## National University of Computer & Emerging Sciences, Karachi



## FAST School of Computing –Artificial Intelligence Department

**Fall 2024, Lab Manual – 11**

|  |  |
| --- | --- |
| **Course Code: CL-2005** | **Course: Database Systems Lab** |
| **Instructor(s):** | **Filza Akhlaq, Sohail Ahmed** |

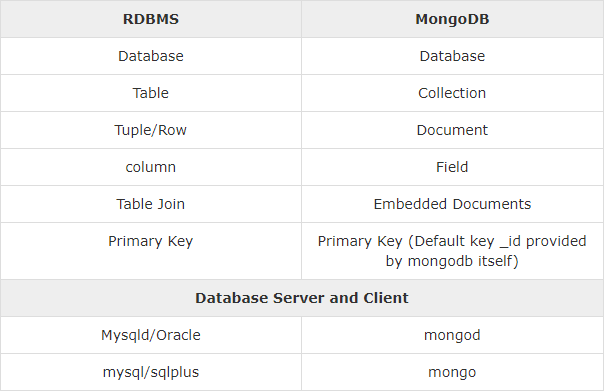
**Contents:**

1. Overview of MongoDB
2. Difference in terminology of MongoDB
3. Installation of MongoDB
4. Designing Schema in MongoDB
5. Some important methods in MongoDB
6. Creating Database
7. Creating collections
8. Inserting single/multiple Documents
9. Querying, Deleting & updating of Documents
10. Logical Operations
11. Implementation of where clause

**Overview of MongoDB**

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. MongoDB is a cross-platform, document-oriented database that provides high performance, high availability, and easy scalability. MongoDB works on the concept of collection and documents.

**Difference in Terminology of MongoDB**



**Figure 1. Difference between RDBMS & MongoDB**

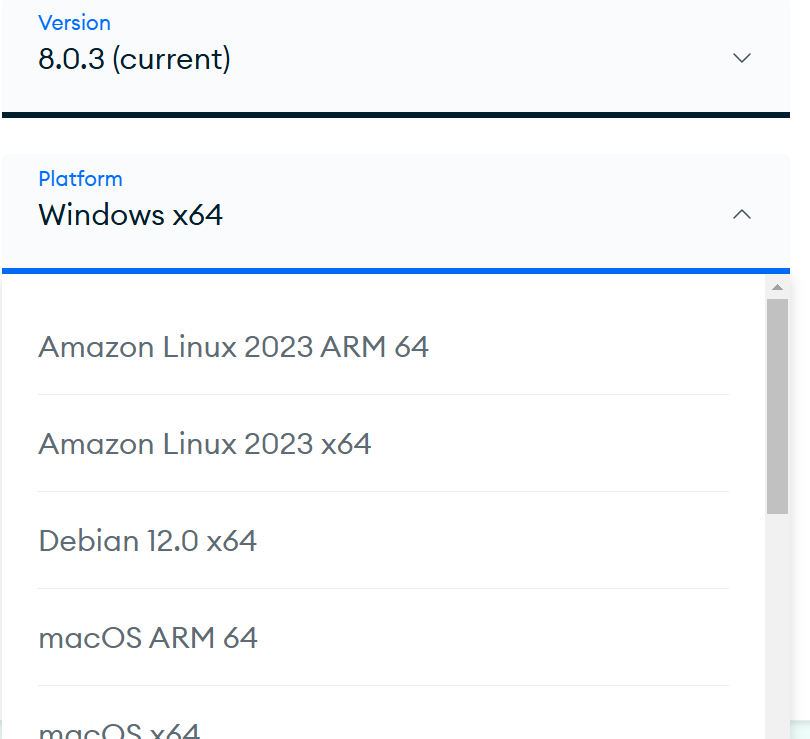
**Database:** Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

**Collection:** Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are for similar or related purposes.

**Document:** A document is a set of key-value pairs. Documents have a dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

**Installing MongoDB on Windows**

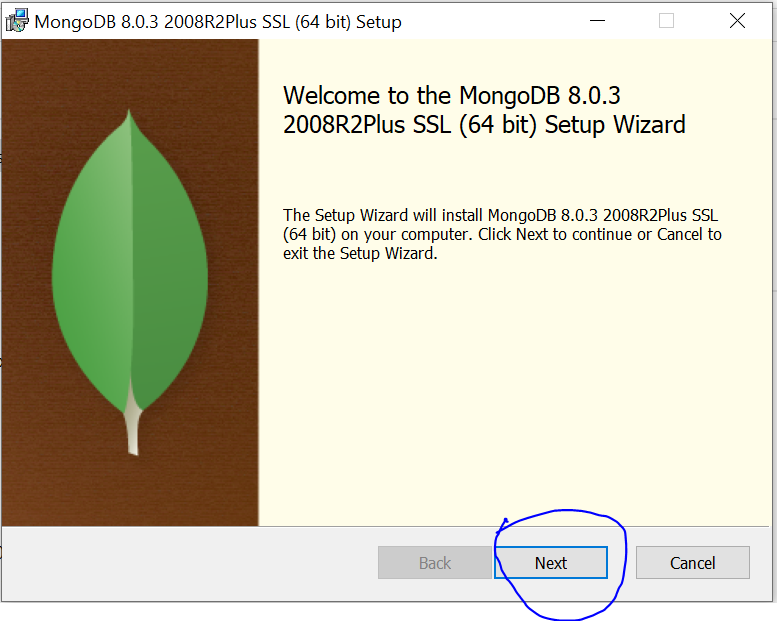
Note for non-window users: You can choose your platform from this dropdown menu while installation:



We can use MongoDB in the following ways:

1. MongoDB on CLI
2. MongoDB Compass (we will use this one)

As for the server, we can either use MongoDB Atlas (Cloud based) or host locally (we will do this one). The community server can be installed from [here.](https://www.mongodb.com/try/download/community) After running the .msi setup, we will get the following screen:



A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

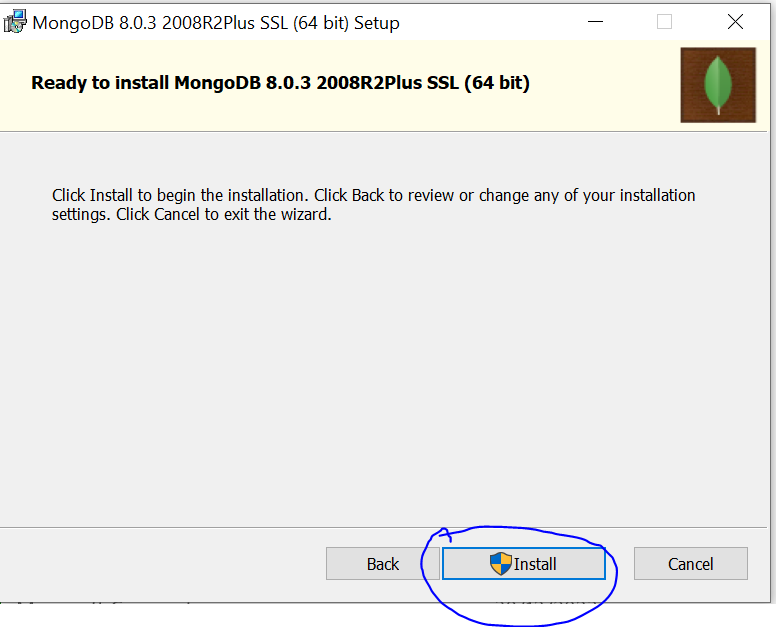
Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

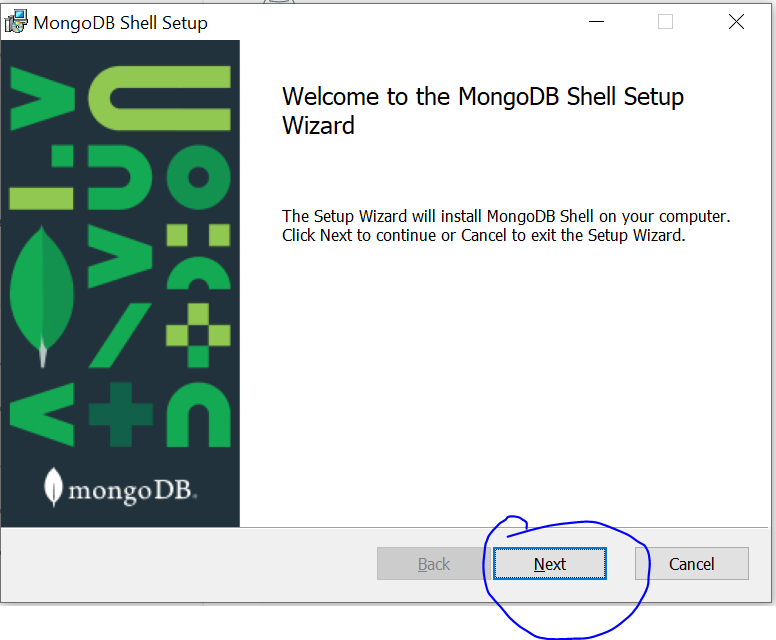


Let the setup finish. This might take a few minutes. If MongoDB Compass does not auto-start after this, start it on your own then open it on your own and minimize it for now. Now we are going to install MongoDB Shell. Under the tools section, download the [MongoDB Shell](https://downloads.mongodb.com/compass/mongosh-2.3.3-x64.msi) (also called mongosh):

A screenshot of a computer

Description automatically generated

Open the .msi setup and follow the steps below:



Keep the destination folder default and install it for all users.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Now that you have installed Mongosh, open CMD and type mongosh and you should see the following:

A screenshot of a computer

Description automatically generated

We will use Mongosh and MongoDB Compass together to visualize what we are doing in this lab. Keep the shell running and go back to Compass:

A screenshot of a computer

Description automatically generated

Enter a name for your connection, do not edit the connection string, and Save and Connect:

A screenshot of a computer

Description automatically generated

You should now be able to see the connection in the left pane of MongoDB Compass:

A screenshot of a search box

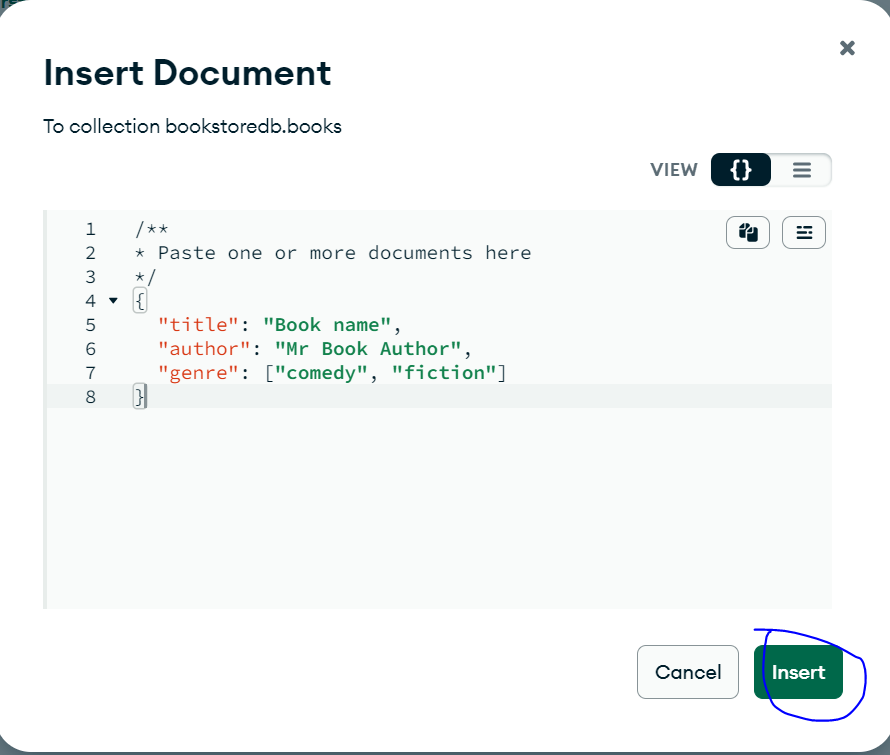
Description automatically generated

Creating a new database named *bookstoredb* using the + icon which you will see when you hover over *myfirstcinnection*.

A screenshot of a computer

Description automatically generated

Now, find the option “Add data” and choose “Insert Document”, then add the following JSON object in the tables’ collection:



Notice that Compass will not let you click “Insert” until your code is error-free. Even if we remove the id field, Mongo automatically adds a unique id to each document on its own:

A screenshot of a computer

Description automatically generated

Insert the following documents in the table collection:

[

{

"title": "A tale of two cities",

"author": "Charles Dickens",

"genre":["historical", "fiction"]

},

{

"title": "The Alchemist",

"author": "Paulo Coelho",

"genre":["fantasy"]

},

{

"title": "Harry Potter and the Philosopher's Stone",

"author": "J. K. Rowling",

"genre": ["children fantasy"]

}

]

Experiment with the Filter and other options.

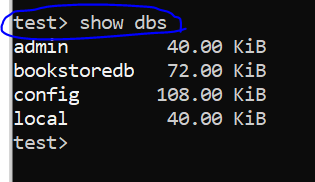
Now we will do the same things as above using Mongosh. We can use mongosh either inside the Compass like this:

A screenshot of a computer

Description automatically generated

or, we can use the command line directly as previously seen.

1. For seeing all the available databases:



1. For selecting a database to work with:

A black background with white text

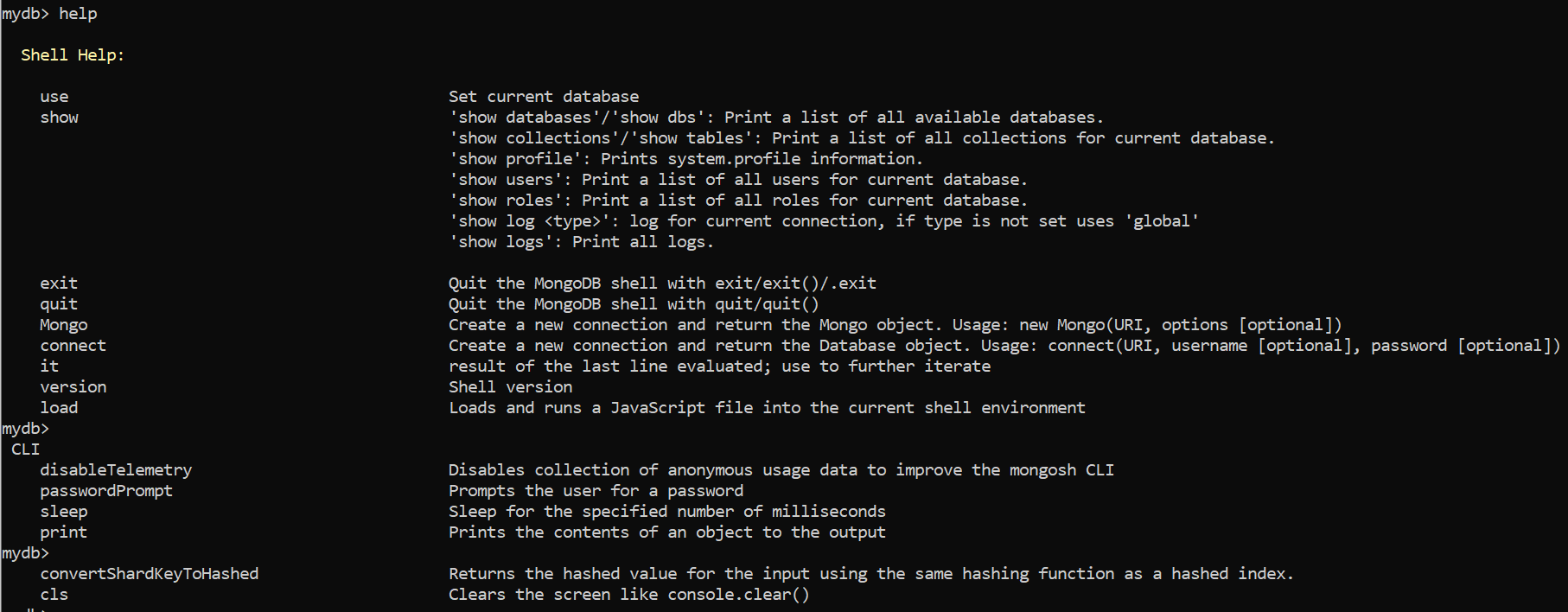
Description automatically generated and you can see that now we are in the bookstoredb>

1. You can go back to test> by simply writing use test.
2. Whenever you use the “use db\_name” command, mongosh will create a db with *db\_name* if it does not exist only after you add some collections/data to this database. For example, after I wrote >use anydatabasenamehere, I get this output even though a database with this name does not exist.

A black background with white text

Description automatically generated

1. You can use *help* command and it will show you all the functions that you can use:



1. Using *db.help()* would return you even more database level functions that you can use:

A computer screen shot of text

Description automatically generated

**Some Considerations while designing Schema in MongoDB**

1. Design your schema according to user requirements.
2. Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should be no need of joins).
3. Duplicate the data (but limited) because disk space is cheap as compared to compute time.
4. Use joins while writing, not on reading the data.
5. Optimize your schema for most frequent use cases.
6. Do complex aggregation in the schema.

**Example**

Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. The website has the following requirements:

1. Every post has a unique title, description and URL.
2. Every post can have one or more tags.
3. Every post has the name of its publisher and total number of likes.
4. Every post has comments given by users along with their name, message, data-time and likes.
5. On each post, there can be zero or more comments

In RDBMS schema, a design for the above requirements will have a minimum of three tables:

A screenshot of a computer

Description automatically generated

Whereas in a MongoDB schema, design will have one collection post and the following structure:

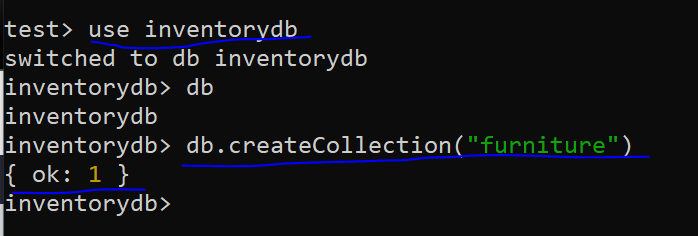
A screenshot of a computer code

Description automatically generated

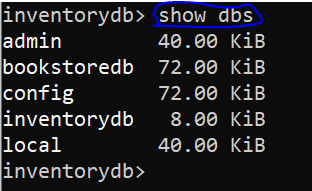
So, while showing the data, in RDBMS you need to join three tables and in MongoDB, data will be shown from one collection only.

**Creating and querying database using Mongosh:**

1. db.createCollection(“*collection\_name\_here*”) is used to add a collection to the database being used inside *db*.



1. Now that we have added a collection in the database “inventory”, we can now see it when we use *show dbs.*

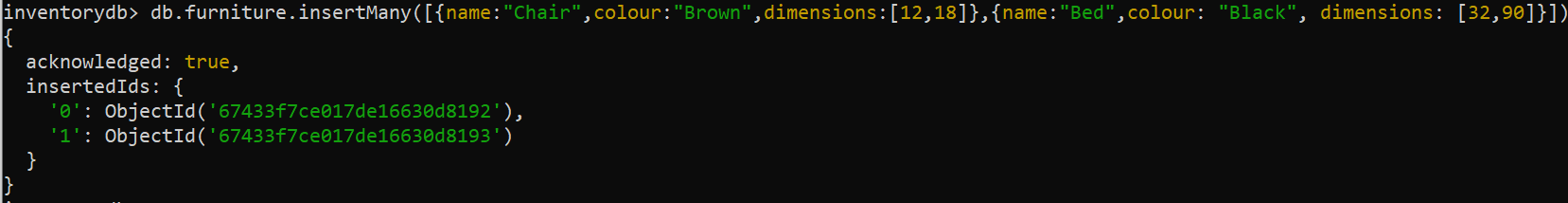


1. Adding documents: insertOne() or insertMany() functions are used to insert one or multiple documents into a collection.

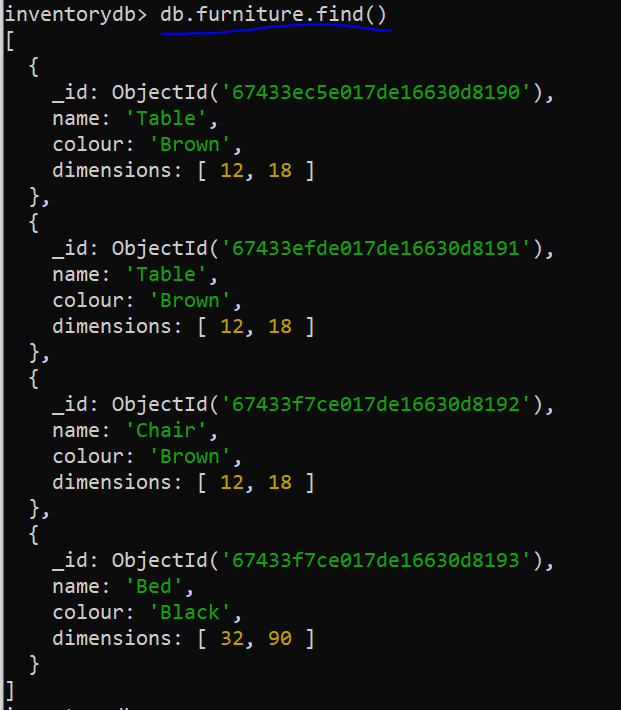
A black screen with green and yellow text

Description automatically generated

The insertOne() method returns an object which shows us true/false for successful insertion and returns the ObjectId of the document we added. insertMany() works similarly, we have to pass an array of objects into it [{},{},…,{}]:



1. Now lets query our documents in the *furniture* collection. find() method will return all the records if no arguments are given:



**RDMS WHERE clause equivalents in MongoDB:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Syntax** | **Example** | **RDBMS**  **Equivalent** |
| **Equals to** | {<key>:<value>} | db.mycollection.find({"by":"Amin Sadiq"}) | where by = ‘Amin Sadiq' |
| **Less Than** | {<key>:{$lt:<value>}} | db.mycollection.find({"likes":{$lt:50}}) | where likes < 50 |
| **Less Than and Equals** | {<key>:{$lte:<value>}} | db.mycollection.find({"likes":{$lte:50}}) | where likes <= 50 |
| **Greater Than** | {<key>:{$gt:<value>}} | db.mycollection.find({"likes":{$gt:50}}) | where likes > 50 |
| **Greater Than and Equals** | {<key>:{$gte:<value>}} | db.mycollection.find({"likes":{$gte:50}}) | where likes >= 50 |
| **Not Equals to** | {<key>:{$ne:<value>}} | db.mycollection.find({"likes":{$ne:50}}) | where likes != 50 |
| **AND** |  | db.mycol.find({$and: [{key1: value1}, {key2:value2}]}) |  |
| **OR** |  | db.mycol.find({$or: [{key1: value1}, {key2:value2}]}) |  |

**Examples:**

1. Find a document where name is equals to Bed:

A computer screen shot of a code

Description automatically generated

1. Find a document where dimensions or dimensions[0] is greater than 30:

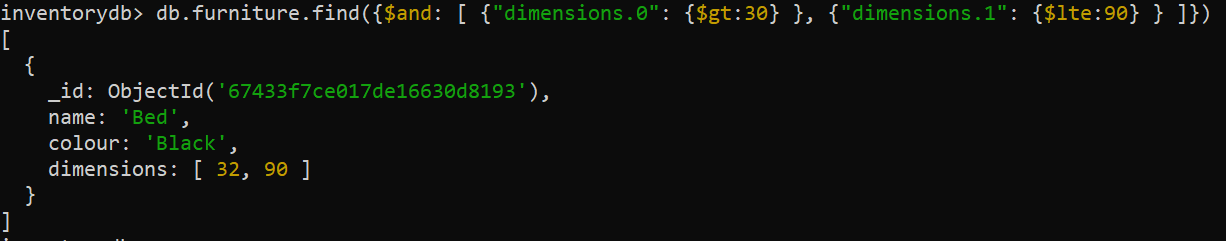
A computer screen shot of a code

Description automatically generated

1. Using AND:

A computer screen with green text

Description automatically generated



1. Using OR:

A computer screen with numbers and letters

Description automatically generated

1. Not equals to:

A computer screen with numbers and letters

Description automatically generated

1. Using AND and OR together (finding those collections where colour is brown, and the name is either table or chair):

A computer screen shot of a black screen

Description automatically generated

**Updating Documents:**

Syntax for updating one document: db.*COLLECTION\_NAME*.updateOne(*SELECTION\_CRITERIA*, *UPDATED\_DATA*)

Syntax for updating multiple document: db.*COLLECTION\_NAME*.updateMany(*SELECTION\_CRITERIA*, *UPDATED\_DATA*)

Let’s update the colour of a furniture to Ivory where dimensions are [32, 90]:

A black screen with yellow text

Description automatically generated

Now, let’s update all the furniture with brown colour and change it to Dark Brown:

A screen shot of a computer

Description automatically generated

**Deleting Documents:**

Syntax for deleting one document: db.COLLECTION\_NAME.deleteOne(DELETION\_CRITERIA)

Syntax for deleting multiple document: db.COLLECTION\_NAME.deleteMany(DELETION\_CRITERIA)

Delete a furniture document where name is Chair:

A black background with white text

Description automatically generated

Delete all the documents where dimension is [12, 18]:



**Dropping a collection:**

Syntax: db.*collection\_name*.drop()

A black background with white text

Description automatically generated

Let’s view the collections now:

A black background with white text

Description automatically generated

Returns nothing, which means collection is dropped.

**Dropping a database using db.dropDatabase():**

A screen shot of a computer code

Description automatically generated

**Some other functions [Comparison, aggregate, index, searching etc]**

### Lab Tasks

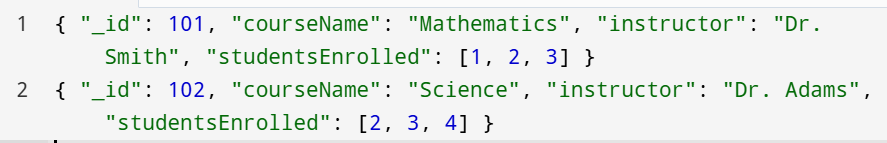
### Submission instructions: Perform the following tasks on Mongosh. Submit your query text along with screenshot of query + output.

1. Create a database named SchoolDB.
2. Create two collections:
   * Students
   * Courses
3. Insert the following documents into the Students collection:

A screenshot of a computer

Description automatically generated

1. Insert the following documents into the Courses collection:



1. Use findOne to retrieve:
   * A student where the math score is >= 85 **and** the age is < 22.
   * A course where the studentsEnrolled array includes 3 **and** the instructor is "Dr. Adams".
2. Use find to retrieve:
   * Students with math score >= 80 **and** science score < 90.
   * Students whose age is < 23 **or** have a math score >= 85.
   * Students with science score >= 80 **and** (either math score < 75 or age > 22).
3. Use updateOne to:
   * Increase the science score of the student where name is "Bob" and math score is >= 75.
4. Use updateMany to:
   * Increase the math score by 5 for students whose science score is < 80 **and** age > 22.
5. Use deleteOne to:
   * Remove a student where name is "Daisy" **and** their science score is < 80.
6. Use deleteMany to:
   * Remove courses where the studentsEnrolled array includes 2 **or** the instructor is "Dr. Smith".
7. Drop the Students collection.
8. Drop the Courses collection.
9. Finally, delete the SchoolDB database.

**1. Counting Documents**

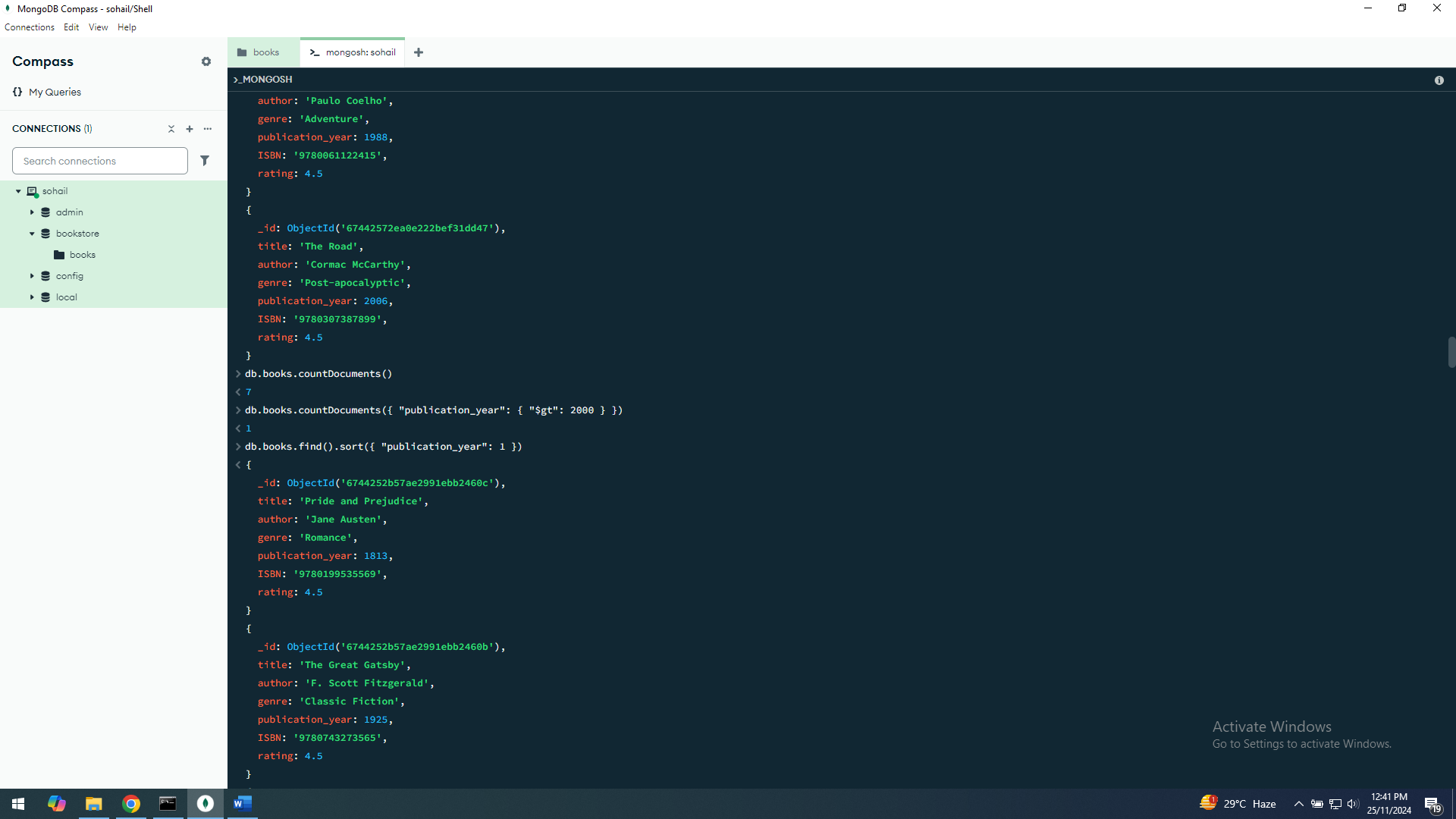
Count the number of documents that match a query or count all documents in a collection.

* **Count All Documents**:

db.books.countDocuments()

* **Count Documents with a Filter** (e.g., count all books published after 2000):

db.books.countDocuments({ "publication\_year": { "$gt": 2000 } })



**2. Sorting Results**

Sort query results by one or more fields.

* **Sort by Publication Year in Ascending Order**:

db.books.find().sort({ "publication\_year": 1 })

A screenshot of a computer program

Description automatically generated

* **Sort by Publication Year (Descending) and Title (Ascending)**:

db.books.find().sort({ "publication\_year": -1, "title": 1 })

A screen shot of a computer

Description automatically generated

**3. Limiting and Skipping Results**

Control the number of documents returned.

* **Limit the Number of Results** (e.g., only return the first 5 books):

db.books.find().limit(5)

* **Skip a Number of Results** (e.g., skip the first 3 books and return the next ones):

db.books.find().skip(3)

* **Combine Skip and Limit for Pagination** (e.g., get books from the second page of results assuming 5 results per page):

db.books.find().skip(5).limit(5)

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**4. Aggregation Pipeline**

Perform complex data transformations and analysis using aggregation.

* **Find the Average Publication Year of All Books**:

db.books.aggregate([

{ "$group": { "\_id": null, "avgPublicationYear": { "$avg": "$publication\_year" } } }

])

* **Group by Genre and Count Books in Each Genre**:

db.books.aggregate([

{ "$group": { "\_id": "$genre", "count": { "$sum": 1 } } }

])

* **Sort Genres by Number of Books in Descending Order**:

db.books.aggregate([

{ "$group": { "\_id": "$genre", "count": { "$sum": 1 } } },

{ "$sort": { "count": -1 } }

])

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

**5. Projection**

Control which fields are returned in query results.

* **Return Only Title and Author**:

db.books.find({}, { "title": 1, "author": 1, "\_id": 0 })

* **Exclude ISBN Field**:

db.books.find({}, { "ISBN": 0 })

A screen shot of a computer

Description automatically generated

**6. Text Search**

If you create a text index on fields like title or author, you can perform text searches.

* **Create a Text Index** (this is a one-time setup):

db.books.createIndex({ "title": "text", "author": "text" })

* **Search for Books with the Word "Road" in Title or Author**:

db.books.find({ "$text": { "$search": "Road" } })

A screen shot of a computer program

Description automatically generated

**7. Find Documents Using Regular Expressions**

Search for documents with partial text matches.

* **Find Books with Titles Starting with "The"**:

db.books.find({ "title": { "$regex": "^The", "$options": "i" } })

* **Find Books by Authors with Last Name "Lee"**:

db.books.find({ "author": { "$regex": "Lee$", "$options": "i" } })

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**8. Update with Increment/Decrement**

Increase or decrease numeric values directly.

* **Increase the Rating of All Books by 1**:

db.books.updateMany({}, { "$inc": { "rating": 1 } })

* **Decrease the Publication Year by 5 for a Specific Book**:

db.books.updateOne({ "title": "1984" }, { "$inc": { "publication\_year": -5 } })

A screen shot of a computer

Description automatically generated

**9. Using findOneAndUpdate and findOneAndDelete**

Find and modify or delete a document in one atomic operation.

* **Find a Book by Title and Update Its Genre**:

db.books.findOneAndUpdate(

{ "title": "The Great Gatsby" },

{ "$set": { "genre": "Classic" } },

{ "returnNewDocument": true }

)

* **Find a Book by Title and Delete It**:

db.books.findOneAndDelete({ "title": "The Catcher in the Rye" })

**Tasks:**

**Count Books by a Specific Author**

* Count the number of books written by "George Orwell."

**2. Find Books Published After a Certain Year**

* Retrieve all books published after the year 2000.

**3. Update the Genre of a Book**

* Change the genre of "The Catcher in the Rye" to "Classic Fiction."

**4. Increase Rating for All Books by 0.5**

* Increase the rating field of all books by 0.5 points.

**5. Find Books Matching a Keyword**

* Perform a text search for books that contain the keyword "Great" in the title or author.

**6. Sort Books by Publication Year**

* Retrieve all books, sorted in descending order by publication year.

**7. Get the Average Publication Year by Genre**

* Calculate the average publication year of books for each genre.

**8. Add a New Field to All Documents**

* Add a new field available (boolean) set to true for all books.

**9. Delete Books Published Before a Certain Year**

* Delete all books published before the year 1950.

**10. List All Unique Genres**

* Retrieve a list of all unique genres in the collection without duplicates.