ScopeTreasure Courses

MONGODB COURSES · LESSON MG.10 · CODE SCOPE SERIES

MongoDB Indexes + Talaash (Search)

Why big collections answer slowly and how an index fixes it — plus unique indexes that refuse duplicates and text search over your posts.

Introduction

Five students answer instantly. Five *lakh* students answer eventually. The gap between those two sentences is the **index**: MongoDB without an index reads every document to answer a filter — a **collection scan** — while an index is a sorted lookup table, like the index at the back of a book, that jumps straight to the right pages. This lesson proves the difference with `explain` (the database's own report card — DB.9's SQL cousin), adds a **unique index** that refuses duplicate emails, and finishes with **text search** over blog posts.

(Roman Urdu: Index = kitaab ke aakhir wali fehrist. Bina fehrist, har sawaal par poori kitaab parhni parti hai.)

Key points

- **No index = COLLSCAN:** for `find({ city: 'Sahiwal' })`, MongoDB opens every document, checks it, moves on. Fine at five documents, fatal at five lakh. The tell-tale number is **docsExamined** — how many documents were touched to answer.
- **An index is a sorted side-table:** `createIndex({ city: 1 })` builds (and forever maintains) a sorted list of city values, each pointing at its document. Lookups become IXSCAN — read the index, fetch only the matches. docsExamined falls to roughly the number of *answers*.
- **explain('executionStats') is the proof, not a feeling:** it reports the winning plan's stage (`COLLSCAN` vs `IXSCAN`), `nReturned` and `totalDocsExamined`. "The query feels slow" is a guess; explain is a measurement. Read it before and after every index you add.
- **Indexes are not free:** every insert and update must also update every index — writes get a little slower and storage grows. Index the fields your app actually filters and sorts by; indexing everything "just in case" taxes every write to speed up queries nobody runs.
- **A unique index is a bouncer:** `createIndex({ email: 1 }, { unique: true })` rejects any insert whose email already exists, with the famous **E11000 duplicate key** error. Uniqueness enforced by the *database* cannot be raced or forgotten by app code — MG.5's validation, at the storage layer.
- **Text indexes power real search:** `createIndex({ title: 'text', body: 'text' })` lets `$text: { $search: 'mongodb' }` find words inside fields — word-based, case-insensitive, rankable by a relevance score. One text index per collection; it is the honest built-in search for small apps.

Code examples

Example 1 — The proof: explain before and after an index

```
db.students.find({ city: 'Sahiwal' }).explain('executionStats')
db.students.createIndex({ city: 1 })
db.students.find({ city: 'Sahiwal' }).explain('executionStats')
```

What it does: The same query, measured twice. First MongoDB has no index on `city`, then we build one (`createIndex` answers with the new index's name, `city_1`), and the plan is measured again. (explain's full report is long — the excerpts below keep the lines that matter.)

What you will see:

```
// BEFORE — no index
queryPlanner: { winningPlan: { stage: 'COLLSCAN' } }
executionStats: { nReturned: 2, totalDocsExamined: 5, totalKeysExamined: 0 }

city_1

// AFTER — index on city
queryPlanner: { winningPlan: { stage: 'FETCH', inputStage: { stage: 'IXSCAN', indexName:
'city_1' } } }
executionStats: { nReturned: 2, totalDocsExamined: 2, totalKeysExamined: 2 }
```

Read the numbers, not the vibes: **5 examined → 2 examined**, and the stage changed from COLLSCAN (read everything) to IXSCAN (index lookup) + FETCH (collect the two matches). At five students this saves nothing you can feel; at five lakh it is the difference between a blink and a coffee break. The rule of the lesson: *nReturned close to totalDocsExamined* means the index is doing its job.

Example 2 — The unique index refuses a duplicate

```
db.users.insertMany([
  { name: 'Ali Raza', email: 'ali@example.com' },
  { name: 'Sara Khan', email: 'sara@example.com' }
])
db.users.createIndex({ email: 1 }, { unique: true })
db.users.insertOne({ name: 'Ali Doosra', email: 'ali@example.com' })
```

What it does: Two honest users land first. Then the email field gets its unique index — and a third insert tries to reuse Ali's email, the classic double-account accident (or attack). The app never gets a vote; the database itself refuses.

What you will see:

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('650000000000000000000301'),
    '1': ObjectId('650000000000000000000302')
  }
}
email_1
MongoServerError: E11000 duplicate key error collection: schoolDB.users index: email_1 dup
key: { email: "ali@example.com" }
```

That red `E11000` is a *feature arriving loudly*: the duplicate never landed. Note where the guarantee lives — not in one careful route, but in the index itself, so no future code path (another route, a script, a sleepy 3 a.m. fix) can sneak a second account past it. Rule worth stealing: **anything that must be unique gets a unique index; app-level checks are just politeness.**

Example 3 — Text search: talaash over blog posts

```
db.posts.insertMany([
  { title: 'MongoDB Seekhna Asaan Hai', body: 'Documents aur collections ki pehli samajh.' },
  { title: 'Meri Pehli Website', body: 'HTML, CSS aur JavaScript se bani meri pehli koshish.'
},
  { title: 'MongoDB vs SQL — Mera Faisla', body: 'Do hafte dono istemal karke ye faisla kiya.'
}
])
db.posts.createIndex({ title: 'text', body: 'text' })
db.posts.find({ $text: { $search: 'mongodb' } }, { title: 1, _id: 0 })
```

What it does: Three posts exist (the blog of MG.11 is warming up). The text index covers title and body — MongoDB answers with its name, `title_text_body_text`. Then `$text` searches the *word* mongodb across both fields, and a projection keeps the answer to titles only.

What you will see:

```
title_text_body_text
[
  { title: 'MongoDB Seekhna Asaan Hai' },
  { title: 'MongoDB vs SQL — Mera Faisla' }
]
```

Two of three posts — the website post, innocent of the word, stays home. The search is case-insensitive ('MongoDB' in the titles matched the small 'mongodb' query) and word-based. For a course blog or a dukaan's product names this honest built-in search carries a small app a long way before any external search engine earns its rent.

Quick recap

- No index → **COLLSCAN**: every document examined. Index → **IXSCAN**: jump, fetch matches only.
- `explain('executionStats')` is the measurement: compare `totalDocsExamined` with `nReturned`, before and after.
- `createIndex({ field: 1 })` builds a maintained sorted lookup; indexes speed reads and tax writes — index what you query.
- `{ unique: true }` makes the *database* refuse duplicates (**E11000**). A **text index** + `$text` = built-in word search.

Common mistakes

- **Indexing by hope:** adding indexes for fields no query filters on. Each one slows every insert and update forever. The evidence-driven way: read your app's real queries (and explain's COLLSCANs), then index exactly those fields.
- **Trusting "it feels fast" on toy data:** five documents make every query instant, indexed or not. Slowness is a *scale* property — measure with explain's docsExamined ratio, which confesses at any size.
  (Roman Urdu: Chhote data par har cheez tez lagti hai — explain ka docsExamined hi sach batata hai.)
- **Catching E11000 and showing it raw to users:** the duplicate-key error is for *you*; the user deserves "this email is already registered" in human words. Database errors are logs, not UI.
- **A second text index:** one collection, one text index — a second `createIndex({... 'text'})` errors. Need more fields covered? Drop the old text index and build one covering them all (`db.posts.dropIndex('title_text_body_text')` first).

Practice

1. On students: run explain for `find({ class: '10-B' })` before any index on `class` exists. Write the stage and docsExamined. Create the index, re-run, and record both numbers again — did `city_1` from Example 1 help this different query at all? Why not?
2. Design the indexes for the dukaan's orders (MG.7): the owner filters by `status` daily and by `customer` when someone calls. Which single-field indexes do you create, and which popular aggregation stage from MG.7 does each one feed?
3. Extend Example 3: search for the word 'faisla' — predict the result before running. Then search 'seekhna asaan' (two words) and describe in one line how $text treats a multi-word search.

Practice quiz

1. Without an index, how does MongoDB answer find({ city: 'Sahiwal' })?

- With an IXSCAN jump
- With a COLLSCAN: it opens every document and checks it; the tell-tale number is docsExamined
- It refuses the query

- It returns only the first document

Answer: With a COLLSCAN: it opens every document and checks it; the tell-tale number is docsExamined

2. What does a unique index do, and what error names a duplicate?

- It speeds reads only
- createIndex({ email: 1 }, { unique: true }) rejects any insert whose email already exists, with the E11000 duplicate key error, enforced by the database itself
- It deletes duplicates silently
- It warns but still inserts

Answer: createIndex({ email: 1 }, { unique: true }) rejects any insert whose email already exists, with the E11000 duplicate key error, enforced by the database itself

3. How many text indexes may one collection have?

- As many as you like
- One; a second text index errors, so drop the old one and build one covering all the fields you need
- None, MongoDB has no text search
- Two, one per field group

Answer: One; a second text index errors, so drop the old one and build one covering all the fields you need

---

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