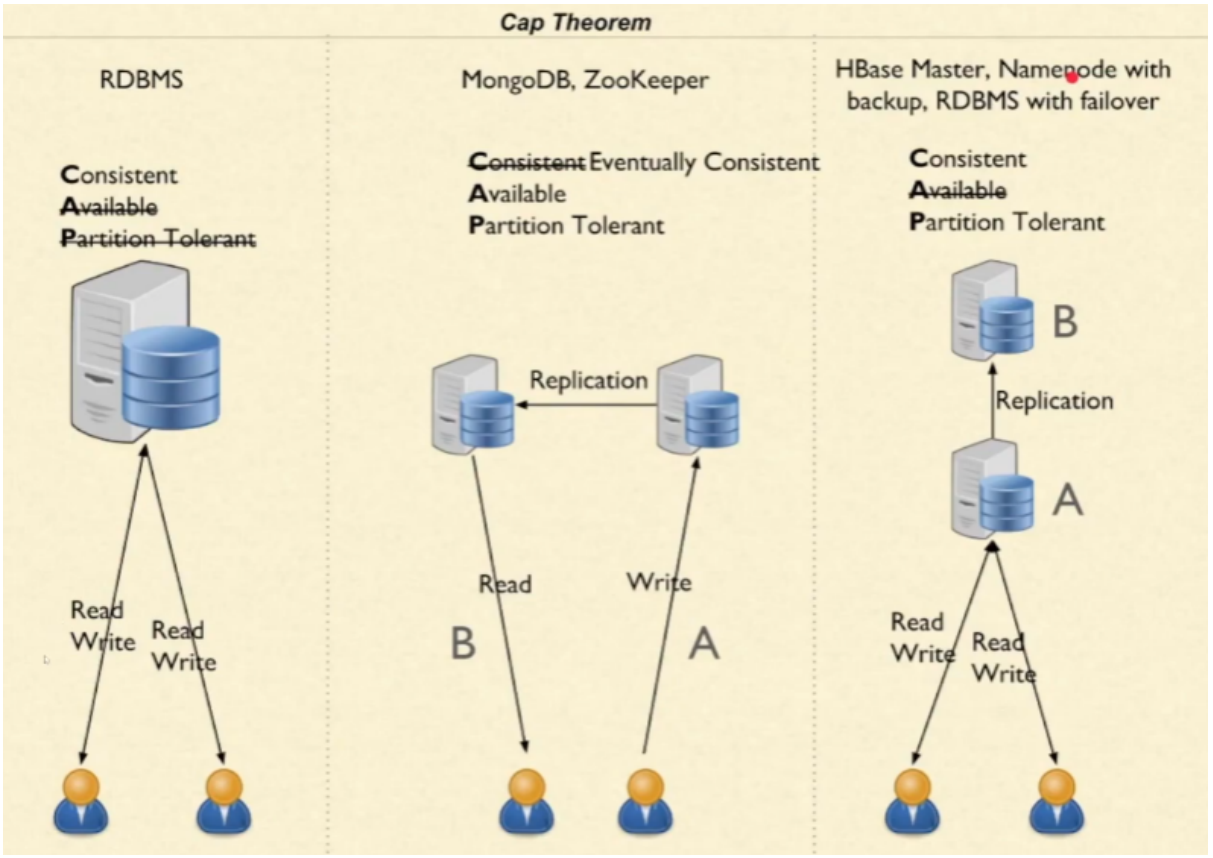
SQL	NoSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have a dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows ACID property	Follows CAP(consistency, availability, partition tolerance)
Examples: MySQL, PostgreSQL, Oracle, MS-SQL Server, etc	**Examples:** MongoDB, GraphQL, HBase, Neo4j, Cassandra, etc

CAP Theorem

- It is very important to understand the limitations of NoSQL database. NoSQL can not provide consistency and high availability together. This was first expressed by Eric Brewer in CAP Theorem. CAP theorem or Eric Brewers theorem states that we can only achieve at most two out of three guarantees for a database: Consistency, Availability and Partition Tolerance.
- **Consistency**: Consistency refers to the property that all nodes in a distributed system see the same data at the same time. In other words, if a data item is updated, all subsequent accesses to that item will reflect the updated value.
- **Availability**: Availability means that the system remains operational and continues to provide responses to client requests, even in the presence of node failures or network partitions. An available system guarantees that every request receives a response, regardless of the state of the system.
- **Partition tolerance**: Partition tolerance refers to the system's ability to function and continue operating even when there is a communication breakdown (partition) between different nodes in a distributed system. Partitions can occur due to network failures or other reasons.
- Out of these three guarantees, no system can provide more than 2 guarantees. Since in the case of a distributed systems, the partitioning of the network is must, the tradeoff is always between consistency and availability

- RDBMS can provide only Consistency but not Partition ToleranceMongodb, HBase and Redis can provide Consistency and Partition Tolerance
- CouchDB, Cassandra can provide only availability and Partition Tolerance but no consistency. SUch database generally settle down for eventually consistency and settle down after some time. Means after while the system will provide consistent data

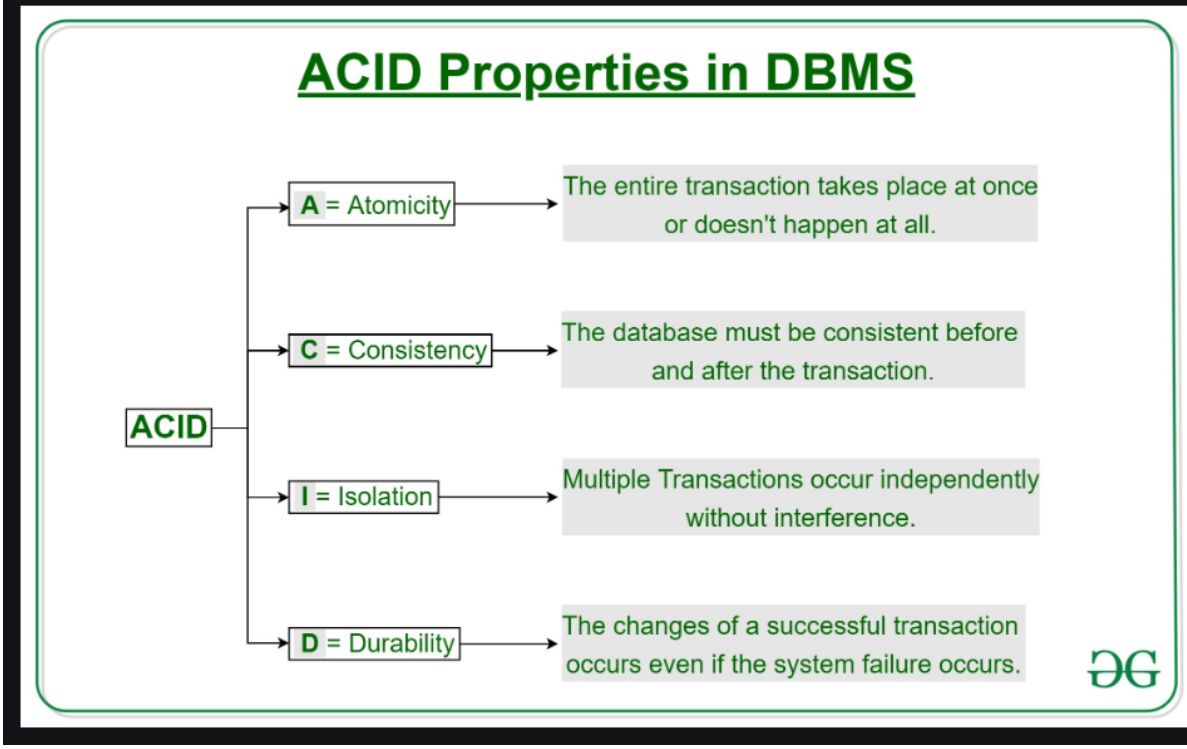


Types of NoSQL

- NoSQL databases are a category of database management systems that provide alternatives to traditional relational databases
    - **Document Databases**: Document databases store and retrieve data in the form of flexible, semi-structured documents, typically using JSON or XML formats. Each document can have its own structure, and fields within the document can vary. Examples of document databases include MongoDB, Couchbase, and CouchDB.
    - **Key-Value Stores**: Key-value stores are simple data stores that associate a unique key with a value. The value can be any type of data, such as strings, numbers, or more complex structures. Key-value stores provide fast and efficient access to data based on the key. Examples of key-value stores include Redis, Amazon DynamoDB, and Riak.
    - **Column-Family Stores**: Column-family stores organise data into column families, which are containers for columns of related data. Each column contains a key-value pair, and column families can be different for each row. Column-family stores are designed for handling large amounts of structured and semi-structured data. Apache Cassandra and Apache HBase are examples of column-family stores.

- **Graph Databases**: Graph databases focus on the relationships between entities and provide efficient storage and traversal of graph-like data structures. They store data as nodes (entities) and edges (relationships) between those nodes. Graph databases are suitable for scenarios such as social networks, recommendation systems, and network analysis. Examples of graph databases include Neo4j, Amazon Neptune, and JanusGraph.
- **Wide-Column Stores**: Wide-column stores (also known as columnar databases) store data in columns rather than rows. They are optimised for handling large-scale, read-heavy workloads and analytical queries. Wide-column stores are particularly useful for scenarios involving time series data or data analytics. Apache Cassandra and Apache HBase can also be considered wide-column stores.

A transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.
In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.



Advantages of using the Document Database

- **Intuitive data model** enables developers to work with data in a way that is more natural and intuitive, facilitating faster development and maintenance of applications. Every data which you need to retrieve, you will keep them into a single document so that data can be retrieved quickly and efficiently. No need to have expensive joins(left join, right join) or divide your data into multiple tables or multiple objects like in RDB.
- **Flexibility and Agility**: The flexibility of document databases allows developers to evolve their data model as the application requirements change over time. They can easily add, modify, or remove fields within documents without disrupting the existing data. This agility empowers developers to iterate quickly, adapt to new business

needs, and make data model changes seamlessly. You can create a document which contains an array and element of the array can further contain different arrays.

- **Easy Scalability**: Document databases offer horizontal scalability, allowing you to scale your database by adding more servers or nodes to a cluster. This scalability makes it easier to handle increasing data volumes and high traffic loads without sacrificing performance.
- **Support for Replication and High Availability**: Document databases often have built-in features for replication and high availability. They can replicate data across multiple nodes or servers to ensure data durability and fault tolerance. This replication enables automatic failover and improved data availability.

Advantages of using the MongoDB

- Open source and free (community edition), also free cloud version available of MongoDB Atlas and you can use that for your small project.
- It is very easy to do various CRUD -create, read, update, delete operations.
- **Rich Querying and Indexing**: MongoDB provides a powerful and flexible query language with support for a wide range of queries, including ad-hoc queries, range queries, geospatial queries, and full-text search. It also offers extensive indexing options to optimise query performance and allows for the creation of compound indexes, geospatial indexes, and text indexes. This makes MongoDB suitable for applications with complex data access requirements.
- **Built-in Replication and High Availability**: MongoDB has built-in support for replication and high availability. It supports replica sets, which are self-healing clusters that provide automatic failover and data redundancy. Replica sets ensure that your data remains available even in the event of server failures.
- **Built-in Data Aggregation**: MongoDB includes powerful aggregation framework capabilities that allow you to perform complex data analysis and aggregation operations directly within the database. It supports functions like grouping, filtering, sorting, and joining data from multiple documents, providing a flexible and efficient way to analyse and transform data.

Difference between the JSON and BSON Documents and the terms with SQL and MongoDB

SQL Terms	MongoDB Terms
Database	Database
Table	Collection
Row	JSON / BSON Document
Column	Field
Index	Index

- **JSON (JavaScript Object Notation)**:
    - JSON is a lightweight data interchange format that is human-readable and easy to understand.
    - It is primarily used for transmitting data between a server and a web application, or between different components of a software system.

    - JSON supports various data types, including strings, numbers, booleans, arrays, and objects.
    - It is based on a subset of the JavaScript programming language syntax, but it is language-independent and can be used with any programming language.
    - JSON documents are text-based and have a file extension of .json.
- **BSON (Binary JSON)**:
    - BSON is a binary representation of JSON-like documents that is designed for efficient storage and traversal.
    - It is commonly used as the data storage and query format in MongoDB, a popular NoSQL database.
    - BSON supports the same data types as JSON, including strings, numbers, booleans, arrays, and objects.
    - In addition to the data types supported by JSON, BSON also includes additional types such as dates, binary data, and specialized numeric types.
    - BSON documents are binary-encoded and have a file extension of .bson.

	JSON (JavaScript Object Notation)	BSON (Binary JSON)
Encoding	UTF-8 String	Binary
Data Support	String, Boolean, Number, Array	String, Boolean, Number (Integer, Float, Long, Decimal128...), Array, Date, Raw Binary
Readability	Human and Machine	Machine Only

Common Errors

- **duplicate key error** occurs when an attempt is made to insert a document with a key value that already exists in the collection. The key value refers to a field or fields that have a unique index constraint applied to them, typically the _id field.

Ad-hoc queries

- queries that are executed on an as-needed basis, typically for one-time or temporary purposes, without the need for predefined or saved queries. These queries are designed to meet immediate requirements and are not part of a structured or predefined query plan.

A schema-less database

- type of database system that allows for flexible and dynamic data structures without requiring a predefined schema or rigid data model. In a schema-less database, each record or document can have its own unique structure, with different fields and field types.
- Unlike traditional relational databases that enforce a fixed schema where the structure of data is defined in advance, schema-less databases provide more agility and adaptability, allowing data to be added or modified without strict schema constraints.

Indexing

- a technique used in databases to improve the speed and efficiency of query execution by creating data structures that allow for faster data retrieval. An index is an additional data structure that is created based on one or more fields in a database

table or collection. It acts as a reference to the actual data, enabling the database to quickly locate and retrieve the desired information.

- **TTL (Time to Live) indexing**
    - is a feature provided by some databases, including MongoDB, to automatically remove documents or records from a collection after a specified amount of time has passed. It is useful for managing data that has an expiration or limited lifespan, such as temporary data, session information, or event logs.
    - TTL indexing works by creating an index on a field that represents the expiration time of the documents. The index contains the timestamp when each document should expire. The database periodically scans the index and removes documents that have surpassed their expiration time.

replica set

- A specific configuration or deployment setup in MongoDB that involves multiple MongoDB instances working together to provide replication. A replica set consists of a primary node, which is the main read-write server, and one or more secondary nodes, which replicate data from the primary.
- replication is a general concept of creating multiple copies of data, while a replica set is a specific configuration in MongoDB that enables replication and provides high availability and automatic failover. Replica sets are commonly used in MongoDB deployments to ensure data redundancy, fault tolerance, and scalability.
- **Replication**
    - a process in which data is copied and synchronized across multiple servers or nodes in a distributed database system. The primary goal of replication is to ensure data availability, reliability, and fault tolerance.
    - By replicating data across multiple servers, replication enhances system resilience, improves read performance, and provides the foundation for various high-availability and disaster recovery strategies.

How to add media

- In MongoDB, you can store media files such as images, videos, or audio files as Binary Large Objects (BLOBs) using the GridFS specification. GridFS is a specification for storing and retrieving large files in MongoDB.
  GridFS
    - a file storage system used in MongoDB to store and retrieve large files that exceed the BSON document size limit of 16 megabytes (MB). It provides a mechanism for dividing and storing files across multiple smaller documents called chunks.
    - GridFS is particularly useful in scenarios where you need to store and retrieve large files, such as images, videos, audio files, or documents, in a MongoDB database. It provides a way to overcome the BSON document size limitation and offers features like metadata storage, indexing, and streaming access for efficient file management within the MongoDB ecosystem.

Sharding

- a technique used in distributed databases to horizontally partition data across multiple servers or nodes called shards. It enables scalability and high-performance data storage and retrieval by distributing the data and query load across multiple machines.

- Sharding is commonly used in large-scale distributed database systems to handle massive amounts of data and high query loads. It allows for horizontal scalability, improved performance, and fault tolerance. Popular databases like MongoDB, Apache Cassandra, and Amazon DynamoDB provide built-in support for sharding, making it easier to scale and manage large datasets in distributed environments.The components involved in sharding typically include:
    - Shard: A shard is a logical or physical division of the database. It contains a subset of the data and is hosted on a separate node or server. Each shard can operate independently and handle a portion of the workload.
    - Shard Key: The shard key is a unique identifier or attribute used to determine which shard a particular piece of data belongs to. It's typically selected based on the data distribution and query patterns to ensure an even distribution of data across the shards.
    - Shard Manager/Coordinator: The shard manager or coordinator is responsible for managing the sharded database. It tracks the location of each shard, handles data routing between shards, and coordinates operations that involve multiple shards.
    - Data Distribution Algorithm: The data distribution algorithm determines how data is assigned to different shards. It aims to evenly distribute the data based on the shard key to prevent hotspots or imbalances that could impact performance.
    - Query Router, Data Migration Mechanism, Shard Monitoring and Management Tools.

Achieving high performance

- a database system involves optimising various aspects of the system to ensure fast and efficient data storage, retrieval, and processing. Here are some strategies and considerations for achieving high performance:
    - Indexing, Query Optimization, Partitioning and Sharding, Hardware Considerations,...

how many nodes can be beset in the replica set

- In MongoDB, a replica set is a group of MongoDB instances that replicate data across multiple nodes for high availability and fault tolerance. The number of nodes that can be configured in a replica set depends on MongoDB's limitations and best practices.
- Minimum Requirement: A replica set must consist of at least three nodes to ensure fault tolerance and maintain a majority for elections in case of node failures.
- Maximum Limit: MongoDB supports up to 50 members (nodes) in a replica set. However, it's important to note that having such a large number of nodes can introduce additional complexity and administrative overhead.
- It's important to design the replica set based on the specific requirements of your application, taking into account factors like read and write scalability, fault tolerance, data redundancy, and operational considerations. Consulting MongoDB's documentation and seeking best practices can help determine the appropriate number of nodes for your replica set based on your use case.

How mongodb handles transaction

- MongoDB introduced multi-document ACID transactions starting from version 4.0. With the addition of transactions, MongoDB provides support for atomicity, consistency, isolation, and durability at the document level.

- It's important to note that transactions in MongoDB are typically used in scenarios that require strong consistency guarantees across multiple document updates. In many cases, designing the data model to minimise the need for transactions, using atomic updates, or leveraging other MongoDB features like document-level validations can provide high performance and scalability without the need for transactions.
- MongoDB's transactional capabilities give developers the flexibility to handle complex operations that require multiple document updates while maintaining data integrity and consistency.

Capped collections

- A type of collection that have a fixed size and follow a circular structure. They maintain insertion order based on insertion time, which means that when the collection reaches its maximum size, it automatically overwrites the oldest documents with new ones.
- They have some limitations, though, such as not supporting document updates that increase the document size or the ability to delete individual documents.

Method Chaining:

- Method chaining is a technique used in MongoDB to chain multiple query operations together using various methods provided by the MongoDB driver or query builder. With method chaining, you can perform operations like filtering, sorting, limiting, and projecting fields on a collection using a chain of method calls.
- Method chaining is a simpler approach used for basic query operations and document manipulation, while aggregation provides a more advanced and flexible way to perform complex data transformations and analysis using pipelines and stages.

Data modeling

- The process of designing the structure and organization of data in a database system. It involves identifying the entities, relationships, attributes, and constraints of the data to create an effective and efficient database schema that meets the requirements of the application.
- The goal of data modeling is to create a well-structured and efficient database schema that accurately represents the data requirements of the application and supports the desired functionality and performance.

Normalization

- the process of organizing and structuring data in a relational database to eliminate redundancy, improve data integrity, and optimize data storage and retrieval.
- data modeling is the process of creating a conceptual representation of the database structure, while normalization is a technique used during the data modeling process to eliminate redundancy and improve data integrity. Data modeling provides the overall framework for organizing and structuring data, while normalization helps ensure that the resulting database design meets specific requirements for data organization and integrity.

Relational data modeling

- The process of designing the structure and organization of data in a relational database management system (RDBMS). It involves identifying entities, relationships, attributes, and constraints to create an effective and efficient database schema that accurately represents the data requirements of an application

Journaling

- A technique used to provide durability and crash recovery capabilities. It is a mechanism that helps ensure data integrity and recoverability in case of system failures or crashes.
- Journaling is a crucial technique in database systems to ensure data durability, recovery, and maintain ACID (Atomicity, Consistency, Isolation, Durability) properties, which are essential for reliable and robust data management.

Embedded data, also known as **embedded documents** or **nested documents**,

- A data modeling technique used in databases where one document contains another document as a field or attribute. In other words, instead of storing data in separate collections and establishing explicit relationships between them, the data is embedded within a single document.

ObjectId data type

- 12 bytes. An ObjectId is a 12-byte identifier generated by MongoDB for each document in a collection. It is primarily used as a unique identifier and is automatically assigned to documents upon insertion.

MongoDB backups and restores

- safeguard your data and recover it in case of data loss, system failures, or other unforeseen events. MongoDB provides several methods and tools for backup and restore operations. Here are some commonly used approaches:
- mongodump and mongorestore, File System Snapshots, Cloud Provider Backup Services.

Storage engine

- A software component responsible for managing the storage and retrieval of data on disk or in memory. It is a fundamental part of the DBMS architecture that interacts directly with the underlying storage system.
- The storage engine handles the low-level details of how data is organized, stored, accessed, and manipulated within the database. It provides mechanisms for reading and writing data, managing indexes, handling transactions, enforcing data integrity, and implementing various storage-related optimizations.
- Types:
    - WiredTiger is the default storage engine in MongoDB since version 3.2.
    - The MMAPv1 storage engine was the default storage engine in MongoDB prior to version 3.2. MongoDB 4.0 marked MMAPv1 as deprecated, and its usage is discouraged in favor of WiredTiger.

What is good alternative to MongoDB?

- Some of the best MongoDB alternatives include Redis, Apache Cassandra, RethinkDB, DynamoDB, OrientDB, CouchDB, and ArangoDB

"Upsert"

- A database operation that combines the functionalities of "update" and "insert." The term "upsert" is a portmanteau of "update" and "insert." It is used to perform an update operation on a document or row in a database, and if the document or row does not exist, it inserts a new document or row instead
- Steps
    - Check if the record already exists: The system examines the target database to determine if a record with a specific key or unique identifier already exists.

    - If the record exists: If the record is found, the system performs an update operation, modifying the existing record with the new data provided in the upsert operation.
    - If the record does not exist: If the record is not found, the system performs an insert operation, creating a new record with the provided data.

why does MongoDB use bson

- BSON serves as a foundational component of MongoDB's architecture, providing efficient data storage, transmission, and compatibility across different programming languages. It helps MongoDB deliver the performance, flexibility, and scalability required for modern applications and their diverse data storage needs.

voting

- the election of a primary node in a replica set. A replica set is a group of MongoDB instances that replicate data across multiple nodes for high availability and fault tolerance. Each replica set typically consists of multiple nodes, where one node serves as the primary and the others function as secondary nodes.
- When a replica set is initially set up or if the primary node becomes unavailable due to a failure or maintenance, the remaining nodes in the replica set need to elect a new primary node to ensure continuity of operations. This election process involves voting among the eligible nodes to determine the next primary.
- Here's a general overview of how voting works in MongoDB replica sets:
    - Each node in the replica set has a voting configuration associated with it. By default, every node has one vote.
    - When an election is triggered due to the unavailability of the primary node, the eligible nodes participate in the voting process. Eligible nodes are those that are reachable, have a data replication oplog that is up-to-date, and have not been configured as non-voting members.
    - Each eligible node casts a vote for itself or for another eligible node. The vote includes information such as the node's ID and the last known election term.
    - The node that receives the majority of the votes becomes the new primary. If no node receives a majority, the election is inconclusive, and the replica set remains in a state without a primary. In such cases, manual intervention is required to resolve the issue and elect a primary node.

Profiler

- A tool or feature that allows you to gather information about the performance and resource usage of database operations. It helps in analyzing and optimizing the execution of queries, identifying bottlenecks, and understanding the behavior of the database system.
- The profiler provides detailed information about the execution of queries, including the time taken to execute each query, the resources consumed (such as CPU usage, memory usage, and disk I/O), and other statistics related to query performance. It helps in identifying slow-running queries, inefficient query plans, or resource-intensive operations.

Covered query, also known as an **index-covered query** or **index-only query**

- a type of database query that can be satisfied entirely using the indexes of a database, without the need to access the actual data stored in the documents or rows. It is a performance optimization technique that can significantly improve query execution time and reduce resource usage.

- It's important to note that not all queries can be fully covered. In some cases, the query requirements or the complexity of the data may necessitate accessing the actual data pages. However, for queries that can be optimized as covered queries, leveraging index-only access can significantly enhance performance and resource utilization.

Pagination

- Therefore, getting a lot of data at once is never the right way. Rather, we should always bring data in chunks as and when necessary.
- This technique is called Pagination where we get data in chunks of a particular size (offset) defined by pages.

Commonly used basic mongosh commands:

- **mongosh**: type in command prompt/any terminal to put us into the interactive mongodb shell.
- **cls**: clear the screen.
- **db**: show current database
- **use <database>**: Switches to the specified database. If the database doesn't exist, MongoDB will create it when you start adding data.
- **show databases / show dbs**: Lists all the available databases on the MongoDB server.
- **show collections**: Lists all the collections in the current database.
- **db.collectionName.find()**: Retrieves first 20 documents from the specified collection, for more type in **it**(iterate). For example, db.students.find() retrieves all documents from the "students" collection.
  Giving a second argument can give only the specified fields, for eg, db.student.find({age: 11},{name: 1, sex: 1}) . First argument can be empty giving all documents mentioned fields as output and it can be used to filter.
- **db.collectionName.findOne()**: Retrieves the first document from the specified collection. For example, db.students.findOne() retrieves the first document from the "students" collection.
- **db.collectionName.insertOne(document)**: Inserts a single document into the specified collection. For example, db.students.insertOne({ SRN: "SRN001", Sname: "John Doe", Degree: "Computer Science", Sem: 5, CGPA: 8.5 }) inserts a document into the "students" collection.
- **db.collectionName.insertMany(documents)**: Inserts multiple documents into the specified collection. For example, db.students.insertMany([{...}, {...}, {...}]) inserts multiple documents into the "students" collection.
- **db.collectionName.updateOne(filter, update)**: Updates a single document in the specified collection that matches the filter. For example, db.students.updateOne({ SRN: "SRN001" }, { $set: { CGPA: 9.0 } }) updates the CGPA field of the document with SRN "SRN001" in the "students" collection.
- **db.collectionName.deleteOne(filter)**: Deletes a single document from the specified collection that matches the filter. For example, db.students.deleteOne({ SRN: "SRN001" }) deletes the document with SRN "SRN001" from the "students" collection.

- **db.collectionName.deleteMany(filter)**: Deletes multiple documents from the specified collection that match the filter. For example, db.students.deleteMany({ Degree: "Computer Science" }) deletes all documents with the Degree "Computer Science" from the "students" collection.
- **db.collection_name.find().count()**: command in MongoDB is used to count the number of documents that match a query in a specific collection. Please note that the count() command is deprecated starting from MongoDB version 4.0. It is recommended to use the countDocuments() command instead, which provides the same functionality.
- **db.collection_name.find().limit(limit_value)**: used to restrict the number of documents returned in a query result. It allows you to specify the maximum number of documents to be retrieved from a collection.
- **db.collection_name.find().sort({ field_name: sort_order })**: used to sort the documents in a collection based on specified criteria. It allows you to specify one or more fields to sort by, along with the sort order (ascending or descending). sort_order with the desired sort order. Use 1 for ascending order and -1 for descending order.
- **db.students.find().skip(4).limit(3)**:display specific documents from the "students" collection in MongoDB, you can use the find() method along with the skip() and limit() methods. Here's how you can display documents 5, 6, and7.
- **distinct()** method is specifically designed to retrieve distinct values for a given field. If you have a large collection, this query may take some time to complete. Additionally, ensure that the field name ("degree" in the example) matches the field name in your actual collection.
- **db.collecion.find({genres:”magic”})**: checks for magic in genres, which is having an array of genres. **db.collecion.find({genres:[”magic”]})**: checks for genres array with only magic as its genre - exact match . **Querying arrays**
- **db.collection.deleteOne(filter, options)**:
    - filter: A document that specifies the criteria to match the document(s) to delete.
    - options: Optional. Additional options for the deletion operation. Such as { writeConcern: { w: "majority" } }
    - Note that **deleteOne()** only deletes a single document, even if there are multiple documents matching the query criteria. If you want to delete multiple documents that match a filter, you can use the **deleteMany()** method instead.
- **db.collection.updateOne(filter, update, options)**:
    - filter: A document that specifies the criteria to match the document(s) to be updated.
    - update: A document specifying the update operation to be performed on the matched document(s).
    - options: Optional. Additional options for the update operation.
    - The **updateOne()** method updates only the first document that matches the filter. If you want to update multiple documents that match the filter, you can use the **updateMany()** method instead.

- **dot notation**:

```
{
  "_id": ObjectId("1234567890"),
  "username": "john_doe",
  "email": "john@example.com",
  "address": {
    "city": "New York",
    "country": "USA"
  },
  "hobbies": ["reading", "painting", "gaming"],
  "friends": [
    {
      "name": "Jane Smith",
      "age": 30
    },
    {
      "name": "Mark Johnson",
      "age": 28
    }
  ]
}
```

  - Querying a nested field: **db.users.find({ "address.city": "New York" })**
  - Querying a nested field within an array: **db.users.find({ "friends.name": "Jane Smith" })**

OPERATORS

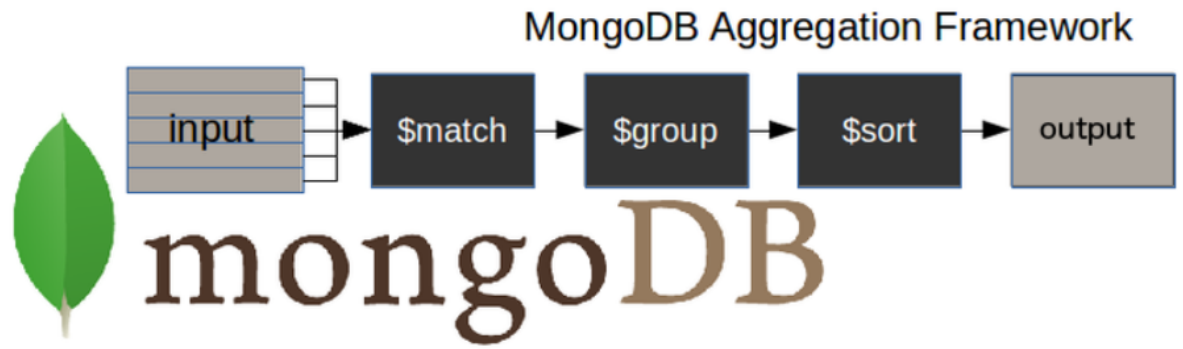
- Used to perform various operations on data stored in the MongoDB database. They are used in query expressions to filter, manipulate, and aggregate data. Here are some common operators used in MongoDB commands:
  - **Comparison Operators**: Used to compare values and perform conditional queries. Common comparison operators include **$eq** (equal to), **$ne** (not equal to), **$gt** (greater than), **$lt** (less than), **$gte** (greater than or equal to), and **$lte** (less than or equal to).
  - **Logical Operators**: Used to combine multiple conditions in a query. Common logical operators include **$and** (logical AND), **$or** (logical OR), and **$not** (logical NOT).
  - **Element Operators**: Used to check for the presence or absence of a field in a document. Common element operators include **$exists** (checks if a field exists) and **$type** (checks the data type of a field).
  - **Array Operators**: Used to query and manipulate arrays. Common array operators include **$in** (matches any value in an array), **$all** (matches all values in an array), and **$size** (checks the size of an array).
  - **Projection Operators**: Used to shape the output of query results. Common projection operators include **$project** (specifies fields to include or exclude), **$limit** (limits the number of documents returned), and **$sort** (sorts the documents).
  - **$set** operator is used within an update operation to modify or add fields in a document. It allows you to set the value of a specific field or add new fields to an existing document. For example: **db.collection.updateOne(filter, { $set: { field1: value1, field2: value2, ... } })**

- **$inc** operator is used within an update operation to increment or decrement the value of a specific field in a document.
  db.collection.updateOne(filter,{ $inc: { field1: value1,field2: value2, ... } })
  Please note that the $inc operator requires that the targeted field is numeric (e.g., integer or floating-point). If the field does not exist, the $inc operator will create it and set the specified increment value as the initial value.
- **$push** and **$pull** operators are used within update operations to add elements to an array field ($push) or remove elements from an array field ($pull) in a document
  db.collection.updateOne(filter, { $push: { arrayField: value } })
  db.collection.updateOne(filter, { $pull: { arrayField: condition } })
- **$each** operator is used in conjunction with the $push operator to add multiple elements to an array field in a document.
  db.collection.updateOne(filter,{$push:{arrayField:{$each:[value1, value2, ...] } } }). It allows you to push multiple values to an array at once.
- **db.collection.find({ field: { $in: [value1, value2, value3] } })**: **$in** Operator: The $in operator selects documents where the value of a field matches any value in the specified array. It acts as an "OR" operator for multiple values.
- **db.collection.find({ field: { $nin: [value1, value2, value3] } })**: **$nin** Operator: The $nin operator selects documents where the value of a field does not match any value in the specified array. It acts as a "NOT IN" operator.
- **Aggregation Operators**: Used in the MongoDB Aggregation Framework to perform complex data analysis and transformations. Common aggregation operators include **$group** (groups documents based on specified criteria), **$match** (filters documents based on specified conditions), and **$sum** (calculates the sum of numeric values).

Aggregation

How does the MongoDB aggregation pipeline work?

Here is a diagram to illustrate a typical MongoDB aggregation pipeline.



- `$match` stage – filters those documents we need to work with, those that fit our needs
- `$group` stage – does the aggregation job
- `$sort` stage – sorts the resulting documents the way we require (ascending or descending)

- A powerful framework for data processing and analysis. It allows you to perform complex operations on collections, such as grouping, filtering, transforming, and computing data. Aggregation pipelines consist of a sequence of stages, each stage performing a specific operation on the data.
- Documents during aggregation process pass through the stages
  db.collection_name.aggregate([
  <stage1>,
  <stage2>,
  <stage3>,
  <stage4>,
  ])
- Aggregation expression refers to the name of the field in input documents, can refer to nested documents using dot notation
- Aggregation stages
  -