Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: MONG • **Updated:** 04 Nov 2026
https://www.mockattempt.com/courses/mongodb-training

MONG • NOSQL DATABASES & DISTRIBUTED DATA ENGINEERING | **BEGINNER TO PRODUCTION MONGODB & ATLAS ARCHITECT**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

MongoDB Training

Master MongoDB from BSON ObjectId internals, Schema Validation, Atomic CRUD Operators ($set, $inc, $push, $elemMatch) & ESR Compound Indexes to Aggregation Pipelines ($match, $group, $lookup, $facet), ACID Transactions, Replica Sets, Sharding & Atlas — with mongosh programs, 386 Topic Mock Tests (17,860 MCQs), and Spoken Voice + MockAttempt AI Interviews at ₹1,799 (Incl. 18% GST).

Duration: 15–17 Weeks • Live + Hands-On Projects (205 Hrs) • **Modules:** 16 • **Live Topics:** 53 • **Subtopics:** 424+ • **Labs & Tasks:** 106 • **Mocks / AI Vivas:** 386 / 386 • **Projects:** 16 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹1,799 (Net Tuition ₹1,525 + Included 18% GST ₹274) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹10/Guided Hr (~₹34/Live Topic)

Faculty: 100% Active Working IT Professionals • **Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots):** Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 17–19 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 19–21 Weeks Weekend Track)

✓ **Trainer LeetCode & HackerRank Mentorship (JavaScript & Node.js Driver): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: MongoDB 8.0 • WiredTiger & BSON • mongosh & Compass • ESR B-Tree Indexes • Aggregation Framework • Replica Sets & Sharding • MongoDB Atlas & Node Driver

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Model high-scale BSON documents choosing accurately between Embedding, Referencing, Bucket, and Outlier patterns within the 16 MB limit
2. Execute atomic updates ($set, $inc, $addToSet, $pull, findOneAndUpdate) and eliminate COLLSCAN bottlenecks using ESR Compound & Multikey Indexes
3. Build multi-stage Aggregation Pipelines ($match, $group, $lookup, $unwind, $facet), Multi-Document ACID Transactions, Replica Sets, and Sharded Clusters
4. Clear 386 Timed Mock Exams (17,500 MongoDB MCQs) and 350 Spoken Voice + MockAttempt AI NoSQL Technical Interviews
5. Access-pattern schema design: embedding vs referencing, bucket/outlier patterns, and `$jsonSchema` validation.
6. Crack LeetCode and HackerRank problems in JavaScript & Node.js Driver with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. High-speed multi-stage Aggregation Pipelines (`$match`, `$lookup`, `$unwind`, `$facet`, `$project`) replacing slow app loops.
8. ESR (Equality-Sort-Range) compound index design, TTL expiration indexes, and `explain("executionStats")` verification.

Eligibility & Prerequisites: Basic familiarity with JSON objects/arrays — starts from NoSQL & document database fundamentals from scratch • Any laptop/PC running MongoDB Community Server / MongoDB Atlas Free Tier, mongosh, and MongoDB Compass • No prior MongoDB industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (16 MODULES • 53 LIVE TOPICS • 424+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (16 Topics / 128 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (13 Topics / 104 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (12 Topics / 96 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (12 Topics / 96 Subtopics)

MODULE 1 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: MongoDB & NoSQL Architecture, WiredTiger Engine, BSON Types & RDBMS Comparison

Understand why document databases excel at polymorphic agile workloads, how WiredTiger manages document-level concurrency & compression, and how 12-byte BSON `ObjectId`s and `Decimal128` work under the hood.

Objectives: ✓ MongoDB Architecture — Document Model, WiredTiger Storage Engine & Document-Level Concurrency — Collections and BSON documents, how MongoDB's default **WiredTiger** engine uses document-level optimistic concurrency control, in-memory cache (~50% of RAM - 1 GB), Snappy/Zstd compression, and 60-second checkpoints + journal durability. • ✓ NoSQL Database Paradigms — Document, Key-Value, Wide-Column, Graph & CAP / PACELC Trade-Offs — Why web-scale systems outgrew rigid single-node tables, the four NoSQL families (Document: MongoDB, Key-Value: Redis, Wide-Column: Cassandra, Graph: Neo4j), CAP theorem under network partitions (`CP` vs `AP`), and BASE vs ACID. • ✓ MongoDB Advantages Over RDBMS — Zero-Downtime Polymorphism, Locality & Native Sharding — Eliminating ORM object-relational impedance mismatch, adding new fields without multi-hour `ALTER TABLE` locks, native Replica Set auto-failover, and built-in horizontal sharding across commodity nodes. • ✓ BSON Data Types — 12-Byte `ObjectId`, `Decimal128`, `ISODate`, `Int32`/`Long`, `BinData` & `16 MB` Cap — Why MongoDB stores Binary JSON (**BSON**) instead of plain text JSON (fast length-prefixed field skipping + rich types!), dissecting the 12-byte `ObjectId` (`4B Unix epoch seconds + 5B process random + 3B counter`), `NumberDecimal()`, and `ISODate()`. • ✓ Build, verify, and ship Module Capstone: Polymorphic Multi-Track Course Catalog BSON Blueprint (`ObjectId` + `Decimal128` + `ISODate`).

1.1 MongoDB Architecture — Document Model, WiredTiger Storage Engine & Document-Level Concurrency

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: WiredTiger storage engine & Document-level concurrency • Core Syntax, CLI & Production APIs: db.serverStatus().wiredTiger, db.stats(), mongod --dbpath • Deep-Dive Mechanics: Snappy & Zstd compression & Working Set fits in RAM • Advanced Patterns: Write-ahead journal checkpoints & 16 MB max BSON document size • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Inspecting Database Storage & WiredTiger Engine Stats in `mongosh` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for MongoDB Architecture

Purpose: To understand and apply MongoDB Architecture — Document Model, WiredTiger Storage Engine & Document-Level Concurrency in real MongoDB systems — specifically: Collections...

What Topic Is Next → Next Topic 1.2: NoSQL Database Paradigms — Document, Key-Value, Wide-Column, Graph & CAP / PACELC Trade-Offs

1.2 NoSQL Database Paradigms — Document, Key-Value, Wide-Column, Graph & CAP / PACELC Trade-Offs

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Document vs Key-Value vs Wide-Column vs Graph & CAP theorem CP behavior • Core Syntax, CLI & Production APIs: Document (MongoDB) vs Key-Value (Redis) vs Wide-Column (Cassandra) vs Graph (Neo4j) • Deep-Dive Mechanics: Data-that-is-accessed-together-stays-together & Horizontal scale-out sharding • Advanced Patterns: Flexible schema evolution & Polyglot persistence • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Co-Located Single-Document Course Aggregate Read (`0` Joins) • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for NoSQL Database Paradigms

Purpose: To understand and apply NoSQL Database Paradigms — Document, Key-Value, Wide-Column, Graph & CAP / PACELC Trade-Offs in real MongoDB systems — specifically: Why web-scale...

What Topic Is Next → Next Topic 1.3: MongoDB Advantages Over RDBMS — Zero-Downtime Polymorphism, Locality & Native Sharding

1.3 MongoDB Advantages Over RDBMS — Zero-Downtime Polymorphism, Locality & Native Sharding

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Zero ORM impedance mismatch & Single-document atomicity • Core Syntax, CLI & Production APIs: Polymorphic documents + Schema Versioning (`schemaVersion: 2`) + Native Replica Sets & Shard... • Deep-Dive Mechanics: Polymorphic attribute pattern & Online schema versioning • Advanced Patterns: Built-in Replica Set failover & Horizontal sharding • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Polymorphic Assessment Documents Sharing a Collection Cleanly • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for MongoDB Advantages Over RDBMS

Purpose: To understand and apply MongoDB Advantages Over RDBMS — Zero-Downtime Polymorphism, Locality & Native Sharding in real MongoDB systems — specifically: Eliminating ORM obj...

What Topic Is Next → Next Topic 1.4: BSON Data Types — 12-Byte `ObjectId`, `Decimal128`, `ISODate`, `Int32`/`Long`, `BinData` & `16 MB` Cap

1.4 BSON Data Types — 12-Byte `ObjectId`, `Decimal128`, `ISODate`, `Int32`/`Long`, `BinData` & `16 MB` Cap

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 12-byte ObjectId (4B time + 5B rand + 3B counter) & Oid.getTimestamp() extraction • Core Syntax, CLI & Production APIs: new ObjectId(), oid.getTimestamp(), NumberDecimal("999.00"), ISODate("2026-10-31T00:00:00Z")... • Deep-Dive Mechanics: NumberDecimal exact currency & ISODate UTC sorting & TTL • Advanced Patterns: Immutable _id primary key & Length-prefixed BSON traversal • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Inspecting 12-Byte `ObjectId` Hex Length & Embedded Timestamp • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for BSON Data Types

Purpose: To understand and apply BSON Data Types — 12-Byte `ObjectId`, `Decimal128`, `ISODate`, `Int32`/`Long`, `BinData` & `16 MB` Cap in real MongoDB systems — specifically: Why...

What Topic Is Next → Module 1 Capstone (Polymorphic Multi-Track Course Catalog BSON Blueprint (`ObjectId` + `Decimal128` + `ISODate`)) → Next Topic 2.1: MongoDB Shell (`mongosh`) & Compass — `mongodb+srv://` URIs, Scripting & Diagnostics

Module 1 Hands-On Capstone Project (100 Marks): Polymorphic Multi-Track Course Catalog BSON Blueprint (`ObjectId` + `Decimal128` + `ISODate`) — Design a polymorphic `courses` BSON collection storing diverse IT tracks (video courses, live cohorts, coding bootcamps) with shared core metadata, `NumberDecimal("999.00")` exact fees, `ISO...

MODULE 2 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: `mongosh` Shell, Collections, `$jsonSchema` Validation & Production Data Modeling

Navigate `mongosh` & Compass like a pro, enforce server-side `$jsonSchema` rules on collections, use Capped/Time-Series collections, and master Embedding vs Referencing design patterns.

Objectives: ✓ MongoDB Shell (`mongosh`) & Compass — `mongodb+srv://` URIs, Scripting & Diagnostics — Modern Node.js-powered `mongosh` (replacing legacy `mongo`), SRV DNS seedlist connection strings (`mongodb+srv://`), `.mongoshrc.js` helpers, Compass visual explain plans, and `db.currentOp()` / `db.killOp()`. • ✓ Creating & Dropping Databases — Lazy Materialization, `db.stats()` & Namespace Hygiene — Why `use academy_db` switches context in memory WITHOUT writing anything to disk until the first collection/document is created, reserved system databases (`admin`, `local`, `config`), and safeguarding `db.dropDatabase()`. • ✓ Collections — `createCollection()`, `$jsonSchema` Validation, Capped & Time-Series Collections — Implicit vs explicit `db.createCollection()`, enforcing server-side `$jsonSchema` (`bsonType`, `required`, `enum`, `minimum`), fixed-size circular **Capped Collections** (`capped: true, size, max`), and MongoDB Time-Series collections (`timeseries: { timeField, metaField }`). • ✓ MongoDB Data Modeling — Embedding vs Referencing, Subset/Bucket Patterns & Unbounded Arrays — 1-to-1 and 1-to-Few (**Embed**), 1-to-Many bounded vs 1-to-Squillions (**Child-to-Parent Reference**!), Subset Pattern (embed latest 10 reviews on product, store full history in `reviews` collection), and Extended Reference Pattern. • ✓ Build, verify, and ship Module Capstone: Validated Student Telemetry & Cohort Schema (`$jsonSchema` + Bucket Pattern).

2.1 MongoDB Shell (`mongosh`) & Compass — `mongodb+srv://` URIs, Scripting & Diagnostics

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Mongosh modern JS shell & Mongodb+srv:// DNS seedlist • Core Syntax, CLI & Production APIs: mongosh "mongodb+srv://cluster.mongodb.net/app", show dbs, show collections, db.currentOp() • Deep-Dive Mechanics: BindIp private network security & Db.currentOp() & db.killOp() • Advanced Patterns: MongoDB Compass visual profiler & Mongosh --file automation • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Listing Collections & Checking Connection Read-State in `mongosh` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for MongoDB Shell (`mongosh`)

Purpose: To understand and apply MongoDB Shell (`mongosh`) & Compass — `mongodb+srv://` URIs, Scripting & Diagnostics in real MongoDB systems — specifically: Modern Node.js-powere...

What Topic Is Next → Next Topic 2.2: Creating & Dropping Databases — Lazy Materialization, `db.stats()` & Namespace Hygiene

2.2 Creating & Dropping Databases — Lazy Materialization, `db.stats()` & Namespace Hygiene

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Lazy database creation on first write & Admin / local / config system DBs • Core Syntax, CLI & Production APIs: use("academy_db"); db.stats(); db.getMongo().getDBNames(); db.dropDatabase(); • Deep-Dive Mechanics: Lowercase snake_case naming rule & Db.stats() storageSize vs dataSize • Advanced Patterns: RBAC block on dropDatabase() & Namespace limits • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Switching Context & Verifying Database Stats Structure • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Creating

Purpose: To understand and apply Creating & Dropping Databases — Lazy Materialization, `db.stats()` & Namespace Hygiene in real MongoDB systems — specifically: Why `use academy_db...

What Topic Is Next → Next Topic 2.3: Collections — `createCollection()`, `$jsonSchema` Validation, Capped & Time-Series Collections

2.3 Collections — `createCollection()`, `$jsonSchema` Validation, Capped & Time-Series Collections

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: $jsonSchema server-side validator & ValidationLevel strict vs moderate • Core Syntax, CLI & Production APIs: db.createCollection("leads", { validator: { $jsonSchema: { bsonType: "object", required: ["e... • Deep-Dive Mechanics: ValidationAction error vs warn & Capped circular collections • Advanced Patterns: Time-Series collections & CollMod online validator update • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Creating a `$jsonSchema`-Validated Collection & Testing Constraint Enforcement • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Collections

Purpose: To understand and apply Collections — `createCollection()`, `$jsonSchema` Validation, Capped & Time-Series Collections in real MongoDB systems — specifically: Implicit vs...

What Topic Is Next → Next Topic 2.4: MongoDB Data Modeling — Embedding vs Referencing, Subset/Bucket Patterns & Unbounded Arrays

2.4 MongoDB Data Modeling — Embedding vs Referencing, Subset/Bucket Patterns & Unbounded Arrays

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Embed (1:Few) vs Reference (1:Many/Squillions) & Unbounded Array 16 MB anti-pattern • Core Syntax, CLI & Production APIs: Embed (1:Few) vs Child->Parent Reference (1:Squillions) + Subset / Bucket / Outlier Patterns • Deep-Dive Mechanics: Subset Pattern (recent N embedded) & Extended Reference denormalization • Advanced Patterns: Bucket Pattern (IoT/metrics) & Read-to-write ratio design • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Subset Pattern: Course Document Embedding Top-2 Preview Topics + Count • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for MongoDB Data Modeling

Purpose: To understand and apply MongoDB Data Modeling — Embedding vs Referencing, Subset/Bucket Patterns & Unbounded Arrays in real MongoDB systems — specifically: 1-to-1 and 1-t...

What Topic Is Next → Module 2 Capstone (Validated Student Telemetry & Cohort Schema (`$jsonSchema` + Bucket Pattern)) → Next Topic 3.1: Inserting Documents — `insertOne()`, `insertMany({ ordered: false })` & Duplicate-Key Idempotency

Module 2 Hands-On Capstone Project (100 Marks): Validated Student Telemetry & Cohort Schema (`$jsonSchema` + Bucket Pattern) — Create a `students` collection protected by server-side `$jsonSchema` validation (`bsonType`, `required`, `minimum`/`maximum` on scores, regex on email) and a `topic_events_buckets` collecti...

MODULE 3 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: CRUD Mastery — Bulk Inserts, `find()` Projections, Cursors, `sort()` & Keyset Pagination

Perform high-speed ordered/unordered batch inserts, shape lean network payloads with projections, stream million-doc cursors without OOM crashes, and replace slow `skip(100000)` with O(log N) range pagination.

Objectives: ✓ Inserting Documents — `insertOne()`, `insertMany({ ordered: false })` & Duplicate-Key Idempotency — How `insertOne` and `insertMany` auto-assign `_id` client-side in the driver before network transmission, `ordered: true` (default — halts at first error) vs `ordered: false` (continues inserting remaining valid docs in parallel shards!), and `E11000 duplicate key error`. • ✓ `find()` & Projections — Nested Dot-Notation, Exact Object Match Trap & Covered Projections — Querying top-level and nested sub-document fields (`"pricing.offer": 999` — must wrap dot-notation keys in quotes!), why `{ pricing: { mrp: 4999, offer: 999 } }` requires EXACT field order & count (never do that — use dot notation!), and second-argument Projections (`{ title: 1, slug: 1, _id: 0 }`). • ✓ MongoDB Cursors — Lazy Batches (`for await`), `batchSize()`, Cursor Timeouts & `countDocuments` vs `estimatedDocumentCount` — Why `db.col.find()` returns a lazy **Cursor** (NOT an array!) fetching ~101 docs initially then `getMore` batches on demand, streaming via `for await (const doc of cursor)` in Node.js, 10-minute `noCursorTimeout` hazards, and exact `countDocuments(filter)` vs O(1) metadata `estimatedDocumentCount()`. • ✓ `sort()`, `limit()` & `skip()` — 100 MB Blocking Sort Cap, BSON Type Sort Order & Keyset Pagination — Ascending (`1`) vs descending (`-1`) sorting, MongoDB's **100 MB in-memory blocking sort limit** (`Sort exceeded memory limit of 104857600 bytes` unless backed by an index or `allowDiskUse()`), deterministic tie-breaker sorts (`{ score: -1, _id: 1 }`), and why `skip(200000)` degrades linearly. • ✓ Build, verify, and ship Module Capstone: High-Throughput Question Bank Importer & Sub-Millisecond Keyset Paginator.

3.1 Inserting Documents — `insertOne()`, `insertMany({ ordered: false })` & Duplicate-Key Idempotency

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: InsertOne vs batched insertMany & Ordered: false continue-on-error • Core Syntax, CLI & Production APIs: db.leads.insertOne(doc); db.leads.insertMany(docs, { ordered: false }); • Deep-Dive Mechanics: Client-driver _id generation & E11000 duplicate key error • Advanced Patterns: RetryWrites=true idempotency & Atomic per-document write • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Unordered Batch `insertMany` Surviving a Duplicate `_id` Cleanly • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Inserting Documents

Purpose: To understand and apply Inserting Documents — `insertOne()`, `insertMany({ ordered: false })` & Duplicate-Key Idempotency in real MongoDB systems — specifically: How `ins...

What Topic Is Next → Next Topic 3.2: `find()` & Projections — Nested Dot-Notation, Exact Object Match Trap & Covered Projections

3.2 `find()` & Projections — Nested Dot-Notation, Exact Object Match Trap & Covered Projections

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Quoted "nested.field" dot notation & Sub-document byte-order equality trap • Core Syntax, CLI & Production APIs: db.courses.find({ "pricing.offer": 999 }, { title: 1, slug: 1, _id: 0 }) • Deep-Dive Mechanics: Projection inclusion/exclusion rule & _id: 0 exception for covered queries • Advanced Patterns: FindOne() vs find() cursor & $slice array projection • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Dot-Notation Nested Query with Lean `_id: 0` Projection • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for `find()`

Purpose: To understand and apply `find()` & Projections — Nested Dot-Notation, Exact Object Match Trap & Covered Projections in real MongoDB systems — specifically: Querying top-l...

What Topic Is Next → Next Topic 3.3: MongoDB Cursors — Lazy Batches (`for await`), `batchSize()`, Cursor Timeouts & `countDocuments` vs `estimatedDocumentCount`

3.3 MongoDB Cursors — Lazy Batches (`for await`), `batchSize()`, Cursor Timeouts & `countDocuments` vs `estimatedDocumentCount`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Lazy cursor getMore batches & Never .toArray() million-doc sets • Core Syntax, CLI & Production APIs: cursor.batchSize(500), for await (const doc of col.find(q)), col.countDocuments(q), col.esti... • Deep-Dive Mechanics: For await...of streaming & 10-minute idle cursor timeout • Advanced Patterns: CountDocuments vs O(1) estimatedDocumentCount & Cursor.explain("executionStats") • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Streaming Cursor Iteration & Comparing Metadata Count Methods • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for MongoDB Cursors

Purpose: To understand and apply MongoDB Cursors — Lazy Batches (`for await`), `batchSize()`, Cursor Timeouts & `countDocuments` vs `estimatedDocumentCount` in real MongoDB system...

What Topic Is Next → Next Topic 3.4: `sort()`, `limit()` & `skip()` — 100 MB Blocking Sort Cap, BSON Type Sort Order & Keyset Pagination

3.4 `sort()`, `limit()` & `skip()` — 100 MB Blocking Sort Cap, BSON Type Sort Order & Keyset Pagination

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Logical order: sort -> skip -> limit & 100 MB in-memory sort limit • Core Syntax, CLI & Production APIs: db.attempts.find({ _id: { $lt: cursorId } }).sort({ _id: -1 }).limit(20) • Deep-Dive Mechanics: Index-backed non-blocking sort & Tie-breaker { score: -1, _id: 1 } • Advanced Patterns: Keyset _id range vs slow skip() & BSON cross-type sort order • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Deterministic Tie-Breaker Sort & Keyset Cursor Slice (`_id < lastSeen`) • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for `sort()`

Purpose: To understand and apply `sort()`, `limit()` & `skip()` — 100 MB Blocking Sort Cap, BSON Type Sort Order & Keyset Pagination in real MongoDB systems — specifically: Ascend...

What Topic Is Next → Module 3 Capstone (High-Throughput Question Bank Importer & Sub-Millisecond Keyset Paginator) → Next Topic 4.1: Updating Documents — `updateOne`, `updateMany`, `replaceOne` & Atomic `findOneAndUpdate`

Module 3 Hands-On Capstone Project (100 Marks): High-Throughput Question Bank Importer & Sub-Millisecond Keyset Paginator — Build an idempotent `insertMany({ ordered: false })` importer loading 10,000 MCQ documents with deterministic `_id` keys, paired with a lean projected `find({ topicId, _id: { $lt: lastSeenId...

MODULE 4 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Atomic Document Updates, Array Operators, Idempotent Upserts & TTL Deletions

Mutate documents atomically in place without read-modify-write race conditions using `$set`, `$inc`, `$push`, `$addToSet`, `$pull`, `findOneAndUpdate()`, `{ upsert: true }`, and self-cleaning TTL indexes.

Objectives: ✓ Updating Documents — `updateOne`, `updateMany`, `replaceOne` & Atomic `findOneAndUpdate` — Targeted field patches with `updateOne`/`updateMany`, atomic compare-and-swap via `findOneAndUpdate(filter, update, { returnDocument: "after" })`, why `updateOne` rejects plain objects lacking `$` operators (`Update document requires atomic operators`), and `matchedCount` vs `modifiedCount`. • ✓ MongoDB Update Operators — `$set`, `$unset`, `$inc`, `$min`/`$max`, `$push`/`$slice`, `$addToSet` & `$pull` — Field operators (`$set`, `$unset`, `$inc`, `$mul`, `$min`, `$max`, `$currentDate`) and array operators (`$push` with `$each` + `$slice` bounded ring-buffer, `$addToSet` unique set append, `$pull` conditional delete, and `$[elem]` `arrayFilters`). • ✓ Idempotent Upserts (`{ upsert: true }`, `$setOnInsert`) & Unique Index Concurrency Safety — Insert-or-update in one command via `{ upsert: true }`, initializing creation-only fields (`createdAt`, `referralCode`) with `$setOnInsert` (untouched on subsequent updates!), and why a **Unique Index** on the filter key is mandatory for race-free concurrent upserts. • ✓ Deleting Documents — `deleteOne()`, `deleteMany()`, Soft Deletes & Self-Expiring **TTL Indexes** — Removing documents via `deleteOne`/`deleteMany`/`findOneAndDelete`, why `db.col.drop()` reclaims disk blocks instantly vs `deleteMany({})` scanning every doc, Soft Deletes (`deletedAt`), and background **TTL (Time-To-Live) Indexes** (`expireAfterSeconds`). • ✓ Build, verify, and ship Module Capstone: Race-Free Seat Counter, Bounded Recent-Activity Ring Buffer & Self-Expiring OTP Store.

4.1 Updating Documents — `updateOne`, `updateMany`, `replaceOne` & Atomic `findOneAndUpdate`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: UpdateOne vs updateMany vs replaceOne & FindOneAndUpdate atomic CAS • Core Syntax, CLI & Production APIs: db.batches.findOneAndUpdate({ code: "MONG", seats: { $gt: 0 } }, { $inc: { seats: -1 } }, {... • Deep-Dive Mechanics: ReturnDocument: "after" option & MatchedCount vs modifiedCount • Advanced Patterns: Aggregation pipeline in update [] & Single-document write atomicity • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Atomic Seat Decrement Guarded by `seatsLeft: { $gt: 0 }` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Updating Documents

Purpose: To understand and apply Updating Documents — `updateOne`, `updateMany`, `replaceOne` & Atomic `findOneAndUpdate` in real MongoDB systems — specifically: Targeted field pa...

What Topic Is Next → Next Topic 4.2: MongoDB Update Operators — `$set`, `$unset`, `$inc`, `$min`/`$max`, `$push`/`$slice`, `$addToSet` & `$pull`

4.2 MongoDB Update Operators — `$set`, `$unset`, `$inc`, `$min`/`$max`, `$push`/`$slice`, `$addToSet` & `$pull`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: $inc atomic counter & $max / $min high-water mark • Core Syntax, CLI & Production APIs: $set, $unset, $inc, $max, $addToSet, $pull, $push: { items: { $each: [x], $slice: -20 } }, a... • Deep-Dive Mechanics: $addToSet + $each trap & $push + $each + $slice ring buffer • Advanced Patterns: $pull conditional array removal & ArrayFilters $[elem] targeted update • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: High-Score `$max`, Unique `$addToSet` (`$each`), and Bounded `$slice` Update • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for MongoDB Update Operators

Purpose: To understand and apply MongoDB Update Operators — `$set`, `$unset`, `$inc`, `$min`/`$max`, `$push`/`$slice`, `$addToSet` & `$pull` in real MongoDB systems — specifically...

What Topic Is Next → Next Topic 4.3: Idempotent Upserts (`{ upsert: true }`, `$setOnInsert`) & Unique Index Concurrency Safety

4.3 Idempotent Upserts (`{ upsert: true }`, `$setOnInsert`) & Unique Index Concurrency Safety

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Upsert: true insert-or-update & $setOnInsert creation-only fields • Core Syntax, CLI & Production APIs: db.daily_metrics.updateOne({ course: "MONG", day: "2026-10-31" }, { $inc: { views: 1 }, $set... • Deep-Dive Mechanics: Unique Index required for race safety & UpsertedId & upsertedCount • Advanced Patterns: Idempotent webhook ingestion & Daily counter rollups • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Idempotent Daily Course Counter Using `upsert: true` & `$setOnInsert` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Idempotent Upserts (`{ upsert: true }`, `$setOnInser...

Purpose: To understand and apply Idempotent Upserts (`{ upsert: true }`, `$setOnInsert`) & Unique Index Concurrency Safety in real MongoDB systems — specifically: Insert-or-update...

What Topic Is Next → Next Topic 4.4: Deleting Documents — `deleteOne()`, `deleteMany()`, Soft Deletes & Self-Expiring **TTL Indexes**

4.4 Deleting Documents — `deleteOne()`, `deleteMany()`, Soft Deletes & Self-Expiring **TTL Indexes**

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: TTL single-field Date index rule & ExpireAfterSeconds: 0 per-doc expiry • Core Syntax, CLI & Production APIs: db.otps.createIndex({ expiresAt: 1 }, { expireAfterSeconds: 0 }); db.sessions.deleteMany({ r... • Deep-Dive Mechanics: 60-second background TTL sweep & DeleteMany({}) vs fast col.drop() • Advanced Patterns: FindOneAndDelete queue pop & Soft Delete Partial Index • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Creating a Per-Document Custom Expiration TTL Index (`expireAfterSeconds: 0`) • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Deleting Documents

Purpose: To understand and apply Deleting Documents — `deleteOne()`, `deleteMany()`, Soft Deletes & Self-Expiring **TTL Indexes** in real MongoDB systems — specifically: Removing...

What Topic Is Next → Module 4 Capstone (Race-Free Seat Counter, Bounded Recent-Activity Ring Buffer & Self-Expiring OTP Store) → Next Topic 5.1: Query & Array Operators — `$gt`/`$in`, `$exists`, `$type`, `$all`, `$size` & The `$elemMatch` Rule

Module 4 Hands-On Capstone Project (100 Marks): Race-Free Seat Counter, Bounded Recent-Activity Ring Buffer & Self-Expiring OTP Store — Implement an atomic cohort seat reservation using `findOneAndUpdate({ code, seatsLeft: { $gt: 0 } }, { $inc: { seatsLeft: -1 }, $push: { recentJoins: { $each: [entry], $slice: -10 } } }, { r...

MODULE 5 OF 16 • STAGE 2 • 10 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Comparison, Logical, Array (`$elemMatch`) & Full-Text Search Query Operators

Filter complex nested BSON structures using `$in`, `$exists`, `$type`, `$elemMatch` multi-condition array matching, boolean `$and`/`$or` index merges, and weighted `$text` search.

Objectives: ✓ Query & Array Operators — `$gt`/`$in`, `$exists`, `$type`, `$all`, `$size` & The `$elemMatch` Rule — Comparison (`$eq`, `$ne`, `$gt`, `$gte`, `$lt`, `$lte`, `$in`, `$nin`), Element (`$exists`, `$type`), and Array operators (`$all`, `$size`, `$elemMatch`) — especially why `{ "scores": { $gt: 80, $lt: 90 } }` matches `{ scores: [50, 95] }` unless wrapped in `$elemMatch`!. • ✓ Logical Query Operators — Implicit vs Explicit `$and`, `$or` Index Merging, `$nor` & `$not` — Implicit comma-AND `{ a: 1, b: 2 }` vs explicit `$and: [{ $or: [...] }, { $or: [...] }]` (required when the same field key repeats twice in JS!), how `$or` requires **EVERY** branch to be indexed to avoid a `COLLSCAN`, and `$not` negation. • ✓ Full-Text Search (`$text`, `$search`, `textScore`) & Anchored Prefix Regular Expressions — Stemming & stop-word tokenization with `{ title: "text", summary: "text" }`, phrase search `"\"replica set\""` and negation `"-legacy"`, sorting by `{ score: { $meta: "textScore" } }`, and why only left-anchored case-sensitive `/^MAA-/` regexes use standard B-Tree indexes. • ✓ Build, verify, and ship Module Capstone: Multi-Faceted Course & Skill Search Engine (`$elemMatch` + Weighted `$text` + Anchored Prefix).

5.1 Query & Array Operators — `$gt`/`$in`, `$exists`, `$type`, `$all`, `$size` & The `$elemMatch` Rule

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: $elemMatch single-element array match & { field: null } matches missing too • Core Syntax, CLI & Production APIs: db.users.find({ attempts: { $elemMatch: { course: "MONG", score: { $gte: 80 } } } }) • Deep-Dive Mechanics: $type: "null" vs $exists: false & $all subset array match • Advanced Patterns: $size cannot take ranges & $in vs slow $ne/$nin • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Proving `$elemMatch` Prevents Cross-Element False Positives • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Query

Purpose: To understand and apply Query & Array Operators — `$gt`/`$in`, `$exists`, `$type`, `$all`, `$size` & The `$elemMatch` Rule in real MongoDB systems — specifically: Compari...

What Topic Is Next → Next Topic 5.2: Logical Query Operators — Implicit vs Explicit `$and`, `$or` Index Merging, `$nor` & `$not`

5.2 Logical Query Operators — Implicit vs Explicit `$and`, `$or` Index Merging, `$nor` & `$not`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Every $or branch needs an index & Prefer $in over same-field $or • Core Syntax, CLI & Production APIs: db.courses.find({ $and: [{ $or: [{ fee: 999 }, { featured: true }] }, { published: true }] }... • Deep-Dive Mechanics: Duplicate JS object key overwrite trap & Explicit $and for multi-$or • Advanced Patterns: $not matches missing fields too & Subplan index intersection • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Combining Multi-Clause `$or` Filters Safely Inside Explicit `$and` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Logical Query Operators

Purpose: To understand and apply Logical Query Operators — Implicit vs Explicit `$and`, `$or` Index Merging, `$nor` & `$not` in real MongoDB systems — specifically: Implicit comma...

What Topic Is Next → Next Topic 5.3: Full-Text Search (`$text`, `$search`, `textScore`) & Anchored Prefix Regular Expressions

5.3 Full-Text Search (`$text`, `$search`, `textScore`) & Anchored Prefix Regular Expressions

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Max 1 text index per collection & Field weights { title: 10 } • Core Syntax, CLI & Production APIs: db.courses.createIndex({ title: "text", tagline: "text" }, { weights: { title: 10, tagline:... • Deep-Dive Mechanics: $meta: "textScore" relevance sort & Space = OR vs quoted phrase = AND • Advanced Patterns: Anchored /^prefix/ B-Tree index seek & Atlas Search Lucene for fuzzy/autocomplete • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Weighted Compound Text Index & Relevance-Sorted `$text` Query • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Full-Text Search (`$text`, `$search`, `textScore`)

Purpose: To understand and apply Full-Text Search (`$text`, `$search`, `textScore`) & Anchored Prefix Regular Expressions in real MongoDB systems — specifically: Stemming & stop-w...

What Topic Is Next → Module 5 Capstone (Multi-Faceted Course & Skill Search Engine (`$elemMatch` + Weighted `$text` + Anchored Prefix)) → Next Topic 6.1: B-Tree Indexing Mastery — Compound **ESR Rule, Multikey Array Limits, Partial & Sparse Indexes**

Module 5 Hands-On Capstone Project (100 Marks): Multi-Faceted Course & Skill Search Engine (`$elemMatch` + Weighted `$text` + Anchored Prefix) — Build a MongoDB search query layer supporting embedded module array filtering via `$elemMatch`, multi-tag `$all` + `$in` filters, weighted full-text relevance ranking (`{ score: { $meta: "te...

MODULE 6 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: B-Tree Indexing Internals, The **ESR Rule**, `explain("executionStats")` & `bulkWrite()`

Eliminate slow `COLLSCAN` queries by mastering MongoDB's **ESR (Equality, Sort, Range)** compound index rule, Multikey/Partial/Wildcard indexes, `.explain("executionStats")` ratio tuning, and `bulkWrite()`.

Objectives: ✓ B-Tree Indexing Mastery — Compound **ESR Rule**, Multikey Array Limits, Partial & Sparse Indexes — Single-field vs Compound indexes, the golden **ESR Rule (Equality -> Sort -> Range)**, **Multikey Indexes** on array fields (cannot index two arrays in one compound index!), and RAM-saving **Partial Indexes** (`partialFilterExpression: { status: "active" }`). • ✓ Query Plan Cache & `.explain("executionStats")` — `IXSCAN` vs `COLLSCAN`, `FETCH` & Covered Queries — How MongoDB's empirical Query Planner races candidate indexes, caches the winning plan (`getPlanCache()`), and how to read `.explain("executionStats")`: `nReturned`, `totalKeysExamined`, `totalDocsExamined`, `IXSCAN`, `COLLSCAN`, `SORT`, and `PROJECTION_COVERED`. • ✓ High-Throughput `bulkWrite()` — Combining `insertOne`, `updateOne`, `replaceOne` & `deleteOne` — Executing mixed heterogenous write batches in a single network round-trip with `collection.bulkWrite([...], { ordered: false })`, inspecting `BulkWriteResult` counters, and chunking 100k+ ETL syncs into 1,000-op batches. • ✓ Build, verify, and ship Module Capstone: ESR Index Redesign, Covered Query Verification & 50,000-Op `bulkWrite` Sync Engine.

6.1 B-Tree Indexing Mastery — Compound **ESR Rule**, Multikey Array Limits, Partial & Sparse Indexes

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: ESR Rule (Equality -> Sort -> Range) & Multikey parallel-array restriction • Core Syntax, CLI & Production APIs: db.attempts.createIndex({ status: 1, createdAt: -1, score: 1 }); // Equality -> Sort -> Rang... • Deep-Dive Mechanics: Compound leftmost prefix rule & Partial Index RAM savings • Advanced Patterns: Sparse vs Partial indexes & $indexStats unused index audit • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Building an ESR Compound Index &

Verifying Index Definitions • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for B-Tree Indexing Mastery

Purpose: To understand and apply B-Tree Indexing Mastery — Compound **ESR Rule**, Multikey Array Limits, Partial & Sparse Indexes in real MongoDB systems — specifically: Single-fi...

What Topic Is Next → Next Topic 6.2: Query Plan Cache & `.explain("executionStats")` — `IXSCAN` vs `COLLSCAN`, `FETCH` & Covered Queries

6.2 Query Plan Cache & `.explain("executionStats")` — `IXSCAN` vs `COLLSCAN`, `FETCH` & Covered Queries [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: IXSCAN vs COLLSCAN & NReturned vs totalDocsExamined 1:1 • Core Syntax, CLI & Production APIs: db.attempts.find(filter, proj).explain("executionStats") • Deep-Dive Mechanics: Covered Query (totalDocsExamined: 0) & Blocking SORT stage detection • Advanced Patterns: Empirical multi-plan race & Plan Cache & Hint() index override caution • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Achieving `totalDocsExamined: 0` with a True Covered Query • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Query Plan Cache

Purpose: To understand and apply Query Plan Cache & `.explain("executionStats")` — `IXSCAN` vs `COLLSCAN`, `FETCH` & Covered Queries in real MongoDB systems — specifically: How Mo...

What Topic Is Next → Next Topic 6.3: High-Throughput `bulkWrite()` — Combining `insertOne`, `updateOne`, `replaceOne` & `deleteOne`

6.3 High-Throughput `bulkWrite()` — Combining `insertOne`, `updateOne`, `replaceOne` & `deleteOne` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Mixed insert/update/delete batch & Single network round-trip • Core Syntax, CLI & Production APIs: db.courses.bulkWrite([{ insertOne: { document: d } }, { updateOne: { filter, update, upsert:... • Deep-Dive Mechanics: Ordered: false parallel shard fanout & 1,000-op batch chunking • Advanced Patterns: BulkWriteResult counters & Idempotent ETL synchronization • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Mixed `insertOne` + `updateOne` + `deleteOne` in One `bulkWrite()` Call • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for High-Throughput `bulkWrite()`

Purpose: To understand and apply High-Throughput `bulkWrite()` — Combining `insertOne`, `updateOne`, `replaceOne` & `deleteOne` in real MongoDB systems — specifically: Executing m...

What Topic Is Next → Module 6 Capstone (ESR Index Redesign, Covered Query Verification & 50,000-Op `bulkWrite` Sync Engine) → Next Topic 7.1: Aggregation Pipeline Architecture — `$match`, `$project`, `$addFields`/`$set`, `$sort`, `$limit` & `$merge`

Module 6 Hands-On Capstone Project (100 Marks): ESR Index Redesign, Covered Query Verification & 50,000-Op `bulkWrite` Sync Engine — Optimize a slow student attempt query (`status == "submitted"`, sorted by `submittedAt DESC`, range `score >= 60`) using an exact ESR compound index `{ status: 1, submittedAt: -1, score: 1 }...

MODULE 7 OF 16 • STAGE 2 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: MongoDB Aggregation Framework (`$match`, `$group`, `$unwind`, `$lookup`, `$facet`) & Geospatial

Build real-time analytical pipelines directly inside MongoDB using `$match`, `$project`, `$group`, `$unwind`, `$lookup` joins, single-scan `$facet` dashboards, and `2dsphere` Geospatial queries.

Objectives: ✓ Aggregation Pipeline Architecture — `$match`, `$project`, `$addFields`/`$set`, `$sort`, `$limit` & `$merge` — Unix-pipe-style stage composition `db.col.aggregate([...])`, automatic optimizer rules (`$sort` + `$limit` top-K coalescence, `$match` pushdown), `$set`/`$addFields` vs destructive `$project`, 100 MB per-stage RAM limit (`allowDiskUse: true`), and materializing summary collections via `$merge`. • ✓ `$group`, `$bucket` & `$facet` — Accumulators (`$sum`, `$avg`, `$topN`) & Single-Scan Dashboards — Grouping by `_id: "$category"` (or `_id: null` for grand totals), accumulators (`$sum: 1` for `COUNT(*)`, `$avg`, `$min`, `$max`, `$addToSet`, MongoDB 5.2+ `$topN`), histogram bucketing (`$bucket`), and running multiple sub-pipelines in ONE scan via `$facet`. • ✓ `$unwind` Array Deconstruction & `$lookup` Cross-Collection Left-Outer Joins — Flattening an N-element array into N separate documents via `$unwind`, keeping empty/missing arrays via `preserveNullAndEmptyArrays: true` (otherwise `$unwind` drops them like an `INNER JOIN`!), and joining collections with `$lookup` (`localField`/`foreignField` + filtered `let`/`pipeline`). • ✓ Geospatial Indexing (`2dsphere`) — GeoJSON `[lng, lat]` Order, `$nearSphere` & `$geoNear` — Storing RFC-7946 **GeoJSON** `{ type: "Point", coordinates: [longitude, latitude] }`, building `{ location: "2dsphere" }` spherical Earth indexes, radius searches via `$nearSphere` (`$maxDistance` in meters), polygon containment (`$geoWithin`), and `$geoNear` (must be Stage 1 in `aggregate`!). • ✓ Build, verify, and ship Module Capstone: Multi-Faceted Executive Analytics Pipeline (`$match` -> `$lookup` -> `$facet`) & Nearby Center Locator.

7.1 Aggregation Pipeline Architecture — `$match`, `$project`, `$addFields`/`$set`, `$sort`, `$limit` & `$merge` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Early $match index pushdown & $sort + $limit Top-K heap coalescence • Core Syntax, CLI & Production APIs: db.orders.aggregate([{ $match: { status: "paid" } }, { $set: { net: { $subtract: ["$mrp", "$... • Deep-Dive Mechanics: $set/$addFields vs $project & 100 MB stage RAM & allowDiskUse • Advanced Patterns: $merge incremental vs $out replace & Field path "$amount" syntax • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Filtered Computed Net-Fee Pipeline Using `$match` + `$set` + Top-K `$sort`/`$limit` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Aggregation Pipeline Architecture

Purpose: To understand and apply Aggregation Pipeline Architecture — `$match`, `$project`, `$addFields`/`$set`, `$sort`, `$limit` & `$merge` in real MongoDB systems — specifically...

What Topic Is Next → Next Topic 7.2: `$group`, `$bucket` & `$facet` — Accumulators (`$sum`, `$avg`, `$topN`) & Single-Scan Dashboards

7.2 `$group`, `$bucket` & `$facet` — Accumulators (`$sum`, `$avg`, `$topN`) & Single-Scan Dashboards [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: _id group key vs _id: null grand total & $sum: 1 row count idiom • Core Syntax, CLI & Production APIs: { $group: { _id: "$courseCode", count: { $sum: 1 }, revenue: { $sum: "$fee" }, avgScore: { $... • Deep-Dive Mechanics: Avoid $push: "$$ROOT" 16MB crash & $topN bounded group top-K • Advanced Patterns: $facet single-pass pagination + count & $bucket score histograms • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Per-Track Enrollment Count & Revenue Summary Using `$group` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation

Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for `$group`

Purpose: To understand and apply `$group`, `$bucket` & `$facet` — Accumulators (`$sum`, `$avg`, `$topN`) & Single-Scan Dashboards in real MongoDB systems — specifically: Grouping...

What Topic Is Next → Next Topic 7.3: `$unwind` Array Deconstruction & `$lookup` Cross-Collection Left-Outer Joins

7.3 `$unwind` Array Deconstruction & `$lookup` Cross-Collection Left-Outer Joins [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: PreserveNullAndEmptyArrays: true & Index foreignField in $lookup • Core Syntax, CLI & Production APIs: { $unwind: { path: "$tags", preserveNullAndEmptyArrays: true } }, { $lookup: { from: "users"... • Deep-Dive Mechanics: Paginate ($limit) BEFORE $lookup & $lookup sub-pipeline with let • Advanced Patterns: $unwind + $group tag frequency & Avoiding N*M unindexed $lookup • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Tag Frequency Analytics Combining `$unwind` + `$group` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for `$unwind` Array Deconstruction

Purpose: To understand and apply `$unwind` Array Deconstruction & `$lookup` Cross-Collection Left-Outer Joins in real MongoDB systems — specifically: Flattening an N-element array...

What Topic Is Next → Next Topic 7.4: Geospatial Indexing (`2dsphere`) — GeoJSON `[lng, lat]` Order, `$nearSphere` & `$geoNear`

7.4 Geospatial Indexing (`2dsphere`) — GeoJSON `[lng, lat]` Order, `$nearSphere` & `$geoNear` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: GeoJSON [longitude, latitude] order! & 2dsphere spherical S2 index • Core Syntax, CLI & Production APIs: db.centers.createIndex({ location: "2dsphere" }); // GeoJSON Point: [longitude, latitude] • Deep-Dive Mechanics: $nearSphere $maxDistance meters & $geoNear MUST be first pipeline stage • Advanced Patterns: $geoWithin Polygon geofencing & GeoJSON Point / Polygon spec • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: 2dsphere GeoJSON Proximity Search Around Bengaluru (`[77.5946, 12.9716]`) • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Geospatial Indexing (`2dsphere`)

Purpose: To understand and apply Geospatial Indexing (`2dsphere`) — GeoJSON `[lng, lat]` Order, `$nearSphere` & `$geoNear` in real MongoDB systems — specifically: Storing RFC-7946...

What Topic Is Next → Module 7 Capstone (Multi-Faceted Executive Analytics Pipeline (`$match` -> `$lookup` -> `$facet`) & Nearby Center Locator) → Next Topic 8.1: Client Sessions & Multi-Document ACID Transactions (`startSession`, `withTransaction`)

Module 7 Hands-On Capstone Project (100 Marks): Multi-Faceted Executive Analytics Pipeline (`$match` -> `$lookup` -> `$facet`) & Nearby Center Locator — Write a single-pass `$facet` MongoDB Aggregation Pipeline returning paginated course leaderboard rows, score distribution buckets (`$bucket`), and overall revenue KPIs simultaneously, plus a...

MODULE 8 OF 16 • STAGE 2 • 10 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Multi-Document ACID Transactions, Read/Write Concerns & SQL-to-MongoDB Translation

Execute multi-collection ACID transactions safely with `session.withTransaction()`, tune durability vs latency via `writeConcern` (`w: "majority"`) & `readConcern`, and translate complex SQL mindsets to MongoDB.

Objectives: ✓ Client Sessions & Multi-Document ACID Transactions (`startSession`, `withTransaction`) — Introduced in MongoDB 4.0 (Replica Sets) & 4.2 (Sharded Clusters): snapshot-isolated multi-document ACID transactions, why `session.withTransaction(fn)` (Callback API) beats manual `startTransaction` (auto-retries `TransientTransactionError` & `UnknownTransactionCommitResult`!), and causal consistency. • ✓ Tunable Durability & Isolation — `writeConcern` (`w: "majority"`, `j: true`) & `readConcern` (`majority`, `snapshot`) — How `w: 1` acknowledges after Primary memory write (risking rollback if Primary crashes before replicating!), why MongoDB 5.0+ defaults to **`w: "majority"`** (acknowledged after majority of Replica Set nodes persist to journal!), and `readConcern` levels (`local`, `available`, `majority`, `linearizable`, `snapshot`). • ✓ SQL-to-MongoDB Translation Guide — DDL, DML, `JOIN`, `GROUP BY` & `HAVING` Equivalents — Side-by-side mental map: `TABLE` -> `Collection`, `ROW` -> `BSON Document`, `COLUMN` -> `Field`, `PRIMARY KEY` -> `_id`, `SELECT a, b WHERE` -> `find(filter, proj)`, `GROUP BY ... HAVING` -> `[$group, $match]`, and `LEFT JOIN` -> `$lookup` (or Embedded Sub-Documents!). • ✓ Build, verify, and ship Module Capstone: Multi-Collection Wallet Debit + Certificate Issuance ACID Transaction Engine.

8.1 Client Sessions & Multi-Document ACID Transactions (`startSession`, `withTransaction`) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Always pass { session } to every op & WithTransaction auto-retry API • Core Syntax, CLI & Production APIs: const session = client.startSession(); await session.withTransaction(async () => { await col... • Deep-Dive Mechanics: Requires Replica Set or Sharded Cluster & Snapshot isolation inside transaction • Advanced Patterns: 60s transactionLifetimeLimitSeconds & Single-doc atomicity first • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Verifying Logical Session Allocation & Session ID in `mongosh` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Client Sessions

Purpose: To understand and apply Client Sessions & Multi-Document ACID Transactions (`startSession`, `withTransaction`) in real MongoDB systems — specifically: Introduced in Mongo...

What Topic Is Next → Next Topic 8.2: Tunable Durability & Isolation — `writeConcern` (`w: "majority"`, `j: true`) & `readConcern` (`majority`, `snapshot`)

8.2 Tunable Durability & Isolation — `writeConcern` (`w: "majority"`, `j: true`) & `readConcern` (`majority`, `snapshot`) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: W: "majority" MongoDB 5.0+ default & W: 1 failover rollback risk • Core Syntax, CLI & Production APIs: writeConcern: { w: "majority", j: true, wtimeout: 5000 }, readConcern: { level: "majority" } • Deep-Dive Mechanics: J: true WiredTiger journal flush & ReadConcern local vs majority vs linearizable • Advanced Patterns: Wtimeout safety bound & Read-Your-Own-Writes causal sessions • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Executing a Write with Explicit `j: true` Journal Acknowledgment • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Tunable Durability

Purpose: To understand and apply Tunable Durability & Isolation — `writeConcern` (`w: "majority"`, `j: true`) & `readConcern` (`majority`, `snapshot`) in real MongoDB systems — sp...

What Topic Is Next → Next Topic 8.3: SQL-to-MongoDB Translation Guide — DDL, DML, `JOIN`, `GROUP BY` & `HAVING` Equivalents

8.3 SQL-to-MongoDB Translation Guide — DDL, DML, `JOIN`, `GROUP BY` & `HAVING` Equivalents [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: WHERE = pre-$group $match & HAVING = post-$group $match • Core Syntax, CLI & Production APIs: SQL WHERE -> $match (before $group) | SQL HAVING -> $match (AFTER $group) | SQL LEFT JOIN ->... • Deep-Dive Mechanics: LEFT JOIN = $lookup or Embedding & ALTER TABLE = online schemaVersion • Advanced Patterns: COUNT(*) = { $sum: 1 } & Query-driven data modeling • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Translating `SELECT track, SUM(fee) ... WHERE active = 1 GROUP BY track HAVING total >= 1500` • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for SQL-to-MongoDB Translation Guide

Purpose: To understand and apply SQL-to-MongoDB Translation Guide — DDL, DML, `JOIN`, `GROUP BY` & `HAVING` Equivalents in real MongoDB systems — specifically: Side-by-side mental...

What Topic Is Next → Module 8 Capstone (Multi-Collection Wallet Debit + Certificate Issuance ACID Transaction Engine) → Next Topic 9.1: MongoDB Replica Sets — Primary Elections, Idempotent `oplog.rs`, Heartbeats & Read Preferences

Module 8 Hands-On Capstone Project (100 Marks): Multi-Collection Wallet Debit + Certificate Issuance ACID Transaction Engine — Implement a Replica-Set-safe `session.withTransaction(async () => { ... }, { readConcern: { level: "snapshot" }, writeConcern: { w: "majority" } })` workflow atomically debiting `wallets`, i...

MODULE 9 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: High Availability **Replica Sets** & Horizontal **Sharding** Architecture

Keep production clusters online 24/7 with 3-Node Replica Set automatic elections & idempotent Oplog replication, and scale petabyte write workloads horizontally across Sharded Clusters (`mongos` + Config Servers + Shard Keys).

Objectives: ✓ MongoDB Replica Sets — Primary Elections, Idempotent `oplog.rs`, Heartbeats & Read Preferences — 3+ odd-numbered voting members (`P-S-S`), 2-second heartbeats & ~10–12s automatic Primary election on crash, how `local.oplog.rs` records idempotent state transitions (`$inc: { x: 1 }` stored as `$set: { x: 6 }`!), Change Streams (built on oplog!), and `readPreference` (`primary`, `secondaryPreferred`). • ✓ Horizontal Sharding Architecture — `mongos` Query Routers, CSRS Config Servers & Chunk Balancing — When vertical scaling hits RAM/IOPS ceilings: partitioning collections across multiple **Shard Replica Sets**, stateless **`mongos`** routers caching Config Server (`CSRS`) range metadata, 128 MB logical ranges (**Chunks**), Targeted vs **Scatter-Gather** broadcast queries, and Zone Sharding for GDPR data residency. • ✓ Shard Key Selection — Cardinality, Frequency (Jumbo Chunks) & Non-Monotonic Hashed/Compound Keys — The three pillars of a great Shard Key: **1. High Cardinality** (millions of distinct values, NOT boolean/status!), **2. Low Frequency** (no single value owns 40% of docs creating un-splittable **Jumbo Chunks**!), and **3. Non-Monotonic Change** (ranged `{ _id: 1 }` hammers the last maxKey chunk on ONE shard — use Hashed `{ _id: "hashed" }` or Compound `{ tenantId: 1, createdAt: 1 }`!). • ✓ Build, verify, and ship Module Capstone: Multi-Tenant SaaS Sharding & Replica-Set Read-Scaling Architecture Blueprint.

9.1 MongoDB Replica Sets — Primary Elections, Idempotent `oplog.rs`, Heartbeats & Read Preferences [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Odd voting members (3 or 5) quorum & Idempotent local.oplog.rs • Core Syntax, CLI & Production APIs: rs.initiate(), rs.status(), rs.conf(), readPreference: "secondaryPreferred" • Deep-Dive Mechanics: ~12s automatic Primary failover & P-S-S vs P-S-Arbiter trade-off • Advanced Patterns: ReadPreference primary vs secondaryPreferred & Change Streams real-time CDC • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Replica Set Quorum & Fault-Tolerance Calculator • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for MongoDB Replica Sets

Purpose: To understand and apply MongoDB Replica Sets — Primary Elections, Idempotent `oplog.rs`, Heartbeats & Read Preferences in real MongoDB systems — specifically: 3+ odd-numb...

What Topic Is Next → Next Topic 9.2: Horizontal Sharding Architecture — `mongos` Query Routers, CSRS Config Servers & Chunk Balancing

9.2 Horizontal Sharding Architecture — `mongos` Query Routers, CSRS Config Servers & Chunk Balancing [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Mongos stateless query router & CSRS Config Server Replica Set • Core Syntax, CLI & Production APIs: sh.enableSharding("academy_db"); sh.shardCollection("academy_db.events", { tenantId: 1, _id:... • Deep-Dive Mechanics: Targeted vs Scatter-Gather query & Chunk splitting & background Balancer • Advanced Patterns: Zone Sharding data residency & Unsharded collections live on Primary Shard • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating `mongos` Targeted vs Scatter-Gather Routing Decision • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Horizontal Sharding Architecture

Purpose: To understand and apply Horizontal Sharding Architecture — `mongos` Query Routers, CSRS Config Servers & Chunk Balancing in real MongoDB systems — specifically: When vert...

What Topic Is Next → Next Topic 9.3: Shard Key Selection — Cardinality, Frequency (Jumbo Chunks) & Non-Monotonic Hashed/Compound Keys

9.3 Shard Key Selection — Cardinality, Frequency (Jumbo Chunks) & Non-Monotonic Hashed/Compound Keys [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: High Cardinality + Low Frequency + Non-Monotonic & Monotonic ObjectId insert hotspot • Core Syntax, CLI & Production APIs: sh.shardCollection("db.logs", { userId: "hashed" }) or { orgId: 1, createdAt: 1 } • Deep-Dive Mechanics: Hashed vs Ranged sharding & Jumbo Chunk low-cardinality trap • Advanced Patterns: Compound { tenantId: 1, createdAt: 1 } & ReshardCollection & refineCollectionShardKey • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Evaluating Candidate Shard Keys Against the 3 Golden Criteria • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Shard Key Selection

Purpose: To understand and apply Shard Key Selection — Cardinality, Frequency (Jumbo Chunks) & Non-Monotonic Hashed/Compound Keys in real MongoDB systems — specifically: The three...

What Topic Is Next → Module 9 Capstone (Multi-Tenant SaaS Sharding & Replica-Set Read-Scaling Architecture Blueprint) → Next Topic 10.1: MongoDB Security — `SCRAM-SHA-256`, Least-Privilege RBAC Roles, Encryption at Rest & Blocking NoSQL Injection

Module 9 Hands-On Capstone Project (100 Marks): Multi-Tenant SaaS Sharding & Replica-Set Read-Scaling Architecture Blueprint — Architect a global SaaS learning telemetry cluster using 3-member `P-S-S` Replica Sets per shard, design a compound shard key `{ tenantId: 1, eventDate: 1 }` avoiding monotonic `_id` / `ISOD...

MODULE 10 OF 16 • STAGE 3 • 10 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Security RBAC, `MongoDB Atlas` Cloud Operations, Backups & Node.js Driver Patterns

Lock down MongoDB with SCRAM-SHA-256, Least-Privilege RBAC roles & NoSQL-injection sanitization, operate managed `MongoDB Atlas` clusters + PITR backups, and write leak-free singleton `MongoClient` services in Node.js.

Objectives: ✓ MongoDB Security — `SCRAM-SHA-256`, Least-Privilege RBAC Roles, Encryption at Rest & Blocking NoSQL Injection — Enabling `security.authorization: enabled`, creating database-scoped `readWrite` users via `db.createUser()` (never run apps as `root` / `userAdminAnyDatabase`!), custom roles (`db.createRole`), network IP allowlists, and blocking `{"$ne": null}` / `$where` NoSQL Injection. • ✓ MongoDB Atlas & Production Backups — Multi-AZ Clusters, `mongodump --oplog`, PITR & Atlas Search — Operating managed multi-cloud Atlas clusters (M10+ dedicated VPC peering/PrivateLink, Performance Advisor slow-query index suggestions, Auto-Scaling), `mongodump --oplog --gzip --archive` vs continuous Cloud Backup Point-in-Time Restore (PITR), and Apache Lucene-powered **Atlas Search** / **Atlas Vector Search**. • ✓ Official Node.js `mongodb` Driver in Production — Singleton `MongoClient` Pool, Timeouts & Change Streams — Connecting Node.js to MongoDB cleanly with `new MongoClient(uri, { maxPoolSize: 20, maxIdleTimeMS: 60000, serverSelectionTimeoutMS: 5000 })`, why creating one global `MongoClient` singleton avoids connection-storm exhaustion, `collection.watch()` real-time Change Streams (with `resumeAfter` tokens!), and graceful shutdown. • ✓ Build, verify, and ship Module Capstone: Hardened Node.js + MongoDB Atlas Production Microservice & Change-Stream Notifier.

10.1 MongoDB Security — `SCRAM-SHA-256`, Least-Privilege RBAC Roles, Encryption at Rest & Blocking NoSQL Injection [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: SCRAM-SHA-256 authentication & Least-Privilege readWrite role • Core Syntax, CLI & Production APIs: db.createUser({ user: "maa_api", pwd: passwordPrompt(), roles: [{ role: "readWrite", db: "mo... • Deep-Dive Mechanics: NoSQL $ne / $gt injection defense & Never use $where on user input • Advanced Patterns: VPC Private Endpoint & TLS & AuditLog & Field-Level Encryption • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Sanitizing Untrusted Query Inputs Against `$` Operator Injection • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for MongoDB Security

Purpose: To understand and apply MongoDB Security — `SCRAM-SHA-256`, Least-Privilege RBAC Roles, Encryption at Rest & Blocking NoSQL Injection in real MongoDB systems — specifical...

What Topic Is Next → Next Topic 10.2: MongoDB Atlas & Production Backups — Multi-AZ Clusters, `mongodump --oplog`, PITR & Atlas Search

10.2 MongoDB Atlas & Production Backups — Multi-AZ Clusters, `mongodump --oplog`, PITR & Atlas Search [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Mongodump --oplog consistent snapshot & Atlas Continuous Cloud PITR • Core Syntax, CLI & Production APIs: mongodump --uri="mongodb+srv://..." --oplog --gzip --archive=backup.gz; mongorestore --oplog... • Deep-Dive Mechanics: PrivateLink / VPC Peering vs 0.0.0.0/0 & Performance Advisor slow query alerts • Advanced Patterns: Atlas Search ($search) & $vectorSearch & Regular restore drills • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Atlas Readiness Checklist Validator (`SRV` + Private Network + Majority Write) • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for MongoDB Atlas

Purpose: To understand and apply MongoDB Atlas & Production Backups — Multi-AZ Clusters, `mongodump --oplog`, PITR & Atlas Search in real MongoDB systems — specifically: Operating...

What Topic Is Next → Next Topic 10.3: Official Node.js `mongodb` Driver in Production — Singleton `MongoClient` Pool, Timeouts & Change Streams

10.3 Official Node.js `mongodb` Driver in Production — Singleton `MongoClient` Pool, Timeouts & Change Streams [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Singleton MongoClient connection pool & Never connect() inside request handler • Core Syntax, CLI & Production APIs: const client = new MongoClient(uri, { maxPoolSize: 20 }); const stream = col.watch([{ $match... • Deep-Dive Mechanics: MaxPoolSize & serverSelectionTimeoutMS & Change Streams watch() + resumeAfter • Advanced Patterns: Graceful SIGTERM client.close() & BSON ObjectId string validation • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Validating 24-Hex `ObjectId` Route Params Safely Before Querying • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Official Node.js `mongodb` Driver in Production

Purpose: To understand and apply Official Node.js `mongodb` Driver in Production — Singleton `MongoClient` Pool, Timeouts & Change Streams in real MongoDB systems — specifically:...

What Topic Is Next → Module 10 Capstone (Hardened Node.js + MongoDB Atlas Production Microservice & Change-Stream Notifier) → Next Topic 11.1: Extended Reference, Subset, Outlier & Bucket Design Patterns

Module 10 Hands-On Capstone Project (100 Marks): Hardened Node.js + MongoDB Atlas Production Microservice & Change-Stream Notifier — Build a production-ready Node.js repository layer using a singleton pooled `MongoClient` (`maxPoolSize: 20`, `serverSelectionTimeoutMS: 5000`), strict scalar/schema sanitization blocking `{"...

MODULE 11 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Production Document Schema Patterns & JSON Schema Validation

Production-grade MongoDB Training engineering module covering Production Document Schema Patterns & JSON Schema Validation with hands-on labs and capstone build.

Objectives: ✓ Extended Reference, Subset, Outlier & Bucket Design Patterns — Keep working sets in WiredTiger RAM and avoid unbounded array growth traps. • ✓ Server-Side $jsonSchema Collection Validation Rules — Enforce required fields and data types inside MongoDB regardless of which app writes. • ✓ Polymorphic Collections & Zero-Downtime Schema Versioning — Store diverse product attributes cleanly and evolve document versions incrementally. • ✓ Build, verify, and ship Module Capstone: Multi-Tenant Catalog & Time-Series Telemetry Data Model.

11.1 Extended Reference, Subset, Outlier & Bucket Design Patterns [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Subset • Integration with Server • Live Debugging, Telemetry & Technical Interview Trade • What Is Extended Reference, Subset, Outlier & Bucket Design Patterns & Why Industry Teams Use It • In MongoDB Training, Extended Reference, Subset, Outlier & Bucket Design Patterns exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Extended Reference, Subset, Outlier & Bucket Design Patterns in real MongoDB systems — specifically: Keep working sets in WiredTiger RAM and avoid...

What Topic Is Next → Next Topic 11.2: Server-Side $jsonSchema Collection Validation Rules

11.2 Server-Side $jsonSchema Collection Validation Rules [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Polymorphic Collections & Zero • Live Debugging, Telemetry & Technical Interview Trade • What Is Server • In MongoDB Training, Server • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Server-Side $jsonSchema Collection Validation Rules in real MongoDB systems — specifically: Enforce required fields and data types inside MongoDB...

What Topic Is Next → Next Topic 11.3: Polymorphic Collections & Zero-Downtime Schema Versioning

11.3 Polymorphic Collections & Zero-Downtime Schema Versioning [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Zero • Integration with Extended Reference, Subset, Outlier & Bucket Design Patterns • Live Debugging, Telemetry & Technical Interview Trade • What Is Polymorphic Collections & Zero • In MongoDB Training, Polymorphic Collections & Zero • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Polymorphic Collections & Zero-Downtime Schema Versioning in real MongoDB systems — specifically: Store diverse product attributes cleanly and evo...

What Topic Is Next → Module 11 Capstone (Multi-Tenant Catalog & Time-Series Telemetry Data Model) → Next Topic 12.1: Multi-Stage $match, $lookup, $facet & $setWindowFields

Module 11 Hands-On Capstone Project (100 Marks): Multi-Tenant Catalog & Time-Series Telemetry Data Model — Design, implement, test, and document 'Multi-Tenant Catalog & Time-Series Telemetry Data Model' applying every production concept from Module 11 (Production Document Schema Patterns & JSON S...

MODULE 12 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Advanced Aggregation Pipelines, Facets, Window Stages & Atlas Search

Production-grade MongoDB Training engineering module covering Advanced Aggregation Pipelines, Facets, Window Stages & Atlas Search with hands-on labs and capstone build.

Objectives: ✓ Multi-Stage $match, $lookup, $facet & $setWindowFields — Compute filtered product pages, brand counts, and price buckets in one fast query. • ✓ Pipeline Optimization: Early $match/$project & Index Pushdown — Ensure aggregation pipelines leverage B-Tree indexes instead of scanning every BSON doc. • ✓ Text/Fuzzy Search, Geospatial 2dsphere Queries & Materialized $merge — Power store-locator radius search and pre-aggregated hourly KPI collections. • ✓ Build, verify, and ship Module Capstone: Single-Roundtrip E-Commerce Faceted Search & Revenue Pipeline.

12.1 Multi-Stage $match, $lookup, $facet & $setWindowFields [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of $lookup • Integration with Pipeline Optimization • Live Debugging, Telemetry & Technical Interview Trade • What Is Multi • In MongoDB Training, Multi • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Multi-Stage $match, $lookup, $facet & $setWindowFields in real MongoDB systems — specifically: Compute filtered product pages, brand counts, and p...

What Topic Is Next → Next Topic 12.2: Pipeline Optimization: Early $match/$project & Index Pushdown

12.2 Pipeline Optimization: Early $match/$project & Index Pushdown [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Index Pushdown • Integration with Text/Fuzzy Search, Geospatial 2dsphere Queries & Materialized $merge • Live Debugging, Telemetry & Technical Interview Trade • What Is Pipeline Optimization • In MongoDB Training, Pipeline Optimization • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Pipeline Optimization: Early $match/$project & Index Pushdown in real MongoDB systems — specifically: Ensure aggregation pipelines leverage B-Tree...

What Topic Is Next → Next Topic 12.3: Text/Fuzzy Search, Geospatial 2dsphere Queries & Materialized $merge

12.3 Text/Fuzzy Search, Geospatial 2dsphere Queries & Materialized $merge [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Geospatial 2dsphere Queries • Integration with Multi • Live Debugging, Telemetry & Technical Interview Trade • What Is Text/Fuzzy Search, Geospatial 2dsphere Queries & Materialized $merge & Why Industry Teams Use It • In MongoDB Training, Text/Fuzzy Search, Geospatial 2dsphere Queries & Materialized $merge exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Text/Fuzzy Search, Geospatial 2dsphere Queries & Materialized $merge in real MongoDB systems — specifically: Power store-locator radius search and...

What Topic Is Next → Module 12 Capstone (Single-Roundtrip E-Commerce Faceted Search & Revenue Pipeline) → Next Topic 13.1: Equality–Sort–Range (ESR) Rule, Partial & TTL Expiring Indexes

Module 12 Hands-On Capstone Project (100 Marks): Single-Roundtrip E-Commerce Faceted Search & Revenue Pipeline — Design, implement, test, and document 'Single-Roundtrip E-Commerce Faceted Search & Revenue Pipeline' applying every production concept from Module 12 (Advanced Aggregation Pipelines, Facets...

MODULE 13 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

ESR Compound Indexing, WiredTiger Profiling & Multi-Doc Transactions

Production-grade MongoDB Training engineering module covering ESR Compound Indexing, WiredTiger Profiling & Multi-Doc Transactions with hands-on labs and capstone build.

Objectives: ✓ Equality–Sort–Range (ESR) Rule, Partial & TTL Expiring Indexes — Eliminate 32MB in-memory sort errors and auto-expire OTP/session documents cleanly. • ✓ ExecutionStats Forensics (totalKeysExamined vs nReturned) & Profiler — Spot low-selectivity indexes and slow >100ms queries before CPU saturates. • ✓ Multi-Document ACID Transactions, Read/Write Concerns & Change Streams — Commit cross-collection ledger updates atomically and stream real-time CDC events. • ✓ Build, verify, and ship Module Capstone: Index Audit & ACID Multi-Document Escrow Settlement Engine.

13.1 Equality–Sort–Range (ESR) Rule, Partial & TTL Expiring Indexes

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Partial • Integration with ExecutionStats Forensics (totalKeysExamined vs nReturned) & Profiler • Live Debugging, Telemetry & Technical Interview Trade • What Is Equality–Sort–Range (ESR) Rule, Partial & TTL Expiring Indexes & Why Industry Teams Use It • In MongoDB Training, Equality–Sort–Range (ESR) Rule, Partial & TTL Expiring Indexes exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Equality–Sort–Range (ESR) Rule, Partial & TTL Expiring Indexes in real MongoDB systems — specifically: Eliminate 32MB in-memory sort errors and au...

What Topic Is Next → Next Topic 13.2: ExecutionStats Forensics (totalKeysExamined vs nReturned) & Profiler

13.2 ExecutionStats Forensics (totalKeysExamined vs nReturned) & Profiler

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Profiler • Integration with Multi • Live Debugging, Telemetry & Technical Interview Trade • What Is ExecutionStats Forensics (totalKeysExamined vs nReturned) & Profiler & Why Industry Teams Use It • In MongoDB Training, ExecutionStats Forensics (totalKeysExamined vs nReturned) & Profiler exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply ExecutionStats Forensics (totalKeysExamined vs nReturned) & Profiler in real MongoDB systems — specifically: Spot low-selectivity indexes and slow...

What Topic Is Next → Next Topic 13.3: Multi-Document ACID Transactions, Read/Write Concerns & Change Streams

13.3 Multi-Document ACID Transactions, Read/Write Concerns & Change Streams

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Read/Write Concerns • Integration with Equality–Sort–Range (ESR) Rule, Partial & TTL Expiring Indexes • Live Debugging, Telemetry & Technical Interview Trade • What Is Multi • In MongoDB Training, Multi • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Multi-Document ACID Transactions, Read/Write Concerns & Change Streams in real MongoDB systems — specifically: Commit cross-collection ledger upda...

What Topic Is Next → Module 13 Capstone (Index Audit & ACID Multi-Document Escrow Settlement Engine) → Next Topic 14.1: Replica Set Elections, Oplog Sizing & Retryable Reads/Writes

Module 13 Hands-On Capstone Project (100 Marks): Index Audit & ACID Multi-Document Escrow Settlement Engine — Design, implement, test, and document 'Index Audit & ACID Multi-Document Escrow Settlement Engine' applying every production concept from Module 13 (ESR Compound Indexing, WiredTiger Profili...

MODULE 14 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Replica Set High Availability, Sharding Keys, RBAC & Backup Ops

Production-grade MongoDB Training engineering module covering Replica Set High Availability, Sharding Keys, RBAC & Backup Ops with hands-on labs and capstone build.

Objectives: ✓ Replica Set Elections, Oplog Sizing & Retryable Reads/Writes — Survive primary node hardware crashes with automatic ~2-second failover. • ✓ Shard Key Selection (Cardinality, Frequency & Monotonic Pitfalls) — Avoid jumbo chunks and hot-shard bottlenecks when scaling past terabytes. • ✓ SCRAM/TLS Encryption, Least-Privilege Roles & Point-in-Time Restore — Lock down production clusters against unauthorized network or credential access. • ✓ Build, verify, and ship Module Capstone: Horizontally Sharded Multi-Region MongoDB Cluster Blueprint.

14.1 Replica Set Elections, Oplog Sizing & Retryable Reads/Writes

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Oplog Sizing • Integration with Shard Key Selection (Cardinality, Frequency & Monotonic Pitfalls) • Live Debugging, Telemetry & Technical Interview Trade • What Is Replica Set Elections, Oplog Sizing & Retryable Reads/Writes & Why Industry Teams Use It • In MongoDB Training, Replica Set Elections, Oplog Sizing & Retryable Reads/Writes exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Replica Set Elections, Oplog Sizing & Retryable Reads/Writes in real MongoDB systems — specifically: Survive primary node hardware crashes with au...

What Topic Is Next → Next Topic 14.2: Shard Key Selection (Cardinality, Frequency & Monotonic Pitfalls)

14.2 Shard Key Selection (Cardinality, Frequency & Monotonic Pitfalls) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Frequency • Integration with SCRAM/TLS Encryption, Least • Live Debugging, Telemetry & Technical Interview Trade • What Is Shard Key Selection (Cardinality, Frequency & Monotonic Pitfalls) & Why Industry Teams Use It • In MongoDB Training, Shard Key Selection (Cardinality, Frequency & Monotonic Pitfalls) exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply Shard Key Selection (Cardinality, Frequency & Monotonic Pitfalls) in real MongoDB systems — specifically: Avoid jumbo chunks and hot-shard bottlen...

What Topic Is Next → Next Topic 14.3: SCRAM/TLS Encryption, Least-Privilege Roles & Point-in-Time Restore

14.3 SCRAM/TLS Encryption, Least-Privilege Roles & Point-in-Time Restore [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Least • Integration with Replica Set Elections, Oplog Sizing & Retryable Reads/Writes • Live Debugging, Telemetry & Technical Interview Trade • What Is SCRAM/TLS Encryption, Least • In MongoDB Training, SCRAM/TLS Encryption, Least • Even if you come from a non • Runnable Code Implementation (JAVASCRIPT)

Purpose: To understand and apply SCRAM/TLS Encryption, Least-Privilege Roles & Point-in-Time Restore in real MongoDB systems — specifically: Lock down production clusters against...

What Topic Is Next → Module 14 Capstone (Horizontally Sharded Multi-Region MongoDB Cluster Blueprint) → Next Topic 15.1: JSON Schema Validation Rules, Polymorphic Patterns & Avoiding Unbounded Arrays

Module 14 Hands-On Capstone Project (100 Marks): Horizontally Sharded Multi-Region MongoDB Cluster Blueprint — Design, implement, test, and document 'Horizontally Sharded Multi-Region MongoDB Cluster Blueprint' applying every production concept from Module 14 (Replica Set High Availability, Sharding...

MODULE 15 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

MongoDB Schema Governance, Time-Series / Bucketing & Change Streams

Self-explanatory list-and-bullet production mastery module covering MongoDB Schema Governance, Time-Series / Bucketing & Change Streams in MongoDB Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ JSON Schema Validation Rules, Polymorphic Patterns & Avoiding Unbounded Arrays — Purpose: Enforce document data quality at the database layer while preventing 16MB BSON growth traps. • ✓ Extended Reference, Subset & Bucket Patterns for High-Volume Workloads — Purpose: Cut WiredTiger working-set RAM usage and disk IOPS on activity feeds and telemetry streams. • ✓ Resumable MongoDB Change Streams & Multi-Document ACID Transaction Boundaries — Purpose: React to live data mutations reliably using resume tokens and keep financial writes atomic. • ✓ Build, verify, and document 'Schema-Validated IoT/Telemetry Bucket Store & Real-Time Change-Stream Consumer' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 JSON Schema Validation Rules, Polymorphic Patterns & Avoiding Unbounded Arrays [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for JSON Schema Validation Rules, Polymorphic Patterns & Avoiding Unbounded Ar... • Performance Optimization, Complexity & Latency Impact of JSON Schema Validation Rules, Polymorphic Patterns & Avoid... • Live Debugging, Stack • Runnable Code Implementation (JAVASCRIPT) • Hands • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for JSON Schema Validation Rules • Production Hardening & Security: $jsonSchema validation, operator sanitization & SCRAM/TLS RBAC

Purpose: To understand and apply JSON Schema Validation Rules, Polymorphic Patterns & Avoiding Unbounded Arrays in real MongoDB systems — specifically: Enforce document data quali...

What Topic Is Next → Next Topic 15.2: Extended Reference, Subset & Bucket Patterns for High-Volume Workloads

15.2 Extended Reference, Subset & Bucket Patterns for High-Volume Workloads [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Extended Reference, Subset & Bucket Patterns for High • Performance Optimization, Complexity & Latency Impact of Extended Reference, Subset & Bucket Patterns for High • Live Debugging, Stack • Runnable Code Implementation (JAVASCRIPT) • Hands • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Extended Reference • Production Hardening & Security: $jsonSchema validation, operator sanitization & SCRAM/TLS RBAC

Purpose: To understand and apply Extended Reference, Subset & Bucket Patterns for High-Volume Workloads in real MongoDB systems — specifically: Cut WiredTiger working-set RAM usag...

What Topic Is Next → Next Topic 15.3: Resumable MongoDB Change Streams & Multi-Document ACID Transaction Boundaries

15.3 Resumable MongoDB Change Streams & Multi-Document ACID Transaction Boundaries [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Resumable MongoDB Change Streams & Multi • Performance Optimization, Complexity & Latency Impact of Resumable MongoDB Change Streams & Multi • Live Debugging, Stack • Runnable Code Implementation (JAVASCRIPT) • Hands • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Resumable MongoDB Change Streams • Production Hardening & Security: $jsonSchema validation, operator sanitization & SCRAM/TLS RBAC

Purpose: To understand and apply Resumable MongoDB Change Streams & Multi-Document ACID Transaction Boundaries in real MongoDB systems — specifically: React to live data mutations...

What Topic Is Next → Module 15 Capstone (Schema-Validated IoT/Telemetry Bucket Store & Real-Time Change-Stream Consumer) → Next Topic 16.1: Equality-Sort-Range (ESR) Index Rule, Covered Queries & COLLSCAN Elimination

Module 15 Hands-On Capstone Project (100 Marks): Schema-Validated IoT/Telemetry Bucket Store & Real-Time Change-Stream Consumer — Architect, implement, test, and document 'Schema-Validated IoT/Telemetry Bucket Store & Real-Time Change-Stream Consumer' applying every bullet-wise engineering concept from Module 15 (Mongo...

MODULE 16 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

MongoDB Atlas/Production Security, ESR Index Audits & NoSQL System Design

Self-explanatory list-and-bullet production mastery module covering MongoDB Atlas/Production Security, ESR Index Audits & NoSQL System Design in MongoDB Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Equality-Sort-Range (ESR) Index Rule, Covered Queries & COLLSCAN Elimination — Purpose: Turn multi-second collection scans into single-millisecond IXSCAN lookups verified via explain("executionStats"). • ✓ Shard Key Cardinality/Frequency Analysis, Hotspot Prevention & Backup Restore — Purpose: Choose shard keys that distribute writes evenly without scatter-gather query penalties. • ✓ SQL vs MongoDB Trade-Offs & Real-World NoSQL Data Modeling Interviews — Purpose: Justify embedding vs referencing and consistency/read-concern choices in technical interviews. • ✓ Build, verify, and document 'ESR Compound Index Optimizer, Profiler Audit & Sharding Key Blueprint' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 Equality-Sort-Range (ESR) Index Rule, Covered Queries & COLLSCAN Elimination [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Equality • Performance Optimization, Complexity & Latency Impact of Equality • Live Debugging, Stack • Runnable Code Implementation (JAVASCRIPT) • Hands • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Equality-Sort-Range (ESR) Index Rule • Production Hardening & Security: $jsonSchema validation, operator sanitization & SCRAM/TLS RBAC

Purpose: To understand and apply Equality-Sort-Range (ESR) Index Rule, Covered Queries & COLLSCAN Elimination in real MongoDB systems — specifically: Turn multi-second collection...

What Topic Is Next → Next Topic 16.2: Shard Key Cardinality/Frequency Analysis, Hotspot Prevention & Backup Restore

16.2 Shard Key Cardinality/Frequency Analysis, Hotspot Prevention & Backup Restore [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Shard Key Cardinality/Frequency Analysis, Hotspot Prevention & Backup Rest... • Performance Optimization, Complexity & Latency Impact of Shard Key Cardinality/Frequency Analysis, Hotspot Preventi... • Live Debugging, Stack • Runnable Code Implementation (JAVASCRIPT) • Hands • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for Shard Key Cardinality/Frequency Analysis • Production Hardening & Security: $jsonSchema validation, operator sanitization & SCRAM/TLS RBAC

Purpose: To understand and apply Shard Key Cardinality/Frequency Analysis, Hotspot Prevention & Backup Restore in real MongoDB systems — specifically: Choose shard keys that distr...

What Topic Is Next → Next Topic 16.3: SQL vs MongoDB Trade-Offs & Real-World NoSQL Data Modeling Interviews

16.3 SQL vs MongoDB Trade-Offs & Real-World NoSQL Data Modeling Interviews [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for SQL vs MongoDB Trade • Performance Optimization, Complexity & Latency Impact of SQL vs MongoDB Trade • Live Debugging, Stack • Runnable Code Implementation (JAVASCRIPT) • Hands • Idiomatic MONG Implementation: bounded document embedding, ESR compound indexes & Aggregation Pipelines • MongoDB 7 WiredTiger Storage Engine, BSON Documents & Oplog Replica Set Execution & Memory Lifecycle for SQL vs MongoDB Trade-Offs • Production Hardening & Security: $jsonSchema validation, operator sanitization & SCRAM/TLS RBAC

Purpose: To understand and apply SQL vs MongoDB Trade-Offs & Real-World NoSQL Data Modeling Interviews in real MongoDB systems — specifically: Justify embedding vs referencing and...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: CatalogSphere Polymorphic Product Catalog, Search & Real-Time Event Store

Module 16 Hands-On Capstone Project (100 Marks): ESR Compound Index Optimizer, Profiler Audit & Sharding Key Blueprint — Architect, implement, test, and document 'ESR Compound Index Optimizer, Profiler Audit & Sharding Key Blueprint' applying every bullet-wise engineering concept from Module 16 (MongoDB Atlas/...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: CatalogSphere Polymorphic Product Catalog, Search & Real-Time Event Store • **Confirmed Role:** MongoDB NoSQL Data Modeling & Aggregation Intern

Domain & Production Infra: Omnichannel Marketplace & IoT Telemetry Client (MongoDB 7 Replica Set + Atlas Search + Compound/TTL Indexes + Change Streams)

- Modeled high-read polymorphic product documents using embedding vs referencing patterns and JSON Schema validation.
- Built multi-stage Aggregation Pipelines ($match, $lookup, $facet, $group) powering sub-40ms storefront facets.

- Configured compound, partial, text, and TTL indexes verified via explain("executionStats") plans.
- Implemented multi-document ACID transactions and Change Stream listeners on a live MongoDB replica set.

Target Job Roles: MongoDB / NoSQL Developer, Backend Data Layer Engineer, MERN Database Specialist, Associate Cloud Data Engineer

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 LPA – ₹8.2 LPA **Mid-Level NoSQL / Backend Engineer (1–3 Yrs):** ₹9.0 LPA – ₹17.5 LPA **Senior Full-Stack / Data Platform Architect (3–6+ Yrs):** ₹18.0 LPA – ₹34.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://www.mockattempt.com/courses/mongodb-training

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.