Notes

1. Relational Databases (RDBMS)

- **Concept:** Data organized into tables (rows/columns) with relationships (keys). Ensures integrity via ACID properties.
- **Key Features:** Tables, schemas, relationships (foreign keys), SQL, and strong consistency.
- **Examples:** MySQL, PostgreSQL, Oracle.
- **Use Cases:** Banking, e-commerce, structured data applications.
- **Pros:** Robust, consistent, standardized query language.
- **Cons:** Rigid schema, complex scaling, less suitable for unstructured data.

2. Non-Relational Databases (NoSQL)

- **Concept:** Flexible data storage (key-value, document, graph, column-family) optimized for scalability and speed.
- **Key Features:** Flexible schemas, varied data models, horizontal scaling, and high performance.
- **Types:**
  - Document: MongoDB, Couchbase (JSON-like).
  - Key-Value: Redis, Memcached.
  - Graph: Neo4j.
  - Column-Family: Cassandra.
- **Use Cases:** IoT, real-time analytics, social networks, unstructured or dynamic data.
- **Pros:** Scalable, flexible schemas, high throughput.
- **Cons:** Weaker consistency, complex joins, less strict ACID adherence.

Comparison Summary

Feature	RDBMS	NoSQL	MongoDB	Mongoose
Structure	Tables/Relations	Schema-less	JSON-like Docs	Node.js + MongoDB Bridge
Scalability	Vertical (mostly)	Horizontal	Horizontal	N/A
Use Cases	Structured data	Unstructured/Flexible	Web apps, analytics	Node.js apps with MongoDB

3. MongoDB (A Document Database)

- **Concept:**
  - Stores data as JSON-like documents (BSON) in collections (analogous to tables).
  - Schema-less, which means documents in the same collection can have different structures.
  - Designed for scalability, high performance, and ease of use.
- **Key Features:**
  - **Documents:** Store data in JSON-like format.
  - **Collections:** Groups of similar documents (like tables in SQL).
  - **Flexible Schema:** No predefined schema is required.
  - **Scalability:** Supports horizontal scaling with sharding.
  - **Indexing:** Supports indexing to speed up query performance.
  - **Querying:** Uses MongoDB Query Language to query documents.
- **Use Cases:**
  - Web applications, content management, mobile apps, real-time analytics.
  - When schema flexibility is required.
- **Pros:**
  - Easy to get started.
  - Flexible schemas.

    - Scalable.
    - Good performance with JSON data.
- **Cons:**
    - Less robust in enforcing data integrity compared to RDBMS.
    - Can be challenging to perform complex multi-document joins (but supports $lookup operator)

4. MongoDB Community Edition (Data Storing)

- **Concept:**
    - The free and open-source version of MongoDB.
    - Provides all core functionalities for data storage and retrieval.
    - Suitable for learning, development, and small-to-medium-sized applications.
- **Key Features:**
    - All core MongoDB data storage capabilities, replication, basic security, indexing.
    - Data is stored in JSON-like BSON format.
- **Use Cases:**
    - Learning MongoDB.
    - Developing prototype applications.
    - Deploying small to medium-sized applications that don't require advanced features (eg. advanced security, enterprise support)

5. MongoDB Compass (Visuals)

- **Concept:**
  - MongoDB's GUI tool to visualize your MongoDB data.
  - Provides an interface for exploring data, running queries, and managing the database.
- **Key Features:**
  - Graphical representation of your collections, documents, indexes, and aggregations.
  - Visual query builder.
  - Schema exploration.
  - Data import and export.
- **Use Cases:**
  - Exploring your MongoDB data.
  - Running queries without needing to use a command line.
  - Debugging data issues.

6. Mongoose (for Node.js and MongoDB connection)

- **Concept:**
  - A Node.js library that makes it easier to interact with MongoDB.
  - Provides a high-level object modeling API to interact with MongoDB database
  - Aids in data validation, schema definition, and data manipulation.
- **Key Features:**

    - **Object Data Modeling:** Maps your JavaScript objects to MongoDB documents.
    - **Schema Definitions:** Allows you to define a schema for your data.
    - **Validation:** Validates data before it gets stored in the database.
    - **Querying:** Simplifies running queries against MongoDB.
    - **Middleware and Hooks:** Allows you to define custom hooks that run before or after database operations.
- **Use Cases:**
    - Building Node.js applications that use MongoDB.
    - Defining models and performing CRUD operations on documents.

7. Schema (in Mongoose)

- **Concept:**
    - Defines the structure and types of data within a MongoDB document.
    - Specifies the data types, required fields, and default values.
    - Acts as a guide for how your data will be structured in your database and used in your node app
    - Provides type checking and validation on the document values.
- **Key Features:**
    - Defines document fields and data types (e.g. String, Number, Boolean, Date, ObjectId, Array, and Nested objects)
    - Enforces data validation rules (e.g., required fields, min/max values, enum types).
    - Defines custom validators.
    - Implements indexes, and unique constraints
- **Use Cases:**
    - Defining the structure of a new collection.
    - Validating data before it's saved to the database.

8. Model (in Mongoose)

- **Concept:**
  - A Mongoose class that is used to perform CRUD operations on database documents using a specific schema.
  - Provides an interface for your application to interact with the database and create, read, update and delete data records
- **Key Features:**
  - Instance methods: allows you to add methods to your documents
  - Static methods: allows you to add methods to your model class
  - Querying: uses different method calls (`find()`, `findById()`, `findOne()`, etc) to fetch data
  - Saving documents to the db, using `save()`
- **Use Cases:**
  - Creating instances of your documents
  - Querying and retrieving documents
  - Updating document values and saving them to the database
  - Deleting documents

In Summary:

- Relational databases are best for structured data with well-defined relationships, while non-relational databases are better for unstructured or rapidly changing data.
- MongoDB is a popular NoSQL document database that is easy to use and scalable.
- Mongoose is a library that simplifies working with MongoDB in Node.js, offering schemas and models.
- MongoDB Compass is a GUI tool that provides a visual way to manage and interact with your data.