

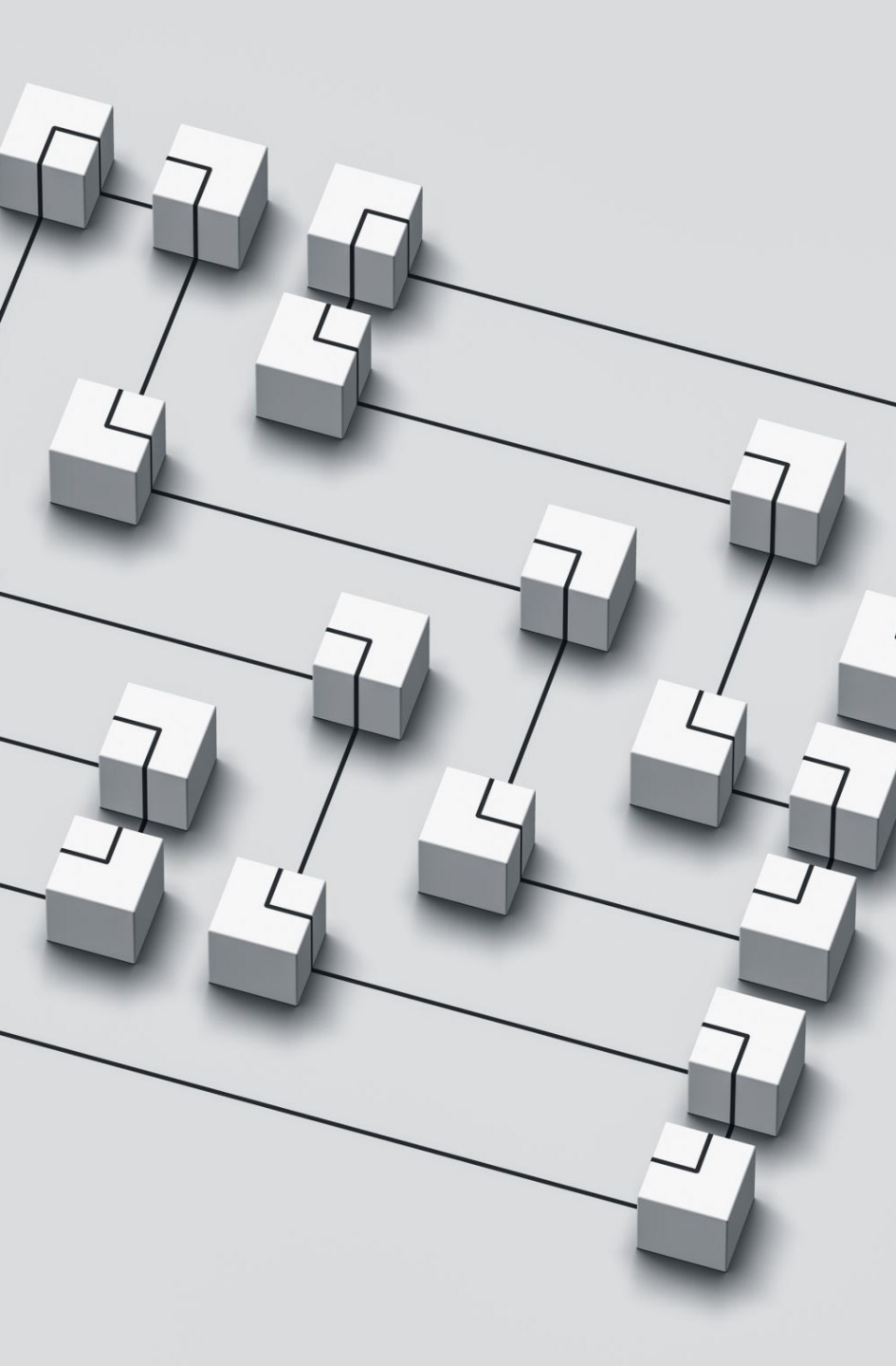


NoSQL and MongoDB

Thusithanjana Thilakarathna

Learning Outcomes

- Differentiate between relational databases and non relational databases.
- Understand the usage of NoSQL
- List different types of NoSQL databases
- Understand the usage of MongoDB
- Understand security aspects of MongoDB



Introduction to NoSQL

- NoSQL databases are tailored for specific data models.
- They have **flexible** schemas for modern applications. charts, replicas
- **Easy to develop**, with great **functionality and scalability**.
- NoSQL databases **use different data models** for data management.
- Optimized for apps **requiring large data volumes, low latency, and flexibility**.
- Achieved by relaxing some **data consistency restrictions** of other databases.

Some modern databases give up strict data accuracy for a short time so that they can handle more users and keep working even if some parts of the system fail.

Relational Databases vs Non-Relational Databases

Everyone sees the same data at the same time.

The system stays online and fast

	Relational Databases	Non-Relational Databases
Structure	Structured, with defined relationships between tables	Flexible, allowing for dynamic and changing data structures
Query Language	Uses SQL for querying data	Uses various querying languages, including JSON and BSON
Data Consistency	Enforces strict data consistency rules	Often prioritizes availability and partition tolerance over consistency
Popularity	Widely used and has a long history in the tech industry	Gained popularity with the rise of big data and modern web applications
Transactions and Operations	Well-suited for complex transactions and operations	Often faster and more scalable than relational databases
Purpose	Best suited for structured data and complex queries	Best suited for unstructured data and high scalability
Scalability	Vertical scaling is more common, but limited <small>Making a single server more powerful</small>	Horizontal scaling is more common, and highly scalable <small>Adding more servers to share the load.</small>
Examples	MySQL, Oracle, Microsoft SQL Server	MongoDB, Cassandra, Couchbase
Data Volume	Good for small to medium data volumes	Good for large data volumes
Cost	Relatively expensive	Relatively cheap
Use Cases	Financial institutions, transactional systems, inventory management	Big data, IoT, content management, cloud computing

vertical horizontal

add more resources (ex -add more ram)

horizontal horizontal

divide work load in multiple machines

Example



In a relational database, a book record is often dissembled (or “normalized”) and stored in separate tables, and relationships are defined by primary and foreign key constraints. In this example, the Books table has columns for ISBN, Book Title, and Edition Number, the Authors table has columns for AuthorID and Author Name,

and finally, the Author-ISBN table has columns for AuthorID and ISBN. The relational model is designed to enable the database to enforce referential integrity between tables in the database, normalized to reduce the redundancy, and generally optimized for storage.

Data is split across multiple tables to avoid repetition and ensure data accuracy

Books Table		
ISBN	Title	Edition
123-A	"Data Structures"	2

Authors Table	
AuthorID	Author Name
1	Alice Johnson

Author-ISBN Table	
AuthorID	ISBN
1	123-A



In a NoSQL database, a book record is usually stored as a [JSON](#) document. For each book, the item, ISBN, Book Title, Edition Number, Author Name, and AuthorID are stored as attributes in a single document. In this model, data is optimized for intuitive development and horizontal scalability.

All related data is stored in one document, often as a JSON object.

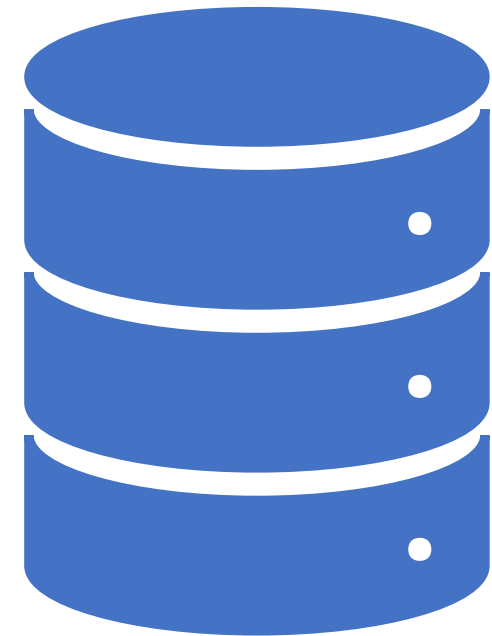
```
{
  "ISBN": "123-A",
  "Title": "Data Structures",
  "Edition": 2,
  "Authors": [
    {
      "AuthorID": 1,
      "AuthorName": "Alice Johnson"
    }
  ]
}
```

Why should you use a NoSQL database?

NoSQL databases are a great fit for many modern applications such as **mobile**, **web**, and **gaming** that require flexible, scalable, high-performance, and highly functional databases to provide great user experiences.

- **Flexibility**: NoSQL databases generally **provide flexible schemas that enable faster and more iterative development**. The flexible data model makes NoSQL databases ideal for **semi-structured and unstructured data**.
- **Scalability**: NoSQL databases are generally designed to **scale out by using distributed clusters of hardware instead of scaling up by adding expensive and robust servers**. Some cloud providers handle these operations behind-the-scenes as a fully managed service.
- **High-performance**: NoSQL database are optimized for **specific data models** and access patterns that enable higher performance than trying to accomplish similar functionality with relational databases.
- **Highly functional**: NoSQL databases **provide highly functional APIs and data types** that are purpose built for each of their respective data models.

scale horizontally using multiple machines (cheaper and more flexible), and cloud services often automate this process to make it easier for developers.





Types of NoSQL Databases

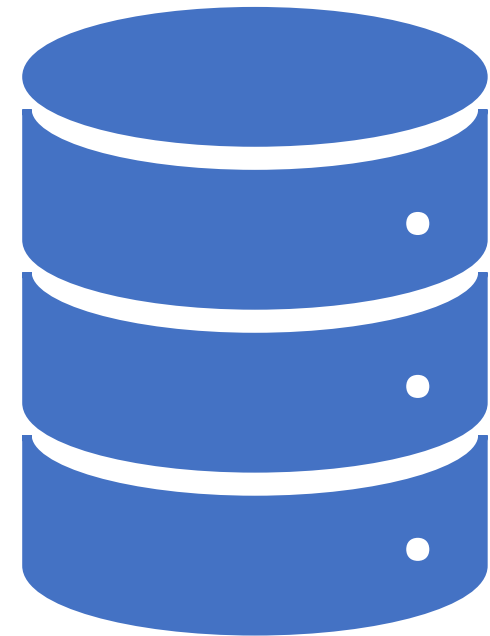
- Key-value
- Document
- Graph
- In-memory
- Wide-column stores
- Object-Oriented

Key-value Databases

- Key-value databases are designed for **horizontal scaling** and high partitioning capabilities.
- **Key-value** databases excel in use cases such as **gaming**, **ad tech**, and **IoT**, where quick data access and update is important.


Use Cases

- **Gaming**: where **quick and efficient data access and updates** are critical. Examples include **leaderboards**, **in-game purchases**, and **user profiles**.
- **Ad tech**: to store and **retrieve user data in real-time**. This allows for targeted advertising and personalized user experiences.
- **IoT**: which **generate large volumes of data** that need to be processed quickly.
- **Distributed caching**: distributed caching systems to improve application performance and **reduce database load**.
- **E-commerce**: to **store and retrieve product information**, **user preferences**, and **shopping cart data** in real-time.
- **Social networks**: to **store user profiles**, **friend lists**, and **activity feeds**.



Document Databases

- In applications, data is often represented as **objects** or **JSON**-like documents.
- Document databases **use the same document model as application code.**
- The **flexible and semi structured** nature of documents allows for easy evolution.
- Document model works well with **catalogs, user profiles, and CMS.**
- Amazon **DynamoDB** and **MongoDB** are popular document databases.
- Document databases offer **powerful and intuitive APIs for flexible development.**

 easy-to-understand and easy-to-use programming

Document Database Use Cases

- **Content management systems:** Document databases are commonly used in content management systems to store and manage content, such as articles, blog posts, and multimedia files.
- **Product catalogs:** It can be used to store product information, including product names, descriptions, images, and pricing information. This information can be easily accessed and updated as needed.
- **User profiles:** It can be used to store and manage user profiles, including personal information, preferences, and activity history. This information can be easily updated and queried to provide a personalized user experience.
- **E-commerce:** It can be used to store and manage product inventory, customer orders, and transaction history. This information can be easily queried and updated to provide a seamless e-commerce experience.
- **IoT:** It can be used to store and manage sensor data from Internet of Things (IoT) devices. This information can be easily analyzed and visualized to monitor device performance and detect anomalies.

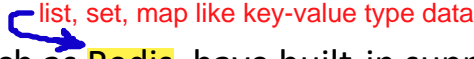
Graph Databases

google map

- Graph databases use a **graph data model** to represent and store data.
- This data model consists of **nodes (entities) and edges (relationships)**.
- Graph databases are **optimized for working with complex, interconnected data**.
- They are particularly useful for applications that involve **analyzing relationships between data points**.
- Graph databases can **scale horizontally** to **handle massive datasets**.
- They are commonly used in fields such as **social media, e-commerce, and recommendation systems**.

store relational data,

In-memory Databases

- In-memory databases are optimized for **handling large amounts of data in RAM**, which allows for **faster data processing** and **querying**.
- They are often used for **real-time applications** that require **very low latency and high throughput**, such as **financial trading platforms or online gaming**.
- In-memory databases can also be used in conjunction with disk-based databases as a **caching layer to improve performance for frequently accessed data**.
- Some in-memory databases, such as **Redis**, have built-in support for advanced data structures like lists, sets, and maps, making them well-suited for **complex data processing tasks**.

- In-memory databases typically **require more memory and may be more expensive than disk-based databases** due to the cost of RAM. However, their **fast performance and ability to handle high volumes of data** can make them a cost-effective solution for certain use cases.
- In-memory databases are used for **modern, microservices applications**
- Tinder **relies on in-memory data stores for real-time response**

Wide Column Stores / column-family stores

- Wide-column stores, also known as column-family stores, are designed to store and manage large amounts of structured data.
- The data is organized into columns, which can be grouped together into column families.
- Column families can have different schema and access patterns, providing flexibility for different use cases.
- Wide-column stores are often used for time-series data, as well as for storing and analyzing large datasets in real-time.
- Popular wide-column stores include Apache Cassandra, Apache HBase, and Google Bigtable.
- These databases are optimized for read and write performance, as well as for high availability and fault tolerance.
- They are also designed to scale horizontally, meaning that they can handle large volumes of data across multiple nodes.
- Wide-column stores are used in a variety of industries, including finance, e-commerce, and social media.
- Some common use cases include ad targeting, fraud detection, and content management systems.
- The data stored in wide-column stores is often accessed using SQL-like languages or APIs.

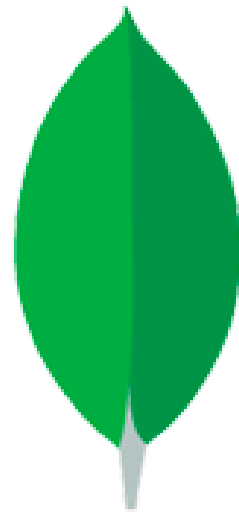
Difference Between Wide-column Stores and Relational Databases

- Data model: Wide-column stores use a column-family data model while relational databases use a table-based data model.
- Schema flexibility: Wide-column stores have flexible schema, meaning that each row in a table can have a different set of columns, whereas relational databases have a fixed schema.
- Scalability: Wide-column stores are designed to scale horizontally by adding more nodes to a cluster, while relational databases are typically scaled vertically by adding more powerful hardware.
- Performance: Wide-column stores are optimized for read-heavy workloads and can perform very well on large-scale data sets, whereas relational databases are better suited for transactions and consistency.
- Query language: Wide-column stores usually use a proprietary query language specific to the database, while relational databases use SQL.
- Use cases: Wide-column stores are well-suited for applications with large-scale data and complex queries, such as big data analytics and real-time data processing, while relational databases are typically used for transactional applications such as e-commerce, financial systems, and human resources management.

Object-Oriented Databases

this process is serialization and de-serialization
convert object into byte stream and vice versa

- Object-Oriented Databases (OODBMS) store objects directly, rather than in tables.
- They allow for complex data structures and support inheritance and polymorphism.
- OODBMS provide efficient and flexible access to complex data, ideal for applications with complex data needs.
- They support complex querying and allow for fine-grained access control.
- OODBMS are popular in industries like finance, telecommunications, and aerospace.
- Examples: db4o, Versant, Objectivity/DB, Apache Cassandra

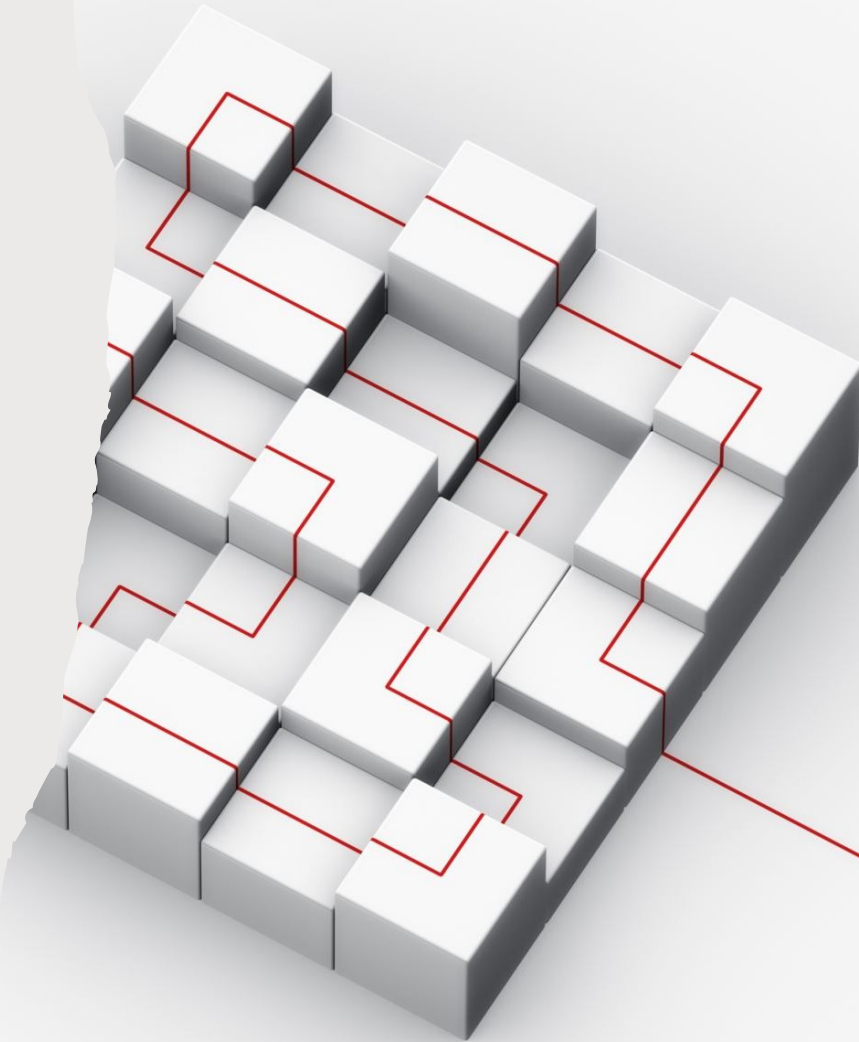


mongoDB®

Introduction to MongoDB

- MongoDB is a document-oriented NoSQL database system that stores data in a flexible, **JSON**-like format called **BSON**.
- This allows for **dynamic schema design** and **easy integration with object-oriented programming languages**.
- Unlike traditional relational databases, MongoDB uses a **distributed architecture** that enables it to **scale horizontally** across multiple servers or clusters.

[fault tolerance facility](#)



Key Features of MongoDB

- **Dynamic schema:** MongoDB has a **flexible schema**, which allows you to **store data of different types** and structures in the same collection. This makes it easier to work with data that doesn't fit neatly into a fixed schema.
- **High scalability:** MongoDB is designed to be highly **scalable, both vertically and horizontally**. You can **easily scale up or down by adding or removing nodes to your cluster**, depending on your needs.
- **High availability:** MongoDB provides **high availability through replication and automatic failover**. You can configure your cluster to **automatically promote a secondary node to become the primary node in the event of a failure**.
- **Rich query language:** MongoDB supports a rich query language that allows you to **perform complex queries on your data**. You can use **aggregation, indexing**, and **full-text search** to analyze and retrieve data.
- **Flexible data model:** MongoDB allows you to **store data of different types and structures in the same collection**. This makes it easy to work with data that doesn't fit neatly into a fixed schema.
- **MapReduce:** MongoDB provides **built-in support for MapReduce**, which is a programming model for **processing large data sets**. This allows you to **perform complex data transformations on large volumes of data**.

Scalability

Primary - Handles writes and can serve reads

Secondary - Copies data from primary, serves reads

Replica - Set Group of servers that stay in sync

Sharding

- MongoDB provides a built-in sharding feature.
- Sharding involves partitioning data across multiple servers, called shards.
- Sharding divides data into smaller chunks and distributes them across servers.
- Each server in the cluster stores a subset of the data.
- The shards are distributed based on a predefined shard key.
- This allows MongoDB to scale horizontally by adding more servers to the cluster as needed.

Sharding is a method to split large data into smaller parts and spread it across multiple servers.

Replica Sets

- MongoDB replica sets allow you to create a group of servers.
- Each server in the replica set stores a copy of the data.
- The replica set can elect a primary server, which serves all write requests.
- The other servers in the replica set serve as secondary servers, which replicate the data from the primary server.
- This allows MongoDB to scale horizontally by adding more secondary servers to the replica set as needed.

A replica set is a group of MongoDB servers that all have the same data.

Distributed Queries

- MongoDB supports distributed queries.
- Distributed queries allow you to run a single query across multiple servers in a sharded cluster.
- This is useful when you need to aggregate data from multiple shards.
- Distributed queries allow MongoDB to scale horizontally by distributing the query workload across multiple servers.

Distributed queries let MongoDB run a single query across multiple servers in a sharded cluster

Availability

If the primary server fails, the set can automatically elect a new one.

Replica Sets: Group of **servers where each stores a copy of the data**, allows automatic election of new primary server if one fails.

Read Preference Modes: Configures **read operations to distribute workload across multiple servers** and continue even if primary server fails. Determines which server (primary or secondary) handles read operations. Helps balance the workload.

Automatic Failover: **Elected new primary server automatically if the current one fails**, ensuring write operations can continue. If the primary server goes down, a new one is automatically elected.

Cluster Manager: **Monitors health of MongoDB cluster and performs recovery actions**, such as removing failed servers and replacing them with new ones.

Monitors the health of all servers in the MongoDB cluster. Can remove failed servers and add new ones as needed.

CAP theorem

Consistency Consistency means that all clients **see the same data at the same time**, no matter which node they connect to. For this to happen, whenever **data is written to one node, it must be instantly forwarded or replicated to all the other nodes** in the system before the write is deemed 'successful.'

Availability Availability means that **any client making a request for data gets a response, even if one or more nodes are down**. Another way to state this—all working nodes in the distributed system return a valid response for any request, without exception.

Partition tolerance A partition is a **communications break within a distributed system—a lost or temporarily delayed connection between two nodes**. Partition tolerance means that the cluster must continue to work despite any number of communication breakdowns between nodes in the system.

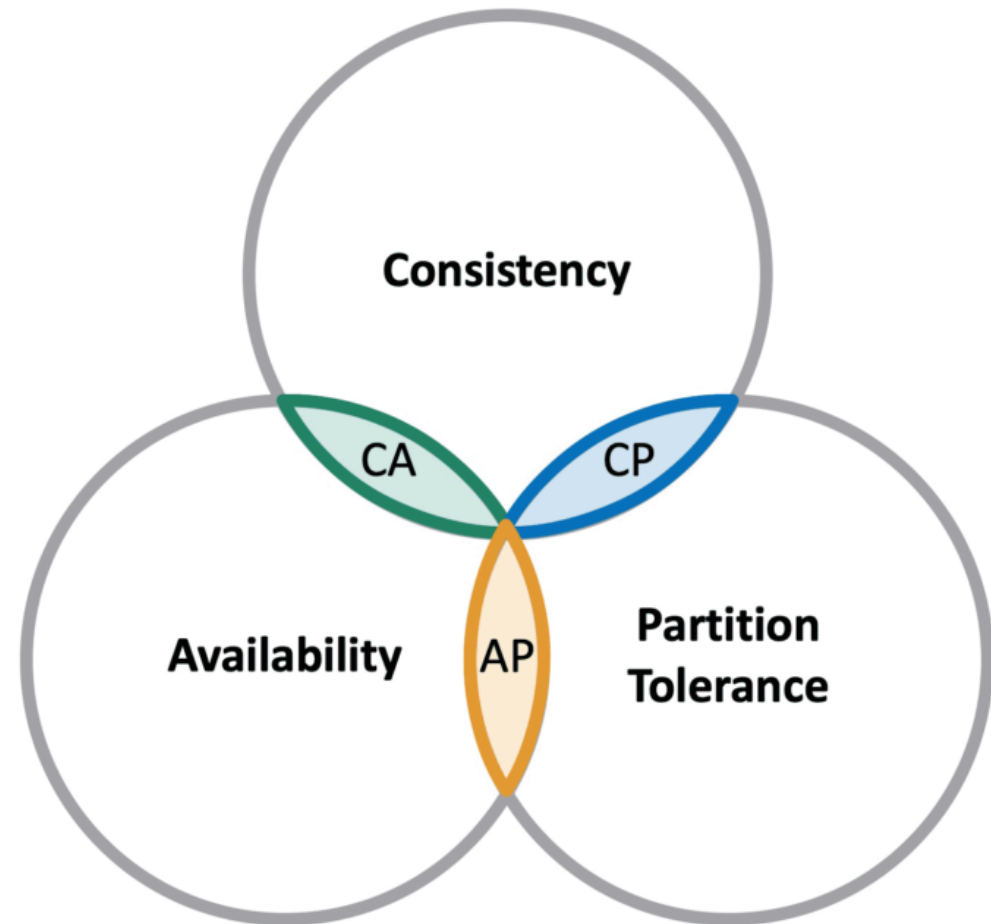


partition is when some servers can't talk to each other due to a network issue or delayed connection. Partition tolerance means the system can keep running, even if some parts can't communicate for a while.

you can't guarantee all 3 properties at once
you can guarantee only 2 of them

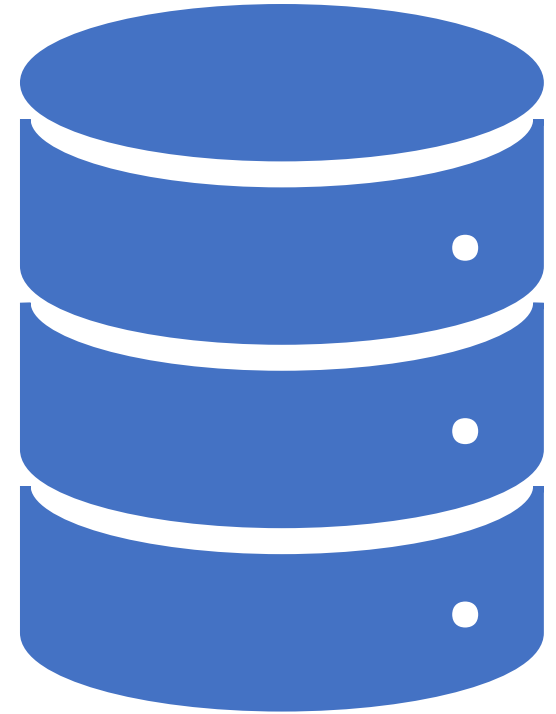
CAP theorem NoSQL database types

- **CP database:** A CP database delivers consistency and partition tolerance at the expense of availability. When a partition occurs between any two nodes, the system has to shut down the non-consistent node (i.e., make it unavailable) until the partition is resolved.
- **AP database:** An AP database delivers availability and partition tolerance at the expense of consistency. When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others. (When the partition is resolved, the AP databases typically resync the nodes to repair all inconsistencies in the system.)
- **CA database:** A CA database delivers consistency and availability across all nodes. It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance.



MongoDB and the CAP theorem

- MongoDB is a **CP** data store, prioritizing consistency over availability when resolving network partitions.
- **Each replica set has one primary node for write operations and secondary nodes that replicate the primary node's operation log.**
- By default, clients read from the primary node, but can specify a read preference to read from secondary nodes.
- **If the primary node fails, the secondary node with the latest operation log becomes the new primary node.**
- Clients can't make write requests during this interval, ensuring data consistency across the network.



Real life examples for MongoDB Applications

- E-commerce and retail
- Social networking and online communities
- Gaming and entertainment
- Healthcare and scientific research
- Financial services
- Internet of Things (IoT)
- Advertising and Marketing
- Content Management

Security

Authentication: MongoDB supports several methods of authentication, including **username/password, X.509 certificates, and LDAP**. By **default, authentication is disabled**, but it is recommended to enable it in production environments.

Authorization: MongoDB **supports role-based access control (RBAC)**, which allows administrators to define roles and privileges for users and applications. **RBAC can be used to restrict access to specific databases or collections, as well as to specific CRUD (create, read, update, delete) operations.**

Encryption: MongoDB supports encryption **at rest and in transit**. **At rest, data can be encrypted using the WiredTiger encryption engine, which encrypts data on disk using AES-256 encryption**. In transit, data can be encrypted **using SSL/TLS, which provides secure communication between clients and servers.**

Auditing: MongoDB provides auditing capabilities that **allow administrators to track user and application activity, including authentication and authorization events, database and collection access, and CRUD operations**. Audit logs can be stored in a separate MongoDB database or exported to a syslog server.

Network security: MongoDB supports several network security features, such as **IP whitelisting, SSL/TLS encryption, and LDAP/SASL authentication**. **Administrators can configure network security settings** to control access to MongoDB servers and to ensure secure communication between clients and servers.

Application security: MongoDB provides several features that help developers write secure applications, such as **parameterized queries, input validation, and SSL/TLS encryption**. Developers should follow best practices for secure coding and use MongoDB drivers that support secure communication protocols.



Getting Started with MongoDB

Will continue in the Tutorial



Thank you!

- <https://www.mongodb.com/what-is-mongodb/features>
- <https://www.mongodb.com/docs/v5.0/security/>
- <https://www.npmjs.com/package/mongodb>
- <https://medium.com/system-design-concepts/dating-application-system-design-aae411412267>
- <https://redis.io/>
- <https://www.mongodb.com/developer/products/mongodb/storing-large-objects-and-files/>
- <https://www.ibm.com/cloud/blog/sql-vs-nosql>