Need for NoSQL Databases and How They Overcome RDBMS Limitations

Traditional **Relational Database Management Systems (RDBMS)** were designed for structured, transactional data. With the growth of **Big Data**, web applications, and real-time systems, RDBMSs face several limitations. **NoSQL databases** were introduced to address these challenges.

Need for NoSQL Databases

The need for NoSQL databases arises due to the following reasons:

- **Massive data volume**
  - Modern applications generate terabytes to petabytes of data
  - RDBMSs struggle to scale efficiently at this level
- **High velocity data**
  - Data is generated continuously in real time
  - RDBMSs are not optimized for high-speed ingestion
- **Variety of data**
  - Data can be structured, semi-structured, or unstructured
  - RDBMSs require fixed schemas
- **Scalability requirements**
  - Web-scale applications need horizontal scaling
  - RDBMSs mainly support vertical scaling
- **High availability and fault tolerance**
  - Distributed applications require continuous uptime
  - RDBMSs are harder to distribute and replicate efficiently

---

Limitations of RDBMS

- Fixed schema makes it difficult to handle changing data structures
- Vertical scaling increases cost and complexity
- Poor performance with large unstructured datasets
- Complex joins reduce performance at scale
- Limited support for distributed systems

How NoSQL Databases Overcome RDBMS Limitations

1. Schema Flexibility

- NoSQL databases use **schema-less or flexible schemas**
- Easy to store semi-structured and unstructured data
- Suitable for frequently changing data models

---

2. Horizontal Scalability

- NoSQL systems scale by adding more machines
- Data is automatically distributed across nodes
- Cost-effective using commodity hardware

---

3. High Performance

- Optimized for fast read/write operations
- Avoids complex joins
- Uses simple data models (key-value, document, column, graph)

---

4. Distributed Architecture

- Built for distributed environments
- Provides replication and partitioning
- Ensures high availability and fault tolerance

---

5. Big Data Compatibility

- Integrates easily with Big Data tools like Hadoop and Spark
- Suitable for real-time analytics and streaming data

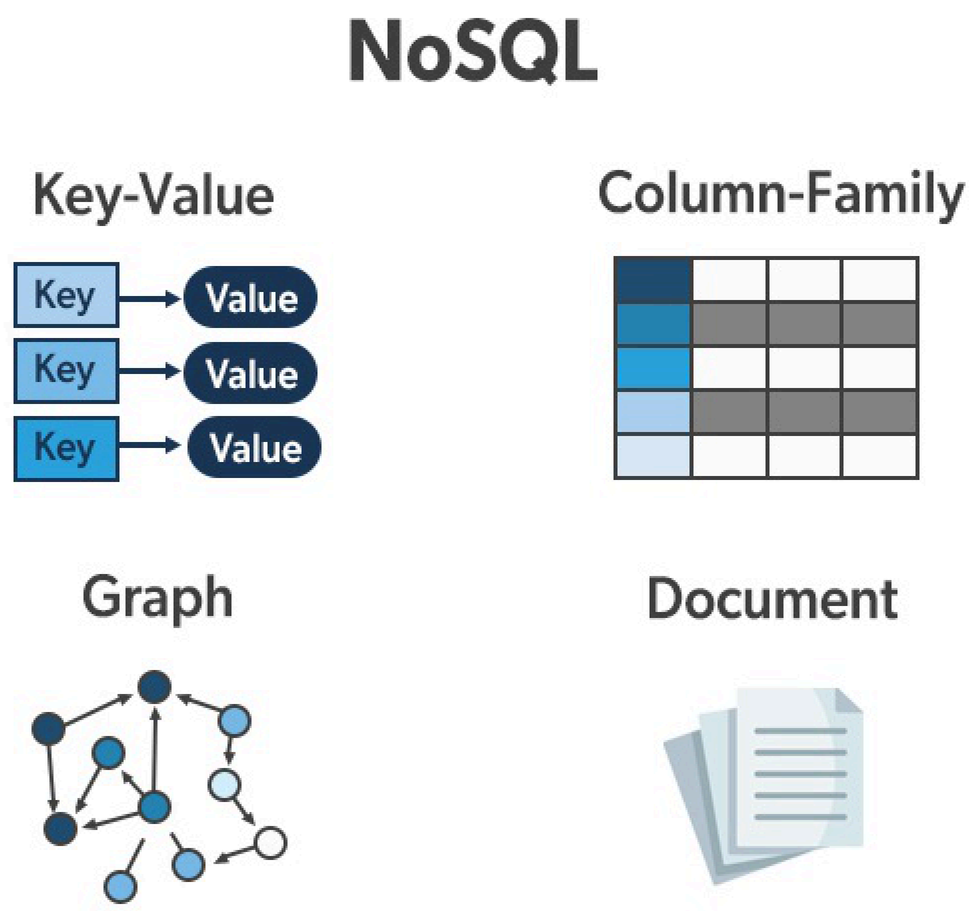
RDBMS vs NoSQL Comparison

Aspect	RDBMS	NoSQL
Schema	Fixed	Flexible / Schema-less
Scalability	Vertical	Horizontal
Data Type	Structured	Structured + Unstructured
Joins	Supported	Limited / Not used
Availability	Moderate	High
Use Cases	Banking, ERP	Big Data, Web apps

Discuss different types of NoSQL databases with examples.

Types of NoSQL Databases (with Examples)

NoSQL databases are designed to handle **large-scale, distributed, and unstructured data**. Based on how data is stored and accessed, NoSQL databases are broadly classified into **four main types**.



1. Key–Value Stores

Description:

- Data is stored as **key–value pairs**
- Each key is unique and maps to a value
- Simplest and fastest NoSQL model

Features:

- High performance
- Easy to scale
- No fixed schema

Examples:

- Redis
- Amazon DynamoDB
- Riak

Use Cases:

- Caching
- Session management
- User profiles

---

2. Document-Oriented Databases

Description:

- Data is stored as **documents** (JSON, BSON, XML)
- Each document contains key–value pairs
- Schema is flexible

Features:

- Easy to read and write
- Supports nested data
- Suitable for semi-structured data

Examples:

- MongoDB
- CouchDB

Use Cases:

- Content management systems
- E-commerce applications
- Mobile and web apps

---

3. Column-Family (Column-Oriented) Databases

Description:

- Data stored in **columns instead of rows**
- Groups related columns into column families
- Optimized for large datasets

Features:

- High write performance
- Scales horizontally
- Suitable for analytical queries

Examples:

- Apache HBase
- Apache Cassandra

Use Cases:

- Time-series data
- IoT data storage
- Real-time analytics

---

4. Graph Databases

Description:

- Data stored as **nodes and relationships**
- Focus on relationships between data
- Uses graph theory concepts

Features:

- Fast relationship traversal
- Flexible schema
- Ideal for connected data

Examples:

- Neo4j
- Amazon Neptune

Use Cases:

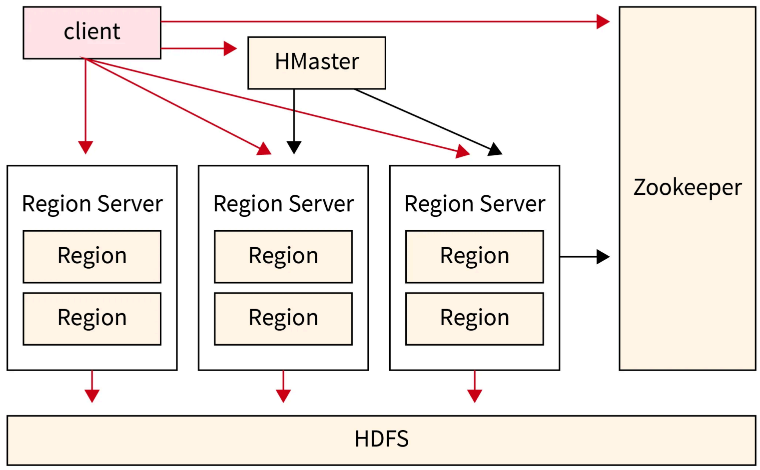
- Social networks
- Recommendation systems
- Fraud detection

Summary Table

NoSQL Type	Data Model	Examples	Use Cases
Key–Value	Key–Value pairs	Redis	Caching
Document	JSON/XML documents	MongoDB	Web apps
Column-Family	Columns & families	HBase	Big Data
Graph	Nodes & edges	Neo4j	Social networks

HBase Architecture and CRUD Operations

HBase is a **distributed, column-oriented NoSQL database** built on top of **HDFS**. It is designed for **real-time read/write access** to very large datasets and follows a **master–slave architecture**.



Apache HBase Architecture

1. HBase Architecture

HBase architecture consists of the following main components:

1.1 HMaster

- Acts as the **master node**
- Manages RegionServers
- Assigns regions to RegionServers
- Handles schema changes (create/alter/delete tables)
- Performs load balancing and failure recovery

1.2 RegionServer

- Runs on slave nodes
- Hosts and manages **regions**

- Handles **read and write requests** from clients
- Each RegionServer contains:
  - Regions
  - MemStore
  - HFiles

1.3 Regions

- Tables are split into **regions**
- Each region stores a range of row keys
- Enables horizontal scalability

1.4 ZooKeeper

- Provides coordination and synchronization
- Maintains configuration information
- Helps in leader election and failure detection

1.5 HDFS

- Used as the underlying storage system
- Stores HFiles permanently
- Provides fault tolerance and replication

---

2. HBase Data Storage Components (Brief)

- **MemStore**
  - In-memory write buffer
  - Stores data before writing to disk
- **HFile**
  - Disk-based storage format

- Stored in HDFS
- Optimized for fast reads

---

3. CRUD Operations in HBase

CRUD stands for **Create, Read, Update, Delete**.

3.1 Create (Put Operation)

- Used to insert data into HBase
- Data is written to MemStore first
- Later flushed to HFiles in HDFS

Example:

```
put 'student', '101', 'info:name', 'Rahul'
```

3.2 Read (Get / Scan Operation)

Get

- Retrieves data for a specific row key

```
get 'student', '101'
```

Scan

- Retrieves multiple rows

```
scan 'student'
```

3.3 Update

- HBase does not update data in place
- New version of data is written with a timestamp
- Old versions are retained (based on configuration)

Example:

```
put 'student', '101', 'info:name', 'Amit'
```

3.4 Delete

- Removes data logically
- Marks data with a delete marker
- Actual deletion happens during compaction

Examples:

```
delete 'student', '101', 'info:name'
deleteall 'student', '101'
```

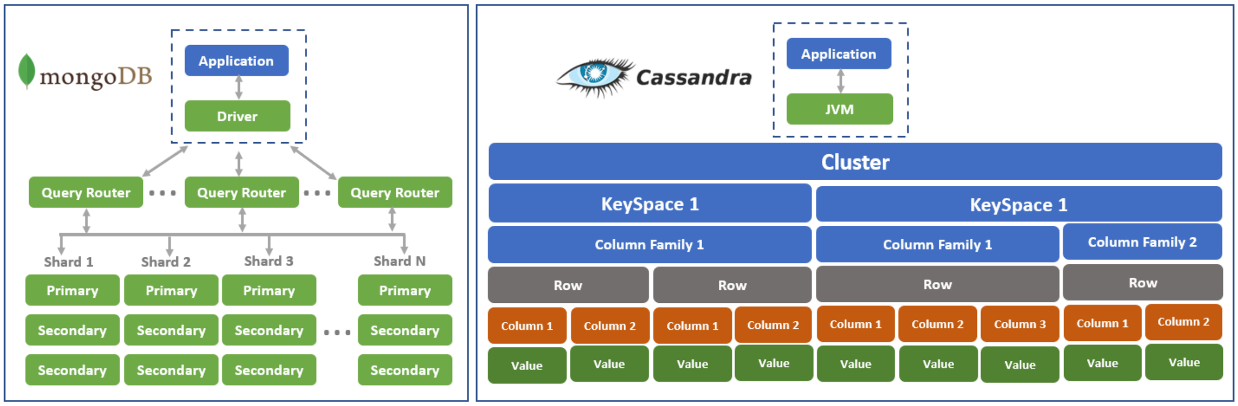
---

4. Advantages of HBase

- Real-time read/write access
- Highly scalable
- Fault tolerant
- Handles massive datasets efficiently

Comparison of MongoDB and Cassandra (Data Model & Scalability)

MongoDB and Cassandra are popular **NoSQL databases**, but they differ significantly in their **data models** and **scalability approaches**, making them suitable for different use cases.



---

1. Data Model Comparison

MongoDB

- Uses a **document-oriented data model**
- Data stored as **JSON/BSON documents**
- Schema is flexible and easy to modify
- Supports nested documents and arrays
- Suitable for semi-structured data

Example:

```
{
"id":101,
"name":"Rahul",
"course":"MCA"
}
```

Cassandra

- Uses a **column-family (wide-column) data model**
- Data stored in rows and columns grouped into column families
- Schema must be designed based on query patterns
- Optimized for large-scale writes

Example:

```
(student_id, course,marks)
```

---

2. Scalability Comparison

MongoDB

- Supports **horizontal scalability using sharding**
- Sharding distributes data across multiple nodes
- Scaling requires shard key selection
- Good scalability, but complex for very large clusters

Cassandra

- Designed for **massive horizontal scalability**
- Uses a **peer-to-peer architecture**
- No single point of failure
- Nodes can be added easily without downtime
- Excellent for large, distributed systems

---

3. Tabular Comparison

Aspect	MongoDB	Cassandra
Data Model	Document-oriented	Column-family
Schema	Flexible	Query-based schema
Data Format	JSON / BSON	Rows and columns
Scalability	Horizontal (Sharding)	Linear horizontal scalability
Architecture	Master–slave (Replica sets)	Peer-to-peer
Write Performance	High	Very high
Availability	High	Very high
Best Use Case	Web & mobile apps	Big Data, IoT, time-series