Lecture 53

Database Integrations Using MongoDB

Listing, search, indexes, and data modeling

Today's objective and agenda

Objective

- Enable listing and search for students using MongoDB.
- Make index behavior visible using concrete examples.
- Model courses and enrollments using references.

Agenda

- Warm-up quiz.
- Live coding: listing and search.
- Activity: courses and enrollments.
- Indexes in practice.
- Embedded documents and references.

Warm-up

Warm-up quiz

- Answer eight multiple choice questions.
- Reply format: 1 / 2 / 3 / 4

Question 1

In a Spring Boot app, what are the two main inputs needed to connect Spring Data MongoDB to a database?

1) A Dockerfile and a MongoDB Compass connection
2) A MongoDB starter dependency and a MongoDB URI in application properties
3) A global exception handler and a seeded dataset
4) A controller annotation and a repository implementation class

Reply format: 1 / 2 / 3 / 4

Answer 1

In a Spring Boot app, what are the two main inputs needed to connect Spring Data MongoDB to a database?

1) A Dockerfile and a MongoDB Compass connection
2) A MongoDB starter dependency and a MongoDB URI in application properties
3) A global exception handler and a seeded dataset
4) A controller annotation and a repository implementation class

Correct answer: 2
Why: The starter enables Mongo support, and the URI tells Spring where the database is and which database name to use.

Reply format: 1 / 2 / 3 / 4

Question 2

What is the main job of Spring MVC in our application?

1) It stores the data in MongoDB collections.
2) It maps incoming HTTP requests to controller methods and produces HTTP responses.
3) It builds the database indexes at runtime for all collections.
4) It generates test data automatically during application startup.

Reply format: 1 / 2 / 3 / 4

Answer 2

What is the main job of Spring MVC in our application?

1) It stores the data in MongoDB collections.
2) It maps incoming HTTP requests to controller methods and produces HTTP responses.
3) It builds the database indexes at runtime for all collections.
4) It generates test data automatically during application startup.

Correct answer: 2
Why: Spring MVC routes requests to controllers and helps build the response, while repositories handle database persistence.

Reply format: 1 / 2 / 3 / 4

Question 3

What does the DispatcherServlet do in a Spring MVC application?

1) It inserts documents into MongoDB when the app starts.
2) It creates unique indexes automatically without configuration.
3) It replaces the need for service classes in the application.
4) It is the front controller that routes requests to the correct controller method.

Reply format: 1 / 2 / 3 / 4

Answer 3

What does the DispatcherServlet do in a Spring MVC application?

1) It inserts documents into MongoDB when the app starts.
2) It creates unique indexes automatically without configuration.
3) It replaces the need for service classes in the application.
4) It is the front controller that routes requests to the correct controller method.

Correct answer: 4
Why: DispatcherServlet is the front controller that receives requests first and forwards them to the right controller handler.

Reply format: 1 / 2 / 3 / 4

Question 4

Which repository method best represents pagination support when listing students?

1) Page<Student> findAll(Pageable pageable)
2) Optional<Student> findById(String id)
3) void deleteAll()
4) List<Student> findAll()

Reply format: 1 / 2 / 3 / 4

Answer 4

Which repository method best represents pagination support when listing students?

1) Page<Student> findAll(Pageable pageable)
2) Optional<Student> findById(String id)
3) void deleteAll()
4) List<Student> findAll()

Correct answer: 1
Why: Pageable carries page, size, and sort, and Page returns page content plus metadata such as total pages and total items.

Reply format: 1 / 2 / 3 / 4

Question 5

Which derived query method name follows Spring Data naming rules and filters by a boolean field?

1) findByActiveTrue()
2) searchStudentsWhereActiveIsTrue()
3) getActiveStudents()
4) findStudentsActiveTrue()

Reply format: 1 / 2 / 3 / 4

Answer 5

Which derived query method name follows Spring Data naming rules and filters by a boolean field?

1) findByActiveTrue()
2) searchStudentsWhereActiveIsTrue()
3) getActiveStudents()
4) findStudentsActiveTrue()

Correct answer: 1
Why: Spring Data parses method names that follow the findBy + field + operator pattern.

Reply format: 1 / 2 / 3 / 4

Question 6

When fetching a student by id, what is the cleanest way for the repository layer to represent "not found"?

1) Return null and let the controller handle it.
2) Return Optional<Student> and let the service decide how to respond.
3) Return an empty string and let the service interpret it.
4) Return ResponseEntity<Student> from the repository.

Reply format: 1 / 2 / 3 / 4

Answer 6

When fetching a student by id, what is the cleanest way for the repository layer to represent "not found"?

1) Return null and let the controller handle it.
2) Return Optional<Student> and let the service decide how to respond.
3) Return an empty string and let the service interpret it.
4) Return ResponseEntity<Student> from the repository.

Correct answer: 2
Why: Optional cleanly represents presence or absence, and the service converts absence into a consistent API response such as 404.

Reply format: 1 / 2 / 3 / 4

Question 7

When two requests try to create the same email at nearly the same time, which layer provides the final guarantee that duplicates cannot be stored?

1) Controller validation annotations
2) A service-layer pre-check only
3) The database unique index
4) MongoDB Compass

Reply format: 1 / 2 / 3 / 4

Answer 7

When two requests try to create the same email at nearly the same time, which layer provides the final guarantee that duplicates cannot be stored?

1) Controller validation annotations
2) A service-layer pre-check only
3) The database unique index
4) MongoDB Compass

Correct answer: 3
Why: A unique index is enforced by MongoDB during the write, even when timing causes two requests to pass service checks.

Reply format: 1 / 2 / 3 / 4

Question 8

Which design best fits these two requirements?
Requirement A: a student has address fields.
Requirement B: an enrollment links one student to one course.

1) Store enrollments as separate documents that keep studentId and courseId.
2) Store everything in one collection and distinguish using a type field.
3) Store both address and enrollments inside the student document.
4) Store enrollments inside the course document.

Reply format: 1 / 2 / 3 / 4

Answer 8

Which design best fits these two requirements?
Requirement A: a student has address fields.
Requirement B: an enrollment links one student to one course.

1) Store enrollments as separate documents that keep studentId and courseId.
2) Store everything in one collection and distinguish using a type field.
3) Store both address and enrollments inside the student document.
4) Store enrollments inside the course document.

Correct answer: 1
Why: Enrollments are relationship records, so storing them as separate documents keeps the model clean and scalable.

Reply format: 1 / 2 / 3 / 4

Implement: Listing & Search

Live coding goal

- Enable a list endpoint that fetches students from MongoDB with filtering, sorting, and pagination.
- Reuse StudentPageResponse so the response returns items and page metadata together.
- Implement changes step by step so behavior is visible after each change.

Files that will be updated

- StudentController for the list endpoint and query parameters.
- StudentService for filtering logic and Pageable construction.
- StudentRepository for DB-backed queries.
- StudentPageResponse for the final response shape.

List endpoint behavior

- The endpoint accepts query parameters that control which students are returned and in what order.
- The endpoint returns a page response so the client can request the next page reliably.

Supported query parameters

- page and size control pagination.
- sortBy and sortDir control sorting.
- active filters by active status when provided.
- q searches by name or email when provided.

Example requests

```
/students?page=0&size=10&sortBy=name&sortDir=asc
/students?active=true&q=ar&page=0&size=5&sortBy=name&sortDir=asc
```

Your turn!

Activity

Goal: Add the core domain building blocks for courses and enrollments.

- Create two new MongoDB document models: Course and Enrollment.
- Create two new repositories: CourseRepository and EnrollmentRepository.
- Keep the implementation minimal and consistent with the existing Student model and repository.
- Note: you are not required to do anything in MongoDB Compass for this activity.

Collections to be created

- courses
- enrollments

Requirement: Course document

- A course is stored as one document in the courses collection.
- Each course needs fields that support searching and display.

Required fields

- id as String, mapped to MongoDB _id.
- title as String.
- code as String, unique.
- active as boolean.

```
{
  "_id": "65b9a0c2a9c7e24e0f4e20a1",
  "title": "Spring Boot Foundations",
  "code": "SB101",
  "active": true
}
```

Requirement: Enrollment document

- An enrollment is stored as one document in the enrollments collection.
- An enrollment connects one student to one course.
- Enrollment stores references so student and course remain independent documents.

Required fields

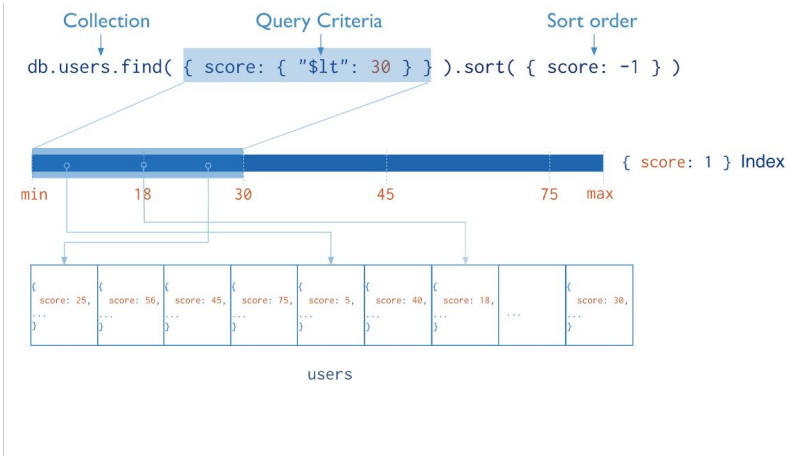
- id as String, mapped to MongoDB _id.
- studentId as String.
- courseId as String.
- enrolledAt as Instant.
- status as String, for example ACTIVE or CANCELLED.

```
{
  "_id": "65b9a13aa9c7e24e0f4e20b2",
  "studentId": "65b7f3b2a9c7e24e0f4e1a91",
  "courseId": "65b9a0c2a9c7e24e0f4e20a1",
  "enrolledAt": "2026-02-01T10:15:00Z",
  "status": "ACTIVE"
}
```

Indexing

Recall: why do indexes matter in an API?

- An index exists to make reads faster by avoiding unnecessary scanning of documents.
- Without an index, MongoDB may need to examine many documents to find matches.
- With an index, MongoDB can jump directly to matching keys and fetch only relevant documents.
- Indexes can also enforce rules such as uniqueness when configured as unique.



Sample: Dataset for index demonstrations

- The examples assume the students collection stores fields like name, email, active, and city.
- The query examples (in upcoming slides) assume many students exist, not just a handful.

```
{ "_id": "65b7f3...a91", "name": "Rahul Kumar", "email": "rahul.k@gmail.com", "active": true,  "city":
"Bengaluru" }
{ "_id": "65b7f3...a92", "name": "Meera N",     "email": "meera.n@gmail.com", "active": false, "city":
"Chennai" }
{ "_id": "65b7f3...a93", "name": "Armaan Khan", "email": "armaan.k@gmail.com","active": true,  "city":
"Bengaluru" }
{ "_id": "65b7f3...a94", "name": "Zoya Farhan", "email": "zoya.f@gmail.com", "active": true,  "city":
"Kochi" }
```

Single field index and the mental model

Query to support efficiently:

```
db.students.find({ city: "Bengaluru" })
```

Index that supports it:

```
db.students.createIndex({ city: 1 })
```

What MongoDB builds:

- MongoDB does not rearrange the documents inside the collection.
- MongoDB builds a separate index structure where keys are stored in sorted order.
- Each index key points to the documents that contain that value.

```
Index on city

"Bengaluru"  -> [ 65b7f3...a91, 65b7f3...a93 ]
"Chennai"    -> [ 65b7f3...a92 ]
"Kochi"      -> [ 65b7f3...a94 ]
```

Unique index as a database rule

- A unique index prevents two documents from having the same value for the indexed field.
- MongoDB enforces this rule during insert and update operations.

Example: email must be unique

```
db.students.createIndex({ email: 1 }, { unique: true })
```

Why this matters:

- Application code can try to prevent duplicates, but what if a query inserting duplicates is executed via MongoDB Compass?
- The database rule also protects data integrity when two requests are processed very close together. Eg:
    - Two requests can be processed concurrently (often on different server threads).
    - If the app does check-then-insert (exists check, then save), both requests may pass the check before either inserts.
    - Without a unique index, both inserts can succeed -> duplicates.
    - With a unique index, one insert succeeds and the other fails with a DuplicateKeyException (which the app should catch and translate to a 409).

Duplicate email scenario described as a timeline

- Two create requests arrive close together for the same email.
- Request A checks and does not see the email yet.
- Before request A finishes saving, request B also checks and does not see the email yet.
- Both requests attempt to insert a new student.

What happens next:

- Without a unique index, both inserts can succeed and duplicates can be stored.
- With a unique index, MongoDB accepts the first insert and rejects the second insert.

Compound index and why order matters

- A compound index is built on more than one field.
- MongoDB stores the combined keys in sorted order.
- The first field creates the primary grouping, and the second field creates the secondary grouping.

Example compound index:

```
db.students.createIndex({ active: 1, city: 1 })
```

```
Index on (active, city)

(false, "Chennai")   -> [ 65b7f3...a92 ]
(true,  "Bengaluru") -> [ 65b7f3...a91, 65b7f3...a93 ]
(true,  "Kochi")     -> [ 65b7f3...a94 ]
```

- This index supports filtering by active efficiently.
- It also supports filtering by active and city together.
- It is not the best fit for queries that only filter by city.

Proving index usage beyond Compass

- Compass can show which indexes exist.
- explain("executionStats") shows which index MongoDB actually used for a query.

Command to run in mongosh:

```
db.students.find({ city: "Bengaluru" }).explain("executionStats")
```

What to look for:

- The plan indicates an index-based scan instead of a full collection scan.
- The execution stats show fewer examined documents when the index is used.

Embedded Docs & References

Recap: Modelling relationships in MongoDB

- MongoDB supports two common relationship patterns.
- You either embed related data inside one document, or store references between documents.

Embedding

- Store a smaller related object inside the parent document.
- Use this when the data is owned by the parent and is usually read together.

Referencing

- Store the related entity as its own document and store its id in another document.
- Use this when the related entity is shared, grows independently, or is queried separately.

Embedded example: Address inside Student

- Address fits well as embedded data because it belongs to a student.
- Address is usually read and updated together with the student.

```
A Student Document:
{
  "_id": "65b7f3b2a9c7e24e0f4e1a91",
  "name": "Rahul Kumar",
  "email": "rahul.k@gmail.com",
  "active": true,
  "address": {
    "line1": "123 MG Road",
    "city": "Bengaluru",
    "state": "Karnataka",
    "pincode": "560001"
  }
}
```

Practical benefit:

- A single read returns the student and address together.
- No join-style lookup is required.

Reference example: Enrollment links Student and Course

- Enrollment should be a separate document because it is a relationship record.
- Many students can enroll into many courses.
- Enrollment data can grow independently and is often queried on its own.

```
{
  "_id": "65b9a13aa9c7e24e0f4e20b2",
  "studentId": "65b7f3b2a9c7e24e0f4e1a91",
  "courseId": "65b9a0c2a9c7e24e0f4e20a1",
  "enrolledAt": "2026-02-01T10:15:00Z",
  "status": "ACTIVE"
}
```

Practical benefit:

- Student and course documents remain small and stable.
- Enrollment queries do not force student documents to grow endlessly.

How this maps to our API and repositories

- Student remains the main API resource.
- Course and Enrollment enable relationship-based features.

Repository usage examples

- EnrollmentRepository.findByStudentId(studentId) lists what a student is enrolled in.
- EnrollmentRepository.findByCourseId(courseId) lists who is enrolled in a course.
- CourseRepository.findByCodeIgnoreCase(code) resolves courses by a human-readable key.

Next extension:

- Add endpoints that read enrollments and expand them into student and course details.

That’s a wrap!