ScopeTreasure Courses

MONGODB COURSES · LESSON MG.1 · CODE SCOPE SERIES

MongoDB Kya Hai? — NoSQL Ki Duniya

What MongoDB is and why apps love it: databases, collections and flexible documents — and the SQL vs NoSQL comparison, told from the MongoDB side.

Introduction

In the SQL series you met the register: neat tables, fixed columns, every row the same shape. That world is powerful — but not every app's data fits a register. A shopping app stores a shirt (needs sizes and colours), a book (needs author and pages) and a phone (needs storage and RAM) in the *same* product list. Force all three into one fixed table and half the columns stay empty for every row.

MongoDB is the most famous answer to that problem: a database that stores **documents** — flexible records that look like JSON — instead of rigid rows. In this first lesson you will meet its three words — **database, collection, document** — read a real document, and settle the famous question from the MongoDB side: **SQL or NoSQL — which one, and when?**

(Roman Urdu: SQL register hai — saaf, sakht. MongoDB file-folder hai — har record apni marzi ki shape ka.)

Key points

- **MongoDB is a NoSQL database:** "NoSQL" means *not only SQL* — data is stored as documents, not tables. MongoDB is the most used document database in the world, and it is free to start (Atlas cloud, MG.2).
- **Three words, one ladder:** a **database** holds **collections**; a collection holds **documents**. Compare with SQL: database → table → row. A document is a record written like a JSON object — field names travel *with* the data.
- **Every document has an `_id`:** MongoDB gives each document a unique `ObjectId` at birth — its identity card. You never have to invent roll numbers again (but you may set your own `_id` if you truly need to).
- **Flexible shape is the superpower:** one document can carry a field its neighbour does not have. No empty columns, no ALTER TABLE ceremony — the shape can grow as the app grows.
- **SQL vs NoSQL — the shape:** SQL stores **fixed-shape tables** (every row, same columns, decided in advance). MongoDB stores **flexible documents** (each record carries exactly what it needs).
- **SQL vs NoSQL — relations:** SQL is built for **connected data** answered with JOINs at question time. MongoDB prefers each document to carry its own world inside itself (embedding), joining only when it must.
- **SQL vs NoSQL — when each wins:** choose **SQL** when the shape is clear and the data is deeply connected — school records, shops, banks, bookings. Choose **MongoDB** when the shape keeps changing or differs per record — product catalogues, social feeds, app content, logs. Most developers learn both; this channel teaches both, and MG.6 is the full decider.

  (Roman Urdu: Saaf shape aur judaa hua data = SQL; badalti shape = MongoDB. Dono seekhna hi asal skill hai.)

- **You talk to MongoDB with mongosh:** the MongoDB Shell — a small program where you type commands like `db.students.find()` and watch documents come back. MG.2 sets it up; from MG.3 it is our daily tool.

Code examples

Example 1 — Meet a document: one student, the MongoDB way

```
db.students.findOne({ name: 'Ali Raza' })
```

What it does: `db.students` is the collection (think: the register's section), and `findOne` fetches the first document matching the filter. In the SQL world this same student was a row of bare values; here every value carries its field name with it.

What you will see:

```
{
  _id: ObjectId('650000000000000000000001'),
  name: 'Ali Raza',
  class: '10-A',
  marks: 88,
  city: 'Sahiwal'
}
```

Read it like a file card: `_id` is the identity, then name, class, marks, city — the same five facts the SQL `students` table held, but self-labelled. Show this card to any programmer on Earth and they understand it without seeing a table design first. That readability is half of MongoDB's charm.

Example 2 — Flexible shape: two products, two shapes

```
db.products.find()
```

What it does: One collection, two very different products. The shirt carries `sizes`; the book carries `author`. Neither wastes a field on the other's facts — no NULL-filled empty columns anywhere.

What you will see:

```
[
  {
    _id: ObjectId('650000000000000000000101'),
    name: 'Cotton Kurta',
    price: 2500,
    sizes: ['S', 'M', 'L'],
    colours: ['white', 'black']
  },
  {
    _id: ObjectId('650000000000000000000102'),
    name: 'JavaScript Ki Kitaab',
    price: 1200,
    author: 'A. Rahman',
    pages: 340
  }
]
```

Notice the **array inside the document** — `sizes` holds three values in one field. In SQL that list would need a second table and a JOIN; here it simply lives where it belongs. Flexible shape is not disorder — each document is still neat; it is just allowed to be *its own* neat.

Example 3 — Grow one document, touch nothing else

```
db.students.updateOne(
  { name: 'Hira Shah' },
  { $set: { scholarship: true } }
)
db.students.findOne({ name: 'Hira Shah' })
```

What it does: Hira won a scholarship, so *her* document gains a brand-new field. No other student is touched, no table is redesigned, and no other document is forced to carry an empty `scholarship` column. The second command reads her card back to prove it.

What you will see:

```
{ acknowledged: true, matchedCount: 1, modifiedCount: 1, upsertedId: null, upsertedCount: 0 }
{
  _id: ObjectId('650000000000000000000004'),
  name: 'Hira Shah',
  class: '10-B',
  marks: 95,
  city: 'Sahiwal',
  scholarship: true
}
```

The acknowledgement line is MongoDB confirming: one document matched, one changed. And Hira's card now has six fields while Ali's still has five — in MongoDB that is not a problem to fix; it is the design working as intended.

Quick recap

- MongoDB = a **NoSQL document database**: database → collection → document (SQL's cousin of database → table → row).
- A document is a JSON-like record whose field names travel with its values; every document is born with a unique `_id` (ObjectId).
- Flexible shape: documents in one collection may carry different fields; arrays and nested data live inside the document.
- **SQL vs NoSQL:** fixed connected tables vs flexible self-carrying documents. Clear connected shape → SQL; changing or mixed shape → MongoDB. MG.6 settles it fully.

Common mistakes

- **Thinking NoSQL means "no structure":** documents are structured — field names, types, nesting. The structure is simply per-record instead of per-table. Sloppy documents are still sloppy data.
- **Calling a collection a table and a document a row in your head — then designing like SQL:** if you split every fact into its own collection and plan to join everything later, you have rebuilt SQL the hard way. MongoDB design starts from *what the app reads together*.

  (Roman Urdu: MongoDB ko SQL ki tarah design karna sab se aam ghalti hai — pehle socho, app ikatthe kya parhta hai.)
- **Fearing the ObjectId:** those long `_id` values look scary but they are just identity cards — guaranteed unique, created for free. You will filter by normal fields (`name`, `city`) in daily work.
- **Choosing a database by fashion:** "everyone uses MongoDB" is not a reason. Shape and connections choose the database — MG.6 gives you the three questions that decide.

Practice

1. Write, on paper, the document for a student of your own invention — name, class, marks, city — plus one field the other students do not have (a hobby, a bus route, anything). Check: could a SQL table hold your extra field without changing? What would it cost?
2. A tailor shop wants to store suits (need measurements), ready-made shirts (need sizes) and fabrics (need metres and material) in one products list. SQL or MongoDB — and why, in two lines?
3. Run `db.students.findOne({ name: 'Sara Khan' })` in your head against this lesson's data: write the exact document you expect, field by field, before MG.2 shows you the real shell.

Practice quiz

1. What is MongoDB's ladder, from biggest to smallest?

- Table, then row, then column
- Database holds collections, a collection holds documents; a document is a JSON-like record whose field names travel with the data
- Collection, then database, then document
- Document, then collection, then database

Answer: Database holds collections, a collection holds documents; a document is a JSON-like record whose field names travel with the data

2. What does every MongoDB document get at birth?

- A roll number you must invent
- A unique _id, an ObjectId, its identity card
- A table name
- Nothing, identity is optional

Answer: A unique _id, an ObjectId, its identity card

3. What is MongoDB's flexible-shape superpower?

- No structure at all
- One document can carry a field its neighbour does not have, with no empty columns and no ALTER TABLE ceremony
- Every document must have identical fields
- Documents cannot hold arrays

Answer: One document can carry a field its neighbour does not have, with no empty columns and no ALTER TABLE ceremony

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/mongodb-intro · More free tools and notes at https://scopetreasure.com