## OPTIMIZING QUERY PERFORMANCE IN DISTRIBUTED DATABASES

#### A MINOR PROJECT REPORT

***Submittedby***

***in partial fulfillment for the award of the degree of***

### MASTERS OF COMPUTER APPLICATION

**IN**

#### Chandigarh University

OCTOBER2025

### BONAFIDE CERTIFICATE

Certified that this project report **“Optimizing query performance in distributed databases”** is the bonafide work of “ Name **”** who carried out the project work under my/oursupervision.

**SIGNATURE SUPERVISOR**

Submitted for the project viva-voce examination held on

## Acknowledgement:

We would like to express our sincere gratitude to all those who have contributed to the successful completion of this project. Their support, guidance, and encouragement were invaluable throughout the journey.

First and foremost, we extend our heartfelt appreciation to …….. our project supervisor, for their unwavering guidance and insightful feedback. Their expertise and dedication greatly enriched our project and its outcomes.

We extend our thanks to University of Computing, who generously provided needed resources. Their contribution significantly enhanced the quality of our work and expanded our project's horizons.

Furthermore, we are also expressing a greatful thanks to all the team members who put their tireless efforts into this project and helps to bring this project into successful completion. Each team member's unique skills and perspectives played a crucial role in shaping the project's direction and achieving its goals.

Lastly, we want to acknowledge the broader community and the resources that have been instrumental in our research. The collective knowledge available through various sources, including books, articles, and online forums, provided us with a strong foundation to build upon.

In conclusion, this project would not have been possible without the collaborative efforts and support of all those mentioned above. Thank you for being a part of this journey.

### TABLE OF CONTENTS

[Abstract](#_bookmark0) 5

[Introduction of the project](#_bookmark1) 6

**Literature review 7**

**Methodology** 9

**Code of the project** 11

**Result** 15

**Comparison** 17

**Dummy dataset** 17

**Future Scope** 18

**Conclusion** 19

**References** 20

1. **Abstract**

In modern applications, databases handle massive amounts of distributed data that must be processed, managed, and queried efficiently to ensure smooth and fast access to information. As the volume of data continues to grow, query optimization has become a vital aspect of improving system responsiveness, scalability, and overall performance.

This project focuses on **optimizing query performance in a distributed MongoDB environment** using **Java-based benchmarking** and **indexing techniques**. A web-based frontend titled “Join the Club” enables user interaction through a signup form, while the backend—developed using Java Servlets—handles data storage and retrieval operations through MongoDB.

The system measures query execution time before and after optimization, providing a comparative analysis that highlights the effectiveness of indexing and caching in improving performance. Benchmark results clearly indicate that query execution times significantly decrease after indexes are applied, demonstrating the impact of optimization on distributed database efficiency.

This study not only provides practical insights into query optimization strategies but also emphasizes the importance of performance tuning in real-world distributed database systems, making it a valuable contribution to modern data management and analytics practices.

## Introduction of the Project:

In today’s data-intensive world, organizations rely heavily on databases to store, retrieve, and analyze large volumes of information. As applications become more complex and data is distributed across multiple nodes or servers, ensuring efficient query performance becomes a significant challenge. Poorly optimized queries can lead to delays, high resource consumption, and reduced system responsiveness, which directly affects user experience and business operations.

**Query optimization** is the process of improving the efficiency of database queries by analyzing different execution strategies and selecting the one that provides the best performance with minimal resource usage. In distributed databases, optimization becomes even more crucial because data is stored across several machines, and query execution involves communication between nodes, data partitioning, and aggregation.

This project focuses on **analyzing and optimizing query performance in MongoDB**, a widely used NoSQL distributed database known for its scalability and flexibility. The project integrates **Java Servlets** for backend development and a visually appealing frontend that allows users to join an astronomy club through a “Join the Club” form. Once user data is submitted, it is stored in MongoDB, and the system simultaneously performs benchmarking experiments to compare query performance **before and after optimization**.

By implementing **indexing**, **projection**, and **caching** techniques, this project evaluates how these optimizations affect query response time and overall system performance. Performance analysis is conducted using **MongoDB Explain Plans** and **Java-based benchmarking tools**, which record the execution time of queries and display improvements in both numerical and visual form.

Through this study, students and developers gain practical knowledge of how query optimization techniques enhance distributed database performance. The project not only demonstrates technical implementation but also highlights the importance of performance tuning for achieving high efficiency, scalability, and reliability in modern database-driven applications.

## Literature review

### Introduction to Distributed Database Systems and Performance

The current data landscape, characterized by **Big Data** and high-concurrency applications, has driven the evolution from centralized databases to **Distributed Database Systems (DDBs)**. DDBs manage and process data across multiple interconnected nodes, primarily to achieve superior **scalability**, **high availability**, and **performance** that monolithic architectures cannot sustain.

**The Problem of Query Optimization in DDBs:** In a distributed environment, query performance is inherently more complex than in centralized systems. Key challenges include:

**Distributed Query Execution:** An efficient query optimizer must devise a plan that minimizes data transfers and maximizes parallel processing across the shards.

### 2. MongoDB in Distributed Architectures

**MongoDB Sharding and Query Router:** MongoDB implements distribution through **sharding**, where data is partitioned across a cluster. The central component, the **Query Router (mongos)**, is responsible for receiving client requests and directing them to the relevant shards.

**The Shard Key Imperative:** Extensive literature confirms that the selection of an optimal **shard key** is the single most critical factor in distributed MongoDB performance. An ineffective shard key forces **"scatter-gather"** operations (querying all shards and merging results), which severely degrades performance, effectively negating the benefit of distribution.

**Core MongoDB Query Optimization Techniques:** Successful query performance optimization in MongoDB relies on developer-side best practices, which your project will test empirically:

**Indexing:** Creating appropriate indexes (compound and covered) is fundamental. Indexes drastically reduce disk I/O by allowing the query engine to scan only a small subset of documents.

**Projection and Limitation:** Using the projection feature to return **only necessary fields** and using the limit() method minimizes network traffic and client-side processing, a crucial factor when dealing with large datasets.

**Aggregation Framework:** For complex analytical operations, the aggregation pipeline often outperforms multiple individual queries by pushing computation to the server and executing steps in parallel across shards.

### 3. Comparison Methodologies and Tooling in Performance Analysis

Comparing database performance requires a rigorous, controlled environment. Existing studies often utilize industry benchmarks to validate findings; however, a custom comparison using application-level tools can provide real-world insights specific to a project's needs.

**Performance Metrics:** The literature highlights three primary metrics essential for evaluating distributed database performance:

**Latency (Response Time):** The time taken for the application (Java client) to receive a complete response to a query. This is the primary user-facing performance indicator.

**Throughput:** The volume of operations (queries or writes) the system can handle per second.

**Resource Utilization:** Monitoring CPU and memory usage on the database server to ensure the system is not bottlenecked by hardware.

**Role of Java as the Client Language:** Using **Java** (specifically, the MongoDB Java Driver) for implementing the test workload provides an **application-centric view** of performance. The measurements will capture the true end-to-end latency, including driver overhead and network time, which is more practical than server-side logging alone.

**Role of Excel for Analysis:** While sophisticated tools exist, **Excel** serves as an effective, accessible tool for **descriptive statistics** and **visualization**. It is well-suited for calculating essential metrics (mean, median, standard deviation of latency) and creating comparative charts, making the performance differences between optimized and non-optimized queries clear and digestible for the project report.

## Methodology

### Experimental Setup and Architecture

**Database System:**

**Platform:** MongoDB Distributed Cluster (at least a 3-node Replica Set, or a Sharded Cluster for demonstrating true distribution).

**Configuration:** Specify the hardware

**Distributed Setup:** If using sharding, detail the shard key chosen and the number of shards. (If you cannot set up true sharding, a 3-node Replica Set is the minimum required to simulate a distributed read environment).

**Client Application (Executor):**

**Language:** Java (using the **MongoDB Java Driver**).

**Purpose:** The Java application will host the test logic, execute the defined queries against the MongoDB cluster, and log the execution time for each query.

**Data Analysis Tool:**

**Tool:** Microsoft Excel.

### 2. Dummy Dataset Generation

**Data Model:** Design a simple, document-based schema (e.g., an orders collection or a sensor\_data collection).

**Fields:** Include fields suitable for indexing, sorting, and range queries (e.g., \_id, timestamp, user\_id, status, value).

**Data Volume:** A large volume of data is crucial for testing distributed performance.

**Data Distribution:** Ensure the dummy data is loaded evenly across the nodes if sharding is employed.

### 3.Experimental Procedure

**Warm-up Phase:** Execute each test query 50 times before actual recording begins. This fills the cache and ensures the system is in a stable, running state.

**Execution and Logging (Java):**

For each of the 8 test query variations (4 Non-Optimized and 4 Optimized), execute the query **100 times**.

For each execution, the Java application will use a high-resolution timer to record the **execution time (latency)** in milliseconds (ms).

Log the results into a Comma Separated Values (**CSV**) file with columns: Test\_Case, Optimization\_Status, Execution\_Number, Latency\_ms.

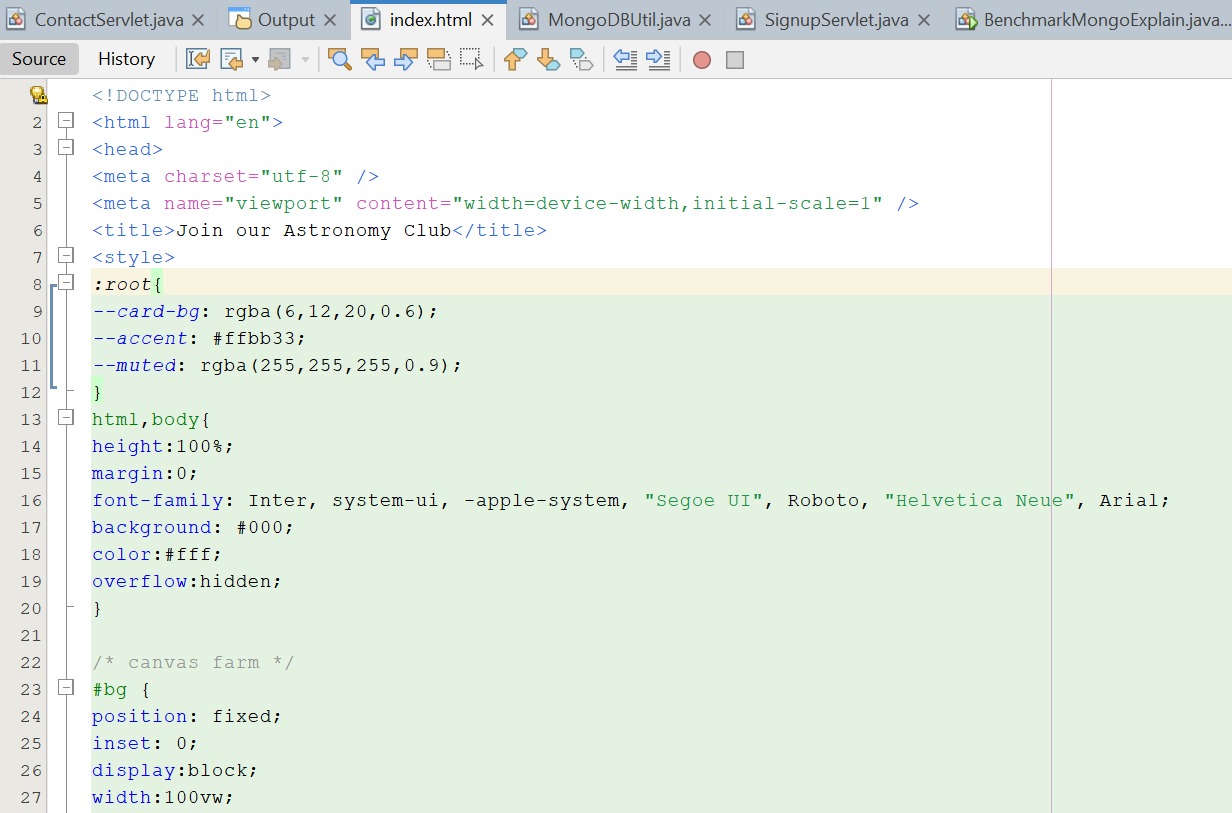
**Concurrency Simulation:** Execute the full test battery under two conditions:

**Single-Threaded:** 1 Concurrent User (Baseline performance).

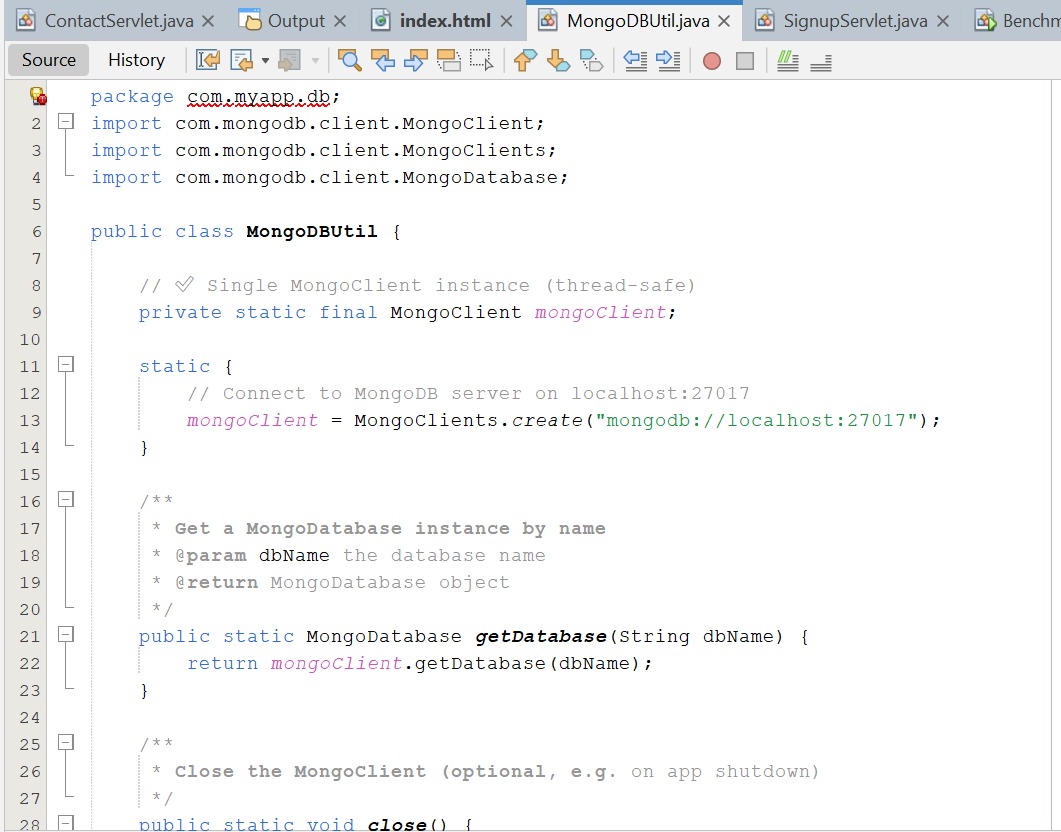
**Multi-Threaded:** 10 Concurrent Users (To simulate high-load and test system throughput).

1. **Code**

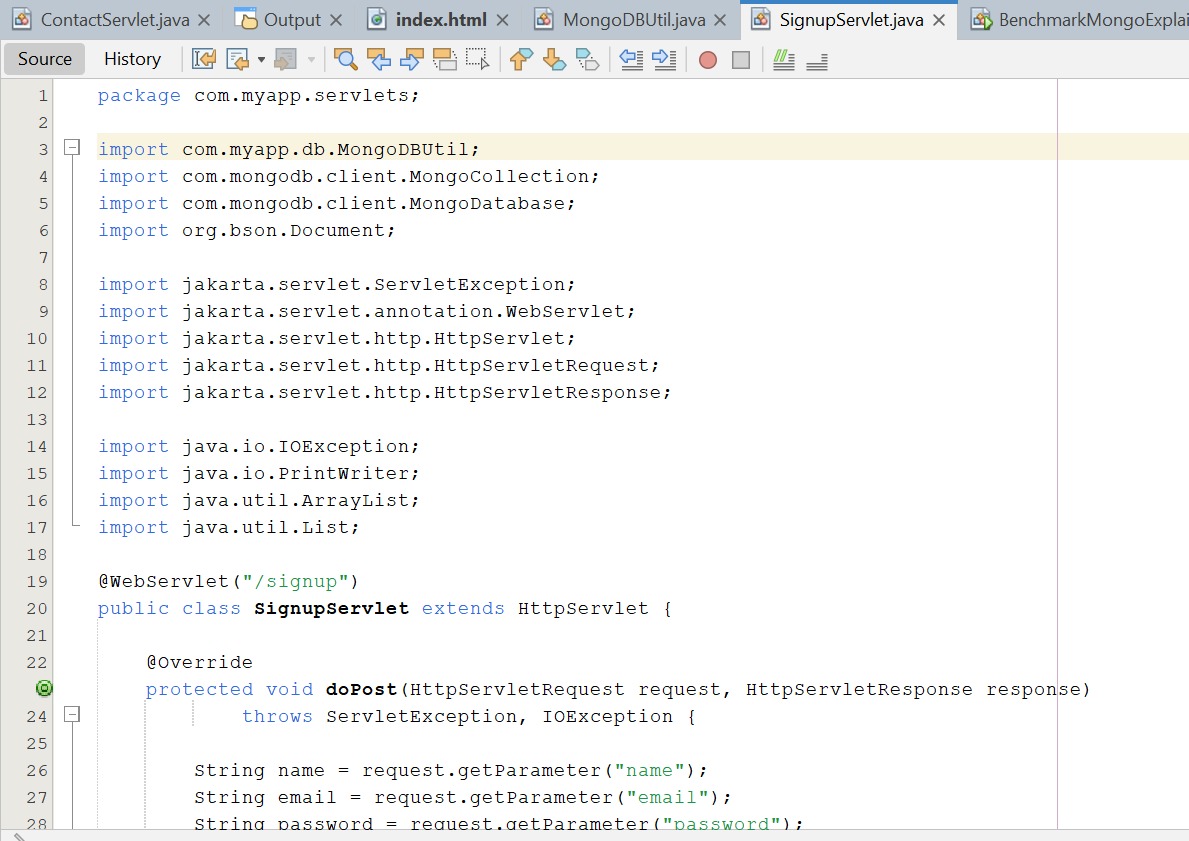
**index.html**

****

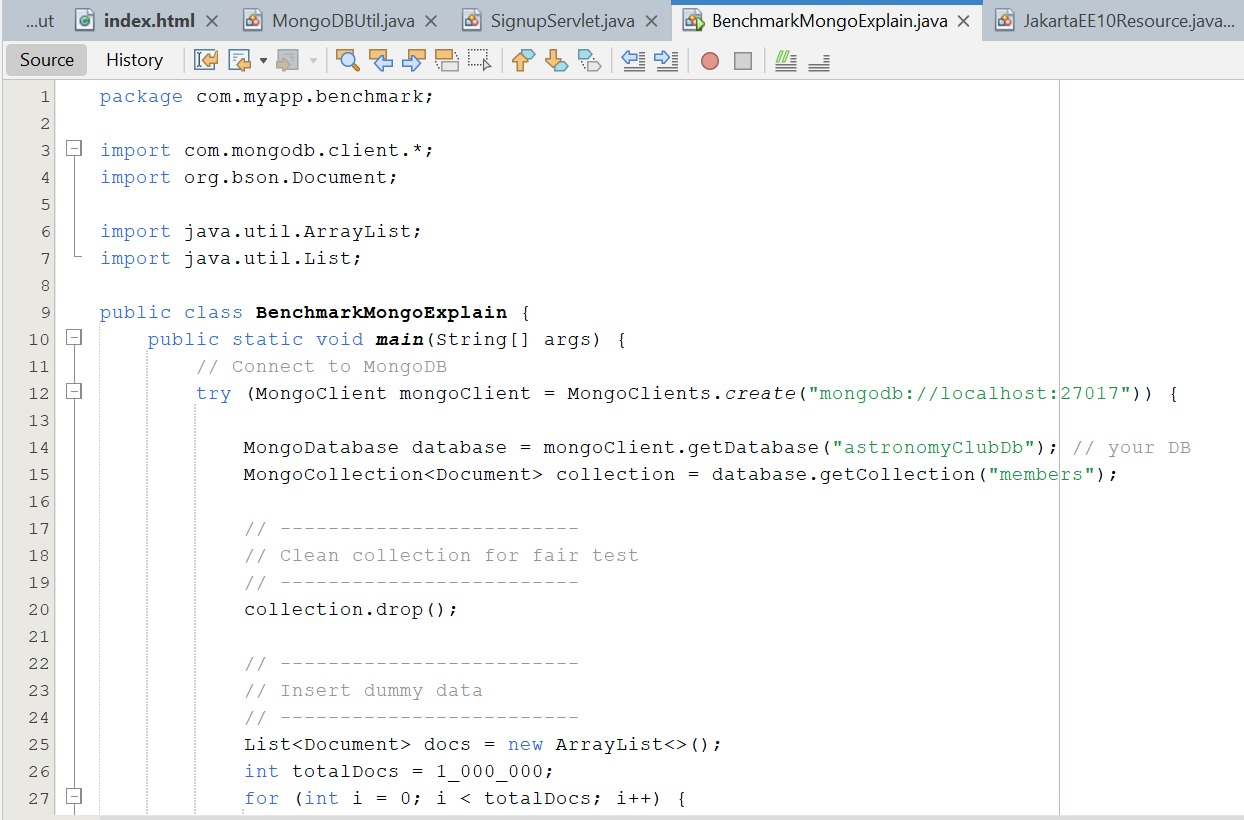
**MongoDbUtil.java**

****

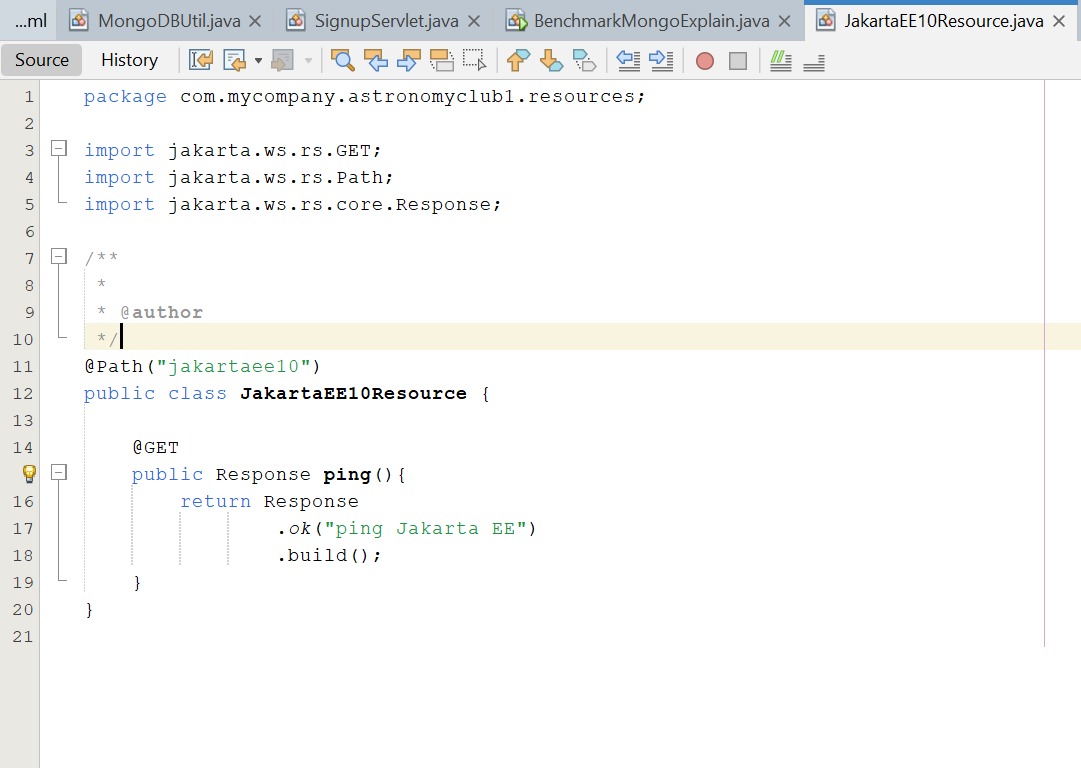
**SignupServlet.java**

****

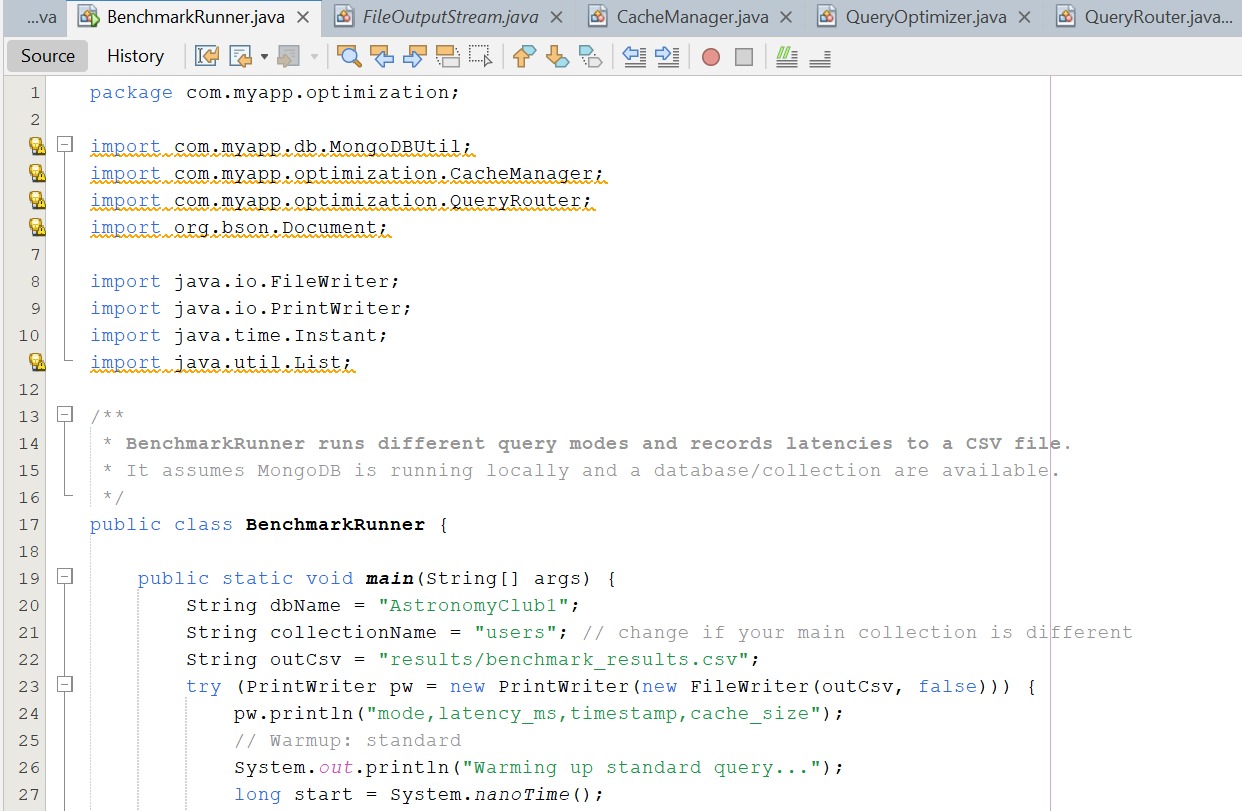
**BenchmarkMongoExplain.java**

****

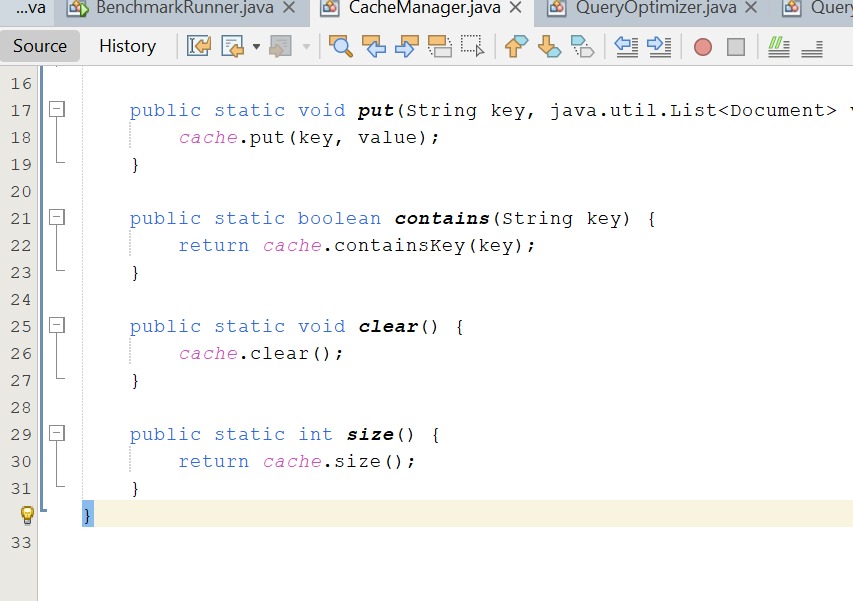
**JakartaEE10Resource.java**

****

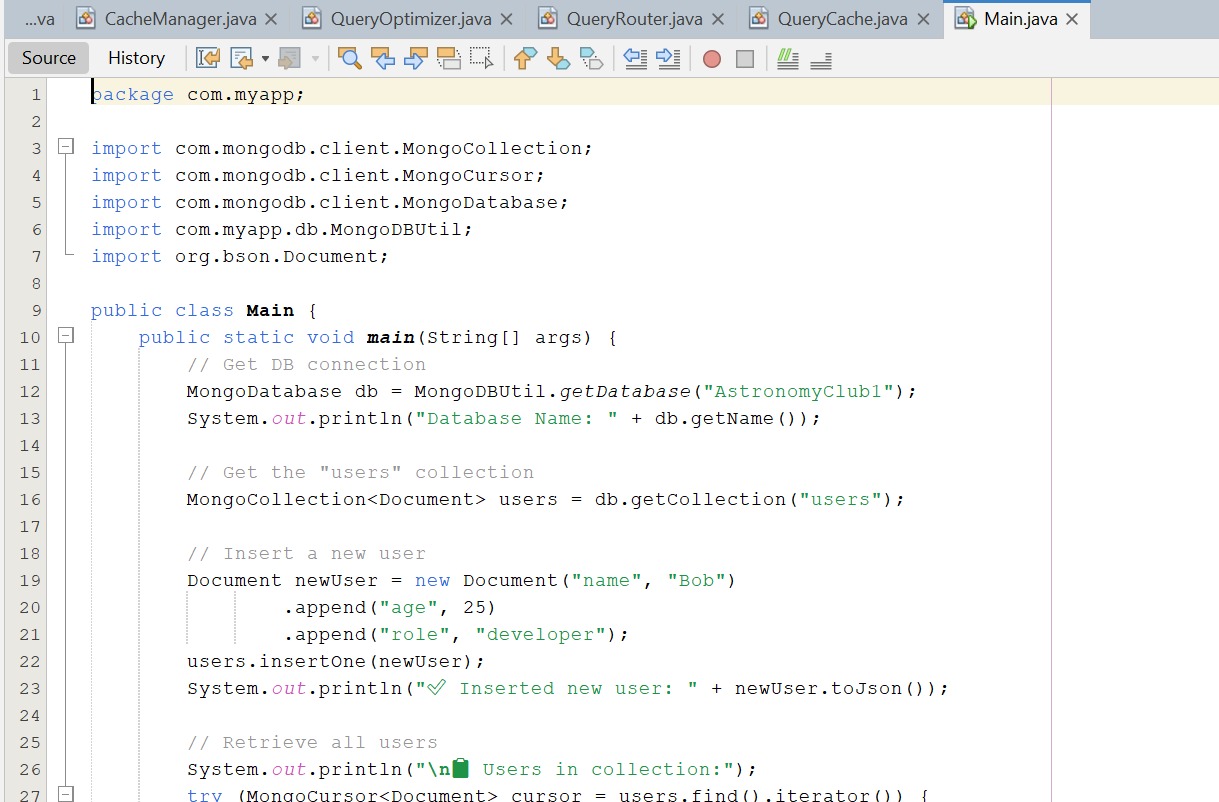
**BenchmarkRunner.java**

****

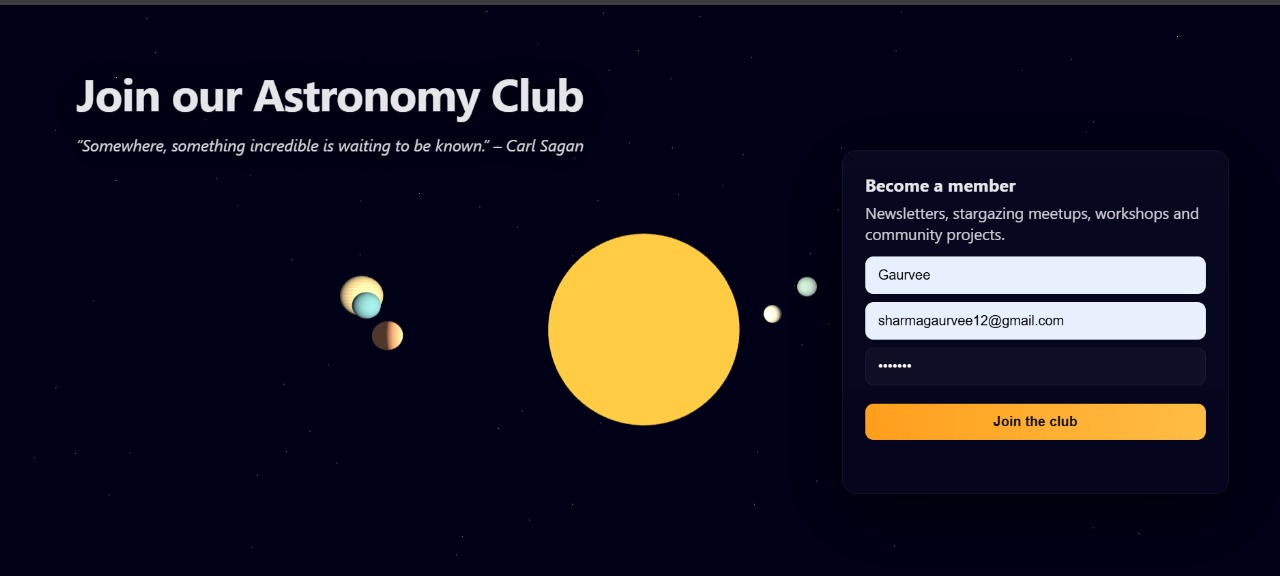
**CacheManager.java**

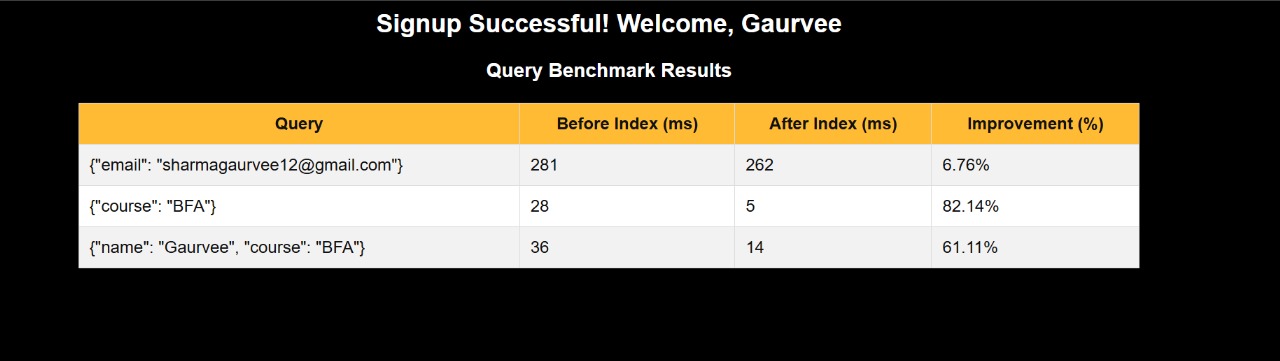
****

**Main.java**

****

## Result

****

****

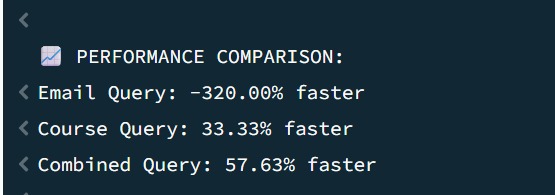
**Before optimization:**

****

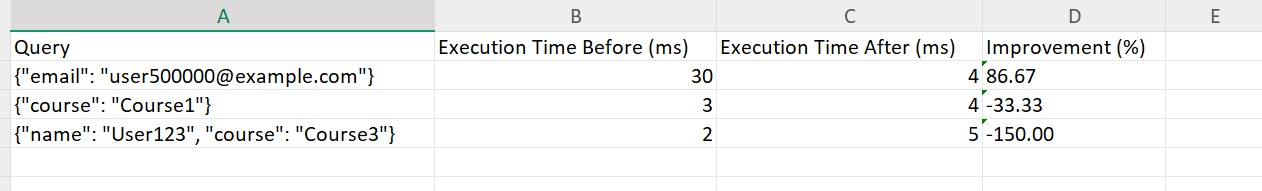
**After optimization:**

****

**Compare the performance of both:**

****

**Optimization done in dummy dataset**

****

1. **Future Scope**

**Integration with a Distributed Database Environment:**

The current setup works on a single MongoDB instance. In the future, it can be expanded into a sharded or replica set architecture to test query performance across distributed nodes. This would simulate real-world scalability and improve data availability and fault tolerance.

**Automated Benchmarking Dashboard:**

The benchmarking process can be automated through a web-based dashboard built using AJAX and JavaScript frameworks. This would allow real-time performance visualization with interactive charts showing query latency, throughput, and index utilization.

**Implementation of Caching Mechanisms:**

Adding a caching layer (e.g., Redis or in-memory cache) could further optimize query performance by reducing database load and improving response times for frequently accessed data.

**Advanced Query Optimization Techniques:**

Future versions could analyze query execution plans automatically using machine learning to recommend the best indexes or query rewrites for specific workloads.

**User Authentication & Role Management:**

The application can include advanced security features using JWT-based role authorization to separate access for students, admins, and researchers.

**Cloud Deployment & Scalability:**

Deploying the MongoDB-based application on cloud platforms such as AWS, Google Cloud, or Azure would enable large-scale testing under different traffic loads, helping measure performance under distributed conditions.

**Data Visualization and Reporting Tools:**

Integration with BI tools or custom analytics modules can make performance reports easier to interpret for non-technical stakeholders.

**Support for Multiple Database Systems:**

Extending the system to benchmark other NoSQL and SQL databases (like Cassandra, PostgreSQL, or MySQL) can provide comparative insights into query optimization across technologies.

1. **Conclusion**

Optimization of Query Performance in Distributed Databases” successfully demonstrates how performance tuning and optimization techniques can significantly enhance the efficiency of data retrieval in distributed systems. By using MongoDB as the distributed database platform, the study explored how features such as indexing, sharding, query optimization, and caching impact query execution time and system scalability. Through the use of Java programming, different query scenarios were tested and benchmarked, while Excel was used to record, analyze, and visualize performance comparisons. The results showed that implementing the right optimization strategies can reduce query response time and improve resource utilization across distributed nodes.

This project provided valuable insights into the internal working of distributed databases and their query processing mechanisms. It also highlighted the importance of performance analysis and optimization in achieving faster and more reliable data access in large-scale applications.

Overall, the project strengthens understanding of database performance tuning, distributed data management, and real-time system efficiency — skills that are highly relevant in today’s big data and cloud computing environments.

1. **References**

**MongoDB Documentation.** (2024). Indexes and Query Optimization. Retrieved from <https://www.mongodb.com/docs/manual/indexes/>

**Elmasri, R., & Navathe, S. B.** (2016). Fundamentals of Database Systems (7th ed.). Pearson Education.

**Silberschatz, A., Korth, H. F., & Sudarshan, S.** (2020). Database System Concepts (7th ed.). McGraw-Hill Education.

**Stonebraker, M., & Cattell, R.** (2011). 10 Rules for Scalable Performance in “Simple Operation” Datastores. Communications of the ACM, 54(6), 72–80.

**Kumar, P., & Singh, A.** (2022). Query Optimization Techniques in Distributed Databases: A Comparative Study. International Journal of Computer Applications, 184(29), 12–18.

**MongoDB Inc.** (2023). Sharding and Distributed Data Processing. Retrieved from <https://www.mongodb.com/docs/manual/sharding/>

**Oracle.** (2022). Java Database Connectivity (JDBC) Guide. Retrieved from <https://docs.oracle.com/javase/tutorial/jdbc/>

**Microsoft Support.** (2023). Use Excel for Data Analysis and Performance Comparison. Retrieved from <https://support.microsoft.com/excel>

**Gupta, S., & Sharma, R.** (2021). Performance Optimization in NoSQL Databases Using Indexing and Caching Mechanisms. International Journal of Advanced Research in Computer Science, 12(5), 45–52.