# Advanced NoSQL Lab Tasks - MongoDB

## Task 7: Working with Embedded Documents

Objective: Learn how to store and retrieve nested JSON objects in MongoDB.

Insert a Student with Course Details:

db.students.insertOne({  
 name: "Hassan Raza",  
 age: 22,  
 semester: 6,  
 courses: [  
 { subject: "Database Systems", marks: 85 },  
 { subject: "Artificial Intelligence", marks: 90 }  
 ]  
})

Retrieve a Student with a Specific Course:

db.students.find({ "courses.subject": "Artificial Intelligence" }).pretty()

Update Marks for a Specific Course:

db.students.updateOne(  
 { name: "Hassan Raza", "courses.subject": "Database Systems" },  
 { $set: { "courses.$.marks": 95 } }  
)

Delete a Course from a Student's Record:

db.students.updateOne(  
 { name: "Hassan Raza" },  
 { $pull: { courses: { subject: "Artificial Intelligence" } } }  
)

## Task 8: Using Array Operators in MongoDB

Insert Multiple Hobbies for a Student:

db.students.insertOne({  
 name: "Sara Ahmed",  
 age: 21,  
 semester: 5,  
 hobbies: ["Reading", "Gaming", "Swimming"]  
})

Find Students with a Specific Hobby:

db.students.find({ hobbies: "Gaming" }).pretty()

Add a New Hobby to an Existing Student:

db.students.updateOne(  
 { name: "Sara Ahmed" },  
 { $push: { hobbies: "Traveling" } }  
)

Remove a Hobby from the Student:

db.students.updateOne(  
 { name: "Sara Ahmed" },  
 { $pull: { hobbies: "Swimming" } }  
)

## Task 9: Implementing Role-Based Access Control (RBAC)

Create an Admin User:

use admin  
db.createUser({  
 user: "labAdmin",  
 pwd: "securepassword123",  
 roles: [{ role: "userAdminAnyDatabase", db: "admin" }]  
})

Enable Authentication in MongoDB:

security:  
 authorization: enabled

Create a User with Read-Only Access:

use StudentDB  
db.createUser({  
 user: "readonlyUser",  
 pwd: "readonly123",  
 roles: [{ role: "read", db: "StudentDB" }]  
})

Authenticate as Read-Only User:

mongo -u readonlyUser -p readonly123 --authenticationDatabase "StudentDB"

## Task 10: Query Optimization and Performance Tuning

Create a Large Dataset (10,000 Documents):

for (let i = 1; i <= 10000; i++) {  
 db.students.insertOne({  
 name: "Student" + i,  
 age: Math.floor(Math.random() \* 10) + 18,  
 semester: Math.floor(Math.random() \* 8) + 1  
 });  
}

Create an Index for Faster Searching:

db.students.createIndex({ name: 1 })

Use Explain Plan to Check Query Performance:

db.students.find({ name: "Student5000" }).explain("executionStats")

Drop an Index:

db.students.dropIndex({ name: 1 })

## Task 11: Implement Data Backup and Restore

Backup the Database:

mongodump --db StudentDB --out /backup/mongo

Restore the Database:

mongorestore --db StudentDB /backup/mongo/StudentDB

## Task 12: Integrate MongoDB with a Web Application

Install Dependencies:

npm install express mongoose body-parser cors

Create a MongoDB Connection in Node.js:

const mongoose = require("mongoose");  
  
mongoose.connect("mongodb://localhost:27017/StudentDB", {  
 useNewUrlParser: true,  
 useUnifiedTopology: true  
}).then(() => console.log("Connected to MongoDB"))  
 .catch(err => console.log("MongoDB connection error:", err));

Define a Student Schema:

const studentSchema = new mongoose.Schema({  
 name: String,  
 age: Number,  
 semester: Number  
});  
const Student = mongoose.model("Student", studentSchema);

Create API Endpoints:

const express = require("express");  
const app = express();  
app.use(express.json());  
  
app.get("/students", async (req, res) => {  
 const students = await Student.find();  
 res.json(students);  
});  
  
app.listen(3000, () => console.log("Server running on port 3000"));

Run the Node.js server:

node server.js

Test the API:

http://localhost:3000/students

## Final Lab Assignment

Students will:  
1. Create a complete database system for a University Management System.  
2. Implement:  
 - Students collection (name, age, semester, courses, grades).  
 - Professors collection (name, department, experience).  
 - Courses collection (course name, professor, credits).  
3. Apply CRUD operations, aggregation, and indexing.  
4. Integrate MongoDB with Python or Node.js for data retrieval.  
5. Secure the database with authentication & backup features.  
6. Submit a report with screenshots of all tasks performed.