International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I9P119

Volume 72 Issue 9, 126-130, September 2024

Original Article

Modern Trends in No SQL Data Bases

Saravanan Subramanian[1], Subhash Saravanan[2]

[1]*Sr Manager, Software Development, Amazon Web Services Inc, Seattle, Washington, USA.*
[2]*Computer Science, University of Washington, Washington, USA.*

[1]*Corresponding Author : subrasar@amazon.com*

Received: 02 August 2024 Revised: 31 August 2024 Accepted: 24 September 2024 Published: 30 September 2024

Abstract *- The exponential growth of data in the digital age, fueled by social media, smartphones, and cloud computing, has exposed the limitations of traditional Relational Database Management Systems (RDBMS) in managing the velocity, volume, and variety of data. This has led to the rise of NoSQL databases as a viable alternative. This article provides an overview of NoSQL databases, highlighting their key properties and various types, including key-value, column-oriented, document-oriented, and graph databases, along with their respective data models. Additionally, it discusses potential limitations such as restricted ACID transactions, proprietary APIs, and the trade-offs involved in the CAP theorem.*

Keywords *- NoSQL, RDBMS, Open Source, Data Bases, Big Data, Gen AI, Large Language Models (LLM), Vector*.

1. Introduction

Relational Database Management Systems (RDBMS) have been the cornerstone of data management for nearly three decades. However, the rapid evolution of technology, characterized by the advent of social media, smartphones, and cloud computing, has led to the generation of large volumes of data at unprecedented velocities. This data ranges from simple text messages to high-resolution video files, presenting a significant challenge for traditional RDBMS. These systems struggle to cope with the velocity, volume, and variety of modern data, revealing a critical research gap in their ability to manage and process such diverse datasets efficiently. Additionally, most RDBMS software is licensed and requires enterprise-class, proprietary hardware, further limiting their accessibility and scalability. This gap has paved the way for the emergence of open-source NoSQL databases, which offer dynamic schemas, distributed architecture, and horizontal scalability on commodity hardware, addressing the limitations of traditional RDBMS in the current data landscape.

2. Properties of NoSQL

2.1. Dynamic Schema

NoSQL databases offer schema flexibility, allowing developers to add new columns dynamically. Rows within these databases may or may not contain values for these columns, and there is no strict enforcement of data types [2]. This flexibility proves advantageous, especially when anticipating frequent schema changes during a product's lifecycle.

2.2. Variety Of Data Types

NoSQL databases can support various types of data. The system enables the storage of structured, semi-structured, and unstructured data. It allows for the storage and manipulation of log files, image files, video files, graphs, jpegs, JSON, and XML in their original format, with no need for pre-processing [2]. As a result, it minimizes the requirement for ETL (Extract, Transform and Load).

2.3. High Availability Clusters

NoSQL databases enable distributed storage through the utilization of commodity hardware. It ensures high availability by expanding horizontally [2]. NoSQL databases can leverage the flexible capabilities of Cloud infrastructure services through this feature.

2.4. Open Source

NoSQL databases are often open-source licensed. The software is available for free, and developers can use a majority of them in commercial products at no cost. The open-source codebases offer flexibility in adjusting and tailoring to meet the demands of businesses [2]. Although most open-source licenses are available for free, there are subtle variations in their usage for commercial purposes on a larger scale.

2.5. API Access

NoSQL databases not only rely on SQL for data retrieval; they also provide rich API interfaces for performing DML (Data Manipulation Language) and CRUD (Create, Read, Update, Delete) operations [2]. These APIs are more developer-friendly and are supported in a variety of programming languages.

3. Variations of NoSQL Databases

There are several types of NoSQL databases, including Key-Value databases, Column-oriented databases, Document-

oriented databases, and Graph databases. Most NoSQL databases share similarities with Relational Database Management Systems (RDBMS) in terms of their data models. Within a database server, there can be multiple databases, each containing one or more tables. These tables consist of rows and columns to store actual data. While this hierarchy is common across all NoSQL databases, the specific terminologies may vary.

3.1. Key Value Pair Database

Key-value databases are a product of the research presented in the Dynamo whitepaper published by Amazon [3]. The Key-Value database enables users to store data in a straightforward <key>:<value> format, where the key helps to retrieve the value. These are very simple data stores but optimized for faster retrieval.

Data Model: The table contains many key spaces, and each key space can have many identifiers to store key-value pairs.

The key space functions as a column in a typical RDBMS, and the group of identifiers presented under the key space are analogues to the non-primary key values of the same row.

This structure is suitable for building simple, non-complex, highly available applications since most of the Key Value Databases support in-memory storage and are used for building cache mechanisms.

Examples: Amazon DynamoDB, Redis

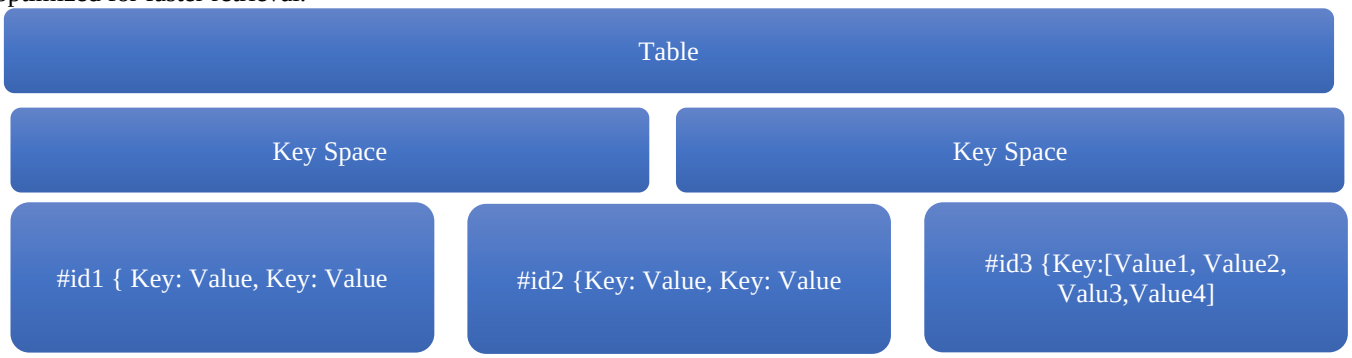

Fig. 1 Diagram of key-value pair database

3.2. Column Oriented Database

Google's white paper on Bigtable introduced column-oriented NoSQL databases, which diverge from the conventional relational model [4]. In these databases, columns can be dynamically added, resulting in wider tables. To manage this breadth, columns are grouped into a 'Column Family' or 'Super Column.' While the use of Column Families is optional in some databases, they prove valuable for handling sparsely distributed values—such as those encountered in medical research projects.

Data Model: The table contains column families (optional). Each column family contains many columns. Key-value pairs might sparsely distribute the values for columns.

Examples: HBase, Apache Cassandra

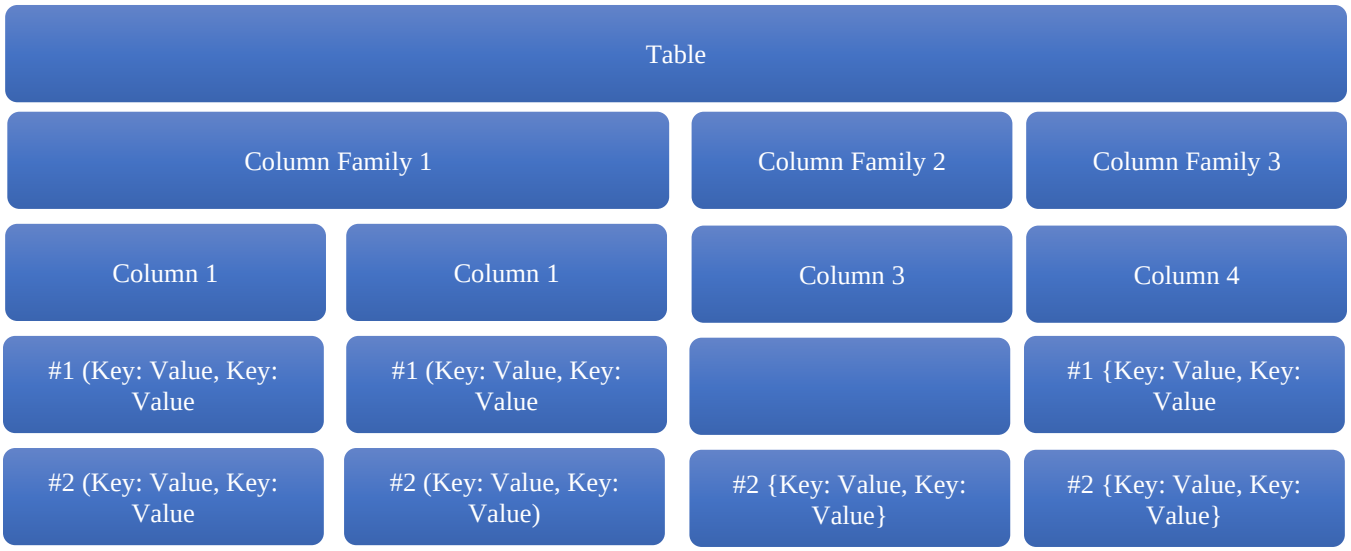

Fig. 2 Diagram of column-oriented database

3.3. Document Oriented Data Stores

Document-oriented databases are designed to store semi-structured data in various formats, including JSON, XML, YAML, and even Word documents [4]. In these databases, a document acts as the unit of data, similar to a row in a relational database management system. A group of these documents is organized into a "Collection," which is analogous to a table in relational databases.

Data Model: The Database encompasses a multitude of collections. A Collection comprises many documents. Each document may contain a JSON, XML, YAML, or even a Word Document. Document databases are well-suited for web-based applications and applications that expose RESTful services.

Examples: MongoDB, Couchbase

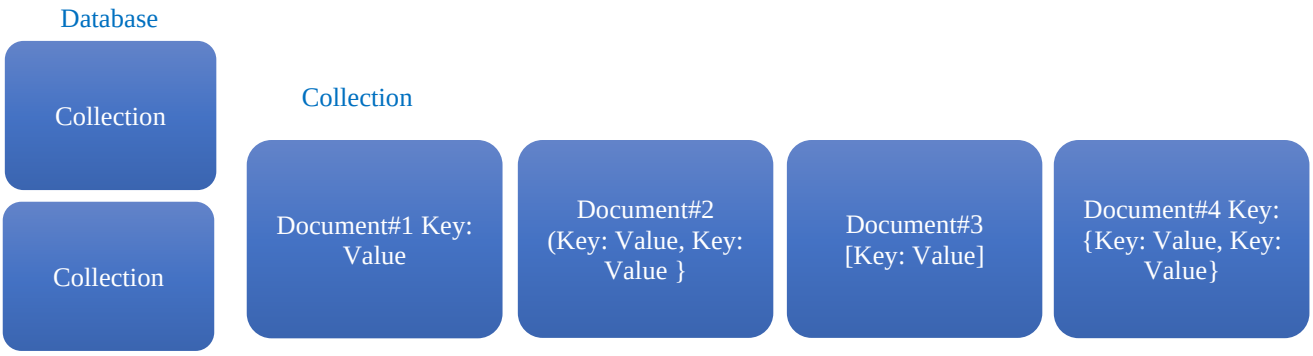

Fig. 3 Diagram of a document-oriented database

3.4. Graph Database

In real-world applications, a graph comprises vertices and edges, which are referred to as nodes and relationships in graph databases [6]. These databases facilitate the storage and execution of data manipulation operations on nodes, relationships, and their associated attributes. Graph databases demonstrate enhanced performance with directed graphs, where explicit relationships exist between nodes.

Data Model: The graph database serves as a two-dimensional depiction of the graph. The graph can be compared to a table. Each graph consists of columns for Node, Node Properties, Relation, and Relation Properties. Each row of these columns will contain corresponding values. The columns in properties can contain key-value pairs. Graph databases are well-suited for addressing social media and network challenges that involve complex queries and multiple joins.

Examples: Neo4j, Amazon Neptune, ArangoDB, OrientDB.

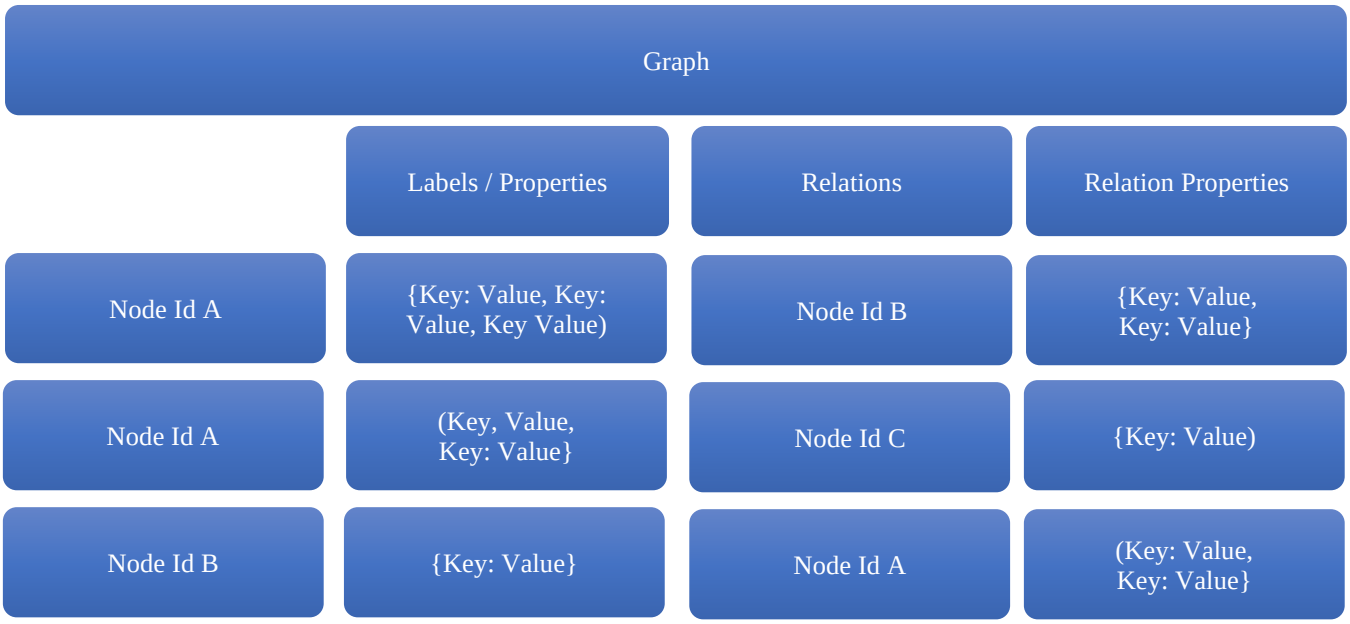

Fig. 4 Diagram of graph database

3.5. Time Series Database

A time-series database is designed to handle time-stamped data, such as metrics, sensor readings, and logs [7]. These databases manage, retrieve, and analyse large volumes of data points indexed by time, facilitating expedited queries and real-time analytics. They are capable of handling high ingestion rates and offer advanced functionalities for time-based queries, including aggregations, down-sampling, and sliding windows. Time-series databases employ data compression techniques to minimize storage requirements and enforce default data retention policies. They are extensively used in monitoring, IoT, and financial applications, enabling the tracking and analysis of trends over time with exceptional performance and scalability.

Data Model: In a time-series database, the data model is constructed based on time-stamped data points, and then it groups them into time series. Each series contains relevant metrics, tags (metadata), and fields (values). Timestamps serve as unique identifiers for the recording time of data.

Examples: Prometheus, InfluxDB, Amazon Timestream

3.6. Vector Database

A vector database handles the storage, indexing, and searching of high-dimensional vector embeddings, which serve as numerical representations for data items like text, images, or audio [8]. In recent days, the use of vector databases in Generative AI products, along with LLM (Large Language Models), has increased. The vector databases store the data to enable the semantic meaning of the data. They excel in similarity searches, so they are primarily used in recommendation systems, natural language processing and computer vision. They also handle high-volume, large-scale, complex data using advanced indexing techniques.

Data Model: The data model of the vector database includes vectors, unique identifiers, metadata, and optional index. A vector is an array of numbers representing feature set data. Unique Identifier to map each vector to allow faster retrieval and association with metadata. Metadata holds data, labels, categories, timestamps, and more. Additional indexes optimize database search and retrieval.

Examples: Pinecode, ElasticSearch, Qdrant

3.7. Search Optimized Database

Search-optimized data stores handle complex search queries efficiently. These data stores index the data to facilitate fast retrieval and complex searches like full-text, patterns and filters [9]. These stores optimize query performance by using inverted indexes to map to documents and supporting ranking algorithms to prioritize relevant results. These are suitable for building search engines, e-commerce platforms, and content management systems. The search-optimized data stores such as Apache Solr and Open Search leverage advanced search algorithms along with search-optimized data structure and low latency data storage systems.

Data Model: The search-optimized stores contain several key components in their data model. The data is stored in an index. Each record in the index is called a document, and a document contains multiple fields. Also, an inverted index that maps keywords in the document in which they appear to enable fast full-text search.

Examples: Apache Solr, Elasticsearch, and Amazon Cloud Search.

3.8. Multi-Model Database

A multi-model database can accommodate various data models within a single database system, including relational, document, key-value pair, columnar, and graph models. This flexibility allows software developers to choose the most appropriate data model for their specific business use case, regardless of the underlying data store system. Additionally, multi-model databases offer a unified query language capable of handling diverse data types stored within the same system. By managing different data types in one place, multi-model databases simplify application management and eliminate the need for multiple databases.

4. General Vs Purpose Built Databases

4.1. Standard Data Bases

Relational databases, such as MySQL and PostgreSQL, are standard, general-purpose systems. They manage diverse applications and data types using a structured, table-based schema and support SQL querying. However, these databases may not be as efficient as purpose-built systems for specialized tasks due to the lack of optimization for specific workloads, leading to performance inefficiencies. Scaling relational databases to handle big data or high transaction rates is complex and resource-intensive. Additionally, adapting to changing data structures is challenging because of the rigidity of schemas. General-purpose databases also lack specialized features and advanced indexing capabilities, which can affect their efficiency for certain tasks.

4.2. Purpose-Built Data Bases

Purpose-built databases deliver optimized performance and specialized features to meet specific use case requirements cost-effectively. For example, InfluxDB is a specialized database system that excels in handling time-stamped data, whereas Neo4j focuses on managing complex data relationships.

4.3. Challenges

To select a purpose-built database for a specific use case, it is important to consider several critical factors. Lack of full ACID transaction support in NoSQL databases (e.g., MongoDB, Couchbase, Cassandra) can compromise data integrity in applications that require strong consistency.

Limited SQL support and proprietary APIs cause learning unique query languages and creating custom adaptors. Relational databases have JOIN as a core feature, unlike many NoSQL databases. Finally, CAP theorem trade-offs are inherent in NoSQL databases, as they typically prioritize only two of the three properties—Consistency, Availability, and Partition Tolerance—requiring careful consideration of application requirements.

5. Conclusion

The exponential surge of big data, driven by advancements in social media, mobile computing, and cloud technologies, has highlighted the inadequacies of traditional RDBMS in handling the velocity, volume, and variety of modern data. NoSQL databases have emerged as an alternative, offering dynamic schemas, diverse data type support, high availability, and open-source accessibility. These paradigms provide specialized data models for various applications, allowing developers to choose the most suitable database technology for their needs rather than relying on a one-size-fits-all relational model. Despite their advantages, NoSQL databases come with trade-offs. Selecting a NoSQL solution requires careful consideration of application needs due to reduced ACID support, proprietary APIs, and challenges posed by the CAP theorem. An optimal data management strategy may involve a combination of both relational and NoSQL approaches. NoSQL databases are evolving to address their limitations by leveraging cloud technologies and incorporating features of traditional RDBMS.

Funding Statement

The authors solely manage the funding through their own sources, with no financial support received from external organizations.

Acknowledgments

Saravanan Subramanian and Subhash Saravanan contributed equally to this work.

References

[1] Neha Bansal, Kanika Soni, and Shelly Sachdeva, "Journey of Database Migration from RDBMS to NoSQL Data Stores," *Big-Data-Analytics in Astronomy, Science, and Engineering*, vol. 13167, pp. 159-177, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[2] Stefanie Scherzinger, and Sebastian Sidortschuck, "An Empirical Study on the Design and Evolution of NoSQL Database Schemas," *Conceptual Modeling*, pp. 441-455, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[3] Giuseppe DeCandia et al., "Dynamo: Amazon's Highly Available Key-Value Store," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205-220, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[4] Fay Chang et al., "Bigtable: A Distributed Storage System for Structured Data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 205-218, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[5] Mateusz Papiernik, and Mark Drake, *An Introduction to Document-Oriented Databases*, DigitalOcean, 2021. [Online]. Available: https://www.digitalocean.com/community/conceptual-articles/an-introduction-to-document-oriented-databases

[6] Graph Database Concepts, Neo4j, 2024. [Online]. Available: https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/

[7] Time Series Database (TSDB) Explained, Technical Paper, InfluxData, 2023. [Online]. Available: https://www.influxdata.com/time-series-database/

[8] Vector Database, AI Skills Challenge, Microsoft, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/cosmos-db/vector-database

[9] Sanket Sharma, Mastering the Art of Optimizing Complex SQL Queries, DEV Community, 2023. [Online]. Available: https://dev.to/thesanketsharma/mastering-the-art-of-optimizing-complex-sql-queries-4ii5