MongoDB 2-Mark Questions and Answers

Name any two tools used to interact with MongoDB and their purpose:

MongoDB Compass: A GUI tool for visually exploring and managing MongoDB data.

Mongo Shell: A command-line tool to interact with MongoDB databases using JavaScript commands.

What is a Document in MongoDB?

A document is a single record in MongoDB stored in BSON format, similar to a JSON object, and contains key-value pairs.

What is an index in MongoDB, and why is it used?

An index is a special data structure that improves the speed of query operations by allowing efficient access to documents.

What is the purpose of the aggregation framework in MongoDB?

It is used to process data records and return computed results, allowing for operations like filtering, grouping, and sorting.

What is the difference between embedding and referencing in MongoDB?

Embedding stores related data in the same document.

Referencing stores related data in separate documents linked by IDs.

What is a Schema in MongoDB?

A schema defines the structure of documents in a collection, such as field names, data types, and validation rules.

What is sharding in MongoDB?

Sharding is a method of distributing data across multiple servers to support large datasets and high throughput operations.

Define Strong Data-Consistency:

Strong data consistency ensures that all users see the same data at the same time, even across

distributed systems.

What is MongoDB Atlas?

MongoDB Atlas is a cloud-based database service that hosts, manages, and scales MongoDB clusters automatically.

What is Vertical Scaling?

Vertical scaling means increasing a server's resources (CPU, RAM, storage) to handle more load.

MongoDB 5-Mark Questions and Answers

Embedding vs. Referencing in MongoDB

Embedding and referencing are two ways of modeling relationships between documents in MongoDB:

- Embedding stores related data in a single document. It is ideal for data that is mostly read together and has a one-to-few relationship. It improves read performance and simplifies queries but can cause document size to grow excessively.
- Referencing stores related data in separate documents using references (usually ObjectIds). It is better for one-to-many or many-to-many relationships, allows for better normalization, and avoids document size limits but requires additional queries or $lookup operations.

Choosing between the two depends on access patterns, data size, and performance requirements.

Schema Validation in MongoDB

MongoDB is schema-less by default, but schema validation allows defining rules for the structure of documents in a collection.

- It uses JSON Schema to enforce data types, required fields, value constraints, and patterns.
- Schema validation helps improve data quality, ensures consistency, and prevents accidental insertion of malformed data.
- You can define different levels of validation using "strict", "moderate", or "off" modes.
- Validators are defined when creating a collection or modified using the collMod command.

Schema validation is especially important in collaborative projects and production environments to maintain data integrity.

Replica Sets in MongoDB

A replica set is a group of MongoDB servers that maintain the same data set to provide high availability and data redundancy.

- It consists of a primary node (receives all writes) and one or more secondary nodes (replicate data from the primary).

- If the primary fails, a new one is automatically elected, ensuring minimal downtime.

- Replica sets enable automatic failover, data backup, and read scaling (by directing reads to secondaries).

- They also support write concern and read preference settings for consistency and availability tuning.

Replica sets are critical for building fault-tolerant, production-grade systems in MongoDB.

Sharding Key Selection in MongoDB

Choosing the right shard key is crucial for effective horizontal scaling in MongoDB.

- A shard key determines how documents are distributed across shards.

- A good shard key should have high cardinality, even distribution, and query isolation to ensure balanced load and optimal performance.

- Common strategies include hashed shard keys (uniform distribution) and ranged shard keys (efficient range queries).

- Poor shard key choices can lead to chunk imbalance, hotspots, or inefficient query routing.

Proper shard key selection requires understanding of the application's data access patterns and write/read operations.

Multi-Cloud Deployment in MongoDB Atlas

MongoDB Atlas supports multi-cloud deployment, enabling a single database cluster to span across AWS, Azure, and Google Cloud.

- It enhances availability, resilience, and vendor independence.

- Data can be replicated across cloud providers using global clusters with location-aware sharding.

- It reduces latency by placing data closer to users in different regions and providers.

- Multi-cloud setups help in avoiding vendor lock-in, disaster recovery, and optimizing for cost or compliance.

MongoDB Atlas simplifies multi-cloud architecture with built-in automation, monitoring, and scalability features.

Horizontal Scaling in MongoDB

Horizontal scaling (scale-out) in MongoDB is achieved through sharding.

- It distributes data across multiple machines (shards), allowing for storage and query handling beyond the capacity of a single server.

- Unlike vertical scaling (adding more resources to a single server), horizontal scaling offers better cost-effectiveness and fault tolerance.

- MongoDB uses a mongos query router and a config server to manage metadata and direct queries.

- It is suitable for handling massive datasets and high-throughput workloads such as analytics, e-commerce, and social networks.

Horizontal scaling ensures MongoDB can grow with your application demands without performance degradation.

Below are the key points for each question, organized by section and option, summarizing the core concepts and components for clarity.

Option A: MongoDB Architecture and Comparison

i) Explain the architecture of MongoDB and its key components

- **Architecture**: MongoDB is a NoSQL, document-oriented database with a distributed architecture supporting scalability and high availability.
- **Key Components**:
  - **Database**: Container for collections.
  - **Collections**: Groups of JSON-like documents (similar to tables in RDBMS).
  - **Documents**: BSON (Binary JSON) records, flexible schema.
  - **Shards**: Partitions of data for horizontal scaling.
  - **Replica Sets**: Primary-secondary nodes for fault tolerance and data redundancy.
  - **Mongod**: Core database process handling queries and storage.
  - **MongoDB Atlas**: Cloud-hosted service for management.

ii) How does MongoDB differ from traditional relational databases?

- **Schema**: MongoDB is schemaless (flexible documents) vs. fixed tables/columns in RDBMS.
- **Data Model**: Document-based (BSON) vs. tabular (rows/columns).
- **Querying**: JSON-like queries vs. SQL.
- **Scaling**: Horizontal (sharding) vs. vertical (hardware upgrades).
- **Joins**: Limited, uses embedding/referencing vs. SQL joins.

iii) Mention two advantages of using NoSQL databases like MongoDB

- **Scalability**: Easily scales horizontally across distributed systems.
- **Flexibility**: Supports dynamic schemas for varied data types.

Option B: MongoDB Setup and NoSQL Context

i) Explain the process of setting up MongoDB on a local machine and in the cloud using MongoDB Atlas

- **Local Setup**:
  - Download MongoDB from the official site.
  - Install and configure `mongod` service.
  - Start server using `mongod` and connect via `mongo` shell or Compass.
- **MongoDB Atlas Setup**:
  - Sign up on MongoDB Atlas.
  - Create a cluster, select cloud provider, and configure settings.
  - Whitelist IP, create admin user, and connect via connection string.

ii) Discuss the types of NoSQL databases and explain where MongoDB fits

- **Types of NoSQL**:
  - **Key-Value**: Simple key-value pairs (e.g., Redis).
  - **Document**: Stores semi-structured documents (e.g., MongoDB).
  - **Column-Family**: Column-based data storage (e.g., Cassandra).
  - **Graph**: Node-edge relationships (e.g., Neo4j).
- **MongoDB's Fit**: Document-based, storing BSON documents in collections, ideal for hierarchical data.

iii) Explain the use of MongoDB Compass

- **Purpose**: GUI tool for MongoDB to visualize, manage, and query data.
- **Features**:
  - Explore collections/documents.
  - Run CRUD operations visually.
  - Analyze schema and create indexes.
  - Debug queries and monitor performance.

Option A: CRUD Operations and Indexing

i) Perform and explain CRUD operations in MongoDB with examples

- **Create**: `db.collection.insertOne({name: "John", age: 30})` – Adds a single document.
- **Read**: `db.collection.find({age: 30})` – Retrieves matching documents.
- **Update**: `db.collection.updateOne({name: "John"}, {$set: {age: 31}})` – Modifies a document.
- **Delete**: `db.collection.deleteOne({name: "John"})` – Removes a document.

ii) Discuss the importance of indexing in CRUD operations and its impact on performance

- **Importance**: Indexes improve query performance by reducing data scanned.
- **Impact**:
  - **Pros**: Faster reads for queries on indexed fields.
  - **Cons**: Slower writes due to index updates, increased storage.
- **Example**: `db.collection.createIndex({age: 1})` speeds up `find({age: 30})`.

iii) What is a Collection in MongoDB?

- A collection is a group of BSON documents, analogous to a table in RDBMS but schemaless, allowing flexible data storage.

Option B: Compound Index and Aggregation

i) What is a compound index? How does it improve query performance?

- **Compound Index**: Index on multiple fields (e.g., `db.collection.createIndex({age: 1, name: 1})`).
- **Performance Improvement**: Optimizes queries filtering/sorting on multiple fields, reducing data scanned.

ii) Explain MongoDB's aggregation framework with an example

- **Aggregation Framework**: Pipeline for data processing (filter, group, sort, etc.).
- **Example**:

```
db.sales.aggregate([
  {$match: {status: "completed"}},
  {$group: {_id: "$product", total: {$sum: "$amount"}}},
  {$sort: {total: -1}}
])
```

  - Matches completed sales, groups by product, sums amounts, and sorts by total.

Option A: Schema Design for Blog Application

i) Explain embedding vs. referencing in schema design. When should each be used?

- **Embedding**: Store related data in a single document (e.g., user details within a post).
  - **Use When**: Data is accessed together, small size, low update frequency.
- **Referencing**: Store related data in separate collections with IDs linking them.
  - **Use When**: Data is large, frequently updated, or queried independently.

ii) Make a Database Design for an Online Blog Application like Medium

- **Collections**:
  - **Users**: `{userId, name, email, bio}`
  - **Posts**: `{postId, title, content, authorId, tags, createdAt, comments: [{commentId, userId, text, createdAt}]}`
  - **Tags**: `{tagId, name, postIds: []}`
- **Design Choices**:
  - Embed comments in posts for quick retrieval.
  - Reference authorId and tagIds for flexibility.

iii) What is Data Modeling?

- Data modeling is designing the structure of data (collections, documents, relationships) to optimize storage, querying, and scalability for an application's needs.

Option B: Schema Design for Social Media Platform

i) Describe the differences between relational database schema design and MongoDB schema design

- **Relational**: Fixed tables, normalized data, SQL joins, rigid schema.
- **MongoDB**: Flexible documents, denormalized (embedding) or referenced data, no joins, schema evolves with needs.

ii) Make a Database Design for a Social Media Platform like Amazon and Write the Code for it

- **Note**: Assuming a social media platform (not Amazon, which is e-commerce).
- **Collections**:
  - **Users**: `{userId, username, email, followers: [userId], following: [userId]}`
  - **Posts**: `{postId, userId, content, likes: [userId], createdAt}`
  - **Comments**: `{commentId, postId, userId, text, createdAt}`
- **Code = **Code Example**:

```
db.users.insertOne({userId: "u1", username: "john", email: "john@example.com", followers: [], following: []});
db.posts.insertOne({postId: "p1", userId: "u1", content: "Hello world!", likes: [], createdAt: new Date()});
```

Option A: CAP Theorem and MongoDB

i) Explain the CAP theorem. How does it define the trade-offs in distributed databases?

- **CAP Theorem**: A distributed system can only guarantee two of three: **Consistency** (all nodes see same data), **Availability** (every request gets a response), **Partition Tolerance** (system works despite network partitions).
- **Trade-offs**: Must sacrifice one (e.g., prioritize consistency over availability or vice versa).

ii) How does MongoDB adhere to the CAP theorem? Discuss how it prioritizes consistency, availability, and partition tolerance

- **MongoDB's Approach**: CP (Consistency and Partition Tolerance).
  - **Consistency**: Ensures strong consistency in replica sets (primary node writes).
  - **Partition Tolerance**: Handles network partitions via replica sets.
  - **Availability**: May sacrifice availability during partitions (secondary nodes don't accept writes).

iii) Why is partition tolerance crucial in modern distributed systems?

- Ensures system functionality during network failures, critical for geographically distributed, high-traffic applications.

Option B: E-commerce and Replication

i) In an e-commerce application, how would you balance availability and consistency? Provide real-world scenarios where availability is preferred over strict consistency

- **Balancing**: Prioritize availability for user-facing operations (e.g., browsing products) and consistency for critical transactions (e.g., payments).
- **Scenarios**:
  - **Product Catalog**: Slightly outdated stock counts are acceptable for browsing.
  - **Shopping Cart**: Temporary inconsistencies in cart items are tolerable.

ii) What is data replication, and how does it help improve fault tolerance and scalability in MongoDB?

- **Data Replication**: Copies data across multiple nodes (replica sets).
- **Fault Tolerance**: Secondary nodes take over if primary fails.
- **Scalability**: Distributes read queries across secondaries.

iii) Mention two key benefits of data replication in a high-traffic database system

- **High Availability**: Ensures uptime during failures.
- **Load Balancing**: Spreads read traffic across nodes.

Option A: Scaling and Atlas Cluster

i) Compare the advantages and disadvantages of horizontal vs. vertical scaling in MongoDB

- **Horizontal Scaling**:
  - **Advantages**: Cost-effective, limitless via sharding, fault-tolerant.
  - **Disadvantages**: Complex setup, potential latency.
- **Vertical Scaling**:
  - **Advantages**: Simpler, no data partitioning.
  - **Disadvantages**: Hardware limits, costly, single-point failure risk.

ii) Describe the steps to set up a MongoDB Atlas cluster

- Sign up on MongoDB Atlas.
- Create cluster, choose cloud provider/region.
- Configure tier, storage, and backups.
- Whitelist IP, set admin credentials.
- Connect via provided connection string.

iii) Explain why MongoDB is highly scalable

- **Sharding**: Distributes data across nodes.
- **Replica Sets**: Ensures redundancy and load balancing.
- **Flexible Schema**: Adapts to growing data needs.

Option B: Backup and Security

i) How does MongoDB handle automatic backup and restore strategies in Atlas?

- **Backups**: Continuous (daily snapshots, point-in-time recovery) or on-demand.
- **Restore**: Download backups or restore to a new cluster via Atlas UI/API.
- **Configuration**: Set retention policies and storage options.

ii) What is the significance of role-based access control (RBAC) in MongoDB security?

- **RBAC**: Assigns specific permissions to users/roles for fine-grained access control.
- **Significance**:
  - Enhances security by limiting data exposure.
  - Ensures compliance with regulations.
  - Prevents unauthorized operations.

These key points cover the essence of each question, focusing on clarity and brevity while addressing the core concepts. Let me know if you need a deeper dive into any specific part!