication factor: in most straightforward terms, the replication factor indicates to the quantity of nodes that will demonstration as duplicates (replicas) of each rows of data. In the event that your replication factor is 4, at that point four nodes in the ring will have duplicates of each line, and this replication is straightforward to clients. The replication factor basically enables you to choose the amount you need to pay in execution to acquire consistency. That is, your consistency level for perusing what's more, composing data depends on the replication factor.

Replica placement strategy: It is only the procedure to put replications in the ring. We have procedures, for example, simple strategy (rack-mindful system), old network topology strategy (rack-mindful procedure), and system topology system (datacenter-shared system) (Hewitt, 2016).

Column families: Similarly, that a database is a compartment for tables, a keyspace is a holder for a column family. A column family is generally equivalent to a table in the relation model, and is a compartment for a gathering of lines. Each line contains requested columns. Column families characterizes the structure of your data. Each keyspace has no less than one and regularly numerous column families. Cassandra characterizes a column family to be a bright division that associates analogous data (Hewitt, 2016).

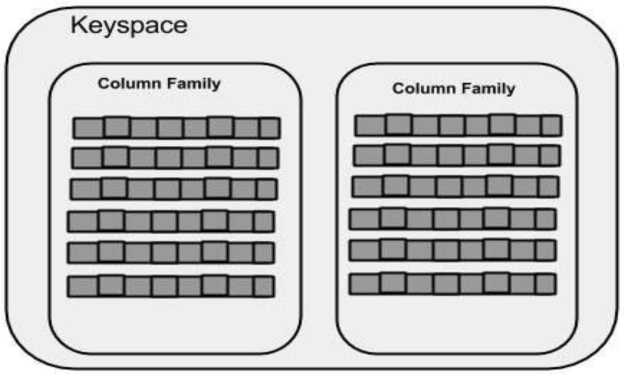


Figure 4.8: Keyspace architecture in Cassandra(Hewitt, 2016)

Assembling many column families, allow us to get the fundamental Cassandra data structures: the column, in relational databases, we're accustomed to putting column names as strings just—there's nothing more to it we're permitted. In any case, in Cassandra, we don't have that constraint. Both row keys and column names can be strings, as relational names, yet they can likewise belong numbers, UUIDs, or any sort of byte array. So, there's some assortment to how your key names can be set. This uncovers another fascinating quality to Cassandra's columns: they don't need to be as basic as predefined name/values sets; you can store helpful data in the key itself, not just in the value. This is fairly regular while making records in Cassandra. Figure 4.9 show how data represent as a column family in Cassandra databases.

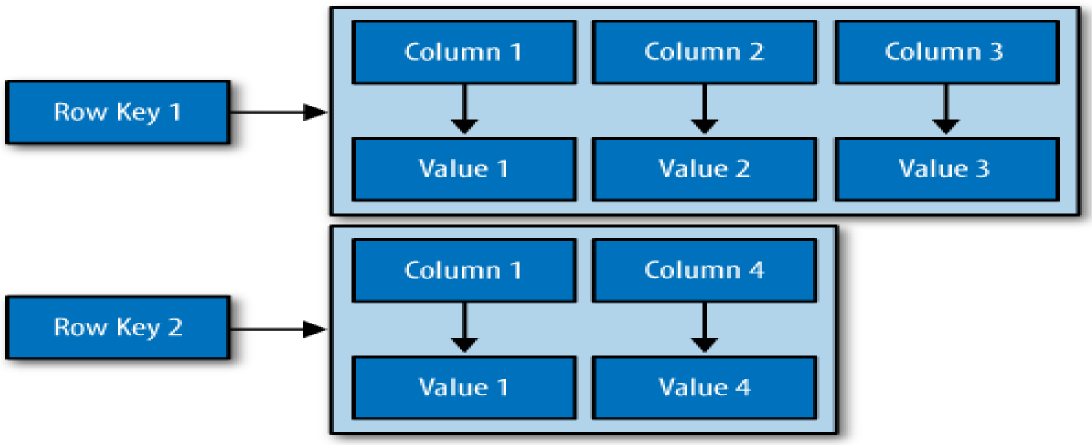


Figure 4.9: A column family (Hewitt, 2016)

4.3.1.2 Availability

Ring Structure: Cassandra clusters work in a Ring fashion, which is in a peer to peer architecture. every node has equal responsibility in the connection, as it were, all nodes are the equal and there are no central nodes that control different nodes (Hewitt, 2016). In this manner, there is no single point of failure due to the replication of data on different nodes. Cassandra, additionally, come up with adaptable replication system, which makes it to handles excess duplicates of data, crosswise over nodes, this implies if any node in the cluster fails, at least one duplicates of that server's data is accessible on different machines in the cluster. Additionally, this replication mechanism works in a big data center, data could be duplicated over multiple data center too and accessible in any one of the data centers when frailer is happening in the ring. Figure 4.10 show how Cassandra distribute all data to individual connected servers.

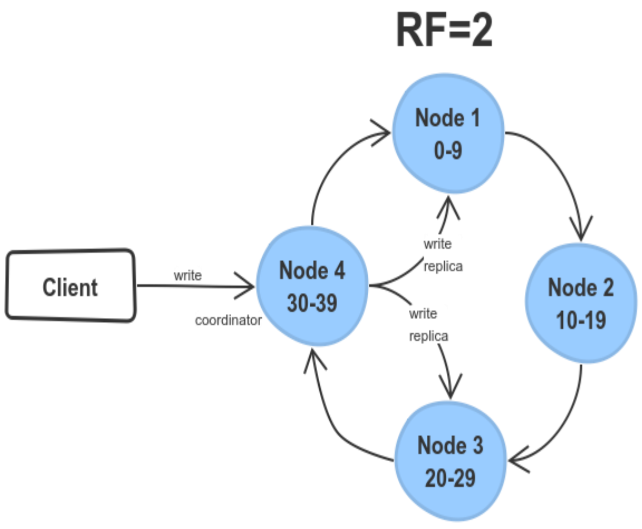


Figure 4.10: Cassandra ring replication example (DZONE, 2016)

Gossip and Failure Detection: To help decentralization and partition tolerance, Cassandra utilizes a Gossip protocol for entire node communication in the cluster with the goal that every node can have state information about other nodes. The gossiper runs each second on a clock.

4.3.1.3 Scalability

Cassandra is predominantly guaranteeing availability and linear scalability at a higher level. It implies that scalability can be expanded by adding new servers/resources or nodes in the cluster. Cassandra supports both scaling up and scaling out. This makes Cassandra structure truly versatile and profoundly accessible. This distributed architecture Improves the general versatility since every one of the nodes in a cluster can serve read or write operations and it promises availability almost 99%. Since all nodes can serve read and compose demands, scaling the framework is as basic as adding new nodes to the group. This makes Cassandra flexibly adaptable in both vertical and horizontal scaling.

Figure 4.11 shows the scalability in Cassandra, the first 2 hubs can deal with 1000 transactions for each second, the second 4 hubs will bolster 2000 transactions/sec and the third 8 hubs will handle 4000 transactions/sec:

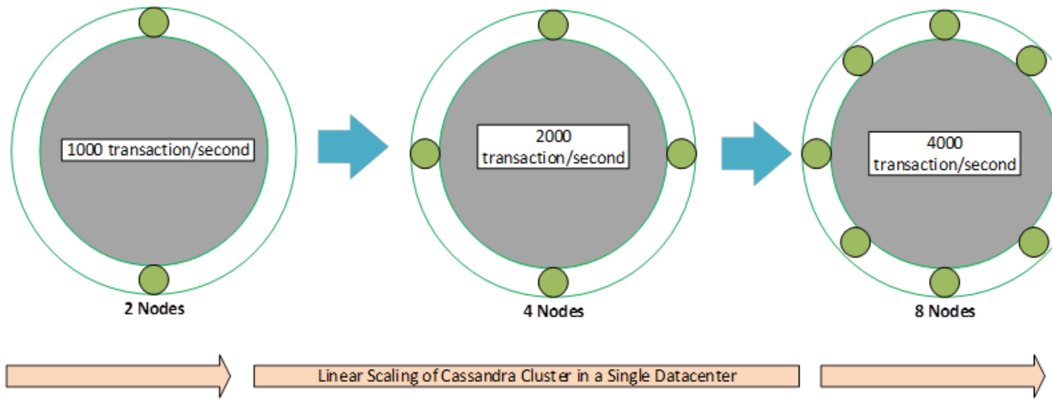


Figure 4.11: *Cassandra scalability example*(RapidValue, 2015)

CQL: it endeavors to be as a near structural query language could reasonably be expected. Given that Cassandra is a NoSQL database management system, a completely included SQL build is beyond the realm of imagination. The primary concern to note about CQL is that it has no idea of GROUP or JOIN, and an extremely restricted usage of ORDER BY. Clients can get to Cassandra through its nodes accessing Cassandra Inquiry Language (CQL). CQL treats the Keyspace as a holder of tables. Software engineers use cqlsh, a command prompt to deal with CQL or along with separate programming language drivers.to make a read and write operation we use a node (facilitator) play an intermediary between the users and the nodes holding the data.

Data types: CQL gives a multi collection of built-in data types, including collection types (list, map, set), UUID, timestamp. Combined with these data types, clients can likewise make their very own custom data types.

Write data: Each writes operations of the server are caught by the commit logs written in the nodes. after that, the data will be caught and put in the mem-table. if the mem-table is full, data will be composed into the table data files. All are dynamically partitioned and reproduced all through the cluster.

Data read: In read actions, Cassandra gets values from the mem-table and checks the blossom filter to locate the suitable SSTable that stores the appropriate data.

4.3.2 Supported languages and Platforms

Cassandra supports for many programming languages. it supports for many web server programming languages and cloud-oriented platforms, operating systems, programming languages such as C#, C++, Java, JavaScript info, Perl, PHP, Python, Ruby are supported by MongoDB. BSD, Linux, OS X, Windows are among the operating systems which are supported by this server.

4.3.2 Documentation, community and support

Cassandra support is provided by Datastax and other companies to assist the developers. The documentation provides about the whole bunch of concepts about the data model, query model conflagration and tools of Cassandra (Apache Cassandra, 2016). Due to the Cassandra has multiple advantages on modern web technologies by processing and holding a large amount of information, many users and technology groups are open their eye to Cassandra. The development of Cassandra is radical and supported by active communities. It provides a platform for training, discussion forum, and mailing service to assist users and developers to know how to design and architecture non-relational databases for dominating web industry. since Cassandra is an open source distributed database system, the complete source code is freely accessible and maintainable regularly (Apache Cassandra, 2016).

4.4 MongoBD

MongoDB is one of the leading NoSQL storage advances which emerged in the mid-2000s with the ability of advanced scale out features.it is written in C++ programming language (solid IT, 2019). MongoDB is an open source non-relational database management framework (DBMS) that utilizes A document-based database program which runs different types of data (Chodorow, 2013). additionally, MongoDB is promptly interesting, not only in light of its scaling procedure but instead in light of its instinctive data modeling. document-based data model makes it preferable to manipulate complex, unstructured data without the needs of tables and join operation which is mostly available in almost all traditional relational databases management system. Data representation procedure in MongoDB is very simple and easy for almost all types of data compared with other database systems. This is due to the help of its document-based nature of data model for storing different types of data (structured or unstructured). the idea of rows in the traditional database system is changed into a document in MongoDB which is highly dynamic, schema-free and conceivable to store complex and various leveled data with a single record (Chodorow, 2013) .

4.4.1 Data Model

Database: a database in MongoDB is a compartment for collections. A single server frequently has various databases.

Collections: collections are the group of documents in MongoDB which is equivalent to table in traditional relational database management system. figure 4.12 shows the how documents represented in a single collection.

Documents: document in MongoDB is equivalent to rows in the relational database system. MongoDB stores data in BSON format which is quick and exceedingly navigable. This implies MongoDB gives clients the convenience and adaptability of JSON documents together with the speed and powerful lightweight binary object (Chodorow, 2013).

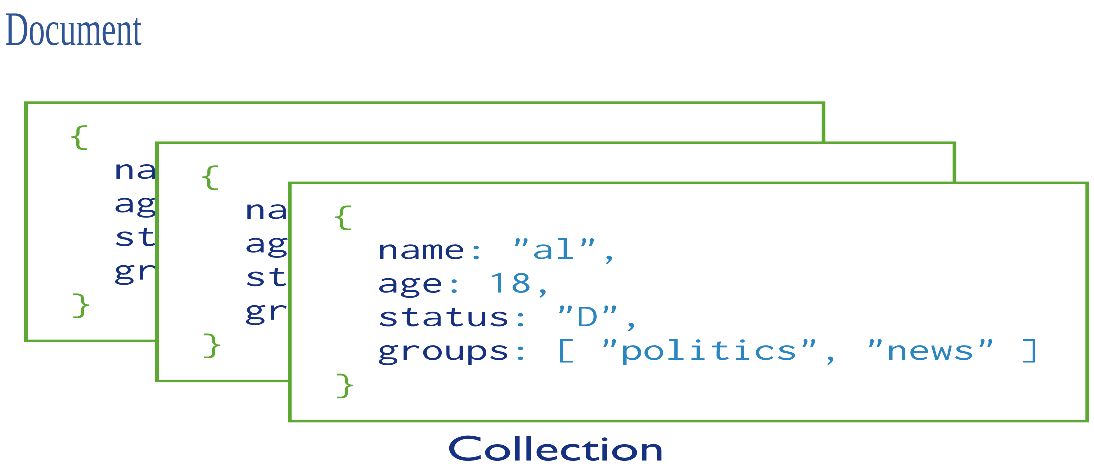
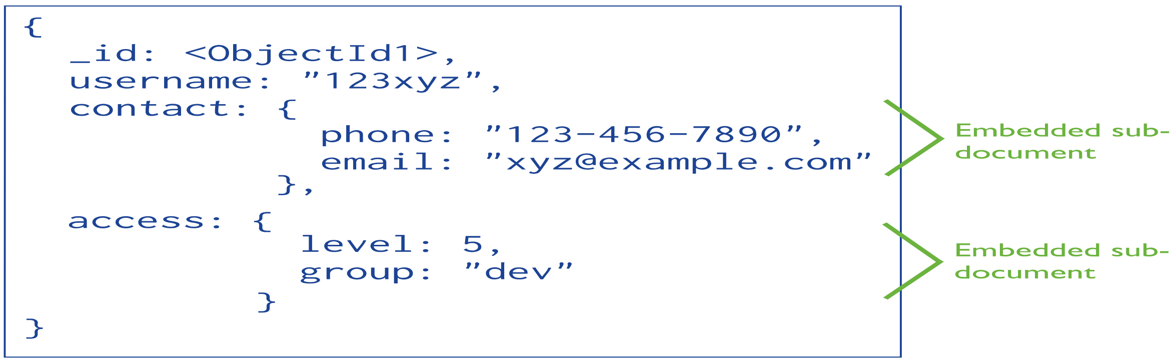


Figure 4.12: MongoDB collection example (MongoDB, 2008a)

Embedded documents

Since MongoDB does not support JOIN operation like other relational database management system. It has other means of representing multiple embedded objects in the documents to make relationship between those documents. This embedded document is stored in the form of arrays and can be easily retrieved by simple query operation. Figure 4.13 show the denormalized embedded documents stored in a single document



```
{
   _id: <ObjectId1>,
   username: "123xyz",
   contact: {
               phone: "123-456-7890",
               email: "xyz@example.com"
             },
   access: {
              level: 5,
              group: "dev"
            }
}
```

Figure 4.13: Denormalized document structure (MongoDB, 2008a)

4.4.1.2 Availability

MongoDB supports data availability using data replication techniques which is always done based on the master-slave configuration like in traditional relational database management system MongoDB manages database replication by using a topology called replica set. The data spread over all nodes in the network by replica set for redundancy and dynamic availability of the replica data on the server. Furthermore, replication is utilized to scale database reads. Specially for read concentrated web application, as is generally the situation on the web, it's conceivable to spread database reads over machine in the replicas set cluster. always the replica set comprises of relatively one primary node and many more secondary nodes in the network. Primary node has permission to accessing both read and write, but secondary nodes are read only. in the event that the primary node stops working or having a failure to continue, the cluster will select and assign one node as a primary node from secondary nodes and consequently advance it and continue the operations. At the point when the previous primary node avoids failure and returns on normal working condition, it will be used as a secondary node since the time the location of the primary nodes has taken by another nodes. figure 4.14 shows how replication works in MongoDB

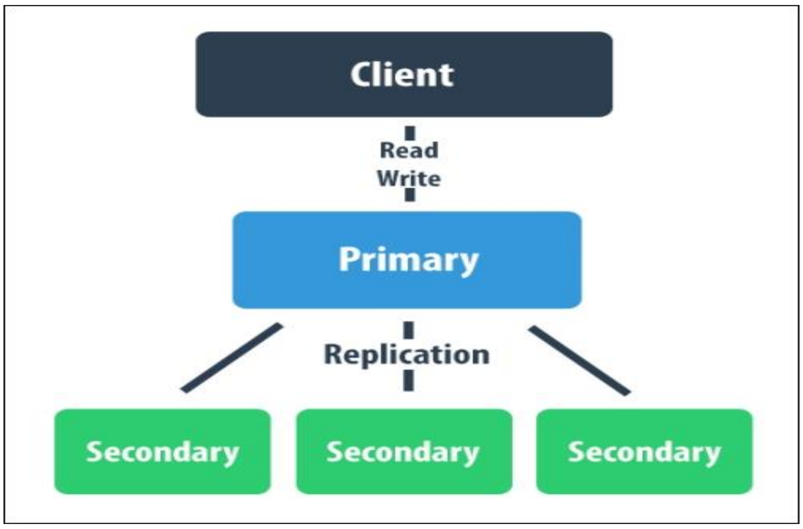


Figure 4.14: Replication in MongoDB(MongoDB, 2008b)

4.3.1.2 Scalability

MongoDB support horizontal scaling. Scaling in MongoDB is directly related to sharding of data over multiple servers. sharding is commonly used by many databases to share data across the connected servers. and it additionally encourages dynamic failover. Singular shards are comprised of a copy set comprising of no less than two nodes. MongoDB uses auto-sharding techniques to guarantee automatic recovery with no single point failure

MongoDB shares the read and write workloads over shards in the cluster, enabling every shard to process a subpart of sharded cluster activity.in this case, the workloads of read and write operation are scaling out horizontally in the cluster. Figure 4.15 shows how sharding works in the database with the combination of sharded and unsharded cluster. ShardA has two collections, collection 1 and collection2. Collection1 is distributed over shardA and shardB. Both read and write is possible for collection1 but shardB has no permission to read and write for collection2.

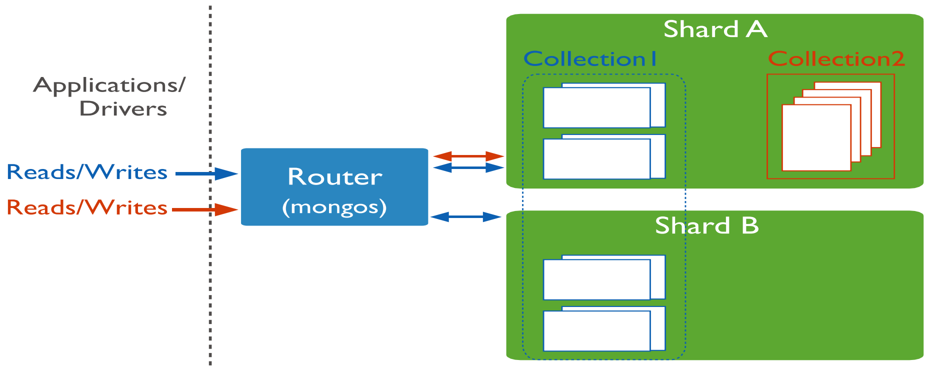


Figure 4.15: MongoDB scalability example(MongoDB, 2008b)

4.3.2 Supported languages and Platforms

MongoDB supports for many programming languages. this database is more compatible with many programming languages than all non-relational databases.it supports for many web server programming languages and cloud-oriented platforms, operating systems, programming languages such as C, C#, C++, D info, Dart info, Java, JavaScript, MATLAB, Perl, PHP, PowerShell info, Python R info, Ruby, Scala, Smalltalk info are supported by MongoDB. Linux, OS X, Solaris, Windows are among the operating systems which are supported by this server.

4.3.2 Documentation, community and support

MongoDB documentation is large and easy compared with Cassandra documentation. It provides deep explanation about the configuration process, libraries, drivers required to manage MongoDB servers(MongoDB, 2019). the documentation also includes all operations with specific examples and use cases to clarify every part of the server. this makes MongoDB more preferable by many developers. It is also supported by many active community and organization. it is accessible under the General Public License (GPA) global permit and is encouraged by immense and dynamic network of open source designers (MongoDB, 2019)

CHAPTER 5

IMPLEMENTATIONS

This chapter deals on implementation of CRUD test for MYSQL, PostgreSQL MongoDB and Cassandra databases. the Test is designed to compare the databases based on computing the execution time taken to perform the CRUD operation on the individual database. the test is incorporated with CRUD commands to manipulate the data to the given database

Test Environment

The experimental platform used for implementing those databases is a 1.6GHz processor with 4 GB memory and 500GB SATA disk drive. The operating system is MacOS Mojave 10.14.

Database software version.

Comparison among database are done based on the current versions. table 5.1 summaries the versions of the database deployed for testing.

Table 5.1 : Database versions

Database name	Version
MYSQL	10.0.4
PostgreSQL	11
Cassandra	3.11.4
MongoDB	4.0.6

5.1 Query Execution Time

For testing purpose, we use UNIX timestamp to measure the time taken by each CRUD operation and the time is represented in seconds. It is essential time estimation techniques applied to manage the execution of code in the program and helps programmers to get the exact amount of time taken by each database to perform all four operations. appendix 1 shows the time counter code that manages the execution of each query in the implementation of database server and the execution of all queries beneath relies upon evaluating and estimating execution time using the strategies depicted here.

5.2 MySQL And PostgreSQL Query

This part represents both MySQL and PostgreSQL query used for testing query capabilities. the execution time of all operations for both databases are executed using PHP programming language in appendix 2 and appendix 3 respectively

Create Table Query

```
" CREATE TABLE IF NOT EXISTS data_table2 (
            Id int(11) NOTNULL,  auto-increment,
            customer_name  VARCHAR(25),
            customer_lastname VARCHAR(25),
            customer_address VARCHAR (25),
            phone INT,
            email VARCHAR (25),
            zip INT,
            state  VARCHAR (25),
            country VARCHAR (25),
            car_model VARCHAR (25),
            car_price double,
            PRIMARY KEY (id))";
```

Insert Query

```
"INSERT INTO data_table2 (customer_name,customer_lastname,customer_address, phone, email, zip, state, country, car_model, car_price) VALUES ('Saron', 'Addal','Addis Ababa', '200195838' , 'Saron0918gmail.com','34533', 'Amhara', 'Ethiopa', '2018 Toyota Avalon', '250000');"
```

"INSERT INTO data_table2 (customer_name,customer_lastname,customer_address, phone, email, zip, state, country, car_model, car_price) VALUES('Helen', 'Danial','bahirdar', '200195838','helene123gmail.com','1186', 'Amhara', 'Ethiopa', '2018 Toyota Avalon', '30000');"

Select Query

'SELECT * FROM data_table2 WHERE id=1232 ';

'SELECT * FROM data_table2 ';

Update Query

"UPDATE data_table2 SET customer_name='Wondwessen', customer_lastname='Haile', customer_address ='Nicosia', phone='53382238', email='wondwessen@gmail.com', zip='4556', state='North Cyprus', country='Turkey',car_model='2011 Lexus RX 350',car_price='3002330' WHERE id=12333");

"UPDATE data_table2 SET customer_name='biruk',customer_lastname='addal', customer_address='Paris',phone='53382238',email='biruk333434@gmail.com',zip='5552' ,state='NorthCyprus',country='Turkey',car_model='1999toyota350',car_price='5700000' WHERE id=1667");

Delete Query

' DELETE * FROM data_table2 WHERE id=1232';

' DELETE * FROM data_table2 WHERE id <= 1232';

5.3 MongoDB Operations

This part represents MongoDB query used for testing query capabilities. the execution time of all operations for MongoDB is executed using PHP programming language in appendix 4

Create collection

$client = new MongoDB\client;

$database = (new MongoDB\Client)->customerinfo;

$database->customer;

Insert Query

insertOne(['customer_name'=>'wondwessen','customer_lastname'=>'haile','customer_address'=>'north_cyprus','phone'=>5338223089,'email'=>'wondwessen533@gmail.com','zip'=>1234,'state'=>'northCyprus','country'=>'turkey','car_model'=>'Toyota2010','car_price'=>23434]);

```
insertOne(['customer_name'=>'wondwessen','customer_lastname'=>'haile','customer_address'=>'north_cyprus','phone'=>5338223089,'email'=>'wondwessen533@gmail.com','zip'=>1234,'state'=>'northCyprus','country'=>'turkey','car_model'=>'Toyota2010','car_price'=>23434]);

insertOne(['customer_name'=>'wondwessen','customer_lastname'=>'haile','customer_address'=>'north_cyprus','phone'=>5338223089,'email'=>'wondwessen533@gmail.com','zip'=>1234,'state'=>'northCyprus','country'=>'turkey','car_model'=>'Toyota2010','car_price'=>23434]);
```

Select Query

```
$customercollection->find();
```

Update Query

```
$customercollection>updateMany(['$set'=>['customer_name'=>'Anteneh','customer_lastname'=>'addal','customer_address'=>'addisababa','phone'=>'9182','email'=>'yohanse@gmial.com','zip' => '1322','state' => 'amhara','country' => 'Ethiopia','car_model' => '2011 Lexus RX 350', 'car_price' =>'3233'] ]);
```

Delete Query

```
$customercollection->deleteMany();
```

5.4 Cassandra Operation

This section represents Cassandra query used for testing query capabilities. the execution time of all operations for Cassandra is executed using PHP programming language in appendix 5.

Create Table Query

```
"CREATE TABLE IF NOT EXISTS data_table5 (id SERIAL PRIMARY KEY, customer_name VARCHAR(25),
        customer_lastname VARCHAR(25),
        customer address VARCHAR(25),
        phone INT,
        email VARCHAR(25),
        zip INT,
        state  VARCHAR(25),
        country VARCHAR(25),
        car_modal VARCHAR(25),
```

car_price INT)";

Insert Query:

```
" INSERT INTO data_table5 (customer_name,customer_lastname,customer_address, phone, email, zip, state, country, car_modal, car_price) VALUES ('Saron', 'Addal','Addis ababa university', '200195838' , 'Saron0918gmail.com','34533', 'Amhara', 'Ethiopa', '2018 Toyota Avalon', '250000');");
```

```
" INSERT INTO data_table5 (customer_name,customer_lastname,customer_address, phone, email, zip, state, country, car_modal, car_price) VALUES ('mohammed', 'Hassen','Addis ababa university', '200195838' , 'Saron0918gmail.com','34533', 'Amhara', 'Ethiopa', '2018 Toyota Avalon', '250000');");
```

Select Query

```
SELECT * FROM data_table5;
```

Update Query

```
"UPDATE data_table5 SET customer_name='Wondwessen',customer_lastname='Haile', customer_address='Nikosia',phone='53382238',email='wondwessen@gmail.com',zip='43 4556', state='NorthCyprus',country='Turkey',car_modal='2011 Lexus RX 350', car_price='3002330'"
```

```
"UPDATE data_table5 SET customer_name='saron',customer_lastname='Haile', customer_address='Nikosia',phone='53382238',email='wondwessen@gmail.com',zip='43 4556', state='NorthCyprus',country='Turkey',car_modal='2011 Lexus RX 350', car_price='3002330'"
```

Delete Query

```
DELETE FROM customerinfo.customer_name;
```

5.5 Testing Results

Testing was implemented based on batch processing on a single node and it follows serial way: INSERT, SELECT UPDATE, DELETE. Each batch of tests was run 5 times for 1000, 5000, 10000,20000,40000,60000,80000 and 100000 records. The performance of the database will measure based on the average execution time of each operation correspond to each database. Lastly, we presented the performance of each database based on graphs and tables

5.5.1 Result for INSERT Operation

Table 5.2 show the average execution times of insert operation for each database server in single node. it shows the number of operations with the time taken to execute by the databases servers of MYSQL, PostgreSQL, MongoDB and Cassandra respectively. Based on this analysis, the performance of non-relational databases is more powerful than relational database management system. among all database servers, the performance of MongoDB for insert operation is higher than all other database servers.in the reverse, the performance PostgreSQL databases for insert operation is slower than all database servers. in this case the performance of Cassandra database server is relatively close to MongoDB database server compared with MySQL and PostgreSQL databases. Figure 5.2 shows the performance of all databases for all records. based on this analysis the rank of those databases is shown here

1. MongoDB
2. Cassandra
3. MYSQL
4. PostgreSQL

Table 5.2: Average Execution Time in seconds for the INSERT Operation

Number of Records	**MYSQL**	**PostgreSQL**	**MongoDB**	**Cassandra**
1000	0.339	0.5	0.287	0.316
5000	1.628	2.079	0.995	1.577
10000	2.986	4.192	2.134	3.038
20000	6.01	8.586	4.159	5.732
40000	12.651	17.501	8.286	10.859
60000	19.105	24.515	12.132	15.631
80000	26.686	33.203	17.937	23.585
100000	32.243	42.282	19.778	26.509

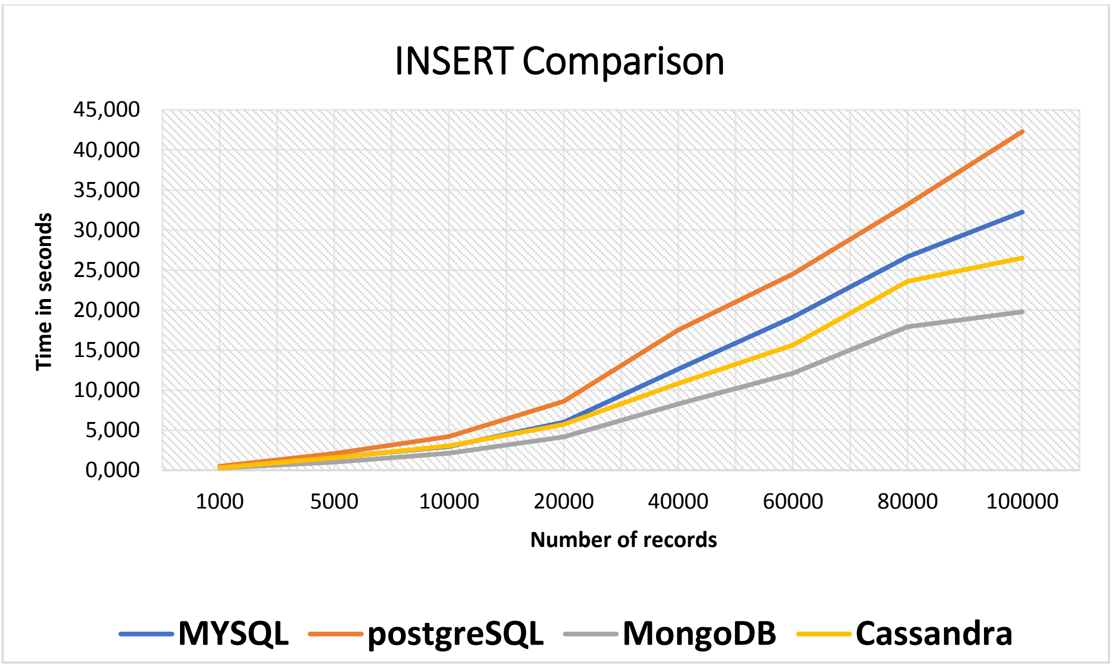


Figure 5.1: Average performance of INSERT operation in seconds

5.5.2 Result for SELECT Operation

The SELECT average time (table5.3) execution times of select operation for each database server in a single node. it shows the number of operations with the time taken to execute by the databases servers of MYSQL, PostgreSQL, MongoDB and Cassandra respectively. in this analysis, the performance of non-relational databases is also more powerful than relational database management system. among all database servers, the performance of MongoDB for select operation is higher than all other database servers.in the reverse, the performance PostgreSQL databases for insert operation is slower than all database servers**.** In this case the performance of MongoDB is twice higher than the performance PostgreSQL databases. Cassandra and MySQL database have relatively similar performance for 1000-20000 number of records. Figure 5.3 shows the performance of all databases for all records. based on this analysis the rank of those databases is shown here.

1. MongoDB
2. Cassandra

3. MYSQL
4. PostgreSQL

Table 5.3: Average Execution Time in seconds for the SELECT Operation

Number of records	MYSQL	PostgreSQL	MongoDB	Cassandra
1000	0.345	0.501	0.308	0.327
5000	1.657	2.082	1.034	1.621
10000	3.044	4.193	2.229	3.137
20000	6.132	8.593	4.313	5.875
40000	12.871	17.502	8.591	11.013
60000	19.466	24.527	12.61	15.744
80000	27.155	33.204	17.374	23.913
100000	32.846	42.283	20.555	26.630

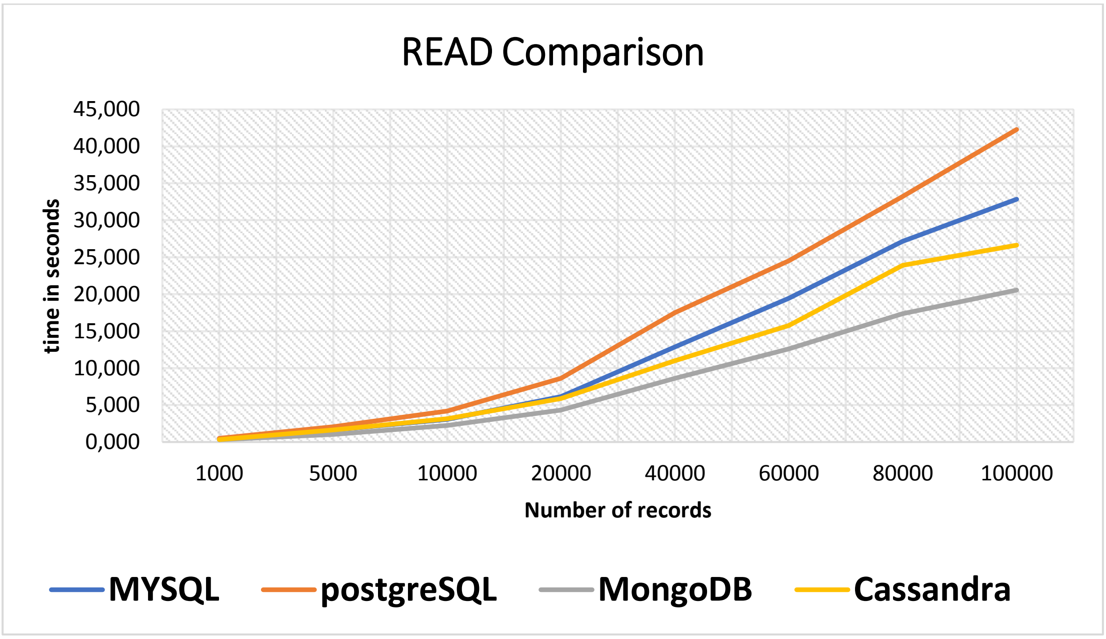


Figure 5.2: Average performance of the SELECT operation in seconds

5.5.3 Result for UPDATE Operation

Table 5.4 presents the performance result of all databases corresponding with a number of records. The same way like insert and select operations, the performance of PostgreSQL for update operation is lower than all other database servers, although it has relatively similar performance with MySQL database server. The performance of MongoDB and Cassandra have almost the same performance for all number of records. The different thing here is the performance of Cassandra for update operation is higher than its performance for insert, select and delete operation. Figure 5.3 shows the performance of all databases for update operation. based on this analysis, the rank of those database divided in two groups.

For inputs less than 40000 number of records is

1. Cassandra
2. MongoDB
3. MYSQL
4. PostgreSQL

For inputs greater than 40000 number of records

1. MongoDB
2. Cassandra
3. MYSQL
4. PostgreSQL

Table 5. 4: Average Execution Time in seconds for the UPDATE Operation

Number of records	MYSQL	PostgreSQL	MongoDB	Cassandra
1000	0.387	0.511	0.381	0.328
5000	1.746	2.124	1.288	1.623
10000	3.204	4.268	2.808	3.139
20000	6.931	8.798	5.343	5.877
40000	14.533	17.970	11.036	11.015
60000	21.989	25.361	15.989	15.745
80000	30.769	34.233	22.405	23.915
100000	37.254	44.059	26.261	26.632

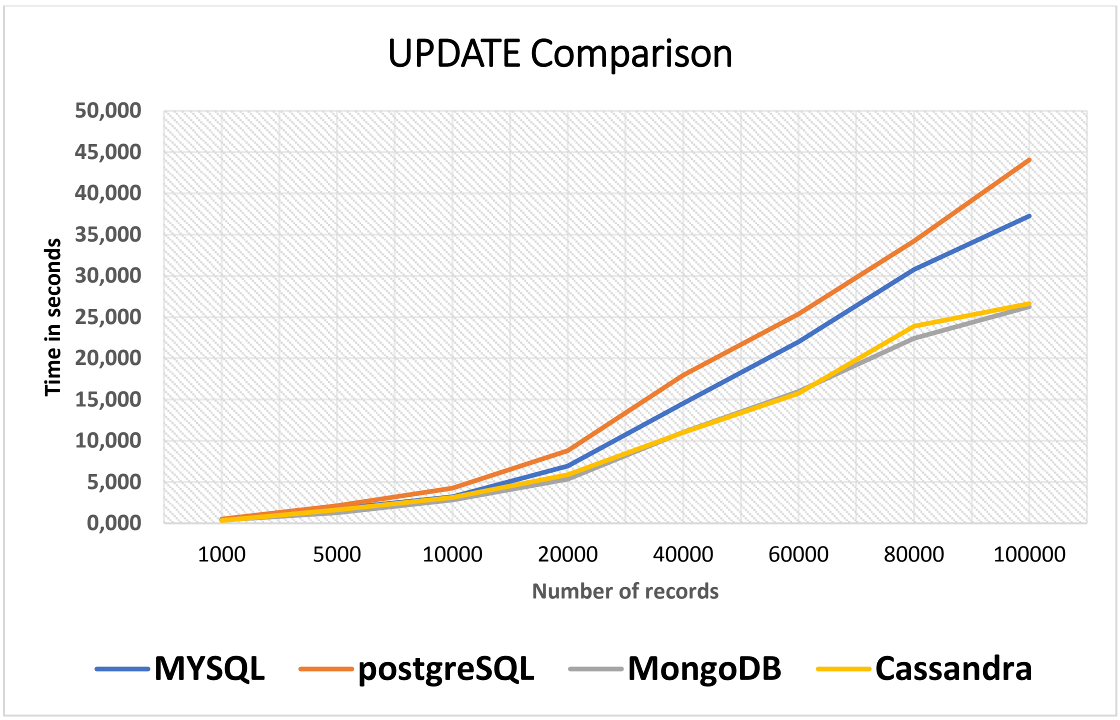


Figure 5.3: Average Performance of the UPDATE Operation in Seconds.

5.5.4 Result for DELETE Operation

Among all database servers, Cassandra has highest performance for the DELETE operation (Figure 5.5) In this case, MySQL, Cassandra and MongoDB have relatively similar performance. the performance of PostgreSQL for delete operation is worse than all operations. Almost all database servers have took similar execution time for records less than 1000. Table 5.4 shows the average execution time of all databases for delete operation. based on this analysis the rank of those databases is shown here.

1. Cassandra
2. MongoDB
3. MYSQL
4. PostgreSQL
5.

Table 5. 4: Average Execution Time in seconds for the DELETE Operation

Number of operations	MYSQL	PostgreSQL	MongoDB	Cassandra
1000	0.404	0.513	0.415	0.490
5000	1.783	2.131	1.418	1.913
10000	3.260	4.283	3.153	3.331
20000	7.026	8.822	5.844	6.105
40000	15.203	18.014	12.196	11.290
60000	22.547	25.490	17.588	16.013
80000	31.971	34.328	26.552	24.300
100000	38.667	43.568	29.581	26.934

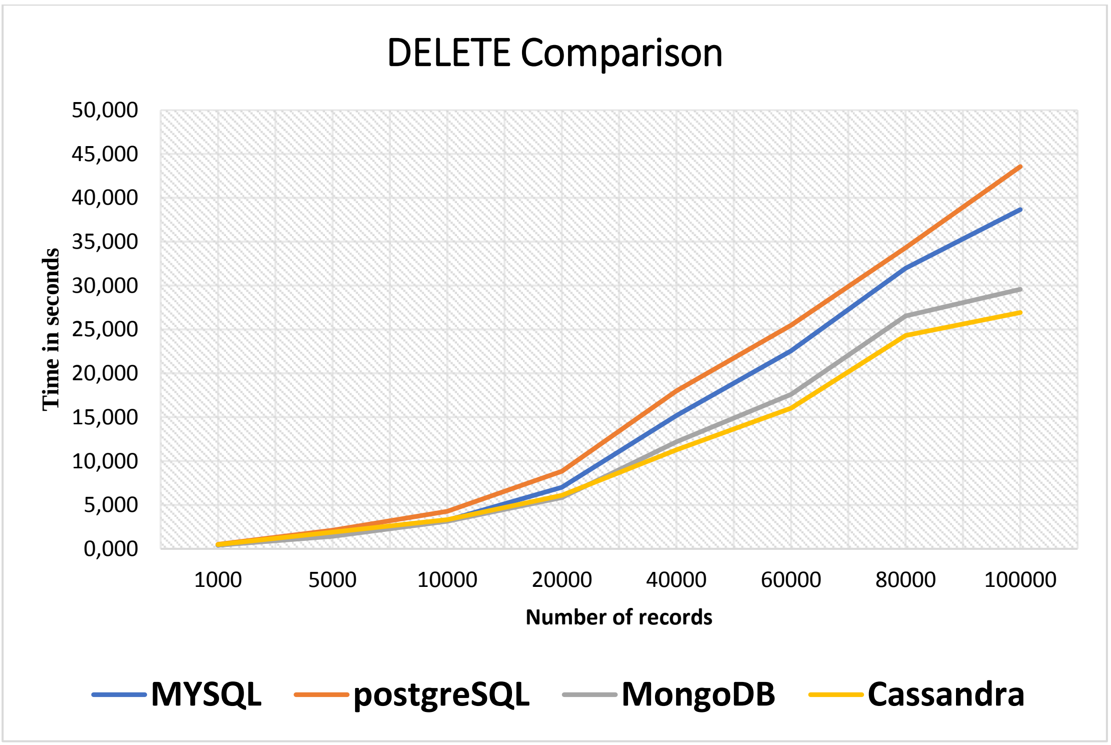


Figure 5.4: Average performance of the DELETE operation in seconds

CHAPTER 6

EVALUATION

chapter 4 and 5 presented about the data model and performance of those 4 databases for CRUD operations as frameworks together with their basics and highlights.in this part, we assess our examination work based on the previous chapters along with new concepts and terminologies regarding the qualities and shortages of all database servers in the web industry. Lastly, we examined all database systems from their community support and area of used.

6.1 Data Model

a data model demonstrate indicates to the legitimate between connections along with data stream among various information components engaged with the different enterprises and organizations.dat model also mainly used as a technique that governs how the given data should be handled and manipulated in different storage systems. data models guarantee strong reliability for various organizations who needs specialized advancement to make the requirements managed properly and matched with the goal of the business. data models assist developers and business by facilitating the communication between the cooperated team and available resources.

As we have seen in chapter 4, MySQL has derived from the SQL standards and trends, new variances in the way certain queries operate on multiple systems. it stores and retrieve data on structured tables, which has columns and rows. it encourages fixed schema since the design of the database depends on tables. MySQL supports complex query in the form of joining multiple tables and making relationship between complex entity’s based on entity relationship model. this leads the performance problem compared to non-relational database management system which supports dynamic schema and few relationships. the data is structured in the form of a spreadsheet. And it supports master-slave configurations and master- master configuration to achieve high availability. It supports vertical scalability and used sharding to achieve scalability. Read operation is highly scalable than write operation. the primary server has ability to provides read and write operation all the time. But Slave or secondary nodes can only access the read operations.

PostgreSQL: is a strong ACID compliant, ACID property is a standard for whether or not the data integrity is applied on across all types of queries and robust enough in order to return the same results without false information. Like other relational database, PostgreSQL is a fixed schema-based database system that stores data in the form of spreadsheet. Complex queries also manipulated by GROUP and JOIN operation this makes to sacrifices it performance quality. only supports master-slave configuration to achieve higher availability. unlike MYSQL, PostgreSQL usually uses the Write Ahead (WAL) log to copy the written file from primary node to slave nodes. the primary server has ability to provides read and write operation all the time. But Slave or secondary nodes can only access the read operations.

MongoDB: It is a document-oriented database management system that stores data in the form of documents. Unlike all relational database system uses SQL languages, MongoDB uses Binary structure object notation (BSON) which is extended from JSON to store and retrieve documents. MongoDB uses free and dynamic schema to make the data manipulation process easy and flexible. Like MySQL and PostgreSQL database system, it supports master-slave configuration to achieve higher data availability. it also supports horizontal scalability and it is achieved by auto-sharding.

```
{
   '_id' : 1,
   'name' : { 'customer name ' : 'John', 'customer_lastname’: 'Backus' },
   'contribs' : [ 'Fortran', 'ALGOL', 'Backus-Naur Form', 'FP' ],
   'awards' : [
      {
         'award' : 'W.W. McDowell Award',
         'year' : 1967,
         'by' : 'IEEE Computer Society'
      }, {
         'award' : 'Draper Prize',
         'year' : 1993,
         'by' : 'National Academy of Engineering'
```

```
            }
        ]
    }
```

BSON like data representation example

Cassandra: it is a column based non-relational database system which uses Cassandra query languages to store and retrieve data. The syntax of CQL is a bit closer to SQL in relational database management system. It doesn’t support GROUP and JOIN operations. Cassandra supports peer-to-peer configuration to achieve higher availability, which indicates that all nodes in the network are the equivalent and there are no central nodes that control different nodes. This makes Cassandra preferable for big data center applications. unlike MongoDB, Cassandra supports both vertical and horizontal scaling. This makes Cassandra structure truly versatile and profoundly accessible for different organizations.

6.2 Database Speed Comparison

A sequences of batch tests was undertaken to measure the performance of the overall execution of MySQL, PostgreSQL, MongoDB, Cassandra. Based on the test results from chapter 5, we comprehend the following points.

- Non-relational databases have better performance than non-relational database system for all CRUD operations.
- MongoDB the first and the fastest database for all CRUD operations, and significantly twice faster than PostgreSQL and database in almost all operations. The performance of PostgreSQL has the worst performance than any other tested databases has.
- Cassandra is the second fastest databases after MongoDB and in some CRUD operations the performance of Cassandra is twice faster that PostgreSQL database.
- MySQL is the fastest database and it is placed in the 3rd place in almost all CRUD tests here.

6.3 Community and Support

Since relational databases has serving for many years, Both MySQL and PostgreSQL databases has more community support than non-relational databases (MongoBD and Cassandra).

Table 6.1: Evaluation of selected databases

Database Name	**MySQL**	**PostgreSQL**	**MongoDB**	**Cassandra**
data model and data storage	Relational data model that stores data in table	Relational data model that stores data in table	Stores data in the form of documents	Store data based on column format
Scalability	Supports scaling in	Supports scaling in	Supports scaling out	Supports scaling out
Availability/ replication	Supports both master to master replication and master to slave replications.	Supports only master slave replication Read operation is highly available	Supports master-slave replication by auto sharding. It has built in replication but need some assistance in getting it to setup	Cassandra supports peer to pear replications. And It has built in replication. Cassandra guarantees 99% data availability
Acid compliant	Acid compliant	Strongly Acid compliant	It is not acid compliant	It is not acid compliant
Data schema	Rigid schema	Rigid schema	Flexible schema	Dynamic and flexible schema
syntax	Easy to learn	Easy to learn	It is a little bet hard to learn MongoDB than learning other languages like SQL and CQL.	It is to learn compared with MongoDB
Architecture	Server-client	Client -server	Distributed	Distributed
Query language and schema	SQL oriented language	SQL oriented query language	BSON oriented query language	Cassandra query language (CQL)
Application area	Designed for large applications due to the query compared with MySQL	It is suitable for transactions and simple websites and others	Due to its accessibility ability it is preferable for distributed and big data environment	Due to its accessibility ability it is preferable for distributed and big data environment

CHAPTER 7
CONCLUSION AND FUTURE WORKS

7.1 Conclusion

the main goal of this paper was to investigate and analyze four database management system based on the data model and performance. therefore, this paper examined MySQL, PostgreSQL, MongoDB and Cassandra databases to choose the right database technologies to maximize the performance of the software products. Even if each database has their own requirements and preferable situation to be chosen, we simply focused on testing performance of the databases in a single server.

from the data model perspective MongoDB and Cassandra has various advantages over MySQL and PostgreSQL databases for various reasons. Both MongoDB and Cassandra support horizontal scaling. Data representation procedure in MongoDB is very simple and easy for almost all types of data compared with other database systems. this is due to the help of its document-based nature of data model for storing different types of data (structured or unstructured).

Cassandra is predominantly guaranteeing availability and linear scalability at a higher level than other database systems. This makes Cassandra structure truly versatile and profoundly accessible since every one of the nodes in a cluster can serve read or write operations and it promises availability almost 99%. Cassandra use Cassandra query language.

Both MongoDB and Cassandra support schema free architecture to store and manipulate the data. This helps developers to design the system freely and in flexible ways than designing in MySQL and PostgreSQL.

from the performance result of all database systems, we recognize that the performance of MongoDB and Cassandra databases is much faster than MySQL and PostgreSQL database in all CRUD operations (create, read, update and delete). the most preferable server was the non-relational MongoDB for most of the operations. the performance of MongoDB for insert, select and update operation is better than all other database servers in a single node.

And also, Cassandra provides the highest performance for delete operations. for all operations.

the performance of MySQL database is better than the performance of PostgreSQL database. In select operation, the performance of MongoDB is twice higher than the performance PostgreSQL databases. the performance PostgreSQL databases is slower than all databases systems for all operations

in general, we can select non-relational database specially MongoDB instead of relational databases for the web system which needs to support multiple data transactions. This is mainly due to non-relational database system has the capability to support higher availability and scalability of the data than relational database. and as we seen in the performance test, MongoDB database has still better performance for all data manipulation operations than most of the other databases. And Cassandra would be the second-choice database for all operations in this study.

As enhancement for the future work of related study would continue on examining the performance of relational and non-relational databases in multiple servers and many users with huge number of transactions to realize the efficiency of the databases in concurrent transactions. This helps developers to choose the best databases by analyzing and testing the scalability, availability and performance of relational and non-relational databases

REFERENCES

Apache Cassandra. (2016). Documentation. Retrieved March 28,2019, at http://cassandra.apache.org/doc/latest/

Arts, G. (2013). *Data Replication for the Distributed Database using Decision Support Systems*. *69*(3), 28–39.

Brewer, E. A. (2003). *Towards robust distributed systems (abstract)*. (May),7. https://doi.org/10.1145/343477.343502

Brittain, J. (2009). The Definitive Guide. In Eben Hewitt (Ed.), *October* (Vol. 7).

Chodorow, K. (2013). *Mongodb_ the Definitive Guide - Kristina Chodorow_1401*.

Douglas K Barry. (2019). ACID Properties. Retrieved April 24, 2019, at https://www.service-architecture.com/articles/database/acid_properties.html

DZONE. (2016). Introduction to Apache Cassandra's Architecture - DZone Database. Retrieved March 4, 2019, at https:// dzone.com/articles/introduction-apache cassandras

Edition, S. (2006). Beginning Databases with PostgreSQL. In *Beginning Databases with PostgreSQL*. https://doi.org/10.1007/978-1-4302-0018-5

Eugen Hoble. (2016). Divide and Conquer: High Scalability With MongoDB Sharding - DZone Database. Retrieved March 4, 2019, from https://dzone.com/articles/divide-and-conquer-high-scalability-with-mongodb-t

Georgi Georgiev. (2016). What is Vertical scaling and Horizontal scaling - Vertical and Horizontal hardware / services scaling, Retrieved February 14, 2019, at http://www.pc-freak.net/blog/vertical-horizontal-server-services-scaling-vertical-horizontal-hardware-scaling/

Gyorodi, C., Gyorodi, R., Pecherle, G., & Olah, A. (2015). A comparative study: MongoDB vs. MySQL. *2015 13th International Conference on Engineering of Modern Electric Systems, EMES 2015*, 0–5. https://doi.org/10.1109/EMES.2015.7158433

Jain, V., & Upadhyay, A. (2017). MongoDB and NoSQL Databases. *International Journal of Computer Applications*, *167*(10), 16–20. https://doi.org/10.5120/ijca2017914385

Kofler, M., & Kramer, D. (2005). The definitive guide to MySQL 5: Third edition. In *The Definitive Guide to MySQL 5:Third Edition*. https://doi.org/10.1007/978-1-4302-00710

Li, Y., & Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings,* (November), 15–19. https://doi.org/10.1109/PACRIM.2013.6625441

Lourenço, J. R., Cabral, B., Bernardino, J., & Vieira, M. (2018). Comparing NoSQL Databases with a Relational Database: Performance and Space. *Services Transactions on Big Data*, *2*(1), 1–14. https://doi.org/10.29268/stbd.2015.2.1.1

Mohamed, M. A., Mohamed, M. A., Altrafi, O. G., & Ismail, M. O. (2014). Relational Vs. NoSQL databases: A survey. *International Journal of Computer and Information Technology*, *03*(03), 2279–0764. Retrieved at www.ijcit.com598

MongoDB, I. (2008a). Data Modeling Introduction — MongoDB Manual. Retrieved April 12, 2019, at https:// docs.mongodb.com/manual/core/data-modeling-introduction/

MongoDB, I. (2008b). Replication — MongoDB Manual. Retrieved June 9, 2019, at https: //docs.mongodb.com/manual/replication/

MongoDB, I. (2019.-a). Community | MongoDB.Retrieved March 29, 2019, at https://www.mongodb.com/community

MongoDB, I. (2019.-b). Technical Support — MongoDB Manual. Retrieved April 28, 2019, at https://docs.mongodb.com/v3.4/support/

Oracle. (2019). MySQL :: MySQL Documentation. Retrieved March 19, 2019, at https://dev.mysql.com/doc/

Patil, M. M., Hanni, A., Tejeshwar, C. H., & Patil, P. (2017). A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retriewal operations using a web/android application to explore load balancing-Sharding in MongoDB and its advantages. https://doi.org/10.1109/I-SMAC.2017.8058365

PostgreSQL. (2019a). PostgreSQL: Community. Retrieved March 24, 2019, at https:// www.postgresql.org/community

Pos Software Testing Help. (2019). Database CRUD Testing Through UI (with Sample Test Cases). Retrieved February 28, 2019, at https://www.softwaretestinghelp.com/crud-testing/

postgreSQL. (2019b). PostgreSQL: Documentation. Retrieved March 25, 2019, at https://www.postgresql.org/docs/

RapidValue. (2015). Cassandra – The Right Data Store for Scalability, Performance, Availability and Maintainability-RapidValue. Retrieved April 6, 2019,at https:// www.rapidvaluesolutions.com/tech_blog/cassandra the right data store forscalability performance-availability-and-maintainability/

Safari Books Online. (2019). 1. MySQL Architecture and History - High Performance MySQL,3rd Edition [Book]. Retrieved April 9, 2019, at https://www.o reilly.com/ library /view/high-performance mysql/9781449332471/ch01.html

Software Testing Help. (2019). Database CRUD Testing Through UI (with Sample Test Cases). Retrieved February 28, 2019, at https://www.softwaretestinghelp.com/crud-testing/

solid IT. (2019). 2019 Database Trends – SQL vs. NoSQL, Top Databases, Single vs. Multiple Database Use-High Scalability-.Retrieved April 28, 2019, at http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html

Syed Sadat Nazrul. (2018). CAP Theorem and Distributed Database Management Systems. Retrieved April 3, 2019, at https://towardsdatascience.com/cap-theorem-and-distributed-database-management-systems-5c2be977950e

Timescale. (2018). High availability and scalable reads in PostgreSQL. Retrieved April 1, 2019, at https://blog.timescale.com/scalable-postgresql-high-availability-read-scalability-streaming-replication-fb95023e2af/

Tony Branson. (2016). Database Scalability : Vertical Scaling vs Horizontal Scaling.

Retrieved April 24, 2019, at http://www.vcloudnews.com/database-scalability-vertical-scaling-vs-horizontal-scaling/

Tudorica, B. G., & Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. *Proceedings - RoEduNet IEEE International Conference*, 1–5. https://doi.org/ 10.1109/RoEduNet.2011.5993686

APPENDICES

APPENDIX 1

CODE FOR ELAPSED TIME OF CRUD OPERATIONS

The following code measures the time taken by each databases operation

```
<!--?<span class="hiddenSpellError" pre=""-->
    <?php
class Timer
{
    private  $start ;
    private  $pause_time ;

    /*  start the timer  */
      public function  __construct($start = 0)
    {
        if($start) { $this->start(); }
    }

    /*  start the timer  */
    public function start()
    {
        $this->start = $this->get_time();
        $this->pause_time = 0;
    }

    /*  pause the timer  */
    public function pause()
    {
        $this->pause_time = $this->get_time();
    }

    /*  unpause the timer  */
    public function unpause()
    {
        $this->start += ($this->get_time() - $this-
>pause_time);
        $this->pause_time = 0;
    }

    /*  get the current timer value  */
    public function get($decimals = 8)
    {
        return round(($this->get_time() - $this->start) ,
$decimals);
    }
    /*  format the time in seconds  */
    public function get_time()
    {
        list($usec,$sec) = explode(' ', microtime());
        return ((float)$usec + (float)$sec);
    }}
    ?>
```

APPENDIX 2

EXECUTION OF MYSQL QUERY IN PHP

The following php script shows the execution time of MySQL queries.

```
<?php
require_once('loadtime.php') ;
require_once('db_connect.php') ;
$timer = new Timer(1);
echo @$num_of_records = $_POST['query_length'];
//////////////////create database connection ///////////////
$mysqli = mysqli_connect("localhost","root","", "lab") or die ("could not
connect to mysql");
 $time1= $timer->get();
 $message1 = "Db connection established at:".$timer->get();
////////////////create table script////////////////////////
$tableCreate = "CREATE TABLE IF NOT EXISTS data_table2 (id
                  int(11) NOT NULL auto_increment ,
                  customer_name VARCHAR(25),
                  customer_lastname VARCHAR(25),
                  customer_address VARCHAR(25),
                  phone INT,
                  email VARCHAR(25),
                  zip INT,
                  state VARCHAR(25),
                  country VARCHAR(25),
                  car_model VARCHAR(25),
                  car_price INT,
                  PRIMARY KEY (id))";
$queryResult = mysqli_query($mysqli , $tableCreate);
 $time2= $timer->get();
 $message2 = "Table created at : " . $timer->get();
/////////////////insert data into the table///////////////
  if ($queryResult === true) {
  for ($i = 1; $i <=$num_of_records; $i++)
 mysqli_query($mysqli, "INSERT INTO data_table2 (customer_name,
customer_lastname,customer_address,phone,email,zip,state,country,
car_model,car_price)VALUES('Saron','Addal','Addisababauniversity','200195
838',
'Saron0918gmail.com','34533','Amhara','Ethiopa','2018ToyotaAvalon','25000
0);");
    }
  } else {
  print "<br />No TABLE created. Check";
  }
   $time3= $timer->get();
///////////////////read data from the table //////////////////
  $message3 = "Data inserted into the table at: " . $timer->get();
  $result = mysqli_query($mysqli , 'SELECT * FROM data_table2') ;
  $arrayResults = array() ;
  while ($row = $result->fetch_assoc()) {
 array_push($arrayResults , $row['customer_name']); }
  $row['customer_name'];
$message4 = "Data is read from table and inserted into an array at:".
$timer->get();
//////////////update all the data in the table///////////////
$sql=mysqli_query($mysqli ,"UPDATE data_table2 SET
customer_name='Wondwessen',customer_lastname='Haile',customer_address='Ni
kosia',phone='53382238',email='wondwessen@gmail.com',zip='434556',state='
North_Cyprus',country='Turkey',car_model='2011LexusRX350',car_price='3002
330'";
 $time5= $timer->get();
  $message5="Data is compately updated at : ". $timer->get();
//////////delete all the data in the table/////////////////
$sql_delete = ' DELETE FROM data_table2';
  if (mysqli_query($conn, $sql_delete)) {
       $time6= $timer->get();
  $message6= "Record deleted successfully at: ". $timer->get();
  } else {
 echo "Error deleting record: ". mysqli_error($conn);
  mysqli_close($conn);
}
///////////////////////******END //////////////////////////
?
```

APPENDIX 3

EXECUTION OF PostgreSQL QUERY IN PHP

The following php script shows the execution time of PostgreSQL queries.

```
<?php
require_once('loadtime.php') ;
$timer = new Timer(1);
/////////////database conneciton script////////////////////
 @$query_length = $_POST['query_length'];
  $host  = "host = 127.0.0.1";
  $port  = "port = 5432";
  $dbname = "dbname = my_testdb";
  $credentials = "user = postgres password= ";
  $db = pg_connect( "$host $port $dbname $credentials" );
  if(!$db) {
 echo "Error : Unable to open database\n";
  } else {
 //echo "Opened database successfully\n";
 $time1 = $timer->get() ;
 $message1 ="Db connection established at: ". $timer->get();
   }
////////////////////////create table script//////////////////
$query = "CREATE TABLE IF NOT EXISTS data_table5 (id SERIAL
                        PRIMARY KEY,customer_name VARCHAR(25),
                        customer_lastname VARCHAR(25),
                        ustomer_address VARCHAR(25),
                        phone INT,
                        email VARCHAR(25),
                        zip INT,
                        state VARCHAR(25),
                        country VARCHAR(25),
                        car_modal VARCHAR(25),
                        car_price INT)";

pg_query($db, $query) or die("Cannot execute query: $query\n");
                        if(!$db) {
             echo "Error : Unable to create the table \n";
```

```
                        } else {
                         $time2 = $timer->get();
 $message2 ="table created  successfully at:". $timer->get();
      }
/////////////////write the data into the table///////////////
       for ($i = 1; $i <=$query_length; $i++) {
 $result=pg_Exec($db, "INSERT INTO data_table5 (customer_name,
customer_lastname,customer_addressphone,email,zip,state,country,car_modal
,car_price)VALUES('Saron','Addal','Addisababauniversity','200195838','Sar
on0918gmail.com','34533','Amhara','Ethiopa','2018ToyotaAvalon','250000');
");
                               }
                  if(!$result) {
            echo "Error : Unable to insert data \n";
       } else {
                  $time3= $timer->get();
$message3 = "data inserted into database successfully at:". $timer-
>get();
            }
/////////////////////////read data from the table /////////////
  $sql =<<<EOF
 SELECT * from data_table5;
  EOF;
  $ret = pg_query($db, $sql);
  if(!$ret) {
   echo pg_last_error($db);
   exit;
  }
  while($row = pg_fetch_row($ret)) {
   "ID = ". $row[0] . "\n";
   "name = ". $row[1] ."\n";
  "price = ". $row[2] ."\n";
  }
  $time4= $timer->get();

 $message4= "Data is read from table at: ". $timer->get();
//////////////update all the data in the table///////////////
```

```
  $sql = "update data_table5 set 
customer_name='Wondwessen',customer_lastname='Haile',customer_address='Ni
cosia',phone='53382238',email='wondwessen@gmail.com',zip='434556',state='
North_Cyprus',country='Turkey',car_modal='2011LexusRX 
350',car_price='3002330'";
            $result = pg_query($db, $sql);
            if(!$result){
             echo pg_last_error($db);
            } else {
$time5= $timer->get(); "Data is successfully updated at : ". $timer-
>get();
             }
////////////////delete data from the table////////////////////
             $sql = "delete from data_table5";
             $result = pg_query($db, $sql);
                  if(!$result){
                   echo pg_last_error($db);
                  } else {
                   $time6= $timer->get();
                        }
                        // Close the connection
                        pg_close($db);
//////////////////////////////******END*******///////////////
      ?>
```

APPENDIX 4
EXECUTION OF MONGODB QUERY IN PHP

The following php script shows the execution time of MongoDB queries.

```
<?php
require 'vendor/autoload.php';
require_once('loadtime.php');
//$num_of_records=1;
@$num_of_records = $_POST['query_length'];
$client = new MongoDB\client;
$timer = new Timer(1);
try {
//$connection = new Mongo();
$client = new MongoDB\client;
$database = (new MongoDB\Client)->customerinfo;
$time1= $timer->get();
 $message1 = "Db connection established at : " . $timer->get() . "<br \>
<br \>" ;
} catch(MongoConnectionException $e) {
die("Failed to connect to database ".$e->getMessage());
}
$customercollection=$database->customer;
for ( $i = 0; $i < $num_of_records; $i++ ){
$insertOneResult=$customercollection>insertOne(['customer_name'=>'wondwes
sen','customer_lastname'=>'haile','customer_address'=>'north_cyprus','pho
ne'=>5338223089,'email'=>'wondwessen533@gmail.com','zip'=>1234,'state'=>'
north_cyprus','country'=>'turkey','car_model'=>'Toyota
2010','car_price'=>23434]);
}
$time3= $timer->get();
 $message3 = "Data inserted into the table at:".$timer->get();
$document=$customercollection->find(['customer_name'=> 'wondwessen']);
foreach($document as $doc){
//var_dump($doc);
}
//var_dump($document);
```

```
$time4= $timer->get();
$message4 = "Data is read from table and inserted into an array at: ".
$timer->get();
try {
$updateResult=$customercollection>updateMany([],['$set'=>['customer_name'
=>'Anteneh','customer_lastname'=>'addal','customer_address'=>'addisababa'
,'phone'=>'9182','email'=>'yohanse@gmial.com','zip'=>'1322','state'=>'amh
ara','country'=>'ethiopia','car_model'=>'2011LexusRX350','car_price'=>'32
33']]);
$message5="Data is compately updated at : ". $timer->get();
 $time5= $timer->get();
 } catch(MongoConnectionException $e) {
 die("Failed to connect to database ".$e->getMessage());
}
$deleteResult=$customercollection>deleteMany(['customer_name'=>
'Anteneh']);
$time6= $timer->get();
 $message6= "Record deleted successfully at: ".$timer->get();

?>
```

APPENDIX 5

EXECUTION OF Cassandra QUERY IN PHP

The following php script shows the execution time of Cassandra queries

```
<?php
require_once('loadtime.php') ;
$timer = new Timer(1);
@$num_of_records = $_POST['query_length'];

//////////////////////create database connection
      $cluster = Cassandra::cluster()                   -
            ->withContactPoints('127.0.0.1')
             ->build();
            $session   = $cluster->connect()

   $session->execute("CREATE KEYSPACE IF NOT EXISTS customer_info WITH
replication = { 'class': 'SimpleStrategy', 'replication_factor': '3' }");
   $session->execute("USE customer_info");
  $time1= $timer->get();
  $message1 =  "Db connection established  at  : " . $timer->get();



$session->execute("CREATE TABLE IF NOT EXISTS customer7 (id uuid,
                                 customer_name text,
                                 customer_lastname text,
                                 customer_address text,
                                 phone int,
                                 email text,
                                 zip int ,
                                 state text,
                                 country text,
                                 car_model text,
                                 car_price int,PRIMARY KEY(id))");
  $time2= $timer->get();
  $message2 =  "table created successfully  at  : " . $timer->get();

/////////////////////insert data into the table///////////////////////

$statement  =  $session->execute(new  Cassandra\SimpleStatement(  "INSERT
INTOcustomer7(id,customer_name,customer_lastname,customer_address,phone,e
mail,zip,state,country,car_model,car_price)VALUES(uuid(),'wondwessen','ha
ile','nicosia',9012323,'wondwessen533@gmail',12323,'northcyprus','turkey'
,'toyota 123323',8922762)"
           ));

  $time3= $timer->get();
  $message3 = "data inserted successfully at : " . $timer->get();
```

```
////////////////////read data from the database
//////////////////////////
  $result = $session->execute("SELECT * FROM customer7");

foreach ($result as $row) { $row['customer_name'] . ": " .
$row['customer_lastname'] . " / " . $row['phone'] . " / " . $row['email']
." / " . $row['zip'] . " / ".$row['state'] . " / " . $row['country'] . "
/ ".$row['car_model'] . " / " . $row['car_price']."<br \> <br \>"
.PHP_EOL;
}
$time4= $timer->get();
 $message4= "data is selected successfully at: ".$timer->get();
      $schema = $session->schema();


///////////////////////////////////

$delete = $session->execute(new Cassandra\SimpleStatement
   ("TRUNCATE customer_info.customer7"));

      $time6= $timer->get();
  $message6 = "data deleted successfully at: ". $timer->get();

 /////////////////////////////******END*******////////////////////


?>
```