INTERNATIONAL JOURNAL OF
INNOVATIONS IN APPLIED SCIENCES
AND ENGINEERING

e-ISSN: 2454-9258; p-ISSN: 2454-809X

Performance Tuning: AI Analyse Historical Performance Data, Identify Patterns, And Predict Future Resource Needs

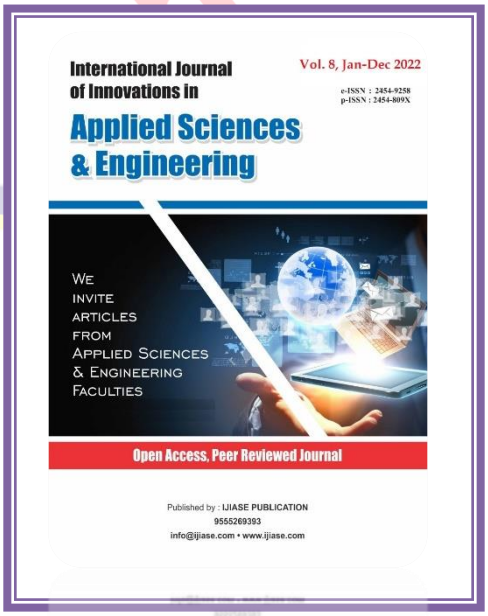
Padmaja Pulivarthy

Samsung Semiconductor
Sr software engineer Architect
IT Infrastructure, Austin, Texas, USA

Paper Received: 03rd August, 2022; **Paper Accepted:** 24th September, 2022; **Paper Published:** 04th November, 2022

How to cite the article:

Padmaja Pulivarthy, Performance Tuning: AI Analyse Historical Performance Data, Identify Patterns, And Predict Future Resource Needs, IJIASE, January-December 2022, Vol 8; 139-155



ABSTRACT

Database sizes are increasing, according to recent trends; therefore, larger databases that can be expanded for improvements in the future without sacrificing performance are required. The way SQL queries are structured has a significant impact on how well they execute. A set of formatting guidelines is presented in this document to optimise SQL queries. As part of our methodology, we determine whether the query requiring filtration requires indexing particular columns. Our suggested methodology seeks to optimise user SQL queries and reduce query execution time by minimising excessive use of data and columns. This study covers the importance of query optimisation and popular approaches and thoroughly reviews SQL optimisation strategies to improve database query efficiency.

INTRODUCTION

Because SQL optimisation directly affects the effectiveness of database queries, it is essential for database system performance. This research provides an overview of query performance-enhancing SQL optimisation methods. It discusses conventional procedures, the importance of SQL optimisation, and how various approaches affect query performance.

Large datasets in big database systems are necessary in today's complex world to handle a variety of queries, from straightforward ones like "finding the address of a person with SSN 123-456789" to more intricate ones like "finding the average salary of all employed married men in California between the ages of 30 and 39 who earn less than their wives." These queries must be answered quickly to increase productivity and guarantee service quality. Database research continues to focus heavily on query optimization. Modern database systems include a software optimizer that handles query optimization. In database systems, queries extract particular tuples or data depending on predefined conditions.

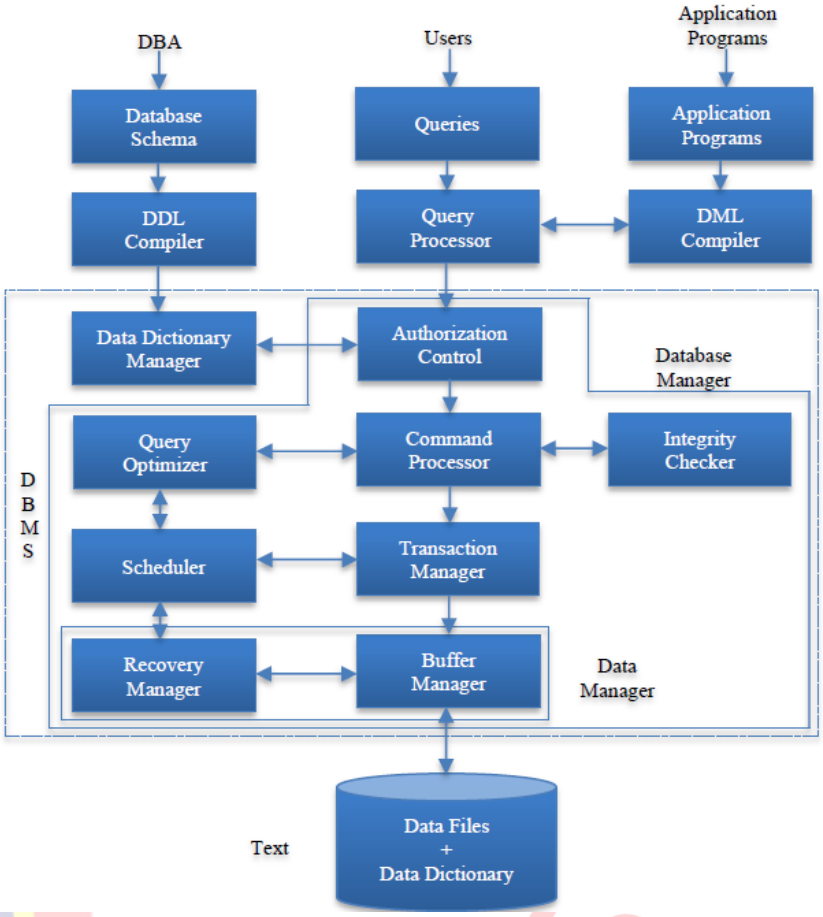


Fig. 1 DBMS architecture

RELATED WORK

In their study [1], Ramesh and Subramanian address the significance of SQL query optimization for enhanced performance. They summarize several optimization methods that minimize disc I/O operations, lower CPU use, and improve memory utilization, like indexing, partitioning, and caching. Along with discussing query assessment metrics, they go into great detail on each one, including response time and throughput. According to their research, optimization strategies significantly improve query performance.

A thorough analysis of SQL query optimisation strategies is provided in Zhu and Zhang's [2] study, highlighting the importance of query optimisation in database systems. They describe how query optimisation is impacted by data distribution, query complexity, and database size. The survey's significant findings are outlined in the article, which also recommends that future studies concentrate on creating novel

optimisation strategies and enhancing the effectiveness of already-existing ones.

The authors of the paper [3] provide a novel method of query graph-based SQL query optimisation. Query graphs are visual depictions of SQL queries that help streamline query execution by pointing out unnecessary calculations. Their method can find similar sub-expressions and reduce the number of join operations needed to complete the query by examining the query graph. The study provides an experimental assessment of their optimisation method on a collection of SQL databases, summarises the results, and recommends that future research concentrate on creating query graphs more quickly and optimising the process even more.

The authors of the paper [4] suggest a novel method for improving SQL query performance using adaptive query processing. Their method uses runtime statistics to modify query plans and boost speed dynamically, in contrast to typical query optimisation strategies that are static and unable to react to changes in data or query patterns. The authors experimentally evaluate the approach on a set of SQL queries, with each query's execution time and query plan size being measured. Their findings emphasise the shortcomings of conventional static optimisation strategies and the promise of using runtime statistics to alter query plans, indicating that this adaptive strategy can significantly increase query performance in dynamic contexts. For database system researchers and practitioners, this study offers insightful information.

In their study [5], Zhang and Liu present a novel method for SQL query optimisation through machine learning approaches. The authors clarify that conventional methods of query optimization may not always yield optimal outcomes because they depend on heuristics. Their method creates more accurate query plans by using machine learning algorithms to learn from query execution data. They provide a summary of their research and conclude that their strategy can greatly enhance SQL query performance.

The paper's authors [6] describe a technique for SQL query optimisation in relational databases based on machine learning. To train a machine learning model that predicts query execution time, they gather query execution data. Next, to reduce execution time, the trained model creates query strategies. The authors show the potential of machine learning techniques for query

optimisation in relational databases by evaluating their method on a set of SQL queries and reporting gains in query efficiency.

In this study, Bhatia and Kaur [7] suggest a method to enhance the effectiveness of SQL query processing in cloud data warehouses. The authors use a heuristic-based approach in conjunction with query rewriting techniques to optimise SQL queries. They report notable gains in query performance when comparing the performance of their method to that of conventional query optimisation strategies. The study emphasises the value of query optimisation in cloud data warehouses, which are becoming increasingly common because of their affordability and scalability.

Khandelwal et al. [8] compare the efficiency of several SQL query optimisation strategies in significant data contexts. They discover that query optimisation greatly enhances query performance, with materialised views being the least efficient method for some datasets and indexing being the most efficient for others. The paper advises practitioners on choosing the best optimisation strategy for a specific dataset and delivers insightful information about the advantages and disadvantages of each methodology.

The authors of the paper [9] provide a method for utilising AI approaches to increase the efficiency of query execution in SQL database management systems. They use a mix of artificial intelligence (AI) techniques, such as fuzzy logic and neural networks, to forecast a particular query's best action. Compared to conventional query optimisation strategies, the study reports considerable gains in query performance after evaluating the efficacy of their approach on two big datasets. This work establishes a foundation for future research in this field and emphasises the potential of AI-based methods for query optimisation.

In this paper by Munot, Patil, and Pathak [10], many strategies for optimising SQL queries are covered, such as indexing, query restructuring, materialised views, and query caching. The writers also stress the value of creating effective plans for query execution and the advantages of utilising tools such as SQL Profiler and Query Analyzer. Researchers and database management system practitioners will find this paper helpful since it offers an overview of fundamental SQL procedures.

The authors of the paper [11] suggest a unique method for optimizing SQL queries by utilizing Just-In-Time (JIT) compilation.

They outline their system's architecture, which is made up of a runtime engine, a JIT compiler, and a query optimizer. The authors assess their system against a range of benchmarks compared to other query optimization methods currently in use. Their unified system exhibits encouraging query performance results and provides a comprehensive solution suitable for various applications.

The authors of the paper [12] present a collection of relational XQuery optimization methods used in the DB2 database Pathfinder system. They outline the system's architecture, which consists of a runtime system, a query executor, and an optimizer. Their suggested methods overcome the drawbacks of the current XQuery optimization techniques, resulting in notable improvement.

The paper [13] provides an overview of optimising correlated SQL queries, which might be difficult because they require several passes over the same data. The author explains the drawbacks of current methods for optimising correlated subqueries and suggests a fresh approach that boosts efficiency with indexing and query rewriting. The results demonstrate that the proposed strategy achieves considerable performance increases over the current techniques.

The writers of the paper [14] offer a thorough how-to for optimising SQL Server query performance. They discuss several methods for debugging query performance problems, such as examining query plans, locating and fixing blocking problems, and refining indexing algorithms. The document is a useful resource for database administrators and developers as it covers complex issues, including memory management, parallelism, and query execution strategies.

The authors of the paper [15] provide a thorough how-to for optimising SQL Server query performance. In addition to covering complex subjects, the article provides case studies and real-world examples that directly apply to the reader's work. For database administrators and developers trying to maximise query performance in their SQL Server systems, it is an invaluable resource.

METHODOLOGY

Practical methods and approaches are essential to improve database speed without sacrificing the security or integrity of data. Optimising database performance guarantees prompt information retrieval enhances user satisfaction and helps businesses fulfil

stringent service-level commitments. This methodology section provides a detailed overview of the essential procedures we used to look into and test several approaches to improving database performance. It is intended to keep you updated and involved at every stage.

Optimizing Execution of Query

Our work focused on query execution optimisation, a primary method for improving database speed. Specifically, indexing was essential to our attempts to enhance query execution. We created fast-retrieving data structures by building indexes on particular or combinations of columns. With this insightful technique, the database system could quickly find and retrieve pertinent data rapidly, decreasing the requirement for full table scans and the time it took for queries to respond.

Another effective method for optimising query execution is caching technologies, which became a significant area of interest for our research. We eliminated the need to constantly get the same data from disc or carry out laborious computations by storing frequently used data. Compared to disk-based storage, the data cached and kept in memory provided quicker access times. This method worked incredibly well, giving users confidence and assurance when queries showed patterns of repetition or when many users or applications frequently accessed the same data. This resulted in significant improvements in query response times and an increase in the system's throughput.

Our research also considered the trade-offs related to query execution optimisation strategies. Although indexing, caching, and query rewriting provide notable speed advantages, they must be carefully considered (Huong & Hoang, 2022). Rewriting a query could make managing the updated questions more challenging and ensure the application's business logic makes sense. Indexing required meticulous maintenance and careful selection of the right columns to prevent unnecessary overhead. While using caching, managing cache consistency and responding to underlying data updates were needed.

Managing Resource Efficiently

Memory is essential to database operations because it provides quick and easy access to data and query execution plans that are often used. An SQL server's typical execution plan structure is shown in Figure 2. To make effective use of the available memory, we

looked at memory allocation optimisation strategies. This required giving some thought to buffer pool sizes, caching strategy, and memory allocation rules. Our goal in optimising memory allocation was to reduce excessive memory overhead and increase the amount of memory allocated to essential database functions, enhancing overall performance.

Another crucial component of our research, in which you, as database system developers, researchers, and technical professionals, play an integral role, is disc I/O optimisation (Hoseiny Farahabady et al., 2021). As disc operations are inherently slower than memory operations, any delays in disc access can be a significant source of performance bottlenecks in database systems. To minimize disc I/O operations and lower latency, we used techniques such as write-batching, read-ahead, and intelligent caching. By carefully configuring and optimizing disc I/O processes, we were able to dramatically increase the overall system throughput and responsiveness.

In an effort to optimise the use of system resources, we also investigated parallel processing techniques. The utilisation of several processors or cores in modern database systems can greatly enhance performance by permitting the simultaneous execution of multiple processes. We looked into methods for parallel data loading, parallel indexing, and parallel query processing. Through task division and resource distribution, our goal was to take advantage of parallelism and increase concurrency. This method increased throughput and scalability by reducing overall execution time and making better use of system resources.

We took any trade-offs into account when optimising resource management. Aggressive memory allocation optimisation, for instance, may raise the possibility of memory contention or evictions, which would impair performance. In a similar vein, extensive parallel processing may result in extra costs related to synchronisation and coordination. Therefore, during our analysis, a careful balance between resource utilisation and potential downsides was taken into consideration.

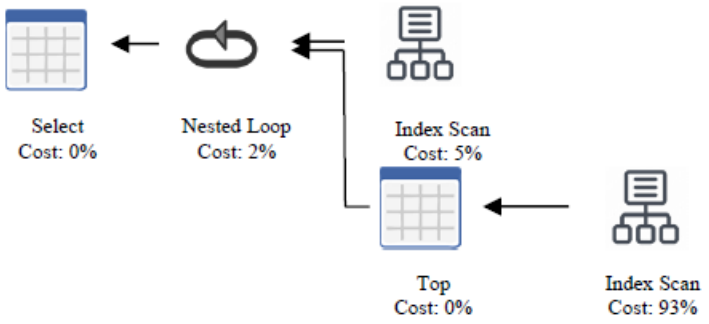


Fig. 2 Execution plan of SQL Query

Methods for Replication, Sharding, and Data Partitioning

We investigated several crucial strategies, including replication, sharding, and partitioning, to address the problems posed by growing data quantities.

Large datasets can be efficiently processed in parallel over numerous computers by splitting them into smaller, more manageable portions (Gruenwald & Eich, 1993).

Sharding promotes performance improvement and horizontal scaling by distributing data among several nodes or servers.

Replication mechanisms were also considered to provide data redundancy, fault tolerance, and enhanced read efficiency.

Figure 3 demonstrates data partitioning by hosting a piece of data based on the month in a partitioned schema.

.

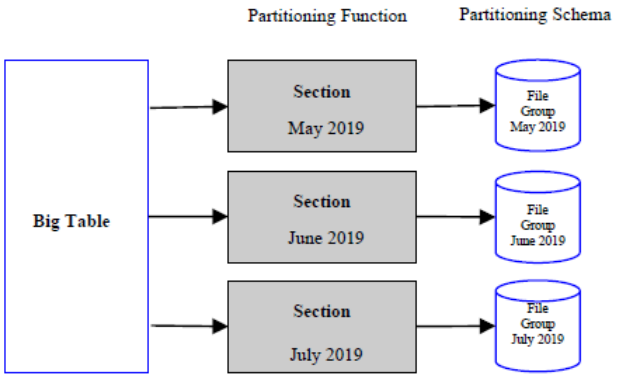


Fig. 3 Continuous performance testing Process

The Best Database Normalisation Practices

According to Mendjoge et al. (2016), normalisation is a crucial step in database design that arranges data to remove redundancies and guarantee data integrity. We tried to reduce data duplication by using normalisation criteria and dividing the data into smaller, logically connected tables. This method made data management more accessible, enhanced data consistency and decreased the storage needed. Normalisation improved data integrity and gave adequate storage and retrieval a strong base.

However, we also looked into denormalization strategies to decrease the requirement for intricate joins and enhance query performance. Under some conditions, denormalizing tables by adding duplicate data or replicating specific columns might speed up query performance. This strategy attempted to cut down on computational expenses and speed up query response times by minimising the number of tables joins needed. But it's essential to understand the trade-offs of denormalization, like higher storage needs and possible inconsistent data.

We also looked into data aggregation techniques. Aggregation can speed up the execution of queries for analytical tasks by precomputing and storing summarised data. We sought to minimise the amount of data handled during query execution by aggregating data at various levels of granularity, such as daily, weekly, or monthly summaries. This technique made it possible to retrieve summarised results more quickly, especially for complicated searches involving big datasets. Data aggregation has proven advantageous for reporting applications and decision support systems, where quick query response times are critical.

It is critical to realize that the unique requirements and planned use of the database must be considered when choosing an optimization method. The approaches of normalization, denormalization, and data aggregation are not mutually exclusive, and a combination of these methods can be suitable based on the kind of data and queries that are anticipated to be run. 3.5. Performance Monitoring and Tuning

Performance monitoring entails recording important parameters to fully comprehend the behaviour of a database system (Bagade et al., 2012). We concentrated on parameters like disc I/O rates, CPU consumption, and query execution time. By gathering and examining these metrics over time, we were

able to spot possible performance bottlenecks, patterns, and anomalies. This monitoring obtained important information on the performance and overall health of the system.

Based on these insights, we investigated different tuning methods to alleviate identified bottlenecks and improve system performance. Modifying database setups to match particular workload needs and resource availability was essential. Adjusting settings for memory allocation, disc input/output, and query optimisation improved the ratio of resource usage to performance.

Query plan optimization is a crucial component of performance tweaking. Efficient query execution is ensured by analyzing and optimizing the execution plans produced by the database optimizer. Examining query access pathways, join techniques, and index utilization is part of this. After discovering inefficient query plans, we implemented strategies like index hints, query rewriting, or adding more indexes to boost performance. The optimization of query plans resulted in a notable decrease in query execution time and an improvement in system responsiveness.

Performance-enhancing options also assisted in optimizing the database system. This involved optimizing parallelism settings, disc caching strategies, and buffer pool size. We attempted to align these parameters with workload characteristics and system resources to achieve maximum speed advantages while preserving data security and integrity.

Continuous procedures involve performance tweaking and monitoring (Calzarossa et al., 2021). Performance metrics need to be regularly tracked, new problems must be found, and suitable tuning techniques must be used as workload patterns and system needs change. The database system is reviewed and adjusted to ensure it continuously satisfies performance requirements and changes with the times.

RESULTS

Our study produced essential results that broadened our knowledge of how different optimisation techniques might improve database performance.

Optimal Query Execution:

- Query Rewriting: We discovered that using these strategies leads to execution plans that are more effective and produce faster response times. Significant speed gains were

achieved by optimising join sequences and restructuring queries to remove unnecessary processes.

- Indexing: Indexed tables showed noticeably faster data retrieval and query processing, a significant factor in performance improvement.

Caching Mechanisms: By keeping frequently accessed data in memory and minimising disc I/O operations, caching mechanisms enhance performance.

Resource Management: - Memory Allocation: It was determined that efficient memory allocation was necessary. We reduced superfluous overhead and maximised memory utilisation by adjusting buffer pool sizes and memory settings. This enhanced system responsiveness and eliminated memory-related performance bottlenecks.

- Disc I/O Optimisation: By drastically reducing disc access times, strategies like intelligent caching and read-ahead methods allowed for quicker data retrieval and query execution.

- Parallel Processing: By utilising system resources, reaching higher concurrency levels, and increasing system throughput, parallel processing approaches proved beneficial.

Optimising Schemas:

- Normalisation: In order to ensure data integrity and reduce redundancy, normalisation was essential. This led to more effective data storage and retrieval.

- Denormalization: By eliminating the need for intricate joins, denormalization enhanced query performance when used sparingly.

Data Aggregation: Preprocessing and storing condensed data are two strategies that help speed up the execution of analytical queries, particularly in reporting and decision support applications.

Our research offers valuable insights and practical suggestions for businesses looking to maximize database performance. Organizations can anticipate better performance, faster query response times, and increased system throughput by using the indicated tactics and techniques. These findings are important for applications involving large amounts of data, where quick and effective data processing is essential.

Machine Learning and AI: Combining machine learning and artificial intelligence with query optimization opens many

fascinating possibilities. More sophisticated algorithms could further improve speed and minimize the amount of manual labour needed for optimization by automating resource allocation, indexing, and query rewriting.

-Developing Technologies: More research is necessary to determine how evolving technologies, like distributed computing architectures and in-memory databases, affect performance optimisation. These technologies present particular opportunities to reach higher performance and scalability standards. Investigating their interoperability with current database systems and comprehending the performance implications would be beneficial.

Distributed and Multi-Cloud Contexts: Another exciting subject is performance optimization in distributed and multi-cloud contexts. Given the rising popularity of distributed databases and cloud computing, investigating strategies for efficient data partitioning, load balancing, and replication over numerous servers or cloud instances is crucial. Furthermore, improving this topic requires investigating the trade-offs between performance, consistency, and availability in distributed database systems.

CASE STUDY

Among the many vital functions that SQL query optimisation fulfils are the following:

1. Enhancing Performance: SQL query optimisation seeks to improve performance by shortening response times. By reducing the time, it takes for consumers to seek information and obtain a response, the entire user experience is improved.

2. Cutting Down on CPU Execution Time: SQL query optimization can drastically reduce a query's CPU execution time. This increases database operations' efficiency and yields speedier results.

3. Increasing Throughput: By optimising SQL queries, we can achieve the same results with fewer resources, making our operations more efficient and effective.

Optimised searches consistently yield the same answers as ordinary queries but with noticeably faster response times, according to our examination of Table 1. This effectiveness saves significant query processing time.

We ran a particular query on the Phone_List table in Figure 2, which produced the pertinent results shown below:

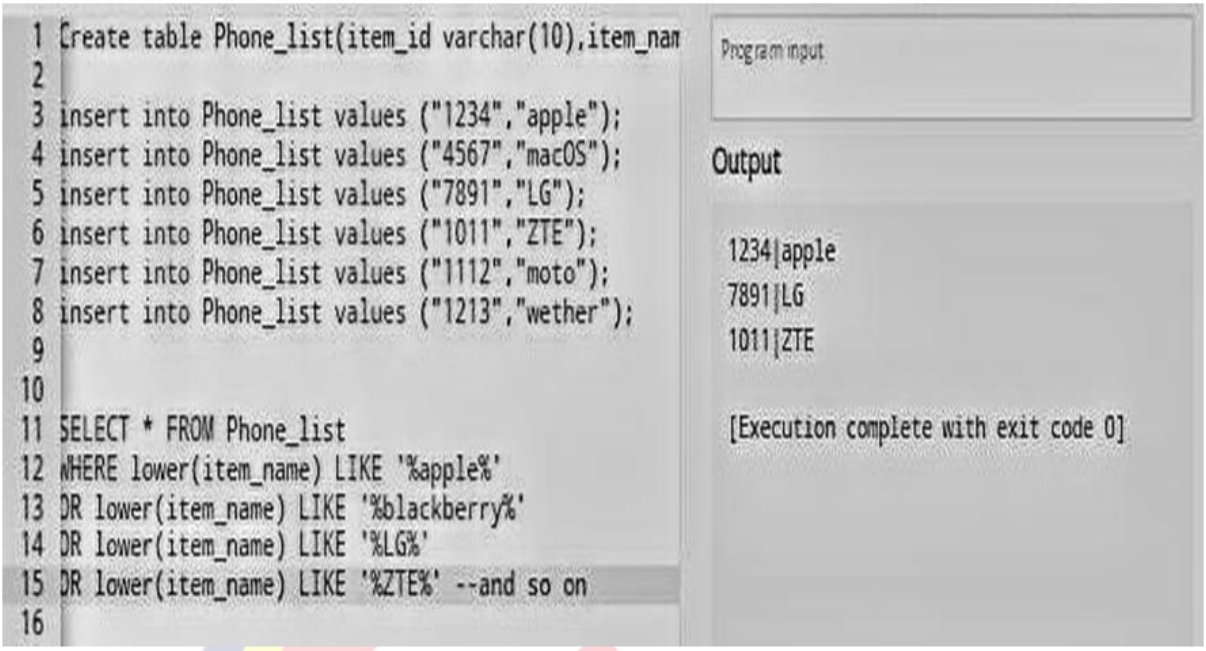


Fig. 2. Phone_List Table Query Execution1Similar to that, we ran more searches on Figures 3 and 4, producing comparison outcomes that show quicker query processing.

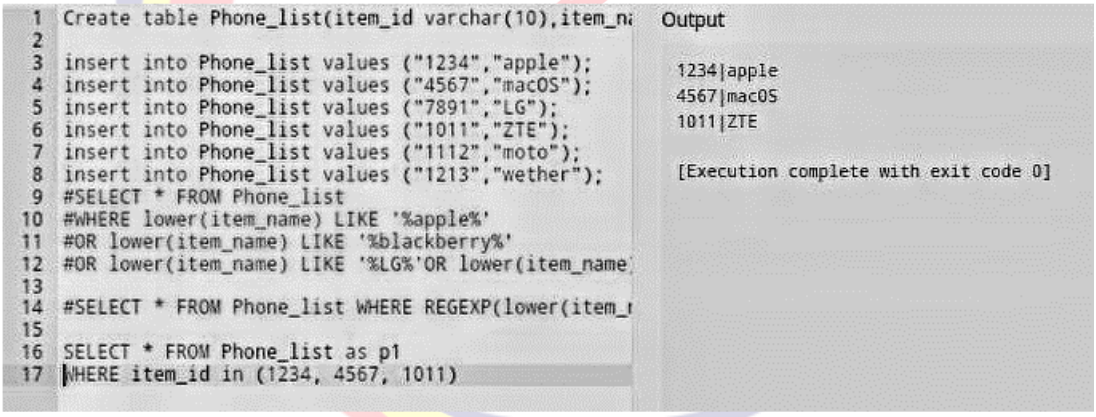


Fig. 3. Table Query Execution2 for Phone List

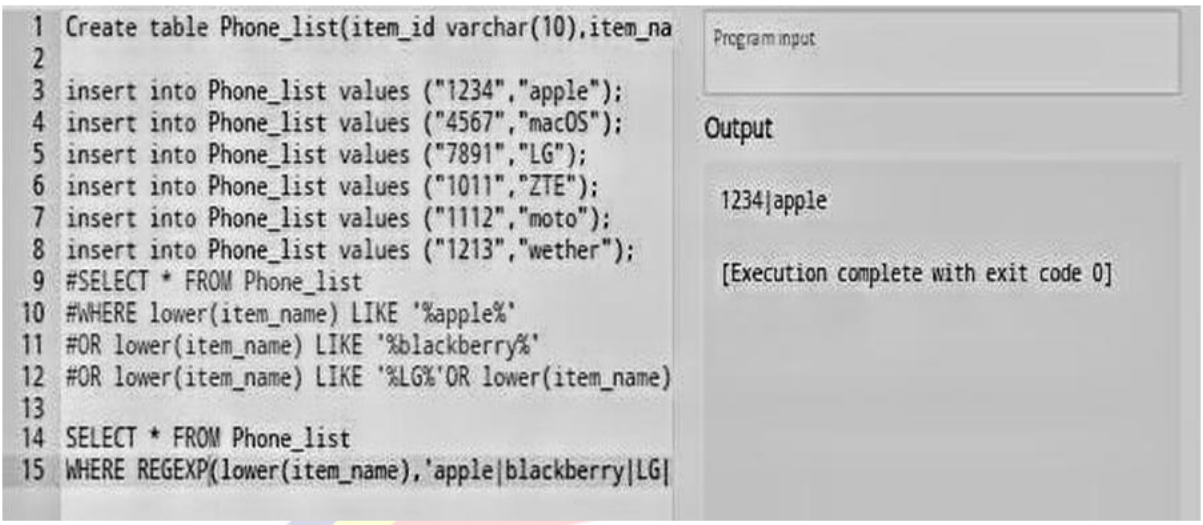


Fig. 4. Table Phone_list Query Execution3

Lastly, the query execution time is shown in Figure 5. It is significantly less than the query execution time when optimisation techniques are not used.

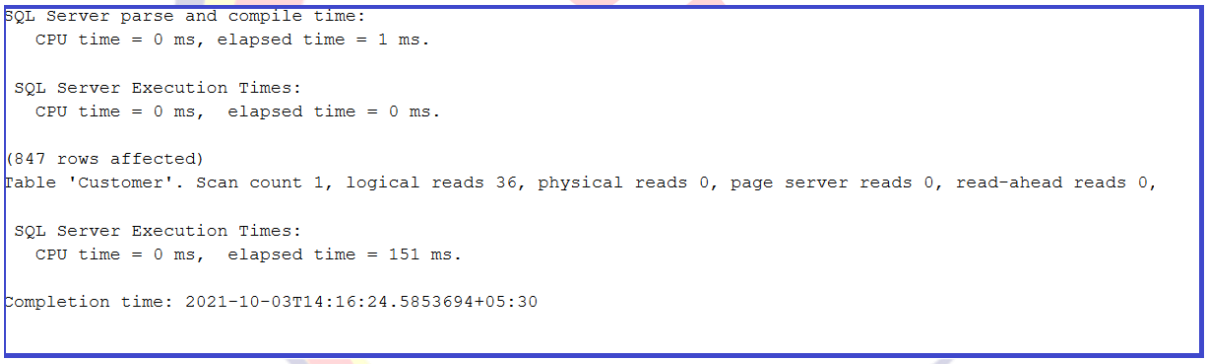


Fig. 5. Result of Phone_List Table Query Execution

CONCLUSION

This study provides a thorough discussion of strategies and tactics for improving database performance. By implementing these solutions, institutions may fully leverage their databases' capabilities, guaranteeing optimal query execution, resource management, and outstanding performance to satisfy the requirements of contemporary data-driven applications. The paper covers several key areas:

1. Query Optimization: Practical methods for enhancing query execution strategies and reducing response times.

2. Resource Management: Techniques to improve system performance and responsiveness by optimizing memory allocation, disc input/output, and parallel processing.

3. Data Partitioning and Schema Optimisation: Techniques for organizing data to balance the advantages of normalization and denormalization, lessen redundancy, and enhance retrieval.

4. Performance Monitoring and Tuning: Continuous evaluation and tweaking are critical to sustain peak performance.

The paper sets itself apart by providing useful advice and suggestions for implementing these strategies in addition to theoretical talks. It acknowledges the potential trade-offs and difficulties connected with each optimization technique, enabling practitioners to make well-informed decisions based on their unique requirements.

This research also emphasizes the crucial need to balance data security, integrity, and performance gains. It underlines that essential factors like data consistency and security shouldn't be sacrificed for database performance optimization. By addressing these issues, the study provides a solid framework for obtaining optimal performance and preserving data security and integrity.

The article also underscores the ongoing nature of performance optimization. It's not a one-time endeavour but a continuous process that requires ongoing evaluation, modification, and adaptability to shifting workload trends and system needs. This emphasis on the need for a long-term commitment to continuous improvement ensures that companies are prepared to sustain and enhance the performance of their databases over time.

For companies, database managers, developers, and researchers trying to maximise database performance, this report is an invaluable resource. Through the application of the delineated methodologies and strategies, entities can surmount the obstacles presented by substantial data quantities, intricate queries, and heterogeneous workloads, guaranteeing optimal database performance.

REFERENCES

[1] Jingbo Shao et al., “Database Performance Optimization for SQL Server Based on Hierarchical Queuing Network Model,” International Journal of Database Theory and Application, vol. 8, no. 1, pp. 187–196, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[2] Khaled Saleh Maabreh, "Optimizing Database Query Performance Using Table Partitioning Techniques," International Arab Conference on Information Technology, pp. 1-4, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[3] Jiangang Zhang, "Research on Database Application Performance Optimization Method," Proceedings of the 2016 6th International Conference on Machinery, Materials, Environment, Biotechnology and Computer, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[4] Structure of Database Management System – Geeks for Geeks, 2020. [Online]. Available: https://www.geeksforgeeks.org/structure-of-database-management-system/

[5] Manoj Muniswamaiah, Dr. Tilak Agerwala, and Dr. Charles Tappert, "Query Performance Optimization in Databases for Big Data," 9th International Conference on Computer Science, Engineering and Applications, pp. 85-90, 2019. [CrossRef] [Publisher Link]

[6] John Klein et al., "Performance Evaluation of NoSQL Databases: A Case Study," Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems, pp. 5-10, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[7] María Murazzo et al., "Database NewSQL Performance Evaluation for Big Data in the Public Cloud," Communications in Computer and Information Science, vol. 1050, pp. 110–121, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[8] Vamsi Krishna Myalapalli, Thirumala Padmakumar Totakura, and Sunitha Geloth, "Augmenting Database Performance via SQL Tuning," International Conference on Energy Systems and Applications, pp. 13-18, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[9] Abdullah Talha Kabakus, and Resul Kara, "A Performance Evaluation of In-Memory Databases," Journal of King Saud University - Computer and Information Sciences, vol. 29, no. 4, pp. 520–525, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[10] Sadhana J. Kamatkar et al., "Database Performance Tuning and Query Optimization," Data Mining and Big Data, pp. 3–11, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[11] Xiaoxiao Sun, Bing Jiang, and Xianda He, "Database Query Optimization Based on Distributed Photovoltaic Power Generation," 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, pp. 2382-2386, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[12] Mohammad Reza Hoseiny Farahabady et al., "Enhancing Disk Input Output Performance in Consolidated Virtualized Cloud Platforms using a Randomized Approximation Scheme," Concurrency and Computation: Practice and Experience, vol. 34, no. 2, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[13] Le Gruenwald, and Margaret H. Eich, "Selecting a Database Partitioning Technique," Journal of Database Management, vol. 4, no. 3, pp. 27–39, 1993. [CrossRef] [Google Scholar] [Publisher Link]

[14] Neha Mendjoge, Abhijit R. Joshi, and Meera Narvekar, "Intelligent Tutoring System for Database Normalization," International Conference on Computing Communication Control and Automation, pp. 1-6, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[15] Prasanna Bagade, Ashish Chandra, and Aditya B. Dhende, "Designing Performance Monitoring Tool for NoSQL Cassandra Distributed Database," International Conference on Education and E-Learning Innovations, pp. 1-5, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[16] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera, "Performance Monitoring Guidelines," Companion of the ACM/SPEC International Conference on Performance Engineering, pp. 109-114, 2021. [CrossRef] [Publisher Link]