Source: [Beginners Guide: MongoDB Basics | MongoDB]

- general-purpose document database designed for modern application development and for the cloud
- stores data as JSON documents
    - ability to express hierarchical relationships and to store structures as arrays
- collections: a group of documents; kind of like a table but more flexible in that they don't enforce schemas, documents in the same collection can have different fields
    - each collection is associated with one MongoDB database
- sharding: distributing data intelligently across multiple machines
- supported languages: Node.js, C, C++, C#, Go, Java, Perl, PHP, Python, Ruby, Rust, Scala, and Swift
- creating a database (via MongoDB shell)
    - **show** dbs → shows/lists all databases in the cluster
    - to create a new database, first need to switch the context to a non-existing database using the use command

      use newdb

      ** but note that database is only created once you store data in it
    - **show** collections → shows/lists all the collections in the db you're in
    - to add a document, use db.collection.insert()

      db.user.**insert**({name: "Ada Lovelace", age: 205})

      ** when we used the **use** command, newdb becomes the current db we're working in
    - **db** command displays the name of the current database

Source: [MongoDB: An introduction - GeeksforGeeks]

- non-relational open source database
- related terms to relational DBMS
    - **collections** are like tables
    - **documents** are like rows
    - **fields** are like columns
    - **default** '_id' is like the primary key

Source: [MongoDB Tutorial (w3schools.com)]

- **document** example

```
{
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
}
```

- searching for all documents with the category field “News”
  - db.posts.**find**({category: "News"})
- show all available databases
  - **show dbs**
- create collections
  - **db.createCollection("posts")**
  - **db.posts.insertOne(object)**

    → object is a valid JavaScript object containing post data
- inserting data
  - **a single document**

```
db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

- **multiple documents**

```
db.posts.insertMany([
  {
    title: "Post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 3",
    body: "Body of post.",
    category: "Technology",
    likes: 3,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 4",
    body: "Body of post.",
    category: "Event",
    likes: 4,
    tags: ["news", "events"],
    date: Date()
  }
])
```

** inserts an array of objects into the database.

- finding data
  - db.posts.**find**() → accepts a query object; if left empty, all documents will be returned
    - examples:

      db.posts.find({category: "News"})

      db.posts.find({}, {title: 1, date: 1}) → only displays title and date fields in result… 1 means include field, 0 means exclude field
  - db.posts.**findOne**() → to select only one document; returns first match
- **update documents**

- db.posts.updateOne({ title: "Post Title 1" }, { $set: { likes: 2 } })

  → updates the “likes” on the post to 2

  ** to check document, just do: db.posts.find({ title: "Post Title 1" })
- db.posts.updateMany({}, { $inc: { likes: 1 } })

  → updates the “likes” on all documents by incrementing by 1

- **delete**
  - db.posts.deleteOne({ title: "Post Title 5" })

    → deletes the first document that matches the query
  - db.posts.deleteMany({ category: "Technology" })

    → deletes all documents that match the query