# Databases

## Introduction

A database is defined as a collection of documents, usually organized in such a way that simplifies the access to information contained in the data. For example, every single case of a webpage that allows its users to register, it most definitely makes use of at least one such database to contain their information. A Database Management System – DBMS, is consisted of one database and a number of programs that consist the interface of the database (with users or other programs). Through the DBMS it is possible to input new data, change the current or retrieve the already stored. These actions can happen either manually or automatically. The automatization of the process essentially, is what makes the proper handling of a big amount of data, possible.

The database systems came into existence to satisfy the need of a better automated process to handle all commercial data. During the 60’s, the time where the need first started to surface, a primitive stage of a database system was used where files organized in folders of an Operating System where simply stored. The ‘interface’ was a set of programs the administrators of the system had to construct themselves, a practice that resulted in many drawbacks1. Databases were therefore created to tackle the following:

* There was no known easy way to automatically check whether a registration was already in the system or not. As a result, the system could very easily end up being filled with duplicate registrations and there was an additional danger of document inconsistencies. For example, after the deletion or change of a document, other documents referring to the same subject would continue to exist in the system while being invalid.
* The lack of flexibility at accessing the data, meaning that the constant use of specific, prepared in advance programs and interfaces for every possible scenario, was not practical enough. Indeed, this was simply burdening the users with additional manual work to be done.
* The writing of programs that facilitated user access would become especially difficult when the files where the data was stored differed based on their structure. Additionally, there was no way to automatically dictate a specific type for all files· their writing was left to the administrator, something that added the parameter of human error in the calculation.
* It was considerably hard to properly adjust the automated control for data consistency. For example, if the value of an arithmetic field had to be limited in a particular range, any future changes to that range were very hard to achieve.
* The integrity of the data was doubtful, in case of a system malfunction.
* There were also problems in cases where a simultaneous update of a registration occurred. When two users updated a file at the same time, there was a danger that either update would fail and thus, one user would be left hanging. The damage however would be much greater if the update was done by more than two people. Such a case would be catastrophic, especially today where the number of people having access to a particular database are huge, thanks to the internet. If that were to continue, databases would lose any possible value and use.
* The implementation of a security system that prevented a user from having access to classified data was also hard to achieve. Especially when new programs are constantly created, this task becomes even so problematic.

All of the above were tackled through the creation of the first Database Management Systems.

## Relational Model

The structure of every database follows a database model. A database model is a collection of logical tools to describe data, the relations between data, the importance of it as well as the consistency limitations. A database model provides a method to describe the design of a database in logical level, physical level and projection level.

The Relational Mode, which was suggested by Edgar F. Codd in his “A Relational Model of Data for Large Shared Data Banks” publication in 1970, describes the database as a collection of tables consisting of the data and their in-between relations. Every table has entries of only one type and to identify the entities to which the entries refer to, special data called keys are used. Ideally, keys are the only type of data which is stored more than once in the database, but the space they take is little compared to the space that is spared through their use.

A simple example is a university database. There is one table with the personal data of the students, another one containing which student registered to which subject and one with all the subjects available. The keys in such a case are the record number for the student (ID) and a code for the subjects. The second table therefore depicts the relation of the first table with the third by matching each other’s keys.

The general idea of the relational model had a significant response since the birth of its concept in 1970. Today, the most known and used query language is SQL (Structured Query Language) which is essentially an alteration of the old relational model. A query based language is a language that allows the easy interface of users with the database through ‘asking questions’.

Some of the most renown Database Management Systems of today that follow the relational model are Oracle, MySQL, Microsoft SQL Server and PostgreSQL.

## NoSQL Databases

A NoSQL database is a database that does not follow the traditional relational model with tables. In essence, this definition identifies NoSQL databases as the file processing systems that were mentioned previously. In that regard, one could even argue they are a weird devolution, however, them resurfacing at the start of the 21st century was not done without important modifications and improvements. Their popularity is mainly attributed to their, till-recently, unsuspected usefulness at applications that need to handle a big volume of data and applications that their content is constantly changing in real-time. A characteristic example is any social network, where the storage of information belonging to millions of users is required, at a time when each and every one of them is capable of changing the content of the webpage at any given time.

NoSQL databases are useful in such applications due to their –usually- simple, unstructured design, that is especially effective at handling unstructured data like text files and multimedia3. NoSQL databases are also potent at easily parallelizing the applications built for them, something that allows the usage of many cheap -and potentially remote- computers in parallel in contrast to one, expensive computer. This renders NoSQL databases as ideal to handle Big Data.

Known examples of popular NoSQL Database Management systems are MongoDB, Cassandra, Redis and HBase.

## MongoDB

### Overview

As mentioned in the official page, MongoDB is “MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need”. MongoDB’s document model is simple for developers to learn and use, while still providing all the capabilities needed to meet the most complex requirements at any scale. They provide drivers for 10+ languages, and the community has built dozens more. An overview of the database could be this:

* MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.
* The document model maps to the objects in their application code, making data easy to work with.
* Ad hoc queries, indexing and real time aggregation provide powerful ways to access and analyze the data.
* MongoDB is a distributed database at its core, so high availability horizontal scaling and geographic distribution are built in and easy to use
* MongoDB is free to use. Previous versions (prior October 16, 2018) are published under the AGPL. All versions released after October 16 however, including fixes for prior versions, are published under the Server Side Public License (SSPL).

Subsequently, a summary of the benefits of using the database could be the following:

* High availability through built-in replication and failover
* Horizontal scalability with native sharding
* End-to-end security
* Native document validation and schema exploration with Compass
* Management tooling for automation, monitoring, and backup
* Fully elastic database as a service with built-in best practices

### Architecture

### Summary

MongoDB is designed to meet the demands of modern apps with a data platform built on three core architectural foundations:

* The document data model and MongoDB Query Language, giving developers the fastest way to innovate in building transactional, operational and analytical applications.
* A multi-cloud global database, giving developers the freedom to run their applications anywhere with the flexibility to move across private and public clouds as requirements evolve – without having to change a single line of application code.
* The MongoDB Cloud, providing a unified developer experience for modern applications than span cloud to edge, in database, search, and the data lake, backed by comprehensive & integrated developer services.

With these capabilities, the developer can build an Intelligent Operational Data Platform, underpinned by MongoDB.

As mentioned, MongoDB is built around JSON-like documents, document databases are both intuitive and flexible for developers to work with. They promise higher developer productivity, and faster evolution with application needs. As developers have experienced these benefits, the document data model has become the most popular alternative to the tabular model used by traditional relational databases. There are four primary advantages of the document data model.

1. Intuitive: Faster and Easier for Developers
   1. Documents in the database directly map to the objects in the code, so they are much more natural to work with. The following example JSON document in MongoDB demonstrates how a customer object is modeled in a single document structure with related data embedded as sub-documents and arrays. This approach collapses what would otherwise be seven separate parent-child tables linked by foreign keys in a relational database. With the document data model, there is no need to decompose data across tables, run expensive JOINs, or integrate a separate ORM layer. Data that is accessed together is stored together, so there have less code to write and the users get higher performance.
2. Flexible Schema: Dynamically Adapt to Change
   1. A document’s schema is dynamic and self-describing, so there is no need to first pre-define it in the database. Fields can vary from document to document and the developer modifies the structure at any time, allowing them to continuously integrate new application functionality, without wrestling with disruptive schema migrations. If a new field needs to be added, it can be created without affecting all other documents in the collection, without updating a central system catalog and without taking the database offline. When the developer needs to make changes to the data model, the document database continues to store the updated objects without the need to perform costly ALTER TABLE operations, update a separate ORM middleware layer, and coordinate all of these changes across multiple developer, DBA, and Ops teams. Documents allow multiple versions of the same schema to exist in the same table space. Old and new applications can co-exist. MongoDB also offers Schema Validation so it is easy can enforce standards over each document’s structure. With schema validation, control is granted to apply data governance to a document schema when the application enters production, while maintaining the benefits of a flexible data model in development.
3. Universal: JSON Documents are Everywhere
   1. Lightweight, language-independent, and human readable, JSON has become an established standard for data communication and storage. Documents are a superset of all other data models so the developer can structure data any way their application needs – rich objects, key-value pairs, tables, geospatial and time-series data, and the nodes and edges of a graph. As a result of these properties, they can serve many more classes of application with a single database. They can work with documents using a single query language, giving them a more consistent development experience however they have chosen to model their data. MongoDB stores data as JSON (JavaScript Object Notation) documents in a binary representation called BSON (Binary JSON). Unlike most databases that store JSON data as primitive strings and numbers, the BSON encoding extends the JSON representation to include additional types such as int, long, date, floating point, and decimal128. This makes it much easier for applications using MongoDB to reliably process, sort, and compare data.
4. Powerful: Serve any Workload
   1. An important difference between databases is the expressivity of the query language, richness of indexing, and data integrity controls. The MongoDB Query Language is comprehensive and expressive. Ad hoc queries, indexing, and real time aggregations provide powerful ways to access, group, transform, and analyze data. Queries can be federated across databases supporting transactional workloads and archived data in the data lake using the same MongoDB Query Language and drivers, all using a single connection string. With ACID transactions the developer can maintain the same all-or-nothing and snapshot isolation guarantees they have used to in relational databases, whether manipulating data in a single document, or with MongoDB’s scale-out architecture, across multiple documents geographically distributed in multiple shards. With strong data consistency, MongoDB eliminates the application complexity imposed by eventually consistent NoSQL systems. MongoDB’s consistency guarantees are fully tunable, allowing to balance data freshness against performance.

### Document Data in Practice

To accelerate developer productivity, MongoDB provides native drivers for all popular programming languages and frameworks. Supported drivers include Java, Javascript, C#/.NET, Go, Python, PHP, Scala, Rust and more. All supported MongoDB drivers are designed to be idiomatic for the given programming language. This makes it much more natural for developers to work with data than string-based languages like SQL, and eliminates the need for cumbersome and fragile ORM abstraction layers. The developer can also interact with MongoDB graphically using MongoDB Compass, the GUI for MongoDB. Through Compass they can explore and manipulate their data, visually create queries and aggregation pipelines from the GUI and then export them as code to their app; view and create indexes; build schema validation rules; and more. Beyond working with documents through the MongoDB drivers, the developer needs to make data accessible to multiple consumers across the organization so they can extract insights and hidden value from their data. MongoDB provides a range of visualization tools and connectors to make this straightforward:

* MongoDB Charts is the fastest and easiest way to create visualizations of richly-structured JSON data. The developer can create graphs and build dashboards, sharing them with other users for collaboration, and embed them directly into their web apps to create engaging user experiences.
* The MongoDB Connector for BI allows to use MongoDB as a data source for existing SQL-based BI and analytics platforms such as Tableau, Microstrategy, Looker, and more.
* The MongoDB Connector for Apache Spark exposes all of Spark’s libraries, including Scala, Java, Python and R. MongoDB data is materialized as DataFrames and Datasets for analysis with machine learning, graph, streaming, and SQL APIs.

To make it easy for businesses to act on data in real time, many developers are building fully reactive, event-driven data pipelines. MongoDB goes beyond many other databases with features like Change Streams that automatically detect and notify consuming applications of any data modifications in the database, while MongoDB Atlas Triggers allow to execute server-side logic in response to database events. With the MongoDB Connector for Apache Kafka, one can build robust data pipelines that move events between systems in real time, using MongoDB as both a source and sink for Kafka. The connector is supported by MongoDB and verified by Confluent. In summary, documents are the best way for developers to work with data, and MongoDB gives them the most productive and fully-featured implementation of a document database anywhere.

### Privacy and Security

With the digital economy becoming so essential for economic prosperity, it’s no surprise that governments and enterprises around the world are responding to growing public concern for the safety of personal data. MongoDB features extensive capabilities to defend, detect, and control access to data:

* Authentication.
  + Simplifying access control to the database, MongoDB offers a strong Challenge-Response mechanism based on SCRAM-256, along with integration to enterprise security infrastructure including LDAP, Windows Active Directory, Kerberos, x.509 certificates, and AWS IAM.
* Authorization.
  + Role-Based Access Control (RBAC) enables one to configure granular permissions for a user or an application based on the privileges they need to do their job.
* Auditing.
  + For regulatory compliance, security administrators can use MongoDB's native audit log to record all database activity and changes.
* Network Isolation
  + MongoDB Atlas users’ data and underlying systems are fully isolated from other users. Database resources are associated with a user group, which is contained in its own Virtual Private Cloud (VPC). Access must be granted by IP whitelisting or VPC Peering.
* Encryption Everywhere.
  + MongoDB data can be encrypted while in motion across the network, while in use in database memory, and while at rest, whether on disk or in backups.

MongoDB’s Client-Side Field Level Encryption (FLE) provides amongst the strongest levels of data privacy and security for regulated workloads. With Client-Side FLE their most sensitive data is automatically encrypted before leaving the application, and so the database only ever works with it as cipher text. Whether resident in memory, in system logs, at-rest in storage, and in backups – all protected data remains unreadable. Client-Side FLE complements existing network and storage encryption to protect the most highly classified, sensitive fields of their records without:

* Developers needing to write additional, highly complex encryption logic.
* Compromising their users’ ability to query encrypted data.
* Significantly impacting database performance.

By securing data with Client-Side FLE you can move to managed services in the cloud with greater confidence. This is because the database only works with encrypted fields, and the developer controls the encryption keys, rather than having the database provider manage the keys for them. This additional layer of security enforces an even finer-grained separation of duties between those who use the database and those who administer and manage the database. Client-Side FLE also enables to more easily comply with “right to erasure” mandates in modern privacy legislation. When a user invokes their right to erasure, the developer can simply destroy the associated encryption key and the user’s Personally Identifiable Information (PII) is rendered unreadable and irrecoverable.

### MongoDB Cloud

Modern data architectures are not limited to the transactional database. Many applications also require analytics and search functionality, which often requires teams to learn, deploy, and manage additional systems. For building mobile apps, one will need to deal with data on the device and syncing it to the backend. They may also find their self building data visualizations, writing a lot of glue code to move data between data services, or creating and operating custom data access APIs. Only through a consistent, unified experience – for both developers and the operations teams supporting them – can companies avoid a sprawl of disparate data silos, each with their own set of APIs, operational models, and security requirements. This is the issue the MongoDB Cloud is uniquely designed to solve.

In MongoDB Cloud, the database is fully integrated with other data services with automatic syncing, data-automatic syncing, data tiering, and federated query. Search indexes run alongside the database and are automatically kept in sync. Aged data can be auto-archived to cloud storage, providing fully managed data tiering while retaining access. Queries are automatically routed to the appropriate data tier without requiring to think about data movement, replication, or ETL. MongoDB Cloud can even automatically sync backend data to an embedded database on mobile devices. Across these services, MongoDB provides a consistent and elegant developer experience. From database to search to analytics on their data lake, here is a common way of working with data that simplifies development. Unlike other cloud data platforms that require to learn entirely different technologies and APIs, MongoDB Cloud presents all its services in a single system and with a consistent interface. MongoDB Cloud runs on multi-cloud infrastructure. The developer may choose underlying infra from AWS, GCP, or Azure and get a consistent experience, deploying and controlling clusters in different clouds from a single UI. Avoid lock-in or take advantage of different cloud vendor’s services with simple data portability across clouds. MongoDB Cloud is fully managed for operational simplicity. From the basics of deployment automation and monitoring to advanced features like auto-scale and intelligent performance advice, MongoDB Cloud gets operations out of their way. Clusters re fully managed, data synchronization between services is automated, and making adjustments is as easy as a button click or API call. MongoDB Cloud works with their ecosystem. Manage their infrastructure as code with Kubernetes and Terraform integrations, plug in their monitoring and alerting tools, integrate with their security environment, connect to data tools like Kafka and Spark, and work with MongoDB integrations in their usual IDEs.

# Bibliography

[1] Abraham Silberschatz, Henry F Korth, Shashank Sudarshan, et al. Database system concepts. Vol. 4.

[2] DB-Engines Ranking - popularity ranking of database management systems. URL: <https://dbengines>.

[3] Neal Leavitt. ``Will NoSQL databases live up to their promise?'' In: Computer 43.2 (2010).

[4] What is MongoDB (https://www.mongodb.com/what-is-mongodb)