International Journal of Scientific Research in Computer Science, Engineering and Information Technology

ISSN : 2456-3307

OPEN ACCESS

Available Online at : www.ijsrcseit.com
https://doi.org/10.32628/CSEIT239074

How to Choose the Right Database for Your Application

Nagaraju Thallapally
University of Missouri Kansas City Missouri, United States

ARTICLEINFO

Article History:

Accepted: 10 Oct 2023

Published: 22 Oct 2023

Publication Issue
Volume 9, Issue 5
September-October-2023
Page Number
414-418

ABSTRACT

Choose the right database; it is a very important choice that determines the speed, scale, and success of any application or system. Various databases now exist with a growing variety, from relational databases to newer NoSQL databases, so development teams, system architects, and business users need to do some homework to find the one that is right for them. In this article, I will show decision-makers how to select the right database based on data structure, scalability, transaction size, consistency, and performance. This article explains what to consider when selecting a database type and contrasts the types.

Keywords: Database, Right Database, Speed, Scale, Success, Application, Relational Databases.

1 Introduction

The critical nature of database selection in modern data-driven environments requires no exaggeration. The database stands as the essential component for effective data storage and retrieval in systems where large volumes of data are generated and used by businesses and applications. Any application relies on its database to store essential information, including user details and transaction records, together with product catalogs, among other data. Applications and systems depend on database performance and stability to achieve functionality and success, which makes selecting the right database a critical choice for development and IT teams.

Choosing the correct database becomes even more crucial today because data types are growing in variety and complexity. Organizations primarily chose relational databases like MySQL, Oracle, and SQL Server historically, and many applications still depend on them today. The emergence of big data together with cloud computing advancements and high-speed application needs has led to the development of new database models. NoSQL databases like MongoDB, Cassandra, and Redis were created to solve the problems traditional relational databases face by delivering increased flexibility and scalability while processing unstructured and semi-structured data faster. NewSQL databases offer organizations facing specialized requirements another choice by combining NoSQL scalability with relational consistency (Cattell, 2011).

The process of selecting an appropriate database cannot be standardized across all organizations. Selecting the ideal database requires knowledge of available database types alongside their respective advantages and limitations, which suit different business requirements and use cases. The organization's requirements must be matched by the selected database through careful evaluation of factors including data structure, scalability, transaction size, consistency, and performance. For enterprises running large-scale real-time applications or generating data-driven insights, database systems must manage increasing data volumes without sacrificing performance or availability.

This paper investigates the most popular industry database types, including relational databases along with NoSQL and NewSQL databases. The selection process will be analyzed by identifying primary influencing factors, and we will evaluate the advantages and disadvantages of each database type. After reading this paper, you will possess a thorough knowledge base to effectively choose databases among numerous possibilities. When developers, system architects, and business leaders understand this information, they will be able to make decisions that improve application performance and efficiency, which will help their organizations achieve success.

2 Types of Databases

2.1 Relational Databases

This is the most popular type of database, a relational database. They maintain the information in tables with schema pre-defined, and they use rows and columns to model entities and relationships. Databases in relational type use Structured Query Language (SQL) to query data and support querying, data manipulation, and transactions (Selinger et al., 1979).

Examples: MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database

Key Features:

- **ACID Compliance:** Relational databases ensure data consistency, integrity, and reliability through ACID (Atomicity, Consistency, Isolation, Durability) properties.
- **Normalization:** Data is often normalized to eliminate redundancy and improve consistency.
- **Complex Queries:** SQL databases are well-suited for complex queries, joins, and aggregations.
- Transactions: Strong support for multi-step transactional operations.
- **Use cases:**
- Applications requiring structured data and transactional consistency (e.g., financial applications, inventory management, e-commerce platforms) (Williams & Lane, 2004).
- Systems that require complex reporting and querying capabilities.

2.2 NoSQL Databases

NoSQL databases are much more extensible than relational databases and store both unstructured and semi-structured data. These databases can scale horizontally to huge amounts of heterogeneous data. They do not have to use a particular schema and can be used for different data models, such as key-value, document, column-family, and graph databases (Cattell, 2011). Examples: MongoDB, Cassandra, Redis, Neo4j

Key Features:

• **Scalability:** NoSQL databases excel in distributed environments, offering horizontal scaling for large amounts of data and traffic.
• **Flexible Schema:** NoSQL databases can handle unstructured and semi-structured data without requiring a rigid schema.
• **High Availability:** Many NoSQL databases are designed with built-in replication and fault-tolerance features.
• **Eventual Consistency:** Unlike SQL databases, NoSQL databases may follow an eventual consistency model for distributed systems.

Use cases:
• Large-scale applications such as social media platforms, recommendation engines, and big data analytics.
• Applications that require high availability, fault tolerance, and horizontal scalability.

2.3 In-Memory Databases

Data is stored in in-memory databases (i.e., not on disk but in the system's main memory (RAM) and therefore can be processed very quickly. They are perfect for applications in real-time with low latency requirements (Wiese, 2015).

Examples: Redis, Memcached, Apache Ignite

Key Features:

• **Speed:** In-memory databases offer faster data access compared to disk based systems due to the absence of I/O bottlenecks.
• **Persistence Options:** Some in-memory databases support persistence options to periodically write data to disk for durability.
• **Low Latency:** Ideal for applications where real-time responses are required, such as caching, session management, or high-frequency trading platforms.

Use cases:
• Real-time applications such as gaming platforms, recommendation systems, and ad targeting.
• Caching frequently accessed data to reduce latency in web applications.

2.4 Graph Databases

Graph databases are databases that are used to store and query data in graphs. They represent relationships through nodes, edges, and properties, which is perfect for scenarios where relationships between data elements are a primary part of the query logic. Below is the

Examples: Neo4j, ArangoDB, Amazon Neptune

Key Features:

• **Relationship-Centric:** Graph databases excel at representing complex relationships and networks of data.
• **Efficient Traversals:** They offer efficient querying and traversal of connected data, such as finding shortest paths or detecting cycles.
• **Flexible Schema:** Like NoSQL databases, graph databases do not require a predefined schema.

Use cases:
• Social networks, fraud detection, and recommendation engines.
• Any system where relationships or connected data are central to the business logic.

3 Key Considerations When Choosing a Database Data Structure:

The first thing to think about while choosing a database is how the data should be stored. Relational databases are for structured data, that is, with explicit relationships, and NoSQL databases are for unstructured or semi-structured data. Graph databases are perfect for applications that need to do a lot of complex relationships and traversals of graphs, and in-memory databases are great for high-performance data loads.

3.1 Scalability:

Handling bigger data volume and traffic requires scale. Relational databases scale vertically on more powerful hardware to handle a larger number of requests. NoSQL databases, on the other hand, scale horizontally and can share data among several servers, processing terabytes of data.

3.2 Transactional Requirements:

A relational database is usually the optimal choice if it involves many transactions or if we need high consistency; then it should be an ACID-compliant database. NoSQL databases might provide eventual consistency, which is suitable for systems where the immediate consistency is not essential, like the large web applications.

3.3 Performance:

Database performance depends on data volume, query complexity, and high availability. In-memory databases are also very fast, and they are great for low latency. Conversely, relational databases can become performance congested by big data or large joins, which can be addressed through indexing and caching.

3.4 Maintenance and Support:

Database maintenance includes regular backups, upgrades, security updates, and performance tuning of databases. Relational databases generally come with proven tools and vendors. NoSQL databases can need more custom management but also have an open-source option.

4 Conclusion

The database that you select is a big decision and should be determined based on the application requirements and objectives. Depending on the data type, scalability, transaction, performance, and maintenance issues, developers and architects can choose the database with the greatest match to their goals. Whether it is a conventional relational database, an extensible NoSQL database, a powerful in-memory database, or a graph database for relationship-oriented queries, all databases provide different benefits based on the use case. And with that in mind, the decision-makers can make smart decisions to optimize performance now and in the future.

References

1. Codd, E. F. (1970). 'A Relational Model of Data for Large Shared DataBanks.' Communications of the ACM, 13(6), 377-387.
2. Abadi, D., Boncz, P., Harizopoulos, S., Idreos, S., & Madden, S. (2013). The design and implementation of modern column-oriented database systems. *Foundations and Trends® in Databases*, *5*(3), 197-280.
3. Kimball, R., Ross, M. (2013). The Data Warehouse Toolkit. Wiley.
4. Bansal, A. (2023). Optimizing RAG with Hybrid Search and Contextual Chunking. In Journal of Engineering and Applied Sciences Technology (pp. 1–5). ScientificResearchandCommunityLtd. https://doi.org/10.47363/jeast/2023(5)e114
5. Elmasri, R. (2008). *Fundamentals of database systems*. Pearson Education India.
6. Cattell, R. (2011). Scalable SQL and NoSQL data stores. *Acm Sigmod Record*, *39*(4), 12-27.

7. Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., & Helland, P. (2018). The end of an architectural era: It's time for a complete rewrite. In *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker* (pp. 463-489).
8. Williams, H. E., & Lane, D. (2004). Web Database Applications with PHP and MySQL: Building Effective Database-Driven Web Sites. " O'Reilly Media, Inc.".
9. Lourenço, J. R., Cabral, B., Carreiro, P., Vieira, M., & Bernardino, J. (2015). Choosing the right NoSQL database for the job: a quality attribute evaluation. Journal of Big Data, 2, 1-26.
10. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). Database system concepts.
11. Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., & Price, T. G. (1979, May). Access path selection in a relational database management system. In Proceedings of the 1979 ACM SIGMOD international conference on Management of data (pp. 23-34).
12. Özsu, M. T., & Valduriez, P. (1999). Principles of distributed database systems (Vol. 2). Englewood Cliffs: Prentice Hall.
13. Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (1998). Density-based clustering in spatial databases: The algorithm gdbscan and its applications. Data mining and knowledge discovery, 2, 169-194.
14. Sadalage, P. J., & Fowler, M. (2013). NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Pearson Education.
15. Kelly, A. (2013). Making Sense of NoSQL: A Guide for Managers and the Rest of Us by Ann Kelly and Dan McCreary.
16. Wiese, L. (2015). Advanced data management: for SQL, NoSQL, cloud and distributed databases. Walter de Gruyter GmbH & Co KG.
17. Strauch, C., Sites, U. L. S., & Kriha, W. (2011). NoSQL databases. Lecture Notes, Stuttgart Media University, 20(24), 79.
18. Moniruzzaman, A. B. M., & Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*.
19. Orend, K. (2010). Analysis and classification of NoSQL databases and evaluation of their ability to replace an object-relational Persistence Layer. *Architecture*, *1*, 1-100.
20. Date, C. J. (1977). *An Introduction to Database Systems*. Addison-Wesley Publishing Company.