

MongoDB and NoSQL Overview



- 1) What do you understand by NoSQL databases? Is MongoDB a NoSQL database? explain.
- 2) Which are the different languages supported by MongoDB?
- 3) What are the different types of NoSQL databases? Give some example.
- 4) Is MongoDB better than other SQL databases? If yes then how?
- 5) What type of DBMS is MongoDB?
- 6) What is the difference between MongoDB and MySQL?
- 7) Why MongoDB is known as best NoSQL database?
- 8) Does MongoDB support primary-key, foreign-key relationship?
- 9) Can you achieve primary key - foreign key relationships in MongoDB?
- 10) Does MongoDB need a lot of RAM? Explain these question in simple but detail



Here are detailed yet simple explanations for each of your questions on MongoDB and NoSQL databases:

1. What do you understand by NoSQL databases? Is MongoDB a NoSQL database? Explain.

- **NoSQL** stands for "Not Only SQL." It refers to a class of databases that do not use traditional relational database structures like tables, rows, and columns. They are designed to handle large amounts of data that may not fit well into the relational model.
- **MongoDB** is a NoSQL database. It stores data in flexible, JSON-like documents (BSON), which allows it to be more scalable and better suited for handling unstructured or semi-structured data, unlike SQL databases that require predefined schemas.

2. Which are the different languages supported by MongoDB?

MongoDB supports multiple programming languages for interaction, including:

- **JavaScript**
- **Python**
- **Java**
- **C++**
- **C#**
- **Go**
- **PHP**
- **Ruby**
- **Node.js** These languages can use MongoDB drivers or libraries to communicate with the database.

3. What are the different types of NoSQL databases? Give some examples.

NoSQL databases can be categorized into four main types:

1. **Document-based:** Stores data in documents (typically JSON or BSON). Example: **MongoDB**.

2. **Key-value stores:** Stores data as key-value pairs. Example: **Redis, Riak**.
3. **Column-family stores:** Data is stored in columns instead of rows. Example: **Cassandra, HBase**.
4. **Graph databases:** Used for storing and querying graph data (nodes, edges). Example: **Neo4j, Amazon Neptune**.

4. Is MongoDB better than other SQL databases? If yes, then how?

- MongoDB is **not necessarily better** than SQL databases, but it offers **advantages** in certain situations, such as:
 - **Scalability:** It scales horizontally, meaning it can handle large volumes of data across multiple machines.
 - **Flexibility:** MongoDB doesn't require a fixed schema, allowing easy storage of dynamic and unstructured data.
 - **Performance:** For large, complex data, MongoDB can outperform traditional SQL databases due to its design.
 - **Speed:** MongoDB is fast at storing and retrieving documents with its simple and flexible design.

However, SQL databases might be better when strict ACID (Atomicity, Consistency, Isolation, Durability) properties and complex querying are required.

5. What type of DBMS is MongoDB?

MongoDB is a **NoSQL** database management system. Specifically, it is a **document-oriented** DBMS because it stores data as JSON-like documents in collections.

6. What is the difference between MongoDB and MySQL?

- **Data Model:** MySQL is a relational database that uses tables to store structured data. MongoDB, a NoSQL database, stores data in flexible documents (JSON-like format) that can vary in structure.
- **Schema:** MySQL requires a fixed schema, meaning the structure of data must be predefined. MongoDB has a dynamic schema, so documents in a collection do not need to have the same structure.
- **Scalability:** MongoDB can scale horizontally (across multiple servers) to handle large amounts of unstructured data, while MySQL traditionally scales vertically (more powerful hardware).
- **Query Language:** MySQL uses **SQL** (Structured Query Language) for queries, while MongoDB uses a query language based on JavaScript and supports a range of operators for filtering documents.

7. Why MongoDB is known as the best NoSQL database?

MongoDB is often considered one of the best NoSQL databases because:

- It provides a **high level of flexibility**, allowing data to be stored in documents without a strict schema.
- It is **scalable** and can handle huge amounts of data with horizontal scaling.
- It supports **replication and sharding**, providing high availability and fault tolerance.
- It has a large **community**, well-documented resources, and support for various programming languages.

8. Does MongoDB support primary-key, foreign-key relationships?

- MongoDB does **not** support traditional **foreign key relationships** like relational databases. It does not enforce constraints between documents in different collections.
- However, MongoDB **does support primary keys**. Each document in a collection has a unique `_id` field that acts as the primary key.

9. Can you achieve primary key - foreign key relationships in MongoDB?

While MongoDB doesn't have native foreign key constraints, you can **simulate** these relationships manually:

- You can reference documents in one collection from another by storing an `_id` from one document as a field in another. This is known as **manual referencing**.
- MongoDB also supports **embedding** documents, where you can include related data within the same document, reducing the need for cross-collection references.

10. Does MongoDB need a lot of RAM?

MongoDB can require **significant RAM** because it caches data in memory for faster access. The more RAM you have, the better MongoDB can perform, especially with large datasets.

- However, MongoDB does not necessarily "need" a lot of RAM, but having more RAM **improves performance** by reducing disk I/O operations and increasing the ability to handle large datasets in memory.
- The amount of RAM needed depends on your dataset size and the operations you are performing (e.g., complex queries or aggregations).



What is BSON?

- 11) Explain the structure of ObjectID in MongoDB.
- 12) Is it true that MongoDB uses BSON to represent document structure?
- 13) What are Indexes in MongoDB?
- 14) By default, which index is created by MongoDB for every collection?
- 15) What is a Namespace in MongoDB?
- 16) What is journaling feature? Can journaling features be used to perform safe hot backups?
- 17) What is Profiler? Why does Profiler use in MongoDB?
- 18) If you remove an object attribute, is it deleted from the database?
- 19) In which language MongoDB is written?
- 20) Does MongoDB need a lot space of Random Access Memory (RAM)?
- 21) What language you can use with MongoDB?
- 22) Does MongoDB database have tables for storing records?
- 23) Do the MongoDB databases have schema?
- 24) What is the method to configure the cache size in MongoDB?
- 25) How to do Transaction/locking in MongoDB?



Here are detailed answers to your MongoDB-related questions:

1) What is BSON?

- **BSON** stands for **Binary JSON**. It is a binary representation of JSON-like documents, used by MongoDB to store data. BSON is more efficient than JSON because it supports additional data types like `Date`, `Binary`, and `ObjectId`, and is designed to be easy to encode and decode, while being fast for storage and retrieval.

11) Explain the structure of ObjectId in MongoDB.

- The **ObjectId** is a unique identifier for documents in MongoDB. It is a 12-byte value, structured as:
 1. **4-byte timestamp**: Represents the creation time of the ObjectId (seconds since Unix epoch).
 2. **5-byte random value**: Ensures uniqueness across machines.
 3. **3-byte incrementing counter**: Ensures uniqueness within a single machine.

This structure allows ObjectIDs to be globally unique and roughly ordered by creation time.

12) Is it true that MongoDB uses BSON to represent document structure?

- **Yes**, MongoDB uses **BSON** to represent document structure. BSON is an efficient binary format that allows MongoDB to store and manipulate data with more flexibility and speed than plain JSON.

13) What are Indexes in MongoDB?

- **Indexes** in MongoDB are data structures that improve the speed of data retrieval operations. They help MongoDB locate and access data more efficiently by reducing the number of documents it needs to scan when performing queries.
- MongoDB supports various types of indexes, including single-field, compound, hashed, and geospatial indexes.

14) By default, which index is created by MongoDB for every collection?

- By default, MongoDB creates a **primary index** on the `_id` field for every collection. This ensures that each document in the collection has a unique identifier and allows for efficient lookups by `_id`.

15) What is a Namespace in MongoDB?

- A **namespace** in MongoDB refers to the combination of the database name and the collection name, which uniquely identifies a collection within a database. For example, in `mydb.mycollection`, "mydb" is the database and "mycollection" is the collection, together forming the namespace `mydb.mycollection`.

16) What is the journaling feature? Can journaling features be used to perform safe hot backups?

- **Journaling** in MongoDB is a feature that provides durability by logging every operation to a journal before applying changes to the database. This ensures that in case of a crash, MongoDB can recover the database to its last consistent state.
- **Yes**, journaling can be used for safe **hot backups**, as MongoDB ensures that data written to the journal is durable and can be recovered after a crash. However, to create proper backups, MongoDB provides additional tools like `mongodump` and `mongorestore`.

17) What is Profiler? Why is Profiler used in MongoDB?

- The **Profiler** in MongoDB is a tool used to capture and log database operations (queries, inserts, updates, and deletes) that take a long time to execute. It helps analyze slow queries and identify performance bottlenecks.
- Profiler is useful for monitoring and troubleshooting database performance issues.

18) If you remove an object attribute, is it deleted from the database?

- **Yes**, if you remove an attribute (field) from a document in MongoDB and save it, the field will be deleted from the document in the database. The operation will update the document, and the deleted field will no longer exist in the document.

19) In which language is MongoDB written?

- MongoDB is primarily written in **C++** for performance reasons.

20) Does MongoDB need a lot of space of Random Access Memory (RAM)?

- MongoDB can benefit from **ample RAM** for better performance, especially when handling large datasets. It uses RAM to cache frequently accessed data, reducing the need for disk I/O operations. The more RAM available, the faster the database can operate, as it can store more data in memory.

21) What language can you use with MongoDB?

- MongoDB supports various programming languages, including:
 - **JavaScript (Node.js)**
 - **Python**
 - **Java**
 - **C#**
 - **Go**
 - **Ruby**
 - **PHP**
 - **C++**
 - **Scala**
 - **R** These languages can interact with MongoDB through drivers or official libraries.

22) Does MongoDB database have tables for storing records?

- No, MongoDB does not use tables like a relational database. Instead, it stores data in **collections**. Collections are analogous to tables in SQL, but they are schema-less and can store documents with different structures.

23) Do the MongoDB databases have schema?

- **No**, MongoDB databases are **schema-less**. While collections can contain documents, each document in a collection can have a different structure. However, you can enforce some schema-like structure using **Mongoose** (for Node.js) or other tools, but it's not a requirement in MongoDB itself.

24) What is the method to configure the cache size in MongoDB?

- The cache size in MongoDB is controlled by the **WiredTiger storage engine** (default in MongoDB). You can configure the cache size using the `--wiredTigerCacheSizeGB` option when starting the MongoDB server. For example, to set the cache size to 2GB:

```
css
```

```
mongod --wiredTigerCacheSizeGB 2
```

25) How to do Transaction/locking in MongoDB?

- MongoDB supports **multi-document transactions** starting from version 4.0 for replica sets and 4.2 for sharded clusters. You can use the `session` object to start, commit, or abort a transaction.
- MongoDB uses **optimistic concurrency control** and **locking** for transaction management. It uses a **write lock** for individual operations and ensures that all operations in a transaction are atomic.

Example of a basic transaction in MongoDB (in JavaScript):

```
javascript
```

```
const session = client.startSession();
session.startTransaction();
try {
  // Perform your operations here
  await collection.updateOne({ _id: 1 }, { $set: { field: "value" } });
  await session.commitTransaction();
} catch (error) {
  await session.abortTransaction();
} finally {
  session.endSession();
}
```



Explain in detail BSON structure



BSON Structure in Detail

BSON (Binary JSON) is a binary-encoded serialization format used by MongoDB to store documents in collections. It is a more efficient and compact format than JSON for data storage and retrieval, allowing MongoDB to support a broader range of data types and optimize performance for database operations. Below is a detailed breakdown of BSON structure:

BSON Structure

BSON documents are composed of **key-value pairs**, where:

- **Key:** A string that represents the name of the field.
- **Value:** The value associated with the key, which can be of various data types.

Unlike JSON, BSON supports additional types that aren't available in regular JSON, which makes it more efficient and capable of representing complex data structures. Some of the core types include integers, binary data, and datetime objects.

BSON Data Types

The key data types in BSON are:

1. Embedded Documents:

- BSON allows documents to be embedded within other documents, creating a hierarchical structure.
- These are equivalent to JSON objects. For example, in MongoDB, a document may have an embedded array or object.

2. Arrays:

- BSON allows arrays to store multiple values, including mixed data types. Arrays in BSON work similarly to JSON arrays, but BSON allows for binary data and other MongoDB-specific data types.

3. Basic Data Types:

- **String:** UTF-8 encoded string.
 - **Integer:** Supports 32-bit and 64-bit integers, allowing for more efficient storage of numbers.
 - **Boolean:** True or False values.
 - **Floating Point:** Double-precision floating-point values.
 - **Decimal128:** A 128-bit decimal-based value for higher precision than floating points.
 - **Null:** Represents a null value, similar to JSON.
 - **Date:** Stores datetime information with millisecond precision.
 - **Binary Data:** Allows the storage of arbitrary binary data, used for things like files, images, etc.
 - **ObjectId:** A 12-byte identifier used to uniquely identify documents in a collection (default primary key in MongoDB).
-

Detailed Breakdown of BSON Object Structure

A BSON document consists of the following components:

1. Length:

- The **first 4 bytes** of a BSON document represent the **total length** of the document (including the length of the document itself and the null-terminating byte for the string). This is used for efficient parsing and ensuring the correct amount of data is read from the stream.

2. Key-Value Pairs:

- **Key:** A string representing the name of the field.
- **Value:** The data associated with the key, which can be a number, string, array, embedded document, or any other BSON-supported data type.

Every key-value pair in BSON is encoded with a **type byte** that indicates the type of value that follows. The type byte is a 1-byte field that precedes the actual value.

3. Null Terminator:

- Each key is null-terminated to signify the end of the key name. This is useful for efficiently parsing BSON documents.

4. Types of BSON Data:

- Each value in BSON is preceded by a **type identifier byte** that specifies the data type. The type identifier is essential for determining how to interpret the subsequent data.

Structure Example

For example, consider the following JSON document:

```
json

{
  "name": "John Doe",
  "age": 30,
  "active": true,
  "birthdate": "1990-01-01T00:00:00Z",
  "address": {
    "street": "123 Main St",
    "city": "Anytown"
  }
}
```

This JSON document, when converted into BSON, will look like this in binary format (simplified view):

- **Length of Document (4 bytes):** Total size of the document.
- **Key-Value Pair 1:**
 - Type byte (string) + "name" (length of string) + "John Doe" (string value).
- **Key-Value Pair 2:**
 - Type byte (integer) + "age" + 30 (integer).
- **Key-Value Pair 3:**
 - Type byte (boolean) + "active" + true (boolean).
- **Key-Value Pair 4:**
 - Type byte (datetime) + "birthdate" + Date("1990-01-01T00:00:00Z").
- **Key-Value Pair 5 (Embedded Document):**
 - Type byte (document) + "address" + embedded document:
 - Type byte (string) + "street" + "123 Main St".
 - Type byte (string) + "city" + "Anytown".
- **End of Document:** Null terminator byte marking the end of the document.

BSON vs JSON

Feature	BSON	JSON
Data Types	More types (Date, ObjectId, Binary)	Limited (only text, numbers, and booleans)
Binary Format	Binary encoded, compact and efficient	Text-based, less compact
Speed	Faster encoding/decoding (binary format)	Slower encoding/decoding (text format)
Use in MongoDB	Native format for MongoDB storage	Primarily for data interchange

Benefits of BSON

- **Efficient Storage:** BSON is more compact than JSON due to its binary encoding.

- **Flexible Data Types:** BSON supports more data types than JSON, including embedded documents, binary data, and timestamps.
- **Fast Processing:** BSON is designed for fast parsing and manipulation of data, making it suitable for database use where performance is critical.

Conclusion

In summary, BSON (Binary JSON) is the format used by MongoDB for storing data in collections. It is an extended, binary version of JSON that supports a richer set of data types, such as binary data, ObjectId, and datetime. BSON is more compact and optimized for high performance, making it ideal for database storage and retrieval.



- 26) Why 32 bit version of MongoDB are not preferred ?
- 27) What is moveChunk? Is it possible to remove old files in the moveChunk directory?
- 28) What is shard? What will have to do if a shard is down or slow and you do a query?
- 29) Explain the covered query in MongoDB.
- 30) What is the importance of covered query?
- 31) What is sharding in MongoDB?
- 32) What is replica set in MongoDB?
- 33) What is primary and secondary replica set in MongoDB?
- 34) By default, which replica sets are used to write data?
- 35) What is CRUD in MongoDB?
- 36) In which format MongoDB represents document structure?
- 37) What will happen when you remove a document from database in MongoDB? Does MongoDB remove it from disk?
- 38) Why are MongoDB data files large in size?
- 39) What is a storage engine in MongoDB?
- 40) Which are the storage engines used by MongoDB?
- 41) What is the usage of profiler in MongoDB?
- 42) Is it possible to configure the cache size for MMAPv1 in MongoDB?
- 43) How to configure the cache size for WiredTiger in MongoDB?
- 44) How does MongoDB provide concurrency?
- 45) What is the difference between MongoDB and Redis database?
- 46) What is the difference between MongoDB and CouchDB?
- 47) What is the difference between MongoDB and Cassandra?
- 48) Is there any need to create database command in MongoDB?



Here are the answers to the questions related to MongoDB:

26) Why 32-bit version of MongoDB is not preferred?

- **Limited Address Space:** The 32-bit version of MongoDB has a limit of 2GB for the maximum size of the data that can be handled by a single database. This makes it unsuitable for larger datasets.
- **Memory Limitations:** 32-bit systems cannot access more than 4GB of RAM, which restricts performance when dealing with larger datasets or high concurrency.

- **Performance Issues:** With 32-bit, MongoDB cannot handle high-performance workloads or large-scale applications due to its limited memory addressing capacity.
- **Compatibility:** Modern MongoDB features and scalability features are built to work on 64-bit systems, so 32-bit MongoDB is deprecated in favor of the more robust and scalable 64-bit systems.

27) What is `moveChunk`? Is it possible to remove old files in the `moveChunk` directory?

- `moveChunk`: This operation in MongoDB is part of **sharding** and is used to move chunks of data from one shard to another to maintain data distribution across shards.
- **Removing old files:** Once the chunk migration is complete, MongoDB automatically removes any old files associated with the migration. However, manual removal of files should be avoided, as it could cause inconsistencies or corruption in the data.

28) What is a shard? What happens if a shard is down or slow during a query?

- **Shard:** In a sharded MongoDB setup, a **shard** is a horizontal partition of data across multiple machines. Shards store a subset of the data.
- **When a shard is down or slow:** MongoDB's **mongos** router coordinates queries across the available shards. If a shard is down, the **mongos** will not send queries to that shard. If the shard is slow, the query response time might be impacted, but the query will still be routed to other functioning shards. MongoDB handles this with **replica sets** for fault tolerance.

29) Explain the covered query in MongoDB.

A **covered query** in MongoDB is one where all the fields required by the query (including the query criteria and the fields returned in the projection) are **indexed**. In such cases, MongoDB can serve the query entirely from the index, without needing to scan the actual documents, leading to faster performance.

30) What is the importance of covered query?

The importance of a covered query lies in its ability to:

- **Improve Performance:** Since MongoDB can serve the query entirely from the index, it reduces the need to access the actual documents, which speeds up the query.
- **Reduced Disk I/O:** It reduces disk I/O, as MongoDB doesn't need to load documents into memory.
- **Efficient Use of Indexes:** A covered query makes optimal use of available indexes, ensuring faster response times.

31) What is sharding in MongoDB?

Sharding is the process of distributing data across multiple servers or clusters to ensure horizontal scalability. MongoDB uses sharding to handle large datasets by breaking them into smaller, more manageable chunks and distributing those chunks across different servers.

32) What is a replica set in MongoDB?

A **replica set** is a group of MongoDB servers that maintain the same data. One of the servers is the **primary** node (write operations happen here), and the other servers are **secondary** nodes (they replicate the data from the primary). Replica sets provide data redundancy and increase availability.

33) What is the primary and secondary replica set in MongoDB?

- **Primary:** The primary replica set node is where all write operations are performed. It can accept reads and writes.
 - **Secondary:** The secondary replica set nodes replicate the data from the primary and serve read requests (if configured). They provide redundancy and can be promoted to primary if the primary goes down.
-

34) By default, which replica sets are used to write data?

By default, write operations are directed to the **primary replica set node**. Only the primary node can accept writes in a replica set.

35) What is CRUD in MongoDB?

CRUD stands for **Create**, **Read**, **Update**, and **Delete**, which are the four basic operations used in MongoDB:

- **Create:** Inserting new documents into a collection.
 - **Read:** Querying and retrieving documents.
 - **Update:** Modifying existing documents.
 - **Delete:** Removing documents from a collection.
-

36) In which format does MongoDB represent document structure?

MongoDB represents documents in **BSON** (Binary JSON) format, which is a binary-encoded version of JSON that supports more data types like ObjectId, binary data, and date.

37) What happens when you remove a document from the database in MongoDB? Does MongoDB remove it from disk?

When a document is removed from MongoDB:

- The document is **removed from memory** immediately.
 - The space occupied by the document is marked as **available for reuse**, but MongoDB does not immediately reclaim disk space.
 - The **disk space** might not be freed up immediately, but it will be reused by future documents. MongoDB uses a mechanism like **WiredTiger** for storage, which handles this efficiently.
-

38) Why are MongoDB data files large in size?

MongoDB data files are large due to:

- **Storage of Deleted Documents:** MongoDB may not immediately reclaim the disk space of deleted documents.
- **Fragmentation:** Data files may have fragmented space after documents are inserted, updated, or deleted.
- **Pre-allocation:** MongoDB may pre-allocate large files for efficient write operations, resulting in larger files.

39) What is a storage engine in MongoDB?

A **storage engine** in MongoDB is the component responsible for how data is stored on disk. It determines how data is organized, accessed, and managed.

40) Which are the storage engines used by MongoDB?

MongoDB supports the following storage engines:

- **WiredTiger:** Default engine, providing high performance and support for compression.
- **MMAPv1:** Older engine, uses memory-mapped files.
- **In-Memory:** Stores all data in RAM, used for ephemeral data.
- **MongoRocks:** A storage engine based on the RocksDB key-value store.

41) What is the usage of the profiler in MongoDB?

The **MongoDB Profiler** is used to log database operations for performance analysis. It helps in identifying slow queries, operations with high resource consumption, and understanding query patterns. This helps optimize database performance.

42) Is it possible to configure the cache size for MMAPv1 in MongoDB?

Yes, it is possible to configure the cache size for the **MMAPv1** storage engine by using the `--mmapv1CacheSize` parameter to set the maximum size of the in-memory cache.

43) How to configure the cache size for WiredTiger in MongoDB?

The cache size for the **WiredTiger** storage engine can be configured using the `storage.wiredTiger.engineConfig.cacheSizeGB` option in the `mongod.conf` configuration file or by using the `--wiredTigerCacheSizeGB` option when starting the `mongod` instance.

44) How does MongoDB provide concurrency?

MongoDB provides concurrency using **multithreading** and **locking** mechanisms.

- **Document-level locking** (in WiredTiger) ensures that operations on different documents do not block each other.
- **Read and Write Concerns:** MongoDB allows configurable consistency and durability levels, which ensure proper concurrency in distributed environments.

45) What is the difference between MongoDB and Redis database?

- **MongoDB** is a **document-oriented** NoSQL database used for storing large datasets in a flexible format (BSON).
- **Redis** is an **in-memory key-value store** primarily used for caching, real-time applications, and fast data access. Redis data is stored in memory, and it's optimized for low-latency read and write operations.

46) What is the difference between MongoDB and CouchDB?

- **MongoDB** is a **document-oriented** database with a flexible schema, using BSON for storage and MongoDB-specific features like sharding and replication.
- **CouchDB** is also a **document-oriented** database, but it uses **JSON** for storage and focuses on being **eventually consistent** and is optimized for **offline applications**.

47) What is the difference between MongoDB and Cassandra?

- **MongoDB** is a **document-oriented NoSQL database** that supports rich querying, secondary indexes, and ACID transactions.
- **Cassandra** is a **wide-column store** optimized for **high availability** and **scalability**. It is better suited for large-scale applications with heavy write workloads and eventual consistency.

48) Is there any need to create a database command in MongoDB?

No, there is no need to explicitly create a database in MongoDB. MongoDB creates a database when you first insert data into a collection within that database. If the database doesn't exist, MongoDB will create it automatically.

These answers provide a detailed overview of the MongoDB-related concepts you asked about! Let me know if you'd like any more information on any specific topic.



49) Features of Mongo DB

50) Does MongoDB support ACID transaction management and locking functionalities?

51) Does MongoDB push the writes to disk immediately or lazily?

52) If you remove a document from database, does MongoDB remove it from disk?

53) Mention the command to insert a document in a database called school and collection called persons.

54) Can you create an index on an array field in MongoDB? If yes, what happens in this case?

55) Does MongoDB provide a facility to do text searches? How?

56) What happens if an index does not fit into RAM?

57) Mention the command to list all the indexes on a particular collection.

58) What is Aggregation in MongoDB?

59) What is Replication in MongoDB? Explain.

60) Why are MongoDB data files large in size?

61) When should we embed one document within another in MongoDB?

62) What is master server? Mention the command to check whether you are on the master server or

not?

63) How can you isolate your cursors from intervening with the write operations?

64) Can one MongoDB operation lock more than one databases? If yes, how?

65) How can concurrency affect replica sets primary?

66) Can you run multiple Javascript operations in a single mongod instance?

67) Which command can be used to provide various information on the query plans used by a MongoDB query?



Here are the answers to your MongoDB-related questions:

49) Features of MongoDB

- **Document-Oriented:** Stores data in BSON format (Binary JSON), offering flexible and schema-less storage.
- **Scalability:** Supports horizontal scaling via **sharding**.
- **High Availability:** Achieved through **replica sets** for redundancy and failover.
- **Rich Query Language:** Provides powerful querying, filtering, and aggregation capabilities.
- **Indexing:** Supports various index types for optimizing queries, including compound and geospatial indexes.
- **Flexible Schema:** No need for a predefined schema, enabling rapid development.
- **ACID Transactions:** Supports multi-document ACID transactions since version 4.0.
- **Aggregation Framework:** Advanced data processing and transformation capabilities.
- **Built-in Replication:** Ensures high availability and fault tolerance.

50) Does MongoDB support ACID transaction management and locking functionalities?

- **ACID Transactions:** Yes, MongoDB supports multi-document ACID transactions starting from version 4.0.
- **Locking:** MongoDB uses **document-level locking** with the WiredTiger storage engine, allowing concurrent operations on different documents within a collection.

51) Does MongoDB push the writes to disk immediately or lazily?

MongoDB writes data to disk **lazily**:

- It uses a write-ahead log and commits writes to the journal file immediately.
- Actual writes to disk happen during the periodic **fsync** process.

52) If you remove a document from a database, does MongoDB remove it from disk?

- MongoDB does not immediately remove it from disk. The document's space is marked as **available for reuse** but is not physically deleted from disk unless a compact operation is performed.

53) Command to insert a document in a database called `school` and collection called `persons`

```
javascript
```

```
db.persons.insertOne({
  name: "John Doe",
  age: 25,
  city: "New York"
})
```

54) Can you create an index on an array field in MongoDB? If yes, what happens in this case?

- Yes, MongoDB allows creating an index on an array field.
- **Behavior:** For each element in the array, a separate index entry is created. This is called a **multikey index**.

55) Does MongoDB provide a facility to do text searches? How?

Yes, MongoDB supports **text searches** using a **text index**.

- To create a text index:

```
javascript
```

```
db.collection.createIndex({ fieldName: "text" });
```

- To perform a text search:

```
javascript
```

```
db.collection.find({ $text: { $search: "search term" } });
```

56) What happens if an index does not fit into RAM?

If an index does not fit into RAM:

- MongoDB must read the index from disk, which significantly slows down query performance.

57) Command to list all the indexes on a particular collection


```
javascript
```

```
db.collection.getIndexes()
```

58) What is Aggregation in MongoDB?

Aggregation in MongoDB is a framework for processing and transforming data. It allows operations like filtering, grouping, sorting, and reshaping using a **pipeline** of stages such as ``$match``, ``$group``, ``$project``, etc.

59) What is Replication in MongoDB? Explain.

Replication is the process of synchronizing data across multiple servers to ensure data redundancy and high availability. It is implemented using **replica sets**, which include:

- **Primary node:** Handles writes.
- **Secondary nodes:** Replicate data from the primary and serve as backups.

60) Why are MongoDB data files large in size?

MongoDB data files are large due to:

- **Pre-allocation** of storage space for efficient writes.
- **Fragmentation** caused by document updates and deletes.
- Retention of unused disk space for future operations.

61) When should we embed one document within another in MongoDB?

Embed one document within another when:

- The relationship between data is **1:1** or **1:N** and tightly coupled.
- You need to retrieve the data together frequently.

62) What is a master server? Mention the command to check whether you are on the master server or not.

- **Master Server:** In MongoDB, the **primary node** of a replica set is the master server, as it handles write operations.
- To check if the current server is the master:

```
javascript
```

```
db.isMaster()
```

63) How can you isolate your cursors from intervening with write operations?

You can use the ``snapshot()`` method to isolate cursors. It ensures that the cursor does not reflect any changes made by concurrent write operations.

64) Can one MongoDB operation lock more than one database? If yes, how?

Yes, operations such as ``db.copyDatabase()`` and ``db.repairDatabase()`` can lock multiple databases because they require a global lock for consistent execution.

65) How can concurrency affect the replica set's primary?

High concurrency can impact the primary node by:

- Increasing **write latency** due to simultaneous write operations.
 - Delaying replication to secondary nodes if the primary cannot handle the load efficiently.
-

66) Can you run multiple JavaScript operations in a single ``mongod`` instance?

Yes, multiple JavaScript operations can be executed concurrently in a single ``mongod`` instance, but each operation runs in its **own thread**.

67) Which command can be used to provide various information on the query plans used by a MongoDB query?

The ``explain()`` method provides details about query plans.

- Example:

```
javascript

db.collection.find({ field: "value" }).explain("executionStats");
```

This returns detailed information about the query plan, including indexes used, execution time, and number of documents scanned.

Let me know if you need more clarification or additional topics!



Just check if any questions missed that can be asked in interview. If yes add 8 more or else leave it



Here are 8 additional MongoDB interview questions that might be asked to cover any gaps:

1) What are capped collections in MongoDB?

Capped collections are fixed-size collections that maintain insertion order and automatically delete the oldest documents when the size limit is reached. They are used for scenarios like logging.

2) How does MongoDB handle large amounts of data?

MongoDB uses techniques like **sharding** to distribute data across multiple servers and supports **indexes** for efficient data access.

3) Explain the concept of GridFS in MongoDB.

GridFS is a specification for storing and retrieving large files, such as images, audio, and video files, in MongoDB by splitting them into smaller chunks and storing them as separate documents.

4) How does MongoDB implement security?

MongoDB provides several security features, including:

- **Role-Based Access Control (RBAC)**
 - **Authentication mechanisms** like SCRAM, LDAP
 - **Encryption** for data in transit and at rest.
-

5) How does MongoDB ensure durability?

MongoDB uses:

- **Journaling** to log write operations.
 - **Write concerns** to control write durability.
 - **Replication** for data redundancy and fault tolerance.
-

6) What is a TTL index in MongoDB?

A TTL (Time-to-Live) index automatically removes documents after a specified expiration time. Useful for managing data like session records.

7) How do you monitor MongoDB performance?

Performance can be monitored using:

- **MongoDB Profiler:** Captures query execution stats.
 - **Monitoring tools:** Like MongoDB Atlas, Cloud Manager, or custom scripts using the ``serverStatus`` and ``dbStats`` commands.
-

8) Can you explain the Aggregation Framework stages in MongoDB?

The Aggregation Framework uses a pipeline with stages like:

- ``$match``: Filters documents.
 - ``$group``: Groups documents by specified fields.
 - ``$project``: Shapes the documents.
 - ``$sort``: Sorts the output.
 - ``$limit``: Limits the number of results.
-

Let me know if you'd like detailed explanations for any of these!