JURTEKSI (Jurnal Teknologi dan Sistem Informasi) ISSN 2407-1811 (Print)
Vol. XII No 3, June 2026, hlm. 461 – 468 ISSN 2550-0201 (Online)
DOI: http://dx.doi.org/ 10.33330/jurteksi.v12i3.4653
Available online at http://jurnal.stmikroyal.ac.id/index.php/jurteksi

COMPARATIVE ANALYSIS OF B-TREE AND HASH INDEXES FOR POSTGRESQL QUERY OPTIMIZATION

Angga Ramdhani[1], Suprih Widodo[1*]
[1]Information Systems and Technology Education, Universitas Pendidikan Indonesia
email: *supri@upi.edu

Abstract: Query performance is a critical factor in managing large-scale databases. One of the most widely used optimization techniques is indexing. This study aims to analyze the impact of indexing on query performance in PostgreSQL, compare the effectiveness of B-Tree and Hash indexes, and evaluate their influence on query planner decisions. A quantitative experimental approach was employed using the TPC-H benchmark dataset at scale factors SF0.1, SF1, and SF10. Experiments were conducted using EXPLAIN ANALYZE on exact match, range, and join queries under three conditions: without indexing, with B-Tree indexing, and with Hash indexing. The results demonstrate that indexing significantly improves query performance. For exact match queries on the SF10 dataset, execution time decreased from 93.36 ms without indexing to 0.034 ms using B-Tree and 0.045 ms using Hash indexes. For join queries, execution time was reduced from 857.77 ms to 0.180 ms using B-Tree and 0.079 ms using Hash indexes. B-Tree showed consistent performance across different query types, while Hash achieved the best results for equality-based queries. Furthermore, index usage influenced query planner decisions in selecting more efficient execution strategies. These findings indicate that appropriate index selection can substantially improve data access efficiency in PostgreSQL.

Keywords: b-tree index; hash index; PostgreSQL; query optimization; query planner

Abstrak: Performa query merupakan faktor penting dalam pengelolaan basis data berskala besar. Salah satu teknik optimasi yang umum digunakan adalah indexing. Penelitian ini bertujuan menganalisis pengaruh penggunaan indexing terhadap performa query pada PostgreSQL, membandingkan efektivitas B-Tree dan Hash index, serta mengevaluasi pengaruhnya terhadap keputusan query planner. Penelitian menggunakan metode eksperimen kuantitatif dengan dataset benchmark TPC-H pada skala SF0.1, SF1, dan SF10. Pengujian dilakukan menggunakan EXPLAIN ANALYZE pada exact match query, range query, dan join query dalam kondisi tanpa index, menggunakan B-Tree index, dan Hash index. Hasil penelitian menunjukkan bahwa indexing meningkatkan performa query secara signifikan. Pada exact match query dataset SF10, execution time menurun dari 93,36 ms tanpa index menjadi 0,034 ms menggunakan B-Tree dan 0,045 ms menggunakan Hash index. Pada join query, execution time berkurang dari 857,77 ms menjadi 0,180 ms menggunakan B-Tree dan 0,079 ms menggunakan Hash index. B-Tree menunjukkan performa yang konsisten pada berbagai jenis query, sedangkan Hash index memberikan performa terbaik pada query berbasis equality. Selain itu, penggunaan index memengaruhi keputusan query planner dalam memilih strategi eksekusi yang lebih efisien. Hasil penelitian menunjukkan bahwa pemilihan metode indexing yang tepat dapat meningkatkan efisiensi akses data pada PostgreSQL

Kata kunci: b-tree index; hash index; optimasi query; PostgreSQL; *query planner*

INTRODUCTION

JURTEKSI (Jurnal Teknologi dan Sistem Informasi) ISSN 2407-1811 (Print)
Vol. XII No 3, June 2026, hlm. 461 – 468 ISSN 2550-0201 (Online)
DOI: http://dx.doi.org/ 10.33330/jurteksi.v12i3.4653
Available online at http://jurnal.stmikroyal.ac.id/index.php/jurteksi

The rapid growth of information technology and digital transformation has led to an unprecedented increase in data volume across various sectors. This condition requires database management systems capable of handling data efficiently, reliably, and at scale. One of the most widely used relational database management systems is PostgreSQL, which is recognized for its stability, open-source nature, and extensive support for query optimization features. PostgreSQL is a robust and secure open-source relational database management system that delivers reliable performance in a wide range of applications [1]. In large-scale data environments, query performance plays a crucial role in determining data access speed and overall system efficiency [2].

Query performance issues commonly arise when database systems process large volumes of data without adequate optimization mechanisms [3]. Queries involving data retrieval, filtering, and table joins may experience significant increases in execution time when the database performs sequential scans over the entire table [4]. Such conditions lead to higher computational resource consumption and reduced system efficiency [5].

One of the most widely adopted techniques for improving query performance is indexing. Indexes provide efficient access structures that enable faster data retrieval without scanning the entire table [6]. Among the various indexing methods available, B-Tree and Hash indexes are the most commonly used. B-Tree is the default indexing method in PostgreSQL and supports both exact-match and range queries due to its ordered structure [6]. In contrast, Hash indexes are specifically designed for equality comparisons and are theoretically more efficient for exact-match queries [7]. These differences indicate that the effectiveness of each indexing method depends on the characteristics of the query being executed [8].

Previous studies have demonstrated that indexing can significantly improve query performance compared to non-indexed databases [4], [9]. However, most existing research primarily focuses on execution time measurements without examining how different indexing methods influence query planner decisions in selecting execution strategies. Furthermore, comparative studies evaluating the performance of B-Tree and Hash indexes across different query types and data scales remain limited. This issue is particularly important because the effectiveness of an indexing method may vary as dataset size increases; an index that performs well on small datasets may not exhibit the same efficiency on larger datasets [10].

Therefore, this study aims to analyze the impact of indexing on query performance in PostgreSQL, compare the effectiveness of B-Tree and Hash indexes for exact-match, range, and join queries, and evaluate their scalability using the TPC-H benchmark dataset at scale factors SF0.1, SF1, and SF10. In addition, this study investigates how different indexing methods influence query planner decisions when selecting query execution strategies. This research contributes by providing a comparative analysis of B-Tree and Hash index performance across various query types, evaluating the impact of data growth on indexing performance, and offering insights into the relationship between indexing methods and query planner behavior in PostgreSQL.

METHOD

Research Approach

This study employed a quantita-

JURTEKSI (Jurnal Teknologi dan Sistem Informasi) ISSN 2407-1811 (Print)
Vol. XII No 3, June 2026, hlm. 461 – 468 ISSN 2550-0201 (Online)
DOI: http://dx.doi.org/ 10.33330/jurteksi.v12i3.4653
Available online at http://jurnal.stmikroyal.ac.id/index.php/jurteksi

tive experimental method with a comparative approach to analyze the impact of indexing on query performance and query planner decisions in PostgreSQL. The experimental method was selected because it enables direct measurement of query performance under controlled testing conditions. Experimental approaches have been widely used in database optimization studies to evaluate the effectiveness of indexing techniques and query execution strategies [9]. This research compares two indexing methods, namely B-Tree and Hash indexes, across different query types and data scales.

Dataset and Experimental Environment

This study utilized the TPC-H benchmark dataset, which is widely used for evaluating the performance of analytical database systems and query processing across different data scales [11]. Experiments were conducted using three dataset scales: Scale Factor (SF) 0.1, SF1, and SF10, representing small, medium, and large datasets, respectively. PostgreSQL version 16.13 was used as the database management system. Query performance was analyzed using the EXPLAIN ANALYZE command to obtain information regarding execution time, planning time, and execution plans.

Experimental Scenarios

The experiments were conducted under three conditions: without indexing (No Index), using a B-Tree index, and using a Hash index. Each condition was evaluated using three query types:

1. Exact-match queries using the equality (=) operator.
2. Range queries using the BETWEEN operator.
3. Join queries between related tables based on relational attributes.

All queries were executed on the SF0.1, SF1, and SF10 datasets. Each experimental scenario was repeated five times. The first execution was used as a cache initialization phase (cold cache), while the remaining four executions were used to calculate the average execution time (warm cache). Repeated executions were performed to minimize result variations caused by caching mechanisms and to obtain more consistent benchmarking results.

Performance Metrics

The primary performance metrics analyzed in this study are presented in Table 1.

Table 1. Measurement Parameters

Metric	Description
Execution time	Query execution time
Planning Time	Time required by the query planner to determine an execution strategy
Scan Type	Type of scan used during query execution
Rows Removed	Number of scanned rows that were not used in the final result
Query Plan	Query execution strategy selected by PostgreSQL

The average query execution time was calculated using the following equation:

$$T = \frac{\sum_{i=1}^{N} T_i}{N} \quad (1)$$

Where:
T = average execution time

JURTEKSI (Jurnal Teknologi dan Sistem Informasi) ISSN 2407-1811 (Print)
Vol. XII No 3, June 2026, hlm. 461 – 468 ISSN 2550-0201 (Online)
DOI: http://dx.doi.org/ 10.33330/jurteksi.v12i3.4653
Available online at http://jurnal.stmikroyal.ac.id/index.php/jurteksi

T_i = execution time of the i-th experiment
N = total number of experimental runs

Query Planner Analysis

Query planner analysis was conducted using the output of EXPLAIN ANALYZE in PostgreSQL to identify query execution strategies based on the cost-based optimization approach [12], [13]. The analysis focused on the utilization of Sequential Scan, Parallel Sequential Scan, Index Scan, and Nested Loop Join.

Unlike previous studies that primarily focused on execution time measurements, this research also evaluates the influence of different indexing methods on query planner decisions when selecting execution strategies. This analysis provides a deeper understanding of the relationship between indexing techniques and query optimization effectiveness in PostgreSQL.

Data Analysis

The experimental results were analyzed using a comparative approach to evaluate query performance under different indexing conditions and dataset scales. The analysis focused on execution time, execution plans, and data access strategies selected by the query planner. In addition, scalability analysis was conducted to assess performance changes as the dataset size increased from SF0.1 to SF10. This comparative approach was employed to identify the relationship between indexing methods, query characteristics, and database system performance [14].

RESULTS AND DISCUSSION

Exact-Match Query Performance

Exact-match query experiments were conducted to evaluate the performance of equality-based data retrieval under three conditions: without indexing, using a B-Tree index, and using a Hash index. Table 2 presents the average execution time of exact-match queries for each experimental scenario.

Table 2. Average Execution Time of Exact-Match Queries (ms)

Dataset	No Index	B-Tree	Hash
SF0.1	5.77	0.25	0.10
SF1	14.92	0.046	0.033
SF10	93.36	0.034	0.045

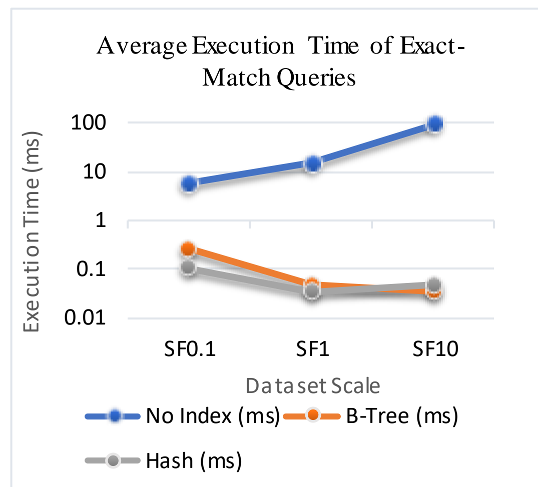

Figure 1. Average Execution Time of Exact-Match Queries

Based on Table 2 and Figure 1, the use of indexes significantly improved query performance compared to the no-index scenario. On the SF10 dataset, execution time decreased from 93.36 ms to 0.034 ms using a B-Tree index and 0.045 ms using a Hash index. These results indicate that both indexing methods substantially reduce data access costs through the use of Index Scan operations.

JURTEKSI (Jurnal Teknologi dan Sistem Informasi) ISSN 2407-1811 (Print)
Vol. XII No 3, June 2026, hlm. 461 – 468 ISSN 2550-0201 (Online)
DOI: http://dx.doi.org/ 10.33330/jurteksi.v12i3.4653
Available online at http://jurnal.stmikroyal.ac.id/index.php/jurteksi

Hash indexes demonstrated sligh tly better performance on the SF0.1 and SF1 datasets, whereas both indexing methods exhibited comparable performa nce on the SF10 dataset. These findings suggest that both B-Tree and Hash indexes are highly effective for exact-match queries, with Hash indexes generally providing advantages for equality-based search operations due to their hash-based lookup mechanism.

Range Query Performance

Range query experiments were conducted using the BETWEEN operator to evaluate the effectiveness of each indexing method in handling range-based data retrieval. Table 3 presents the average execution time of range queries for each experimental scenario.

Table 3. Average Execution Time of Range Queries (ms)

Dataset	No Index	B-Tree	Hash
SF0.1	3.46	3.46	4.01
SF1	18.32	0.335	21.52
SF10	55.77	0.172	56.62

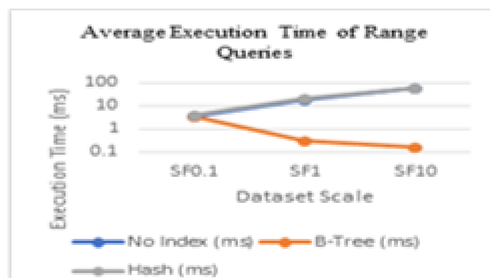

Figure 2. Comparison of Average Execution Time for Range Queries

Based on Table 3 and Figure 2, the results indicate that B-Tree is the most effective indexing method for range queries. On the SF10 dataset, execution time decreased from 55.77 ms to 0.172 ms after applying a B-Tree index. In contrast, the Hash index did not provide a significant performance improvement, as PostgreSQL continued to utilize a Sequential Scan strategy.

This behavior occurs because Hash indexes do not maintain data in a sorted order, making them unsuitable for efficient range-based searches. Consequently, PostgreSQL cannot effectively utilize Hash indexes for queries involving the BETWEEN operator. These findings highlight the superiority of B-Tree indexes for range queries and demonstrate their flexibility in supporting different query workloads.

Join Query Performance

Join query experiments were conducted to analyze the impact of indexing on table join operations. Table 4 presents the average execution time of join queries for each experimental scenario.

Table 4. Average Execution Time of Join Queries (ms)

Dataset	No Index	B-Tree	Hash
SF0.1	21.36	0.13	0.16
SF1	122.93	0.136	0.136
SF10	857.77	0.18	0.079

Figure 3. Comparison of Average Execution Time for Join Queries

Based on Table 4 and Figure 3, the use of indexes resulted in a substantial improvement in query performance compared to the no-index scenario. On the SF10 dataset, execution time decreased from 857.77 ms to 0.180 ms using a B-Tree index and 0.079 ms using a

JURTEKSI (Jurnal Teknologi dan Sistem Informasi) ISSN 2407-1811 (Print)
Vol. XII No 3, June 2026, hlm. 461 – 468 ISSN 2550-0201 (Online)
DOI: http://dx.doi.org/ 10.33330/jurteksi.v12i3.4653
Available online at http://jurnal.stmikroyal.ac.id/index.php/jurteksi

Hash index.The results indicate that Hash indexes achieved the best performance for equality-based join queries. This advantage can be attributed to the hashing mechanism, which enables faster lookup operations than tree traversal. Furthermore, index utilization reduced the number of rows processed during query execution, thereby improving the overall efficiency of join operations. These findings demonstrate the importance of appropriate index selection in optimizing join query performance, particularly when processing large-scale datasets.

Query Planner Behavior

The output of EXPLAIN ANALYZE was examined to identify the query execution strategies selected by PostgreSQL under different experimental conditions.

Table 5. Query Planner Decision Patterns

Condition	Query Planner Strategy
No Index	Sequential Scan/Parallel Seq Scan
B-Tree Exact Match	Index Scan
B-Tree Range Query	Index Scan
Hash Exact-Match Query	Index Scan
Hash Range query	Sequential Scan
Join Query With Indexes	Nested Loop + Index Scan

The results indicate that PostgreSQL adaptively selects execution strategies based on query characteristics and index availability. Under the no-index condition, PostgreSQL relied on Sequential Scan or Parallel Sequential Scan, resulting in increased execution time as the dataset size grew. When B-Tree indexes were applied, the query planner consistently selected Index Scan for both exact-match and range queries. In contrast, Hash indexes were utilized only for equality-based operations, while range queries continued to use Sequential Scan because Hash indexes do not support ordered data access. For join queries, PostgreSQL selected a combination of Nested Loop and Index Scan, which significantly improved data access efficiency. These findings show that the execution strategy chosen by PostgreSQL is closely related to both query characteristics and the indexing method applied.

Discussion

The results demonstrate that indexing significantly improves query performance compared to the no-index condition. As the dataset size increased from SF0.1 to SF10, execution time in the absence of indexes increased substantially, whereas both B-Tree and Hash indexes maintained consistently low execution times. These findings indicate that indexing provides good scalability in maintaining data access efficiency as data volume grows.

B-Tree proved to be the most versatile indexing method, delivering performance improvements for exact-match, range, and join queries. Its ordered data structure enables PostgreSQL to efficiently utilize Index Scan operations across various query patterns [8]. In contrast, Hash indexes achieved the best performance for exact-match and equality-based join queries but were ineffective for range queries because they do not support ordered data access [15].

In addition to the indexing method, the results also indicate that query optimization effectiveness is influenced by the execution strategies selected by the query planner. The query planner consistently utilized Index Scan when suitable indexes were available and re-

JURTEKSI (Jurnal Teknologi dan Sistem Informasi) ISSN 2407-1811 (Print)
Vol. XII No 3, June 2026, hlm. 461 – 468 ISSN 2550-0201 (Online)
DOI: http://dx.doi.org/ 10.33330/jurteksi.v12i3.4653
Available online at http://jurnal.stmikroyal.ac.id/index.php/jurteksi

verted to Sequential Scan when an indexing method could not efficiently support a particular query type. Therefore, query performance improvements depend not only on the presence of indexes but also on PostgreSQL's ability to utilize those indexes effectively through its cost-based optimization mechanism.

CONCLUSION

This study demonstrates that indexing significantly improves query performance in PostgreSQL compared to the no-index condition. B-Tree proved to be the most versatile indexing method, providing substantial performance improvements for exact-match, range, and join queries. In contrast, Hash indexes achieved the best performance for exact-match and equality-based join queries but were ineffective for range queries. The results also indicate that the indexing method influences query planner decisions in selecting query execution strategies.

The main contribution of this study is the comparative analysis of B-Tree and Hash index performance across different query types and data scales, as well as the evaluation of their impact on query planner behavior in PostgreSQL. Future research may extend this work by investigating other indexing methods, such as BRIN, GiST, and GIN, and by evaluating their performance using larger datasets and more complex workloads.

BIBLIOGRAPHY

[1] S. V. Salunke and A. Ouda, “A Performance Benchmark for the PostgreSQL and MySQL Databases,” *Future Internet*, vol. 16, no. 10, pp. 1–22, 2024, doi: 10.3390/fi16100382.

[2] D. Zolotukhina, “Comparative analysis of indexing strategies in PostgreSQL under various load scenarios,” *Программные систе мы и вычислительные методы*, no. 1, pp. 21–31, Jan. 2025, doi: 10.7256/2454-0714.2025.1.73138.

[3] S. Huang, Y. Qin, X. Zhang, Y. Tu, Z. Li, and B. Cui, “Survey on performance optimization for database systems,” *Science China Information Sciences*, vol. 66, no. 2, p. 121102, Feb. 2023, doi: 10.1007/s11432-021-3578-6.

[4] V. B. Ramu, “Optimizing Data base Performance: Strategies for Efficient Query Execution and Resource Utilization,” *Internatio nal Journal of Computer Trends and Technology*, vol. 71, no. 7, pp. 15–21, Jul. 2023, doi: 10.14445/22312803/ijctt-v71i7p103.

[5] K. Sirigiri, “Enhancing SQL Query Performance: A Case Study on Optimizing Enterprise Data Processing,” *International Journal of Basic and Applied Sciences*, vol. 14, no. 5, pp. 353–360, Sep. 2025, doi: 10.14419/x2wqqh31.

[6] A. Anchlia, “Enhancing Query Performance Through Relational Database Indexing,” *International Journal of Computer Trends and Technology*, vol. 72, no. 8, pp. 130–133, Aug. 2024, doi: 10.14445/22312803/IJCTT-V72I8P119.

[7] I. C. Saidu, M. Yusuf, F. C. Nemariyi, and A. C. George, “Indexing techniques and structu red queries for relational databases management systems,” *Journal of the Nigerian Society of Physical*

JURTEKSI (Jurnal Teknologi dan Sistem Informasi) ISSN 2407-1811 (Print)
Vol. XII No 3, June 2026, hlm. 461 – 468 ISSN 2550-0201 (Online)
DOI: http://dx.doi.org/ 10.33330/jurteksi.v12i3.4653
Available online at http://jurnal.stmikroyal.ac.id/index.php/jurteksi

Sciences, vol. 6, no. 4, Nov. 2024, doi: 10.46481/jnsps.2024.2155.

[8] M. Müller, L. Benson, and V. Leis, "B-Trees Are Back: Engineering Fast and Pageable Node Layouts," *Proceedings of the ACM on Management of Data*, vol. 3, no. 1, pp. 1–26, Feb. 2025, doi: 10.1145/3709664.

[9] M. Abbasi, M. V. Bernardo, P. Váz, J. Silva, and P. Martins, "Revisiting Database Indexing for Parallel and Accelerated Computing: A Comprehensive Study and Novel Approaches," *Information (Switzerland)*, vol. 15, no. 8, Aug. 2024, doi: 10.3390/info15080429.

[10] Kurniadi, V. A. Andrea, J. Panjaya, and K. Jingga, "Performance Comparison of Relational Database Management Systems for Processing Large Amount of Text Data," *Procedia Comput. Sci.*, vol. 269, pp. 150–160, 2025, doi: 10.1016/j.pro cs.2025.08.268.

[11] L. Qu *et al.*, "Are current benchmarks adequate to evaluate distributed transactional databa ses?," *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 2, no. 1, p. 100031, Mar. 2022, doi: 10.1016/j.tbench.2022.100031.

[12] H. Lan, Z. Bao, and Y. Peng, "A Survey on Advancing the DBMS Query Optimizer: Cardinality Estimation, Cost Model, and Plan Enumeration," *Data Sci. Eng.*, vol. 6, no. 1, pp. 86–101, Mar. 2021, doi: 10.1007/s41019-020-00149-7.

[13] J. R. Haritsa, "Robust Query Processing: A Survey," *Founda tions and Trends in Databases*, vol. 15, no. 1, pp. 1–114, Dec. 2024, doi: 10.1561/1900000089.

[14] D. Samoladas, C. Karras, A. Karras, L. Theodorakopoulos, and S. Sioutas, "Tree Data Structures and Efficient Indexing Techniques for Big Data Management: A Comprehensive Study," in *Procee dings of the 26th Pan-Hellenic Conference on Informatics*, New York, NY, USA: ACM, Nov. 2022, pp. 123–132. doi: 10.1145 /3575879.3575977.

[15] S. Nakazono, Y. Bessho, H. Kawashima, and T. Nakamori, "Griffin: Fast Transactional Data base Index with Hash and B+-Tree," in *2024 IEEE 20th International Conference on e-Science (e-Science)*, IEEE, Sep. 2024, pp. 1–10. doi: 10.1109/e-Science62913.2024.10678674.