

EXPERIMENT NO. 7 - MongoDB

Name of Student	Bhagyesh Patil
Class Roll No	D15A_35
D.O.P.	13/02/2025
D.O.S.	20/03/2025
Sign and Grade	

AIM: To study CRUD operations in MongoDB

PROBLEM STATEMENT:

1. Create a new database to storage student details of IT dept(Name, Roll no, class name) and perform the following on the database
 - a. Insert one student details
 - b. Insert at once multiple student details
 - c. Display student for a particular class
 - d. Display students of specific roll no in a class
 - e. Change the roll no of a student
 - f. Delete entries of particular student
2. Create a set of RESTful endpoints using Node.js, Express, and Mongoose for handling student data operations. The endpoints should support:
 - a. Retrieve a list of all students.
 - b. Retrieve details of an individual student by ID.
 - c. Add a new student to the database.
 - d. Update details of an existing student by ID.
 - e. Delete a student from the database by ID.Connect the server to MongoDB using Mongoose, and store student data with attributes: name, age, and grade.

OUTPUT:

Step 1: Use or Create a Database and Create and Use a Collection (e.g., students)

use IT_Dept_Students

db.createCollection("students")

```
>_MONGOSH
> use IT_Dept_Students
< switched to db IT_Dept_Students
> db.createCollection("students")
< { ok: 1 }
```

Step 2:

a) Insert One Student Detail

```
> db.students.insertOne({
  name: " Bhagyesh patil ",
  roll_no: 35,
  class_name: "IT-D15A"
})
< {
  acknowledged: true,
  insertedId: ObjectId('67fd01f908aa5808a6f9fec8')
}
```

b) Insert Multiple Student Details at Once

```
> db.students.insertMany([
  { name: "Kartik Bhatt", roll_no: 03, class_name: "IT-D15A" },
  { name: "Prajwal Pandey", roll_no: 32 , class_name: "IT-D15A" },
  { name: "Aryan Dangat", roll_no: 12, class_name: "IT-D15A" },
  {name: "Swaraj Patil", roll_no: 39, class_name: "IT-D15A"}
])
```

c) Display Students of a Particular Class:

```
> db.students.find({ class_name: "IT-D15A" })
< {
  _id: ObjectId('67fcffbe08aa5808a6f9fec4'),
  name: 'Kartik Bhatt',
  roll_no: 3,
  class_name: 'IT-D15A'
}
{
  _id: ObjectId('67fcffbe08aa5808a6f9fec5'),
  name: 'Prajwal Pandey',
  roll_no: 32,
  class_name: 'IT-D15A'
}
{
  _id: ObjectId('67fcffbe08aa5808a6f9fec6'),
  name: 'Aryan Dangat',
  roll_no: 12,
  class_name: 'IT-D15A'
}
{
  _id: ObjectId('67fcffbe08aa5808a6f9fec7'),
  name: 'Swaraj Patil',
  roll_no: 39,
  class_name: 'IT-D15A'
}
```

d) Display Students of a Specific Roll No in a Class

```
> db.students.find({ class_name: "IT-D15A", roll_no: 25 })
< {
  _id: ObjectId('67fd01f908aa5808a6f9fec8'),
  name: 'Bhagyesh Patil',
  roll_no: 25,
  class_name: 'IT-D15A'
}
```

e) Change the Roll No of a Student

```
> db.students.updateOne(
  { name: "Bhagyesh Patil" }, // Filter
  { $set: { roll_no: 10035 } } // Update
)
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

f) Delete Entry of a Particular Student

```
> db.students.deleteOne({ name: "Swaraj Patil" })
< {
  acknowledged: true,
  deletedCount: 1
}
```

Restful API:

a. Retrieve a list of all students.

The screenshot shows a REST client interface with a GET request to `http://localhost:3000/api/students`. The response is a JSON array of three student objects, displayed in the 'Body' tab with the 'Pretty' format selected.

```
POST http://localhost:3000/api/...
GET http://localhost:3000/api/students

POST http://localhost:3000/api/students

Params Authorization Headers (7) Body Pre-request Script Test Results

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

1 {
2   {
3     "_id": "67fcffbe08aa5808a6f9fec4",
4     "name": "Kartik Bhatt",
5     "roll_no": 3,
6     "class_name": "IT-D16A"
7   },
8   {
9     "_id": "67fcffbe08aa5808a6f9fec5",
10    "name": "Prajwal Pandey",
11    "roll_no": 32,
12    "class_name": "IT-D15A"
13  },
14  {
15    "_id": "67fcffbe08aa5808a6f9fec6",
16    "name": "Aryan Dangat",
17    "roll_no": 12,
18    "class_name": "IT-D15A"
19  }
20 }
```

b. Retrieve details of an individual student by ID.

The screenshot shows a REST client interface with a GET request to `http://localhost:3000/api/students/67fcffbe08aa5808a6f9fec4`. The response body is displayed in JSON format, showing the details of a student with ID `67fcffbe08aa5808a6f9fec4`, name `Kartik Bhatt`, roll number `3`, and class name `IT-D15A`.

```
GET http://localhost:3000/api/ + ...
```

`http://localhost:3000/api/students/67fcffbe08aa5808a6f9fec4`

GET `http://localhost:3000/api/students/67fcffbe08aa5808a6f9fec4`

Params Authorization Headers (6) Body Pre-request Script Tests

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "67fcffbe08aa5808a6f9fec4",
3   "name": "Kartik Bhatt",
4   "roll_no": 3,
5   "class_name": "IT-D15A"
6 }
```

c. Add a new student to the database.

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/api/students/`. The request body is in JSON format, containing the details of a new student: name `Riya Shah`, age `20`, and grade `A`. The response status is `201 Created`, and the response body shows the newly created student with an assigned ID `67fd1c0c6d088dafaa2cb049` and a version number `0`.

```
POST http://localhost:3000/ap + ...
```

`http://localhost:3000/api/students/`

POST `http://localhost:3000/api/students/`

Params Authorization Headers (9) Body Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON

```
1 {
2   "name": "Riya Shah",
3   "age": 20,
4   "grade": "A"
5 }
6
```

Body Cookies Headers (7) Test Results Status: 201 Created

Pretty Raw Preview Visualize

```
{ "name": "Riya Shah", "age": 20, "grade": "A", "_id": "67fd1c0c6d088dafaa2cb049", "__v": 0 }
```

d. Update details of an existing student by ID.

The screenshot shows a REST client interface with a PUT request to `http://localhost:3000/api/students/67fd1c0c6d088dafaa2cb049`. The request body is a JSON object: `{ "name": "Riya Mehta", "grade": "A+" }`. The response status is 200 OK, and the response body is a JSON object: `{ "_id": "67fd1c0c6d088dafaa2cb049", "name": "Riya Mehta", "age": 20, "grade": "A+", "__v": 0 }`.

```
PUT http://localhost:3000/api/students/67fd1c0c6d088dafaa2cb049
```

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   "name": "Riya Mehta",
3   "grade": "A+"
4 }
```

Body Cookies Headers (7) Test Results Status: 200 OK

Pretty Raw Preview Visualize

```
{ "_id": "67fd1c0c6d088dafaa2cb049", "name": "Riya Mehta", "age": 20, "grade": "A+", "__v": 0 }
```

e. Delete a student from the database by ID.

The screenshot shows a REST client interface with a DELETE request to `http://localhost:3000/api/students/67fd1c0c6d088dafaa2cb049`. The response status is 200 OK, and the response body is a JSON object: `{ "message": "Student deleted successfully" }`.

```
DEL http://localhost:3000/api/students/67fd1c0c6d088dafaa2cb049
```

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   "name": "Riya Mehta",
3   "grade": "A+"
4 }
```

Body Cookies Headers (7) Test Results Status: 200 OK

Pretty Raw Preview Visualize

```
{ "message": "Student deleted successfully" }
```

CONCLUSION

In this experiment, we successfully performed CRUD operations in **MongoDB** and implemented a **RESTful API** using **Node.js, Express, and Mongoose**. We learned how to create, read, update, and delete student records both via **MongoDB shell commands** and **API endpoints**.