Main title

Interview questions

MongoDB

Table of Contents

- How does MongoDB handle geographic data and geospatial queries?

What is MongoDB?

MongoDB is a document database built on a horizontal scale-out architecture that uses a flexible schema for storing data. Founded in 2007, MongoDB has a worldwide following in the developer community.

Table of Contents

Why use MongoDB?

As a document database, MongoDB makes it easy for developers to store structured or unstructured data. It uses a JSON-like format to store documents. This format directly maps to native objects in most modern programming languages, making it a natural choice for developers, as they don't need to think about normalizing data. MongoDB can also handle high volume and can scale both vertically or horizontally to accommodate large data loads.

Table of Contents

What are the Advantages of MongoDB?

- A Powerful Document-Oriented Database
- Developer User Experience
- Scalability and Transactionality
- Platform and Ecosystem Maturity

Table of Contents

When Should You Use MongoDB?

- Integrating large amounts of diverse data
- Describing complex data structures that evolve
- Delivering data in high-performance applications
- Supporting hybrid and multi-cloud applications
- Supporting agile development and collaboration

Table of Contents

How does MongoDB exactly stores the data?

In MongoDB, records are stored as documents in compressed BSON files. The documents can be retrieved directly in JSON format, which has many benefits:

- It is a natural form to store data.
- It is human-readable.
- Structured and unstructured information can be stored in the same document.
- You can nest JSON to store complex data objects.
- JSON has a flexible and dynamic schema, so adding fields or leaving a field out is not a problem.
- Documents map to objects in most popular programming languages.

Most developers find it easy to work with JSON because it is a simple and powerful way to describe and store data.

Table of Contents

MongoDB vs. RDBMS: What are the differences?

One of the main differences between MongoDB and RDBMS is that RDBMS is a relational database while MongoDB is nonrelational. Likewise, while most RDBMS systems use SQL to manage stored data, MongoDB uses BSON for data storage.

While RDBMS uses tables and rows, MongoDB uses documents and collections. In RDBMS a table -- the equivalent to a MongoDB collection -- stores data as columns and rows. Likewise, a row in RDBMS is the equivalent of a MongoDB document but stores data as structured data items in a table. A column denotes sets of data values, which is the equivalent to a field in MongoDB.

Table of Contents

How does mongodb scale?

mongo scales the same way as any distributed application would scale, throw more resources at it(`Vertical Scaling`) or distribute the data on multiple physical servers or nodes (`Horizontal Scaling`) .

or in other words:

Vertical Scaling involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space.

Horizontal Scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required.

While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server.

Table of Contents

How does mongodb scale horizontally?

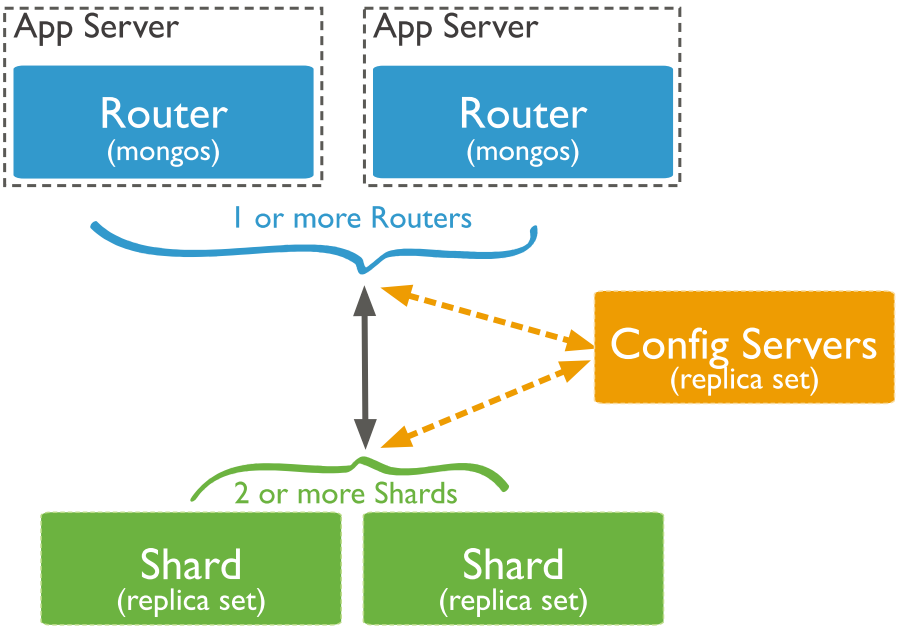
MongoDB supports *horizontal scaling* through sharding.

Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

A MongoDB sharded cluster consists of the following components:

- shard: Each shard contains a subset of the sharded data. Each shard must be deployed as a replica set.
- mongos: The `mongos` acts as a query router, providing an interface between client applications and the sharded cluster. `mongos` can support hedged reads to minimize latencies.
- config servers: Config servers store metadata and configuration settings for the cluster. As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).

The following graphic describes the interaction of components within a sharded cluster:



MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster.

Table of Contents

What are the advantages of sharding?

Reads / Writes

MongoDB distributes the read and write workload across the shards in the sharded cluster, allowing each shard to process a subset of cluster operations. Both read and write workloads can be scaled horizontally across the cluster by adding more shards.

For queries that include the shard key or the prefix of a compound shard key, `mongos` can target the query at a specific shard or set of shards. These targeted operations are generally more efficient than broadcasting to every shard in the cluster.

`mongos` can support hedged reads to minimize latencies.

Storage Capacity

Sharding distributes data across the shards in the cluster, allowing each shard to contain a subset of the total cluster data. As the data set grows, additional shards increase the storage capacity of the cluster.

High Availability

The deployment of config servers and shards as replica sets provide increased availability.

Even if one or more shard replica sets become completely unavailable, the sharded cluster can continue to perform partial reads and writes. That is, while data on the unavailable shard(s) cannot be accessed, reads or writes directed at the available shards can still succeed.

What methods can we use for sharding in mongodb?

MongoDB supports two sharding strategies for distributing data across sharded clusters.

Hashed Sharding

Hashed Sharding involves computing a hash of the shard key field's value. Each chunk is then assigned a range based on the hashed shard key values.

Ranged Sharding

Ranged sharding involves dividing data into ranges based on the shard key values. Each chunk is then assigned a range based on the shard key values.

Table of Contents

How does atomicity and transaction word in mongodb?

In MongoDB, a write operation is atomic on the level of a single document, even if the operation modifies multiple embedded documents *within* a single document.

When a single write operation (e.g. `db.collection.updateMany()`) modifies multiple documents, the modification of each document is atomic, but the operation as a whole is not atomic.

For situations that require atomicity of reads and writes to multiple documents (in a single or multiple collections), MongoDB supports distributed transactions, including transactions on replica sets and sharded clusters.

Table of Contents

What is an Aggregation Pipeline in mongodb?

An aggregation pipeline consists of one or more stages that process documents:

- Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, and calculate values.
- The documents that are output from a stage are passed to the next stage.
- An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.

Table of Contents

Where should I use an index in mongodb?

If your application is repeatedly running queries on the same fields, you can create an index on those fields to improve performance.

Although indexes improve query performance, adding an index has negative performance impact for write operations. For collections with a high write-to-read ratio, indexes are expensive because each insert must also update any indexes.

Table of Contents

How would you choose a indexing strategy in mongo and what are some common considerations we need to care about?

The best indexes for your application must take a number of factors into account, including the kinds of queries you expect, the ratio of reads to writes, and the amount of free memory on your system.

The best overall strategy for designing indexes is to profile a variety of index configurations with data sets similar to the ones you'll be running in production to see which configurations perform best. Inspect the current indexes created for your collections to ensure they are supporting your current and planned queries. If an index is no longer used, drop the index.

These are some considerations you need to take for your indexing strategy:

- Use the ESR (Equality, Sort, Range) Rule

  The ESR (Equality, Sort, Range) Rule is a guide to creating indexes that support your queries efficiently.

- Create Indexes to Support Your Queries

An index supports a query when the index contains all the fields scanned by the query. Creating indexes that support queries results in greatly increased query performance.

- Use Indexes to Sort Query Results

  To support efficient queries, use the strategies here when you specify the sequential order and sort order of index fields.

- Ensure Indexes Fit in RAM

  When your index fits in RAM, the system can avoid reading the index from disk and you get the fastest processing.

- Create Indexes to Ensure Query Selectivity

  Selectivity is the ability of a query to narrow results using the index. Selectivity allows MongoDB to use the index for a larger portion of the work associated with fulfilling the query.

Table of Contents

How does MongoDB ensure data consistency and reliability?

MongoDB ensures data consistency and reliability through several mechanisms:

- Write Concern: Allows you to specify the level of acknowledgment required from MongoDB for write operations.
- Replica Sets: Provides data redundancy and high availability.
- Journaling: Ensures data durability by writing operations to a journal before applying them to the data files.
- Data Validation: Allows you to enforce document structure.

Table of Contents

What are MongoDB's backup and restore options?

MongoDB offers several backup and restore options:

- mongodump and mongorestore: Command-line tools for creating binary exports of database contents and restoring them.
- Filesystem snapshots: For creating point-in-time snapshots of data.
- MongoDB Atlas Backup: Continuous backups and point-in-time recovery for cloud-hosted databases.
- MongoDB Ops Manager: For on-premises deployments, offering backup automation and point-in-time recovery.

How does MongoDB handle concurrency?

MongoDB handles concurrency through:

- Document-level locking: Allows multiple clients to read and write different documents simultaneously.
- WiredTiger storage engine: Provides document-level concurrency control and compression.
- Multi-version Concurrency Control (MVCC): Allows readers to see a consistent view of data without blocking writers.

Table of Contents

What security features does MongoDB offer?

Once again we have many options for security as well, like:

- Authentication: Supports various authentication mechanisms (SCRAM, x.509 certificates, LDAP, Kerberos).
- Authorization: Role-Based Access Control (RBAC) for fine-grained access control.

- Encryption: Supports encryption at rest and in transit (TLS/SSL).
- Auditing: Allows tracking of system and user activities.
- Network isolation: Supports IP whitelisting and VPC peering.

Table of Contents

How does MongoDB manage schema design and changes?

MongoDB uses a flexible, document-based model for schema design:

- Schemaless: Collections don't enforce document structure by default.
- Dynamic Schema: Documents in a collection can have different fields.
- Schema Validation: Optional rules can be set to enforce document structure.
- Indexing: Supports various index types to optimize query performance.

For schema changes:

- Adding fields: Simply update documents with new fields.
- Removing fields: Delete fields from documents or use $unset in updates.
- Changing field types: Update documents with new data types.
- Large-scale changes: Can be done programmatically or using tools like Mongoose for Node.js.

Table of Contents

What are the limitations or disadvantages of MongoDB?

Some limitations and disadvantages of MongoDB include:

- Limited JOIN functionality compared to relational databases.
- Document size limit of 16MB.
- Lack of built-in data integrity constraints (like foreign key constraints).
- Higher storage space requirements due to data duplication in denormalized models.
- Steeper learning curve for those accustomed to relational databases.
- Less mature ecosystem compared to some traditional RDBMSs.

Table of Contents

How do you perform migrations from RDBMS to MongoDB?

Migrating from RDBMS to MongoDB typically involves these steps:

1. Analyze the relational schema and design a document model.
2. Map relational tables to MongoDB collections.
3. Convert relational data to JSON format.
4. Use tools like mongoimport or write custom scripts to import data.
5. Verify data integrity after migration.
6. Update application code to work with MongoDB instead of SQL.
7. Test thoroughly to ensure functionality and performance.

Tools that can assist in this process include:

- MongoDB Compass for visualizing and manipulating data.
- Official MongoDB connectors for various programming languages.
- Third-party ETL (Extract, Transform, Load) tools.

Table of Contents

What is MongoDB Atlas and what benefits does it provide?

MongoDB Atlas is the cloud-hosted database-as-a-service (DBaaS) platform for MongoDB. Benefits include:

- Automated deployment and scaling of MongoDB clusters.
- Multi-cloud and multi-region deployment options.
- Built-in security features (network isolation, encryption, access controls).
- Automated backups and point-in-time recovery.
- Performance monitoring and optimization tools.
- Easy integration with other cloud services.
- Reduced operational overhead for database management.
- Automatic updates and patches for the database software.

Table of Contents

How can you monitor and optimize MongoDB performance?

Monitoring and optimizing MongoDB performance involves several strategies:

Monitoring:

- Use MongoDB's built-in tools like mongotop and mongostat
- Leverage MongoDB Compass for visual performance insights
- Implement MongoDB Atlas monitoring for cloud deployments
- Use third-party monitoring tools like Prometheus with MongoDB exporter

Optimization:

- Analyze and optimize slow queries using explain() method
- Create appropriate indexes based on query patterns
- Use proper schema design to minimize data duplication
- Configure appropriate write concern and read preferences
- Optimize server and storage configurations
- Use appropriate shard keys for distributed systems

Table of Contents

What is a replica set in MongoDB and how does it work?

A replica set in MongoDB is a group of mongod processes that maintain the same data set. It provides:

- High Availability: If the primary node fails, an election occurs to choose a new primary from the secondary nodes.
- Data Redundancy: Data is replicated across multiple nodes.
- Read Scaling: Secondary nodes can handle read operations.

How it works:

- One primary node accepts all write operations
- Multiple secondary nodes replicate data from the primary
- Optionally, arbiter nodes can participate in elections but don't hold data
- Automatic failover occurs if the primary becomes unavailable

Table of Contents

How do you handle data integrity in MongoDB?

While MongoDB doesn't have built-in referential integrity like relational databases, you can maintain data integrity through:

- Schema Validation: Define JSON Schema to enforce document structure
- Application-Level Checks: Implement integrity checks in your application code
- Atomic Operations: Use multi-document transactions for operations that must succeed or fail as a unit
- Unique Indexes: Ensure uniqueness of certain fields
- Data Validation: Use $jsonSchema operator to define validation rules
- Consistent Data Modeling: Design schemas to minimize the need for complex integrity constraints

Table of Contents

how does MongoDB fit into CAP theorm it?

The CAP theorem states that a distributed system can only provide two of three guarantees: Consistency, Availability, and Partition tolerance. MongoDB's position in the CAP theorem can be described as:

Partition Tolerance: MongoDB is designed to handle network partitions in distributed deployments. **Consistency vs. Availability**: MongoDB allows you to tune the balance between consistency and availability:

With default settings, MongoDB leans towards **CP** (Consistency and Partition Tolerance) By adjusting write concern and read preferences, you can shift towards **AP** (Availability and Partition Tolerance)

In practice:

- For strong consistency, use majority write concern and read preference
- For higher availability at the cost of potential inconsistency, use lower write concerns or read from secondaries

Table of Contents

How do you handle large data volumes in MongoDB?

MongoDB offers several strategies for handling large data volumes:

- Sharding: Distribute data across multiple machines to handle large datasets and high throughput operations.
- Indexing: Create appropriate indexes to improve query performance on large collections.
- Aggregation Pipeline: Use for efficient data processing and analysis on large datasets.
- GridFS: Store and retrieve large files efficiently.
- Data Compression: WiredTiger storage engine provides data compression to reduce storage requirements.
- Capped Collections: Use for high-volume data that doesn't require long-term storage.
- Time Series Collections: Optimize storage and querying for time series data.

Table of Contents

What are the best practices for designing a MongoDB schema?

Key best practices for MongoDB schema design include:

- Embed related data in a single document when possible for faster reads.
- Use references when data is used in multiple places or for very large datasets.
- Design schema based on application query patterns.
- Avoid deeply nested documents (keep nesting to 2-3 levels for better performance).
- Use appropriate data types for fields.
- Create indexes to support common queries.

- Consider document growth when designing schemas.
- Use schema versioning for easier updates and migrations.
- Normalize data only when necessary (e.g., for data consistency across multiple collections).
- Use array fields for one-to-many relationships within reasonable limits.

Table of Contents

How does MongoDB handle geographic data and geospatial queries?

MongoDB provides robust support for geographic data and geospatial queries:

- Geospatial Indexes: Support for 2d and 2dsphere indexes for efficient geospatial queries.
- GeoJSON Objects: Store location data using standard GeoJSON format.
- Geospatial Queries: Support for various types of geospatial queries:
  - Proximity: Find documents near a point.
  - Intersection: Find geometries that intersect with a specified geometry.
  - Within: Find geometries contained within a specified geometry.
- Aggregation Pipeline: Geospatial stages like $geoNear for complex geo-based analytics.
- Coordinate Systems: Support for both legacy coordinate pairs and GeoJSON objects.
- Geospatial Operators: $near, $geoWithin, $geoIntersects, etc., for flexible querying.
- Spherical Geometry: 2dsphere index supports queries on an Earth-like sphere.

Table of Contents