https://doi.org/10.31891/2307-5732-2026-365-18
УДК 004.652.1

ZAKHAROV YURII
Kharkiv national university of radioelectronics
https://orcid.org/0009-0005-3449-4664
e-mail: yurii.zakharov.cpe@nure.ua
GOLIAN NATALIYA
Kharkiv national university of radioelectronics
https://orcid.org/0000-0002-1390-3116
e-mail: natalia.golian@nure.ua

ANALYSIS OF THE IMPACT OF INDEXING ON SQL QUERY PERFORMANCE

This work explores methods of SQL queries performance improvement in the context of relational database management systems by systematically applying and evaluating contemporary indexing strategies. As data volumes expand rapidly and the complexity of both transactional and analytical workloads, ensuring efficient query execution has become a major challenge for modern databases. Indexes play a critical role in speeding up data retrieval, reducing disk input/output operations, and enhancing the overall scalability of database systems. However, poorly designed or excessive indexing may result in increased storage requirements, slower update operations, and additional maintenance burdens.

This study examines the effects of different index types—including B-tree, hash, bitmap, and composite indexes—on query performance across different dataset sizes and access patterns. The experiments were carried out in a controlled computing environment, maintaining consistent datasets, query sets, and execution conditions to ensure fair and reproducible results. Performance was measured using several key indicators, including execution duration, and the time required for index creation and maintenance, index storage footprint, and effects on data modification operations. Findings indicate that carefully planned indexing strategies can substantially boost query efficiency: hash indexes are highly effective for equality searches, B-tree and composite indexes show strong performance for range queries and multi-attribute filtering, and bitmap indexes are particularly suitable for analytical tasks involving low-cardinality columns.

Additionally, the study presents a practical methodology for choosing index structures depending on the features of the workload and the properties of the underlying data. By tailoring indexing approaches to specific query patterns, database systems can achieve notable improvements in speed, responsiveness, and scalability while keeping the overhead from maintaining indexes during write-heavy operations to a minimum. This guidance provides practical guidance for database administrators and developers seeking to enhance the performance of relational database systems in real-world scenarios.

Keywords: *database performance, indexing, query optimization, B-tree, hash, bitmap*

ЗАХАРОВ ЮРІЙ, ГОЛЯН НАТАЛІЯ
Харківський національний університет радіоелектроніки, Харків, Україна

АНАЛІЗ ВПЛИВУ ІНДЕКСАЦІЇ НА ПРОДУКТИВНІСТЬ SQL-ЗАПИТІВ

У роботі представлено підхід до підвищення продуктивності SQL-запитів у реляційних системах керування базами даних шляхом використання сучасних методів індексації. Це дослідження розглядає вплив різних типів індексів, включаючи B-дерево, хеш-індекси, растрові карти та композитні індекси, на продуктивність запитів для різних розмірів наборів даних та шаблонів доступу. Експерименти проводилися в контрольованому обчислювальному середовищі, підтримуючи узгодженість наборів даних, наборів запитів та умов виконання для забезпечення справедливих та відтворюваних результатів.

Ключові слова: *продуктивність бази даних, індексування, оптимізація запитів,, B-tree, hash, bitmap*

Стаття надійшла до редакції / Received 11.02.2026
Прийнята до друку / Accepted 11.03.2026
Опубліковано / Published 28.05.2026

Problem statement in a general form and its relationship with important scientific or practical tasks

Relational database systems still functions as a fundamental component of informational infrastructures, supporting a wide variety of applications, such as financial services, e-commerce environments, Internet of Things (IoT) ecosystems, and analytical platforms. As the volume of operated data continues to increase at a rapid pace, the efficiency of SQL query execution has become a critical factor influencing system responsiveness, effective resource management, and overall scalability. Poorly optimized queries can significantly lower application performance, increase operational expenses, and in certain situations prevent timely data processing within required performance constraints. Indexing is widely recognized as one of the most effective techniques for improving query efficiency, as it allows the database engine to access relevant data more directly and substantially lower the need for costly full table scans.

Well-designed indexing strategies can dramatically improve query execution speed while reducing the computational demands for the database system. However, improper indexing practices—such as the lack of necessary indexes, excessive index creation, or inefficient index design—can produce the opposite effect. These issues may lead to higher storage consumption, increased overhead during insert, update, and delete operations, and greater complexity in database administration and maintenance. The benefits provided by indexes depend on several significant factors, such as the size of the dataset, values distribution within indexed columns, query selectivity, and typical access patterns. Different index types, including B-tree, hash-based, block-range (BRIN), and bitmap indexes, are optimized for specific

query scenarios and workload characteristics. As a result, choosing the most appropriate indexing strategy requires careful consideration and remains a critical task in both academic research and practical database optimization.

In addition to improving query execution speed, indexes also influence other important aspects of database system performance, such as concurrency management, efficiency of transaction processing, and overall system reliability. In modern database environments, where systems must handle a combination of transactional and analytical workloads simultaneously, inefficient indexing decisions can gradually accumulate and negatively impact general system performance. Even relatively small inefficiencies in index usage can lead to measurable slowdowns when queries are executed frequently or when working with large-scale datasets. Therefore, gaining a clear understanding of how different index types perform under monitored experimental conditions and across different data volumes is essential for predicting their effectiveness in real-world applications. This study provides a structured and reproducible experimental evaluation of multiple indexing approaches, allowing for a detailed comparison of their performance characteristics and their suitability for practical implementation in relational database systems.

Analysis of research and publications

Database indexing research has progressed in response to the increasing data volumes, diverse workloads, and modern hardware architectures. Indexing remains crucial for reducing query execution time, with effectiveness depending on alignment between index structures, data distribution, and hardware characteristics [1]. Comparative studies show no single index type is optimal: tree-based indexes excel for selective queries, while bitmap indexes suit low-cardinality analytical workloads [2].

Automatic and adaptive indexing mechanisms dynamically recommend or create indexes based on workload monitoring, reducing manual tuning, though often tied to commercial systems [3]. Surveys highlight trade-offs among query speed, storage, and update costs [4], while vendor documentation and textbooks offer hands-on guidance for implementing indexes [5–8].

Although considerable research exists, only a limited number of studies conduct controlled experiments comparing fundamental index types under consistent conditions. This gap motivates the present study, aiming to evaluate multiple indexing strategies across data sizes and query patterns to provide practical performance insights.

Formulation of the objectives of the article

This study is designed to measure the influence of indexing on SQL query performance in relational databases through a number of experiments, with PostgreSQL selected as the primary platform for analysis. The research quantitatively analyzes how various index structures—including B-tree, hash, composite, covering, partial, and BRIN indexes—affect the execution efficiency of selection queries involving equality and range conditions, as well as operations for modification which include insert and update. Special emphasis is placed on comparing query performance in indexed and non-indexed scenarios under identical experimental conditions, ensuring that the specific contribution of each indexing technique to overall query efficiency can be accurately identified and measured. This approach allows for a precise assessment of how indexing alters execution behavior across different query types.

By developing a reproducible experimental framework and providing detailed empirical observations, the study aims to address limitations and gaps found in existing research. In particular, the work emphasizes systematic benchmarking of multiple index types within a unified and controlled environment, the generation of datasets with deterministic structure and correlated attribute distributions, and the use of high-resolution timing combined with statistical aggregation of repeated measurements. These methodological elements ensure consistency and reliability of the results, enabling objective comparisons between indexing strategies across varying dataset sizes and query patterns. As a result, the study allows for a clearer and more comprehensive understanding of both the strengths and weaknesses of different indexing approaches in realistic database scenarios.

Beyond measuring direct performance gains, the research also examines the broader impact of indexing on database schema design and workload optimization. By evaluating how query execution time, index construction cost, and storage overhead change as dataset size increases, the study identifies the point at which indexing shifts from being a performance enhancement to a fundamental requirement for maintaining acceptable system responsiveness. This analysis highlights the trade-offs between improved read performance and the additional costs associated with index creation and maintenance. The findings contribute to practical, evidence-based recommendations for selecting appropriate index types in PostgreSQL-based systems, particularly in environments where scalability, storage efficiency, and write performance must be carefully balanced to ensure long-term system stability and efficiency.

Presentation of the main material

The proposed methodology evaluates the impact of indexing on SQL query performance in PostgreSQL using a controlled and reproducible experimental setup. Test tables with sizes from 100,000 to 1,000,000 rows were generated deterministically, ensuring identical data distributions across all experiments. For each dataset, baseline performance was first measured without indexes, after which B-tree, hash, composite, covering, partial, and BRIN indexes were created and evaluated individually. Three classes of SQL operations were considered: exact-match selection, range-based selection, and data modification queries. Each query was executed repeatedly following a warm-up phase, and median execution times were recorded to reduce the influence of caching effects and transient system fluctuations. In addition to query performance, index build time and storage overhead were also measured.

The results indicate that indexing provides increasingly significant performance benefits as data volume grows. For exact-match queries, B-tree, composite, and covering indexes consistently achieved the highest acceleration, reducing

execution time by more than an order of magnitude even for smaller tables, and by several hundred times for datasets of one million rows.

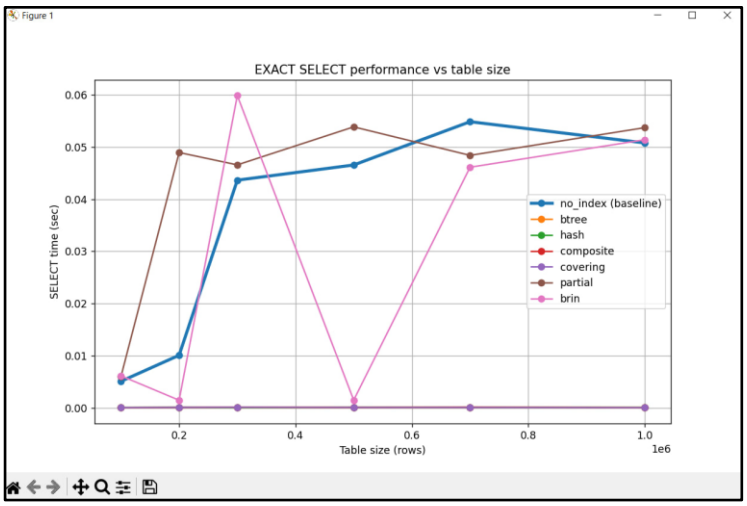

Fig. 1: Execution time of exact match queries for different index types across table sizes

For range-based queries, B-tree, composite, and covering indexes again demonstrated substantial improvements, particularly for larger datasets, whereas hash indexes showed little benefit due to their focus on equality lookups. BRIN indexes exhibited moderate gains only under certain conditions, reflecting their dependence on data ordering and correlation.

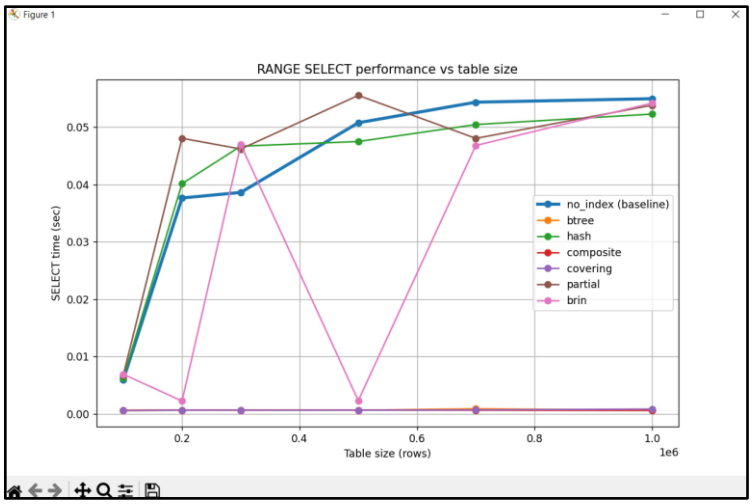

Fig. 2: Execution time of range queries for different index types across table sizes

Analysis of data modification operations revealed a clear trade-off between read performance and write overhead. Composite and covering indexes incurred the highest costs during insert and update operations, while hash and BRIN indexes introduced comparatively lower overhead.

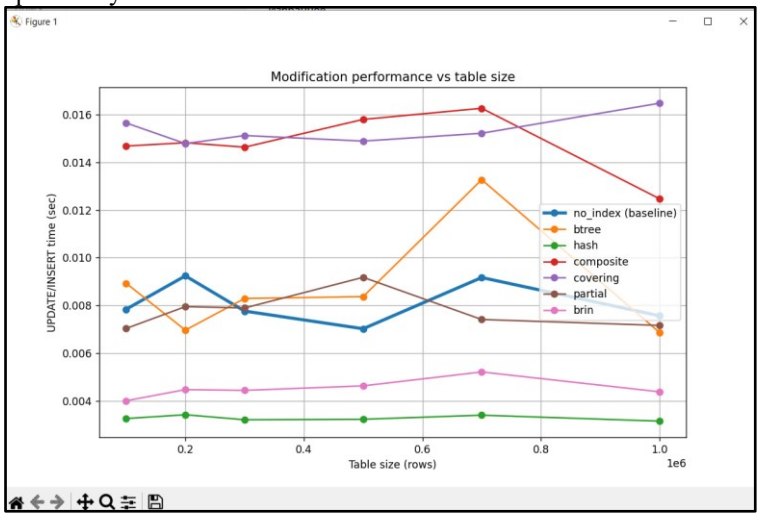

Fig. 3: Modification operation time with different index types

Additional measurements showed that hash and composite indexes require greater build time and storage space, especially for large tables, whereas BRIN indexes exhibit minimal construction cost and disk usage. However, this lower overhead does not necessarily translate into superior query performance, particularly for exact-match workloads.

Overall, the experiments confirm that no single index type is universally optimal. B-tree and covering indexes provide the best general performance for selective read workloads, hash indexes are most effective for equality-based access, and partial or BRIN indexes are beneficial only under specific data and query characteristics. These results emphasize the importance of selecting indexing strategies based on workload patterns, data volume, and performance priorities.

Conclusions of this study and prospects for further research in this area

This study shows that the proper use of indexing plays a critical role in improving SQL query performance, particularly when working with large-scale datasets. Experimental results confirm that different index structures influence execution efficiency in distinct ways, depending on the query type and workload characteristics. Among the evaluated approaches, B-tree, composite, and covering indexes delivered the most significant performance improvements for both exact-match and range-based queries. These benefits are primarily achieved by avoiding full table scans and enabling faster data access through well-organized index traversal mechanisms.

However, the findings also indicate that indexing effectiveness depends heavily on context, and not all index types provide universal advantages. For example, hash indexes show strong performance in equality-based searches but offer limited usefulness for range queries. Similarly, BRIN and partial indexes demonstrate effectiveness only under specific data distributions and access patterns. In some cases, particularly when applied to smaller tables or poorly matched query conditions, certain indexes performed similarly to non-indexed baselines. This highlights the importance of carefully aligning index design with query structure, data organization, and workload requirements to achieve optimal performance gains.

The experimental methodology used in this study—based on deterministic dataset generation, controlled execution environments, repeated testing, and median-based statistical analysis—proved to be reliable and reproducible. This structured approach enabled a thorough and objective evaluation of index efficiency, as well as associated costs such as index creation time and storage consumption. By maintaining consistent testing conditions, the study ensured accurate comparisons between different indexing strategies across varying table sizes and query patterns.

The results also points out the dependency between improved read performance and increased overhead during data modification operations. More advanced index types, such as composite and covering indexes, offer substantial benefits in read-heavy workloads but introduce additional costs in terms of storage usage and slower insert, update, and delete operations. In contrast, lightweight indexing methods like BRIN require less maintenance and consume less storage space, but typically provide smaller performance improvements and are suitable only for specific scenarios.

Overall, the study confirms that different indexing methods are optimal in different cases. Effective database optimization requires careful selection and, in many cases, a combination of multiple index types based on query behavior, workload intensity, and dataset size. The results reinforce the importance of workload-aware indexing strategies and highlight the need for balanced design decisions. Future work may extend this research by investigating more complex query structures, evaluating mixed read–write workloads, and exploring adaptive indexing techniques capable of dynamically adjusting to changing database usage patterns.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

1. Abbasi M., et al. Revisiting Database Indexing for Parallel and Accelerated Architectures // Information. –2024. –Vol. 15, No. 8:429. https://www.mdpi.com/2078-2489/15/8/429. Accessed: 10.11.2025.

2. Hecht R., et al. Tree-based Indexes versus Bitmap Indexes: A Performance Study // [Electronic resource]. – Available at: https://scispace.com/pdf/tree-based-indexes-versus-bitmap-indexes-a-performance-study-10hlbyeqxl.pdf. Accessed: 16.11.2025.

3. Chakkappen S., Kunjibettu S., McGreer D., Javidi M. Automatic Indexing in Oracle // Proceedings of the VLDB Endowment (PVLDB). –2025. –Vol. 18:4924. https://www.vldb.org/pvldb/vol18/p4924-chakkappen.pdf. Accessed: 12.11.2025.

4. Afridi S., et al. A Survey on Indexing Techniques for Big Data: Taxonomy and Performance Evaluation // [Electronic resource]. –2015. – Available at: https://www.researchgate.net/publication/273082158_A_survey_on_Indexing_Techniques_for_Big_Data_Taxonomy_and_Performance_Evaluation. Accessed: 14.11.2025.

5. SQLite Consortium. SQLite: Speed vs Other DBMS — Notes and Best Practices // [Electronic resource]. – Available at: https://sqlite.org/speed.html. Accessed: 17.11.2025.

6. Garcia-Molina H., Ullman J. D., Widom J. Database Systems: The Complete Book. –2nd ed. –Upper Saddle River: Pearson, 2009.

7. Elmasri R., Navathe S. B. Fundamentals of Database Systems. –6th ed. –Boston: Pearson Education, 2010.

8. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. –6th ed. –New York: McGraw-Hill, 2011.