Evaluating Field Flexibility Approaches in Relational Databases: A Performance Study of JSON and Column-Oriented Models in Library Systems

Rizal Fathoni Aji, Nilamsari Putri Utami
Faculty of Computer Science, Universitas Indonesia, Indonesia

***Abstract*—This study examines two approaches for achieving field flexibility in library systems using relational databases: column-oriented tables and JSON data types. To evaluate the performance and practicality of flexible schema strategies, a dataset of 41,000 library records was implemented using both column-oriented and JSONB-based schemas in PostgreSQL. Five representative queries based on typical search operations in library applications were executed repeatedly on each model, and average execution times were measured in a controlled environment. Results show that JSONB consistently outperforms the column-oriented approach across all query scenarios, benefiting from reduced structural overhead and more direct access to semi-structured data. However, the flexibility of JSONB introduces risks of inconsistent data structures and reduced schema enforcement compared to the more rigid but uniform column-oriented method. The findings highlight a trade-off between performance and data consistency, suggesting that JSONB is advantageous for dynamic, metadata-rich systems, while column-oriented storage remains preferable when strict structural integrity is required. Future work should explore hybrid models and schema validation layers to combine flexibility with reliable data governance.**

***Keywords*—Field flexibility; RDBMS; column-oriented model; JSON; library systems**

I. Introduction

Relational databases have been the foundation of data management for decades, particularly in systems like library information systems, where maintaining consistency, data integrity, and efficient querying is crucial [1]. Libraries often store vast collections that encompass a wide range of materials, books, journals, and digital media, each with unique metadata attributes such as authorship, publication date, genre, format, or edition. The structured schema of relational databases ensures that these attributes are stored in a highly organized manner. However, this rigid structure can become a constraint when new or evolving data needs arise, such as adding metadata fields for digital content or custom attributes for special collections [2].

Library systems differ considerably in nature. Libraries worldwide are managed by diverse organizations with varying management styles and differing attitudes toward collections and technology. Libraries have evolved from early research platforms into more fully developed applications, typically within selected content domains [3]. Alongside the classical research area of electronic searching, known as information retrieval, library systems have progressed from conventional search methods toward richer information retrieval systems. Modern library systems must facilitate data exchange among geographically distributed libraries and accommodate differing system requirements. Moreover, libraries employ various metadata standards, such as Dublin Core [4] and MARC [5], which require flexibility in field implementation. For example, journal article records may need to include information about editors and publishers, whereas thesis records may not require such details.

The predefined schema in relational databases requires that all data types be known upfront, making it difficult to accommodate unexpected or variable metadata without significant modifications. As a result, adapting a relational system to handle new types of collection metadata can lead to increased development effort, added complexity, and potential downtime [6]. This challenge is amplified as the library grows or incorporates more diverse types of collections, requiring frequent schema updates and risking disruption to the overall system.

NoSQL databases offer considerable flexibility in handling unstructured or semi-structured data, making them ideal for applications where schema changes are frequent, or data structures are unpredictable [6,7]. However, this flexibility comes at a cost, particularly when it comes to complex querying and table relationships. One significant challenge is the difficulty in performing efficient joins between tables, a core functionality that relational databases excel at. This limitation can pose problems for systems like library information management, where table relations are critical for day-to-day operations [7].

In a library system, various activities, such as book check-ins, check-outs, and reporting, rely heavily on inter-table relationships to maintain data consistency and integrity. For example, checking a book in or out requires joining tables that hold data on books, borrowers, and loan history. Moreover, libraries often need to generate comprehensive reports, which require data aggregation from multiple tables, like generating overdue reports, catalog updates, or usage statistics. The absence of native join capabilities in NoSQL databases can result in performance bottlenecks or force developers to implement workarounds, such as data duplication or complex client-side logic, which can introduce additional complexity and potential inefficiencies [7].

Thus, while NoSQL provides unmatched schema flexibility, its shortcomings in handling relational data make it less suited

for systems like library management, where efficient table relationships and complex joins are crucial for smooth operations and reporting [7,8].

This study delves into several innovative approaches designed to enhance the flexibility of relational database management systems (RDBMS). These methods enable developers to introduce and manage changes dynamically, without the need for extensive schema redesigns or additional overhead. By doing so, organizations can ensure their data systems remain adaptable, scalable, and better suited to meet the ever evolving demands of modern information management.

Previous studies have investigated the performance implications of using the JSONB data type in PostgreSQL for managing semi-structured data [9, 10]. Although JSONB offers schema flexibility, compact binary storage, and advanced indexing mechanisms such as GIN indexes, empirical evaluations reveal a clear trade-off between flexibility and performance. Comparative benchmarks consistently show that JSONB substantially outperforms the plain JSON type and regular column indexing, with reported query time reductions of up to fourfold under certain workloads [9, 10]. However, most existing work concentrates on generic database workloads or compares JSONB with document-oriented databases, with limited attention to domain-specific systems such as digital libraries.

In library information systems, schema flexibility is a critical requirement because bibliographic data exhibit substantial structural diversity. Libraries manage a wide range of resource types, including books, journal articles, theses, and heritage collections, each requiring different metadata fields and descriptive elements. Some collections, such as heritage records, demand highly customized and evolving metadata to capture provenance, physical condition, restoration history, and cultural context, which are often not supported by standard schemas. These needs are typically addressed through multiple metadata standards (such as MARC and Dublin Core) and their integration, which increases structural complexity within the database [11–13]. Rigid relational schemas often lead to frequent schema modifications and increasingly complex table designs, whereas JSONB offers a promising alternative by enabling flexible, self-describing metadata storage within a relational database environment. Despite its potential, empirical evidence on JSONB performance under realistic library workloads remains limited, particularly in comparison with traditional column-oriented schemas. This study, therefore, investigates the performance of JSONB in a digital library setting, directly comparing it with column-based designs to evaluate the costs and benefits of schema flexibility for bibliographic metadata management.

II. Field Flexibility in Relational Database

Field flexibility refers to the ability to modify the structure of data within an application without causing significant disruptions to the underlying systems. In traditional databases, data structures are often rigid, requiring a defined schema upfront, which can make changes laborious and time-consuming. However, field flexibility enables the system to evolve as new requirements emerge, allowing for the addition of new fields, modification of existing ones, or even changes in data types, all while minimizing impact on the rest of the application. Achieving this flexibility can be crucial for applications that face frequent changes in data requirements or need to support dynamic use cases. Below are several strategies commonly employed to implement field flexibility:

A. Incremental Schema Changes

One approach for achieving field flexibility is through incremental schema changes. This strategy involves gradually evolving the database schema by adding or modifying fields as needed, without requiring a full database redesign. For instance, in a library information system, if new data fields, such as a book's digital format (e.g., PDF, ePub), are needed to support eBook checkouts, developers can add these fields incrementally.

Tools like database migration frameworks (e.g., Alembic for SQLAlchemy in Python or Liquibase) can automate schema updates and track changes, ensuring that no downtime or significant reengineering is needed. This method allows developers to maintain the structure and performance advantages of a relational database while still adapting to new requirements. Incremental changes can include:

- Adding nullable columns to accommodate new data types.
- Using default values for new columns to ensure compatibility with existing data.
- Creating new indexes as needed to maintain query performance.

If a library needs to store new metadata for each book, such as its genre or digital format, a new column can be added to the books table incrementally:

```
ALTER TABLE books ADD COLUMN genre VARCHAR(100);

ALTER TABLE books ADD COLUMN digital_format VARCHAR(50) DEFAULT 'None';
```

Existing records remain unaffected, while new records can start using the additional fields immediately.

B. Column-Oriented Tables

Another strategy for field flexibility involves using column-oriented tables, which are structured to allow easy addition and modification of fields. In column-oriented databases, data is stored in columns rather than rows, making it highly efficient to add new fields or update existing ones. These databases (e.g., Apache HBase or Google Bigtable) are especially useful in systems where not all records share the same structure. For example, in a library system, you might have some books that contain additional information, such as author biographies or associated multimedia content, while others do not.

Using a column-oriented approach, these additional fields can be added for specific entries without forcing the entire dataset to conform to the new structure. This also allows efficient querying of specific fields, improving flexibility when handling large amounts of varied data. For example, if only certain types of books (e.g., rare manuscripts) require special

attributes like "restoration history", a columnar table structure allows these fields to be included only for relevant records without affecting the rest of the data.

C. Using JSON Data Types

A highly flexible and modern approach to enhancing field flexibility is by utilizing JSON data types within a relational database. Several RDBMSs, such as PostgreSQL and MySQL, now offer native support for JSON data, allowing developers to store semi-structured or unstructured data alongside traditional relational data. This strategy is particularly useful when the data structure is not fully known in advance or when rapid changes in data fields are expected.

Using JSON, fields can be added or modified on the fly, without altering the overall schema of the table. This is ideal for systems like library databases, where books, authors, and other entities may have variable attributes that are not always predictable. For instance, a library might need to store additional metadata for certain books, such as links to multimedia resources, custom annotations, or external references. Rather than modifying the schema each time a new attribute is needed, a JSON column can be used to store this variable data, offering flexibility without compromising the integrity of the relational data. PostgreSQL implements JSON records using the JSONB data type. It is an efficient format for storing JSON data in a decomposed binary representation [14].

D. Flexibility for Library Application

Comparing among those three methods, each method offers distinct advantages depending on the specific requirements of a system. Incremental schema changes are ideal for systems that rely heavily on predefined structures and require the assurance of relational integrity, such as library management systems. This method allows for controlled, gradual evolution of the database, ensuring that new fields can be added with minimal disruption to existing data or queries. However, it requires careful management of migrations and can become cumbersome when faced with frequent or unpredictable changes.

On the other hand, column-oriented tables excel in environments where data structures vary significantly across records, providing efficient querying of specific fields without affecting the entire dataset. This approach is particularly useful in large-scale analytics systems, where different records may not have a uniform set of fields. However, it lacks some traditional relational capabilities required for complex transactions, making it less suited for systems where relationships between entities are key.

Using JSONB data types, meanwhile, strikes a balance between flexibility and relational structure. It allows for schema-less, semi-structured data to be stored directly within a relational database, providing the benefits of both worlds. JSONB fields are perfect for handling dynamic, evolving data without the need for frequent schema modifications. This is particularly useful when the application must handle irregular or custom attributes, such as in a library system that needs to store metadata for certain books or patrons. However, JSONB based queries can be less efficient than traditional SQL queries, and overuse can lead to performance degradation if not carefully managed.

In summary, while incremental schema changes offer stability and control, column-oriented tables excel in scenarios requiring high flexibility at the cost of relational power. JSONB data types provide a versatile middle ground, allowing for both flexibility and structure, though they must be used judiciously to avoid potential performance issues. The choice between these methods largely depends on the balance between flexibility, performance, and the need for relational integrity in the application.

Both column-oriented tables and the JSONB data type offer significant flexibility to library systems, particularly when adapting to evolving data needs without requiring modifications to the underlying table structure. Column-oriented tables provide a stable and structured approach, ensuring efficient data retrieval for specific fields, especially when dealing with large datasets. They are highly optimized for read-heavy operations and analytical queries, offering a more consistent and predictable performance. However, the rigidity of this structure means that it may not be as agile when accommodating diverse or irregular data.

In contrast, JSONB data types offer a more dynamic and flexible way to handle semi-structured data within a relational database. By allowing for schema-less data storage, JSONB enables library systems to handle varying data attributes, such as custom metadata for books or patrons, without frequent schema alterations. This makes JSONB particularly powerful for managing diverse or evolving datasets. However, the flexibility of JSONB comes with the potential for field inconsistencies, as there is no enforced structure across all records. Despite this, JSONB has the advantage of using standardized formats for metadata, making it ideal for interoperability with external systems and APIs.

This study aims to compare these two approaches, column-oriented tables and JSONB data types, specifically in terms of query performance within the context of a library management system. Understanding the trade-offs between stability and flexibility, and how they impact query efficiency, will provide valuable insights into which method is best suited for specific use cases in a dynamic, data-rich environment.

III. Methods

To conduct a thorough comparison between the two methods of field flexibility, a comprehensive dataset was collected from the University of Indonesia's Faculty of Computer Science library. This dataset consists of 41000 records, each encompassing five essential fields: call number, title, author, publisher, and the number of collections. These fields were chosen for their relevance in typical library operations, allowing to focus on the core aspects of library data management. The records represent a diverse array of library materials, making this dataset particularly suitable for analyzing the performance of different data storage formats in a real-world context.

The collected data was stored in two distinct formats: one using column-oriented tables and the other using JSONB data types. This dual approach was to evaluate how each method handles common library operations. Then, a series of standard queries was executed in library applications, such as searching

for specific titles, retrieving collections by author, and counting the total number of items in each category.

Employing these queries was to simulate everyday usage scenarios that librarians encounter, ensuring the results reflect practical performance metrics. Fig. 1 and Fig. 2 illustrate the implementations of the tables for both column-oriented and JSONB data type methods, offering a clear visual representation of them.

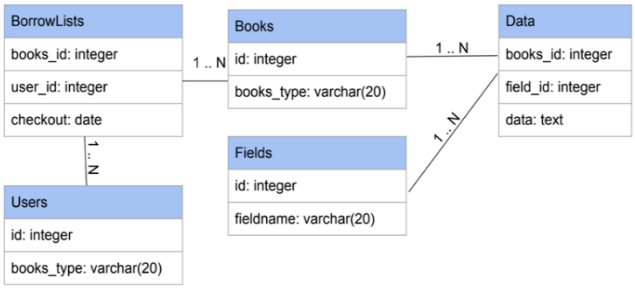

Fig. 1. Simplified tables implementation of the column-oriented method.

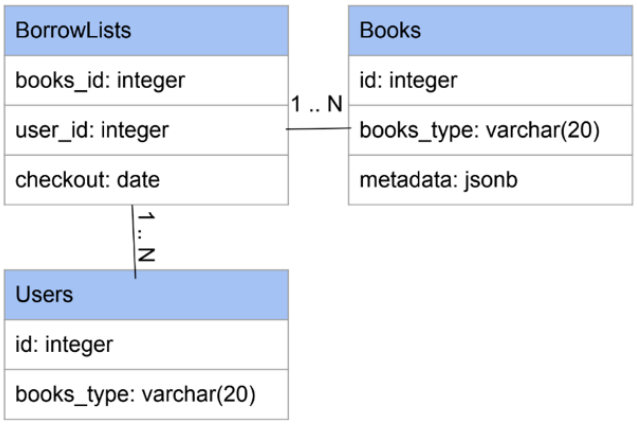

Fig. 2. Simplified table implementation of the JSONB data type method.

In addition, Table I specifies the queries used in this experiment, emphasizing the data retrieval processes employed. PostgreSQL was utilized for database running on an Apple M1 CPU with 8GB of RAM to support these implementations. This enables analyzing the data related to experimental methods.

Finally, the analysis of average execution times of each query across both data formats was performed to draw meaningful conclusions about their efficiency. Measuring how quickly each method processed the queries aimed to identify any significant differences in performance that could influence the choice of data handling strategy for library systems.

To ensure the accuracy and reliability of the results, each query was executed five times, and the average execution time for each query was calculated. To maintain fairness and accuracy between queries, the database server is restarted after each run. This step clears the query cache, preventing any residual data from influencing subsequent results. Averaging the execution times can gain a more accurate representation of how each data format performs under consistent conditions.

TABLE I. QUERIES USED IN THE EXPERIMENT

ID	Task	SQL Query on column-oriented method	SQL Query on JSONB data type method
Q1	Retrieve books with keyword ‘Donald’ in author name	select books_id from data d, fields f where f.fieldname='author' and f.id=d.field_id and d.data like '%Donald%';	select books_id from books where metadata->>'author' like '%Donald%';
Q2	Retrieve books with keyword ‘Donald’ or ‘Knuth’ in author name	select books_id from data d, fields f where f.fieldname='author' and f.id=d.field_id and (d.data like '%Donald%' or d.data like ‘%Knuth%’);	select books_id from books where metadata->>'author' like '%Donald%' or metadata->>'author' like ‘%Knuth%’;
Q3	Retrieve books with keyword ‘Donald’ and ‘Knuth’ in author name	select books_id from data d, fields f where f.fieldname='author' and f.id=d.field_id and (d.data like '%Donald%' and d.data like ‘%Knuth%’);	select books_id from books where metadata->>'author' like '%Donald%' and metadata->>'author' like ‘%Knuth%’;
Q4	Retrieve books with keyword ‘Donald’ but not ‘Knuth’ in author name	select books_id from data d, fields f where f.fieldname='author' and f.id=d.field_id and (d.data like '%Donald%' and d.data not like ‘%Knuth%’);	select books_id from books where metadata->>'author' like '%Donald%' and metadata->>'author' not like ‘%Knuth%’;
Q5	Retrieve books with keyword ‘Donald’ in author name and keyword ‘Computer’ in title	select d1.books_id from data d1, data d2, fields f1, fields f2 where (d1.data like '%Donald%' and f1.fieldname='author' and d1.field_id=f1.field_id) and (d2.data like '%Computer%' and f2.fieldname='title' and d2.field_id=f2.field_id) and d1.books_id=d2.books_id;	select books_id from books where metadata->>'author' like '%Donald%' and metadata->>'title' like ‘%Computer%’;

This repeated execution also allows observing patterns and identifying any potential performance bottlenecks in the system. For instance, certain queries may initially execute faster due to caching, but when repeated multiple times, underlying inefficiencies may become apparent. Running the queries in a controlled environment and taking multiple measurements can assess the stability and consistency of each method over time, which is crucial for applications that demand high availability and consistent performance.

Moreover, calculating the average time across multiple runs provides a more robust foundation for comparison between the two data formats. It ensures that the findings are not based on a one-time performance outlier, but reflect the typical performance behavior of both approaches. This method of testing is essential for drawing reliable conclusions about which data format offers better query execution performance, particularly for the dynamic and often data-intensive operations of a library management system.

IV. RESULTS

This section presents the findings from performance analysis of the two data storage methods: column-oriented tables and JSONB data types. The comparison of execution times of various common queries used in library management systems highlights any significant differences between the two approaches. Additionally, the implications of these results are examined in terms of how each method performs in terms of query efficiency, flexibility, and scalability. The insights gained from this analysis will help determine which approach is better suited for managing dynamic and evolving library data.

A. Table Size

Before diving into the performance analysis, first compare the storage sizes of the tables in both the column-oriented and JSONB data type approaches. This comparison is essential for understanding the storage efficiency of each method, as data size can directly impact query performance and system resources. The findings revealed that the column-oriented approach resulted in a larger table size, approximately 13 MB, while the JSONB data type approach had a smaller footprint, occupying only around 9 MB.

The larger size of the column-oriented table can be attributed to its structured nature, where each field is explicitly defined and indexed. On the other hand, the JSONB data type offers more compact storage because it allows for flexible, semi-structured data representation without requiring predefined schema constraints. This reduced storage size may improve space efficiency, especially when dealing with dynamic or variable data. This size comparison lays the groundwork for evaluating the overall efficiency of each approach in handling real-world library data.

B. Query Performance

Table II presents the mean execution time recorded for each query across all scenarios. These averages were computed by running each query multiple times and aggregating the results to reduce the influence of outliers or transient system fluctuations.

TABLE II. AVERAGE TIME OF EACH QUERY

Query	Column-oriented method	JSON data type method
Q1	58.03 ms	33.53 ms
Q2	77.40 ms	42.60 ms
Q3	41.74 ms	36.46 ms
Q4	58.52 ms	27.90 ms
Q5	78.22 ms	32.69 ms

By focusing on average performance rather than single run outcomes, the analysis ensures a more reliable representation of each data model's behavior under typical operating conditions. Fig. 3 presents a comparative performance chart derived from the data in Table II.

The results indicate that the JSONB data type consistently demonstrates faster query execution times compared to the column-oriented approach. This performance advantage can be attributed to the reduced need for data transformation and parsing when working directly with native JSON structures. In contrast, column-oriented storage often requires additional processing steps, such as schema enforcement, column reconstruction, or intermediate serialization, which increase computational overhead during query execution.

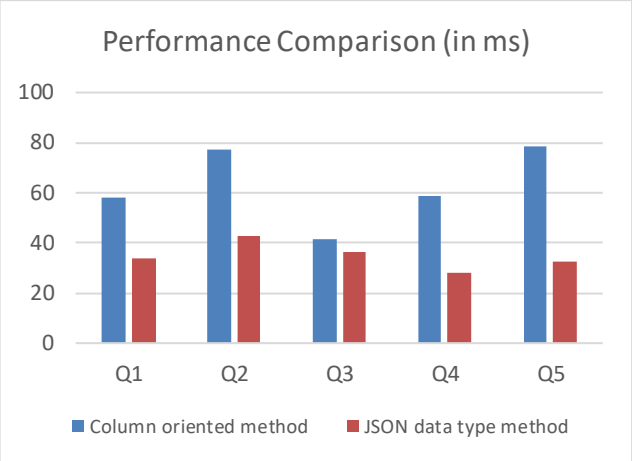

Fig. 3. Performance comparison chart.

V. DISCUSSION

From a systems-level perspective, the superior performance of JSONB indicates that workloads involving semi-structured or nested data benefit significantly from storage models optimized for hierarchical formats. By allowing complex metadata to be stored and accessed directly without extensive schema transformation, JSONB reduces the processing overhead commonly associated with traditional column-oriented designs. This efficiency becomes particularly important in library information systems, where metadata attributes vary widely across collections and evolve over time, requiring a storage model that can adapt without sacrificing performance.

These findings also emphasize the importance of aligning data representation with query patterns and application requirements. When search operations frequently involve flexible attributes such as author names, titles, or custom metadata fields, JSONB enables more direct and expressive querying compared to the join-intensive structure of column-oriented tables. As a result, JSONB minimizes relational reconstruction costs and simplifies query logic, leading to lower latency and more predictable performance in real-world usage scenarios. This demonstrates that performance is not solely determined by hardware or indexing strategies, but is strongly influenced by how well the data model matches the access patterns of the system.

Overall, the consistent outperformance of JSONB across all evaluated scenarios underscores its effectiveness for environments that demand both flexibility and fast access to complex data structures. While column-oriented storage offers stronger structural enforcement and uniformity, its rigidity introduces overhead when handling dynamic metadata. JSONB, therefore, represents a compelling alternative for modern library systems, where adaptability and performance are equally critical, provided that appropriate validation and governance mechanisms are implemented to preserve data quality and consistency.

VI. Conclusion

The results of this study show that the JSONB data type consistently delivers faster query execution times than the column-oriented method across all evaluated scenarios. This performance benefit is largely due to JSONB's ability to store and access hierarchical, semi-structured data without requiring extensive transformation. However, despite its advantages in speed and flexibility, the JSONB data type also presents notable limitations. Its schema's flexible nature can complicate data validation, especially in structured domains such as library information systems, where consistent formats for book records, borrower data, and classification codes are essential.

While the JSONB data type can improve performance and reduce overhead, it also creates challenges in maintaining data consistency and integrity. In a library system, variation in field naming, missing attributes, or inconsistent structures can arise easily due to the lack of enforced schema rules. JSON documents may also contain duplicated or redundant nested data, making synchronization and updates more error-prone. Additionally, the nested structure of JSON can complicate indexing and query optimization, which are critical for fast retrieval of books, circulation statistics, or user records. In contrast, column-oriented storage, with its rigid schema and normalized structure, inherently promotes uniformity and reduces inconsistency across records.

This research has both theoretical and practical implications for the design and management of database systems in library information environments. From a theoretical perspective, JSONB can bridge the gap between rigid relational models and fully schema-less NoSQL approaches. Hierarchical storage within relational systems can significantly improve query performance for semi-structured metadata. This study also extends existing literature by providing concrete performance evidence in the underexplored domain of library and heritage metadata management, thereby offering a foundation for future research on hybrid data models and adaptive schema design.

From a practical perspective, it provides guidance for system architects that library systems can adopt flexible metadata storage without incurring significant query overhead. At the same time, the findings highlight the need for complementary governance mechanisms to mitigate the risks associated with schema flexibility. By balancing performance, flexibility, and data consistency, this research informs the development of scalable and sustainable digital library systems capable of adapting to future metadata requirements.

Future work should explore mechanisms to mitigate the consistency challenges associated with JSON-based storage. Further evaluation across larger, more complex library datasets may also clarify the conditions under which JSON's performance benefits outweigh its consistency drawbacks. Additionally, examining the impact of different indexing strategies, storage engines, and transaction loads could provide deeper insights into optimizing JSON for use in library information systems and other domains requiring both flexibility and high data reliability.

Acknowledgment

This study's research is supported by the Faculty of Computer Science, Universitas Indonesia, Grant number: NKB-20/UN2.F11.D/HKP.05.00/2025.

Declaration on Generative AI

The authors acknowledge the use of generative AI to assist in improving the clarity, grammar, and structure of the manuscript. The content, analysis, and conclusions remain the sole responsibility of the authors.

References

[1] G. Feuerlicht. "Database Trends and Directions: Current Challenges and Opportunities". Proceedings of DATESO. 2010.

[2] S. Akinola. "Trends in Open Source RDBMS: Performance, Scalability and Security Insights". Journal of Research in Science and Engineering (JRSE). 2024.

[3] T.R. Kochtanek. "Library information systems : from library automation to distributed information access solutions". Libraries Unlimited. 2002.

[4] S. Weibel. "The Dublin Core: A Simple Content Description Model for Electronic Resources". Bulletin of the American Society for Information Science and Technology. 1997.

[5] Library of Congress. "MARC STANDARDS". https://www.loc.gov/marc/index.html (accessed January 20, 2026).

[6] Z.H. Liu, D. Gawlick. "Management of Flexible Schema Data in RDBMSs-Opportunities and Limitations for NoSQL". CIDR. 2015.

[7] Hermansyah, Y. Ruldeviyani, R.F. Aji. "Enhancing query performance of library information systems using NoSQL DBMS: Case study on library information systems of Universitas Indonesia". 2016 International Workshop on Big Data and Information Security (IWBIS). 2016

[8] N. Bansal, S. Sachdeva, L.K. Awasthi. "Are NoSQL Databases Affected by Schema?". IETE J. Res. 70. 2024

[9] G. Turutin, M. Puzevich. "PostgreSQL JSONB-based vs. Typed-column Indexing: A Benchmark for Read Queries". International Journal of Computer (IJC). 2025.

[10] Y. Vazquez Ortíz, L.M. Pierre, A.R. Sotolongo León. "Semi-structured Data Management in PostgreSQL: Competing with MongoDB's Performance". Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management. 2016.

[11] A. Amato. "Towards the Interoperability of Metadata for Cultural Heritage". Lecture Notes on Data Engineering and Communications Technologies 176. 2023.

[12] F.C. Paletta, C. Wijesundara. "Metadata Principles, Guidelines and Best Practices: A Case Study of Brazil and Sri Lanka". Proceedings of the International Conference on Dublin Core and Metadata Applications. 2024.

[13] M. Rosyihan Hendrawan, A. Mat Isa, A. Zam Hariro Samsudin. "Metadata Interoperability for Cultural Heritage Digital Repositories: A Case Study in Indonesian World Heritage Site Memory Institutions". International Journal of Academic Research in Business and Social Sciences. 2024.

[14] PostgreSQL. "PostgreSQL 18.1 Documentation". https://www.postgresql.org/files/documentation/pdf/18/postgresql-18-A4.pdf (accessed January 20, 2026).