 ISSN:2302-8149
e-ISSN:2540-9719

Performance Evaluation of Relational, JSON, and Hybrid Models in MariaDB for Dynamic Form Builders

[1]Muhammad Salmansyah Mousavi Akhyar*, [2]Triyanna Widiyaningtyas
[1,2]Program Studi Teknik Informatika, Fakultas Teknik, Universitas Negeri Malang
*e-mail: muhammad.salmansyah.2305356@students.um.ac.id, triyannaw.ft@um.ac.id

(***received*:** 1 October 2026, ***revised*:** 15 October 2026, ***accepted*:** 16 October 2026)

Abstract

Dynamic form builders require database structures that can accommodate variable attributes while maintaining efficient query performance. However, the performance implications of different storage representations within the same database management system remain insufficiently characterized, particularly for relational, raw JSON, and hybrid relational-JSON models. This study evaluates the performance of these three storage representations for dynamic-form data using MariaDB 10.6.19. A controlled quantitative experiment was conducted using datasets of 10,000, 50,000, and 100,000 records and six read and aggregation workloads covering point retrieval, dynamic-field search, multi-field filtering, counting, averaging, and grouping. Each workload was executed using 10 warm-up iterations followed by 30 measured repetitions. Query latency, storage consumption, execution plans, and statistical differences among representations were analyzed. The results show that performance varied according to workload and dataset scale. Statistically significant differences were observed in 13 of 18 workload-dataset combinations. The representation with the lowest median latency also varied across conditions, indicating that no single storage representation consistently provided the lowest query latency. Execution-plan analysis for DYNAMIC_FIELD_SEARCH at 100,000 records showed indexed access for relational storage and full-table scans for the evaluated raw JSON and hybrid configurations. Raw JSON required the lowest storage consumption, followed by the hybrid representation. These findings indicate that storage representation should be selected according to workload characteristics, dataset scale, and structural requirements rather than based on a single overall performance criterion.

Keywords: *database benchmarking, dynamic form builder, JSON storage, query performance, relational database.*

1 Introduction

Dynamic-form applications require database structures that accommodate variable attributes while supporting efficient data retrieval and aggregation. Relational storage provides structured access to predefined attributes, whereas raw JSON allows variable attributes to be stored without requiring a fixed column for every field. A hybrid relational–JSON representation combines these approaches by storing selected attributes relationally and retaining other attributes in JSON. However, the practical performance trade-offs among these representations depend on the workloads being executed and the physical access mechanisms available in the database system.

Previous research on database performance benchmarking has highlighted limitations in experimental realism and reporting detail, which can affect the reproducibility and interpretation of performance comparisons [1]. These differences make it difficult to isolate the performance implications of the storage representation itself. A controlled comparison of relational-only, raw JSON, and hybrid relational–JSON representations under the same DBMS environment and equivalent workloads is therefore needed to characterize their workload- and dataset-dependent performance trade-offs.

This study evaluates these three storage representations for dynamic-form data using MariaDB 10.6.19. The experiment uses logically equivalent datasets at 10,000, 50,000, and 100,000 records and evaluates six workloads: point read, dynamic-field search, multi-field filtering, COUNT, AVG, and GROUP BY. Raw JSON and hybrid configurations are evaluated without generated columns or JSON-specific indexes to establish a controlled baseline. Query latency, storage consumption, execution plans, and statistical differences are analyzed using repeated benchmark measurements.

 ISSN:2302-8149
e-ISSN:2540-9719

The study addresses three research questions:

(RQ1) How does query performance differ among relational, raw JSON, and hybrid relational-JSON storage representations for dynamic-form workloads in MariaDB?

(RQ2) How do these representations differ across dynamic-field retrieval, multi-field filtering, and aggregation workloads at different dataset scales?

(RQ3) What trade-offs among query performance, storage consumption, execution strategy, and structural flexibility can be identified from the three representations?

The contribution of this study is a controlled intra-DBMS comparison of relational-only, raw JSON, and hybrid relational-JSON storage using the same logical data, workloads, dataset scales, and measurement procedure. Rather than identifying a universally superior representation, the study characterizes how performance differences emerge under specific workload and dataset conditions.

2 Literature Review

Research on relational databases has also explored how RDBMSs can accommodate unstructured and semi-structured data, including the use of JSON to represent multivalued attributes and many-to-many relationships [2]. Studies of multi-model data warehouses have compared fully relational, multi-model, and native-model designs, identifying trade-offs between query performance and flexibility, extensibility, and evolvability [3]. Related research on the logical design of multi-model data warehouses has examined relational, document-oriented, and graph-based representations in terms of querying, storage, and ETL requirements [4].

Research on semi-structured data has examined methods for mapping heterogeneous data structures into relational tables [5]. Related work has also investigated the conversion and unified control of relational SQL data and JSON-based NoSQL data in hybrid cloud environments [6]. Schema evolution research has further proposed approaches for managing structural changes across relational and NoSQL databases [7]. These studies provide relevant background for dynamic-form applications, where attributes may vary across forms and database structures may need to accommodate changes over time.

JSON-based storage offers flexibility in representing data with varying structures, while efficient retrieval depends partly on how JSON data is indexed and queried. Research on JSON-based search systems has demonstrated that combining JSON document storage with indexing mechanisms can improve query response time compared with relational implementations in specific application contexts [8]. More recent research has examined algebraic approaches to JSON query optimization and reported reduced query costs compared with unoptimized execution strategies [9]. These studies highlight the importance of indexing and query-processing strategies when evaluating the performance of JSON-based data representations.

Physical data organization is another factor affecting query performance in semi-structured and document-oriented databases. Research on content-and-structure indexing has demonstrated the importance of considering both attribute paths and values when designing indexes for hierarchical data [10]. Studies on document-store design have also shown that alternative document structures and embedding or referencing choices can affect query performance, storage requirements, and document complexity [11]. These findings suggest that performance comparisons across data representations should consider the physical access mechanisms and storage designs used in each configuration, in addition to their logical data models.

A hybrid representation combines structured relational attributes with variable attributes stored in JSON. Research on multi-model query languages has examined mechanisms for querying heterogeneous data representations within a unified database environment [12]. In dynamic-form systems, this concept can be applied by storing stable or frequently queried attributes in relational columns while retaining variable attributes in JSON. Workload-driven database design further indicates that schema decisions should account for data access patterns and expected workloads because these factors influence query performance and storage requirements [13]. A hybrid representation therefore offers a design option for balancing structured access and schema flexibility, although its performance advantage depends on the workload and implementation.

Database performance comparisons depend on the experimental conditions under which they are conducted. A systematic literature review of DBMS performance studies found that benchmark

 ISSN:2302-8149
e-ISSN:2540-9719

workloads and reporting practices often limit the extent to which results can be reproduced or generalized to real-world use cases [1]. A broader review of SQL and NoSQL database architectures has also documented performance comparisons across different database technologies and evaluation criteria [14]. Similarly, empirical benchmarking of PostgreSQL and MySQL illustrates the use of defined query types and experimental conditions to evaluate database performance [15]. These studies highlight the importance of specifying the DBMS, data representation, dataset scale, workload, and measurement procedure when interpreting benchmark results.

Workload definition is an important consideration in database benchmarking. Research on workload duplication has shown that the similarity between synthetic workloads and real application workloads affects the credibility of performance evaluations [16]. Research on reproducibility in database performance testing has further identified dataset characteristics, schema design, queries, hardware configurations, DBMS parameters, software versions, and implementation code as factors that should be documented when reproducing experiments [17]. Microbenchmark research also demonstrates that repeated performance evaluations can help detect performance changes, although the findings depend on the coverage and design of the benchmark suite [18]. These considerations support the use of clearly defined workloads and controlled measurement procedures when comparing storage configurations under equivalent logical operations.

Benchmark studies comparing relational and document-oriented databases have reported performance differences that depend on the tested operations and experimental settings. A comparative study of MongoDB and document-based MySQL evaluated CRUD operations across different data volumes, illustrating the importance of workload characteristics in performance assessment [19]. More recent research comparing PostgreSQL and MongoDB found that relative read performance varied by query type and data volume, with PostgreSQL outperforming MongoDB on the complex analytical queries examined in that study [20]. Performance differences have also been observed among document-oriented database systems, where the leading system varied depending on the workload and scalability measure [21]. These findings reinforce the need to evaluate storage representations under clearly defined workloads rather than assuming a universal performance advantage.

Database optimization research provides additional context for interpreting query performance. A survey of database performance optimization identifies indexes, configuration parameters, SQL design, storage management, and other tuning mechanisms as factors that can affect database performance [22]. Research on DBMS suboptimality has also shown that query optimizers may select execution plans that perform less efficiently than alternative plans, with suboptimality associated with schema complexity, data characteristics, query structure, and optimizer behavior [23]. These findings support examining execution plans alongside measured latency to better understand the access strategies used by different storage representations. However, execution-plan characteristics should be interpreted as supporting evidence rather than as a standalone explanation of observed performance differences.

Although previous studies have investigated data representation, JSON processing, indexing, multi-model databases, and database benchmarking, direct comparisons remain difficult when database platforms, workloads, and physical configurations differ. Prior research provides a foundation for understanding these factors, but the performance trade-offs among relational-only, raw JSON, and hybrid relational-JSON representations under equivalent conditions within the same DBMS warrant further empirical examination. This gap motivates a controlled comparison that evaluates these representations across multiple workloads and dataset scales.

3 Research Method

This study employed a controlled quantitative experimental approach to compare three storage representations for dynamic-form data: relational, raw JSON, and hybrid relational–JSON. The experiment was conducted using MariaDB 10.6.19 with logically equivalent datasets, identical dataset scales, equivalent query workloads, and consistent measurement procedures. Controlled workload definition and experimental configuration are important considerations in database performance evaluation because differences in workload and experimental conditions can influence observed results [1], [17].

 ISSN:2302-8149
e-ISSN:2540-9719

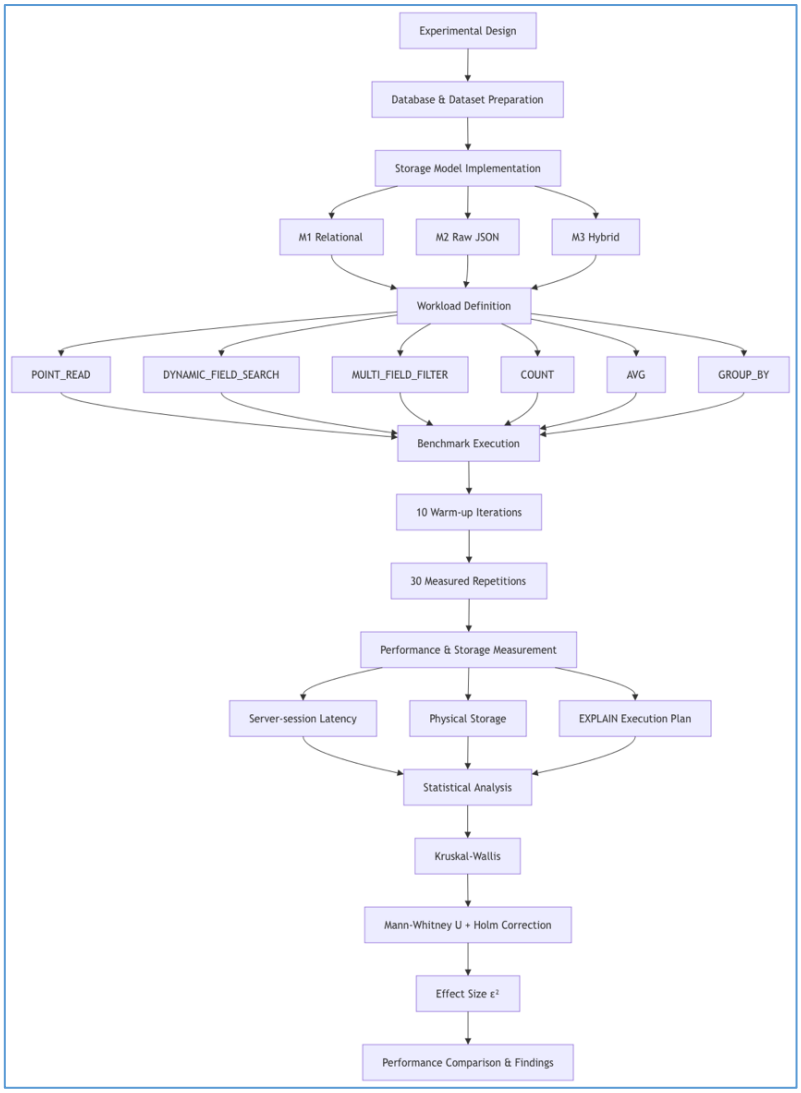


Figure 1 Research method framework

Based on Figure 1, the research consisted of sequential stages covering database and dataset preparation, storage-model implementation, workload definition, benchmark execution, performance measurement, and statistical analysis. The three storage representations were evaluated using equivalent logical data and workloads under the same database environment.

3.1 Research Design

The research used a controlled experimental design in which three database storage representations were evaluated under equivalent experimental conditions. The evaluated representations consisted of relational, raw JSON, and hybrid relational–JSON storage.

The experiment used three dataset scales of **10,000, 50,000, and 100,000 records** and six query workloads covering point retrieval, dynamic-field search, multi-field filtering, and aggregation. The same logical data, workload definitions, dataset scales, and measurement procedures were applied to all three storage representations.

The experimental design was intended to provide a controlled intra-DBMS comparison in which differences in database engine, server environment, dataset content, and workload definition were minimized. Database benchmarking research emphasizes that comparisons should explicitly define workload characteristics and experimental conditions to improve the interpretability and reproducibility of performance results [1], [16].

The benchmark was conducted using a dedicated database separated from the production database. Production tables were not modified during the experiment.

 ISSN:2302-8149
e-ISSN:2540-9719

3.2 Database Storage Models

Three storage models were implemented to represent alternative approaches for storing dynamic-form data.

M1 — Relational. Stable and dynamic attributes were represented as relational columns. Indexes were applied to selected structured attributes used by the benchmark workloads to provide direct relational access.

M2 — Raw JSON. Dynamic attributes were stored in a JSON field. No generated columns or JSON-specific indexes were introduced. This configuration represents a baseline raw JSON implementation rather than a fully optimized JSON configuration. Previous research has shown that JSON retrieval performance can be influenced by indexing mechanisms and query-processing strategies [8], [9].

M3 — Hybrid. Stable core attributes, consisting of id, form_id, user_id, status, and created_at, were stored as relational columns, while variable attributes were stored in a JSON field. Indexing was applied only to the stable relational attributes.

The three models therefore represent the specified storage configurations within the same database environment. The experiment does not attempt to isolate storage representation as an independent causal factor from the physical access mechanisms used by each configuration. This distinction is important because physical organization and access mechanisms can affect query performance independently of the logical data representation [10], [11].

3.3 Benchmark Procedure and Experimental Environment

The benchmark was executed on **MariaDB 10.6.19** using a dedicated benchmark database separate from the production database. Production tables were not modified during the experiment. The benchmark environment used the same database server and connection configuration for all storage representations.

For each combination of storage model, dataset scale, and workload, **10 warm-up iterations** were executed before **30 measured repetitions**. The measured executions were performed in a fixed order: **relational, raw JSON, and hybrid**. The same workload definitions and dataset conditions were maintained throughout the experiment.

Repeated measurements under controlled conditions are commonly used in database benchmarking to account for measurement variability and improve the reliability of observed performance differences [17], [18].

Query latency was measured at the database-session level using microsecond-resolution timestamps immediately before and after query execution. The measurement therefore represents server-session elapsed latency and does not separately isolate CPU processing, disk I/O, or network components.

Storage consumption was obtained from the database table and index size information. Query execution plans were inspected using EXPLAIN to identify access types, indexes, and estimated rows examined.

The fixed execution order was maintained consistently across the benchmark. However, execution order may interact with caching and other system-state effects; therefore, this aspect is considered when interpreting the benchmark results.

3.4 Performance and Storage Measurement

The primary performance variable was **server-session elapsed latency in milliseconds**, summarized using the median of the 30 measured repetitions for each experimental condition. Latency was recorded using microsecond-resolution timestamps immediately before and after query execution.

Median latency was selected to reduce the influence of extreme observations in repeated benchmark measurements. The use of repeated measurements and robust descriptive summaries is consistent with methodological considerations in database microbenchmarking [18].

Because the same database server, connection configuration, workload definitions, dataset scales, warm-up procedure, repetition count, and measurement mechanism were applied across all storage representations, the measured latency was used for comparative evaluation under a common experimental environment rather than as an isolated measure of database-engine execution time.

 ISSN:2302-8149
e-ISSN:2540-9719

Storage consumption was measured as the **sum of reported table data and index sizes** for each benchmark table. Execution plans were analyzed descriptively using EXPLAIN to identify differences in access paths, index usage, and estimated rows examined. Examination of execution plans is relevant because query optimizers and access-path selection can affect observed query performance [22], [23].

The benchmark did not use generated columns, JSON-specific indexes, or additional physical optimizations for the raw JSON and hybrid models. Therefore, the measured results represent the specified baseline configurations rather than the maximum performance achievable through JSON-specific optimization. This distinction is particularly relevant because JSON indexing and query optimization can change the performance characteristics of JSON-based workloads [8], [9].

This experiment compares the specified storage configurations rather than isolating storage representation as a single causal factor independent of physical access mechanisms.

3.5 Statistical Analysis

Statistical analysis was conducted independently for each combination of dataset scale and workload, resulting in 18 workload–dataset conditions. Because the latency observations were not assumed to satisfy parametric distributional assumptions, the **Kruskal–Wallis test** was used as a non-parametric omnibus procedure to compare the three storage representations [24].

For workload–dataset conditions with a statistically significant Kruskal–Wallis result, pairwise Mann–Whitney U tests were subsequently performed for the relational–raw JSON, relational–hybrid, and raw JSON–hybrid comparisons. The resulting 39 pairwise comparisons were adjusted using Holm's step-down procedure to control the family-wise error rate [25].

The magnitude of the omnibus differences was additionally evaluated using **epsilon-squared (ε^2)**. Effect-size measures complement p-values by providing information about the magnitude and practical relevance of findings from non-parametric tests [26].

Statistical significance and effect size were interpreted together with median latency and execution-plan characteristics rather than as independent indicators of overall performance.

4 Results and Analysis

The benchmark results are presented according to the experimental stages described in Section 3. The analysis covers storage consumption, query latency, latency distribution, execution-plan characteristics, and statistical differences among the relational, raw JSON, and hybrid relational-JSON representations. The results are interpreted within the MariaDB 10.6.19 environment and the predefined dataset scales and query workloads.

4.1 Storage Consumption

Figure 2 presents the total storage consumption of the three storage representations at 10,000, 50,000, and 100,000 records.

 ISSN:2302-8149
e-ISSN:2540-9719

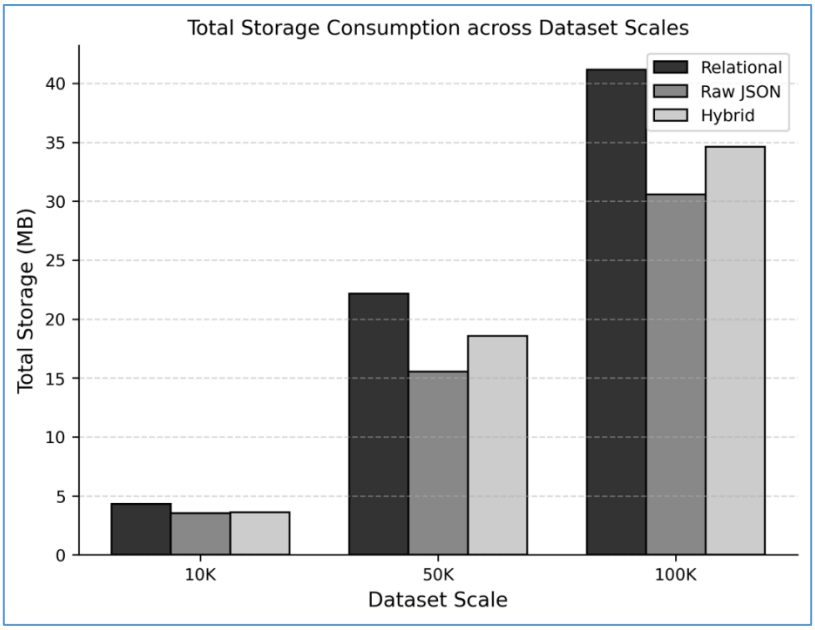


Figure 2 Total storage consumption across dataset scales.

The total storage requirement increased with dataset size for all three representations. At 10,000 records, the relational, raw JSON, and hybrid configurations occupied 4.2969 MB, 3.5156 MB, and 3.5938 MB, respectively. At 50,000 records, the corresponding values were 22.1563 MB, 15.5469 MB, and 18.5781 MB. At 100,000 records, the relational configuration occupied 41.1563 MB, compared with 30.5625 MB for raw JSON and 34.6250 MB for the hybrid configuration.

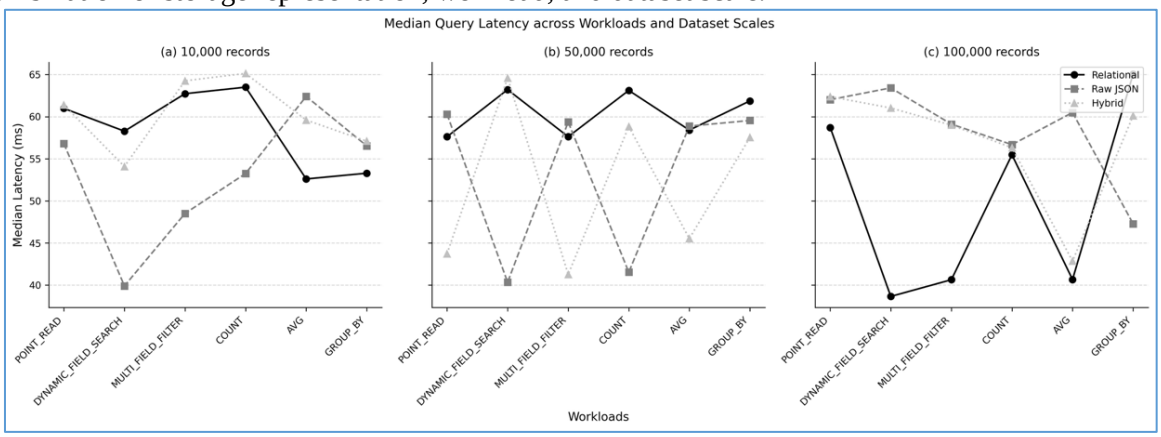
At 100,000 records, raw JSON therefore occupied approximately 25.7% less total storage than the relational configuration, while the hybrid configuration occupied approximately 15.9% less. The hybrid representation consistently occupied more storage than raw JSON but less than the relational representation across the three dataset scales.

The observed differences reflect the physical configurations evaluated in this study. The relational representation stored dynamic attributes as relational columns and used indexes required by the evaluated workloads, whereas raw JSON stored variable attributes in a JSON document without generated columns or JSON-specific indexes. The hybrid representation combined relational storage for stable core attributes with JSON storage for variable attributes.

Therefore, the observed storage differences should not be interpreted as an inherent storage advantage of JSON over relational representation. Rather, they reflect the combined effects of data representation and the physical configuration used in the benchmark. Previous studies have likewise demonstrated that indexing and physical data organization are important factors in database storage and processing characteristics [7], [9], [10].

4.2 Query Latency across Workloads and Dataset Scales

Figure 3 presents the median latency obtained from the 30 measured repetitions for each combination of storage representation, workload, and dataset scale.



 ISSN:2302-8149
e-ISSN:2540-9719

Figure 3 Median query latency across workloads and dataset scales.

At 10,000 records, raw JSON produced the lowest median latency for DYNAMIC_FIELD_SEARCH, MULTI_FIELD_FILTER, and COUNT, with values of 39.900 ms, 48.505 ms, and 53.246 ms, respectively. For DYNAMIC_FIELD_SEARCH, the median latency was 58.274 ms for relational storage, 39.900 ms for raw JSON, and 54.087 ms for the hybrid representation. Thus, raw JSON exhibited approximately 31.5% lower median latency than relational storage for this workload at the smallest dataset scale. However, raw JSON produced the highest median latency for AVG, at 62.405 ms. For POINT_READ, the median latencies were 60.981 ms for relational storage, 56.805 ms for raw JSON, and 61.379 ms for the hybrid representation.

At 50,000 records, the relative performance differed across workloads. The hybrid representation produced the lowest median latency for POINT_READ, MULTI_FIELD_FILTER, and AVG, with values of 43.723 ms, 41.265 ms, and 45.522 ms, respectively. Raw JSON produced the lowest median latency for DYNAMIC_FIELD_SEARCH and COUNT, at 40.336 ms and 41.531 ms. For DYNAMIC_FIELD_SEARCH, the median latency was 63.208 ms for relational storage, 40.336 ms for raw JSON, and 64.594 ms for the hybrid representation. Raw JSON therefore remained the fastest representation for this workload at 50,000 records. The three representations showed relatively close median latency for GROUP_BY, ranging from 57.531 ms to 61.868 ms.

At 100,000 records, relational storage produced the lowest median latency for POINT_READ, DYNAMIC_FIELD_SEARCH, MULTI_FIELD_FILTER, and AVG, with values of 58.712 ms, 38.656 ms, 40.640 ms, and 40.665 ms, respectively. For DYNAMIC_FIELD_SEARCH, raw JSON and the hybrid representation recorded median latencies of 63.415 ms and 61.028 ms, respectively. Unlike the results at 10,000 and 50,000 records, relational storage was therefore the fastest representation for this workload at the largest dataset scale. Raw JSON produced the lowest median latency for GROUP_BY, at 47.279 ms. For COUNT, the median latency values were relatively close, ranging from 55.467 ms to 56.676 ms.

The DYNAMIC_FIELD_SEARCH results illustrate how the relative performance of the storage representations changed with dataset scale. Raw JSON achieved the lowest median latency at 10,000 and 50,000 records, whereas relational storage achieved the lowest median latency at 100,000 records. These results do not establish a universally faster representation; instead, they show that the observed performance depended on both the workload and the dataset scale. The execution-plan analysis discussed in Section 4.3 provides additional evidence for interpreting the result at 100,000 records. In that configuration, relational storage used indexed access, whereas raw JSON and hybrid storage used full-table scans. Because execution plans were examined only at 100,000 records, however, the same access-path explanation cannot be assumed for the smaller dataset scales.

More broadly, median latency did not increase monotonically with dataset size for every workload and storage representation. Several combinations produced lower median latency at 100,000 records than at 50,000 records. Because the measurements represent server-session query latency rather than isolated CPU or I/O execution time, this variation may reflect execution-state and caching conditions within the benchmark environment. These factors should be considered when interpreting differences across dataset scales.

The workload-dependent pattern is consistent with previous database benchmarking research, which emphasizes that performance comparisons are influenced by workload characteristics, datasets, database environments, and experimental procedures [1], [14]. Research concerning workload reproducibility further emphasizes the importance of controlling and documenting benchmark conditions when interpreting performance measurements [16], [17].

4.3 Latency Distribution and Execution Characteristics

Median latency describes the central tendency of the repeated measurements, but it does not fully describe their distribution. Figure 4 therefore focuses on the DYNAMIC_FIELD_SEARCH workload at 100,000 records, where the three representations exhibited a substantial difference in median latency.

 ISSN:2302-8149
e-ISSN:2540-9719

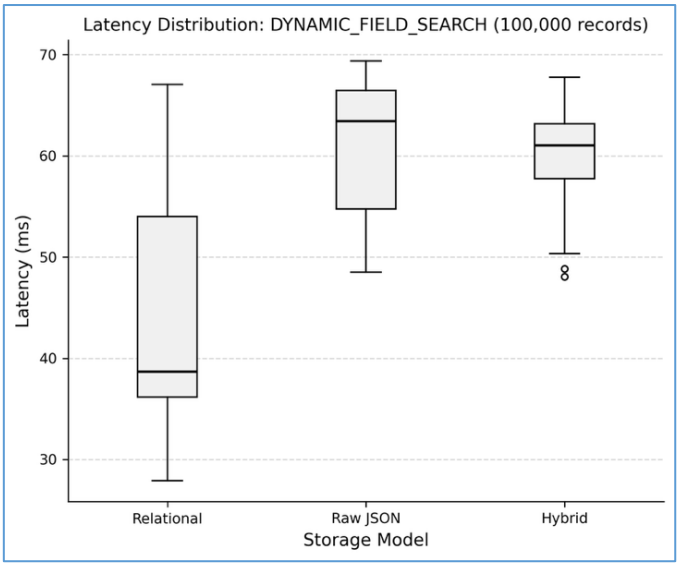


Figure 4 Latency distribution for dynamic-field search at 100,000 records.

The relational representation produced a median latency of 38.656 ms, compared with 63.415 ms for raw JSON and 61.028 ms for hybrid. The distribution shows that the difference was not attributable to a single measurement, although the observed variability differed among the three representations. Table 1 presents the execution plans for the three storage representations, providing additional context for the observed latency differences.

Table 1 Execution plan for dynamic-field search at 100,000 records

Model	Access Type	Key	Estimated Rows	Extra
Relational	ref	idx_kota	≈19,370	Using where; Using index
Raw JSON	ALL	—	≈98,124	—
Hybrid	ALL	—	≈98,644	—

The relational configuration used an indexed access path with type=ref, using idx_kota, with an estimated 19,370 rows and Using where; Using index in the execution plan. In contrast, the raw JSON and hybrid configurations used type=ALL without an applicable key, with estimated rows of approximately 98,124 and 98,644, respectively.

This difference indicates that the relational configuration had a direct indexed access path for the structured attribute used in the workload, whereas the corresponding variable JSON attribute was accessed through a full-table scan under the experimental configuration.

Previous research on JSON indexing has shown that specialized indexing mechanisms can affect the efficiency of JSON search operations [8]. Research on JSON query optimization has likewise examined how JSON representation and query-processing mechanisms influence query execution [9]. Therefore, the result observed in this experiment represents the behavior of **raw JSON without JSON-specific physical optimization**, rather than a general limitation of JSON storage.

The hybrid representation also performed a full scan for the evaluated dynamic-field search because the queried variable attribute remained inside the JSON document. Thus, storing stable core attributes relationally did not provide an indexed access path for a query targeting a variable JSON attribute.

This result illustrates the distinction between structural flexibility and query accessibility. JSON storage allows variable attributes to be represented without mapping every possible field to a separate

 ISSN:2302-8149
e-ISSN:2540-9719

relational column, but querying those attributes without JSON-specific indexing may require more extensive data scanning.

In contrast to the 100,000-record condition, the DYNAMIC_FIELD_SEARCH workload at 10,000 records exhibited an inverse latency pattern in which raw JSON achieved a significantly lower median latency (39.900 ms, IQR = 17.361 ms) than both the indexed relational model (58.274 ms, IQR = 7.414 ms) and the hybrid model (54.087 ms, IQR = 16.814 ms). Although EXPLAIN output was only captured for the 100,000-record condition (as raw EXPLAIN logs for smaller scales were not retained during the benchmark run), the storage decomposition and statistical distributions explain why the indexing advantage does not materialize at 10,000 records. At the 10,000-record scale, the relational table's secondary indexes account for 64.73% of its total storage footprint (2.7813 MB of index data versus 1.5156 MB of row data), whereas raw JSON stores all records in 3.5156 MB with zero index overhead. When the entire table fits comfortably in memory and the query predicate matches a substantial fraction of records (comparable to the ≈ 19.4% selectivity in Table 1), traversing the secondary B-tree index (idx_kota) and performing repeated index-to-clustered-record lookups in the relational model incurs higher cumulative overhead than sequentially scanning the compact, contiguous data pages of the raw JSON table. As the dataset scales to 100,000 records—where raw JSON data expands to 30.5625 MB and relational index structures occupy 28.6406 MB alongside 12.5156 MB of row data—the CPU and memory bandwidth cost of scanning and parsing nearly 100,000 unindexed JSON documents outweighs index lookup overhead, shifting the latency advantage decisively to the indexed relational representation.

4.4 Statistical Analysis Results

The Kruskal-Wallis test was performed independently for each combination of dataset scale and workload, resulting in 18 statistical tests.

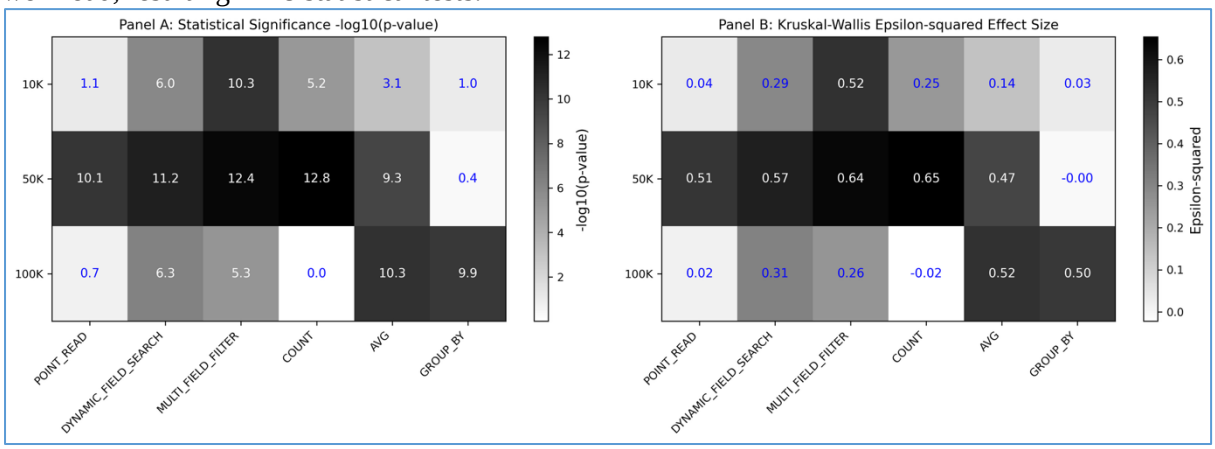


Figure 5 Statistical significance and effect size across dataset scales and workloads.

Figure 5 summarizes the statistical significance and effect-size results across the 18 workload–dataset combinations. Panel A maps the omnibus statistical significance as $-\log_{10}(p\text{-value})$, where darker cells indicate stronger statistical evidence against the null hypothesis of equal latency distributions (with values ranging from 0.0 for COUNT at 100,000 records to 12.8 for COUNT at 50,000 records), while Panel B maps the corresponding Kruskal–Wallis epsilon-squared (ε^2) effect sizes. Visually, both panels highlight that performance divergence among the three storage models is most pronounced at the 50,000-record scale, which exhibits a continuous band of dark cells across five workloads (POINT_READ, DYNAMIC_FIELD_SEARCH, MULTI_FIELD_FILTER, COUNT, and AVG) corresponding to large effect sizes ($\varepsilon^2 \geq 0.26$, ranging from 0.470 to 0.654). Conversely, the lightest cells in both panels identify the five non-significant conditions where latency distributions did not differ significantly ($\varepsilon^2 \leq 0.035$), namely POINT_READ and GROUP_BY at 10,000 records, GROUP_BY at 50,000 records, and POINT_READ and COUNT at 100,000 records.

 ISSN:2302-8149
e-ISSN:2540-9719

At 10,000 records, statistically significant differences were observed for four of the six workloads: DYNAMIC_FIELD_SEARCH, MULTI_FIELD_FILTER, COUNT, and AVG. POINT_READ and GROUP_BY were not statistically significant.

At 50,000 records, five of the six workloads showed statistically significant differences: POINT_READ, DYNAMIC_FIELD_SEARCH, MULTI_FIELD_FILTER, COUNT, and AVG. GROUP_BY was not statistically significant.

At 100,000 records, four workloads showed statistically significant differences: DYNAMIC_FIELD_SEARCH, MULTI_FIELD_FILTER, AVG, and GROUP_BY. POINT_READ and COUNT were not statistically significant.

Overall, 13 of the 18 workload–dataset combinations (72.2%) showed statistically significant differences among the three storage representations at $\alpha = 0.05$. Table 2 summarizes the Kruskal–Wallis test results for each workload–dataset combination, including the H statistic, p-value, epsilon-squared (ε^2) effect size, and statistical significance outcome. These results identify the conditions in which the observed latency distributions differed significantly among the three storage representations.

Table 2 Kruskal-Wallis test results

Dataset	Workload	H	p-value	ε^2	Result
10K	POINT_READ	5.061	0.0796	0.035	Not Significant
10K	DYNAMIC_FIELD_SEARCH	27.535	<0.001	0.294	Significant
10K	MULTI_FIELD_FILTER	47.270	<0.001	0.520	Significant
10K	COUNT	23.849	<0.001	0.251	Significant
10K	AVG	14.321	<0.001	0.142	Significant
10K	GROUP_BY	4.690	0.0959	0.031	Not Significant
50K	POINT_READ	46.637	<0.001	0.513	Significant
50K	DYNAMIC_FIELD_SEARCH	51.693	<0.001	0.571	Significant
50K	MULTI_FIELD_FILTER	57.258	<0.001	0.635	Significant
50K	COUNT	58.919	<0.001	0.654	Significant
50K	AVG	42.861	<0.001	0.470	Significant
50K	GROUP_BY	1.655	0.4370	≈0	Not Significant
100K	POINT_READ	3.440	0.1791	0.017	Not Significant
100K	DYNAMIC_FIELD_SEARCH	28.940	<0.001	0.310	Significant
100K	MULTI_FIELD_FILTER	24.453	<0.001	0.258	Significant
100K	COUNT	0.085	0.9585	≈0	Not Significant
100K	AVG	47.210	<0.001	0.520	Significant
100K	GROUP_BY	45.806	<0.001	0.504	Significant

The largest effect sizes occurred at 50,000 records for COUNT ($\varepsilon^2 = 0.654$), MULTI_FIELD_FILTER ($\varepsilon^2 = 0.635$), and DYNAMIC_FIELD_SEARCH ($\varepsilon^2 = 0.571$). At 100,000 records, relatively large effects were observed for AVG ($\varepsilon^2 = 0.520$) and GROUP_BY ($\varepsilon^2 = 0.504$).

Two non-significant conditions produced slightly negative raw epsilon-squared estimates because the corresponding H statistics were very small. These estimates were treated as approximately zero for interpretation and visualization rather than as substantive negative effects.

These results support the descriptive finding that differences among storage representations varied across workload and dataset scale.

4.5 Post-hoc Pairwise Comparisons

For workload–dataset combinations with statistically significant Kruskal–Wallis results, pairwise Mann–Whitney U tests were conducted, with p-values adjusted using Holm's step-down procedure [25]. Table 3 summarizes the adjusted pairwise comparisons between relational and raw JSON,

 ISSN:2302-8149
e-ISSN:2540-9719

relational and hybrid, and raw JSON and hybrid storage. The results identify which pairs of storage representations differed significantly within each workload–dataset condition after adjustment for multiple comparisons. Comparisons are reported as statistically significant or not significant based on the adjusted p-values and the significance threshold of $\alpha = 0.05$.

Table 3 Post-hoc pairwise comparisons with Holm adjustment

Dataset	Workload	Relational-Raw JSON	Relational-Hybrid	Raw JSON-Hybrid
10K	AVG	Significant	Significant	Not Significant
10K	COUNT	Significant	Not Significant	Significant
10K	DYNAMIC_FIELD_SEARCH	Significant	Not Significant	Significant
10K	MULTI_FIELD_FILTER	Significant	Not Significant	Significant
50K	AVG	Not Significant	Significant	Significant
50K	COUNT	Significant	Not Significant	Significant
50K	DYNAMIC_FIELD_SEARCH	Significant	Not Significant	Significant
50K	MULTI_FIELD_FILTER	Not Significant	Significant	Significant
50K	POINT_READ	Not Significant	Significant	Significant
100K	AVG	Significant	Not Significant	Significant
100K	DYNAMIC_FIELD_SEARCH	Significant	Significant	Not Significant
100K	GROUP_BY	Significant	Significant	Significant
100K	MULTI_FIELD_FILTER	Significant	Significant	Not Significant

Note: Significant indicates a statistically significant Holm-adjusted comparison at $\alpha = 0.05$. Not Significant indicates a non-significant Holm-adjusted comparison at $\alpha = 0.05$.

At 10,000 records, the relational-raw JSON comparison was significant for all four workloads with significant omnibus results. The relational-hybrid comparison was significant only for AVG, whereas raw JSON-hybrid was significant for COUNT, DYNAMIC_FIELD_SEARCH, and MULTI_FIELD_FILTER.

At 50,000 records, the pairwise relationships followed two distinct patterns depending on the leading storage representation. For workloads where the hybrid model achieved the lowest median latency (`AVG`, `MULTI_FIELD_FILTER`, and `POINT_READ`), statistically significant differences were observed between the hybrid representation and both relational and raw JSON storage, whereas the relational–raw JSON comparison was not significant after Holm adjustment across the 39 tests. Conversely, for workloads where raw JSON achieved the lowest median latency (`COUNT` and `DYNAMIC_FIELD_SEARCH`), significant differences were observed between raw JSON and both relational and hybrid storage, while the relational–hybrid comparison was not significant.

At 100,000 records, GROUP_BY was the only evaluated workload across the entire benchmark in which all three pairwise comparisons were statistically significant after Holm adjustment. In contrast, DYNAMIC_FIELD_SEARCH and MULTI_FIELD_FILTER showed significant differences involving the relational representation (Relational–Raw JSON and Relational–Hybrid) but no significant difference between raw JSON and hybrid storage.

These findings demonstrate that a significant Kruskal–Wallis result does not imply that every pair of storage representations differs. Instead, the pairwise results provide more specific evidence regarding which pairs of storage representations differed significantly within each workload–dataset condition.

4.6 Discussion and Research Question Analysis

The benchmark results demonstrate that relational, raw JSON, and hybrid relational-JSON representations exhibit different characteristics in terms of storage consumption, query latency, and

 ISSN:2302-8149
e-ISSN:2540-9719

execution strategy. However, the observed differences were conditional on workload and dataset scale rather than consistent across all experimental conditions.

Regarding **RQ1**, the observed query performance differed across the evaluated storage representations in a workload- and scale-dependent manner. Statistically significant differences were observed in 13 of the 18 workload-dataset combinations, while the representation with the lowest median latency varied across conditions. Therefore, the results do not indicate a universally superior storage representation.

Regarding **RQ2**, the three representations exhibited workload- and scale-dependent behavior for dynamic-field retrieval, multi-field filtering, and aggregation. For DYNAMIC_FIELD_SEARCH at 10,000 records, raw JSON achieved a median latency of 39.900 ms, outperforming both relational storage (58.274 ms) and the hybrid representation (54.087 ms). Pairwise Mann–Whitney U tests with Holm adjustment confirmed that raw JSON was significantly faster than both relational ($U = 777.0, p_{\text{Holm}} < 0.001$) and hybrid storage ($U = 200.0, p_{\text{Holm}} = 0.0057$), whereas relational and hybrid models did not differ significantly ($p_{\text{Holm}} = 0.6801$). This finding demonstrates that secondary indexing in the relational model did not provide a latency advantage at the smallest data scale. Because the relational table's secondary indexes constitute 64.73% of its 4.2969 MB total size at 10,000 records, index traversal and bookmark lookups across matching rows incur greater overhead than a sequential in-memory scan of the unindexed 3.5156 MB raw JSON table. At 100,000 records, however, this relationship reversed: relational storage achieved the lowest median latency at 38.656 ms, significantly outperforming both raw JSON (63.415 ms; $U = 136.0, p_{\text{Holm}} < 0.001$) and hybrid storage (61.028 ms; $U = 160.0, p_{\text{Holm}} < 0.001$). Execution-plan analysis at 100,000 records (Table 1) confirmed that relational storage utilized an indexed access path (type=ref on idx_kota, examining $\approx$ 19,370 rows), whereas raw JSON and hybrid configurations required full-table scans (type=ALL across $\approx$ 98,124 and $\approx$ 98,644 rows, respectively). Because formal EXPLAIN plans were recorded only for DYNAMIC_FIELD_SEARCH at 100,000 records, these access-path observations should not be overgeneralized to all workloads or scales.

Regarding **RQ3**, the three representations involve different trade-offs between storage, query accessibility, and structural flexibility. Raw JSON occupied the least storage under the evaluated configuration, while relational storage occupied the most. Relational storage provided an indexed access path for the structured attribute examined in the dynamic-field search, whereas the corresponding JSON-based configurations required full scans. The hybrid representation occupied an intermediate storage footprint and retained relational access for stable core attributes while preserving JSON-based flexibility for variable attributes.

The observed behavior is consistent with previous research showing that JSON query performance is influenced by indexing and query-processing mechanisms [8], [9]. Studies on content-and-structure indexing and document-store database design likewise demonstrate that physical organization can affect database processing characteristics [10], [11]. Consequently, the latency observed for raw JSON in this experiment should be interpreted as the result of the evaluated baseline configuration rather than as an intrinsic limitation of JSON storage.

The findings also align with database benchmarking literature emphasizing the importance of workload definition and experimental control. Previous reviews have demonstrated substantial variation in DBMS performance studies due to differences in database systems, datasets, workloads, and evaluation procedures [1], [14]. Research on workload duplication and reproducibility further emphasizes the importance of controlling and documenting experimental conditions when interpreting benchmark results [16], [17]. In the present study, the DBMS, logical data, workload definitions, dataset scales, repetition count, and storage representations were controlled within the same experimental environment to reduce these sources of variation.

From an application-design perspective, the results indicate that structural flexibility and query accessibility represent different design considerations. JSON provides a flexible representation for variable attributes, whereas relational columns provide direct access paths for structured attributes when appropriate indexes are available. The hybrid configuration combines these properties by maintaining stable core attributes relationally while retaining variable attributes in JSON. However, the results show that this hybrid structure does not automatically improve queries targeting unindexed variable attributes.

 ISSN:2302-8149
e-ISSN:2540-9719

4.7 Study Limitations

Several methodological limitations of this study should be considered when interpreting the findings. First, the benchmark used a fixed execution order of relational, raw JSON, and hybrid configurations rather than randomized execution. Second, temporal dependence among repeated latency observations was not explicitly modeled; because the benchmark used a fixed execution order, buffer-pool caching and system-state effects may have influenced successive measurements, and the statistical findings should therefore be interpreted within the scope of the benchmark protocol. Third, query latency was measured as server-session elapsed time using MariaDB NOW(6) timestamps and does not separately isolate CPU processing, disk I/O, or network overhead. Fourth, the experiment focused on single-session read and aggregation workloads and did not evaluate concurrent transactions, write-heavy workloads, or lock contention. Fifth, generated columns and JSON-specific indexes were intentionally excluded from the raw JSON and hybrid configurations to establish an unindexed baseline, meaning the results do not reflect the maximum performance achievable through JSON-specific physical tuning. Sixth, EXPLAIN execution plans were only captured for the 100,000-record DYNAMIC_FIELD_SEARCH baseline, limiting direct verification of optimizer access paths at smaller scales. Finally, the experiment was conducted in a single MariaDB 10.6.19 environment, limiting direct generalization to other DBMS engines.

Overall, the findings indicate that the choice among relational, raw JSON, and hybrid storage representations should be aligned with the expected workload, dataset scale, and structural variability of the application. The contribution of this study is therefore the controlled empirical characterization of these three representations within a common MariaDB environment, rather than the identification of a universally optimal storage model.

5 Conclusion

This study evaluated the performance characteristics of relational, raw JSON, and hybrid relational-JSON storage representations for dynamic-form data within a MariaDB 10.6.19 environment. The evaluation considered three dataset scales, six query workloads, query latency, storage consumption, execution plans, and statistical differences among the three representations.

The results demonstrate that the performance characteristics of the three storage representations were workload- and scale-dependent. Statistically significant differences were observed in 13 of 18 workload-dataset combinations, while the remaining conditions did not show statistically significant differences. The median latency results similarly showed that no single representation consistently produced the lowest latency across all workloads and dataset scales. This indicates that storage representation should be considered together with workload characteristics rather than evaluated through a single overall performance measure.

For DYNAMIC_FIELD_SEARCH at 100,000 records, the execution-plan analysis showed indexed access for relational storage and full-table scans for the evaluated raw JSON and hybrid configurations. These findings describe the access paths observed under the tested baseline configuration and should not be generalized to all workloads or dataset scales.

The findings answer the research questions by demonstrating that the three storage representations involve different trade-offs among query performance, storage consumption, execution strategy, and structural flexibility. Relational storage provides direct access mechanisms for structured and indexed attributes, raw JSON provides greater flexibility for variable attributes and a smaller storage footprint under the evaluated configuration, while the hybrid representation combines relational storage for stable core attributes with JSON storage for variable attributes. However, none of these characteristics resulted in a universally superior representation across all evaluated workloads.

From a practical standpoint, these results indicate that system architects developing dynamic-form applications should align storage representation choices with anticipated dataset growth and query patterns—leveraging indexed relational columns or hybrid core attributes when high-volume filtering dominates, while utilizing JSON representations when schema flexibility and compact storage are prioritized at smaller scales. Future studies may extend this baseline evaluation by investigating concurrent read/write workloads, randomized execution orders, multi-scale execution-

 ISSN:2302-8149
e-ISSN:2540-9719

plan profiling, and virtual generated columns with secondary indexes on JSON paths to examine how physical optimization alters these performance trade-offs.

References

[1] T. Taipalus, "*Database Management System Performance Comparisons: A Systematic Literature Review*," *Journal of Systems and Software*, Vol. 208, hlm. 111872, Feb 2024, DOI: 10.1016/j.jss.2023.111872.

[2] K. ElDahshan, E. Selim, A. Ismail Ebada, M. Abouhawwash, Y. Nam, dan G. Behery, "*Handling Big Data in Relational Database Management Systems*," *Computers, Materials & Continua*, Vol. 72, No. 3, hlm. 5149–5164, 2022, DOI: 10.32604/cmc.2022.028326.

[3] S. Bimonte, E. Gallinucci, P. Marcel, dan S. Rizzi, "*Data Variety, Come as You are in Multi-Model Data Warehouses*," *Information Systems*, Vol. 104, hlm. 101734, Feb 2022, DOI: 10.1016/j.is.2021.101734.

[4] S. Bimonte, E. Gallinucci, P. Marcel, dan S. Rizzi, "*Logical Design of Multi-Model Data Warehouses*," *Knowl Inf Syst*, Vol. 65, No. 3, hlm. 1067–1103, Mar 2023, DOI: 10.1007/s10115-022-01788-0.

[5] G. Yuan, J. Lu, Z. Yan, dan S. Wu, "*A Survey on Mapping Semi-Structured Data and Graph Data to Relational Data*," *ACM Comput. Surv.*, Vol. 55, No. 10, hlm. 1–38, Okt 2023, DOI: 10.1145/3567444.

[6] L. Zhang, K. Pang, J. Xu, dan B. Niu, "*JSON-based Control Model for SQL and NoSQL data Conversion in Hybrid Cloud Database*," *J Cloud Comp*, vol. 11, no. 1, hlm. 23, Des 2022, doi: 10.1186/s13677-022-00302-9.

[7] A. H. Chillón, M. Klettke, D. S. Ruiz, dan J. G. Molina, "*A Generic Schema Evolution Approach for NoSQL and Relational Databases*," *IEEE Trans. Knowl. Data Eng.*, Vol. 36, No. 7, hlm. 2774–2789, Jul 2024, DOI: 10.1109/TKDE.2024.3362273.

[8] T. Whairit, B. Phadermrod, dan V. Attasena, "*JINDEX: JSON and Index Search System for Plant Germplasm Database*," *Journal of King Saud University - Computer and Information Sciences*, Vol. 35, No. 8, hlm. 101701, Sep 2023, DOI: 10.1016/j.jksuci.2023.101701.

[9] A. J. Langerak, F. Frasincar, dan J. Klinkhamer, "*JAL: An Algebra for JSON Query Optimization*," *World Wide Web*, Vol. 28, No. 3, hlm. 28, Mei 2025, DOI: 10.1007/s11280-025-01336-0.

[10] K. Wellenzohn, M. H. Böhlen, S. Helmer, A. Pietri, dan S. Zacchiroli, "*Robust and Scalable Content-and-Structure Indexing*," *The VLDB Journal*, Vol. 32, No. 4, hlm. 689–715, Jul 2023, DOI: 10.1007/s00778-022-00764-y.

[11] M. Hewasinghage, S. Nadal, A. Abelló, dan E. Zimányi, "*Automated Database Design for Document Stores with Multicriteria Optimization*," *Knowl Inf Syst*, Vol. 65, No. 7, hlm. 3045–3078, Jul 2023, DOI: 10.1007/s10115-023-01828-3.

[12] Q. Guo, C. Zhang, S. Zhang, dan J. Lu, "*Multi-Model Query Languages: Taming the Variety of Big Data*," *Distrib Parallel Databases*, Vol. 42, No. 1, hlm. 31–71, Mar 2024, DOI: 10.1007/s10619-023-07433-1.

[13] L. Chen, A. Davoudian, dan M. Liu, "*A Workload-Driven Method for Designing Aggregate-Oriented NoSQL Databases*," *Data & Knowledge Engineering*, Vol. 142, hlm. 102089, Nov 2022, DOI: 10.1016/j.datak.2022.102089.

[14] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, dan B. Luo, "*SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review*," *BDCC*, Vol. 7, No. 2, hlm. 97, Mei 2023, DOI: 10.3390/bdcc7020097.

[15] S. V. Salunke dan A. Ouda, "*A Performance Benchmark for the PostgreSQL and MySQL Databases*," *Future Internet*, Vol. 16, No. 10, hlm. 382, Okt 2024, DOI: 10.3390/fi16100382.

[16] S. Weng *dkk.*, "*Lauca: A Workload Duplicator for Benchmarking Transactional Database Performance*," *IEEE Trans. Knowl. Data Eng.*, Vol. 36, No. 7, hlm. 3180–3194, Jul 2024, DOI: 10.1109/TKDE.2024.3360116.

[17] A. Ouared, M. Amrani, A. Chadli, dan P.-Y. Schobbens, "*Deep Variability Modeling to Enhance Reproducibility of Database Performance Testing*," *Cluster Comput*, Vol. 27, No. 8, hlm. 11683–11708, Nov 2024, DOI: 10.1007/s10586-024-04533-0.

[18] M. Grambow, D. Kovalev, C. Laaber, P. Leitner, dan D. Bermbach, "*Using Microbenchmark Suites to Detect Application Performance Changes*," *IEEE Trans. Cloud Comput.*,Vol. 11, No. 3, hlm. 2575–2590, Jul 2023, DOI: 10.1109/TCC.2022.3217947.
[19] C. A. Győrödi, D. V. Dumşe-Burescu, D. R. Zmaranda, dan R. Ş. Győrödi, "*A Comparative Study of MongoDB and Document-based MySQL for Big Data Application Data Management*," *BDCC*, Vol. 6, No. 2, hlm. 49, Mei 2022, DOI: 10.3390/bdcc6020049.
[20] J. Urnikienė, V. Steponavičienė, dan S. Atanasov, "*Comparative Read Performance Analysis of PostgreSQL and MongoDB in E-Commerce: An Empirical Study of Filtering and Analytical Queries," BDCC*, Vol. 10, No. 2, hlm. 66, Feb 2026, DOI: 10.3390/bdcc10020066.
[21] I. Carvalho, F. Sá, dan J. Bernardino, "*Performance Evaluation of NoSQL Document Databases: Couchbase, CouchDB, and MongoDB*," *Algorithms*, Vol. 16, No. 2, hlm. 78, Feb 2023, DOI: 10.3390/a16020078.
[22] S. Huang, Y. Qin, X. Zhang, Y. Tu, Z. Li, dan B. Cui, "*Survey on Performance Optimization for Database Systems*," *SCI. China Inf. SCI.*, Vol. 66, No. 2, hlm. 121102, Feb 2023, DOI: 10.1007/s11432-021-3578-6.
[23] S. Currim, R. T. Snodgrass, dan Y.-K. Suh, "*Identifying the Root Causes of DBMS Suboptimality*," *ACM Trans. Database Syst.*, Vol. 49, No. 1, hlm. 1–40, Mar 2024, DOI: 10.1145/3636425.
[24] D. Chicco, A. Sichenze, dan G. Jurman, "*A Simple Guide to the use of Student's t-Test, Mann-Whitney U Test, Chi-Squared Test, and Kruskal-Wallis Test in Biostatistics*," *BioData Mining*, Vol. 18, No. 1, hlm. 56, Agu 2025, DOI: 10.1186/s13040-025-00465-6.
[25] O. J. M. Guilbaud, "*Note on a Stepwise Procedure for Rejecting at Least k out of m Hypotheses: A Simple Holm-Type Formulation and Proof*," *Biometrical J*, Vol. 64, No. 2, hlm. 377–383, Feb 2022, DOI: 10.1002/bimj.202000206.
[26] F. Fiel Peres, "*Effect Sizes for Nonparametric Tests*," *Biochem. med. (Online)*, Vol. 36, No. 1, hlm. 5–16, Feb 2026, DOI: 10.11613/BM.2026.010101.