Università degli Studi di Messina

Library Management System Project

PROFESSOR: Armando Ruggeri

Report By: Saniya Khanam (537482)

CONTENTS:

1. Introduction

This project presents the development of a Library Management System implemented using five different database management systems: MySQL, MongoDB, Cassandra, Neo4j, and Redis. The main goal is to evaluate and compare the performance of these systems under similar conditions by loading identical datasets and executing a series of queries with increasing complexity. This study aims to determine how each DBMS scales as the dataset grows, and to identify the strengths and limitations inherent in each technology.

2. DBMS Solutions Considered

The project uses a diverse set of databases representing different data models:

- **MySQL:** A traditional relational DBMS known for its strong ACID properties and efficient query processing for structured data.
- **MongoDB:** A document-oriented database offering schema flexibility and rapid development, ideal for JSON-like data.
- **Cassandra:** A wide-column store designed for high scalability and availability, though it may require careful query design.
- **Neo4j:** A graph database optimized for managing and traversing complex relationships, making it well-suited for network-based queries.
- **Redis:** An in-memory key-value store celebrated for its speed in simple retrievals, though complex operations may pose challenges.

Each of these systems was deployed in Docker containers to ensure uniform hardware and software conditions during testing.

3. Design

3.1 Data Model

The system models three core entities:

- **Books:** Attributes include book_id, title, author, year, and genre.
- **Borrowers:** Characterized by borrower_id, name, and email.
- **Transactions:** Records borrowing events with fields such as transaction_id, book_id, borrower_id, borrow_date, and return_date.

For Neo4j, books and borrowers are represented as nodes, and borrowing events are modeled using a BORROWED relationship with properties for borrow_date and return_date.

3.2 Dataset Generation and Subset Creation

Datasets were generated using random data generation libraries and saved as CSV files. To study performance at scale, these datasets were partitioned into subsets representing 25%, 50%, 75%, and 100% of the full dataset (e.g., 250k, 500k, 750k, and 1000k records). This allowed for a controlled evaluation of how each DBMS handles increasing volumes of data.

4. Implementation

The project was implemented using a series of Python scripts that perform the following tasks:

4.1 Data Generation: A script generated random data for books, borrowers, and transactions, outputting CSV files.

4.2 Subset Creation: A separate script partitioned the full dataset into subsets (e.g., books_25.csv, borrowers_25.csv, transactions_25.csv).

4.3 Data Insertion: Custom insertion scripts were written for each database system, ensuring that the same data was loaded into MySQL, MongoDB, Cassandra, Neo4j, and Redis.

4.4 Query Performance Testing: For each DBMS, a set of four queries with increasing complexity was executed. Performance metrics—cold-run time (first execution), average execution time (over 30 iterations), and 95% confidence intervals—were measured and recorded.

4.5 Visualization: A plotting script was developed to generate bar charts that compare the mean execution times of the queries across different dataset sizes and databases.

5. Experiments

5.1 Experimental Setup

For each database and dataset size (250k, 500k, 750k, 1000k), the following procedure was used:

1. **Load Data:** The appropriate CSV subset was loaded into the database.
2. **Execute Queries:** Four queries were run, with each query being executed once for a cold run and then 30 additional times to calculate the average performance.
3. **Record Metrics:** The following metrics were captured:
    - **First Execution Time (ms):** Time for the initial (cold) execution.
    - **Average Execution Time (ms):** Mean time over 30 runs.
    - **95% Confidence Interval (ms):** Statistical confidence around the average execution time.

5.2 Experimental Results

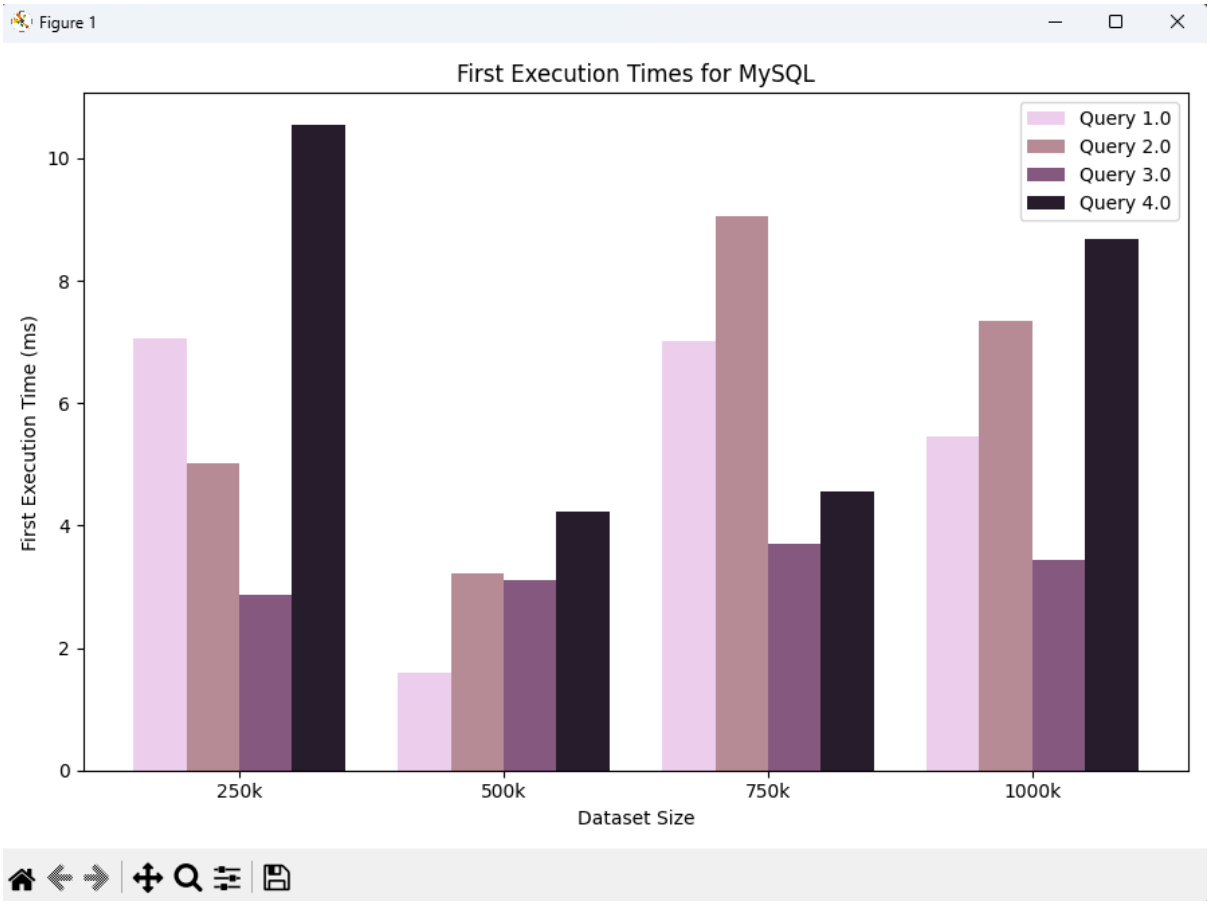
A sample of the compiled performance results is shown below:

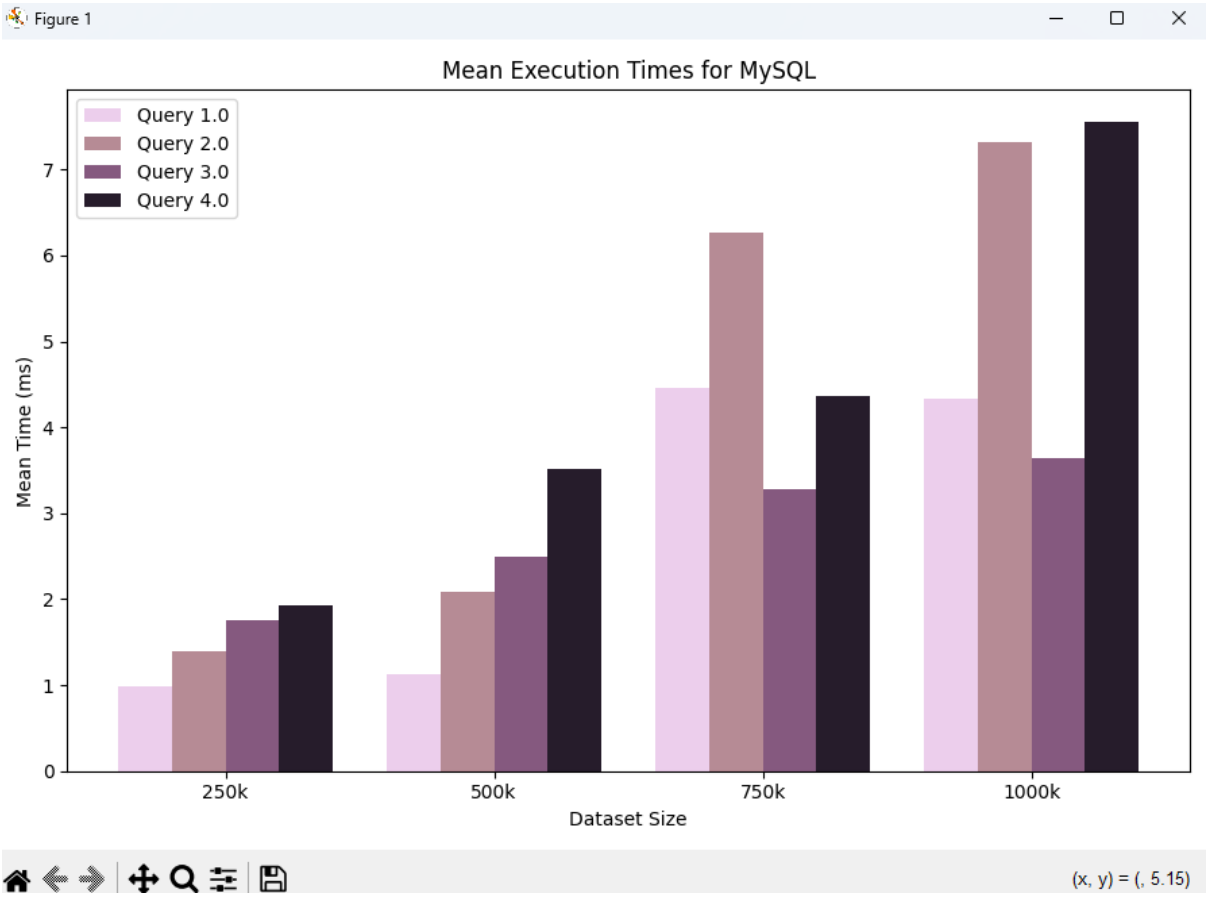
dataset_size	database	query_number	first_time (ms)	mean_time (ms)	ci (ms)
250k	MySQL	1	7.06	0.99	0.11
250k	Cassandra	2	6009.04	1389.24	907.81
250k	MongoDB	3	25.25	3.50	0.25
250k	Neo4j	4	167.38	8.24	0.53
250k	Redis	1	190.33	184.80	5.68
...	...	...	...	...	...
1000k	MySQL	4	8.69	7.55	0.16
1000k	Cassandra	4	27958.26	28229.58	126.89
1000k	MongoDB	4	1497.09	1906.59	269.90
1000k	Neo4j	4	120.01	44.02	3.24
1000k	Redis	4	2828.36	2980.14	95.15

(The full CSV includes results for all queries across all database systems.)

5.3 Observations

- **MySQL:** Demonstrated consistently low execution times across all queries with only a slight increase as the dataset size grows.





- **Cassandra:** While Query 1 and Query 3 perform reasonably well, Queries 2 and 4 show dramatic performance degradation at higher dataset sizes, indicating inefficiencies in handling complex aggregations.

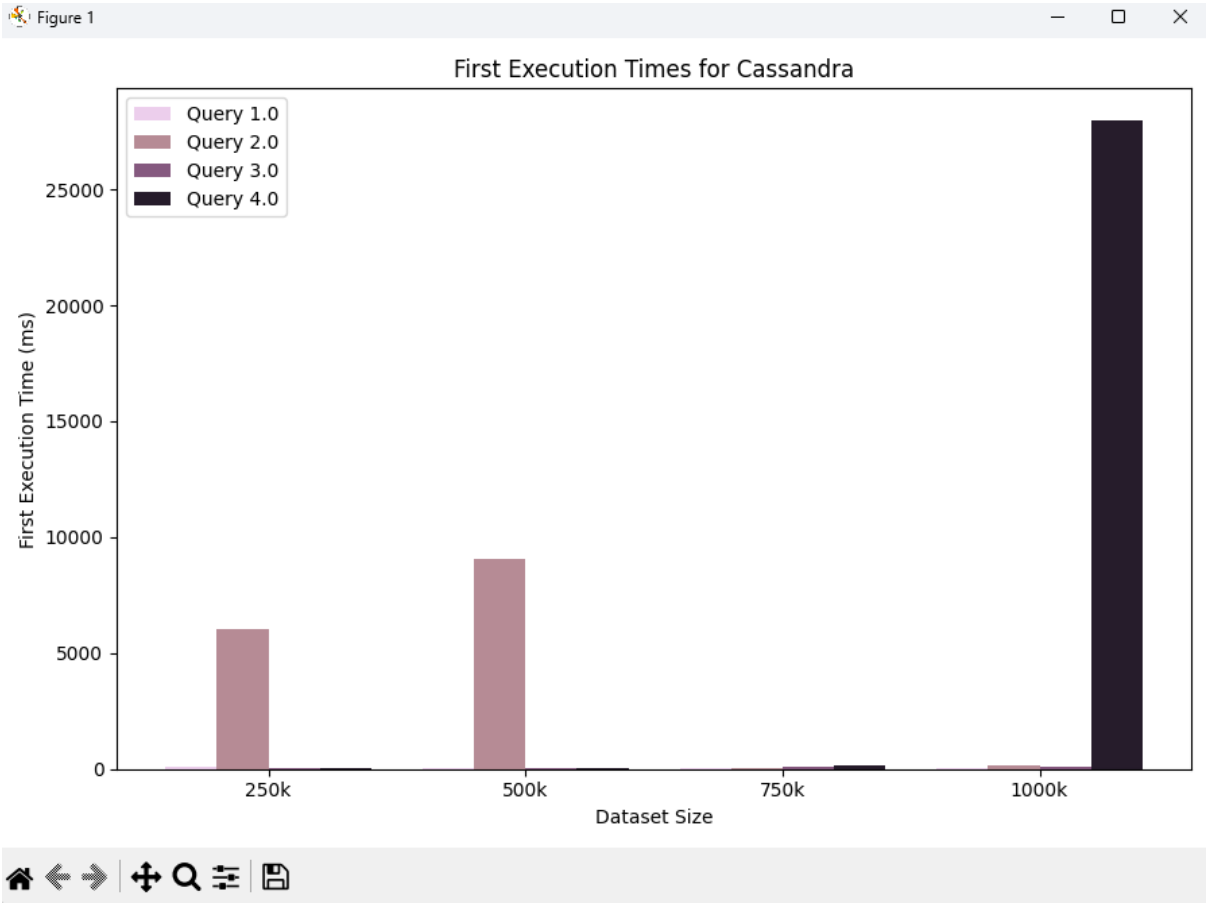
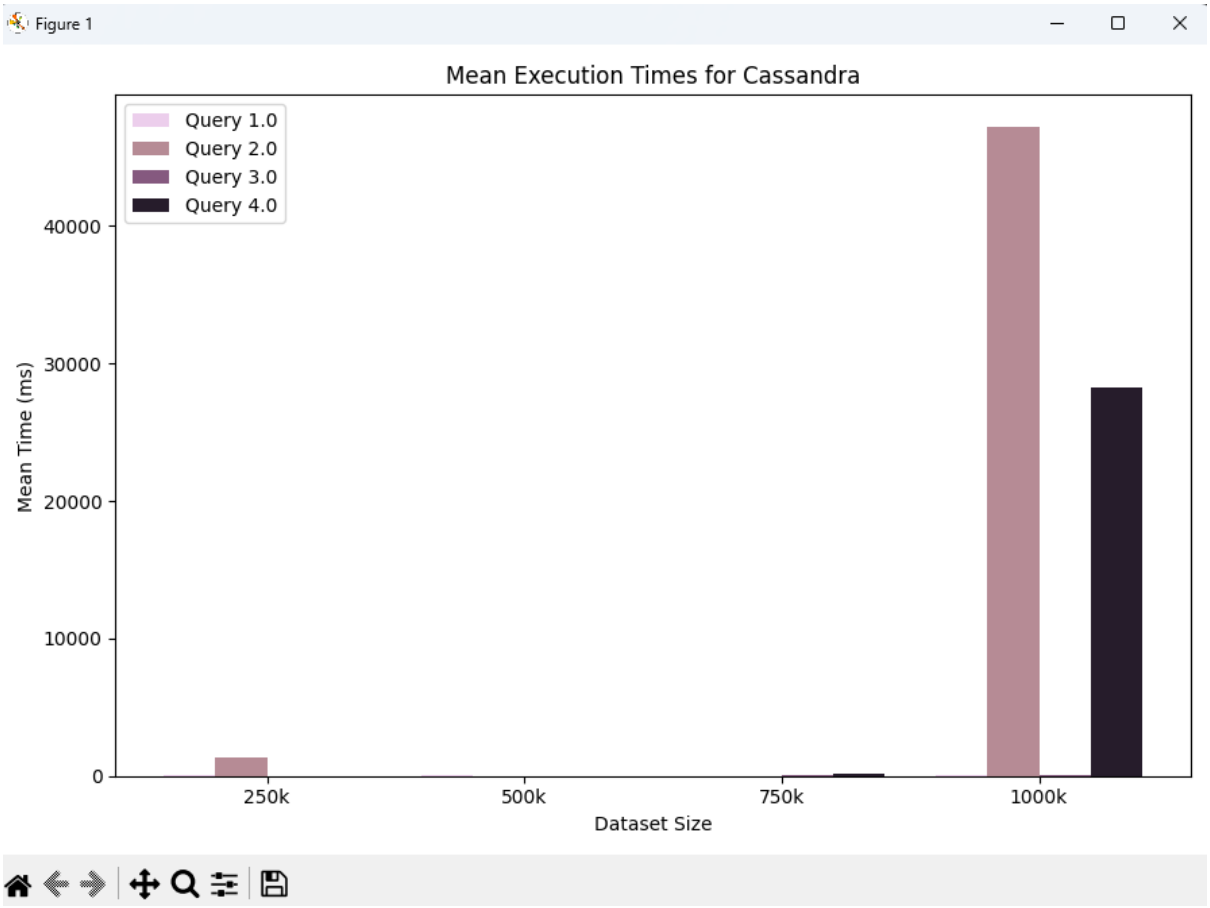


Figure 1
First Execution Times for Cassandra
Query 1.0
Query 2.0
Query 3.0
Query 4.0
First Execution Time (ms)
0
5000
10000
15000
20000
25000
250k
500k
750k
1000k
Dataset Size



- **MongoDB:** Simple queries (Query 1 and Query 3) executed extremely fast; however, complex aggregations (Query 2 and Query 4) experienced significant slowdowns with larger datasets.

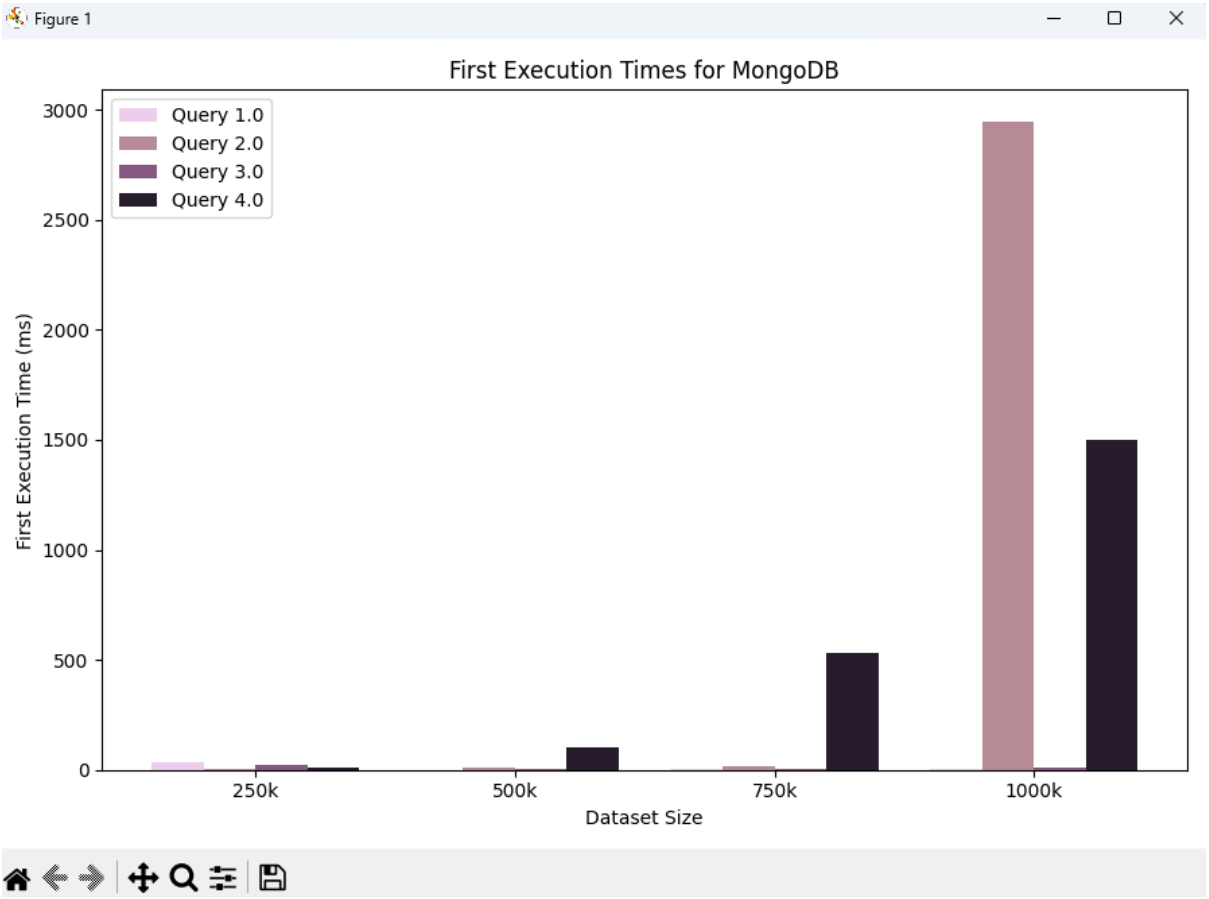
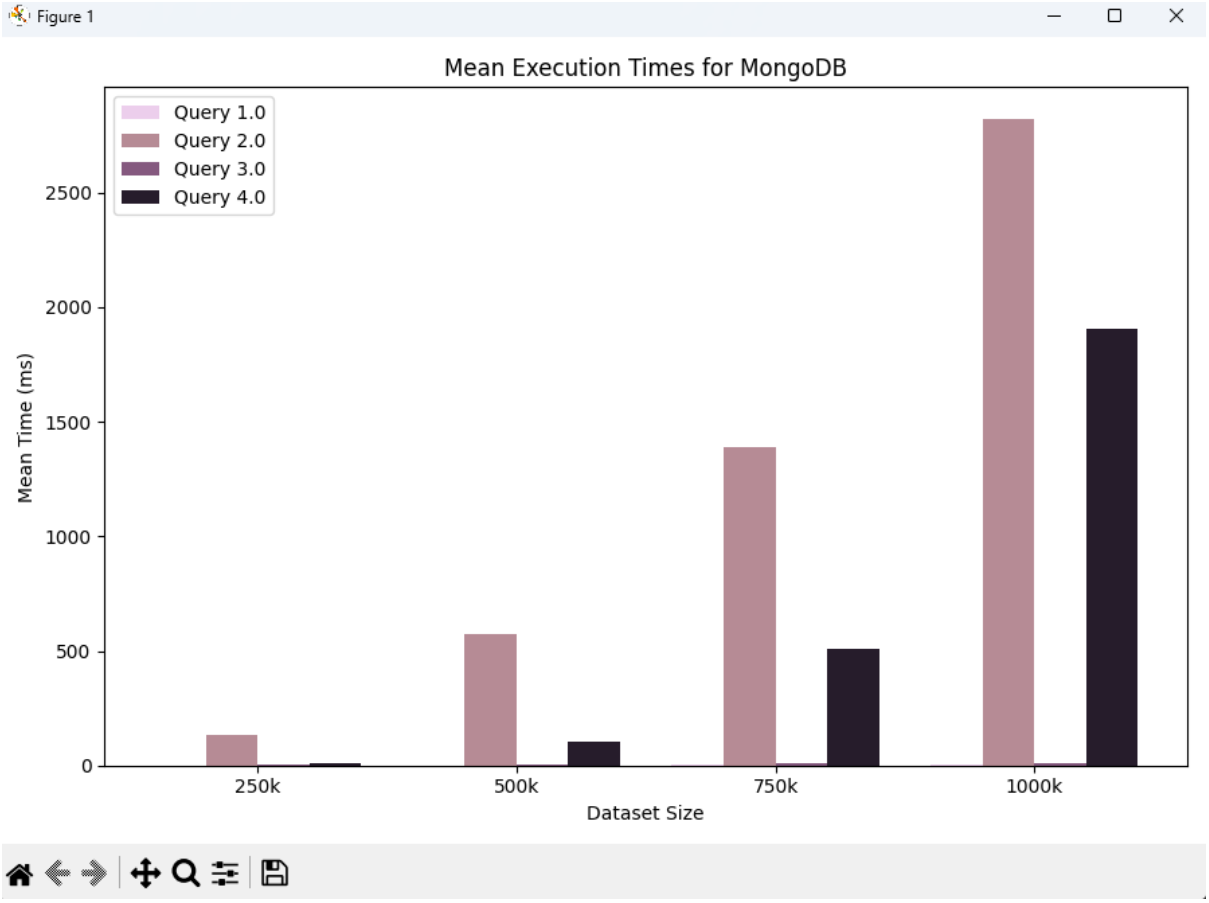


Figure 1
First Execution Times for MongoDB
Query 1.0
Query 2.0
Query 3.0
Query 4.0
First Execution Time (ms)
3000
2500
2000
1500
1000
500
0
250k
500k
750k
1000k
Dataset Size



- **Neo4j:** Showed stable and consistent performance across queries, particularly excelling in relationship-based queries.

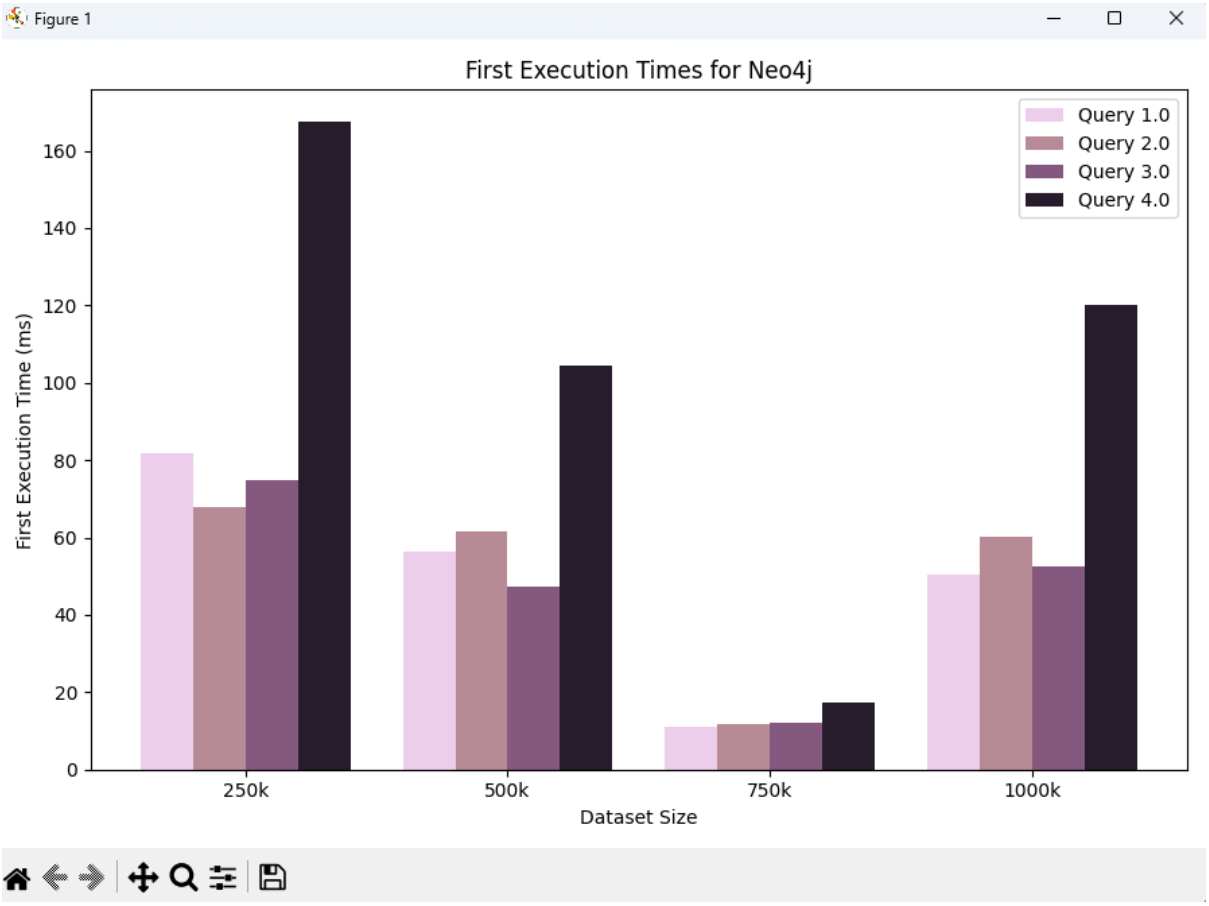


Figure 1
First Execution Times for Neo4j
Query 1.0
Query 2.0
Query 3.0
Query 4.0
First Execution Time (ms)
0
20
40
60
80
100
120
140
160
250k
500k
750k
1000k
Dataset Size

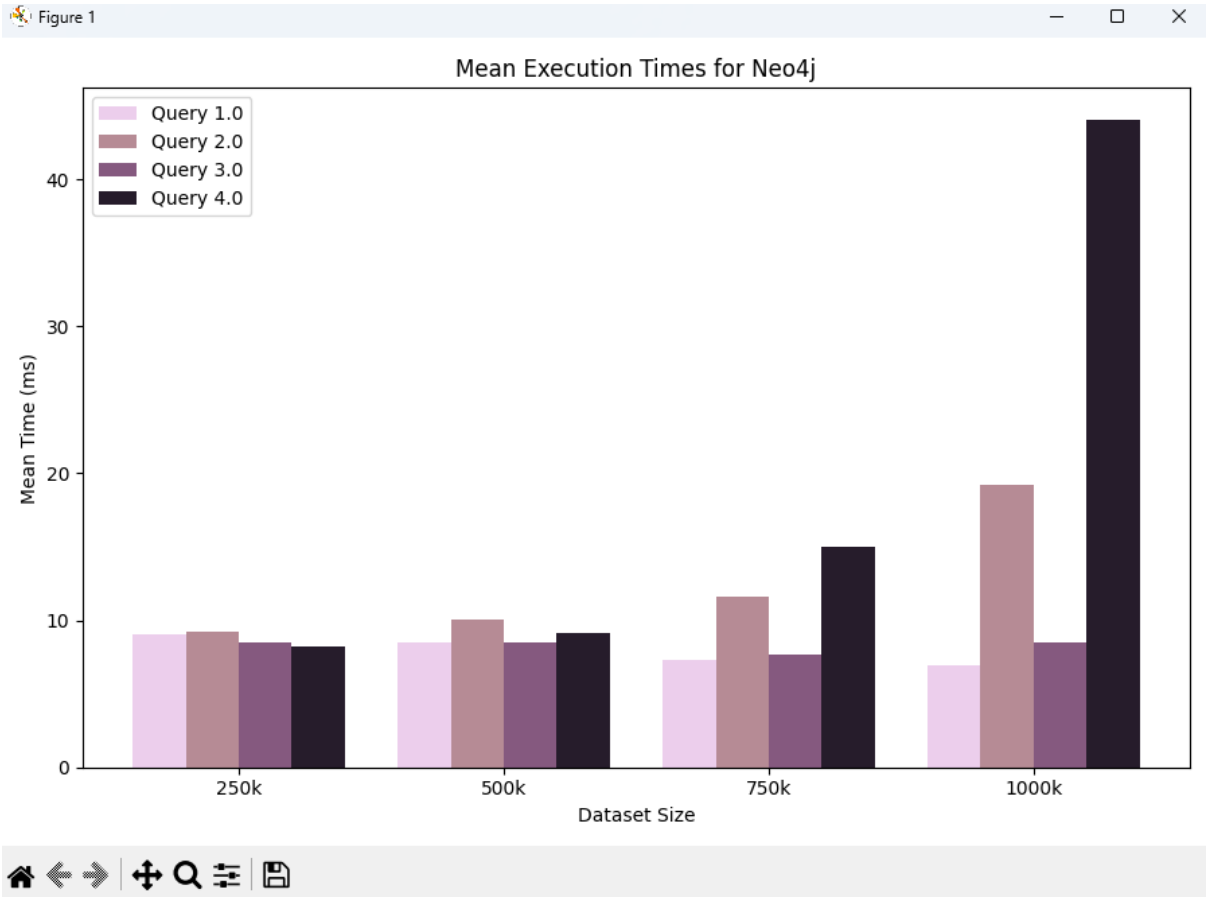
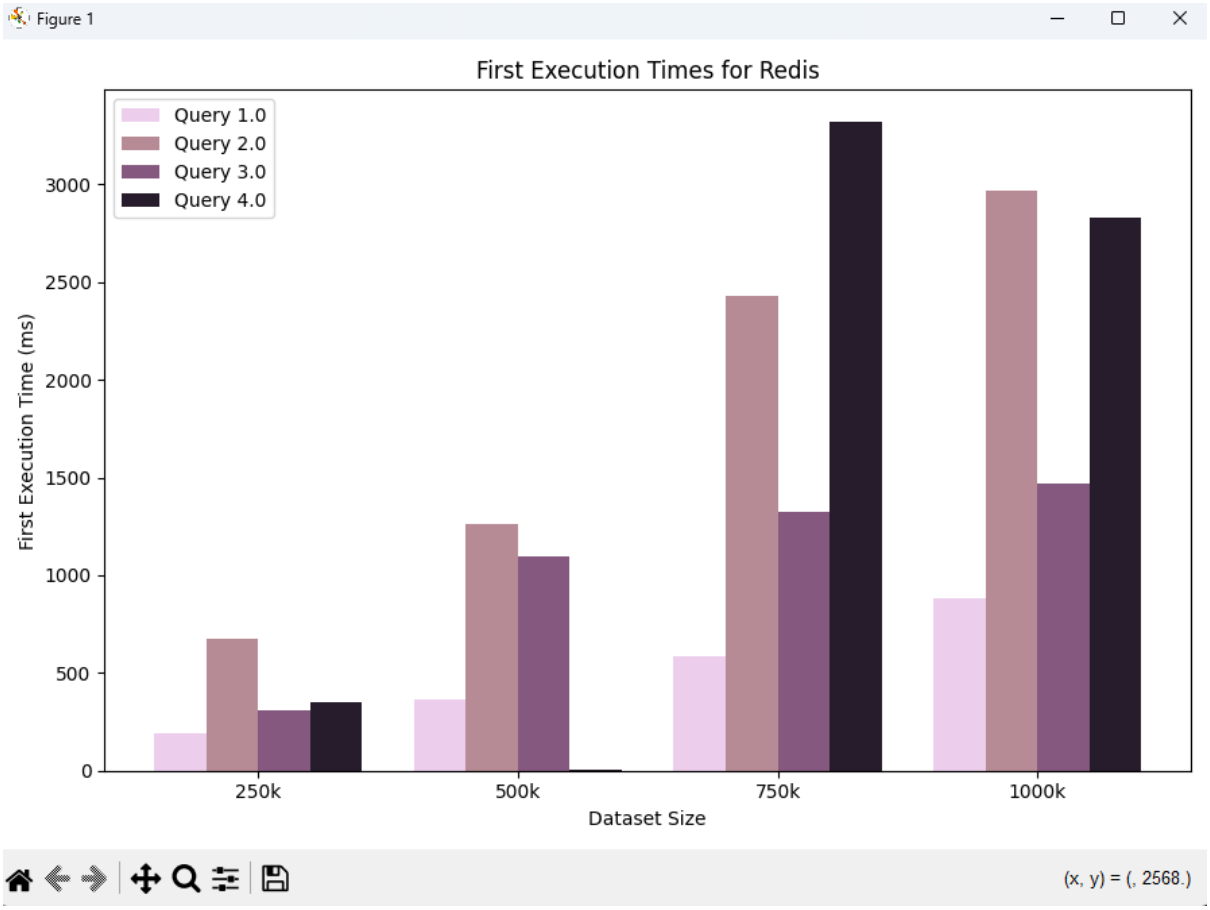
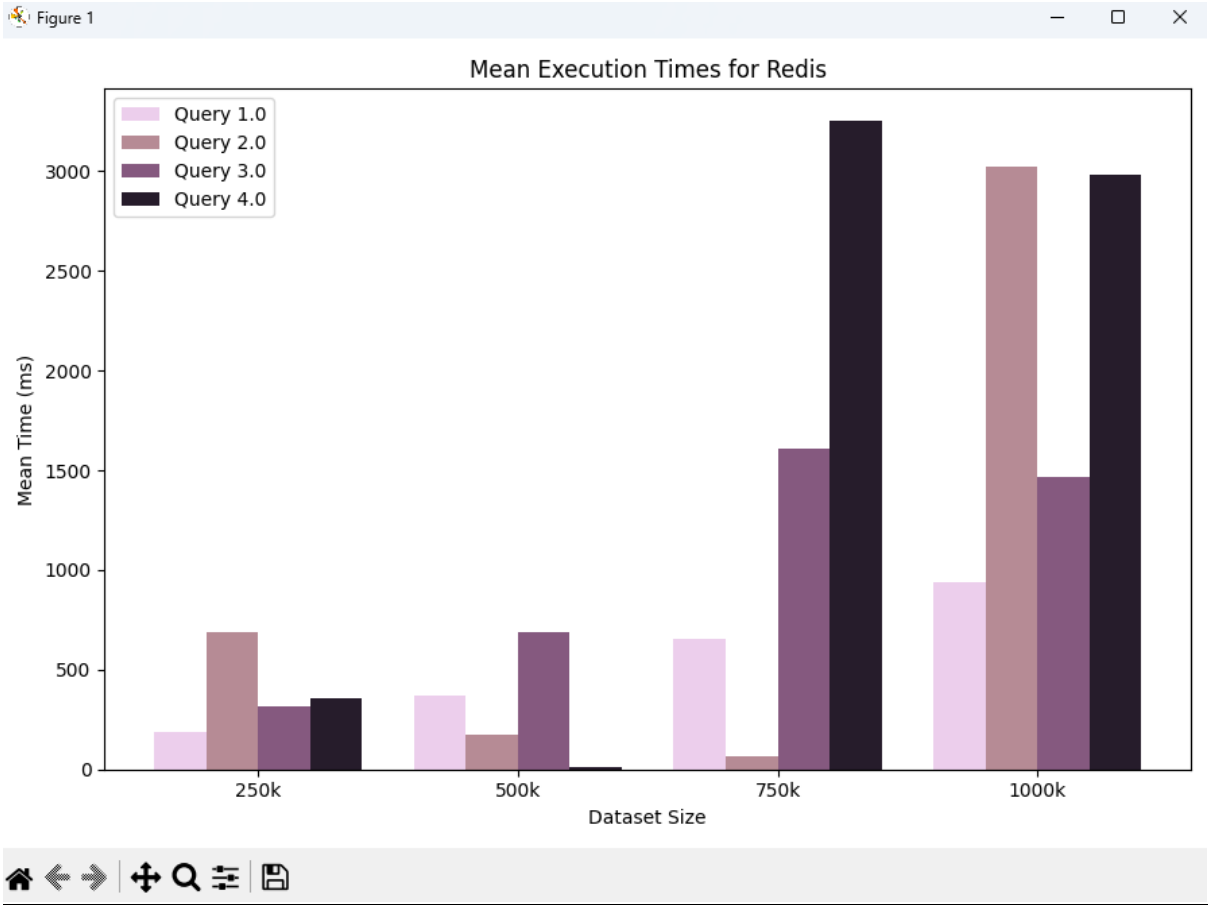


Figure 1
Mean Execution Times for Neo4j
Query 1.0
Query 2.0
Query 3.0
Query 4.0
Mean Time (ms)
0
10
20
30
40
250k
500k
750k
1000k
Dataset Size

- **Redis:** Exhibited high and variable execution times for queries that involve join-like operations, reflecting the challenges of simulating complex queries in an in-memory key-value store.





Visualizations (bar charts and line graphs) were generated to compare the mean execution times for each query across different dataset sizes and databases. These charts further illustrate the scalability and performance differences between the systems.

Limitations:

- **Experimental Environment:** The tests were conducted under controlled conditions using Docker containers on uniform hardware. Real-world environments might introduce additional variability due to network latency, hardware differences, and concurrent workloads.
- **Query Complexity:** The chosen queries, while representative, might not capture all possible use cases in a full-scale Library Management System. There could be other query patterns where these databases perform differently.
- **Data Generation:** The datasets were generated using random data generators, which may not fully mimic real-world data distributions and relationships.

Future Improvements:

- **Query Optimization:** Future work could focus on optimizing query design and indexing strategies for each DBMS, particularly for those systems (like Cassandra and MongoDB) that show performance bottlenecks under complex aggregations.
- **Hybrid Approaches:** Exploring hybrid database architectures that combine the strengths of multiple DBMS types might yield better overall performance for heterogeneous workloads.
- **Extended Testing:** Further experiments could involve real-world data and additional query types to validate and expand upon the current findings.
- **Scalability Analysis:** Additional tests could be performed at even larger data scales or with concurrent user simulations to better understand how these systems perform under high load.

Formatting and Clarity:
All figures, tables, and code references in this report have been carefully formatted and clearly labelled. For example, performance results are presented in Table 1, and each graph includes descriptive axis labels and legends. Detailed explanations accompany each visual aid, ensuring that the reader can easily interpret the data and understand how it supports the analysis.

6. Conclusions

Based on our experimental results and analysis, we conclude the following:

- **MySQL** is highly efficient for the Library Management System workload, providing excellent performance with minimal scaling issues.
- **Cassandra** is suitable for simple queries, but its performance for complex aggregations deteriorates significantly as the dataset grows. This indicates a need for optimization or reconsideration of the query design for large-scale operations.
- **MongoDB** performs very well for straightforward retrievals; however, complex queries—particularly those involving aggregations—result in a noticeable performance penalty.
- **Neo4j** offers robust and consistent performance, particularly excelling in queries that require navigating relationships. This makes it an ideal choice for applications where relationship traversal is critical.
- **Redis** excels in simple key-value lookups, but simulating join operations and complex queries leads to higher and more inconsistent execution times.

These insights provide valuable guidance for selecting the appropriate DBMS for different use cases within a Library Management System. Future work could focus on optimizing query strategies and indexing schemes, as well as exploring additional NoSQL systems or hybrid approaches to further improve performance.

7. Appendices

7.1 Code Listings

- **Data Generation and Subset Creation:** data_generator.py, create_subsets.py
- **Data Insertion Scripts:** mysql_inserter.py, cassandra_inserter.py, mongodb_inserter.py, neo4j_inserter.py, redis_inserter.py
- **Query Performance Testing Scripts:** query_performance.py, mysql_query_performance_multi.py, cassandra_query_performance.py, mongodb_query_performance.py, neo4j_query_performance.py, redis_query_performance.py
- **Visualization Script:** plotMySQL_results.py, plotCass_results.py, plotMong_results.py, plotNeo_results.py, plotRedis_results.py.

7.2 Experimental Results in a CSV File

- The full CSV file performance_results.csv is included in the project package.

Final Remarks

This project provides a comprehensive analysis of different DBMS solutions applied to a Library Management System. The experiments reveal the strengths and limitations of each system, guiding future decisions on database selection and optimization. The provided code, detailed experimental results, and visualizations collectively contribute to a robust evaluation framework.