Module 4A - Database Integrations using MongoDB

MongoDB Schema Design for Real Systems

Warm-up Quiz

Warm-up

- 8 questions, multiple choice
- Reply format: 1 / 2 / 3 / 4
- Topics
  - Search and filters
  - Matching rules
  - Pagination
  - Sorting
  - Recap: MongoDB schema

Warm-up Q1: Filters

- Which request applies two filters at the same time?

- 1. GET /students?name=karthik&email=karthik.rao@gmail.com
- 2. GET /students?page=0&size=5
- 3. GET /students?sortBy=name&sortDir=asc
- 4. GET /students/18

- Reply format: 1 / 2 / 3 / 4

Answer 1

- Which request applies two filters at the same time?

- 1. GET /students?name=karthik&email=karthik.rao@gmail.com
- 2. GET /students?page=0&size=5
- 3. GET /students?sortBy=name&sortDir=asc
- 4. GET /students/18

- Correct answer: 1
- Why: name and email are both filter inputs. The others are pagination, sorting, or path-based access.

Warm-up Q2: Matching rule

- We said the name filter uses contains and is case-insensitive. Which option correctly describes what should match name=an?

- 1. Only Ananya Menon
- 2. Only Sanjana
- 3. Ananya Menon and Sanjana
- 4. None

- Reply format: 1 / 2 / 3 / 4

Answer 2

- We said the name filter uses contains and is case-insensitive. Which option correctly describes what should match name=an?

- 1. Only Ananya Menon
- 2. Only Sanjana
- 3. Ananya Menon and Sanjana
- 4. None

- Correct answer: 3
- Why: an appears inside both Ananya and Sanjana. Case does not matter.

Warm-up Q3: AND logic

- If the API uses AND logic when multiple filters are present, what is true?

- 1. A student matches if name matches OR email matches
- 2. A student matches only if name matches AND email matches
- 3. Filters are ignored if both are present
- 4. Email filter runs only if name filter is missing

- Reply format: 1 / 2 / 3 / 4

Answer 3

- If the API uses AND logic when multiple filters are present, what is true?

- 1. A student matches if name matches OR email matches
- 2. A student matches only if name matches AND email matches
- 3. Filters are ignored if both are present
- 4. Email filter runs only if name filter is missing

- Correct answer: 2
- Why: AND logic means all provided conditions must pass.

Warm-up Q4: Empty filters

- How should the API treat this request?

- GET /students?name=&email=gmail.com

- 1. Treat name= as a filter for empty names
- 2. Treat name= as “not provided” and filter only by email
- 3. Reject the request with 400
- 4. Return zero students always

- Reply format: 1 / 2 / 3 / 4

Answer 4

- How should the API treat this request?

- GET /students?name=&email=gmail.com

- 1. Treat name= as a filter for empty names
- 2. Treat name= as “not provided” and filter only by email
- 3. Reject the request with 400
- 4. Return zero students always

- Correct answer: 2
- Why: an empty filter value should behave like the filter is not provided, so clients do not accidentally wipe results.

Warm-up Q5: Pagination meaning

- What does page=2&size=5 mean?

- 1. Return 2 students, each with 5 fields
- 2. Return the 2nd page, with 5 students per page
- 3. Return the 3rd page, with 5 students per page
- 4. Return students with id between 2 and 5

- Reply format: 1 / 2 / 3 / 4

Answer 5

- What does page=2&size=5 mean?

- 1. Return 2 students, each with 5 fields
- 2. Return the 2nd page, with 5 students per page
- 3. Return the 3rd page, with 5 students per page
- 4. Return students with id between 2 and 5

- Correct answer: 2
-

Warm-up Q6: Sorting inputs

- Which request sorts by email descending using our contract?

- 1. GET /students?sortBy=email&sortDir=desc
- 2. GET /students?sort=email,desc
- 3. GET /students?sortDir=email&sortBy=desc
- 4. GET /students?email=desc

- Reply format: 1 / 2 / 3 / 4

Answer 6

- Which request sorts by email descending using our contract?

- 1. GET /students?sortBy=email&sortDir=desc
- 2. GET /students?sort=email,desc
- 3. GET /students?sortDir=email&sortBy=desc
- 4. GET /students?email=desc

- Correct answer: 1
- Why: sortBy chooses the field, sortDir chooses the direction.

Warm-up Q7: MongoDB recall

- Which is the best description of an embedded document in MongoDB?

- 1. A document stored in a different collection and linked by id
- 2. A document stored inside another document as a nested object or array
- 3. A document that cannot have an _id field
- 4. A document that must be stored only in Atlas

- Reply format: 1 / 2 / 3 / 4

Answer 7

- Which is the best description of an embedded document in MongoDB?

- 1. A document stored in a different collection and linked by id
- 2. A document stored inside another document as a nested object or array
- 3. A document that cannot have an _id field
- 4. A document that must be stored only in Atlas

- Correct answer: 2
- Why: embedded means nested inside the parent document, commonly as an object or array.

Warm-up Q8: MongoDB schema thinking

- You want to store a student and their addresses. A student can have multiple addresses, and addresses are always shown when viewing a student profile. Best fit?

- 1. Store addresses in a separate addresses collection and reference address ids
- 2. Store addresses as an embedded array inside the student document
- 3. Store addresses as plain strings in a single comma-separated field
- 4. Store addresses only in application memory, not in MongoDB

- Reply format: 1 / 2 / 3 / 4

Answer 8

- You want to store a student and their addresses. A student can have multiple addresses, and addresses are always shown when viewing a student profile. Best fit?

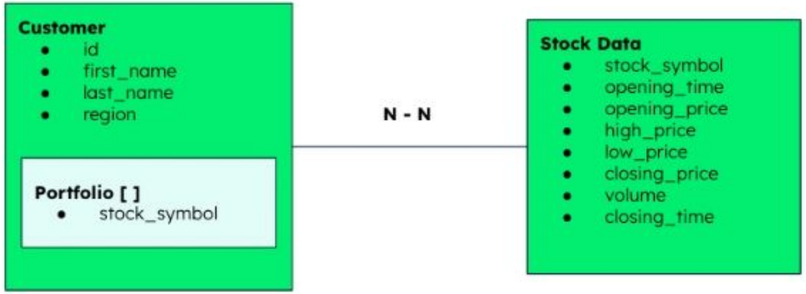
- 1. Store addresses in a separate addresses collection and reference address ids
- 2. Store addresses as an embedded array inside the student document
- 3. Store addresses as plain strings in a single comma-separated field
- 4. Store addresses only in application memory, not in MongoDB

- Correct answer: 2
- Why: addresses are tightly owned by the student and commonly fetched together, so embedding as an array is a clean fit.

Section 3A - Schema Design Mindset

What schema design means in MongoDB

- MongoDB *does not* force a rigid schema, but your application still depends on structure.
- Schema design means choosing:
  - what fields a document contains
  - which data is embedded vs stored separately
  - how relationships are represented
- A good schema makes common reads and writes easy, and reduces accidental corruption.



Baseline student document shape

- We will model one student as one document in the students collection.
- Fields we will rely on in examples:
    - _id is the unique identity for the document
    - name and email are searchable fields
    - active supports filter use cases

```
{
  "_id": "65b7e1d2c4f3a12b9e8d77a1",
  "name": "Aditi Sharma",
  "email": "aditi.sharma@gmail.com",
  "active": true
}
```

Schema design principles that matter in real systems

- Design for your most common queries.
  - Store together what you read together most of the time.
- Avoid duplication unless it buys you real query speed.
  - Duplication increases update risk.
- Prefer bounded growth inside a document.
  - A small list is fine, an unbounded list becomes painful
- Optimize for change.
  - Expect new fields later, but keep the meaning stable for existing fields.

A decision tool to reuse

- When deciding embed vs reference, ask:
  - Do I need this data every time I fetch the parent?
  - Will this data change frequently and must stay consistent everywhere?
  - Can this list grow without a clear limit?
  - Do I need to query this data independently, without the parent?
- If the answers trend toward always together and small, embed is usually better.
- If the answers trend toward shared and queried separately, reference is usually better.

Section 3B - Relationships and Modeling Choices

Relationships in MongoDB

- Real applications have relationships, even in NoSQL.
- MongoDB gives multiple representation options:
    - Embedding
    - References
    - Relationship documents
- The best choice depends usually on:
    - Query patterns
        - What do we fetch most often (student with enrollments, course with students, etc.)?
        - Ex: Profile page always shows student + enrollments. So, query enrollments by studentId
    - Update patterns
        - What changes often, and must stay consistent everywhere?
        - Ex:: Course title changes. So, keep course in courses, reference courseId from enrollments (avoid embedding course details in every student).
    - Growth patterns
        - Will this related data stay small and bounded, or can it grow without limit?
        - Ex: A course can have thousands of enrollments. So, store as enrollments collection, not as a giant embedded list

Relationship shapes in the student domain

- Entities: Student, Course, Enrollment.
- We will model:
  - students collection holds student identity and searchable fields
  - courses collection holds course catalog data
  - enrollments collection holds the relationship between students and courses
- This keeps relationships explicit and queryable.

Consider: One student enrolled in many courses

- Consider a scenario where we need to store the detail of a student enrolled in several courses.
- Two options:
  - Array of course ids inside student:
    - fast to list courses for a student
    - hard when enrollment needs extra fields like status, score, enrolledAt
  - Enrollment as its own collection:
    - best for rich relationships and reporting
    - one more query step, but cleaner long-term

Option 1: Embed course ids inside student

- Use when the relationship is simple and you only need the course identity list.
- Easy query to manage: show all course ids for a student.
- Hard changes to make later: add fields for enrolledAt, progress & status; perform reporting by course.

```
Collection: students
{
  "_id": "65b7e1d2c4f3a12b9e8d77a1",
  "name": "Aditi Sharma",
  "email": "aditi.sharma@gmail.com",
  "active": true,
  "courseIds": [
    "65b7e1d2c4f3a12b9e8d77b9",
    "65b7e1d2c4f3a12b9e8d77c3"
  ]
}
```

Option 2: Use an enrollments collection

- Use when the relationship needs its own fields and must be queried independently.
- Easy query to manage: list all students in a course, filter by status, track progress.

```
Collection: enrollments
{
  "_id": "65b7f9a0d4a8c91f2a3b1111",
  "studentId": "65b7e1d2c4f3a12b9e8d77a1",
  "courseId": "65b7e1d2c4f3a12b9e8d77b9",
  "status": "ACTIVE",
  "enrolledAt": "2026-01-25"
}
```

Quick check

- Where tracking the progress of a course, where should it live?
    - 1. Inside the students document
    - 2. Inside the courses document
    - 3. Inside an enrollments document
    - 4. As a random field in both student and course

- Answer: 3
- Why: Course progress is relationship data. It’s not a property of the student alone or the course alone. It’s “this student’s progress in this specific course”

Section 3C - Embedded vs References

Embedded documents

- Embedding means storing related data inside the same document.
- Best when:
  - the embedded data is used whenever the parent is fetched
  - the embedded data is small and bounded
  - the embedded data does not need independent queries
- Example: a student profile contains a small, fixed set of addresses.

References

- References mean storing only ids, and keeping the full data in another collection.
- Best when:
    - the referenced data is shared across many parents
    - the referenced data changes and must remain consistent everywhere
    - you need to query the referenced data independently
- Example: courses are shared across many students, so store courses separately.

Embedded vs references: a comparison

- Embed when:
  - You fetch one student and show this data every time
  - The list is small and will not explode in size
- Embed is usually right when the data is tightly owned, read together, and stays small.
- Example: student contactPreferences displayed on every profile view.
- Reference when:
  - the data is shared across many documents
  - the data must be updated in one place
  - you need independent queries over that data
- Reference is usually right when the data is shared, updated independently, or queried on its own.
- Example: course details live in courses and are referenced from enrollments.

Quick check

- Scenario: You need to show a student’s current enrollments with
    - course name
    - status
    - enrolled date
    - progress percentage
- Pick the best modeling approach:
    - 1. Store full course objects inside students
    - 2. Store only course ids in students
    - 3. Use an enrollments collection that references student and course
    - 4. Store everything in one giant document per student that grows forever

- Answer: 3
- Why: Enrollment has its own fields and must be queryable and maintainable over time.

Section 3D - Apply the Model to the Student Domain

The student domain we will model

- We will model three collections:
    - students stores student identity and searchable fields
    - courses stores course catalog data
    - enrollments stores the relationship between students and courses
- This supports:
    - student search,
    - course listing
    - enrollment tracking, and
    - progress.

Collections shape

- One document per course in the courses collection.

```
{
  "_id": "65b7e1d2c4f3a12b9e8d77b9",
  "title": "Spring Boot Fundamentals",
  "level": "BEGINNER"
}
```

- One document per student in the students collection.

```
{
  "_id": "65b7e1d2c4f3a12b9e8d77a1",
  "name": "Aditi Sharma",
  "email": "aditi.sharma@gmail.com",
  "active": true
}
```

- One document per enrollment in the enrollments collection.

```
{
  "_id": "65b7f9a0d4a8c91f2a3b1111",
  "studentId": "65b7e1d2c4f3a12b9e8d77a1",
  "courseId": "65b7e1d2c4f3a12b9e8d77b9",
  "status": "ACTIVE",
  "enrolledAt": "2026-01-25"
}
```

Model Design on Queries

- What queries does this design makes easy?
    - List a student’s enrollments
        - Query: find enrollments by studentId
    - List all students in a course
        - Query: find enrollments by courseId
    - Filter enrollments by status
        - Query: find enrollments by status
    - Join-like behavior at the application layer
        - Query: fetch courses by courseId list when needed

Quick check

- Query: Find all students enrolled in course 65b7e1d2c4f3a12b9e8d77b9 with status ACTIVE.
- Where does the query start?
  - ○ 1. students
  - ○ 2. courses
  - ○ 3. enrollments
  - ○ 4. It cannot be done in MongoDB

- Answer: 3
- Why: The filter conditions (courseId and status) live on the relationship, so the query must start in enrollments

Section 4 - Schema Choice Drills

Micro-activity: Introduction

- We will practice choosing where data should live in MongoDB.
- Assume the domain collections are: **students, courses, enrollments**.
- Decision
  - First ask: is this data owned by the parent?
  - Then ask: is it shared across many parents?
  - Then ask: does the relationship need its own fields?
  - If the relationship has fields, default to a relationship collection.
- For each requirement, pick the best modeling choice.
  - Choice 1: Embed inside the parent document
  - Choice 2: Reference by id and fetch from another collection
  - Choice 3: Relationship collection for links with their own fields
  - Choice 4: Unsure
- Reply format: 1 / 2 / 3 / 4

Drill 1

- Requirement
  - A student stores only the current mentorId, and mentor details live separately (name, email, experience, etc.).
- Where does the query start?
  - 1. Embed
  - 2. Reference
  - 3. Relationship collection
  - 4. Unsure

- Answer: 2
- Why: Mentor details are independent data and can be reused across many students, so store the mentor in its own collection and reference it from the student.

Drill 2

- Requirement
    - A course can have thousands of students. A student can enroll in many courses. We must store enrolledAt and progress per enrollment
- Where does the query start?
    - 1. Embed
    - 2. Reference
    - 3. Relationship collection
    - 4. Unsure

- Answer: 3
- Why: This is many-to-many, and the relationship has its own fields (enrolledAt, progress), so it belongs in a relationship collection (enrollments).

Drill 3

- Requirement
  - Store the last 5 login timestamps for each student. It is only used inside the student profile.
- Where does the query start?
  - 1. Embed
  - 2. Reference
  - 3. Relationship collection
  - 4. Unsure

- Answer: 1
- Why: This is a small, bounded list that belongs to the student and is read with the student profile, so embed it (array inside the student document)

Section 5 - Bridge Back to Spring Boot

Where does MongoDB fits in our Spring Boot project?

- We are moving from an in-memory repository to a real database.
- We keep the same layers:
  - Controller handles HTTP input and output
  - Service holds business rules and validation
  - Repository handles storage and retrieval
- MongoDB replaces the store students in a list or map logic.
- Spring Data MongoDB is Spring’s integration for working with MongoDB.
  - It provides:
    - mapping between Java objects and MongoDB documents
    - repository interfaces for common database operations
    - query helpers so you do not write raw Mongo queries for every case

MongoRepository: An Introduction

- MongoRepository is a Spring Data interface for database operations.
- You define it as follows: MongoRepository<Student, String>
- What the generics mean:
  - Student is the document type stored in the collection
  - String is the id type
    - MongoDB _id is an ObjectId in MongoDB
    - In our Java model we commonly represent it as String (simple to get started)
- Why is MongoRespository is useful?
  - Spring can generate standard CRUD behavior automatically for your document type
  - Without writing implementation code, you get methods like:
    - save
    - findAll
    - findById
    - deleteById
  - This keeps your repository layer small and readable.

MongoDB-specific Annotations

- **@Document**
    - maps a Java class to a MongoDB collection.
    - Example: @Document(collection = "students")
- **@Id**
    - marks the field that maps to MongoDB _id.

What are we going to do next?

- We’ll wire up our Spring Boot project to use a local MongoDB server.
- What stays the same:
  - API endpoints and controller inputs
  - service rules like validation and error handling
- What changes:
  - repository becomes MongoDB-backed
  - data survives application restarts

Quick check

- Which layer talks to MongoDB
  - 1. Controller
  - 2. Service
  - 3. Repository
  - 4. GlobalExceptionHandler

- Answer: 3
- Why: Controllers handle HTTP, services handle rules, repositories handle storage

That’s a wrap!

Key takeaways

- MongoDB schema design is about making common reads and writes predictable and safe.
- Embed for owned, small, always-read-together data.
- Reference for shared, independently changing, independently queried data.
- Use relationship collections when the link has its own fields or is truly many-to-many.