1. Basics of MongoDB

1. What is MongoDB? How is it different from RDBMS?
   MongoDB is a NoSQL, document-oriented database that stores data in flexible, JSON-like documents. It's designed for high availability, performance, and scalability. Unlike a traditional Relational Database Management System (RDBMS) like MySQL or PostgreSQL, which uses a fixed, tabular schema with rows and columns, MongoDB uses a dynamic schema.
   - **Schema**: In an RDBMS, the schema is **static** and **pre-defined**. Each row in a table must conform to the same column structure. In MongoDB, the schema is **dynamic**; documents within the same collection don't need to have the same fields or data types. This offers greater flexibility.
   - **Data Model**: RDBMS uses a **relational model** with tables and joins. MongoDB uses a **document model**, where related data can be embedded within a single document, reducing the need for complex joins.
   - **Scalability**: RDBMS typically scales **vertically** (adding more CPU, RAM to a single server). MongoDB scales **horizontally** (adding more servers to a distributed cluster) through sharding, which is a key advantage for handling large datasets.
2. **What are the key features of MongoDB?**
   - **Document Model**: Stores data in flexible, JSON-like documents.
   - **High Availability**: Achieved through **replica sets**, which provide automatic failover and data redundancy.
   - **Horizontal Scalability**: Achieved through **sharding**, which distributes data across multiple servers.
   - **Rich Query Language**: Supports a powerful query language with a wide range of operators for filtering, sorting, and aggregating data.
   - **Indexing**: Supports various types of indexes (single, compound, geospatial, text) to optimize query performance.
   - **Aggregation Framework**: A powerful pipeline-based framework for performing complex data transformations and analytics.
3. **What are documents and collections in MongoDB?**
   - Document: A document is the basic unit of data in MongoDB. It's a BSON (Binary JSON) object containing field and value pairs. It's similar to a row in an RDBMS table, but with a flexible structure.
     Example:
     { "name": "John Doe", "age": 30, "city": "New York" }
   - **Collection**: A **collection** is a group of documents. It's analogous to a table in an RDBMS. A single database can have multiple collections. Unlike a table, a collection does not enforce a schema, so documents can have different structures.
4. What is BSON in MongoDB?
   BSON stands for Binary JSON. It's a binary-encoded serialization of JSON-like documents used for data storage and network transfer in MongoDB. It extends JSON with additional data types like Date, ObjectID, and Binary Data, making it more efficient for parsing, storing, and manipulating data.
5. Difference between SQL vs NoSQL databases?
   | Feature | SQL (e.g., MySQL, PostgreSQL) | NoSQL (e.g., MongoDB, Cassandra) |

| :--- | :--- | :--- |
Structure	Relational, with a fixed schema	Non-relational, with a dynamic schema
Data Model	Tabular (rows and columns)	Various models: document, key-value, graph, etc.
Query Language	Structured Query Language (SQL)	Varies: MQL (MongoDB Query Language), GraphQL, etc.
Transactions	ACID compliant	Varies; some support ACID, others BASE
Scalability	Primarily vertical (scale up)	Primarily horizontal (scale out)
Best for	Structured data, complex joins, transactional systems	Unstructured/semi-structured data, high-volume data, flexibility

6. What is a replica set in MongoDB?
   A replica set is a group of mongod instances that maintain the same dataset. It consists of one primary node that handles all write operations, and multiple secondary nodes that replicate the data from the primary. This provides high availability and data redundancy. If the primary node fails, an election is held, and a new primary is automatically chosen from the secondaries.
7. What is sharding in MongoDB?
   Sharding is a method for distributing data across multiple machines (shards) to support larger datasets and higher throughput. It's a form of horizontal scaling. MongoDB uses a shard key to determine how documents are distributed across the shards. A mongos router directs queries to the correct shards, and config servers store the cluster's metadata.
8. **Difference between MongoDB and MySQL?**
   - **MongoDB**: NoSQL, document-oriented, flexible schema, scales horizontally, ideal for unstructured data and real-time analytics.
   - **MySQL**: RDBMS, tabular, fixed schema, scales vertically, ideal for structured data and complex joins.
9. **When should you use MongoDB?**
   - **Flexible Data Structures**: When your data schema is evolving or not strictly defined.
   - **High Write/Read Volume**: For applications that require high throughput and performance, like IoT, content management, or real-time analytics.
   - **Scalability**: When you anticipate rapid data growth and need to scale horizontally.
   - **Geospatial Data**: For location-based applications that benefit from MongoDB's geospatial query capabilities.
   - **Content Management Systems**: Storing and retrieving content (e.g., blog posts, user profiles) is a natural fit for the document model.
10. **What are the advantages and disadvantages of MongoDB?**
    - **Advantages**:
      - **Flexibility**: Dynamic schema allows for rapid development and iteration.
      - **Scalability**: Horizontal scaling through sharding handles massive datasets and traffic.
      - **Performance**: Fast reads and writes due to the document model and embedded data.
      - **High Availability**: Replica sets provide data redundancy and automatic failover.

        - **Rich Query Language**: Powerful and expressive query capabilities.
    - **Disadvantages**:
        - **Data Consistency**: Some early versions didn't support multi-document ACID transactions, though this has improved in newer versions.
        - **Joins**: Performing joins can be more complex and less efficient than in an RDBMS.
        - **High Memory Usage**: Can consume more memory due to its data model and in-memory operations.
        - **Complexity**: Managing a sharded cluster can be more complex than a single RDBMS instance.

---

2. CRUD Operations

11. **How to insert data in MongoDB?** (insertOne, insertMany)
    - insertOne(): Inserts a single document into a collection.
      Example:
      db.users.insertOne({ "name": "Alice", "age": 28, "status": "active" })
    - insertMany(): Inserts multiple documents into a collection.
      Example:
      db.users.insertMany([{ "name": "Bob", "age": 35 }, { "name": "Charlie", "age": 42 }])
12. **How to query documents in MongoDB?** (find, findOne)
    - find(): Retrieves documents from a collection that match a query filter. It returns a cursor, which can be iterated over to access the documents.
      Example: db.users.find({ "age": { "$gt": 30 } })
    - findOne(): Retrieves a single document that matches a query filter. It returns a document object or null if no document is found.
      Example: db.users.findOne({ "name": "Alice" })
13. **Difference between updateOne, updateMany, and replaceOne?**
    - updateOne(): Updates a single document that matches the filter.
      Example: db.users.updateOne({ "name": "Alice" }, { "$set": { "age": 29 } })
    - updateMany(): Updates all documents that match the filter.
      Example: db.users.updateMany({ "status": "inactive" }, { "$set": { "status": "archived" } })
    - replaceOne(): Replaces a single document with a completely new document. The new document must not contain any update operators.
      Example: db.users.replaceOne({ "name": "Bob" }, { "name": "Robert", "city": "London" })
14. **How to delete data in MongoDB?** (deleteOne, deleteMany)
    - deleteOne(): Deletes a single document that matches the filter.
      Example: db.users.deleteOne({ "name": "Charlie" })
    - deleteMany(): Deletes all documents that match the filter.
      Example: db.users.deleteMany({ "status": "archived" })

15. What are operators in MongoDB?
    Operators are special keywords used in queries and update operations.
    - $gt, $lt: Comparison operators. $gt (greater than), $lt (less than).
      Example: {"age": {"$gt": 30}}
    - $in: Inclusion operator. Matches any of the values in an array.
      Example: {"status": {"$in": ["active", "pending"]}}
    - $and, $or: Logical operators. Combine multiple query expressions.
      Example: {"$and": [{"age": {"$gt": 30}}, {"status": "active"}]}
16. **What are projection and filtering in MongoDB queries?**
    - Filtering: The first argument in a find() query is the filter (or query) document. It specifies the criteria for which documents to return.
      Example: db.users.find({ "age": { "$gt": 30 } })
    - Projection: The second optional argument in a find() query is the projection document. It specifies which fields to include or exclude from the returned documents. This helps reduce network traffic and memory usage.
      Example: db.users.find({}, { "name": 1, "age": 1, "_id": 0 })
17. What is aggregation in MongoDB?
    The Aggregation Framework is a powerful feature for processing data and returning computed results. It uses a pipeline concept, where documents pass through a series of stages. Each stage performs a specific operation on the data, and the output of one stage becomes the input for the next. This allows for complex data transformations and analytics in a single, efficient operation.
18. **Difference between `$match`, `$group`, `$project` in aggregation?**
    - `$match`: **Filters** the documents to pass into the next stage. It's similar to the `find()` query and should be used early in the pipeline to reduce the number of documents being processed.
    - `$group`: **Groups** documents by a specified key and performs aggregate functions (e.g., `sum`, `avg`, `count`) on the grouped data.
    - `$project`: **Reshapes** each document in the stream, often by including, excluding, or adding new fields based on existing ones. It's used to select which fields to output.
19. What is $lookup in MongoDB? (joins)
    $lookup is an aggregation stage that performs a left outer join from one collection to another within the same database. It's used to join documents from two collections based on a shared field. It's a way to model one-to-many or one-to-one relationships in a denormalized schema.
    Example: Joining an orders collection with a products collection.
    db.orders.aggregate([{ "$lookup": { "from": "products", "localField": "productId", "foreignField": "_id", "as": "product_details" } }])
20. **What is the difference between `find()` and `aggregate()`?**
    - `find()` is used for **simple queries** to retrieve documents, optionally with filtering, sorting, and projection. It's generally faster for basic retrieval.
    - `aggregate()` is used for **complex data processing** and analytics. It's a multi-stage pipeline that can filter, group, transform, and join data, making it suitable for tasks like generating reports or performing calculations across multiple documents. While you can achieve some of the same results, aggregation is designed for more advanced operations.

3. Data Modeling

21. How does MongoDB store data internally?
    MongoDB stores data in BSON (Binary JSON) format. A document is serialized into a BSON object, which is then stored in a WiredTiger storage engine file. These files are organized into collections. Internally, MongoDB also uses various data structures like B-trees for indexing to efficiently locate and retrieve documents.
22. **Difference between embedding vs referencing in MongoDB?**
    - Embedding: Nests related data within a single document. This is ideal when the related data is tightly coupled and accessed together.
      Example: A post document with an array of comments embedded within it.
      Pros: Faster reads (one query), atomic updates.
      Cons: Can lead to large documents; complex updates if the embedded data is large.
    - Referencing: Stores related data in separate collections and links them using a reference (like an _id). This is similar to a foreign key in RDBMS.
      Example: An order document that references a customer document by its _id.
      Pros: Smaller documents, avoids duplication, good for frequently updated or large related data.
      Cons: Requires multiple queries (or $lookup) to retrieve related data, which can be slower.
23. What is schema design in MongoDB?
    Schema design in MongoDB is the process of structuring your data. It's not about creating a rigid schema like in SQL, but about deciding whether to embed or reference related data. The key principle is to design your schema based on your application's query patterns. The goal is to minimize the number of queries required to retrieve the data your application needs.
24. When to use denormalization in MongoDB?
    Denormalization in MongoDB means storing related data in a single document (embedding) to improve read performance. You should use it when:
    - **Data is frequently read together**: If you always fetch a product with its reviews, embed the reviews.
    - **Data is small and static**: For example, embedding a user's address in an order document if the address is not expected to change frequently.
    - **There is a one-to-one or one-to-many relationship**: Especially when the "many" side is relatively small.
25. What are capped collections in MongoDB?
    A capped collection is a fixed-size collection that works like a circular buffer. It maintains insertion order and automatically overwrites the oldest documents when it reaches its maximum size. Capped collections are ideal for use cases like:
    - **Logging**: Storing a constant stream of logs.
    - **Caching**: A simple in-memory cache.
    - **Real-time data**: Storing and serving the most recent data points.
26. What is TTL (Time To Live) collection?
    A TTL collection is a collection with a special index that allows MongoDB to automatically delete documents after a specified time has passed. This is useful for data that has a limited lifespan, such as session data, temporary user data, or event

logs.
Example: Creating a TTL index on a createdAt field that expires after 3600 seconds.
db.logs.createIndex({ "createdAt": 1 }, { expireAfterSeconds: 3600 })

27. What are indexes in MongoDB?
    Indexes are special data structures that store a small portion of the collection's data in an easy-to-traverse form. They improve the speed of read operations by allowing the database to quickly find documents without scanning every single document in a collection.
28. **Difference between single-field and compound indexes?**
    - Single-field Index: An index on a single field. It's useful for queries that filter or sort by that specific field.
      Example: db.users.createIndex({ "email": 1 })
    - Compound Index: An index on multiple fields. The order of the fields is important as it determines the sort order of the index. Compound indexes are used to support queries that involve multiple fields in the filter or sort.
      Example: db.users.createIndex({ "status": 1, "created_at": -1 })
29. **What is text index and geospatial index?**
    - Text Index: Supports efficient text search queries on string content within your documents. It uses a special algorithm to tokenize and analyze text.
      Example: db.products.createIndex({ "description": "text" })
    - Geospatial Index: Supports queries for location-based data. MongoDB offers a variety of geospatial index types, such as 2dsphere for Earth-like spheres and 2d for a flat, Cartesian plane.
      Example: db.places.createIndex({ "location": "2dsphere" })
30. What is covered query in MongoDB?
    A covered query is a query that can be entirely satisfied using an index. All the fields in the query's filter and projection are part of the index. This is extremely efficient because MongoDB doesn't have to access the actual documents, only the index data, which is already in RAM.

---

4. Replication & Sharding

31. What is replication in MongoDB? Why is it needed?
    Replication is the process of synchronizing data across multiple servers. It's needed for:
    - **High Availability**: If the primary node fails, a secondary can take over.
    - **Data Redundancy**: Protects against data loss from a single point of failure.
    - **Read Scalability**: You can distribute read operations across secondary nodes.
    - **Disaster Recovery**: Replicating data to a different data center protects against regional outages.
32. **What is primary and secondary in replica sets?**
    - **Primary**: The single node in a replica set that receives all **write operations**. It records all changes in an **oplog** (operations log).
    - **Secondary**: A member of the replica set that **replicates data from the primary**. It applies the operations from the primary's oplog to its own dataset. Secondaries can serve read operations, providing read scalability.

33. What is oplog (operations log) in MongoDB?
    The oplog is a special capped collection that stores a history of all the write operations performed on the primary node. Secondaries use the oplog to asynchronously replicate and apply the changes to their own data, ensuring that all members of the replica set have a consistent dataset.
34. How does automatic failover work in MongoDB?
    Automatic failover is a key feature of replica sets. If the primary node becomes unresponsive (e.g., due to a crash or network issue), the other members of the replica set detect the failure via a "heartbeat" mechanism. An election is then triggered among the remaining secondary members. The member with the most recent oplog entry and highest priority is elected as the new primary, ensuring minimal downtime.
35. What is sharding in MongoDB?
    Sharding is MongoDB's method for horizontal scaling. It distributes data across multiple machines (shards) to handle datasets too large for a single machine and to increase the read/write throughput of the database.
36. **What is config server and mongos in sharding?**
    - config server: A mongod instance that stores the cluster's **metadata**, including a map of the data (chunks) and where they are located on each shard.
    - mongos: A **routing service** that acts as an interface between client applications and the sharded cluster. It routes queries to the correct shards and aggregates the results.
37. **Difference between horizontal and vertical scaling?**
    - **Vertical Scaling (scaling up)**: Increasing the capacity of a single server by adding more CPU, RAM, or storage. It has physical limits.
    - **Horizontal Scaling (scaling out)**: Adding more servers to a system. It's highly scalable and often more cost-effective for large-scale applications. MongoDB's sharding is a prime example of horizontal scaling.
38. What are shard keys in MongoDB?
    A shard key is a field or a compound index that determines how documents are distributed across the shards in a sharded cluster. Choosing a good shard key is crucial for performance. An ideal shard key should have high cardinality (many unique values) and ensure an even distribution of data to prevent "hotspots" (a single shard handling a disproportionate amount of traffic).
39. What is balanced cluster in sharding?
    A balanced cluster is a sharded cluster where the data is evenly distributed across all shards. The balancer is a background process that monitors the cluster and automatically migrates chunks of data from shards with too much data to shards with too little, ensuring an even distribution and preventing performance bottlenecks.
40. **Pros and cons of sharding?**
    - **Pros**:
        - **Scalability**: Allows for handling massive amounts of data and traffic.
        - **High Performance**: Distributes load and reduces bottlenecks.
        - **Fault Tolerance**: If one shard fails, the rest of the cluster can continue to operate.
    - **Cons**:

        - **Complexity**: A sharded cluster is more complex to set up and manage.
        - **Shard Key Selection**: A poor shard key choice can lead to uneven data distribution and performance issues.
        - **Query Overhead**: Queries that don't use the shard key may need to be broadcast to all shards.

---

5. Transactions & Concurrency

41. Does MongoDB support ACID transactions?
    Yes, as of MongoDB 4.0, it supports multi-document ACID transactions for replica sets, and as of 4.2, for sharded clusters. This means operations across multiple documents and collections are atomic, consistent, isolated, and durable.
42. **Difference between transactions in MongoDB vs RDBMS?**
    - **RDBMS Transactions**: Transactions are a core, fundamental part of the relational model. They are typically used for a wide range of operations, including single-row updates.
    - **MongoDB Transactions**: While now supported, transactions in MongoDB are generally used for complex, multi-document operations that can't be handled by the document model's inherent atomicity. For example, updating a document within a single collection is already atomic, so a transaction isn't necessary.
43. What are write concerns in MongoDB?
    A write concern specifies the level of acknowledgment that MongoDB requests for a write operation. It determines the durability of the write.
    - `w: 1` (default): Acknowledges the write only after it has been written to the primary.
    - `w: "majority"`: Acknowledges the write only after it has been written to the majority of replica set members, ensuring higher durability.
44. What is read concern in MongoDB?
    A read concern specifies the consistency and isolation guarantees for a read operation. It controls the level of data freshness you're willing to accept.
    - `"local"` (default): Returns the most recent data from the current member (primary or secondary).
    - `"majority"`: Returns data that has been acknowledged by the majority of replica set members, ensuring the data is durable.
45. What is journaling in MongoDB?
    Journaling is a feature that writes data changes to a journal file before they are written to the main data files. This ensures that even if a server crashes, the database can recover to a consistent state using the journal. It's a key part of ensuring data durability.
46. What are locks in MongoDB?
    Locks are mechanisms to ensure that only one process can modify a database resource at a time, preventing data corruption. MongoDB uses a system of granular locks (database, collection, or document level) to allow for high concurrency. For

most read and write operations, MongoDB uses a shared/exclusive lock system to ensure data integrity.

47. How does MongoDB ensure consistency in replica sets?
    MongoDB ensures consistency in a replica set through asynchronous replication. The primary node writes to the oplog, and the secondaries continuously poll and apply these operations. While this is an "eventual consistency" model, write and read concerns like "majority" can be used to enforce a stronger consistency model.
48. What is causal consistency in MongoDB?
    Causal consistency is a stronger form of consistency than eventual consistency. It guarantees that if one write causally depends on a previous read, a subsequent read will see the updated data. In a replica set, this is achieved by using a causal consistency session, where the client driver tracks the logical timestamp of its operations.
49. How does optimistic concurrency work in MongoDB?
    Optimistic concurrency is a strategy where you assume that conflicts between simultaneous updates are rare. Instead of locking the data, you check for a version number (or some other unique identifier) before committing the update. If the version has changed, it means another process has modified the document, and the update is retried. This is a pattern used to prevent "lost updates" in an application and is not a built-in feature of MongoDB itself.
50. **Difference between durability and consistency in MongoDB?**
    - **Durability**: Ensures that committed transactions remain permanent, even in the event of a system crash. In MongoDB, journaling and write concerns like `w: "majority"` ensure durability.
    - **Consistency**: Guarantees that a transaction brings the database from one valid state to another. In MongoDB, this is achieved through transactions and read concerns.

---

6. Performance & Optimization

51. **How to improve query performance in MongoDB?**
    - **Indexing**: The most important factor. Create indexes on fields used in `find()` queries and `sort()` operations.
    - **Schema Design**: Design your schema around your application's access patterns, using embedding to minimize the number of queries.
    - **Projection**: Use projection to retrieve only the fields you need.
    - **Aggregation Pipeline Optimization**: Place the `$match` stage early in the pipeline to filter documents as soon as possible.
    - **Hardware**: Ensure you have enough RAM to hold your working set (the data and indexes most frequently accessed).
52. What is explain plan in MongoDB?
    The explain() command is used to analyze a query's execution plan. It provides detailed information on how MongoDB executes a query, including:
    - Which index was used.
    - The number of documents scanned (`nscannedObjects`).
    - The number of index entries scanned (`nscanned`).

    - The execution time.
      This helps in identifying inefficient queries and optimizing them by adding or modifying indexes.
53. How does indexing improve performance?
    Indexes improve performance by allowing MongoDB to avoid a collection scan (or table scan), where it has to check every single document. Instead, it can use a much smaller, pre-sorted index to quickly locate the specific documents that match the query, drastically reducing the I/O and CPU overhead.
54. **Difference between covered query and non-covered query?**
    - **Covered Query**: A query that can be entirely satisfied by an index. MongoDB only needs to read the index, not the documents themselves. This is the most efficient type of query.
    - **Non-covered Query**: A query where MongoDB has to read the actual documents in addition to the index to return the results. This is less efficient than a covered query.
55. What is index cardinality in MongoDB?
    Index cardinality refers to the number of unique values in an indexed field. A field with high cardinality (e.g., _id or email) is a good candidate for an index because it has many unique values, making the index very selective and efficient. A field with low cardinality (e.g., status with values "active" or "inactive") is less effective for filtering on its own, but can still be useful in a compound index.
56. What is aggregation pipeline optimization?
    MongoDB's aggregation framework has an optimizer that automatically reshapes and reorders pipeline stages to improve performance. For example, it might move a $match stage earlier in the pipeline if it can be executed before a resource-intensive stage like $sort or $group. It also performs various internal optimizations to run stages more efficiently.
57. What is write amplification in MongoDB?
    Write amplification occurs when a single logical write operation to the database results in multiple physical I/O writes. This can happen in MongoDB when:
    - A large document is updated, requiring the entire document to be rewritten.
    - An index is updated, requiring both the document and the index B-tree to be modified.
    - A replica set write concern requires the write to be replicated to multiple nodes.
58. **How to avoid large documents in MongoDB?**
    - **Use Referencing**: Instead of embedding a large array of frequently-updated or large data, use referencing to link to a separate collection.
    - **Chunking**: Break large, binary files into smaller pieces using GridFS.
    - **Use `$set` for small updates**: Use update operators like `$set` to modify only a part of the document instead of rewriting the entire document.
59. What is MongoDB Atlas?
    MongoDB Atlas is a fully managed, cloud-based database service for MongoDB. It automates the setup, management, and scaling of your database infrastructure, including replica sets and sharded clusters, across major cloud providers like AWS, Azure, and Google Cloud.
60. **Best practices for MongoDB schema design?**

- **Design for your Queries**: Understand your application's read and write patterns before designing your schema.
- **Use Embedding for Co-located Data**: If data is always read together, embed it to minimize queries.
- **Use Referencing for Independent Data**: If data is large, updated frequently, or shared across multiple collections, use referencing.
- **Use Arrays Sparingly**: If an array can grow indefinitely, it might be a sign to use a separate collection.
- **Consider Indexing**: Always plan your indexes during schema design to support your queries.

---

7. Security

61. **How to secure MongoDB database?**
    - **Enable Authentication**: Don't run MongoDB without a user.
    - **Role-Based Access Control (RBAC)**: Grant users the least amount of privilege they need.
    - **TLS/SSL Encryption**: Encrypt data in transit between clients and the database.
    - **Network Access Control**: Restrict network access to the MongoDB server.
    - **Auditing**: Enable auditing to track all activities on the database.
62. **Difference between authentication and authorization in MongoDB?**
    - **Authentication**: The process of **verifying the identity** of a user. In MongoDB, this is done with a username and password.
    - **Authorization**: The process of **granting privileges** to an authenticated user. It determines what a user is allowed to do (e.g., read, write, or drop a collection).
63. What is role-based access control in MongoDB?
    Role-based access control (RBAC) is a security model where permissions are grouped into roles. You then assign these roles to users, rather than assigning individual permissions to each user. This simplifies user management and security policy enforcement.
64. What is SCRAM authentication in MongoDB?
    SCRAM (Salted Challenge Response Authentication Mechanism) is the default authentication mechanism in MongoDB. It's a secure, challenge-response protocol that prevents a password from being sent over the network in plaintext, protecting against man-in-the-middle attacks.
65. How to enable TLS/SSL in MongoDB?
    You enable TLS/SSL by providing the path to a valid certificate and key file in the MongoDB configuration file (mongod.conf) and setting the net.tls.mode to a secure setting like requireTLS or allowTLS. This encrypts the data transmitted over the network.
66. What is auditing in MongoDB?
    Auditing is the process of tracking and logging database events, such as a user's login, a specific query, or a data modification. Auditing logs provide a record of all activity, which is crucial for security compliance and incident investigation.

67. How to prevent injection attacks in MongoDB?
    The most effective way to prevent MongoDB injection attacks is to use parameterized queries or drivers that sanitize input. This prevents malicious user input (e.g., query operators like $where) from being executed as code. Always treat user input as data, not as executable code.
68. **Difference between database-level and collection-level security?**
    - **Database-level security**: Security settings that apply to an entire database. A user with `read` access to a database can read from any collection within that database.
    - **Collection-level security**: Security settings that apply to a specific collection. You can grant a user `read` access to one collection but not another within the same database.
69. How does encryption at rest work in MongoDB?
    Encryption at rest encrypts the data stored on the disk. MongoDB Enterprise Edition uses the WiredTiger storage engine with native encryption, which encrypts the data files on the disk. The data is only decrypted in memory. This protects against data theft if the physical server or storage media is compromised.
70. What is field-level encryption in MongoDB?
    Field-level encryption (FLE) is a more granular form of encryption where you can encrypt individual fields within a document. The encryption and decryption happen on the client side, meaning the server never sees the plaintext data. This is ideal for protecting highly sensitive information like social security numbers or credit card details.

---

8. Advanced MongoDB

71. **Difference between MongoDB and Cassandra?**
    - **MongoDB**: Document-oriented, supports dynamic schema, strong consistency with transactions, and provides high availability with replica sets.
    - **Cassandra**: Column-family database, fixed schema, designed for extremely high write throughput and availability with tunable consistency. It's best for time-series data and applications that require linear scalability and can tolerate eventual consistency.
72. **Difference between MongoDB and Redis?**
    - **MongoDB**: A full-featured, persistent database for storing documents.
    - **Redis**: An in-memory key-value store, primarily used for **caching**, session management, and real-time analytics. While Redis can persist data, its main strength is high-speed access to data in RAM.
73. What is GridFS in MongoDB? (large file storage)
    GridFS is a specification for storing and retrieving large files (e.g., images, audio, video) in MongoDB. Instead of storing the entire file in a single document (which has a 16 MB limit), GridFS divides the file into smaller chunks and stores them in a fs.chunks collection. Metadata about the file is stored in a fs.files collection.
74. What is change stream in MongoDB?
    A change stream is a feature that allows applications to subscribe to and receive real-time notifications of changes happening on a collection, database, or a deployment. It's similar to a database trigger but provides a more powerful and

flexible way to build real-time applications and microservices that react to data changes.

75. What is capped collection?
    A capped collection is a fixed-size collection that automatically removes the oldest documents as new ones are added. It's useful for high-volume, log-like data where you only need the most recent information. A capped collection is a special type of collection, so this question is a duplicate of number 25.
76. What are aggregation stages in MongoDB?
    An aggregation stage is a data processing step within the aggregation pipeline. Each stage takes a stream of documents as input, performs an operation, and produces a stream of documents as output. Common stages include $match, $group, $project, $sort, $limit, and $lookup.
77. How to perform joins in MongoDB?
    You perform joins in MongoDB using the $lookup stage in the aggregation pipeline. It's the equivalent of a LEFT OUTER JOIN in SQL. It's not a native join operation in the storage engine but an aggregation framework feature.
78. What is $facet in MongoDB?
    $facet is an aggregation stage that allows you to run multiple aggregation pipelines on the same set of input documents. The output is a single document that contains the results of all the sub-pipelines. This is useful for creating multi-faceted search results or dashboards with multiple views of the same data.
79. **What is `$bucket` and `$bucketAuto` in MongoDB?**
    - `$bucket`: An aggregation stage that groups documents into fixed-size **buckets** based on a specified field. You define the boundaries of the buckets yourself.
    - `$bucketAuto`: Automatically determines the optimal number of buckets to evenly distribute the documents across the buckets. This is useful when you don't know the data distribution beforehand.
80. What is MapReduce in MongoDB?
    MapReduce is a data processing paradigm for handling large datasets. It has been largely superseded by the more flexible and performant Aggregation Framework in modern versions of MongoDB. The Map function processes each document and emits key-value pairs, and the Reduce function aggregates the values for each key.

---

9. Real-world & DevOps

81. **How to backup and restore a MongoDB database?**
    1. **Backup**:
        - **`mongodump`**: A command-line tool that creates a binary export of your database.
        - **Filesystem Snapshot**: Taking a snapshot of the underlying storage, which is the preferred method for large deployments.
    2. **Restore**:
        - **`mongorestore`**: A command-line tool to restore data from `mongodump` backups.
82. **What are `mongod` and `mongos` processes?**
    1. `mongod`: The **main MongoDB daemon** process. It's the core database server that runs the database.

    2. `mongos`: The **sharded cluster query router** process. It's the entry point for client applications in a sharded cluster.
83. **Difference between standalone, replica set, and sharded cluster?**
    1. **Standalone**: A single `mongod` instance. It has no redundancy and is not suitable for production.
    2. **Replica Set**: A group of `mongod` instances with one primary and multiple secondaries. It provides **high availability** and data redundancy.
    3. **Sharded Cluster**: A group of replica sets (shards) that distribute data horizontally. It provides **horizontal scalability** and high throughput.
84. **How to monitor MongoDB performance?**
    1. **`mongostat` & `mongotop`**: Command-line tools for real-time monitoring of database activity.
    2. **`db.serverStatus()`**: Provides a rich set of metrics about the server's performance.
    3. **MongoDB Atlas Monitoring**: A managed service that provides comprehensive dashboards for monitoring performance.
    4. **Third-party tools**: Solutions like Datadog or Prometheus.
85. **What are common use cases of MongoDB?**
    1. **Content Management Systems**: Blog platforms, product catalogs.
    2. **IoT (Internet of Things)**: Storing and processing massive streams of time-series data.
    3. **Real-time Analytics**: Dashboards and reporting.
    4. **Mobile Applications**: Backend for apps that need a flexible, scalable database.
    5. **Gaming**: Storing user profiles, game state, and leaderboards.
86. How does MongoDB handle big data?
    MongoDB handles big data through horizontal scaling via sharding. This allows it to distribute a dataset that is too large for a single machine across many servers, providing high throughput and storage capacity. The flexible document model also makes it easy to ingest and process unstructured data from various sources.
87. What is oplog size and why is it important?
    The oplog size is the amount of disk space allocated to the operations log. It's a capped collection. A sufficient oplog size is crucial for replication:
    1. If the oplog is too small, a secondary that goes offline for a while might not be able to catch up, forcing a full resync.
    2. A larger oplog can handle network latency and temporary outages, preventing unnecessary resyncs.
88. What is journaling?
    Journaling is a key part of MongoDB's durability. It writes data changes to an on-disk journal file before they are applied to the main data files. This allows for a fast and consistent recovery after an unclean shutdown, as the database can replay the operations from the journal.
89. **How to migrate data from SQL to MongoDB?**
    1. **Schema Translation**: Map the relational schema (tables and joins) to a document-based schema (embedding and referencing). This is the most complex step.
    2. **Export SQL Data**: Use tools like `mysqldump` to export data from the SQL database.

3. **Transform Data**: Write a script (e.g., in Python or Node.js) to transform the exported data into BSON/JSON documents.
4. **Import to MongoDB**: Use `mongoimport` to load the transformed data into MongoDB.

90. How to integrate MongoDB with Node.js (Mongoose)?
    Mongoose is an Object Data Modeling (ODM) library for Node.js and MongoDB. It provides a schema-based solution to model your application data, enforcing a structure on your collections.
    1. **Connect**: Establish a connection to the MongoDB database.
    2. **Define Schema**: Create a Mongoose schema that defines the structure of your documents.
    3. **Define Model**: Compile the schema into a Model, which is a class that represents a collection.
    4. **Use Model**: Use the Model to perform CRUD operations on your documents in a type-safe and consistent manner.

---

10. Latest Trends

91. What is MongoDB Atlas?
    MongoDB Atlas is the official database-as-a-service (DBaaS) for MongoDB. It provides a fully managed, cloud-based platform for deploying, managing, and scaling MongoDB databases. It's the most common way to run MongoDB in production today.
92. What is a multi-cloud cluster in MongoDB Atlas?
    A multi-cloud cluster in MongoDB Atlas allows you to deploy a single database across multiple cloud providers (e.g., AWS, Google Cloud, Azure). This provides greater resilience against cloud provider outages and allows you to locate data closer to users in different regions for lower latency.
93. What is Realm in MongoDB?
    Realm is a mobile and web application development platform that is part of MongoDB Atlas. It includes a mobile database, a backend-as-a-service (BaaS), and serverless functions. Realm makes it easier to build real-time, data-driven applications by handling the syncing of data between the client and the cloud.
94. How does MongoDB support serverless apps?
    MongoDB supports serverless applications through MongoDB Atlas Serverless Instances and Realm Functions. Serverless instances automatically scale compute and storage based on demand, and you only pay for what you use. Realm Functions are a way to write serverless code that runs directly on MongoDB, allowing you to build a complete backend without managing any servers.
95. **Difference between MongoDB community vs enterprise edition?**
    - **Community Edition**: The free, open-source version of MongoDB. It includes the core database features and is suitable for development and small-scale production.
    - **Enterprise Edition**: A commercial version of MongoDB that includes additional features for security, monitoring, and compliance, such as native encryption at rest, advanced auditing, and an in-memory storage engine.

96. What is MongoDB Stitch?
    MongoDB Stitch was a serverless platform for building applications. It has been rebranded and evolved into MongoDB Realm (now just part of the Atlas platform), which now provides all the serverless functionality.
97. What is field-level encryption in MongoDB Atlas?
    Field-level encryption (FLE) in MongoDB Atlas is a client-side encryption feature where you can encrypt individual fields within a document before they are sent to the database. This ensures that the data is encrypted both in transit and at rest, and the server never has access to the unencrypted data.
98. What is a search index in MongoDB Atlas?
    A search index in MongoDB Atlas is a powerful indexing feature that integrates with MongoDB's search functionality. It allows for rich text search capabilities, including relevance scoring, synonyms, and multi-language support, that are more advanced than a standard text index.
99. What is a Data Lake in MongoDB?
    A Data Lake in MongoDB Atlas allows you to query data stored in various formats in Amazon S3, Azure Blob Storage, or Google Cloud Storage using the MongoDB Query Language (MQL). This allows you to combine and analyze data from your MongoDB cluster with other data in your cloud storage without having to move or transform it.
100. Future of MongoDB in cloud-native applications?
    The future of MongoDB in cloud-native applications is strong, driven by MongoDB Atlas. It is positioned as a developer-friendly, multi-cloud platform that automates database management and provides serverless functions, change streams, and a data lake. This makes it an ideal fit for modern, scalable, and resilient cloud-native architectures.