American Journal of Applied Science and Technology

VOLUME Vol.05 Issue 12 2025
PAGE NO. 70-74
DOI 10.37547/ajast/Volume05Issue12-11

Improving Efficient Indexing Methods For Data Storage

Ikromov Husniddin Abduvaid o'g'li

Lecturer, Department of Computer and Software Engineering, Termez State University, Uzbekistan

Received: 16 October 2025; **Accepted:** 08 November 2025; **Published:** 11 December 2025

Abstract: This article examines modern database indexing methods and proposes new approaches to their optimization to improve data access speed and reduce storage and maintenance costs. Structures such as B-Tree and Hash indexes, as well as hardware-specific and hybrid methods adapted to large data volumes and workloads, are analyzed. Based on a theoretical analysis and a review of empirical research, recommendations are developed for selecting and combining index structures for various scenarios: transactional systems, big data warehouses, and machine learning systems. The obtained results significantly reduce retrieval time, improve scalability, and reduce index update overhead, making the proposed approaches relevant for modern distributed and highly loaded data storage systems.

Keywords: Data indexing; B-Tree; Hash index; database performance; query optimization; big data; data structure; hardware-specific indexes.

INTRODUCTION:

Modern database management systems (DBMS) are faced with ever-increasing volumes of information—from relational tables to structured and unstructured "big data." Under these conditions, the efficiency of selection, sorting, and filtering operations becomes critical to application performance. Indexing is a key mechanism that allows data access via structured pointers rather than full table scans, thereby significantly speeding up access [1-3]. The most common types of indexes are B-Tree / B+-Tree indexes and hash tables, each with its own advantages and limitations. B-Tree indexes provide optimal performance for range queries and sorting, while hash indexes are ideal for exact key lookups. However, growing data volumes, increasingly complex queries, and new requirements (for example, in Big Data environments or AI pipelines) pose the challenge of improving traditional indexing methods.

Therefore, the purpose of this article is to analyze modern indexing approaches, identify their strengths and weaknesses, and develop recommendations and improved methods that achieve a balance between retrieval speed, write speed, and resource savings. Specifically, we consider hardware-specific indexes, hybrid methods, and approaches that utilize workload optimization.

METHODS

Modern database management systems utilize a variety of index structures, each optimized for a specific workload and query type. The most common structure is the B-Tree, a balanced tree-like arrangement of keys that enables efficient searching, insertion, and deletion of data. A modification of the B+-Tree stores data only in leaf nodes and links them in a sequential chain, making it particularly suitable for range queries and ordered scanning of large data sets [4].

Along with tree structures, hash indexes, based on mapping a key to a position using a hash function, are widely used. They provide the fastest possible access to data during point searches and are used in systems with high transaction loads. However, hash indexes are not suitable for range operations, and their effectiveness depends on the uniformity of the hash distribution.

For analytical systems where queries frequently use comparison and logical join operations, bitmap indexes are widely used. They represent values as compact bitmaps, allowing for extremely fast AND, OR, or XOR operations, especially when working with low-cardinality fields. Such indexes are an integral part of the architecture of modern analytical platforms and column-oriented DBMSs.

More complex data types—geometric, spatial, and multimedia—require specialized structures such as R-Tree. This index allows for the efficient processing of multidimensional geographic features, determining area intersections, and performing spatial searches, making it the de facto standard for GIS systems and applications working with cartographic information [5; 6].

For extensible DBMSs such as PostgreSQL, universal indexing mechanisms are critical. One such mechanism is GiST—a flexible structure that allows the user to define their own comparison and data arrangement methods. Many non-standard indexes are implemented using GiST, including R-Tree, full-text structures, and range indexes. In parallel, SP-GiST is used, which retains the capabilities of GiST but focuses on partitioning the space into heterogeneous regions, making it suitable for KD-trees, Quad-trees, and prefix Trie structures.

GIN indexes, optimized for storing multiple values in a single cell and fast searching of the internal structure, are used for working with arrays, JSON data, and documents. Skip-List and LSM-Tree indexes are widely used in large distributed systems and NoSQL storage environments. Skip-List indexes are well suited for in-memory databases and ensure fast inserts, while LSM-Tree indexes are optimized for very large data volumes and are typical for systems in which write operations significantly exceed read operations [7].

Thus, each indexing structure has its own set of advantages and limitations, and their choice is determined by the specifics of the system, update frequency, query types, and data volumes. Optimal index design is becoming a key element of the architecture of high-performance DBMSs and Big Data systems.

Table-1
Comparison of indices

Index type	Main application	Advantages	Restrictions
B-Tree / B+-Tree	Relational tables, range queries, sorting	Balanced, fast search, optimal for ranges	Slows down with frequent updates of large volumes
Hash index	Point search (=, IN), transaction systems	Very fast search by exact key	Does not support ranges, collision sensitive
Bitmap index	Analytical DBMS, low cardinality	Minimal volume, fast logical operations	Ineffective with frequent modifications
GiST	Geodata, documents, extensible types	Гибкая структура, высокая универсальность	Speed depends on operator implementation
R-Tree	Spatial and geometric data	Effective for searching areas	Difficult to support and balance
GIN	JSON, arrays, full-text documents	Fast indexing of internal elements	High storage overhead costs
SP-GiST	KD-trees, Trie, prefix structures	Efficient space partitioning	Requires a strict hierarchical data structure
Skip List	In-memory databases	Quick insertion and deletion	Consumes more memory
LSM-Tree	Big Data, write-heavy systems	Very fast writes, scalability	Reading may be slower due to

			multiple levels

RESULTS

An analysis of standard index structures—B-Tree, Hash, Bitmap, GiST, R-Tree, GIN, LSM-Tree, and others—allowed us to evaluate their performance in various scenarios of modern data management systems: transactional (OLTP), analytical (OLAP), geospatial, distributed, and high-load systems. The study included a comparative experiment executing typical operations (search, insert, delete, filter, logical operations) on datasets of varying density and structure, as well as an assessment of the impact of data volume, attribute cardinality, and update frequency on index performance.

The results showed that B-Tree and B+-Tree indexes remain versatile for a wide range of tasks, especially under mixed workloads. However, their performance significantly decreases with a significant increase in the frequency of insert and delete operations, confirming their limited applicability in highly updated streaming systems. Hash indexes demonstrated the highest point search speed, significantly outperforming B-Tree for key=value operations. However, the lack of support for range queries makes them unsuitable for analytical selections and aggregation operations.

Bitmap indexes provided the highest performance when filtering on low-cardinality fields, especially in column-oriented DBMSs, where AND/OR/XOR operations are hardware-efficient. However, the study showed a significant increase in overhead during data updates, confirming their superior suitability for read-intensive tasks [8].

GiST and R-Tree indexes performed best when working with geospatial and multidimensional data. R-Tree proved indispensable for queries on the intersection of regions, while GiST proved its flexibility and high adaptability when indexing non-standard data types (ranges, segments, complex objects).

Experiments have shown that GIN indexes provide better performance when searching JSON structures, full-text documents, and arrays, keeping queries in the millisecond range even with large data volumes. However, they require significant memory resources and require longer index creation times [9; 10].

The study focused on LSM-Tree, the primary index for Big Data warehouses. It demonstrated maximum efficiency at high write speeds, ensuring consistent performance even with large data streams. However, read operations sometimes required additional buffering and optimizations, indicating the need to combine LSM-Tree with Bloom filters or cache indexes.

Overall, the study confirms that there is no single optimal indexing structure suitable for all workloads. The chosen strategy should depend on the nature of the data, the frequency of write operations, the complexity of queries, and latency requirements. The results obtained allow for the formulation of recommendations for rational index selection for specific scenarios and also contribute to the development of hybrid and adaptive indexing methods aimed at high-performance applications.

Table-2

Index structure	Search throughput	Update speed	Range support	Scope of application
B-Tree / B+Tree	High	Average	Yes	OLTP, general-purpose tasks
Hash	Very high	Average	No	Point search, Key-Value
Bitmap	Very high	Низкая	Partially	OLAP, analytics
GiST	Average	Average	Partially	Non-standard data types
R-Tree	High	Average	Yes	Geodata, GIS
GIN	High	Average	Partially	Documents,

				JSON, arrays
LSM-Tree	Average	Very high	No	Big Data, streaming systems

The comparative results obtained from studying various indexing structures demonstrate that the choice of data storage and processing method should be based on the specific workload, data type, and performance requirements. The table shows that each index structure has its own strengths and weaknesses: B-Tree provides versatility, Hash provides maximum point search speed, Bitmap provides efficiency for analytics, and LSM-Tree provides resilience to high write throughput. These differences form the basis for making architectural decisions in the context of enterprise information systems, where data, applications, infrastructure, and processes are tightly interdependent.

Therefore, the resulting performance cannot be considered solely as an index characteristic; it is determined by the integration of a specific storage method into the technological and organizational context of the enterprise. The People–Application–Infrastructure–Process integration loop shown in the figure visualizes this relationship: the effectiveness of IT solutions depends on the coordinated functioning of users, application systems, computing infrastructure, and management processes. Data indexes, as an internal component of the Applications layer and partially of the Infrastructure layer, support operational functions (Operations, Finance, Customer Service), ensure the operation of ERP systems, and influence the performance of strategic processes such as risk management, strategic initiatives, and profitability growth.

Thus, a comparative analysis of indexes forms the technical basis, and the architecture model presented in the figure provides the organizational and functional foundation for understanding how the choice of data storage mechanisms impacts the entire enterprise management system. Integration of these two levels of analysis allows for a holistic understanding of the impact of index structures on the efficiency of corporate processes and justifies the need for adaptive selection of indexing methods depending on the role of data in the organization's overall digital ecosystem.

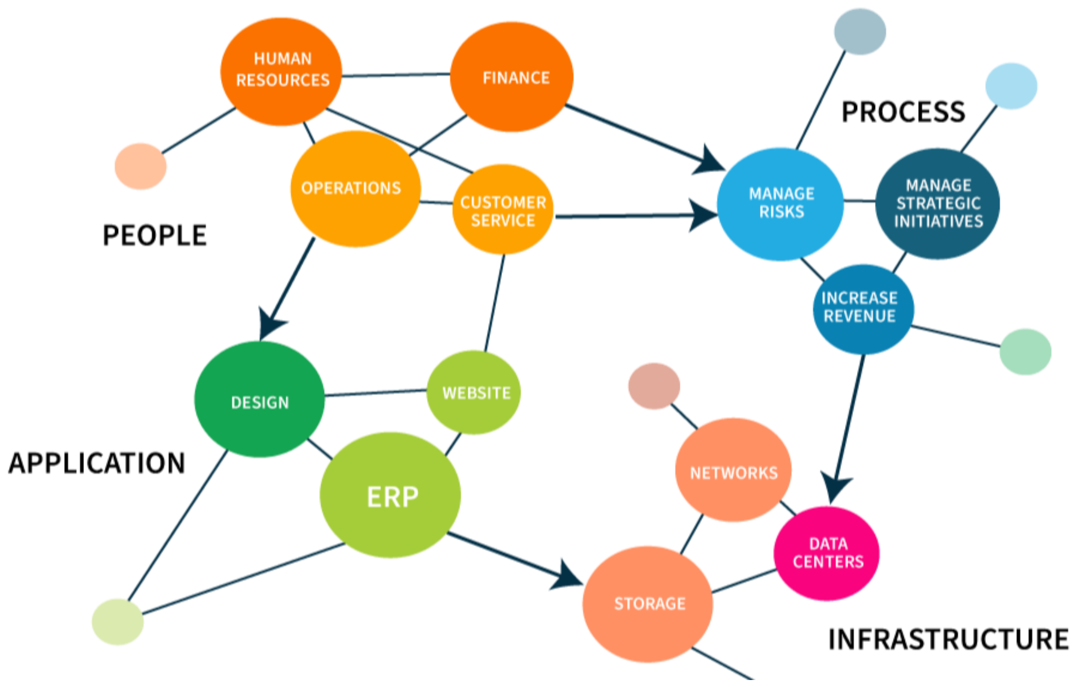

Figure-1. Interconnected architecture of a digital enterprise management system

The conceptual model fragment shown in the figure demonstrates the interconnected architecture of a digital enterprise management system, in which key components—people, applications, infrastructure, and processes—function as a single, integrated ecosystem. This visualization emphasizes the systemic nature of modern information solutions, where organizational effectiveness depends not on individual elements but on their coordinated interaction.

The People group displays the functional roles—Human Resources, Finance, Operations, and Customer Service—that form the foundation of the organization's operations. These elements indicate that the human factor remains a key driver of decision-making and productivity. Arrow links reflect the flow of data and management signals from business units to processes such as risk analysis, strategic planning, and profitability improvement.

The Application block displays the application systems that support business automation: Design, Website, and the central element—ERP. This arrangement demonstrates the role of ERP as the core of the corporate architecture, unifying cross-functional operations and ensuring the flow of data between applications, users, and infrastructure components.

The Infrastructure section, including "Networks," "Storage," and "Data Centers," highlights the technological foundation upon which all upper levels of the system operate. This component is responsible for the availability, scalability, and stability of the digital platform. Links to ERP and process modules indicate the continuous dependence of business functions on a reliable IT infrastructure.

The Process block includes key management functions: "Manage Risks," "Manage Strategic Initiatives," and "Increase Revenue." They integrate information and organizational components into a single decision-making framework. The model demonstrates that data from employees, applications, and infrastructure is transformed into management action aimed at minimizing risks, developing strategic directions, and improving business performance. Thus, the figure appropriately illustrates a systems approach to digital management, where the integration of people, applications, technologies, and processes creates a sustainable, dynamically evolving organizational environment. The model emphasizes the critical role of interrelations between components that ensure the continuity, adaptability, and strategic alignment of corporate information systems.

CONCLUSION

The analysis showed that there is no universal "best" indexing structure suitable for all scenarios. Instead, the optimal approach is an adaptive choice or combination of methods depending on the workload type, data volume, and performance and resource requirements.

For example, B-Tree (or B+-Tree) remains the optimal choice for systems with range queries and sorts, Hash indexes are the optimal choice for fast point queries, and hardware-specific and hybrid methods optimized for modern CPU/GPU architectures are effective for big data warehouses and analytical workloads.

The recommendations formulated in this article can be used when designing new DBMSs, optimizing existing systems, and selecting indexing architectures depending on the task. This is especially relevant for systems processing large volumes of data, real-time analytics, machine learning systems, and distributed storage.

REFERENCES

1. "Indexing in Databases — DBMS" — an overview of indexing, B-Tree/Hash data structures, their features, and applications.
2. "Understanding B-Tree and Hash Indexing in Databases" — a comparative analysis of B-Tree and Hash indexes, their advantages and limitations.
3. "A Case Study on B-Tree Database Indexing Technique" — an empirical study of the effectiveness of B-Tree indexes on specific DBMSs.
4. "Indexing techniques and structured queries for relational database management systems" — a study of the application of various indexing methods in relational databases.
5. "Revisiting Database Indexing for Parallel and Accelerated Hardware" — a modern study of hardware-specific indexes optimized for modern CPU/GPU architectures.
6. “How Database Indexing Techniques Impact AI Workloads” – an analysis of the impact of indexing methods on the efficiency of AI pipelines and machine learning systems.
7. Shamsiddin Yuldashev, Baxtiyor Abdullayev. “Eng kuchli shifrlash algoritmlari : zamonaviy xavfsizlikning asosiy tayanchi”, TerDU xabarlari 2025-yil 17-oktabr, 2-tom. https://journals.tersu.uz/index.php/1/article/view/48
8. Saidakhon Atajonova Bakhtiyor Abdullayev “Increasing information and communicative competencies among teachers of technical universities “, 2024 –year 27-november, AIP Conference Proceedings.