1. The Need for NoSQL and its Advantages (5 Marks)

NoSQL (Not Only SQL) databases were developed because traditional databases (like MySQL or Oracle) struggled with the massive, unstructured data and high demands of modern web and mobile applications.

5 Key Points on the Need and Advantages of NoSQL:

1. Handling Massive Scale (Horizontal Scaling): Traditional databases can only 'scale up' (buy a bigger server). NoSQL lets you 'scale out' by easily distributing data across many smaller, cheaper servers.

2. Managing Unstructured/Flexible Data: NoSQL uses a flexible schema that allows adding new data types without redesigning the database.

3. Higher Performance (Speed): Avoiding complex joins allows NoSQL to read/write faster than RDBMS.

4. Lower Cost (Commodity Hardware): Runs efficiently on standard hardware instead of expensive specialized servers.

5. Developer Flexibility and Agility: Models like documents (MongoDB) align with modern data structures like JSON.

2. Taxonomy of NoSQL Data Models (5 Marks)

NoSQL includes four main database types optimized for different use cases.

1. Key-Value Store: Simplest model acting like a scalable dictionary (e.g., Redis).

2. Document-Oriented Database: Stores data as JSON/BSON documents (e.g., MongoDB).

3. Column-Family Store: Organizes data in columns for big data analytics (e.g., Cassandra).

4. Graph Database: Stores relationships using nodes and edges (e.g., Neo4j).

3. The CAP Theorem and NoSQL Assumptions (5 Marks)

CAP Theorem states that distributed systems can ensure only two of Consistency, Availability, and Partition Tolerance.

1. Consistency (C): Every client sees the same data instantly.

2. Availability (A): System always responds to requests.

3. Partition Tolerance (P): Operates despite network failures.

4. The Choice (C vs A): NoSQL prioritizes either Consistency or Availability, keeping Partition Tolerance mandatory.

5. BASE over ACID: NoSQL often adopts the BASE model for high scalability, allowing eventual consistency.

4. Strengths and Weaknesses of NoSQL (5 Marks)

Strengths:

1. Better Data Modeling: Flexible and intuitive structures.

2. Superior Scaling and Performance: Handles huge data with horizontal scaling.

Weaknesses:

3. Lack of Standardized Query Language: Each NoSQL uses unique query syntax.

4. Eventual Consistency Trade-off: Might return outdated data temporarily.

5. Complex Multi-Document Transactions: Hard to guarantee atomic operations across multiple documents.

5. MongoDB Architecture and Core Features (5 Marks)

1. Document Data Model: Data stored in JSON-like documents grouped into collections.

2. Schema-Less Structure: Documents in a collection can differ in fields.

3. BSON Format: Binary JSON for faster processing.

4. High Availability with Replica Sets: Automatic failover for reliability.

5. Indexing: Indexes speed up queries significantly.

6. MongoDB CRUD Operations and Data Types (5 Marks)

CRUD = Create, Read, Update, Delete — the basic data management operations.

1. Create: db.collection.insertOne()/insertMany().

2. Read: db.collection.find().

3. Update: db.collection.updateOne()/updateMany().

4. Delete: db.collection.deleteOne()/deleteMany().

5. Key Data Types: ObjectId, Date, Arrays, Embedded Documents, etc.

7. MongoDB Advanced Features: Indexing and Aggregation (5 Marks)

1. Indexing: Optimizes query performance by avoiding full scans.

2. Creating Indexes: db.collection.createIndex().

3. Aggregation Pipeline: Performs data processing and transformations.

4. Pipeline Stages: $match, $group, $project, etc.

5. Use Case: Summaries, reports, and analytical queries.

8. MongoDB Advanced Feature: Sharding for Scaling (5 Marks)

1. Purpose: Distributes data across multiple machines for scaling.

2. Shards: Individual servers holding parts of data.

3. Shard Key: Determines how data is divided.

4. Routing (mongos): Directs queries to correct shards.

5. Auto-Balancing: Moves data automatically for load balancing.

9. MongoDB Advanced Feature: Replica Sets (High Availability) (5 Marks)

1. Purpose: Ensures data redundancy and automatic failover.
2. Primary & Secondary Nodes: Primary handles writes; secondary replicates data.
3. Replication: Data copied asynchronously across members.
4. Automatic Failover: Secondary promoted to Primary during failure.
5. Read Scaling: Secondary nodes can handle reads for faster performance.

10. Managing Complex Queries: Join Operations and Pagination (5 Marks)

1. $lookup (Joins): Used to combine data from multiple collections.
2. When to Use: Prefer embedding, use $lookup only when necessary.
3. Pagination Need: Retrieve data in smaller chunks for performance.
4. Methods: skip() and limit() for paging results.
5. Improved Pagination: Cursor/key-set method for faster page loads.