National University of Computer & Emerging Sciences, Peshawr

FAST School of Computing –Artificial Intelligence Department
Spring 2025, Lab Manual – 11

Course Code: CL-2005	Course: Database Systems Lab
Instructor:	Yasir Arfat

Contents:

Overview of MongoDB

MongoDB is an open-source document database and leading NoSQL database. MongoDBis 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

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

Figure 1. Difference between RDBMS & MongoDB

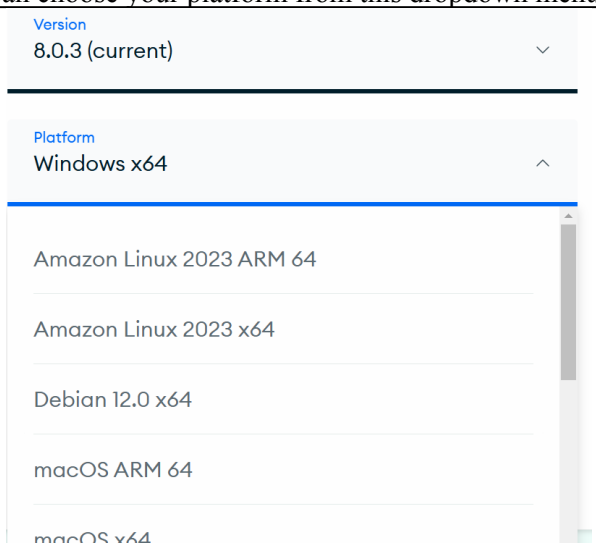
Database: Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

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

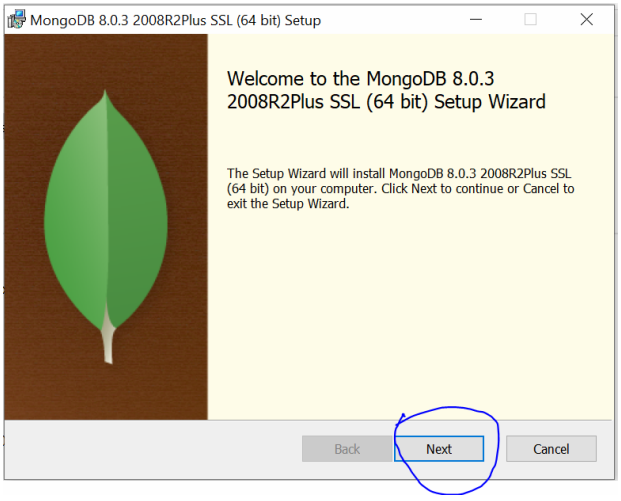
Note for non-window users: You can choose your platform from this dropdown menu while installation:

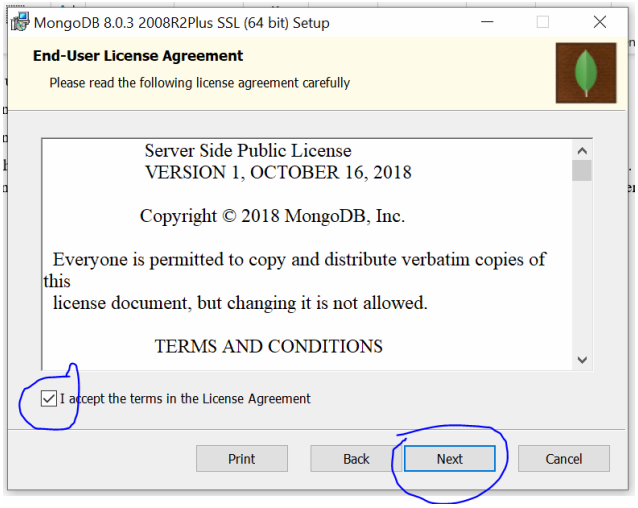


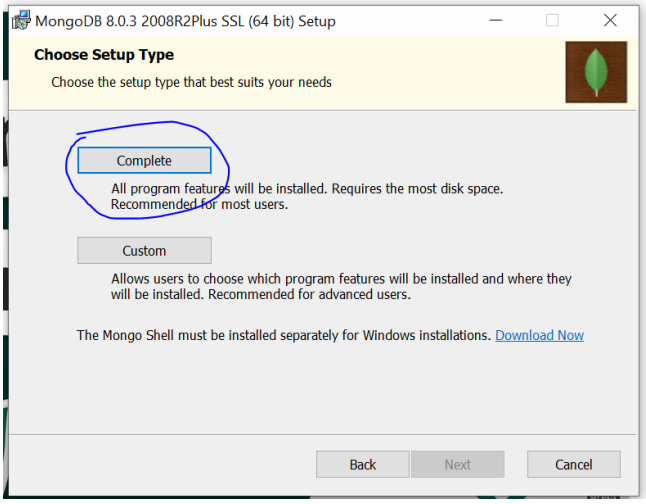
We can use MongoDB in the following ways:

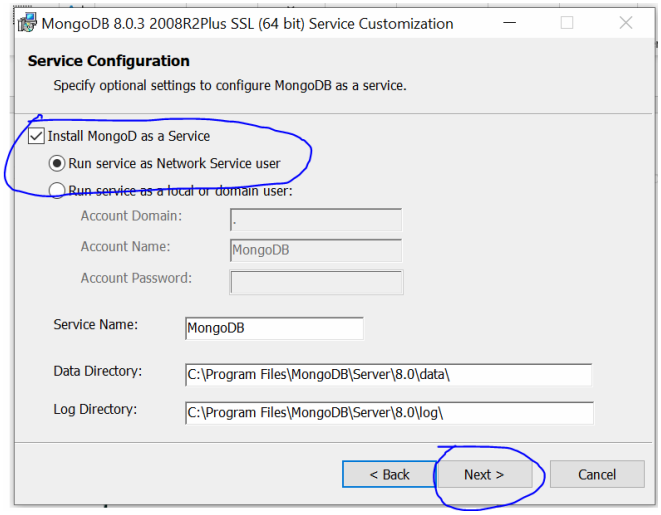
1. MongoDB on CLI
2. MongoDB Compass (we will use this one)

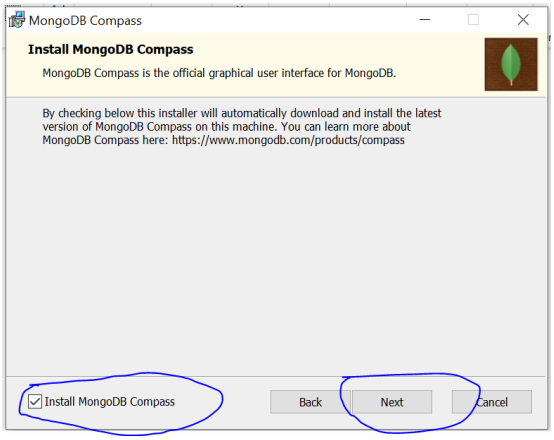
As for the server, we can either use MongoDB Atlas (Cloud based) or host locally (we will do this one). The community server can be installed from here. After running the .msi setup, we will get the following screen:

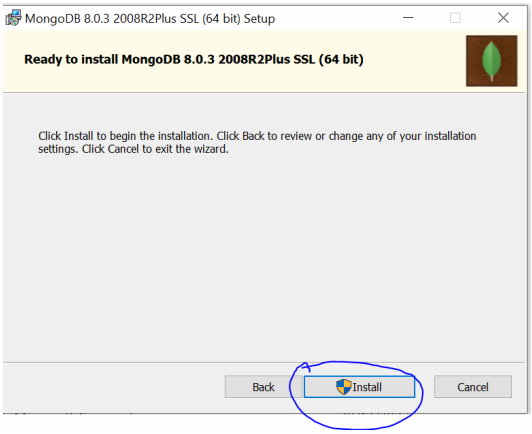



MongoDB 8.0.3 2008R2Plus SSL (64 bit) Setup
End-User License Agreement
Please read the following license agreement carefully
Server Side Public License
VERSION 1, OCTOBER 16, 2018
Copyright © 2018 MongoDB, Inc.
Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.
TERMS AND CONDITIONS
I accept the terms in the License Agreement
Print
Back
Next
Cancel

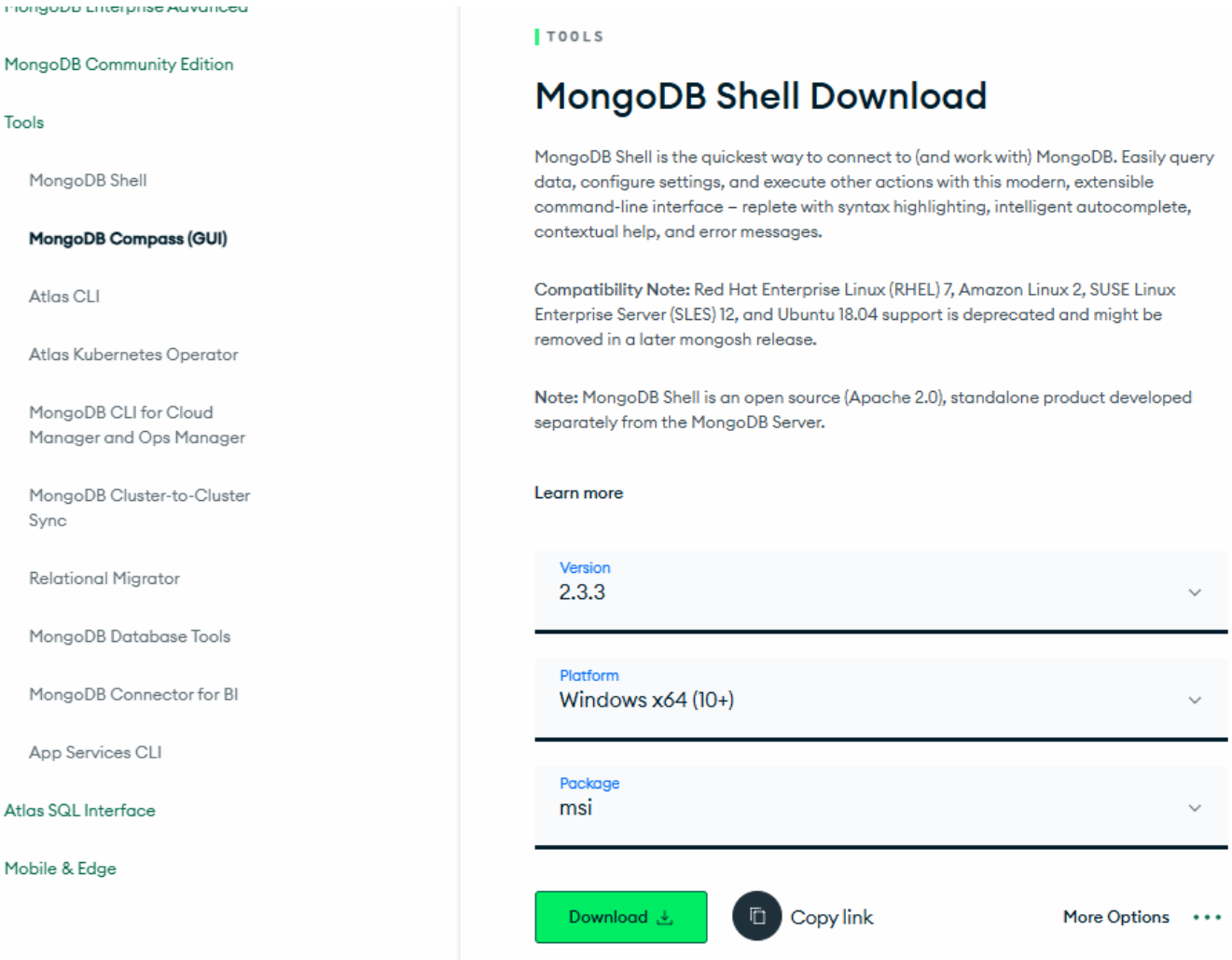

MongoDB 8.0.3 2008R2Plus SSL (64 bit) Setup
Choose Setup Type
Choose the setup type that best suits your needs
Complete
All program features will be installed. Requires the most disk space.
Recommended for most users.
Custom
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.
The Mongo Shell must be installed separately for Windows installations. Download Now
Back
Next
Cancel


MongoDB 8.0.3 2008R2Plus SSL (64 bit) Service Customization
Service Configuration
Specify optional settings to configure MongoDB as a service.
Install MongoD as a Service
Run service as Network Service user
Run service as a local or domain user:
Account Domain:
Account Name:
MongoDB
Account Password:
Service Name:
MongoDB
Data Directory:
C:\Program Files\MongoDB\Server\8.0\data\
Log Directory:
C:\Program Files\MongoDB\Server\8.0\log\
< Back
Next >
Cancel

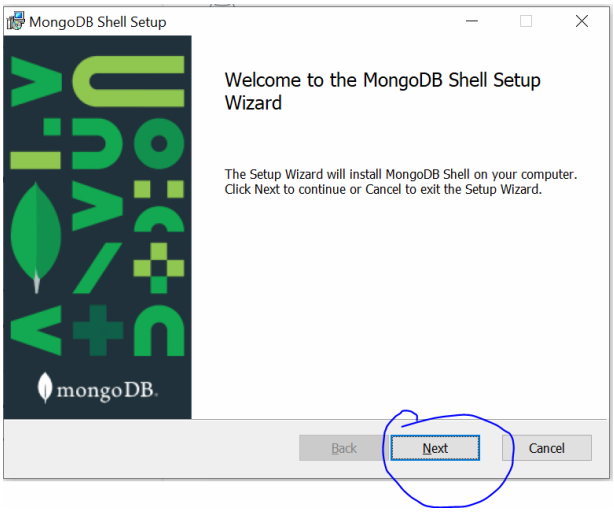

MongoDB Compass
Install MongoDB Compass
MongoDB Compass is the official graphical user interface for MongoDB.
By checking below this installer will automatically download and install the latest version of MongoDB Compass on this machine. You can learn more about MongoDB Compass here: https://www.mongodb.com/products/compass
Install MongoDB Compass
Back
Next
Cancel



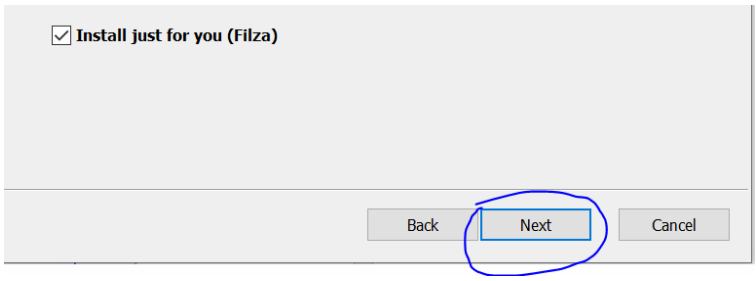
Let the setup finish. This might take a few minutes. If MongoDB Compass does not auto-start after this, start it on your own then open it on your own and minimize it for now. Now we are going to install MongoDB Shell. Under the tools section, download the [MongoDB Shell] (also called mongosh):

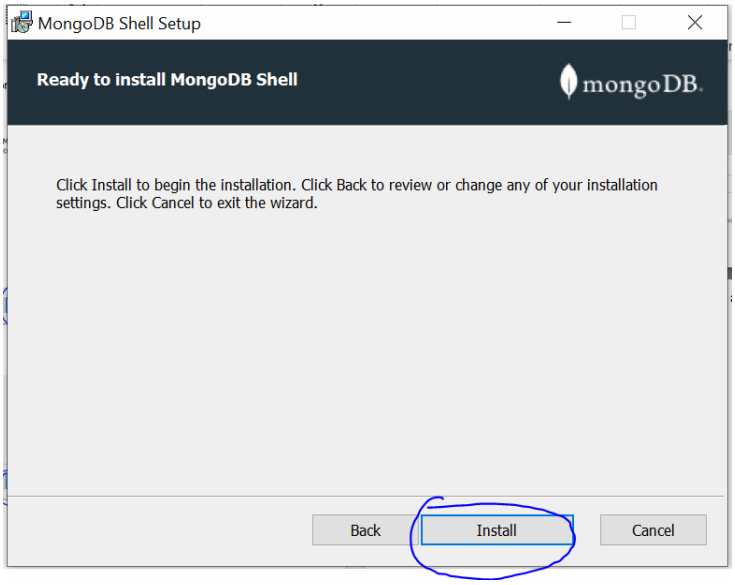


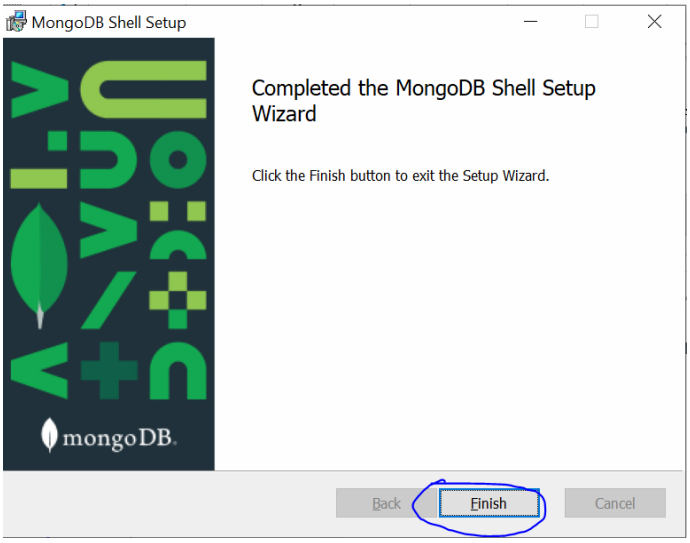
Open the .msi setup and follow the steps below:



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



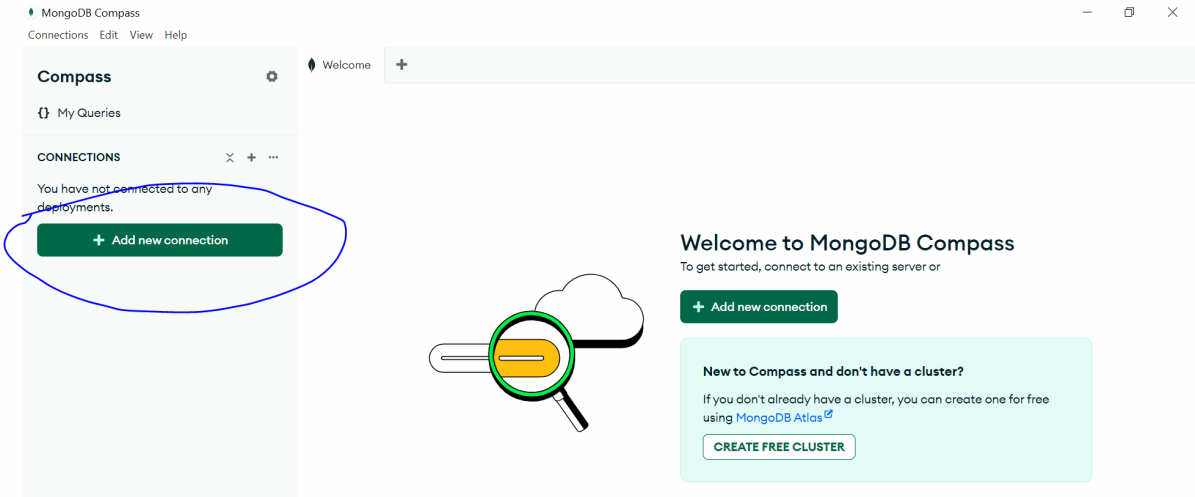




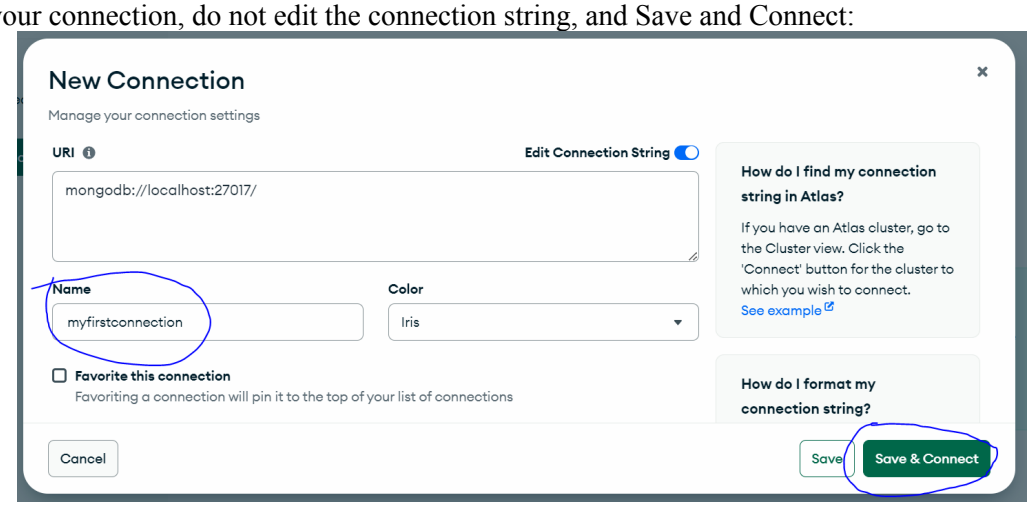
Now that you have installed Mongosh, open CMD and type mongosh and you should see the following:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
(c) Microsoft Corporation. All rights reserved.

C:\Users\PMLS>mongosh
Current Mongosh Log ID: 674316b7398a01adf00d818f
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=
mongosh+2.3.3
Using MongoDB:          8.0.3
Using Mongosh:          2.3.3

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.m
ongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2024-11-24T16:44:22.246+05:00: Access control is not enabled for the database. Read and write access to data
 and configuration is unrestricted
------

test>
```

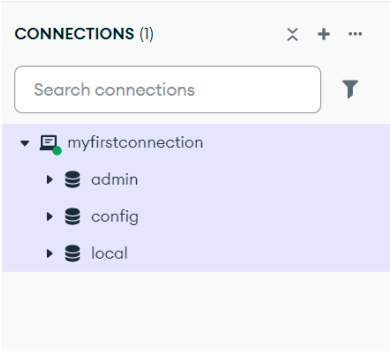
We will use Mongosh and MongoDB Compass together to visualize what we are doing in this lab. Keep the shell running and go back to Compass:



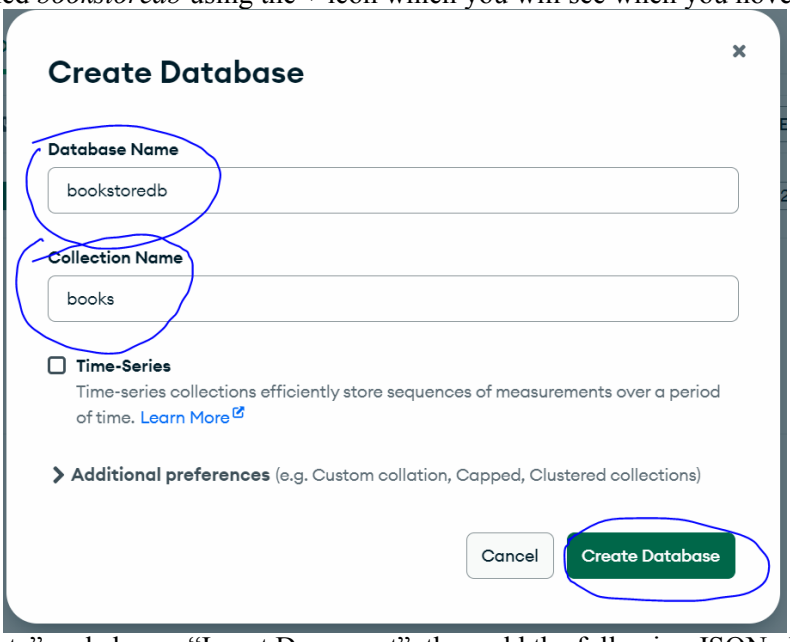
Enter a name for your connection, do not edit the connection string, and Save and Connect:



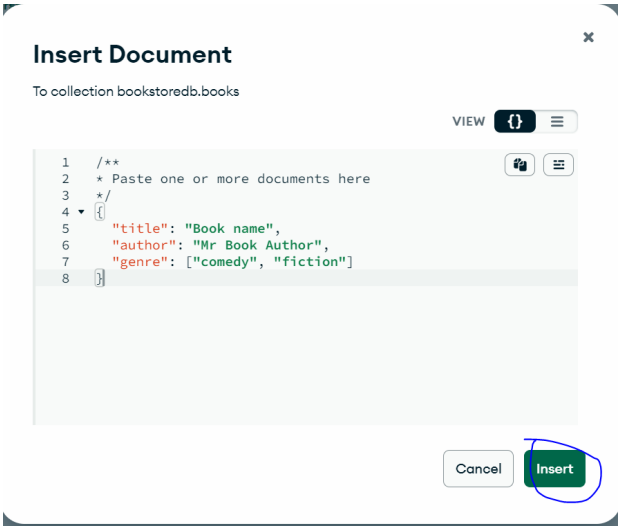
You should now be able to see the connection in the left pane of MongoDB Compass:



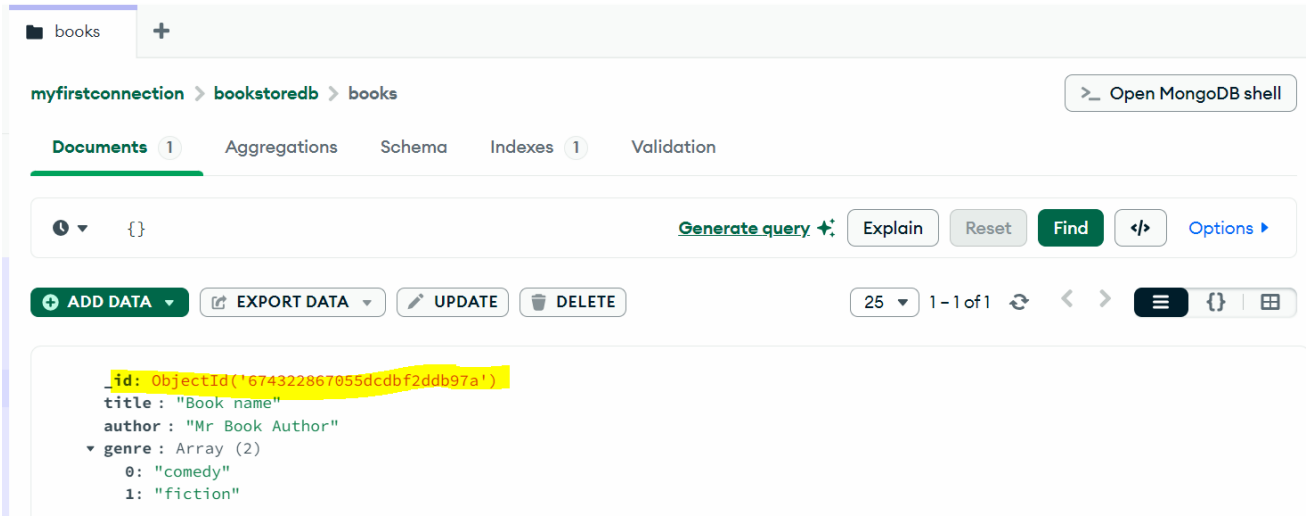
Creating a new database named *bookstoredb* using the + icon which you will see when you hover over *myfirstcinnection*.



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:

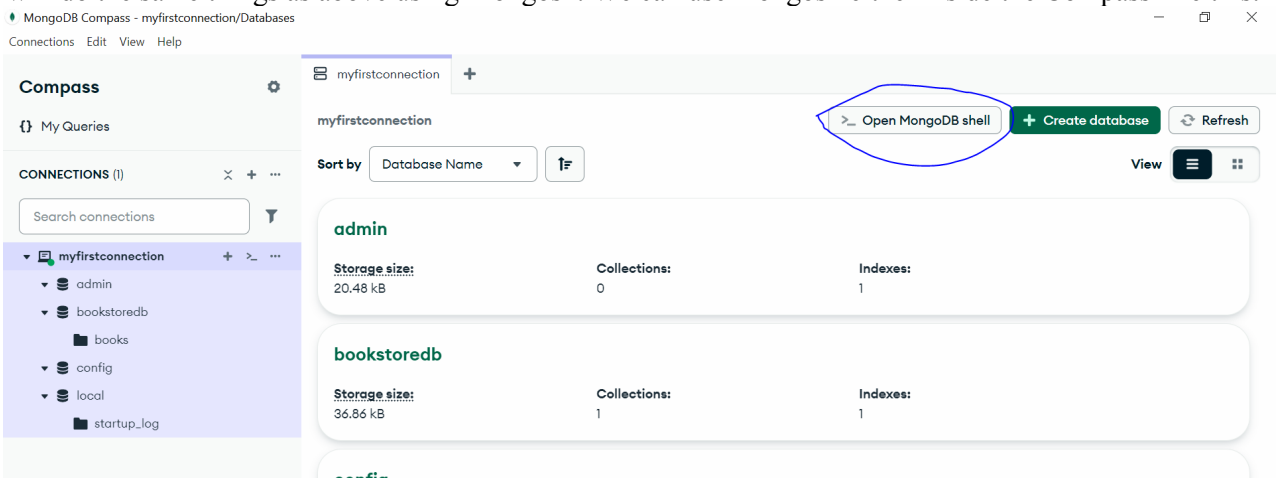


Insert the following documents in the table collection:

```
[
 {
  "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:



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

1. For seeing all the available databases:

```
test> show dbs
admin          40.00 KiB
bookstoredb    72.00 KiB
config        108.00 KiB
local          40.00 KiB
test>
```

2. For selecting a database to work with:

```
test> use bookstoredb
bookstoredb>
```

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

3. You can go back to test> by simply writing use test.
4. 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.

```
anydatabasenamehere>
switched to db anydatabasenamehere
```

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

```
mydb> help

  Shell Help:

    use                                     Set current database
    show                                    'show databases'/'show dbs': Print a list of all available databases.
                                            'show collections'/'show tables': Print a list of all collections for current database.
                                            'show profile': Prints system.profile information.
                                            'show users': Print a list of all users for current database.
                                            'show roles': Print a list of all roles for current database.
                                            'show log <type>': log for current connection, if type is not set uses 'global'
                                            'show logs': Print all logs.

    exit                                    Quit the MongoDB shell with exit/exit()/.exit
    quit                                    Quit the MongoDB shell with quit/quit()
    Mongo                                   Create a new connection and return the Mongo object. Usage: new Mongo(URI, options [optional])
    connect                                 Create a new connection and return the Database object. Usage: connect(URI, username [optional], password [optional])
    it                                      result of the last line evaluated; use to further iterate
    version                                 Shell version
    load                                    Loads and runs a JavaScript file into the current shell environment
mydb>
 CLI
    disableTelemetry                        Disables collection of anonymous usage data to improve the mongosh CLI
    passwordPrompt                          Prompts the user for a password
    sleep                                   Sleep for the specified number of milliseconds
    print                                   Prints the contents of an object to the output
mydb>
    convertShardKeyToHashed                 Returns the hashed value for the input using the same hashing function as a hashed index.
    cls                                     Clears the screen like console.clear()
```

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

```
mydb> db.help()

  Database Class:

    getMongo                                Returns the current database connection
    getName                                 Returns the name of the DB
    getCollectionNames                      Returns an array containing the names of all collections in the current database.
    getCollectionInfos                      Returns an array of documents with collection information, i.e. collection name and options, for the current database.
    runCommand                              Runs an arbitrary command on the database.
    adminCommand                            Runs an arbitrary command against the admin database.
    aggregate                               Runs a specified admin/diagnostic pipeline which does not require an underlying collection.
    getSiblingDB                            Returns another database without modifying the db variable in the shell environment.
    getCollection                           Returns a collection or a view object that is functionally equivalent to using the db.<collectionName>.
    dropDatabase                            Removes the current database, deleting the associated data files.
    createUser                              Creates a new user for the database on which the method is run. db.createUser() returns a duplicate user error if the user a
lready exists on the database.
    updateUser                              Updates the user's profile on the database on which you run the method. An update to a field completely replaces the previou
s field's values. This includes updates to the user's roles array.
    changeUserPassword                      Updates a user's password. Run the method in the database where the user is defined, i.e. the database you created the user.
    logout                                  Ends the current authentication session. This function has no effect if the current session is not authenticated.
    dropUser                                Removes the user from the current database.
    dropAllUsers                            Removes all users from the current database.
    auth                                    Allows a user to authenticate to the database from within the shell.
    grantRolesToUser                        Grants additional roles to a user.
    revokeRolesFromUser                     Removes a one or more roles from a user on the current database.
    getUser                                 Returns user information for a specified user. Run this method on the user's database. The user must exist on the database o
n which the method runs.
    getUsers                                Returns information for all the users in the database.
    createCollection                        Create new collection
    createEncryptedCollection               Creates a new collection with a list of encrypted fields each with unique and auto-created data encryption keys (DEKs). This
 is a utility function that internally utilises ClientEnryption.createEncryptedCollection.
    createView                              Create new view
    createRole                              Creates a new role.
    updateRole                              Updates the role's profile on the database on which you run the method. An update to a field completely replaces the previou
s field's values.
    dropRole                                Removes the role from the current database.
    dropAllRoles                            Removes all roles from the current database.
    grantRolesToRole                        Grants additional roles to a role.
    revokeRolesFromRole                     Removes a one or more roles from a role on the current database.
```

Some Considerations while designing Schema in MongoDB

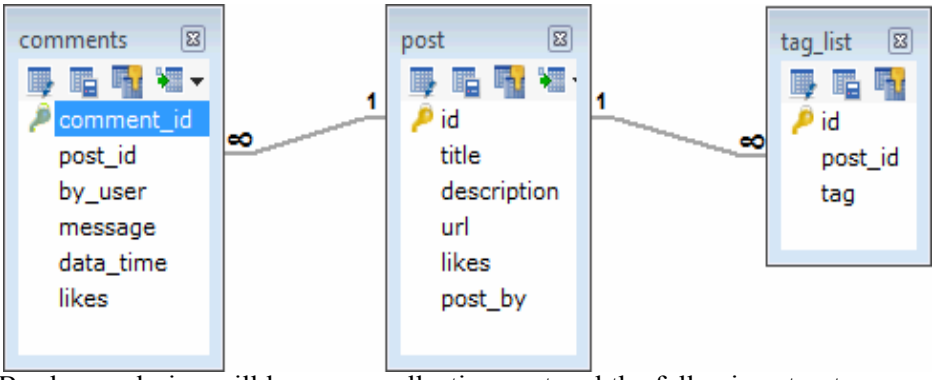
1. Design your schema according to user requirements.
2. Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should be no need of joins).
3. Duplicate the data (but limited) because disk space is cheap as compared to compute time.
4. Use joins 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:



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

```
{
   _id: POST_ID
   title: TITLE_OF_POST,
   description: POST_DESCRIPTION,
   by: POST_BY,
   url: URL_OF_POST,
   tags: [TAG1, TAG2, TAG3],
   likes: TOTAL_LIKES,
   comments: [
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      },
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      }
   ]
}
```

So, while showing the data, in RDBMS you need to join three tables and in MongoDB, datawill 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*.

```
test> use inventorydb
switched to db inventorydb
inventorydb> db
inventorydb
inventorydb> db.createCollection("furniture")
{ ok: 1 }
inventorydb>
```

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

```
inventorydb> show dbs
admin        40.00 KiB
bookstoredb  72.00 KiB
config       72.00 KiB
inventorydb   8.00 KiB
local        40.00 KiB
inventorydb>
```

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

```
inventorydb> db.furniture.insertOne({name:"Table",colour:"Brown",dimensions:[12,18]})
{
  acknowledged: true,
  insertedId: ObjectId('67433efde017de16630d8191')
}
```

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 [{},{},…,{}]:

```
inventorydb> db.furniture.insertMany([{name:"Chair",colour:"Brown",dimensions:[12,18]},{name:"Bed",colour: "Black", dimensions: [32,90]}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67433f7ce017de16630d8192'),
    '1': ObjectId('67433f7ce017de16630d8193')
  }
}
```

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

```
inventorydb> db.furniture.find()
[
  {
    _id: ObjectId('67433ec5e017de16630d8190'),
    name: 'Table',
    colour: 'Brown',
    dimensions: [ 12, 18 ]
  },
  {
    _id: ObjectId('67433efde017de16630d8191'),
    name: 'Table',
    colour: 'Brown',
    dimensions: [ 12, 18 ]
  },
  {
    _id: ObjectId('67433f7ce017de16630d8192'),
    name: 'Chair',
    colour: 'Brown',
    dimensions: [ 12, 18 ]
  },
  {
    _id: ObjectId('67433f7ce017de16630d8193'),
    name: 'Bed',
    colour: 'Black',
    dimensions: [ 32, 90 ]
  }
]
```

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:

```
inventorydb> db.furniture.find({name:"Bed"})
[
  {
    _id: ObjectId('67433f7ce017de16630d8193'),
    name: 'Bed',
    colour: 'Black',
    dimensions: [ 32, 90 ]
  }
]
```

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

```
inventorydb> db.furniture.find({dimensions:{$gt:30}})
[
  {
    _id: ObjectId('67433f7ce017de16630d8193'),
    name: 'Bed',
    colour: 'Black',
    dimensions: [ 32, 90 ]
  }
]
inventorydb> db.furniture.find({"dimensions.0":{$gt:30}})
[
  {
    _id: ObjectId('67433f7ce017de16630d8193'),
    name: 'Bed',
    colour: 'Black',
    dimensions: [ 32, 90 ]
  }
]
```

3. Using AND:

```
inventorydb> db.furniture.find({$and: [ {"dimensions.0": {$gt: 30} }, {colour: "Black" } ]})
[
  {
    _id: ObjectId('67433f7ce017de16630d8193'),
    name: 'Bed',
    colour: 'Black',
    dimensions: [ 32, 90 ]
  }
]
```

```
inventorydb> db.furniture.find({$and: [ {"dimensions.0": {$gt:30} }, {"dimensions.1": {$lte:90} } ]})
[
  {
    _id: ObjectId('67433f7ce017de16630d8193'),
    name: 'Bed',
    colour: 'Black',
    dimensions: [ 32, 90 ]
  }
]
```

4. Using OR:

```
inventorydb> db.furniture.find({$or: [ {"dimensions.0": {$gt:30} }, {"dimensions.1": {$lte:0} } ]})
[
  {
    _id: ObjectId('67433f7ce017de16630d8193'),
    name: 'Bed',
    colour: 'Black',
    dimensions: [ 32, 90 ]
  }
]
```

5. Not equals to:

```
inventorydb> db.furniture.find( {colour: { $ne: "Brown" } } )
[
  {
    _id: ObjectId('67433f7ce017de16630d8193'),
    name: 'Bed',
    colour: 'Black',
    dimensions: [ 32, 90 ]
  }
]
```

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

```
inventorydb> db.furniture.find({$and: [ {colour: "Brown"}, {$or: [ {name: "Table"}, {name: "Chair"} ]} ] })
[
  {
    _id: ObjectId('67433ec5e017de16630d8190'),
    name: 'Table',
    colour: 'Brown',
    dimensions: [ 12, 18 ]
  },
  {
    _id: ObjectId('67433efde017de16630d8191'),
    name: 'Table',
    colour: 'Brown',
    dimensions: [ 12, 18 ]
  },
  {
    _id: ObjectId('67433f7ce017de16630d8192'),
    name: 'Chair',
    colour: 'Brown',
    dimensions: [ 12, 18 ]
  }
]
```

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]:

```
inventorydb> db.furniture.updateOne({dimensions: [32, 90]}, {$set: {colour: "Ivory"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

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

```
inventorydb> db.furniture.updateMany({colour: "Brown"}, {$set: {colour: "Dark Brown"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
```

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:

```
inventorydb> db.furniture.deleteOne({name: "Chair"})
{ acknowledged: true, deletedCount: 1 }
inventorydb>
```

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

```
inventorydb> db.furniture.deleteMany({dimensions: [12, 18]})
{ acknowledged: true, deletedCount: 2 }
```

Dropping a collection:

Syntax: db.*collection_name*.drop()

```
inventorydb> db.furniture.drop()
true
```

Let's view the collections now:

```
inventorydb> show collections

inventorydb>
```

Returns nothing, which means collection is dropped.

Dropping a database using db.dropDatabase():

```
inventorydb> use inventorydb
already on db inventorydb
inventorydb> db.dropDatabase()
{ ok: 1, dropped: 'inventorydb' }
inventorydb>
```

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:

```
{ "_id": 1, "name": "Alice", "age": 20, "scores": { "math": 85, "science": 90 } }
{ "_id": 2, "name": "Bob", "age": 22, "scores": { "math": 78, "science": 82 } }
{ "_id": 3, "name": "Charlie", "age": 21, "scores": { "math": 92, "science": 88 } }
{ "_id": 4, "name": "Daisy", "age": 23, "scores": { "math": 68, "science": 74 } }
```

4. Insert the following documents into the Courses collection:

```
{ "_id": 101, "courseName": "Mathematics", "instructor": "Dr. Smith", "studentsEnrolled": [1, 2, 3] }
{ "_id": 102, "courseName": "Science", "instructor": "Dr. Adams", "studentsEnrolled": [2, 3, 4] }
```

5. 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".
6. 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).
7. Use updateOne to:
   - Increase the science score of the student where name is "Bob" and math score is >= 75.
8. Use updateMany to:
   - Increase the math score by 5 for students whose science score is < 80 **and** age > 22.
9. Use deleteOne to:
   - Remove a student where name is "Daisy" **and** their science score is < 80.
10. Use deleteMany to:
    - Remove courses where the studentsEnrolled array includes 2 **or** the instructor is "Dr. Smith".
11. Drop the Students collection.
12. Drop the Courses collection.
13. 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

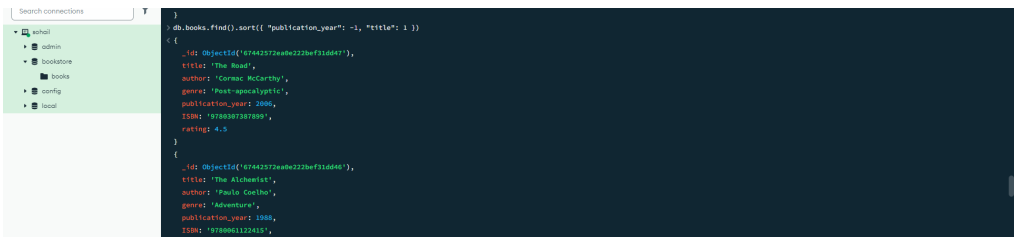
Sort query results by one or more fields.

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

```
db.books.find().sort({ "publication_year": 1 })
```

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

```
db.books.find().sort({ "publication_year": -1, "title": 1 })
```



3. Limiting and Skipping Results

Control the number of documents returned.

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

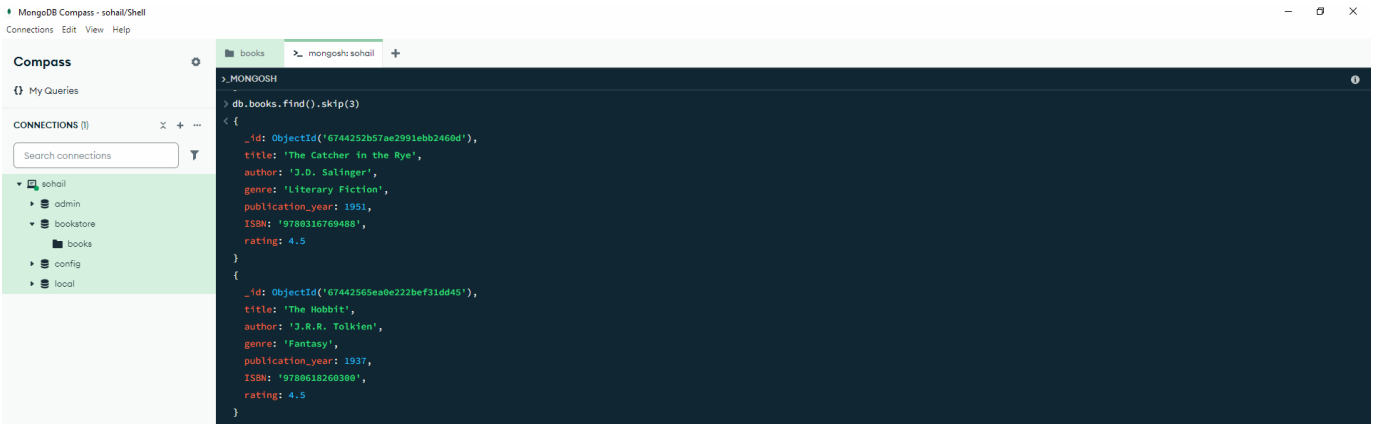
  ```
  db.books.find().limit(5)
  ```

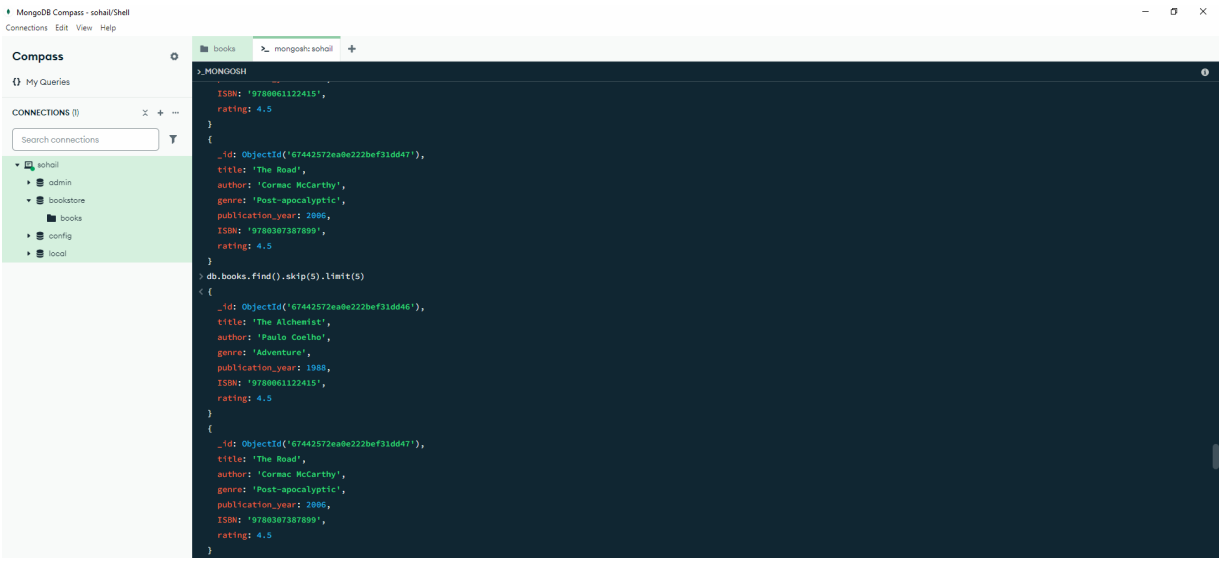
- **Skip a Number of Results** (e.g., skip the first 3 books and return the next ones):

  ```
  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)
  ```





4. Aggregation Pipeline

Perform complex data transformations and analysis using aggregation.

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

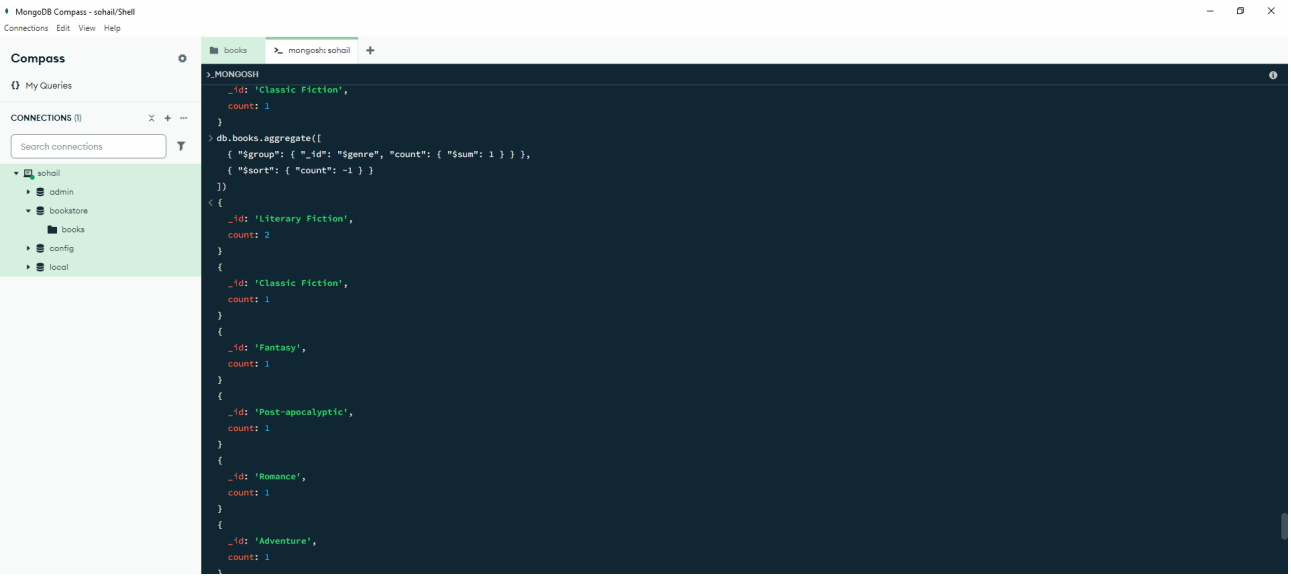
```
db.books.aggregate([
  { "$group": { "_id": null, "avgPublicationYear": { "$avg":
"$publication_year" } } }
])
```

- **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 } }
])
```



5. Projection

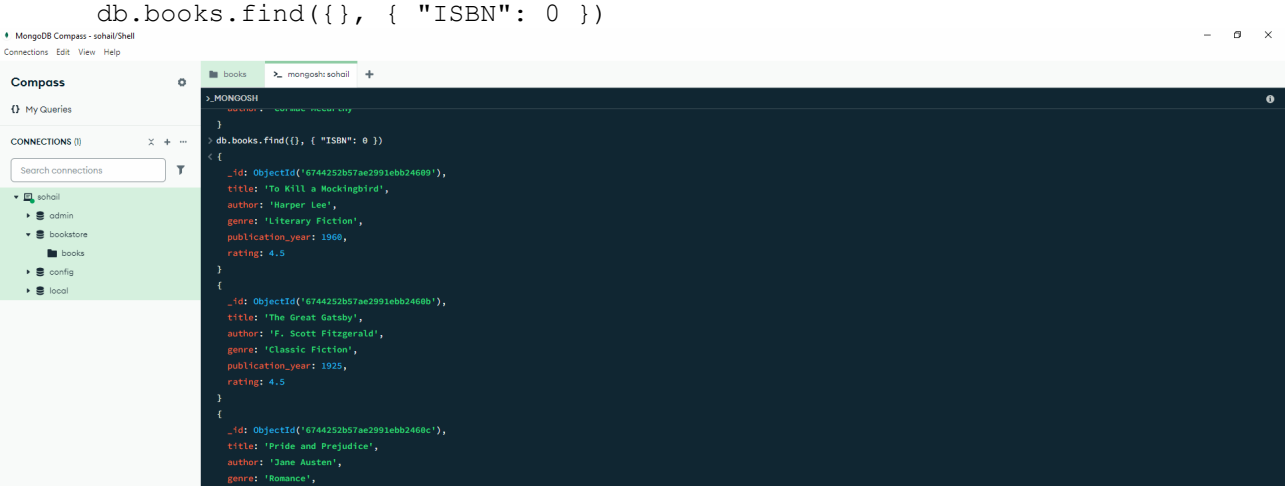
Control which fields are returned in query results.

- **Return Only Title and Author**:

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

- **Exclude ISBN Field**:

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



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" } })
```

7. Find Documents Using Regular Expressions

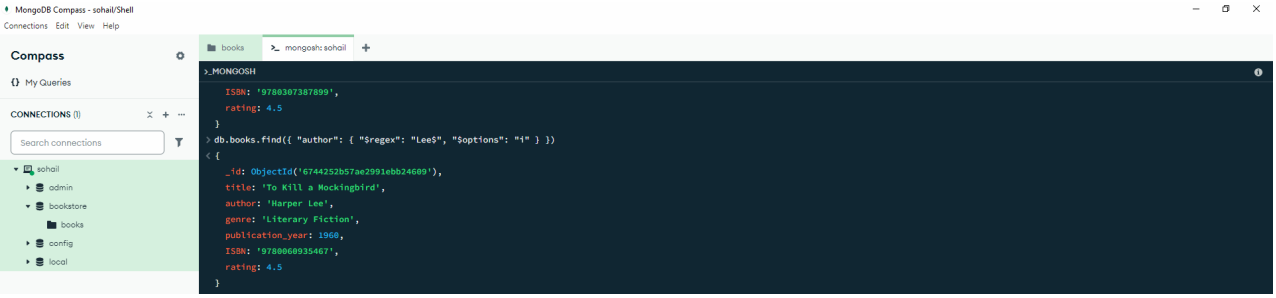
Search for documents with partial text matches.

- **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" } })
```



8. Update with Increment/Decrement

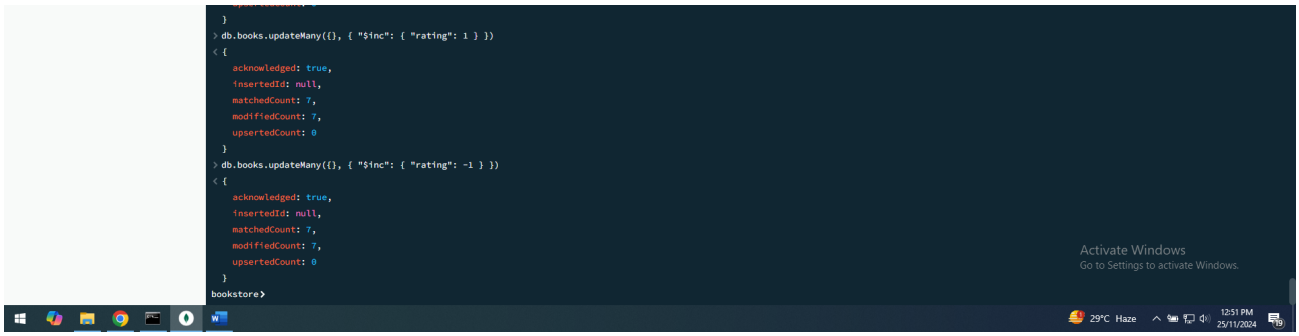
Increase or decrease numeric values directly.

- **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 } })
```



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.