Let’s go step by step on how to work with **MongoDB and Java**.

**1. Add MongoDB Java Driver**

To connect Java with MongoDB, you need the official driver. If using **Maven**, add this to your pom.xml:

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver-sync</artifactId>

<version>5.1.0</version> <!-- check latest -->

</dependency>

For **Gradle**:

implementation 'org.mongodb:mongodb-driver-sync:5.1.0'

**2. Connect to MongoDB**

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoDatabase;

public class MongoConnect {

public static void main(String[] args) {

// connect to local MongoDB

try (MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = mongoClient.getDatabase("testdb");

System.out.println("Connected to database: " + db.getName());

}

}

}

**3. Insert Documents**

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

import org.bson.Document;

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoClient;

public class InsertExample {

public static void main(String[] args) {

try (MongoClient client = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = client.getDatabase("testdb");

MongoCollection<Document> collection = db.getCollection("users");

Document user = new Document("name", "Alice")

.append("age", 25)

.append("email", "alice@example.com");

collection.insertOne(user);

System.out.println("Document inserted!");

}

}

}

**4. Query Documents**

import com.mongodb.client.MongoCursor;

public class QueryExample {

public static void main(String[] args) {

try (MongoClient client = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = client.getDatabase("testdb");

MongoCollection<Document> collection = db.getCollection("users");

// find all users

MongoCursor<Document> cursor = collection.find().iterator();

while (cursor.hasNext()) {

System.out.println(cursor.next().toJson());

}

// find with filter

Document query = new Document("name", "Alice");

Document result = collection.find(query).first();

System.out.println("Found: " + result.toJson());

}

}

}

**5. Update and Delete**

import com.mongodb.client.model.Updates;

import com.mongodb.client.model.Filters;

public class UpdateDeleteExample {

public static void main(String[] args) {

try (MongoClient client = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = client.getDatabase("testdb");

MongoCollection<Document> collection = db.getCollection("users");

// Update

collection.updateOne(Filters.eq("name", "Alice"),

Updates.set("age", 26));

// Delete

collection.deleteOne(Filters.eq("name", "Alice"));

System.out.println("Update/Delete complete");

}

}

}

**6. POJO Mapping (Optional, Cleaner)**

MongoDB supports mapping Java classes to documents with mongodb-driver-sync + PojoCodecProvider.

import org.bson.codecs.configuration.CodecRegistry;

import org.bson.codecs.pojo.PojoCodecProvider;

import static org.bson.codecs.configuration.CodecRegistries.fromProviders;

import static org.bson.codecs.configuration.CodecRegistries.fromRegistries;

CodecRegistry pojoCodecRegistry = fromRegistries(

MongoClientSettings.getDefaultCodecRegistry(),

fromProviders(PojoCodecProvider.builder().automatic(true).build())

);

MongoDatabase db = client.getDatabase("testdb").withCodecRegistry(pojoCodecRegistry);

Then you can use plain Java classes (User class with fields) instead of Document.

✅ So in summary:

1. Add MongoDB driver.
2. Connect with MongoClients.
3. Perform CRUD using MongoCollection<Document>.
4. Optionally use POJO mapping.

here’s a compact, **runnable core-Java** example that shows **CRUD + searching + sorting + pagination** on an Employee entity using the MongoDB **sync** driver with POJO mapping.

**What you get**

* POJO Employee
* Connection + POJO codec setup
* Create/Insert (single & bulk)
* Read (by id, all)
* Search: equality, range, regex “contains”, membership (skills)
* Sorting (multi-field), Pagination
* Update (single & bulk)
* Delete (single & bulk)
* Helpful indexes

### MongoEmployeeDemo.java

import com.mongodb.MongoClientSettings;

import com.mongodb.client.\*;

import com.mongodb.client.model.\*;

import org.bson.Document;

import org.bson.codecs.configuration.CodecRegistry;

import org.bson.codecs.pojo.PojoCodecProvider;

import org.bson.types.ObjectId;

import org.bson.conversions.Bson;

import java.time.Instant;

import java.util.\*;

import static com.mongodb.client.model.Filters.\*;

import static com.mongodb.client.model.Updates.\*;

import static com.mongodb.client.model.Sorts.\*;

import static org.bson.codecs.configuration.CodecRegistries.fromProviders;

import static org.bson.codecs.configuration.CodecRegistries.fromRegistries;

public class MongoEmployeeDemo {

// ----- POJO -----

public static class Employee {

private ObjectId id; // \_id

private String name;

private String department;

private double salary;

private Instant hiredAt; // requires recent driver; use java.util.Date if you prefer

private List<String> skills;

private boolean active;

public Employee() {} // REQUIRED by POJO codec

public Employee(String name, String department, double salary,

Instant hiredAt, List<String> skills, boolean active) {

this.name = name;

this.department = department;

this.salary = salary;

this.hiredAt = hiredAt;

this.skills = skills;

this.active = active;

}

// Getters & setters (required)

public ObjectId getId() { return id; }

public void setId(ObjectId id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getDepartment() { return department; }

public void setDepartment(String department) { this.department = department; }

public double getSalary() { return salary; }

public void setSalary(double salary) { this.salary = salary; }

public Instant getHiredAt() { return hiredAt; }

public void setHiredAt(Instant hiredAt) { this.hiredAt = hiredAt; }

public List<String> getSkills() { return skills; }

public void setSkills(List<String> skills) { this.skills = skills; }

public boolean isActive() { return active; }

public void setActive(boolean active) { this.active = active; }

@Override public String toString() {

return "Employee{" +

"id=" + id +

", name='" + name + '\'' +

", department='" + department + '\'' +

", salary=" + salary +

", hiredAt=" + hiredAt +

", skills=" + skills +

", active=" + active +

'}';

}

}

public static void main(String[] args) {

// 1) POJO codec setup

CodecRegistry pojoRegistry = fromRegistries(

MongoClientSettings.getDefaultCodecRegistry(),

fromProviders(PojoCodecProvider.builder().automatic(true).build())

);

// 2) Connect

try (MongoClient client = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = client.getDatabase("company").withCodecRegistry(pojoRegistry);

MongoCollection<Employee> employees = db.getCollection("employees", Employee.class);

// Clean slate for demo

employees.drop();

// 3) Indexes (help searches)

employees.createIndex(Indexes.ascending("name"));

employees.createIndex(Indexes.ascending("department", "salary"));

employees.createIndex(Indexes.descending("hiredAt"));

employees.createIndex(Indexes.ascending("skills"));

// 4) Seed data (CREATE - insertMany)

seedData(employees);

// 5) CREATE - insertOne

Employee e = new Employee("Karthik", "Sales", 6.2, Instant.now(),

List.of("negotiation", "excel"), true);

employees.insertOne(e);

System.out.println("Inserted one: " + e.getId());

// 6) READ - find all

System.out.println("\nAll employees:");

employees.find().forEach(System.out::println);

// 7) READ - find by id

Employee foundById = employees.find(eq("\_id", e.getId())).first();

System.out.println("\nFound by \_id: " + foundById);

// 8) SEARCH - by equality

System.out.println("\nSearch: department == 'Engineering'");

employees.find(eq("department", "Engineering")).forEach(System.out::println);

// 9) SEARCH - regex contains (case-insensitive)

System.out.println("\nSearch: name contains 'an' (case-insensitive)");

employees.find(regex("name", "an", "i")).forEach(System.out::println);

// 10) SEARCH - range + AND

System.out.println("\nSearch: dept=Engineering AND 7.0 <= salary <= 10.0");

employees.find(and(eq("department", "Engineering"), gte("salary", 7.0), lte("salary", 10.0)))

.forEach(System.out::println);

// 11) SEARCH - set membership

System.out.println("\nSearch: skills contains 'react'");

employees.find(eq("skills", "react")).forEach(System.out::println);

// 12) SORTING (multi-field)

System.out.println("\nSorted: by salary DESC, then name ASC");

employees.find()

.sort(orderBy(descending("salary"), ascending("name")))

.forEach(System.out::println);

// 13) PAGINATION (page=2, size=3)

int page = 2, size = 3;

System.out.println("\nPagination: page " + page + ", size " + size);

employees.find()

.sort(ascending("name"))

.skip((page - 1) \* size)

.limit(size)

.forEach(System.out::println);

// 14) UPDATE - single

System.out.println("\nUpdate: set Karthik's salary = 6.8 and add skill 'crm'");

UpdateResult ur1 = employees.updateOne(eq("\_id", e.getId()),

combine(set("salary", 6.8), addToSet("skills", "crm")));

System.out.println("Matched: " + ur1.getMatchedCount() + ", Modified: " + ur1.getModifiedCount());

System.out.println("After update: " + employees.find(eq("\_id", e.getId())).first());

// 15) UPDATE - bulk (give 10% raise to Engineering)

System.out.println("\nBulk Update: +10% salary for Engineering");

UpdateResult ur2 = employees.updateMany(eq("department", "Engineering"),

mul("salary", 1.10));

System.out.println("Matched: " + ur2.getMatchedCount() + ", Modified: " + ur2.getModifiedCount());

// 16) DELETE - single

System.out.println("\nDelete: remove Karthik by \_id");

DeleteResult dr1 = employees.deleteOne(eq("\_id", e.getId()));

System.out.println("Deleted: " + dr1.getDeletedCount());

// 17) DELETE - bulk (inactive)

System.out.println("\nBulk Delete: remove inactive employees");

DeleteResult dr2 = employees.deleteMany(eq("active", false));

System.out.println("Deleted: " + dr2.getDeletedCount());

// 18) Final list

System.out.println("\nRemaining employees:");

employees.find().forEach(System.out::println);

}

}

private static void seedData(MongoCollection<Employee> employees) {

List<Employee> batch = List.of(

new Employee("Ananya", "Engineering", 9.5, Instant.parse("2022-03-10T00:00:00Z"),

List.of("java", "spring", "mongodb"), true),

new Employee("Rohan", "Engineering", 7.2, Instant.parse("2023-07-01T00:00:00Z"),

List.of("python", "flask"), true),

new Employee("Meera", "HR", 5.1, Instant.parse("2021-11-20T00:00:00Z"),

List.of("recruiting", "communication"), true),

new Employee("Vikram", "Engineering", 10.4, Instant.parse("2020-01-15T00:00:00Z"),

List.of("react", "node", "docker"), true),

new Employee("Sana", "Design", 6.8, Instant.parse("2024-02-05T00:00:00Z"),

List.of("figma", "ux", "html"), false),

new Employee("Nandini", "Finance", 8.0, Instant.parse("2019-06-25T00:00:00Z"),

List.of("excel", "tally"), true),

new Employee("Arun", "Engineering", 7.9, Instant.parse("2023-01-03T00:00:00Z"),

List.of("java", "spring", "kafka"), true),

new Employee("Ishaan", "Engineering", 8.7, Instant.parse("2022-08-19T00:00:00Z"),

List.of("go", "kubernetes"), true)

);

employees.insertMany(batch);

System.out.println("Seeded " + batch.size() + " employees.");

}

}

### How to run

1. Add dependency (Maven):
2. <dependency>
3. <groupId>org.mongodb</groupId>
4. <artifactId>mongodb-driver-sync</artifactId>
5. <version>5.1.0</version> <!-- or latest -->
6. </dependency>
7. Ensure MongoDB is running locally at mongodb://localhost:27017.
8. Compile & run:
9. javac -cp .:mongo.jar MongoEmployeeDemo.java
10. java -cp .:mongo.jar MongoEmployeeDemo

### Notes / Tweaks

* If Instant causes issues on older driver versions, switch hiredAt to java.util.Date.
* In real apps, avoid dropping collections; use migrations/seeders.
* Consider **validation rules** and **unique indexes** (e.g., unique email/empCode).
* For large lists, prefer **server-side pagination** using a stable sort + skip/limit or **range pagination** with filters.