**Course Code:** CSE 3513

**Course Name:** NoSQL Data Management

**Course Instructor**: Sunil Sahoo

**Lab Session**: 02

**Activity**: Do lab experiments to perform CRUD (create, read, update and delete).

# Learning Objectives (Los)

**LO1:** Explain the significance of CRUD operations in database systems**. LO2:** Illustrate commands to create new documents within a collection.

**LO3:** Retrieve specific documents or sets of documents using different query parameters.

**LO4:** Update existing documents based on certain conditions.

**LO5:** Delete documents from a collection using specific criteria.

The CRUD operations—Create, Read, Update, and Delete—represent the essential functions needed for data management in database systems, including MongoDB. Here's why they are so crucial:

# Foundational to Data Management

* + **Create:** Without the ability to add new records, a database would be static and quickly become obsolete. "Create" enables the initialization and extension of datasets.
  + **Read:** Reading is perhaps the most frequent operation carried out on databases. The "R" in CRUD allows for data retrieval, making the stored data useful and actionable.
  + **Update:** The ability to update records is essential for maintaining the accuracy and relevance of the dataset. This ensures the database remains a reliable source of information.
  + **Delete:** Over time, some data may become irrelevant, incorrect, or need to be purged for regulatory compliance. The "Delete" operation allows for effective data management and optimization of storage.

# Flexibility and Scalability

* + In MongoDB, the schema-less design allows you to perform CRUD operations without a fixed schema, giving you more flexibility. You can add or remove fields easily.
  + MongoDB is designed to be scaled across many servers, allowing for more effective distribution and partitioning of data. CRUD operations in MongoDB can be made more efficient by utilizing sharding and replication, improving the system's scalability.

# Real-time Applications

* + Real-world applications demand real-time CRUD operations. For example, social media platforms, online marketplaces, and IoT applications require immediate and reliable CRUD operations to manage vast datasets.

# Essential for Business Logic

* + CRUD operations are integral to the business logic of any application. Whether it's adding a new user, updating an inventory item, or deleting an obsolete data entry, CRUD operations form the backbone of most application functionalities.

# User Experience

* + CRUD operations directly impact the user experience. Fast and reliable CRUD operations can enhance the interactivity and responsiveness of an application.

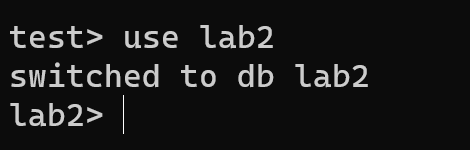
# Integration and APIs

* + CRUD operations often form the basis for RESTful APIs, which allow for seamless interaction between different services and applications.

**Summary:** CRUD operations are the building blocks of any database system, acting as the interface between the user or application and the database. Proper implementation and optimization of CRUD operations are critical for the effective and efficient use of MongoDB or any other database system.

**CREATE Operation in MongoDB Step 1:** Create a new database

* + Use the use command followed by the name of the database you want to switch to (or create).
  + If "lab2" doesn't exist, MongoDB won't create it immediately, but it will switch the context to that database in the shell.



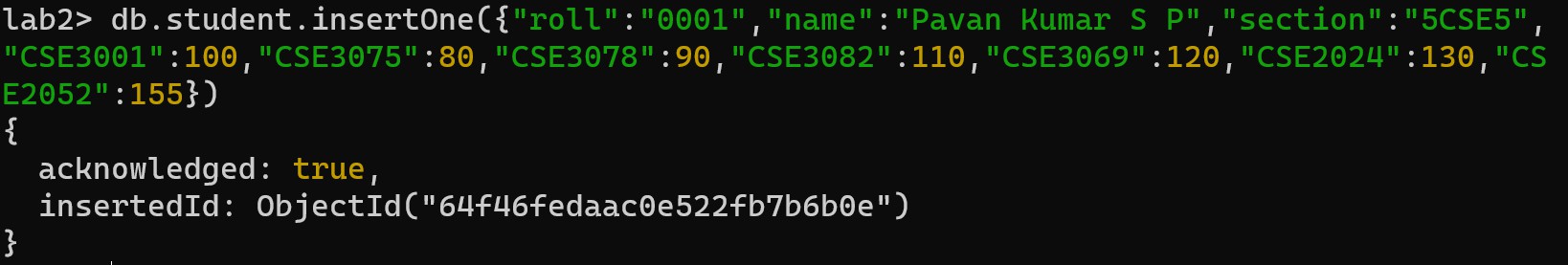
**Step 2:** Create a new Collection in the database

* + You can create a collection using the createCollection() method

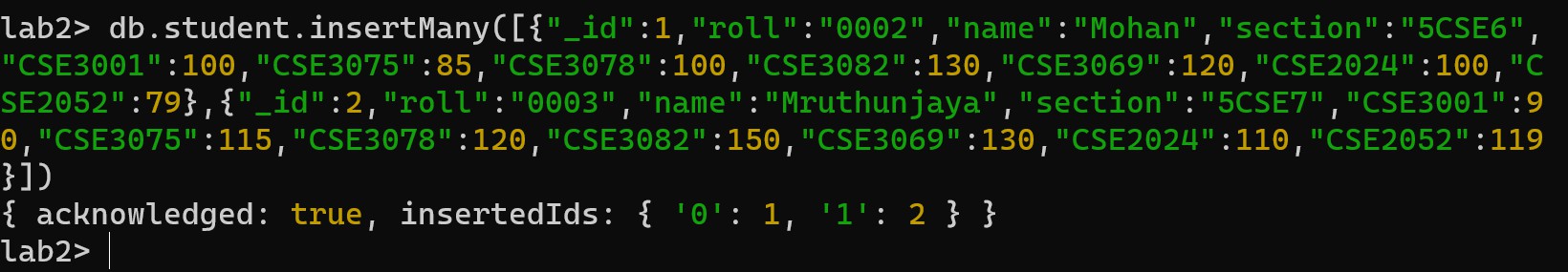


**Step 3:** Insert a new Document into the collection created in the previous step.

* + Inserting a new document into a MongoDB collection can be achieved using several methods, but the most common ones are **insertOne()** and **insertMany().**
  + **InsertOne Method-** This method is used to insert a single document into a collection.



* + **insertMany()-**This method allows you to insert multiple documents into a collection with a single command.



# READ /RETRIEVAL operation in MongoDB

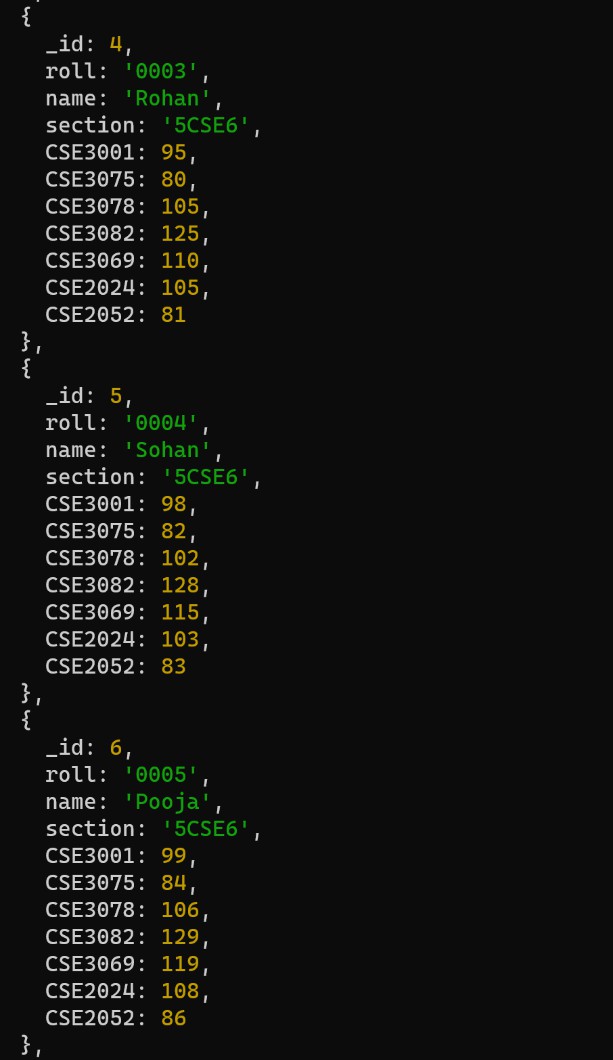
In MongoDB, the primary method to read or retrieve data from collections is using the find() method. Alongside find(), there's the findOne() method, which returns only one document as the name suggests.

**Syntax :** db.collectionName.find(query, projection)

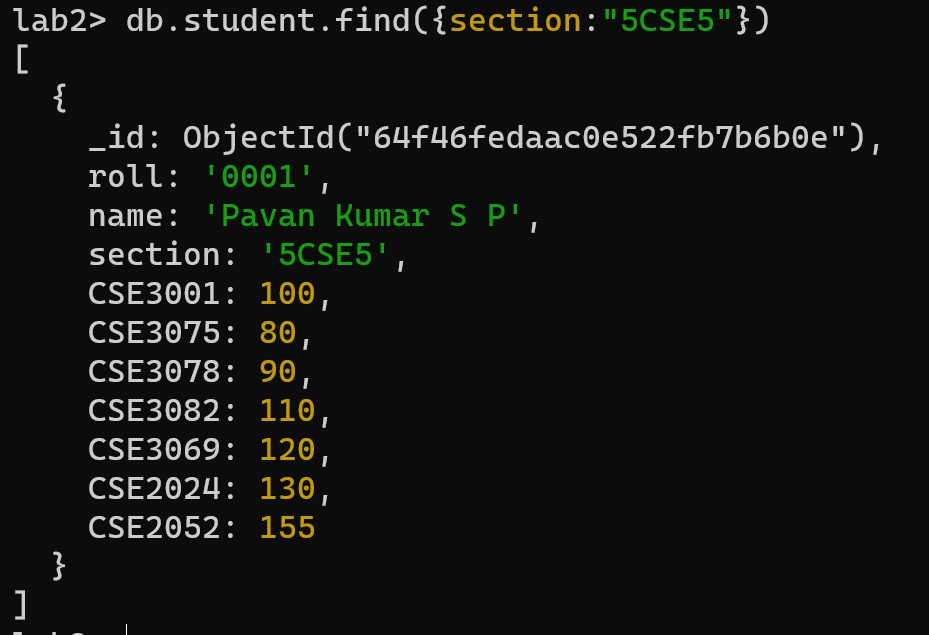
**query:** (Optional) This is a filter for searching specific documents. If you omit this, all documents in the collection are returned.

**projection:** (Optional) This defines which fields to include or exclude from the results. A value of 1 indicates inclusion, and 0 indicates exclusion.

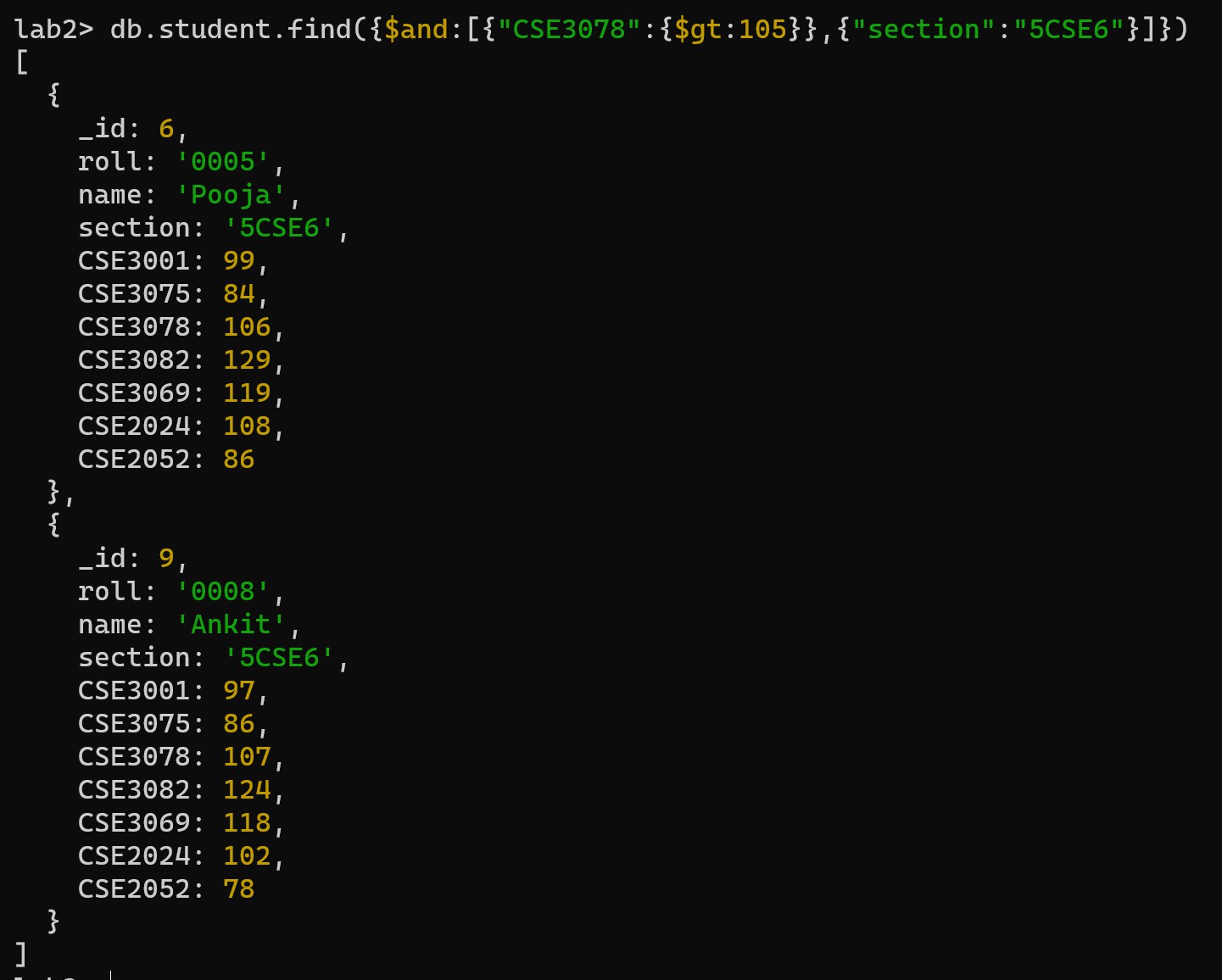
**Retrieve all documents from the students collection:** db.student.find()



**With Query:** Retrieve students section :5CSE5



With Query : Retrieve students section:5CSE6 and CSE3078>105



**Examples:** comparison operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **MongoDB Query** | **Explanation** |
| **$gt** | Greater Than | db.student.find({ CSE3001: {  $gt: 100 } }) | Finds all students who have scored more than 100 in subject CSE3001. |
| **$gte** | Greater Than or Equal | db.student.find({ CSE3075: {  $gte: 100 } }) | Retrieves students who have scored 100 or more in subject CSE3075. |
| **$lt** | Less Than | db.student.find({ CSE3078: {  $lt: 90 } }) | Selects students who have a score of less than 90 in CSE3078. |
| **$lte** | Less Than or Equal | db.student.find({ CSE3082: {  $lte: 80 } }) | Finds students with scores of 80 or less in subject CSE3082. |
| **$ne** | Not Equal | db.student.find({ CSE3069: {  $ne: 110 } }) | Selects all students who did not score 110 in subject CSE3069. |
| **$in** | In Array | db.student.find({ CSE2024: {  $in: [100, 110, 120] } }) | Retrieves students who scored  either 100, 110, or 120 in subject CSE2024. |
| **$nin** | Not In Array | db.student.find({ CSE2052: {  $nin: [85, 89] } }) | Fetches students who have scores other than 85 and 89 in CSE2052. |

**Examples:** Logical operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **MongoDB Query** | **Explanation** |
| **$and** | Logical AND | db.student.find({ $and: [{ CSE3001: { $gt: 90 } }, {  CSE3075: { $lt: 100 } }] }) | Finds students who scored more than 90 in CSE3001 AND scored less than 100 in CSE3075. |
| **$or** | Logical OR | db.student.find({ $or: [{  CSE3001: { $gt: 90 } }, {  CSE3075: { $lt: 100 } }] }) | Retrieves students who either scored more than 90 in CSE3001 OR scored less than 100 in CSE3075. |
| **$not** | Logical NOT | db.student.find({ CSE3001: {  $not: { $lt: 90 } } }) | Selects students who did not score less than 90 in CSE3001. In essence, this returns students who scored 90 or more. |
| **$nor** | Logical NOR | db.student.find({ $nor: [{ CSE3001: { $gt: 90 } }, {  CSE3075: { $lt: 100 } }] }) | Finds students who neither scored more than 90 in CSE3001 NOR scored less than 100 in CSE3075. |

# UPDATE operation in MongoDB

The update operation in MongoDB can be used to modify existing documents in a collection. The updateOne(), updateMany(), and replaceOne() methods are commonly used for this purpose.

1. **updateOne()-**The updateOne() method updates the first document that matches the given filter. If multiple documents match the filter, only the first occurrence is updated. **Syntax :** db.collection.updateOne(filter, update, options)

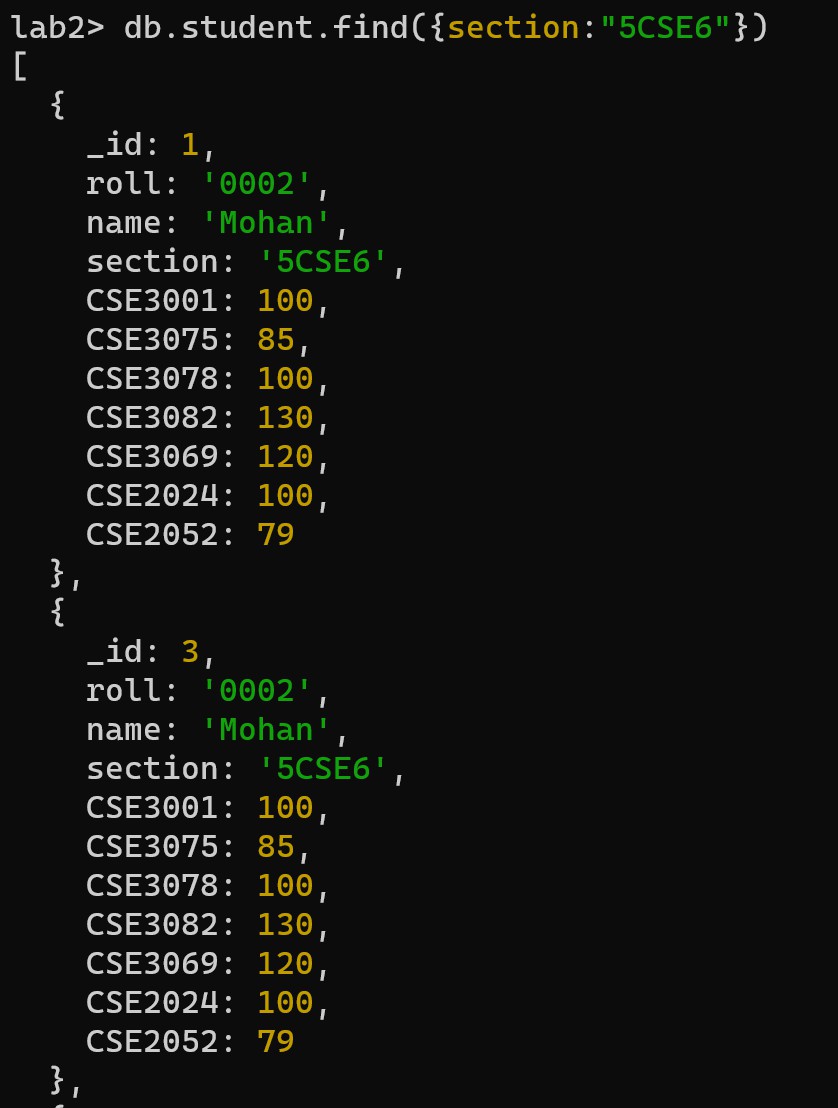
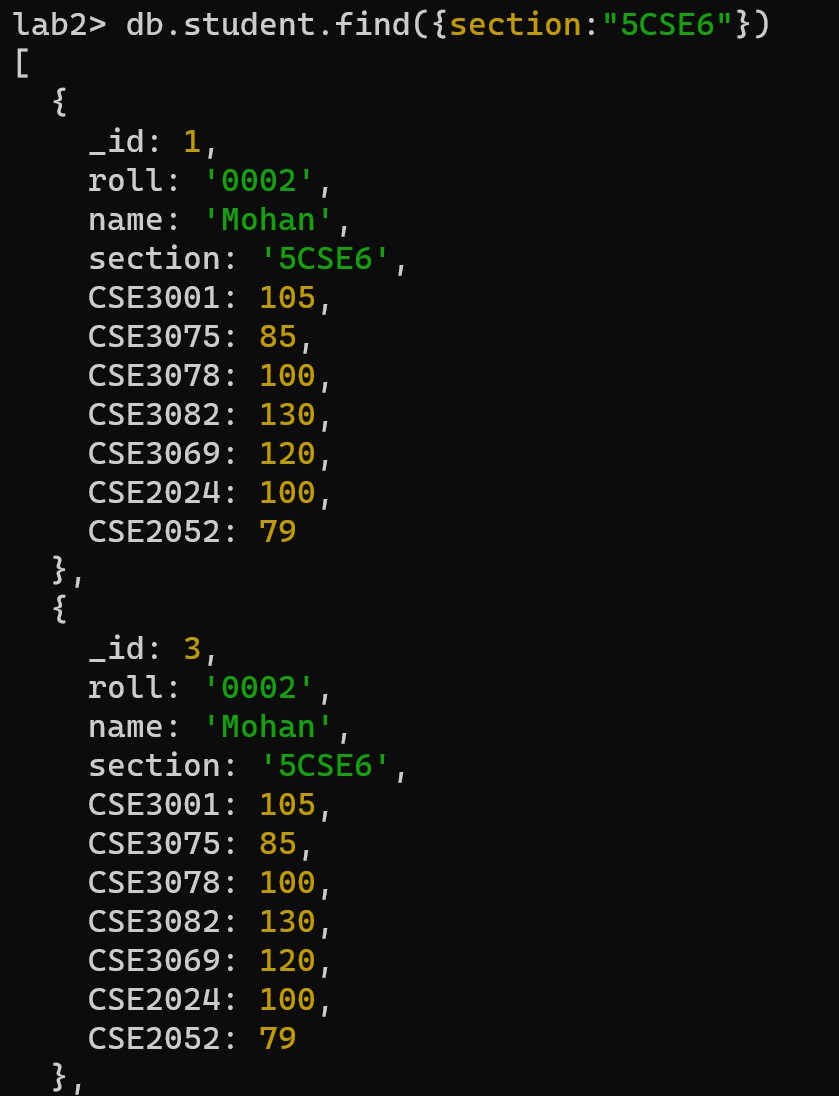
**Example:** Let's say you want to update the section of the student with roll "0001" to '5CSE4'.



1. **updateMany() -** method updates all documents that match the given filter**. Syntax :** db.collection.updateMany(filter, update, options)

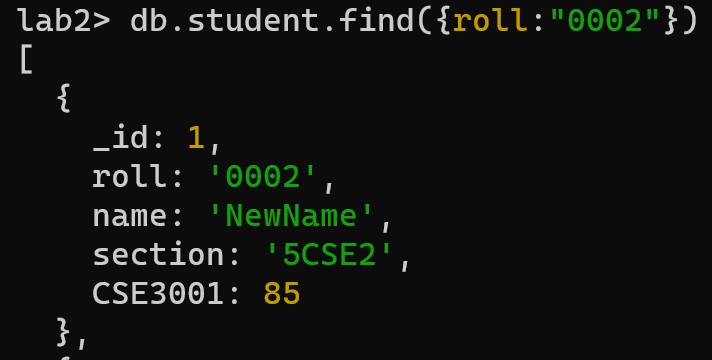
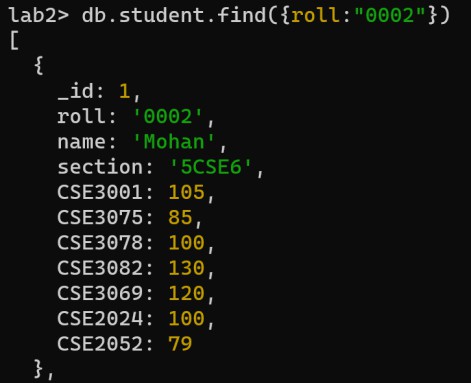
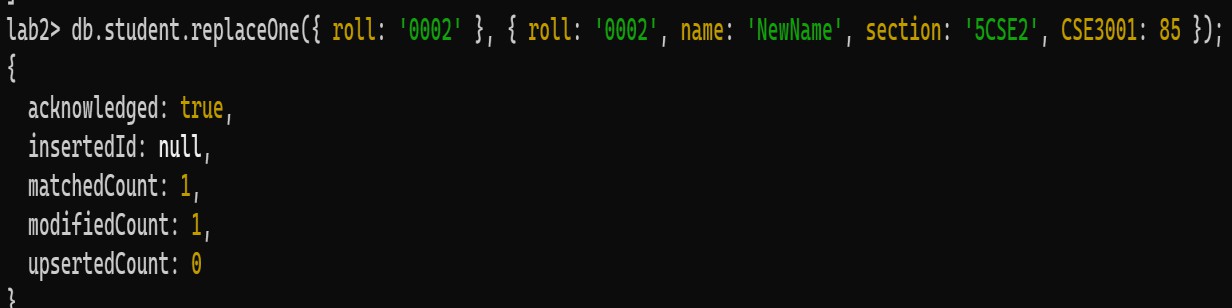
**Example :** Increase the CSE3001 score by 5 marks for all students in section '5CSE6'.



1. **replaceOne()-**The replaceOne() method replaces the first document that matches the filter with a new document. The replacement document must only contain field/value pairs; to include operators such as $set, use the updateOne() method instead.

**Syntax :** db.collection.replaceOne(filter, replacement, options)

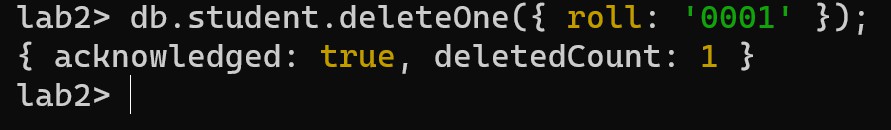
**Example:** Replace the student data with roll "0002" with a new set of data

# DELETE Operations in MongoDB

In MongoDB, you can perform deletion operations on documents using the deleteOne() and deleteMany() methods. These methods allow you to remove documents from a collection based on certain criteria.

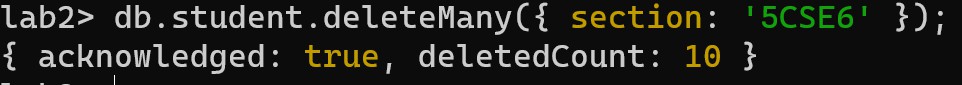
1. **deleteOne()**- method deletes the first document that matches the provided filter. If multiple documents match the criteria, only the first one encountered (according to natural order) will be removed.

**Syntax:** db.collection.deleteOne(filter, options)

**Example:** Delete the student with the roll "0001"

1. **The deleteMany()** method removes all documents that match the provided filter.

**Syntax :** db.collection.deleteMany(filter, options)

**Example :** Delete all students in the section '5CSE6'

# Lab challenge 1: Library Management System Background:

A local community library aims to digitize its operations to efficiently manage books and patrons. The new system should allow the librarian to add new books, update book details, delete books, and retrieve details about books. Additionally, the librarian should be able to manage patron information.

# Requirements:

1. **Book Entity**
   * Attributes: BookID, Title, Author, Genre, PublishedDate, ISBN, AvailableCopies
   * **C - Create/Add a new book**- A function to input the details of a new book and add it to the library's digital collection.
   * **R - Retrieve/View book details**.- A function to input a BookID or Title and get the corresponding book's details.
   * A function to list all books in the library.
   * A function to list books based on Author or Genre.
   * **U - Update book details**.-A function to update the details of an existing book using its BookID. This can include updating the number of AvailableCopies when a book is borrowed or returned.
   * **D - Delete a book.** A function to remove a book from the library's collection using its BookID.

# Patron Entity

* + Attributes: PatronID, Name, Email, Phone, MembershipDate
  + **C - Register a new patron**- A function to input the details of a new patron and add them to the library's database.
  + **R - Retrieve/View patron details**.- A function to input a PatronID or Email and get the corresponding patron's details.
  + A function to list all patrons in the library.
  + **U - Update patron details**- A function to update an existing patron's contact details or other information using their PatronID.
  + **D - Delete patron details**.- A function to remove a patron from the library's database using their PatronID.

# Real-Time Scenario:

Imagine that the librarian receives new books to add to the library's collection. Using the digital system, the librarian can easily input the book details, and they get added to the database. When a patron borrows a book, the AvailableCopies attribute for that book is updated. If a patron damages a book beyond repair, the librarian can remove that book's record from the system. Additionally, when new members join the library, their details can be efficiently stored, and if they move out of the community, their records can be deleted.

Happy Learning