Lesson: A Document Database

What you'll learn

- The difference between the **document model** and the relational (table) model.
- When a NoSQL document store is a good fit — and when it isn't.
- How to install MongoDB and connect with `mongosh`.
- Basic CRUD (Create, Read, Update, Delete) operations.
- Vocabulary you'll see in app stacks: database, collection, document.

By the end you can recognise a MongoDB-backed app, connect to it, and run basic reads and writes.

The lesson

1. Document model vs relational

In a relational database you store rows in fixed tables with a declared schema. In a **document database** like **MongoDB**, you store **documents** — JSON-like objects — grouped into **collections**. There is no rigid table shape; each document can have its own fields.

The vocabulary maps roughly like this:

```
   Relational (SQL)            MongoDB (document)
   ----------------            ------------------
   database          <-->      database
   table             <-->      collection
   row               <-->      document   (a JSON-like object)
   column            <-->      field
   JOIN across tables <-->     often: embed related data in one document
```

A user document might look like:

```
{
  "_id": "...",
  "name": "Layla",
```

```
  "email": "layla@example.com",
  "roles": ["editor", "admin"],
  "address": { "city": "Riyadh", "zip": "11564" }
}
```

Notice `roles` is an array and `address` is a nested object — that's all one document. In a relational DB this might be three tables joined together. MongoDB stores it internally as **BSON** (Binary JSON). Every document gets a unique `_id` automatically.

2. When does a document store fit?

A document database fits well when:

- Your data is naturally **object-shaped / nested** (a product with variant lists, a CMS page with mixed blocks).
- The **schema evolves often** or varies between records — flexible fields with no migrations.
- You usually read/write a **whole object at once** by its id, rather than joining many tables.
- You want easy **horizontal scaling** (sharding across machines).

It fits poorly when:

- You need **multi-table transactions** and strict relational integrity (banking-style).
- Your queries are **heavily relational** (lots of JOINs, aggregate reporting across entities).
- You value an **enforced schema** to keep data clean.

Rule of thumb: relational for structured, interrelated, transaction-heavy data; document for flexible, self-contained, object-shaped data. Many real systems use both. In our lab, MongoDB and Redis appear inside app stacks (for example, a CMS storing pages as documents).

3. Install MongoDB

MongoDB ships from its own repository. On Ubuntu (steps; versions change, check the official docs):

```
# add MongoDB's GPG key + apt repo (per official install guide), then:
sudo apt update
sudo apt install -y mongodb-org
sudo systemctl enable --now mongod
systemctl status mongod        # expect "active (running)"
```

The server daemon is `mongod`. Its config is `/etc/mongod.conf` (where the data path, port `27017`, and `bindIp` live). By default MongoDB listens only on localhost; to accept lab connections an admin

widens `bindIp` and enables authentication.

4. Connect with mongosh

`mongosh` is the modern MongoDB shell — and it's a full JavaScript environment.

```
mongosh                                    # connect to localhost:27017
mongosh "mongodb://10.100.100.20:27017"    # connect to a remote server
```

Once inside:

```
show dbs                 // list databases
use appdb                // switch to (and create on first write) appdb
db                       // show current database name
show collections         // list collections in this database
```

A database and a collection spring into existence the first time you write to them — no `CREATE TABLE` step.

5. CRUD — Create

Insert documents into a collection (`users` here):

```
use appdb

db.users.insertOne({ name: "Layla", email: "layla@example.com", roles: ["editor"] })

db.users.insertMany([
  { name: "Omar", email: "omar@example.com", age: 30 },
  { name: "Sara", email: "sara@example.com", age: 27 }
])
```

Each insert returns the generated `_id`. Note Omar and Sara have an `age` field that Layla doesn't — that's fine, documents need not match.

6. CRUD — Read

Query with `find` (returns many) or `findOne` (returns one). The argument is a **filter** document:

```
db.users.find()                          // all documents
db.users.find({ name: "Omar" })          // exact match
```

```
db.users.find({ age: { $gt: 28 } })          // age greater than 28
db.users.findOne({ email: "sara@example.com" })

db.users.find({ age: { $gte: 27 } }).sort({ age: -1 }).limit(2)   // filter+sort+limit
db.users.countDocuments({ roles: "editor" })
```

Operators start with `$`: `$gt` (greater than), `$gte`, `$lt`, `$in`, etc. `find()` returns a cursor; `.sort()`, `.limit()`, and `.skip()` chain onto it.

7. CRUD — Update and Delete

Update needs a filter **and** an update operator like `$set`:

```
db.users.updateOne(
  { name: "Omar" },
  { $set: { age: 31, city: "Jeddah" } }     // sets fields; adds city if absent
)

db.users.updateMany(
  { roles: "editor" },
  { $set: { active: true } }
)
```

`$set` adds or changes fields. `$inc` increments a number; `$push` appends to an array. Delete:

```
db.users.deleteOne({ name: "Sara" })
db.users.deleteMany({ active: false })
```

Be careful: `find({})`, `updateMany({}, ...)`, and `deleteMany({})` with an empty filter touch **every** document. Always double-check your filter before an update or delete.

8. Operating notes

- Service: `sudo systemctl status mongod`, logs in `/var/log/mongodb/mongod.log`.
- Default port `27017`; in production/lab, **enable authentication** and create an app user (least privilege, one db per app — same rule as the SQL lesson).
- Back up with `mongodump`, restore with `mongorestore`.

Dig deeper

- MongoDB Manual (official docs)
- MongoDB CRUD Operations guide
- mongosh (MongoDB Shell) documentation
- DigitalOcean: How To Install MongoDB on Ubuntu
- DigitalOcean: SQL vs NoSQL databases explained

Search terms

- `mongodb document model vs relational explained`
- `when to use nosql vs sql database`
- `mongosh crud tutorial insertOne find updateOne`
- `install mongodb ubuntu official guide`
- `mongodb query operators $gt $set $push`

Check yourself

1. What MongoDB concept corresponds to a relational *table*, and what corresponds to a *row*?
2. Give one situation where a document database fits well and one where a relational database is the better choice.
3. What command switches to (and lazily creates) a database in `mongosh`?
4. Write a query that finds all users whose `age` is greater than 28.
5. Why must an update use an operator like `$set`, and what is the danger of `deleteMany({})`?

Revision #3
Created 2026-05-19 18:00:00 UTC by Eslam Shapsough
Updated 2026-05-19 18:04:00 UTC by Eslam Shapsough