NoSQL, MongoDB with ODM

Interview Questions

(Practice Project)

PW SKILLS

Interview Questions

Easy

Q. Define NoSQL and MongoDB

Ans -
NoSQL is a type of database designed to handle a wide variety of data models, including key-value, document, columnar, and graph formats. It is often used for its flexibility, scalability, and performance advantages over traditional relational databases.
MongoDB is a NoSQL, document-oriented database that stores data in flexible, JSON-like documents. It is known for its high performance, high availability, and easy scalability.

Q. What are the advantages/benefits of using MongoDB

Ans -
The advantages of using MongoDB include -

1. Flexible Data model
2. Scalability
3. Speed and Performance
4. Real-time analytics
5. Internet of Things (IoT) applications

Q. Differentiate between collections, documents, and fields in MongoDB
Ans -

Feature	Description	Relational Database Equivalent
Collection	A container that holds documents. Similar to a table in a relational database, but with some key differences.	Table
Document	A flexible data structure that stores information about a single entity. Documents are like records in a table, but they can have different fields (columns) and can embed other documents.	Record (Row)
Field	A key-value pair within a document. Similar to a column in a table, fields can hold various data types and a document can have different sets of fields.	Column

Q. What are Databases in MongoDB

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

- admin
- local
- config

Q. What does ODM stand for and define Mongoose

Ans -
The ODM stands for Object Data Modeling and Mongoose is an ODM library for MongoDB and Node.js. It provides a schema-based solution to model data, including built-in type casting, validation, query building, and business logic hooks.

Q. How do you create an index on a specific field in a MongoDB collection using the createIndex command?

Ans:
To create an index on a specific field in a MongoDB collection, you can use the createIndex command with the db.collection.createIndex() method. For example, to create a single-field ascending index on the "name" field in a "users" collection, you can use the following command:

Example

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

Q. Explain how the db.collection.find() command works in MongoDB and provides an example of its usage for querying documents.

Ans:
The db.collection.find() command is used to query documents in a MongoDB collection. It allows you to specify query criteria and retrieve documents that match those criteria. For example, to find all documents in the "orders" collection where the "status" field is "shipped," you can use the following command:

Example -

```
db.orders.find({status: "shipped"})
```

Q. What is the db.collection.update() command used in MongoDB, and how do you use it to update documents in a collection?

Ans:
The db.collection.update() command is used to modify existing documents in a MongoDB collection. To update documents, you provide a query to identify the documents to be updated and specify the changes to be made. For example, to update the "quantity" field of all documents in the "products" collection with a "category" of "electronics" to 50, you can use the following command:

Example -

```
db.products.update({category: "electronics"}, {$set:
{quantity:50}}, {multi:true})
```

Q. Describe the purpose of the db.collection.aggregate() command in MongoDB and provide an example of an aggregation pipeline.

Ans:
The db.collection.aggregate() command in MongoDB is used for data transformation and aggregation. It allows you to define a sequence of operations in an aggregation pipeline to process and analyze data.
For example, to find the average price of products in the "electronics" category in the "products" collection, you can use the following aggregation pipeline:

Example -

```
db.products.aggregate([
{$match: {category:"electronics"}},
{$group: {_id: null, avgPrice: {$avg: "$price"}}}
])
```

Q. How do you use the db.collection.remove() command to delete documents from a MongoDB collection, and what options can you specify with it for deletion criteria?

Ans:
The db.collection.remove() command is used to delete documents from a MongoDB collection. To specify deletion criteria, you provide a query object. For example, to delete all documents in the "customers" collection with a "status" of "inactive," you can use the following command:

Example -

```
db.customers.remove({status: "inactive"})
```

Medium

Q. What is Mongoose referencing and how does it relate to MongoDB?

Ans:
Mongoose referencing serves as a mechanism for establishing connections between various data models in MongoDB, much like the way we establish relationships using foreign keys in relational databases. Mongoose referencing encompasses two primary types:

1. **Reference-Based Approach:** In this approach, we store only the ObjectID of the referenced document. To access the actual document, a secondary database query is required. This approach prioritizes efficiency in write operations but may be relatively slower when it comes to read operations.
2. **Value-Based (Embedded Documents) Approach:** This method involves storing the actual object within the parent document, resulting in quicker read operations. However, it may introduce some delays in write operations due to the embedded data structure.

Q. How do you create a reference between two schemas in Mongoose?

Ans:
In Mongoose, establishing a connection between two schemas involves defining the ObjectId type within the schema where the reference will be stored. Additionally, you specify a 'ref' option, which should indicate the name of the schema to which the ObjectId will be pointing.

Example:-
For instance, consider a straightforward scenario where we have two models, namely 'User' and 'Post,' and a user can possess multiple posts. In such a case, we could structure our models as follows:

```
import mongoose, { Document, Schema } from 'mongoose'

interface IUser extends Document {
name: string;
posts: IPost['_id'][];
}

interface IPost extends Document {
title: string;
content: string;
}

const postSchema= new Schema<IPost>({
title: String,
content: String
});

const userSchema= new Schema<IUser>({

name: String,
posts: [{ type: Schema. Types.ObjectId, ref: 'Post' }]
});

const User = mongoose.model<IUser>('User', userSchema);
const Post = mongoose.model<IPost>('Post', postSchema);
```

In this example, every user document will contain an array of ObjectIDs corresponding to the posts they have authored.

Q. How do you retrieve/populate referenced documents in Mongoose?

Ans:
To fetch or load referenced documents in Mongoose, the .populate() method comes into play. With this method, you can substitute the designated path within the document, where the reference ObjectId is stored, with the real referenced document.

For instance, consider a straightforward scenario where we have two models, namely 'User' and 'Post,' and a user can possess multiple posts. In this example, every user document will contain an array of ObjectIds corresponding to the posts they have authored. Now, let's try to retrieve all users along with their posts.

```
User
.findOne({ name: 'John Doe' })
.populate('posts')
.exec((err: Error, user: IUser) => {
if (err) throw err;

console.log('The user is %s', user.name); console.log('The
posts are %s', user.posts);

});
```

Q. Explain the structure of ObjectID in MongoDB

Ans:
ObjectID is a 12-byte BSON type. These are:

- 4 bytes value representing seconds
- 3 byte machine identifier
- 2 byte process id
- 3 byte counter

Q. Can you please explain the difference between a relational database and a NoSQL database like mongoDB

Ans -
Relational databases use structured query language (SQL) and have a predefined schema with tables, rows, and columns. NoSQL databases like MongoDB use flexible, schema-less structures, such as collections and documents, and can handle unstructured and semi-structured data.

Q. How do you define Schema in Mongoose?

Ans -

```
const schemaName = new mongoose.Schema({
  name: {
    first: String,
    last: { type: String, trim: true }
  },
  age: { type: Number, min: 0 },
  posts: [ { title: String, url: String, date: Date } ],
  updated: { type: Date, default: Date.now }
});
```

Q. What do you understand by replica set in MongoDB?

Ans
A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and high availability. A replica set consists of a primary node and one or more secondary nodes that replicate data from the primary.

Q. Can you some of the benefits of Replica set in MongoDB

Ans -
Some of the benefits of Replica set include -

1. Data Redundancy and High Availability
2. Disaster Recovery
3. Automatic Failover
4. Read Scaling

Q. Mention some of the benefits of Sharding

Ans -
The benefits of Shariding include -

1. Balance Data Distribution
2. Flexible deployment
3. Write Scalability
4. Query Isolation

Q. Tell me something about transactions in MongoDB

Ans -
Transactions in MongoDB allow you to group multiple operations together as a single atomic unit. They provide ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure data integrity and consistency when working with multiple documents or collections

Key points about transactions in MongoDB include -

- **Atomicity:** All operations in a transaction must succeed for the transaction to complete successfully. If any operation fails, the entire transaction is aborted and rolled back.
- **Consistency:** Transactions help maintain data consistency by ensuring that the database moves from one valid state to another.
- **Isolation:** Transactions provide isolation, meaning that changes made within a transaction are not visible to other operations until the transaction is committed.
- **Durability:** Once a transaction is committed, the changes are durable and will persist even in the event of a system failure.
- **Multi-Document Transactions:** MongoDB supports transactions across multiple documents, collections, and databases.
- **Distributed Transactions:** MongoDB also supports distributed transactions, which are transactions that span multiple shards

Here is an example of performing transactions in mongoDB

```
const performTransaction = async () => {
  const session = await mongoose.startSession();
  session.startTransaction();

  try {
    // Perform operations
    const user = new User({ name: 'Jane Doe', email:
'jane@example.com', age: 25 });
    await user.save({ session });

    const order = new Order({ userId: user._id, product:
'Laptop', quantity: 1 });
    await order.save({ session });

    // Commit the transaction
    await session.commitTransaction();
    console.log('Transaction committed.');
  } catch (error) {
    // Abort the transaction
    await session.abortTransaction();
    console.error('Transaction aborted due to error:',
error);
  } finally {
    session.endSession();
  }
};

performTransaction();
```

Hard

Q. What are Indexes in MongoDB?

Ans:
Within MongoDB, indexes serve as tools for optimizing query performance. In the absence of indexes, MongoDB is compelled to execute a collection scan, meaning it needs to examine each document in a collection to identify those that satisfy the query conditions. When a suitable index is available for a query, MongoDB can leverage that index to restrict the number of documents it needs to scrutinize.

Q. Why Profiler is used in MongoDB?

Ans:
MongoDB employs a database profiler to analyze the attributes of each operation carried out on the database. This profiler can be utilized to discover and inspect queries and write operations.

Profiling can be configured to capture specific types of database operations, like slow queries or all queries, and you can set various profiling levels (0, 1, 2) to control the amount of detail in the profiler output. It's a useful tool for both performance optimization and troubleshooting in MongoDB.

Q. What is Sharding in MongoDB?

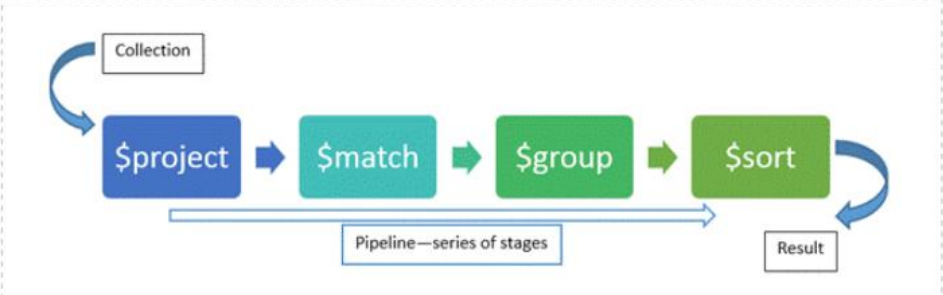
Ans:
Sharding in MongoDB is a technique used to horizontally partition a large database into smaller, more manageable pieces called "shards." It is a method of distributing data across multiple servers or clusters, allowing MongoDB to efficiently store and retrieve data as the dataset grows beyond the capacity of a single server.

For more information - https://www.mongodb.com/docs/manual/sharding/

Q. What is the Aggregation Framework in MongoDB?

Ans:
The aggregation framework in MongoDB comprises a suite of analytical tools enabling the analysis of documents in one or multiple collections.

It operates on the principle of a pipeline, where documents from a MongoDB collection are processed through a series of stages, each performing distinct operations on the input data (refer to the diagram below). In this pipeline, the output of one stage serves as the input for the subsequent stage. Both the inputs and outputs for all stages are in the form of documents, creating a continuous flow of document data.



Q. Given the MongoDB command: db.collection.find({ age: { $gt: 30 } }).limit(3).sort({ age: -1 }), what will this command return?

Ans:
This command will return the first three documents from the collection where the "age" field is greater than 30, sorted in descending order of age.

Q. Using the MongoDB command: db.collection.aggregate([{ $group: { _id: "$city", avgSalary: { $avg: "$salary" } } }]), what will be the result of this aggregation pipeline?

Ans:
The result will be a list of documents, each representing a distinct "city" value in the collection, along with the average "salary" for each city.

Q. Given the MongoDB command: db.collection.aggregate([{ $match: { status: "active" } }, { $group: { _id: "$category", totalProducts: { $sum: 1 } } }]), what will the result of this aggregation pipeline be?

Ans:
The result will be a list of documents, where each document represents a "category" with the total count of documents having a "status" of "active" in that category.

Q. Using the MongoDB command: db.collection.distinct("category"), what will this command return for a collection with various product categories?

Ans:
This command will return an array containing the distinct "category" values found in the collection, without duplicates.

Q. Given the MongoDB command: db.collection.stats(), what information will be returned when querying the statistics of a specific collection in the database?

Ans:
This command will return various statistics about the collection, including the number of documents, storage size, index details, and more, providing insights into the collection's characteristics and usage.