ScopeTreasure Courses

MONGODB COURSES · LESSON MG.3 · CODE SCOPE SERIES

MongoDB CRUD

The four jobs every app does with data: insert, find, update and delete documents — with the safety habits that keep a live database safe.

Introduction

Every app you will ever build does four jobs with its data: **Create, Read, Update, Delete** — CRUD. A school app creates a student, reads the class list, updates a mark, deletes a leaver. In MG.2 the Create half arrived (`insertOne`, `insertMany`). This lesson completes the set in mongosh: reading with aim (`find`, `findOne`), changing documents safely (`updateOne` with `$set`), and removing them (`deleteOne`) — with the same safety habits the SQL series taught, translated to documents.

(Roman Urdu: CRUD har app ki chaar naukriyaan hain — banana, parhna, badalna, mitana. MongoDB mein ye chaaron documents par hote hain.)

Key points

- **find is the Read:** `db.students.find()` returns every document; `find({ city: 'Sahiwal' })` returns only matches — the filter is a small document describing *what you want*. `findOne` returns the first match only (or `null` — no match is an answer too).
- **A projection trims the answer:** `find({}, { name: 1, marks: 1, _id: 0 })` means *give me only name and marks, hide the _id*. `1` = show, `0` = hide. Apps use projections constantly — fetch the fields you need, not the whole card.
- **updateOne + $set is the safe change:** `updateOne(filter, { $set: { marks: 55 } })` — the filter picks the document, `$set` names the fields to change. Other fields are untouched. `$inc` is its cousin: `{ $inc: { marks: 5 } }` adds to a number instead of replacing it.
- **The reply tells the truth:** every update answers with `matchedCount` and `modifiedCount`. Matched 0 means your filter found nobody — a silent spelling mistake in a name looks exactly like this, so *read the counts*.
- **deleteOne / deleteMany remove documents:** `deleteOne(filter)` removes the first match; `deleteMany(filter)` removes them all — and `deleteMany({})` with an empty filter removes **everything**. The reply's `deletedCount` is your receipt.
- **The safety habit is identical to SQL's:** before an UPDATE or DELETE, run the same filter as a `find` (or `countDocuments`) and look at exactly who is about to be touched. SELECT-then-UPDATE in SQL; find-then-update here.

Code examples

Example 1 — Read with aim: filter + projection

```
db.students.find({ city: 'Sahiwal' })
db.students.find({}, { name: 1, marks: 1, _id: 0 })
```

What it does: The first command asks for Sahiwal's students only. The second asks for the whole class but trims every answer to two fields — a class list the way a teacher actually wants it: names and marks, no ObjectIds, no noise.

What you will see:

```
[
  {
    _id: ObjectId('650000000000000000000001'),
    name: 'Ali Raza',
    class: '10-A',
    marks: 88,
    city: 'Sahiwal'
  },
  {
    _id: ObjectId('650000000000000000000004'),
    name: 'Hira Shah',
    class: '10-B',
    marks: 95,
    city: 'Sahiwal',
    scholarship: true
  }
]
[ { name: 'Ali Raza', marks: 88 }, { name: 'Sara Khan', marks: 92 }, { name: 'Bilal Ahmed',
marks: 76 }, { name: 'Hira Shah', marks: 95 }, { name: 'Usman Tariq', marks: 64 } ]
```

Two answers, two shapes. Notice Hira's `scholarship` field from MG.1 travelled along in the full-document answer — and vanished politely in the projected one, which shows only what was asked. The filter chose *which documents*; the projection chose *which fields*.

Example 2 — Update safely: $set and $inc on a test document

```
db.students.insertOne({ name: 'Test Wala', class: '9-B', marks: 50, city: 'Okara' })
db.students.updateOne({ name: 'Test Wala' }, { $set: { marks: 55 } })
db.students.updateOne({ name: 'Test Wala' }, { $inc: { marks: 5 } })
db.students.findOne({ name: 'Test Wala' })
```

What it does: A throwaway sixth student is created so dangerous commands can be practised honestly. `$set` *replaces* his marks with 55; `$inc` then *adds* 5 — 55 + 5 = 60. The final `findOne` is the proof read: always read back after a change.

What you will see:

```
{ acknowledged: true, insertedId: ObjectId('650000000000000000000006') }
{ acknowledged: true, matchedCount: 1, modifiedCount: 1, upsertedId: null, upsertedCount: 0 }
{ acknowledged: true, matchedCount: 1, modifiedCount: 1, upsertedId: null, upsertedCount: 0 }
{
  _id: ObjectId('650000000000000000000006'),
  name: 'Test Wala',
  class: '9-B',
  marks: 60,
  city: 'Okara'
}
```

Both updates report matched 1, modified 1 — exactly one document changed, twice, and the read-back shows marks 60 with every other field untouched. This is the rhythm of safe updates: **filter → update → read the counts → read the document.**

Example 3 — Delete safely: count first, delete once

```
db.students.countDocuments({ name: 'Test Wala' })
db.students.deleteOne({ name: 'Test Wala' })
db.students.countDocuments()
```

What it does: The count is the safety SELECT — it proves the filter touches exactly one document *before* anything is destroyed. Then `deleteOne` removes him, and the full count confirms the class is back to its original five.

What you will see:

```
1
{ acknowledged: true, deletedCount: 1 }
5
```

Three small numbers, one complete safety story: 1 in danger, 1 deleted, 5 remain. The real students were never touched — every risky command in this lesson was practised on a document created to be sacrificed.

Quick recap

- **CRUD** = insert (create), find (read), update (change), delete (remove) — on documents.
- `find(filter)` reads many, `findOne(filter)` reads the first; a **projection** (`{ name: 1, _id: 0 }`) trims fields.

- `updateOne(filter, { $set: {...} })` changes named fields; `$inc` adds to numbers; **matchedCount / modifiedCount** are your proof.
- `deleteOne` removes one, `deleteMany` removes all matches — and the safety habit never changes: **count with the filter first, then strike.**

Common mistakes

- **Updating without $set:** `updateOne({ name: 'Ali' }, { marks: 55 })` — modern mongosh refuses with *"Update document requires atomic operators"* (older tools would have replaced the whole document with just `{ marks: 55 }` — name, class, city gone). The `$set` wrapper is what makes an update a *change* instead of a *replacement*.
- **The empty-filter disaster:** `deleteMany({})` deletes the whole collection; `updateOne({}, ...)` touches the first document it finds. An empty filter is never "no filter" — it is "every filter". Type your filters completely, then count.

  (Roman Urdu: Khaali filter {} ka matlab "koi nahi" nahi — "sab" hai. Pehle count, phir delete.)
- **Ignoring matchedCount 0:** you filter `{ name: 'Ali Raza ' }` (a trailing space) and the update "succeeds" — matched 0, modified 0, nothing changed, no error shown. The counts are the only witness; read them every time.
- **Expecting find to return one document:** `find` always returns a list (possibly of one, possibly empty). Code that treats the list as a single card breaks quietly — use `findOne` when you want one document.

Practice

1. Using a test document of your own (never a real student), practise the full cycle: insert, `$set` a new city, `$inc` the marks by 10, read back, count, delete — and write down every count MongoDB reported.
2. Write the projection that shows only `name` and `city` for the whole class, *with* the `_id` hidden. How many fields does each answer document have?
3. A filter is written as `{ ctiy: 'Lahore' }` (a typo in the field name). What does `find` return, what does `updateOne` report, and why does MongoDB not call it an error? (This is the flexible shape's dark side — remember it in MG.5, where schemas fight back.)

Practice quiz

1. What is the difference between find and findOne?

- There is none
- find returns every matching document as a list, findOne returns the first match only, or null when there is no match
- find returns one, findOne returns many
- find works only with filters

Answer: find returns every matching document as a list, findOne returns the first match only, or null when there is no match

2. What does the projection { name: 1, marks: 1, _id: 0 } mean?

- Give me only _id
- Give me only name and marks, and hide the _id; 1 means show, 0 means hide
- Delete those fields
- Sort by name and marks

Answer: Give me only name and marks, and hide the _id; 1 means show, 0 means hide

3. What does deleteMany({}) do, and what must you do first?

- It deletes nothing, an empty filter means no filter
- It removes everything in the collection, because an empty filter matches all; run the same filter as a find or countDocuments first and look at exactly who is about to be touched
- It deletes only the first document
- It is an error, empty filters are rejected

Answer: It removes everything in the collection, because an empty filter matches all; run the same filter as a find or countDocuments first and look at exactly who is about to be touched

---

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-crud · More free tools and notes at https://scopetreasure.com