MySQL vs. MongoDB: Unraveling the Performance Enigma in Database Systems

Aniruddha Raghavendra
The George Washington University
Washington D.C, U.S
aniruddha.malamandiraghavendra@gwu.edu

Swathi Murali Srinivasan
The George Washington University
Washington D.C, U.S
swathi.muralisrinivasan@gwu.edu

Ashwin Muthuraman
The George Washington University
Washington D.C, U.S
ashwin.muthuraman@gwu.edu

***Abstract*—Modern applications demand robust and efficient database management systems for optimal performance. As MySQL and MongoDB stand as prominent choices in this realm, understanding their performance nuances becomes imperative. This research undertakes a comprehensive comparison, evaluating the query performance, scalability, and ease of use of MySQL and MongoDB. Through rigorous benchmarking and analysis, we aim to unravel the performance enigma surrounding these two database systems.**

Keywords: MySQL, MongoDB, Database Management, Performance Comparison, Scalability, Query Performance.

I. Introduction

A. Background

In the dynamic landscape of web applications, effective database management plays a pivotal role in determining overall system performance and user experience. Databases serve as the backbone for data storage, retrieval, and management, influencing the responsiveness of web applications. Among the myriad of available database systems, MySQL and MongoDB stand out as popular choices, each with its unique features and structures.

MySQL, a well-established relational database management system, has been a cornerstone for web developers since its inception in 1995. Its structured query language (SQL) and ACID properties make it a robust choice for applications demanding strong data consistency. In contrast, MongoDB represents the NoSQL paradigm, designed with scalability in mind. Its flexible, document-oriented approach allows for dynamic data structures, catering to the needs of modern, high-performance applications.

B. Problem Statement

The increasing diversity of web applications necessitates careful consideration of database selection, with performance being a critical factor. Developers face the challenge of choosing between traditional relational databases like MySQL and newer NoSQL alternatives like MongoDB. This research addresses the pressing need to comprehensively compare the performance aspects of MySQL and MongoDB to empower developers and businesses with informed decision-making.

C. Objectives

This research aims to conduct a thorough performance analysis of MySQL and MongoDB, focusing on key metrics such as query speed, scalability, and ease of use. The primary objectives include:

- Evaluate the query performance of MySQL and MongoDB under various scenarios.
- Assess the scalability of both database systems.
- Analyze the ease of use and user experience aspects for developers working with MySQL and MongoDB.

By achieving these objectives, this research seeks to provide valuable insights that aid developers and businesses in selecting the most suitable database system for their specific requirements.

II. Comparative Study

TABLE I: MySQL vs. MongoDB: Performance & Scalability Comparison

Aspect	MySQL	MongoDB
Data Model	Relational (Tables) Structured	NoSQL (BSON Documents) Flexible
Query Language	SQL Standardized	JSON-based Query Language Expressive
Performance	ACID Compliance Read-Heavy Workloads	Write-Heavy Workloads Read and Write Intensive
Scalability	Vertical Scaling Limited Horizontal Scaling	Horizontal Scaling Excellent Horizontal Scaling
Use Cases	Traditional Web Apps Enterprise Solutions	Modern Web Apps Real-time Analytics

III. Queries

A. Time Series Indexing:

Time-series indexing is a key optimization method in database management for handling time-stamped data efficiently, applicable to both SQL and MongoDB. In SQL, it involves creating specialized indexes like clustered or non-clustered indexes on the timestamp column or using composite indexes. In MongoDB, B-tree structures are utilized, with an ascending index on the timestamp field being common. This strategy significantly improves query performance, reduces resource consumption, and supports advanced analytical tasks. Integrating time-series indexing into database design highlights its crucial role in optimizing temporal data retrieval across diverse database systems.

B. *Data Compression:*

Data compression is a fundamental concept in both SQL and MongoDB, aimed at reducing the storage footprint of datasets to optimize resource utilization and enhance performance. In SQL databases, compression techniques involve minimizing redundant data through methods like page-level compression or column store compression, thereby conserving storage space and accelerating query execution. In MongoDB, data compression is achieved through WiredTiger, the default storage engine, which implements a combination of compression algorithms to efficiently store data on disk. The use of compression in both database systems contributes to improved data retrieval speeds, reduced storage costs, and overall enhanced efficiency in managing and processing large volumes of information.

C. *Text search:*

Text search with indexing is a fundamental concept in both SQL and MongoDB databases, enhancing the efficiency of search operations within large datasets. In SQL, full-text indexing allows for the rapid retrieval of information from text columns by creating a searchable index, significantly improving query performance. This indexing process involves parsing and storing keywords in a structured format, enabling faster search operations. Similarly, MongoDB employs text indexing to facilitate text search functionality. With MongoDB's text indexes, users can efficiently search for specific terms or phrases within documents. The indexing process in MongoDB involves tokenizing and stemming words, creating an index that accelerates search queries. Incorporating text search with indexing in both SQL and MongoDB not only enhances search speed but also contributes to a more streamlined and effective data retrieval process in diverse applications.

D. *Sub-queries:*

Sub-queries in MySQL are nested queries used within larger queries, enabling operations like fetching data based on results of inner queries. In MongoDB, sub-queries are implemented through the aggregation pipeline, especially with the 'lookup' stage, allowing for data retrieval and aggregation across collections, akin to join operations.

E. *Windows function vs Aggregation pipeline:*

In the context of time series data analysis, both SQL and MongoDB employ distinct approaches to achieve similar outcomes. SQL utilizes window functions, which are specialized functions that operate within a specified range or "window" of rows in the result set. These functions, such as LAG, LEAD, and ROW-NUMBER, facilitate computations over a defined temporal context, enabling efficient time-based analysis. On the other hand, MongoDB employs the Aggregation Pipeline, a framework that allows the transformation of documents through a sequence of processing stages. By leveraging stages like 'match', 'group', and 'project', MongoDB achieves similar time series analyses as SQL's window functions. Despite the differences in syntax and methodology, both SQL's window functions and MongoDB's Aggregation Pipeline serve as powerful tools for temporal data manipulation, showcasing the adaptability of these database systems in handling time- oriented queries.

F. *Pagination:*

Pagination in MySQL involves using the LIMIT and OFF- SET clauses within queries to break down large datasets into smaller pages. The LIMIT keyword specifies the number of records to retrieve, while OFFSET determines the starting point within the dataset. On the other hand, in MongoDB, pagination relies on the skip() and limit() methods. limit() sets the number of records to return, and skip() determines the starting point for fetching records.

G. *Materialized Views:*

In MySQL, materialized views are created using scheduled events or triggers to update tables representing the view's data. In MongoDB lacks its achieved using the aggregation framework or separate collections to store precomputed or aggregated data, necessitating manual updates to maintain the 'materialized' representation.

IV. Results

A. *Clustered Column Chart*

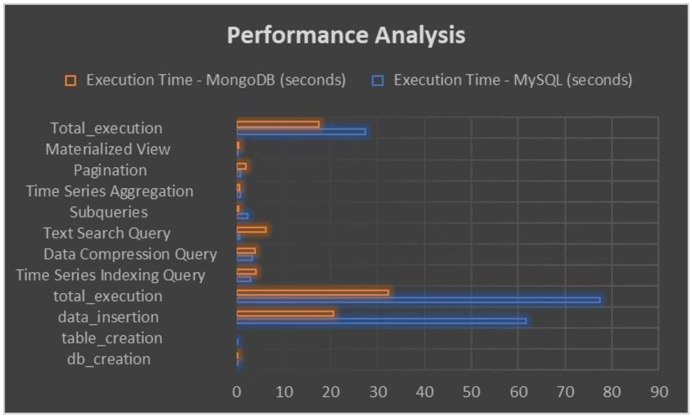
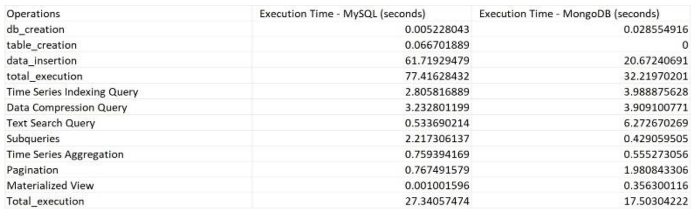


Fig 1: Final results of performance analysis

B. *Execution time*



Operations	Execution Time - MySQL (seconds)	Execution Time - MongoDB (seconds)
db_creation	0.005228043	0.028554916
table_creation	0.066701889	0
data_insertion	61.71929479	20.67240691
total_execution	77.41628432	32.21970201
Time Series Indexing Query	2.805816889	3.988875628
Data Compression Query	3.232801199	3.909100771
Text Search Query	0.533690214	6.272670269
Subqueries	2.217306137	0.429059505
Time Series Aggregation	0.759394169	0.555273056
Pagination	0.767491579	1.980843306
Materialized View	0.001001596	0.356300116
Total_execution	27.34057474	17.50304222

Fig 2: Execution times of operations

V. CONCLUSION

Following an analysis of the COVID dataset utilizing MySQL and MongoDB, it became clear that MongoDB consistently excelled over MySQL in database and table creation as well as insertion queries, displaying notably quicker execution times. Nevertheless, MySQL showcased superior query processing compared to MongoDB, specifically in tasks involving text search, sub-queries, aggregation, pagination, materialized views, and indexing. These observations underscore MySQL's potential as a more efficient option for handling these specific operations. Looking forward, for upcoming analyses entailing data pre-processing, mining, and modeling, a structured database appears to hold an advantage over a NoSQL database due to its suitability in managing such complex operations.

References

1) "Comparison of MySQL and MongoDB with focus on performance," 2020.
2) "Relational Database and NoSQL Inspections using MongoDB and Neo4j on a Big Data Application," 2022.
3) "NoSQL Database — RavenDB ACID NoSQL Document Database," 2019. https://ravendb.net/
4) "Performance Analysis of Neo4j and MySQL Databases using Public Policies Decision Making Data," 2020.
5) I. MongoDB, "Mongodb architecture guide: Overview," 08 2019.
6) M. Diogo, B. Cabral, and J. Bernardino, "Consistency models of nosql databases," Future Internet, vol. 11, p. 43, 02 2019.
7) Patel, S.; Kumar, S.; Katiyar, S.; Shanmugam, R.; Chaudhary, R. MongoDB Versus MySQL: A Comparative Study of Two Python Login Systems Based on Data Fetching Time. In Research in Intelligent and Computing in Engineering; Springer: Singapore, 2021;