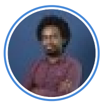
When to Use SQL vs NoSQL: An Expert Guide to Databases



Nebiyu Elias
Apr 24th, 2023

Blog

Given the plethora of database options, how do you decide which type of database is best for your application? Are you going to go for a SQL (relational) or a NoSQL (non-relational) database?

As developers, the choice is ours. And the success of a project will often depend on the choices you have made at the start.

It's worth realizing that there is no perfect choice when it comes to software engineering. Instead, everything has tradeoffs, and it's best to make decisions based on your project's requirements. By all means, avoid trying to make *perfect* decisions. Instead, make an *informed* decision.

Overview:

1. What are SQL and NoSQL databases?
2. What are the differences between SQL vs NoSQL databases?
3. What are the pros and cons of SQL vs NoSQL databases?
4. What are some common SQL and NoSQL databases use case?
5. When do you best use a SQL vs NoSQL database?

What Are SQL Databases?

SQL databases are relational databases that store data in related tables with rows and columns.

The theoretical basis for relational databases was first invented by Edgar F. Codd in 1969 while working at IBM. After the theoretical basis was laid out, the first relational databases emerged in the 1970s.

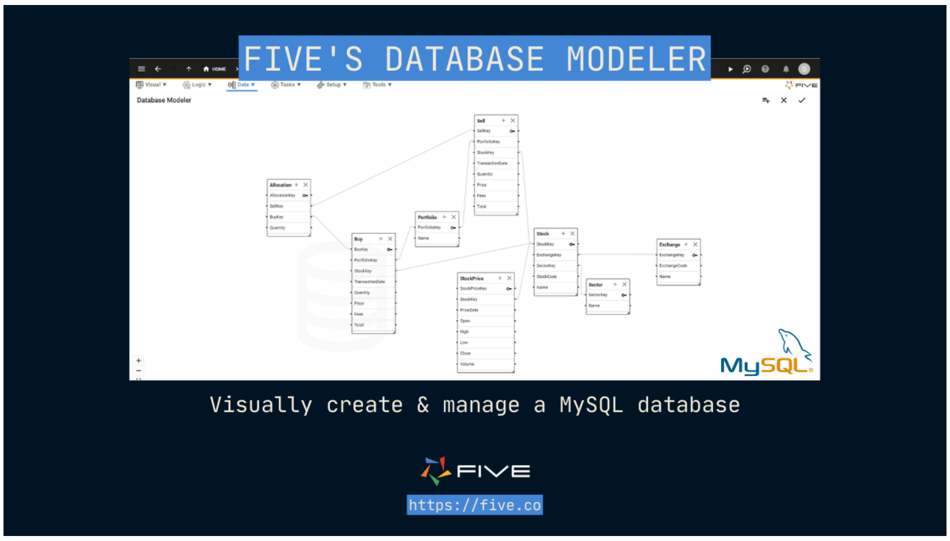
With over fifty years of history, SQL databases have been around for a long time. The technology and developers' understanding of it is very mature. There is also an active community of developers working hard to constantly make the technology better, more accessible, and more powerful.

SQL databases are designed with a focus on reducing data duplication and ensuring data integrity. To accomplish this, data is organized across tables, and these tables can have relationships. That's where the term "relational" comes from.

Each table will have rows and columns. But you need to define the schema (structure) of the database before you store any data. That's why SQL databases are well suited for structured data. The schema typically includes:

- **Tables:** You define the tables inside your database by giving them names.
- **Columns**: Each table will have one or more columns. Each column will have a name, data type, and any constraints or rules that apply to the data.
- **Relationships:** You define the relationship between the tables, such as one-to-one, one-to-many, or many-to-many relationships.
- **Constraints:** Define the rules or conditions that must be satisfied by the data in the database.
- **Views:** Virtual tables that provide a way to represent data from one or more tables in a customized way.
- **Keys**: Primary Keys and Foreign Keys are core concepts of relational databases. They serve as unique identifiers for each record and enforce integrity in a table or between tables.

Database schemas are often shown as Entity-Relationship (ER) Diagrams, which visually map out tables and their relationships. For example, this is how inside Five, developers can create and manage their MySQL database.



Examples of popular relational databases that follow the SQL way include:

- MySQL,
- Microsoft SQL Server,
- PostgreSQL, or
- SQLite

What Are NoSQL Databases?

NoSQL is a general term used to mean “no SQL” or “not only SQL.” NoSQL databases are databases that do not follow the relational model first introduced through SQL databases. Instead, they are designed to deal with different types of data, which is less structured.

With the proliferation of the internet, we started experiencing the so-called data explosion phenomenon.

According to IBM, over 2.5 quintillion bytes of data are created every day. And not all data is structured data that fits neatly into relational databases. Instead, semi-structured or polymorphic data, which does not follow a predefined schema, became more and more prevalent. Traditional SQL databases were not well suited to handle such large volumes and types of data. Thus, we needed a different way to handle data.

In the late 2000s, NoSQL databases emerged to handle large amounts of unstructured data and high user loads. It was perfect timing for these kinds of databases because as data volumes increased, the cost of storing data went down dramatically.

NoSQL databases are well-suited for semi-structured or unstructured data. Unstructured data is information that is not arranged according to a preset data model or schema. Instead, NoSQL databases can deal with a dynamic schema for unstructured data.

NoSQL databases come in a variety of types based on their data model. The main types are:

- **Document Databases:** Store data in a document-oriented way, where each document has a unique identifier and contains key-value pairs, and the data is stored in a semi-structured format, like JSON, or XML. Examples of document-based NoSQL databases include MongoDB, Couchbase, Apache Cassandra, and CouchDB.
- **Key-Value:** Store data as key-value pairs, where each value is associated with a unique key. Redis, which is

often used for caching, session management, and real-time data processing, is one such example.

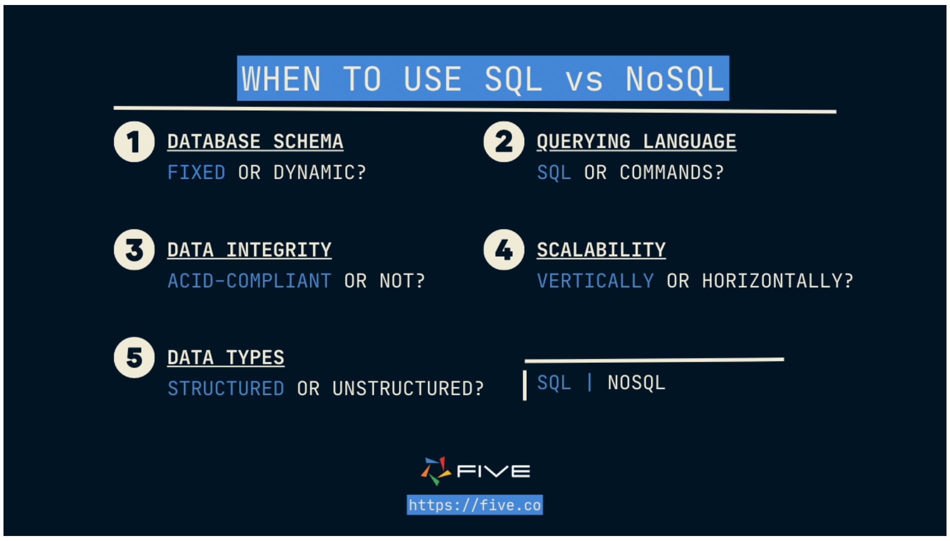
- **Wide-Column:** Store data in a table-like structure, where each row can have a different set of columns. Examples of wide-column-based NoSQL databases include Apache Cassandra, Apache HBase, and ScyllaDB.
- **Graph:** Store data in nodes and edges, where nodes represent entities and edges represent the relationships between those entities. Neo4j is a popular graph-based NoSQL database that is often used for social networking, recommendation engines, and fraud detection.

Amongst the NoSQL databases mentioned, MongoDB is the most popular. It was first released in 2009. For comparison, MySQL, the world's most popular open-source relational database management system, was first released fourteen years earlier, in 1995.

When to Use SQL vs NoSQL? Five Important Criteria

Fundamentally, both SQL and NoSQL databases are used to store data. They also allow us to retrieve it whenever we need it.

But deep down, there are some key differences between them. So what's the difference between SQL vs NoSQL? Let's try to see in what key areas these databases differ.



1. Schema

SQL and NoSQL databases differ fundamentally in the way they store data.

As we have seen earlier, SQL databases store data in tables that follow a predefined schema. This helps to optimize storage and ensure data integrity, but it also makes SQL databases more rigid. Changing the schema or a data type can be a complex and time-consuming process.

NoSQL databases, on the other hand, follow a dynamic schema that doesn't require a predefined data structure. As a result, NoSQL databases offer a greater level of flexibility and are easier to adapt and change.

For those who think that this sounds like a clear win for NoSQL databases, don't be fooled. Defining how your database stores data is one of the most important decisions in application development. SQL databases are more rigid, but it also means that you have to carefully think about the schema before writing your first line of code. This can lead to cleaner code and less work down the line.

2. Querying Language

At the heart of relational databases is **Structured Query Language (SQL),** which is a standardized programming language used to query, manipulate, and change data.

Once you've defined the schema for your database, you will use SQL to insert, query, and manipulate the data. Since NoSQL databases come in a variety of flavors, there is no standard language used for querying purposes. MongoDB uses the MongoDB Query Language (MQL), while on the other hand, Redis typically uses commands based on the Redis command set.

3. Data Integrity

SQL databases enforce data integrity rules. This helps prevent the storage of inconsistent data.

If you define a unique constraint for a particular column in a table, the database will make sure that there aren't two rows with the same value for that column. Making sure data integrity rules are being followed comes at a price.

The database has to do extra work to enforce these rules. This problem becomes amplified when you have large volumes of data.

NoSQL databases, on the other hand, tend to be more fluid compared to SQL databases when it comes to data integrity. They don't have strict data integrity rules. These kinds of databases prioritize scalability and performance over data integrity.

4. Scalability

What happens when you start having data in the order of tens of terabytes? You need to scale to adapt to the increasing workload and demands. Otherwise, your system will perform poorly.

SQL databases primarily scale vertically, which means you add more computing resources (CPU, RAM, and storage) to a single database server. It is very difficult to scale SQL databases horizontally. But some clever people have found ways of scaling SQL databases horizontally as well. Here is a great blog post written by Vamsi Ponnekanti, a software engineer at Quora, on how they scaled their MySQL database horizontally.

NoSQL databases are primarily designed to scale horizontally, meaning that they use multiple nodes in a cluster to handle increased workloads. This is one of the advantages of NoSQL databases.

5. Data Types

What type of data are you planning to store? Traditional SQL databases work best for structured data, which is typically found in applications, such as customer relationship management systems (CRM), inventory management systems or other transactional systems. The data is structured and follows a defined schema.

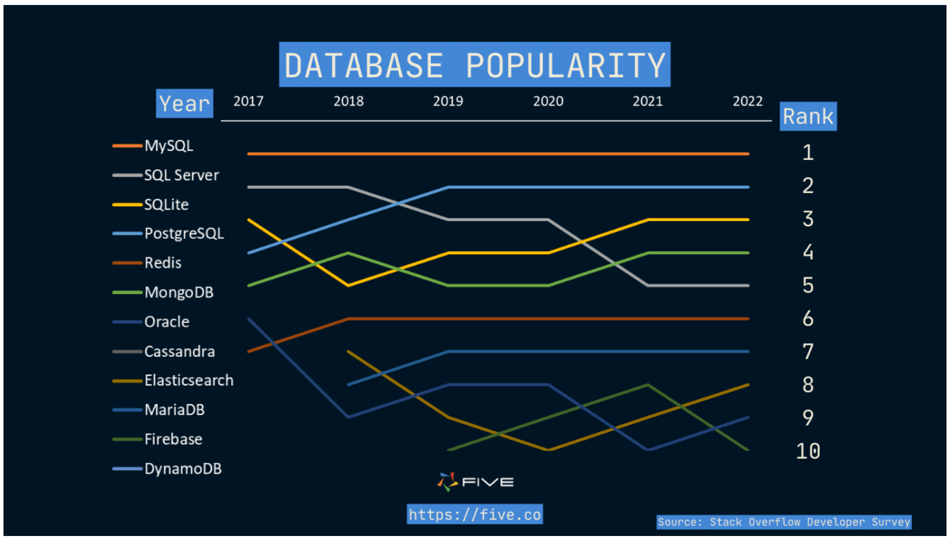
NoSQL databases work best when you require a dynamic schema for unstructured data. Media, text, audio, video or image data are all examples of unstructured data.

When To Use SQL vs NoSQL: The Pros and Cons

Because NoSQL databases emerged recently, it may seem like they are a replacement for SQL databases. But nothing can be further from the truth.

SQL databases continue to be used by enterprises despite being decades old. Old is not necessarily bad in this case, as opposed to aging. The fact that the technology has been around for decades means it has gotten a lot more mature.

Here at Five, for example, we use MySQL (a relational database) as the underlying database for all applications built with our low-code IDE. When you start developing a new application in Five, you typically set out by creating the tables for your MySQL database. We chose MySQL because it's the most popular database among developers, has a great support community, and is cheap, reliable, and highly performant.



Both SQL and NoSQL databases have their strengths and weaknesses.

Pros and Cons of SQL vs NoSQL Databases

It's crucial to understand SQL and NoSQL databases' differences in order to choose the right database for your needs. As we have seen earlier, these databases are designed to solve different problems.

SQL databases try to ensure data integrity while avoiding data duplication. NoSQL databases, on the other hand, are about scaling to a larger demand.

In the next subsections, we will look at the key criteria you need to answer the question when to use SQL vs NoSQL.

When To Use SQL Over NoSQL

You should choose a SQL database when:

1. Your Application Requires Transactions.

A transaction is an indivisible unit of work that should be treated as a single, discrete operation. Either all the changes made by the transaction should be applied, or none of them should be applied.

For example, in banking applications, we use a database transaction to do money transfers. Technically speaking, money transfers involve more than one database operation. Money has to be deducted from the sender and credited to the receiver. But with transactions, we think about the money transfer as if it’s just a single operation.

SQL databases are well suited for handling transactions, and they have a special property called ACID:

- **Atomicity:** Ensures either all of the operations or nothing is completed.
- **Consistency:** A transaction should take the database from one consistent state to another consistent state.
- **Isolation:** Two transactions should be able to work independently and in parallel. They should not interfere with one another.
- **Durability:** Once a transaction completes its changes should be persisted.

2. You have complex queries: SQL databases have a capability that allows you to perform complex queries, such as joining multiple tables, aggregating data, filtering, and sorting. Here’s what Deexith Reddy, data engineer at Slalom, has to say about the ability of SQL databases to perform complex queries:

“This is invaluable for applications that need to perform in-depth data analysis or generate intricate reports. The ability to leverage complex queries makes SQL databases well-suited for business intelligence and analytical applications.”

3. Your Application Will Not Grow Exponentially

SQL is really great for your data storage needs if you’re not working with an excessively large volume of data. SQL will ensure the integrity and consistency of your data. It’s only when you have large workloads and demands that it performs poorly.

Unless you start to get into the terabyte realm, SQL should be more than enough. In fact, most of the world’s most popular apps we’re using, like Twitter, Facebook, and Instagram, started on a SQL database. But over time, these apps started growing exponentially. It was at that point that they started looking into other database options that could handle really large workloads.

For most application development projects, a SQL database is more than enough. Unless you store time-series data or expect millions of users worldwide to store data in your application, SQL databases will offer your application plenty of room to grow.

When To Use NoSQL Over SQL

You should choose a NoSQL database when:

1. You Need Flexibility: If you’re looking for flexibility without adhering to a fixed structure, then you should use a NoSQL database. This means that you can add new fields or change the structure of the data without needing to modify the schema or migrate the data. These databases can be advantageous for applications with rapidly evolving data requirements.

2. You Have a Very Large Dataset: As we've seen earlier, NoSQL databases scale horizontally easily by distributing data across multiple nodes in a cluster. This enables NoSQL databases to handle very large datasets and high-velocity read and write operations, making them a good fit for big data applications, real-time analytics, and high-traffic web applications.

3. You Need Very High Availability: Even though both SQL and NoSQL databases can be configured for high availability, NoSQL databases are generally considered to be a better choice for applications that require high availability. NoSQL databases are often designed with features like automatic failover, data replication, and sharding, which can help ensure that data is always available, even in the event of a node failure or network outage.

4. You Can Accept That Some Data Gets Lost or is Stored in a Non-ACID-Compliant Way. Not all data is as critical as banking account balances or other commercial transactions. Some data, such as likes, upvotes, or reviews can be stored in NoSQL databases and at the risk of data loss.

SQL Database Use Case: Manipulating Relational Databases

Customer Relationship Management (CRM)

One of the key applications of SQL databases is in managing customer data for CRM systems. SQL databases provide an efficient and reliable way to store and retrieve customer information, track interactions, analyze buying patterns, and provide personalized experiences. With SQL, businesses can ensure data integrity, scalability, and ease of integration with other systems.

E-commerce and Inventory Management

SQL databases are well-suited for e-commerce platforms and inventory management systems. They can handle complex product catalogs, track inventory levels, and process transactions with ease. SQL enables efficient querying to retrieve product information, manage orders, and provide seamless shopping experiences to customers. Businesses can leverage SQL databases to optimize their inventory systems and enhance overall operational efficiency.

Financial Systems and Reporting

SQL databases play a crucial role in financial systems, powering applications for banking, accounting, and financial reporting. They provide a secure and reliable storage mechanism for financial data, enabling efficient transaction processing and complex query capabilities. SQL databases ensure accuracy, consistency, and compliance in financial operations, supporting critical functions such as generating financial reports and conducting audits.

Content Management Systems (CMS)

SQL databases form the backbone of content management systems, facilitating efficient storage and retrieval of website content. From blog posts to multimedia assets, SQL databases offer the flexibility to handle diverse content types, enable robust search functionality, and support multi-user collaboration. CMS powered by SQL databases provide a reliable foundation for managing and delivering content to end-users.

Data Analytics and Business Intelligence

SQL databases serve as a foundation for data analytics and business intelligence applications. They enable efficient querying, aggregation, and analysis of large datasets, helping organizations gain valuable insights from their data. SQL’s declarative nature and rich set of analytical functions make it a powerful tool for driving data-driven decision-making. SQL databases can support complex analytical queries, allowing businesses to uncover trends, identify opportunities, and make informed decisions.

NoSQL Database Use Case: Large Scale and Unstructured Data

Big Data and Real-Time Analytics

NoSQL databases excel in handling massive volumes of data in real-time. They are well-suited for use cases that involve storing and processing large datasets, such as sensor data, social media feeds, log files, and machine-generated data. NoSQL databases can easily scale horizontally to accommodate the growing data volume and provide low-latency access for real-time analytics, enabling businesses to derive valuable insights and make data-driven decisions.

Highly Scalable Web Applications

NoSQL databases are widely used in building scalable web applications that require high performance and flexibility. These databases offer horizontal scalability, allowing developers to distribute data across multiple nodes and handle increased traffic and user loads effortlessly. NoSQL databases are well-suited for use cases like user profiles, session management, content management systems, and e-commerce platforms, where the ability to scale and handle unpredictable workloads is critical for delivering a seamless user experience.

Conclusion: When to Use SQL vs NoSQL

In this article, we tried to answer the question of when to use SQL vs NoSQL. We introduced these two types of databases and analyzed the pros and cons of SQL and NoSQL.

To sum up: SQL databases work best when dealing with structured data that needs to be stored efficiently and in an ACID-compliant manner. NoSQL databases serve a different use case. They are built for scale and for dealing with unstructured data. For most use cases, SQL databases will work just fine and they are almost always a good place to start. No one ever got fired for choosing MySQL.

But if you do understand the differences between SQL and NoSQL well, then you can choose the right database for your needs. It's worth mentioning that in the real world, things are not as black and white as portrayed in this article. There is no definite answer as to when to use SQL vs NoSQL, as technologies are evolving. SQL databases now have support for storing JSON data. On the other hand, NoSQL databases such as MongoDB now have support for multi-document ACID transactions.

Recent Posts

The 3 Best Cloud Development Environments

3 Apr, 2024

White Label SaaS: Made Easy

27 Mar, 2024

8 SaaS Ideas You Can Start Today

27 Mar, 2024

Start Developing Your First Application!

Get Started For Free Today

Sign Up Free

Book a demo →

Low-Code For Real Developers

The Next Generation Of Software Engineering, Simplified

Use Cases

Pricing

Case Studies

Blog

Themes

About

Careers

Contact

Corporate Social Responsibility

Community

Free Access

Privacy Policy

Terms of Service