**Information Research Communications, 2025; 2(1):47-70.**
https://inforescom.org

Research Article

# Comparative Analysis of SQL and NoSQL Databases: Data Models, Use Cases, and Performance Insights

**Dionne Teh Wooi Ying[1], Choo Yan Jie[1], Cheah Shaoren[1], Hong Chang Eui[1], Chin Wey Ken[1], Sathishkumar Veerappampalayam Easwaramoorthy[1,*], Usha Moorthy[2]**

[1]School of Computing and Artificial Intelligence, Faculty of Engineering and Technology, Sunway University, No. 5, Jalan Universiti, Bandar Sunway, Selangor Darul Ehsan, MALAYSIA.
[2]Department of Information Technology, Manipal Institute of Technology Bengaluru, Manipal Academy of Higher Education, Manipal, Karnataka, INDIA.

## ABSTRACT

**Aim/Background:** The primary aim of this study is to conduct a comparative analysis of SQL and NoSQL databases based on their data models, performance characteristics, and suitability for various application scenarios. It specifically investigates relational, key-value, graph, document, and wide-column models, focusing on their operational implications, such as data integrity, query performance, scalability, and security. **Methodology:** This research adopted a mixed-methods approach, combining qualitative and quantitative evaluations. It involved literature review, official DBMS documentation, and performance benchmarking. The study utilized five DBMSs-Oracle (SQL), Neo4j (graph), Cassandra (wide-column), Redis (key-value), and MongoDB (document). Performance metrics like data creation, manipulation, retrieval, access control, and data integrity were analyzed. Scenario-based analyses (e-commerce and social media analytics platforms) were used to examine database suitability under different real-world conditions. **Results:** The results indicated that NoSQL databases generally outperformed SQL databases in terms of scalability, data flexibility, and runtime performance, especially under large-scale data operations. SQL databases like Oracle demonstrated strong data integrity and complex querying capabilities but lagged in scalability and schema flexibility. NoSQL databases like MongoDB and Neo4j provided ACID compliance with dynamic schemas, while Redis and Cassandra excelled in high-speed data operations with eventual consistency. Scenario analysis confirmed the contextual suitability of each model. **Discussion:** While NoSQL databases offer superior performance for unstructured data and scalable applications, SQL databases remain indispensable for structured, transaction-heavy systems due to their robust consistency and integrity mechanisms. The preference for a database model depends heavily on application context, data structure, and performance requirements. The findings highlight that no single model is universally superior; rather, optimal selection depends on the specific use case. **Conclusion:** The study concludes that both SQL and NoSQL databases have distinct strengths and weaknesses. SQL databases are best suited for structured, transactional systems, whereas NoSQL models are ideal for modern, data-intensive, and scalable applications. Organizations should evaluate their specific data requirements and system demands when selecting an appropriate database solution.

**Keywords:** SQL, NoSQL, Oracle, Cassandra, Redis, Neo4j, MongoDB, Relational model, Key-value model, Graph model, Wide-column model, Document model.

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

**Received:** 07-02-2025;
**Revised:** 28-03-2025;
**Accepted:** 19-05-2025.

## INTRODUCTION

Modern systems and applications are evidently reliant on data. This raises the importance of implementing efficient and effective database as there is a vast amount of information to be managed and analyzed. There are two main database types: SQL (Structured Query Language) and NoSQL (Not Only SQL). SQL databases, which are characterized by their relational model and adherence to ACID (Atomicity, Consistency, Isolation and Durability) properties, excel in handling structured data with complex relationships and transactions. Conversely, NoSQL databases offer flexibility and scalability for unstructured and semi-structured data which is suitable for modern applications with dynamic and varied data needs. NoSQL databases come in a plethora of models including key-value, document, wide column, and graph. Deciding between a SQL database and the various types of NoSQL database is a critical decision to organizations

**DOI:** 10.5530/irc.2.1.5

as it brings an impact on performance, scalability, and data management strategies.

## Goals

The aim of this report is to analyse and compare SQL and NoSQL databases from multiple aspects by answering 3 questions:

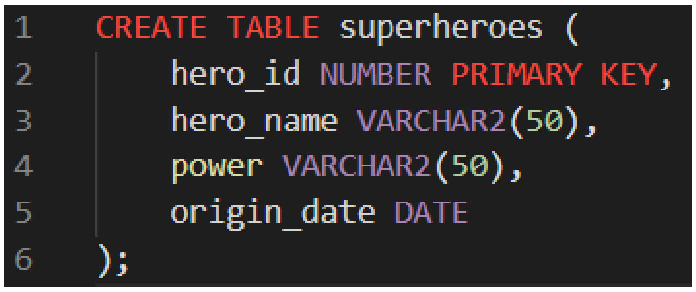
- What is the fundamental difference between SQL and NoSQL databases?
- How do SQL and different types of NoSQL databases differ in terms of data creation, manipulation, integrity, retrieval and access control across different use cases?
- In certain scenarios, what factors are preferred to NoSQL databases over SQL databases?

The first is to explore the fundamental differences between SQL and NoSQL databases. This includes analysing the underlying principles, structure, and data models. Next, we will investigate how SQL and different types of NoSQL databases differ in terms of data creation, manipulation, integrity, retrieval and access control. For comparison, we will discuss the differences between the two database types in detail through two scenario cases, which is e-commerce and social media analytics platform, and discuss in which situations each database is more appropriate. Lastly, we want to discover the factors that favor NoSQL databases over SQL databases in certain scenarios. This will be useful for identifying situations where NoSQL can be more advantageous than SQL databases.

## Importance of databases and importance of reporting

In modern information systems, databases play a critical role. Beyond just storing data, databases help various applications or systems operate smoothly by providing functions such as search, management, modification, and deletion. Because each database has its own advantages and disadvantages, it is important to choose according to its purpose and environment.

Our findings can be utilised by database administrators and developers to guarantee effective data management and optimize database systems. Additionally, it may help others in designing and implementing suitable database solutions catered to a variety



```
CREATE TABLE superheroes (
    hero_id NUMBER PRIMARY KEY,
    hero_name VARCHAR2(50),
    power VARCHAR2(50),
    origin_date DATE
);
```

**Figure 1:** Create Table in Oracle.

of business demands by knowing the advantages and traits of NoSQL and SQL databases, respectively.

This report will take a quantitative and qualitative approach to compare the performance of SQL and NoSQL databases. Our main goal will be to comprehend database management concepts in a variety of contexts and to identify practical variations by applying current theories and models. Next, we will set up two real-world applications to assess each database model's performance based on its advantages and disadvantages. This paper will aim to offer a clear foundation for comprehending the distinctions between NoSQL and SQL databases based on the findings.

## LITERATURE REVIEW

A data model is a conceptual framework that describes how data would be structured, stored, and manipulated within a database. Data models are generally represented using various notations. SQL, or relational databases, is represented through Entity-Relationship (ER) diagrams. NoSQL databases have various representations: key-value models are often depicted with keys and values, graph models use nodes and edges to represent relationships, document models display hierarchical structures of data, and wide column models organize data into tables with rows having a variable number of columns, grouped into column families. Data models are integral to logical database design as they determine the schema and constraints that ensure data integrity and consistency. They also influence physical implementation by dictating how data is stored and accessed on hardware, affecting performance, scalability, and storage efficiency. Thus, a well-defined data model bridges the gap between conceptual data requirements and practical database architecture, guiding the development and optimization of database systems.

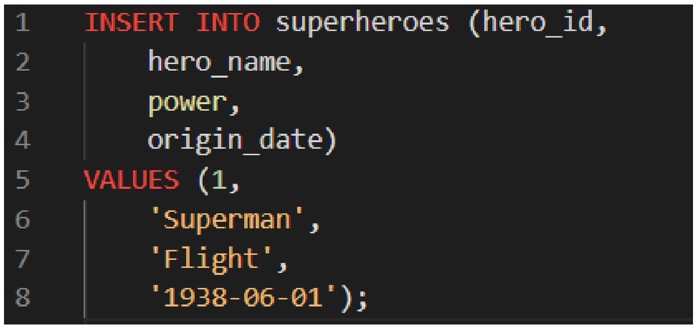
### Oracle (Relational Model)

Oracle is a famous example of SQL database that can efficiently manage and process large amounts of data using relational database models. In short, this means that the data is stored in tables that resemble spreadsheets. These tables have rows and columns. Each row is a record, such as a single entry in a spreadsheet, and each column is a property that describes the record, such as "name" or "date" (https://www.oracle.com/database/what-is-a-relational-database/).

### Data Creation

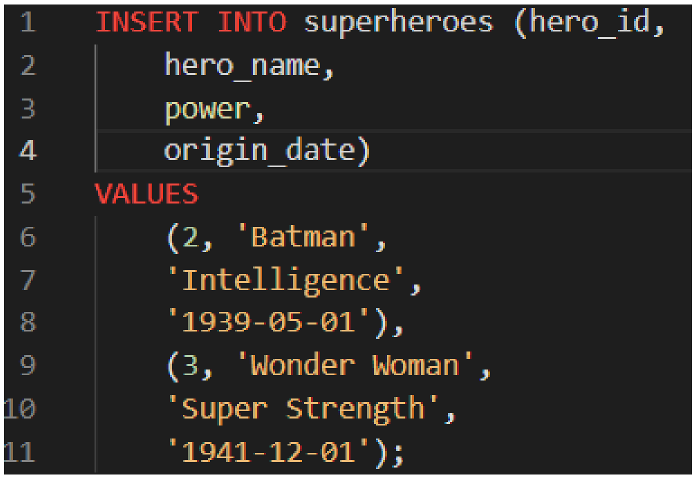
To begin creating data in Oracle, the first step is to create tables and then add data to them. You use the command CREATE TABLE, literally, to create a table.

Now, the command in Figure 1 makes a table with columns for hero ID, hero name, power, and origin date. To add data to this table, use the command INSERT INTO as shown in Figures 2 and 3.



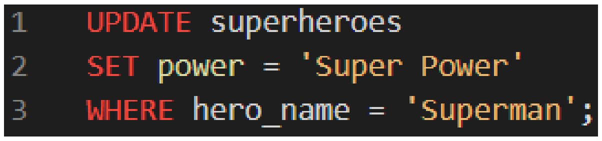
```sql
INSERT INTO superheroes (hero_id,
    hero_name,
    power,
    origin_date)
VALUES (1,
    'Superman',
    'Flight',
    '1938-06-01');
```

**Figure 2:** Insert Data in Oracle.



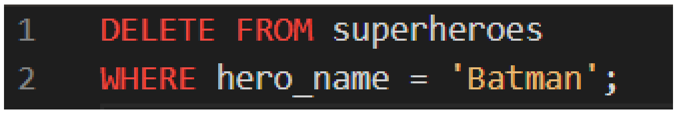
```sql
INSERT INTO superheroes (hero_id,
    hero_name,
    power,
    origin_date)
VALUES
    (2, 'Batman',
    'Intelligence',
    '1939-05-01'),
    (3, 'Wonder Woman',
    'Super Strength',
    '1941-12-01');
```

**Figure 3:** Insert Multiple Data in Oracle.



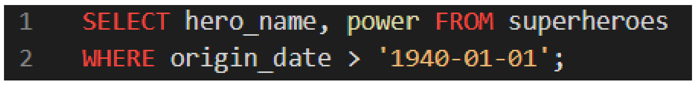
```sql
UPDATE superheroes
SET power = 'Super Power'
WHERE hero_name = 'Superman';
```

**Figure 4:** Update Data in Oracle.



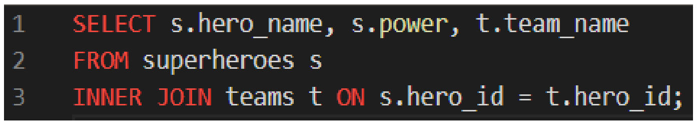
```sql
DELETE FROM superheroes
WHERE hero_name = 'Batman';
```

**Figure 5:** Delete Data in Oracle.



```sql
SELECT hero_name, power FROM superheroes
WHERE origin_date > '1940-01-01';
```

**Figure 6:** Select Data in Oracle.



```sql
SELECT s.hero_name, s.power, t.team_name
FROM superheroes s
INNER JOIN teams t ON s.hero_id = t.hero_id;
```

**Figure 7:** Select Data With Condition in Oracle.

## Data Manipulation

To change data which already exists in the table, use the command UPDATE as shown in Figure 4.

This time, use the DELETE command to remove the data as shown in Figure 5.

## Data Retrieval

Getting data out of the database is done with the SELECT command.

The command in Figure 6 shows the names and powers of superheroes that started after January 1, 1940. Sometimes, you need to get data from more than one table. This is done by joining tables.

The command in Figure 7 gets the hero names, powers, and their team names.

## Access Control

Oracle lets you create users and give them specific permissions. First you create a user to grant access to as shown in Figures 8 and 9.

Oracle also uses roles to manage permissions more easily. A role is a collection of permissions that you can assign to users. Instead of granting each user access one by one, you can create a role that is granted with multiple accesses and then grant the role to users like in Figure 10. Roles makes it easier to manage permissions as you only need to update the permissions of the role and all users with the role will have updated permissions.

## Data Integrity

Ensuring that data within a database remains accurate and consistent is called data integrity. Oracle uses several key mechanisms to maintain this feature.

The primary key ensures that each table's record is unique, preventing duplicate entries. Foreign keys maintain relationships between tables and ensure that values in one table match valid entries in another. A unique constraint ensures that all values in the column are different, and a check constraint ensures that the values in a column meet certain conditions (https://docs.oracle.com/cd/E18283_01/server.112/e16508/datainte.htm). For example, you can set conditions to ensure that a date is in the past, and so on.

In addition, the Not Null constraint does not allow NULL values in columns, so that all items have values. The Default constraint loads default values in columns when no values are specified.

Oracle also has a transaction feature which is able to maintain data integrity. When commands are executed in a transaction, all commands must execute without any failures to commit the transaction. If in any command in the transaction fails, the entire transaction will roll back and no changes will be made to the database. This ensures that the database is consistent and only updates if all the intended commands are executed. This mechanism ensures the reliability of the data stored in the database and preserves consistency to prevent errors and maintain the integrity of the information (https://docs.oracle.com/cd/B14117_01/server.101/b10743/transact.htm).

## Neo4j (Graph Model)

The graph model stores information in nodes that are connected by relationships (https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/; https://www.ibm.com/topics/nosql-databases). This model is made up of several components that share the same characteristics from relational databases (Table 1).

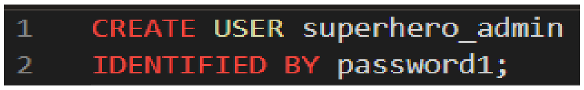
Neo4j, a leading NoSQL DBMS that utilizes graph model, is a prime example to illustrate the characteristics of this model. However, it uses a more advanced database model called **property graph** database model, where the relationships are not only connections but also carries a name and properties (Figure 11) (https://www.dataversity.net/what-is-a-property-graph). The properties are stored in key-value pairs used to describe nodes and relationships.

## Data Creation

The data is stored as a true graph model from the top to the storage level as Neo4j is a native graph database. Each node is connected with relationships to form a complex graph network in a more flexible approach, optimizing data traversal to process complex relationships among data.

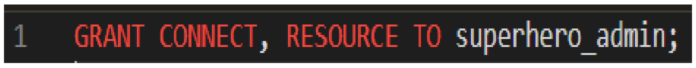
Neo4j uses their own declarative query language, Cypher, where it is similar to SQL but more efficient by supporting expressive queries to explore unknown data connections and clusters in the graph model (https://neo4j.com/docs/cypher-manual/current/introduction/cypher-overview/). In the context of data creation, Neo4j populates the database by defining the nodes and their relationships using Cypher in ascii-art type of syntax (https://neo4j.com/docs/cypher-manual/current/introduction/cypher-overview/).

(nodes)-[:CONNECT_TO]→(otherNodes)



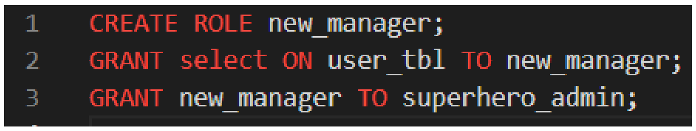
```
CREATE USER superhero_admin
IDENTIFIED BY password1;
```

**Figure 8:** Create User in Oracle.



```
GRANT CONNECT, RESOURCE TO superhero_admin;
```

**Figure 9:** Grant Access in Oracle.



```
CREATE ROLE new_manager;
GRANT select ON user_tbl TO new_manager;
GRANT new_manager TO superhero_admin;
```

**Figure 10:** Create Role and Assign Role to User in Oracle.

Unlike traditional relational databases, the graph model in Neo4j is schema-flexible, meaning that it offers a greater degree of flexibility as each node or relationship does not have a specific property defined (https://neo4j.com/docs/cypher-manual/current/introduction/cypher-overview/). Thus, it typically does involve schema implementation which creates a fixed structure of tables to store data as compared to SQL databases (Figure 12). Users can add new attributes and relationships as the graph expands and evolves.

## Data Manipulation

Data manipulation is also achievable through the use of Cypher by using syntaxes like SET and DELETE to manipulate the property or relationships of nodes. Unlike SQL databases, it does not have a fix schema which uses the MATCH syntax to determine the node to be manipulated in the graph network by using pattern matching (Figures 13-15).

Within the property graph model database of Neo4j, data manipulation is almost similar to SQL. However, Neo4j's schema is much more flexible. When deleting nodes, its related relationships are detached from other nodes to ensure data accuracy and integrity. This flexibility allows the graph schema to adapt over time and changes to add or remove new relationships according to the user needs (https://neo4j.com/docs/getting-started/get-started-with-neo4j/graph-database/).

## Data Retrieval

Neo4j also supports CRUD operations with the use of Cypher. The data in the property graph model can be retrieved by using the MATCH syntax to identify nodes with the corresponding properties in both the nodes and relationships.

For example, the query in Figure 16. Finds all Person nodes labelled as Actor connected through ACTED_IN relationships to Movie nodes, which are in turn connected through DIRECTED relationships to other Person nodes (Figure 16). The RETURN clause then displays these matched nodes and relationships. By using this method, data retrieval is dynamic and highly flexible by specifying the patterns of nodes and relationships to efficiently traverse the graph and access interconnected data.

## Access Control

In the context of access control, Neo4j has built-in native authentication and authorization (Table 2). It contains features that stores user and role information in the system database for authentication purposes and manages authorization using Role-Based Access Control (RBAC) (https://neo4j.com/docs/operations-manual/current/authentication-authorization/). The RBAC method assigns privileges to roles then assigned to users, limiting their actions on the database. This can be done in Neo4j by using Cypher to modify the access rights to users. Neo4j also contains built-in roles and privileges that can be modified to suit the user's needs.

## Data Integrity

Despite being a type of NoSQL database that utilizes a graph model, Neo4j can ensure data integrity by being fully ACID compliant (https://neo4j.com/docs/operations-manual/current/database-internals/). This is achieved via write-ahead transaction log to ensure durability, where it keeps track of all write operations to ensure data consistency and enable recovery (https://neo4j.com/docs/operations-manual/current/database-internals/transaction-logs/). The transactions performed in Neo4j are atomic, where any operations will be rolled back if any part fails resulting in no changes to the databases to maintain consistency. Neo4j only allows default read-committed isolation level where data can only be read when it is committed by other transactions in a complete state. It also provides optional write locks to provide isolation by preventing other transactions from affecting data that the user is working with. Lastly, durability is ensured with built-in deadlock detection and robust transaction management, making committed transactions permanently stored and recoverable (https://neo4j.com/docs/operations-manual/current/database-internals/transaction-logs/).

**Table 1:** **Components in graph model (https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/).**

| Component | Functionality |
|---|---|
| Node | Represent entities in a domain and can have zero or more labels to define their type. |
| Label | Classify nodes by defining their type. |
| Relationship | Describe connections between nodes, have a direction, and must have a type to classify the connection. |

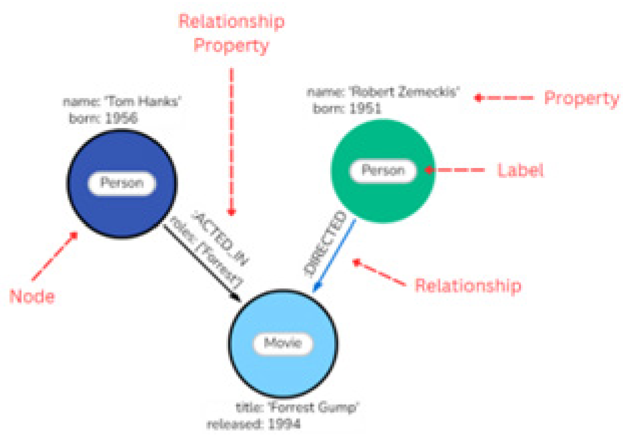


**Figure 11:** Example of Property Graph Database Model in Neo4j (https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/).

## Cassandra (Wide Column Model)

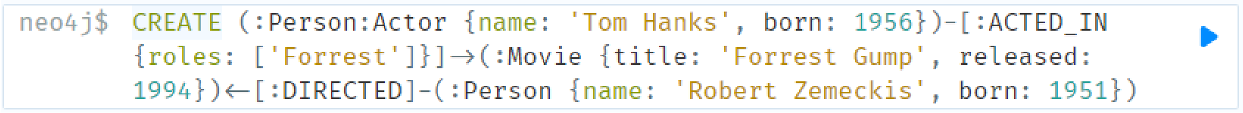
The wide column model organizes data into columns rather than rows. This data model is particularly well-suited for handling large volumes of structured data and offers flexibility in how data can be stored and accessed. Unlike traditional relational models, where each row has a fixed set of columns, wide column databases allow each row to contain a different number of columns like in Figure 17. This feature provides the ability to store sparse data efficiently. Hence, the model excels in scenarios that require high write throughput and the ability to perform complex queries on large datasets (https://www.scylladb.com/glossary/wide-column-database/).

Cassandra is a distributed database system designed to manage large volumes of data for low-cost servers. It is the top choice for wide column database because it ensures high availability without a single point of failure (https://www.datastax.com/guides/what-is-cassandra). Cassandra architecture runs on masterless and peer-to-peer model where any node can handle any request. This allows the system to scale horizontally while maintaining consistent performance under a heavy load. Cassandra is well-suited for applications that require high reliability and robustness because of the data replication and fault tolerance (https://cassandra.apache.org/_/index.html).

Cassandra Query Language (CQL) is the main tool to interact with Cassandra. It is easier to use for those familiar with relational databases as it has similar syntax to SQL. CQL has structured language to define keyspaces, column families and the data types to accommodate the unique aspects and constraints (https://docs.datastax.com/en/cql/hcd-1.0/overview/cql-about.html).

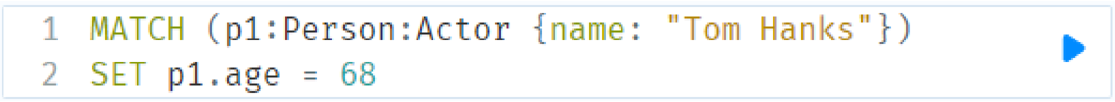
## Data Creation

In Cassandra, data is stored in structures known as keyspaces, similar to databases in relational models. Each keyspace contains



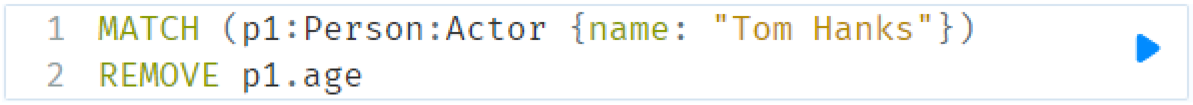
```
neo4j$ CREATE (:Person:Actor {name: 'Tom Hanks', born: 1956})-[:ACTED_IN
       {roles: ['Forrest']}]→(:Movie {title: 'Forrest Gump', released:
       1994})←[:DIRECTED]-(:Person {name: 'Robert Zemeckis', born: 1951})
```

**Figure 12:** Data Creation in Neo4j.



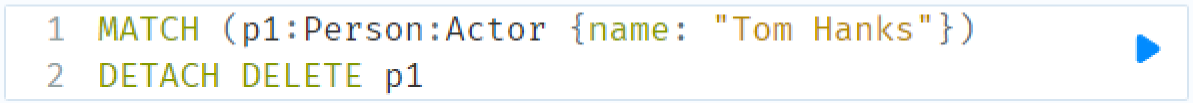
```
MATCH (p1:Person:Actor {name: "Tom Hanks"})
SET p1.age = 68
```

**Figure 13:** Update Property in Neo4j.



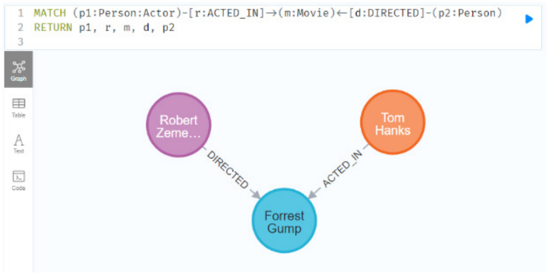
```
MATCH (p1:Person:Actor {name: "Tom Hanks"})
REMOVE p1.age
```

**Figure 14:** Remove Property in Neo4j.



```
MATCH (p1:Person:Actor {name: "Tom Hanks"})
DETACH DELETE p1
```

**Figure 15:** Delete Node and Relationship in Neo4j.



```
MATCH (p1:Person:Actor)-[r:ACTED_IN]→(m:Movie)←[d:DIRECTED]-(p2:Person)
RETURN p1, r, m, d, p2
```

**Figure 16:** Data Retrieval in Neo4j.

**Table 2: Built-in Roles in Neo4J (https://neo4j.com/docs/cypher-manual/current/introduction/cypher-overview/).**

| Role | Privileges |
|---|---|
| PUBLIC | Access to the home database for executing procedures, user-defined functions, and loading data. |
| reader | Access to all databases for reading data and viewing schema constructs. |
| editor | Access to all databases for reading and writing data, with limited schema modification permissions. |
| publisher | Access to all databases for reading and writing data and viewing schema constructs. |
| architect | Access to all databases for reading, writing, and managing schema constructs. |
| admin | Full access to all databases for managing data, schema, and privileges, and executing administrative procedures. |

multiple column families, which are analogous to tables (https://docs.datastax.com/en/cql/hcd-1.0/overview/cassandra-structure.html). However, unlike traditional tables, each row in a column family can have a different set of columns. Data creation involves defining a keyspace and its associated column families, specifying the primary key and clustering columns that determine data retrieval order as shown in Figures 18 and 19.

### Data Manipulation

Cassandra supports various data manipulation operations, including inserting, updating, and deleting data. The INSERT statement is used to add new rows, the UPDATE statement modifies existing rows and the DELETE statement removes rows or specific columns within a row (Figures 20-22). Data manipulation is schema-less, allowing each row to have unique columns.

### Data Retrieval

Data retrieval in Cassandra is highly efficient due to its partitioning and clustering keys. For example, the query in Figure 23 allows users to query data based on the primary key using the SELECT statement. Complex queries are supported, including range queries and filtering using secondary indexes.

### Access Control

Cassandra uses RBAC to manage permissions. It allows administrators to define roles with specific permissions and assign them to users. This ensures secure data access and operation controls within the database environment (https://www.datastax.com/blog/role-based-access-control-cassandra).

For example, the query in Figure 24 creates a role 'ecommerce_user' and the query in Figure 25 grants access to select, insert and update on the customers table to it.

### Data Integrity

Cassandra ensures data integrity through several mechanisms, including consistency levels, write-ahead logging, and tunable consistency. Users can configure consistency levels for reads and writes, balancing between consistency and availability. The database supports eventual consistency, ensuring that all replicas converge to the same state over time (https://docs.datastax.com/en/cassandra-oss/3.x/cassandra/dml/dmlAboutDataConsistency.html).

For example, the query in Figure 26 sets the consistency level to QUORUM, meaning that a majority of replicas must agree on the value for it to be returned or written.

## Redis (Key-value Model)

A key-value database stores each data in a simple pair of key and value. The key is a simple string that is unique, followed by the value which could be an arbitrary large data field of any scalar data types (integers, strings, JSON, BLOB etc.,). A hash table keeps track of the unique keys and links it to the corresponding data value with pointers.

To search for a record, you can only query and search the key, not the value. Referring to Figure 27 as example, when you query "Paul" (key), you get the phone number (value) associated with "Paul", which is "(091) 9786453778". There is also no query language in key-value databases. In contrast to relational databases that uses SQL (Structured Query Language) to perform complex operations like joining multiple tables, key-pair database does not support querying languages (https://redis.io/nosql/key-value-databases/).

Redis, which stands for Remote Dictionary Server, is a key-value store that excels in speed and efficiency. This is because it operates in memory instead of on a disk or Solid-State Drive (SSD), making it exceptionally fast for read and write operations (https://www.ibm.com/topics/redis). Redis also supports various data structures, including strings, hashes, lists, sets, and sorted sets, which adds flexibility in how data can be stored and manipulated (https://redis.io/docs/latest/develop/data-types/).

## Data Creation

Data creation in Redis involves adding key-value pairs to the database. This can be done using the SET command in Redis as shown in Figure 28 and Table 3.

In Figure 29, SET mykey "Hello" means to create a key named "mykey" and associate it with the value of "Hello". If the key previously has a value, using SET on the key will overwrite the previous record (https://redis.io/docs/latest/commands/set/).

In Figure 30, the MSET command allows user to set multiple key-value pairs in one command. The MSET will overwrite the previous value of the key, similar to SET.

## Data Manipulation

To **update** the value of a key, use SET and it will overwrite the previous value stored. To **delete** a key in Redis, the DEL command is used. If the key does not exist, the command will be ignored. The DEL command can also delete multiple keys in one command (https://redis.io/docs/latest/commands/del/).

In Figure 31, "key1" and "key2" were deleted. "key3" does not exist; hence, it was ignored and not deleted. The returned integer indicates the number of keys that were removed from the database.

## Data Retrieval

The GET command is used to fetch the value stored at a key as shown in Figure 32. If the key does not exist, nil will be returned. However, GET command can only return the value if it is a string (https://redis.io/docs/latest/commands/get/).

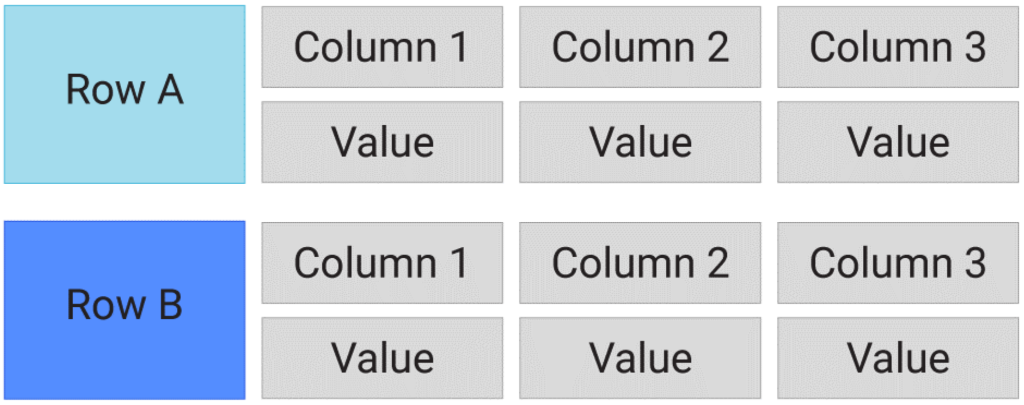
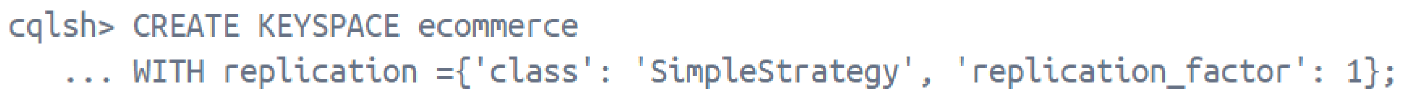
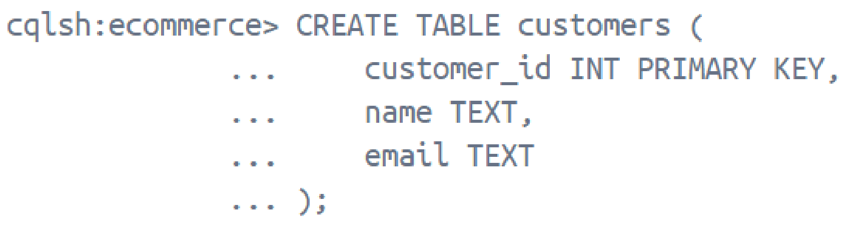


**Figure 17:** Structure of Wide Column Database (https://www.scylladb.com/glossary/wide-column-database/).



```
cqlsh> CREATE KEYSPACE ecommerce
   ... WITH replication ={'class': 'SimpleStrategy', 'replication_factor': 1};
```

**Figure 18:** Create Keyspace in Cassandra.



```
cqlsh:ecommerce> CREATE TABLE customers (
             ...     customer_id INT PRIMARY KEY,
             ...     name TEXT,
             ...     email TEXT
             ... );
```

**Figure 19:** Create Table in Cassandra.

### Access Control

When a database is created, users have full access to the database as *unauthenticated access* is selected by default. There are three access methods: unauthenticated access, password-only authentication, and using ACL only (https://redis.io/docs/latest/commands/get/).

In Figure 33, unauthenticated access would mean anyone could access the database without any credentials; password-only authentication would require users to authenticate with their configured password using the AUTH command; using ACL only would require users to set up RBAC with Access Control Lists (ACLs).

### Role-Based Access Control (RBAC)

Redis has RBAC to manage users' access privileges. The RBAC allows creation of roles and definition of each role's access privileges. It also allows creation of users then assigning roles to them.

### Cluster Access vs Database Access

In Redis there are two types of access:

Cluster access: allows performing management-related actions (create database, view statistics).

Database access: allows performing data-related actions (read and write data).

Each role can be granted either or both accesses. This is useful to manage and control who can access the databases and who can access cluster management for security purposes.

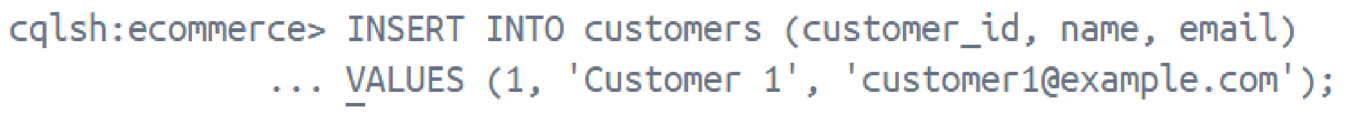
### Data Integrity

Data integrity in Redis is maintained through its atomic operations. Commands like SET and DEL are atomic, where they either complete fully or not at all. Additionally, Redis supports transactions through the use of the MULTI and EXEC commands, which allow multiple commands to be executed as a single atomic operation. Using WATCH could also provide a Check-and-Set (CAS) behaviour. Keys that have the WATCH command will be monitored to check for any changes. If any WATCH-ed keys are modified before the EXEC command, the whole transaction will be aborted to prevent race condition (https://redis.io/docs/latest/develop/interact/transactions/).

If the code in Figure 34 is run by multiple clients at the same time, the mykey value would be incremented at the same time and affect data integrity. By WATCH-ing mykey, if another client tries to modify the result of val in time between the call to WATCH and EXEC, the transaction will fail.

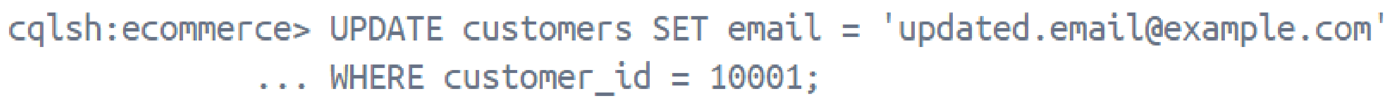
### MongoDB (Document Model)

In a document model, data is stored in documents in various formats such as JSON, BSON and XML. These documents can



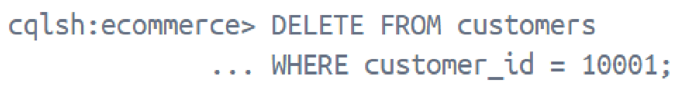
```
cqlsh:ecommerce> INSERT INTO customers (customer_id, name, email)
             ... VALUES (1, 'Customer 1', 'customer1@example.com');
```

**Figure 20:** Insert Data in Cassandra.



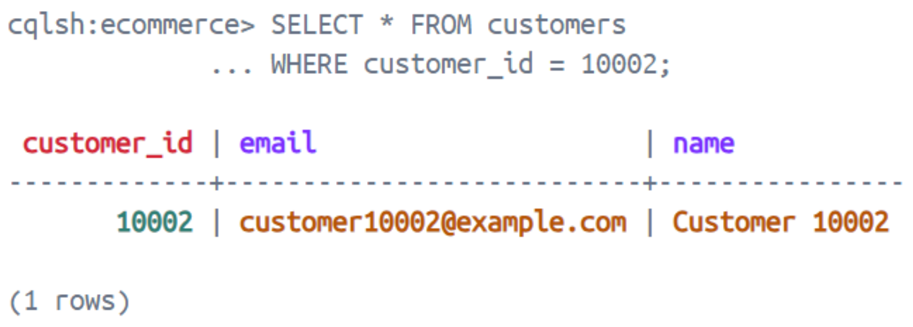
```
cqlsh:ecommerce> UPDATE customers SET email = 'updated.email@example.com'
             ... WHERE customer_id = 10001;
```

**Figure 21:** Update Data in Cassandra.



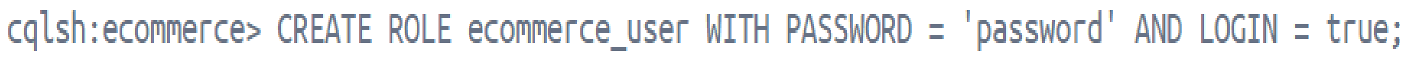
```
cqlsh:ecommerce> DELETE FROM customers
             ... WHERE customer_id = 10001;
```

**Figure 22:** Delete Data in Cassandra.



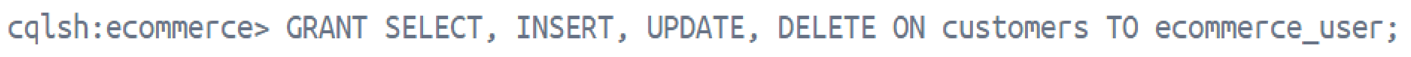
```
cqlsh:ecommerce> SELECT * FROM customers
             ... WHERE customer_id = 10002;

 customer_id | email                     | name
-------------+---------------------------+----------------
       10002 | customer10002@example.com | Customer 10002

(1 rows)
```

**Figure 23:** Data Retrieval in Cassandra.



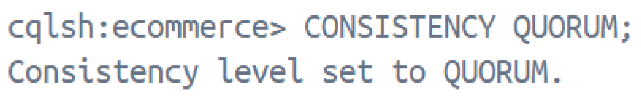
```
cqlsh:ecommerce> CREATE ROLE ecommerce_user WITH PASSWORD = 'password' AND LOGIN = true;
```

**Figure 24:** Create Role in Cassandra.



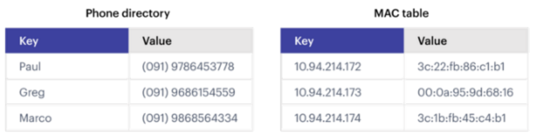
```
cqlsh:ecommerce> GRANT SELECT, INSERT, UPDATE, DELETE ON customers TO ecommerce_user;
```

**Figure 25:** Grant Access to Role in Cassandra.



```
cqlsh:ecommerce> CONSISTENCY QUORUM;
Consistency level set to QUORUM.
```

**Figure 26:** Set Consistency Level in Cassandra.



Phone directory

| Key | Value |
|---|---|
| Paul | (091) 9786453778 |
| Greg | (091) 9686154559 |
| Marco | (091) 9868564334 |

MAC table

| Key | Value |
|---|---|
| 10.94.214.172 | 3c:22:fb:86:c1:b1 |
| 10.94.214.173 | 00:0a:95:9d:68:16 |
| 10.94.214.174 | 3c:1b:fb:45:c4:b1 |

**Figure 27:** Structure of Key-value Model.

have varying structures and flexible schemas and are able to contain

To showcase how data is created and manipulated in a document model, we will utilize MongoDB as an example with the queries being executed through MongoDB Shell or mongosh. Mongosh is a JavaScript and Node.js environment for interacting with MongoDB databases and can be used to test queries or interact with the database (https://www.mongodb.com/docs/mongodb-shell/). However, not all document databases may use the same syntax or queries in their creation or manipulation of data.

## Data Creation

In the case of MongoDB, in order to store data, we first have to create a database and a collection. Documents which store the data are gathered together in a collection while a database stores one or more collections or documents. In the context of a relational database, a collection can be compared to a table. In order to create a database, we can utilize the insertOne() method as shown as Figure 35. If the target database and collection does not exist, it will automatically create a new database and collection. The code below demonstrates how to create a database with the name "mynewDB" and a collection with the name "myNewCollection1" (https://www.mongodb.com/docs/manual/core/databases-and-collections/). Alternatively, the user can utilize the createCollection() method to create a collection with various options.

In order to insert data into the collection or insert a document, we can use the insertOne() and insertMany() methods. The insertOne() inserts a single document while insertMany() inserts multiple documents into the collection. It is also important to note that if the user does not specify the _id that acts as a primary key, MongoDB will automatically generate an object for the _id field. Figures 36 and 37 are examples on how to utilize the insertOne() and insertMany() method (https://www.mongodb.com/docs/manual/tutorial/insert-documents/).

## Data Manipulation

To update the document, there are 3 methods available to the user. They can utilize the updateOne(), updateMany() and replaceOne() methods. The updateOne() method is used to update the first document where the condition is met whereas updateMany() updates all the documents. The replaceOne() method is used to replace the entire content of a document except for the _id primary key. Figures 38-40 demonstrate how to utilize all 3 methods.

For delete operations, you can utilize the deleteOne() and deleteMany() methods (Figures 41 and 42). The deleteOne() and deleteMany() methods work similarly to the updateOne and updateMany() methods.

## Data Retrieval

Moving on to read operations, we can use the find() method. The find() method also supports the typical 'or' and 'and' conditions that we normally use in a relational database as shown in Figures 43 and 44. The following is an example of how to retrieve data utilizing both the 'or' and 'and' operation and its equivalent in a SQL statement.

## Access Control

In the context of security and access control, MongoDB supports RBAC; however other databases may have different levels of security and access control. MongoDB itself comes with several default roles to simplify access control. The administrator of the database can also create users and assign custom made roles to them (https://www.mongodb.com/docs/manual/core/authorization/). In order to create a user, the createUser() method in Figure 45 is used while the createRole() method is used to create a role. The administrator can assign a role upon user creation or use the grantRolesToUser() method after creating a user. The following shows an example on how to use the createUser() method to create a user and assign a role to them.

## Data Integrity

As for data integrity, it can vary between the different document model databases however it generally prioritizes availability over immediate consistency. Meanwhile the flexibility of the schemas in document models leads to data redundancy as duplicate data might be present across different documents. However, this can lead to issues in maintaining data consistency as every instance of the data needs to be updated (https://atlan.com/relational-vs-document-database/).

## Research Methodology

This study employs a mix of quantitative and qualitative approach to investigate the differences between SQL (relational) and NoSQL (graph, wide column, key-value and document) databases and identify the factors that drive the preference for each database models in distinct scenarios. The research focuses on theoretical evaluation and practical insights to provide a comprehensive understanding of database performance and application.

**Table 3:** **Keywords of Data Creation in Redis.**

| Keywords | Functionality |
|---|---|
| EX *seconds* | Set specified expire time in seconds. |
| PX *milliseconds* | Set specified expire time in milliseconds. |
| EXAT *timestamp-seconds* | Set the specified Unix time at which the key will expire in seconds. |
| PXATtimestamp-milliseconds | Set the specified Unix time at which the key will expire in milliseconds. |
| NX | Only set key if it does not already exist. |
| XX | Only set the key if it already exists. |
| KEEPTTL | Retain the time to live associated with the key. |
| GET | Return the old string stored at key, or nil if key did not exist. An error is returned and SET aborted if the value stored at key is not a string. |

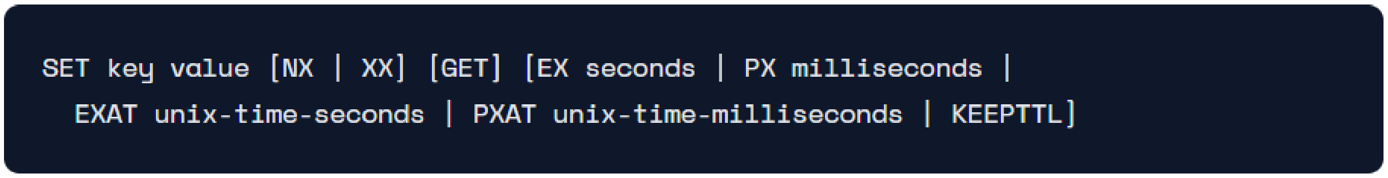
## Data Collection Methods

The study began with collecting data and information on SQL and NoSQL databases from various sources to address our objectives. Initially, a literature review was conducted to study the characteristics of each database model implemented in their respective DBMS (Oracle, Neo4j, Cassandra, Redis and MongoDB).

The data is collected by using academic databases such as Google Scholar, Science Direct, and Sunway Library to ensure the accuracy and validity of information. Mediums such as article, journal publication, and thesis papers are primarily used as our source of information. This research also refers to official DBMS documentations and online blogs for additional insights on the context of each database model in different use cases.

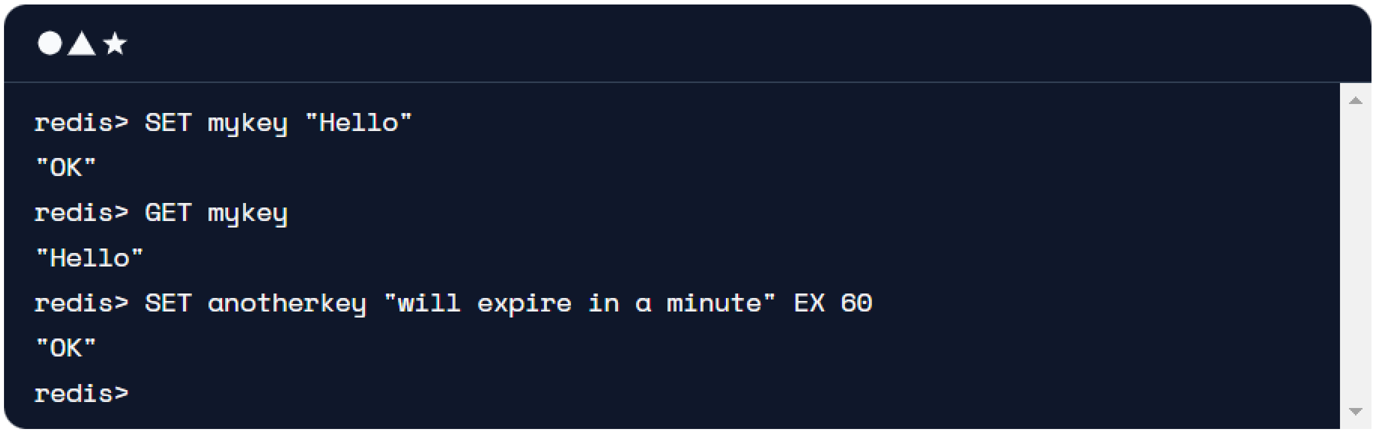
## Data Analysis Procedures

The research will be conducted in two different parts, which is performance evaluation and scenario analysis. The performance evaluation will analyse the runtime performance and traits of each database model implemented in their respective DBMS by using quantitative results from reliable academic sources. This process will analyse and evaluate each database model in five different contexts: Data Creation, Data Manipulation, Data Retrieval, Access Control, and Data Integrity, where the results are justified



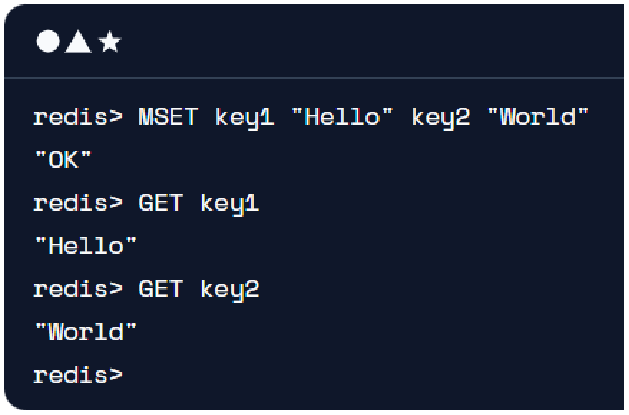
```
SET key value [NX | XX] [GET] [EX seconds | PX milliseconds |
  EXAT unix-time-seconds | PXAT unix-time-milliseconds | KEEPTTL]
```

**Figure 28:** Syntax of Data Creation in Redis (https://redis.io/docs/latest/commands/set/).



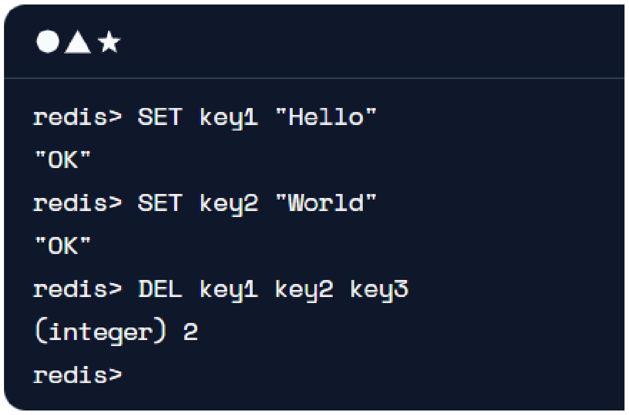
```
redis> SET mykey "Hello"
"OK"
redis> GET mykey
"Hello"
redis> SET anotherkey "will expire in a minute" EX 60
"OK"
redis>
```

**Figure 29:** Example 1 of Data Creation in Redis.



```
redis> MSET key1 "Hello" key2 "World"
"OK"
redis> GET key1
"Hello"
redis> GET key2
"World"
redis>
```

**Figure 30:** Example 2 of Data Creation in Redis.



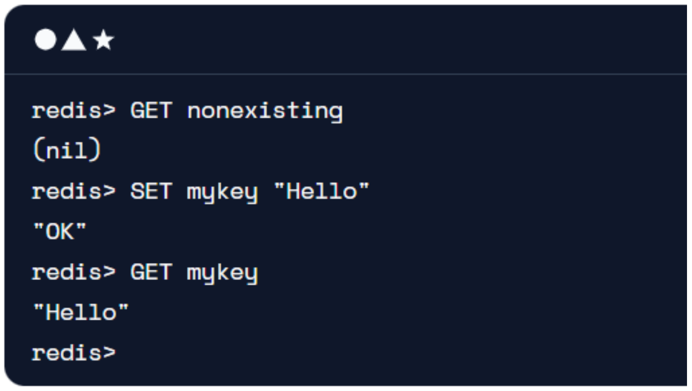
```
redis> SET key1 "Hello"
"OK"
redis> SET key2 "World"
"OK"
redis> DEL key1 key2 key3
(integer) 2
redis>
```

**Figure 31:** Example of Deleting Key in Redis.

from our findings on the study.The second part would be the scenario analysis where a theoretical evaluation of each database model will be conducted in two distinct scenarios to perform a qualitative review of their performance. This method utilizes theory-based evaluation approach which evaluates each database model in a conceptual context to highlight their comparative differences and identify the preference of each database model in a particular scenario. The two selected scenarios for theoretical evaluation will be distinct from each other to simulate their strength and weaknesses when implemented in a specific scenario with different requirements.

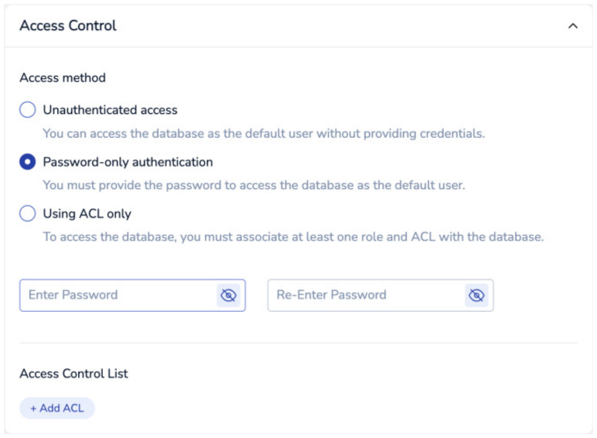
## Justification of Methodological Choices

The team decided to use a mix of quantitative and qualitative approach for this study to understand the characteristics and capabilities of each database models in two different levels of context. The qualitative analysis is used to study the performance of each database model in terms of run-time performance and traits without having much external variable affecting the judgement of our findings. This is purely to gain insight into the performance of each database model implemented in their respective DBMS and study the factors that contribute to their performance based on their characteristics. The insights gained



```
redis> GET nonexisting
(nil)
redis> SET mykey "Hello"
"OK"
redis> GET mykey
"Hello"
redis>
```

**Figure 32:** Data Retrieval in Redis.



Access Control

Access method

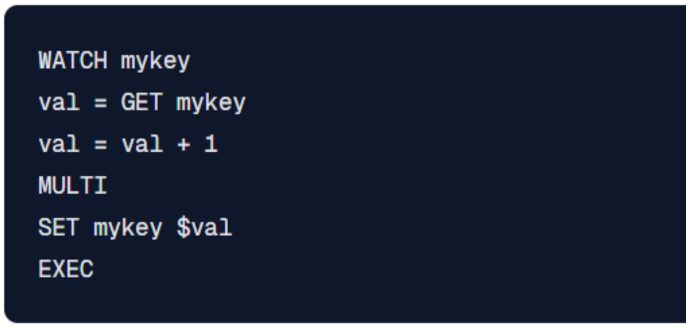
Unauthenticated access
You can access the database as the default user without providing credentials.

[✓ Password-only authentication]
You must provide the password to access the database as the default user.

Using ACL only
To access the database, you must associate at least one role and ACL with the database.

Access Control List

**Figure 33:** Access Control in Redis.



```
WATCH mykey
val = GET mykey
val = val + 1
MULTI
SET mykey $val
EXEC
```

**Figure 34:** Data Integrity in Redis.

are then used as a foundation of reasoning for the upcoming qualitative evaluation with more factors to be considered.

The qualitative analysis is then used to study and simulate how each database model will perform under distinct scenarios on a conceptual level. This method is used to evaluate their performance when implemented in an actual ecosystem or environment which differs from just testing their run time performance in a controlled environment as there are many other factors that will affect their effectiveness. Thus, a mix of quantitative and qualitative is method is employed to study the overall performance of each database models in a more comprehensive and complete manner for accuracy and relevancy of result.

## RESULTS

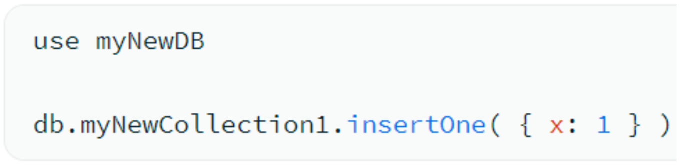
This section presents the performance of each data model implemented in their respective DBMS and scenario analysis to highlight the characteristics of SQL and NoSQL databases.

### Performance Evaluation

The results below are referenced from Comparison of query performance in relational a non-relational databases by Roman Čerešňák and Michal Kvet, (Čerešňák and Kvet, 2019). which showcases the query performance of each data model in their respective DBMS. The performance testing is conducted in a controlled environment with sample records of 10k and 100k respectively. This result is used as a benchmark for evaluation of each data model in their respective DBMS to ensure accuracy and relevancy in our findings.

### Data Creation

Oracle and Cassandra require table creation before inserting data while MongoDB, Redis, and Neo4j do not require a table to store data due to their nature of NoSQL database. Instead, they use other forms to store data that are often schema-less, except Cassandra which uses keyspaces and column families, a structure similar to the relational model that involves table creation.Figure 46 shown that the SQL database (Oracle) demonstrated a longer execution time compared to the remaining NoSQL databases (MongoDB, Redis, Neo4j, Cassandra). Although Oracle is capable of handling larger amount of volume with only small increment in execution time, its performance is still found lacking compared to other NoSQL databases. The remaining NoSQL databases demonstrated similar performance where no major deviation in execution time when handling larger volume of data, highlighting their scalability capabilities.



```
use myNewDB

db.myNewCollection1.insertOne( { x: 1 } )
```

**Figure 35:** Create Database in MongoDB.

### Data Manipulation

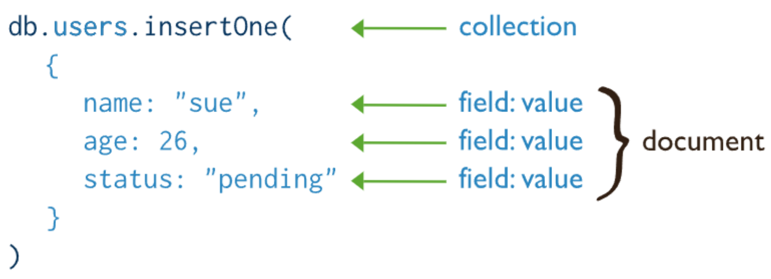
Figures 47 and 48 showcase the query performance of data manipulation for each DBMS in the context of updating data and data deletion. In the context of data manipulation, the performance of SQL database is found lacking compared other NoSQL databases, especially in handling large volume of data deletion for Oracle. This is due to SQL databases being ACID compliant which reduces performance when handling larger amounts of data or performance requests where data is required to be processed in a strict order. NoSQL databases on the other hand can process records at any time which reduces wait time and improves performance due to being BASE compliant (https://aws.amazon.com/compare/the-difference-between-acid-and-base-database/). However, MongoDB and Neo4j is fully ACID compliant despite being a type of NoSQL database without sacrificing their runtime performance.

### Data Retrieval

Figure 49 shown that SQL database demonstrated a decrease in performance when retrieving larger volume of data while the performance of NoSQL databases is similar with only slight deviations. This is due to the nature of SQL databases which relies on strict, predefined schema which affects how data is retrieved due to data being stored throughout multiple tables in the schema. Unlike SQL databases, NoSQL databases are often deemed schemaless where it does not involve JOIN operations, allowing them to run queries faster even when handling large volume of data.

### Access Control

Table 4 summarises the access control mechanisms implemented in each DBMS. Both SQL and NoSQL databases supports RBAC implementation although the extent and granularity can vary significantly. In SQL databases like Oracle offers cell-level granularity, which allows for very find-grained access control, ensuring that permissions can be precisely managed down to individual cells within tables. NoSQL databases implement RBAC with varying levels of granularity. Neo4J operates at graph-level, managing permissions for entire graphs and sub-graphs. Cassandra uses column-family-level, controlling access to set of columns. Redis uses cluster-level, managing permissions across



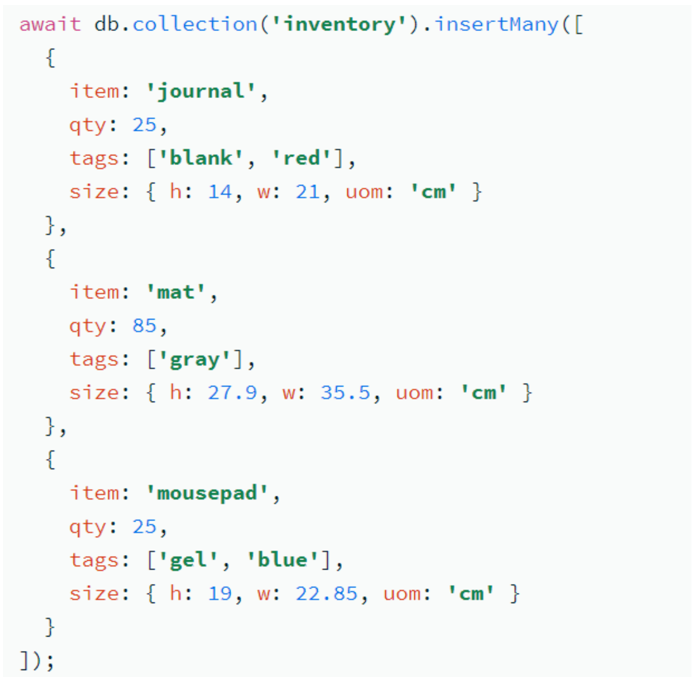
```
db.users.insertOne(
   {
      name: "sue",
      age: 26,
      status: "pending"
   }
)
```

**Figure 36:** Example of Inserting Data in MongoDB (https://www.mongodb.com/docs/manual/crud/).

entire clusters. MongoDB uses collection-level, applying access control to entire collections of documents.

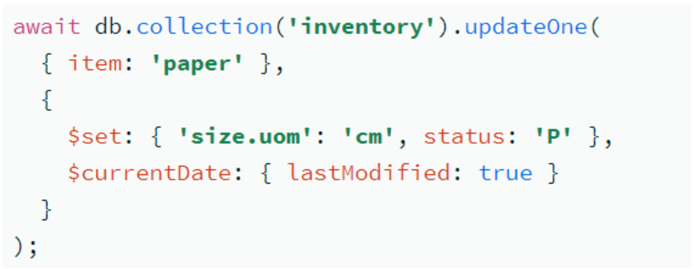
Table 5 also shows that the Oracle and MongoDB consist of extensive built-in roles including administrative and user permissions. It helps to streamline the management of security without the need for creating extra custom roles. The Neo4J has basics built-in roles. Cassandra has limited built-in roles, so it needs additional effort to create custom roles for other access. Redis include built-in roles with administrative functions. All databases support creating custom roles except Redis.

Out of these databases, Oracle which is a SQL database is the easiest to implement because of its comprehensive roles and strong management tools. Neo4J and Cassandra is moderate to implement but it require additional effort to create extra custom roles. Redis is the hardest to implement as it only have limited administrative roles and does not support creating custom roles. Finally, MongoDB has a moderate difficulty of implementation with the extensive built-in roles even though configuring RBAC can be complicated due to its schema-less design.



```
await db.collection('inventory').insertMany([
  {
    item: 'journal',
    qty: 25,
    tags: ['blank', 'red'],
    size: { h: 14, w: 21, uom: 'cm' }
  },
  {
    item: 'mat',
    qty: 85,
    tags: ['gray'],
    size: { h: 27.9, w: 35.5, uom: 'cm' }
  },
  {
    item: 'mousepad',
    qty: 25,
    tags: ['gel', 'blue'],
    size: { h: 19, w: 22.85, uom: 'cm' }
  }
]);
```

**Figure 37:** Example of Inserting Multiple Data in MongoDB (https://www.mongodb.com/docs/manual/tutorial/insert-documents/).



```
await db.collection('inventory').updateOne(
  { item: 'paper' },
  {
    $set: { 'size.uom': 'cm', status: 'P' },
    $currentDate: { lastModified: true }
  }
);
```

**Figure 38:** Example of Updating One Data in MongoDB (https://www.mongodb.com/docs/manual/tutorial/update-documents/).

## Data Integrity

Table  shows whether each database is BASE or ACID compliant, along with the implementation difficulty.

Oracle supports easy implementation of various constraints like primary keys, foreign keys, unique constraints and check constraints in order to enforce data integrity. Triggers can also be used enforce more complex business rules, data constraints and relationships. Meanwhile, MongoDB utilizes JSON schema validation to enforce data integrity and embeds related data in a single document to ensure consistency, however, it does not enforce these constraints as strictly as relational databases. Neo4j supports unique constraints on node labels and relationships as well as write-ahead transaction logs to enforce data integrity. These constraints are implemented through Cypher queries but they are more limited in types compared to relational databases. Cassandra supports a primary key but lack other constraints such as foreign keys and unique constraints. Instead, it relies on application logic to enforce data integrity. And finally, Redis supports atomic operations but does not enforce any schema constraints and uses application logic similar to Cassandra. This makes enforcing data integrity with Redis and Cassandra difficult compared to other databases.Scenario AnalysisE-Commerce Platform

This scenario features an e-commerce platform such as Shopee and Lazada which handles large volume of transaction requests that needs to be processed concurrently. This is usually achieved by implementing Online Transactional Processing (OLTP) database which excels in real-time execution of large numbers of transactions performed by many users (https://www.ibm.com/topics/oltp). The e-commerce platform also requires a structured database model that is capable of storing user and product data from the sellers, ensuring data are interconnected and consistent when a transaction occurs.

Most of the OLTP database exists are relational databases which can accommodate large volume of concurrent users and frequent query processing, which is crucial in the e-commerce platform. The nature of this scenario requires data to be constantly updated in fast response times due to the frequent transactions performed by many users. Thus, the ideal database model should support rapid query processing and ensure that multiple users have access to same data all the time without compromising data integrity. For future proof, the proposed database model should also be scalable due to the dynamic evolving nature of e-commerce platforms that needs to accommodate increasing numbers of users.

## Relational Model (Oracle)

### *Strengths*

Efficient Query Performance-Data stored are indexed in the table which is used for querying, searching, and retrieval to perform transactions efficiently. (https://www.ibm.com/topics/oltp).

ACID Compliant-Transactions carried out are fully ACID compliant which ensures data integrity and consistency throughout the schema despite how data is modified frequently due to the referential constraints.

Complex Query Processing-Supports complex SQL operations which provides flexibility in accessing or processing data from multiple entities such as retrieving product data and user details for a transaction.

Robust RBAC Mechanism-Consists of robust built-in RBAC and data encryption feature that is easy to implement to ensure privacy of user data and prevention of unauthorized access.

### Weaknesses

Rigid Schema-Implementation of a rigid schema makes relational model inflexible in accommodating new types of data or structural changes in the e-commerce platform.

Scalability Issue-The rigid schema leads to scalability issues where it is less efficient when handling larger volume of data due to the constraint and checks of the schema which affects performance in the platform.

Costly-Requires high overhead to modify the rigid structure of the schema just to scale according to business needs in this highly dynamic e-commerce environment as it involves major restructuring and development to ensure ACID compliant.

## Graph Model (Neo4J)

### *Strengths*

Complex Relationship Management-Excel in representing and querying complex relationships between entities such as, customers, products, and orders for rapid processing by interconnecting the data when a user perform a transaction.

Real-Time Analytics Capabilities-Support real-time analytics and insights based on user behaviour due to complex relationship management which stores the dynamic behaviour of users in the e-commerce platform

Scalability-Highly scalable to accommodate more data due to the schema optional nature of graph model which allows the e-commerce platform to accommodate more dynamic data of users and transactional records.

### Weaknesses

Scalability with High Write Throughput-Struggles with high write throughput in an OLTP context, especially when compared to relational databases which is optimized for e-commerce operations.

Inadequate Query Performance-Graph model may excel in query processing, but it is found lacking when comparing with relational databases in performing highly transactional operations such as updating order status and inventory management.

Complexity in Maintaining Graph Structure-Frequent transactions performed by multiple users concurrently will constantly modify the data and affect the overall graph structure, leading to performance bottleneck.

## Wide Column Model (Cassandra)

### *Strengths*

Horizontal Scalability and High Availability-Supports large volumes of data and handles high transaction rates, ensuring consistent performance during peak times like sales or promotions (https://databasetown.com/wide-column-database-use-cases/).

Seamless Scaling Without Downtime-Scale out seamlessly without downtime, maintaining consistent performance and high availability, which is critical for maintaining customer trust and prevent revenue loss (https://databasetown.com/wide-column-database-use-cases/).

High Write Throughput-Efficiently handles frequent updates such as inventory changes, user activity logs, and order processing, essential for real-time data processing (https://databasetown.com/wide-column-database-use-cases/).

Flexible Schema Model-Easy to add new fields without disrupting existing data, good at adapting to changing business needs, such as introducing new product attributes or supporting diverse data formats (https://databasetown.com/wide-column-database-use-cases/).

Fault-Tolerant-Ensures data replication across multiple nodes to eliminate single points of failure and provide high resilience against outages (https://databasetown.com/wide-column-database-use-cases/).

### Weaknesses

Eventual Consistency-May not provide the immediate consistency required in e-commerce platform such as financial transactions or inventory management (https://databasetown.com/wide-column-database-use-cases/).

Complex Data Modelling-Requires good understanding of the CQL to design an efficient schema. Can cause data redundancy and increased storage requirements because of the lacks support for joins (https://databasetown.com/wide-column-database-use-cases/).

Limited Support for ACID Transactions-Not ideal for applications that require strong transactional consistency across multiple rows or tables (https://databasetown.com/wide-column-database-use-cases/).

Limited Query Flexibility-Lacks support for complex querying such as joins or ad hoc queries which can be restrictive for e-commerce platform that requires robust data analysis and reporting capabilities (https://databasetown.com/wide-column-database-use-cases/).

Increased Latency with Large Datasets-May experience increased latency and potentially causing performance bottlenecks as the data set grows. This is a concern for e-commerce platforms that deal with extensive product catalogues and high user interaction volumes (https://databasetown.com/wide-column-database-use-cases/).

## Key-Value Model (Redis)

### *Strengths*

Fast Data Handling-In-memory storage allows for fast data handling with minimal latency to support the processing of large volume of data in real-time such as processing orders and managing products (https://redis.io/nosql/key-value-databases/).Flexible Schema-Supports various types of entities which lets the platform perform changes such as adding new product information or details without much effort (https://redis.io/nosql/key-value-databases/).

Efficient Caching-Caching capability reduces memory usage by storing frequently accessed data such as information of popular products and payment details of user (https://redis.io/nosql/key-value-databases/).

Partially ACID-Compliant-Supports atomic operations which help ensure consistency and improve accuracy in processes such as order processing and inventory management (https://redis.io/nosql/key-value-databases/).

Scalability-Can be scaled horizontally to handle increased traffic during peak periods such as sales and holidays, which increases overall performance (https://redis.io/nosql/key-value-databases/).

## Weaknesses

High Cost-In-memory storage utilizes RAM instead of traditional hard drives which can be costly to prepare the necessary hardware and infrastructure to support a large data volume present in an e-commerce platform (https://alronz.github.io/Factors-Influencing-NoSQL-Adoption/site/Redis/Results/Strengths%20and%20Weaknesses/).

Slow Backups-Regular backups performed by the database can slow the system down as memory dumps are used to create snapshots which may cause reoccurring periods of lag while performing backups (https://alronz.github.io/Factors-Influencing-NoSQL-Adoption/site/Redis/Results/Strengths%20and%20Weaknesses/).

Lack of Complex Querying Capabilities-Data can only be accessed via key and there are no relationships in a key-value pair model which can complicate development and features such as product filtering (https://redis.io/nosql/key-value-databases/).

## Document Model (MongoDB)

### *Strengths*

Flexible Schema-Has a schema that is easily adaptable to changes. Can be used to store a wide range of information such as product details and customer information.

Complex Data Structures-Supports nested arrays and embedded documents which can help further embed information reducing the need for multiple collections and joins such as embedding the payment details in the user collection.

Real-Time Analytics-Provides real-time data analytics allowing for analysis of data such as customer behaviour, inventory levels, and sales data.

Simplified Data Handling-Related data can be stored in a single document which reduces the need for joining. This makes retrieval and manipulation of data easier such as fetching and modifying product and order details (https://atlan.com/relational-vs-document-database/).

Strong RBAC-Employs RBAC and Client-Side Field Level Encryption (CSFLE) which help ensure that sensitive data such as customer information and payment details are only accessible to those with the proper permission (https://www.mongodb.com/docs/manual/core/csfle/).

Scalability-Can be scaled horizontally to handle increased traffic during peak periods such as sales and holidays, which increases overall performance.

ACID-Compliant-Supports atomic operations across multiple documents and provides replica sets for data redundancy and consistency in processes such as payment processing and order handling (https://www.mongodb.com/resources/basics/databases/acid-transactions; https://www.mongodb.com/docs/manual/replication/).

## Weaknesses

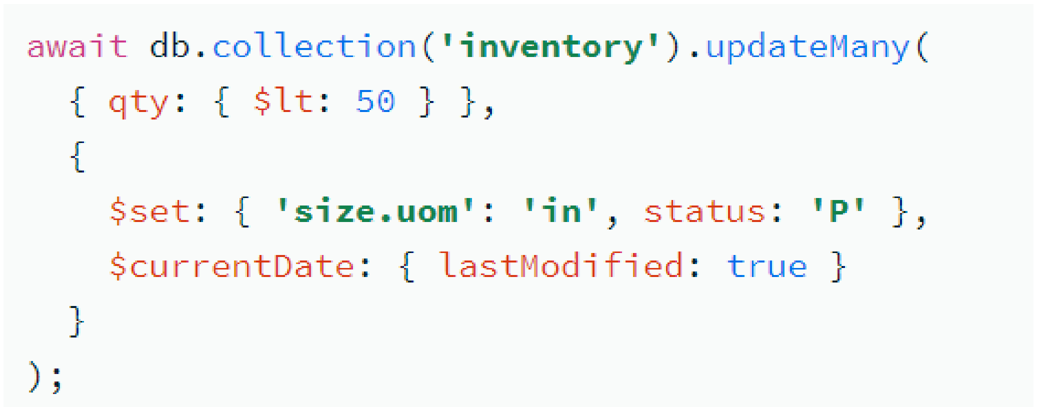
Inconsistency and Complexity-The flexibility of the schema can make managing old and new documents challenging which can result in inconsistencies when handling different data structures such as when new customer data is gathered, while old profiles lack these data fields (https://atlan.com/relational-vs-document-database/).

Storage Requirement and Memory Usage-Replica sets can lead to higher storage requirements and memory usage which can cost a lot when handling a large data volume in an e-commerce platform (https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb).

Lack Join Capabilities-The nature of a document model lacks join capabilities which can make data retrieval from multiple documents challenging during processes such as user interactions and generating reports or order information (https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb).

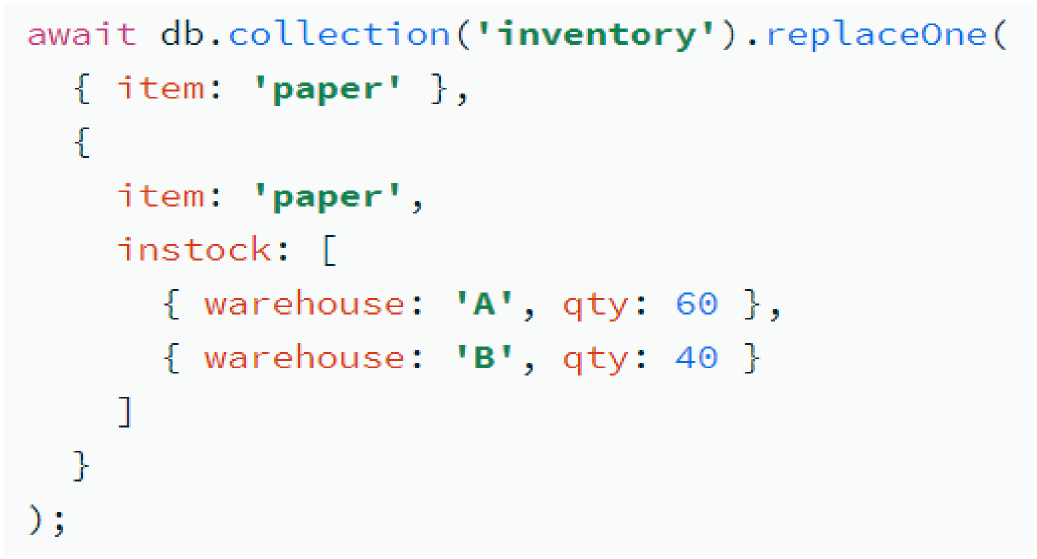
## Social Media Analysis

This second scenario features social media analytics applications or platforms such as Instagram Insights and TikTok Analytics. These applications serve as a tool to gather information regarding social media performance and audience engagement in digital marketing which are widely utilized by companies nowadays (https://blog.hootsuite.com/social-media-analytics/). Unlike OLTP databases, they usually consist of utilizing Online Analytical Processing (OLAP) databases, where they perform rapid and complex queries execution for multidimensional analysis on large volumes of data in a data repository (https://www.ibm.com/topics/oltp). Instead of performing business operations such as e-commerce transactions, it is mainly used



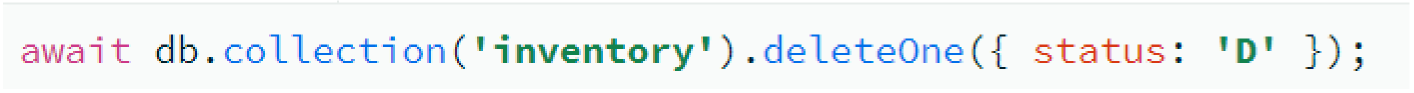
```
await db.collection('inventory').updateMany(
  { qty: { $lt: 50 } },
  {
    $set: { 'size.uom': 'in', status: 'P' },
    $currentDate: { lastModified: true }
  }
);
```

**Figure 39:** Example of Updating Multiple Data in MongoDB. (https://www.mongodb.com/docs/manual/tutorial/update-documents/).



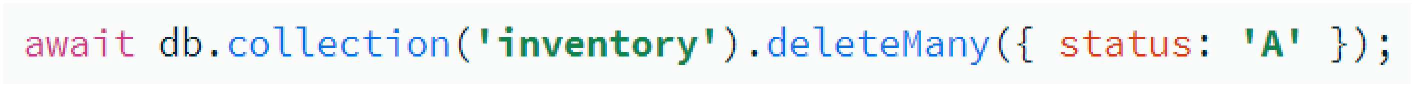
```
await db.collection('inventory').replaceOne(
  { item: 'paper' },
  {
    item: 'paper',
    instock: [
      { warehouse: 'A', qty: 60 },
      { warehouse: 'B', qty: 40 }
    ]
  }
);
```

**Figure 40:** Example of Replacing Data in MongoDB. (https://www.mongodb.com/docs/manual/tutorial/update-documents/).



```
await db.collection('inventory').deleteOne({ status: 'D' });
```

**Figure 41:** Example of Deleting Data in MongoDB. (https://www.mongodb.com/docs/manual/tutorial/remove-documents/).



```
await db.collection('inventory').deleteMany({ status: 'A' });
```

**Figure 42:** Example of Deleting Multiple Data in MongoDB. (https://www.mongodb.com/docs/manual/tutorial/remove-documents/).

for Business Intelligence (BI), decision support, and business forecasting which a social media analysis perform to generate insights on their digital marketing (https://www.ibm.com/topics/oltp).

The system needs to have the capability to handle extremely high volumes of data generated by users through posts, comments, likes, shares, and other interactions to produce valuable insight. Unlike the first scenario, this system is characterized by massive and continuously growing datasets, where schema complexity is varied and constantly evolving due to the dynamic and interconnected nature of social media data.

## Relational Model (Oracle)

### *Strengths*

Structured Data Management-Excels in organizing data into well-defined tables, ensuring data consistency and integrity through constraints like primary keys and foreign keys.

Robust Querying Capabilities-Allows for detailed analysis, such as tracking user engagement and identifying trends, which are essential for making informed decisions in social media strategies.

ACID Properties-Maintain data accuracy and ensure reliable transaction management in dynamic changes such as posts and comments.

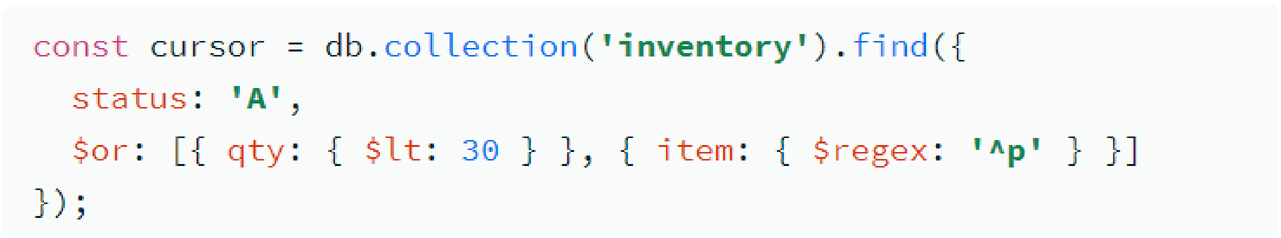
Comprehensive Access Control-Fine-grained permissions to protect user information and ensure compliance with data protection regulations (https://databasetown.com/relational-database-benefits-and-limitations/).

### Weaknesses

Scalability Challenges-Might has struggles to efficiently manage vast and unstructured datasets commonly found in social media platforms (https://databasetown.com/relational-database-benefits-and-limitations/).

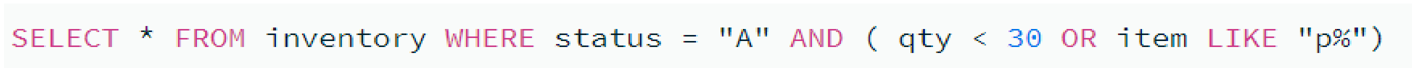
Rigid Schema Requirement-Inflexible schema causes difficulty in adapting to the evolving nature of social media data, which further complicates the addition of new data types or structural changes.

Performance Overhead-Enforcement of data integrity and ACID properties can lead to performance overhead, especially under high transaction loads, impacting the speed of real-time analytics (https://databasetown.com/relational-database-benefits-and-limitations/).



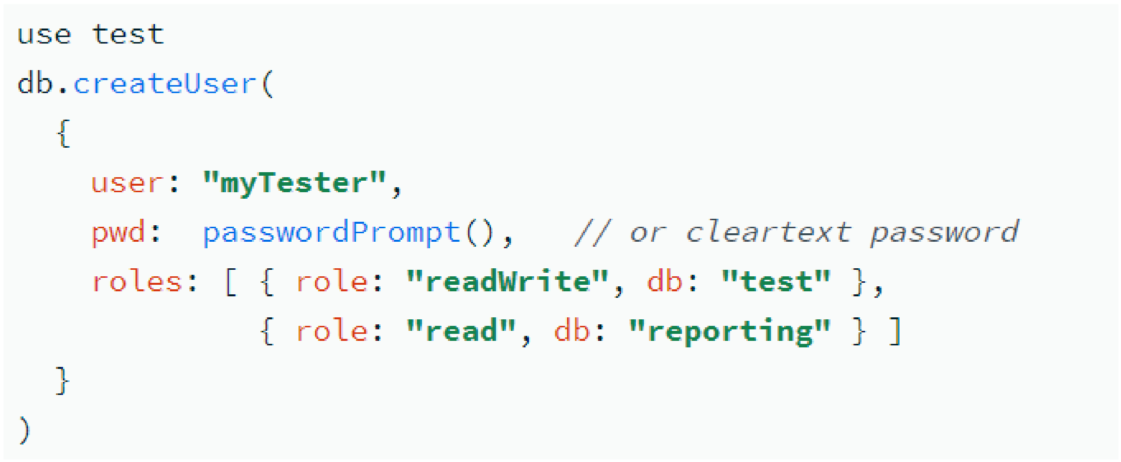
```
const cursor = db.collection('inventory').find({
  status: 'A',
  $or: [{ qty: { $lt: 30 } }, { item: { $regex: '^p' } }]
});
```

**Figure 43:** Data Retrieval Using OR Operation in MongoDB. (https://www.mongodb.com/docs/manual/tutorial/query-documents/).



```
SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
```

**Figure 44:** Data Retrieval Using AND Operation in MongoDB. (https://www.mongodb.com/docs/manual/tutorial/query-documents/).



```
use test
db.createUser(
  {
    user: "myTester",
    pwd:  passwordPrompt(),    // or cleartext password
    roles: [ { role: "readWrite", db: "test" },
             { role: "read", db: "reporting" } ]
  }
)
```

**Figure 45:** Access Control in MongoDB. (https://www.mongodb.com/docs/manual/tutorial/create-users/).

Complex Data Representation-Intricate relationships and hierarchical data in a relational model leads to inefficient queries and cumbersome database design once data volumes increase (https://databasetown.com/relational-database-benefits-and-limitations/).

Cost Considerations-Oracle databases can be expensive to license and maintain, particularly for configurations that demand high availability and performance (https://databasetown.com/relational-database-benefits-and-limitations/).

Limited Support for Unstructured Data-The relational model's limited support for unstructured data, such as text, images, and videos, poses a significant challenge. Integrating unstructured data into a relational framework can be complex and resource-intensive, potentially limiting the effectiveness of analytics.

## Graph Model (Neo4j)

### *Strengths*

Rapid Query Capabilities-Utilizes pattern matching to quickly traverse through complex and dynamic relationships between nodes for researching user behaviour from social media posts.

High Flexibility-Nature of schema optional allows new identified relationships to be added easily into the graph structure which accommodates the dynamic and evolving nature of user behaviour within social media platforms to be analysed.

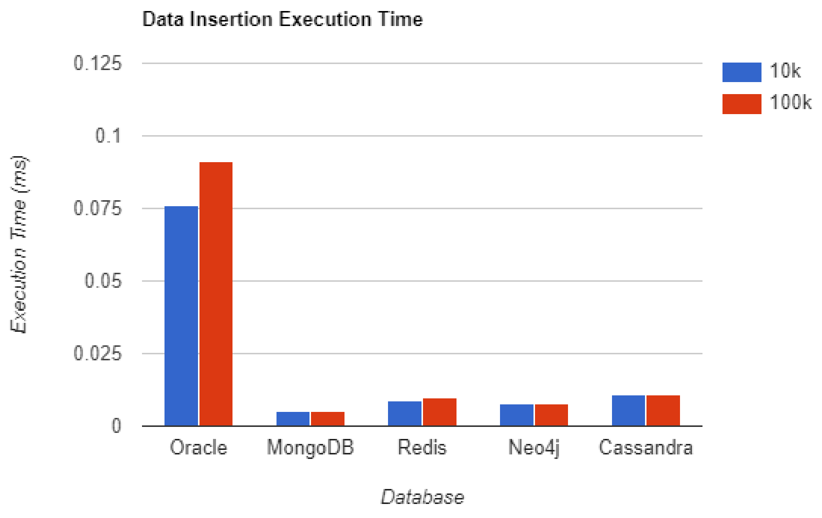


**Figure 46:** Query Performance of Data Insertion for Each DBMS (Čerešňák and Kvet, 2019; https://aws.amazon.com/compare/the-difference-between-acid-and-base-database/).

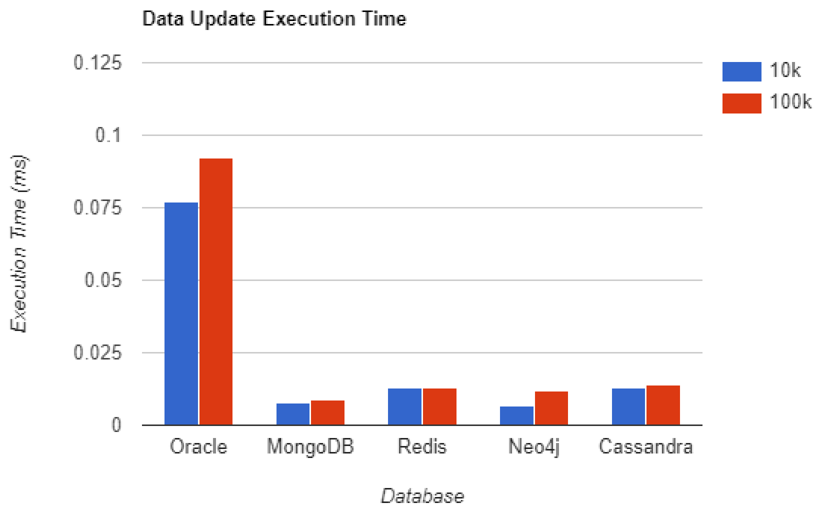


**Figure 47:** Query Performance of Data Update for Each DBMS (Čerešňák and Kvet, 2019).

Scalability-Excels in scaling horizontally which utilizes partitioning on different servers to parallelly process graph queries for processing huge amounts of social media data (https://aws.amazon.com/compare/the-difference-between-graph-and-relational-database/).

Fully ACID Compliant-Transactions performed are fully ACID compliant despite being a NoSQL database and schema optional which ensures data integrity and consistency within the complex social media graph network.

RBAC Mechanism-Supports RBAC mechanisms that comes with built-in roles that can be delegated to researchers with different level of access to social media data.

## Weaknesses

Only Effective for Relationship-Heavy Scenarios-Graph model only excels in handling complex relationship scenarios like this social media analytics platform, where it may find lacking in simple data storage and retrieval tasks.

Significant Computational Resources-Requires high memory consumption and significant computational resources to handle complex queries and large datasets for optimal performance in a social media analytics platform.

Complex Maintenance and Configuration-Requires extensive maintenance and configuration of data clusters within the social media graph network to ensure query efficiency and execution in a distributed environment.

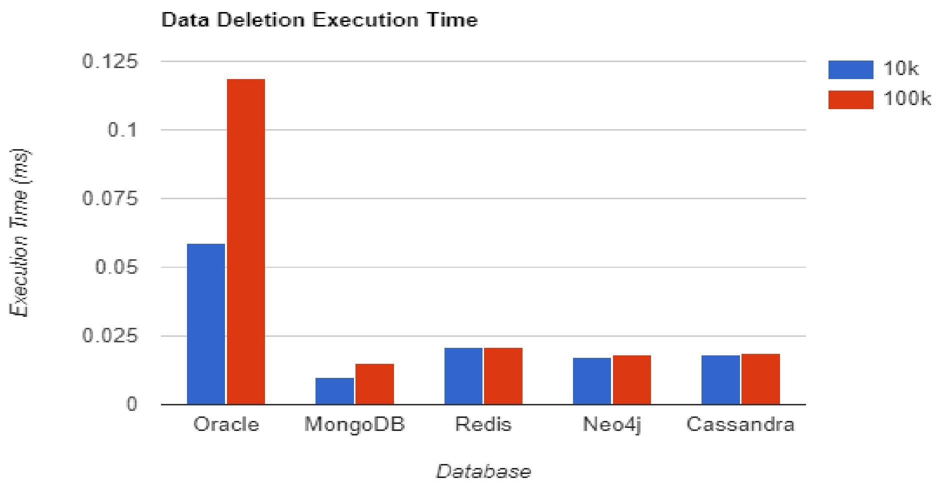


**Figure 48:** Query Performance of Data Deletion for Each DBMS (Čerešňák and Kvet, 2019).

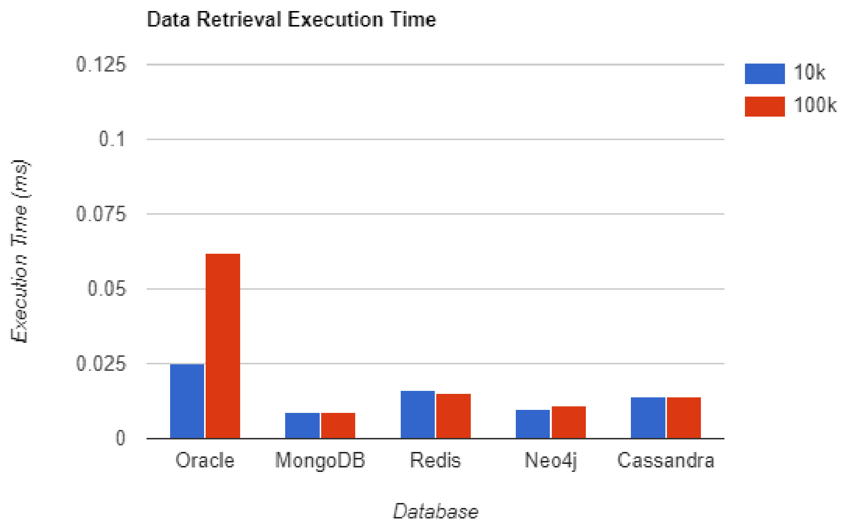


**Figure 49:** Query Performance of Data Retrieval for Each DBMS (Čerešňák and Kvet, 2019).

## Wide Column Model (Cassandra)

### *Strengths*

High Write Throughput-Optimized for high write throughput with low latency. Ideal for social media analytics since large amounts of data are ingested rapidly, such as logging social media interactions in real-time.

Scalability-Scales horizontally across multiple nodes with ease, allowing for distributed data storage and processing rapid growing data volumes.

Flexible Schema-Allows to add new columns without altering the existing table structure, which is good when dealing with varying types of social media data such as posts, comments and likes.

High Availability and Fault Tolerance-Able to provide high availability and fault tolerance. This is important as the analytics platform could remain operational even during node failures.

### Weaknesses

CQL Limitations-Less features when compared to SQL, lacks support for complex joins, subqueries and aggregate functions. This can be limiting when performing analytics or querying across multiple datasets in social media contexts.

Data Model Complexity-Requires careful planning and deep understanding of the query patterns upfront. Mistakes can lead to inefficient data retrieval, especially problematic in dynamic and evolving social media analytics environment.

Read Performance-Can suffer if the data is not partitioned appropriately, leading to potential bottlenecks in social media analytics where quick access to data is crucial.

Eventual Consistency-Wide column model follows an eventual consistency model, which might not be ideal for social media analytics where immediate consistency is required.

Operational Overhead-Requires significant operational expertise which can become complex and resource-intensive, especially when handling large volumes of social media data across multiple regions.

## Key-Value Model (Redis)

### *Strengths*

Fast Data Handling-In-memory storage allows for fast data handling with minimal latency to support the processing of large volume of data in real-time such as user interaction, activity feed and trending topics (https://redis.io/nosql/key-value-databases/).

Flexible Schema-Supports various types of entities which lets the platform perform changes such as adding new metrics or interactions without much effort (https://redis.io/nosql/key-value-databases/).

Efficient Caching-Caching capability reduces memory usage by storing frequently accessed data such as user profile and popular content (https://redis.io/nosql/key-value-databases/).

Partially ACID-Compliant-Supports atomic operations which help ensure consistency and improve accuracy in processes such as updating follower counts and interactions between users (https://redis.io/nosql/key-value-databases/).

Scalability-Can be scaled horizontally to handle increased traffic during high traffic periods such as viral and trending events, which increases overall performance (https://redis.io/nosql/key-value-databases/).

### Weaknesses

High Cost-In-memory storage utilizes RAM instead of traditional hard drives which can be costly to prepare the necessary hardware and infrastructure to support a large data volume present in a social media analytics platform that handles a large volume of posts and user interactions (https://alronz.github.io/Factors-Influencing-NoSQL-Adoption/site/Redis/Results/Strengths%20and%20Weaknesses/).

Slow Backups-Regular backups performed by the database can slow the system down as memory dumps are used to create snapshots which may cause reoccurring periods of lag while performing backups (https://alronz.github.io/Factors-Influencing-NoSQL-Adoption/site/Redis/Results/Strengths%20and%20Weaknesses/).

Lack of Complex Querying Capabilities-Data can only be accessed via key and there are no relationships in a key-value model which can complicate development and features such as user analysis and filtering of interactions (https://redis.io/nosql/key-value-databases/).

## Document Model (MongoDB)

### *Strengths*

Flexible Schema-Has a schema that is easily adaptable to changes. Can be used to store a wide range of information such as user profiles, posts and comments.

Complex Data Structures-Supports nested arrays and embedded documents which can help further embed information reducing the need for multiple collections and joins such as embedding the user interactions and metadata in a single document.

Real-Time Analytics-Provides real-time data analytics allowing for analysis of data such as user behaviour, statistics, and content engagement data.

Simplified Data Handling-Related data can be stored in a single document which reduces the need for joining. This makes retrieval and manipulation of data easier such as updating comments and posts when user's profile is changed (https://atlan.com/relational-vs-document-database/).

Strong RBAC-Employs RBAC and CSFLE which help ensure that sensitive data such as private messages and user analytics are only accessible to those with the proper permission (https://www.mongodb.com/docs/manual/core/csfle/).

Scalability-Can be scaled horizontally to handle increasing data volume as the amount of posts increases, which increases overall performance.

ACID-Compliant-Supports atomic operations across multiple documents and provides replica sets for data redundancy and consistency in processes such as user authentication and interactions (https://www.mongodb.com/resources/basics/databases/acid-transactions; https://www.mongodb.com/docs/manual/replication/).

## Weaknesses

Inconsistency and Complexity-The flexibility of the schema can make managing old and new documents challenging which can result in inconsistencies when handling different data structures such as when new user profile is created, while old profiles lack these data fields (https://atlan.com/relational-vs-document-database/).

Storage Requirement and Memory Usage-Replica sets can lead to higher storage requirements and memory usage which can cost a lot when handling a large data volume in a social media analytics platform (https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb).

Lack Join Capabilities-The nature of a document model lacks join capabilities which can make data retrieval from multiple documents challenging during processes such as user interactions and generation of content metrics (https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb).

## Summary of Results

Based on both performance evaluation and scenario analysis, the database models of NoSQL (graph, wide column, key-value, document) databases demonstrated overall better performance than SQL (relational) databases. The major preference factor of NoSQL databases is due to their flexibility in storing data in a schemaless structure which improves read and write operations and scalability. Traditional SQL databases on the other hand adheres to a strict and rigid structure schema which compromises performance to ensure data integrity may be found lacking to accommodate the dynamicity of business needs in this evolving digital era. However, the overall performance of the characteristics of each database model is not sufficient to be the only determining factor.

In Figure 50, Oracle DBMS that utilizes relational model is still the leading choice and popularity for users despite the emergence of NoSQL databases that demonstrated better overall performance in this study. This is due to the long history of SQL databases, where it is supported by a large community of users that are familiar with the user-friendly SQL and understands the reliability of using traditional SQL databases (https://www.testgorilla.com/blog/sql-vs-nosql). The ease of usability and rigidity of schema may overwhelm the trade-offs of using NoSQL databases due to their trustworthy and robust reputation which is still relevant for many businesses. However, this also does not mean that SQL databases are better than NoSQL databases in implementation.

**Table 4: Summarisation of Implementation of RBAC.**

| Database | RBAC Granularity | Built-in Roles | Custom Roles | Difficulty of Implementation |
|---|---|---|---|---|
| Oracle | Cell-level | Extensive | image50 | Easy |
| Neo4J | Graph-level | Basic | image50 | Moderate |
| Cassandra | Column-family-level | Limited | image50 | Moderate |
| Redis | Cluster-level | Limited to administrative roles | image52 | Hard |
| MongoDB | Collection-level | Extensive | image50 | Moderate |

**Table 5: Summarisation of Data Integrity.**

| Database | BASE | ACID | Implementation Difficulty |
|---|---|---|---|
| Oracle | image52 | image50 | Easy |
| MongoDB | image52 | image50 | Moderate |
| Redis | image50 | Partially | Difficult |
| Neo4j | image52 | image50 | Moderate |
| Cassandra | image50 | image52 | Difficult |

In Figure 51, database models of NoSQL databases have shown a significant increase in popularity over relational DBMS. This phenomenon is due to the dynamic nature of business which is constantly evolving, where new DBMS models are developed to accommodate certain needs and different scale of data. However, SQL database that utilizes relational model is still relevant in this era as not all businesses are required to handle such large scale of data operations. Thus, there is no saying that NoSQL databases are better than SQL databases as the performance of each database model heavily relies on the context of application and usage.

## DISCUSSION AND CONCLUSION

All databases have its advantage, making it more suitable for specific applications. The table below summarizes and compares the data models in 5 aspects: security, integrity, query performance, availability, and scalability.

### Security

Oracle's relational model provides strong protection with advanced access control mechanisms, including RBAC and relational view/virtual tables. Redis offers RBAC for defining roles and managing access, but its security features are more basic compared to Oracle. Neo4j secures graph data using RBAC, ensuring controlled access to the relationships and nodes within the graph. MongoDB provides detailed access control and data protection with the features of RBAC and CSFLE. Cassandra supports multi-tenant environments and extensive data protection via complex security configurations.

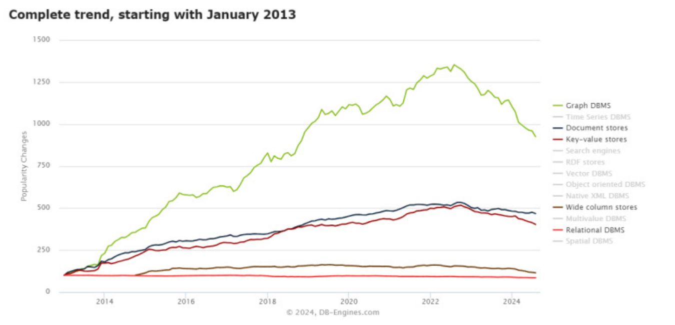


**Figure 51:** Ranking of Database Models Ranging 2013 to 2024 August (https://db-engines.com/en/ranking_categories).

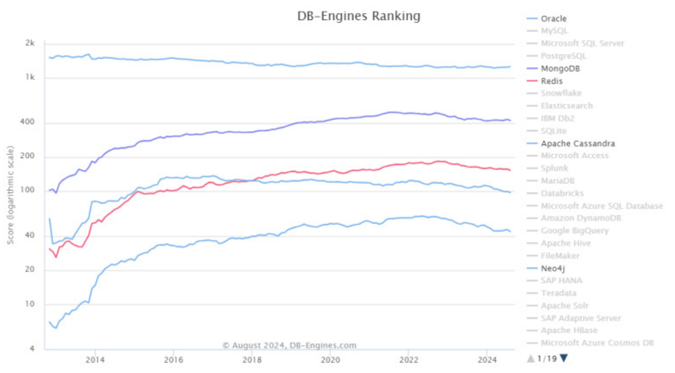


**Figure 50:** Ranking of Database Engines From 2014 to 2024 August (https://db-engines.com/en/ranking_trend).

### Integrity

Oracle has ACID compliance and referential integrity to ensure consistent data relationships. Redis only provides basic data integrity and lacks support for complex transactions. Neo4j is fully ACID compliant, which is makes it suitable for accurate analytics and decision-making in graph-based data. MongoDB also supports ACID transactions, which ensures documents are consistent. Cassandra has consistency levels that can be tuned, based on the requirements needed.

### Query performance

Oracle's relational model has great query performance with the use of SQL but the performance will decrease when there are complex joins and large datasets. Redis is quick in simple key-value operations due to its in-memory storage but it lacks advanced querying features. Neo4j is strong in querying complex relationships through its advanced graph traversal capabilities. MongoDB offers efficient querying through embedded documents and indexing, and also supports real-time analytics for dynamic and evolving data. Cassandra performs well with large-scale data and complex queries.

### Availability

High availability is achieved through clustering and sharding by Oracle. However, it can be complex and costly to implement. Redis does it through clustering and replication, but availability is generally limited by its in-memory nature. Neo4j also maintains high availability with clustering and replication but it is difficult to maintain consistency across extensive graphs. MongoDB uses replica sets to provide redundancy and failover capabilities, and also supports horizontal scaling. Cassandra is highly available with built-in replication across multiple nodes and data centers, offering redundancy and fault tolerance. However, it may be complex to set up and manage.

### Scalability

Oracle's rigid schema and complex dataset management may affect its scalability. Redis is highly scalable horizontally by adding more keys but is constrained by memory limits and associated costs. Neo4j scales well with interconnected data but high computational resources would be needed. MongoDB offers flexible horizontal scalability through sharding and hence, can handle large volumes of data and adapt to evolving structures. Cassandra is designed for horizontal scaling and effectively manages large data volumes and high throughput.

By providing a comparative examination of several data models across important database concerns, this study additionally builds on earlier research. Previous studies were mostly focused on individual models or specific applications. Hence, this research focused on offering a wider perspective on how various models function in diverse settings. It also emphasizes the usefulness of selecting a database model in accordance with particular use cases, in which we have chosen social media analytics platforms and e-commerce platforms as example.

Through this report, the team has learnt that each database model excels in different areas but also has its limitations. When deciding on which database model is most suitable, we should consider the specific requirements and priorities of the application. As no single model is universally superior, each database models pros and cons should be carefully considered to implement an efficient and effective database that could cater to the application's need.

## LIMITATIONS

There are several limitations in this study. The quantitative data in this study were collected only under specific conditions and environments because we adopted only two from academic papers instead of own testing. This is due to the lack of existing computational resources, complexity of factors that would affect database performance and the team's limited experience in conducting controlled environment tests. There are also various complexities in the actual operating environment that may not be sufficiently reflected. For example, this study mainly focused on the performance evaluation of the basic database. However, in the real environment, more factors such as security, recovery time, and maintenance cost determine the performance and efficiency of the database.

Furthermore, since the amount of data collected in the experimental environment of the academic paper referred was small (10k and 100k queries), practical variables were not sufficiently reflected. For example, the research results were concluded without considering failures, traffic peaks, and unexpected data patterns. More than millions of records can be processed in the real environment, so the database was not evaluated based on sufficient samples.

Due to these limitations, the gap with the actual operating environment may be significant.

## FUTURE STUDIES

The research only compares the SQL and NoSQL database without considering the possibility of a more efficient database that absorbs the advantages of each database. In other words, there is a desire to study hybrid database systems.

Modern applications have to process both structured and unstructured data. For example, an e-commerce platform ideally will need to store transaction data in a SQL database and unstructured data, such as users' review boards, in a NoSQL database.

In addition, the scalability of database systems has become a hot topic as the amount of data grows exponentially. SQL databases and NoSQL databases have strengths in vertical and horizontal expansion, respectively. There will also be an increasing demand for databases that provide balanced scalability that absorbs both strengths. Furthermore, by combining the features of SQL databases where integrity and data consistency are important and NoSQL databases where fast data access and flexibility are important, answers to meet both requirements can be studied upon.

### Group member contributions

| Student Name (ID) | Percentage of contribution in each section (1-6) | Percentage of the overall participation |
|---|---|---|
| Dionne Teh Wooi Ying (21074356) | 2 (20%), 5 (50%), proofread and edit report | 100% |
| Cheah Shaoren (21051982) | 2 (20%), 3 (100%), 4 (33%) | 100% |
| Chin Wey Ken (21048012) | 2 (20%), 4 (33%), 6 (100%) | 100% |
| Choo Yan Jie (21061734) | 2 (20%), 4 (33%), compile report | 100% |
| Hong Chang Eui (23037476) | 1 (100%), 2 (20%), 5 (50%) | 100% |

## CONFLICT OF INTEREST

The authors declare that there is no conflict of interest.

## ABBREVIATIONS

**SQL:** Structured Query Language; **NoSQL:** Not Only SQL; **DBMS:** Database Management System; **RDBMS:** Relational Database Management System; **JSON:** JavaScript Object Notation; **API:** Application Programming Interface.

## REFERENCES

Al-Saeedi, B. Strengths and weaknesses-factors influencing NoSQL adoption. GIThub. Retrieved August 05, 2024, https://alronz.github.io/Factors-Influencing-NoSQL-Adoption/site/Redis/Results/Strengths%20and%20Weaknesses/

Amazon web services. "ACID vs BASE databases-difference between databases-AWS," Amazon Web Services, Inc. Retrieved August 08, 2024, https://aws.amazon.com/compare/the-difference-between-acid-and-base-database/

Amazon web services. "Graph vs relational databases-difference between databases-AWS," Amazon Web Services, Inc. Retrieved August 08, 2024, https://aws.amazon.com/compare/the-difference-between-graph-and-relational-database/

Atlan. (December 19, 2023). Relational vs document database: 9 key differences! Atlan. Retrieved August 03, 2024, https://atlan.com/relational-vs-document-database/

AWS. What is a document database? Amazon web services, Inc. Retrieved August 03, 2024, https://aws.amazon.com/nosql/document/

Cassandra, A. (2024). 'Apache Cassandra | Apache Cassandra Documentation,' Cassandra. Retrieved August 03, 2024, https://cassandra.apache.org/_/index.html

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

DatabaseTown. "Relational database benefits and limitations (advantages & disadvantages)-DatabaseTown," Database Town. Retrieved August 08, 2024, https://databasetown.com/relational-database-benefits-and-limitations/

DataStax. (April 17, 2024). "What is Apache Cassandra?" | Open Source Database," DataStax. Retrieved August 03, 2024, https://www.datastax.com/guides/what-is-cassandra

DataStax. "About the Cassandra query language (CQL)" | CQL for DataStax Hyper-Converged Database | DataStax Docs," DataStax. Retrieved August 03, 2024, https://docs.datastax.com/en/cql/hcd-1.0/overview/cql-about.html

DataStax. "Apache Cassandra Structure | CQL for DataStax Hyper-Converged Database | DataStax Docs," DataStax. Retrieved August 03, 2024, https://docs.datastax.com/en/cql/hcd-1.0/overview/cassandra-structure.html

DataStax. "How are consistent read and write operations handled?" | Apache Cassandra 3.x," DataStax. Retrieved August 03, 2024, https://docs.datastax.com/en/cassandra-oss/3.x/cassandra/dml/dmlAboutDataConsistency.html

DB-engines, "DB-Engines Ranking per database model category," DB-Engines. Retrieved August 08, 2024, https://db-engines.com/en/ranking_categories.

DatabaseTown, "Wide Column Database (Use Cases, Example, Advantages & Disadvantages)-DatabaseTown," DatabaseTown. Retrieved August 08, 2024, https://databasetown.com/wide-column-database-use-cases/.

DB-Engines. "historical trend of the popularity ranking of database management systems," DB-engines. Retrieved August 08, 2024, https://db-engines.com/en/ranking_trend

EvalCommunity. Theory-based evaluation approach-EvalCommunity. Eval. Community. Retrieved August 05, 2024, https://www.evalcommunity.com/career-center/theory-based-evaluation-approach/

IBM. What are NoSQL databases? | IBM. IBM. Retrieved August 03, 2024, https://www.ibm.com/topics/nosql-databases

IBM. What is OLTP? | IBM. IBM. Retrieved August 08, 2024, https://www.ibm.com/topics/oltp

IBM. What is redis explained? | IBM. IBM. Retrieved August 03, 2024, https://www.ibm.com/topics/redis

Knight, M. (April 28, 2021). What is a property graph? DATAVERSITY. https://www.dataversity.net/what-is-a-property-graph/

KT. (November 11, 2022). Retrieved August 08, 2024, https://www.testgorilla.com/blog/sql-vs-nosql. SQL vs. NoSQL: Full comparison of features, differences, and more. TestGorilla.

Lutkevich, B., Alexander, S. G., & Biscobing, J. (June 01, 2024). What is a relational database? Tech. Target. Retrieved August 05, 2024, https://www.techtarget.com/searchdatamanagement/definition/relational-database

Mongo, D. B. (2024). 'Client-Side Field Level Encryption,' MongoDB. Retrieved August 05, 2024, https://www.mongodb.com/docs/manual/core/csfle/

Mongo, D. B. ACID properties in DBMS explained. MongoDB. Retrieved August 05, 2024, https://www.mongodb.com/resources/basics/databases/acid-transactions

Mongo, D. B. Create a user-MongoDB Manual. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/docs/manual/tutorial/create-users/

Mongo, D. B. Databases and collections-MongoDB Manual. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/docs/manual/core/databases-and-collections/

Mongo, D. B. Delete documents-MongoDB Manual. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/docs/manual/tutorial/remove-documents/

Mongo, D. B. Insert documents-MongoDB Manual. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/docs/manual/tutorial/insert-documents/

Mongo, D. B. JSON and BSON. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/resources/basics/json-and-bson

Mongo, D. B. MongoDB CRUD operations-MongoDB Manual. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/docs/manual/crud/

Mongo, D. B. MongoDB Shell (mongosh)-MongoDB Shell. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/docs/mongodb-shell/

Mongo, D. B. Query documents. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/docs/manual/tutorial/query-documents/

Mongo, D. B. Replication. MongoDB. Retrieved August 05, 2024, https://www.mongodb.com/docs/manual/replication/

Mongo, D. B. Role-based access control-MongoDB Manual. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/docs/manual/core/authorization/

Mongo, D. B. Update documents-MongoDB Manual. MongoDB. Retrieved August 03, 2024, https://www.mongodb.com/docs/manual/tutorial/update-documents/

Mongo, D. B., "Transactions-MongoDB Manual," MongoDB. Retrieved August 05, 2024, https://www.mongodb.com/docs/manual/core/transactions/

Neo4j. (2024a). 'Graph database concepts-Getting Started,' Neo4j. Retrieved August 03, 2024, https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/

Neo4j. (2024b). 'Overview-Cypher Manual,' Neo4j. Retrieved August 03, 2024, https://neo4j.com/docs/cypher-manual/current/introduction/cypher-overview/

Neo4j. (2024c). 'What is a graph database? -Getting Started,' Neo4j. Retrieved August 03, 2024, https://neo4j.com/docs/getting-started/get-started-with-neo4j/graph-database/

Neo4j. (2024d). 'Authentication and authorization-Operations Manual,' Neo4j. Retrieved August 03, 2024, https://neo4j.com/docs/operations-manual/current/authentication-authorization/

Neo4j. (2024e). 'Built-in roles and privileges-Operations Manual,' Neo4j. Retrieved August 03, 2024, https://neo4j.com/docs/operations-manual/current/authentication-authorization/built-in-roles/

Neo4j. (2024f). 'Database internals and transactional behavior-Operations Manual,' Neo4j. Retrieved August 03, 2024, https://neo4j.com/docs/operations-manual/current/database-internals/

Neo4j. (2024g). 'Transaction logging-Operations Manual,' Neo4j. Retrieved August 03, 2024, https://neo4j.com/docs/operations-manual/current/database-internals/transaction-logs/

Newberry, C. (June 27, 2024). 21 of the best social Media Analytics tools for 2024. Hootsuite. Retrieved August 05, 2024, https://blog.hootsuite.com/social-media-analytics/

Nile. Retrieved August 05, 2024, https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb. KnowledgeNile, "Ultimate guide MongoDB: Definition, advantages & disadvantages." Knowledge.

Oracle. Data integrity. Oracle. Retrieved August 03, 2024, https://docs.oracle.com/cd/E18283_01/server.112/e16508/datainte.htm.

Oracle. Database security guide. Oracle. https://docs.oracle.com/database/121/DBSEG/authorization.htm#DBSEG124.

Oracle. Maintaining data integrity through constraints. Oracle. Retrieved August 05, 2024, https://docs.oracle.com/cd/B10500_01/appdev.920/a96590/adg05itg.htm.

Oracle. Transaction management. Oracle. Retrieved August 03, 2024, https://docs.oracle.com/cd/B14117_01/server.101/b10743/transact.htm.

Oracle. What is a relational database? Oracle. Retrieved August 03, 2024, https://www.oracle.com/database/what-is-a-relational-database/.

Redis. (2024). 'GET | Docs,' redis. Retrieved August 03, 2024, https://redis.io/docs/latest/commands/get/

Redis. Access control | Docs. Redis. Retrieved August 03, 2024, https://redis.io/docs/latest/operate/rs/security/access-control/

***53***. Redis. ACL | Docs. Redis. Retrieved August 05, 2024, https://redis.io/docs/latest/operate/oss_and_stack/management/security/acl/

Redis. Del | Docs. Redis. Retrieved August 03, 2024, https://redis.io/docs/latest/commands/del/

Redis. Docs Redis. "Transactions. Retrieved August 03, 2024, https://redis.io/docs/latest/develop/interact/transactions/

Redis. Set | Docs. Redis. Retrieved August 03, 2024, https://redis.io/docs/latest/commands/set/

Redis. Understand Redis data types. Redis. Retrieved August 03, 2024, https://redis.io/docs/latest/develop/data-types/

Redis. What is a key-value database? Redis. Retrieved August 03, 2024, https://redis.io/nosql/key-value-databases/

Scylla, D. B. T.C. development. "What is a wide-column database? Definition & FAQs." Retrieved August 03, 2024, https://www.scylladb.com/glossary/wide-column-database/

Stax. (March 17, 2015). Retrieved August 03, 2024, https://www.datastax.com/blog/role-based-access-control-cassandra. DataStax, "Role based access control in Cassandra." Data.

Words doctorate. (June 04, 2021). "Write a methodology in research proposal example," Words Doctorate. Retrieved August 05, 2024, https://www.wordsdoctorate.com/blog-details/methodology-in-research-proposal-example/

**Cite this article:** Ying DTW, Jie CY, Shaoren C, Eui HC, Ken CW, Sathishkumar VE, *et al*. Comparative Analysis of SQL and NoSQL Databases: Data Models, Use Cases, and Performance Insights. Info Res Com. 2025;2(1):47-70.