Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

15th February 2026. Vol. 5. No. 2; e-ISSN: 2808-4519

Optimizing the Execution Time of JOIN Queries and Subqueries Using MySQL

Muhammad Hamdi Yahya[1*], Satriaji Ammarulloh[2], Gathan Hilabi[3], Muhammad Zaki Musyaffa[4], Dicky Anggriawan Nughroho[5], Imam Prayogo Pujiono[6]

[1, 2, 3, 4, 5, 6]*Program Studi Informatika, Fakultas Ekonomi dan Bisnis Islam, Universitas Islam Negeri K.H. Abdurrahman Wahid Pekalongan*

muhammad.hamdi.yahya24035@mhs.uingusdur.ac.id[1]*, satriaji.ammarulloh24017@mhs.uingusdur.ac.id[2], gathan.hilabi24059@mhs.uingusdur.ac.id[3], muhammad.zaki.musyaffa24021@mhs.uingusdur.ac.id[4], dicky.anggriawannugroho@uingusdur.ac.id[5], imam.prayogopujiono@uingusdur.ac.id[6]

Abstract

Relational database systems form the backbone of modern information management. However, the escalating volumes of data and increasing complexity of queries present substantial performance challenges in data retrieval operations. This study investigates the execution time differences between Subqueries and five join methods: Inner Join, Left Join, Right Join, AsOf Join and Lateral Join, in MySQL environments. An experimental methodology was employed, utilising two simulated relational tables containing 100, 1,000, and 10,000 rows of data. Each query method was executed three times under identical system conditions to establish reliable average execution times. The findings demonstrate that join operations substantially outperform subqueries across all tested datasets. Inner Join, Left Join and Right Join maintained execution times below 0.04 seconds, even with the most extensive dataset. Conversely, subqueries exhibited significant performance degradation, with execution times increasing to tens of seconds as the data volume increased. This performance disparity stems from the iterative processing inherent to subqueries, which intensifies proportionally with dataset scale, whereas join operations leverage more efficient simultaneous data processing and merging algorithms. The research concludes that join methods constitute the more appropriate choice for medium to large-scale data scenarios, offering practical optimisation guidance for database developers and administrators implementing MySQL-based systems.

Keywords: *Execution Time, Join, MySQL, Query Optimisation, Subquery*

1. Introduction

The information system ecosystem in the current era of technological development positions relational databases as a crucial foundation for data storage and management. As data continues to grow rapidly, the primary challenge faced by software developers and database administrators no longer lies solely in data storage, but rather in optimising the efficiency of data retrieval processes[1]. MySQL, widely recognised as one of the most commonly used open-source relational database management systems, is often identified as experiencing significant performance degradation when handling complex queries. This issue serves as the primary context of this study, in which execution time efficiency is positioned as a key parameter in evaluating the response time of database systems[2].

This study was conducted because, in terms of query syntax, retrieving data from a database using subqueries is often perceived as straightforward and intuitive, aligning with human logic. This technique can be analogised to instructing a librarian to retrieve books by repeatedly walking back and forth to the shelves for each title requested. Although the process is easy to understand conceptually, in practice, it is highly exhausting and burdensome for the computer system, as it forces the machine to perform tasks that could actually be completed simultaneously repeatedly[3].

Field observations indicate that many applications experience performance degradation due to the excessive use of subqueries, particularly in systems that handle a high volume of transactions. Studies have shown that unoptimized queries can lead to slow response times, increased CPU and memory usage, as well as excessive disk I/O operations, all of which negatively impact overall system performance[4]. Experimental tests have also demonstrated that specific subquery methods can have a substantial impact on query performance, especially when combined with high transactional workloads[5].

Recent studies reveal that optimised queries can improve execution speed by up to 70% compared to non-optimised queries[6]. This phenomenon has motivated further research, as significant differences in execution time, reaching up to tens of times faster, were observed between queries using subqueries and those optimised with Join on production datasets containing millions of rows. A case study conducted on a MySQL database system with a dataset of 64.75 GB and 194.70 million rows demonstrated that optimised methods were able to reduce execution time by up to 70%[7].

In the current era of technological advancement, modern database systems, such as MySQL, are designed to operate more intelligently by utilising Join methods[8]. This Join method is similar to the analogy previously described. However, in practice, many database system

developers are not fully aware that choosing an inappropriate query approach often becomes the primary reason for severe performance degradation, especially when the stored data exceeds millions of rows. This issue becomes even more relevant since the latest versions of MySQL already provide features such as Lateral Join and Hash Join, which can significantly speed up the process. Yet, these features remain underutilised due to a lack of understanding regarding how they work[9].

The main problem identified in this study is the high response latency that occurs when the system processes reports involving relationships between tables. Based on the principles of relational database theory, join operations (Inner Join, Left Join, and Right Join) utilise a set-based approach that manipulates collections of data simultaneously. This approach is theoretically proven to be more efficient than the iterative mechanism of row-by-row processing commonly found in the execution of correlated subqueries, where the query is executed repeatedly for each row of data[10]. In addition, MySQL version 8.0 and above has introduced new functionalities such as Lateral Join, which allows referencing columns from a preceding table within the clause. A lack of awareness regarding the appropriate context for utilising each specific type of Join, including the simulation of AsOf Join related to time-series data, often results in systems operating below their optimal performance threshold[11].

The research problems can be formulated into several key questions: How significant is the difference in execution time between the Subquery and Join methods? Under what data conditions does each technique demonstrate optimal performance? How do dataset characteristics influence the effectiveness of each approach? Do the new MySQL features, such as Lateral Join, provide substantial performance benefits? Considering the research problems mentioned above, this study aims to conduct a comparative analysis to optimise query execution time. The main objective is to evaluate and compare the execution speed between Subquery implementation and various Join methods (Inner, Left, Right, Lateral, and AsOf Join). This research seeks to explore and produce best practice guidelines for database practitioners in selecting the most efficient query writing techniques tailored to specific dataset characteristics, making the findings highly relevant to the topic as a whole[12].

The expected outcome of this study is the development of a comprehensive, evidence-based guideline that will assist database administrators and application developers in making informed decisions regarding effective query writing techniques. This research is expected to enhance the overall efficiency of database systems, accelerate application response time, and ultimately improve end-user satisfaction, as database optimisation has been proven to have a direct impact on service performance. In addition, this study is expected to serve as an academic reference that contributes to the body of knowledge on database query optimisation, particularly in the context of MySQL, which is one of the most widely used database management systems worldwide[13].

2. Literature Review

Query optimisation is a crucial step in database management, ensuring that SQL commands are executed as efficiently as possible. Its objective is to determine the most optimal execution plan by minimising the use of system resources such as processing time, memory, and disk I/O operations. This process involves evaluating various methods for executing a query and selecting the strategy with the lowest cost based on its cost analysis model. The quality of database design and query efficiency are key factors that determine the performance of an information system, especially when the volume of data increases rapidly[14].

The structure of queries written by developers has a direct impact on performance. One of the common dilemmas is deciding between using subqueries and join operations. Subqueries are often considered more intuitive and easier to write because they reflect a step-by-step procedural logic. However, this approach frequently becomes a significant source of inefficiency in large-scale databases.

Systematic literature studies have demonstrated significant inefficiencies associated with the use of nested subqueries in large-scale databases, which often lead to dramatic increases in execution time that exceed the tolerance limits of applications. As a remedial approach, the literature recommends reconfiguring SQL syntax by substituting subqueries with Join structures. This method has been empirically evaluated across various dataset scenarios containing millions of rows[15]. Experimental findings indicate a drastic improvement in execution time efficiency, with some cases achieving more than a 90% reduction for correlated subquery scenarios. This demonstrates that modifying the logical structure of a query often yields a more substantial performance impact than merely adjusting server configuration parameters[16].

In general, Join operations often demonstrate superior performance compared to subqueries, especially in database management systems that have not yet implemented advanced subquery rewriting techniques. However, the execution efficiency of Joins greatly depends on the indexing strategy applied. Without adequate indexing on the join predicate columns, the query optimiser tends to utilise a full table scan to process the data. This operation is known to incur high computational cost and cause significant latency, particularly in tables with data volumes reaching millions of rows[17]. Another literature study highlights the challenges posed by SQL Join performance degradation caused by the absence of indexing on foreign key columns, which results in full table scans on large tables. The methodological framework employed consists of a comparative experimental design that evaluates Join operations with and without composite indexing, using EXPLAIN ANALYZE to measure query costs. The investigation revealed an average increase of 43.68 per cent in execution speed when proper indexing was applied, thereby confirming the essential role of indexing strategies as a fundamental support for Join operations across tables[18]. Within the category of subqueries itself, there are syntactic variants such as IN and EXISTS. Developers often misunderstand when to use one over the other, which leads to the adoption of suboptimal methods, especially when dealing with high-volume data. Addressing the performance differences between subquery variants (IN versus EXISTS), which are often misunderstood by developers and lead to the adoption of suboptimal methodologies when handling high-volume data, requires a comparative analysis of both clauses with indexing support. This can be achieved by utilising MySQL 8.0 benchmarks to evaluate execution time and memory consumption. The results indicate that EXISTS provides greater stability, with a 25–30 per cent speed improvement compared to IN on large datasets, although its performance remains significantly lower than Join operations unless adequately supported by optimal indexing[19]. Modern MySQL (version 8.0 and above) offers significant improvements in Join operation algorithms, including the implementation of Hash Join, which replaces the Block Nested Loop for non-indexed operations, providing far more efficient performance on large datasets. However, the validity of performance measurements is often biased by the tools used[20]. The integration of recent studies indicates that query optimisation is no longer a one-dimensional process, but rather a holistic approach that combines syntactic efficiency, adaptive indexing strategies, and a deep understanding of execution plans. Furthermore, the Lateral Derived Tables feature in MySQL 8.0 has blurred the rigid boundaries between subqueries and Joins, enabling subqueries to correlate with external tables more flexibly while still maintaining execution performance equivalent to Joins in complex cases. This research aims to establish a comprehensive optimisation framework by simultaneously analysing the interactions of these variables[21].

3. Research Methodology

3.1. Research Stages

The research stages in this study were arranged based on a workflow that describes the process of testing the performance of Join and Subquery queries in MySQL. The following is the flowchart:

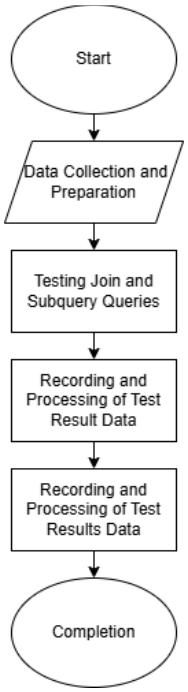

Fig.1. Research Flow Diagram

3.2. Data Collection

The data collection method in this study is experimental, utilising primary data generated through simulation. The primary data is structured in the form of a simple database consisting of two main tables, namely the Student Table (storing unique IDs, names, and majors) and the Grade Table (storing unique IDs, student_id as a foreign key, academic_semester and gpa). The relationship between these two tables enables performance testing of Join and Subquery queries. The amount of data in both tables is varied, ranging from tens to thousands of rows, to simulate different levels of system workload. The main structure of the tables is described in the table as the basis for dataset formation.

No	Table	Column	Data Type
1	**Student**	student_id (PK)	INT
		name	VARCHAR(100)
		major	VARCHAR(50)
2	**Grades**	id (PK)	INT
		student_id	INT
		academic_semester	INT
		gpa	DECIMAL(3,2)

Fig. 2. Testing Database Structure

3.3. Recording and Processing of Test Data

Recording and processing of data were carried out to obtain accurate execution time values for each testing scenario. Each join and Subquery query was executed three times for each dataset size, namely 100, 1.000, and 10.000 rows. Data recording was performed manually using Excel, referencing the execution time values displayed in the performance_schema.events_statements_summary_by_digest table in MySQL. This method allows the researchers to capture execution values in detail and consistently transfer them into the data processing sheet.

The execution results were then organised into testing tables to facilitate analysis and comparison. Each table contains the query type, dataset size, the results of three executions, and the calculated average value to minimise the influence of system fluctuations. The average calculation is performed using the formula :

$$Average\ Execution\ Time = \frac{t1 + t2 + t3}{3}$$

The data are presented in tabular form to demonstrate the performance differences between Join and Subquery queries across various dataset sizes. Presenting the results in tables enables researchers to observe patterns of execution time changes in a structured and measurable manner, which can then serve as the basis for analysis in subsequent stages.

3.4. Result Analysis

An analysis of the research results was conducted to understand how the performance of Join and Subquery queries changes across different dataset sizes, based on the execution times obtained from three trials in each scenario. The analysed object was the average execution time previously recorded and processed, providing an accurate representation of the efficiency of both query methods. This analysis was performed after all data had been obtained from a controlled testing environment by executing the queries in MySQL Workbench on the same device under stable system conditions.

The results show that for small-sized datasets, the difference in execution time between Join and Subquery remains relatively small. This is reasonable since the amount of processed data does not yet impose a significant load on the database engine. However, when the dataset size increases to 1.000 rows, the performance differences become more apparent. Joins demonstrate more consistent execution time due to MySQL's optimised table merging mechanism, whereas Subqueries require additional processing before generating results, leading to a tendency for higher execution times.

The performance gap becomes more pronounced when the dataset reaches 10.000 rows. Joins continue to provide lower execution times compared to Subqueries, indicating that direct table merging methods are more efficient for large data volumes. In contrast, Subqueries begin to show significant execution time increases because the data retrieval process within a Subquery requires additional processing stages that become more burdensome as the number of rows grows.

From these results, it can be concluded that dataset scale has a direct influence on the performance of both query techniques, and Joins tend to be more stable and efficient for medium to large datasets. This analysis provides a crucial foundation for determining effective query writing strategies in database environments with diverse workloads.

4. Results and Discussion

This study employed MySQL Workbench as the primary platform to execute the entire series of tests on Join and Subquery queries. The testing environment was run on a laptop with the following specifications:

a) Processor: AMD Ryzen 5 6600H with a base frequency of 3.30 GHz
b) Memory: 16 GB RAM
c) Operating System: Windows 11 64-bit
d) Storage: 512 GB NVMe SSD

All testing activities were conducted in a controlled manner to ensure stable results, utilising a single application, MySQL Workbench, throughout the entire process. This step was intended to minimise interference from other applications that could affect CPU or memory usage. Before each batch of testing began, the laptop was restarted, and the MySQL cache was cleared to ensure the system was in optimal condition and not influenced by any remaining processes from previous executions.

The testing was performed on two data retrieval techniques, namely Join and Subquery, each executed three times on every dataset size to obtain stable and consistent results. The Join variations tested included Inner Join, Left Join, Right Join, AsOf Join, and Lateral Join, to allow a comprehensive comparison of their performance. For the Subquery category, the structured Join queries were rewritten into Subquery form while still producing the same output, ensuring a fair comparison between the two techniques. This approach enabled a more precise evaluation of the operational differences between Joins and Subqueries as well as their impact on execution time under varying data volumes. The distribution of testing scenarios is as follows:

a) The first to third tests used a dataset containing 100 rows
b) The fourth to sixth tests used a dataset containing 1.000 rows
c) The seventh to ninth tests used a dataset containing 10.000 rows

The execution time results from each trial were manually recorded into Microsoft Excel based on the data displayed through performance_schema.events_statements_summary_by_digest in MySQL. The collected data were then averaged for each scenario as a more stable and objective representation of execution performance. These values served as the basis for evaluating the effectiveness of Join and Subquery in processing data across different dataset sizes. They became the primary reference for the performance analysis in the subsequent section.

4.1. Inner Join and Subquery

The experiment in this section was conducted to compare the performance of Inner Join and Subquery using three different data sizes, namely 100, 1.000, and 10.000 rows. Each test was executed three times for each dataset size, resulting in stabilised average execution values. A summary of the measured computation time for both methods is presented in Table 1.

Table 1: Result of Inner Join and Subquery Testing

Test Order	Number of Rows	Inner Join Execution Time (seconds)	Subquery Execution Time (seconds)
1	100	0.0009	0.0019
2	100	0.0009	0.0010
3	100	0.0010	0.0009
4	1.000	0.0017	0.0022
5	1.000	0.0022	0.0014
6	1.000	0.0030	0.0039
7	10.000	0.0343	0.0404
8	10.000	0.0337	0.0325
9	10.000	0.0348	0.0384

The test results show that Inner Join consistently provides shorter execution times across all dataset sizes compared to Subquery. The performance difference for small to medium datasets is not highly significant, yet Inner Join still demonstrates superiority. When the data size increases to 10.000 rows, Inner Join continues to maintain its performance, whereas Subquery experiences a sharper increase in execution time. This condition indicates that direct table merging mechanisms are more efficient than the Subquery approach, which requires the execution of separate selection commands before the merging process takes place.

4.2. Left Join and Subquery

In this section, the performance of the Left Join is analysed and compared to the Subquery. Similar to the previous test, all three dataset sizes were tested three times to obtain representative average execution time values. A summary of the average computation results is presented in Table 2.

Table 2: Result of Left Join and Subquery Testing

Test Order	Number of Rows	Left Join Execution Time (seconds)	Subquery Execution Time (seconds)
1	100	0.0012	0.0099
2	100	0.0009	0.0097
3	100	0.0010	0.0088
4	1.000	0.0019	0.5023
5	1.000	0.0027	0.4704
6	1.000	0.0026	0.5105
7	10.000	0.0335	24.2317
8	10.000	0.0203	24.7429
9	10.000	0.0177	24.7803

Left Join demonstrates stable performance across all dataset sizes, with the increase in computation time remaining proportional to the growth in data volume. Subquery, on the other hand, experiences a very drastic increase in execution time on datasets containing 1.000 and 10.000 rows. This phenomenon indicates that the repetitive data retrieval process performed by Subquery results in a much higher processing load. Therefore, Left Join is considerably more efficient in scenarios involving large table structures.

4.3. Right Join and Subquery

In this test, a performance evaluation was conducted comparing the Right Join and Subquery methods. All three data size variations were tested three times each, and the average values were then calculated to obtain more stable results. The data obtained from the test are presented in Table 3.

Table 3: Result of Right Join and Subquery Testing

Test Order	Number of Rows	Right Join Execution Time (seconds)	Subquery Execution Time (seconds)
1	100	0.0009	0.0016
2	100	0.0008	0.0012
3	100	0.0009	0.0014
4	1.000	0.0016	0.0034
5	1.000	0.0021	0.0034
6	1.000	0.0018	0.0033
7	10.000	0.0183	0.0392
8	10.000	0.0180	0.0388
9	10.000	0.0179	0.0391

Right Join demonstrates competitive performance with low execution times across all dataset sizes, following a similar pattern to Inner Join. The increase in execution time remains controlled as the amount of data grows. Subquery, however, produces higher execution times at all scales, particularly on the largest dataset. This indicates that the table-matching approach used in Right Join is more computationally efficient than Subquery, which performs data retrieval independently.

4.4. AsOf Join and Subquery

AsOf Join testing was conducted to evaluate its performance under the same variations of dataset sizes. Each scenario was executed three times, and the average values were then calculated to ensure that the results were not affected by fluctuations in execution. The computation time data are presented in Table 4.

Table 4: Result of AsOf Join and Subquery Testing

Test Order	Number of Rows	AsOf Join Execution Time (seconds)	Subquery Execution Time (seconds)
1	100	0.0065	0.0143
2	100	0.0068	0.0097
3	100	0.0092	0.0080
4	1.000	0.2971	0.2603
5	1.000	0.3000	0.2501
6	1.000	0.3027	0.4382
7	10.000	28.7683	24.4295
8	10.000	28.7519	24.3718
9	10.000	28.6200	24.2472

AsOf Join shows relatively high execution time even when the dataset size is still small, and it experiences a substantial spike when the dataset reaches 10.000 rows. Subquery demonstrates slightly better performance in several scenarios, but remains within a high execution

time category. These findings indicate that AsOf Join has considerable operational complexity, making it a less suitable option for large-scale data or dense table relationships.

4.5. Lateral Join and Subquery

In this section, testing was conducted on Lateral Join and Subquery using the same testing procedures. All dataset size scenarios were executed three times. The test results are presented in Table 5.

Table 5: Result of Lateral Join and Subquery Testing

Test Order	Number of Rows	Lateral Join Execution Time (seconds)	Subquery Execution Time (seconds)
1	100	0.0148	0.0077
2	100	0.0077	0.0085
3	100	0.0102	0.0079
4	1.000	0.3047	0.2938
5	1.000	0.2977	0.2951
6	1.000	0.2947	0.3871
7	10.000	28.5472	28.7994
8	10.000	28.6004	28.6488
9	10.000	28.8743	28.5859

Lateral Join exhibits relatively high computation time even with small dataset sizes, and it increases sharply as the amount of data grows. The subquery appears slightly more efficient in several trials, yet it still falls within a range of high and unstable execution times overall. This condition indicates that Lateral Join introduces significant processing overhead due to its row-by-row evaluation, making it less recommended for use with large datasets.

4.6. Comparison of Computation Time Across Join Methods

In this section, a performance comparison is conducted for all five join queries (Inner Join, Left Join, Right Join, AsOf Join, and Lateral Join) against the Subquery based on the average execution time. The values presented are the averaged results from three tests on each dataset size, namely 100, 1.000, and 10.000 rows. This approach was chosen to ensure that single execution fluctuations do not influence the measurement results and provide a more stable representation of the actual performance. A summary of the test results is shown in Table 6 and Figure 3.

Table 6: Average Execution Time of Five Join Queries and Subquery (seconds)

Amount of Data	100	1.000	10.000
Inner Join	0.0009	0.0023	0.0343
Subquery Inner Join	0.0013	0.0077	0.0371
Left Join	0.0010	0.0024	0.0282
Subquery Left Join	0.0095	0.4944	24.5850
Right Join	0.0009	0.0018	0.0181
Subquery Right Join	0.0014	0.0034	0.0390
AsOf Join	0.0075	0.2999	28.7134
Subquery AsOf Join	0.0107	0.3162	24.3495
Lateral Join	0.0109	0.2990	28.6740
Subquery Lateral Join	0.0080	0.3253	28.6780

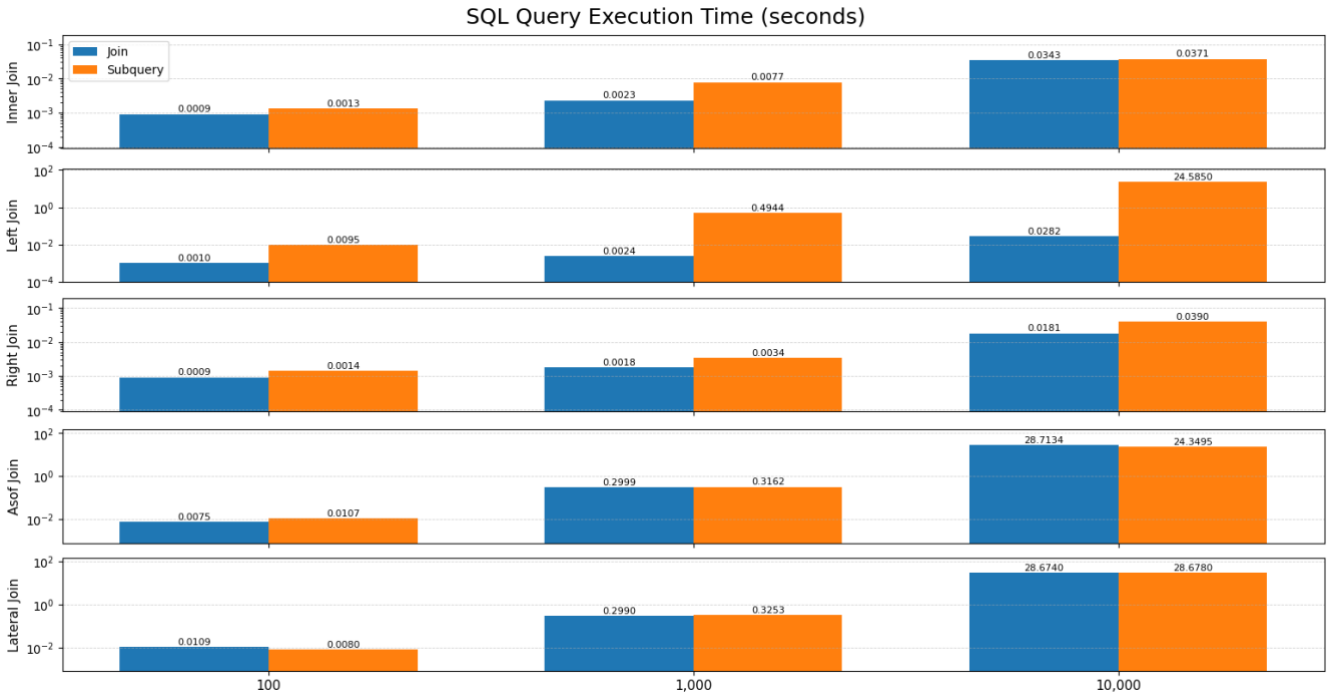

Fig. 3. Average Query Execution Time Chart for Five Types of Joins and Subqueries

Based on Table 6 and Figure 3, which present the testing results of five Join queries (Inner Join, Left Join, Right Join, AsOf Join, and Lateral Join) compared to Subquery, the following conclusions can be drawn:

a. Query execution time (seconds)
b. For the dataset containing 100 rows, all types of Join and Subquery still show very low execution times and do not demonstrate significant differences. Inner Join, Left Join, and Right Join perform very quickly within the range of 0.0009-0.0010 seconds, while their Subquery counterparts also remain efficient with values below 0.01 seconds. AsOf Join and Lateral Join start to show slightly higher execution times compared to the basic Join operations, yet the values are still considered lightweight. Testing with 100 rows indicates that all methods can operate optimally without imposing a considerable computational load on the system.
c. For the dataset containing 1.000 rows, the performance differences between Join and Subquery become more noticeable compared to the previous dataset. Inner Join, Left Join, and Right Join maintain very low execution times, all of which are below 0.003 seconds. The Subquery versions of these three operations exhibit increases in execution time, but the growth remains reasonable and within an acceptable efficiency level. AsOf Join and Lateral Join begin to show much larger spikes in execution time, reaching around 0.29 to 0.32 seconds. The Subquery execution values for both operations are also within a similar range, indicating that computational complexity increases significantly when the dataset reaches 1.000 rows. This dataset size becomes the point at which differences in the ability of each method to handle computational workload become more distinct, particularly for more complex Join operations.
d. For the dataset containing 10.000 rows, the performance differences among methods become highly significant. Inner Join, Left Join, and Right Join continue to show high efficiency with execution times remaining below 0.04 seconds. Subquery for Left Join experiences an extreme increase, exceeding 24 seconds, while Subquery for AsOf Join and Lateral Join reach execution times around 24 to 28 seconds. On the Join side, both AsOf Join and Lateral Join also exhibit substantial increases, surpassing 28 seconds. This condition indicates that operations with more complex searching patterns are susceptible to dataset growth. At large data scales, the efficiency differences among methods become clearly visible.

The overall results between Join and Subquery indicate that Join consistently outperforms Subquery in almost all testing conditions. Join can maintain low execution times even as the data volume increases, particularly in Inner Join, Left Join, and Right Join, which consistently remain below 0.04 seconds even on the 10.000-row dataset. Subquery often experiences significantly higher execution spikes, especially in more complex operations such as Left Join, AsOf Join, and Lateral Join, with execution times increasing to tens of seconds. This occurs because Subquery execution generally requires additional computation steps before producing the final output. Therefore, Join is a far more efficient, stable, and suitable method for large-scale datasets compared to Subquery.

5. Conclusion

This study demonstrates that join methods outperform Subqueries in terms of execution performance across various data scales. Inner Join, Left Join, and Right Join exhibit stable processing times, even when the dataset reaches 10.000 rows, with execution times remaining within the range of milliseconds. This indicates that the set-based approach used by Join is more efficient in processing data compared to the iterative nature of Subqueries.

Subqueries result in a significant increase in execution time, particularly in complex operations such as Left Join, AsOf Join, and Lateral Join. Execution times reaching tens of seconds indicate that repetitive data retrieval in Subqueries imposes a substantial system load. AsOf Join and Lateral Join also exhibit inefficient performance on large
datasets, making them unsuitable for use in production environments with heavy data workloads.

Based on these findings, the application of Join is concluded to be a more effective strategy for data retrieval in relational databases, especially in applications that require fast response and stable performance.

System developers and database administrators are advised to use Join as the primary approach in data processing, particularly when dealing with medium to large datasets. Subqueries should only be used when the logic cannot be represented using a Join or when the query objective is simple and does not significantly affect system performance.

Future research is suggested to evaluate the impact of single and composite indexing on Join performance, as indexing strategies often play a crucial role in accelerating table merging operations. The study can be expanded to larger data scales, including hundreds of thousands to millions of rows, to simulate real-world production system conditions.

Further studies may also compare performance in other database management systems, such as PostgreSQL or MariaDB, to determine whether the observed performance patterns are general or specific to MySQL. Evaluating internal execution methods, such as Hash Join and Block Nested Loop, is also worthwhile to understand the effectiveness of execution algorithms for each query type.

References

[1] C. A. Putra, R. Pratama, and T. Sutabri, "Analisis Manfaat Machine Learning Pada Next-Generation Firewall Sophos Xg 330 Dalam Mengatasi Serangan Sql Injection," *J. Manaj. Inform. Sist. Inf.*, vol. 6, pp. 197–204, 2023.

[2] A. Misbullah, Nazaruddin, Rasudin, and Zulfan, "Analisa Proses Migrasi Mysql Non-Cluster Ke Cluster Dalam Menangani Fail-Over Sistem," *J. Pendidik. Teknol. Inf.*, vol. 4, pp. 40–49, 2020.

[3] E. Budiman, M. Fadli, D. Kurniawan, and E. R. Susanto, "Tinjauan Literatur Dengan Pendekatan Systematic Literature Review Untuk Optimasi Kueri Dalam Basis Data," *J. Ilm. Tek. dan Ilmu Komput.*, vol. 4, no. 2, pp. 82–91, 2025.

[4] Vasudevan Senathi Ramdoss, "Optimizing Database Queries: Cost And Performance Analysis," *Int. J. Sci. Res. Arch.*, vol. 2, no. 2, pp. 293–297, Aug. 2021, doi: 10.30574/ijsra.2021.2.2.0025.

[5] T. Oktavia and S. Sujarwo, "Evaluation of Sub Query Performance in SQL Server," *EPJ Web Conf.*, vol. 68, p. 00033, Mar. 2014, doi: 10.1051/epjconf/20146800033.

[6] B. Triaji, W. Andriyani, T. Suprawoto, M. A. Nugroho, and R. Kartadie, "Query Execution Performance Analysis of Column-Oriented Database in Dashboard," *J. Intell. Softw. Syst.*, vol. 1, no. 2, p. 122, Dec. 2022, doi: 10.26798/jiss.v1i2.768.

[7] T. Alyas, A. Alzahrani, Y. Alsaawy, K. Alissa, Q. Abbas, and N. Tabassum, "Query Optimization Framework for Graph Database in Cloud Dew Environment," *Comput. Mater. Contin.*, vol. 74, no. 1, pp. 2317–2330, 2023, doi: 10.32604/cmc.2023.032454.

[8] A. D. Riawati, M. Irfan, Khaeruddin, and A. Faruq, "High Availability Dynamic Sharding Database Server Dengan Metode Fail Over dan Clustering," *J. Manaj. Inform. Sist. Inf.*, vol. 5, pp. 1–10, 2022.

[9] E. Firmansyah, H. Firdaus, and S. Samidi, "Optimasi Performa Query Subsidi Debitur dengan Index and Table Partition, Subquery and Indexing, dan Parallel Query Execution," *J. Pendidik. dan Teknol. Indones.*, vol. 5, no. 6 SE-, pp. 1769–1785, Jul. 2025, doi: 10.52436/1.jpti.824.
[10] Y. Remil, A. Bendimerad, R. Mathonat, P. Chaleat, and M. Kaytoue, "What Makes My Queries Slow?': Subgroup Discovery For SQL Workload Analysis," Aug. 2021, [Online]. Available: http://arxiv.org/abs/2108.03906
[11] G. Raphaela *et al.*, "Optimasi Query Sql Server Dengan Teknik Indexing Dan Performance Monitoring," *JATI (Jurnal Mhs. Tek. Inform.*, vol. 9, no. 2, pp. 3094–3099, 2025.
[12] K. E. Permana, M. K. Sophan, A. Muntasa, and A. B. Rahmat, "Perbandingan Kinerja Query Sql Join Tables Dengan Menggunakan Index," *J. SimanteC*, vol. 11, no. 2, pp. 241–248, 2023.
[13] M. Raihan Siddik, M. Arief Hasan, A. Fajar Kesuma, N. Sari, S. Dwi Putri, and Q. Uyun Harahap, "Implementasi Query Tuning Untuk Peningkatan Performa Pada Database Barang Mini Market Nan," *JATI (Jurnal Mhs. Tek. Inform.*, vol. 9, no. 2, pp. 3183–3187, 2025, doi: 10.36040/jati.v9i2.13217.
[14] N. Hettiarachchi and P. Yapa, "Advancements in SQL Query Optimization: A Review of Join Order and Index Selection," in *2025 5th International Conference on Machine Learning and Intelligent Systems Engineering (MLISE)*, 2025, pp. 31–39. doi: 10.1109/MLISE66443.2025.11100192.
[15] A. Sander and R. Wauer, "Integrating Terminologies Into Standard SQL: A New Approach For Research On Routine Data," *J. Biomed. Semantics*, vol. 10, no. 1, p. 7, Dec. 2019, doi: 10.1186/s13326-019-0199-z.
[16] A. Hajdini, L. Fazliu, and D. Gjoshi, "Comparative Study of Join Algorithms in MySQL: Cross, Inner, Outer, and Self Joins," *J. Comput. Data Technol.*, vol. 1, no. 1, pp. 1–9, Jun. 2025, doi: 10.71426/jcdt.v1.i1.pp1-9.
[17] I. Sabek and T. Kraska, "The Case for Learned In-Memory Joins," Mar. 2022, [Online]. Available: http://arxiv.org/abs/2111.08824
[18] A. Meleková and M. Kvet, "Effect of JOIN Type on Query Performance," in *2025 37th Conference of Open Innovations Association (FRUCT)*, 2025, pp. 179–184. doi: 10.23919/FRUCT65909.2025.11007985.
[19] R. Marcus, "Learned Query Superoptimization," Jul. 2023, [Online]. Available: http://arxiv.org/abs/2303.15308
[20] A. H. Fathulloh and H. I. Adauwiyah, "Perbandingan Tingkat Efisiensi Waktu Query SELECT pada Database Interface Navicat dan SQLYog di MySQL DBMS," *Appl. Inf. Syst. Manag.*, vol. 4, no. 2, pp. 101–105, Oct. 2021, doi: 10.15408/aism.v4i2.18369.
[21] S. C. Nurzanah, M. S. Armilah, F. Arianto, S. Supriadi, and H. P. Utomo, "Comparison of Support Vector Machine and Naïve Bayes to Sentiment Analysis of Military Barracks Program," *J. Comput. Networks, Archit. High Perform. Comput.*, vol. 7, no. 3, pp. 854–864, Jul. 2025, doi: 10.47709/cnahpc.v7i3.6515.