

Analysis of optimization capabilities of selected database management systems

Paweł Tarkiewicz*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The study is devoted to the analysis of optimization features and mechanisms available in Oracle Database, SQL Server, and PostgreSQL. The purpose of the study was to identify differences in the available functionalities and to assess the impact of selected mechanisms on SQL query performance. The research included a theoretical analysis based on database system documentation and a practical part conducted in a prepared test environment. The research scenarios examined the impact of indexes, materialized views, data partitioning, and query optimizer behaviour. The analysis indicated that Oracle Database offers the most extensive set of optimization functionalities compared with the other examined systems. The results of the practical tests showed that the use of appropriate optimization mechanisms can significantly reduce query execution time, while the scale of improvement depends on the type of query and the way in which a given mechanism is implemented.

Keywords: database optimization; query execution; indexes; materialized views; partitioning; performance analysis.

*Corresponding author

Email address: s95594@pollub.edu.pl (P. Tarkiewicz)

Published under Creative Common License (CC BY 4.0 Int.)

1. Introduction

In the present era, characterized by the dynamic development of information technologies and the growing volume of processed data, the performance of database operations plays a crucial role in the functioning of modern information systems. Relational databases remain among the most chosen solutions [1]. Due to their broad support and advanced capabilities in data management, environment configuration, integrity, and security, they constitute the foundation of many business systems [2].

When choosing an appropriate database system, it is essential to consider optimization features and tools provided by a given technology, as they make it possible to effectively adapt performance to the requirements of a particular project and ensure appropriate and efficient query processing. Differences in the implementation of optimization mechanisms across individual systems may significantly affect the final performance of systems, as well as processing time under different workloads and types of operations. Therefore, it is important to make an informed choice of the appropriate technology and the set of features and tools it offers to achieve the highest possible performance and system stability.

The motivation behind this study was the need to provide both a theoretical and practical comparison of popular relational database systems in terms of their optimization capabilities and the available tools for performance analysis and monitoring. Conducting such an analysis makes it possible to gain a more detailed understanding of the differences in optimization mechanisms among the examined database systems and allows to identify solutions that meet specific requirements,

ensuring optimal and stable operation of the analysed database systems.

The purpose of this study is to provide a comparative analysis of methods and tools dedicated to the optimization and analysis of the execution of various database operations, with particular emphasis on query processing. This objective was achieved through research scenarios conducted on three selected relational database systems: Oracle Database, SQL Server, and PostgreSQL.

2. Related Work

2.1. Database management systems

In the paper “Comparison of Semi-structured Data on MSSQL and PostgreSQL”, the authors compared the optimization of semi-structured data processing in JSON format in SQL Server and PostgreSQL [3]. The study results indicate that PostgreSQL provides more advanced capabilities for indexing and manipulating JSON data, while SQL Server is more limited in this regard, both in terms of functionality and performance. These findings highlight the architectural and technological differences between the analysed systems.

The article “Microsoft SQL Server and Oracle: Comparative Performance Analysis” provides an in-depth comparative analysis of Oracle and SQL Server database systems [4]. The comparison covers theoretical differences between the databases, the basic characteristics of the systems, and query execution plans. The aim of the article was to identify the advantages and disadvantages of the selected database systems. The results indicate that Microsoft SQL Server achieved significantly better performance than Oracle, both for simple queries on single tables and for JOIN queries involving multiple tables. The results of the study suggest that the

the database.

The test environment was built using Docker technology, which enables the creation of isolated containers for running instances of individual database management systems. The use of containers ensured an isolated, consistent, and repeatable configuration for each of the analyzed database management systems. Each system was deployed in a dedicated container based on the official image of the selected database in the following versions:

- Oracle Database 26ai Free,
- Microsoft SQL Server 2022,
- PostgreSQL 14.3.

At a later stage, ready-made SQL scripts provided with the database were used to recreate the database structure and to generate and populate it with test data.

The practical tests were conducted on a local machine whose technical specifications are presented in Table 1.

Table 1: System specifications

Component	Specification
Operating system	Fedora Linux 42
Processor	Intel Core i5-11400F
RAM	16 GB
Storage	SSD NVMe 1 TB

3.2. Research procedure

The research was divided into two main stages: a theoretical analysis and a practical analysis. This approach enabled the identification of optimization tools and methods available in the analyzed systems, as well as differences in their implementation.

3.2.1. Theoretical analysis

The goal of the theoretical analysis was to identify and compare the optimization functionalities and tools available in the analyzed database systems. This analysis made it possible to identify differences between the systems in terms of the availability of optimization mechanisms and tools, their complexity, and the way they are implemented.

The theoretical analysis covered the following aspects:

- optimizer operation and query execution plan generation,
- types and categories of indexes,
- mechanisms for creating and managing views and materialized views,
- data partitioning capabilities,
- tools for analysing and generating statistics for queries executed in the database,
- mechanisms for caching data and query results.

A analysis was conducted based on the official documentation of the selected DBMSs [11, 12, 13].

This approach enabled the identification of common and unique features of each system and served as a basis

for selecting the mechanisms to be examined in the practical analysis.

3.2.2. Practical analysis

The purpose of the practical analysis was to verify and compare how the implementation of selected optimization mechanisms affects query execution performance in the tested database systems. The practical verification enabled an assessment of the effectiveness of these mechanisms under real query execution conditions.

The research was conducted in two stages. In the first stage, tests were performed in a non-optimized test environment by repeatedly executing prepared complex SQL queries. The queries and operations were executed 100, 1,000, and 10,000 times, which made it possible to observe the behavior of the analyzed systems under different workloads and data volume scales. Additionally, the cache was cleared before each query execution attempt to eliminate the influence of previous results and ensure consistent measurements.

In the next stage, optimization mechanisms were introduced according to the capabilities of each system. These included the addition of indexes, the use of materialized views, optimizer analysis based on query execution plans, and data partitioning. As part of the optimizer analysis, sets of queries returning identical data were prepared, with each query written in a different syntactic form. For each statement, execution plans were examined to analyze the behavior of the optimizer and evaluate the performance of individual query variants. The examined parameters primarily included the execution time of a single query operation.

Time measurements were performed using a script developed in C#. The Stopwatch class was used to measure the execution time of a single query, with the operation time recorded in milliseconds. In each iteration, the query was executed separately, and the obtained execution time was stored as an individual measurement result. The script also included cache-clearing operations. In the case of PostgreSQL, due to the specific characteristics of the test environment, the container had to be restarted after each query execution to reduce the influence of cached data on subsequent measurements.

3.3. Research scenarios

To conduct performance tests of the database systems, research scenarios were prepared to cover selected optimization mechanisms. The analysis examined the impact of indexes, materialized views, table partitioning, and query optimizer mechanisms. Performance measurements were performed both before and after the implementation of the selected optimization mechanisms.

3.3.1. Scenario 1 – composite indexes

The research scenario involved executing an aggregation query that combines data from the table containing customer order information, the table storing order sta-

tus history, and the table containing order line details. Then, three composite indexes are created on the tables included in the query. The indexes cover columns used in filtering conditions, table join operations, and data aggregation. The purpose of this scenario is to demonstrate the impact of multi-column, or composite, indexes on the performance of an aggregation query that processes data from multiple tables.

3.3.2. Scenario 2 – materialized view

The scenario involves executing an aggregation query based on data from four tables to produce statistical information about customer orders, including the number of orders, total order value, number of order items, order status changes, and values assigned to specific statuses. Materialized views are used as an optimization mechanism in Oracle Database and PostgreSQL, while SQL Server uses an indexed view as an equivalent solution. The purpose of this scenario is to assess the impact of these mechanisms on the performance of complex aggregation queries processing large volumes of data.

3.3.3. Scenario 3 – data partitioning

The scenario involves executing a query that retrieves data from the order history table and filters it by the status change date. To optimize this operation, range partitioning is applied to the table based on the filtering column, with yearly partitions. This allows the data to be divided into date ranges and limits the number of rows scanned to the relevant partitions during query execution. The purpose of this scenario is to assess the impact of range partitioning on the performance of queries that filter data within a single partition range.

3.3.4. Scenario 4 - optimizer

The scenario assumes the execution of queries returning a list of customers who have never placed an order using a specific shipping method, divided into three variants differing in their data exclusion approach. The first variant uses the NOT IN operator, the second uses NOT EXISTS with a correlated subquery, and the third uses LEFT JOIN with a NULL check on the joined table side. The scenario aims to analyze the optimizer's behavior by comparing execution plans for logically equivalent queries, as well as to evaluate the impact of each method on query execution performance.

4. Theoretical Analysis

4.1. Query optimizers

All selected databases use cost-based query optimizers, but they differ in cost estimation methods, the scope of statistics used, adaptive mechanisms and the degree of control over optimizer behavior. Oracle Database provides the most extensive set of adaptive optimization and statistics management mechanisms. SQL Server focuses strongly on automation, plan reuse, and mechanisms for detecting and correcting performance regressions. PostgreSQL, in contrast, follows a more traditional and transparent cost-based optimization model,

offering fewer automatic adaptation mechanisms but greater predictability of optimizer behavior.

4.2. Indexing mechanisms

The analyzed database management systems provide extensive indexing support. Oracle Database offers a broad and highly specialized set of index structures, such as B-tree, bitmap, function-based, and invisible indexes, which allow the system to support both transactional workloads and analytical processing. Microsoft SQL Server is based primarily on clustered and nonclustered indexes, which support efficient access to table data, while also offering columnstore indexes for covering queries and analytical workloads. PostgreSQL, in contrast, provides a flexible indexing architecture based on multiple access methods, including B-tree, Hash, GiST, SP-GiST, GIN, and BRIN.

4.3. View and Materialized View

The analyzed systems support standard view mechanisms, but they differ in their support for materialized views and their integration with the query optimizer. Oracle Database provides advanced materialized view functionality with automated refresh and query rewrite support. SQL Server implements indexed views, which materialize view data by creating a unique clustered index that stores the result of the view-defining query similarly to a table. Indexed views are also maintained synchronously by the database engine. PostgreSQL supports materialized views, but they require explicit refresh and are not automatically used by the optimizer for query rewrite.

4.4. Data partitioning

The analyzed systems provide data partitioning mechanisms, but they differ in the available partitioning methods, integration with the query optimizer, and automation of partition management. Oracle Database offers a wide range of partitioning methods, close integration with the query optimizer, automatic partition creation, and greater design flexibility due to support for global indexes. In SQL Server, partitioning is mainly range-based and is strongly linked to tables and indexes through partition functions and partition schemes. PostgreSQL supports native range, list, and hash partitioning, where partitions are physically separate tables, providing flexibility but requiring more manual management.

4.5. Statistical and diagnostic tools

The selected database systems provide mechanisms for query performance analysis and statistical data collection, but they differ significantly in functionality, automation, and data presentation. Oracle Database offers the most extensive set of integrated diagnostic tools, supporting both real-time and historical analysis, automatic detection of performance issues, and optimization recommendations. SQL Server provides practical query monitoring mechanisms, especially Query Store, which enables the analysis of execution plan history and query

statistics and supports automatic plan correction. PostgreSQL offers a narrower range of built-in performance analysis and statistics tools, but it can be extended with extensions, and external tools.

4.6. Data caching mechanisms

The selected database systems provide caching mechanisms used to reduce the number of input-output operations. Oracle Database caches data blocks read from disk, execution plans, and query results through the Result Cache mechanism. SQL Server mainly focuses on caching data pages, indexes, and execution plans, while providing a high level of automation in cache management. PostgreSQL uses a two-layer caching model that combines its own shared buffer with the operating system cache. Unlike Oracle Database, PostgreSQL and SQL Server do not include a native mechanism for caching complete SQL query results.

4.7. Summary

The analysis showed that the examined systems offer a comparable set of basic functionalities, but differ in their level of advancement, automation, and administrator control. Oracle Database provides the most extensive and integrated set of optimization mechanisms, including advanced query optimization techniques, broad statistical capabilities, and comprehensive analytical and diagnostic tools. Microsoft SQL Server focuses mainly on automation and execution plan stability, although it offers less diverse partitioning mechanisms. PostgreSQL provides high transparency and flexibility, but requires greater user involvement in performance design, configuration, and tuning, and offers a narrower range of native analytical tools. Table 2 summarizes the key optimization mechanism available in selected database management systems.

Table 2: Overview of optimization mechanisms in the analyzed systems

Mechanism	Oracle Database	SQL Server	PostgreSQL
Query optimizer	Adaptive CBO	CBO with plan caching	CBO + GEQO
Indexing	B-tree, bit-map, invisible, specialized	Clustered, Nonclustered, columnstore	B-tree, Hash, GiST, SP-GiST, GIN, BRIN
Views	Materialized views, query rewrite	Indexed views	Materialized views (manual refresh)
Partitioning	Advanced, automatic	Rangebased	Native, table-based
Diagnostic tools	AWR, ASH, ADDM	Query Store, DMV	ANALYZE, extensions
Caching	Data and result cache	Data and plan cache	Data and OS cache

5. Results

This chapter presents the results of individual scenarios for the selected databases. The results are presented in bar charts comparing the average single-query execution times for individual measurement series, including the minimum and maximum values.

5.1. Scenario 1

Figure 2 presents a bar chart comparing the average execution times of a single query for scenario 1 in Oracle Database. The use of composite indexes resulted in a decrease in the average query execution time by approximately 45% to 47%, depending on the number of executions. For the series of 10,000 executions, the average time after applying indexes decreased from 235.89 ms to 125.87 ms, which is a difference of 110.02 ms.

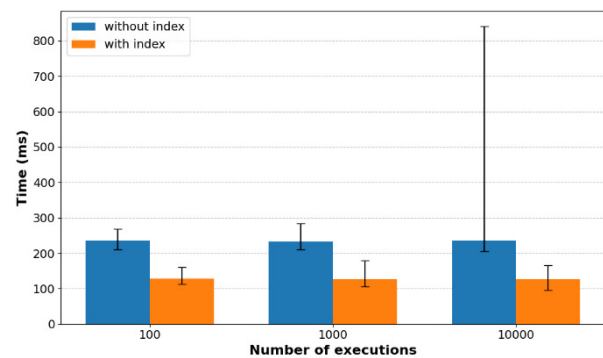


Figure 2: Average query execution times before and after applying the composite indexes in Oracle Database

Figure 3 presents a bar chart comparing the average execution times of a single query for scenario 1 in SQL Server. The use of composite indexes resulted in a decrease in the average query execution time by approximately 55% to 56%, depending on the number of executions. For the series of 10,000 executions, the average time after applying indexes decreased from 78.7 ms to 3.52 ms, which represents a reduction of 43.18 ms.

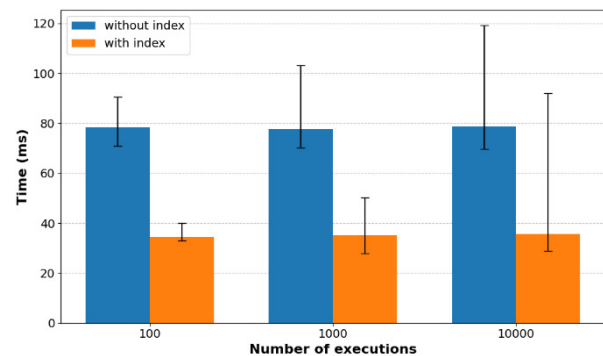


Figure 3: Average query execution times before and after applying the composite indexes in SQL Server

Figure 4 presents a bar chart comparing the average execution times of a single query for scenario 1 in PostgreSQL. The use of composite indexes resulted in a decrease in the average query execution by approximately 12%. For the series of 10,000 executions, the average time after applying indexes decreased from 171.19 ms to 150.21 ms, which represents a reduction of 20.98 ms.

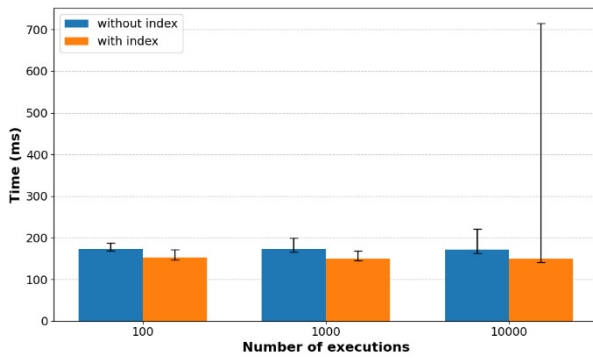


Figure 4: Average query execution times before and after applying the composite indexes in PostgreSQL

5.2. Scenario 2

Figure 5 presents a bar chart comparing the average execution times of a single query for scenario 2 in Oracle Database. The creation of the materialized view significantly reduced the average query execution time by approximately 94%. For the series of 10,000 executions, the average execution time decreased from 1588.56 ms to 92.43 ms, which represents a reduction of 1496.13 ms.

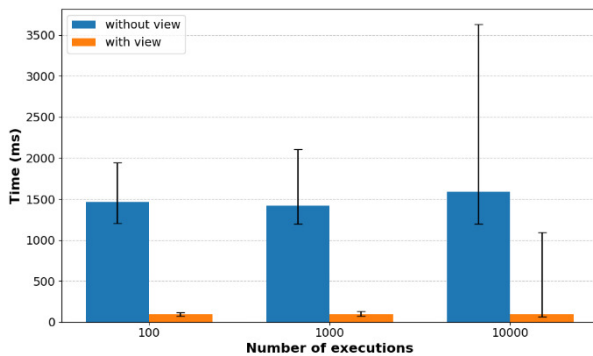


Figure 5: Average query execution times before and after applying the materialized view in Oracle Database

Figure 6 presents a bar chart comparing the average execution times of a single query for scenario 2 in SQL Server. The creation of the materialized view significantly reduced the average query execution time by approximately 88%. For the series of 10,000 executions, the average execution time decreased from 264.29 ms to 30.9 ms, which represents a reduction of 233.39 ms.

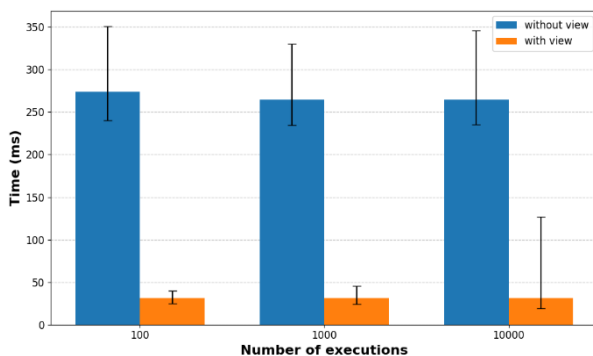


Figure 6: Average query execution times before and after applying the indexed view in SQL Server

Figure 7 presents a bar chart comparing the average execution times of a single query for scenario 2 in PostgreSQL. The creation of the materialized view significantly reduced the average query execution time by approximately 98%. For the series of 10,000 executions, the average execution time decreased from 1015.29 ms to 23.02 ms, which represents a reduction of 992.27 ms.

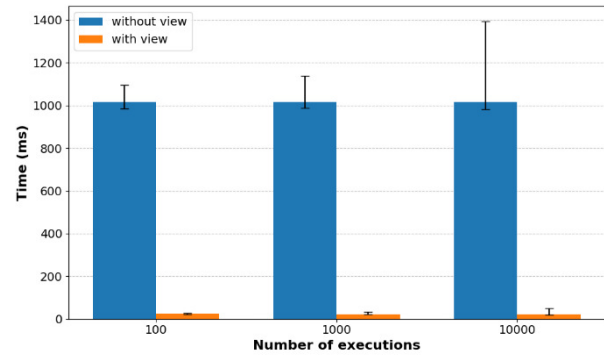


Figure 7: Average query execution times before and after applying the materialized view in PostgreSQL

5.3. Scenario 3

Figure 8 presents a bar chart comparing the average execution times of a single query for scenario 3 in Oracle Database. Partitioning the table resulted in a reduction in the average query execution time by approximately 31% to 33%, depending on the number of executions. For the series of 10,000 executions, the average time after partitioning decreased from 171 ms to 114.2 ms, which represents a reduction of 56.8 ms.

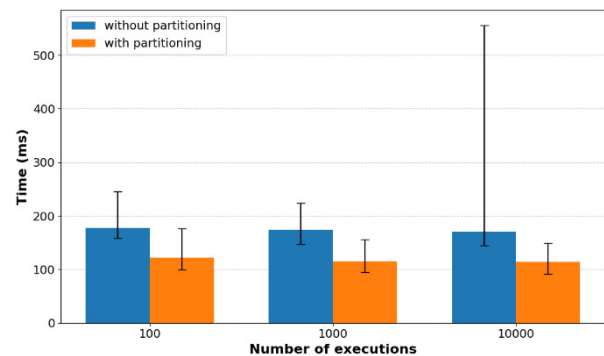


Figure 8: Average query execution times before and after applying the partitioning in Oracle Database

Figure 9 presents a bar chart comparing the average execution times of a single query for scenario 3 in SQL Server. Partitioning the table resulted in a reduction in the average query execution time by approximately 21% to 24%, depending on the number of executions. For the series of 10,000 executions, the average time after partitioning decreased from 61.43 ms to 48.51 ms, which represents a reduction of 12.92 ms.

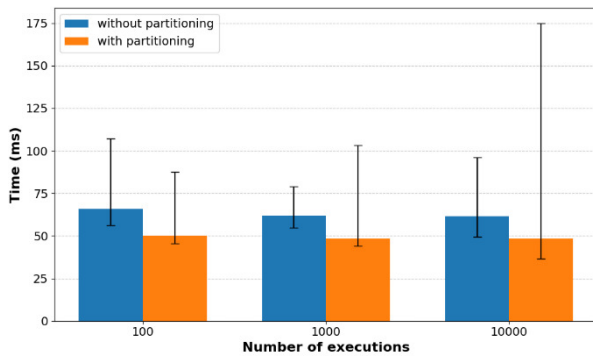


Figure 9: Average query execution times before and after applying the partitioning in SQL Server

Figure 10 presents a bar chart comparing the average execution times of a single query for scenario 3 in PostgreSQL. Partitioning the table resulted in a reduction in the average query execution time by approximately 34% to 37%, depending on the number of executions. For the series of 10,000 executions, the average time after partitioning decreased from 104.72 ms to 66.28 ms, which represents a reduction of 38.44 ms.

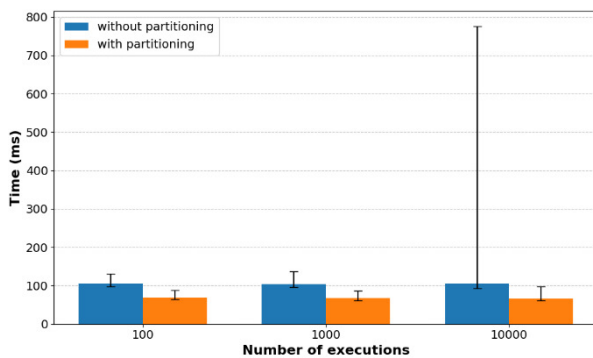


Figure 10: Average query execution times before and after applying the partitioning in PostgreSQL

5.4. Scenario 4

The scenario involved executing three exclusion queries using NOT IN, NOT EXISTS, and LEFT JOIN, each returning the same data.

In Oracle Database, the shortest average execution time was obtained for the NOT EXISTS variant, while the longest was recorded for the NOT IN variant. For the series of 10,000 executions, the average times were 179.69 ms for NOT IN, 158.16 ms for NOT EXISTS, and 167.49 ms for LEFT JOIN. The execution plan analysis indicated that the NOT IN and NOT EXISTS variants were transformed into anti-joins, whereas the LEFT JOIN variant was executed as an outer join with additional filtering.

In SQL Server, the longest average execution time was obtained for the NOT IN variant, while NOT EXISTS and LEFT JOIN achieved comparably lower times. For the series of 10,000 executions, the average times were 109.98 ms for NOT IN, 45.43 ms for NOT EXISTS, and 44.90 ms for LEFT JOIN. The execution plans showed that NOT EXISTS and LEFT JOIN were executed in a similar way, using a mechanism that elim-

inates records with matching values. The NOT IN variant also used an anti-join mechanism, but its plan was more complex, which may have resulted from the need to handle exclusion logic involving potential NULL values.

In PostgreSQL, the shortest average execution times were observed for the NOT EXISTS variant, while the longest were recorded for the LEFT JOIN variant. For the series of 10,000 executions, the average times were 128.71 ms for NOT IN, 103.21 ms for NOT EXISTS, and 152.54 ms for LEFT JOIN. The execution plans showed that each variant was processed differently. The NOT IN variant used a hashed subplan for the subquery results, the NOT EXISTS variant applied an anti-join, and the LEFT JOIN variant performed an outer join with an additional filter that retained only unmatched records.

6. Conclusions

The aim of the study was to analyze the optimization mechanisms available in Oracle Database, SQL Server, and PostgreSQL. It was achieved through a theoretical analysis of the available optimization mechanisms and practical tests covering scenarios involving indexes, materialized or indexed views, data partitioning, and query execution plan analysis.

The theoretical analysis indicated that Oracle Database provides greater optimization capabilities than the other examined systems, primarily due to the availability and level of advancement of its optimization mechanisms. The results of the practical tests demonstrated that the use of selected optimization mechanisms can significantly affect query execution time, with the scale of improvement depending on the examined system, the type of query, and the way in which a given mechanism was applied.

The greatest performance improvement across all examined systems was observed when materialized views or their equivalents were used. It was also observed that the use of composite indexes in PostgreSQL resulted in a smaller performance improvement compared with the other examined systems.

The query optimizer analysis indicated that logically equivalent queries may be processed differently depending on the database system. The differences in generated execution plans were reflected in varying execution times for individual query variants.

Further research could include an analysis of the impact of larger data volumes, more complex table structures, tests in a multi-user environment, and an assessment of memory and CPU usage when applying individual optimization mechanisms.

References

- [1] Stack Overflow Developer Survey 2025 – Technology: Databases, <https://survey.stackoverflow.co/2025/technology#1-databases>, [13.02.2026].
- [2] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and

- Maintainable Systems, O'Reilly Media, Sebastopol, 2017.
- [3] L. Alves, P. Oliveira, Comparison of Semi-structured Data on MSSQL and PostgreSQL, *Marketing and Smart Technologies, Smart Innovation, Systems and Technologies*, Springer 337 (2023) 31–43, https://doi.org/10.1007/978-981-19-9099-1_3.
 - [4] E. K. Naka, V. G. Guliashki, Optimization Techniques in Data Management: A Survey, *Proceedings of the 2021 7th International Conference on Computing and Data Engineering (ICCDE 2021)*, Phuket, Thailand (2021) 8–13, <https://doi.org/10.1145/3456172.3456214>.
 - [5] M. Ilić, L. Kopanja, D. Zlatković, M. Trajković, D. Čurguz, Microsoft SQL Server and Oracle: Comparative Performance Analysis, In *7th International Conference Knowledge Management and Informatics, Vrnjačka Banja, Serbia* (2021) 33–39.
 - [6] Q. Wang, PostgreSQL database performance optimization, Bachelor thesis, Business Information Technology, Haaga-Helia University of Applied Sciences, 2011.
 - [7] S. Lindgren, Performance comparison between two relational database management systems: B-tree indexing in PostgreSQL and MySQL, Bachelor thesis, Department of Computing Science, Umeå University, Umeå, 2024.
 - [8] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, T. Neumann, How Good Are Query Optimizers, Really?, *Proceedings of the VLDB Endowment* 9(3) (2015) 204–215, <https://doi.org/10.14778/2850583.2850594>.
 - [9] M. Radoev, New Opportunities for a Query Performance Analysis in MS SQL Server 2016 using the Query Store, *Proceedings of the 5th International Conference on Application of Information and Communication Technology and Statistics in Economy and Education (ICAICTSEE 2015)*, Sofia, Bulgaria (2015) 541–546.
 - [10] How to Set Up the Sample Bookstore Database, <https://www.databasestar.com/sample-bookstore-database/>, [28.09.2025].
 - [11] Documentation. Oracle Database documentation, <https://docs.oracle.com/en/database/oracle/oracle-database/>, [06.09.2025].
 - [12] Documentation. Microsoft SQL Server documentation, <https://learn.microsoft.com/pl-pl/sql/?view=sql-server-ver17>, [06.09.2025].
 - [13] Documentation. PostgreSQL documentation, <https://www.postgresql.org/docs/>, [06.09.2025].