Most Asked 50+ NoSQL Interview Questions & Answers

1. What is NoSQL? How is it different from SQL (relational databases)?

Answer:

NoSQL stands for **"Not Only SQL"**. It refers to non-relational databases that store data in a format other than traditional relational tables.

Key differences:

- **Schema:** SQL requires predefined schemas; NoSQL is schema-less or flexible.
- **Scalability:** SQL scales vertically (add more power to one machine); NoSQL scales horizontally (add more machines).
- **Structure:** SQL stores data in tables with rows and columns; NoSQL stores in formats like key-value pairs, documents, or graphs.

2. Why would someone use a NoSQL database instead of a traditional relational database?

Answer:

- To handle **large volumes of unstructured or semi-structured data**.
- For **flexibility in schema** (e.g., frequent changes).
- When **horizontal scaling** is required for performance.
- In use cases like **real-time analytics**, caching, and **IoT** where traditional RDBMS may be too rigid.

3. Can you name some popular NoSQL databases and what they're best used for?

Answer:

- **MongoDB** – Document database; ideal for dynamic schema like blogs, catalogs.
- **Redis** – Key-value store; used for caching, real-time analytics.
- **Cassandra** – Wide-column store; high availability, good for time-series data.

- **Neo4j** – Graph database; excellent for social networks, recommendation engines.

4. What are the different types of NoSQL databases?

Answer:

1. **Document** – e.g., MongoDB, CouchDB
2. **Key-Value** – e.g., Redis, DynamoDB
3. **Wide-Column** – e.g., Cassandra, HBase
4. **Graph** – e.g., Neo4j, ArangoDB

5. What kind of data is best stored in a document-oriented database like MongoDB?

Answer:

- **Semi-structured data** like JSON.
- Data with **variable fields per record** (e.g., user profiles).
- Nested or hierarchical data (e.g., product catalogs).

6. What is a key-value store? Can you give a real-life example?

Answer:

- It's a database where data is stored as a **pair** of a key and its corresponding value.

Example:

```json
"user:123": {"name": "Alice", "age": 25}
```

- Used in **Redis**, ideal for caching sessions or user preferences.

7. What is denormalization in NoSQL and why is it common?

Answer:

Denormalization means **storing related data together**, even if it's redundant, to reduce the need for joins and improve **read performance**.

It’s common in NoSQL because these databases don't support joins like SQL databases do.

8. How does NoSQL handle relationships between data?

Answer:

- **Embedding documents** (in MongoDB).
- Using **references/IDs** to link documents.
- Graph databases use **edges** to model direct relationships.

9. What is eventual consistency in NoSQL systems?

Answer:

In distributed systems, **eventual consistency** means data will become consistent **over time**, not immediately.

Used for better availability and performance (CAP theorem – prioritize AP).

10. What is CAP Theorem and how does it relate to NoSQL?

Answer:

CAP = Consistency, Availability, Partition Tolerance

NoSQL databases often sacrifice **Consistency** for **Availability and Partition Tolerance**.

NoSQL Type	Prioritizes
MongoDB	CP (Consistency + Partition Tolerance)
Cassandra	AP (Availability + Partition Tolerance)

11. Explain the structure of a MongoDB document.

Answer:

A MongoDB document is a **BSON** object (Binary JSON).

Example:

```json
{
  "_id": "001",
  "name": "John Doe",
  "email": "john@example.com",
  "skills": ["Java", "MongoDB"]
}
```

12. How would you perform a CRUD operation in MongoDB?

Answer:

- **Create:**

```js
db.users.insertOne({ name: "Alice", age: 30 });
```


- **Read:**

```js
db.users.find({ name: "Alice" });
```

- **Update:**

```js
db.users.updateOne({ name: "Alice" }, { $set: { age: 31 } });
```

- **Delete:**

```js
db.users.deleteOne({ name: "Alice" });
```

13. Can NoSQL databases be ACID compliant?

Answer:

Yes, **some NoSQL databases** like MongoDB support ACID for single or multi-document transactions.

But many favor **BASE** (Basically Available, Soft state, Eventually consistent) over strict ACID.

14. How do NoSQL databases scale?

Answer:

They scale **horizontally** — add more servers to handle increased load.

This is done via **sharding** and **replication**.

15. When should you NOT use a NoSQL database?

Answer:

- If the application needs **complex joins** or **strict ACID compliance**.

- For systems with **fixed schema** or **strong consistency requirements**, like banking.

16. What is sharding in NoSQL?

Answer:

Sharding is the process of **splitting large datasets** across multiple machines (shards) to improve performance and scalability.

17. What's the difference between MongoDB and Cassandra?

Feature	MongoDB	Cassandra
Type	Document-oriented	Wide-column
Best for	Flexible schemas	Heavy writes
Consistency	Strong (CP)	Tunable (AP)

18. What tools or commands have you used to query NoSQL databases?

Answer:

- MongoDB Shell: db.collection.find()
- Mongo Compass (GUI)
- Redis CLI: GET key
- Mongoose (Node.js ODM for MongoDB)

19. How does indexing work in NoSQL databases like MongoDB?

Answer:

- Indexes improve **query performance** by avoiding full collection scans.
- MongoDB uses B-Tree indexes.

```
js                                                    Copy    Edit

db.users.createIndex({ email: 1 });
```

20. Have you ever worked on a project that used a NoSQL database? What challenges did you face?

Answer:

Yes, I used MongoDB in a project for a content management app.

Challenges:

- Data modeling was tricky without predefined schema.
- Managing relationships required embedding or referencing.
- Performance tuning required creating proper indexes.

21. What is BSON in MongoDB and how is it different from JSON?

Answer:

BSON (Binary JSON) is a binary-encoded serialization format used by MongoDB to store documents.

Difference from JSON:

Feature	JSON	BSON
Format	Text-based	Binary

Support	Basic data types	Additional types (Date, BinData)
Speed	Slower parsing	Faster for machines to parse
Size	Compact, but no length prefix	Includes length prefixes, larger

Example:

```
json                                                    Copy    Edit

JSON: {"name": "Alice", "age": 25}
BSON stores it with extra metadata like type and length.
```

22. What are collections in MongoDB? How are they different from tables in RDBMS?

Answer:

A **collection** in MongoDB is a group of documents, similar to a table in relational databases.

Difference:

Feature	MongoDB Collection	RDBMS Table

Schema	Schema-less	Fixed schema
Data Format	BSON documents (flexible)	Rows with fixed columns
Validation	Optional	Enforced

23. What is a replica set in MongoDB? Why is it used?

Answer:

A **replica set** is a group of MongoDB servers that maintain the same data set.

Usage:

- Provides **high availability**.
- One **primary** node accepts writes; others are **secondaries** for redundancy.
- Automatic **failover** if the primary goes down.

24. What is the difference between find() and findOne() in MongoDB?

Answer:

- **find()** returns a **cursor** to all matching documents.
- **findOne()** returns the **first matching document**.

Example:

```js
db.users.find({ age: 25 })      // returns a cursor to all users aged 25
db.users.findOne({ age: 25 })   // returns the first user aged 25
```

25. How would you update a nested field in a MongoDB document?

Answer:

You can use dot notation in the $set operator.

Example:

```js
db.users.updateOne(
  { name: "Alice" },
  { $set: { "address.city": "Delhi" } }
)
```

This updates the city field inside the nested address object.

26. What is MapReduce in NoSQL databases?

Answer:

MapReduce is a data processing paradigm used to perform complex data aggregation and transformation.

- **Map**: Filters and maps data.
- **Reduce**: Aggregates results.

Example in MongoDB:

```js
db.orders.mapReduce(
  function() { emit(this.customerId, this.amount); },
  function(key, values) { return Array.sum(values); },
  { out: "order_totals" }
)
```

27. How is data modeled in NoSQL compared to SQL?

Answer:

Feature	NoSQL	SQL
Structure	Flexible, document/key-value	Fixed, tabular
Relationships	Embedded or referenced	Foreign keys
Normalization	Often denormalized	Normalized

NoSQL emphasizes performance and scalability by allowing nested and redundant data structures.

28. Explain the importance of schema-less design in NoSQL.

Answer:

- Allows storing different structures in the same collection.
- Helps in **rapid development**.
- Ideal for **evolving data models**.
- Reduces **migration overhead**.

Example:

```json
// Same collection, different fields
{ "name": "Alice", "age": 25 }
{ "name": "Bob", "email": "bob@example.com" }
```

29. How do you handle data integrity in NoSQL systems?

Answer:

While NoSQL systems often **sacrifice strict ACID compliance** for performance, you can use:

- **Schema validation** (e.g., in MongoDB via validator)
- **Atomic operations** (on single documents)
- **Application-level validation**
- **Multi-document transactions** (supported in newer MongoDB versions)

30. What is the role of "_id" in MongoDB documents?

Answer:

- Every MongoDB document **must have a unique _id** field.
- It acts as the **primary key**.
- If not provided, MongoDB automatically generates one as an ObjectId.

Example:

```json
{
  "_id": ObjectId("60abc123456789"),
  "name": "Alice"
}
```

31. What is the difference between vertical and horizontal scaling? Which one does NoSQL support better?

Answer:

Feature	Vertical Scaling	Horizontal Scaling
Also Known As	Scale Up	Scale Out
How It Works	Add more CPU/RAM to one server	Add more machines/nodes to the system
Downtime	Usually needs downtime	Minimal downtime

Cost	Can get expensive quickly	More cost-effective in long term
NoSQL Preference	Less preferred	Most NoSQL databases are designed for it

NoSQL databases are optimized for horizontal scaling, making them ideal for distributed and cloud-based applications.

32. What happens if two documents have the same _id in MongoDB?

Answer:

If you try to insert a document with a duplicate _id in the **same collection**, **MongoDB will throw a duplicate key error** because _id must be **unique**.

Example:

```js
db.users.insert({ _id: 1, name: "Alice" });
db.users.insert({ _id: 1, name: "Bob" });
// ❌ Error: E11000 duplicate key error
```

33. What is TTL (Time To Live) index in MongoDB and how does it work?

Answer:

A **TTL index** in MongoDB is used to automatically remove documents after a certain time.

- You create a TTL index on a **date field**.
- MongoDB runs a background thread to delete expired documents.

Example:

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

This removes sessions 1 hour (3600 seconds) after the createdAt timestamp.

34. Can NoSQL databases handle transactions? If yes, how?

Answer:

Yes, some NoSQL databases like MongoDB (v4.0+) **support multi-document transactions**, providing ACID compliance.

Example:


```js
const session = db.getMongo().startSession();
session.startTransaction();
try {
  db.accounts.updateOne({ _id: 1 }, { $inc: { balance: -100 } }, { session });
  db.accounts.updateOne({ _id: 2 }, { $inc: { balance: 100 } }, { session });
  session.commitTransaction();
} catch (e) {
  session.abortTransaction();
}
```

35. What's the difference between embedded documents and references in MongoDB?

Answer:

Feature	Embedded Document	Reference
Structure	Child document is inside the parent	Child stored separately with _id link
Speed	Faster reads	Slower reads, needs separate query
Normalization	Denormalized	Normalized
Use Case	When data is frequently read together	When data is reused across documents

Example:

```js
// Embedded
{ name: "John", address: { city: "Delhi", zip: "110001" } }

// Reference
{ name: "John", addressId: ObjectId("...") }
```

36. How would you store hierarchical data (like categories and subcategories) in a NoSQL database?

Answer:

Common approaches in MongoDB:

Parent referencing

```js
{ _id: 1, name: "Electronics", parentId: null }
{ _id: 2, name: "Mobile Phones", parentId: 1 }
```

Embedded documents

```js
{
  name: "Electronics",
  subcategories: [{ name: "Mobiles" }, { name: "Laptops" }]
}
```

Materialized path

```js
{ _id: 1, name: "Mobile Phones", path: "Electronics/Mobile Phones" }
```


37. What is aggregation in MongoDB and how is it used?

Answer:

Aggregation is a way to process data and return computed results.

It allows:

- Filtering
- Grouping
- Sorting

- Joining collections

38. What is the use of aggregate() pipeline in MongoDB?

Answer:

The aggregate() function uses a **pipeline** of stages to process data step-by-step.

Example:

```js
db.orders.aggregate([
  { $match: { status: "delivered" } },
  { $group: { _id: "$customerId", total: { $sum: "$amount" } } }
]);
```

This returns total delivered orders per customer.

39. What is a write concern in MongoDB?

Answer:

Write concern defines the level of acknowledgment requested from MongoDB for write operations.

Write Concern	Description
w: 1	Acknowledged by primary only (default)
w: 0	Unacknowledged (fire-and-forget)
w: "majority"	Acknowledged by majority of replica set

Example:

```
js                                                    Copy    Edit

db.users.insert({ name: "John" }, { writeConcern: { w: "majority" } });
```

40. What is read preference in MongoDB and why is it important in replica sets?

Answer:

Read preference determines **which member of a replica set** to read data from.

Read Preference	Description

primary	Always read from the primary
secondary	Read from a secondary (for read scaling)
nearest	Reads from node with lowest latency

It's important to **optimize performance and reduce load** on the primary node in large-scale systems.

41. Explain the difference between Redis and MongoDB.

Answer:

Feature	Redis	MongoDB
Type	Key-Value Store	Document-Oriented Database
Data Storage	In-memory (optional disk persistence)	Disk-based
Speed	Very fast (in-memory)	Slower compared to Redis (but fast for disk)

Use Case	Caching, Pub/Sub, Session Storage	General purpose database with flexible schema
Data Format	Simple key-value (Strings, Lists, etc.)	JSON-like documents (BSON)

Use **Redis** for performance-critical, short-lived data.
Use **MongoDB** for long-term storage of structured/semi-structured data.

42. What kind of applications are best suited for key-value databases like Redis?

Answer:

Key-value stores like Redis are best for:

- Caching (e.g., API responses, user sessions)
- Leaderboards (gaming apps)
- Real-time analytics (IoT, monitoring)
- Queue systems (using Redis Lists)
- Pub/Sub messaging systems

They are ideal when:

- Data access is frequent and performance is critical.
- The data model is simple (key → value).

43. What is the difference between a graph database and a document store?

Answer:

Feature	Graph Database (e.g., Neo4j)	Document Store (e.g., MongoDB)
Data Model	Nodes (entities) and Edges (relationships)	JSON-like documents
Best For	Highly connected data (e.g., social networks)	Flexible semi-structured data
Query Language	Cypher (Neo4j)	MongoDB Query Language
Relationship Handling	Native and fast	Manual embedding or referencing

Use **graph DB** for traversals, social graphs, fraud detection.
Use **document DB** for catalogs, user profiles, and content management.

44. What is a wide-column store and give an example.

Answer:

A **wide-column store** stores data in **tables with dynamic columns**, optimized for fast writing and querying of large datasets.

- Rows can have **different columns**.
- Each row is grouped into **column families**.

Examples:

- **Apache Cassandra**
- **Google Bigtable**

Use case: Time-series data, logs, recommendation systems.

45. Explain consistency, availability, and partition tolerance with examples (CAP Theorem).

Answer:

CAP Theorem states that in a distributed system, you can **only guarantee 2 out of 3** at any given time:

Property	Description
Consistency	All nodes see the same data at the same time
Availability	Every request gets a response (success/failure)
Partition Tolerance	System continues working even if network partitions exist

Examples:

- **CP (Consistency + Partition):** MongoDB (with strong consistency mode)
- **AP (Availability + Partition):** CouchDB, Cassandra
- **CA (Consistency + Availability):** Not possible in distributed systems

46. What is the difference between strong consistency and eventual consistency?

Answer:

Type	Description	Example Systems
Strong Consistency	Every read returns the most recent write	MongoDB (with write concern), SQL databases
Eventual Consistency	Reads may return stale data, but will eventually be up-to-date	Cassandra, DynamoDB

Use **strong consistency** for financial transactions.
Use **eventual consistency** for scalable, distributed apps like social feeds.

47. What are the advantages of using NoSQL for big data applications?

Answer:

- **Horizontal scalability** (easy to distribute data)
- **Schema flexibility** (store varied data formats)

- **High performance** for reads/writes
- **Better fit for semi-structured or unstructured data**
- **Easy integration with distributed systems** (like Hadoop, Kafka)
- **Optimized for specific use cases** (e.g., graph traversal, key-value lookup)

48. Can NoSQL databases support joins? If yes, how? If not, why not?

Answer:

Some NoSQL databases do support joins but with limitations.

- **MongoDB:** Supports $lookup for joining collections (since v3.2)
- **Cassandra, Redis:** Don't support joins natively

Reason: Joins are expensive in distributed systems, so NoSQL encourages **denormalization or embedding** instead.

49. How do you ensure backup and recovery in NoSQL databases?

Answer:

- **MongoDB:** Use mongodump / mongorestore or filesystem snapshots
- **Redis:** Uses RDB/AOF files and can replicate data
- **Cassandra:** nodetool snapshot for backups

Regular backup strategies:

- Schedule automatic dumps
- Use cloud storage for backup
- Set up replication and disaster recovery environments

50. What are some security concerns with NoSQL databases and how can they be addressed?

Answer:

Common Security Concerns:

- Lack of authentication/authorization by default
- Unencrypted data in transit or at rest
- Open ports exposed to public

How to address:

- Enable **authentication and role-based access control (RBAC)**

- Use **SSL/TLS encryption**
- Configure **firewalls and IP whitelisting**
- Regularly **patch and update** the database engine
- Enable **audit logging** for suspicious activities