MongoDB

MongoDB is a powerful and flexible database system that offers scalability, performance, and flexibility, making it well-suited for a wide range of use cases, from web and mobile applications to real-time analytics and content management systems.

Here's a brief introduction to MongoDB:

- **NoSQL Database:** MongoDB falls under the category of NoSQL databases, which means it doesn't rely on the traditional tabular relations used in relational databases like SQL. Instead, MongoDB uses a flexible, document-oriented data model that can store data in JSON-like documents with dynamic schemas.
- **Document-Oriented:** In MongoDB, data is stored in documents, which are essentially JSON-like data structures composed of field-value pairs. These documents are organized into collections, which are similar to tables in relational databases but are schema-less and can store documents with varying structures.
- **Scalability:** MongoDB is designed to scale horizontally across multiple servers, making it suitable for handling large volumes of data and high-traffic applications. It supports sharding, which involves distributing data across multiple machines, and replica sets, which provide redundancy and high availability.
- **High Performance:** MongoDB is optimized for high performance, with features like indexing, query optimization, and support for in-memory computing. It also offers features like aggregation pipelines for complex data processing tasks.
- **Query Language:** MongoDB provides a rich query language that supports a wide range of operations for querying and manipulating data. It supports CRUD operations (Create, Read, Update, Delete), as well as advanced operations like aggregation, text search, geospatial queries, and more.
- **Flexible Schema:** Unlike relational databases with fixed schemas, MongoDB offers a flexible schema that allows documents in the same collection to have different structures. This flexibility makes it easier to iterate on data models and accommodate changes in application requirements over time.
- **Community and Ecosystem:** MongoDB has a large and active community of developers and contributors, as well as a rich ecosystem of tools, libraries, and services built around it. This ecosystem includes tools for data visualization, monitoring, management, and integration with other technologies.

MongoDB was created by a company '10Gen', which is now known as MongoDB Inc.

NoSQL VS SQL

let's discuss the differences between SQL (relational databases) and NoSQL databases in detail, along with suitable examples:

1. Data Model

- **SQL (Relational Databases)**
  - Relational databases follow a structured data model based on tables with rows and columns.
  - Data is organized into tables, each defining a specific entity, with relationships established through foreign keys.
  - Examples: MySQL, PostgreSQL, SQLite.

- **NoSQL Databases**
  - NoSQL databases offer various data models, including document-based, key-value pairs, column-oriented, and graph databases.
  - Each NoSQL database type provides a different way of structuring and organizing data, offering flexibility in data modeling.
  - Examples: MongoDB (document-based), Redis (key-value pairs), Cassandra (column-oriented), Neo4j (graph database).

2. Schema

- **SQL (Relational Databases)**
  - Relational databases enforce a fixed schema, where the structure of data (tables, columns, data types) must be predefined before inserting data.
  - Changes to the schema typically require altering the database schema, which can be complex and time-consuming.

- **NoSQL Databases**
  - NoSQL databases are schema-less or have a flexible schema, allowing for dynamic changes to the data structure without requiring a predefined schema.
  - Each document or record in a NoSQL database can have its own structure, enabling easier adaptation to evolving application requirements.

3. Query Language

- **SQL (Relational Databases)**
  - SQL (Structured Query Language) is the standard language for querying and manipulating data in relational databases.
  - SQL provides a declarative syntax for performing CRUD operations (Create, Read, Update, Delete), as well as complex queries and joins.

- **NoSQL Databases**
  - NoSQL databases typically have their own query languages optimized for specific data models.
  - Query languages may vary between different NoSQL databases but generally offer capabilities for basic CRUD operations and specific querying features tailored to the database type.

4. Scalability

- **SQL (Relational Databases)**
  - Relational databases are traditionally scaled vertically (scale-up), by adding more resources (CPU, memory) to a single server.
  - Horizontal scaling (scale-out) can be challenging and may involve complex partitioning strategies.

- **NoSQL Databases**
  - NoSQL databases are designed for horizontal scalability, allowing data to be distributed across multiple servers or nodes.
  - NoSQL databases can handle large volumes of data and high throughput by adding more nodes to the cluster, without requiring significant changes to the application or database architecture.

5. Examples

- **SQL (Relational Databases)**
  - Example: Consider an e-commerce app storing data about customers, orders, & products in relational tables (e.g., Customers, Orders, Products).
  - Tables would have predefined schemas with fixed columns, and relationships between tables would be established through foreign keys.

- **NoSQL Databases**
  - Example: In the same e-commerce application, using MongoDB as a NoSQL database, each order could be represented as a JSON document within a collection.
  - The structure of the order document could vary based on the items ordered, without requiring a fixed schema for all orders.
  - This flexibility allows for easier handling of nested data, such as customer details and order items, within a single document.

Hence, SQL databases (relational databases) and NoSQL databases differ in their data models, schema flexibility, query languages, scalability approaches, and usage scenarios. The choice between SQL and NoSQL databases depends on factors such as data structure, scalability requirements, and application complexity.

Use-Cases : NoSQL

NoSQL databases are often the best option to use in the following scenarios:

- **Handling Large Volumes of Data:** NoSQL databases are designed to scale horizontally, making them well-suited for handling large volumes of data distributed across multiple servers or nodes. They can efficiently manage big data workloads without sacrificing performance.
- **Flexible Schema Requirements:** NoSQL databases offer flexible schema options, allowing developers to store data with varying structures within the same collection or table. This flexibility is beneficial for applications with evolving or unpredictable data schemas.
- **High Availability and Fault Tolerance:** NoSQL databases, particularly those with features like automatic failover and replication (e.g., MongoDB replica sets), provide high availability and fault tolerance. They can automatically recover from hardware failures and ensure continuous operation.
- **Scalability Requirements:** NoSQL databases excel at horizontal scalability, allowing for seamless scaling out by adding more nodes to the database cluster. This makes them suitable for applications with growing data and user loads, without the need for complex partitioning strategies.
- **Real-time Analytics and IoT Applications:** NoSQL databases are commonly used in real-time analytics and IoT (Internet of Things) applications, where data is

generated and processed rapidly. Their ability to handle high-throughput workloads and semi-structured data makes them ideal for capturing and analyzing real-time data streams.

- **Document-centric Applications:** NoSQL document databases like MongoDB are particularly well-suited for document-centric applications, such as content management systems, e-commerce platforms, and social media applications. They allow developers to store and query complex, nested data structures (e.g., JSON documents) efficiently.
- **Agile Development and Rapid Prototyping:** NoSQL databases offer schema flexibility and agile development capabilities, allowing developers to iterate quickly on data models and application features. They are conducive to agile development practices, enabling faster time-to-market for new applications and features.
- **Geospatial and Graph Data:** Some NoSQL databases provide specialized features for handling geospatial data (e.g., MongoDB's geospatial indexes) or graph data (e.g., Neo4j). These features are beneficial for applications requiring spatial queries, location-based services, or graph-based data modeling.

Hence, NoSQL databases are a preferred choice for applications requiring scalability, flexibility, high availability, and real-time data processing capabilities. They are well-suited for modern, data-intensive applications across various industries, including e-commerce, gaming, social networking, IoT, and analytics.

Aspects of MongoDB

MongoDB is a feature-rich and versatile database system that offers scalability, flexibility, performance, and robustness, making it well-suited for modern applications across various industries and use cases.

- **Data Model:** MongoDB stores data in flexible, JSON-like documents called BSON (Binary JSON), which allows for a more natural representation of hierarchical relationships and complex data structures. Each document can have a different structure, with varying fields and data types, within the same collection.
- **Collections and Documents:** MongoDB organizes data into collections, which are analogous to tables in relational databases. Each collection contains multiple documents, and documents within a collection can have different structures. Collections and documents are schema-less, meaning there is no requirement for predefined schemas or data types.
- **Indexes:** MongoDB supports various types of indexes to improve query performance. Indexes can be created on single fields, compound fields, arrays, and even text fields for full-text search. Indexes are essential for efficiently querying large datasets and ensuring fast data retrieval.
- **Replication:** MongoDB provides replication functionality through replica sets, which are self-healing clusters of MongoDB servers. Replica sets maintain multiple copies of data across different servers to ensure data redundancy, fault tolerance, and high availability. If one server fails, another server can automatically take over to ensure continuous operation.
- **Sharding:** Sharding is MongoDB's approach to horizontal scaling, allowing the distribution of data across multiple servers or shards. Sharding improves scalability by distributing data and query load evenly across multiple machines. MongoDB's sharding architecture is based on a sharded cluster, which consists of multiple shards, config servers, and query routers (mongos).
- **Query Language and Aggregation:** MongoDB's query language is rich and expressive, supporting a wide range of operations for querying and manipulating data. It includes CRUD operations (Create, Read, Update, Delete), as well as advanced query features such as aggregation pipelines, which allow for complex data transformations, grouping, filtering, and analysis directly within the database.
- **Transactions:** MongoDB introduced multi-document transactions in version 4.0, allowing developers to perform atomic operations across multiple documents within a single transaction. This feature ensures data integrity and consistency,

especially in scenarios where operations across multiple documents need to be coordinated.

- **Security:** MongoDB provides various security features to protect data, including authentication, authorization, encryption, and auditing. It supports role-based access control (RBAC), allowing administrators to define granular access controls and permissions for users and applications accessing the database.
- **Scalability and Performance:** MongoDB is designed for high performance and scalability, with features such as in-memory computing, automatic load balancing, and horizontal scaling through sharding. It is optimized for both read and write-intensive workloads, making it suitable for a wide range of applications with diverse performance requirements.
- **Community and Support:** MongoDB has a vibrant and active community of developers, users, and contributors. It offers comprehensive documentation, online forums, user groups, and official support options to help users get started, troubleshoot issues, and optimize their MongoDB deployments.

Best Fit Consideration: MongoDB

- MongoDB is a versatile database system that can be used in a wide range of projects and software applications, including content management systems, e-commerce platforms, real-time analytics, mobile app backends, IoT, gaming, and location-based services.
- Its flexibility, scalability, performance, and ease of development make it a preferred choice for many developers and organizations across various industries.
- MongoDB is well-suited for a wide range of projects and software applications, particularly those that benefit from its flexible data model, scalability, performance, and ease of development.

Here are some types of projects and applications for which MongoDB is often considered the best fit:

- **Content Management Systems (CMS):** MongoDB's document-oriented data model is well-suited for managing content in CMS applications. It allows for flexible schema design, making it easy to store and retrieve structured and unstructured content, such as articles, blog posts, images, and videos.

- **E-commerce Platforms:** MongoDB is suitable for e-commerce applications, where product catalogs, customer data, and order information need to be stored and managed efficiently. Its scalability and performance make it ideal for handling large volumes of product listings, customer transactions, and inventory data.
- **Real-time Analytics and Big Data:** MongoDB is commonly used in real-time analytics and big data applications, where data is ingested, processed, and analyzed in real-time. Its ability to handle high-speed data ingestion and complex queries makes it well-suited for applications that require real-time insights and data-driven decision-making.
- **Social Media Platforms:** MongoDB is a popular choice for building social media platforms and networking applications. Its flexible data model allows developers to store user profiles, social interactions, messages, and multimedia content in a scalable and efficient manner.
- **Mobile App Backends:** MongoDB serves as an excellent backend database for mobile applications, providing offline synchronization, geospatial queries, and real-time data updates. Its support for JSON-like documents and flexible schema design simplifies data synchronization between mobile devices and backend servers.
- **Internet of Things (IoT):** MongoDB is well-suited for IoT applications, where sensor data, telemetry, and device information need to be stored and analyzed in real-time. Its ability to handle time-series data, geospatial queries, and complex event processing makes it an ideal choice for IoT data management.
- **Gaming Platforms:** MongoDB is commonly used in gaming platforms and online gaming applications. Its scalability, performance, and support for real-time data updates make it suitable for managing player profiles, game states, leaderboards, and in-game transactions.
- **Location-based Services:** MongoDB's support for geospatial indexes and queries makes it suitable for location-based services and geospatial applications. It can efficiently store and query geospatial data, such as maps, points of interest, and location-based user interactions.
- **Prototyping and Rapid Development:** MongoDB's schema flexibility, agile development capabilities, and ease of use make it an ideal choice for prototyping and rapid development projects. It allows developers to iterate quickly on data models and application features, accelerating the development process.

JSON Vs BSON

- JSON is a human-readable, text-based data interchange format widely used for transmitting and storing structured data, while BSON is a binary serialization format optimized for storage and transmission efficiency in MongoDB.
- JSON is commonly used in web development and APIs, while BSON is specific to MongoDB and is used internally for storing documents in the database.
- Both JSON (JavaScript Object Notation) and BSON (Binary JSON) are data interchange formats used to represent structured data.

JSON (JavaScript Object Notation)

- JSON is a lightweight, text-based data interchange format that is easy for humans to read and write, and easy for machines to parse and generate.
- It is commonly used for transmitting data between a web server and a web client, as well as for storing configuration settings and exchanging data between systems.
- JSON data is represented as key-value pairs, where keys are strings and values can be strings, numbers, booleans, arrays, or nested objects.
- **Format:** JSON is a lightweight, text-based data interchange format inspired by JavaScript object syntax.
- **Human-readable:** JSON is easy for humans to read and write, and it's also easy for machines to parse and generate.

  Example of JSON data:

```
{
  "name": "Nikhil garg",
  "age": 14,
  "languages": ["JavaScript", "React", "Node"],
  "address": {
    "city": "Agra",
    "zipcode": "10001"          }}
```

- **Data Types:** JSON supports several data types, including:
  - Objects: Key-value pairs enclosed in curly braces {}.
  - Arrays: Ordered lists of values enclosed in square brackets [].
  - Strings: Sequences of characters enclosed in double quotes " " or single quotes ' '.
  - Numbers: Integer or floating-point numbers.
  - Booleans: true or false.
  - Null: null.

BSON (Binary JSON)

- BSON is a binary serialization format used to store and transfer data in MongoDB, a NoSQL database system.
- Unlike JSON, which is text-based, BSON is a binary format, which makes it more compact and efficient for storage and transmission.
- BSON extends JSON by adding additional data types such as Date, Binary Data, and ObjectId, which are commonly used in MongoDB documents.
- BSON documents have a similar structure to JSON documents, with key-value pairs representing the data fields, but they are encoded in binary format for storage and transmission efficiency.
- **Format:** BSON is a binary serialization format used primarily in MongoDB for storing and transferring data.
- **Efficiency:** BSON is designed to be more compact and efficient than JSON, making it suitable for storage and transmission over networks.

  Example of BSON data (same as JSON but represented in BSON format):

```
{
  "name": "Nikhil garg",
  "age": 14,
  "languages": ["JavaScript", "React", "Node"],
  "address": {
    "city": "Agra",
    "zipcode": "10001"          }}
```

- **Data Types:** BSON extends JSON by adding additional data types, including:
  - **Date:** Represents date and time information.
  - **Binary Data:** Represents binary data in various formats.
  - **ObjectId:** Represents a unique identifier used in MongoDB.
  - **Regular Expression:** Represents regular expression patterns.
  - **Timestamp:** Represents a time stamp.
  - **Decimal128:** Represents high-precision floating-point numbers.

BSON documents consist of a series of fields and values encoded in a binary format.

For example:

```
\x31\x00\x00\x00\x02hello\x00\x06\x00\x00\x00world\x00\x00
```

BSON is specific to MongoDB and is used internally for storing documents in collections. It allows MongoDB to efficiently store and retrieve data, perform queries, and manage indexes.

\x31\x00\x00\x00\x02hello\x00\x06\x00\x00\x00world\x00\x00

The string \x31\x00\x00\x00\x02hello\x00\x06\x00\x00\x00world\x00\x00 is a representation of a BSON document encoded in hexadecimal. Each pair of hexadecimal digits represents a byte in the BSON data.

Let's break down the structure of this BSON document:

- **\x31\x00\x00\x00:** This represents the total size of the BSON document, including itself. The total size is 49 bytes (\x31 in hexadecimal is 49 in decimal).
- **\x02:** This byte represents the data type of the field "hello". In BSON, 0x02 represents a string type.
- **hello\x00:** This is the field name. The string "hello" followed by a null terminator (\x00).
- **\x06\x00\x00\x00:** This represents the length of the string value "world". The length is 6 bytes (\x06\x00\x00\x00 in little-endian byte order).
- **world\x00:** This is the string value "world" followed by a null terminator (\x00).
- **\x00:** This marks the end of the BSON document.

Hence, the string \x31\x00\x00\x00\x02hello\x00\x06\x00\x00\x00world\x00\x00 represents a BSON document with a single field named "hello" with the value "world". The total size of the BSON document is 49 bytes.

Don't Forget

- MongoDB is designed for flexibility, scalability, high performance and handling unstructured & semi-structured data.
- Unstructured and semi-structured data are two types of data formats that do not fit neatly into traditional structured formats, such as rows and columns in a relational database.
- Unstructured data lacks a predefined structure and is often challenging to process, while semi-structured data retains some level of organization or metadata, making it more accessible but still requiring preprocessing for analysis.
- Both types of data are common in modern applications and require specialized tools and techniques for effective management and analysis.

Unstructured Data

- Definition: Unstructured data refers to data that does not have a predefined data model or organization. It lacks a formal structure, making it challenging to process and analyze using traditional database management systems.
- Characteristics
  - Unstructured data can include text documents, images, audio files, video files, social media posts, emails, sensor data, and more.
  - It often contains natural language text, multimedia content, or raw data without a specific schema or format.
  - Unstructured data is typically stored in file systems, content management systems, NoSQL databases, or data lakes.
- Examples
  - Text documents (e.g., Word documents, PDFs)
  - Images (e.g., photographs, scanned documents)
  - Audio files (e.g., voice recordings, podcasts)
  - Video files (e.g., movies, surveillance footage)
- Challenges
  - Extracting meaning and insights from unstructured data requires advanced techniques such as natural language processing (NLP), image processing, audio analysis, and machine learning.
  - Unstructured data can be voluminous and difficult to manage, requiring specialized storage and processing solutions.

Semi-Structured Data

- Definition: Semi-structured data refers to data that does not conform to a rigid structure like relational databases but contains some level of organization or metadata that enables partial understanding.
- Characteristics
  - Semi-structured data retains some form of organizational structure or metadata, making it more accessible than unstructured data.
  - It may have a flexible schema or a loosely defined schema, allowing for variations in data representation within the same dataset.
  - Semi-structured data often includes formats such as JSON (JavaScript Object Notation), XML (eXtensible Markup Language), YAML (YAML Ain't Markup Language), CSV (Comma-Separated Values), and log files.
- Examples
  - JSON documents (e.g., API responses, NoSQL databases)
  - XML files (e.g., RSS feeds, configuration files)
  - CSV files (e.g., spreadsheets, tabular data)
  - Log files (e.g., server logs, application logs)
- Advantages
  - Semi-structured data offers a balance between flexibility and organization, making it suitable for scenarios where data schemas may evolve over time.
  - It can be processed using a combination of traditional database techniques and modern data processing tools like Apache Spark, Hadoop, and Elasticsearch.
- Challenges
  - Semi-structured data still requires preprocessing and transformation to extract meaningful insights, particularly when dealing with nested or complex structures.
  - Managing schema evolution and data quality can be challenging, especially in large and heterogeneous datasets.

Inside MongoDB database, we have collections and inside every collections, we have documents which is in json/bson format.

MongoDB Architecture

In MongoDB, the server, storage engine, and database work together to store, manage, and access data efficiently.

How they work

- **MongoDB Server**
  - The MongoDB server (mongod) is responsible for handling client connections, executing database operations, and coordinating data storage and retrieval.
  - The server process listens for incoming client requests over the network and communicates with client applications using the MongoDB wire protocol.
  - The server manages multiple databases and collections, each containing documents organized in BSON format.
  - The server supports various administrative operations, including creating databases, collections, and indexes, as well as managing user authentication and authorization.
- **Storage Engine**
  - The storage engine is responsible for managing data storage, indexing, and retrieval on disk. MongoDB supports multiple storage engines, including WiredTiger (default) and MMAPv1 (deprecated).
  - The storage engine handles data persistence, ensuring that data modifications are durably stored on disk and can be recovered in case of server failures or crashes.
  - It manages storage structures such as data files, journal files, indexes, and in-memory caches to optimize data access and performance.
  - The storage engine implements data compression, concurrency control, and transaction management to ensure data consistency and integrity.
- **Database**
  - A database in MongoDB is a logical container for storing collections of documents. Each database is identified by a unique name and can contain multiple collections.
  - Databases are created dynamically as needed and do not require predefined schemas or data types. MongoDB supports flexible schema design, allowing documents within the same collection to have different structures.
  - Each database has its own set of users and privileges, allowing for fine-grained access control and security policies.

Hence, you can understand that

- The MongoDB server acts as the core component responsible for handling client connections, executing database operations, and managing data storage and retrieval.
- The storage engine is responsible for managing data storage on disk and optimizing data access and performance.
- Databases provide logical containers for organizing collections of documents and support flexible schema design and fine-grained access control. MongoDB databases are automatically sharded and distributed across multiple nodes in sharded clusters for horizontal scalability and fault tolerance.

Together, these components work together to provide a flexible, scalable, and high-performance database system for modern applications.

MongoDB Installation & Setup

- MongoDB Community Server
  https://www.mongodb.com/try/download/community

  through the setup installation, don't forget to check "Install MongoDB Compass"

- MongoDB Shell
  https://www.mongodb.com/try/download/shell

- MongoDB Command Line Database Tools
  https://www.mongodb.com/try/download/database-tools

- Now add the following path to System Environment variables
  C:\Program Files\MongoDB\Server\7.0\bin

- To check the version of mongodb server, use following command in command prompt
  Mongod –version

Now we don't need to start/run the mongodb server manually because its automatically gets started. You can check it by your system's task manager processes log.

Using shell for execution

- Use "mongosh" command in command prompt to enter inside mongodb shell.
- To check the available database, use command
  show dbs
  or show databases
- To create a database in mongodb, or to switch between databases, use command
  use 'database_name'
  for ex: use emp_records

  now If you want to see the available databases, using "showdbs" command then you will not get the name of newly created database, because no any collection or document is available inside the database.

  So, we need to create a collection or document inside the newly created database.

- To create collection inside the database, use command:
  use database_name
  db.createCollection('collection_name') //to create a single collection
  for ex: db.createCollection('emp_data')

- To check the available connection inside the database, use command
  show collections

- To delete the available collection inside the database, use command
  db.emp_data.drop()

- To delete the database, use command
  db.dropDatabase()