**MongoDB**

MongoDB is a document-oriented, open-source database program that is platform-independent. MongoDB, like some other NoSQL databases (but not all!), stores its data in documents using a JSON structure. This is what allows the data to be so flexible and not require a schema.

Some of the more important features are:

* You have support for many of the standard query types, like matching (==), comparison (<, >), or even regex
* You can store virtually any kind of data - be it structured, partially structured, or even polymorphic
* To scale up and handle more queries, just add more machines
* It is highly flexible and agile, allowing you to quickly develop your applications
* Being a document-based database means you can store all the information regarding your model in a single document
* You can change the schema of your database on the fly
* Many relational database functionalities are also available in MongoDB (e.g. indexing)

## What is NoSQL Database

NoSQL or “non-SQL” a non-structured database.

It provides a facility for storage and retrieval of data using fields.

While in SQL the data stores in a tabular form.

Companies are using NoSQL database in big data and real-time applications.

NoSQL offers “eventual consistency” so that it may not meet the real-time application requirements. Still, its use to merits over relational databases.

**WHY NOSQL**

* Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
* Long gone is the twelve-to-eighteen month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.
* Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users.
* Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

## NoSQL Database Types

* **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.(MongoDB)
* **Graph stores** are used to store information about networks of data, such as social connections. Graph stores include Neo4J and Giraph.
* **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value. Examples of key-value stores are Riak and Berkeley DB. Some key-value stores, such as Redis, allow each value to have a type, such as 'integer', which adds functionality.
* **Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

## Features of NoSQL/mongoDB

## *Dynamic Schemas*

## Relational databases require that schemas be defined before you can add data. For example, you might want to store data about your customers such as phone numbers, first and last name, address, city and state – a SQL database needs to know what you are storing in advance.

## This fits poorly with [agile development](https://www.mongodb.com/agile-development) approaches, because each time you complete new features, the schema of your database often needs to change. So if you decide, a few iterations into development, that you'd like to store customers' favorite items in addition to their addresses and phone numbers, you'll need to add that column to the database, and then migrate the entire database to the new schema.

If the database is large, this is a very slow process that involves significant downtime. If you are frequently changing the data your application stores – because you are iterating rapidly – this downtime may also be frequent. There's also no way, using a relational database, to effectively address data that's completely unstructured or unknown in advance.

NoSQL databases are built to allow the insertion of data without a predefined schema. That makes it easy to make significant application changes in real-time, without worrying about service interruptions – which means development is faster, code integration is more reliable, and less database administrator time is needed. Developers have typically had to add application-side code to enforce data quality controls, such as mandating the presence of specific fields, data types or permissible values. More sophisticated NoSQL databases allow validation rules to be applied within the database, allowing users to enforce governance across data, while maintaining the agility benefits of a dynamic schema.

B) Auto-Sharding

Because of the way they are structured, relational databases usually **scale vertically** – *a single server has to host the entire database to ensure acceptable performance for cross- table joins and transactions*.

This gets expensive quickly, places limits on scale, and creates a relatively small number of failure points for database infrastructure.

**The solution to support rapidly growing applications is to *scale horizontally*, by adding servers instead of concentrating more capacity in a single server.**

**Manual Sharding in relational databases and the problems associated with it.**

**‘Sharding'** a database across many server instances can be achieved with SQL databases, *but usually is accomplished through SANs and other complex arrangements for making hardware act as a single server.*

Because the database does not provide this ability natively, development teams take on the work of deploying multiple relational databases across a number of machines. Data is stored in each database instance autonomously. Application code is developed to distribute the data, distribute queries, and aggregate the results of data across all of the database instances. Additional code must be developed to handle resource failures, to perform joins across the different databases, for data rebalancing, replication, and other requirements. Furthermore, many benefits of the relational database, such as transactional integrity, are compromised or eliminated when employing manual sharding.

**AutoSharding in NoSQL DATABASES**

NoSQL databases support **auto-sharding**, meaning that they natively and automatically spread data across an arbitrary number of servers, without requiring the application to even be aware of the composition of the server pool.

Data and query load are automatically balanced across servers, and when a server goes down, it can be quickly and transparently replaced with no application disruption.

[Cloud computing](https://www.mongodb.com/big-data-explained) makes this significantly easier, with providers such as Amazon Web Services providing virtually unlimited capacity on demand, and taking care of all the necessary infrastructure administration tasks. Developers no longer need to construct complex, expensive platforms to support their applications, and can concentrate on writing application code. Commodity servers can provide the same processing and storage capabilities as a single high-end server for a fraction of the price.

## Replication

Most NoSQL databases also support automatic database replication to maintain availability in the event of outages or planned maintenance events. More sophisticated NoSQL databases are fully self-healing, offering automated failover and recovery, as well as the ability to distribute the database across multiple geographic regions to withstand regional failures and enable data localization. Unlike relational databases, NoSQL databases generally have no requirement for separate applications or expensive add-ons to implement replication.

**NoSQL vs. SQL Summary**

|  |  |  |
| --- | --- | --- |
|  | SQL Databases | NoSQL Databases |
| Types | One type (SQL database) with minor variations | Many different types including key-value stores, [document databases](https://www.mongodb.com/document-databases), wide-column stores, and graph databases |
| Development History | Developed in 1970s to deal with first wave of data storage applications | Developed in late 2000s to deal with limitations of SQL databases, especially scalability, multi-structured data, geo-distribution and agile development sprints |
| Examples | MySQL, Postgres, Microsoft SQL Server, Oracle Database | MongoDB, Cassandra, HBase, Neo4j |
| Data Storage Model | Individual records (e.g., 'employees') are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., 'manager,' 'date hired,' etc.), much like a spreadsheet. Related data is stored in separate tables, and then joined together when more complex queries are executed. For example, 'offices' might be stored in one table, and 'employees' in another. When a user wants to find the work address of an employee, the database engine joins the 'employee' and 'office' tables together to get all the information necessary. | Varies based on database type. For example, key-value stores function similarly to SQL databases, but have only two columns ('key' and 'value'), with more complex information sometimes stored as BLOBs within the 'value' columns. Document databases do away with the table-and-row model altogether, storing all relevant data together in single 'document' in JSON, XML, or another format, which can nest values hierarchically. |
| Schemas | Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline. | Typically dynamic, with some enforcing data validation rules. Applications can add new fields on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically. |
| Scaling | Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required, and core relational features such as JOINs, referential integrity and transactions are typically lost. | Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary. |
| Development Model | Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database) | Open-source |
| Supports multi-record ACID transactions | Yes | Mostly no. MongoDB 4.0 and beyond support multi-document ACID transactions. [Learn more](https://www.mongodb.com/transactions) |
| Data Manipulation | Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE… | Through object-oriented APIs |
| Consistency | Can be configured for strong consistency | Depends on product. Some provide strong consistency (e.g., MongoDB, with tunable consistency for reads) whereas others offer eventual consistency (e.g., Cassandra). |

## When you should use it

Mongo represents data as a tree. If your data is tree shaped, or can be made tree shaped, Mongo is great. If your data is a web or a network which can't be flattened out, you likely have relational data, and Mongo is perhaps not for you this time.

If you have unstructured data to store which can be represented as a series of nested lists Mongo will make your life more enjoyable.

Say you have a webpage full of widgets, and each of those widgets can contain arbitrary information. This is a semi-structured tree and Mongo might be a very good choice.

If you have big customer data to store, and each customer record contains lists of communications, subscriptions, etc, the data is tree shaped, and Mongo would again be a good chice.

If you have big data and you want to query it in interesting and complex ways, pulling useful aggregated data out the other side in suprisingly short timeframes, Mongo is perfect.

On the other hand, if your data looks like a web: comments, purchases, kittens, customers, sharks, exploding hats, etc, all linking to each other in a web, then you have relational data, and you may wish to stick with a relational database like Postgres, MySQL or MS SQL Server.