Study Guide: NoSQL and MongoDB

Andrés Oswaldo Calderón Romero, PhD.

November 18, 2025

Learning Objectives

By the end of this guide, you should be able to:

- Understand the differences between NoSQL and traditional relational databases.
- Explain the types of NoSQL databases.
- Understand the core concepts of MongoDB.
- Perform basic CRUD operations in MongoDB.
- Apply MongoDB features to real-world scenarios.

1 Introduction to NoSQL

1.1 What is NoSQL?

- NoSQL stands for **Not Only SQL**.
- Designed for large volumes of unstructured or rapidly changing data.
- Common in web applications, big data, and distributed systems.

1.2 Key Characteristics

Feature	Description
Schema-less	Flexible structure without fixed schema
Scalability	Horizontal scaling (e.g., sharding)
High Availability	Built-in replication and failover
Performance	Optimized for reads/writes over complex joins

1.3 Why Not SQL?

NoSQL databases overcome RDBMS limitations in distributed, large-scale, or real-time applications. They work well for document storage, graphs, key-value pairs, and wide columns.

2 Types of NoSQL Databases

Type	Description	Examples
Document-based	Stores documents (e.g., JSON/BSON)	MongoDB, CouchDB
Key-Value	Simple key and value storage	Redis, DynamoDB
Column-family	Stores data in columns	Cassandra, HBase
Graph-based	Represents networks and relationships	Neo4j, ArangoDB

3 NoSQL Limitations

While NoSQL databases offer significant advantages in flexibility and scalability, they also have several important limitations:

1. **Lack of Standardization:** Unlike SQL, NoSQL does not have a common query language. Each database (e.g., MongoDB, Cassandra, Redis) has its own API and query methods.
2. **Eventual Consistency:** Many NoSQL systems follow an eventual consistency model instead of strong consistency, which can lead to temporary inconsistencies in distributed systems.
3. **Limited Query Capabilities:** Features like joins, subqueries, and advanced filtering are often not supported or are more difficult to implement than in relational databases.
4. **Weaker ACID Guarantees:** Full ACID transactions are either limited or unsupported in some NoSQL systems, which may be unsuitable for applications that require strong consistency and atomic operations.
5. **Maturity and Tooling:** The ecosystem for some NoSQL databases may lack robust tools for monitoring, analytics, backup, and administrative tasks compared to mature RDBMS platforms.
6. **Steeper Learning Curve:** Developers familiar with relational databases must adapt to new paradigms, data models, and APIs specific to each NoSQL solution.

4 Suitable Scenarios: RDBMS vs NoSQL

The table 1 summarizes typical use cases and highlights whether a relational database (RDBMS) or a NoSQL solution is more suitable for each scenario.

5 Introduction to MongoDB

5.1 What is MongoDB?

- Document-oriented NoSQL database.
- Stores data in BSON (Binary JSON).
- Developed by MongoDB Inc.

Scenario	Best Choice	Reason
Complex transactions (e.g., banking systems)	RDBMS	Strong ACID compliance and integrity constraints
Dynamic, frequently changing data models	NoSQL	Schema flexibility
Structured data with clear relationships	RDBMS	Support for joins and relational modeling
High-speed, low-latency key-value lookups	NoSQL	Optimized for fast retrieval using keys
Large-scale web apps with horizontal scaling needs	NoSQL	Designed for distributed architecture and sharding
Data warehousing and analytical processing	RDBMS	Mature support for complex queries and aggregations
IoT or real-time event data ingestion	NoSQL	Handles high-velocity unstructured or semi-structured data
Applications requiring strict data integrity	RDBMS	Enforces referential integrity and transactions
Content management, catalogs, or user profiles	NoSQL	Suits document-based flexible data structures
Multi-record consistency and rollback requirements	RDBMS	Transaction support across multiple tables

Table 1: NoSQL vs RDBMS typical use cases.

5.2 MongoDB Architecture

- Hierarchy: **Database** → **Collection** → **Document**
- Documents resemble JSON objects.
- Collections group similar documents.

6 Core MongoDB Operations

To install and configure the MongoDB Shell, users should follow the official MongoDB documentation guidelines available at https://www.mongodb.com/docs/mongodb-shell/install/.

6.1 Connecting to MongoDB

To start a MongoDB shell type in your terminal:

```
mongosh
```

6.2 Creating a Database

The `use` command is used to switch to a specific database. If the specified database does not exist, MongoDB will create it when data is inserted.

```
use myDatabase
```

6.3 Creating a Collection

The `createCollection` method explicitly creates a new collection named `students`. Collections are created automatically when data is inserted, but this method allows manual creation.

```
db.createCollection("students")
```

6.4 Inserting Documents

Documents can be added to a collection using `insertOne` for a single document or `insertMany` for multiple documents. Each document is represented as a JSON-like structure.

```
db.students.insertOne({ name: "Alice", age: 23, major: "CS" })
db.students.insertMany([{ name: "Bob" }, { name: "Carol" }])
```

6.5 Reading Documents

The `find` method retrieves documents from a collection. Without parameters, it returns all documents. Filters can be applied using MongoDB query operators like `$gt` (greater than).

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

6.6 Updating Documents

The `updateOne` method modifies the first document that matches a given condition. The `$set` operator is used to update specific fields within the document.

```
db.students.updateOne({ name: "Alice" }, { $set: { age: 24 } })
```

6.7 Deleting Documents

The `deleteOne` method removes the first document that matches the specified condition from the collection.

```
db.students.deleteOne({ name: "Bob" })
```

7 Advanced Features

7.1 Indexing

Indexes improve the performance of queries by allowing MongoDB to quickly locate documents that match specified criteria, instead of scanning every document in a collection. The command below creates an ascending index on the `name` field, which speeds up queries that sort or filter by name.

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

7.2 Aggregation Framework

The aggregation framework is used for data processing and transformation, allowing operations such as filtering, grouping, and calculating summary statistics. In the example below, documents where `major` is `"CS"` are filtered using `$match`, then grouped by `age`, and a count is calculated for each group using `$sum`.

```
db.students.aggregate([
  { $match: { major: "CS" } },
  { $group: { _id: "$age", count: { $sum: 1 } } }
])
```

7.3 Replication and Sharding

Replication is MongoDB's built-in mechanism to ensure high availability and data redundancy. It uses a configuration called a *replica set*, which consists of a primary node and one or more secondary nodes. All write operations go to the primary node, and secondaries replicate the data. If the primary fails, one of the secondaries is automatically elected as the new primary, providing fault tolerance and minimal downtime.

Key benefits of replication:

- High availability through automatic failover
- Redundant copies of data across different nodes
- Option to distribute read operations across secondaries (read scaling)

Sharding is MongoDB's strategy for *horizontal scaling*, which distributes large datasets across multiple servers (shards). Each shard stores a portion of the data, determined by a *shard key*. A special query router component called `mongos` directs client queries to the appropriate shards.

Key benefits of sharding:

- Enables linear scalability as data volume and query load grow
- Supports large datasets that exceed the storage capacity of a single server
- Helps balance workloads across multiple machines

Together, replication and sharding form the foundation of MongoDB's distributed architecture, supporting both high availability and high scalability in large-scale production environments.

8 Expanded Use Cases for MongoDB

MongoDB is widely used across various industries due to its flexible document model, horizontal scalability, and real-time capabilities. Below are several common categories where MongoDB provides strong advantages:

E-commerce and Product Catalogs

Why MongoDB? Its flexible schema allows storing products with varying attributes such as size, color, and price. **Typical Features:**

- Dynamic product schemas
- Embedded reviews, inventory, and pricing details
- Fast filtering and search via indexes

Mobile and Web Applications

Why MongoDB? Real-time, user-centric data such as preferences, sessions, and settings are easily modeled using documents. **Typical Features:**

- User profile and session storage
- Support for push notifications and messaging
- Offline data sync with MongoDB Realm[1]

Location-Based Services

Why MongoDB? Built-in support for geospatial data types and queries makes it ideal for geo-aware applications. **Typical Features:**

- Proximity-based search and geofencing
- Delivery and ride-tracking systems
- Store locator and navigation services

[1]MongoDB Realm is a serverless platform and mobile database solution provided by MongoDB, designed to make it easier to build real-time, responsive applications—especially on mobile and edge devices. More info here.

Real-Time Analytics and Dashboards

Why MongoDB? Aggregation pipelines enable fast analysis and transformation of large-scale or streaming data. **Typical Features:**

- IoT sensor data processing
- Real-time dashboards and monitoring
- Business intelligence with dynamic schemas

Content Management Systems (CMS)

Why MongoDB? Easily handles diverse content types like articles, blogs, and multimedia. **Typical Features:**

- Articles with embedded metadata and media
- Tagging and categorization with nested fields
- Document versioning and history tracking

Healthcare and Scientific Research

Why MongoDB? Complex, variable data structures such as patient records or experiment results can be modeled and evolved over time. **Typical Features:**

- Patient histories with embedded diagnostic data
- Genomic and lab test result storage
- Audit logs and regulatory compliance

Artificial Intelligence and Machine Learning

Why MongoDB? Flexible enough to store models, training metadata, and results from experimentation. **Typical Features:**

- Storage of features extracted from raw data
- Model performance tracking and metadata logging
- Dataset and experiment versioning

Authentication and User Identity

Why MongoDB? Supports customizable user profiles and identity management systems. **Typical Features:**

- User role and permission structures
- Integration with OAuth, LDAP, or custom auth flows
- Activity logs and login history

9 Recommended Resources

To further your understanding of MongoDB and NoSQL databases, the following resources offer a mix of theoretical foundations, hands-on tutorials, and cloud-based tools:

- **MongoDB University** — A free educational platform provided by MongoDB Inc. It offers structured courses and learning paths for developers, DBAs, and data engineers. Topics include MongoDB basics, aggregation pipelines, performance tuning, and application integration. Each course includes videos, quizzes, and a certificate upon completion.
- ***MongoDB: The Definitive Guide*** by Kristina Chodorow — This book is one of the most comprehensive print resources available on MongoDB. It covers the architecture, query language, data modeling, indexing, and deployment strategies. Suitable for both beginners and intermediate users, it provides in-depth explanations and practical examples.
- **MongoDB Official Website** — The central hub for all the MongoDB ecosystem, including official documentation, community forums, release notes, tools, and APIs. It also provides links to integrations with programming languages like Python, JavaScript, Java, and Go.
- **MongoDB Atlas** — MongoDB's fully managed cloud database service. It supports automated deployment, backups, monitoring, and scaling. Atlas enables developers to build and run applications without managing database infrastructure. It includes support for multi-region replication, serverless functions, and real-time data services.
- **Coursera: Introduction to NoSQL Databases** — A beginner-friendly course that introduces key NoSQL concepts and MongoDB fundamentals. Modules 1 and 2 specifically cover the basics of NoSQL data models and MongoDB's architecture and operations. The course is offered by the University of Michigan and includes an *Audit* option, allowing you to access the material for free.

10 Practice Questions

1. What are the advantages of NoSQL over traditional relational databases?
2. Describe the differences between document-based and key-value databases.
3. Write a query to find all documents where `age > 30`.
4. Explain the role of indexing in MongoDB.
5. How does MongoDB handle horizontal scalability?

Solutions

1. What are the advantages of NoSQL over traditional relational databases?

- **Schema flexibility:** No predefined schema; allows dynamic changes.

- **Horizontal scalability:** Easily scales across multiple machines.
- **High performance:** Optimized for fast read/write operations.
- **Unstructured data handling:** Suitable for JSON/BSON or nested data.
- **Agile development:** Schema-less design supports rapid iteration.

2. Describe the differences between document-based and key-value databases.

Feature	Document-Based DB	Key-Value DB
Structure	Documents (e.g., JSON/BSON)	Simple key-value pairs
Querying	Supports queries on document fields	Access by exact key only
Use Cases	Nested data, rich querying	Caching, session storage
Examples	MongoDB, CouchDB	Redis, DynamoDB

3. Write a query to find all documents where `age > 30`.

```
db.collection.find({ age: { $gt: 30 } })
```

Explanation: `$gt` stands for ”greater than”. This query returns all documents where the `age` field is greater than 30.

4. Explain the role of indexing in MongoDB.

- Indexes speed up query execution by allowing fast lookup of documents.
- Without indexes, MongoDB scans the entire collection.
- Indexes are used for filtering, sorting, and aggregation operations.
- Common index types: single-field, compound, and text indexes.

Example:

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

5. How does MongoDB handle horizontal scalability?

MongoDB uses **sharding** to distribute data across multiple servers. A **shard key** is used to determine how data is partitioned, then each **shard** holds part of the data and can be queried independently. The **mongos** query router directs client requests to the appropriate shard(s). So, *sharding* enables high throughput and large dataset handling.