MONGO DB

- What is Mongo DB?
  MongoDB is **an open-source NoSQL database management program.**

- How does MongoDB work?
  MongoDB is a NoSQL, document-oriented database management system that stores data as JSON-like documents. **It is flexible and scalable**. It uses a dynamic schema, allowing for changes in the document and structure to be made without affecting all other documents in the collection.

- Where is Mongo DB used?
  Mongo DB is used as a Web application as a back-end database.
  Mongodb uses Mozilla's Spidermonkey Javascript Engine.

- What is Relational DB (SQL)?
  Ex -MySQL, PostgreSQL, ORACLE, MariaDB

- What is a non-relational DB (NOSQL)?
  Ex- Mongo DB, Apache Cassandra, OrientDB, DynamoDB, Google's - BigTable.

- What is mongoose?

  Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward and schema-based solution for modeling and interacting with MongoDB databases from Node.js applications. Mongoose simplifies many common tasks associated with working with MongoDB, such as defining schemas, performing CRUD (Create, Read, Update, Delete) operations, and handling data validation

- **Collection** - A collection is a grouping of MongoDB documents. A collection is the equivalent of a table in a relational database system.

- **Documents** - **a way to organize and store data as a set of field-value pairs**. an organised store of documents in MongoDB, usually with common fields between documents.

- **JSON** - JSON is a data format that represents the values of objects, arrays, numbers, strings, booleans, and nulls.
    - **JSON.stringify()**: This method is used to convert a JavaScript object or value into a JSON string. It takes the object as an argument and returns the JSON representation as a string.
    - **JSON.parse()**: This method is used to parse a JSON string and convert it into a JavaScript object or value. It takes the JSON string as an argument and returns the corresponding JavaScript object.
    -

- **BSON** - MongoDB stores documents (objects) in a format called BSON. The reason why BSON is used is BSON supports many more data types than JSON

- **Extended JSON** - EXTENDED JSON is used to represent BSON data types.

- **BSON types** - Double, string, object, array, boolean

- **CRUD**
    - C - Create
    - R - Read - find,findOne
    - U - Update - update, updateOne , updateMany,replaceOne
    - D - Delete - deleteOne,deleteMany

- **Comparison Operators** - $in, $nin, $ne, $eq, $gt,$lt,$gte, $lte

- **Logical Operators** - $and, $or, $not, $nor

- **Operators** - $set, $unset, $inc, $rename,$addToSet
  sort(), skip(), limit()

- **upsert() -** upsert is a method that is used to insert and update the value in any operation. the MongoDB upsert method is a combination of insert and update. By default, the upsert method's value is always false. If the document matches the specified query and the method's value is set to true, the update operation will update the matching documents. If the document does not match the specified query and the method's value is true, this method inserts a new document in the collection. This new document contains the fields that indicate the operation.

- **Upsert vs save** - If a document exists with the specified _id value, the save() method performs an update, replacing all fields in the existing record with the fields from the document

- **Array update Operators** - $push, $pop, $pull, $pullall, $ - positional operator $all(), $size(),$addtoset()
  $elemmatch- The `$elemMatch` operator matches documents that contain an array field with at least one element that matches all the specified query criteria.
  $exists - checking the existence of the field in the specified collection **using the $exists operator**.

- **Field Filters** - **db.person.find({}, { name:1 , age:0})**
  {name:/^tes/} - name starting with tes - same with regex
  {name:/tes$/} - name ending with tes - same with regex
  $regex** - **db.person.find({name: {$regex:/rel/i}}) output - a combination of rel will be printed ex **rel**and, au**rel**ia

- **Atomic operators -** An atomic transaction is an indivisible and irreducible series of database operations such that either all occur, or nothing occurs.

We can clearly explain atomic operations in MongoDB by using the acronym ACID, which stands for Atomicity, Consistency, Isolation, and Durability.

- Here is a simple rule of atomicity for every single operation, “either all or none.”
- The consistency property will play a crucial role in atomicity. It will ensure that each transaction will ultimately lead to a valid state of some kind.
- The isolation property of the database will play a part in guaranteeing the systematic process of the concurrent transaction, which means one by one.
- Finally, the durability property will come into play. This property ensures the permanency of the database after each database transaction regardless of errors, crashes, and power loss.

- **Transactions** - A transaction is a sequence of actions that are executed as an indivisible unit. The **main goal of a transaction is to ensure that the data remains in a consistent state even in the face of errors or failures.** Transactions usually provide Atomicity, Consistency, Isolation, and Durability (ACID) guarantees, which ensure that the data is consistent and reliable. Atomicity means that all actions in the transaction are treated as a single, unbreakable unit, either all actions are executed or none are. This helps to prevent data corruption and maintain the integrity of the data.

- **Aggregation Framework** - Aggregation is a way of processing a large number of documents in a collection by means of passing them through different stages. The stages make up what is known as a pipeline. The stages in a pipeline can filter, sort, group, reshape, and modify documents that pass through the pipeline.
  $match -, **$group** - group documents, **$project** - filters fields, **$sort** - sort objects **$count** -, **$limit** -, **$skip** -, The **$out** stage is the last stage in the pipeline and it writes the resulting documents to a specified collection.
  $expr can build query expressions that compare fields from the same document in a **$match stage**.
  $addFields - $addFields appends new fields to existing documents. You can include one or more $addFields stages in an aggregation operation.
  Unwind - $unwind Deconstructs an array field from the input documents to output a document for each element.

  { _id: 1, customer: "John", items: ["Apple", "Banana", "Orange"]}
  db.orders.aggregate([{ $unwind: "$items" }])

- **Aggregation Pipeline** - The aggregation pipeline refers to a specific flow of operations that processes, transforms, and returns results. In a pipeline, successive operations are informed by the previous result. **Let's take a typical pipeline: Input -> $match -> $group -> $sort -> output.**

- **Allowdiskuse** - all aggregation stages can use a maximum of 100 MB of RAM. Use allowDiskUse() to either allow or prohibit writing temporary files on disk when a pipeline stage exceeds the 100 MB limit. Syntax: {allowDiskUse: true}

- **Accumulators** - Accumulators are operators that maintain their state as documents progress through the pipeline. are **used in $group stage**
  $sum, $min, $max, $avg
- **Unary operators** - A unary operation is an operation with only one operand that performs an operation for each document. **used in $project stage**
  $type, $or, $lt, $gt, $and, $multiply

- **Lookup** - The $lookup operator is an aggregation operator or an aggregation stage, which is used to join a Document from one collection to a document of another collection of the same database.
  db.orders.aggregate([{**$lookup**:{
  from: "products",**localField**: "product_id",**foreignField**:"_id", **as**: "orderDetails"
  }}])

- **Mongo DB Utilities** - MongoDB utilities are a set of tools provided by MongoDB to manage and maintain MongoDB databases.
  - **Mongoexport** - Exports MongoDB collection data into JSON or CSV format
  - **Mongoimport** - Imports MongoDB collection data in JSON or CSV format
  - **Mongodump** - Mongodump is a MongoDB utility to create a backup of the content of a database in BSON format.
  - **Mongorestore** - used to restore the binary backups of MongoDB, indexes will be recreated.
  - **Mongostat** - real-time MongoDB statistics.
  - **Mongotop** - time is taken to track MongoDB's current read and write operations

- **Indexes** - In MongoDB, an index is a data structure that improves the performance of queries by allowing them to quickly locate the specific documents they are looking for.
  Default _id index is unique. **MongoDB uses a B-Tree index data structure.**
  {background: true} - Create an index in the background. other operations will not be blocked.
  Types of indexes - Single Field, Compound Index, Multikey Index, Geospatial Index, Text Search Indexes, Hashed Index, Clustered Indexes.
  - **Unique** - A unique index ensures that the indexed fields do not store duplicate values; i.e. enforces uniqueness for the indexed fields. By default, MongoDB creates a unique index on the _id field while creating a collection.

- **DRAWBACKS**
    - **Storage Overhead**: Indexes require additional storage space. For large databases or datasets, the size of the indexes can become significant. This can lead to increased storage costs.

    - **Maintenance Overhead**: Indexes need to be maintained as data is inserted, updated, or deleted. This maintenance can be resource-intensive, especially in high-transaction systems, and can slow down write operations.

    - **Complexity**: As the database grows and more indexes are added, the management and optimization of indexes can become complex. Database administrators need to carefully consider which columns to index and how to balance the benefits of indexing with the drawbacks.

- **Distinct()** - the distinct() method finds the distinct values for a given field across a single collection and returns the results in an array. Ex - db.person.distinct("eyeColor").

- **Capped collection** - **Fixed-size collections** are called capped collections in MongoDB. While creating a collection, the user must specify the collection's maximum size in bytes and the maximum number of documents it would store. **Syntax- isCapped()**

- **CAP Theorem** - The CAP theorem states that it is not possible to guarantee all three of the desirable properties – **C**onsistency, **A**vailability, and **P**artition tolerance at the same time in a distributed system with data replication. The theorem states that networked shared-data systems can only strongly support two of the following three properties: partition will always work.
- **MongoDB is a CP data store that resolves network partitions by maintaining consistency while compromising on availability**.

- **Replication** - MongoDB replication is **the process of creating a copy of the same data set in more than one MongoDB server**. This can be achieved by using a

Replica Set. A replica set is a group of MongoDB instances that maintain the same data set and pertain to any MongoDB process.

- **Replica sets** - Primary - (Read, Insert, Update)
  Secondary - (Read)

- **Normalization** - When you normalize your data, you are **dividing your data into multiple collections with references between those collections**.

- **Denormalization** - Denormalization is the technique of combining the data by reference to make data retrieval faster.

- **Sharding** - **Database sharding is splitting data into smaller chunks, called shards, and storing them across several database servers**. Ex - horizontal scaling
  Chunks - A chunk **consists of a subset of sharded data**.

  - **Vertical Scaling** - Vertical Scaling is defined as the ability to increase an existing system's capacity by adding resources. Ex - (RAM / CPU)

  - **Horizontal Scaling** - Horizontal Scaling is defined as the ability to extend capacity by interfacing different hardware or software entities.

- **Journaling** - Journaling in MongoDB is a feature that ensures the durability of data by maintaining an on-disk journal. The journal records all write operations before they are written to the main data files. This ensures that even if MongoDB crashes, data can be recovered from the journal, preventing data loss or corruption.

- **Atlas** - MongoDB Atlas is **a fully managed cloud database that handles all the complexity of deploying, managing, and healing your deployments. It provides features like automatic sharding, backup and recovery, monitoring, and scaling.**

- **Trigger** - Database Triggers allow you to execute server-side logic whenever a document is added, updated, or removed in a linked MongoDB Atlas cluster.

- **Alias -** In MongoDB, an alias is a name that you can use to refer to a field or a variable in a query or aggregation operation. Aliases allow you to change the name of the field or variable that is returned in the result set without modifying the original data.

- **Gridfs** - GridFS is **a specification for storing and retrieving files that exceed the BSON-document size limit of 16 MB**.

- **Namespace** - The namespace is a combination of the database name and the name of the collection or index, All documents belong to a namespace.