Structured Query Language vs. Not Only SQL: A Comparative Analysis

Md Radif Rafayet Chowdhury **Saborni Barua** **Zahin Un Nafi**

103539316 103512168 103539510

Swinburne University of Technology

November 2023

Abstract

This comparative analysis compares SQL and NoSQL databases in the context of contemporary data management needs. It examines SQL's structured, table-based approach, as used in MySQL, against NoSQL's flexible storage models, like those in MongoDB. By constructing and analyzing custom databases with MariaDB and MongoDB, the study evaluates CRUD operations, data modeling, and performance, particularly for large datasets.

The shift of major companies toward NoSQL is scrutinized, with findings indicating NoSQL's superior scalability and efficiency for larger data volumes. Performance tests reveal that NoSQL can outpace SQL in high-demand scenarios, although SQL remains a robust option for certain applications. The review synthesizes evidence from multiple sources, establishing a nuanced understanding of the databases' respective strengths and informing the ongoing debate in database technology adoption.

Key words: database management, data scalability, performance benchmarking, technology adoption

JEL category: C88

1. Introduction

In the landscape of data management, the significance of efficient and scalable databases is paramount. SQL, an acronym for Structured Query Language, has been a mainstay for relational databases since the 1970s. Developed by IBM, SQL and its relational database model are still foundational to modern data systems, as seen in widely-used platforms like MySQL and Oracle. These databases organize data into tables marked by rows and columns, where primary keys ensure the integrity of each record.

MySQL, a prominent open-source relational database management system, exemplifies the traditional architecture of SQL-based data storage. It is constructed using C and C++, offering the flexibility for adaptations and capable of storing vast quantities of diverse data types. Its design, rooted in the client-server model, enables integration with various programming languages, facilitating complex data operations.

In contrast, NoSQL databases represent a paradigm shift, favoring a schema-less, non-relational approach that allows for the rapid handling of large, unstructured data sets. MongoDB, a leading NoSQL database, typifies this approach with its document-oriented structure that contrasts the rigid, table-based format of SQL systems.

This paper aims to elucidate the operational differences between SQL and NoSQL databases, focusing on query execution speed and syntactic complexity. Through a carefully designed comparative analysis, the study will explore the two database models' efficacy in handling modern data management tasks. Given the increasing volume and complexity of data in various domains, the findings will shed light on the suitability of each database type for contemporary applications, ultimately guiding businesses and developers in their choice of data management solutions.

The structure of the paper is as follows: Section 2 provides an overview of SQL and NoSQL databases, delineating key aspects such as language, scalability, and data management. Section 3 offers a detailed comparison of query syntax and structure. In Section 4, the focus shifts to practical implementations of each database type. Section 5 interprets these findings, and the paper culminates with a conclusion in Section 6, summarizing the insights gathered.

2. Overview

SQL (Structured Query Language) databases are relational, designed to handle structured data with defined relationships. This traditional model relies on a schema-based, table-like structure where data is stored in rows and columns, and operations are performed using SQL queries. They excel in ACID compliance (Atomicity, Consistency, Isolation, Durability), ensuring reliable transactions, making them ideal for complex queries and reports.

NoSQL databases, or "Not Only SQL," are non-relational and provide a more flexible data model that can handle unstructured and semi-structured data. These databases do not require a fixed schema and can store data in various formats like key-value pairs, documents, wide-column stores, or graphs. They are

designed to excel in horizontal scaling and are optimized for large data sets and real-time web applications, offering high performance for specific types of data and queries.

Below is a table that succinctly outlines the fundamental differences between SQL and NoSQL databases:

Feature	SQL (Relational)	NoSQL (Non-Relational)
Data Structure	Structured, schema-dependent tables.	Flexible, schema-less models like key-value, document, graph, wide-column stores.
Schema	Fixed, requires migration for alterations.	Dynamic, allowing for on-the-fly modifications.
Query Language	Structured Query Language (SQL).	Varies (e.g., NoSQL query language, JavaScript Object Notation (JSON)).
Scalability	Vertical (scale-up by adding more powerful hardware).	Horizontal (scale-out by adding more nodes to the system).
Transactions	ACID-compliant, suited for complex transactions.	BASE (Basically Available, Soft state, Eventual consistency), allowing for fast and flexible transactions.
Consistency	Strong consistency model.	Eventual or tuneable consistency models.
Best Use Cases	Complex queries, data integrity, and relational data structures.	Large-scale, distributed data with variable structures and agile development needs.
Examples	MariaDB, MySQL, Oracle, SQL Server.	MongoDB, Cassandra, Couchbase, Redis.

Table 1: Differences SQL and NoSQL

3. Comparative Analysis

The central thesis of this comparative analysis is that the choice between SQL and NoSQL databases should be dictated by the specific requirements of the application, with a clear understanding of the underlying differences in language, scalability, sharding, partitioning, and data replication mechanisms. SQL databases offer a rich, structured query language and strong consistency models, which can be optimal for transactions requiring high levels of data integrity. In contrast, NoSQL databases provide flexible schema design, superior scalability options, and varied data replication techniques that cater to large-scale distributed systems and handle massive volumes of data more efficiently.

1. Language

SQL databases, exemplified by MariaDB, employ a standardized Structured Query Language (SQL) that is highly expressive, allowing for complex queries and transactions. It is designed around a predefined schema and operates on a row-and-column-based architecture. NoSQL databases like MongoDB use a more dynamic query language that caters to unstructured data and document-like storage systems, often using JSON-like formats. This can lead to a more simplified development process with fewer constraints on how data is ingested and managed.

2. Scalability

When it comes to scalability, SQL databases traditionally scale vertically, requiring more powerful hardware to handle increased loads. This can become a limitation when dealing with very large datasets or high throughput demands. NoSQL databases, however, are engineered to scale horizontally, across commodity servers, making them well-suited for cloud-based services and applications that experience massive, unpredictable growth.

3. Sharding

Sharding, the process of distributing data across multiple servers, is a critical component of scalability. SQL databases typically implement sharding through custom, often complex architectures. MariaDB, for example, may require additional configuration and management to effectively shard data. NoSQL databases like MongoDB, however, have built-in support for automatic sharding, enabling more straightforward distribution of data across a cluster of machines.

4. Partitioning

Partitioning refers to dividing a database into parts to make it easier to manage. SQL databases usually partition data based on ranges or lists, which can complicate queries across partitions. In contrast, NoSQL systems frequently utilize a more flexible partitioning scheme, often distributing data based on a hash of a key attribute, which can facilitate faster access and management of data.

5. Data Replication

For ensuring data consistency and availability, SQL databases, such as MariaDB, implement several replication strategies:

- **Asynchronous Replication:** Changes made to the primary database are recorded and then replayed on the replica databases. This approach does not guarantee that the replicas are immediately consistent with the primary, but it offers the advantage of less impact on the performance of the primary server because the transaction does not need to wait for the replica to acknowledge the write.

- **Synchronous Replication:** In this method, a transaction is only considered complete when changes have been fully applied to both the primary and the replica databases. This ensures strong consistency but can introduce latency, as the primary database transaction must wait for the replica to confirm the write.

NoSQL databases, such as MongoDB, also provide various replication models:

- **Master-Slave Replication:** This model includes one primary node that receives all write operations and multiple secondary nodes that replicate the primary node's data. The primary node handles all write operations while secondary nodes can be used to scale out read operations and provide redundancy.

- **Peer-to-Peer Replication:** In a peer-to-peer (or multi-master) setup, replication occurs across all nodes in the system, and each node can accept write operations. This allows for a distributed system that can handle read and write operations across multiple nodes, potentially increasing availability and fault tolerance.

3.1 Query Differences

In this portion of the comparative analysis, we aim to illustrate the fundamental differences in query syntax and structure between SQL and NoSQL database systems. To provide a clear and applicable comparison, we will utilize MariaDB—a popular SQL relational database management system—as the representative for SQL databases. For NoSQL, MongoDB, known for its flexible schema and document-oriented storage, will serve as the example. By comparing specific queries from each system, we can observe how similar outcomes are achieved through their distinct languages and paradigms. The following table showcases these differences, presenting side-by-side examples of how common database operations are executed within MariaDB and MongoDB:

Operation	SQL (MariaDB)	NoSQL (MongoDB)
Create Database	CREATE DATABASE database_name;	use database_name
Create Table/Collection	CREATE TABLE table_name (column1 datatype, column2 datatype, ...);	db.createCollection('collection_name')
Insert Data	INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);	db.collection_name.insert({key1: value1, key2: value2, ...})
Select Data	SELECT column1, column2 FROM table_name;	db.collection_name.find({})
Update Data	UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;	db.collection_name.update({key: value}, {$set: {key1: value1, key2: value2}})
Delete Data	DELETE FROM table_name WHERE condition;	db.collection_name.remove({key: value})

Table 2: Differences SQL and NoSQL Queries

4. Real-World Analysis

We examine the detailed performance metrics of MySQL and MongoDB. The tests were designed to measure how each database handles a variety of operations with a dataset consisting of 5311 entries. Our comparison includes reading datasets, single query performance, multiple consecutive queries, sorting of entries, and both single and bulk insert operations. Additionally, we evaluate the performance impact of dropping collections in each database system.

4.1 Performance Comparison Table

The results are consolidated into a comprehensive table that reflects the time taken by MySQL and MongoDB to complete each specified task. This tabulation is crucial for our analysis as it provides clear, numeric evidence of the performance capabilities of both databases under identical test conditions.

SQL Database (MySQL)	NoSQL Database (MongoDB)
1. Read all data	1. Read all data
2. Single Query	2. Single Query
3. Five Consecutive Queries	3. Five Consecutive Queries
4. Sorting All	4. Sorting All
5. Three Queries and Sort	5. Three Queries and Sort
6. Insert	6. Insert
7. Insert Multiple	7. Insert Multiple

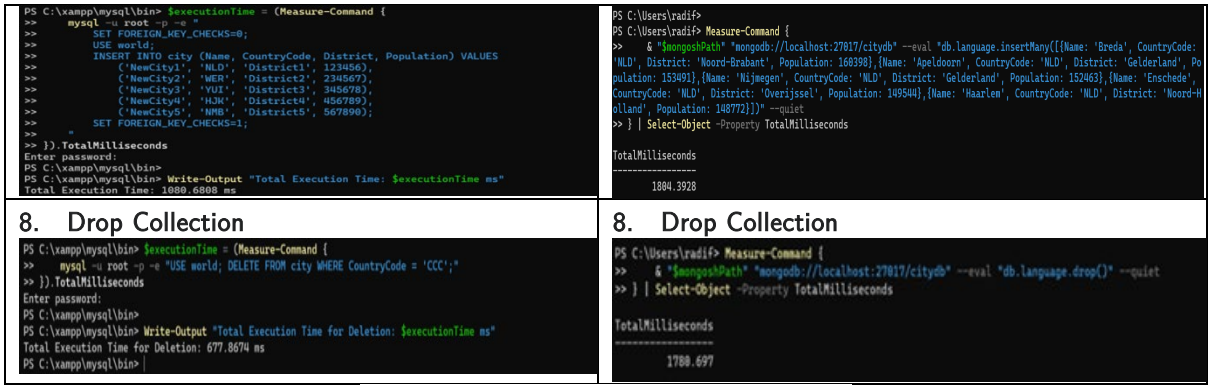

Table 3: MySQL vs MongoDB Test Results

4.2 Graphical Analysis of Results

This subsection presents a graphical analysis that translates our data into a visual format. The graph is designed to provide an at-a-glance comparison of the performance differences between MySQL and MongoDB. By plotting the time taken for each operation, we can visually assess the relative efficiency of the two database systems.

In Figure 1, the graph displays time metrics in scientific notation on the vertical axis, allowing for a precise comparison of performance times across a range of magnitudes. The horizontal axis enumerates the various tests conducted, with the length of the bars indicating the time taken to execute the corresponding task.

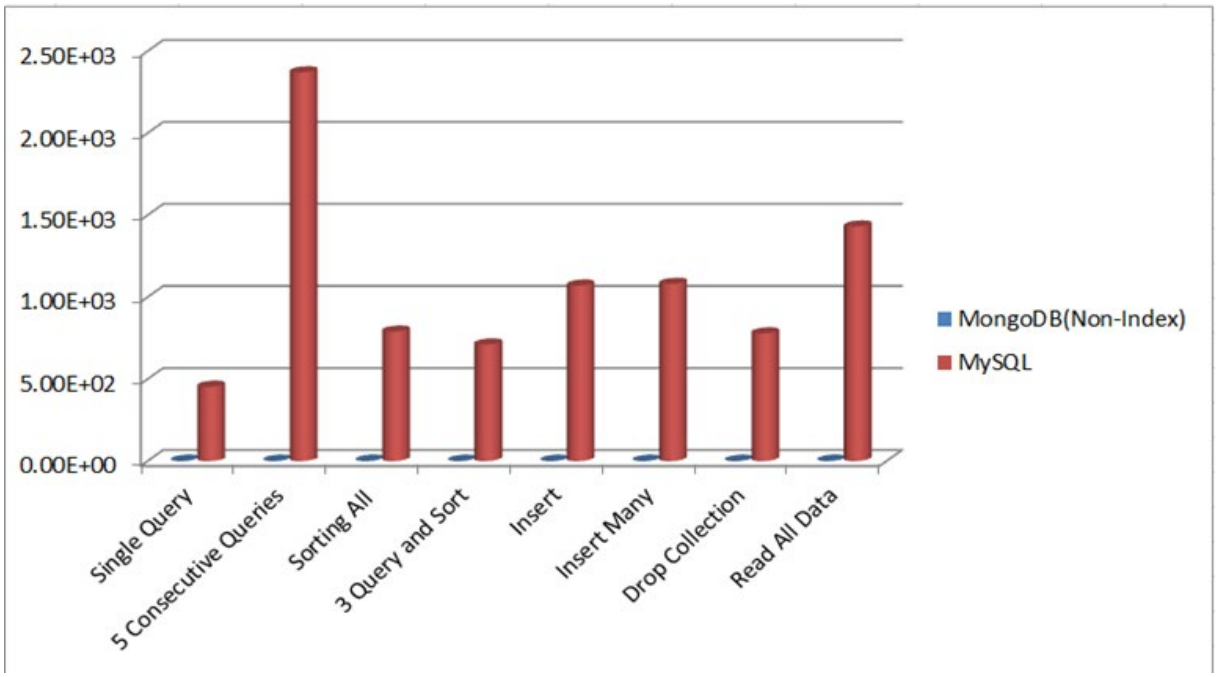

Figure 1: MySQL vs MongoDB Graphical Analysis

Time needed to insert 5311 entries for MongoDB was 956.5151 seconds

```
PS C:\Users\radif> Measure-Command { & mongoimport --uri="mongodb://localhost:27017/citydb" --collection="language" --ty
pe="csv" --file="D:\dataset\world_1.csv" --headerline }
2023-11-04T23:49:14.928+1100    connected to: mongodb://localhost:27017/citydb
2023-11-04T23:49:14.993+1100    5311 document(s) imported successfully. 0 document(s) failed to import.


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 956
Ticks             : 9565151
TotalDays         : 1.10707766203704E-05
TotalHours        : 0.000265698638888889
TotalMinutes      : 0.0159419183333333
TotalSeconds      : 0.9565151
TotalMilliseconds : 956.5151
```

Figure 2: MongoDB Performance Testing

Whereas, time needed to insert 5311 entries for MySQL was 2362.4678 seconds

```
PS C:\Users\User> $Env:PATH += ";C:\xampp\mysql\bin"
PS C:\Users\User> Measure-Command {
>>    Get-Content "C:\COS20015\data\world.sql" -Raw | mysql -u root -p world
>> }
Enter password:


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 2
Milliseconds      : 362
Ticks             : 23624678
TotalDays         : 2.734337731481481E-05
TotalHours        : 0.000656241055555555
TotalMinutes      : 0.0393744633333333
TotalSeconds      : 2.3624678
TotalMilliseconds : 2362.4678
```

Figure 3: MySQL Performance Testing

5. Discussion

The findings of our research indicate that SQL databases, NoSQL databases are faster when performing read or write operations on a single entity database. The performance metrics indicate that MongoDB, a NoSQL database, generally exhibits swifter read and write operations within the scope of a single entity database. However, it would be an oversimplification to deem NoSQL databases superior across all dimensions of database management.

Relational databases, such as MySQL, excel in scenarios where data integrity, normalization, and structured schema are paramount. The relational model's strength lies in its ability to reduce redundancy and ensure consistency, especially in complex datasets where relationships between chunks of data are crucial. The atomicity, consistency, isolation, and durability (ACID) properties of relational databases cannot be understated when transactional data is involved.

On the other hand, NoSQL databases like MongoDB shine in handling unstructured data that does not fit neatly into a tabular structure. Its schema-less nature allows for greater agility in storing and retrieving varied data types, from JSON documents to key-value pairs. This makes NoSQL databases particularly adept at scaling horizontally and managing large volumes of data that may not require the stringent consistency guarantees of relational databases.

In cases where transactions involve complex joins and data normalization is critical, SQL databases may prove more performant and reliable. Conversely, for applications demanding high throughput for simple queries over large, unstructured datasets, NoSQL databases could provide the needed efficiency. Therefore, our findings encourage a pragmatic approach to database selection, one that carefully considers the unique requirements of each application.

6. Conclusion

In conclusion, our comparative study highlights the distinctive roles of SQL and NoSQL databases in data management. SQL databases excel in complex queries and transactional integrity, ideal for structured data and applications that prioritize relational data and schema stability. NoSQL databases, with their flexible schemas and horizontal scalability, are better suited for handling large volumes of diverse data, excelling in performance for single-entity operations as demonstrated by MongoDB in our tests.

The choice between SQL and NoSQL hinges on suitability rather than superiority, with each catering to specific needs. Decisions should be based on data requirements, query nature, and application scale. As data challenges grow, the emergence of hybrid databases combining SQL and NoSQL features may offer a more versatile approach, warranting further exploration in future research.

7. References

1. "5 Best Practices For Improving MongoDB Performance" n.d., *MongoDB*, accessed <https://www.mongodb.com/basics/best-practices>.

2. "How To Monitor MongoDB's Performance | DigitalOcean" 2022, *How To Monitor MongoDB's Performance | DigitalOcean*, accessed <https://www.digitalocean.com/community/tutorials/how-to-monitor-mongodb-s-performance>.

3. "MongoDB Performance" n.d., *MongoDB Performance — MongoDB Manual*, accessed <https://www.mongodb.com/docs/manual/administration/analyzing-mongodb-performance/>.

4. "MySQL :: MySQL 8.0 Reference Manual :: 8 Optimization" n.d., *MySQL :: MySQL 8.0 Reference Manual :: 8 Optimization*, accessed <https://dev.mysql.com/doc/refman/8.0/en/optimization.html>.

5. "SQL versus noSQL (speed)" 2012, *Stack Overflow*, accessed <https://stackoverflow.com/questions/13397979/sql-versus-nosql-speed>.

6. "SQL vs. MySQL: Differences, Similarities, Uses, and Benefits" n.d., *Coursera*, accessed <https://www.coursera.org/articles/sql-vs-mysql>.

7. "Tips and Best Practices for MongoDB Performance Tuning" 2023, *codedamn news*, accessed <https://codedamn.com/news/databases/mongodb-performance-tuning-tips>.

8. Anderson, B 2022, "SQL vs. NoSQL Databases: What's the Difference? - IBM Blog," *IBM Blog*, accessed <https://www.ibm.com/blog/sql-vs-nosql/>.

9. Jevtic, G 2020, "MySQL Performance Tuning: 14 Optimization Tips | phoenixNAP KB," *Knowledge Base by phoenixNAP*, accessed <https://phoenixnap.com/kb/improve-mysql-performance-tuning-optimization>.

10. Mauer, L 2022, "SQL vs NoSQL: A Performance Comparison - RestApp," *RestApp*, accessed <https://restapp.io/blog/sql-vs-nosql-a-performance-comparison/>.

11. SentinelOne 2021, "SentinelOne | SQL vs NoSQL Performance: Where One Outperforms the Other," *SentinelOne*, accessed <https://www.sentinelone.com/blog/sql-vs-nosql-performance/>.

12. Vuollet, P 2018, "Improve MySQL Performance With This Tutorial," *Stackify*, accessed <https://stackify.com/mysql-tutorial-improve-performance/>.

13. Wang, R & Yang, Z n.d., "SQL vs NoSQL: A Performance Comparison," *University of Rochester*.