جامعة الزعيم الازهري
المجلة العلمية المركزية

Improving PostgreSQL Database Performance for Large-Scale Data Management Using Indexing and Table Partitioning Techniques

Tayseer Awad Abd El Rahman Mohammed

Supervised by: Dr. Ammar Mohammed Abdo

Faculty of Graduate Studies, Department of Computer Sciences and Information Technology, University of Alzaiem Alazhari , Sudan

Abstract

The rapid growth of digital technologies has increased the demand for database systems capable of efficiently handling massive volumes of data. PostgreSQL is one of the most widely used open-source relational database management systems, offering performance-optimization techniques such as indexing and table partitioning; however, selecting the optimal strategy remains challenging for large datasets. This study evaluates the impact of indexing and table partitioning on PostgreSQL performance in a large-scale data management environment through an experimental study. An experimental database was created using PostgreSQL 17 containing 1,000,103 records to simulate a large-scale real-world information system, and four scenarios were tested: no optimization, indexing only, table partitioning only, and the combination of both. Query execution times and execution plans were analyzed using the EXPLAIN ANALYZE tool. The results show that indexing significantly improves search performance, while table partitioning enhances the processing of large datasets; combining both techniques achieves the highest overall performance by reducing query execution time and improving resource utilization. The study offers practical recommendations for database administrators and developers seeking to optimize PostgreSQL performance in large-scale data management environments.

Keywords— PostgreSQL; Large-Scale Data; database performance; indexing; table partitioning; query optimization; SQL.

I. Introduction

In recent years, rapid advancements in digital technologies, cloud computing, the Internet of Things (IoT), and artificial intelligence (AI) have led to exponential data growth, creating significant challenges for database management systems (DBMSs) in terms of storage, query performance, and scalability. Consequently, database performance optimization has become essential for reliable data management. PostgreSQL, a widely adopted open-source relational database management system (RDBMS), provides advanced optimization features such as indexing, query optimization, parallel query execution, and table partitioning, making it well suited for large-scale, data-intensive applications. Although NoSQL databases such as MongoDB have emerged as effective solutions for large-scale and semi-structured data through flexible models and horizontal scalability, relational database management systems remain essential for applications requiring strong consistency, transactional reliability, and complex query processing. Among PostgreSQL's optimization techniques, indexing and table partitioning are widely used to improve query performance.

This study evaluates the impact of indexing and table partitioning on PostgreSQL performance in large-scale data management environments. The experiments were conducted using PostgreSQL 17 on a database containing 1,000,103 records to simulate a large-scale workload, and performance metrics were measured before and after applying the proposed optimization techniques.

II. Research Problem

Although PostgreSQL is one of the most widely used relational database management systems — valued for its reliability, flexibility, and support for modern standards — its performance can be affected when handling vast datasets, particularly during complex queries and operations that process large numbers of records. PostgreSQL offers optimization techniques such as indexing and table partitioning designed to accelerate search and retrieval and reduce execution time. However, the effectiveness of these techniques varies with the nature and volume of the data and the types of queries employed, and many organisations and developers lack a clear understanding of how each technique affects actual system performance.

The research problem therefore lies in the need to evaluate the impact of indexing and table partitioning techniques on PostgreSQL performance when handling large-scale data environments, and to measure the extent to which they contribute to reducing query execution time and improving resource utilisation.

III. Research Objectives

To address the identified research problem and evaluate the effectiveness of PostgreSQL optimization techniques for large-scale data management, this study aims to investigate the impact of indexing and table partitioning on database performance. Specifically, the objectives of this study are as follows:

1. To study the concept of large-scale data management and its impact on database performance.

2. To examine the characteristics of PostgreSQL and its available performance-optimization techniques.
3. To evaluate the impact of indexing on query performance.
4. To evaluate the impact of table partitioning on query performance.
5. To compare performance before and after implementing optimization techniques.
6. To identify best practices for optimizing PostgreSQL performance when handling in large-scale data management environments.

IV. Related Work

Previous studies have investigated various optimization techniques in relational database management systems such as PostgreSQL — including indexing, table partitioning, query optimization, and execution-plan analysis — to improve query-processing efficiency and reduce execution time. Table I summaries a comparison of previous studies on improving PostgreSQL performance in a large-scale datasets environment.

Table I. Summary of Previous Studies

No	Study (Year)	Techniques Used	Key Findings	Research Gap
1	Eltabakh et al. (2006)	SP-GiST, B+-Tree, R-Tree	Implemented the SP-GiST indexing framework in PostgreSQL and demonstrated significant improvements in search and update performance for spatial and multidimensional data compared with conventional indexing methods.	Focused on specialized indexing for spatial and multidimensional data without evaluating table partitioning.
2	Cook et al. (2006)	SP-GiST	Demonstrated that specialized indexing structures significantly improve the performance of spatial queries compared with traditional indexing techniques.	Limited to spatial databases and did not investigate general-purpose PostgreSQL workloads or table partitioning.

No	Study (Year)	Techniques Used	Key Findings	Research Gap
3	Stonebraker et al. (2018)	Advanced Database Optimization Techniques	Demonstrated that modern relational database systems remain highly competitive for large-scale data applications when supported by advanced optimization techniques.	Focused on general optimization concepts rather than providing detailed experimental evaluation.
4	Cheng et al. (2019)	Indexing and Database Optimization	Reported that appropriate indexing strategies and database optimization techniques significantly improve query execution performance on large datasets.	Did not evaluate the impact of table partitioning or its integration with indexing.
5	Al-Subaiee et al. (2019)	Table Partitioning	Demonstrated that table partitioning effectively reduces query processing overhead and improves database scalability.	Investigated partitioning only without considering indexing or the combined optimization approach.
6	Harth et al. (2020)	Indexing and Table Partitioning	Confirmed that combining indexing with table partitioning enhances database scalability and query performance.	Did not compare different indexing strategies or evaluate their effectiveness across diverse PostgreSQL workloads**.**
7	Fittkau et al. (2020)	Performance Monitoring, Indexing	Demonstrated that continuous performance monitoring combined with appropriate indexing strategies significantly improves database efficiency and response time.	Did not evaluate table partitioning or compare multiple optimization configurations.

No	Study (Year)	Techniques Used	Key Findings	Research Gap
8	Salunke & Ouda (2024)	Benchmarking	Reported that PostgreSQL achieved competitive performance across multiple benchmark workloads.	Did not analyze the combined effect of indexing and table partitioning.
9	Sirigire (2025)	SQL Query Optimization, Indexing	Found that indexing significantly reduces query execution time while improving overall system resource utilization.	Focused on indexing-based query optimization without experimentally evaluating table partitioning.
10	Khaerunnisa et al. (2025)	Table Partitioning and Indexing	Demonstrated that combining partitioning and indexing produced the best performance, reducing execution time by up to **55.13%**.	Focused on a data warehouse workload and did not evaluate diverse query types in a general-purpose PostgreSQL environment.
11	Toktomusheva (2025)	B-Tree, Hash, GiST, SP-GiST, GIN, and BRIN	Compared PostgreSQL index types and recommended selecting index structures according to workload characteristics.	Did not experimentally evaluate the integration of partitioning and indexing.
12	This Study (2026)	Indexing, Table Partitioning, and Their Combination	Experimentally compares four optimization scenarios: no optimization, indexing only, partitioning only, and indexing combined with partitioning.	Provides a comprehensive experimental evaluation of individual and combined optimization techniques.

Although previous studies have demonstrated the benefits of indexing and partitioning individually, limited research has experimentally evaluated their combined impact on PostgreSQL performance using a unified experimental environment and multiple optimization scenarios. Most studies emphasize either indexing or partitioning separately rather than providing a comprehensive comparison of different optimization configurations. Therefore, this study experimentally investigates

PostgreSQL performance under four scenarios — without optimization, indexing only, table partitioning only, and the combination of indexing and partitioning — to provide a comprehensive evaluation and practical recommendations for a large-scale datasets management.

V. Research Methodology

This study adopted an experimental research methodology to evaluate the effectiveness of indexing and table partitioning in improving PostgreSQL performance in a large-scale datasets environment. A controlled experimental setup was implemented using PostgreSQL 17 with a dataset of 1,000,103 records. Performance was evaluated under four scenarios: (1) without optimization, (2) using indexing, (3) using table partitioning, and (4) using both indexing and table partitioning. The same SQL queries — including search, date-range, join, and aggregate queries — were executed in each scenario to ensure a consistent comparison. Performance was measured using the EXPLAIN ANALYZE command, with query execution time serving as the primary evaluation metric.

Table II. SQL Queries Used in the Experiment

Query Type	SQL Statement
Primary Key Search	SELECT * FROM employees WHERE employee_id=750000;
Salary Search	SELECT * FROM employees WHERE salary=50000;
Date Range Search	SELECT * FROM employees WHERE hire_date BETWEEN '2023-01-01' AND '2023-12-31';

Query Type	SQL Statement
Join Query	SELECT e.employee_id, e.first_name, d.department_name, j.job_title FROM employees e JOIN departments d ON e.department_id=d.department_id JOIN jobs j ON e.job_id=j.job_id;
COUNT	SELECT COUNT(*) FROM employees;
AVG salary	SELECT AVG(salary) FROM employees;

Experimental Environment

The experiments were conducted on a personal computer with the following configuration; the default PostgreSQL server configuration was used to simulate a realistic environment. Performance may differ in production environments where memory parameters (e.g., shared buffers, work_mem, and effective_cache_size) are tuned for large-scale workloads

Table III. Experimental Environment

Component	Specifications
Operating System	Windows 10 Pro
Processor	Intel Core i5-10210U CPU @ 1.60 GHz
RAM	8 GB (7.71 GB usable)
DBMS	PostgreSQLL 17
Administration Tool	pgAdmin 4
Query Language	SQL

Experimental Database Design

The physical database was designed to model real-world information systems. To ensure scalability, a synthetic dataset of 1,000,103 records was generated using SQL scripts, simulating the increasing workloads typical of a large-scale dataset application. Four scenarios were formulated: PostgreSQL without optimization, PostgreSQL using indexing, PostgreSQL using table partitioning, and PostgreSQL using both indexing and table partitioning.

Experimental Procedure

The experimental procedure followed four sequential phases:

1. Execute all queries without optimization and record the execution plans and times.

2. Create the database indexes and repeat the experiments.

3. Apply table partitioning and repeat the experiments.

4. Combine indexing with partitioning and perform the final evaluation.

The results obtained from these phases were then compared to determine the effectiveness of each optimization strategy.

VI. Experimental Setup

The experiments were designed to evaluate the impact of indexing, table partitioning, and their combination on PostgreSQL query performance using a large-scale dataset. The experimental database was implemented using PostgreSQL 17 and managed through pgAdmin. All experiments were conducted under the same hardware and software environment described above, while the database configuration remained unchanged throughout the evaluation process to ensure consistency, fairness, and reliable comparison among the different optimization scenarios.

Database Schema

A relational database was designed to represent a simplified enterprise information system, consisting of three interconnected tables:

- departments (department_id [PK], department_name, location).
- jobs (job_id [PK], job_title, min_salary, max_salary).
- employees (employee_id [PK], first_name, last_name, gender, email, phone, hire_date, salary, department_id [FK], job_id [FK]).

Table Partitioning Configuration

Range partitioning was implemented using the hire_date column, dividing the employees table into yearly partitions to reduce the amount of data scanned during query execution. Date-based partitioning was chosen because it is widely used in enterprise applications and supports efficient processing of historical records.

Query Workload

The following SQL operations were executed in every experimental scenario, and the resulting execution plans generated by PostgreSQL are summarized in Table II. Each query was executed multiple times and

the average execution time was recorded to minimize random variation.

Table IV Query Execution Plans Generated by PostgreSQL

Query	Execution Plan
Primary Key Search	Index Scan
Salary Search	Index Scan
Date Range Search	Sequentiall Scan , Bitmap Index Scan
Join Query	Hash Join
COUNT	Aggregate
AVG Salary	Aggregate

Performance Measurement

Database performance was evaluated using PostgreSQL's EXPLAIN ANALYZE command. The metrics collected were the query execution time (ms) and the execution plan, which together provide a comprehensive evaluation of the effectiveness of indexing and table partitioning.

This section presents the experimental results obtained from evaluating PostgreSQL performance under the four optimization scenarios, using identical datasets and SQL queries. Query execution time and execution plans were measured using EXPLAIN ANALYZE. Table III reports the performance comparison before and after applying indexing and table partitioning.

VII. Results and Discussion

Table V. Performance Comparison of PostgreSQL Queries Before and After Optimization

Experiment	SQL Query	Before (ms)	After Indexing	After Partitioning	Index + Partitioning	Improvement (%)
Primary Key	PK	0.518	0.036	4.905	3.262	93.05%
Salary	Salary	207.575	51.503	205.965	42.988	79.29%
Date Range	Hire Date	119.837	52.024	31.629	24.759	79.34%

Experiment t	SQL Query	Before (ms)	After Indexing	After Partitioning	Index + Partitioning	Improvement (%)
Join Query	JOIN	682.346	564.912	621.571	560.734	17.82%
COUNT	Aggregate	93.933	84.542	120.82	104.689	10.00%
AVG Salary	Aggregate	156.118	134.541	164.931	134.282	13.99%

Table III presents the performance evaluation of different optimization scenarios, including no optimization, indexing, table partitioning, and the combined approach of indexing and partitioning. The results demonstrate that optimization techniques significantly reduced query execution time compared with the optimized database.

For the Primary Key Search query, indexing achieved the best performance with an execution time of 0.036 ms, resulting in a 93.05% improvement compared with the original execution time of 0.518 ms. The combined indexing and partitioning approach did not provide additional benefits for this query because primary key lookups already rely on direct index access, making partition pruning less effective.

The Salary Search query showed a significant improvement after applying the combined indexing and partitioning approach, reducing execution time from 207.575 ms to 42.988 ms, which represents a 79.29% improvement. This indicates that indexing improves data retrieval efficiency, while partitioning reduces the amount of data scanned during query execution.

Similarly, the Date Range Search query achieved the best performance using the combined optimization approach, decreasing execution time from 119.837 ms to 24.759 ms with a 79.34% improvement. This result confirms that partitioning is particularly effective for range-based queries, especially when combined with appropriate indexing strategies.

For the Join Query, the execution time was reduced from 682.346 ms to 560.734 ms, achieving a 17.82% improvement. Although the improvement was lower than other query types, the results indicate that indexing and partitioning can still enhance join performance by reducing data access overhead. However, join operations depend on several factors, including join conditions, table size, and query execution plans.

The Aggregate queries showed different behaviors. The COUNT query achieved a 10.00% improvement, with indexing providing the best result (84.542 ms), while partitioning alone increased execution time due to the overhead of scanning multiple partitions. The AVG Salary query achieved a 13.99% improvement, where the combined optimization approach slightly outperformed indexing alone.

Overall, the results indicate that the effectiveness of optimization techniques

depends on query characteristics. Indexing provides the greatest benefit for point-based searches, while the combination of indexing and partitioning is more effective for large-scale range queries and complex workloads. These findings demonstrate that selecting appropriate optimization strategies according to query patterns is essential for improving PostgreSQL performance.

VIII. Conclusion

The experimental evaluation demonstrated that both indexing and table partitioning significantly improved PostgreSQL query performance, although the level of improvement varied by query type. Prior to optimization, PostgreSQL relied mainly on sequential scans, resulting in higher execution times. Table partitioning was particularly effective for date-range queries, where partition pruning reduced the data scanned, and the combination of indexing and partitioning achieved the best overall performance, with improvements of up to 79.34% compared with the baseline. JOIN queries showed only moderate gains, indicating that their execution is strongly influenced by data distribution and the query plan, while aggregate queries showed mixed results.

The findings indicate that the effectiveness of database optimization techniques depends on workload characteristics: indexing is most beneficial for selective search queries, whereas table partitioning is most effective for queries that filter on the partitioning key. The study concludes that selecting optimization techniques according to query patterns and data characteristics is essential for improving PostgreSQL performance and supporting efficient a large-scale datasets management.

IX. Future Work

This study focused on evaluating the impact of indexing and partitioning on PostgreSQL performance; several directions remain for future research:

7. Using larger datasets to further test scalability.
8. Studying different types of indexes (e.g., Hash, GiST, GIN, BRIN).
9. Evaluating different data-partitioning strategies.
10. Investigating advanced query enhancements.
11. Evaluating performance in cloud environments.

References

[1] PostgreSQL Global Development Group, PostgreSQL Documentation, 2025.

[2] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, 7th ed. Pearson Education, 2016.

[3] R. Ramakrishnan and J. Gehrke, Database Management Systems, 3rd ed. McGraw-Hill, 2003.

[4] H. Garcia-Molina, J. D. Ullman, and J. Widom, Database Systems: The Complete Book, 2nd ed. Pearson, 2009.

[5] M. Kleppmann, Designing Data-Intensive Applications. O'Reilly Media, 2017.

[6] P. J. Sadalage and M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging

World of Polyglot Persistence. Addison-Wesley, 2012.

[7] K. Chodorow, MongoDB: The Definitive Guide, 2nd ed. O'Reilly Media, 2013.

[8] M. Stonebraker and J. M. Hellerstein, Readings in Database Systems, 4th ed. MIT Press, 2005.

[9] S. V. Salunke and A. Ouda, "A performance benchmark for the PostgreSQL and MySQL databases," Future Internet, vol. 16, no. 10, art. 382, 2024.

[10] G. Toktomusheva, "Indexing in PostgreSQL: Performance Evaluation and Use Cases," Preprints.org, Nov. 2025. doi: 10.20944/preprints202511.2170.v1.

[11] L. S. Khaerunnisa et al., "Optimizing data warehouse performance using partitioning and indexing in PostgreSQL," 2025.

[12] 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, 2025, doi: 10.14419/x2wqqh31.

[13] E. Dritsas and M. Trigka, "Database Systems in the Big Data Era: Architectures, Performance, and Open Challenges," IEEE Access, vol. 13, pp. 95068–95084, 2025, doi: 10.1109/ACCESS.2025.3572059.

[14] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," Mobile Networks and Applications, vol. 19, pp. 171–209, 2014, doi: 10.1007/s11036-013-0489-0.

[15] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 7th ed. New York, NY, USA: McGraw-Hill Education, 2020.