Jurnal Teknik Informatika (JUTIF) Vol. 6, No. 3, Juni 2025, Page. 1457-1468
P-ISSN: 2723-3863 https://jutif.if.unsoed.ac.id
E-ISSN: 2723-3871 DOI: https://doi.org/10.52436/1.jutif.2025.6.3.4523

Performance Optimization of ERD Designs Using Cost-Based Optimization for Large-Scale Query Processing

Juanda Hakim Lubis*[1], Sri Handayani[2], Herman Mawengkang[3], Fajrul Malik Aminullah Napitupulu[4]

[1,2]Information Technology, Universitas Pembinaan Masyarakat Indonesia, Indonesia
[3]Computer Science, Universitas Sumatera Utara, Indonesia
[4]Department of Computer Science, Institute Modern of Architecture and Technology, Indonesia

Email: [1]juandahakim@gmail.com

Received: Mar 16, 2025; Revised: Jun 17, 2025; Accepted: Jun 17, 2025; Published: Jun 23, 2025

Abstract

The rapid growth of stored data, particularly on magnetic disks, is doubling annually for each department within a company, creating a pressing need for efficient database management. While database design is a fundamental step in establishing a high-performance system, it alone is insufficient to ensure optimal efficiency. Query optimization plays a critical role in improving data transaction speed, reducing query execution time, and enhancing overall system responsiveness. This study evaluates various relational database models under different data volumes to analyze their impact on query performance. Using the Cost-Based Optimizer method and access time measurements, we assess query costs and determine the factors influencing performance. The results indicate that among the three database models analyzed, ERD-3 consistently delivers superior performance, especially in handling complex queries. This is attributed to its modular structure, strategic indexing, and reduced full table scans, which collectively minimize query execution costs. Additionally, several key factors significantly affect query performance, including record count, attribute size, query complexity, primary and unique key usage, indexing strategies, order-by clauses, index sequences, and SQL function application. This research contributes to the field of database optimization by demonstrating that ERD structuring and cost-based query analysis significantly improve system efficiency in large-scale environments. These findings emphasize the necessity of a well-structured, scalable database model and efficient query processing techniques to accommodate large-scale data growth. The study's conclusions provide a foundation for advanced optimization strategies, ensuring that modern database systems remain efficient and adaptable to evolving data demands.

Keywords: *Cost-Based Optimizer, Database Design, Query Optimization, Query Performance, Relational Database Model.*

1. INTRODUCTION

A Database system performance is an increasingly critical issue as computerized database systems grow larger and more complex. A survey conducted by the University of California at Berkeley highlighted that the amount of data stored on magnetic disks doubles annually per department and per company, underscoring the pressing need for organizations to maintain optimal database performance [1]. This rapid data growth necessitates the development of robust performance assessment and enhancement strategies tailored to specific system types, whether operational systems or Decision Support Systems (DSS). These strategies must account for the evolving nature of technology and the diverse objectives of various database applications [2].

The design and development of a database system directly influence its performance. Advanced computer technology can address performance bottlenecks, but when systems become obsolete, even the most skilled database administrators may struggle to maintain high-performance levels [3].

Moreover, the effectiveness of a database system depends significantly on its initial planning and design. Investing in robust database design from the outset is thus a critical factor in achieving sustained system efficiency [4].

Effective database design involves constructing a data model that accurately represents real-world scenarios while accommodating potential improvements. This foundational step ensures the system's adaptability to organizational needs and supports long-term performance optimization [5]. In practice, a well-structured database design can significantly reduce processing time for business operations, offering substantial performance gains in specific use cases [6].

Beyond initial design efforts, optimizing database performance often involves enhancing the speed of data transactions. One impactful approach is optimizing query processing speeds, which can boost performance by 25% to 100%, or even more in certain scenarios [7]. The strategic use of representative indexes, normalization processes, and table designs further contributes to improved system efficiency. These design principles can yield performance gains comparable to hardware enhancements, such as deploying high-performance storage solutions like RAID (Redundant Array of Inexpensive Disks) [1], [8].

Recent advancements have introduced models to enhance query performance, enabling database administrators to manage materialized views effectively and improve operational efficiency through faster report generation [4], [12]. Additionally, research has examined rule-based and cost-based optimization techniques, concluding that cost-based optimization offers a more efficient strategy by estimating and selecting execution plans with the lowest costs [13]. These studies highlight diverse approaches to improving database system performance through both software and hardware optimization.

Despite these advancements, there remains a need to explore the interaction between database design, storage technology, and query optimization in relational database systems. This research seeks to address this gap by investigating relational database models with varying data volumes, performing tests on Direct-Attached Storage (DAS) systems using identical queries, and analyzing query costs using cost-based optimization methods [14]. Additionally, disk access times will be evaluated to propose practical strategies for database administrators.

Query optimization is a crucial component in database management systems (DBMS) that significantly impacts performance. Various optimization techniques have been proposed in the literature to enhance the efficiency of query execution. Traditionally, query optimization relies on rule-based and cost-based approaches, where heuristics and estimated costs guide the selection of the optimal query execution plan [15]. These techniques focus on reducing query execution time by restructuring queries, leveraging indexes, and optimizing join operations. Early research in query optimization introduced fundamental strategies such as predicate pushdown, join reordering, and materialized views, which remain relevant in modern database systems [16].

One of the widely studied approaches is cost-based query optimization (CBO), where the optimizer estimates the cost of different query execution plans and selects the most efficient one. Selinger et al. [17] introduced the System R optimizer, which pioneered cost-based optimization by evaluating alternative join orders and access paths using statistical estimates. Since then, several enhancements have been made to CBO, including histograms for data distribution estimation [18] and machine learning techniques to improve cost estimation accuracy [19]. Despite its effectiveness, CBO suffers from high computational overhead, especially in complex queries involving multiple joins and subqueries.

Rule-based optimization (RBO) is another approach that applies predefined transformation rules to optimize queries. Unlike CBO, RBO does not consider cost estimates but instead relies on heuristic rules to rewrite queries into more efficient forms. System-generated rules, such as predicate pushdown and join elimination, are commonly used in commercial database systems [20]. While RBO is efficient for simple queries, it lacks adaptability to workload changes and complex query structures. Hybrid approaches combining RBO and CBO have been proposed to balance efficiency and adaptability [21].

With the advent of big data and distributed database systems, query optimization has extended to parallel and distributed environments. Modern distributed query optimizers employ strategies such as data partitioning, query decomposition, and distributed join processing to improve performance [22]. MapReduce-based query optimizers, such as those used in Apache Hive and Spark SQL, leverage parallel processing frameworks to execute queries efficiently over large datasets [23]. Recent research explores deep reinforcement learning techniques to optimize distributed query execution dynamically [24].

Additionally, adaptive query optimization (AQO) has gained attention as an approach to dynamically adjust execution plans based on runtime feedback. Traditional optimizers rely on static estimates, which may be inaccurate due to data distribution changes. AQO techniques, such as re-optimization and feedback-based learning, help mitigate this issue [25]. Database systems like IBM Db2 and PostgreSQL have incorporated adaptive optimization features to refine query execution over time [26]. As workloads become more unpredictable, AQO is expected to play a significant role in modern query optimizers.

Query optimization techniques have evolved from rule-based and cost-based approaches to more adaptive and distributed methodologies. With advancements in machine learning and AI-driven optimizations, future research is likely to explore more intelligent and autonomous optimization strategies [27]. As database workloads grow in complexity, optimizing query performance remains a critical area of research with ongoing developments in both traditional and emerging database architectures.

Ultimately, the findings of this research aim to provide actionable insights for database system administrators. By designing relational database models effectively, leveraging appropriate storage technologies, and optimizing queries for data retrieval, organizations can ensure sustained database performance. This research endeavors to contribute to the broader field of database optimization by offering practical guidelines for managing the complexities of modern database systems.

2. METHOD

This chapter outlines the research methodology employed to analyze query performance based on different database model designs and query costs. The study focuses on a company's inventory system and utilizes a simulation application designed specifically for evaluating query efficiency rather than user interface considerations. The primary performance metrics include query cost, measured in disk I/O operations, and response time, assessed in milliseconds.

The system workflow (Figure 1) illustrates the process of evaluating query performance and optimization using Cost-Based Optimization (CBO) applied to three relational model designs derived from ER diagrams (ER-1, ER-2, and ER-3). The process begins with inserting data at large scales (100K to 100M entities). After that, one of the three relational model designs is selected for testing.

Next, a set of queries is chosen—including Simple, Aggregate, Join, Subquery, and Complex queries—which are sent to the Plan Generator module to produce multiple alternative execution plans. These plans are then evaluated by the Cost Estimator using available data statistics. The optimizer selects the most efficient plan based on the estimated cost. The system then records the average response time and estimated cost for each query. The process concludes after evaluating all query combinations across the selected models and data volumes.

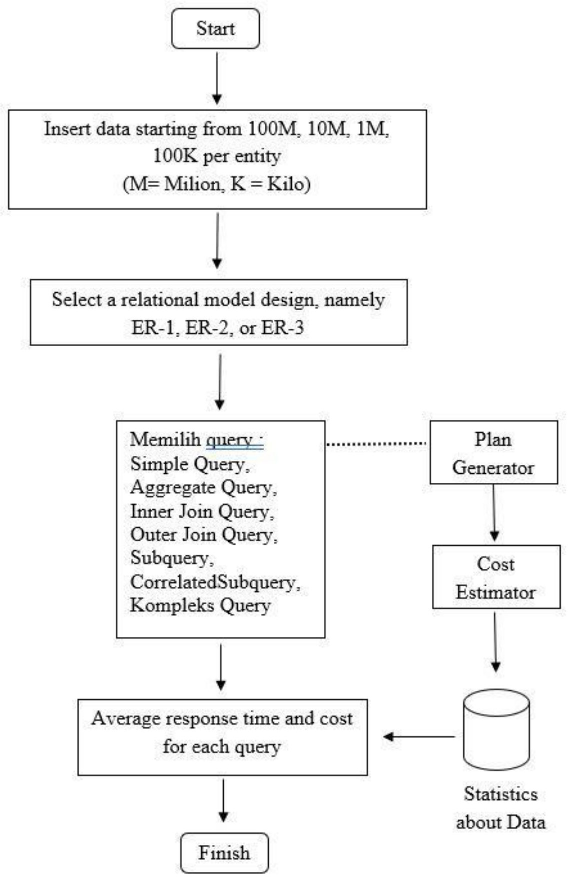
Start
Insert data starting from 100M, 10M, 1M, 100K per entity (M= Milion, K = Kilo)
Select a relational model design, namely ER-1, ER-2, or ER-3
Memilih query : Simple Query, Aggregate Query, Inner Join Query, Outer Join Query, Subquery, CorrelatedSubquery, Kompleks Query
Plan Generator
Cost Estimator
Statistics about Data
Average response time and cost for each query
Finish

Figure 1. System Workflow

2.1. Query Configuration & Hardware Setups

The research application is configured to handle a wide range of data sizes, varying from 100,000 to 100,000,000 entities, to simulate both small-scale and large-scale database operations. While the testing environment consists of a computer system with the following specifications:

1. Processor: Intel Core i3-3220 (3.3 GHz, dual-core, 4 threads).
2. Memory: 8 GB DDR3 RAM.
3. Storage: 2 × 500GB HDD (spinning disk drives).
4. Operating System: Windows 7 Ultimate (64-bit).
5. Database Management System: Oracle Database 11g Release 2.

These hardware and software configurations are selected to simulate real-world database environments where storage, processing power, and memory availability impact performance.

The study evaluates seven distinct query types, each representing a different aspect of database performance (Table 1).

Table 1. Query Operation Types

Query Types	Operation
Simple Query	Basic selection and retrieval of records.
Aggregate Query	Summarization using functions such as COUNT, SUM, AVG, MIN, and MAX.
Inner Join	Retrieving data across multiple tables with matching key relationships.
Outer Join	Combining related data while retaining unmatched records.
Subquery	Nested queries used for filtering and processing data conditions.

Correlated Subquery	Subqueries that reference columns from the outer query and execute row-by-row.
Complex Query	Queries combining multiple operations, including joins, aggregations, and subqueries

2.2. Database Design

To evaluate the impact of different database modeling strategies on query performance, three entity-relationship (ER) models are implemented:

1. ER-1 (Basic Model): A straightforward relational model including core entities such as goods, employees, materials, suppliers, transactions, and supply records.
2. ER-2 (IS-A Hierarchical Model): Implements an IS-A relationship within the material_doc entity to distinguish between different document types and goods classifications.
3. ER-3 (Separated Entity Model): Instead of using IS-A relationships, the material_doc entity is split into two distinct tables: material_doc_ambil (material retrieval) and material_doc_terima (material receipt).

These models are designed to assess how different approaches to data structuring impact query efficiency, index utilization, and overall system performance.

2.3. Testing Phase

Each database design undergoes extensive testing using the predefined query types. The evaluation process consists of:

1. Executing queries under different data loads to assess scalability.
2. Measuring the average response time (millisecond) to evaluate query execution speed.
3. Calculating the total query cost (disk I/O operations) to determine efficiency in data retrieval.

The study aims to provide actionable insights for database administrators, helping them choose optimal data structuring techniques to improve transaction speed, reduce computational overhead, and optimize resource usage. The experimental results serve as a benchmark for understanding the trade-offs between normalized and denormalized designs in real-world applications.

3. RESULT

3.1. ERD Performance Analysis in Various Query Types

The performance analysis of different query types highlights the efficiency of ERD-3 in handling large-scale datasets.

In **Simple Queries** (Figure 2), ERD-1 and ERD-3 outperform ERD-2, demonstrating better efficiency, particularly for large datasets, due to their use of direct index access rather than complex operations like nested loops.

For **Aggregate Queries** (Figure 3), ERD-3 again proves to be the best choice, with execution times up to 94% faster than ERD-2. ERD-1 struggles with full table scans, while ERD-2's performance is hindered by nested loops. Optimization using hash joins could enhance the efficiency of ERD-1 and ERD-2.

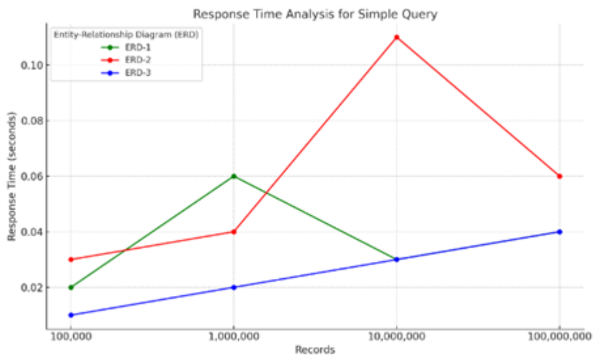

Figure 2. Simple Query

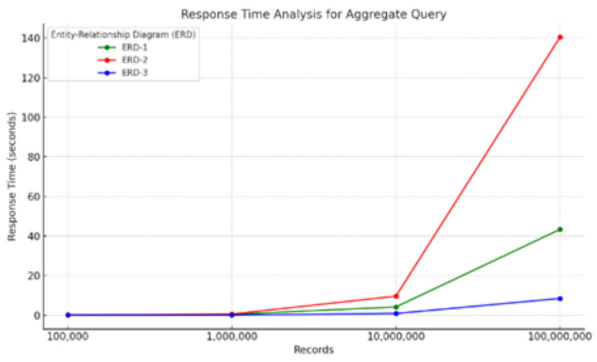

Figure 3. Aggregate Query

In **Inner Join Queries** (Figure 4), ERD-3 is the fastest, with response times 74% better than the other ERDs. This advantage stems from its optimized indexing and hash join utilization, whereas ERD-1 and ERD-2 rely heavily on full table scans, leading to slower performance.

Similarly, for **Outer Join Queries** (Figure 5), ERD-3 remains the most efficient, processing queries 71% faster than ERD-2. The slower response times of ERD-1 and ERD-2 result from excessive full table scans, which could be improved by reducing these operations and increasing index utilization.

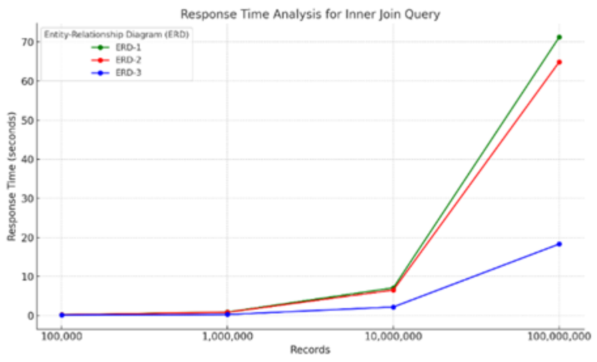

Figure 4. Inner Join Query

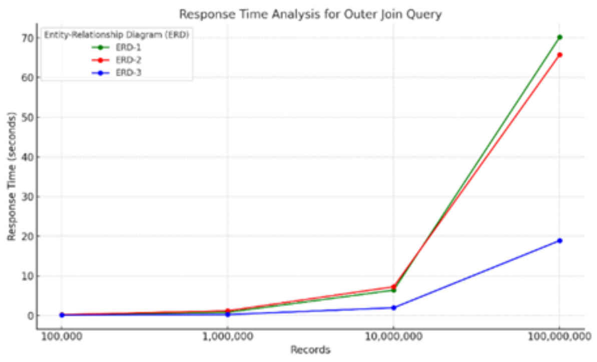

Figure 5. Outer Join Query

When evaluating **Subqueries** (Figure 6), all three ERDs display nearly identical performance, with ERD-2 slightly ahead (about 5% faster). The similarity in their query structures results in minimal differences, although index-based optimization could improve overall efficiency.

For **Correlated Queries** (Figure 7), ERD-3 maintains its dominance with execution times 21.5% faster than ERD-1 and 17.8% faster than ERD-2. This efficiency is attributed to its better index utilization, though all ERDs still rely on nested loops, which could be optimized further.

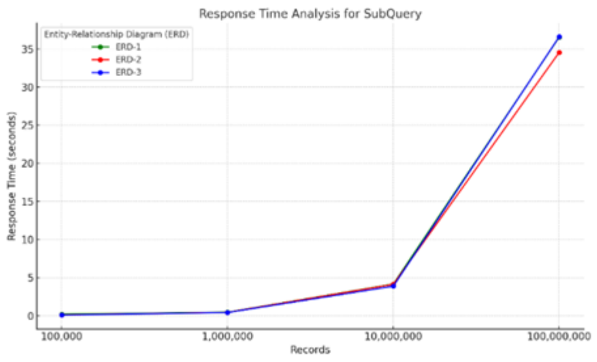

Figure 6. Subquery

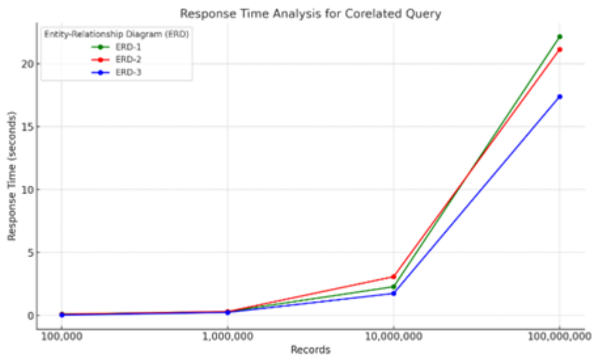

Figure 7. Corelated Query

In the case of **Complex Queries** (Figure 8), ERD-3 once again outperforms the others, being 34.4% faster than ERD-1 and 25.9% faster than ERD-2. ERD-3 achieves this through the reduction of full table scans and the frequent use of unique index scans, making it the most effective option. Meanwhile, ERD-1 and ERD-2 struggle with handling multi-table queries due to their heavier reliance on inefficient join operations.

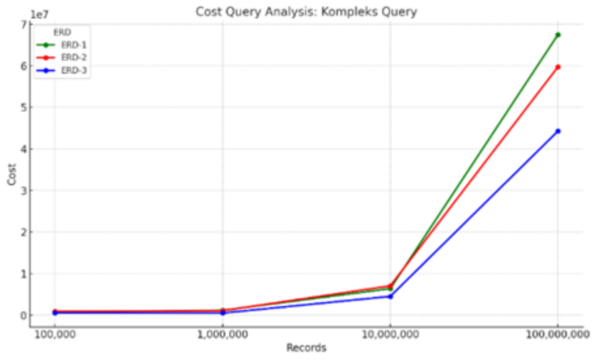

Figure 8. Complex Query

Overall (Figure 9), the findings indicate that ERD-3 is the best choice for processing large-scale data due to its modular and efficient design. While ERD-1 has a simple structure, it is inefficient in handling joins, particularly in many-to-many relationships. ERD-2 improves upon ERD-1 by introducing hierarchical relationships and better entity grouping, yet it still suffers from performance limitations due to the use of merge join cartesian and nested loops. In contrast, ERD-3 overcomes these inefficiencies by implementing a modular approach, optimizing relationships, and reducing reliance on full table scans. The consistent superiority of ERD-3 across all query types, especially for large datasets (100 million records), makes it the most scalable and efficient solution for complex data processing.

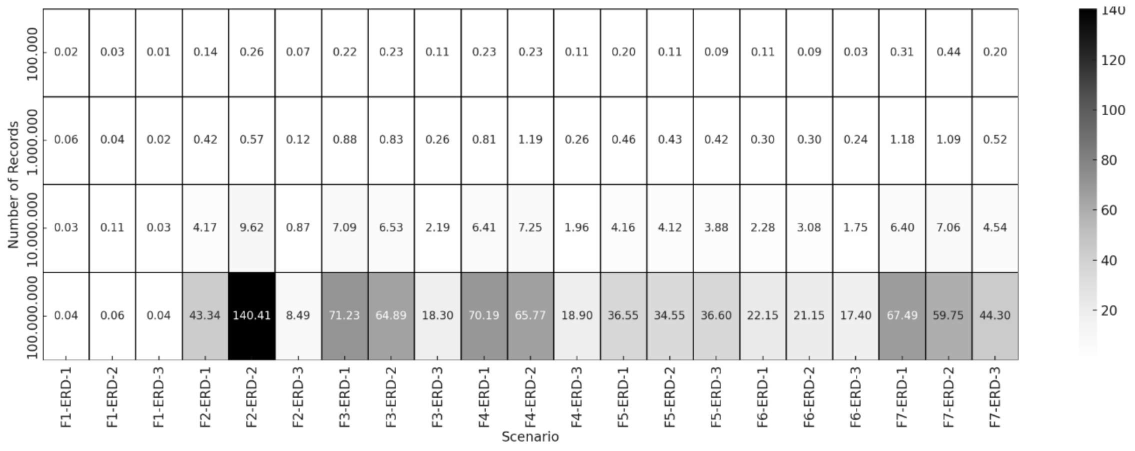

Figure 9. Response Time (second) per Scenario from Figure 2 to Figure 8

3.2. Performance Evaluation of ERD Designs Based on Cost-Based Optimization (CBO) Analysis

Based on the Cost-Based Optimization (CBO) analysis (Figure 10), ERD-3 consistently outperforms ERD-1 and ERD-2, particularly in handling high-complexity queries. In Simple Queries, ERD-1 and ERD-3 have the same query cost of 8, whereas ERD-2 incurs a cost of 13, making it 62.5% more expensive than the other two designs. The advantage of ERD-3 becomes even more evident in Aggregate Queries, where its cost is only 44,620, which is 79.4% lower than ERD-1 (217,208) and

92.7% lower than ERD-2 (608,290). This significant efficiency gain is attributed to ERD-3's modular design and optimized index usage, which enhances the performance of aggregate data calculations.

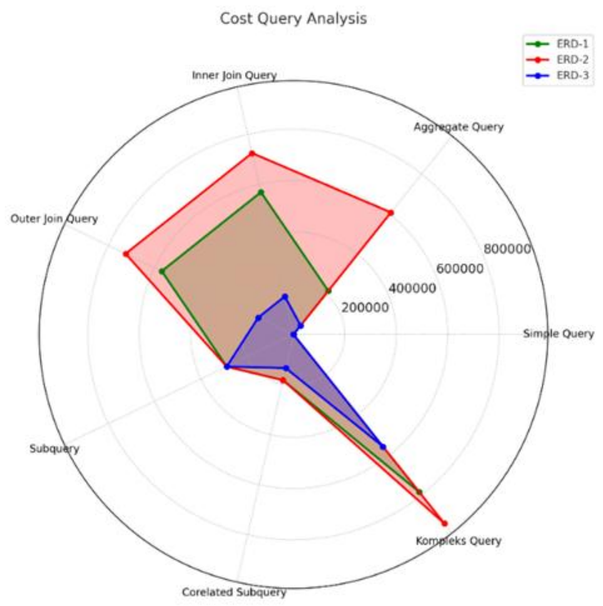

Figure 10. Time Cost for each Data and Query Types

For queries involving table joins, such as Inner Join and Outer Join Queries, ERD-3 again proves to be superior. Its query cost is 152,220, which is 73.2% lower than ERD-1 (568,896) and 79.0% lower than ERD-2 (725,082). This improvement is due to ERD-3's modular structure, where the Transactions entity is split into two separate entities, Transaksi_1 and Transaksi_2, reducing the complexity of join operations. For Correlated Subqueries, ERD-3 has a cost of 134,192, which is 26.5% lower than both ERD-1 and ERD-2, each of which has a cost of 182,444. In Complex Queries, ERD-3 remains the best performer with a cost of 558,058, which is 28.9% lower than ERD-1 (785,209) and 40.8% lower than ERD-2 (942,261). Overall, ERD-3's modular, efficient design, along with optimal index utilization, enables cost reductions of over 90% in certain query types. This makes ERD-3 the ideal choice for large-scale data processing systems.

3.3. Factors Influencing Query Cost and Performance

Beyond the CBO analysis, factors like query operations, attribute size, indexing, statistics, and complexity affect query cost and performance (Table 2).

Table 2. Factors Affecting Query Execution Performance

Factor Names	Effects
Query Operation	Projection and selection operations generally reduce execution costs, whereas join operations increase complexity and cost. The use of the ORDER BY clause can increase query cost due to the sorting process required.
Attribute Size	The smaller the attribute size in a projection operation, the lower the execution cost.
Index Utilization	Using a unique key or primary key with an equality (=) operator in selection operations helps avoid full table access. Irrelevant indexes or the use of SQL functions on columns in the WHERE clause may cause the optimizer to ignore indexing.

Database Statistics	Updated statistics allow the Cost-Based Optimizer (CBO) to generate more efficient execution plans.
Query Type and Complexity	Complex queries involving multiple joins or subqueries significantly increase execution cost, particularly on large datasets

4. DISCUSSIONS

Understanding these key factors enables organizations to optimize ERD design, leverage indexing efficiently, and maintain accurate database statistics, thereby improving query performance significantly across different data processing needs. Several optimization strategies can be employed:

1) Index Optimization.
   Implement indexing strategies that prioritize frequently queried fields. Use composite indexes where appropriate to support multi-column filtering and joining operations. Avoid redundant or unused indexes that increase maintenance overhead without improving performance.
2) Query Structure Refinement.
   Rewrite queries to minimize expensive operations such as full table scans and nested loops. Use appropriate join strategies such as hash joins instead of nested loops for large datasets. Optimize correlated subqueries by replacing them with joins or common table expressions (CTEs) when possible.
3) Efficient Table Design.
   Normalize data structures to reduce redundancy while ensuring that joins remain efficient. Decompose many-to-many relationships into intermediary tables to improve join performance. Use partitioning techniques to distribute large tables into smaller, more manageable segments. Compared to the findings of Lindner et al. [4], our approach using ERD-3 shows better scalability in cost reduction, suggesting a modular ERD structure as a practical design principle.
4) Use of Materialized Views.
   Precompute and store the results of expensive aggregation queries to improve response times. Refresh materialized views periodically to maintain data accuracy while reducing query execution time.
5) Caching Strategies.
   Implement query caching for frequently executed queries to minimize redundant computations. Use application-level caching mechanisms such as Redis or Memcached to store commonly accessed data.
6) Load Balancing and Resource Allocation.
   Distribute query loads across multiple database servers to prevent performance bottlenecks. Allocate adequate CPU and memory resources to database instances handling complex queries.

This study informs best practices for relational schema design, particularly for transaction-heavy systems, which are common in both enterprise and government systems.

5. CONCLUSION

The findings from this study underscore the importance of a well-optimized ERD in ensuring efficient query execution. Among the three ERD designs analysed, ERD-3 consistently demonstrates superior performance across all query types, particularly in high-complexity scenarios. The cost analysis highlights how modular design, strategic indexing, and reduced full table access contribute to significant cost reductions, making ERD-3 the preferred model for large-scale database applications. The implementation of modular ERD design and strategic indexing demonstrates tangible benefits in query performance, representing a practical advancement in relational database optimization in Informatics.

By incorporating best practices such as indexing optimization, query restructuring, and table design improvements, organizations can further enhance query performance and database efficiency. Future research could explore real-world implementations of ERD-3 across different database management systems and evaluate its adaptability to varying workload patterns. Additionally, integrating machine learning techniques for dynamic query optimization may offer new opportunities for performance enhancements in complex data environments.

Overall, this study provides valuable insights into query optimization strategies and reinforces the need for efficient database design in handling large-scale data processing challenges. By leveraging these findings, organizations can build scalable, high-performance database systems that effectively support growing data demands.

REFERENCES

[1] Y. Jani, "Optimizing database performance for large-scale enterprise applications," Int. J. Sci. Res. (IJSR), vol. 11, no. 10, pp. 1394–1396, 2022. https://doi.org/10.21275/SR24716121211.

[2] H. Singh, "Adaptive search optimization: Dynamic algorithm selection and caching for enhanced database performance," Int. J. Comput. Appl., vol. 182, no. 31, pp. 25–32, 2023. https://doi.org/10.13140/RG.2.2.34751.69281.

[3] J. Lao et al., "GPTuner: A manual-reading database tuning system via GPT-guided Bayesian optimization," Proc. VLDB Endow., vol. 17, no. 8, pp. 1939–1952, Apr. 2024. https://doi.org/10.14778/3659437.3659449.

[4] D. Lindner, D. Ritter, and F. Naumann, "Enabling data dependency-based query optimization," Proc. VLDB Endow., vol. 17, no. 6, pp. 1025–1037, 2024. https://doi.org/10.48550/arXiv.2406.06886.

[5] J. Ba and M. Rigger, "CERT: Finding performance issues in database systems through the lens of cardinality estimation," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2023, pp. 1600–1612. https://doi.org/10.1145/3597503.3639076.

[6] T. Gautier, S. Nagar, N. Agrawal, dan A. Kumar, "Hybrid Transactional/Analytical Graph Processing in Modern Memory Hierarchies," *in Proc. Scalable Data Manag. Systems for Future Hardware* (Lecture Notes in Computer Science), no. 13531, Singapore, Jan. 2025, pp. 91–116. https://doi.org/10.1007/978-3-031-74097-8_4.

[7] V. B. Ramu, "Optimizing database performance: Strategies for efficient query execution and resource utilization," Int. J. Comput. Trends Technol., vol. 71, no. 7, pp. 15–21, 2023. https://doi.org/10.14445/22312803/IJCTT-V71I7P103.

[8] H. Song, W. Zhou, H. Cui, X. Peng, and F. Li, "A survey on hybrid transactional and analytical processing," Data Sci. J., vol. 23, no. 1, pp. 45–67, Jun. 2024, https://doi.org/10.1007/s00778-024-00858-9.

[9] J. Li, Q. Wang, S. Han, and P. P. C. Lee, "The Design and Implementation of a High-Performance Log-Structured RAID System for ZNS SSDs," *arXiv preprint arXiv:2402.17963*, Feb. 2024. https://doi.org/10.48550/arXiv.2402.17963.

[10] G. Gottlob, M. Lanzinger, D. M. Longo, C. Okulmus, R. Pichler, and A. Selzer, "Structure-Guided Query Evaluation: Towards Bridging the Gap from Theory to Practice," *arXiv preprint arXiv:2303.02723*, Mar. 2023. https://doi.org/10.48550/arXiv.2303.02723.

[11] Z. Yang, "Machine learning for query optimization," Ph.D. dissertation, Univ. California, Berkeley, CA, USA, 2020. ISBN: 979-8-3529-5188-0.

[12] A. R. Raipurkar and M. B. Chandak, "Optimized execution method for queries with materialized

views: Design and implementation," J. Intell. Fuzzy Syst., vol. 41, no. 6, pp. 6191–6205, 2021. https://doi.org/10.3233/JIFS-202821.

[13] R. Marcus and T. Kraska, "OneShotSTL: One-Shot Seasonal-Trend Decomposition For Online Time Series Anomaly Detection And Forecasting," Proc. VLDB Endow., vol. 16, no. 6, pp. 1432–1444, Feb. 2023, https://doi.org/10.14778/3583140.3583155.

[14] M. Gonthier, D. D. Sanchez-Gallegos, H. Pan, B. Nicolae, S. Zhou, H. D. Nguyen, V. Hayot-Sasson, J. G. Pauloski, J. Carretero, K. Chard, dan I. Foster, "D-Rex: Heterogeneity-Aware Reliability Framework and Adaptive Algorithms for Distributed Storage," *arXiv preprint arXiv:2506.02026*, May 2025. https://doi.org/10.48550/arXiv.2506.02026.

[15] B. Ding, R. Zhu, and J. Zhou, "Learned Query Optimizers", *Foundations and Trends® in Databases*, vol. 9, no. 3, pp. 173–247, 2024. https://doi.org/10.1561/1900000082.

[16] A. Margara, G. Cugola, N. Felicioni, and S. Cilloni, "A model and survey of distributed data-intensive systems," *ACM Comput. Surv.*, vol. 56, no. 1, Article 1, pp. 1–71, Jun. 2023. https://doi.org/10.1145/3604801.

[17] R. Heinrich, M. Luthra, J. Wehrstein, H. Kornmayer, and C. Binnig, "How good are learned cost models, really? Insights from query optimization tasks," Proc. on arXiv, Feb. 2025, https://doi.org/10.48550/arXiv.2502.01229.

[18] A. Mhedhbi, A. Deshpande, and S. Salihoğlu, "Modern Techniques for Querying Graph-Structured Databases," *Foundations and Trends® in Databases*, vol. 14, no. 2, pp. 72–185, Oct. 2024. https://doi.org/10.1561/1900000090.

[19] A. Kipf et al., "Learned cardinalities: Estimating correlated joins with deep learning," in Proc. Conf. Innov. Data Syst. Res. (CIDR), Jan. 2019. https://doi.org/10.48550/arXiv.1809.00677.

[20] T. Kraska et al., "The case for learned index structures," in Proc. ACM SIGMOD Int. Conf. Manage. Data, Jun. 2018, pp. 489–504. https://doi.org/10.1145/3183713.3196909.

[21] N. Vasilenko, A. Demin, and D. Ponomaryov, "Adaptive Cost Model for Query Optimization," *arXiv preprint arXiv:2409.17136*, Sep. 2024. https://doi.org/10.48550/arXiv.2409.17136.

[22] R. Marcus et al., "Neo: A learned query optimizer," Proc. VLDB Endow., vol. 12, no. 11, pp. 1705–1718, Jul. 2019. https://doi.org/10.14778/3342263.3342644.

[23] Y. Ji, D. Amagata, Y. Sasaki, and T. Hara, "PolyCard: A learned cardinality estimator for intersection queries on spatial polygons," *Journal of Intelligent Information Systems*, vol. 63, pp. 873–891, Jan. 2025. https://doi.org/10.1007/s10844-025-00921-z.

[24] X. Wang, M. L. Sapino, W.-S. Han, A. El Abbadi, G. Dobbie, Z. Feng, Y. Shao, and H. Yin, "*Database Systems for Advanced Applications: 28th International Conference,*" *DASFAA 2023, Tianjin, China, April 17–20, 2023, Proceedings, Part I.* Singapore: Springer, 2023. https://doi.org/10.1007/978-3-031-30637-2.

[25] A. Kamali, V. Kantere, C. Zuzarte, and V. Corvinelli, "Robust plan evaluation based on approximate probabilistic machine learning," *arXiv preprint arXiv:2401.15210*, Jan. 2024, https://doi.org/10.48550/arXiv.2401.15210.

[26] Z. M. Khalid and S. R. M. Zeebaree, "Big data analysis for data visualization: A review," Int. J. Sci. Bus., vol. 5, no. 2, pp. 64–75, 2021, https://doi.org/10.5281/zenodo.4481357.

[27] R. Zhu, W. Chen, B. Ding, X. Chen, A. Pfadler, Z. Wu, and J. Zhou, "Lero: A learning-to-rank query optimizer," Proc. on arXiv, Feb. 2023, https://doi.org/10.48550/arXiv.2302.06873.