Information Research Communications, 2024; 1(2):104-119.
https://inforescom.org

Research Article

Comparative Study on Oracle, Neo4J, Cassandra, Redis, and MongoDB

Sathishkumar Veerappampalayam Easwaramoorthy*, Klaudia Ong Yun Xuan, Laksamana Putra, Ng Chi Ern, Tan Jun Sheng

School of Engineering and Technology, Sunway University, Jalan Universiti, Bandar Sunway, Selangor Darul Ehsan, MALAYSIA.

ABSTRACT
Database models are an essential part in defining how database systems store and manage their data. Database models can have multiple forms and it might cause confusion on selecting the appropriate model for their needs. Thus, this paper helps to examine the performance mechanism of various database models, including Oracle, Neo4J, Cassandra, Redis and MongoDB by identifying their key characteristics and concepts of each model to provide a comprehensive guide for readers regarding database selection. The research methodology incorporates academic references to contribute to creating an evaluation framework for comparison analysis of each model. The comparison framework takes several aspects of data security, data retrieval, data creation and data manipulation into consideration to facilitate in-depth comparison. Our findings reveal that each model has its own strength and uniqueness. In terms of data integrity and access control, Oracle stands out among others because of its strong mechanism. However, Neo4J excels in data creation and fast data retrieval using its record searches whilst Cassandra is best at controlling large data volumes. As for data manipulation, Oracle supports reliability while Cassandra focuses on scalability and nodes distribution. MongoDB as well as Redis also perform strong scalability through their sharding features.

Keywords: Comparison, Data, Evaluation, Framework, Performance.

Correspondence:
Dr. Sathishkumar Veerappampalayam Easwaramoorthy
School of Engineering and Technology, Sunway University, No. 5, Jalan Universiti, Bandar Sunway-47500, Selangor Darul Ehsan, MALAYSIA.
Email: sathishv@imail.sunway.edu.my

Received: 09-11-2024;
Revised: 29-11-2024;
Accepted: 15-12-2024.

INTRODUCTION

Database is an important factor that can contribute to the success or failure of a business. They consist of collection of data, saved in a systematic and organized structure to serve a specific purpose (Goldmeier, 2024). The data structure can come in different models such as relational model, graph-based model, key-value model, document model, or wide-column model. It is used to define database structures that are formed based on a collection of concepts. They contain entities, relationships and attributes that represents real-world concept. It can represent relationship between an employee to a manager or an organization to their client. In short, data models can be described as frameworks of databases that usually used to store data in a database management system (Elmasri & Navathe, 2016). However, due to its variations, it might cause confusion to people to choose the most appropriate database models. Thus, leads to this paper which aims to assess each database model based on several factors as the threshold. It will involve a thorough analysis of database models' performance and their scalability ratio. The analysis conducted will provide insights to guide readers in selecting the better overall performance database models based on users' needs. Resulting to easier progress in striving for organizational business objectives and better decision-making processes. To highlight the importance of data integrity and data security, oftentimes, database models contain vital information about a company which can only be retrieved by authorized users within the company, if necessary. Such as when the users share their data model content, only some users or user groups are given access to edit classified information like bonuses or salaries, while the others are only allowed to view the data. This is to prevent unauthorized access or modifications to the restricted data which is the purpose of data security. Data security is an attribute of a database management system that prevents illegal access to the content inside the database. It usually involves an authentication mechanism or data encryption, which is used to add another protection layer by creating passwords or coding algorithm layer that needs to be deciphered before being able to access the data. These data security features are then frequently checked and maintained in database audits to check for the existence of unauthorized operations. This type of access operation is facilitated by database management systems to manage, create and regulate database access/control to people who are responsible for maintaining data consistency, integrity and data security. Moreover, data integrity is an important factor in ensuring reliable and consistent database models.

DOI: 10.5530/irc.1.2.13

Upholding data consistency could significantly contribute to higher data accuracy. The consistency of data within the database is maintained by enforcing integrity constraints such as unique primary key constraints and referential integrity constraints to the data models (Elmasri & Navathe, 2016). Nevertheless, it can be challenging for stakeholders into decide which database model to incorporate to the company as each data model can have different access control and constraint mechanisms. Therefore, this paper aims to provide readers with a comprehensive understanding and a detailed analysis of a variety of database models, including relational models and not only SQL (NoSQL) models. Grasping the concept, benefits and disadvantages of the processes that each model underlines will provide us with the knowledge to be well-prepared to make an informed decision. In this study, we aim to address these objective questions:

RQ1. Which database model is suitable for different characteristics in of application scenario?

RQ2. What are the key differences between relation databases and a nonrelational database?

RQ3. What are the strengths and weaknesses based on several factors for comparison?

LITERATURE REVIEW

Database Model Aspects

A foundation for tackling issues of managing data is provided by Database Management Systems (DBMS), which are made to manage and oversee databases. This allows organizations to manage, organize and use their data with ease. Data management systems include several integral aspects that ensure a successful process, which are data creation, data manipulation, data retrieval, access control and data integrity. DBMS allows users to create database through its data creation, or Data Definition Language (DDL). DDL is used to define database views and storage structures, e.g. creating a table and altering a table (Punit *et al.*, 2024). Another similar feature of DBMS is data manipulation using Data Manipulation Language (DML). Examples of data manipulation are inserting, update and delete functions (Punit *et al.*, 2024). A subset of DML known as Data Query Language (DQL) is responsible for the SELECT statement. This command is important in retrieving data from a table without resulting in any changes to the table (Glimm & Horrocks, 2004). In massive databanks, each user has different levels of authority to view or edit data. To grant or revoke access for users to manage the database, Data Control Language (DCL) is used to act as access specifier (Fehily, 2020). Examples of statements used to manage user control are GRANT and REVOKE. Lastly, ensuring data integrity in databases requires maintaining the accuracy and consistency of data over its lifecycle. To achieve this quality, enforcing constraints at the database level (e.g. primary keys and foreign keys) would maintain the validity of data. Moreover, using Transactional Control Language (TCL) to control all transactions of data would uphold the safe keeping of data. Examples of commands from TCL are Roll Back (used to undo changes), Commit (used to apply or save changes) and Save Point (used to save data on a temporary databank in the database) (Punit *et al.*, 2024).

Data Model

Data model is an abstract representation of a database structure that defines the real-world entities' relationships and constraints between entities (Sebastian-Coleman, 2022). Data models are frequently represented by using different notations, which includes Chen notation, Crow's Foot notation and Unified Modeling Language (UML) notation. The Chen notation, pioneered by Peter Chen in 1976 developed the entity-relationship model which differentiates entities, attributes and relationships with the use of unique shapes (Chen, 1976). Crow's foot diagrams depict entities as boxes and relationships as lines connecting the boxes (Hammerschmied & Bork, n.d) .The shapes at the endpoints of these lines show the relationship's relative cardinality. However, this notation does not support attributes. Lastly, UML notation is a more versatile language that can be used for various modelling purposes, including Entity Relation Diagram (ERD). It is the standard language for software development and documentation that is effective at modelling large and complicated systems (Alkoshman, 2015). Its cardinality is represented by characters such as "1...1". As data modelling is merely a visual depiction of a system design intended for unified understanding of data, logical data design and physical implementation can help developing a more refined version of the model (Sebastian-Coleman, 2022). A logical data model thoroughly explores the conceptual model. For platform-independent implementation, it diagrammatically expresses relationships, entity names and data restrictions. On the other hand, physical implementation enhances the logical data model for usage with a particular database system. To facilitate effective data storage, retrieval and manipulation, logical data models and physical implementations specify the rules, organization and structure of the data.

Database Model

Nowadays, there are a variety of database model options available to choose from, such as Oracle, Neo4J, Cassandra, Redis and MongoDB. The flexibility of choosing different models based on needs, allow users to meet their objective in an efficient manner due to each of these database models has their own features that distinguish them with each other.

Oracle (Relational Model)

Oracle provide a platform that assists users with creating design, modifications and deployment of database applications using web browser with minimal complexity to allow a more beginner-friendly approach while maintaining robust features

(Baggia *et al.*, 2018). In relation to that, Oracle is considered as a relational database management systems due to its utilization of table-form relation models. Relational model is a database model that focused on managing relational databases. Relational models are beneficial because it ensures reliability, robustness and scalability. Data integrity in relational model is backed by ACID properties that stands for Atomicity, Consistency, Isolation and Durability (Jatana *et al.*, 2012). Atomicity verifies a valid transaction so there will be no partial transaction done. Consistency ensures that database remain consistent throughout the transaction. Isolation allows more than one transaction to be done without affecting one another. Finally, durability ensures that transaction records are saved permanently in the event of system errors. Oracle uses SQL to create and modify data stored in the Oracle database. Incorporating SQL in Oracle database leads to easier quicker query as the nature of SQL as a simple, beginner-friendly language. Furthermore, SQL statement such as add, create, update and delete processed inside Oracle tools like Oracle Apex will grant user influence for data creation and data manipulation (Jatana *et al.*, 2012). Oracle also integrated multiple security layers at database level to mitigate unauthorized access or modifications done within the database (Ilić *et al.*, 2021). It will create a secured environment for users to conveniently manage or modify their database.

Neo4J (Graph Model)

Neo4J is a NoSQL database classified as graph databases. The database follows the concept of mathematical tree concept where each node and relationship within the 'tree' is stored with data (Minder *et al.*, 2024). Graph-based models in Neo4J are widely used as a replacement for relational databases due to severe limitations caused by relational databases. These limitations include a fixed number of columns and table. Thus, it is mitigated by using graph-based databases to allow more flexibility and dynamic data (Minder *et al.*, 2024). This versatility is the result of a non-structured repository where the database nodes manipulation is open-ended and does not require historical data design, enabling higher scalability. Next, Neo4J follows ACID behaviour to support consistent database transactions. Neo4J also implement a high availability feature inside their system by implying the master-slave cluster concept. The master-slave concept works by creating two different partitions consist of the database and required cluster components for management purposes. The master cluster will then be taking the role of writing operations. Aside from that, there are mechanism working alongside the partitions to allow constant synchronization and centralized control by prioritizing the master cluster (Lopez & Cruz, 2015)

Cassandra (Wide-column Model)

Cassandra is a column-family NoSQL database created by Apache Software that is written in Java (Abramove & Bernardino, 2013). A column-family contains an unlimited number of columns where the reading and writing is done by columns instead of rows, allowing fast access (Čerešňák & Kvet, 2019). The model ensures that the rows are indexed by primary keys without any modification (Popescu & Radu, 2020). Cassandra often shows similarity to a relational database in terms of their design and implementation, like their data structure and tables (Okman *et al.*, 2011). Cassandra distributes their data to different nodes using a peer-to-peer clustering architecture, allowing no downtime to replace a failed node, providing high availability and scalability (Abramove & Bernardino, 2013). Additionally, their effortless scaling is also backed through native sharding and replication mechanisms (Popescu & Radu, 2020). Cassandra's transaction support also utilizes ACID properties (Gundigara & Mehta, n.d). Cassandra also uses their own query language known as CQL (Cassandra Query Language) that is like SQL to provide a more user friendly and structured way to manage data compared to Thrift API, their base client (Okman *et al.*, 2011).

MongoDB (Document Model)

MongoDB is a NoSQL that uses a document infrastructure to store and retrieve data (Mehrabani, 2014). MongoDB relies on Binary JavaScript Object Notion (BSON) files to store their data which is commonly used and allows flexibility in the schema of the database ("What is a Document Database?", 2024). It has great performance and scalability due to its features like sharding and replication while handling large data structures (Mehrabani, 2014). MongoDB uses a replication procedure to create and maintain multiple dataset copies across different server and this is supported by their sharding concept to distribute it evenly across, offering performance, scalability and availability (Mehrabani, 2014). MongoDB's sharding concept is done through two scaling methods: horizontal and vertical. The vertical scaling method is done through the administrator increasing the resources and capacity to existing server, horizontal scaling method increases the server's capacity by combining multiple separate servers (Mohan *et al.*, 2024).

Redis (Key-value Model)

Redis is an in-memory database model that uses key-value storage system to provide high performance, replication and a unique data model (Carlson, 2013). It supports multiple data structures or variables such as lists, strings, hashes, sets and sorted sets (Mohan *et al.*, 2024). As it utilizes memory, it can execute data request with higher performance as compared to disks because of high throughput and low latency (Carlson, 2013). Redis uses a single-threaded design written in ANSI-C that requires very minimal memory to process client requests asynchronously by overlapping network I/O processing (Zhang *et al.*, 2015; Chinnachammy, 2013). Redis availability when a server crash or power off takes two forms: point-in-time dump, which takes periodic snapshot of the entire dataset and save them to single disk or append-only-file and that writes logs of every operation to a file (Chen *et al.*, 2016). Redis has also introduced a distributed version called Redis Cluster that allows client-side sharding to distribute data across multiple databases on client side (Zhang *et al.*, 2015). Additionally, Redis' replication is done through a master/slave process where the slaves receive a copy of the full database when connected to the master (Carlson, 2013). Table 1 summarizes the database management aspects, data model and database model discussed in our literature review.

METHODOLOGY

This section describes the methods and process we used for our research, including literature review, conceptual synthesis and result derivation. First, we search for relevant papers using academic databases such as Google Scholar, ScienceDirect and ResearchGate. We used specific keywords to find relevant research articles and ensured their quality by assessing their research methods and relevance to our study. Moreover, we investigated their research strategy, data collection techniques and statistical analysis results. We focus on studies with clear methods and meaningful statistical results, which enables us to apply proven approaches to our research content. These selected research

Table 1: Literature Review Summary

Summary of Literature Review				
Data Model **An abstract representation of database structure**				
Chen Notation The model differentiates entity, attribute and relationships. Cardinality can be represented with numbers or letters.	Crow's Foot Depict entities as boxed and relationships as line. Does not support attributes. Cardinality shown on Crow's Foot Symbol.	UML A more versatile notation. Effective at large modelling system. Cardinality is represented as characters.		
Logical Data Model Diagrammatically expresses relationships, entity names, and data restrictions.	Physical Implementation Specify the rules, organization, and structure of the data.			
5 Common Data Models				
Oracle (Relational Model) Utilize of table-form relation models ACID properties Uses SQL language to create and modify data Integrated multiple security layers.	Neo4J (Graph Model) Formed the base of graph with relationships and nodes Database node manipulation is open-ended (versatile) ACID properties Implying Master-slave cluster concept.	Cassandra (Wide-column model) Writing is done by columns Use peer-to-peer clustering architecture Sharding and replication mechanism to backup ACID properties Uses CQL language to manage data.	Redis (Key-value model) Uses key-value storage system Single threaded design written in American National Standard Institute (ANSI-C) Use point-in-time dump or append-only file to backup Use client-side sharding Replication through master-slave process.	MongoDB (Document model) Uses a document to retrieve data Rely on BSON files to store Great performance with sharding and replication techniques Two sharding method: Vertical: Increase resources and capacity. Horizontal: Increases the server's capacity by combining multiple separate servers.
Database Management Aspect **Include several integral aspects to let the user manage data with ease**				
DDL (Data Creation) Defines database design and structure.	DML (Data Manipulation) Manipulation of data within a database.	DQL (Data Retrieval) Querying data within a database.	DCL (Access Control) Manage data retrieval operations.	TCL (Data Integrity) Control database transactions.

papers provided a solid foundation for our analysis. To manage our research effectively, we divided tasks among group members. Each member was assigned specific tasks such as searching for articles, evaluating sources, or integrating findings. We held regular meetings to discuss our progress and shared valuable insight. All findings were documented in a shared document and each member reviewed others' work to provide feedback and ensure accuracy. Teamwork was crucial throughout this process, especially when facing challenges. We worked together and helped each other overcome obstacles, making sure that all parts of the research were covered. Next, we categorized the literature findings into themes relevant to our research questions. This includes summarizing key points and identifying patterns. We looked for comparisons and contrasts in the methods and results of the selected papers to identify the common themes. Then, we make a simple table to organize the collected information. We focused on key aspects like data creation, data manipulation, data retrieval, access control and data integrity and explained these concepts through several scenarios. We created a robust framework for our analysis by using examples and criteria from the reviewed paper. This framework allows us to evaluate the strengths and weaknesses of each data model based on established academic standards. Based on the combined information, we derived our results by bringing together insights from multiple studies. This method allowed us to draw a strong and reliable conclusion. This approach ensured that our findings were well supported and reliable, providing a strong base for further research and discussion. Figure 1 illustrates a flowchart of our methodology steps.

RESULTS

Databases that are both consistent and connected, as well as management that is both effective and efficient, are critical concerns in the modern information technology era (Khan *et al.*, 2023). Following the associated procedure, we categorized the findings into five main categories- Access Control, Data Integrity, Data Creation, Data Manipulation and Data Retrieval. These categories will evaluate how different data models influence the mechanisms.

Access Control and Data Integrity

Access control and data integrity are considered as one of the most important measures of DBMS security. Table 2 provides the basic security mechanism characteristics of Oracle compared to another DBMS concluded by Ilić *et al.* (2021). Oracle has a strong security mechanism. Access control in Oracle is achieved through strong authentication mechanisms. As Ilić *et al.* (2021) has stated for Table 2 "this table highlights that Oracle confirms user identity at the database level and bases this confirmation on operating system roles" (Ilić *et al.*, 2021). Unlike the others, this multi-layer security ensures that only authorized users can access the database and prevent unauthorized access easily.

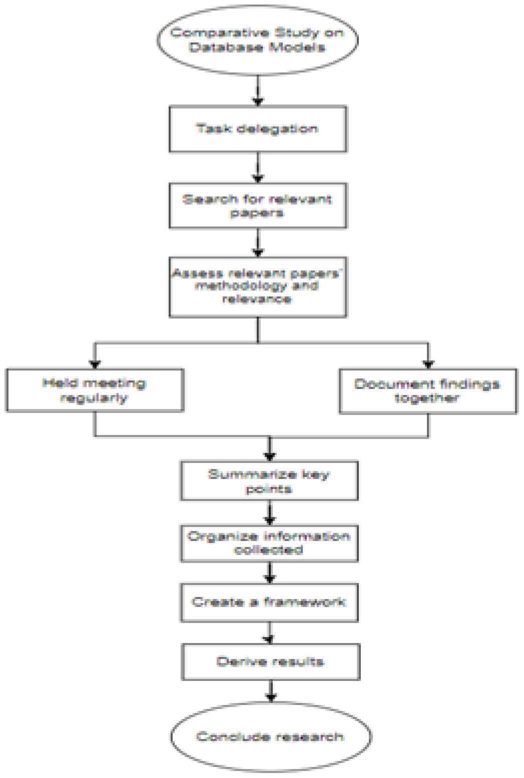

Comparative Study on Database Models
Task delegation
Search for relevant papers
Assess relevant papers' methodology and relevance
Held meeting regularly
Document findings together
Summarize key points
Organize information collected
Create a framework
Derive results
Conclude research

Figure 1: Methodology Steps.

Table 2: Oracle Security Mechanism

Security features	Microsoft SQL Server	Oracle
Security type	Simple security	Multi-layer security
Authentication.	User authentication at the instance level and at the database level.	Confirmation of user identity at the database level and based on OS roles.
Database sharing	Users cannot share databases.	Users can share databases.
Prone to errors and data corruption.	Chances are high.	Chances are low.
Types of backups.	Full, partial and incremental.	Full, partial, incremental and differential.

Sources: Ilić et al., 2021

Compared to others, Oracle has a strong mechanism in place of data integrity. It is designed to minimize the chances of errors and data corruption. As a result, Oracle has a low chance of errors and data corruption. Moreover, Oracle supports a variety of backup formats such as full, partial, incremental and differential backups. These backup options are crucial elements for restoring data if it is loss or corruption and ensure that the data can be recovered precisely when needed.

For graph databases like Neo4J were not designed with solid security features. Neo4J does not have data-level security and data encryption (Gupta & Agrawal, 2018). However, Neo4J offers a new plug-in architecture that allows users to build and deploy customauthentication and authorization controls. Figure 2 illustrates the enhancement of third-party tools such as blockchain on the database security mechanism Blockchain is a reliable third-party tool. Figure 2 is a simple representation of a blockchain-based system specifically designed for graph databases. By leveraging blockchains unique characteristics, Shkokani and Altamimi (2020), proposed that this tool addresses the security gaps in graph databases especially against analytic attack. Blockchain only allows authorized users to access and its cryptographic algorithm plays an important role in assuring data security. Table 3 summarizes the security mechanisms for various databases in terms of data integrity and access control by Noiumkar and Chomsiri (2014). MongoDB, Cassandra and Redis have a common data integrity problem which is there is no data files encryption. The developer can only protect the database by encrypting the data in application level before recording them into the data files. Another way for the developer is to add an appropriately determined permission in an operating system to prevent data hacking from the unauthorized users. In terms of access control, MongoDB and Cassandra have weak performance in client/server authentication and inter-cluster authentication that requires different specified requirements to activate the authentication, which indicates that they have ineffective security mechanisms (Okman *et al.*, 2011). Tables 4 and 5 provide the security control mechanism for MongoDB and Cassandra.

Redis, on the other hand, has no encryption for both authentications. This means anyone can access and get the value if the key is known because the data is stored in the form of key value pair. It does not provide enough security for the data. Significantly, Redis is more vulnerable to attacks. There is no statistical data that mentions that Redis have been attacked by Script Injection or Denial of Service.

Data Retrieval

Data retrieval is crucial for database query performance. It affects how quickly and efficiently users can access the information when required. Effective data retrieval mechanisms ensure faster and more precise query results. To analyse Oracle's data retrieval performance, we have made conclusion based on several research papers. The performance results for Oracle were done by Čerešňák and Kvet (2019) and Kolonko (2018). Kolonko (2018) has run some tests for Oracle and MongoDB with 6 defined

Table 3: **Summary of Nosql Security Mechanism.**

Security Issues	Databases				
	MongoDB	Cassandra	CouchDB	Hypertable	Redis
Data files encryption	No encrypt	No encrypt	No encrypt	No encrypt	No encrypt
Client/Server Authentication/ Encryption	Weak	Weak	SSL	No authen/no encrypt	No authen/no encrypt
Inter-cluster Authentication/ Encryption	Weak	Weak	SSL	No authen/no encrypt	No authen/no encrypt
Script Injection	Vulnerable	Not vulnerable	Vulnerable	Not vulnerable	Not vulnerable
Denial of service attack	Not vulnerable	Vulnerable	Vulnerable	Not vulnerable	Not vulnerable

Sources: Noiumkar & Chomsiri, 2014.

Table 4: **MongoDB Security Mechanism.**

Category	Status	Recommendations
Data at rest.	Unencrypted.	Protect with OS level mechanisms.
Authentication for native connections.	Available only in unsharded configurations.	Enable if possible.
Authentication for native connections.	READ/READ-WRITE/Admin levels, only in unsharded configurations.	Enable, if possible, requires enabled authentication.
Auditing.	Not available in MongoDB.	
AAA (authentication, authorization auditing) for RESTful connections	User and permissions are maintained externally.	Available if configured on a reverse proxy.
Database Communication	Encryption is not available.	
Injection attacks.	Possible, via JavaScript or string concatenation.	Verify that the application does reasonable input validation.

Sources: Okman *et al.*, 2011.

Table 5: Cassandra Security Mechanism.

Category	Status	Recommendations
Data at rest.	Unencrypted.	Protect with OS level mechanisms.
Authentication for native connections.	The available solution isn't production ready.	Implement a custom IAuthentication provider.
Authentication for native connections.	Done at the CF granularity level. The available solution isn't of production quality.	Implement a custom IAuthority provider.
Auditing.	Not available OOTB.	Implement as part of the authentication and authorization solution.
AAA (authentication, authorization auditing) for RESTful connections.	Encryption is available.	Enable this using a private CA.
Database Communication.	No encryption is available.	Add packet-filter rules to prevent unknown hosts from connection. Re-implement the Thrift server-side to use the SSL transport in Thrift 0.6. Add timeouts for silent connections in the Thrift server side, and cap the number of acceptable client connection.
Injection attacks.	Possible in CQL.	If using the Java driver, prefer Prepared Statements to Statements. Always perform input in the application.

Sources: Okman *et al.*, 2011.

workloads. However, we only take 1 workload as our observation with 3 main operations in it. (Read, Read-Modify-Write, Update) The workload is defined as:

Read operations-total of 1000 operations summed as 500 reads in the workload definition and 500 reads from read-modify-writes.

Read-modify-write operations-total of 500 read-modify-writes.

Updates-total of 500 writes from read-modify-write.

Table 6 provides the comparison of operation (Read, Read-Modify-Write and Update) performance. Table 6 demonstrated that Oracle has a significantly longer runtime (8446 ms) compared to MongoDB. Besides, Oracle's throughput is much lower at 118.4 ops/sec. As it is visible in Table 6, Oracle shows considerably higher latencies and less performance in Read, Read-Modify-Write and Update operations compared to NoSQL databases. Additionally, Čerešňák and Kvet (2019) tested Oracle query performance with a variety of databases. Figure 3 and Table 7 shows the overview of the result time for relational databases and nonrelational database. This result was expected due to differences in data storage methods. Relational databases like Oracle need normalized data to avoid duplicity and redundancy. While normalization helps manage data effectively, it also hampers performances, leading to longer query times. Next, Neo4J as a graph database has an easily mutable schema. It is flexible to restructure the entire schema every time a new relationship added. Due to its structure, Neo4J able to find the nodes that meet the search criteria instead of searching the whole data set. Moreover, it only looks at the records that are directly connected to other records. Therefore, when the datasets become larger, it will not significantly increase the retrieval times. This will be proven with an experiment conducted by Batra and Tyagi (2012) on Neo4J data retrieval compared to Oracle. Figures 4 and 5 provide the experiment implementation details and result. It can be observed that the retrieval times of graph databases (Neo4J) are less than relational databases (Oracle). And increasing the

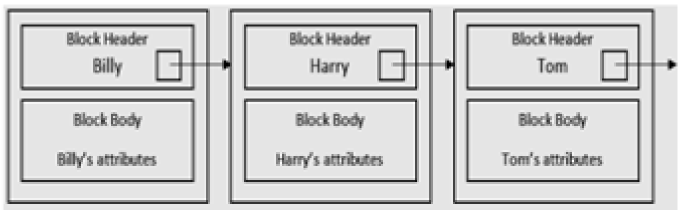


Figure 2: Example of Blockchain. (Shkokani & Altamimi, 2020).

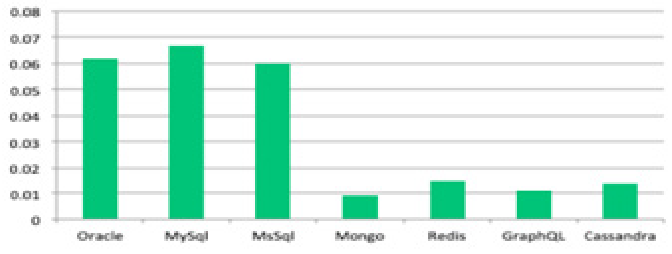


Figure 3: Query performance in miliseconds. (Čerešňák & Kvet, 2019).

Table 6: Comparison of operation performance.

MongoDB Workload F Total runtime: 839 ms Throughput: 1191.90 ops/sec			
Average latency [µs]	434.29	1073.04	445.24
Min latency [µs]	82.0	234.0	147.0
Max latency [µs]	155755.0	186239.0	14495.0
95th percentile latency [µs]	537.0	1339.0	770.0
99th percentile latency [µs]	1031.0	2175.0	1604.0
Oracle Workload F Total runtime: 8446 ms Throughput: 118.40 ops/sec			
Average latency [µs]	2967.66	1110.91	8633.53
Min latency [µs]	83.0	4312.0	4112.0
Max latency [µs]	565247.0	24111.0	19663.0
95th percentile latency [µs]	4155.0	14447.0	10479.0
99th percentile latency [µs]	4591.0	15127.0	11271.0

Sources: (Čerešňák & Kvet, 2019).

number of nodes from one hundred to five hundred does not significantly increase the retrieval time for Neo4J. Following, we use the result of Yahoo! Cloud Serving Benchmark (YCSB) benchmark test tested by Abramove and Bernardino (2013) to conclude our Cassandra database. The scenarios tested are read, write and update operations performed on randomly chosen records. Because our focus is on several operations, some workloads will not be used. The used scenarios are:

Workload C: Read only. This workload is 100% read.

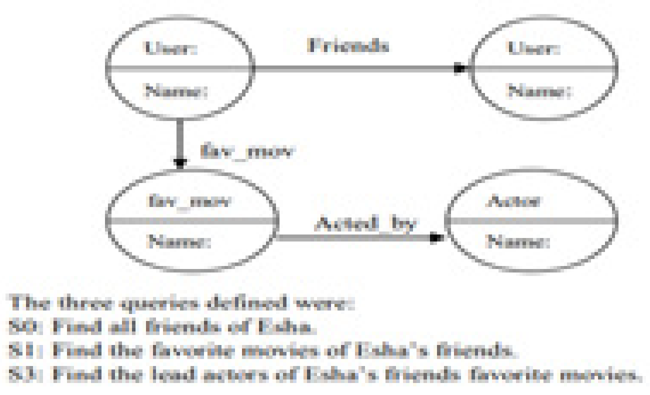
Workload F: Read-modify-write. In this workload, the client will read a record, modify it and write back the changes.

Workload H: Update only. This workload is 100% updated.

Table 7: Query performance of databases with 100000 records in milliseconds.

Type/ Operation	Oracle	MySql	MsSql	Mongo	GraphQL	Cassandra
Insert	0.091	0.038	0.093	0.005	0.008	0.011
Update	0.092	0.068	0.075	0.009	0.012	0.014
Delete	0.119	0.047	0.171	0.015	0.018	0.019
Select	0.062	0.067	0.060	0.009	0.011	0.014

Sources: (Čerešňák & Kvet, 2019).



The three queries defined were:
S0: Find all friends of Esha.
S1: Find the favorite movies of Esha's friends.
S3: Find the lead actors of Esha's friends favorite movies.

Figure 4: Implementation details. (Batra & Tyagi, 2012).

Figures 6-8 illustrates the operation (Read, Read-Modify-Write, Update) results for Cassandra compared to MongoDB. In Figure 6, Cassandra has 1.75 faster speed than MongoDB when using 700k records in Workload C. In Figure 7, Cassandra is 1.8 faster than MongoDB when using 700k records in Workload F. Surprisingly, Figure 8 shows Cassandra has greater results compared to MongoDB with 25 to 43 times better in Workload H.

In Figures 6 and 7, we can observe that Cassandra has weaker performance than MongoDB when processing 100K records. However, the performance of Cassandra improved simultaneously with increasing data sizes in read operations and read-modify-write operations. Cassandra has better execution time with high volume of data. In Figure 8, Cassandra has stable and well performance regardless of database size when it comes to upload operation. With other experiment conducted as well, Abramove and Bernardino (2013) has concluded that Cassandra shows the best result for almost every scenario.

To evaluate MongoDB and Redis performance, Mohan *et al.* (2024) has conducted a test of multiple databases across a broad range of data volumes.

Figure 9 illustrates the query execution time for MongoDB. This is due to MongoDB's inbuilt sharding capabilities, which support horizontal scaling. It allows MongoDB to handle amount of data sets and maintain high output by distributing data across multiple shards.

Although Mongo's scaling is high, its performance may begin to degrade and show poor results as the volume of data grow beyond certain limits. This can be proven with the previous experiment by Abramove and Bernardino (2013) and shown in Figures 6-8.

In the same experiment, Redis takes longer to execute the queries compared to other databases. Figure 10 illustrates the query execution time for Redis. In Figure 10, Redis shows a best-case scenario of taking 3x shorter in sf1, smaller dataset. However, the performance of Redis reduces when the datasets become bigger. It is because Redis suffers from the data schema. Assigning all features to a single key makes each record very large, which will slow down query execution and increases central processing unit (CPU) usage.

Data Creation

Data creation ensure that the data inserted is correctly formatted and saved in a way that maintains the structure of the database. In our result, we will use data loading as our key observation value. Data loading is a critical component of data creation. Effective data loading essential for maintaining database consistency and optimal performance as data volumes increase and the system scales.

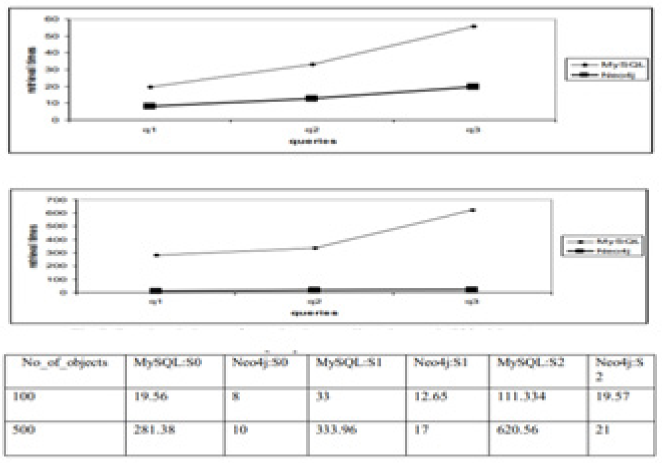
Oracle performs well in large volumes of data from flat files. Oracle database has a file bulk loading mechanism, SQL*Loader. It is a command-line tool for Oracle designed to improve data loading into Oracle database. The SQL* Loader is used to import massive data into the database in parallel when the burst data is all stored in the form of files. Additionally, it supports various data formats and control files for mapping data to database structures. It also tracks errors and creates records of successful and failed data loads (Liu *et al*., 2022).

For Neo4J, Gupta and Agrawal (2018) mentioned that it delivers lightning-fast read and write performances while still protecting data integrity. As mentioned earlier, Neo4J has the advantage of providing the opportunity to build new applications. This database is queried through Cypher Query Language. The cypher language allows efficient query execution and update of graph database. It provides flexibility for complex data loading scenario (Francie *et al*., 2018).

To evaluate Neo4J data loading, we will take the result from Rosberg (2022) Tables 8 and 9 provides the data loading test with Neo4J performed by Rosberg (2022).

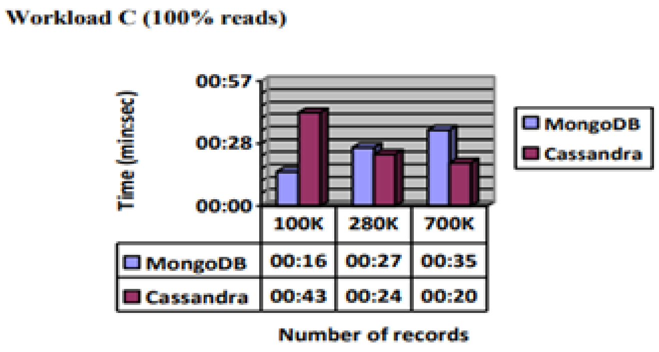
We can observe that Neo4J perform efficient data loading capabilities. The average time to load relationships and rows increases moderately with the volume of data. It takes about 1.58 sec to load 100000 relationships and 0.77 seconds to load 100000 rows. This experiment has concluded that Neo4J is a quite fast and scalable for handling large datasets loading.

When it comes to data loading times, Cassandra's architecture provides significantly to its fast data loading capabilities. Data is first written to a commit log to ensure data durability. Then, data is indexed and written into a memTable, an in-memory structure. After the memory structures become full, data is written in the form of a SSTable data. This process keeps the write operations



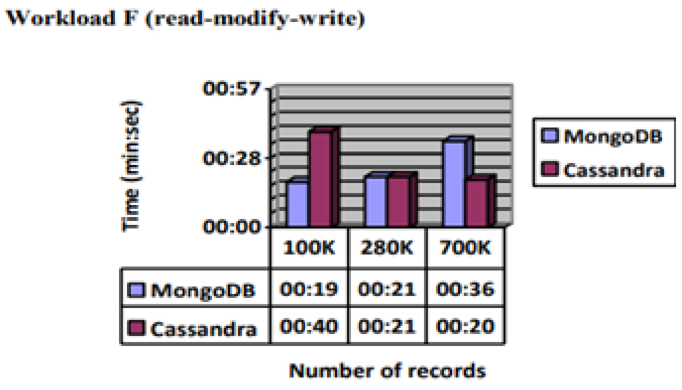
No_of_objects	MySQL:S0	Neo4j:S0	MySQL:S1	Neo4j:S1	MySQL:S2	Neo4j:S2
100	19.56	8	33	12.65	111.334	19.57
500	281.38	10	333.96	17	620.56	21

Figure 5: Experiment results. (Batra & Tyagi, 2012).



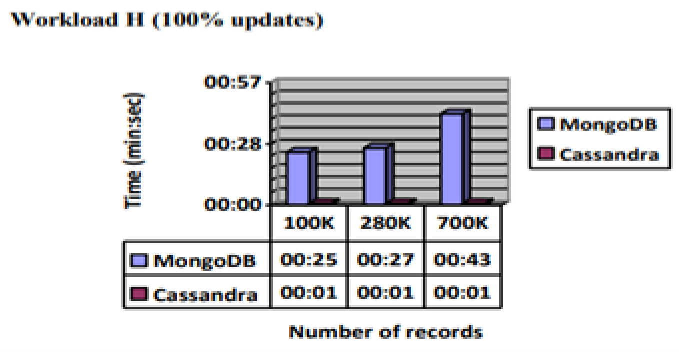
	100K	280K	700K
MongoDB	00:16	00:27	00:35
Cassandra	00:43	00:24	00:20

Figure 6: Read operation results. (Abramove & Bernardino, 2013).



	100K	280K	700K
MongoDB	00:19	00:21	00:36
Cassandra	00:40	00:21	00:20

Figure 7: Read-Modify-Write operation results. (Abramove & Bernardino, 2013)



	100K	280K	700K
MongoDB	00:25	00:27	00:43
Cassandra	00:01	00:01	00:01

Figure 8: Update operation results. (Abramove & Bernardino, 2013)

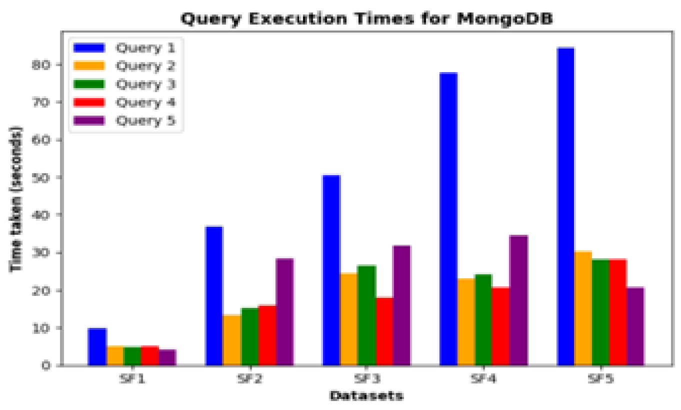


Figure 9: Query execution time for MongoDB. (Mohan et al., 2024)

Table 8: Import Nodes.

Number of row (n)	First test (s)	Second test (s)	Third test (s)	Average time (s)
100	0.012	0.016	0.011	0.013
1000	0.033	0.023	0.02	0.02533333333
10000	0.052	0.045	0.047	0.048
50000	0.272	0.23	0.282	0.26133333333
100000	0.996	0.683	0.623	0.76733333333

Sources: Rosberg, 2022.

fast, resulting in the relatively short data loading times (Jyothi, 2022).

Table 10 summarizes the load times for different datasets across different databases; this can be referred for MongoDB and Redis' data loading time.

We can observe that MongoDB has longer data loading times. Due to increased indexing and write operations overhead and its limited sharding capabilities have possibility led to increased data ingestion. MongoDB's reliance on memory-mapped files for storage may result in higher disk I/O operations and worse write performance as dataset sizes increase (Mohan *et al.*, 2024).

Table 9: Import Relation.

Number of row (n)	First	Second	Third	Average
100	0.043	0.029	0.023	0.03166666667
1000	0.055	0.046	0.047	0.04933333333
5000	0.082	0.083	0.086	0.08366666667
10000	0.213	0.246	0.203	0.2206666667
50000	0.754	0.784	0.704	0.7473333333
100000	1.437	1.582	1.732	1.583666667
309357	5.217	5.288	5.363	5.289333333

Sources: Rosberg, 2022.

Table 10: Data loading time.

Databases	SF1	SF2	SF3	SF4	SF5
PostgresSQL	37s	275s	857s	1089s	1481s
MongoDB	90s	1250s	1701s	2275s	2810s
ArangoDB	295s	2249s	3964s	12169s	15162s
Redis	1495s	3245s	5023s	7748s	10289s
Apache Kudu	42s	95s	146s	192s	240s

Sources: Mohan *et al.*, 2024.

In the same experiment, Redis displays the slowest performance in data loading times across all datasets, which can be referred to in Table 10. The authors aggregated data features into a single list, resulting in records with a single key and 40 value. Since Redis is single threaded, it requires two operations for each insertion. With data scaling, it will increase the load time for Redis (Mohan *et al.*, 2024).

Data Manipulation

Data manipulation is a vital mechanism in databases to ensure data is easily accessible and manageable. It supports flexibility and scalability as the data growth when an organization expands.

Oracle has two architectures to consolidate data manipulation capabilities to ensure the data is always available, consistent and protected during complex operations.

Data Guard is a high availability, data protection and discover recovery architecture. It provides simplicity, performance and reliability. It helps offload backup work from production database while protect against data loss and downtime at the same time. Data Guard will automatically resynchronize (resynch) the standby using archived redo generated at primary database if there is any failure (Nawaz & Soomro, 2013). Figure 11 illustrates an Active Data Guard.

Besides that, Oracle also offers backup recovery architecture called Real Application Clusters (RAC). RAC is a cluster database

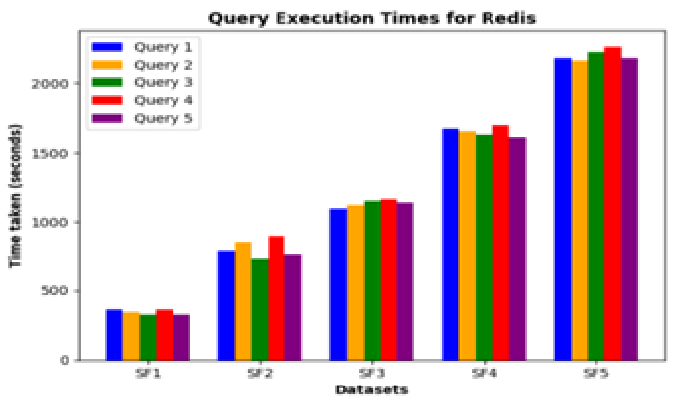


Figure 10: Query execution time for Redis. (Mohan et al., 2024)

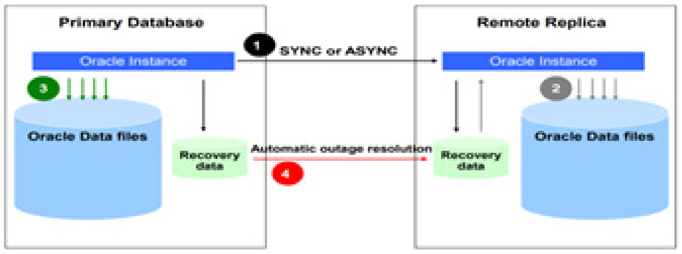


Figure 11: Active Data Guard. (Walters, 2011)

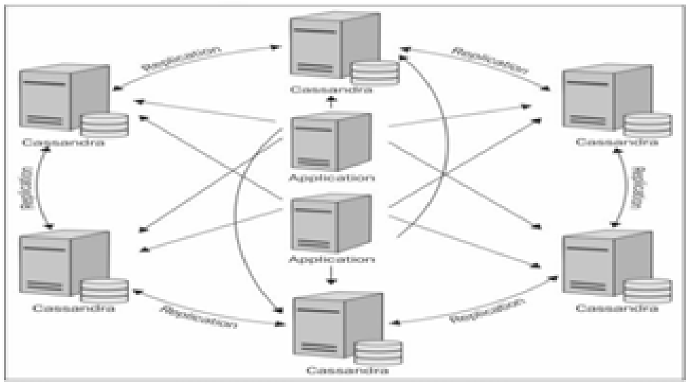


Figure 12: Data Replication in Cassandra. (Gundigara & Mehta, n.d)

with a shared cache architecture that provides highly scalable and available database solutions. It protects against hardware failures and operating system or server crashes. In the case of node failover, service connections are seamlessly redirected to another node, users can continue to access the service (Kadam *et al.*, 2011).

Neo4J excels in handling complex relationships and large volumes of connected data. Neo4J has high availability due to its unique characteristics, master-slave cluster (Lopez & Cruz, 2015). It follows a non-structured repository model, which allow for flexible manipulation of nodes. The performance of Neo4J is outstanding as it seeks on the set of nodes linked, providing a shorter path for queries (Blimm & Horrocks, 2004).

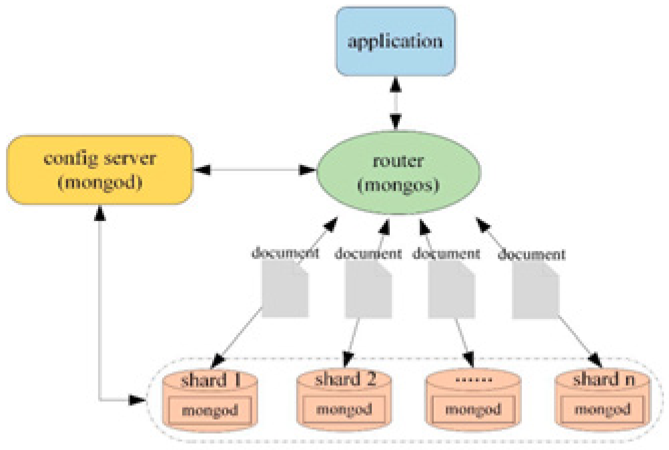


Figure 13: Sharding techniques (Shuang, 2019).

However, Neo4J has weaknesses in data manipulation. It struggles with horizontal scalability compared to other database systems and has an upper limit on the size of the graph. Neo4J's writing performance can be bottleneck, especially for very heavy writing operations. It requires the ability to distribute data across multiple machines. Moreover, Neo4J can be resource-intensive in memory and CPU usage. Large graphs and complex queries can consume major memory (Pokorný, 2015).

Following, Cassandra is a highly scalable database that can manage large amounts of data across many commodity servers. It ensures high availability with a symmetric architecture that has no single point of failure and replicates data across multiple nodes. Figure 12 illustrates Cassandra's data replication process. Cassandra uses the Gossip Protocol, a peer-to-peer mechanism to allow the nodes to communicate and detect any faulty nodes in the cluster (Gundigara & Mehta, n.d).

To evaluate Cassandra scalability, Popescu and Radu (2020) have tested it with other databases in benchmarking with YCSB framework. Table 11 summarizes the results of the benchmark throughput and latency.

Cassandra delivers the highest performance across all dataset sizes. Cassandra has the lowest latency with below 70 milliseconds even at a massive 1TB scale. Overall, Cassandra demonstrated the most consistent and impressive scalability in intensive read-write workloads on huge dataset.

MongoDB offers high scalability and performance for handling large-scale deployments. It leverages capabilities to achieve low latency which can be noticed in the Table 11 as well. MongoDB improves reliability and support horizontal scaling with two distribution models. Firstly, MongoDB supports replication through replica sets. These replica sets create multiple copies of data on different servers, to ensure data redundancy and high availability. In addition, automatic failover keeps the

Table 11: Benchmark throughout and latency.

Database	10GB	100GB	1TB
MongoDB	Throughput: 15,000 ops/sec	Throughput: 18,500 ops/sec	Throughput: 22,000 ops/sec
	Latency: 68 ms	Latency: 72 ms	Latency: 82 ms
Cassandra	Throughput: 18,200 ops/sec	Throughput: 24,000 ops/sec	Throughput: 29,000 ops/sec
	Latency: 52 ms	Latency: 58 ms	Latency: 62 ms
HBase	Throughput: 12,000 ops/sec	Throughput: 16,300 ops/sec	Throughput: 17,200 ops/sec
	Latency: 78 ms	Latency: 86 ms	Latency: 92 ms
Couchbase	Throughput: 10,500 ops/sec	Throughput: 14,800 ops/sec	Throughput: 17,200 ops/sec
	Latency: 72 ms	Latency: 78 ms	Latency: 83 ms

Sources: Popescu & Radu, 2020.

Table 12: Strength and weaknesses for comparison.

Factor	Oracle (RDBMS)	Neo4J (Graph)	Cassandra (wide column)	MongoDB (Document-Oriented)	Redis (Key-value)
Security	Strong with user identity at database level and role authority. Strong access control.	No data-level security and encryption.	No data-level security and encryption. Work performance in client/server and inter-cluster authentication.	No data-level security and encryption. Work performance in client/server and inter-cluster authentication.	No data-level security and encryption for authentication.
Data Retrieval	Slowest, with stable and predictable times. (slower than non-RDBMS).	Good for data retrieval as it does not scan the whole graph.	Good for read-only, read-modify-write, update only. Performance improves as data size increases.	Good performance in execution speeds.	Slower execution conquered to NoSQL. Performance reduces when dataset size increases.
Creation	Performs well with large data volumes from flat files.	Performs well for rapid data ingestion.	Fast loading times due to its architecture.	Long data loading times due to limited sharding.	Slowest data loading times due to single threading.
Manipulation	Supports vertical and horizontal scalability. High availability, data protection and isolation.	Struggles with horizontal scalability, having upper limit on size of graph. High availability through master-slave cluster architecture.	High scalable across many commodities servers. High availability due to symmetric architecture with low failure.	High scalable through sharding. Availability through primary node selection.	High scalable through distribution of nodes. High availability.

system running smoothly. It enhances reliability and minimizes downtime (Panpaliya, 2012).

Apart from that, sharding techniques are used by MongoDB to achieve data distribution and load balancing. This technique divides data into smaller chucks among several nodes. It allows multiple nodes to manage to read and write requests concurrently, which will improve performance and scalability (Shuang, 2019). Figure 13 illustrates the sharding techniques.

Lastly, Redis is a powerful in-memory database that support various data manipulation operations. It excels in fast-read operations and can effectively modify large objects with low latency (Chen, 2016). Redis has an extensive range of data structures and includes basic operations available in Redis Enterprise. It is an enhanced version of Redis. It expands Redis capabilities to high availability with features like active geo-replication and operation-based CRDT (House *et al*., 2021).

Although Redis can perform reliably as a single-node service, its clustering service faces challenges with reliability and scalability based on the summary given by (Spal & Kaur, 2018). Decentralized design can lead to performance issues due to an inefficient data indexing mechanism. Additionally, node failures are more common in the distributed setup. Redis's partial synchronization strategy also carries the risk of data loss during system crashes.

Scenarios

The following scenarios will allow readers to understand the different models and their mechanisms better in different applications. For example, comparing a hospital database system and a social media database system. Both databases share a common characteristic to manage large volumes and time-critical data, requiring quick data performance and high security.

Social media generates a large unmeasurable volume of data, known as big data to most. That is because of their constant real-time users throughout the day, who are generating billions of data every second. In this case, database models like Neo4J, Cassandra and MongoDB would be much suited as they have high-performing data creation and retrieval that can handle the real-time large volume of data, as compared to Redis and Oracle. However, if we were to look at a hospital database, they would require a high to medium query performance, especially since hospitals are not constantly creating or retrieving data daily and does not carry as much volume of data as social media. In this case, any of the five models would be suitable for use as their data creation and retrieval are on par.

However, when taking data security into consideration, the model selection for a hospital database would narrow down to Oracle. This is because Oracle has strong access control and multiple levels of authorization. It also uses ACID compliance and revision control, ensuring no loss of validity for data. The four NoSQL models on the other hand tend to have weaker data-level security and encryption, making personal data vulnerable. Similarly, security would be a concern for social media data as well, however social media personal data are less risky compared to a hospital's, hence Neo4J, Cassandra and MongoDB can still be used with blockchain implementation to enhance security.

If we look at an application caching, Redis would be the suitable model for it. This is because its in-memory store can allow quick data manipulation and retrieval, as compared to the other models. Oracle has a likely chance to encounter bottleneck while the other models require a lot of query performance to ensure efficiency. Redis also can scale up to enterprise availability and scalability, however although it sounds ideal for a hospital and social media database, Redis' slow data creation is a loss for other applications. Hence, it is mostly used for application cache, quick response database and data analytics.

Objective Question

Based on the results above, we can answer our objective question that was stated above, in the following:

RQ1. Which database model is suitable for different characteristics in of application scenario?

Different application is ideal with certain model. Based on our findings, we observed that Oracle is more suitable for critical business data, especially where security is concerned. Neo4J is most suited for cloud management and social networking. Both Cassandra and MongoDB are ideal for social media, finance and real-time systems. Lastly, Redis is preferable for quick-response system such as e-commerce query search.

RQ2. What are the key differences between relation databases and a nonrelational database?

Based on the studies presented in this research, RDBMSs are slower when it comes to outright performance compared to non-RDBMSs, at the cost of consistency. In addition, RDBMSs are more suited for applications that call for strong data integrity, whereas non-RDBMSs are more suited towards real time applications that call for performance that delivers when needed (Key-value), large-scale storage solutions (Wide-column), network analysis (Graph) and content management or web applications (Document-oriented).

RQ3. What are the strengths and weaknesses based on several factors for comparison?

Table 12 summarizes the strengths and weaknesses for comparison that answer our research question 3.

DISCUSSION

The comparison of these five database systems highlights the factors to consider when selecting the right database solution for any organization:

Security: Oracle offers the strongest data-level security with access control, crucial for safeguarding sensitive user data for any applied scenario.

Performance: Oracle is slower than NoSQL but has stable performance. Neo4J is faster for simple and less complex queries. Cassandra outperforms MongoDB while handling data growth, but MongoDB handles most requests during scaling. Redis performs the slowest, especially with large datasets.

Integrity: Oracle and Neo4J appears to be reliable for data integrity. Oracle's Data Guard feature ensures stability, while Neo4J's speed is crucial for ensuring isolation from underlying corruption and speed respectively.

Manipulation: Oracle and Cassandra highly support scalability and data distribution, essential for user growth while maintaining reliability. MongoDB and Redis also excel in scalability through sharding and node distribution.

Therefore, these presented results imply several key insights:

The Need to Evaluate Trade Offs to Better Suit Needs

Each database system has own strengths and drawbacks across each measured performance factor.

This implies there is no perfect database solution and organizations need to carefully evaluate the pros and cons of each system to find out what suits their business model and use case.

Complementary Solutions

Since the result yield shows that different database systems excel in different areas, organizations might benefit from adopting multi-database solutions where each strength of the database systems can be used to address diverse requirements if needed.

Importance of Scalability

As active user counts increase globally, so does the amount of data that is collected and used for business insights. Therefore, scalability is an important factor to consider for organizations, as it is paramount for them to accommodate increasing data volumes and user base without hiccups.

Continuous Evaluation

As database technologies continue to evolve, organizations should periodically evaluate their database solutions to ensure alignment with evolving business needs and technologies.

Previous studies have provided insights into database security, performance, integrity, availability and scalability. This research extends these. This research builds upon and extends upon prior research by comparing all 5 database models instead of a pair of databases. These findings offer several contributions highlighting the unique strengths of NoSQL and SQL models along with RDMSs and non-DBMSs. Moreover, we learned about the methods employed to ensure data integrity, such as sharding that is employed by MongoDB to ensure scalability and Oracle's Data Guard which employs in memory database replication.

The limitations in the research paper includes resource constraints, data model differences, and limited research on certain databases. We faced difficulty in finding existing research and gathering broad data information on certain models such as Neo4J and Redis. Compared to other data model, Neo4J being relatively new chapter in graph-based model domain. There is a limitation in research on Neo4J available, making it difficult to find detailed insights and experiences from researchers. Similarly, Redis has fewer researchers compared to other models like Cassandra and MongoDB. This lack of relative research may limit our ability to provide deeper insights.

Another limitation is the variation in operations tested across research. Some focuses on read operation, write operation, update operations or mixed operations. This inconsistency makes it difficult to determine which data model is good in query performance, only assessing how well each model performs under certain types of operations. This limitation hinders our ability to draw a complete conclusion about the overall query performance.

As part of future work, we propose to focus on adding security technologies into different data models to enhance data protection. We also plan to explore new way to maintain data integrity such as backup discovery. In term of query performance, we can apply Artificial Intelligence (AI) and Machine learning technique to discover predictive insights. The comparison of data availability in on-premises and cloud-based environments is another area of our interest. We are also interested in doing research on scalable architecture designs that support the growth of data. Lastly, research can be extended across various hybrid data models to identify their implementations.

CONCLUSION

Consistent with our expectations, the primary outcomes of this study are that each database system - Oracle, Neo4J, Cassandra, MongoDB and Redis - are ideal in different scenarios, as outlined in our analysis. This study contributes insights to DBMS knowledge offering a practical guide for organizations to optimize their database solutions. While our study provides a solid foundation, future research should focus on emerging technologies, such as graph databases and NoSQL variants, to ensure that organizations are well-equipped to address the evolving demands of data management.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest.

ACKNOWLEDGEMENT

The authors of this paper appreciate the help given by individuals and institutions that led to the completion of this research. Researchers are grateful to lecturer Dr. Satishkumar VE for the continuous feedback that contributed to the improvement of this paper. Furthermore, researchers also extend their appreciation to Sunway University for providing a holistic research environment that fosters knowledge and academic growth.

ABBREVIATIONS

AAA: Authentication, authorization, auditing; **ACID:** Atomicity, Consistency, Isolation, and Durability; **AI:** Artificial Intelligence; **ANSI-C:** American National Standards Institute C language; **API:** Application programming interface; **BSON:** Binary JavaScript Object Notion; **CA:** Certificate authority; **CF:** CompactFlash; **CPU:** Central processing unit; **CQL:** Cassandra Query Language; **CRDT:** Conflict-Free Replicated Data Type; **DB:** Database; **DBMS:** Database Management Systems; **DCL:** Data Control Language; **DDL:** Data Definition Language; **DML:** Data Manipulation Language; **DQL:** Data Query Language; **ERD:** Entity Relationship Diagram; **I/O:** Input/Output; **NoSQL:** not only SQL; **OS:** Operating system; **OOTB:** Out-of-the-box; **RAC:** Real Application Cluster; **RDBMS:** Relational database management system software; **REST:** Representational State Transfer; **RQ:** Research question; **SQL:** Standard Query Language; **SSL:** Secure Sockets Layer; **SSTable:** Sorted Strings Table; **TCL:** Transactional Control Language; **UML:** Unified Modeling Language; **YCSB:** Yahoo! Cloud Serving Benchmark.

REFERENCES

Abramove, V. & Bernardino, J. (2013). NoSQL databases: MongoDB vs Cassandra. *C3S2E13: International C* Conference on Computer Science and Software Engineering*, *Porto, Portugal*.10.1145/2494444.2494447

Alkoshman, M.M. (2015). Unified Modeling Language and Enhanced Entity Relationship: An Empirical Study. *International Journal of Database Theory and Application*, *8*(3), 215–227. https://doi.org/10.14257/ijdta.2015.8.3.18.

Baggia, A., Mali, A., Grlica, A., & Leskovar, R. (2018). Oracle APEX in Higher Education. *37th International Conference on Organizational Science Development, Portoroz, Slovenia*. https://www.researchgate.net/publication/324911051_Oracle_APEX_in_Higher_Education

Batra, S., & Tyagi, C. (2012). Comparative Analysis of Relational and Graph Databases. *International Journal of Soft Computing and Engineering (IJSCE)*, *2*(2). https://citeseerx.ist.psu.edu/document?repid=rep1&andtype=pdf&anddoi=d9e4877d25fe4138196719f5b64255f7d71c91b5

Carlson, J. (2013). *Redis in Action* (1st ed). Manning Publications. https://books.google.com.my/books/about/Redis_in_Action.html?id=xjszEAAAQBAJ&redir_esc=y

Čerešňák, R., & Kvet, M. (2019). Comparison of query performance in relational a non-relation database. *Transportation Research Procedia*, *40*, 170–177. 10.1016/j.trpro.2019.07.027

Chen, P. P.-S. (1976). The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, *1*(1), 9–36. https://doi.org/10.1145/320434.320440.

Chen, S., Tang, X., Wang, H., Zhao, H., & Guo, M. (2016). Towards Scalable and Reliable In-Memory Storage System: A Case Study with Redis. *2016 IEEE Trustcom/BigDataSE/ISPA*. 10.1109/TrustCom.2016.0255

Chinnachamy, A. (2013). *Redis Optimization How-to* (1st ed.). Packt Publishing. https://ebookcentral.proquest.com/lib/sunway/reader.action?docID=1215005

Elmasri, R., & Navathe, S.B. (2016). *Fundamentals of Database Systems* (7th ed.). Upper Saddle River, NJ, USA: Pearson. https://www.auhd.edu.ye/upfiles/elibrary/Azal2020-01-22-12-28-11-76901.pdf

Fehily, C. (2020). *SQL Database Programming (Fifth Edition*). USA: Questing Vole Press.

Francie, N., *et al.* (2018). Cypher: An Evolving Query Language for Property Graph. *SIGMOD '18: Proceedings of the 2018 International Conference on Management of Data*. https://doi.org/10.1145/3183713.3190657

Glimm, B., & Horrocks, A. (2004). *Query answering systems in the semantic web*. Oxford University Research Archive. https://ora.ox.ac.uk/objects/uuid:7d3184ca-fd69-4813-9e9e-4a9b593e1296/files/m7db6dbeb5578c838c0219322e513eca3

Gundigara, K.U., & Mehta, V.H. Cassandra as a Big Data Modeling Methodology for Distributed Database System. *International Journal of Engineering development and research (IJEDR)*, *5*(3). https://www.ijedr.org/papers/IJEDR1703135.pdf

Gupta, N., & Agrawal, R. (2018). Chapter Four-NoSQL Security. *Advances in Computers, Elsevier Science*, *109*, 101-132. 10.1016/bs.adcom.2018.01.003

Hammerschmied, G., & Bork, D. (2022). *Multi-Notation Support for a Hybrid VS Code Modeling Tool*. TU Wien. https://model-engineering.info/publications/theses/thesis-hammerschmied.pdf.

Harold, G. Ed.D. (2024). Database. *Salem Press Encyclopedia* [Press release]. https://research.ebsco.com/c/lzo7om/viewer/html/kummripqzv

House, D., Kuang, H., Surendran, K., & Paul, C. (2021). Toward Fast and Reliable Active-Active Geo-Replication for a Distributed Data Caching Service in the Mobile Cloud. *Procedia Computer Science*, *191*(7), 119-126. 10.1016/j.procs.2021.07.018

Ilić, M., Kopanja, L., Zlatković, D., Trajković, M. & Ćurguz, D. (2021) Microsoft Sql Server and Oracle: Comparative Performance Analysis. *7th International conference Knowledge management and informatics*. https://kmi.vtsns.edu.rs/KMI_2021/radovi/1-KMI_Informatika/KMI_informatika-1.5.pdf

Jatana, N., Puri, S., Ahuja, M., Kathuria, I., & Gosain, D. (2012). A Survey and Comparison of Relational and Non-Relational Database. *International Journal of Engineering Research & Technology (IJERT)*, *1*(6). https://www.academia.edu/download/76957411/a-survey-and-comparison-of-relational-and-non-relational-database.pdf

Jyothi, J. (2022). Cassandra is a Better Option for Handling Big Data in a NoSQL Database. *International Journal of Research Publication and Reviews*, *3*(9), 880-883. 10.55248/gengpi.2022.3.9.27

Kadam, D., Bhalwarkar, N., Neware, R., Sapkale, R., & Lamge, R. (2011). Oracle Real Application Clusters. *International Journal of Scientific and Engineering Research*, *2*(6). https://www.bibme.org/apa/journal-citation/search?q=https%3A%2F%2Fwww.researchgate.net%2Fpublication%2F316510400_Oracle_Real_Application_Clusters

Khan, W., *et al.* (2023). SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review. *Big Data Cognitive Computing*, *7*(2), 97. 10.3390/bdcc7020097

Kolonko, K. (2018). Performance Comparison of the Most Popular Relational and Non-Relational Database Management Systems. *Blekinge Tekniska Högskola*. https://www.diva-portal.org/smash/get/diva2: 1199667/FULLTEXT02.pdf

Liu, Y., Kumar, R., Tripathi, A., Sharma, A., & Rana, M. (2022). The Application of Internet of Things and Oracle database in the research of intelligent data management system. *Informatica*, *46*(3). 10.31449/inf.v46i3.4019.

Lopez, F.M.S., & Cruz, E.G.S.D.L. (2015). Literature Review about Neo4J Graph Database as a Feasible Alternative for Replacing RDBMS. *Journal of the Faculty of Industrial Engineering*, *18*(2), 135. 307180380_Literature_review_about_Neo4j_graph_database_as_a_feasible_alternative_for_replacing_RDBMS

Mehrabani, A. (2014). *MongoDB High Availability* (1st ed.). Packt Publishing, 2014. https://ebookcentral.proquest.com/lib/sunway/detail.action?docID=1753325

Minder, J., Brandenberger, L., Salamanca, L., & Schweitzer, F. (2024). Data2Neo-A Tool for Complex Neo4J Data Integration. *arXiv*. arXiv:2406.04995

Mohan, R.K., Kanmani, R.R.S., Ganesan, K.A., & Ramasubramanian, N. (2014). Evaluating NoSQL Databases for OLAP Workloads: A Benchmarking Study of Mongodb, Redis, Kudu and Arangodb. *arXiv*. arXiv:2405.17731

Nawaz, R., & Soomro, T.R. (2013). Role of Oracle Active Data Guard in High Availability Database Operations. *International Journal of Applied Information System (IJIAS)*, *5*(5). 10.5120/ijais13-450914

Noiumkar, P. & Chomsiri, T. (2014). A Comparison the Level of Security on Top 5 Open Source Nosql Databases. *The 9th International Conference on Information Technology and Applications (ICITA2014)*, *Sydney, Australia*. https://www.researchgate.net/publication/301633978_A_Comparison_the_Level_of_Security_on_Top_5_Open_Source_NoSQL_Databases

Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., & Abramov, J. (2011). Security Issues in NoSQL Databases. *Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, USA*, 541-547. 10.1109/TrustCom.2011.70

Panpaliya, T. (2012). Benchmarking Mongodb Multi-Document Transactions in A Sharded Cluster. *Master's Projects*. 10.31979/etd.ykxw-rr89

Pokorný, J. (2015). Graph Databases: Their Power and Limitations. *IFIP International Conference on Computer Information Systems and Industrial Management*. 10.1007/978-3-319-24369-6_5

Popescu, E., & Radu, A. (2020). A Comparative Study of Scalability and Performance in NoSQL Databases for Big Data Storage and Retrieval. *International Journal of Social Analytics (IJSA)*, *5*(12), 16–27. https://norislab.com/index.php/IJAHA/article/view/44

Punit, Shrivastava, V., & Pandey, A. (2014). A Comprehensive Study on Modern Database Management Systems. *International Journal of Research Publication and Reviews*, *5*(4), 2133-2136. https://ijrpr.com/uploads/V5ISSUE4/IJRPR24839.pdf

Rosberg, O. (2022). An Interactive Web Tool for Importing Data to a Graph Database. *Dissertation*. https://www.diva-portal.org/smash/get/diva2: 1679807/FULLTEXT01.pdf

Sebastian-Coleman, L. (2022). *Meeting the Challenges of Data Quality management* (1st ed.). Academic Press. https://doi.org/10.1016/C2019-0-03993-3

Shkokani, M., & Altamimi, A.M. (2020). Graph Database Security: Blockchain Solution and Open Challenges. *International Journal of Simulation: Systems, Science & Technology*. 10.5013/IJSSST.a.21.01.09

Shuang, W., *et al.* (2019). A Distributed Storage and Access Approach for Massive Remote Sensing Data in MongoDB. *International Journal of Geo-Information*, *8*(12), 533. 10.3390/ijgi8120533

Walters, G. (2011). *Oracle Active Data Guard – Overview*. Indiana Oracle Users Group. https://dmdc.unimap.edu.my/images/pdf/Active-Data-Guard-Overview.pdf

What is a Document Database? (2024, July 19). *MongoDB*. https://www.mongodb.com/resources/basics/databases/document-databases#:~:text=An%20intuitive%20data%20model%20that,to%20evolve%20as%20application%20needs

Zhang, H., Chen, Z., Ooi, B.C., Tan K.L., & Zhang, M. (2015). In-Memory Big Data Management and Processing: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, *27*(7), 1920-1948. 10.1109/TKDE.2015.2427795.

Cite this article: Sab CM, Ahmed KKM Scientometric Insights into Social Media Analytics: Trends and Impact in Communication Studies. Info Res Com. 2024;1(2):104-19.