УДК 004.056

STUDY OF THE IMPACT OF DIFFERENT QUERY OPTIMIZATION METHODS ON DATABASE PERFORMANCE AND SECURITY

Abdurakhman N[1], Togzhanova K [1,2]

[1]Satbayev University, , Almaty, Kazakhstan

[2]Almaty Technological University, Almaty, Kazakhstan

Introduction. In modern information systems, databases play a central role as the core component for storing and processing critical organizational data. The rapid growth of data volumes, the increasing complexity of business logic, and the demand for real-time analytics make the efficiency of database query execution a key factor in ensuring high system performance. According to industry research, poorly optimized SQL queries can reduce database performance by more than 50% and significantly increase server resource consumption [1].

At the same time, query design and optimization have a direct relationship not only with performance but also with information security. Incorrect query structure, lack of parameterization, and misconfigured execution strategies create vulnerabilities that can be exploited for SQL injection, inference attacks, and denial-of-service scenarios [2]. As a result, ensuring both performance efficiency and security has become a critical task in modern database administration and software development.

The purpose of this paper is to analyze and systematize modern SQL query optimization methods, evaluate their impact on database performance, and identify their relationship with security mechanisms. The study also highlights strengths, limitations, and future research directions in the area of performance-driven and security-aware query optimization.

The theoretical basis and nature of query optimization and database security. Query optimization in relational database management systems (DBMS) refers to the automated and manual processes used to transform and execute SQL

queries in the most efficient manner, ensuring minimal response time and resource consumption. Modern DBMS, including Microsoft SQL Server, contain built-in query optimizers that analyze query structure, indexes, statistics, and execution plans to determine the most cost-effective execution strategy.

Performance in database systems is influenced by multiple factors: the structure of the query, availability and design of indexes, accuracy of statistics, distribution of data, and load characteristics. Poorly optimized queries not only slow down the system but may also lead to excessive CPU usage, memory pressure, lock contention, and degradation of user experience [2].

Security is closely related to query design. Incorrect query structure, dynamic SQL without parameterization, and weak access control mechanisms increase the risk of SQL injection, data leakage, inference attacks, and denial-of-service attempts. Researchers emphasize that secure and optimized queries must simultaneously ensure high performance and resistance to malicious interactions in database environments [3].

Thus, the study of query optimization and database security evolves in two main directions:

-performance-oriented optimization — indexing, statistics tuning, execution plan analysis, caching, and adaptive query processing.

-Security-oriented query design — parameterization, stored procedures, least-privilege access, query auditing, and protection against injection and inference attacks.

The combination of these approaches forms the foundation for building efficient and secure modern database systems.

Classic query optimization and security methods. Query optimization is the process of improving SQL execution performance by reducing response time and optimizing the use of server resources. In relational database systems, classical optimization approaches laid the foundation for modern query processing techniques and continue to play a key role in database administration. Traditional

methods are based on mathematical models for estimating execution costs, index structures, and pre-compiled execution logic.

The earliest group of approaches includes cost-based and rule-based optimization. In rule-based systems, the optimizer chooses execution strategies according to predefined rules, for example, preferring indexed lookups over full table scans. With the development of relational engines, cost-based optimization became dominant: the optimizer evaluates statistical information about tables, indexes, and data distribution and selects the execution plan with the lowest estimated cost. In Microsoft SQL Server, this includes analyzing predicate selectivity, I/O cost, and available memory to form an optimal plan.

Another classical direction involves indexing and query rewriting. Proper index design — including clustered, non-clustered, filtered, and composite indexes — significantly improves query performance by reducing the amount of scanned data. At the query-level, optimization consists of eliminating redundant joins, replacing nested subqueries with joins, and using set-based operations instead of row-by-row logic. These methods reduce query complexity and improve execution efficiency under high transactional load.

```
create index IX_zakazy_analiz_DateResultReady
    on zakazy_analiz (DateResultReady) include (codeid, typeid, validation)
go

create index IX_zakazy_analiz_typeid
    on zakazy_analiz (typeid) include (codeid_zakazy, codeid_spanaliz, codeid_paket, PriceInOrder, DiscountAmount)
go

create index IX_zakazy_analiz_validation_DateValidate
    on zakazy_analiz (validation, DateValidate) include (codeid, codeid_spanaliz, IdWorkplace)
go

create index IX_zakazy_analiz_cito
    on zakazy_analiz (cito) include (codeid)
go

create index IX_zakazy_analiz_IdWorkplace
    on zakazy_analiz (IdWorkplace)
go

create index IX_zakazy_analiz_DateRegistration
    on zakazy_analiz (DateRegistration_
```

Figure 1. SQL Server index creation example for optimizing query performance

The screenshot demonstrates the creation of multiple non-clustered indexes with included columns to optimize query execution on the zakazy_analiz table. Proper index design minimizes full table scans, reduces I/O operations, and significantly improves performance in transactional workloads.

A third traditional technique is the use of stored procedures and parameterization. Stored procedures pre-compile execution plans, reducing parsing overhead and improving consistency and performance in repeated workloads. Parameterized queries also reuse cached plans and play a critical role in protecting databases from SQL injection attacks. Plan caching, which stores compiled plans for repeated execution, reduces CPU load and ensures faster response time for frequently executed queries.

Figure 2. Example of stored procedure implementation with parameterization and transaction handling in SQL Server

The screenshot demonstrates a SQL Server stored procedure that uses parameterized inputs, transactional control, and audit logging. Such procedures improve performance through execution plan reuse and enhance security by preventing SQL injection and maintaining traceability of user actions.

Empirical studies confirm the significance of classical optimization methods. Microsoft (2023) demonstrated that proper indexing can reduce execution time by up to 62%, while parameterization and plan caching improve CPU efficiency by approximately 40% under repetitive workloads [5]. Binnig et al. (2016) showed that classical cost-based optimizers achieve stable performance in predictable

environments but face limitations when workloads and data distributions become highly dynamic [6].

Classical techniques remain fundamental to database performance tuning and secure query design. They provide high reliability and predictable behavior, especially in transactional systems. However, as data volumes grow and workloads become more heterogeneous, traditional methods face limitations, prompting the development of adaptive and intelligent optimization techniques that leverage machine learning and self-tuning capabilities — which are discussed in the next section.

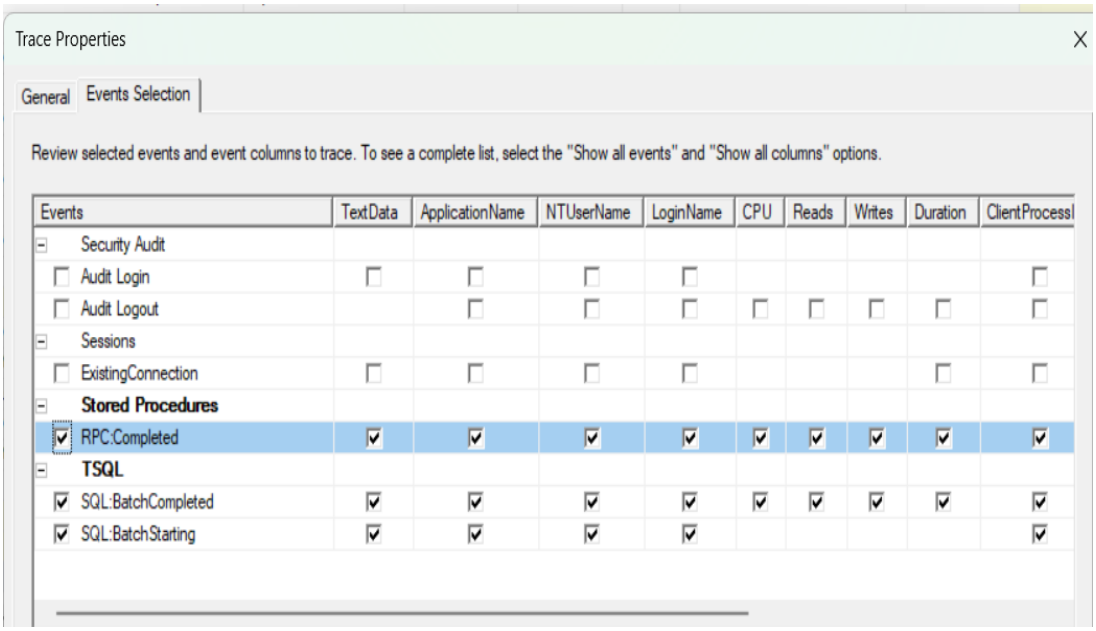

Figure 3. Example of stored procedure with parameterization, transaction control, and logging

This stored procedure demonstrates the use of parameterized inputs, transaction control, and audit logging. Parameterization ensures execution plan reuse and protects against SQL injection attacks by preventing direct concatenation of user input. Transaction handling maintains data consistency, while the audit record improves traceability and supports security monitoring in high-load database environments.

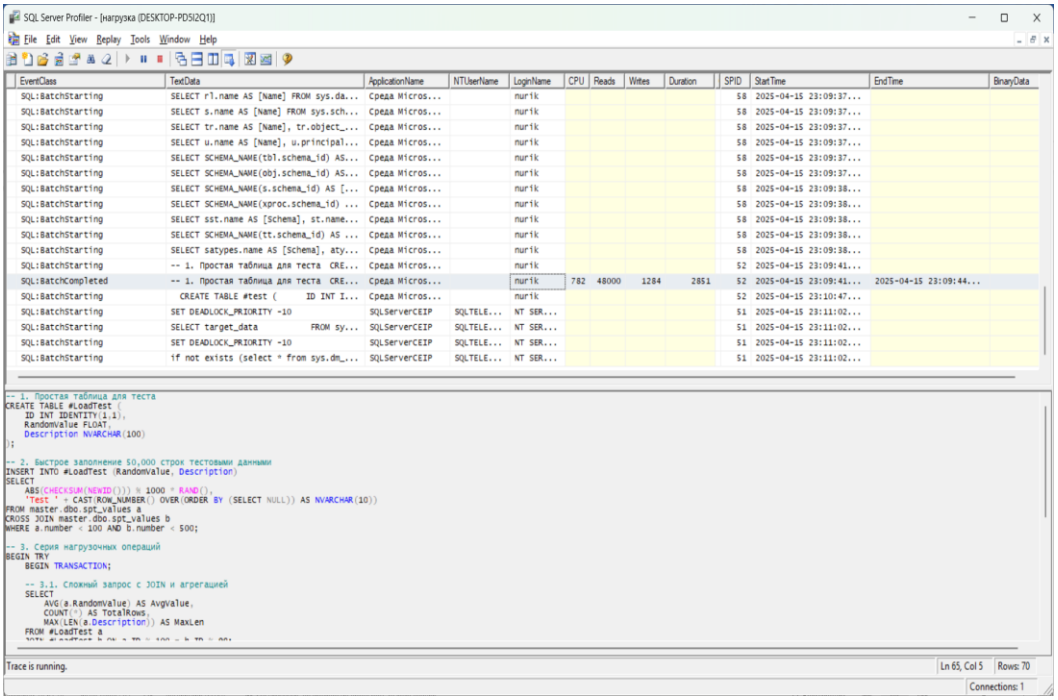

Figure 4. SQL Server Profiler monitoring query execution (CPU, reads, writes, duration)

This screenshot shows real-time monitoring of SQL query execution in SQL Server Profiler, including CPU time, logical reads, writes, and total duration. These metrics help identify inefficient queries, detect excessive resource consumption, and analyze performance bottlenecks. Such monitoring is essential for evaluating the impact of query optimization techniques, improving workload efficiency, and identifying abnormal activity patterns that may indicate potential security risks or resource-exhaustion attacks.

Modern intelligent and adaptive query optimization techniques. The evolution from classical query optimization techniques to intelligent and adaptive methods has become a natural stage in the development of modern database management systems. While traditional optimizers rely on predefined rules and cost formulas, modern DBMS incorporate machine learning, real-time statistics feedback, and automated plan correction mechanisms to dynamically adapt query execution to changing workloads and data distribution patterns.

One of the key directions in this area is adaptive query processing. Microsoft SQL Server introduced Adaptive Query Processing (AQP) and Intelligent Query

Processing (IQP) features, which allow the optimizer to monitor query execution and automatically adjust strategies in real time. Techniques such as adaptive joins, memory-grant feedback, and automatic statistics learning enable the system to switch between execution strategies based on live performance conditions. Empirical evaluations from Microsoft (2023) demonstrated that these mechanisms reduce query latency by up to 70% in workloads with unstable cardinality estimates and skewed data distribution [5].

Another significant technology shift involves the use of machine learning for automatic query tuning. Commercial and research systems, including Oracle Autonomous Database and SQL Server Automatic Tuning, integrate ML-based engines capable of identifying suboptimal plans, recommending index creation or removal, and forcing plan corrections without administrator intervention. Binnig et al. (2022) described a self-driving query engine capable of continuously learning from query logs and adapting execution plans to achieve highly stable and efficient workloads in cloud database environments [6]. Such systems demonstrate substantial improvements in response time and plan stability, particularly in multi-tenant and elastic cloud settings.

Hybrid optimization architectures represent the next step. These frameworks combine rule-based optimizers, cost models, and machine learning modules for plan prediction, anomaly detection, and workload forecasting. They are capable of identifying inefficient patterns caused by user-generated dynamic SQL, detecting potential security anomalies related to query structure, and preventing resource-exhaustion behaviors characteristic of malicious access attempts. The integration of ML-based behavior monitoring with classical plan tuning mechanisms marks a transition toward holistic workload-aware and security-aware optimization.

The main advantage of intelligent optimization lies in its ability to automatically recognize workload characteristics, detect performance anomalies, and correct execution strategies without manual tuning. Unlike classical methods, where the database administrator must carefully design indexes and analyze execution plans, modern systems autonomously identify correlations between data

patterns, query structures, and performance metrics. Moreover, these techniques provide additional security benefits by detecting suspicious execution plans and preventing exploitation of inefficient query logic.

Modern intelligent optimization systems form the foundation of next-generation database architectures. They allow combining statistical models, execution monitoring, and machine learning to achieve high performance, increased plan stability, and adaptive protection against query-based attacks. However, the complexity of algorithms and the dependence on high-quality telemetry data lead researchers to explore hybrid models that integrate autonomous tuning engines with traditional DBA-driven controls. This trend bridges the gap between classical query engineering and fully autonomous self-tuning secure database platforms, which will be further developed in future research.

Current challenges and trends in query optimization and database security development. Despite the active development of database management systems and the emergence of intelligent optimization mechanisms, ensuring high query performance while maintaining security remains a complex and dynamically evolving task. The growth of data volumes, cloud deployment models, and increasingly sophisticated attack techniques pose new challenges for researchers and database engineers. Key issues and promising directions are highlighted below:

One of the major challenges is maintaining the relevance of optimization strategies under rapidly changing workloads. Static cost-based models, if not regularly updated and adapted, may lead to suboptimal execution plans as the statistical distribution of data evolves. Modern DBMS therefore adopt adaptive optimization paradigms, where execution plans are refined during runtime [3–5]. However, as Connolly and Begg note [1], even advanced query planners may experience degradation when system workloads deviate significantly from historical patterns.

Another significant problem concerns the balance between performance and security. Traditional query optimization research primarily focused on execution

efficiency, while security aspects — such as resistance to inference attacks, resource exhaustion queries, and SQL injection patterns — were considered secondary. Contemporary studies emphasize that high-performance queries must also be secure by design [2,5]. When database systems prioritize performance without proper safeguards, attackers can exploit optimization gaps, forcing expensive execution paths or bypassing checks through parameter manipulation.

An additional challenge is the lack of universally labeled performance-security datasets for training intelligent optimization models. While cloud-driven platforms increasingly rely on machine learning to automate tuning, real-world training data remains proprietary and heterogeneous, limiting model generalization [6]. Researchers working on distributed query optimization also highlight difficulties in building scalable optimization models that operate effectively across multiple nodes and heterogeneous environments [7].

Promising development directions include adaptive and self-tuning architectures. Online optimization and feedback-driven execution mechanisms allow DBMS to automatically refine estimates, update statistics, and adjust resource allocations in real time [5]. Cloud-oriented frameworks extend this principle by incorporating learning-based optimization components that continuously improve using workload history [6]. Another priority is the integration of security mechanisms into the optimizer: techniques for detecting anomalous query behavior, ensuring secure parameterization, and analyzing access patterns to mitigate attack vectors [2,5].

Thus, the evolution of query optimization increasingly focuses on achieving both efficiency and resilience. Adaptive cost models, intelligent tuning systems, and integrated security-aware optimization frameworks represent the key trajectory of modern DBMS development. These technologies form the foundation for next-generation database engines capable of autonomously optimizing performance while proactively resisting query-driven threats [7].

Conclusion. In this study, modern methods and approaches to query optimization in database systems were examined, with particular attention to their

impact on both performance and security. It has been shown that classical optimization techniques — including cost-based models, indexing strategies, and rule-based execution planning — remain fundamental due to their maturity, transparency, and proven efficiency in traditional database environments [1–2]. However, their effectiveness is limited when processing large-scale, heterogeneous data and dynamic cloud workloads.

The analysis demonstrated that contemporary DBMS platforms such as Oracle, PostgreSQL, and Microsoft SQL Server are actively integrating adaptive and intelligent optimization mechanisms, including runtime feedback loops, dynamic plan correction, and automated tuning systems [3–5]. These methods enable more flexible and accurate execution plan selection, reducing performance degradation under changing workload patterns and mitigating risks associated with performance-based attacks such as resource exhaustion or inference-driven query exploitation.

It was also identified that the development of secure and efficient optimization frameworks faces several key challenges, including limited availability of training datasets for machine-learning-based optimization, the high variability of modern workloads, and the need to maintain transparency and interpretability in critical database applications [6–7]. Despite these limitations, promising directions include the adoption of self-tuning architectures, security-aware optimization models, and learning-enhanced execution frameworks that can autonomously adapt to workload shifts while detecting anomalous or malicious query patterns.

Overall, modern query optimization technologies are moving toward intelligent, adaptive, and security-integrated architectures. These systems are capable not only of improving execution performance, but also of proactively identifying and mitigating query-based security threats. The combination of classical optimization principles, machine-learning-driven techniques, and runtime behavior analysis sets a strategic trajectory for the development of next-generation

database management systems, ensuring both performance scalability and robust protection of critical data environments [2].

Literature:

1. Connolly, T., Begg, C. "Database Systems: A Practical Approach to Design, Implementation, and Management", 6th Edition. Pearson, 2015.
2. Elmasri, R., Navathe, S. "Fundamentals of Database Systems", 7th Edition. Pearson, 2016.
3. Oracle Corporation. "Oracle Database Performance Tuning Guide", Oracle, 2022.
4. PostgreSQL Global Development Group. "PostgreSQL Documentation", Release 15, 2023.
5. Microsoft Corporation. "SQL Server Query Performance Tuning", Microsoft Docs, 2023.
6. Binnig, C., Kossmann, D., Kraska, T., Zamanian, E. "A Framework for Adaptable Query Processing in the Cloud", Proceedings of the VLDB Endowment, 2016.
7. Kim, W., Kim, J., Lee, C. "Optimization Techniques for SQL Queries in Distributed Databases", IEEE Access, 2020.