IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, May 2026
Impact Factor: 8.2

A Survey of Data Storage, Retrieval, and Optimization Methods in Modern Databases

Dr. Neha Upadhyay
Lakshmi Narain College of Technology (MCA), Bhopal, Madhya Pradesh, India 462022
neha.upadhyay887@gmail.com

Abstract: *Modern database systems require performance optimization techniques and data management approaches to handle the increasing challenges which modern data-driven applications face at larger operational scales. This paper presents a complete review of all current database systems which includes their data storage methods and retrieval techniques and their optimization approaches. The paper examines advanced storage technologies which include relational databases and NoSQL databases and hybrid row-column databases to assess their capabilities for handling both structured data and semi-structured data. The research study investigates NVM-aware storage systems together with data compression technologies which enable better storage efficiency and quicker access performance. The research examines efficient query execution techniques which include indexing systems based on hash and tree ISAM methods and cache-aware optimization techniques that improve execution performance. The paper presents fundamental performance tuning techniques which include query optimization methods and resource allocation strategies and adaptive self-tuning mechanisms that sustain system performance through varying workload conditions. The paper establishes a comprehensive framework which describes how current database technologies impact system scalability, operational performance and fast data processing*

Keywords: Database Systems, Performance Optimization, Data Storage, Query Processing, Indexing Techniques, Scalability.

I. INTRODUCTION

Modern applications produce large amounts of data which need efficient storage systems that can retrieve data quickly while processing information reliably. Database performance optimization needs to occur because it enables systems to handle complex workloads while maintaining their responsiveness and scalability requirements. From the viewpoints of users as well as database system operators and maintenance staff, database performance optimization is essential[1]. Performance requirements have been severely challenged by database application systems due to the ever-growing complexity and diversity of database workloads[2]. In service-level agreements (SLAs), it is standard practice to limit the proportion of queries or transactions that perform poorly. Therefore, performance optimization—that is, keeping an eye on the database system's operational conditions and resolving issues with performance regressions—is a major responsibility of the operation and maintenance staff. In this context, optimization also involves efficient query processing, indexing strategies, caching mechanisms, and workload management to ensure consistent system performance.

The database used to hold the data is one of the most crucial components when creating an application. NoSQL (Not Only SQL (Structured Query Language)) databases were required as an alternative method of data storage because conventional relational databases were unable to manage these massive amounts of data and process it promptly[3][4]. A non-relational database that does not store data in the conventional relational format is called a NoSQL database. Because NoSQL is not based on tables, it may not always fully meet the requirements of atomicity, consistency, isolation, and durability. The two systems provide two dynamic data models which enable users to scale their storage

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, May 2026
Impact Factor: 8.2

capabilities. This enables them to manage semi-structured data and unstructured data which modern applications require.

Large sets of organized patterns, such as frequent item sets and association rules retrieved from transactional data, sequential motifs inferred from logs and clickstreams, and subgraphs extracted within a networked environment, are frequently produced by modern data analytics pipelines[5]. While algorithms have advanced, the scalability of pattern discovery and the usefulness of pattern mining ultimately depend on how well these findings are collected, indexed, stored, and queried[6][7]. Efficient retrieval of these patterns requires advanced indexing structures and query-processing techniques to enable fast, accurate access.

In general, a distributed database cluster consists of several data storage locations and a coordinated control site. The stored data is divided and dispersed across these locations in accordance with a certain rule[8]. In the meantime, data shards usually have many backups spread across several locations to provide high availability of the distributed database cluster. Selecting a suitable query execution strategy is crucial to lowering data retrieval overhead and accelerating query response. Techniques such as data partitioning, replication, and distributed query optimization play a vital role in improving retrieval efficiency and system scalability.

A. Structure of the paper

The paper is structured as follows: Section II discusses data storage mechanisms in modern databases. Section III covers efficient query processing and execution. Section IV presents performance tuning techniques. Section V provides a critical literature review, and Section VI outlines future research directions.

II. DATA STORAGE MECHANISMS IN MODERN DATABASES

Modern relational database systems often struggle to handle large volumes of data efficiently, especially for quick, cost-effective analysis and processing.Large-scale data applications necessitate the development of novel data storage methods outside of conventional databases[9]. There are two types of data storage: relational databases and non-relational databases[10]. They vary in the kinds of information they hold, how they are constructed, and how they store data. Relational and NoSQL databases have been contrasted extensively, as Figure 1 illustrates.

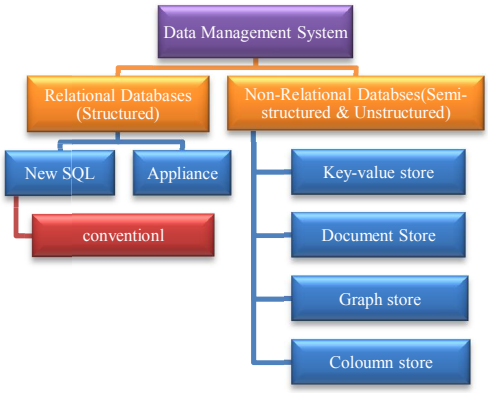

Figure 1: Data collection and storage.

A. Row-Column Hybrid Storage Models in Modern Databases

Row-column hybrid storage models represent a significant evolution in modern database systems, combining advantages of both row-oriented and column-oriented storage approaches[11]. Row storage and column storage are the two types of storage modes used in relational database systems. Tuples are used to store data in the row storage mode. Each tuple's characteristics are all kept together. They must read the full tuple before they can query it for an attribute value. Row-storage technology is used by the majority of relational database systems in use today. While distinct

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, May 2026 Impact Factor: 8.2

columns are kept separately, column stores group all of a table's columns together for storage. The two most common ways to store data are in rows and columns[12]. However, the bottleneck of the conventional approach is rapidly forming in the following areas due to the introduction of new business data, such as the IoT and mobile Internet:

Storage technology single, the existence of short-board technology:Only row or column storage may be used to pick physical data, but both programs have some technological drawbacks and are unable to account for the benefits of row and column storage.

Storage space waste is serious: The existing data warehouse has a row storage structure with a relatively poor data compression ratio, which makes it easy to waste a lot of storage space. Mobile Internet and IoT businesses include both dense and big sparse data.

Some business data storage scheme is not reasonable, as the flexibility is poor: a business analysis system with a lot of data entities, a lot of attributes, and a variety of characteristics between the columns, some of which are dense and highly important for the row to store, while others are sparse and lowly important and appropriate for column storage.

Cannot realize the mixture of the sparse data and the dense data, the highly efficient storage: Compared to the conventional dense data, the new business class data introduced by the sub-system is sparser, and the model is often modified. The two forms of data typically coexist in real-world applications, and neither the relational model nor the non-relational model can adequately address the issue of efficiently storing "hybrid data sets."

Investment costs and business support costs remain high: Column storage is used for online analytical processing, whereas line storage is used for online transactions. However, two or more sets must be deployed since online analytical processing and online transaction processing products are independent of one another. In addition to increasing investment costs, business-oriented solutions necessitate data transfer across several applications, which raises administration and maintenance expenses.

In fact, the database system that relies on row and column storage technologies can only fulfil one of the read and write optimizations; it is unable to address the issue of simultaneous reading and writing optimization. As a result, a database system built on hybrid storage is required[13].

B. NVM-Aware Storage Architectures and File Systems

In order to avoid the operating system page cache and I/O stack and minimize software-induced overhead and delay, NVM-aware file systems often employ a method known as Direct Access (DAX). Applications may access data directly using memory load and save instructions by mapping files on an NVM-aware file system into their address space thanks to DAX. Conversely, DAX is not supported by conventional file systems[14]. Furthermore, certain conventional file systems are built to depend on the sector-write atomicity of the storage media. The Block Translation Table (BTT) is an extra software layer that DCPMMs must maintain in order to handle various file systems and ensure the atomic sector update. The speed difference between NVM-aware file systems and conventional file systems is further increased by the expense of maintaining BTT.

NVM-aware versions of XFS and Ext4 that allow Direct Access are called XFS-DAX and Ext4-DAX. BPFS is an NVM-aware file system that minimizes write amplification and data copying through the use of shadow paging. PMFS supports DAX for programs through the map interface and offers reliable, consistent modifications to file system information. NOVA uses strategies from log-structured file systems to keep a distinct log for every inode, providing great consistency and fast speed.

C. Scalable Data Compression Approaches in Database Systems

The act of transforming a data set into a code to reduce the requirement for data storage and transmission while facilitating data transfer is known as compression. A can be compressed to save time and memory (storage)[15]. The Huffman, Lempel-Ziv Welch, Run Length Encoding, Tunstall, and Shannon Fano methods are only a few of the compression algorithm strategies that may be used and work well.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, May 2026
Impact Factor: 8.2

The process of reducing a file's size from a big to a smaller one is called compression. To make it easier to send a big file with lots of characters, it is compressed. Finding patterns of repetition in the data and substituting them with a certain symbol is how compression operates. There are two techniques for this kind of compression: lossless compression and lossy compression:

Lossless compression is the technique of compressing the original material to make it more succinct without sacrificing information.

Lossy compression is the procedure that creates compressed data from the original data. Different values exist, yet the value of this difference is taken into account without diminishing the significance of the original data.

Figure 2 explains the process of data compression.When data is not compressed, it remains in its original form. The uncompressed data can be processed using any compression technique, including lossless compression. In lossless compression, data is reduced in size without losing any information, resulting in a smaller file than the original.

Figure 2: The data processing of data compression

Some common compression algorithm techniques are as follows:

Shannon Fano Algorithm:

Data compression is a fairly practical way of compression employed in implosion compression, which is utilised in zip files or files in the .rar format. The Modalism SE 6.4 simulator may be used to build the Shannon Fano algorithm using VHDL code, which compresses the data, which can be done by following Equation (1),

Amount Of Compression (Ratio Compression)

$$\frac{AmountDataBitsBeforeCompression}{AmountDataBitsAfterCompression} \quad (1)$$

Run Length Encoding (RLE)

One method for compressing data so that the generated data is lower than the original size is the RLE algorithm[16]. A set of identical data values are saved as a single data item in the simplest kind of lossless data compression, known as it. Icon files, line drawings, and animations are examples of data containing several repeated values in a sequence that benefit greatly from this approach. For typical data, this technique is inappropriate since it raises.

Lempel Ziv Welch:

The LZW algorithm is a dictionary-based lossless compression method. A dictionary is created during the LZW compression process. There are several uses for the LZW technique in compression. This experiment compares the proposed MLZW coding with the traditional LZW coding. Dictionary-based compression outcomes. The compressed bit or codeword result, which must be less than the original file, is the output of the LZW algorithm. The technique may be used on any Bangla compressed text with ease, as it has been modified to comply with the Unicode standard.

III. EFFICIENT QUERY PROCESSING AND EXECUTION IN MODERN DATABASES

Optimizing query execution is essential to improving database speed. One of the first things to concentrate on for improving database speed overall is query optimization, and indexing was crucial in the pursuit of better query execution[17].They can construct data structures that enable quick data retrieval by building indexes on certain columns or combinations of columns.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, May 2026
Impact Factor: 8.2

Indexing Techniques for Efficient Query Execution

Indexes eliminated the need for complete table scans by enabling the database system to swiftly find and retrieve pertinent data. Faster query response times were made possible by the existence of indexes, which improved system efficiency overall and sped up query processing. One or more columns from a database table can be used to create an index that enables effective access to ordered objects as well as rapid random lookups. Two popular designs are used by most index structures. They may be index structures based on trees or hashes:

Hash-based index structures:

In Hashing is a method used in hash-based index structures to efficiently locate records with a certain search key value[18].By calculating a function on the record's search key value, the addresses of the disk blocks that contain the requested record are immediately acquired. Formally, let B represent the set of all bucket addresses and K represent the set of all search key values. A function from K to B is called a hash function (h). Compute h(Ki), which provides the disk block address, in order to insert a record with the search key Ki, as seen in Figure 3.

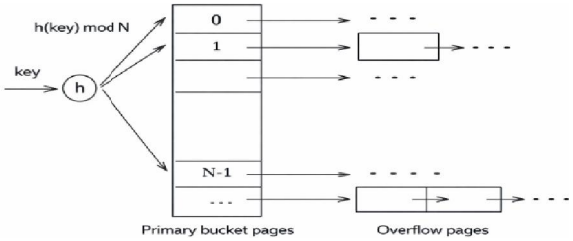

Figure 3: Concept of Hashing

Tree-based index structures:

A tree is a popular data structure that depicts a hierarchical structure with sub-nodes called child nodes and a root node called the parent node. A collection of connected nodes is typically used to depict the tree data structure. The B-Tree, B+-Tree, R-Tree, R+ Tree, and the Indexed Sequential Access Method are common instances of tree-based index structures.

This Table I compares different indexing techniques used in databases based on their working principles, structures, and advantages. It demonstrates that hash-based indexing allows for quick exact searches, while tree-based and ISAM methods provide efficient data retrieval through both range and sequential access methods.

Table 1: Comparison of Database Indexing Techniques

Index Type	Working Principle	Structure	Advantages
Hash-Based Indexing	Uses a hash function h(K) to map search key values directly to bucket addresses	Flat structure (buckets)	Very fast for exact-match queries; reduces lookup time
Tree-Based Indexing	Organises data hierarchically using parent-child relationships	Tree structure (linked nodes)	Efficient for both exact-match and range queries; scalable
ISAM (Indexed Sequential Access Method)	Uses a static index with pointers to records	Fixed tree-like index	Fast retrieval; small index size; efficient sequential access

This approach allows for the retrieval of certain records without searching the complete data set since it includes an index of tables whose items are pointers[19]. The databases can access only the records they require because this index is tiny and fast to scan, often using a binary search. Index nodes, which are fixed when an ISAM file is formed, include an index of tables whose contents are pointers, allowing certain entries to be obtained without searching the entire data

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, May 2026 Impact Factor: 8.2

set. The databases can access only the records they require because this index is tiny and fast to scan, often using a binary search. Index nodes are fixed upon the creation of an ISAM file.

B. Cache-Aware Query Optimization Techniques

In contemporary designs, cache use is becoming increasingly crucial. DBMSs suffer from excessive memory-related processor delays while operating on current architectures, according to a number of workload characterization studies that offer a thorough examination of the temporal breakdown in DBMS execution on a modern CPU[20]. Because of the large number of TCP network I/O, instruction cache misses are another significant cause of performance deterioration in distributed databases[21][22].Some studies concentrate on rearranging the data structure by grouping all values of each attribute in a N-ary Storage Model (NSM), utilizing a Decomposition Storage Model (DSM), or fully organizing the records in a column store in order to make better use of the cache[23][24]. This type of optimization has a detrimental effect on intra-tuple cache locality but advantages the OLAP workload, which usually only requires a few columns. Other methods, such compression and coloring, can also be used to maximize cache use for the main data structure.

In summary, to optimize cache utilization, the following important factors should be taken into consideration:

- Cache line length:These features reveal spatial proximity, thus accessing nearby data would be more effective.
- Cache size:Keeping commonly used data inside at least the L3 cache size would be more effective as well.
- Cache replacement policy:LRU, one of the most often used replacement rules, replaces the least recently used cache line in the event of a cache miss. To attain excellent performance, temporal locality should be utilized.

IV. PERFORMANCE TUNING TECHNIQUES IN MODERN DATABASES

In order to fully comprehend the behaviour of the database system, performance monitoring entails tracking important performance parameters. It concentrated on important parameters including CPU use, disk I/O speeds, and query execution time. Finding trends and anomalies through the process of gathering and evaluating measurements over a lengthy period of time allows for the identification of possible performance bottlenecks.

The involved adjusting memory allocation and disk I/O parameters and query optimization through its fine-tuning process. The team optimized the system configurations to reach an ideal state, which balanced resource usage with system performance. The second essential component of performance optimization involved optimizing query execution plans [25]. Database optimizer execution plan analysis with execution plan optimization are essential procedures needed for achieving effective query execution.

Furthermore, context that improves speed aids in optimizing the database system. Buffer pool sizes, disk caching rules, and parallelism environments were all optimized. The goal is to achieve performance advantages while preserving data security and integrity by matching this context with the workload characteristics and system resources.

A. Real-Time Transaction Optimization via Workload critically

The characteristics of a real-time transaction object include significance, deadline, and priority. Deadline and priority are typical object characteristics. As a result, the deadline and priority properties may have different values in each transaction class instance[26]. Additionally, many transaction class objects may share the same deadline and priority properties. Even when a transaction is being executed, the priority property may change. Nevertheless, when the system's transactions are diverse, some are more significant than others. Unlike the deadline feature, the transaction's relevance is independent of its arrival time.

Priority-based conflict resolution was a feature of the majority of earlier real-time concurrency control methods. Here, transactions are given "priorities" based on their deadlines, criticality, or both, either explicitly or implicitly. A transaction's criticality reveals how important it is. In reality, though, these two criteria might occasionally conflict with one another. As a result, they replace priority in the conflict resolution of optimistic concurrency control with the significance (or criticalness) of the transactions. This avoids the problem of priority-based conflict resolution while

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, May 2026
Impact Factor: 8.2

integrating deadlines and criticality such that the ultimate objective is to maximize the net worth of the performed transactions for the system, in addition to ensuring that the more important transactions meet their deadlines.

B. Efficient Resource Allocation & Management

Static configurations and rule-based policies are the foundation of conventional database resource management. Although they offer rudimentary workload segregation, Oracle's Resource Manager and other technologies need a lot of human adjustment[27]. Adaptive techniques that react to shifting workload characteristics have been the subject of recent research. For heterogeneous CPU-GPU systems, they suggest clever data placement and query routing, obtaining notable performance gains through workload-aware scheduling. But because their strategy relies on analytical models rather than learning-based techniques, it is less flexible in response to unexpected workload patterns.

The challenge of scheduling heterogeneous database workloads has received increasing attention. This research focuses on optimizing workload distribution while creating detailed modeling methods which support resource management operations in big data environments. The systems exhibit the complexity of modern database workloads, yet rely on conventional optimization methods. Het Exchange provides CPU-GPU parallelism through its JIT-compiled engine, which demonstrates the advantages of heterogeneous processing yet lacks a mechanism for dynamic resource allocation among competing workloads.

The operation of multi-tenant database systems introduces additional challenges for task scheduling. The system uses workload prediction to optimise queries in parallel DBMSs,demonstrating the importance of understanding resource requirements in advance. Their method optimizes individual queries instead of managing resources throughout the entire system.

C. Adaptive and Self-Tuning Systems

The field of automated tuning has become essential to the development of DBMS systems as their size and complexity continue to expand. The traditional tuning methods, which rely on both manual setup and expert knowledge, have become insufficient to address the extensive parameter requirements and multiple system interactions in contemporary database management systems.

ML techniques have been successfully applied to various aspects of automated database tuning. An ML-based system handles automatic index selection for databases. The system uses historical query data to recommend indexes which will enhance query performance[28]. The method outperforms traditional heuristics, but it fails to consider how index selection interacts with other optimization methods used for query plan selection and buffer pool management[29][30].

The development of ML technology has improved the efficiency of buffer pool management, an essential optimization task. The iBTune system uses machine learning to adjust buffer pool memory settings in response to real-time changes in workload patterns. The method enhances memory usage and improves query execution results. However, it can only optimize one specific area of database performance tuning.

V. LITERATURE REVIEW

This section reviews modern DBMS advancements in storage, retrieval, and optimization, highlighting scalability and efficiency. Prior studies are comparatively summarized in Table II, outlining key approaches, performance metrics, and future research directions in contemporary database systems.

Tikotikar et al. (2026)modern applications use contemporary DBMS systems, which have evolved over time and gained new architectural features and operational capabilities. The research studies distributed systems together with cloud-based architectures, NoSQL databases, NewSQL databases and intelligent database systems to show how these systems scale and perform while maintaining data security and consistency. The research uses conceptual analysis and system evaluation to show how modern DBMS systems meet new computing requirements while they work to solve ongoing problems with data integrity, latency and ethical data governance[31].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, May 2026 Impact Factor: 8.2

E. Dritsas and M. Trigka, (2025) The Big Data era database ecosystem requires a complete and organized study which demonstrates its development throughout time. The study tracks the historical evolution, which began with traditional relational DBMS (RDBMS) systems and reached contemporary database systems, while demonstrating the driving forces behind changes and the resulting benefits and innovations. The research investigates database classification methods which use data models, deployment methods,scalability approaches and consistency frameworks to create an organized system that shows the different database capabilities[32].

P. Koukaras and C. Tjortjis,(2025) Examine the underlying data types and pattern outputs, as well as the various query types and popular retrieval techniques used in practical applications. Because each technique performs well with particular pattern representations and workload types, the study examines key indexing structures, including prefix trees, inverted indices, hash-based techniques, and bitmap-based methods. The study examines storage architectures, with an emphasis on redundancy-reduction techniques, format selection, and metadata annotation. The authors of the study assess several query optimization methods, including caching, ranking systems, and index-aware traversal[33].

G. S. Kopparthi (2024) examines PL/SQL and SQL work together to accomplish three key goals: improved performance, data integrity preservation, and simpler execution of challenging database operations. The research provides comprehensive information about PL/SQL, including its fundamental components—stored procedures, functions, triggers, and explicit/implicit cursors—which professionals illustrate using actual database management scenarios. These methods, which include indexing, partitioning, and data compression, are thoroughly examined in order to increase the effectiveness of data storage and retrieval in the future[34].

M. Besta et al., (2024) purpose is to identify and analyze the basic categories of these systems (such as document stores, tuple stores, native graph database systems, or object-oriented systems), the related graph models (such as Resource Description Framework or Labeled Property Graph), data organization strategies (such as storing graph data in indexing structures or splitting data into records), and various aspects of data distribution and query execution (such as support for sharding and Atomicity, Consistency, Isolation, Durability). Neo4j, OrientDB, and Virtuoso are among the fifty-one graph database systems that are described and contrasted. They describe graph database queries and their connections to related fields (dynamic graph algorithms, graph streaming, and NoSQL storage)[35].

G. R. Enjam, (2023) provided a thorough investigation of PostgreSQL optimization in the high-volume insurance workload, covering hardware usage, partitioning, query tuning, indexing techniques, connection pools, and caching. Additionally, it looks at secure backup and recovery methods such Point-In-Time Recovery (PITR), Write-Ahead Logging (WAL) archiving, encryption, logical and physical backups, and cloud-native backup methods[36].

H. Gadde, (2022) examines the possibilities and difficulties of integrating AI methods into conventional SQL query processing systems. They start by highlighting the main challenges, including the requirement for efficient integration frameworks, algorithm selection, and data quality. Three benefits of the system are identified by the research: enhanced user query processing, predictive analytics, and automated query optimization. In order to demonstrate the efficacy and limits of current AI models and frameworks that facilitate query processing, the researchers[37]

Table 2: Comparative Analysis of Database Storage, Retrieval, and Optimization Techniques in Modern Systems

Authors (Year)	Focus Area	Objectives	Key Performance Metrics	Approaches	Future Work
E. Dritsas and M. Trigka, (2025)	Database ecosystem in Big Data	To analyze evolution from RDBMS to modern databases and classify database systems	Throughput, latency, fault tolerance, cost efficiency	Comparative analysis of database models, deployment strategies, and consistency models	Explore emerging database technologies and hybrid models
P. Koukaras and C. Tjortjis,(2025)	Data retrieval and indexing	To study data types, query patterns, and	Query efficiency, indexing performance, storage	Use of prefix trees, inverted indices, hash-based and	Development of adaptive indexing and intelligent

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, May 2026 Impact Factor: 8.2

	techniques	retrieval operations	optimization	bitmap indexing, query optimization	query optimization methods
G. S. Kopparthi, (2024)	PL/SQL and database optimization	To enhance database performance and simplify operations using PL/SQL	Query execution efficiency, data consistency	Stored procedures, triggers, cursors, indexing, partitioning, compression	Integration with advanced analytics and automation tools
M. Besta et al., (2024)	Graph database systems	To analyse graph database categories, models, and query execution	Scalability, consistency (ACID), query performance	Study of RDF, LPG models, sharding, distributed query execution	Optimization of dynamic graph processing and real-time analytics
G. R. Enjam, (2023)	PostgreSQL optimization in high-volume systems	To optimize PostgreSQL performance in enterprise workloads	Query latency, throughput, recovery time (RTO)	Query tuning, indexing, partitioning, caching, WAL, PITR, backups	Further enhancement using cloud-native and AI-driven optimization
H. Gadde, (2022)	AI in SQL query processing	To explore integration of AI in traditional query systems	Query optimization accuracy, processing efficiency	AI models for automated optimization, predictive analytics	Address data quality issues and develop robust AI integration frameworks

VI. CONCLUSION AND FUTURE WORK

Modern database systems have developed through substantial changes to handle the increasing difficulties and larger data volumes which modern data-intensive applications bring. The study shows that advanced storage models, which use hybrid row-column storage methods to handle different data types, show better performance results for both reading and writing operations. NVM-aware systems, which constitute new storage architectures, deliver better performance through two improvements, which include faster access times and simpler data retrieval. The system achieves effective query processing through strong indexing methods combined with cache-aware optimizations which reduce processing costs. The system uses performance tuning methods, which include query plan optimization and, dynamic resource allocation and adaptive workload management to achieve system stability and responsiveness. Database management now uses automated systems through intelligent self-tuning mechanisms. The combination of these methods helps modern databases to achieve better scalability and reliability, which enables them to handle different analytical and transactional workloads in various computing environments.

Future research should focus on AI-driven autonomous database systems, adaptive hybrid storage models, and intelligent indexing techniques. The development of next-generation database systems will benefit from research that investigates real-time workload prediction combined with energy-efficient data management and improved distributed system integration.

REFERENCES

[1] C. Shekhar Pareek, “Chaos Testing: A Proactive Framework for System Resilience in Distributed Architectures,” Int. J. Sci. Res., vol. 13, no. 11, pp. 851–855, Nov. 2024, doi: 10.21275/SR241110081650.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, May 2026 Impact Factor: 8.2

[2] S. Huang, Y. Qin, X. Zhang, Y. Tu, Z. Li, and B. Cui, "Survey on performance optimization for database systems," Sci. China Inf. Sci., vol. 66, no. 2, p. 121102, Feb. 2023, doi: 10.1007/s11432-021-3578-6.
[3] H. B. Dama, "A Survey of MySQL Database Administration Techniques and Best Practices," ESP J. Eng. Technol. Adv., vol. 6, no. 1, pp. 89–98, February, 2026.
[4] C. A. Győrödi, D. V Dumşe-Burescu, R. Ş. Győrödi, D. R. Zmaranda, L. Bandici, and D. E. Popescu, "Performance Impact of Optimization Methods on MySQL Document-Based and Relational Databases," Appl. Sci., vol. 11, no. 15, p. 6794, Jul. 2021, doi: 10.3390/app11156794.
[5] A. Tomitan Bocces, A. Baldassin, D. Carlos Guimarães Pedronette, and A. Allberson Bruno de Oliveira Dantas, "Optimization in Information Retrieval: A Systematic Review of Techniques for Performance and Scalability," IEEE Access, vol. 14, pp. 2576–2591, 2026, doi: 10.1109/ACCESS.2025.3648134.
[6] Imran, F. Qayyum, D.-H. Kim, S.-J. Bong, S.-Y. Chi, and Y.-H. Choi, "A Survey of Datasets, Preprocessing, Modelling Mechanisms, and Simulation Tools Based on AI for Material Analysis and Discovery," Materials (Basel), vol. 15, no. 4, p. 1428, Feb. 2022, doi: 10.3390/ma15041428.
[7] S. Murumkar and S. Sen, "Managing IT Revolutions in Regulated Industries," Int. J. Artif. Intell. Data Sci. Mach. Learn., vol. 6, no. 2, pp. 190–194, May 2025, doi: 10.63282/3050-9262.IJAIDSML-V6I2P121.
[8] Y. Du, Z. Cai, and Z. Ding, "Query Optimization in Distributed Database Based on Improved Artificial Bee Colony Algorithm," Appl. Sci., vol. 14, no. 2, p. 846, Jan. 2024, doi: 10.3390/app14020846.
[9] S. Singamsetty, "AI-Enabled Data Stewardship Real-Time Alignment of Privacy and Storage Policies Across Global Systems using Deep CNN-RNN Techniques," in 2025 5th Asian Conference on Innovation in Technology (ASIANCON), IEEE, Aug. 2025, pp. 1–6. doi: 10.1109/ASIANCON66527.2025.11281010.
[10] S. Ramzan, I. S. Bajwa, R. Kazmi, and Amna, "Challenges in NoSQL-Based Distributed Data Storage: A Systematic Literature Review," Electronics, vol. 8, no. 5, Apr. 2019, doi: 10.3390/electronics8050488.
[11] M. Chanda, "A Low-Cost System for Acquiring Login/Logout Data for On-Ground Racks of in-Flight Entertainment Systems," California State University, 2016.
[12] F. E. R. Rabelo Ferreira and R. do Nascimento Fidalgo, "A Performance Analysis of Hybrid and Columnar Cloud Databases for Efficient Schema Design in Distributed Data Warehouse as a Service," Data, vol. 9, no. 8, p. 99, Aug. 2024, doi: 10.3390/data9080099.
[13] H. Ravilla, Predictive Analytics for Customer Churn in Salesforce Service Cloud. Springer Nature Switzerland, 2025.
[14] G. Zhu, J. Han, S. Lee, and Y. Son, "An Empirical Evaluation of NVM-Aware File Systems on Intel Optane DC Persistent Memory Modules," Electronics, vol. 10, no. 16, p. 1977, Aug. 2021, doi: 10.3390/electronics10161977.
[15] L. A. Fitriya, T. Purboyo, and A. Prasasti, "A review of data compression techniques," Int. J. Appl. Eng. Res., vol. 12, no. 19, pp. 8956–8963, 2017.
[16] A. Anup, R. Ashok, and P. Raundale, "Comparative Study of Data Compression Techniques," Int. J. Comput. Appl., vol. 178, no. 28, pp. 15–19, Jun. 2019, doi: 10.5120/ijca2019919104.
[17] V. B. Ramu, "Optimizing Database Performance: Strategies for Efficient Query Execution and Resource Utilization," Int. J. Comput. Trends Technol., vol. 71, no. 7, pp. 15–21, Jul. 2023, doi: 10.14445/22312803/IJCTT-V71I7P103.
[18] I. C. Saidu, M. Yusuf, F. C. Nemariyi, and A. C. George, "Indexing techniques and structured queries for relational database management systems," J. Niger. Soc. Phys. Sci., p. 2155, 2024.
[19] M. Abbasi, M. V Bernardo, P. Váz, J. Silva, and P. Martins, "Revisiting Database Indexing for Parallel and Accelerated Computing: A Comprehensive Study and Novel Approaches," Information, vol. 15, no. 8, p. 429, Jul. 2024, doi: 10.3390/info15080429.
[20] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang, "In-Memory Big Data Management and Processing: A Survey," IEEE Trans. Knowl. Data Eng., vol. 27, no. 7, pp. 1920–1948, Jul. 2015, doi: 10.1109/TKDE.2015.2427795.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, May 2026 Impact Factor: 8.2

[21] A. K. Padhy, T. Pravinbhai Patel, V. Soni, S. Shivam, G. B. Thokala, and B. Vulugundam, "Machine Learning-Based Fault Prediction in Large- Scale Distributed Systems," in 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC), IEEE, Feb. 2026, pp. 1–6. doi: 10.1109/ICAIC67076.2026.11395778.
[22] S. R. Sirikonda, S. Bhat, R. Jain, and V. Katoch, "Failure Isolation And Blast Radius Control In Large-Scale Distributed Systems," Int. J. Adv. SIGNAL IMAGE Sci., vol. 12, no. 3, pp. 2003–2022, March, Mar. 2026, doi: 10.29284/mkwrh621.
[23] M. Patel and U. Korat, "Swarm Optimization Algorithm-Enhanced Clustering Techniques for Reliable Wireless Sensor Networks Communication," in 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC), IEEE, Feb. 2026, pp. 1–6. doi: 10.1109/ICAIC67076.2026.11395724.
[24] Z. Zhou, D. Zhao, G. Hancke, L. Shu, and Y. Sun, "Cache-Aware Query Optimization in Multiapplication Sharing Wireless Sensor Networks," IEEE Trans. Syst. Man, Cybern. Syst., vol. 48, no. 3, pp. 401–417, 2018, doi: 10.1109/TSMC.2016.2598398.
[25] S. Gaddam, "Database Performance Optimization: Strategies that Scale," Int. J. Comput. Eng., vol. 7, no. 11, pp. 1–12, 2025, doi: 10.47941/ijce. 2967.
[26] N. Hartono and Z. Masyhur, "Optimizing transaction data performance in database management systems," MATRIX J. Manaj. Teknol. dan Inform., vol. 13, no. 2, pp. 106–114, Jul. 2023, doi: 10.31940/matrix.v13i2.106-114.
[27] S. Xing, Y. Wang, and W. Liu, "Self-Adapting CPU Scheduling for Mixed Database Workloads via Hierarchical Deep Reinforcement Learning," Symmetry (Basel)., vol. 17, no. 7, p. 1109, Jul. 2025, doi: 10.3390/sym17071109.
[28] M. Abbasi, M. V Bernardo, P. Váz, J. Silva, and P. Martins, "Adaptive and Scalable Database Management with Machine Learning Integration: A PostgreSQL Case Study," Information, vol. 15, no. 9, p. 574, Sep. 2024, doi: 10.3390/info15090574.
[29] A. R. Bairi, V. P. Rambabu, and B. Yakkanti, "AI-Enhanced Test Prioritization in Continuous Integration for SaaS Platforms," Am. J. Auton. Syst. Robot. Eng., vol. 2, pp. 110–145, Aug. 2022.
[30] R. Palwe, "Onboarding for AI features: Reducing friction at the first use," Int. J. Comput. Artif. Intell., vol. 6, no. 2, pp. 393–400, Jul. 2025, doi: 10.33545/27076571.2025.v6.i2e.227.
[31] A. Tikotikar, K. Sakibaev, U. Srilakshmi, and R. K. P, "Innovative Study On Database Management Systems In Modern Applications," Int. J. Adv. Signal Image Sci., vol. 12, no. 2, pp. 199–207, February, 2026, doi: 10.29284/n0s4rk20.
[32] E. Dritsas and M. Trigka, "Database Systems in the Big Data Era: Architectures, Performance, and Open Challenges," IEEE Access, vol. 13, pp. 95068–95084, 2025, doi: 10.1109/ACCESS.2025.3572059.
[33] P. Koukaras and C. Tjortjis, "Data Organisation for Efficient Pattern Retrieval: Indexing, Storage, and Access Structures," Big Data Cogn. Comput., vol. 9, no. 10, pp. 1–36, Oct. 2025, doi: 10.3390/bdcc9100258.
[34] G. S. Kopparthi, "Data Storage and Retrieval with PL/SQL," J. Informatics Educ. Res., vol. 4, no. 2, pp. 3635–3643, 2024.
[35] M. Besta et al., "Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries," ACM Comput. Surv., vol. 56, no. 2, pp. 1–40, Feb. 2024, doi: 10.1145/3604932.
[36] G. R. Enjam, "Optimizing PostgreSQL for High-Volume Insurance Transactions & Secure Backup and Restore Strategies for Databases," Int. J. Emerg. Trends Comput. Sci. Inf. Technol., vol. 4, no. 1, pp. 104–111, 2023, doi: 10.63282/3050-9246.IJETCSIT-V4I1P112.
[37] H. Gadde, "Integrating AI into SQL Query Processing: Challenges and Opportunities," Int. J. Adv. Eng. Technol. Innov., vol. 01, no. 03, p. 3, 2022.