Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Comparative Performance Evaluation of SQL and NoSQL Databases Under Different Indexing and Cache Conditions

Amir Khan

amirkhan421.official@gmail.com

Thal University Bhakkar

Sawaira Gull

Government College University, Faisalabad

Arooj Shahzadi

Government College University, Faisalabad

Research Article

Keywords: Database performance, SQL, NoSQL, query optimization, indexing, MySQL, MongoDB

Posted Date: November 13th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11275727/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

A Comparative Performance Evaluation of SQL and NoSQL Databases Under Different Indexing and Cache Conditions

Amir Khan [1] , Sawaira Gull [2] , Arooj Shahzadi [3]

[1] Department of Computer Science & IT, Thal University Bhakkar, Pakistan

[2,3] Department of Computer Science, Government College University Faisalabad, Pakistan

Corresponding Author: Amir Khan (amirkhan421.official@gmail.com)

Abstract

This study presents a comparative benchmark of MySQL 8.0.46 and MongoDB 8.0.32 using bulk-write, point-lookup, and filtered-aggregation workloads. Experiments were conducted on a single Ubuntu-based computer using synthetic transaction datasets containing 10K, 100K, and 1M records. Performance was measured under unindexed, indexed, and warm-cache conditions, with five measured iterations per configuration. MongoDB recorded lower bulk-write latency at all dataset sizes. Single-field indexing substantially reduced point-lookup latency in both database systems, with reductions exceeding 99% at one million records. Aggregation results varied according to dataset size and configuration. At one million records, MySQL's indexed aggregation latency was higher than its unindexed latency; an independent remeasurement reproduced this result under the tested conditions. These findings demonstrate that database performance depends on workload, dataset size, indexing, and execution state. The study does not establish a universal advantage for either database model. Instead, it provides a controlled, single-host comparison and identifies directions for further investigation, including composite indexes, concurrent workloads, and distributed deployments.

Keywords: Database performance, SQL, NoSQL, query optimization, indexing, MySQL, MongoDB

1 Introduction

The increasing volume of data generated by enterprise applications, web services, financial systems, and other data-intensive applications has made database performance an important consideration in system design. Database management systems must support efficient data storage and retrieval while maintaining predictable performance as workloads and dataset sizes increase. The choice between relational SQL databases and NoSQL databases is therefore influenced not only by data modeling requirements but also by workload characteristics, query patterns, indexing strategies, and scalability requirements. [1] [2] [3] [4]

Relational database management systems (RDBMSs) such as MySQL organize data using predefined schemas, tables, and structured query languages. They provide mature mechanisms for transaction management, indexing, aggregation, and data integrity. In contrast, document-oriented NoSQL systems such as MongoDB store records as flexible BSON documents and provide a different data model and query-processing architecture. These architectural differences can result in different performance characteristics for write-intensive operations, point queries, and analytical workloads. [5] [1] [2] [4]

Query optimization is particularly important when the volume of stored data increases. An operation that performs adequately on a small dataset may exhibit substantially different latency when executed against hundreds of thousands or millions of records. Indexes are among the most widely used mechanisms for reducing query search costs, but their effect may depend on the query structure, indexed attributes, database engine, and workload.[6] [4] Consequently, evaluating indexing only through a single query type or dataset size may not provide a complete picture of database performance.

Previous database benchmarking studies have examined SQL and NoSQL systems under different workloads and deployment environments. However, performance measurements can vary considerably according to hardware configuration, software versions, dataset characteristics, query design, indexing strategy, caching state, and measurement methodology.[1] [7] [2] [3] [4] A controlled experiment using identical datasets and clearly defined workload conditions can therefore provide useful empirical evidence about the behavior of different database systems under comparable conditions.

Although previous studies have compared MySQL and MongoDB using CRUD workloads, cloud environments, large datasets, concurrent clients, and indexing or optimization techniques, these dimensions are often investigated separately or under substantially different experimental conditions [1]-[8]**.** For example, previous work has examined cloud-based and concurrent workloads, large-scale CRUD operations, and index-oriented optimization, while other studies have focused on document-oriented data models or distributed database architectures. A controlled single-host experiment that combines multiple dataset scales with bulk writes, equality lookups, filtered aggregation, single-field indexing, and a defined warm-cache condition provides a complementary perspective on latency behavior under comparable execution conditions.

This study presents a controlled empirical comparison of MySQL 8.0.46 and MongoDB 8.0.32 using transaction-oriented synthetic datasets containing 10,000, 100,000, and 1,000,000 records. The experiment evaluates three workload categories: bulk data ingestion, equality lookups, and filtered aggregation queries. Read workloads are measured in unindexed, indexed, and warm-cache configurations, while performance is recorded using repeated high-resolution timing measurements. The same generated datasets and equivalent logical operations are used for both database systems.

The study addresses the following research questions:

> **RQ1:** How do MySQL and MongoDB differ in bulk-write latency as dataset size increases from 10K to 1M records?
>
> **RQ2:** How does single-field indexing affect equality-lookup and aggregation latency in MySQL and MongoDB?
>
> **RQ3:** How does query latency scale across unindexed, indexed, and warm-cache configurations as dataset size increases?

The main contributions of this study are as follows:

- **Controlled empirical benchmark:** A reproducible Python-based benchmarking framework is used to compare MySQL and MongoDB under identical synthetic transaction datasets and defined workload conditions.
- **Multi-scale performance evaluation:** Database performance is evaluated at three dataset scales—10K, 100K, and 1M records—to examine how latency changes as the amount of stored data increases.
- **Indexing and cache analysis:** The study measures equality-lookup and aggregation performance under unindexed, indexed, and warm-cache conditions, allowing the effects of these configurations to be examined separately.
- **Independent remeasurement of an anomalous observation:** The higher measured latency of the MySQL indexed aggregation at the 1M-record scale is independently remeasured without modifying the original benchmark dataset, providing additional evidence that the observation was reproducible under the tested configuration.

The remainder of this paper is organized as follows. Section II reviews related research on SQL and NoSQL database performance, indexing, and benchmarking. Section III describes the experimental environment, datasets, database configurations, workloads, and measurement methodology. Section IV presents and analyzes the experimental results. Section V discusses the observed performance differences and their implications. Section VI describes threats to validity and limitations. Section VII concludes the study and identifies directions for future work.

2 Related Work

The performance characteristics of relational and NoSQL database systems have been investigated in a variety of application and workload settings. Existing studies generally compare database systems using metrics such as insertion time, query response time, retrieval performance, and scalability. However, reported performance differences depend strongly on the selected workload, dataset, hardware environment, database configuration, and optimization techniques.

2.1 Direct MySQL–MongoDB Comparisons

Several studies have directly compared MySQL and MongoDB under different experimental conditions. These studies have examined database performance using transaction processing, CRUD operations, read/write workloads, and different dataset sizes [9][2][10][4]. Their findings indicate that the relative performance of MySQL and MongoDB can vary according to workload characteristics, data scale, and execution environment. These studies provide a basis for direct SQL–NoSQL comparison, while the present work focuses specifically on bulk writes, point lookups, and filtered aggregation under controlled indexing and warm-cache conditions.

2.2 Broader SQL/NoSQL Performance Studies

Broader research has also examined SQL and NoSQL database performance across different workloads and application requirements. [5] evaluates database performance under specific workload conditions, illustrating that performance differences cannot be attributed solely to the underlying SQL or NoSQL classification. A systematic review presented in [1] synthesized findings from a broad body of SQL

and NoSQL performance studies and highlighted the importance of workload, experimental configuration, and evaluation methodology when interpreting comparative results.

2.3 Indexing and Query Optimization

Indexing and query optimization represent another important area of database performance research. Index structures can reduce the amount of data examined during query execution, but their effectiveness depends on query structure, data distribution, and database configuration. Research on indexing strategies has demonstrated that different index configurations can produce different retrieval behavior under different query conditions [6]. Other comparative research has also considered optimization and indexing-related effects when evaluating MySQL and MongoDB [4]. These findings motivate the present study's separation of unindexed and indexed conditions rather than treating indexing as a fixed database property.

2.4 Cloud, Concurrent, and Distributed Benchmarks

Database benchmarking has also been conducted under concurrent, cloud-based, distributed, and larger-scale environments. Previous research has evaluated database systems under concurrent workload conditions [7], while other studies have investigated database performance in distributed and cloud-oriented environments [8][2]. Large-scale database evaluations have also examined CRUD and read/write performance as dataset size increases [3]. These studies demonstrate that concurrency, deployment architecture, and dataset scale can influence measured database performance. In contrast, the present study deliberately uses sequential execution on a single physical host to provide a controlled comparison of read and write latency.

Taken together, previous studies demonstrate that SQL and NoSQL performance is strongly dependent on workload, indexing, dataset size, concurrency, and experimental configuration [1]-[8]. Direct MySQL–MongoDB comparisons provide evidence of workload-dependent performance differences, while indexing, concurrent execution, cloud deployment, and larger datasets introduce additional factors that can influence measured latency. The present study builds on this literature by evaluating MySQL and MongoDB using the same logical transaction data, the same physical host, three dataset sizes, and explicitly separated unindexed, indexed, and warm-cache conditions.

Despite this broad body of work, the experimental conditions used across studies vary considerably in hardware, software versions, data models, workload definitions, concurrency, and indexing configurations. The present study therefore provides a complementary controlled benchmark using the same physical host, identical logical transaction data, three dataset sizes, and clearly separated unindexed, indexed, and warm-cache read conditions. Unlike studies focused only on CRUD operations or concurrent throughput, the present benchmark specifically examines bulk writes, user-ID equality lookups, and filtered aggregation on datasets ranging from 10K to 1M records.

The present work also differs from a simple SQL-versus-NoSQL comparison by independently validating an unexpected aggregation result observed during the benchmark. This additional validation helps distinguish a potentially anomalous measurement from a reproducible observation under the specific experimental configuration. The resulting analysis provides empirical evidence about the performance behavior of the two systems while explicitly limiting conclusions to the tested environment and workload conditions.

3 Experimental Methodology

3.1 Experimental Environment

All experiments were conducted on a single local host to provide a consistent execution environment and to avoid variability caused by network communication between the benchmark client and database servers. The host ran Ubuntu 24.04.4 LTS and contained an Intel Core i5-5200U processor with four logical CPU cores, 8 GB of RAM, and a 256 GB SSD. The benchmark software was executed using Python 3.12.3 inside an isolated Python virtual environment.

The relational database system was MySQL Server 8.0.46 using the InnoDB storage engine. The NoSQL database system was MongoDB Community Server 8.0.32 using its WiredTiger storage engine. Python communicated with MySQL through PyMySQL 1.2.3 and with MongoDB through PyMongo 4.18.1. The benchmark used Python's time.perf_counter_ns() for high-resolution elapsed-time measurement.

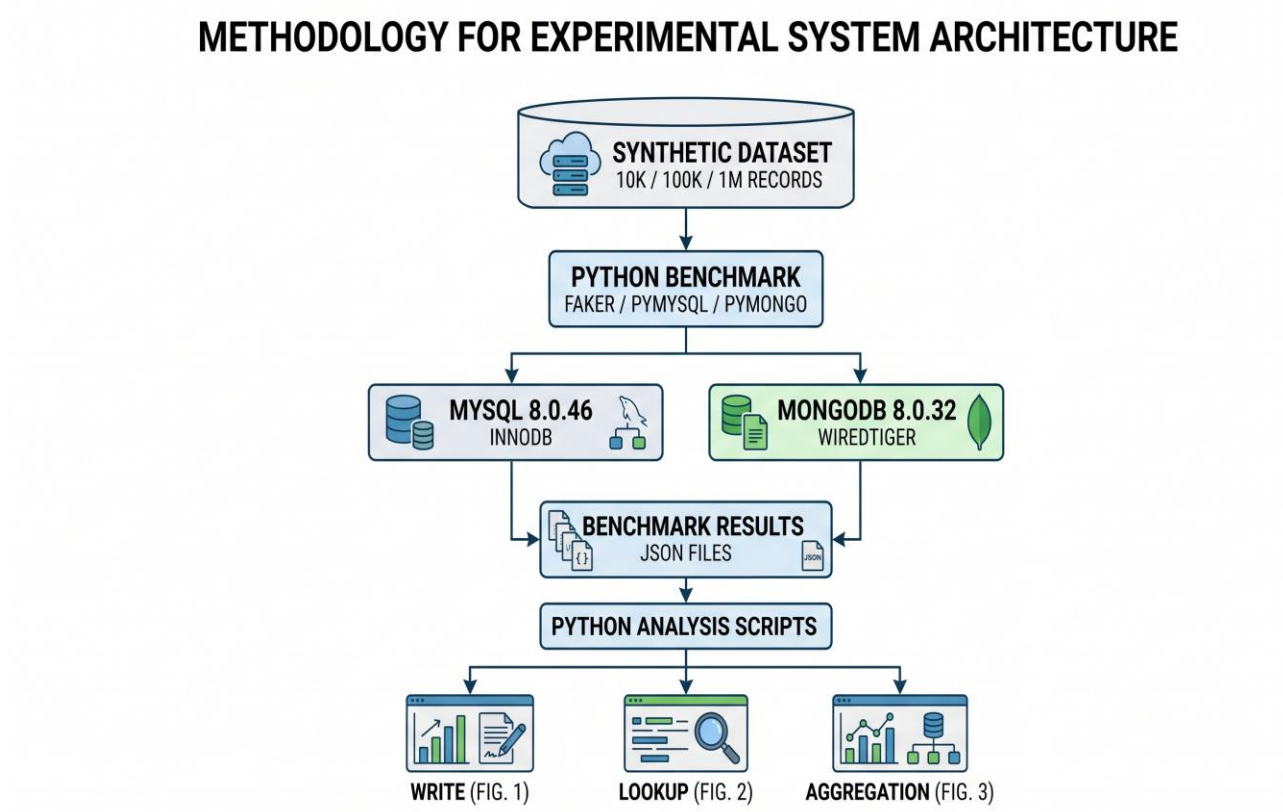


Fig 1. Experimental benchmark architecture and data-analysis workflow.

Table 1. Experimental Environment

Component	Configuration
Operating system	Ubuntu 24.04.4 LTS
Processor	Intel Core i5-5200U, 4 logical cores
Memory	8 GB DDR3
Storage	256 GB SSD
Python	3.12.3
Relational DBMS	MySQL Server 8.0.46
MySQL storage engine	InnoDB
NoSQL DBMS	MongoDB Community Server 8.0.32
MongoDB storage engine	WiredTiger
MySQL driver	PyMySQL 1.2.3
MongoDB driver	PyMongo 4.18.1
Timing method	time.perf_counter_ns()

3.2 Dataset Generation

A synthetic transaction dataset was generated using the Python Faker library. A fixed random seed of 42 was used so that the same logical dataset could be reproduced across benchmark executions. Three dataset sizes were generated to represent increasing data volumes: 10,000 records (10K), 100,000 records (100K), and 1,000,000 records (1M).

Each record contained six fields: transaction_id, user_id, amount, category, status, and created_at. The transaction identifier was generated by the MySQL database through an auto-incrementing primary key, while the corresponding MongoDB documents used equivalent transaction identifiers. The generated source data contained the common workload attributes user_id, amount, category, status, and created_at.

The transaction amount was represented as DECIMAL(10,2) in MySQL and BSON Decimal128 in MongoDB to maintain comparable decimal representation. The timestamp was stored as MySQL DATETIME and MongoDB BSON datetime, respectively.

3.3 Database Schema and Configuration

The MySQL benchmark used a transactions table with the following logical structure:

```
transaction_id  INT, AUTO_INCREMENT, PRIMARY KEY
user_id         INT
amount          DECIMAL(10,2)
category        VARCHAR(50)
status          VARCHAR(20)
created_at      DATETIME
```

The corresponding MongoDB collection was named transactions and stored equivalent transaction information as BSON documents.

Two secondary indexes were created for the indexed benchmark configuration:

MySQL:
CREATE INDEX idx_user_id ON transactions(user_id);
CREATE INDEX idx_category ON transactions(category);

MongoDB:
index on user_id
index on category

The benchmark therefore evaluates **single-field indexing,** rather than composite indexing. The original benchmark did not create a composite (category, status) index.

Indexes were created only after the unindexed measurements had been completed. This procedure prevented the indexed configuration from affecting the preceding unindexed benchmark state.

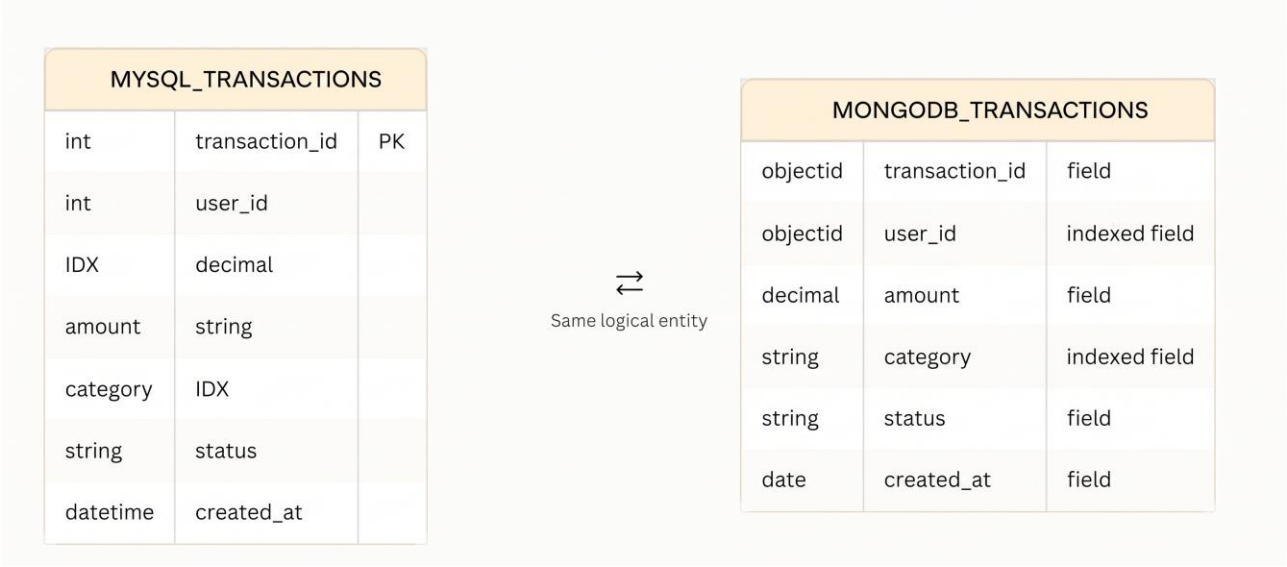


Fig 2: Logical representation of the benchmark data model in MySQL and MongoDB.

3.4. Benchmark Workloads

Three workload categories were evaluated.

3.4.1. Bulk Write Workload

Bulk ingestion measured the elapsed time required to insert the complete dataset into each database. The benchmark used batches of 5,000 records for the insertion process. The generated CSV data were loaded and normalized before timing so that data preparation and conversion overhead were not included in the database write measurement.

The reported write latency represents the total elapsed database-ingestion time in milliseconds.

3.4.2. Equality lookup Workload

The equality-lookup workload measured retrieval of records matching a selected user_id. The logical SQL operation was equivalent to:

```
SELECT *
FROM transactions
WHERE user_id = ?;
```

The MongoDB operation was equivalent to:

```
find({"user_id": value})
```

The benchmark measured the complete query operation, including result retrieval/materialization rather than measuring only the database command submission time.

Equality lookups were evaluated in both unindexed and indexed configurations and were additionally measured under a warm-cache condition.

3.4.3. Aggregation Workload

The aggregation workload evaluated a filtered grouping operation. The MySQL query was:

```
SELECT
   status,
   SUM(amount)
FROM transactions
WHERE category = ?
GROUP BY status;
```

The corresponding MongoDB operation applied a category filter followed by grouping by status and summing the transaction amount.

The benchmark used Electronics as the category value for the aggregation workload.

Unlike the equality lookup, the aggregation benchmark was affected by both the category filtering operation and the subsequent grouping operation. Therefore, the presence of a category index does not imply that the complete aggregation can be satisfied directly from the index.

3.5. Benchmark Configurations

Workload	Unindexed	Indexed	Warm-cache
Bulk write	✓	—	—
Equality lookup	✓	✓	✓
Aggregation	✓	✓	✓

The indexed configuration used single-field indexes on user_id and category. The warm-cache configuration was measured with the indexes present and after one unmeasured execution of the corresponding read operations.

Read workloads were evaluated under three conditions.

Unindexed configuration: The relevant secondary indexes were absent during measurement. This represents the baseline state without the tested secondary indexes.

Indexed configuration: Single-field indexes on user_id and category were present. These indexes were created after the unindexed measurements.

Warm-cache configuration: With the single-field indexes already present, one unmeasured point lookup and one unmeasured aggregation query were executed before the five measured iterations to establish a warmed database/memory state. This condition should not be interpreted as the use of MySQL Query Cache.

The warm-cache condition was applied to read workloads only.

3.6. Measurement Procedure

Each benchmark condition was executed for five measured iterations. Query latency was measured using Python's time.perf_counter_ns() and converted to milliseconds. The arithmetic mean of the five measured observations was reported as the primary latency value.

For each dataset size, the benchmark therefore collected measurements for MySQL and MongoDB across the defined workload and configuration combinations.The benchmark framework stored the resulting mean latency values in a structured JSON file. The

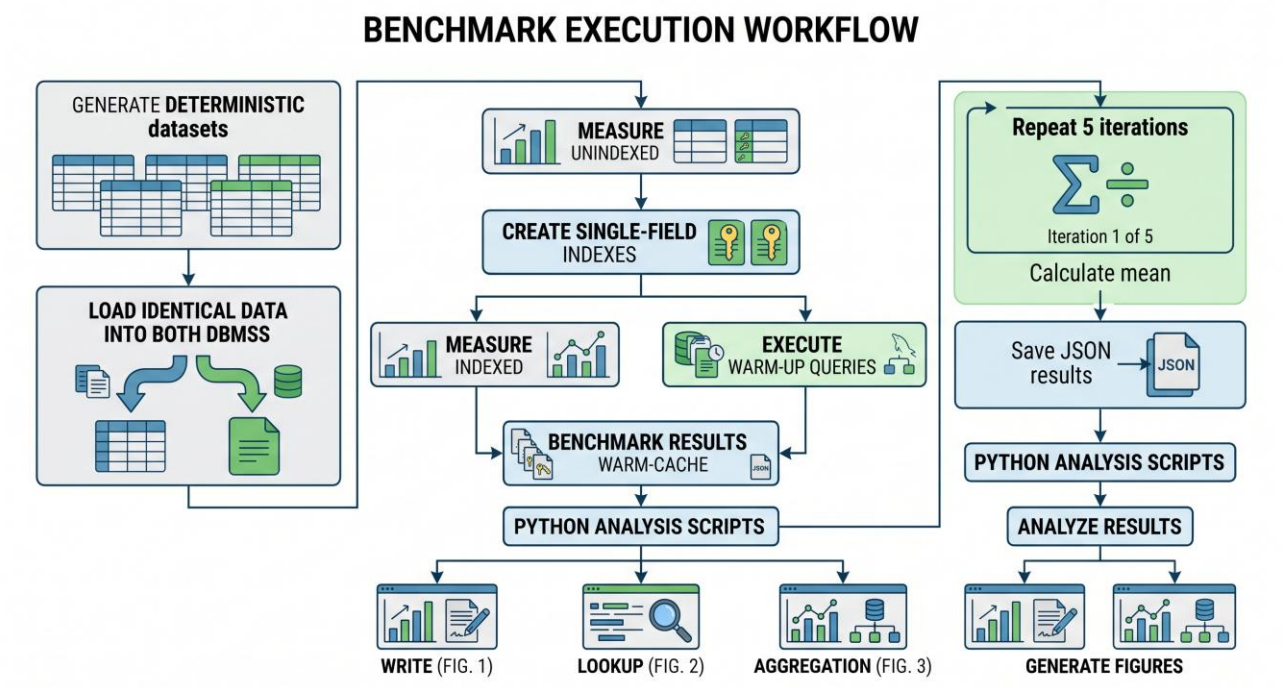


Fig 3: Benchmark execution and measurement workflow.

finalized benchmark results were subsequently analyzed using Python scripts during the data-analysis phase. Three publication-quality figures were generated to visualize bulk-write, equality-lookup, and aggregation performance.

3.7. Validation Procedure

During analysis, the 1M-record MySQL aggregation result showed higher measured latency in the indexed configuration than in the corresponding unindexed configuration. Because this observation differed from the substantial improvement observed for equality lookups, an independent remeasurement was performed without modifying the original benchmark dataset.

The validation repeated five indexed aggregation measurements, removed the idx_category index, repeated five unindexed measurements, and then recreated the index. The database state was restored after validation.

The validation produced an average indexed latency of **764.391 ms** and an average unindexed latency of **614.816 ms**, reproducing the direction of the original observation. The original benchmark had measured **1,069.269 ms** for the indexed configuration and **704.294 ms** for the unindexed configuration.

This validation was used to strengthen confidence that the original observation was reproducible under the tested configuration. It does not establish that indexing generally reduces or increases aggregation performance across MySQL workloads.

3.8. Reproducibility and Data Integrity

The experiment was designed to maintain consistent data and configuration across the two database systems. The synthetic source datasets were generated deterministically using a fixed seed, and the same logical transaction records were used for both database systems. Dataset sizes and benchmark configurations were kept constant across comparisons.

The finalized Phase 2 benchmark results were independently checked during Phase 3 to confirm that the numerical values used for analysis matched the stored benchmark results. The final MySQL and MongoDB database states were also verified to contain 1,000,000 records after the completion of the benchmark procedures.

4 Results And Analysis

The benchmark results are presented for three dataset sizes: 10K, 100K, and 1M records. Performance was evaluated using bulk-write latency, equality-lookup latency, and aggregation latency. For read workloads, unindexed, indexed, and warm-cache configurations were measured. All reported benchmark values represent the arithmetic mean of five measured iterations.

4.1. Bulk Write Performance

Table 2. presents the measured bulk-write latency for MySQL and MongoDB.

Table 2. Bulk Write Latency

Dataset size	MySQL (ms)	MongoDB (ms)	MongoDB reduction
10K	832.416	429.521	48.40%
100K	3,239.424	1,082.221	66.59%
1M	29,617.741	12,547.836	57.63%

MongoDB recorded lower measured write latency at all three dataset sizes. At 10K records, MongoDB required 429.521 ms compared with 832.416 ms for MySQL. At 100K records, the measured latencies increased to 1,082.221 ms and 3,239.424 ms, respectively. At the 1M-record scale, MongoDB required 12,547.836 ms, whereas MySQL required 29,617.741 ms.

The relative difference corresponds to approximately 1.94×, 2.99×, and 2.36× lower elapsed write time for MongoDB at 10K, 100K, and 1M records, respectively. These measurements indicate a consistent write-latency advantage for MongoDB under the specific single-host, batch-insertion configuration used in this experiment. The measurements alone do not establish the architectural cause of this difference.

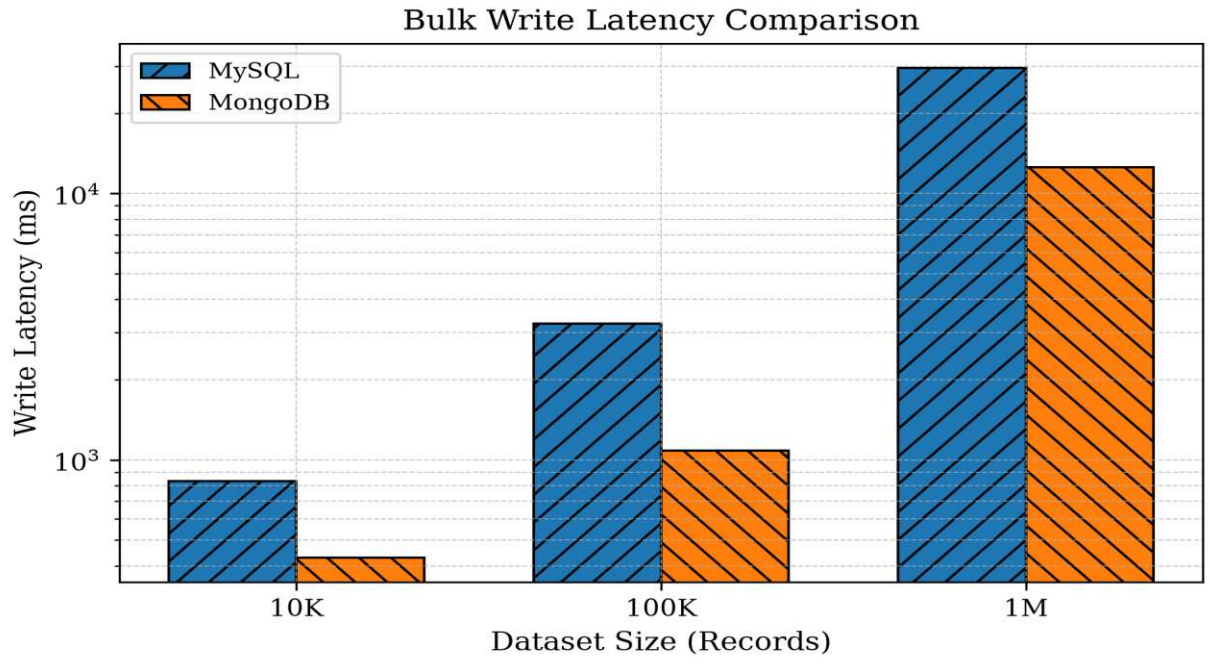


Fig 4. Bulk-write latency across dataset sizes.

4.2. Equality lookup Performance

Equality-lookup latency was measured before indexing, after creating single-field indexes on user_id, and under the warm-cache configuration.

Table 3. Equality Lookup Latency

Dataset	Configuration	MySQL (ms)	MongoDB (ms)
10K	Unindexed	9.962	11.806
10K	Indexed	0.891	1.574
10K	Warm-cache	0.671	0.812
100K	Unindexed	55.262	48.104
100K	Indexed	0.725	1.119
100K	Warm-cache	0.738	0.908
1M	Unindexed	540.517	518.589
1M	Indexed	1.192	1.984
1M	Warm-cache	1.137	1.214

The unindexed equality lookup exhibited a substantial increase in latency as dataset size increased. For MySQL, latency increased from 9.962 ms at 10K records to 540.517 ms at 1M records. MongoDB showed a similar increase from 11.806 ms to 518.589 ms.

The introduction of the user_id index substantially reduced measured equality-lookup latency in both systems. At 1M records, MySQL decreased from 540.517 ms unindexed to 1.192 ms indexed, corresponding to a 99.78% reduction. MongoDB decreased from 518.589 ms to 1.984 ms, corresponding to a 99.62% reduction.

The same pattern was observed at smaller scales. For MySQL, indexing reduced latency by 91.06% at 10K records and 98.69% at 100K records. For MongoDB, the corresponding reductions were 86.67% and 97.67%.

The warm-cache measurements were also substantially lower than the corresponding unindexed measurements. At 1M records, the warm-cache latency was 1.137 ms for MySQL and 1.214 ms for MongoDB. These results indicate that the tested equality-lookup workload benefited strongly from the indexed and warmed configurations.

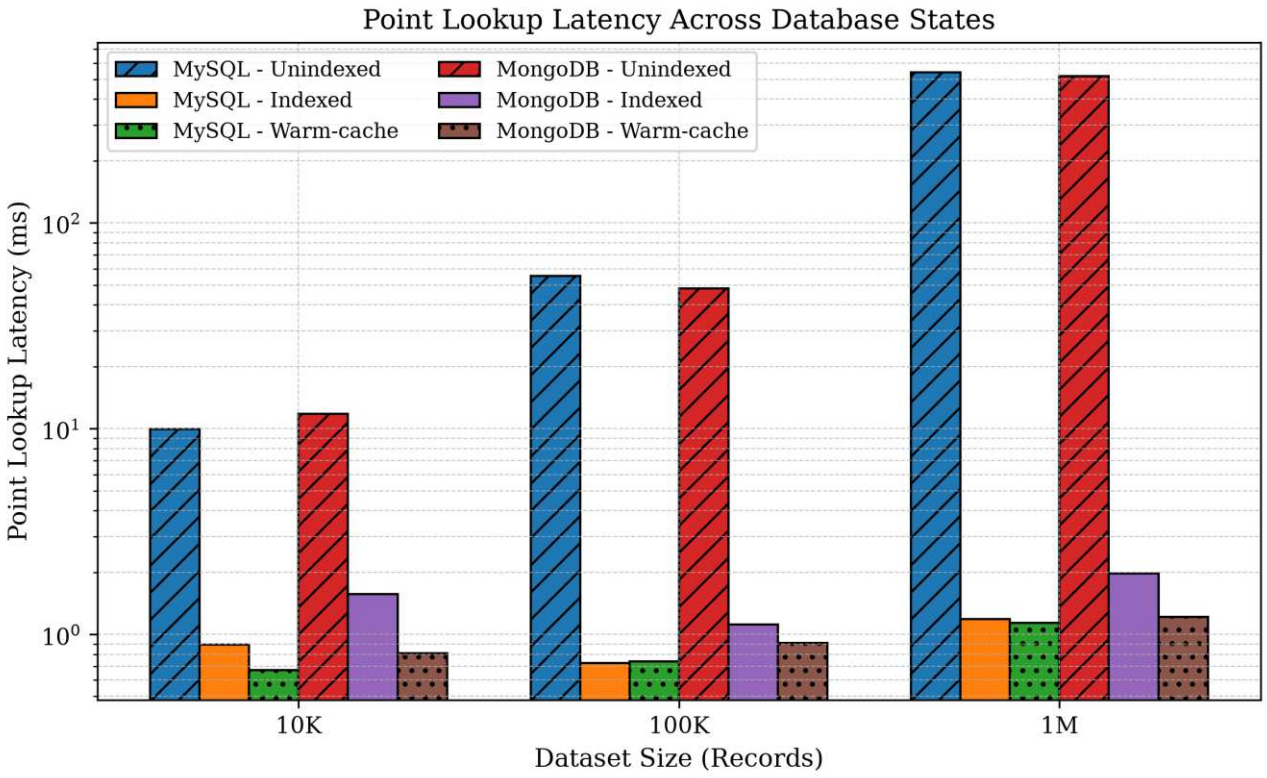


Fig 5: Point-lookup latency under unindexed, indexed, and warm-cache configurations.

4.3. Aggregation Performance

The aggregation workload filtered records by category = 'Electronics', grouped the matching records by status, and calculated the sum of amount. Table 4. presents the measured results.

Table 4. Aggregation Latency

Dataset	Configuration	MySQL (ms)	MongoDB (ms)
10K	Unindexed	9.903	20.231
10K	Indexed	7.468	6.533
10K	Warm-cache	5.777	4.915
100K	Unindexed	59.744	50.668
100K	Indexed	58.983	30.494
100K	Warm-cache	46.334	32.408
1M	Unindexed	704.294	590.154
1M	Indexed	1,069.269	304.607
1M	Warm-cache	733.467	304.103

Aggregation behavior differed from the equality-lookup results. At 10K records, the unindexed MySQL aggregation required 9.903 ms compared with 20.231 ms for MongoDB. After indexing, the

corresponding measurements were 7.468 ms and 6.533 ms. Under the warm-cache condition, MySQL measured 5.777 ms while MongoDB measured 4.915 ms.

At 100K records, MongoDB measured lower latency in all three aggregation configurations. The unindexed measurements were 59.744 ms for MySQL and 50.668 ms for MongoDB. With indexing, the measurements were 58.983 ms and 30.494 ms, respectively. Under warm-cache conditions, MySQL measured 46.334 ms while MongoDB measured 32.408 ms.

At 1M records, the difference between the systems became more pronounced. The unindexed aggregation measured 704.294 ms for MySQL and 590.154 ms for MongoDB. In the indexed configuration, MySQL measured 1,069.269 ms, whereas MongoDB measured 304.607 ms. The warm-cache measurements were 733.467 ms for MySQL and 304.103 ms for MongoDB.

An important observation is that the MySQL indexed aggregation at 1M records had higher measured latency than its corresponding unindexed measurement. The indexed value was 1,069.269 ms compared with 704.294 ms unindexed. This represents a 51.82% increase relative to the unindexed measurement in this particular experimental configuration.

The result should not be interpreted as evidence that indexes generally make MySQL aggregation slower. The benchmark used a single-field category index for a query that also performed grouping by status. An examination of the MySQL execution plan showed that the idx_category index was selected, while the plan also reported Using temporary. Thus, the observed result represents the behavior of this particular query, index structure, dataset, and database state.

The observation was independently validated after the original benchmark. The validation produced an average indexed latency of 764.391 ms and an average unindexed latency of 614.816 ms, again showing higher measured latency for the indexed configuration. The index was restored after the validation.

MongoDB showed a different aggregation pattern at the 1M scale. Its indexed latency of 304.607 ms was 48.39% lower than its unindexed latency of 590.154 ms. The warm-cache value of 304.103 ms was also substantially lower than the unindexed measurement.

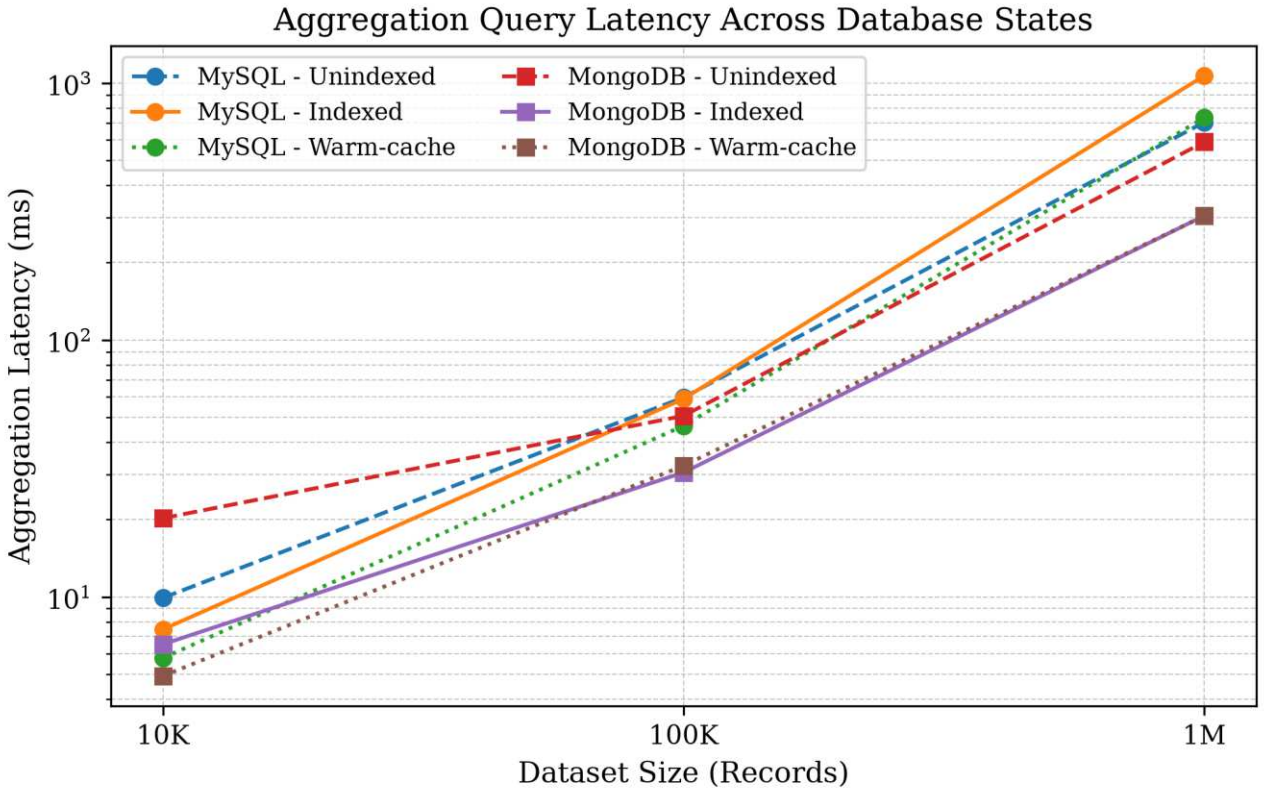


Fig 6. Aggregation latency under unindexed, indexed, and warm-cache configurations.

4.4. Scaling Behavior

The measured results demonstrate that the effect of increasing dataset size depended strongly on the workload and configuration.

For unindexed equality lookups, increasing the dataset from 10K to 1M records increased MySQL latency from 9.962 ms to 540.517 ms, representing a 5,325.79% increase. MongoDB increased from 11.806 ms to 518.589 ms, representing a 4,292.59% increase.

In contrast, indexed equality lookups showed much smaller absolute growth. MySQL increased from 0.891 ms at 10K to 1.192 ms at 1M, a 33.78% increase. MongoDB increased from 1.574 ms to 1.984 ms, a 26.05% increase.

Aggregation latency showed substantially stronger scaling effects. MySQL's unindexed aggregation increased from 9.903 ms at 10K to 704.294 ms at 1M, while MongoDB increased from 20.231 ms to 590.154 ms. The indexed aggregation showed different behavior: MySQL increased from 7.468 ms to 1,069.269 ms, while MongoDB increased from 6.533 ms to 304.607 ms.

These observations indicate that dataset growth does not affect all query types equally. equality lookups benefited strongly from the tested indexes, whereas aggregation performance remained more sensitive to the amount of data processed and the structure of the query.

4.5. Comparative Observations

The benchmark provides several direct observations across the tested workload categories.

First, MongoDB had lower measured bulk-write latency at all three dataset sizes. The largest relative difference occurred at 100K records, where MongoDB's measured write latency was 66.59% lower than MySQL's.

Second, equality lookups benefited substantially from the user_id index in both database systems. At the 1M-record scale, the reduction exceeded 99% for both systems relative to their corresponding unindexed measurements.

Third, aggregation behavior was more dependent on database and configuration. MongoDB produced lower indexed aggregation latency than MySQL at all three dataset sizes, while MySQL was lower than MongoDB for the 10K unindexed aggregation. At larger scales, MongoDB maintained lower measured aggregation latency in the tested configurations.

Fourth, the warm-cache condition generally reduced read latency relative to the corresponding unindexed condition, but the magnitude of the reduction varied by workload and database. The warm-cache measurements therefore provide additional evidence that execution state can influence observed latency.

Finally, the benchmark does not identify a universally superior database system. Instead, the results show that relative performance changed according to the operation being measured, dataset size, indexing configuration, and cache condition. These observations directly address the research questions by demonstrating that database selection and query optimization should be considered in relation to the workload rather than evaluated using a single performance metric.

5 Discussion

The experimental results demonstrate that query and write performance differed between MySQL and MongoDB across dataset sizes and workload types. However, the results do not indicate a universal performance advantage for either database system. Instead, the measured latency depended on the operation, dataset size, indexing configuration, and cache condition.

5.1. Bulk Write Performance

MongoDB recorded lower bulk-write latency than MySQL for all three dataset sizes. At 10K records, MongoDB required 429.521 ms compared with 832.416 ms for MySQL, representing approximately 48.40% lower measured latency. At 100K records, the difference increased to 66.59%, while at 1M records MongoDB recorded 12,547.836 ms compared with 29,617.74 1 ms for MySQL, corresponding to approximately 57.63% lower latency.

These results show a consistent write-latency advantage for MongoDB under the batch-insertion workload used in this study. Similar workload-dependent MongoDB write advantages have been reported in previous MySQL–MongoDB comparisons, although the magnitude of the difference varies substantially with data model, hardware, workload design, and execution environment [9][7].

However, the experiment was conducted on a single host using a synthetic dataset and a fixed batch size of 5,000 records. Therefore, the measurements should be interpreted as observations under the

tested configuration rather than as evidence that MongoDB generally provides superior write performance for all workloads.

5.2. Effect of Indexing on equality lookups

Indexing produced a substantial reduction in equality-lookup latency in both database systems. For MySQL, the indexed equality lookup decreased from 9.962 ms to 0.891 ms at 10K records, from 55.262 ms to 0.725 ms at 100K records, and from 540.517 ms to 1.192 ms at 1M records. The corresponding reductions were approximately 91.06%, 98.69%, and 99.78%.

MongoDB showed a similar pattern. Indexed equality-lookup latency decreased from 11.806 ms to 1.574 ms at 10K records, from 48.104 ms to 1.119 ms at 100K records, and from 518.589 ms to 1.984 ms at 1M records. These correspond to reductions of approximately 86.67%, 97.67%, and 99.62%.

The results indicate that the single-field user_id indexes were highly effective for the equality-lookup workload. The improvement became particularly pronounced as the dataset increased to 1M records. This supports the importance of appropriate indexing when applications frequently retrieve individual records using a selective attribute [6][4]**.**

5.3. Aggregation Performance

The aggregation workload produced more varied results than the equality-lookup workload. Without indexes, MongoDB was faster than MySQL at 100K and 1M records, while MySQL was faster at 10K records. With the category index, MongoDB was faster at all three dataset sizes.

At 1M records, for example, the indexed aggregation latency was 1,069.269 ms for MySQL and 304.607 ms for MongoDB. The corresponding warm-cache measurements were 733.467 ms and 304.103 ms, respectively.

The behavior of MySQL at 1M records is particularly notable. The indexed aggregation measurement was higher than its unindexed measurement: 1,069.269 ms compared with 704.294 ms. This does not imply that indexes generally reduce MySQL aggregation performance. Rather, it shows that the particular single-column category index used in this experiment did not provide a latency improvement for this aggregation workload at the largest dataset size.

The EXPLAIN output provides additional context. MySQL selected idx_category for the aggregation query, but the execution plan reported Using temporary. The query filtered records by category and then grouped the resulting records by status. Since the benchmark did not test a composite (category, status) index, no conclusion can be drawn about whether such an index would improve this workload.

A separate validation experiment was therefore performed for the 1M-record MySQL aggregation. With idx_category present, the validation average was 764.391 ms, while removing the index produced an average of 614.816 ms. Although these measurements differed from the original benchmark values, they reproduced the same direction of the observation: the indexed configuration was slower than the unindexed configuration under the tested conditions. The index was restored after validation.

5.4. Scaling Behavior

Dataset size had a substantial effect on unindexed query latency. [2][3][4] For MySQL, unindexed equality-lookup latency increased from 9.962 ms at 10K records to 540.517 ms at 1M records, while

unindexed aggregation increased from 9.903 ms to 704.294 ms. MongoDB showed similar growth, with unindexed equality lookup increasing from 11.806 ms to 518.589 ms and aggregation increasing from 20.231 ms to 590.154 ms.

In contrast, indexed equality lookups remained close to the millisecond range even at 1M records. MySQL increased from 0.891 ms at 10K records to 1.192 ms at 1M records, while MongoDB increased from 1.574 ms to 1.984 ms. This indicates that the tested indexes substantially reduced the growth in equality-lookup latency as the dataset increased.

Aggregation queries did not show the same behavior. Their latency continued to increase substantially with dataset size, particularly for MySQL. This difference illustrates that an index designed for a selective equality lookup does not necessarily provide the same performance benefit for an aggregation that filters and then groups a large number of records.

5.5. Warm-Cache Observations

The warm-cache measurements generally reduced equality-lookup latency relative to the corresponding unindexed measurements. At 1M records, warm-cache equality lookup was 1.137 ms for MySQL and 1.214 ms for MongoDB, compared with 540.517 ms and 518.589 ms in the unindexed configuration.

Warm-cache aggregation also improved several measurements, particularly for MongoDB. At 1M records, MongoDB's aggregation latency was 304.103 ms in the warm-cache condition compared with 590.154 ms in the unindexed condition. MySQL, however, recorded 733.467 ms in the warm-cache condition compared with 704.294 ms in the unindexed condition.

These observations show that cache state can influence measured query latency. The warm-cache condition in this study represents repeated execution after an initial query and should not be interpreted as evidence about MySQL Query Cache or as a controlled measurement of a particular internal caching mechanism.

5.6. Implications for the Research Questions

The results provide evidence for all three research questions.

For RQ1, MongoDB recorded lower bulk-write latency than MySQL at all tested dataset sizes, with the largest relative difference occurring at 100K records.

For RQ2, single-field indexing substantially reduced equality-lookup latency in both systems. Its effect on aggregation latency was less consistent and differed between database systems and dataset sizes.

For RQ3, query latency increased considerably with dataset size for unindexed operations, whereas indexed equality lookups remained relatively low. Aggregation workloads continued to show stronger sensitivity to dataset size.

Overall, the findings indicate that database performance is workload-dependent. The results support evaluating database systems using representative workloads rather than relying on a general assumption that either relational or document-oriented databases will perform better across all operations.

Taken together, the present observations are consistent with the broader literature in showing that comparative database performance is workload-dependent rather than determined solely by whether a

system is relational or document-oriented. Previous studies have reported different performance characteristics for MySQL and MongoDB across write, retrieval, CRUD, concurrent, and cloud-based workloads [1][10]. The present benchmark extends this perspective by evaluating indexing and warm-cache conditions alongside dataset scaling on a common single-host environment.

6 Threats To Validity and Limitations

Several limitations should be considered when interpreting the results of this study.

6.1. Hardware and Deployment Environment

The experiments were conducted on a single physical computer with an Intel Core i5-5200U processor, 8 GB of RAM, and a 256 GB SSD. MySQL and MongoDB were therefore evaluated under the same local hardware environment rather than under separate servers or distributed deployments. Differences in hardware architecture, storage devices, memory capacity, processor characteristics, or network latency may produce different performance results.

6.2. Synthetic Dataset

The benchmark used synthetically generated transaction data with 10K, 100K, and 1M records. Although the datasets were generated deterministically using a fixed random seed and contained the same logical records for both systems, they may not represent the distributions, data correlations, access patterns, or data-quality characteristics of production workloads. Consequently, the findings should not be interpreted as directly representative of every real-world database application.

6.3. Limited Workload Types

Only three workload categories were evaluated: bulk insertion, equality lookup, and filtered aggregation. The benchmark did not evaluate other important workloads such as concurrent transactions, complex joins, multi-document operations, range queries, updates, deletes, full-text search, or transactional contention. Therefore, the conclusions are limited to the operations included in the experiment.

6.4. Limited Index Configurations

The study evaluated single-field indexes on user_id and category. The aggregation workload used the category index while grouping by status. Composite indexes, covering indexes, alternative index designs, and database-specific indexing strategies were not evaluated. In particular, the results should not be interpreted as evidence that a composite index such as (category, status) would perform poorly or that the tested index configuration represents the optimal configuration for the aggregation workload.

6.5. Measurement Procedure

Each measured workload condition was executed for five iterations and the arithmetic mean was reported. Although this provides repeated measurements, a larger number of repetitions could provide stronger statistical characterization of latency variability. The original benchmark also retained the averaged measurements rather than the complete raw iteration-level dataset for every workload condition, limiting some forms of post-hoc statistical analysis.

Because the benchmark reports arithmetic means rather than confidence intervals or statistical significance tests, the results should be interpreted primarily as descriptive latency measurements under the tested conditions.

6.6. Cache and System-State Effects

The warm-cache condition was produced by executing an unmeasured query before the measured iterations. This approach represents a warmed database and memory state but does not provide complete control over operating-system caches, database buffer state, background processes, or other system-level effects. The warm-cache measurements should therefore be interpreted as observations under the defined experimental procedure rather than as a complete isolation of caching effects.

6.7. Concurrency and Distributed Scalability

The experiments used sequential execution on a single host and did not evaluate multiple concurrent clients. Consequently, the study does not measure throughput under contention, lock behavior under concurrent transactions, connection-management overhead, or distributed scaling. These factors can be particularly important when evaluating database systems intended for high-throughput production environments. Previous benchmarking studies have examined database performance under concurrent workloads and more demanding execution conditions [7],[4] , providing broader perspectives on performance beyond the sequential execution used in the present study.

Cloud-based and distributed database studies have also evaluated deployment environments beyond a single-host configuration [10],[2], whereas the present experiment intentionally excludes such architectures.

6.8. Generalizability

The results describe the behavior of MySQL 8.0.46 and MongoDB 8.0.32 under the specific schema, indexes, workload definitions, dataset sizes, hardware, software environment, and measurement procedure used in this study. They should therefore be interpreted as empirical observations for the tested configuration rather than as universal performance rankings between SQL and NoSQL database systems.

Despite these limitations, deterministic dataset generation, identical logical input data, explicit workload definitions, repeated measurements, documented index configurations, and independent remeasurement of the unusual 1M-record MySQL aggregation result improve the reproducibility and transparency of the experiment.

7 Conclusion And Future Work

This study presented an empirical comparison of MySQL 8.0.46 and MongoDB 8.0.32 using bulk-write, equality-lookup, and filtered-aggregation workloads across datasets containing 10K, 100K, and 1M transaction records. The experiments evaluated unindexed, indexed, and warm-cache conditions using a deterministic synthetic dataset and a consistent benchmarking procedure.

The results showed that MongoDB recorded lower bulk-write latency than MySQL at all three dataset sizes. At 1M records, MongoDB required 12,547.836 ms compared with 29,617.741 ms for MySQL. Equality-lookup performance showed a strong response to single-field indexing in both systems. At 1M

records, indexing reduced equality-lookup latency by approximately 99.78% for MySQL and 99.62% for MongoDB relative to their corresponding unindexed measurements.

Aggregation performance showed more workload-dependent behavior. MongoDB produced lower indexed aggregation latency than MySQL at all three dataset sizes, while the difference between the systems varied across configurations. The 1M-record MySQL aggregation also produced a higher measured latency with the tested category index than without it. An independent remeasurement reproduced the same direction of this observation under the tested conditions. However, this result should not be generalized to other index designs because composite and alternative indexing strategies were outside the scope of the experiment.

Overall, the findings demonstrate that database performance depends strongly on workload characteristics, dataset size, indexing strategy, and execution state. The experiment therefore does not establish a universal performance advantage for either relational or document-oriented databases. Instead, it provides empirical evidence that database selection and query optimization should be evaluated against the specific access patterns and workload requirements of an application.

Future work can extend this study in several directions. First, additional index configurations, including composite and covering indexes, can be evaluated to investigate their effect on aggregation workloads, building on previous research that examined simple and compound indexing strategies and their effects on query performance [6]. Second, concurrent clients and higher request rates can be introduced to measure throughput, latency under contention, and scalability, extending the type of concurrent benchmarking examined in previous studies [7],[4]. Third, larger and more realistic datasets can be used to evaluate behavior beyond 1M records, extending the larger-scale evaluations reported in previous studies [3],[4]. Fourth, additional workloads such as updates, deletes, range queries, joins, and complex aggregations can provide a broader comparison. Finally, the experiment can be extended to multi-node or distributed deployments to examine network overhead, horizontal scalability, and fault-tolerance characteristics of both database systems, building on previous work involving distributed and cloud-based database environments [10],[2].

8 Declarations

Funding

No funding was received for conducting this research or for the preparation of this manuscript.

Data Availability Statement

The datasets generated during and analysed during the current study, together with the associated benchmarking scripts and experimental results, are available in Zenodo at https://doi.org/10.5281/zenodo.23348760. The source code and project documentation are also available at https://github.com/amirkhan421/sql-nosql-benchmark-data.

Ethics, Consent to Participate, and Consent to Publish declarations

Ethics, Consent to Participate, and Consent to Publish declarations: not applicable.

References

[1] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, and B. Luo, "SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review," Jun. 01, 2023, *MDPI*. doi: 10.3390/bdcc7020097.

[2] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, "Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments," *IEEE Access*, vol. 8, pp. 110656–110668, 2020, doi: 10.1109/ACCESS.2020.3002164.

[3] C. A. Győrödi, D. V. Dumşe-Burescu, D. R. Zmaranda, R. Győrödi, G. A. Gabor, and G. D. Pecherle, "Performance analysis of nosql and relational databases with couchdb and mysql for application's data storage," *Applied Sciences (Switzerland)*, vol. 10, no. 23, pp. 1–21, Dec. 2020, doi: 10.3390/app10238524.

[4] S. Keshavarz, "Analyzing Performance Differences Between MySQL and MongoDB," 2021.

[5] C. A. Győrödi, D. V. Dumşe-Burescu, D. R. Zmaranda, and R. Győrödi, "A Comparative Study of MongoDB and Document-Based MySQL for Big Data Application Data Management," *Big Data and Cognitive Computing 2022, Vol. 6,* vol. 6, no. 2, May 2022, doi: 10.3390/BDCC6020049.

[6] C. A. Győrödi, D. V. Dumşe-Burescu, R. Győrödi, D. R. Zmaranda, L. Bandici, and D. E. Popescu, "Performance impact of optimization methods on MySQL document-based and relational databases," *Applied Sciences (Switzerland)*, vol. 11, no. 15, Aug. 2021, doi: 10.3390/app11156794.

[7] R. Pandey, "Performance Benchmarking and Comparison of Cloud-Based Databases MongoDB (NoSQL) Vs MySQL (Relational) using YCSB."

[8] E. Khashan, A. Eldesouky, and S. Elghamrawy, "An adaptive spark-based framework for querying large-scale NoSQL and relational databases," *PLoS One*, vol. 16, no. 8 August, Aug. 2021, doi: 10.1371/journal.pone.0255562.

[9] M. Zohaib Khan Software, D. Engineer at SMBBIT Karachi, F. Uz Zaman, M. Adnan Engineer at KPT Karachi, A. Imroz, and M. Abdul Rauf, "Comparative Case Study: An Evaluation of Performance Computation Between SQL And NoSQL Database," 2022.

[10] H. A. Mumtahana, "Optimization of Transaction Database Design with MySQL and MongoDB," *SinkrOn*, vol. 7, no. 3, pp. 883–890, Jul. 2022, doi: 10.33395/sinkron.v7i3.11528.