MongoDB Class 1

What is Database?

A database is an organized collection of data that can be easily accessed, managed, and updated.

Think of it like a filing cabinet:

- You store data in a structured way.
- You can retrieve or update it quickly when needed.

Examples:

- Contact list in your phone
- Student records in a school
- Product details in an online store

Why do we need databases?

Without a database, data would be stored in:

- Plain text files
- Excel sheets
- Random formats (slow, insecure, unscalable)

Which is

- Hard to manage
- Stored in plain files (slow and insecure)
- Difficult to *search*, filter, or *connect*

We need databases to:

- Store large amounts of data efficiently
- Retrieve data quickly
- Manage users, security, and access
- Ensure consistency (avoid duplicate/incorrect data)
- Allow multiple users to work with the same data safely

Types of Databases?

Databases are primarily two types

1. ### Relational Databases (SQL)

   Data is stored in tables (like Excel spreadsheets)

   - Tables with defined schema
   - Strong ACID compliance (Atomicity, Consistency, Isolation, Durability)
   - Ideal for:
     - Financial systems
     - Inventory systems
     - Applications needing complex JOINs

   Important Concepts:

   - Primary Key
   - Foreign Key
   - Normalization
   - Relationships: One-to-One, One-to-Many, Many-to-Many

   Pros:

   - Strong data consistency
   - Powerful querying with SQL
   - Good for complex relationships

   Cons:

   - Rigid schema
   - Scaling horizontally is harder

   Examples: MySQL, PostgreSQL, SQLite, Microsoft SQL Server

2. ### Non-Relational Databases (NoSQL)

   Data is stored in flexible formats like JSON-like documents, key-value pairs, graphs, etc

   Example:

```
{
  "name": "Nitesh",
  "email": "nitesh@example.com",
  "role": “Tech”
}
```

Store data in flexible formats like:

- Documents (MongoDB)
- Key-Value Pairs (Redis)
- Graphs (Neo4j)
- Column Stores (Cassandra)

Pros:

- Flexible schema (easy to evolve)
- Scales easily for large apps
- Stores nested and complex data directly

Cons:

- Less strict data consistency (eventual consistency in distributed systems)
- Queries can be less powerful for complex joins

Examples: MongoDB, Cassandra, Redis, Firebase

When we choose SQL or MongoDB

Feature	SQL (MySQL/PostgreSQL)	MongoDB (NoSQL)
Data Structure	Table-based (rows & columns)	Document, Key-Value, Graph, Column-oriented
Schema	Fixed, pre-defined tables	Flexible, dynamic schema
Data Relationships	Strong support via joins	Weak (manual reference or embedding)

Transactions	Fully ACID-compliant	Often eventual consistency, but some NoSQL DBs provide ACID support
Scaling	Vertical (scale-up)	Horizontal (scale-out)
Query Language	SQL (Structured Query Language)	Custom queries per database (e.g., MongoDB uses BSON query)
Use Cases	Banking, ERPs, Inventory	Real-time apps, CMS, analytics

When to Use SQL (Relational Database)

Use SQL when:

- Your data has strong relationships (e.g., customers and orders).
- You need ACID transactions for accuracy (e.g., financial systems).
- Schema is well-defined and doesn’t change frequently.
- You need complex queries and joins.

Use Case Example:

Banking system - You need accurate balance updates, foreign key constraints, and transaction support.

When to Use NoSQL (Non-Relational Database)

Use NoSQL when:

- Data is semi-structured or unstructured (e.g., JSON).
- Schema may evolve over time.
- You want fast, scalable performance with big data.
- You're building real-time or distributed applications.

Use Case Example:

Social media platforms - Posts, comments, user profiles may vary in structure and grow rapidly. NoSQL like MongoDB fits well here.

What is Schema?

Definition: A schema is the structure or blueprint of how data is organized in a database.

Think of a schema like a form or template – it tells what kind of data can be stored, how it's labeled, and the types (like text, numbers, dates).

In a database, schema defines:

- Field names (like name, email, age)
- Data types (string, number, boolean, date)
- Constraints (like required, unique, default, etc.)

Fixed Schema (Relational Databases like MySQL, PostgreSQL)

- In a fixed schema, table structure is predefined.
- Every row in the table must follow the same structure.

Example (Relational Table):

ID	Name	Email	Company
1	Munit	munit@gmail.com	Amazon
2	Priyanshu	priyanshu@gmail.com	Microsoft

Here, the columns are fixed. You can’t add extra fields without altering the whole table.

Pros:

- Well-structured and strict.
- Data integrity is high.
- Easy to use with complex joins and relationships.

Cons:

- Less flexible for rapidly changing data needs.

- Schema changes (like adding a column) need migrations.

Flexible Schema (MongoDB and other NoSQL Databases)

- In MongoDB, schema is flexible by default.
- You can store documents with different structures in the same collection.

Example (MongoDB Collection):

```
// Document 1
{
  "name": "Munit",
  "email": "munit@gmail.com",
  "company": “Amazon”
}

// Document 2
{
  "name": "Priyanshu",
  "phone": "1234567890",
  "location": "Delhi"
}
```

Both documents are in the same collection, but their fields are different.

Pros:

- Great for fast-changing data and agile development.
- Ideal when data fields are optional or vary across records.

Cons:

- Can lead to inconsistent data if not handled carefully.
- No built-in enforcement unless using schema validation (like Mongoose).

What is Scalability?

Definition: Scalability is the ability of a system to handle increased load.

There are two types:

Type	Meaning
Vertical Scaling	Increasing power of a single server (CPU, RAM, SSD)
Horizontal Scaling	Adding more servers to handle the load (distributing the work)

RDBMS = Vertically Scalable

Why?

- Relational databases (like MySQL, PostgreSQL, Oracle) are monolithic — designed to run on one central server.
- Data is strictly structured with relationships (foreign keys, joins), which makes splitting across servers very complex.

Example: If you're joining 5 tables with millions of rows, and you try to split across servers, the JOINs can get extremely slow or even break.

How do you scale?

- You upgrade your existing server: add more RAM, CPU, SSD.
- But... this has limits: hardware gets expensive and cannot scale infinitely.

Analogy:
Think of vertical scaling like buying a more powerful laptop – faster and better, but only one person can use it.

NoSQL = Horizontally Scalable

Why?

- NoSQL databases like MongoDB, Cassandra, DynamoDB, etc., are built for distributed systems.
- They don’t rely on complex joins or strict schemas, so data can be easily partitioned (aka sharded) across multiple machines.

Example: In MongoDB, if you have 1 crore documents, they can be spread across 10 servers – each one stores 10 lakh documents.

How do you scale?

- You add more cheap servers as load increases.
- Each server handles part of the data = scales horizontally.

Analogy:
Think of horizontal scaling like a team of delivery people. One person can deliver only 10 packages a day. Instead of giving 50 to one person, hire 5 people.

MongoDB Installation and Tools

MongoDB Ecosystem Tools

Tools	Purpose
mongod	MongoDB *server process* (daemon). Responsible for handling requests, storing data, and replication.
mongosh	MongoDB Shell, used to interact with the database via CLI (replaces the older mongo shell).
mongos	Router for sharded clusters, used in distributed systems to route queries to the correct shard.
compass	GUI client for MongoDB, useful for visualizing data, querying collections, and performing CRUD operations.

To Download Mongod:

https://www.mongodb.com/try/download/community

To Download Mongosh Shell:

https://www.mongodb.com/try/download/shell

To Download MongoDB Compass:

https://www.mongodb.com/try/download/compass