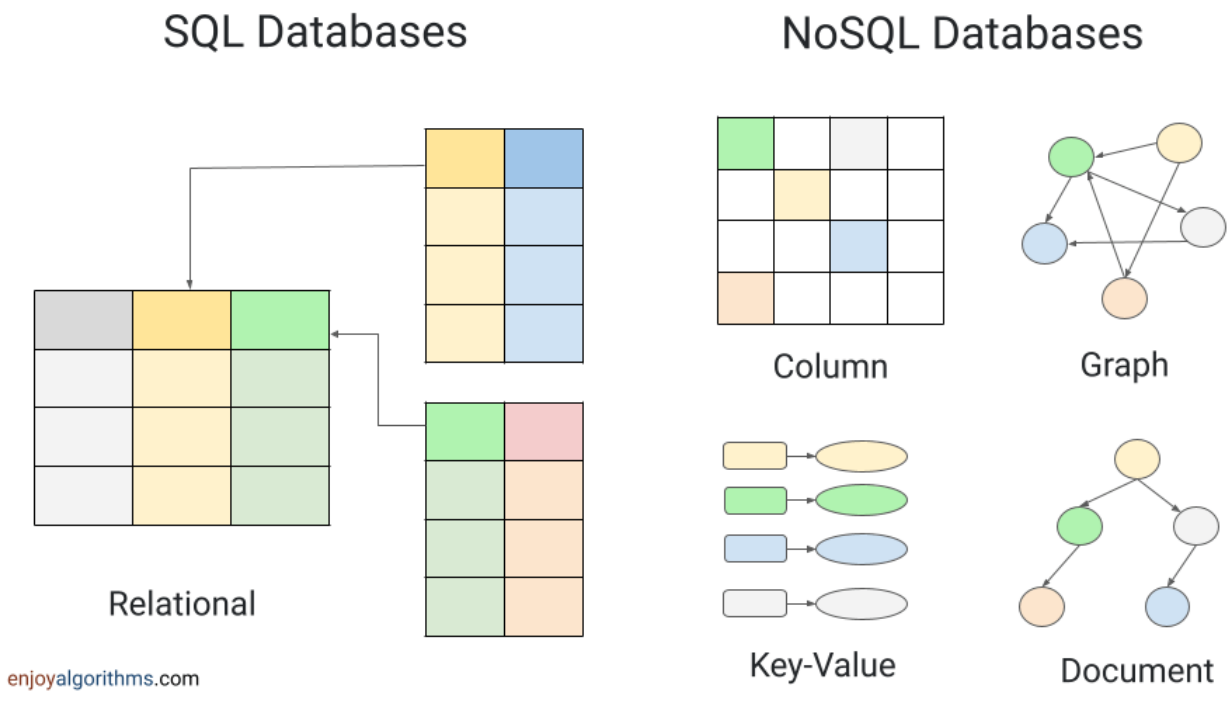
SQL vs NoSQL Databases: What is the Difference?

Introduction

There are two types of databases in distributed systems: SQL and NoSQL databases. Each has its own strengths and weaknesses. SQL databases are the best choice for consistency, complex query, and ability to handle structured data, while NoSQL databases are the best for scalability, flexibility, and ability to handle unstructured data.

So the decision between SQL and NoSQL will depend on the project-specific requirements: data size, data types, relationships between data, query complexity, data consistency requirements, availability requirements, scalability requirements, flexibility of the data model, etc.

Let's move forward to learn more about properties, advantages, disadvantages, use cases and comparison of both databases. We will first start with the SQL databases and then we will discuss NoSQL databases.



What is SQL?

In relational databases, data is organised into tables (rows and columns) with predefined relationships between them. Here each column represents a specific data field and each row contains a set of actual data values. In a practical scenario, one table entry may connect to many other table entries that are related to many entries in another table!

So, SQL (Structured Query Language) is a programming language used to query and manage data in relational databases. It provides a set of commands to perform various queries (search, insert, update, delete, etc) and commands to create new tables, define relationships between tables, create indexes, etc. We highly recommend you explore various SQL commands.

- SQL databases are optimized to efficiently handle **complex queries** that involve multiple tables, filtering conditions, aggregations, sorting, etc.
- Before starting with SQL databases, It is important to define the **structured schema**. What is a structured schema? It is a predefined plan that outlines the tables, columns, data types, relationships, constraints, and other aspects of the database design.

SQL databases can **scale vertically** i.e. we can increase the capacity of the same machine by adding more CPU, RAM, or other hardware. Horizontal scaling is not well-supported in traditional SQL databases because they only work with one main database server and replicas. So how can we do it? There are two options:

1. One idea is to write partition logic in the application code to partition data. But this will add complexity because we need to manage multiple connections and query the appropriate partition.
2. Another idea is to use MySQL Cluster or Vitess for MySQL or Citus for PostgreSQL. Their built-in partitioning helps us grow the database without adding partitioning logic to the application. For example, Vitess enables live re-partitioning with minimal read-only downtime. But they are not without limitations!

ACID Compliance

SQL databases exhibit ACID compliance (Atomicity, Consistency, Isolation, Durability). ACID compliance guarantees the integrity of data transactions and ensures a high level of reliability. This makes them suitable for financial systems and other applications that require reliable transaction processing.

- **Atomicity:** Each transaction is treated as a single unit of work and all changes to the database made by a transaction must either succeed or fail as a whole. If any part of the transaction fails, all changes made by the transaction must be rolled back.
- **Consistency:** Transaction must preserve the consistency of the database and any changes made by a transaction must not violate any integrity constraints!
- **Isolation:** Concurrent transactions do not interfere with each other. In other words, each transaction must be executed in isolation, without any interference from other transactions.
- **Durability:** Once a transaction is committed, its changes are permanent and cannot be undone, even in the case of system failures.

Some other facts

- Several types of SQL databases are available like MySQL, Oracle, PostgreSQL, Microsoft SQL Server, etc. Each one of them are are widely used, and provides unique features with extensive documentation.
- SQL databases have a long history and a huge community base.

What is NoSQL?

NoSQL (Not only SQL) is a term used to describe databases that differ from traditional SQL databases. Instead of following a structured data model like SQL, NoSQL databases are designed to handle large volumes of structured, unstructured, and semi-structured data. **Note:** NoSQL can

also store data found within SQL databases, but it stores it differently!

- They have very **flexible schemas**, which makes NoSQL databases a good choice for storing diverse data types in a scalable manner. This flexibility makes it easier to handle data that may change over time or make changes to the database because of changes in requirements.
- Changing the schema will not impact development cycles or create any downtime for the application. Due to this, we can iterate quickly and continuously integrate new application features.

NoSQL can deliver **high performance** in terms of read and write operations. This makes them suitable for applications that require fast access to data. In certain cases, queries in NoSQL databases can be faster compared to SQL databases. The reason is: data in SQL databases are often structured and normalized i.e. retrieving information may require joining data from multiple tables. As the size of the tables increases, performing joins on large datasets can add performance overhead.

Unlike SQL, NoSQL databases provide built-in features to **scale horizontally** by adding cheaper commodity servers to distribute the load. This helps us increase storage capacity, throughput and handle high volumes of concurrent read and write operations with no single point of failure. In other words, NoSQL databases are distributed databases i.e. data is stored on various servers to ensure availability, performance and scalability. If some of the database servers go offline, the remaining servers can continue to run.

There are several **types of NoSQL databases**: key-value stores, document-oriented databases, column-oriented databases, and graph databases. Each one is optimized for different use cases. Let's understand some of them:

- Due to simplicity and high performance, **key-value stores** are best for fast data retrieval. For example, e-commerce can use **Redis** to cache frequently accessed data: product catalogues, user sessions, etc.

- Due to flexible schema and the ability to handle a wide variety of data types, **document-oriented databases** are best for storing unstructured data in JSON format. For example, a blogging platform can use **MongoDB** to store data related to blogs: title, content, author, date, tags, and other metadata.
- Due to efficient aggregation and the ability to query specific columns, **column-oriented databases** are best to store structured data in a columnar format. For example, data warehousing applications can use **Apache Cassandra** to store sales data, financial data, customer data or any other structured data that requires complex aggregations for generating reports.
- Due to the efficient modelling of relationships and ability to handle complex graph queries, **graph databases** are the best for storing interconnected data. For example, social media can use **Neo4j** to store data related to users and followers: User profiles, posts, comments, likes, and other interactions.

Some other popular NoSQL databases are HBase, Couchbase, DynamoDB, Apache CouchDB, and Riak. These days, community support for these databases is growing at a very fast rate.

BASE Properties and CAP Theorem

To handle various requirements, NoSQL databases follow various trade-offs based on the CAP theorem. Most of the time, they follow BASE compliance (Basically Available, Soft state, Eventually consistent), which prioritizes availability and partition tolerance over immediate consistency.

- **Basically Available:** The system guarantees availability even if it returns outdated data. In other words, the system always provides a response and remains available even in the event of a failure.
- **Soft State:** The state of the data can change over time due to the lack of immediate consistency. So the system allows for temporary inconsistencies across nodes during updates or failures.
- **Eventually Consistent:** The data is replicated to different nodes and eventually reaches a consistent state, but consistency is not

guaranteed at a transaction level. Until the system eventually converges to a consistent state, data reads are still possible, even though they may not reflect updated data.

So in other words: Strict consistency may not always be desirable. So BASE compliance provides a more relaxed idea to maintain data consistency (eventual consistency) in NoSQL databases.

But the above scenario is not always the case in modern NoSQL databases. Sometimes they also prefer strong consistency over availability. This is another side of a practical story based on the CAP theorem! These days, NoSQL databases provide configurations to change degrees of consistency levels to handle various requirements. We highly recommend to explore tradeoffs related to C, A and P in NoSQL databases like Redis, MongoDB, Cassandra, HBase and Neo4j.

When to use SQL Databases?

1. Ideal for managing structured data with well-defined schemas.
2. Good choice for complex queries that involve multiple tables.
3. When an application requires strong data consistency and integrity (ACID properties).

When to use NoSQL Databases?

1. When we have a lot of data, data types and the amount of data will only grow over time.
2. When we need to scale up and scale down the system because of changing requirements.
3. When we require complex data models that cannot be accommodated by a relational model.
4. When high availability is our priority and data consistency and integrity are not a major concern.
5. When we need to constantly add new features and data types.
6. When we require high performance for read and write queries.

Drawbacks of SQL databases

1. Challenging to scale because database architecture is limited by the hardware capacity (due to vertical scaling). This can require additional hardware which can increase the cost.
2. Due to the strict schema, it is not a good choice when data requirements are changing.
3. Not suitable for storing and querying unstructured data.
4. Rely heavily on data consistency and integrity constraints.
5. As data volumes increase, SQL databases can lead to slower response times.

Drawbacks of NoSQL databases

1. NoSQL databases do not support ACID properties. This can be a constraint when strong data consistency and integrity are required. Note: Some NoSQL databases like MongoDB can integrate ACID rules.
2. Not optimized for reducing data duplication. This can result in larger database sizes if the same data is stored redundantly across multiple nodes. But this trade-off can be acceptable when performance and scalability are more important. Note: Many NoSQL databases do support compression and other optimization techniques to reduce storage requirements.
3. A single type of NoSQL database may not be able to cover all use cases. So one has to operate with multiple databases and data models. For example, graph databases are excellent for analyzing relationships in data but may not provide retrieval of data like range queries.
4. NoSQL databases do not have any standard query language like SQL. In other words, they lack the standard interface like SQL. For example, many NoSQL databases have unique data manipulation languages constrained by specific structures. So there is poor consistency between NoSQL languages.

	SQL Database	NoSQL Database
Data Model	Tables with rows and columns	Key-Value, Document-oriented, Column-oriented, and Graph
Schemas	Rigid	Flexible
Scaling	Vertical scaling (Require powerful hardware)	Horizontal scaling (Require commodity hardware)
Properties	ACID properties	Either CAP theorem or BASE properties
Development Year	Developed in 1970s with a focus on reducing data duplication.	Developed in the late 2000s with a focus on scaling and allowing for rapid application change.
Query Language	Support Structured Query Languages	Does not have any standard query language
Use Cases	Best for complex queries, multi-row transactions	Best for unstructured data, Not ideal for complex queries
Data Distribution	It can only run on a single system and does not follow distribution of data.	Designed to follow data distribution features like, partitioning, replication.
Examples	Oracle, MySQL, Microsoft SQL Server, PostgreSQL, etc.	MongoDB, CouchDB, Redis, DynamoDB, Cassandra, HBase, Neo4j, Amazon Neptune, etc.

enjoyalgorithms.com

Combining the best of SQL and NoSQL

In some cases, it may be useful to use both SQL and NoSQL databases in a single application to take advantage of each. For example, YouTube stores video content in a NoSQL database and user metadata and other information in a SQL database. This allows them to leverage the flexibility and scalability of NoSQL databases for storing large amounts of unstructured data, while also taking advantage of the structured data and complex query capabilities of SQL databases.

There are also databases that offer features of both SQL and NoSQL databases. For example, MySQL Document Store combines the structure of an SQL database with the features and flexibility of a NoSQL database. Similarly, MongoDB, a NoSQL database, also offers ACID transactions. This can be useful for applications that need to handle both structured and unstructured data and perform complex queries.

If you have any queries or feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!