Unit-1

I. Need of Database

- Efficient Data Storage and Management
  - Databases provide structured storage for large volumes of data.
  - Unlike file systems, databases allow for efficient data retrieval and organization.
- Data Consistency and Integrity
  - Ensures data accuracy through constraints like primary keys, foreign keys, and unique constraints.
  - Transactions maintain consistency through ACID (Atomicity, Consistency, Isolation, Durability) properties.
- Faster Data Retrieval and Querying
  - Databases use indexing and optimized search algorithms for quick data access.
  - SQL (Structured Query Language) allows users to retrieve specific data efficiently.
- Data Security
  - Role-based access control (RBAC) ensures only authorized users can access or modify data.
  - Encryption and authentication mechanisms protect sensitive information.
- Elimination of Data Redundancy
  - Databases normalize data, reducing duplication and saving storage space.
  - Relationships between tables prevent unnecessary repetition.
- Scalability and Performance
  - Databases support horizontal and vertical scaling to accommodate growing data needs.
  - Technologies like indexing, caching, and partitioning enhance performance.
- Concurrent Access and Multi-User Support
  - Multiple users can access the database simultaneously without conflicts.
  - Locking mechanisms and isolation levels prevent data corruption.
- Backup and Recovery
  - Databases provide automated backup and recovery options to prevent data loss.

- Transaction logs help restore data in case of failure.

➢ Data Relationships and Complex Queries
- Relational databases allow complex relationships between different datasets.
- Joins and aggregations help analyze large datasets efficiently.

➢ Support for Big Data and Analytics
- Modern databases integrate with big data tools for analytics and business intelligence.
- NoSQL and NewSQL databases handle unstructured and semi-structured data efficiently.

➢ Integration with Applications
- Databases serve as the backend for web and mobile applications.
- APIs and ORM (Object-Relational Mapping) frameworks enable seamless interaction with applications.

II. Different types of databases

1. Relational Databases (RDBMS)

- **Definition:**
  - Stores data in structured tables with rows and columns.
  - Relationships between tables are maintained using keys.
- **Characteristics:**
  - Uses **SQL (Structured Query Language)** for querying.
  - Follows **ACID (Atomicity, Consistency, Isolation, Durability)** properties.
  - Data normalization eliminates redundancy.
- **Examples:**
  - MySQL
  - PostgreSQL
  - Microsoft SQL Server
  - Oracle Database

2. NoSQL Databases

- **Definition:**
  - Handles unstructured, semi-structured, and distributed data.
  - Designed for high scalability and flexibility.
- **Characteristics:**
  - Flexible schema (supports JSON, XML, key-value pairs, etc.).

- Horizontally scalable (handles big data efficiently).
- Categorized into different models:

(a) Key-Value Stores

- **Definition:** Stores data as key-value pairs (like a dictionary).
- **Examples:**
  - Redis
  - Amazon DynamoDB
  - Riak

(b) Document-Oriented Databases

- **Definition:** Stores data in document formats like JSON or BSON.
- **Examples:**
  - MongoDB
  - CouchDB
  - Firebase Firestore

(c) Column-Family Stores

- **Definition:** Organizes data into columns instead of rows for efficient big data analytics.
- **Examples:**
  - Apache Cassandra
  - HBase

(d) Graph Databases

- **Definition:** Stores data as nodes and relationships instead of tables.
- **Examples:**
  - Neo4j
  - Amazon Neptune
  - ArangoDB

3. Object-Oriented Databases (OODBMS)

- **Definition:**
  - Stores data as objects, similar to object-oriented programming.
- **Examples:**
  - ObjectDB
  - db4o

4. Time-Series Databases (TSDB)

- **Definition:**
  - Optimized for handling time-stamped or sequentially ordered data.

- **Examples:**
  - InfluxDB
  - TimescaleDB

5. Graph Databases

- **Definition:**
  - Used to store and analyze relationships between entities.
- **Examples:**
  - Neo4j
  - ArangoDB

6. Hierarchical Databases

- **Definition:**
  - Data is structured in a tree-like hierarchy (parent-child relationships).
- **Examples:**
  - IBM Information Management System (IMS)

7. Network Databases

- **Definition:**
  - Similar to hierarchical databases but allows multiple parent-child relationships (many-to-many).
- **Examples:**
  - Integrated Data Store (IDS)

8. Multi-Model Databases

- **Definition:**
  - Supports multiple database types (relational, document, key-value) within a single system.
- **Examples:**
  - ArangoDB
  - Cosmos DB

9. NewSQL Databases

- **Definition:**
  - Combines the benefits of SQL databases with NoSQL scalability.
- **Examples:**
  - Google Spanner
  - CockroachDB

III. Relational vs Non-Relational Database

Relational Databases (RDBMS)

- **Definition:** Store data in structured tables with rows and columns, maintaining relationships between tables using keys.
- **Features:**
  - Uses **SQL (Structured Query Language)** for queries.
  - Ensures **ACID (Atomicity, Consistency, Isolation, Durability)** compliance for data integrity.
  - Data is **structured and normalized** to reduce redundancy.
- **Examples:** MySQL, PostgreSQL, Oracle, SQL Server.
- **Use Cases:** Banking, CRM, ERP, and transactional systems.

Non-Relational Databases (NoSQL)

- **Definition:** Designed for unstructured, semi-structured, or large-scale distributed data.
- **Features:**
  - Flexible schema (supports JSON, XML, key-value, graphs, etc.).
  - Highly **scalable and optimized for big data**.
  - Different models: **Key-Value, Document, Column-Family, Graph**.
- **Examples:** MongoDB, Cassandra, Redis, Neo4j.
- **Use Cases:** Big data applications, real-time analytics, content management, IoT.

IV. **Introduction to NoSQL database**

NoSQL (Not Only SQL) databases are a category of databases designed to handle **large volumes of unstructured, semi-structured, and structured data** efficiently. Unlike traditional **relational databases (RDBMS)**, NoSQL databases offer **flexibility, scalability, and high performance**, making them ideal for modern applications.

Key Features of NoSQL Databases:

- **Schema-less:** No fixed table structure, allowing flexible data storage.

- **Scalability:** Supports **horizontal scaling** to handle big data and high traffic.
- **High Performance:** Optimized for fast read and write operations.
- **Distributed Architecture:** Often used in cloud-based and distributed systems.
- **Multiple Data Models:** Supports key-value, document, column-family, and graph storage models.

V. NoSQL features

1. **Schema Flexibility**
    - No fixed schema; data structure can change dynamically.
    - Supports **JSON, BSON, XML, or key-value pairs** instead of rigid tables.
2. **Scalability**
    - **Horizontally scalable** (adding more servers instead of upgrading hardware).
    - Handles large-scale applications efficiently.
3. **High Performance**
    - Optimized for **fast read/write operations** with low latency.
    - Uses **distributed storage** for quick data access.
4. **Distributed Architecture**
    - Data is replicated across multiple nodes for fault tolerance.
    - Ensures high availability, even in case of hardware failures.
5. **Variety of Data Models**
    - Supports multiple formats, including:
        - **Key-Value Stores** (e.g., Redis, DynamoDB)
        - **Document-Oriented** (e.g., MongoDB, CouchDB)
        - **Column-Family Stores** (e.g., Cassandra, HBase)
        - **Graph Databases** (e.g., Neo4j, ArangoDB)
6. **Eventual Consistency**
    - Ensures **availability and partition tolerance** over strict consistency.
    - Follows **BASE (Basically Available, Soft-state, Eventually consistent)** principles instead of ACID.
7. **Support for Big Data and Real-Time Applications**
    - Efficiently manages **high volumes of unstructured data**.
    - Used in **IoT, social media, and analytics** applications.
8. **Easy Integration with Cloud & Microservices**
    - Works well with **cloud platforms** (AWS, Google Cloud, Azure).

- Supports **microservices architectures** for modern web applications.

9. **High Availability & Fault Tolerance**
   - Automatic data replication ensures **continuous uptime**.
   - Reduces **single points of failure** in distributed systems.
10. **Automatic Sharding & Partitioning**

- Data is **automatically split across servers** for better performance.
- Ensures **load balancing** without manual intervention.

VI. Different types of NoSQL database

NoSQL databases are designed to handle large volumes of structured, semi-structured, and unstructured data. They are flexible, scalable, and optimized for high performance. NoSQL databases can be classified into four main types:

---

1. Document-Oriented Databases

Structure: Store data in the form of JSON, BSON, or XML documents, making them ideal for hierarchical data representation.
Best for: Content management systems, e-commerce catalogs, and applications that require flexible schemas.
Examples:

- **MongoDB:** Uses BSON (Binary JSON) format and supports indexing, aggregation, and replication.
- **CouchDB:** Uses JSON for documents, JavaScript for queries, and HTTP for API communication.

Advantages:

✓ Flexible schema allows dynamic changes.

✓ High scalability and fast read/write operations.

✓ Suitable for hierarchical and nested data structures.

Disadvantages:

✕ Querying can be slower compared to relational databases if indexing is not optimized.

✕ Transactions are limited compared to relational databases.

---

2. Key-Value Stores

Structure: Store data as key-value pairs, similar to a dictionary.
Best for: Caching, session management, and real-time applications.

Examples:

- **Redis:** An in-memory data store supporting key-value operations with high-speed performance.
- **DynamoDB:** Amazon's fully managed key-value database with built-in security and scalability.
- **Riak:** A distributed key-value database with fault tolerance and scalability.

Advantages:

✓ Extremely fast read and write operations.

✓ Highly scalable and distributed.

✓ Simple and efficient data retrieval.

Disadvantages:

✕ Not suitable for complex queries or relationships.

✕ Lack of advanced querying mechanisms like joins.

---

3. Column-Family Stores

Structure: Store data in columns rather than rows, making them highly efficient for read-heavy applications.

Best for: Big data analytics, data warehousing, and real-time reporting.

Examples:

- **Apache Cassandra:** A distributed database that provides high availability and fault tolerance.
- **HBase:** A NoSQL database built on Hadoop for handling large-scale data.
- **ScyllaDB:** A high-performance replacement for Apache Cassandra.

Advantages:

✓ Optimized for read-heavy and write-heavy applications.

✓ Highly scalable and supports distributed architecture.

✓ Efficient in handling large-scale datasets.

Disadvantages:

✕ Complex data modeling compared to relational databases.

✕ Requires additional effort for indexing and query optimization.

---

4. Graph Databases

Structure: Store data as nodes (entities) and edges (relationships), making them ideal for connected data.

Best for: Social networks, fraud detection, recommendation engines, and network topology analysis.

Examples:

- **Neo4j:** A widely used graph database with Cypher query language.
- **Amazon Neptune:** A managed graph database for connected applications.
- **ArangoDB:** A multi-model database supporting graph, document, and key-value data models.

Advantages:

✓ Excellent for relationship-based queries.

✓ Flexible schema allows changes without affecting performance.

✓ Optimized for deep link analysis and path traversal queries.

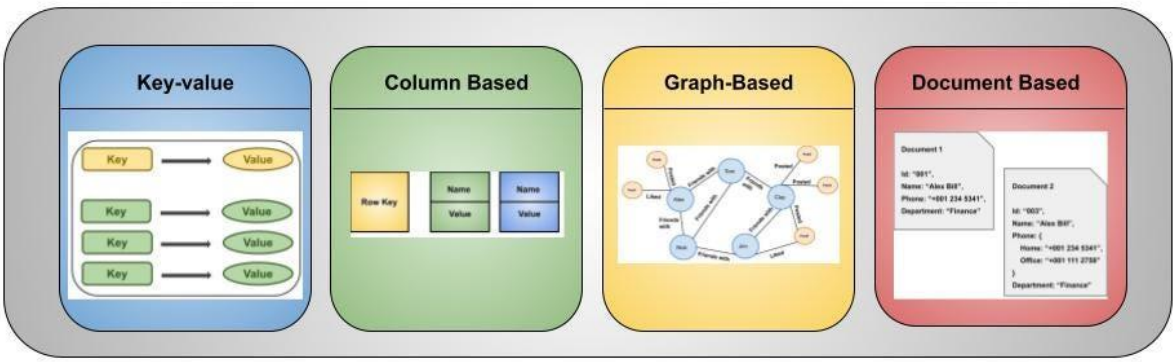
Disadvantages:

✕ Not ideal for transactional applications with structured data.

✕ Can be more complex to scale compared to other NoSQL types.

Comparison of NoSQL Database Types

Type	Data Model	Use Case Example	Scalability	Query Complexity
Document	JSON, BSON	CMS, E-commerce	High	Medium
Key-Value	Key-Value Pairs	Caching, Sessions	Very High	Low
Column-Family	Columns	Big Data Analytics	High	High
Graph	Nodes & Edges	Social Networks, Fraud Detection	Medium	Very High



VII. Introduction to MongoDB

MongoDB is a popular **NoSQL database** that stores data in a **document-oriented** format. It is designed for **high performance, scalability, and flexibility**, making it suitable for modern applications that require handling large volumes of unstructured or semi-structured data.

Key Features of MongoDB

✅ Document-Oriented Storage

- Data is stored in **JSON-like BSON (Binary JSON) format** instead of traditional rows and columns.
- Each **document** contains key-value pairs, making it flexible and easy to manage.

✅ Schema Flexibility

- Unlike relational databases, MongoDB **does not require a fixed schema**.
- New fields can be added without affecting existing data.

✅ Scalability & High Performance

- Uses **sharding** (horizontal scaling) to handle large amounts of data.
- **Indexing** and **replication** ensure fast read/write operations.

✅ Rich Query Language

- Supports **CRUD operations** (Create, Read, Update, Delete).
- Allows **filtering, aggregation, sorting, and text search**.

✅ Replication & High Availability

- **Replica sets** ensure data redundancy and automatic failover.

✅ Aggregation Framework

- Similar to SQL's **GROUP BY** and aggregate functions, enabling powerful data processing.

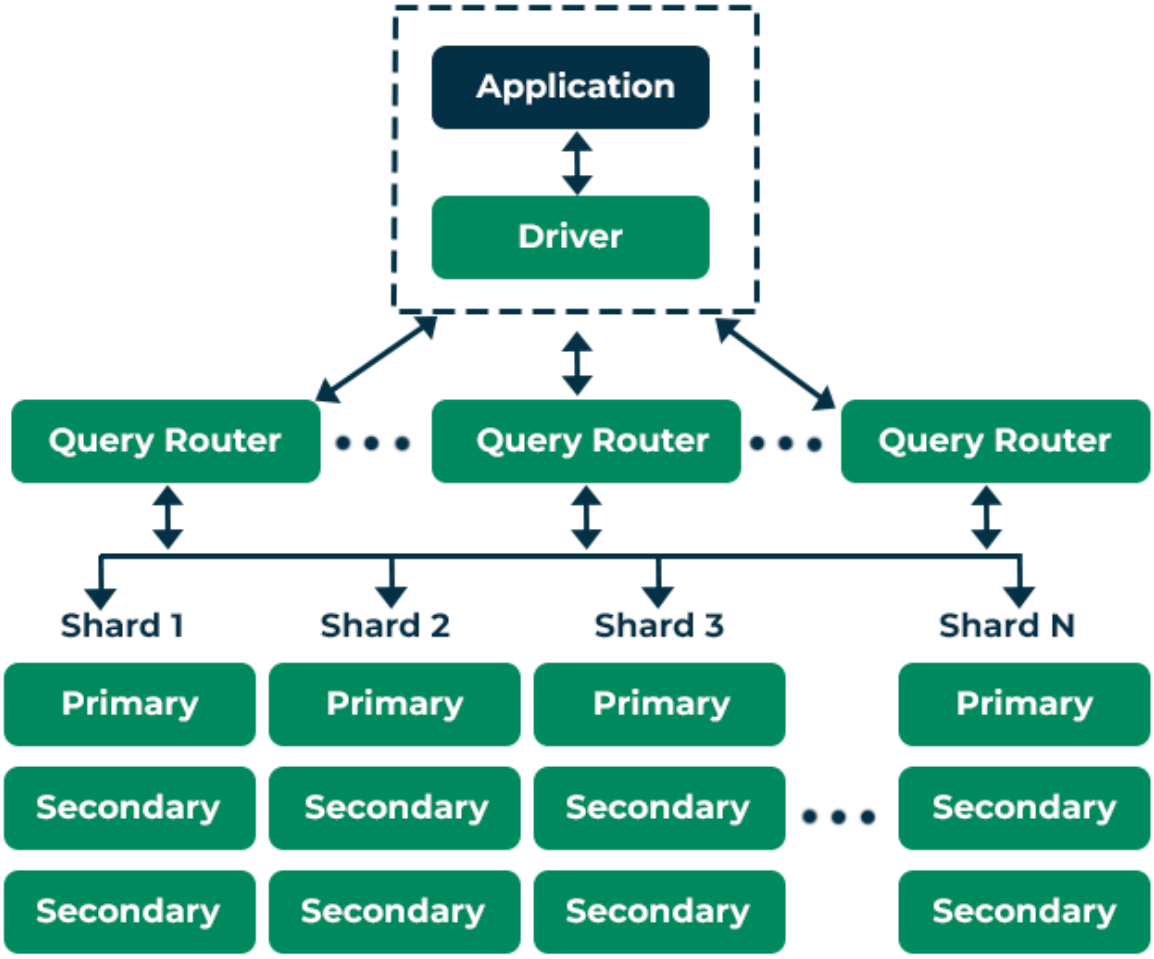
Basic MongoDB Concepts

Concept	Description
Database	A container for collections (similar to a relational database).
Collection	A group of documents (similar to a table in SQL).
Document	A JSON-like data structure containing key-value pairs (similar to a row).

Field	A key-value pair inside a document (similar to a column in SQL).
Index	Speeds up query performance by creating efficient data structures.
Replica Set	A set of MongoDB instances that provide redundancy and high availability.
Sharding	A method of distributing data across multiple servers for scalability.

VIII. MongoDB Architecture

MongoDB's architecture design involves several important parts that work together to create a strong and flexible database system. these are the following MongoDB's architecture



1. Drivers & Storage Engine

MongoDB store the data on the server but that data we will try to retrieve from our application. So that time how the communication is happening between our application and MongoDB server.

Any application which is written in python, .net and java or any kind of frontend application, these application are trying to access the data from these physical storage in server. First they will interact with driver which will communicate with **MongoDB** server. What happen is once the request is going from the **frontend application** through the driver then driver will change

appropriate query by using query engine and then the query will get executed in MongoDB data model. Left side is security which provides security to the database that who will access the data and right side is management this management will manage all these things.

Drivers

Drivers are client libraries that offer **interfaces** and **methods** for applications to communicate with MongoDB databases. Drivers will handle the translation of documents between **BSON objects** and mapping application structures.

.NET, Java, JavaScript, Node.js, Python, etc are some of the widely used drives supported by MongoDB.

Storage Engine

The storage engine significantly influences the performance of applications, serving as an intermediary between the MongoDB database and persistent storage, typically disks. MongoDB supports different storage engines:

- **MMAPv1** – It is a traditional **storage engine** based on **memory mapped files**. This storage engine is optimized for workloads with high volumes of read operations, insertions, and in-place updates. It uses **B-tress** to store indexes. Storage Engine works on multiple reader single writer lock. A user cannot have two write calls to be processes in parallel on the same collection. It is fast for reads and slow for writes.
- **Wired Tiger** – Default Storage Engine starts from MongoDB 3version. No locking Algorithms like **hash pointer**. It yields 7x-10x better write operations and 80% of the file system compression than MMAP.
- **InMemory** – Instead of storing documents on disk, the engine uses in-memory for more **predictable data latencies**. It uses 50% of physical **RAM** minimum 1 GB as default. It requires all its data. When dealing with large datasets, the in-memory engine may not be the most suitable choice.

2. Security

- Authentication
- Authorization
- Encryption on data
- Hardening (Ensure only trusted hosts have access)

3. MongoDB Server

It serves as the **central element** and is in charge of **maintaining**, **storing**, and **retrieving data** from the database through a number of interfaces. The system's heart is the MongoDB server. Each mongod server instance is in charge of handling client requests, maintaining data storage, and performing database operations. Several mongod instances work together to form a cluster in a typical MongoDB setup.

4. MongoDB Shell

For dealing with MongoDB databases, MongoDB provides the MongoDB Shell **command-line interface** (CLI) tool. The ability to handle and query MongoDB data straight from the terminal is **robust and flexible**. After installing MongoDB, you may access the MongoDB Shell, often known as mongo. It interacts with the database using JavaScript-based syntax. Additionally, it has built-in help that shows details about possible commands and how to use them.

5. Data Storage in MongoDB

5.1 Collections

A database can contain as many collections as it wishes, and MongoDB stores **data inside collections**.

5.2 Documents

Documents themselves represent the individual records in a specific collection.

6. Indexes

Indexes are data structures that make it simple to navigate across the collection's data set. They help to execute queries and find documents that match the query criteria without a collection scan.

These are the following different types of indexes in MongoDB:

6.1 Single field

MongoDB can traverse the indexes either in the **ascending** or **descending order** for single-field index

db.students.createIndex({“item”:1})

In this example, we are creating a single index on the item field and 1 here represents the filed is in ascending order.

A **compound index** in MongoDB contains multiple single filed indexes separated by a comma. MongoDB restricts the number of fields in a compound index to a maximum of 31.

db.students.createIndex({“item”: 1, “stock”:1})

Here, we create a compound index on item: 1, stock:1

6.2 Multi-Key

When indexing a filed containing an **array value**, MongoDB creates **separate index** entries for each array component. MongoDB allows you to create multi-key indexes for arrays containing scalar values, including strings, numbers, and nested documents.

db.students.createIndex({<filed>: <1 or -1>})

6.3 Geo Spatial

Two geospatial indexes offered by MongoDB are called **2d indexes** and **2d sphere indexes**. These indexes allow us to query geospatial data. On this case, queries intended to locate data stored on a two-dimensional plane are supported by the 2d indexes. On the other hand, queries that are used to locate data stored in spherical geometry are supported by 2D sphere indexes.

6.4 Hashed

To maintain the entries with **hashes** of the values of the indexed field we use Hash Index. MongoDB supports hash based sharding and provides hashed indexes.

db.<collection>.createIndex({ item: “hashed” })

7. Replication

Within a MongoDB cluster, data replication entails keeping several copies of the same data on various servers or nodes. Enhancing **data availability** and **dependability** is the main objective of data replication. A replica may seamlessly replace a failing server in the cluster to maintain service continuity and data integrity.

- **Primary Node (Primary Replica):** In a replica set, the primary node serves as the main source for all write operations. It's the only node that accepts write requests. The main node is where all data modifications begin and are implemented initially.

- **Secondary Nodes:** Secondary nodes duplicate data from the primary node (also known as secondary replicas). They are useful for dispersing read workloads and load balancing since they are read-only and mostly utilized for read activities.

8. Sharding

Sharding is basically **horizontal scaling** of databases as compared to the traditional vertical scaling of **adding more CPUS** and ram to the current system.

The partioning of data in a sharded environment is done on a range based basis by deciding a field as a shard key.