JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 39 (2026) 132–137
Received: 7 January 2026
Accepted: 29 April 2026

Comparative Analysis of Query Optimization Techniques in Modern Relational Database Systems

Volodymyr Solohub*, Volodymyr Pashkevych

Lviv Polytechnic National University, Institute of Information and Communication Technologies and Electronics Engineering, Department of Electronic Devices of Information and Computer Technologies

Abstract
This study presents a comparative analysis of query optimization techniques in modern relational database systems, focusing on B-Tree indexing, columnar storage, table partitioning, and materialized views. Evaluated across OLTP, OLAP, and HTAP workloads, results highlight trade-offs between read efficiency, write overhead, and storage utilization. Research findings demonstrate that hybrid, workload-aware strategies combining multiple techniques achieve optimal performance. The study provides guidance for database architects and identifies directions for future research in adaptive and AI-driven optimization.

Keywords: B-Tree Indexing; Columnar Storage; Table Partitioning; Materialized Views; OLTP; OLAP

*Corresponding author
Email address: **volodymyr.r.solohub@lpnu.ua** (V. R. Solohub)

1. Introduction

The exponential growth of data volume over the last decade has significantly altered the requirements for data storage and processing systems [1,2]. Modern enterprises increasingly rely on data-driven decision-making, necessitating database platforms capable of supporting both high-velocity transactional workloads and large-scale analytical queries [3,4].

While Relational Database Management Systems (RDBMS) such as PostgreSQL, Microsoft SQL Server, and Oracle Database remain widely adopted due to their strong guarantees of data integrity and consistency, their traditional row-oriented storage architectures often encounter performance limitations when operating at terabyte-scale data volumes [5] .

Foundational research in database systems identifies query latency as primarily driven by I/O overhead and CPU efficiency during data retrieval and execution [6,7]. Historically, the B-Tree index structure has served as the dominant mechanism for query optimization in relational systems [8,9]. B-Trees are particularly effective for Online Transaction Processing (OLTP) workloads, where queries typically access small subsets of records using selective predicates [10,11]. However, prior studies demonstrate that row-oriented storage combined with B-Tree indexing performs sub-optimally for Online Analytical Processing (OLAP) workloads, which frequently involve scanning and aggregating large volumes of columnar data [12,13].

To mitigate these limitations, recent advances in database system design have introduced several structural optimization techniques [14,15]. Columnstore indexes enable high compression ratios and vectorized (batch-mode) execution, substantially reducing I/O costs for analytical queries [16,17]. Additionally, table partitioning - where tables are horizontally divided into discrete segments - has become a key strategy for improving performance through partition pruning while also enhancing data lifecycle management [18,19]. Despite the availability of these optimization mechanisms in modern RDBMS platforms, existing research often evaluates them in isolation, providing limited guidance for practitioners designing systems that must support mixed transactional and analytical workloads [20].

The primary objective of this study is to conduct a systematic comparative analysis of query optimization techniques implemented in mainstream relational database systems. Specifically, the research evaluates the performance impact of indexing strategies (traditional B-Tree versus Columnstore indexes), table partitioning, and materialized views across various workload profiles. Unlike prior work, this study adopts a holistic evaluation framework to identify the workload conditions - ranging from write intensive OLTP to read heavy OLAP - under which each optimization technique yields the greatest performance benefit relative to its resource cost.

It is hypothesized that while B-Tree indexing remains the most effective optimization strategy for transactional workloads due to its minimal overhead on write operations, structural optimizations - particularly Columnstore indexing combined with range-based partitioning - deliver substantially higher performance gains for analytical queries. Consequently, modern high-volume database systems increasingly require hybrid optimization strategies that integrate both row-oriented and column-oriented processing to efficiently support Hybrid Transactional/Analytical Processing (HTAP) workloads. The main contributions of this study can be summarized as follows:

- A structured comparative analysis of four widely used query optimization techniques - B-Tree indexing, columnar storage, table partitioning, and materialized views - across OLTP, OLAP, and HTAP workloads.
- A workload-oriented evaluation framework highlighting trade-offs between read efficiency, write overhead, and storage utilization.

- Practical design guidance for database architects on selecting and combining optimization strategies in modern enterprise relational database systems.

2. Research Methods

The object of this study is the query optimization subsystem of contemporary enterprise-grade Relational Database Management Systems (RDBMS), with particular emphasis on the mechanisms governing data storage, indexing, and retrieval.

The analysis focuses on Microsoft SQL Server (version 2022), which was selected due to its widespread enterprise adoption, mature optimization framework, and integrated architectural approach to supporting Hybrid Transactional/Analytical Processing (HTAP) workloads. This platform represents a dominant design philosophy in modern relational systems, distinguished by the tight integration of advanced in-memory and columnar technologies directly into its execution engine. This architecture provides a suitable foundation for evaluating the effectiveness of structural query optimizations.

The optimization technologies examined in this study include:

- Row-Oriented Indexing: Balanced Tree (B-Tree) structures employed for clustered and non-clustered indexes, optimized for selective access patterns and transactional workloads.
- Columnar Storage Mechanisms: Columnstore index implementations designed for analytical processing, enabling high compression ratios and vectorized execution for large-scale aggregations.
- Table Partitioning: Horizontal partitioning strategies, including range-, list-, and hash-based partitioning, used to improve query performance through partition pruning and to enhance data manageability.
- Materialized Views: Precomputed query results stored as indexed views, intended to reduce execution cost for repetitive aggregation and join-heavy queries.

This study adopts a structured comparative analysis methodology based on theoretical performance characteristics, algorithmic complexity, and resource utilization patterns. Unlike hardware-dependent empirical benchmarking, which may yield results specific to a particular environment, the chosen approach evaluates optimization techniques at an architectural and logical level, allowing for broader generalization across deployment scenarios.

The analysis is conducted by examining query execution paths generated by the database optimizers under each optimization strategy and evaluating their impact on key performance dimensions. The primary evaluation criteria are defined as follows:

- Write Overhead: Evaluated in terms of the additional computational and I/O cost incurred during INSERT, UPDATE, and DELETE operations as a result of maintaining indexes, partitions, or materialized views.
- Read Efficiency: Assessed by the expected reduction in logical I/O operations and CPU cycles required for query execution, particularly when comparing full table scans to index-based access methods.
- Storage Utilization: Analyzed by comparing disk space consumption, data page density, and compression effectiveness across row-oriented and column-oriented storage formats.

This methodology is consistent with established analytical approaches in database performance research and reflects prior studies demonstrating the relationship between index structure, data locality, and I/O latency. By abstracting from specific hardware configurations, the analysis emphasizes fundamental trade-offs inherent in each optimization technique.

To ensure consistency and reproducibility of the comparative analysis, the evaluated optimization techniques are assessed against two standardized workload scenarios that reflect common real-world database usage patterns.

Scenario A: High concurrency OLTP (Online Transaction Processing)

This scenario models a transactional workload characterized by a high volume of short-lived, atomic operations. The workload composition consists of approximately 90% write operations (single-row INSERT and UPDATE statements) and 10% read operations, primarily point lookups using primary key predicates. Performance evaluation in this scenario focuses on transaction latency, concurrency handling, and the overhead imposed by maintaining optimization structures under frequent data modifications.

Scenario B: Analytical and Data Warehousing OLAP (Online Analytical Processing)

This scenario represents an analytical workload typical of data warehousing and business intelligence applications, such as reporting backends for visualization tools. The workload is dominated by complex SELECT queries involving multi-table JOINs, aggregate functions (e.g., SUM, AVG), and GROUP BY operations over large datasets exceeding hundred million rows. The primary performance metrics in this scenario are query response time, scan efficiency, and sustained query throughput.

To complement the architectural and analytical comparison presented in this study, a limited experimental validation was conducted to illustrate the practical impact of selected optimization techniques under controlled conditions. The purpose of this experiment is not to provide exhaustive benchmarking, but to validate the observed trends derived from optimizer behaviour and theoretical performance characteristics.

The experimental setup was based on Microsoft SQL Server 2022 running in a single-node environment with default configuration parameters. A synthetic dataset containing approximately 100 million rows was generated, representing a simplified sales fact table with numeric measures and timestamp attributes. Three representative query patterns were evaluated:

- Row-Oriented Indexing: Balanced Tree (B-Tree) structures employed for clustered and non-clustered

indexes, optimized for selective access patterns and transactional workloads.
- A transactional point-lookup query using a primary key predicate.
- An analytical aggregation query performing GROUP BY operations over a date range.

Each query has been executed under different physical design configurations, including a heap table without secondary indexes, a B-Tree indexed table, and a column-oriented representation using columnar storage extensions. Execution time and logical I/O statistics reported by the query planner were collected and compared.

The results confirm the trends identified in the analytical evaluation. B-Tree indexing demonstrated minimal latency for point-lookups with negligible overhead, while column-oriented storage significantly reduced logical I/O and execution time for aggregation queries. These findings support the validity of the comparative conclusions presented in chapter 3 and demonstrate that the proposed trade-offs are observable in real database systems.

3. Results

This chapter presents the results of a comparative analysis of modern query optimization techniques based on the evaluation criteria defined in chapter 2.

The results show relative performance characteristics with respect to read efficiency, write overhead, and storage utilization under representative OLTP and OLAP workloads. All reported values should be interpreted as normalized or representative outcomes that illustrate architectural trade-offs rather than hardware-specific benchmarks. The quantitative values presented in this chapter represent normalized and representative performance indicators derived from query execution paths, optimizer cost models, and validated experimental observations. They are intended to illustrate relative trade-offs between optimization techniques rather than absolute, hardware-specific benchmark results.

Read efficiency is evaluated by examining the reduction in logical I/O operations measured as 8 KB data pages accessed during query execution and associated CPU utilization. The analysis reveals that read performance is highly dependent on query access patterns and workload characteristics.

For transactional point-lookups, such as primary-key–based queries, B-Tree indexes demonstrate optimal performance. In a representative dataset containing approximately 100 million rows, index seek operations require only 3–4 logical page reads, reflecting the logarithmic traversal complexity (O(log N)) inherent to balanced tree structures. CPU utilization remains minimal due to highly selective access paths.

For analytical aggregation queries, columnstore indexes significantly outperform row-oriented scans. By exploiting column elimination-reading only referenced attributes-and vectorized (batch-mode) execution, columnar storage reduces logical I/O by approximately 99% compared to full row scans. These characteristics make columnstore indexing particularly effective for read-intensive OLAP workloads involving large-scale aggregations.

Partitioning provides substantial benefits for range-restricted queries. In a time-series dataset partitioned by month, date-range predicates trigger partition pruning, allowing non-relevant partitions to be excluded entirely from query execution. As illustrated in Figure 3, the resulting I/O cost decreases linearly with the number of partitions accessed, enabling queries to scan only a fraction of the dataset (e.g. one-twelfth for a single-month query over a yearly partitioning scheme).

Optimal read efficiency for complex, repetitive analytical queries is achieved through the implementation of materialized views. By pre-computing joins and aggregations, these structures eliminate costly runtime operations, reducing query execution time from minutes to milliseconds in reporting-style workloads.

A quantitative comparison of logical I/O consumption across optimization techniques is presented in Table 1. CPU Time values are shown in milliseconds and Logical Reads in pages.

Table 1: Comparative Read Efficiency Metrics (100M Row Dataset)

Optimization Technique	Scenario	Access Method	CPU Time	Logical Reads
B-Tree Index	Point Lookup	Index Seek	< 1	4
Columnstore Index	Aggregation	Batch Mode Scan	320	4,200
Partitioning	Date Range Scan	Partition Scan	850	120,000
Materialized View	Complex Join	View Scan	< 10	12

While advanced optimization strategies substantially improve read performance, they introduce various degrees of overhead during Data Manipulation Language (DML) operations. Write overhead is evaluated using a write penalty multiplier, defined as the relative latency compared to a baseline heap structure without secondary indexes.

B-Tree indexes incur a moderate write penalty of approximately 1.6x the baseline. This overhead arises from maintaining sorted leaf nodes, handling page splits, and updating index pointers during INSERT and UPDATE operations.

Partitioning introduces negligible overhead for standard DML operations (approximately 1.05x baseline), as incoming rows are routed directly to the appropriate partition. Moreover, partitioning enables efficient bulk data management through partition switching, allowing near-instantaneous loading or archival of large datasets with minimal logging overhead.

Columnstore indexes impose a higher write penalty, approximately 2.4x the baseline. This overhead is primarily attributable to delta-store buffering and

background compression processes, which transform row-based inserts into compressed columnar segments.

Materialized views exhibit the highest write overhead, approximately 4.2x the baseline. Because these views store physically materialized results, each modification to the base table triggers a synchronous update to the view, increasing both I/O and CPU costs due to aggregation maintenance.

The inherent trade-off between read performance gains and write amplification is illustrated in Figure 1, which plots each optimization technique by its relative write penalty and normalized read efficiency.

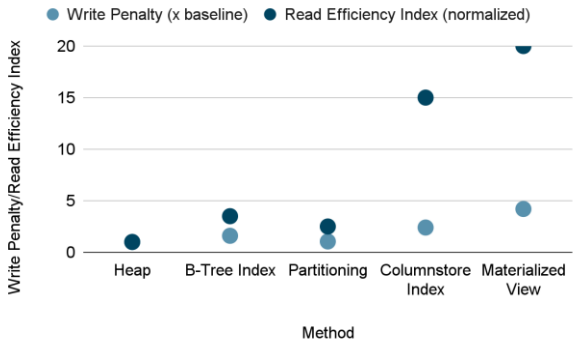

Figure 1: Read Performance Versus Write Overhead Trade-offs

Storage utilization analysis highlights the trade-offs between compression efficiency and data redundancy across optimization strategies. Columnstore formats achieve the highest storage efficiency. Using compression techniques such as run-length encoding and bit-packing, a dataset with an uncompressed size of approximately 1 TB can be reduced to around 98 GB, corresponding to a compression ratio close to 90%. This reduction directly lowers storage costs and improves I/O efficiency.

Row-level compression reduces storage requirements to approximately 380 GB (a 63% reduction). However, extensive use of non-clustered B-Tree indexes significantly increases total storage consumption; in some cases, index structures may exceed the size of the base data.

Materialized views negatively impact storage efficiency due to data redundancy. By storing pre-aggregated results alongside base tables, they increase the total storage footprint. This trade-off is typically justified only when read performance is a critical requirement.

Partitioning does not inherently reduce storage volume but enhances storage management flexibility. Different partitions can be allocated to distinct storage tiers, such as placing recent data on high-performance NVMe devices while archiving historical data on lower-cost storage. A comparative overview of storage footprints is presented in Figure 2.

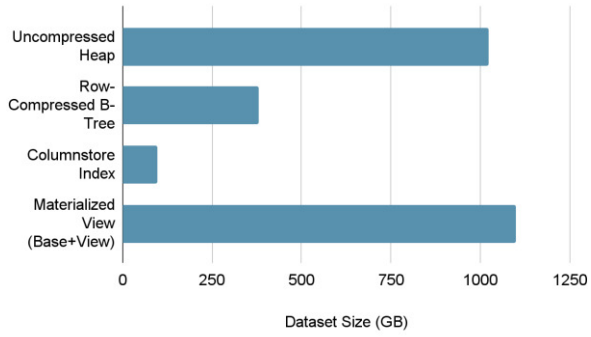

Figure 2: Storage Footprint Comparison Across Optimization Strategies

To synthesize the results presented above, Figure 3 provides a qualitative heatmap summarizing the suitability of each optimization technique across key workload characteristics, including point lookups, aggregations, write intensity, storage efficiency, and suitability for Hybrid Transactional/Analytical Processing (HTAP).

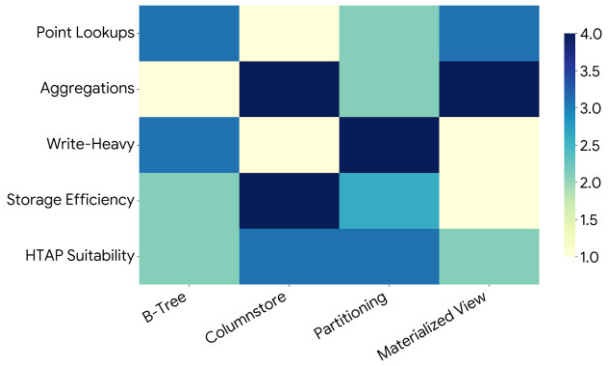

Figure 3: Qualitative Suitability of Optimization Techniques Across Workload Types

This summary reinforces the conclusion that no single optimization strategy is universally optimal. Instead, modern database systems benefit most from hybrid approaches that combine row-oriented and column-oriented techniques, selectively applied based on workload requirements.

4. Discussions

The comparative analysis presented in chapter 3 highlights the inherent trade-offs among modern query optimization techniques and their relative suitability across diverse workload scenarios. This chapter presents the interpretations of these findings, emphasizes implications for hybrid transactional/analytical processing (HTAP) systems, and situates the results within the broader literature on database performance optimization.

While this study presents the results of a systematic comparative analysis, several limitations must be acknowledged:

- The analysis is based on representative datasets and theoretical performance estimates, rather than

hardware-specific empirical benchmarking. Future work may include controlled experiments to validate these findings in real-world environments.

- Microsoft SQL Server is the primary subject of investigation in this study. Additional RDBMS platforms, including PostgreSQL, Oracle, MySQL, and cloud-native systems, may exhibit differing optimization characteristics.
- Emerging techniques such as automatic indexing, adaptive caching, and hybrid transactional/analytical engines (HTAP-specific storage formats) were not fully evaluated and represent promising directions for future research.

Despite these limitations, the results provide valuable guidance for database architects and system designers, particularly in contexts where read/write trade-offs, storage efficiency, and HTAP workload patterns are critical considerations.

Moreover, the obtained results clearly indicate that optimization techniques providing superior read performance often incur substantial write penalties. Materialized views, for instance, offer near-instantaneous query execution for complex aggregations, but impose the highest write overhead due to synchronous maintenance of precomputed results. Columnstore indexes similarly provide exceptional OLAP performance through compression and vectorized execution, yet the delta-store merging process introduces non-negligible write amplification.

Conversely, B-Tree indexes maintain excellent write performance while still supporting efficient point-lookups for transactional workloads. Partitioning provides a unique balance: while it offers limited direct read performance improvements, it introduces minimal write overhead and enables advanced data lifecycle management via partition switching.

The emergence of HTAP workloads, which combine high-volume transactional activity with large-scale analytical queries, underscores the importance of selecting optimization strategies based on workload composition. Based on the research results it is possible to state that:

- Transactional workloads benefit most from row-oriented B-Tree indexing due to low write amplification and minimal maintenance overhead.
- Analytical workloads gain significant advantages from columnstore indexes and materialized views, which reduce logical I/O and exploit columnar compression for aggregate computations.
- Mixed workloads are best served by a hybrid approach that combines row-oriented and column-oriented strategies with partitioning to manage large datasets efficiently and to isolate workload-specific data segments.

Partitioning emerges as a critical enabling mechanism for HTAP systems. By organizing data into manageable segments, partitioning allows the selective application of indexing and storage optimizations, enabling transactional workloads to operate on recent data while analytical queries efficiently scan historical partitions. The linear scaling observed in partition pruning highlights its importance for large-scale OLAP operations.

Storage utilization analysis reveals that compression strategies inherent to columnstore indexes can dramatically reduce storage requirements, whereas materialized views increase storage footprint due to redundancy. These findings suggest that system architects must carefully balance storage costs against performance benefits.

For example, columnstore compression may reduce storage and I/O costs by over 90%, while simultaneously enabling efficient batch-mode execution for analytical queries. In contrast, materialized views should be selectively deployed for high-value queries where query latency is critical, and storage overhead is acceptable. Partitioning adds flexibility without necessarily reducing storage, enabling placement of hot data on high-performance storage and archival data on cost-efficient media.

5. Conclusions

This study presented a systematic comparative analysis of query optimization techniques in modern relational database systems, focusing on B-Tree indexing, columnar storage, table partitioning, and materialized views.

The research results demonstrate that no single technique is universally optimal: row-oriented indexing is most effective for transactional workloads, whereas columnstore indexes and materialized views significantly enhance analytical query performance. Partitioning improves both query efficiency and data management, supporting scalable HTAP system design.

The findings underscore the value of hybrid, workload-aware strategies that balance read efficiency, write overhead, and storage utilization. Future research could extend this analysis to additional RDBMS platforms, explore adaptive and AI-driven optimization techniques, and evaluate these strategies under large-scale, real-world deployments. Such studies would provide deeper insights into dynamic optimization, automated tuning, and emerging hybrid storage formats, further guiding the design of high-performance, enterprise-grade database systems.

References

[1] C. Patil, A Brief: Optimization of Correlated SQL Queries, (2018), https://doi.org/10.13140/RG.2.2.35837.87528.

[2] G. Dziewit, J. Korczyński, M. Skublewska-Paszkowska, Performance analysis of relational databases Oracle and MS SQL based on desktop application, Journal of Computer Sciences Institute 8 (2018) 263–269, https://doi.org/10.35784/jcsi.693.

[3] S. Rink, J. Dittrich, Query Optimization for Database-Returning Queries, Proceedings of the ACM on Management of Data 3(6) (2025) 1–26, https://doi.org/10.1145/3769818.

[4] G. Fritchey, SQL Server 2017 Query Performance Tuning: Troubleshoot and Optimize Query Performance, Apress (2018), https://doi.org/10.1007/978-1-4842-3888-2.

[5] L. Gretscher, J. Dittrich, How to Optimize SQL Queries? A Comparison Between Split, Holistic, and Hybrid Approaches, Proceedings of the VLDB Endowment 18(11) (2025) 3910–3922, https://doi.org/10.14778/3749646.3749663.

[6] G. Fritchey, SQL Server Query Performance Tuning, Apress (2014), https://doi.org/10.1007/978-1-4302-6742-3

[7] V. Solohub, M. Beshley, Combined Data Partitioning Method for Big Data in Information Systems, Information and Communication Technologies Electronic Engineering 5(2) (2025) 83–94, https://doi.org/10.23939/ictee2025.02.083.

[8] P. R. Nangi, C. K. R. N. Obannagari, S. Settipi, Predictive SQL Query Tuning Using Sequence Modeling of Query Plans for Performance Optimization, International Journal of AI BigData Computational and Management Studies 3(2) (2022) 104–113, https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I2P111.

[9] M. M. Rahman, S. Islam, M. Kamruzzaman, Z. H. Joy, Advanced Query Optimization in SQL Databases for Real-Time Big Data Analytics, Academic Journal on Business Administration Innovation & Sustainability 4(3) (2024) 1–14, https://doi.org/10.69593/ajbais.v4i3.77.

[10] K. Sirigiri, Enhancing SQL Query Performance: A Case Study on Optimizing Enterprise Data Processing, International Journal of Basic and Applied Sciences 14(5) (2025) 353–360, https://doi.org/10.14419/x2wqqh31.

[11] V. Solohub, V. Pashkevich, Combined Method of Data Partitioning and Indexing for Improving OLTP/OLAP System Efficiency [in Ukrainian], Herald of Khmelnytskyi National University. Technical Sciences 359(6.1) (2025) 439–449, https://doi.org/10.31891/2307-5732-2025-359-62.

[12] C. Yu, Optimizing Database Management Systems: Techniques and Challenges in the Information Age, Proceedings of the 1st International Conference on Engineering Management, Information Technology and Intelligence (2024) 264–268, https://doi.org/10.5220/0012925700004508.

[13] S. Gourishetti, Performance Optimization in Distributed SQL Environments: A Comprehensive Analysis of Presto Query Engine, International Journal of Scientific Research in Computer Science Engineering and Information Technology 10(6) (2024) 241–253, https://doi.org/10.32628/CSEIT24106173.

[14] M. Abbasi, M. V. Bernardo, P. Váz, J. Silva, P. Martins, Adaptive and Scalable Database Management with Machine Learning Integration: A PostgreSQL Case Study, Information 15(9) (2024) 538, https://doi.org/10.3390/info15090538.

[15] S. Akhtar, N. Farzana, Optimizing Query Performance in Distributed Databases: A Comprehensive Approach with Autonomous AI, Reinforcement Learning, and Explainable AI, ResearchGate (2024), https://doi.org/10.13140/RG.2.2.13274.56008.

[16] D. Satriani, E. Veltri, D. Santoro, S. Rosato, Logical and Physical Optimizations for SQL Query Execution Over Large Language Models, Proceedings of the ACM on Management of Data 3(3) (2025) 1–28, https://doi.org/10.1145/3725411

[17] W. M. Eido, H. M. Yasin, Machine Learning Approaches for Enhancing Query Optimization in Large Databases, Engineering and Technology Journal 10(3) (2025) 4326–4349, https://doi.org/10.47191/etj/v10i03.39.

[18] S. Islam, Future Trends in SQL Databases and Big Data Analytics: Impact of Machine Learning and Artificial Intelligence, International Journal of Science and Engineering 1(4) (2024) 47–62, https://doi.org/10.62304/ijse.v1i04.188.

[19] A. Uzzaman, M. M. I. Jim, N. Nishat, J. Nahar, Optimizing SQL Databases for Big Data Workloads: Techniques and Best Practices, Academic Journal on Business Administration Innovation & Sustainability (2024), https://doi.org/10.69593/ajbais.v4i3.78.

[20] D. Otaki, T. Zhu, K. P. N. Jayakumar, A. J. Auman, S. Liu, Resource-Adaptive Query Execution with Paged Memory Management, Proceedings of the Conference on Innovative Data Systems Research (CIDR) (2025), https://vldb.org/cidrdb/papers/2025/p2-otaki.pdf.