MongoDB Interview Q&As (Part 1)

🔹 Section 1: Fundamentals

Q1. What is MongoDB and why is it used?
Answer: MongoDB is a NoSQL, document-oriented database that stores data in JSON-like BSON format. It's used for scalability, flexibility, and handling unstructured or semi-structured data.
💡 **Tip:** Always highlight schema flexibility and scalability in interviews.

Q2. What is a Document in MongoDB?
Answer: A document is the basic unit of data in MongoDB, stored in BSON format. Example:

```
{ name: "Aman", age: 25, skills: ["MongoDB", "Node.js"] }
```

💡 **Tip:** Think of a document like a "row" in SQL but more flexible.

Q3. What is a Collection in MongoDB?
Answer: A collection is a group of documents, similar to a table in SQL, but without a fixed schema.
💡 **Tip:** Collections allow documents with different fields.

Q4. What is BSON and how is it different from JSON?
Answer: BSON (Binary JSON) is MongoDB's internal storage format. It supports extra data types like Date, ObjectId, and is more efficient for storage and traversal.
💡 **Tip:** Mention ObjectId is BSON-specific and often tested.

Q5. How does MongoDB differ from SQL databases?
Answer:

- SQL → Tables, Rows, Columns, Fixed Schema
- MongoDB → Collections, Documents, Flexible Schema
  💡 **Tip:** Interviewers love the phrase: *"MongoDB = schema-less JSON database."*

🔹 Section 2: CRUD Operations

Q6. How do you insert a document in MongoDB?

```
db.users.insertOne({ name: "Aman", age: 25 })
```

💡 **Tip:** Use insertMany() for bulk inserts.

Q7. How do you retrieve specific fields from a collection?

```
db.users.find({}, { name: 1, age: 1, _id: 0 })
```

💡 **Tip:** Projections reduce payload size → better performance.

Q8. How do you update a field inside a document?

```
db.users.updateOne({ name: "Aman" }, { $set: { age: 26 } })
```

💡 **Tip:** Use $inc, $push, $addToSet for common updates.

Q9. What is the difference between updateOne() and updateMany()?
Answer:

- updateOne() → Updates the first matching document.
- updateMany() → Updates all matching documents.
  💡 **Tip:** Always clarify which one you're using in code.

Q10. How do you delete a document?

```
db.users.deleteOne({ name: "Aman" })
```

💡 **Tip:** Use deleteMany() carefully — can wipe large datasets.

🔹 Section 3: Querying & Operators

Q11. How do you filter documents with conditions?

```
db.users.find({ age: { $gt: 21 } })
```

💡 **Tip:** Operators like $gt, $lt, $gte, $lte are standard.

$gt → greater than

$lt → less than

$gte → greater than or equal to

$lte → less than or equal to

Q12. How do you perform pattern matching in MongoDB?

You can use the $regex operator to search for string patterns.
Example:

```
db.users.find({ name: { $regex: /^A/, $options: "i" } })
```

- ^A → Matches names that **start with "A"**.
- $options: "i" → Makes the search **case-insensitive** (so "Aman" and "aman" both match).

💡 **Tip:** Regex queries are usually slow because they scan documents, but **prefix-matched regex** (like ^A) can still use indexes and perform better.

Q13. What are logical operators in MongoDB?

- $and → Match all conditions
- $or → Match any condition
- $not → Negate condition
  💡 **Tip:** Most used in filtering queries.

Q14. How do you use $in and $nin operators?

```
db.users.find({ age: { $in: [21, 22, 23] } })
```

```
db.users.find({ status: { $nin: ["inactive", "banned"] } })
```

- status → The field being checked.
- $nin: [...] = **NOT IN** → Excludes documents where status is "inactive" or "banned"

💡 **Tip:** Good for matching against multiple values.

Q15. How do you sort and limit query results?

```
db.users.find().sort({ age: -1 }).limit(5)
```

1. **db.users.find()** → Fetches all documents from the users collection.
2. **.sort({ age: -1 })** → Sorts the documents by the age field in **descending order** (-1 = descending, 1 = ascending).
   - So the oldest users come first.
3. **.limit(5)** → Returns only the **top 5 results** after sorting.

If users have ages [18, 21, 25, 30, 40, 50] → query returns [50, 40, 30, 25, 21].

💡 **Tip:** Combine sort + limit for top-N queries.

🔹 Section 4: Indexing & Performance

Q16. What is an Index in MongoDB and why is it used?
Answer: Indexes speed up queries by avoiding full collection scans.
Example:

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

💡 **Tip:** Indexes = faster reads, slower writes.

Q17. What is a Compound Index?
Answer: Index with multiple fields:

```
db.orders.createIndex({ customerId: 1, orderDate: -1 })
```

💡 **Tip:** Order of fields in compound indexes matters.

Q18. What is a Covered Query?
Answer: A query where all required fields are in the index → no need to fetch full document.
💡 **Tip:** Helps in optimizing read-heavy queries.

Q19. What is a TTL Index and where is it used?
Answer: Time-to-live index auto-deletes documents after expiry.

```
db.sessions.createIndex({ createdAt: 1 }, { expireAfterSeconds: 3600 })
```

💡 **Tip:** Perfect for sessions, cache, logs.

Q20. How do you analyze query performance in MongoDB?
Answer: Use .explain("executionStats") to see index usage and scanned docs.
💡 **Tip:** Compare nReturned vs totalDocsExamined.

🔷 Section 5: Aggregation Framework

Q21. What is the Aggregation Framework?
Answer: A pipeline-based framework for transforming and analyzing documents using stages like $match, $group, $project.
💡 **Tip:** Think of it like SQL's GROUP BY + functions.

Q22. How do you use $group in aggregation?

```
db.sales.aggregate([
  { $group: { _id: "$product", total: { $sum: "$amount" } } }
])
```

💡 **Tip:** Common for totals, averages, counts.

Q23. What is $match and how is it different from find()?
Answer:

- $match → Used inside aggregation pipelines.
- find() → Standalone query method.
  💡 **Tip:** $match can leverage indexes too.

Q24. What is $lookup in MongoDB?
Answer: Performs a left outer join with another collection.

```
db.orders.aggregate([
 { $lookup: {
    from: "customers",
    localField: "customerId",
    foreignField: "_id",
    as: "customerInfo"
 }}
])
```

💡 **Tip:** Great for joins but expensive if not indexed.

Q25. What is $unwind used for?
Answer: Deconstructs array fields → outputs one document per element.
💡 **Tip:** Combine with $group to aggregate arrays.

🔹 Section 6: Schema Design & Advanced Concepts

Q26. When do you Embed vs Reference data?
Answer:

- Embed → small, bounded, frequently accessed together (e.g., user + address).
- Reference → large, unbounded, or shared data (e.g., users ↔ posts).
  💡 **Tip:** Classic interview question — always give example.

Q27. What is the maximum document size in MongoDB?
Answer: 16 MB. Larger data must use GridFS or multiple documents.
💡 **Tip:** Mention GridFS if asked about large files.

Q28. What are Capped Collections?
Answer: Fixed-size collections that overwrite oldest data when full.

💡 **Tip:** Great for logs and real-time data.

Q29. What is Sharding in MongoDB and why is it needed?
Answer: Horizontal partitioning of data across multiple servers for scalability.
💡 **Tip:** Choosing the right shard key is critical.

Q30. What is Replication and why is it important?
Answer: Storing copies of data across multiple servers for high availability. Replica set = Primary + Secondaries.
💡 **Tip:** Mention automatic failover in replica sets.