INTERNATIONAL JOURNAL OF MODERN COMPUTER SCIENCE AND IT INNOVATIONS (IJMCSIT)

eISSN: 3087-4289

Volume. 02, Issue. 02, pp. 10-18, February 2025"

CONVERGENT DATABASE ARCHITECTURES: MULTI-MODEL DESIGN AND QUERY OPTIMIZATION IN NEWSQL SYSTEMS

Dr. Arjun S. Patel
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Prof. Elena D. Petrovna
Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Russia

Article received: 24/12/2024, Article Revised: 19/01/2025, Article Accepted: 21/02/2025
DOI: https://doi.org/10.55640/ijmcsit-v02i02-02

ABSTRACT

The contemporary data landscape, characterized by diverse data types and escalating volumes, has driven a significant evolution in database management systems. Traditional Relational Database Management Systems (RDBMS) often struggle with the scalability and flexibility required for modern applications, while NoSQL databases, though scalable, frequently compromise on transactional consistency. NewSQL systems have emerged as a hybrid solution, aiming to combine the ACID guarantees of RDBMS with the horizontal scalability of NoSQL. This report explores the critical need for multi-model capabilities within NewSQL DBMSs to efficiently manage heterogeneous data adhering to various data models—including relational, document, graph, and key-value—within a single, unified database. It delves into the architectural considerations for supporting such diversity, examining storage paradigms like row-store, column-store, and hybrid approaches, alongside data distribution strategies such as sharding and partitioning. Furthermore, the paper investigates advancements in multi-model query languages, particularly SQL++ and extensions in SQL:2016, and discusses query optimization techniques essential for handling complex, hybrid workloads. Finally, it addresses the performance evaluation of these systems in big data environments, highlights current limitations, and outlines future research directions for achieving truly versatile and high-performing data management solutions.

Keywords: Convergent database architectures, multi-model databases, NewSQL systems, query optimization, hybrid data models, scalable data management, relational and NoSQL integration, unified query processing, database performance, modern data architectures.

INTRODUCTION

A. The Evolving Data Landscape and Database Paradigms

The digital age has ushered in an era of unprecedented data growth, characterized not only by sheer volume but also by immense variety. Enterprises today grapple with structured, semi-structured, and unstructured data, demanding database solutions that can handle this heterogeneity while ensuring high performance and scalability.[38] Traditional Relational Database Management Systems (RDBMS), which have long served as the backbone of data management, excel at handling structured data with strong transactional integrity (ACID properties: Atomicity, Consistency, Isolation, Durability).[39, 40] However, their inherent design, often optimized for vertical scaling (upgrading existing server hardware), presents significant limitations when faced with the demands of massive datasets and high transaction volumes.[41, 42] These limitations include scalability challenges, fixed schemas that are difficult to adapt, performance bottlenecks with complex joins, and complexities in maintaining high availability in distributed environments.[41, 42]

In response to these challenges, NoSQL (Not only SQL)

databases emerged as a viable alternative, prioritizing horizontal scalability and flexibility for unstructured and semi-structured data.[40, 43, 44, 45] NoSQL databases, which include document-oriented (e.g., MongoDB), key-value (e.g., Redis), and column-family (e.g., Cassandra) stores, can distribute data across multiple nodes, handling increased workloads and massive data volumes with relative ease.[43, 44] However, this flexibility often comes at a cost: many NoSQL systems relax ACID compliance in favor of availability and partition tolerance (as per the CAP theorem), leading to eventual consistency models.[43, 45, 46] This trade-off makes NoSQL unsuitable for applications requiring strict transactional integrity, such as financial systems or healthcare applications, where real-time consistency is critical.[43, 46]

B. NewSQL: Bridging the Gap

To bridge the functional gap between the transactional consistency of SQL and the horizontal scalability of NoSQL, a new class of database systems, termed NewSQL, emerged.[38, 39, 40, 42, 43, 44, 45, 47] NewSQL databases are engineered to combine the best features of both worlds: they retain the familiar SQL interface and full ACID compliance of traditional RDBMS while offering the distributed, scalable performance typically associated with NoSQL systems.[38, 39, 40, 42, 43, 45, 47]

Key characteristics of NewSQL systems include:

- SQL Compatibility: They support the widely used SQL query language, minimizing re-learning and migration overheads for developers.[38, 39, 45]

- ACID Compliance: They maintain full ACID guarantees, even in distributed environments, ensuring reliable transaction processing.[38, 39, 40, 43, 48]

- Distributed, Shared-Nothing Architecture: NewSQL databases typically adopt a distributed system architecture, partitioning data across multiple nodes to improve scalability and availability, and ensuring no single point of contention.[39, 40, 43, 48]

- Horizontal Scalability: They are designed to scale out seamlessly by adding more servers, distributing workloads and data across multiple machines.[38, 39, 40, 43, 48]

- High Availability and Fault Tolerance: Mechanisms like replication and automatic failover ensure continuous data accessibility even in the event of hardware or network failures.[39, 45]

C. The Imperative for Multi-Model Capabilities in NewSQL

The increasing diversity of data types in modern enterprises necessitates database systems capable of handling multiple data models simultaneously. NewSQL DBMSs are evolving to provide robust multi-model support, integrating various data paradigms within a unified system.[49, 50] This capability allows organizations to store and manage heterogeneous collections of values—structured, semi-structured, and unstructured—in a single database, optimizing for hybrid workloads and ensuring continuous data availability.[49, 50]

Key aspects of multi-model support in NewSQL include:

- Relational Data: As their foundation, NewSQL systems inherently support the relational model, providing strong consistency and integrity guarantees for structured data.[49, 50]

- Document Data (JSON, XML): NewSQL databases are increasingly incorporating native support for semi-structured data formats like JSON and XML. This allows for flexible schemas, where new fields can be added without costly migrations, similar to NoSQL document stores.[38, 54]

- Graph-Oriented Data: Some NewSQL systems are extending their capabilities to handle graph data, enabling the storage and querying of complex relationships, which is crucial for applications like social networks or recommendation engines.[49, 50]

- Key-Value and Column-Family Data: While often associated with NoSQL, the underlying storage layers of some NewSQL databases, like TiKV in TiDB, can function as key-value stores, providing high performance for specific access patterns.[53, 54]

- Unified Design Methodology: A comprehensive methodological approach for designing multi-model databases in NewSQL involves identifying and defining heterogeneous collections of values and integrating the design processes of relational and NoSQL databases across multiple levels of abstraction.[49, 50] This allows designers to leverage the Nested Relational Model as a "Pivot Model" to automatically generate external schemas for virtual NoSQL databases, enabling users to interact with the data store as if it were a native NoSQL database.[49, 50]

D. Article Structure

The remainder of this article is structured as follows: Section II provides an overview of foundational database concepts and architectural elements relevant to multi-model NewSQL systems. Section III delves into multi-model query processing and optimization techniques. Section IV discusses performance evaluation, current limitations, and future research directions. Finally,

Section V concludes the paper, summarizing the key insights and implications.

II. Foundational Concepts and Architectural Elements

A. Evolution of Database Management Systems

The journey of Database Management Systems (DBMS) has been a continuous adaptation to evolving data needs and technological advancements.[44] It began with Hierarchical Databases in the 1960s, organizing data in a tree-like structure with parent-child relationships, suitable for specific applications but lacking flexibility for complex interconnections.[44] The 1970s marked a significant shift with Relational Database Management Systems (RDBMS), which store data in tables with well-defined schemas and introduced SQL (Structured Query Language) as the standard for querying and managing data.[44] Popular RDBMS like Oracle, MySQL, and SQL Server emerged during this era.[44]

As applications became more object-oriented, Object-Oriented Database Management Systems (OODBMS) appeared, storing data as objects, but their adoption was limited by the dominance of RDBMS.[44] The 21st century witnessed the rise of NoSQL databases, designed to handle unstructured and semi-structured data, making them ideal for big data and web applications. NoSQL encompasses various types, including document-oriented (e.g., MongoDB), key-value (e.g., Redis), and column-family (e.g., Cassandra) stores.[44] Subsequently, NewSQL databases emerged, combining the scalability of NoSQL with the ACID compliance of RDBMS, targeting high-performance, transactional applications.[44] More specialized categories like Graph Databases, excelling at handling data with complex relationships (e.g., social networks), and Time-Series Databases, optimized for time-stamped data (e.g., IoT), further illustrate the dynamic nature of DBMS evolution.[44]

B. Limitations of Traditional RDBMS

Despite their long-standing role as the bedrock of data management, traditional RDBMS face several inherent limitations that challenge their effectiveness in modern, data-intensive environments:

- Scalability Challenges: RDBMS typically scale vertically, meaning that to handle increased workloads, hardware upgrades (CPU, memory) on a single server are required.[42] This vertical scaling becomes prohibitively expensive and eventually hits physical limits when dealing with extremely large datasets or high transaction volumes.[41] Horizontal scaling (adding more servers) is often complicated and requires significant effort to maintain performance and consistency.[41]

- Fixed Schema Rigidity: Traditional RDBMS enforce a fixed schema, where the structure of the database must be defined in advance.[41, 42] Modifying this schema, such as adding new columns or changing relationships, can be a cumbersome process, often requiring downtime or significant changes to existing applications. This lack of flexibility is a major drawback in fast-evolving environments that demand agile data structures.[41]

- Performance Bottlenecks with Complex Joins: In relational databases, operations involving complex joins across multiple tables can lead to severe performance degradation.[41, 42] As the number of tables and the complexity of relationships increase, query execution time rises significantly, especially with large datasets.[41]

- Concurrency Issues: Managing data concurrency when multiple users or applications simultaneously access and modify the database can be challenging.[41] Locking mechanisms, while ensuring data consistency, can lead to performance degradation under heavy load.[41]

- Limited Support for Unstructured Data: RDBMS are primarily designed for structured data and are not well-suited for handling unstructured or semi-structured data like images, documents, or JSON objects.[41] While some RDBMS offer specialized data types for such content, they are generally less efficient than NoSQL databases for these workloads.[41]

C. Challenges of NoSQL Databases

While NoSQL databases offered a compelling solution to the scalability limitations of RDBMS, they introduced their own set of challenges, particularly concerning data consistency and reliability:

- Relaxed Consistency Models: Most NoSQL databases prioritize availability and partition tolerance over strong consistency, adhering to the CAP theorem.[43] This often results in "eventual consistency," where data might not be immediately synchronized across all nodes.[43] While beneficial for high availability and low-latency responses, this model is unsuitable for applications that require strict transactional integrity, such as financial transactions or healthcare applications.[43, 46]

- Complexity in Maintaining Data Consistency: The eventual consistency model of NoSQL can create significant challenges for applications that demand real-time consistency.[43] Developers often need to implement additional application-level logic to handle data synchronization and resolve conflicts, which complicates development for critical systems.[43]

- Lack of Standardized Query Language: Unlike SQL for RDBMS, NoSQL databases typically lack a single, universally adopted query language. Each NoSQL database often has its own unique API or query language, increasing the learning curve and hindering interoperability across different NoSQL solutions.

- Limited Support for Complex Queries: While NoSQL excels at simple key-value lookups or document retrievals, performing complex analytical queries, multi-table joins, or aggregations can be challenging or inefficient compared to SQL.[52]

D. Advantages of NewSQL DBMSs

NewSQL databases represent a strategic evolution, combining the strengths of traditional RDBMS and NoSQL systems to address the demands of modern applications. Their key advantages include:

- SQL Compatibility with Scalability: NewSQL systems are designed to retain full SQL support while overcoming the performance and scalability limitations of conventional SQL databases.[38, 39, 40] They introduce distributed, scalable architectures that allow for horizontal scaling, efficiently distributing workloads across multiple nodes without sacrificing SQL functionality.[39]

- Full ACID Compliance: Unlike many NoSQL databases, NewSQL systems maintain full ACID guarantees (Atomicity, Consistency, Isolation, Durability) even in distributed environments.[38, 39, 40, 43, 48] This ensures reliable transaction processing and strong data consistency, critical for applications like financial systems and e-commerce platforms.[45]

- Distributed, Shared-Nothing Architecture: NewSQL databases typically adopt a shared-nothing architecture, where each node operates independently without sharing memory or disk with other nodes.[39] This design eliminates single points of contention, making them ideal for handling big data workloads and ensuring high availability.[39] Data is partitioned across many servers, providing fault tolerance and seamless scale-out capabilities.[39]

- High Availability and Fault Tolerance: Many NewSQL databases are built with high availability and fault tolerance in mind, utilizing mechanisms such as replication and automatic failover to ensure continuous data accessibility.[39, 48] Systems like Google Spanner exemplify this by providing geographically distributed databases that ensure data is always available, even across data centers.[39, 48]

- Hybrid Workload Optimization (HTAP): Some NewSQL systems are designed to handle both Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) workloads efficiently within a single database, a concept known as Hybrid Transactional/Analytical Processing (HTAP).[49, 50, 51, 53] This eliminates the need for separate databases for transactional and analytical tasks, simplifying architecture and reducing data movement.

E. Multi-Model Data Support in NewSQL

The increasing diversity of data types in modern enterprises necessitates database systems capable of handling multiple data models simultaneously. NewSQL DBMSs are evolving to provide robust multi-model support, integrating various data paradigms within a unified system.[38, 49, 50, 54] This capability allows organizations to store and manage heterogeneous collections of values—structured, semi-structured, and unstructured—in a single database, optimizing for hybrid workloads and ensuring continuous data availability.[49, 50]

Key aspects of multi-model support in NewSQL include:

- Relational Data: As their foundation, NewSQL systems inherently support the relational model, providing strong consistency and integrity guarantees for structured data.[49, 50]

- Document Data (JSON, XML): NewSQL databases are increasingly incorporating native support for semi-structured data formats like JSON and XML. This allows for flexible schemas, where new fields can be added without costly migrations, similar to NoSQL document stores.[38, 54]

- Graph-Oriented Data: Some NewSQL systems are extending their capabilities to handle graph data, enabling the storage and querying of complex relationships, which is crucial for applications like social networks or recommendation engines.[49, 50]

- Key-Value and Column-Family Data: While often associated with NoSQL, the underlying storage layers of some NewSQL databases, like TiKV in TiDB, can function as key-value stores, providing high performance for specific access patterns.[53, 54]

- Unified Design Methodology: A comprehensive methodological approach for designing multi-model databases in NewSQL involves identifying and defining heterogeneous collections of values and integrating the design processes of relational and NoSQL databases across multiple levels of abstraction.[49, 50] This allows designers to leverage the Nested Relational Model as a "Pivot Model" to automatically generate external schemas for virtual NoSQL databases, enabling users to interact with the data store as if it were a native NoSQL database.[49, 50]

F. Storage Architectures: Row-Store, Column-Store, and Hybrid

The internal storage architecture significantly impacts a NewSQL DBMS's performance for different workloads. The primary layouts include row-store, column-store, and hybrid-store.[55]

- Row-Store: In a row-store, each entity (or record) is assigned a dedicated row, and all attributes of a tuple are stored contiguously.[55] This layout is highly efficient for Online Transaction Processing (OLTP) workloads, which typically involve interactive transactions like retrieving all attributes from a single entity or adding new entities, as the entire row can be written in a single operation.[55] However, for analytical queries that only need a few columns, row-stores can be inefficient due to unnecessary data transfer.[55]

- Column-Store: Column-oriented database systems store tables of tuple attributes contiguously by column.[55] This design is highly efficient for Online Analytical Processing (OLAP) workloads, such as data warehousing, decision support, and business intelligence applications, where queries often involve aggregating data across many rows but only a few columns.[55, 56] Column-stores can perform orders of magnitude better than row-stores on analytical workloads.[55] However, they are less efficient for write-intensive operations or when retrieving entire rows.[55]

- Hybrid-Store: Recognizing the trade-offs, hybrid-store architectures combine both row-store and column-store characteristics.[55, 56] This approach allows for storing insert and update-intensive data in row-store components while analytical and historical data reside in column-store components.[55] For example, StarRocks supports hybrid row-column storage, where data is stored in both fashions, enabling high-concurrency, low-latency point queries and partial column updates, while still delivering efficient analytical capabilities.[56] This hybrid approach aims to optimize for diverse workloads within a single system.

G. Data Distribution Strategies: Sharding and Partitioning

To achieve horizontal scalability and high availability, NewSQL DBMSs employ sophisticated data distribution strategies, primarily sharding and partitioning.[40, 43, 48, 57] While often used interchangeably, they refer to distinct levels of data organization:

- Partitioning: This typically refers to dividing a single database (and often a single table) into multiple segments called partitions within the same database server or cluster node.[57] All partitions remain part of one overall database instance, collectively representing one logical table split based on a key like date or an ID range.[57] Partitioning improves manageability and query performance on very large tables by allowing queries to scan only relevant segments (partition pruning).[57] However, it does not increase the total processing power or storage beyond what one server provides.[57]

- Sharding (Horizontal Partitioning): Sharding involves distributing portions of the data across multiple separate database servers or instances, with each server holding a subset of the data known as a shard.[40, 48, 57] The primary motivation for sharding is to distribute the load, allowing the overall system to handle more users or transactions than a single server could.[57] Each shard might have the same schema but contains only the rows pertaining to a certain portion of the data (e.g., a subset of customers based on a shard key).[57] NewSQL systems often integrate middleware for automatic and transparent sharding, fragmenting tables and indexes horizontally and distributing them across geographically dispersed servers.[40, 48, 50] While powerful for scaling, sharding introduces complexities such as increased application complexity, potential for uneven data distribution (hotspots), and difficulties with cross-shard operations.[57]

III. Multi-Model Query Processing and Optimization

A. Unified Query Languages for Multi-Model Data

The ability to manage diverse data models within a single NewSQL DBMS necessitates query languages that can seamlessly interact with these varied structures. Traditional SQL, designed for strictly relational data, requires extensions to handle semi-structured, nested, and graph-oriented data effectively.

- SQL++: This is a prominent SQL extension designed to relax SQL's strictness regarding both object structure (from flat to nested) and schema (from mandatory to optional).[58] SQL++ views relational data as a subset of a more flexible object model and naturally supports collections of document data (e.g., JSON).[58] It provides syntax and semantics to comprehensively access, query, and construct nested data while naturally composing with standard SQL features.[58] The goal of SQL++ is to broaden the scope of SQL itself, making it backward-compatible while handling schema-optional data.[58]

- SQL:2016 Extensions: The SQL:2016 standard introduced significant enhancements to support JSON data, including functions for creating, querying, and manipulating JSON documents within a relational database context.[1] This allows SQL to interact more directly with semi-structured data, blurring the lines between traditional relational and document models. Furthermore, the upcoming SQL/PGQ (Property Graph Queries) standard (ISO/IEC 9075-16:2023) aims to

standardize graph query capabilities within SQL, enabling direct querying of graph data.[2] These extensions allow NewSQL systems to provide a unified query interface that can span multiple data models, reducing the need for developers to learn different query languages for different data types.[38, 52]

B. Query Optimization Techniques

Efficient query processing is paramount in multi-model NewSQL DBMSs, especially when dealing with large, heterogeneous datasets and hybrid workloads. Query optimization techniques are crucial for ensuring high performance and scalability.[50, 52]

- Indexing: Creating indexes on frequently queried fields is a fundamental technique to improve query performance.[52] NewSQL systems support various indexing strategies, including traditional B-trees [3] and specialized indexes for different data models (e.g., inverted indexes for document data or graph indexes for graph traversals). Join indexes [4] and materialized views [5] can further accelerate complex queries and analytical workloads by pre-computing and storing results.

- Caching and Buffering: Implementing caching mechanisms (e.g., Redis or Memcached) helps reduce the load on the database by storing frequently accessed data in memory, thereby improving data throughput and performance.[40, 52] Optimizing cache invalidation strategies is essential to ensure data consistency and freshness.[52]

- Query Rewriting and Simplification: Breaking down complex queries into simpler, more efficient sub-queries can significantly improve execution time.[52] Query planners and cost optimizers within NewSQL systems analyze query structures and data distribution to determine the most efficient execution plan, often leveraging distributed execution capabilities.[53]

- Distributed Query Processing: NewSQL databases are designed for distributed query execution, where queries are fragmented and processed in parallel across multiple shards or nodes.[40, 48, 57] This involves routing queries to the correct shards based on shard keys and handling the aggregation of results from multiple nodes.[57] Techniques like partition pruning, where the database "cuts away" partitions not needed for a given query, further reduce I/O and improve efficiency.[57]

- Storage Model Optimization: Choosing a data storage format optimized for specific use cases, such as column-store for analytical queries or row-store for transactional workloads, is critical.[52] Hybrid storage architectures allow NewSQL to dynamically adapt to different query patterns.[55, 56]

C. Unified Data Access Layer

A key architectural goal of multi-model NewSQL DBMSs is to provide a unified data access layer. This layer abstracts away the complexities of underlying heterogeneous data models and distributed storage, presenting a consistent interface to applications and users.[47, 53]

- Simplified Development: By offering a single query language (like an extended SQL) and a consistent API, developers can interact with diverse data types without needing to learn multiple database-specific languages or manage complex data transformations.[38, 52] This significantly reduces development time and effort.

- Data Silo Elimination: A unified access layer helps eliminate data silos, where different data types are isolated in separate, specialized databases.[47] This integration fosters a more holistic view of enterprise data and enables cross-model queries that were previously difficult or impossible.

- Hybrid Workload Support: The unified layer facilitates Hybrid Transactional/Analytical Processing (HTAP) by allowing both OLTP and OLAP queries to run efficiently against the same data store, regardless of its underlying multi-model structure.[53] This is achieved through sophisticated query routing and optimization that leverages the appropriate storage and processing mechanisms for each query.

- Interoperability: The goal is to provide a platform where various workloads, such as data warehousing and machine learning, can be supported on a unified platform.[47] This includes the ability to query SQL, NoSQL, and NewSQL data from a single platform, enhancing overall system versatility.[38]

IV. Performance Evaluation and Future Outlook

A. Performance Evaluation for Big Data Workloads

Evaluating the performance of multi-model NewSQL DBMSs for big data workloads is crucial to validate their effectiveness in real-world scenarios. These evaluations typically focus on their ability to handle hybrid transactional and analytical processing (HTAP) efficiently, as well as their scalability under increasing data volumes and user loads.[49, 50, 51]

- Benchmarking: Performance is often assessed through benchmarks that compare NewSQL databases against traditional RDBMS (e.g., MySQL) in cloud environments.[59] These comparisons measure response times under various workload configurations, including different mixes of read/write operations and query complexities.[59]

- Scalability Metrics: Key metrics include

throughput (transactions per second), latency (response time for queries), and the ability to maintain performance as the number of nodes or data volume increases.[39, 52] NewSQL systems are expected to demonstrate linear scalability, where performance increases proportionally with added resources.

- Hybrid Workload Performance: Evaluations specifically test the system's capability to handle concurrent transactional and analytical queries without significant degradation in either. This is a core promise of HTAP-enabled NewSQL databases.[49, 50, 51]

B. Limitations and Open Issues

Despite their significant advancements, multi-model NewSQL DBMSs still face several limitations and open issues that require ongoing research and development:

- Increased Design Complexity: The integration of multiple data models and distributed architectures inherently increases the complexity of database design.[50] Designers must make intricate choices regarding storage models, access paths, and sharding strategies to optimize for diverse workloads, which can be challenging.[49, 50]

- Data Distribution Challenges: While sharding enables horizontal scalability, it introduces complexities such as potential for uneven data distribution (hotspots) and difficulties with cross-shard operations.[43, 48, 57] Ensuring consistent data synchronization across multiple servers in a distributed environment is difficult due to network latency, server failures, and replication delays, potentially leading to temporary inconsistencies.[48]

- Learning Curve and Compatibility Issues: The relative novelty of NewSQL technologies means that each new implementation may require a non-overlapping learning curve for developers and administrators.[45] Furthermore, compatibility issues with existing data models and schemas can lead to considerable migration efforts for legacy applications.[45]

- Usability Concerns: While NewSQL systems aim for flexible installation and maintenance, the complexity of distributed systems can make query formulation difficult for end-users, potentially affecting user experience.[60] Improving compiler error messages and providing constructive hints for error resolution are areas for improvement.[60]

- Partial Access to Traditional Features: Some NewSQL solutions may offer only partial access to the rich features available in traditional RDBMS, which could be a disadvantage for certain use cases.[54]

C. Future Research Directions

The continuous evolution of data management presents several promising avenues for future research in multi-model NewSQL DBMSs:

- Enhanced Usability and Developer Experience: Future research should focus on simplifying the design, deployment, and management of multi-model NewSQL systems. This includes developing more intuitive tools, improving error messaging, and providing better guidance for optimizing complex, hybrid workloads.[60]

- Adaptive Query Optimization: Developing more intelligent and adaptive query optimizers that can dynamically adjust execution plans based on real-time workload characteristics and data distribution patterns is crucial. This could involve leveraging AI-driven query optimization techniques.[38]

- Automated Data Model Selection and Mapping: Research into automated mechanisms for identifying the most suitable data model for specific data types and workloads, and for seamlessly mapping between different models, would further enhance multi-model capabilities.

- Advanced Consistency Models: Exploring more nuanced consistency models beyond strict ACID or eventual consistency that can be dynamically tuned to specific application requirements, offering a balance between performance, availability, and consistency.

- Serverless and Cloud-Native Architectures: Further integration with serverless computing and cloud-native architectures can enhance the elasticity, cost-efficiency, and operational simplicity of NewSQL deployments.[38]

- Cross-Model Query Language Standardization: Continued efforts to standardize query languages like SQL/PGQ [2] and further extend SQL++ will be vital for broader adoption and interoperability across multi-model NewSQL systems.

V. CONCLUSION

The advent of NewSQL DBMSs marks a pivotal moment in database technology, effectively bridging the historical divide between the strong consistency of traditional RDBMS and the horizontal scalability of NoSQL databases. This report has underscored the increasing necessity for multi-model capabilities within these hybrid systems, enabling enterprises to manage the burgeoning volume and diversity of data—from structured relational tables to flexible JSON documents and intricate graph structures—within a single, unified platform.

We have explored the foundational architectural elements that underpin multi-model NewSQL, including adaptable storage paradigms (row-store, column-store,

and hybrid) and sophisticated data distribution strategies (sharding and partitioning). The evolution of query languages, particularly SQL++ and the extensions within SQL:2016, is critical for providing a unified interface to interact with these diverse data models, complemented by advanced optimization techniques to ensure high performance for complex, hybrid workloads.

While NewSQL offers compelling advantages for big data environments, challenges related to design complexity, data distribution, and the learning curve for new implementations persist. Addressing these limitations through continued research into enhanced usability, adaptive query optimization, and further standardization will be crucial. Ultimately, the ongoing development of multi-model NewSQL DBMSs promises to deliver increasingly versatile, efficient, and resilient data management solutions, capable of meeting the dynamic demands of the modern digital landscape.

REFERENCES

Guo Q, Zhang C, Zhang S, Lu J. Multi-model query languages: taming the variety of big data. Distributed and Parallel Databases, 2024, 42: 31–71.

Lu J, Holubova I. Multi-model Databases: A New Journey to Handle the Variety of Data. ACM Computing Surveys, 2019, Vol. 0, No. 0.

Michels J, Hare K, Kulkarni K, Zuzarte C, Liu Z H, Hammerschmidt B, Zemke F. The New and Improved SQL: 2016 Standard. SIGMOD Record, June 2018, Vol. 47, No. 2.

Ong K W, Papakonstantinou Y, Vernoux R. The SQL++ Unifying Semi-structured Query Language, and an Expressiveness Benchmark of SQL-on-Hadoop, NoSQL and NewSQL Databases. arXiv: 1405.3631, Dec. 2015.

Krishnappa M S, Harve B M, Jayaram V, Nagpal A, Ganeeb K K, Ingole B S. ORACLE 19C Sharding: A Comprehensive Guide to Modern Data Distribution. IJCET, Sep-Oct 2024, Volume 15, Issue 5.

Akinola S. Trends in Open Source RDBMS: Performance, Scalability and Security Insights. Journal of Research in Science and Engineering (JRSE), July 2024, Volume-6, Issue-7.

Miryala N K. Emerging Trends and Challenges in Modern Database Technologies: A Comprehensive Analysis. International Journal of Science and Research (IJSR), November 2024, Volume 13 Issue 11.

Muhammed A, Abdullah Z H, Ismail W, Aldailamy A Y, Radman A, Hendradi R, Afandi R R. A Survey of NewSQL DBMSs focusing on Taxonomy, Comparison and Open Issues. IJCSMC, December 2021, Volume 11, Issue 4.

Khasawneh T N, Alsahlee M, Safieh A. SQL, NewSQL, and NOSQL Databases: A Comparative Survey. In 2020 11th International Conference on Information and Communication Systems (ICICS).

Murazzo M, Gómez P, Rodríguez N, Medel D. Database NewSQL Performance Evaluation for Big Data in the Public Cloud. In Book Communications in Computer and Information Science ((CCIS, volume 1050)), Naiouf, M., Chichizola, F., Rucci, E. (eds) Cloud Computing and Big Data. JCC&BD 2019.

Pavlo A, Aslett M. What's Really New with NewSQL? SIGMOD Record, June 2016, Vol. 45, No. 2.

Maia F C M B O. Sharding by Hash Partitioning A database scalability pattern to achieve evenly sharded database clusters. 17th ICEIS At: Barcelona, Spain, April 2015.

Moniruzzaman A. NewSQL: Towards Next-Generation Scalable RDBMS for Online Transaction Processing (OLTP) for Big Data Management. arXiv preprint arXiv: 1411.7343, 2014.

Tankoano J. Providing in RDBMSs the flexibility to Work with Various Non-Relational Data Models. Global Journal of Computer Science and Technology: H Information & Technology, 2023, Volume 23 Issue 2 Version 1.0. DOI:(https://doi.org/10.34257/GJCSTHVOL23IS2PG1.

Chen P. The entity-relationship model - toward a unified view of data. ACM TODS, March 1976, Volume 1, Issue 1, pp 9–36.

Object Modeling Group. Unified Modeling Language Specification. October 2012, Version 2.5.

Valduriez P, Khoshajian S, Copeland G. Implementation Techniques of Complex Objects. 12th Int. Conference on Very Large Data Bases - Kyoto, August 1986.

Lahiri T, Abiteboul S, Widom J. Ozone: Integrating Structured and Semistructured Data. 7th Int. Workshop on Database Programming Languages: Research Issues in Structured and Semi-structured Database Programming, December 1999.

Natti, M. (2023). Reducing PostgreSQL read and write latencies through optimized fillfactor and HOT percentages for high-update applications. International Journal of Science and Research Archive, 9(2), 1059–1062. https://doi.org/10.30574/ijsra.2023.9.2.0657

Scholl M H. Extensions to the Relational Data Model. Available from:(https://www.researchgate.net/publication/238121

7_Extensions_to_the_Relational_Data_Model) (accessed 29 March 2025).

Tankoano J. Modèle relationnel imbriqué. In SGBD relationnels – Tome 2, Vers les Bases de données Réparties, Objet, Objet-relationnelles, XML, … Available from:(https://www.researchgate.net/publication/366548683_SGBD_relationnels_-_Tome_2_Vers_les_Bases_de_donnees_Reparties_Objet_Objet-relationnelles_XML) (accessed 29 March 2025).

Ozsoyoglu Z M, Yuan L Y. On the normalization in Nested Relational Databases. LNCS, 1989, volume 361.

Abadi D J, Madden S R, Hachem N. Column-Stores vs. Row-Stores: How Different Are They Really? SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.

ORACLE. Oracle Database SQL Language. Reference 23ai, F47038-19, November 2024.

Comer D. The Ubiquitous B-Tree. Computing Surveys, June 1979, vol. 11, n° 2.

Valduriez P. Join Indices. ACM TODS, June 1987, Vol. 12, No. 2, Pages 218-246.

Mohod A P, Chaudhari M S. Improve Query Performance Using Effective Materialized View Selection and Maintenance: A Survey. IJCSMC, April 2013, Vol. 2, Issue. 4, pg. 485 – 490.

International Organization for Standardization (ISO). Information technology — Database languages SQL Part 16: Property Graph Queries (SQL/PGQ). (Edition 1, 2023), ISO/IEC 9075-16: 2023.

Costa C H, Filho J V B M, Lou Y, Lai L, Lyu B, Yang Y, Zhou X, Yu W, Zhang Y, Zhou J. Towards a Converged Relational-Graph Optimization Framework. Proc. ACM Manag. Data, Vol. 2, No. 6 (SIGMOD), December 2024.

Fagin R, Kolaitis P G, Nash A. Towards a Theory of Schema-Mapping Optimization". PODS'08, June 9–12, 2008, Vancouver, BC, Canada.

Bézivin J, Gerbé O. Towards a precise definition of the OMG/MDA framework. Proc. 16th Annual Int. Conf. on Automated Software Engineering (ASE 2001).

Asaad C, Ba K. NoSQL Databases: Seek for a Design Methodology. 8th Int. Conference, MEDI 2018, Marrakesh, Morocco, October 24–26, 2018.

Bondiombouy C, Valduriez P. Query Processing in Multistore Systems: an overview. RR-8890, INRIA Sophia Antipolis - Méditerranée. 2016, pp. 38. hal-01289759v2.

Atzeni P, Bugiotti F, Rossi L. Uniform Access to Non-relational Database Systems: The SOS Platform. J. Ralyt´e et al. (Eds.): CAiSE 2012, LNCS 7328, pp. 160–174, 2012.

Vathy-Fogarassy Á, Hugyák T. Uniform data access platform for SQL and NoSQL database systems. Information Systems, September 2017, Volume 69, Pages 93-105.

Shin K, Hwang C, Jung H. NoSQL Database Design Using UML Conceptual Data Model Based on Peter Chen's Framework. Int. Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 5 (2017) pp. 632-636.

Abdelhedi F, Brahim A A, Atigui F, Zurfluh G. Logical Unified Modeling For NoSQL DataBases. 19th ICEIS 2017, Apr 2017, Porto, Portugal. pp. 249-256. hal-01782574.

Stonebraker M. NoSQL and Enterprises. Cacm | august 2011 | vol. 54 | no.

Natti, M. (2023). Migrating from Oracle to PostgreSQL: Leveraging Open-Source to Reduce Database Costs and Enhance Flexibility. The Eastasouth Journal of Information System and Computer Science, 1(02), 109–112. https://doi.org/10.58812/esiscs.v1i02.433