<u>Lab Manual on Non-relational (NoSQL) data models, featuring Document Data Model</u>

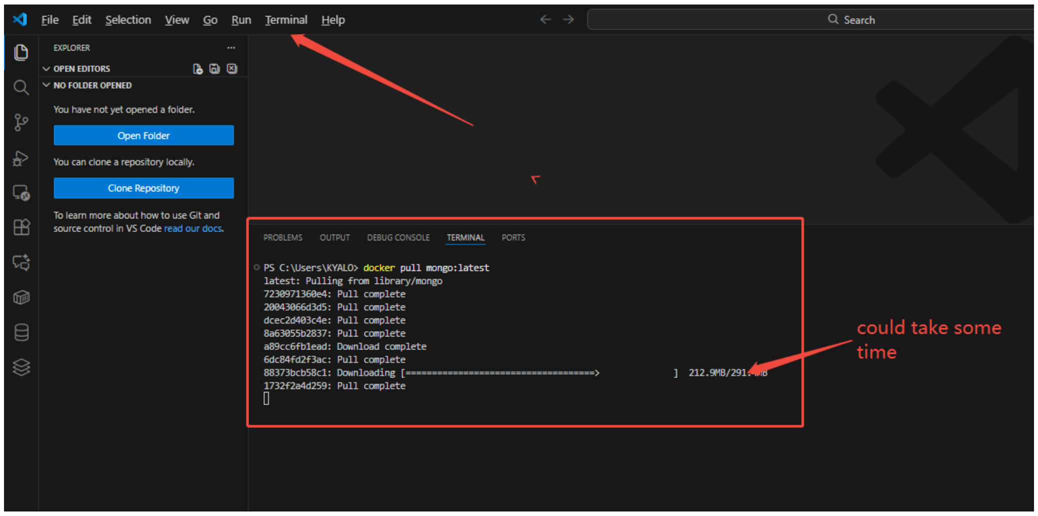
Required Software

- Docker Desktop - https://docs.docker.com/desktop/setup/install/windows-install/
- MongoDB Instance
- VS Code - https://code.visualstudio.com/

<u>PHASE 1: SETUP INSTRUCTIONS</u>

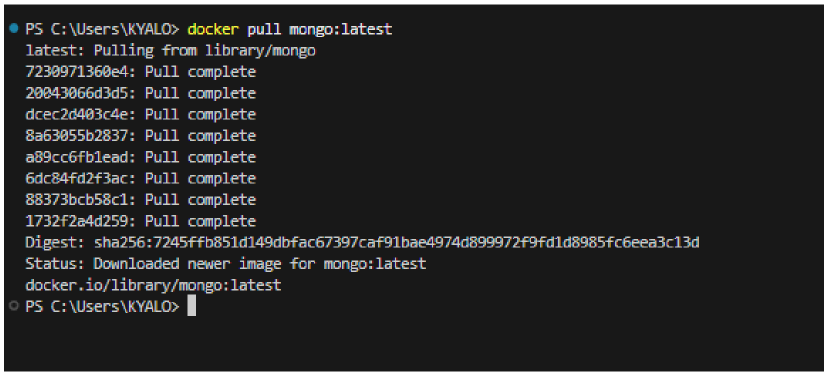
1. Set up MongoDB container with Docker. (launch a single instance of the server).
   ✓ Pull the latest stable MongoDB image. *(Run the commands in VS Code)*

Command: docker pull mongo:latest



```
PS C:\Users\KYALO> docker pull mongo:latest
latest: Pulling from library/mongo
7230971360e4: Pull complete
20043066d3d5: Pull complete
dcec2d403c4e: Pull complete
8a63055b2837: Pull complete
a89cc6fb1ead: Download complete
6dc84fd2f3ac: Pull complete
88373bcb58c1: Downloading [====================================>            ]  212.9MB/291.
1732f2a4d259: Pull complete
```

Output after completion!

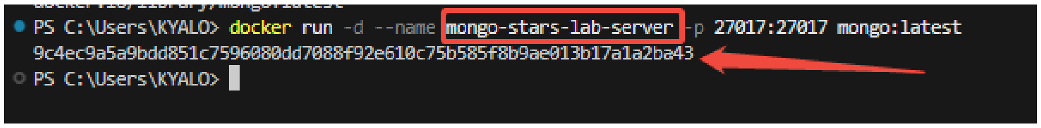


```
PS C:\Users\KYALO> docker pull mongo:latest
latest: Pulling from library/mongo
7230971360e4: Pull complete
20043066d3d5: Pull complete
dcec2d403c4e: Pull complete
8a63055b2837: Pull complete
a89cc6fb1ead: Pull complete
6dc84fd2f3ac: Pull complete
88373bcb58c1: Pull complete
1732f2a4d259: Pull complete
Digest: sha256:7245ffb851d149dbfac67397caf91bae4974d899972f9fd1d8985fc6eea3c13d
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
PS C:\Users\KYALO>
```

2. Run the MongoDB Container

   Command: docker run -d --name mongo-stars-lab-server -p 27017:27017 mongo:latest

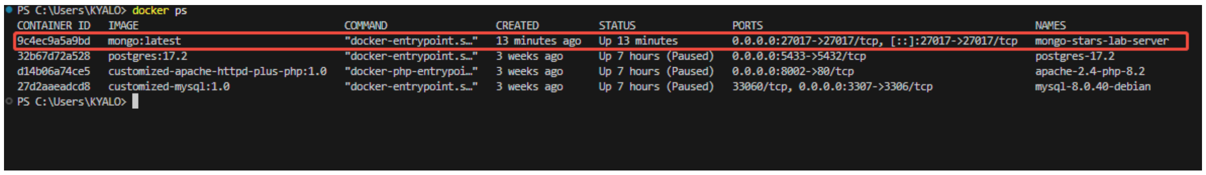
   (You can name the container a name of your choice)



```
PS C:\Users\KYALO> docker run -d --name mongo-stars-lab-server -p 27017:27017 mongo:latest
9c4ec9a5a9bdd851c7596080dd7088f92e610c75b585f8b9ae013b17a1a2ba43
PS C:\Users\KYALO>
```

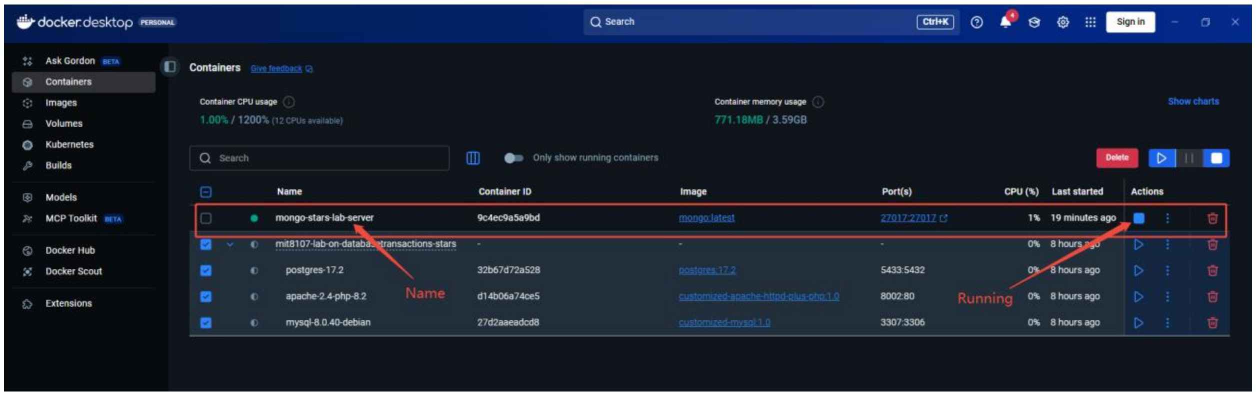
✓ Check Container Status – This can be done through Docker desktop or the terminal on the VS-code. Other Containers are installed in my Docker for your case you will see just one DB container if no other container was installed.

Command: docker ps



```
PS C:\Users\KYALO> docker ps
CONTAINER ID   IMAGE                                 COMMAND                  CREATED          STATUS                PORTS                                          NAMES
9c4ec9a5a9bd   mongo:latest                          "docker-entrypoint.s…"   13 minutes ago   Up 13 minutes         0.0.0.0:27017->27017/tcp, [::]:27017->27017/tcp   mongo-stars-lab-server
32b67d72a528   postgres:17.2                         "docker-entrypoint.s…"   3 weeks ago      Up 7 hours (Paused)   0.0.0.0:5433->5432/tcp                         postgres-17.2
d14b06a74ce5   customized-apache-httpd-plus-php:1.0  "docker-php-entrypoi…"   3 weeks ago      Up 7 hours (Paused)   0.0.0.0:8002->80/tcp                           apache-2.4-php-8.2
27d2aaeadcd8   customized-mysql:1.0                  "docker-entrypoint.s…"   3 weeks ago      Up 7 hours (Paused)   33060/tcp, 0.0.0.0:3307->3306/tcp              mysql-8.0.40-debian
PS C:\Users\KYALO>
```

Or, open the docker desktop app and see if the container is running



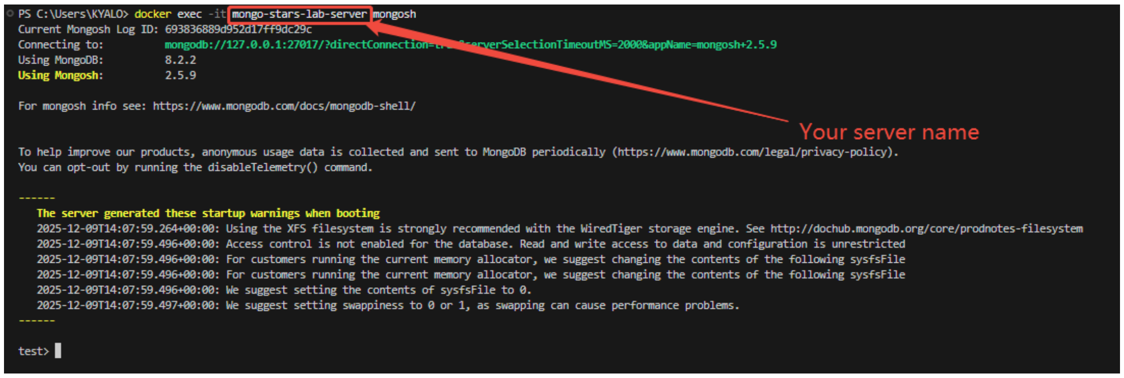
Container CPU usage
1.00% / 1200% (12 CPUs available)

Container memory usage
771.18MB / 3.59GB

Name	Container ID	Image	Port(s)	CPU (%)	Last started
mongo-stars-lab-server	9c4ec9a5a9bd	mongo:latest	27017:27017	1%	19 minutes ago
mit8107-lab-on-databa...transactions-stars	-	-	-	0%	8 hours ago
postgres-17.2	32b67d72a528	postgres:17.2	5433:5432	0%	8 hours ago
apache-2.4-php-8.2	d14b06a74ce5	customized-apache-httpd-plus-php:1.0	8002:80	0%	8 hours ago
mysql-8.0.40-debian	27d2aaeadcd8	customized-mysql:1.0	3307:3306	0%	8 hours ago

3. Connect the Database shell
   - ✓ Now that the database server is running, we can access its shell (mongosh) and execute the required commands.
   - ✓ Using docker exec command, run the MongoDB Shell

     Command: docker exec -it mongo-stars-lab-server mongosh

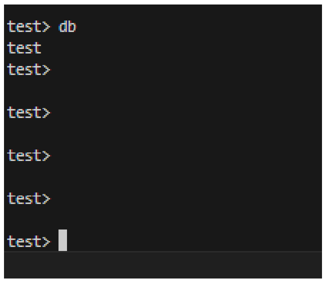


```
PS C:\Users\KYALO> docker exec -it mongo-stars-lab-server mongosh
Current Mongosh Log ID: 693836889d952d17ff9dc29c
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.9
Using MongoDB:          8.2.2
Using Mongosh:          2.5.9

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2025-12-09T14:07:59.264+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
   2025-12-09T14:07:59.496+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2025-12-09T14:07:59.496+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
   2025-12-09T14:07:59.496+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
   2025-12-09T14:07:59.496+00:00: We suggest setting the contents of sysfsFile to 0.
   2025-12-09T14:07:59.497+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.
------

test> 
```

   - ✓ Test means you are connected to the default test DB.
   - ✓ Do some other command to check status

     Command: db

     Output



```
test> db
test
test>

test>

test>

test>

test> 
```

At this point your Environment is completely setup

NB:

- **Ensure that the docker desktop is running as you execute commands on the created lab.**
- **Docker desktop, vs code and Mongo DB should be in their latest versions**

<u>**Versions**</u>

- **Running on Docker version 28.5.1.**
- **Running on mongosh –version 2.5.9.**

PHASE 2: BASIC OPERATIONS

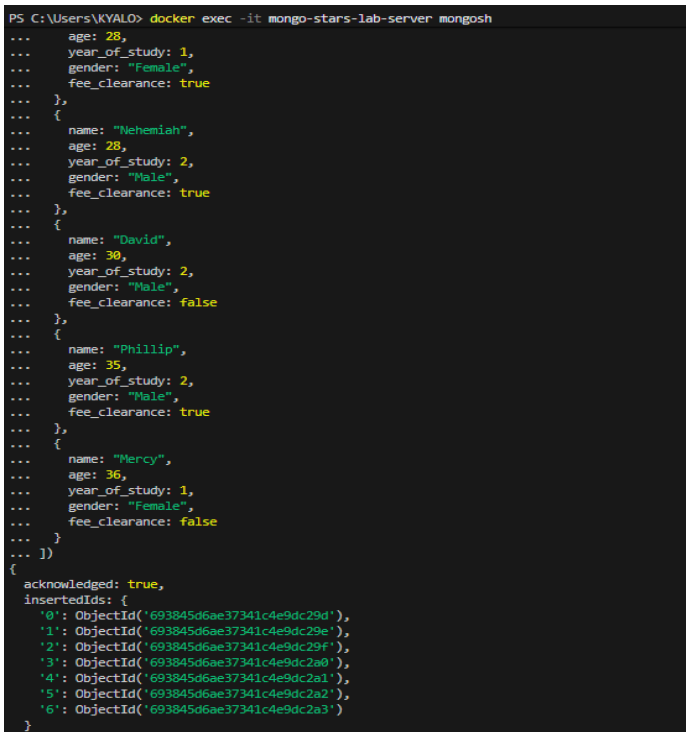
Operation 1 : Insert

1. There are two types of insertions where you can insert a single document or can insert multiple documents. Either way the procedure is the same in our case we will insert multiple documents and the DB name is students.

Commands:

```
db.students.insertMany([
 {
  name: "Elijah",
  age: 32,
  year_of_study: 1,
  gender: "Male",
  fee_clearance: true
 },
 {
  name: "Tonny",
  age: 32,
  year_of_study: 1,
  gender: "Male",
  fee_clearance: true
 },
 {
  name: "Janet",
  age: 28,
  year_of_study: 1,
  gender: "Female",
  fee_clearance: true
},
{
  name: "Nehemiah",
  age: 28,
  year_of_study: 2,
  gender: "Male",
  fee_clearance: true
},
{
  name: "David",
  age: 30,
  year_of_study: 2,
  gender: "Male",
  fee_clearance: false
},
{
  name: "Phillip",
  age: 35,
  year_of_study: 2,
  gender: "Male",
  fee_clearance: true
},
{
  name: "Mercy",
  age: 36,
   year_of_study: 1,

   gender: "Female",

   fee_clearance: false

  }

])
```

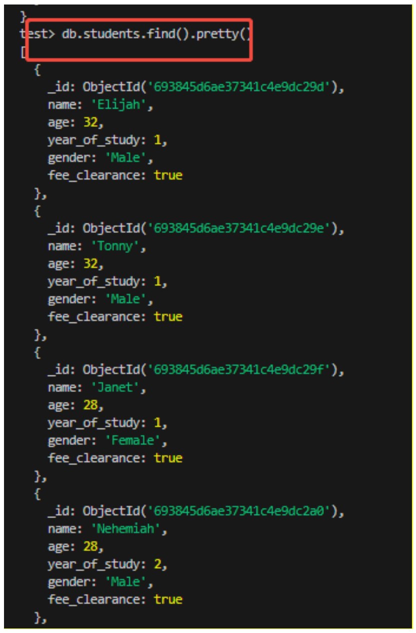
Out put



```
PS C:\Users\KYALO> docker exec -it mongo-stars-lab-server mongosh
...       age: 28,
...       year_of_study: 1,
...       gender: "Female",
...       fee_clearance: true
...     },
...     {
...       name: "Nehemiah",
...       age: 28,
...       year_of_study: 2,
...       gender: "Male",
...       fee_clearance: true
...     },
...     {
...       name: "David",
...       age: 30,
...       year_of_study: 2,
...       gender: "Male",
...       fee_clearance: false
...     },
...     {
...       name: "Phillip",
...       age: 35,
...       year_of_study: 2,
...       gender: "Male",
...       fee_clearance: true
...     },
...     {
...       name: "Mercy",
...       age: 36,
...       year_of_study: 1,
...       gender: "Female",
...       fee_clearance: false
...     }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('693845d6ae37341c4e9dc29d'),
    '1': ObjectId('693845d6ae37341c4e9dc29e'),
    '2': ObjectId('693845d6ae37341c4e9dc29f'),
    '3': ObjectId('693845d6ae37341c4e9dc2a0'),
    '4': ObjectId('693845d6ae37341c4e9dc2a1'),
    '5': ObjectId('693845d6ae37341c4e9dc2a2'),
    '6': ObjectId('693845d6ae37341c4e9dc2a3')
  }
```

2. Verify the insertion
    - ✓ By running a find () Query to check whether documents were created as required. Your Query should run without errors

      Command: db.students.find().pretty()



```
}
test> db.students.find().pretty()
[
  {
    _id: ObjectId('693845d6ae37341c4e9dc29d'),
    name: 'Elijah',
    age: 32,
    year_of_study: 1,
    gender: 'Male',
    fee_clearance: true
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc29e'),
    name: 'Tonny',
    age: 32,
    year_of_study: 1,
    gender: 'Male',
    fee_clearance: true
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc29f'),
    name: 'Janet',
    age: 28,
    year_of_study: 1,
    gender: 'Female',
    fee_clearance: true
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a0'),
    name: 'Nehemiah',
    age: 28,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true
  },
```

Operations 2: Read

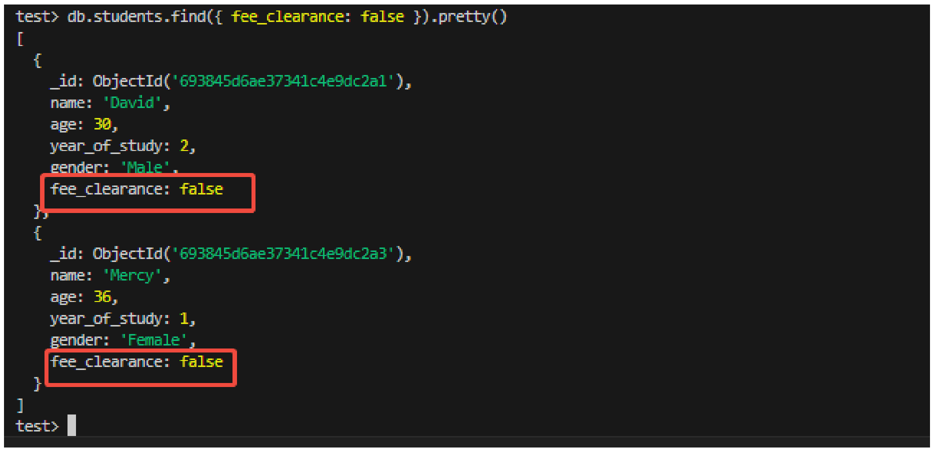
1. For example, I am going to perform two read functions, you can go ahead and read other functions within your context. I will do for query by single criteria and also query by multiple criteria.

- ✓ **Query by single criteria**

  Scenario 1: Read the students who haven't completed their school fee

  Command:db.students.find({ fee_clearance: false }).pretty()

  This filters out documents for mercy and David because at fee_clearance the status is False



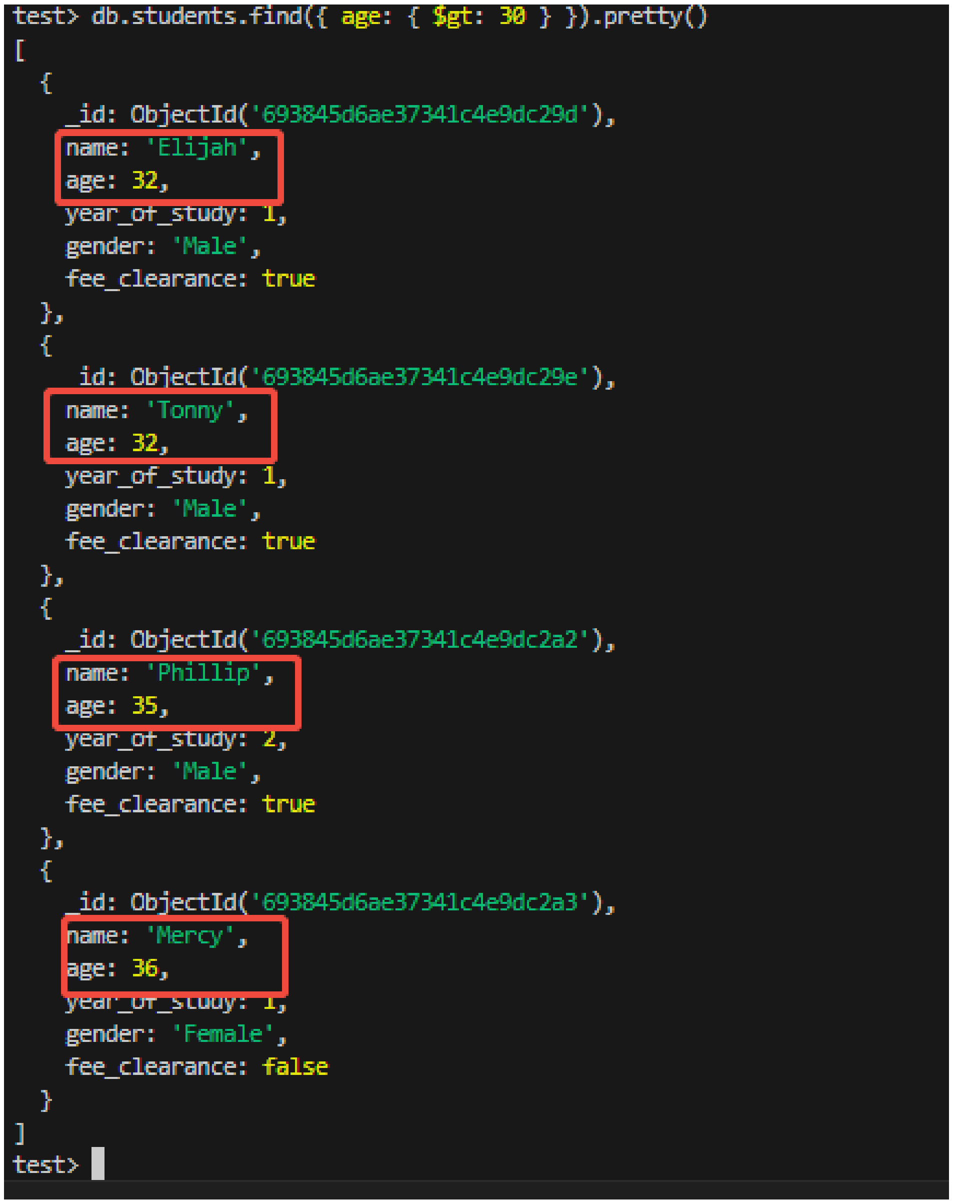
```
test> db.students.find({ fee_clearance: false }).pretty()
[
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a1'),
    name: 'David',
    age: 30,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: false
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a3'),
    name: 'Mercy',
    age: 36,
    year_of_study: 1,
    gender: 'Female',
    fee_clearance: false
  }
]
test> 
```

Scenario 2: Read documents of all students older than 30 years.

Command: db.students.find({ age: { $gt: 30 } }).pretty()

The name $gt is for greater than and used as comparison operator of the attribute age.

This filers out documents for Elijah, Tonny, Phillip and Mercy



```
test> db.students.find({ age: { $gt: 30 } }).pretty()
[
  {
    _id: ObjectId('693845d6ae37341c4e9dc29d'),
    name: 'Elijah',
    age: 32,
    year_of_study: 1,
    gender: 'Male',
    fee_clearance: true
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc29e'),
    name: 'Tonny',
    age: 32,
    year_of_study: 1,
    gender: 'Male',
    fee_clearance: true
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a2'),
    name: 'Phillip',
    age: 35,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a3'),
    name: 'Mercy',
    age: 36,
    year_of_study: 1,
    gender: 'Female',
    fee_clearance: false
  }
]
test> 
```

- ✓ **Querry By Multiple Criteria using (AND Logic)**

  NB: if you list multiple conditions separated by commas inside the find () query object, MongoDB treats it as an AND operation that means all conditions must be true for the document to be returned.

  Scenario 1: Let's find all year two students who haven't cleared school fee.

  Command: db.students.find({

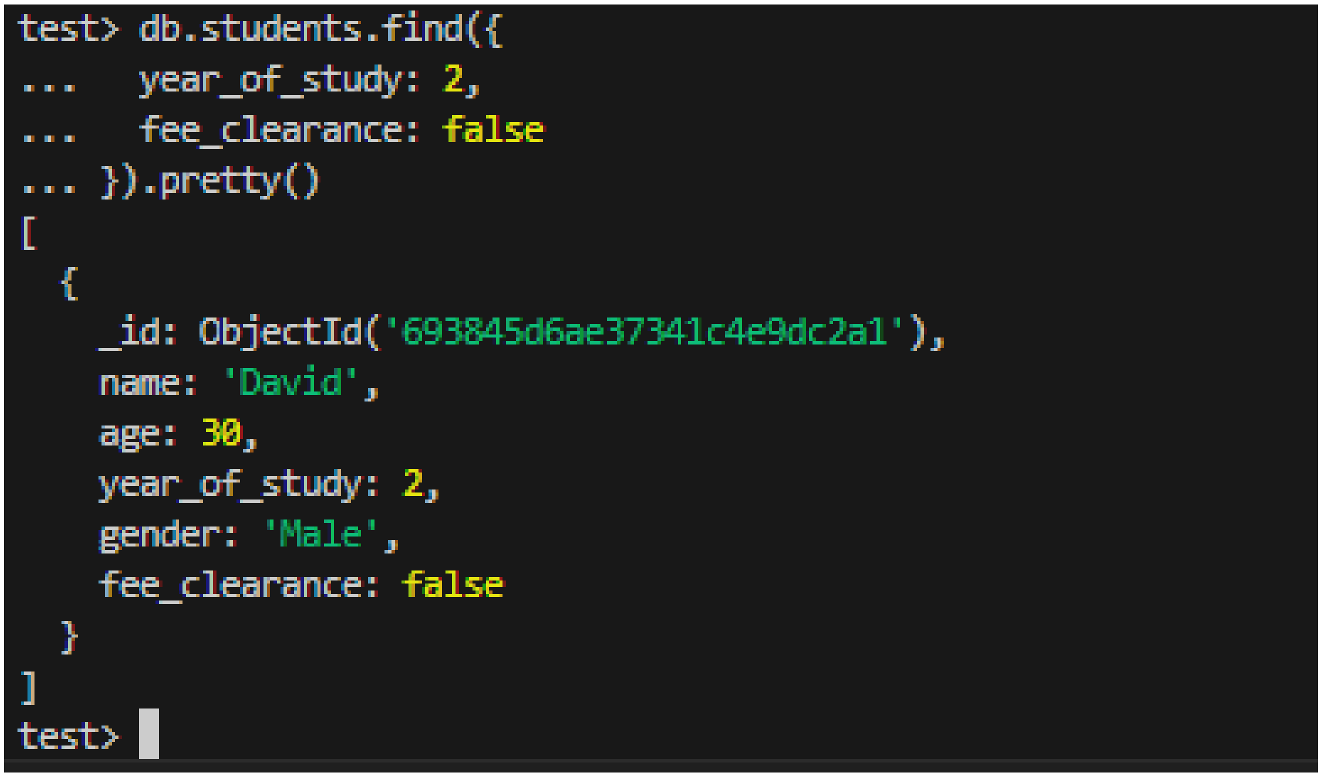
  year_of_study: 2,

  fee_clearance: false

  }).pretty()

For this command to return the query then year of study must be 2 and fee clearance status must be false.

This returns the document for David who meets that criteria.



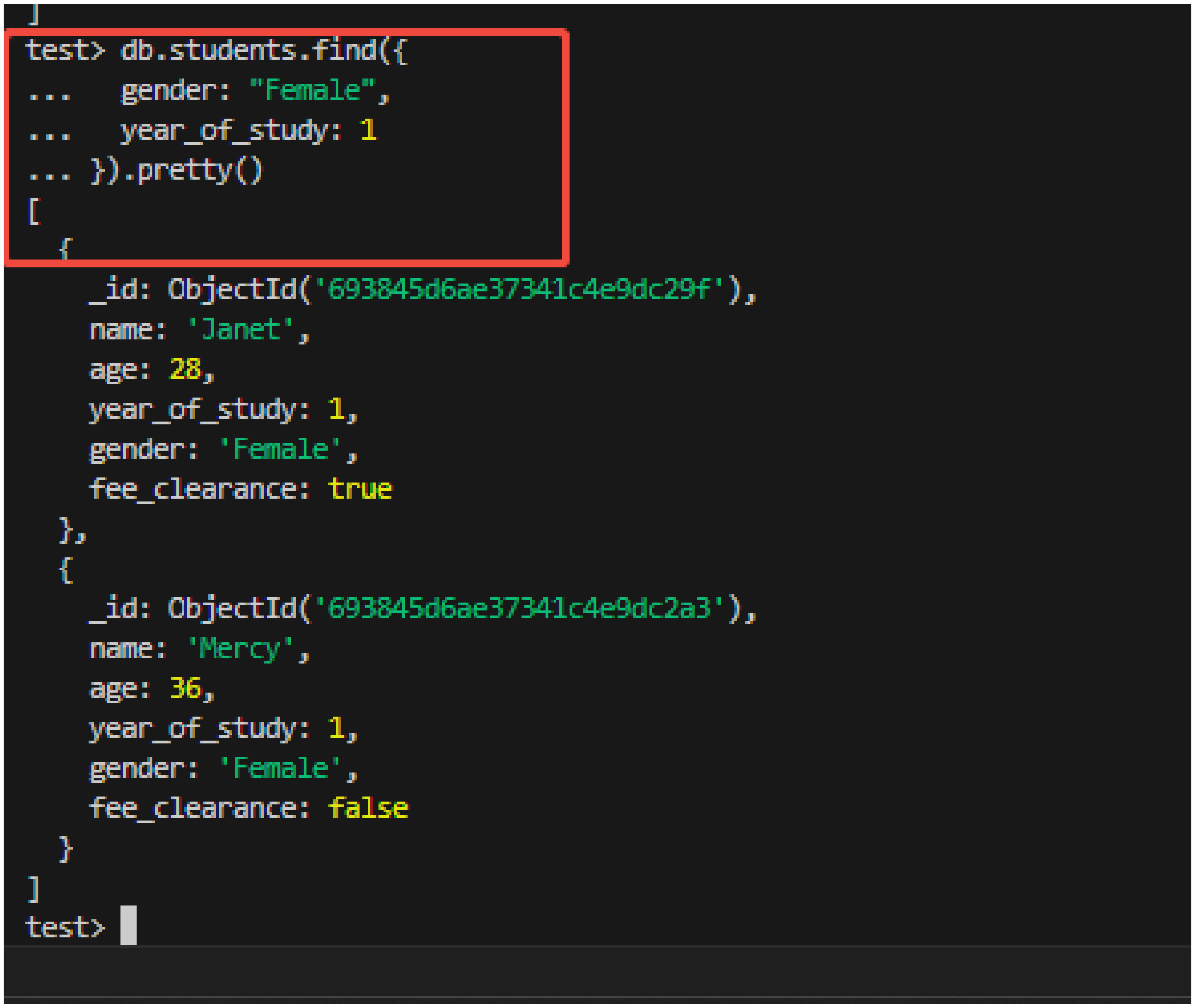
```
test> db.students.find({
...   year_of_study: 2,
...   fee_clearance: false
... }).pretty()
[
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a1'),
    name: 'David',
    age: 30,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: false
  }
]
test> 
```

Scenario 2: Find all female students and are in year 1.

Command:

```
db.students.find({
  gender: "Female",
  year_of_study: 1
}).pretty()
```

Returns

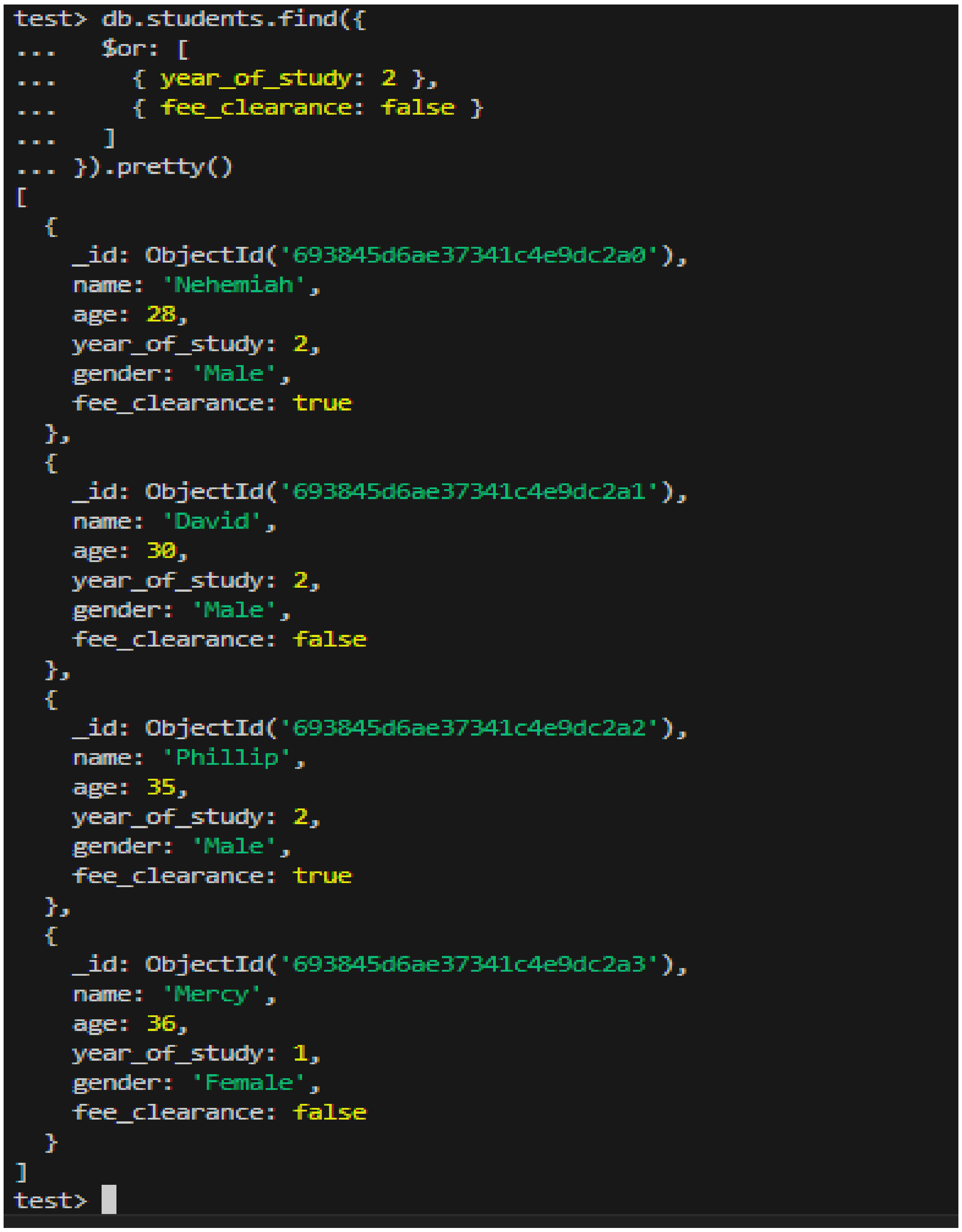


```
]
test> db.students.find({
...   gender: "Female",
...   year_of_study: 1
... }).pretty()
[
  {
    _id: ObjectId('693845d6ae37341c4e9dc29f'),
    name: 'Janet',
    age: 28,
    year_of_study: 1,
    gender: 'Female',
    fee_clearance: true
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a3'),
    name: 'Mercy',
    age: 36,
    year_of_study: 1,
    gender: 'Female',
    fee_clearance: false
  }
]
test>
```

✓ **Querry (OR Logic)**

- Finding documents where **at least one** of several conditions is true. In my example I will querry Students Who Are in Year 2 OR Have Not Cleared Fees.
  Command: db.students.find({ $or: [{ year_of_study: 2 }, { fee_clearance: false }] }).pretty()

  Returns



```
test> db.students.find({
...   $or: [
...     { year_of_study: 2 },
...     { fee_clearance: false }
...   ]
... }).pretty()
[
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a0'),
    name: 'Nehemiah',
    age: 28,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a1'),
    name: 'David',
    age: 30,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: false
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a2'),
    name: 'Phillip',
    age: 35,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a3'),
    name: 'Mercy',
    age: 36,
    year_of_study: 1,
    gender: 'Female',
    fee_clearance: false
  }
]
test> 
```

Operations 3: Update

- ✓ **Single document Update:**

Update operation: David has cleared his school fee:

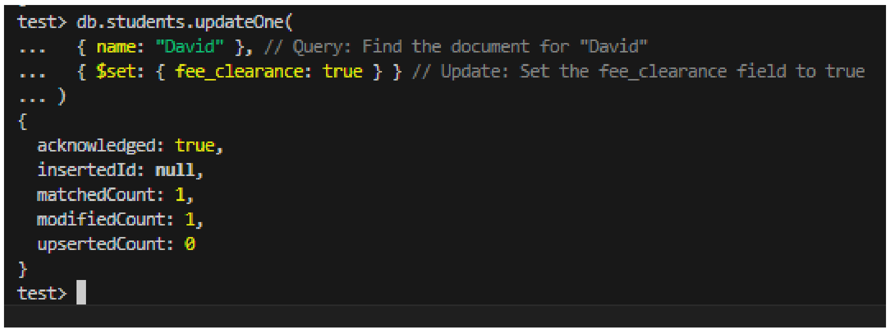
Command: db.students.updateOne(

{ name: "David" }, // Query: Find the document for "David"

{ $set: { fee_clearance: true } } // Update: Set the fee_clearance field to true

)

Returns:



```
test> db.students.updateOne(
...   { name: "David" }, // Query: Find the document for "David"
...   { $set: { fee_clearance: true } } // Update: Set the fee_clearance field to true
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test> 
```

Update operation: Nehemiah has moved to year 3.

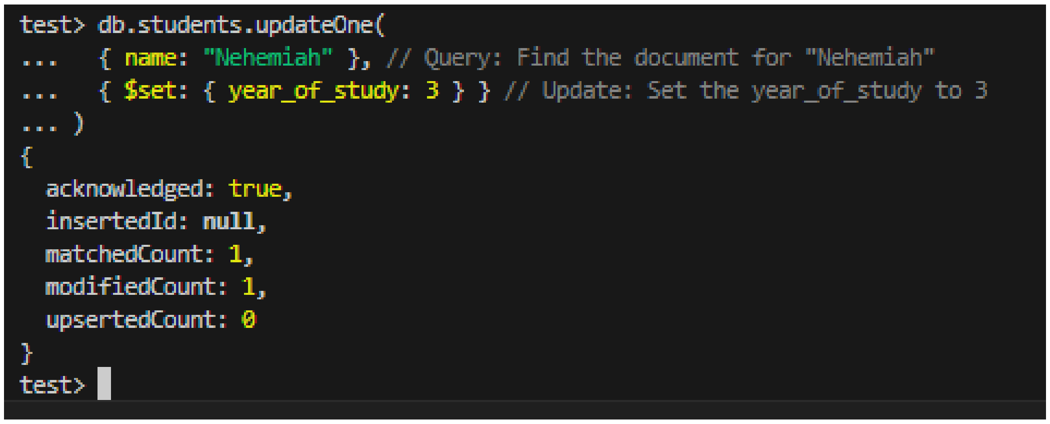
Command: db.students.updateOne(

{ name: "Nehemiah" }, // Query: Find the document for "Nehemiah"

{ $set: { year_of_study: 3 } } // Update: Set the year_of_study to 3

)

Returns



```
test> db.students.updateOne(
...   { name: "Nehemiah" }, // Query: Find the document for "Nehemiah"
...   { $set: { year_of_study: 3 } } // Update: Set the year_of_study to 3
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test> 
```

- ✓ **Multiple document Update**

Update operation: all year one students have moved to year 2.

Command: db.students.updateMany(

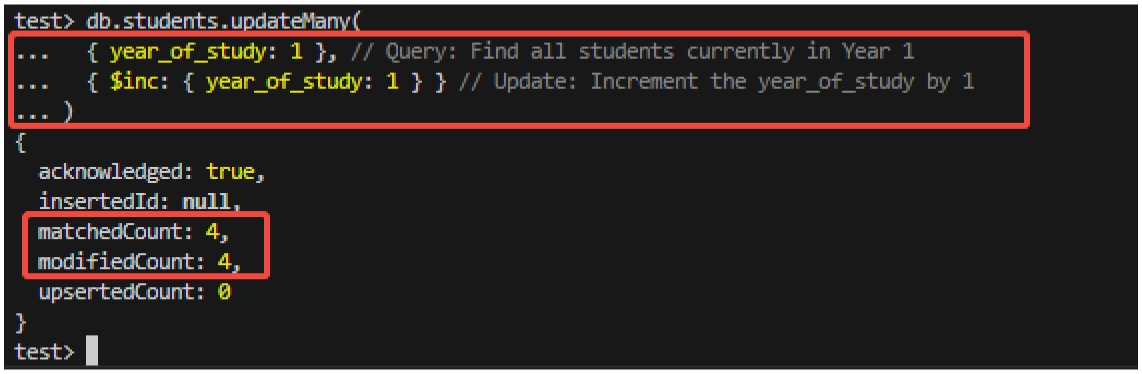
{ year_of_study: 1 }, // Query: Find all students currently in Year 1

{ $inc: { year_of_study: 1 } } // Update: Increment the year_of_study by 1

)

Returns

Elijah, Tonny, Janet, and Mercy moves to year 2

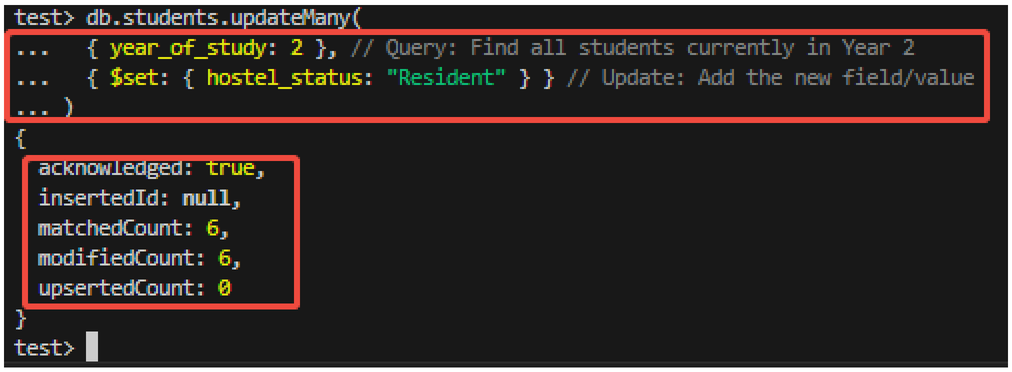


```
test> db.students.updateMany(
...   { year_of_study: 1 }, // Query: Find all students currently in Year 1
...   { $inc: { year_of_study: 1 } } // Update: Increment the year_of_study by 1
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
test> 
```

Update operation: lets add hostel_status: "Resident" to a group of students.i.e all students in year 2

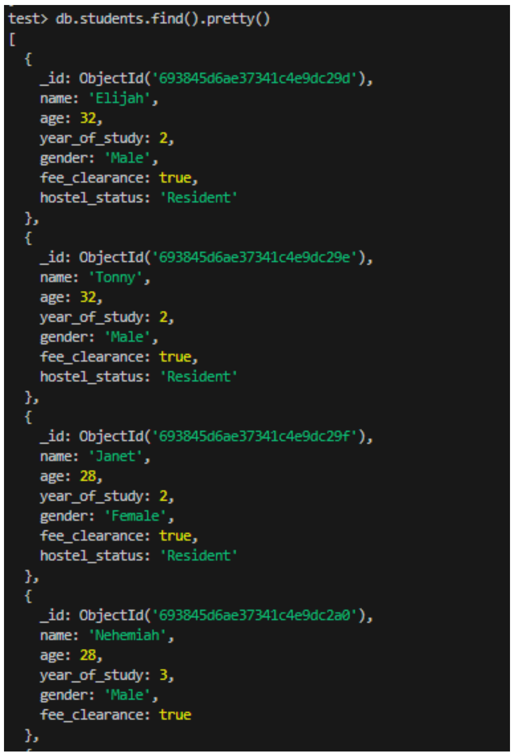
Command: db.students.updateMany({ year_of_study: 2 }, // Query: Find all students currently in Year 2 { $set: { hostel_status: "Resident" } } // Update: Add the new field/value)

Returns

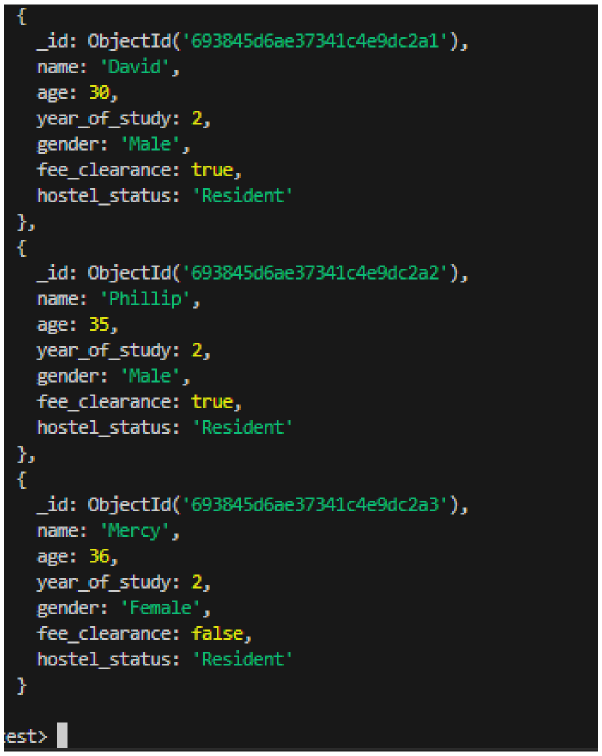


```
test> db.students.updateMany(
...   { year_of_study: 2 }, // Query: Find all students currently in Year 2
...   { $set: { hostel_status: "Resident" } } // Update: Add the new field/value
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 6,
  modifiedCount: 6,
  upsertedCount: 0
}
test> 
```

✓ Finally confirm the updates with a read command.
(In operation 2 of the read function) confirm your updates are as updated.



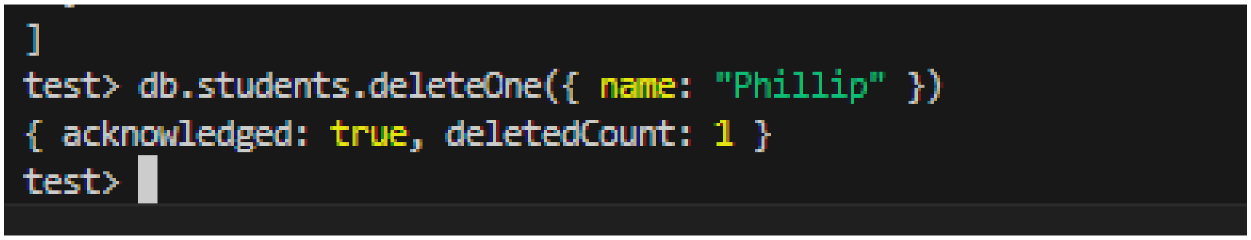
```
test> db.students.find().pretty()
[
  {
    _id: ObjectId('693845d6ae37341c4e9dc29d'),
    name: 'Elijah',
    age: 32,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true,
    hostel_status: 'Resident'
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc29e'),
    name: 'Tonny',
    age: 32,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true,
    hostel_status: 'Resident'
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc29f'),
    name: 'Janet',
    age: 28,
    year_of_study: 2,
    gender: 'Female',
    fee_clearance: true,
    hostel_status: 'Resident'
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a0'),
    name: 'Nehemiah',
    age: 28,
    year_of_study: 3,
    gender: 'Male',
    fee_clearance: true
  },
```



```
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a1'),
    name: 'David',
    age: 30,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true,
    hostel_status: 'Resident'
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a2'),
    name: 'Phillip',
    age: 35,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true,
    hostel_status: 'Resident'
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a3'),
    name: 'Mercy',
    age: 36,
    year_of_study: 2,
    gender: 'Female',
    fee_clearance: false,
    hostel_status: 'Resident'
  }

est>
```

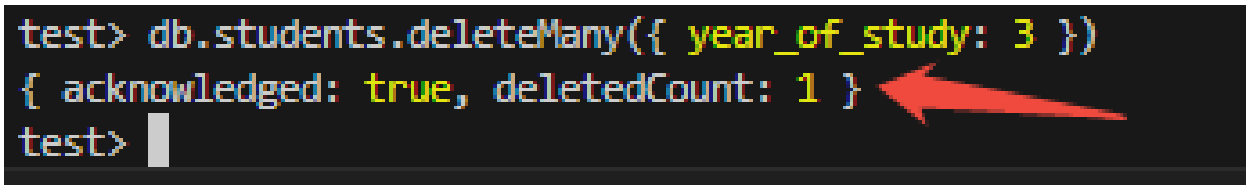
Operation 4: Delete

✓ Single document deletetion: Lets delete Philip alone
Command: db.students.deleteOne({ name: "Phillip" })
Returns



```
]
test> db.students.deleteOne({ name: "Phillip" })
{ acknowledged: true, deletedCount: 1 }
test>
```

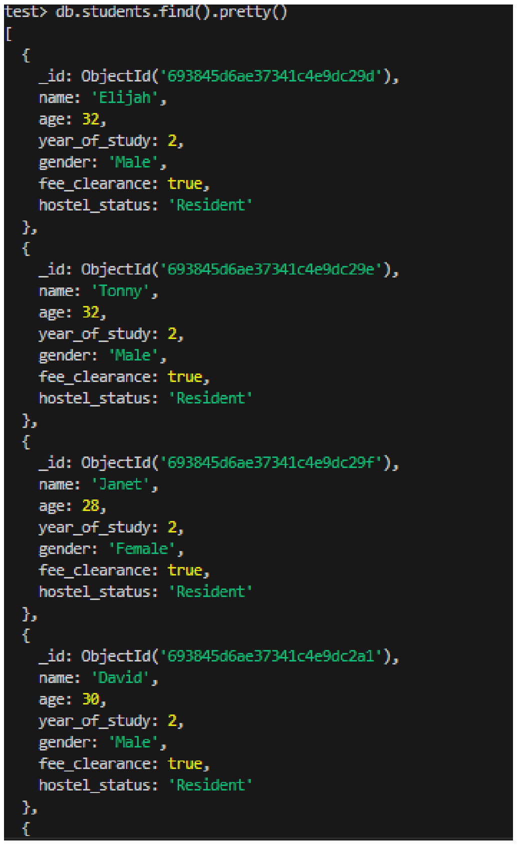
✓ Multiple Documents deletion: Lets delete all students in year 3:
Command: db.students.deleteMany({ year_of_study: 3 })
Returns:



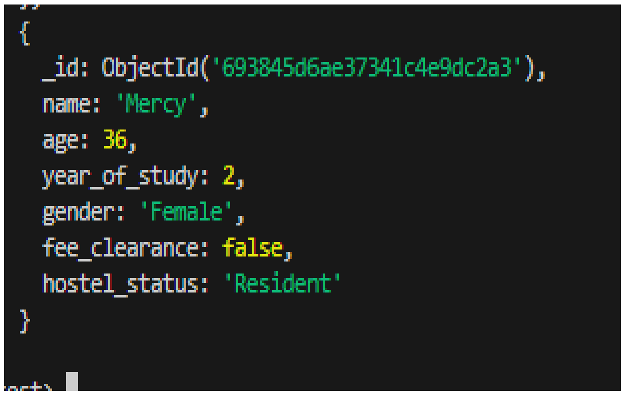
```
test> db.students.deleteMany({ year_of_study: 3 })
{ acknowledged: true, deletedCount: 1 }
test>
```

- ✓ Verify the deletion using the read command

Returns



```
test> db.students.find().pretty()
[
  {
    _id: ObjectId('693845d6ae37341c4e9dc29d'),
    name: 'Elijah',
    age: 32,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true,
    hostel_status: 'Resident'
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc29e'),
    name: 'Tonny',
    age: 32,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true,
    hostel_status: 'Resident'
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc29f'),
    name: 'Janet',
    age: 28,
    year_of_study: 2,
    gender: 'Female',
    fee_clearance: true,
    hostel_status: 'Resident'
  },
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a1'),
    name: 'David',
    age: 30,
    year_of_study: 2,
    gender: 'Male',
    fee_clearance: true,
    hostel_status: 'Resident'
  },
  {
```



```
  {
    _id: ObjectId('693845d6ae37341c4e9dc2a3'),
    name: 'Mercy',
    age: 36,
    year_of_study: 2,
    gender: 'Female',
    fee_clearance: false,
    hostel_status: 'Resident'
  }
```

You find that phillip and Nehemiah stop existing in the database.