MongoDB.

- Database : A collection of organized info. on particular subject.

- Show databases

- use database.name

- db.coures

- db.find()

- db.courses.insertOne({name:"JS"})

* MongoDB.

- It is a document db.

- It stores data in type of JSON in the form of key value pair

- CRUD Operation

Create, Read, Update, Delete

- Create : db.collection.insertOne() / .insertMany()
- Read : db.collection.find()
- Updat : db.collection.updateOne() / updateMany() / replaceOne()
- Delet : db.collection.deleteOne() / deleteMany.

Date:- / /
Page No.:-

- Mongoose

- npm init -y
- npm i mongoose
- npm i express

- It is a library that simplifies interacting with MongoDB in Node.js application

- It provides a schema based soln for modelling & managing data. offering features like type casting, Validation & query building.

- Mongoose act as bridge b/w Node.js & MongoDB.

MongoDB Interview Questions

Last updated on Dec 23, 2024

Introduction to MongoDB

When dealing with data, there are two types of data as we know – (i) structured data and (ii) unstructured data. Structured data is usually stored in a tabular form whereas unstructured data is not. To manage huge sets of unstructured data like log or IoT data, a NoSQL database is used.

What is MongoDB ?

- MongoDB is an open-source NoSQL database written in C++ language. It uses JSON-like documents with optional schemas.
- It provides easy scalability and is a cross-platform, document-oriented database.
- MongoDB works on the concept of Collection and Document.
- It combines the ability to scale out with features such as secondary indexes, range queries, sorting, aggregations, and geospatial indexes.
- MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).

MongoDB Basic Interview Questions

1. What are some of the advantages of MongoDB?

Some advantages of MongoDB are as follows:

- MongoDB supports field, range-based, string pattern matching type queries. for searching the data in the database
- MongoDB support primary and secondary index on any fields
- MongoDB basically uses JavaScript objects in place of procedures
- MongoDB uses a dynamic database schema
- MongoDB is very easy to scale up or down
- MongoDB has inbuilt support for data partitioning (Sharding).

2. When to use MongoDB?

You should use MongoDB when you are building internet and business applications that need to evolve quickly and scale elegantly. MongoDB is popular with developers of all kinds who are building scalable applications using agile methodologies.
MongoDB is a great choice if one needs to:

- Support a rapid iterative development.
- Scale to high levels of read and write traffic - MongoDB supports horizontal scaling through Sharding, distributing data across several machines, and facilitating high throughput operations with large sets of data.
- Scale your data repository to a massive size.
- Evolve the type of deployment as the business changes.
- Store, manage and search data with text, geospatial, or time-series dimensions.

3. What are the data types in MongoDB?

MongoDB supports a wide range of data types as values in documents. Documents in MongoDB are similar to objects in JavaScript. Along with JSON's essential key/value–pair nature, MongoDB adds support for a number of additional data types. The common data types in MongoDB are:

- Null
  `{"x" : null}`
- Boolean
  `{"x" : true}`
- Number
  `{"x" : 4}`
- String
  `{"x" : "foobar"}`
- Date
  `{"x" : new Date()}`
- Regular expression
  `{"x" : /foobar/i}`
- Array
  `{"x" : ["a", "b", "c"]}`
- Embedded document
  `{"x" : {"foo" : "bar"}}`
- Object ID
  `{"x" : ObjectId()}`
- Binary Data
  Binary data is a string of arbitrary bytes.
- Code
  `{"x" : function() { /* ... */ }}`

- Evolve the type of deployment as the business changes.
- Store, manage and search data with text, geospatial, or time-series dimensions.

3. What are the data types in MongoDB?

MongoDB supports a wide range of data types as values in documents. Documents in MongoDB are similar to objects in JavaScript. Along with JSON's essential key/value–pair nature, MongoDB adds support for a number of additional data types. The common data types in MongoDB are:

- Null
  `{"x" : null}`
- Boolean
  `{"x" : true}`
- Number
  `{"x" : 4}`
- String
  `{"x" : "foobar"}`
- Date
  `{"x" : new Date()}`
- Regular expression
  `{"x" : /foobar/i}`
- Array
  `{"x" : ["a", "b", "c"]}`
- Embedded document
  `{"x" : {"foo" : "bar"}}`
- Object ID
  `{"x" : ObjectId()}`
- Binary Data
  Binary data is a string of arbitrary bytes.
- Code
  `{"x" : function() { /* ... */ }}`

4. How to perform queries in MongoDB?

The `find` method is used to perform queries in MongoDB. Querying returns a subset of documents in a collection, from no documents at all to the entire collection. Which documents get returned is determined by the first argument to `find`, which is a document specifying the query criteria.

Example:
`> db.users.find({"age" : 24})`

5. How do you Delete a Document?

The CRUD API in MongoDB provides `deleteOne` and `deleteMany` for this purpose. Both of these methods take a filter document as their first parameter. The filter specifies a set of criteria to match against in removing documents.

For example:
`> db.books.deleteOne({"_id" : 3})`

6. How do you Update a Document?

Once a document is stored in the database, it can be changed using one of several update methods: `updateOne`, `updateMany`, and `replaceOne`. `updateOne` and `updateMany` each takes a filter document as their first parameter and a modifier document, which describes changes to make, as the second parameter. `replaceOne` also takes a filter as the first parameter, but as the second parameter `replaceOne` expects a document with which it will replace the document matching the filter.

For example, in order to replace a document:

```
{
   "_id" : ObjectId("4b2b9f67a1f631733d917a7a"),
   "name" : "alice",
   "friends" : 24,
   "enemies" : 2
}
```

7. How to add data in MongoDB?

The basic method for adding data to MongoDB is "inserts". To insert a single document, use the collection's `insertOne` method:

```
> db.books.insertOne({"title" : "Start With Why"})
```

For inserting multiple documents into a collection, we use `insertMany`. This method enables passing an array of documents to the database.

8. What are some features of MongoDB?

- **Indexing:** It supports generic secondary indexes and provides unique, compound, geospatial, and full-text indexing capabilities as well.
- **Aggregation:** It provides an aggregation framework based on the concept of data processing pipelines.
- **Special collection and index types:** It supports time-to-live (TTL) collections for data that should expire at a certain time
- **File storage:** It supports an easy-to-use protocol for storing large files and file metadata.
- **Sharding:** Sharding is the process of splitting data up across machines.

9. How does Scale-Out occur in MongoDB?

For inserting multiple documents into a collection, we use `insertMany`. This method enables passing an array of documents to the database.

8. What are some features of MongoDB?

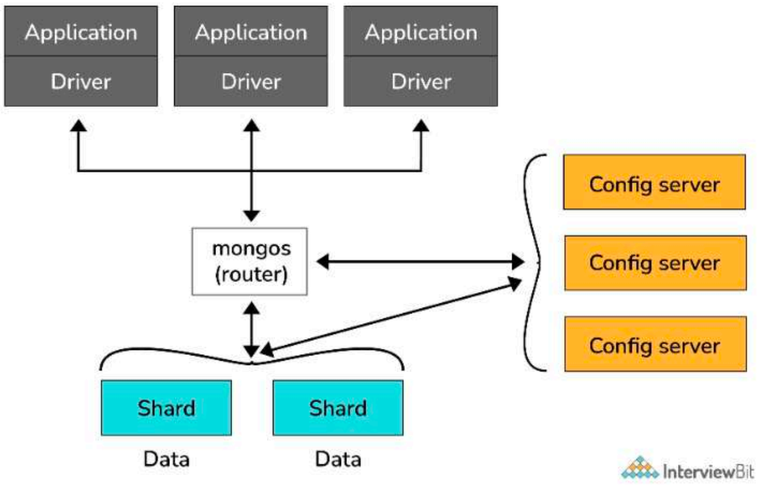
- **Indexing:** It supports generic secondary indexes and provides unique, compound, geospatial, and full-text indexing capabilities as well.
- **Aggregation:** It provides an aggregation framework based on the concept of data processing pipelines.
- **Special collection and index types:** It supports time-to-live (TTL) collections for data that should expire at a certain time
- **File storage:** It supports an easy-to-use protocol for storing large files and file metadata.
- **Sharding:** Sharding is the process of splitting data up across machines.

9. How does Scale-Out occur in MongoDB?

The document-oriented data model of MongoDB makes it easier to split data across multiple servers. Balancing and loading data across a cluster is done by MongoDB. It then redistributes documents automatically.

The mongos acts as a query router, providing an interface between client applications and the sharded cluster.

Config servers store metadata and configuration settings for the cluster. MongoDB uses the config servers to manage distributed locks. Each sharded cluster must have its own config servers.



10. What is the Mongo Shell?

It is a JavaScript shell that allows interaction with a MongoDB instance from the command line. With that one can perform administrative functions, inspecting an instance, or exploring MongoDB.

To start the shell, run the mongo executable:

```
$ mongod
$ mongo
MongoDB shell version: 4.2.0
connecting to: test
>
```

The shell is a full-featured JavaScript interpreter, capable of running arbitrary JavaScript programs. Let's see how basic math works on this:

```
> x = 100;
200
> x / 5;
20
```

11. What are Databases in MongoDB?

MongoDB groups collections into databases. MongoDB can host several databases, each grouping together collections.
Some reserved database names are as follows:
admin
local
config

12. What is a Collection in MongoDB?

A collection in MongoDB is a group of documents. If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.
Documents within a single collection can have any number of different "shapes.", i.e. collections have dynamic schemas.
For example, both of the following documents could be stored in a single collection:

```
{"greeting" : "Hello world!", "views": 3}
{"signoff": "Good bye"}
```

13. What is a Document in MongoDB?

A Document in MongoDB is an ordered set of keys with associated values.
JavaScript, documents are represented as objects:
`{"greeting" : "Hello world!"}`

```
{"signoff": "Good bye"}
```

13. What is a Document in MongoDB?

A Document in MongoDB is an ordered set of keys with associated values. It is represented by a map, hash, or dictionary. In JavaScript, documents are represented as objects:

```
{"greeting" : "Hello world!"}
```

Complex documents will contain multiple key/value pairs:

```
{"greeting" : "Hello world!", "views" : 3}
```

MongoDB Intermediate Interview Questions

14. How is Querying done in MongoDB?

The `find` method is used to perform queries in MongoDB. Querying returns a subset of documents in a collection, from no documents at all to the entire collection. Which documents get returned is determined by the first argument to find, which is a document specifying the query criteria.

For example: If we have a string we want to match, such as a "username" key with the value "alice", we use that key/value pair instead:

```
> db.users.find({"username" : "alice"})
```

15. Explain the SET Modifier in MongoDB?

If the value of a field does not yet exist, the "$set" sets the value. This can be useful for updating schemas or adding user-defined keys.

Example:

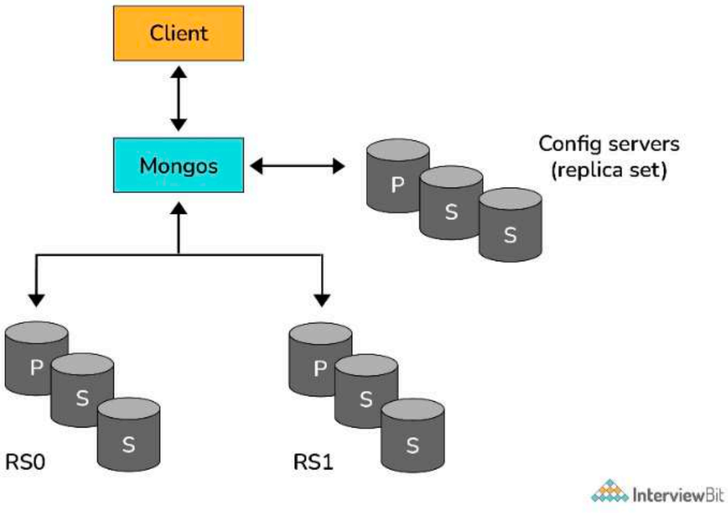
```
> db.users.findOne()
{
   "_id" : ObjectId("4b253b067525f35f94b60a31"),
   "name" : "alice",
   "age" : 23,
   "sex" : "female",
   "location" : "India"
}
```

To add a field to this, we use “$set”:

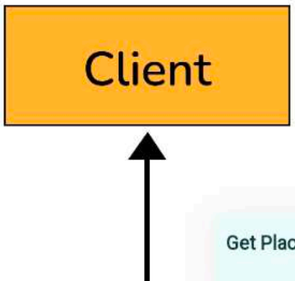
```
> db.users.updateOne({"_id" :
ObjectId("4b253b067525f35f94b60a31")},
... {"$set" : {"favorite book" : "Start with Why"}})
```

16. Explain the process of Sharding.

Sharding is the process of splitting data up across machines. We also use the term “partitioning” sometimes to describe this concept. We can store more data and handle more load without requiring larger or more powerful machines, by putting a subset of data on each machine.
In the figure below, RS0 and RS1 are shards. MongoDB’s sharding allows you to create a cluster of many machines (shards) and break up a collection across them, putting a subset of data on each shard. This allows your application to grow beyond the resource limits of a standalone server or replica set.

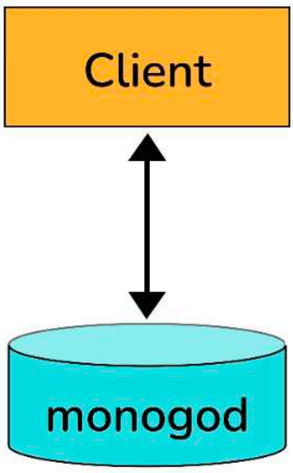


Non



RS0 RS1

Non



17. What are Geospatial Indexes in MongoDB?

MongoDB has two types of geospatial indexes: 2dsphere and 2d. 2dsphere indexes work with spherical geometries that model the surface of the earth based on the WGS84 datum. This datum model the surface of the earth as an oblate spheroid, meaning that there is some flattening at the poles. Distance calculations using 2sphere indexes, therefore, take the shape of the earth into account and provide a more accurate treatment of distance between, for example, two cities, than do 2d indexes. Use 2d indexes for points stored on a two-dimensional plane.

2dsphere allows you to specify geometries for points, lines, and polygons in the GeoJSON format. A point is given by a two-element array, representing [longitude, latitude]:

```
{
   "name" : "New York City",
   "loc" : {
       "type" : "Point",
       "coordinates" : [50, 2]
   }
}
```

A line is given by an array of points:

```
{
   "name" : "Hudson River",
   "loc" : {
       "type" : "LineString",
       "coordinates" : [[0,1], [0,2], [1,2]]
   }
}
```

18. Explain the term “Indexing” in MongoDB.

In MongoDB, indexes help in efficiently resolving queries. What an Index does is that it stores a small part of the data set in a form that is easy to traverse. The index stores the value of the specific field or set of fields, ordered by the value of the field as specified in the index.
MongoDB's indexes work almost identically to typical relational database indexes.

Indexes look at an ordered list with references to the content. These in turn allow MongoDB to query orders of magnitude faster. To create an index, use the `createIndex` collection method.

For example:

```
> db.users.find({"username": "user101"}).explain("executionStats")
```

Here, `executionStats` mode helps us understand the effect of using an index to satisfy queries.

MongoDB Advanced Interview Questions

19. What do you mean by Transactions?

A transaction is a logical unit of processing in a database that includes one or more database operations, which can be read or write operations. Transactions provide a useful feature in MongoDB to ensure consistency.

MongoDB provides two APIs to use transactions.

- **Core API:** It is a similar syntax to relational databases (e.g., start_transaction and commit_transaction)
- **Call-back API:** This is the recommended approach to using transactions. It starts a transaction, executes the specified operations, and commits (or aborts on the error). It also automatically incorporates error handling logic for "TransientTransactionError" and"UnknownTransactionCommitResult".

20. What are MongoDB Charts?

MongoDB Charts is a new, integrated tool in MongoDB for data visualizatic

operations, and commits (or aborts on the error). It also automatically incorporates error handling logic for "TransientTransactionError" and"UnknownTransactionCommitResult".

20. What are MongoDB Charts?

MongoDB Charts is a new, integrated tool in MongoDB for data visualization.

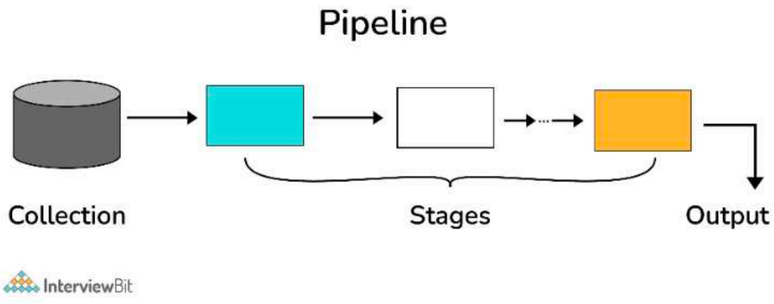
MongoDB Charts offers the best way to create visualizations using data from a MongoDB database.
It allows users to perform quick data representation from a database without writing code in a programming language such as Java or Python.

The two different implementations of MongoDB Charts are:

- MongoDB Charts PaaS (Platform as a Service)
- MongoDB Charts Server

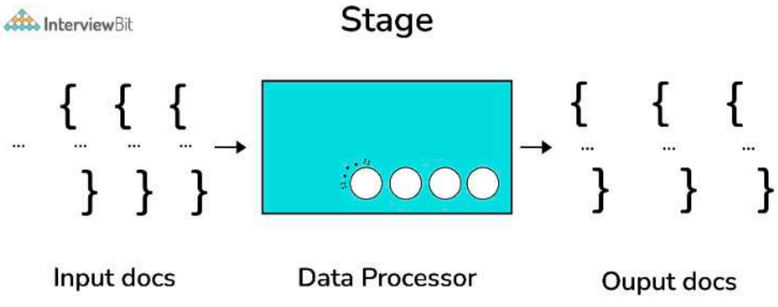
21. What is the Aggregation Framework in MongoDB?

- The aggregation framework is a set of analytics tools within MongoDB that allow you to do analytics on documents in one or more collections.
- The aggregation framework is based on the concept of a pipeline. With an aggregation pipeline, we take input from a MongoDB collection and pass the documents from that collection through one or more stages, each of which performs a different operation on its inputs (See figure below). Each stage takes as input whatever the stage before it produced as output. The inputs and outputs for all stages are documents—a stream of documents.



22. Explain the concept of pipeline in the MongoDB aggregation framework.

An individual stage of an aggregation pipeline is a data processing unit. It takes in a stream of input documents one at a time, processes each document one at a time, and produces an output stream of documents one at a time (see figure below).



23. What is a Replica Set in MongoDB?

To keep identical copies of your data on multiple servers, we use replication. It is recommended for all production deployments. Use replication to keep your application running and your data safe, even if something happens to one or more of your servers.

Such replication can be created by a replica set with MongoDB. A replica set is a group of servers with one primary, the server taking writes, and multiple secondaries, servers that keep copies of the primary's data. If the primary crashes, the secondaries can elect a new primary from amongst themselves.

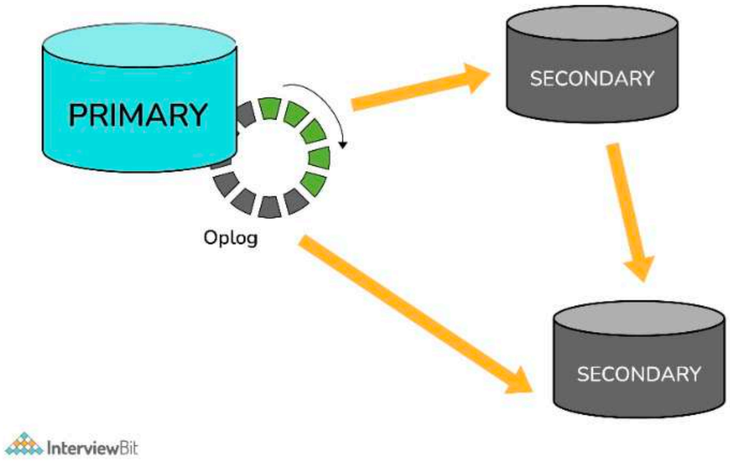
24. Explain the Replication Architecture in MongoDB.

The following diagram depicts the architecture diagram of a simple replica set cluster with only three server nodes – one primary node and two secondary nodes:

writes, and multiple secondaries, servers that keep copies of the primary's data. If the primary crashes, the secondaries can elect a new primary from amongst themselves.

24. Explain the Replication Architecture in MongoDB.

The following diagram depicts the architecture diagram of a simple replica set cluster with only three server nodes – one primary node and two secondary nodes:



- In the preceding model, the PRIMARY database is the only active replica set member that receives write operations from database clients. The PRIMARY database saves data changes in the Oplog. Changes saved in the Oplog are sequential—that is, saved in the order that they are received and executed.
- The SECONDARY database is querying the PRIMARY database for new changes in the Oplog. If there are any changes, then Oplog entries are copied from PRIMARY to SECONDARY as soon as they are created on the PRIMARY node.
- Then, the SECONDARY database applies changes from the Oplog to its own datafiles. Oplog entries are applied in the same order they were inserted in the log. As a result, datafiles on SECONDARY are kept in sync with changes on PRIMARY.
- Usually, SECONDARY databases copy data changes directly from PRIMARY. Sometimes a SECONDARY database can replicate data from another SECONDARY. This type of replication is called Chained Replication because it is a two-step replication process. Chained replication is useful in certain replication topologies, and it is enabled by default in MongoDB.

25. What are some utilities for backup and restore in MongoDB?

The mongo shell does not include functions for exporting, importing, backup, or restore. However, MongoDB has created methods for accomplishing this, so that no scripting work or complex GUIs are needed. For this, several utility scripts are provided that can be used to get data in or out of the database in bulk. These utility scripts are:

- mongoimport
- mongoexport
- mongodump
- mongorestore

Conclusion

26. Conclusion

MongoDB is a powerful, flexible, and scalable general-purpose database. It combines the ability to scale out with features such as secondary indexes, range queries, sorting, aggregations, and geospatial indexes.
Thus, in conclusion, MongoDB is:

- Supports Indexing
- Designed to scale
- Rich with Features
- High Performance
- Load Balancing
- Supports sharding

Although MongoDB is powerful, incorporating many features from relational systems, it is not intended to do everything that a relational database does. For some functionality, the database server offloads processing and logic to the client-side (handled either by the drivers or by a user's application code). Its maintenance of this streamlined design is one of the reasons MongoDB can achieve such high performance.

Here are few References to understand MongoDB in-depth:

- https://www.mongodb.com/2
- https://docs.mongodb.com

Recommended Tutorials:

- SQL Interview Questions
- SQL Server Interview Questions
- MySQL Interview Questions
- DBMS Interview Questions
- Database Testing Interview Questions
- MongoDB vs MySQL

the differences between SQL and **NoSQL databases**, how MongoDB structures and stores data, and basic operations like **CRUD** (Create, Read, Update, Delete).

1. What is MongoDB, and How Does It Differ from Traditional SQL Databases?

- **MongoDB** is a NoSQL database which means it does not use the **traditional table-based** relational database structure. Instead of it uses a flexible and document-oriented data model that stores data in **BSON** (Binary JSON) format.
- Unlike SQL databases that use rows and columns, MongoDB stores data as JSON-like documents, making it easier to handle unstructured data and providing greater flexibility in terms of schema design.

2. Explain BSON and Its Significance in MongoDB.

BSON (Binary JSON) is a **binary-encoded serialization** format used by MongoDB to store documents. BSON extends JSON by adding support for data types such as dates and binary data and it is designed to be efficient in both storage space and scan speed. The binary format allows MongoDB to be more efficient with data retrieval and storage compared to text-based JSON.

3. Describe the Structure of a MongoDB Document.

A MongoDB document is a set of **key-value** pairs similar to a JSON object. Each key is a string and the value can be a variety of data types including strings, numbers, arrays, nested documents and more.

Example:

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "name": "Alice",
  "age": 25,
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA"
  },
  "hobbies": ["reading", "cycling"]
}
```

4. What are Collections And Databases In MongoDB?

- **Database**: A container for collections, equivalent to a database in SQL.
- **Collection**: A group of documents, similar to tables in SQL, but schema-less.
- For example, a `users` collection can be part of the `mydatabase` database.

5. How Does MongoDB Ensure High Availability and Scalability?

- MongoDB ensures high **availability** and **scalability** through its features like **replica sets** and **sharding**.
- Replica sets provide redundancy and failover capabilities by ensuring that data is always available.
- Sharding distributes data across multiple servers, enabling horizontal scalability to handle large volumes of data and high traffic loads.

6. Explain the Concept of Replica Sets in MongoDB.

- A **replica set** in MongoDB is a group of mongod instances that maintain the same data set.
- A replica set consists of a primary node and multiple **secondary nodes**.
- The primary node receives all **write operations** while secondary nodes replicate the **primary's data** and can serve read operations.
- If the primary node fails, an automatic election process selects a new primary to maintain high availability.

7. What are the Advantages of Using MongoDB Over Other Databases?

- **Flexibility**: MongoDB's document-oriented model allows for dynamic schemas.
- **Scalability**: Built-in sharding enables horizontal scaling.
- **High Availability**: Replica sets provide redundancy and automatic failover.
- Performance: Efficient storage and retrieval of BSON format.

- If the primary node fails, an automatic election process selects a new primary to maintain high availability.

7. What are the Advantages of Using MongoDB Over Other Databases?

- **Flexibility**: MongoDB's document-oriented model allows for dynamic schemas.
- **Scalability**: Built-in sharding enables horizontal scaling.
- **High Availability**: Replica sets provide redundancy and automatic failover.
- **Performance**: Efficient storage and retrieval with BSON format.
- **Ease of Use**: JSON-like documents make it easier for developers to interact with data.

8. How to Create a New Database and Collection in MongoDB?

To create a new database and collection in MongoDB, you can use the mongo shell:


```
use mydatabase
db.createCollection("mycollection")
```

This command switches to **mydatabase** (creating it if it doesn't exist) and creates a new collection named **mycollection**.

9. What is Sharding, and How Does It Work in MongoDB?

Sharding is a method for distributing data across multiple servers in MongoDB. It allows for horizontal scaling by splitting large datasets into smaller, more manageable pieces called **shards**.

- Each shard is a separate database that holds a portion of the data.
- MongoDB automatically balances data and load across shards, ensuring efficient data distribution and high performance.

10. Explain the Basic Syntax of MongoDB CRUD Operations.

CRUD operations in MongoDB are used to create, read, update, and delete documents.

- **Create:** db.collection.insertOne({ name: "Alice", age: 25 })
- **Read:** db.collection.find({ name: "Alice" })
- **Update:** db.collection.updateOne({ name: "Alice" }, { $set: { age: 26 } })
- **Delete:** db.collection.deleteOne({ name: "Alice" })

11. How to Perform Basic Querying in MongoDB?

Basic querying in MongoDB involves using the find method to retrieve documents that match certain criteria.

Example:

```
db.collection.find({ age: { $gte: 20 } })
```

This query retrieves all documents from the collection where the **age** field is greater than or equal to 20.

12. What is an Index in MongoDB, and How to Create One?

An **index** in MongoDB is a data structure that improves the speed of data retrieval operations on a collection. You can create an index using the createIndex method.

For example, to create an index on the name field:

```
db.collection.createIndex({ name: 1 })
```

13. How Does MongoDB Handle Data Consistency?

MongoDB provides several mechanisms to ensure data consistency:

- **Journaling:** MongoDB uses write-ahead logging to maintain data integrity.
- **Write Concerns:** It specify the level of acknowledgment requested from MongoDB for write operations (e.g., acknowledgment from primary or secondaries).

```
db.collection.createIndex({ name: 1 })
```

13. How Does MongoDB Handle Data Consistency?

MongoDB provides several mechanisms to ensure data consistency:

- **Journaling:** MongoDB uses write-ahead logging to maintain data integrity.
- **Write Concerns:** It specify the level of acknowledgment requested from MongoDB for write operations (e.g., acknowledgment from primary only, or acknowledgment from primary and secondaries).
- **Replica Sets:** Replication ensures data is consistent across multiple nodes, and read concerns can be configured to ensure data consistency for read operations.

14. How to Perform Data Import and Export in MongoDB?

To perform data import and export in MongoDB, you can use the mongoimport and mongoexport tools. These tools allow you to import data from **JSON, CSV or TSV** files into MongoDB and export data from MongoDB collections to JSON or CSV files.

Import Data:

```
mongoimport --db mydatabase --collection mycollection --file data.json
```

This command imports data from data.json into the mycollection collection in the mydatabase database.

Export Data:

```
mongoexport --db mydatabase --collection mycollection --out data.json
```

This command exports data from the mycollection collection in the mydatabase database to data.json.

15. What are MongoDB Aggregation Pipelines and How are They Used?

The aggregation pipeline is a framework for data aggregation, modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into aggregated results. Each stage performs an operation on the input documents and passes the results to the next stage.

```
db.orders.aggregate([
  { $match: { status: "A" } },           // Stage 1: Filter documents by status
  { $group: { _id: "$cust_id", total: { $sum: "$amount" } } },  // Stage 2:
Group by customer ID and sum the amount
  { $sort: { total: -1 } }               // Stage 3: Sort by total in
descending order
])
```

In this example:

- Stage 1 ($match) filters documents by status "A".
- Stage 2 ($group) groups documents by customer ID and calculates the total amount for each group.
- Stage 3 ($sort) sorts the results by total amount in descending order.

Aggregation pipelines are powerful and flexible, enabling complex data processing tasks to be executed within MongoDB.

MongoDB Intermediate Interview Questions

MongoDB Intermediate Interview Questions explore **advanced concepts** and **features**, such as **schema design**, **aggregation pipelines**, **indexing strategies**, and **transaction management**. These questions help gauge your ability to utilize MongoDB efficiently in more complex scenarios.

1. Describe the Aggregation Framework in MongoDB

- The Aggregation Framework in MongoDB is a powerful tool for performing data processing and transformation on documents within a collection.
- It works by passing documents through a multi-stage pipeline, where each stage performs a specific operation on the data, such as **filtering, grouping, sorting, reshaping and computing aggregations.**
- This framework is particularly useful for creating complex data transformations and analytics directly within the database.

2. How to Perform Aggregation Operations Using MongoDB?

Aggregation operations in MongoDB are performed using the aggregate method. This method

- Stage 3 ($sort) sorts the results by total amount in descending order.

Aggregation pipelines are powerful and flexible, enabling complex data processing tasks to be executed within MongoDB.

MongoDB Intermediate Interview Questions

MongoDB Intermediate Interview Questions explore **advanced concepts** and **features**, such as **schema design**, **aggregation pipelines**, **indexing strategies**, and **transaction management**. These questions help gauge your ability to utilize MongoDB efficiently in more complex scenarios.

1. Describe the Aggregation Framework in MongoDB

- The Aggregation Framework in MongoDB is a powerful tool for performing data processing and transformation on documents within a collection.
- It works by passing documents through a multi-stage pipeline, where each stage performs a specific operation on the data, such as **filtering, grouping, sorting, reshaping and computing aggregations.**
- This framework is particularly useful for creating complex data transformations and analytics directly within the database.

2. How to Perform Aggregation Operations Using MongoDB?

Aggregation operations in MongoDB are performed using the aggregate method. This method takes an array of pipeline stages, each stage representing a step in the data processing pipeline.

Example: Calculate total sales for each product:

```
db.sales.aggregate([
  { $match: { status: "completed" } },  // Filter completed sales
  { $group: { _id: "$product", totalSales: { $sum: "$amount" } } },  // Group
by product and sum the sales amount
  { $sort: { totalSales: -1 } }  // Sort by total sales in descending order
])
```

3. Explain the Concept of Write Concern and Its Importance in MongoDB

Write Concern in MongoDB refers to the level of acknowledgment requested from MongoDB for write operations. It determines how many nodes must confirm the write operation before it is considered successful. Write concern levels range from "**acknowledged**" (default) to "**unacknowledged**," "**journaled**," and various "**replica acknowledged**" levels.

The importance of write concern lies in balancing between data durability and performance. Higher write concern ensures data is safely written to disk and replicated, but it may impact performance due to the added latency.

4. What are TTL Indexes, and How are They Used in MongoDB?

TTL (Time To Live) Indexes in MongoDB are special indexes that automatically remove documents from a collection after a certain period. They are commonly used for data that needs to expire after a specific time, such as **session information**, **logs**, or **temporary data**. To create a TTL index, you can specify the expiration time in seconds

Example: Remove documents 1 hour after `createdAt`:

```
db.sessions.createIndex({ "createdAt": 1 }, { expireAfterSeconds: 3600 })
```

This index will remove documents from the sessions collection 1 hour (3600 seconds) after the createdAt field's value.

5. How to Handle Schema Design and Data Modeling in MongoDB?

Schema design and data modeling in MongoDB involve defining how data is organized and stored in a document-oriented database. Unlike SQL databases, MongoDB offers flexible schema design, which can be both an advantage and a challenge. Key considerations for schema design include:

- **Embedding** vs. **Referencing**: Deciding whether to embed related data within a single document or use references between documents.
- **Document Structure**: Designing documents that align with application query patterns for efficient read and write operations.
- **Indexing**: Creating indexes to support query performance.
- **Data Duplication**: Accepting some level of data duplication to optimize for read performance.
- **Sharding**: Designing the schema to support sharding if horizontal scaling is required.

6. What is GridFS, and When is it Used in MongoDB?

GridFS is a specification for storing and retrieving large files in MongoDB. It is used when files exceed the BSON-document size limit of 16 MB. Use GridFS when you need to perform efficient retrieval of

- **Sharding**: Designing the schema to support sharding if horizontal scaling is required.

6. What is GridFS, and When is it Used in MongoDB?

GridFS is a specification for storing and retrieving large files in MongoDB. It is used when files exceed the BSON-document size limit of 16 MB or when you need to perform efficient retrieval of specific file sections.

GridFS splits a large file into smaller chunks and stores each chunk as a separate document within two collections: **fs.files** and **fs.chunks**. This allows for efficient storage and retrieval of large files, such as images, videos, or large datasets.

7. Explain the Differences Between WiredTiger and MMAPv1 Storage Engines

Feature	WiredTiger	MMAPv1
Concurrency	Document-level concurrency, allowing multiple operations simultaneously.	Collection-level concurrency, limiting performance under heavy write operations.
Compression	Supports data compression, reducing storage requirements.	Does not support data compression.
Performance	Generally offers better performance and efficiency for most workloads.	Limited performance, especially under heavy workloads.
Journaling	Uses write-ahead logging for better data integrity.	Basic journaling; less advanced than WiredTiger.
Status	Modern and default storage engine.	Legacy engine, deprecated in favor of WiredTiger.
Implementation	Advanced implementation with additional features.	Simple implementation but lacks advanced features.

8. How to Handle Transactions in MongoDB?

MongoDB supports multi-document ACID transactions by allowing us to perform a series of read and write operations across multiple documents and collections in a transaction. This ensures data consistency and integrity. To use transactions we typically start a session, begin a transaction, perform the operations and then **commit** or **abort** the transaction.

Example in JavaScript:

```
const session = client.startSession();

session.startTransaction();

try {
  db.collection1.insertOne({ name: "Alice" }, { session });
  db.collection2.insertOne({ name: "Bob" }, { session });
  session.commitTransaction();
} catch (error) {
  session.abortTransaction();
} finally {
  session.endSession();
}
```

9. Describe the MongoDB Compass Tool and Its Functionalities

MongoDB Compass is a graphical user interface (GUI) tool for MongoDB that provides an easy way to visualize, explore, and manipulate your data. It offers features such as:

- **Schema Visualization**: View and analyze your data schema, including field types and distributions.
- **Query Building**: Build and execute queries using a visual interface.
- **Aggregation Pipeline**: Construct and run aggregation pipelines.
- **Index Management**: Create and manage indexes to optimize query performance.
- **Performance Monitoring**: Monitor database performance, including slow queries and resource utilization.
- **Data Validation**: Define and enforce schema validation rules to ensure data integrity.
- **Data Import/Export**: Easily import and export data between MongoDB and JSON/CSV files.

10. What is MongoDB Atlas, and How Does It Differ From Self-Hosted MongoDB?

```
  session.abortTransaction();
} finally {
  session.endSession();
}
```

9. Describe the MongoDB Compass Tool and Its Functionalities

MongoDB Compass is a graphical user interface (GUI) tool for MongoDB that provides an easy way to visualize, explore, and manipulate your data. It offers features such as:

- **Schema Visualization**: View and analyze your data schema, including field types and distributions.
- **Query Building**: Build and execute queries using a visual interface.
- **Aggregation Pipeline**: Construct and run aggregation pipelines.
- **Index Management**: Create and manage indexes to optimize query performance.
- **Performance Monitoring**: Monitor database performance, including slow queries and resource utilization.
- **Data Validation**: Define and enforce schema validation rules to ensure data integrity.
- **Data Import/Export**: Easily import and export data between MongoDB and JSON/CSV files.

10. What is MongoDB Atlas, and How Does it Differ From Self-Hosted MongoDB?

MongoDB Atlas is a fully managed cloud database service provided by MongoDB. It offers automated deployment, scaling, and management of MongoDB clusters across various cloud providers (AWS, Azure, Google Cloud). Key differences from self-hosted MongoDB include:

- **Managed Service**: Atlas handles infrastructure management, backups, monitoring, and upgrades.
- **Scalability**: Easily scale clusters up or down based on demand.
- **Security**: Built-in security features such as encryption, access controls, and compliance certifications.
- **Global Distribution**: Deploy clusters across multiple regions for low-latency access and high availability.
- **Integrations**: Seamless integration with other cloud services and MongoDB tools.

11. How to Implement Access Control and User Authentication in MongoDB?

Access control and user authentication in MongoDB are implemented through a role-based access control (RBAC) system. You create users and assign roles that define their permissions. To set up access control:

- **Enable Authentication**: Configure MongoDB to require authentication by starting the server with --auth or setting security.authorization to enabled in the configuration file.
- **Create Users**: Use the db.createUser method to create users with specific roles.

```
db.createUser({
  user: "admin",
  pwd: "password",
  roles: [{ role: "userAdminAnyDatabase", db: "admin" }]
});
```

- **Assign Roles:** Assign roles to users that define their permissions, such as read, write, or admin roles for specific databases or collections.

12. What are Capped Collections, and When are They Useful?

Capped collections in MongoDB are fixed-size collections that automatically overwrite the oldest documents when the specified size limit is reached. They maintain insertion order and are useful for scenarios where you need to store a fixed amount of recent data, such as logging, caching, or monitoring data.

Example of creating a capped collection:

```
db.createCollection("logs", { capped: true, size: 100000 });
```

13. Explain the Concept of Geospatial Indexes in MongoDB

Geospatial indexes in MongoDB are special indexes that support querying of geospatial data, such as locations and coordinates. They enable efficient queries for proximity, intersections, and other spatial relationships. MongoDB supports two types of geospatial indexes: **2d** for **flat** geometries and **2dsphere** for spherical geometries.

Example of creating a 2dsphere index:

```
db.createCollection("logs", { capped: true, size: 100000 });
```

13. Explain the Concept of Geospatial Indexes in MongoDB

Geospatial indexes in MongoDB are special indexes that support querying of geospatial data, such as locations and coordinates. They enable efficient queries for proximity, intersections, and other spatial relationships. MongoDB supports two types of geospatial indexes: **2d** for **flat** geometries and **2dsphere** for spherical geometries.

Example of creating a 2dsphere index:

```
db.places.createIndex({ location: "2dsphere" });
```

14. How to Handle Backups and Disaster Recovery in MongoDB?

Handling backups and disaster recovery in MongoDB involves regularly creating backups of your data and having a plan for restoring data in case of failure. Methods include:

- **Mongodump/Mongorestore**: Use the mongodump and mongorestore utilities to create and restore binary backups.
- **File System Snapshots**: Use file system snapshots to take consistent backups of the data files.
- **Cloud Backups**: If using MongoDB Atlas, leverage automated backups provided by the service.
- **Replica Sets**: Use replica sets to ensure data redundancy and high availability. Regularly test the failover and recovery process.

15. Describe the Process of Upgrading MongoDB to a Newer Version

Upgrading MongoDB to a newer version involves several steps to ensure a smooth transition:

- **Check Compatibility**: Review the release notes and compatibility changes for the new version.
- **Backup Data**: Create a backup of your data to prevent data loss.
- **Upgrade Drivers**: Ensure that your application drivers are compatible with the new MongoDB version.
- Upgrade MongoDB: Follow the official MongoDB upgrade instructions, which typically involve stopping the server, installing the new version, and restarting the server.
- **Test Application**: Thoroughly test your application with the new MongoDB version to identify any issues.
- **Monitor**: Monitor the database performance and logs to ensure a successful upgrade.

16. What are Change Streams in MongoDB, and How are They Used?

Change Streams in MongoDB allow applications to listen for real-time changes to data in collections, databases, or entire clusters. They provide a powerful way to implement event-driven architectures by capturing insert, update, replace, and delete operations. To use Change Streams, you typically open a change stream cursor and process the change events as they occur.

Example:

```
const changeStream = db.collection('orders').watch();
changeStream.on('change', (change) => {
  console.log(change);
});
```

This example listens for changes in the orders collection and logs the change events.

17. Explain the Use of Hashed Sharding Keys in MongoDB

Hashed Sharding Keys in MongoDB distribute data across shards using a hashed value of the shard key field. This approach ensures an even distribution of data and avoids issues related to data locality or uneven data distribution that can occur with range-based sharding. Hashed sharding is useful for fields with monotonically increasing values, such as timestamps or identifiers.

Example:

```
db.collection.createIndex({ _id: "hashed" });
sh.shardCollection("mydb.mycollection", { _id: "hashed" });
```

18. How to Optimize MongoDB Queries for Performance?

Optimizing MongoDB queries involves several strategies:

- **Indexes:** Create appropriate indexes to support query patterns.
- **Query Projections:** Use projections to return only necessary fields.
- **Index Hinting:** Use index hints to force the query optimizer to use a specific index.
- Query Analysis: Use the explain() method to analyze query execution plans and identify

```
db.collection.createIndex({ _id: "hashed" });
sh.shardCollection("mydb.mycollection", { _id: "hashed" });
```

18. How to Optimize MongoDB Queries for Performance?

Optimizing MongoDB queries involves several strategies:

- **Indexes:** Create appropriate indexes to support query patterns.
- **Query Projections:** Use projections to return only necessary fields.
- **Index Hinting:** Use index hints to force the query optimizer to use a specific index.
- **Query Analysis:** Use the explain() method to analyze query execution plans and identify bottlenecks.
- **Aggregation Pipeline:** Optimize the aggregation pipeline stages to minimize data processing and improve efficiency.

19. Describe the Map-Reduce Functionality in MongoDB

Map-Reduce in **MongoDB** is a data processing paradigm used to perform complex data aggregation operations. It consists of two phases: the map phase processes each input document and emits key-value pairs, and the reduce phase processes all emitted values for each key and outputs the final result.

Example:

```
db.collection.mapReduce(
  function() { emit(this.category, this.price); },
  function(key, values) { return Array.sum(values); },
  { out: "category_totals" }
);
```

This example calculates the total price for each category in a collection.

20. What is the Role of Journaling in MongoDB, and How Does It Impact Performance?

Journaling in MongoDB ensures data durability and crash recovery by recording changes to the data in a journal file before applying them to the database files. This mechanism allows MongoDB to recover from unexpected shutdowns or crashes by replaying the journal. While journaling provides data safety, it can impact performance due to the additional I/O operations required to write to the journal file.

21. How to Implement Full-Text Search in MongoDB?

Full-Text Search in MongoDB is implemented using text indexes. These indexes allow you to perform text search queries on string content within documents.

Example:

```
db.collection.createIndex({ content: "text" });
db.collection.find({ $text: { $search: "mongodb" } });
```

In this example, a text index is created on the content field, and a text search query is performed to find documents containing the word "mongodb."

22. What are the Considerations for Deploying MongoDB in a Production Environment?

Considerations for deploying MongoDB in a production environment include:

- **Replication**: Set up replica sets for high availability and data redundancy.
- **Sharding**: Implement sharding for horizontal scaling and to distribute the load.
- **Backup and Recovery**: Establish a robust backup and recovery strategy.
- **Security**: Implement authentication, authorization, and encryption.
- **Monitoring**: Use monitoring tools to track performance and detect issues.
- **Capacity Planning**: Plan for adequate storage, memory, and CPU resources.
- **Maintenance**: Regularly update MongoDB to the latest stable version and perform routine maintenance tasks.

23. Explain the Concept of Horizontal Scalability and Its Implementation in MongoDB

- **Horizontal Scalability** in MongoDB refers to the ability to add more servers to distribute the load and data. This is achieved through sharding, where data is partitioned across multiple shards.
- Each shard is a replica set that holds a subset of the data. Sharding allows MongoDB to handle large datasets and high-throughput applications by distributing the workload.

- **Capacity Planning**: Plan for adequate storage, memory, and CPU resources.
- **Maintenance**: Regularly update MongoDB to the latest stable version and perform routine maintenance tasks.

23. Explain the Concept of Horizontal Scalability and Its Implementation in MongoDB

- **Horizontal Scalability** in MongoDB refers to the ability to add more servers to distribute the load and data. This is achieved through sharding, where data is partitioned across multiple shards.
- Each shard is a replica set that holds a subset of the data. Sharding allows MongoDB to handle large datasets and high-throughput operations by distributing the workload.

24. How to Monitor and Troubleshoot Performance Issues in MongoDB?

Monitoring and troubleshooting performance issues in MongoDB involve:

- **Monitoring Tools**: Use tools like MongoDB Cloud Manager, MongoDB Ops Manager, or third-party monitoring solutions.
- **Logs**: Analyze MongoDB logs for errors and performance metrics.
- **Profiling**: Enable database profiling to capture detailed information about operations.
- **Explain Plans**: Use the explain() method to understand query execution and identify bottlenecks.
- **Index Analysis**: Review and optimize indexes based on query patterns and usage.
- **Resource Utilization**: Monitor CPU, memory, and disk I/O usage to identify resource constraints.

25. Describe the Process of Migrating Data from a Relational Database to MongoDB

Migrating data from a relational database to MongoDB involves several steps:

- **Schema Design**: Redesign the relational schema to fit MongoDB's document-oriented model. Decide on embedding vs. referencing, and plan for indexes and collections.
- **Data Export**: Export data from the relational database in a format suitable for MongoDB (e.g., CSV, JSON).
- **Data Transformation**: Transform the data to match the MongoDB schema. This can involve converting data types, restructuring documents, and handling relationships.
- **Data Import**: Import the transformed data into MongoDB using tools like mongoimport or custom scripts.
- **Validation**: Validate the imported data to ensure consistency and completeness.
- **Application Changes**: Update the application code to interact with MongoDB instead of the relational database.
- **Testing**: Thoroughly test the application and the database to ensure everything works as expected.
- **Go Live**: Deploy the MongoDB database in production and monitor the transition.

MongoDB Query Based Interview Questions

MongoDB Query-Based Interview Questions focus on your ability to write efficient and optimized queries to interact with databases. These tasks include retrieving specific data using **filters**, **sorting** and **paginating results**, and **utilizing projections** to select desired fields. Below is a sample dataset we'll use to demonstrate various queries.

Sample Dataset

The following dataset represents a collection named **`employees`**, containing documents about employees in an organization. Each document includes details such as the **employee's name**, **age**, **position**, **salary**, **department**, and hire date.

```
"[
    {
        ""_id"": 1,
        ""name"": ""John Doe"",
        ""age"": 28,
        ""position"": ""Software Engineer"",
        ""salary"": 80000,
        ""department"": ""Engineering"",
        ""hire_date"": ISODate(""2021-01-15"")
    },
    {
        ""_id"": 2,
        ""name"": ""Jane Smith"",
        ""age"": 34,
        ""position"": ""Project Manager"",
        ""salary"": 95000,
        ""department"": ""Engineering"",
        ""hire_date"": ISODate(""2019-06-23"")
    },
```

The following dataset represents a collection named **employees**, containing documents about employees in an organization. Each document includes details such as the **employee's name**, **age**, **position**, **salary**, **department**, and hire date.

```
"[
    {
        ""_id"": 1,
        ""name"": ""John Doe"",
        ""age"": 28,
        ""position"": ""Software Engineer"",
        ""salary"": 80000,
        ""department"": ""Engineering"",
        ""hire_date"": ISODate(""2021-01-15"")
    },
    {
        ""_id"": 2,
        ""name"": ""Jane Smith"",
        ""age"": 34,
        ""position"": ""Project Manager"",
        ""salary"": 95000,
        ""department"": ""Engineering"",
        ""hire_date"": ISODate(""2019-06-23"")
    },
    {
        ""_id"": 3,
        ""name"": ""Emily Johnson"",
        ""age"": 41,
        ""position"": ""CTO"",
        ""salary"": 150000,
        ""department"": ""Management"",
        ""hire_date"": ISODate(""2015-03-12"")
    },
    {
        ""_id"": 4,
        ""name"": ""Michael Brown"",
        ""age"": 29,
        ""position"": ""Software Engineer"",
        ""salary"": 85000,
        ""department"": ""Engineering"",
        ""hire_date"": ISODate(""2020-07-30"")
    },
    {
        ""_id"": 5,
        ""name"": ""Sarah Davis"",
        ""age"": 26,
        ""position"": ""UI/UX Designer"",
        ""salary"": 70000,
        ""department"": ""Design"",
        ""hire_date"": ISODate(""2022-10-12"")
    }
]"
```

1. Find all Employees Who Work in the "Engineering" Department.

Query:

```
db.employees.find({ department: "Engineering" })
```

Output:

```
[
    {
        "_id": 1,
        "name": "John Doe",
        "age": 28,
        "position": "Software Engineer",
        "salary": 80000,
        "department": "Engineering",
        "hire_date": ISODate("2021-01-15")
    },
    {
        "_id": 2,
        "name": "Jane Smith",
        "age": 34,
        "position": "Project Manager",
        "salary": 95000,
        "department": "Engineering",
        "hire_date": ISODate("2019-06-23")
```

```
        "hire_date": ISODate("2022-10-12")
    }
]"
```

1. Find all Employees Who Work in the "Engineering" Department.

Query:

```
db.employees.find({ department: "Engineering" })
```

Output:

```
[
    {
        "_id": 1,
        "name": "John Doe",
        "age": 28,
        "position": "Software Engineer",
        "salary": 80000,
        "department": "Engineering",
        "hire_date": ISODate("2021-01-15")
    },
    {
        "_id": 2,
        "name": "Jane Smith",
        "age": 34,
        "position": "Project Manager",
        "salary": 95000,
        "department": "Engineering",
        "hire_date": ISODate("2019-06-23")
    },
    {
        "_id": 4,
        "name": "Michael Brown",
        "age": 29,
        "position": "Software Engineer",
        "salary": 85000,
        "department": "Engineering",
        "hire_date": ISODate("2020-07-30")
    }
]
```

Explanation: This query finds all employees whose department field is "Engineering".

2. Find the Employee with the Highest Salary.

Query:

```
db.employees.find().sort({ salary: -1 }).limit(1)
```

Output:

```
[
    {
        "_id": 3,
        "name": "Emily Johnson",
        "age": 41,
        "position": "CTO",
        "salary": 150000,
        "department": "Management",
        "hire_date": ISODate("2015-03-12")
    }
]
```

Explanation: This query sorts all employees by salary in descending order and retrieves the top document, which is the employee with the highest salary.

3. Update the Salary of "John Doe" to 90000.

Query:

```
db.employees.updateOne({ name: "John Doe" }, { $set: { salary: 90000 } })
```

Output:

```
        "hire_date": ISODate("2015-03-12")
    }
]
```

Explanation: This query sorts all employees by salary in descending order and retrieves the top document, which is the employee with the highest salary.

3. Update the Salary of "John Doe" to 90000.

Query:

```
db.employees.updateOne({ name: "John Doe" }, { $set: { salary: 90000 } })
```

Output:

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Explanation: This query updates the salary of the employee named "John Doe" to 90000.

4. Count the Number of Employees in Each Department.

Query:

```
db.employees.aggregate([
    { $group: { _id: "$department", count: { $sum: 1 } } }
])
```

Output:

```
[
    { "_id": "Engineering", "count": 3 },
    { "_id": "Management", "count": 1 },
    { "_id": "Design", "count": 1 }
]
```

Explanation: This query groups the employees by the department field and counts the number of employees in each department.

5. Add a New Field Bonus to All Employees in the "Engineering" Department with a Value of 5000.

Query:

```
db.employees.updateMany({ department: "Engineering" }, { $set: { bonus: 5000 } })
```

Output:

```
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
```

Explanation: This query adds a new field bonus with a value of 5000 to all employees in the "Engineering" department.

6. Retrieve All Documents in the Employees Collection and Sort Them by the Length of Their Name in Descending Order.

Query:

```
db.employees.aggregate([
    { $addFields: { nameLength: { $strLenCP: "$name" } } },
    { $sort: { nameLength: -1 } },
    { $project: { nameLength: 0 } }
])
```

Output:

```
[
    {
        "_id": 2,
        "name": "Jane Smith",
        "age": 34,
        "position": "Project Manager",
        "salary": 95000,
```

Explanation: This query adds a new field bonus with a value of 5000 to all employees in the "Engineering" department.

6. Retrieve All Documents in the Employees Collection and Sort Them by the Length of Their Name in Descending Order.

Query:

```
db.employees.aggregate([
    { $addFields: { nameLength: { $strLenCP: "$name" } } },
    { $sort: { nameLength: -1 } },
    { $project: { nameLength: 0 } }
])
```

Output:

```
[
    {
        "_id": 2,
        "name": "Jane Smith",
        "age": 34,
        "position": "Project Manager",
        "salary": 95000,
        "department": "Engineering",
        "hire_date": ISODate("2019-06-23")
    },
    {
        "_id": 3,
        "name": "Emily Johnson",
        "age": 41,
        "position": "CTO",
        "salary": 150000,
        "department": "Management",
        "hire_date": ISODate("2015-03-12")
    },
    {
        "_id": 1,
        "name": "John Doe",
        "age": 28,
        "position": "Software Engineer",
        "salary": 80000,
        "department": "Engineering",
        "hire_date": ISODate("2021-01-15")
    },
    {
        "_id": 4,
        "name": "Michael Brown",
        "age": 29,
        "position": "Software Engineer",
        "salary": 85000,
        "department": "Engineering",
        "hire_date": ISODate("2020-07-30")
    },
    {
        "_id": 5,
        "name": "Sarah Davis",
        "age": 26,
        "position": "UI/UX Designer",
        "salary": 70000,
        "department": "Design",
        "hire_date": ISODate("2022-10-12")
    }
]
```

Explanation: This query calculates the length of each employee's name, sorts the documents by this length in descending order, and removes the temporary nameLength field from the output.

7. Find the Average Salary of Employees in the "Engineering" Department.

Query:

```
db.employees.aggregate([
    { $match: { department: "Engineering" } },
    { $group: { _id: null, averageSalary: { $avg: "$salary" } } }
])
```

Output:

```
]
```

Explanation: This query calculates the length of each employee's name, sorts the documents by this length in descending order, and removes the temporary nameLength field from the output.

7. Find the Average Salary of Employees in the "Engineering" Department.

Query:

```
db.employees.aggregate([
    { $match: { department: "Engineering" } },
    { $group: { _id: null, averageSalary: { $avg: "$salary" } } }
])
```

Output:

```
[
    { "_id": null, "averageSalary": 86666.66666666667 }
]
```

Explanation: This query filters employees to those in the "Engineering" department and calculates the average salary of these employees.

8. Find the Department with the Highest Average Salary.

Query:

```
db.employees.aggregate([
    { $group: { _id: "$department", averageSalary: { $avg: "$salary" } } },
    { $sort: { averageSalary: -1 } },
    { $limit: 1 }
])
```

Output:

```
[
    { "_id": "Management", "averageSalary": 150000 }
]
```

Explanation: This query groups employees by department, calculates the average salary for each department, sorts these averages in descending order, and retrieves the department with the highest average salary.

9. Find the Total Number of Employees Hired in Each Year.

Query:

```
db.employees.aggregate([
    { $group: { _id: { $year: "$hire_date" }, totalHired: { $sum: 1 } } }
])
```

Output:

```
[
    { "_id": 2015, "totalHired": 1 },
    { "_id": 2019, "totalHired": 1 },
    { "_id": 2020, "totalHired": 1 },
    { "_id": 2021, "totalHired": 1 },
    { "_id": 2022, "totalHired": 1 }
]
```

Explanation: This query groups employees by the year they were hired, which is extracted from the hire_date field, and counts the total number of employees hired each year.

10. Find the Highest and Lowest Salary in the "Engineering" Department.

Query:

```
db.employees.aggregate([
    { $match: { department: "Engineering" } },
    {
        $group: {
            _id: null,
            highestSalary: { $max: "$salary" },
            lowestSalary: { $min: "$salary" }
```

```
    { "_id": 2021, "totalHired": 1 },
    { "_id": 2022, "totalHired": 1 }
]
```

Explanation: This query groups employees by the year they were hired, which is extracted from the hire_date field, and counts the total number of employees hired each year.

10. Find the Highest and Lowest Salary in the "Engineering" Department.

Query:

```
db.employees.aggregate([
    { $match: { department: "Engineering" } },
    {
        $group: {
            _id: null,
            highestSalary: { $max: "$salary" },
            lowestSalary: { $min: "$salary" }
        }
    }
])
```

Output:

```
[
    { "_id": null, "highestSalary": 95000, "lowestSalary": 80000 }
]
```

Explanation: This query filters employees to those in the "Engineering" department, then calculates the highest and lowest salary within this group.

Conclusion

Preparing for a **MongoDB interview** becomes much simpler with the right resources. This guide has provided a **detailed collection** of MongoDB interview questions, covering **fundamental concepts**, **advanced topics**, and **real-world applications**. By practicing these questions, you'll enhance your knowledge of MongoDB, strengthen your **problem-solving skills**, and build the confidence needed to excel in your interview. Stay consistent in your preparation, and you'll be ready to tackle any **MongoDB challenge** that comes your way!

Next Article
Top 50 MongoDB Interview Questions with Answers for 2025

Similar Reads

1. Top 50 Database Interview Questions and Answers for 2025
2. Top 50 PostgreSQL Interview Questions and Answers
3. Top 50 SQLite Interview Questions for 2024
4. 30 SQL Interview Questions For Business Analyst in 2025
5. Top 10 MongoDB Tools for 2025
6. How to Manage Data with MongoDB
7. MongoDB Roadmap: A Complete Guide [2025 Updated]
8. A Better Developer Experience with the MongoDB Atlas Data API
9. How to Migrate a Microservice from MySQL to MongoDB
10. What is the Format of Document in MongoDB?