13th KM Stone, Bannur Road, Mysore - 560 028

Department of CSE–Artificial Intelligence & Machine Learning

(Academic Year 2025-26)

LABORATORY MANUAL

SUBJECT: MANGO DB LABORATORY

SUB CODE: BDSL456B

SEMESTER: IV

SCHEME: 2022

Prepared By	Verified By	Approved by
Ms. GEETHA.B Instructor	Ms. LIKITHA D Assistant Professor Dept. of CSE-AI &ML	Dr. ANIL KUMAR C.J Assoc. Professor & Head Dept. of CS-AI &ML

Institute Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Institute Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

Department Vision

To impart technical education in the field of Artificial intelligence and machine learning of topnotch quality with a high level of professional competence, social obligation, and global cognizance among the students.

Department Mission

- To impart technical education that is up to date, relevant and makes students to compete at global level
- Fostering an ambiance where students can adopt the suitable moral, intellectual, emotional, and physical attributes to shine as the leaders of tomorrow's society.

- To strive to meet ever higher educational standard.

Program Outcomes (PO’s)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

Program Educational Objectives (PEO's):

PEO1: Graduates will be able to hone their problem-solving abilities and capacity to offer solutions to challenges that arise in the actual world.

PEO2: Able to design and develop AI based solutions to real-world problems in a business, research, or social environment.

PEO3: Graduates shall acquire and inculcate corporate culture, core attributes, and leadership qualities as well as professional etiquette's and lifelong learning.

Program Specific Outcomes (PSO's)

PSO1: Ability to design and develop artificial intelligent based solutions by applying optimal algorithms to solve real world issues.

PSO2: Ability to apply suitable AI tools and techniques to offer solutions in the various domains of engineering.

MongoDB		Semester	4
Course Code	**BDSL456B**	CIE Marks	50
Teaching Hours/Week (L: T:P: S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	24	Total Marks	100
Credits	01		

Sl.NO	**Experiments**
1	Illustration of Where Clause, AND,OR operations in MongoDB. a. Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection. (Note: use any collection).
2	a. Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection. **b**.Develop a MongoDB query to display the first 5 documents from the results obtained in a. [use of limit and find].
3	a. Execute query selectors (comparison selectors, logical selectors) and list out the results on any collection b. Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection.
4	Create and demonstrate how projection operators ($, $elematch and $slice) would be used in the MondoDB.
5	Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators).
6	Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project, $skip etc. students encourage to execute several queries to demonstrate various aggregation operators)
7	a. Find all listings with listing_url, name, address, host_picture_url in the listings And Reviews collection that have a host with a picture url **b**.Using E-commerce collection write a query to display reviews summary.
8	a. Demonstrate creation of different types of indexes on collection (unique, sparse, compound and multikey indexes) using indexes. b. Demonstrate optimization of queries
9	a. Develop a query to demonstrate Text search using catalog data collection for a given word b. Develop queries to illustrate excluding documents with certain words

	and phrases
10	Develop an aggregation pipeline to illustrate Text search on Catalog data collection.

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

1. Make use of MangoDB commands and queries.
2. Illustrate the role of aggregate pipelines to extract data.
3. Demonstrate optimization of queries by creating indexes.
4. Develop aggregate pipelines for text search in collections.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to **20 marks** (40% of the maximum marks).

The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.

INTRODUCTION

What is NoSQL?

NoSQL alias Non SQL alias Non-Relational.

In simple terms: A NoSQL database is a Key-Value Database
A NoSQL database provides

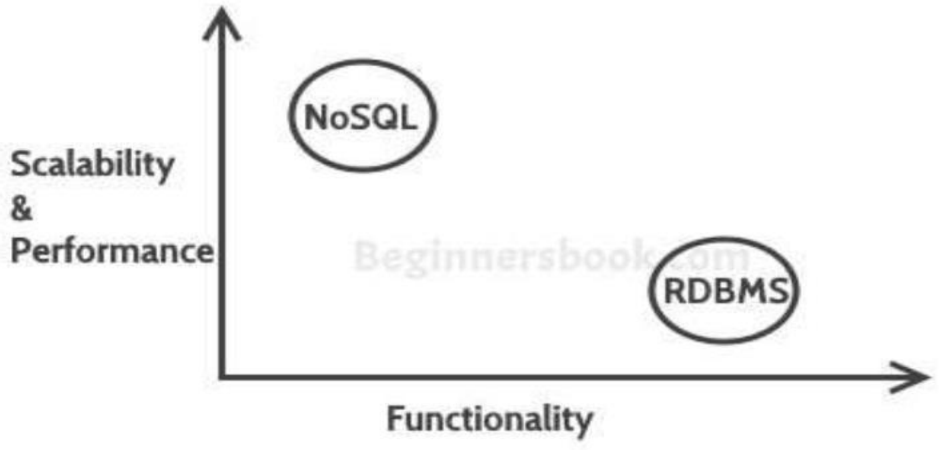
- A mechanism for storage and retrieval of data.
- The data is NOT modelled in relational and tabular format.

A large variation and flavours are available in the market-MongoDB, AmazonDocument DB, Google datastore,
Amazon DynamoDB etc.

Extremely useful, powerful, high performance database in large big data applications, large distributed network architecture apps etc.

Advantages of NoSQL

- One of the advantage of NoSQL database is that they are really easy to scale and they are much
- faster in most types of operations that we perform on database.
- When you are dealing with huge amount of data then NoSQL database is your best choice.



When to go for NoSQL

When you would want to choose NoSQL over relational database:
1. When you want to store and retrieve huge amount of data.
2. The relationship between the data you store is not that important
3. The data is not structured and changing over time
4. Constraints and Joins support is not required at database level
5. The data is growing continuously and you need to scale the database regular to handle the data.

What is MongoDB?

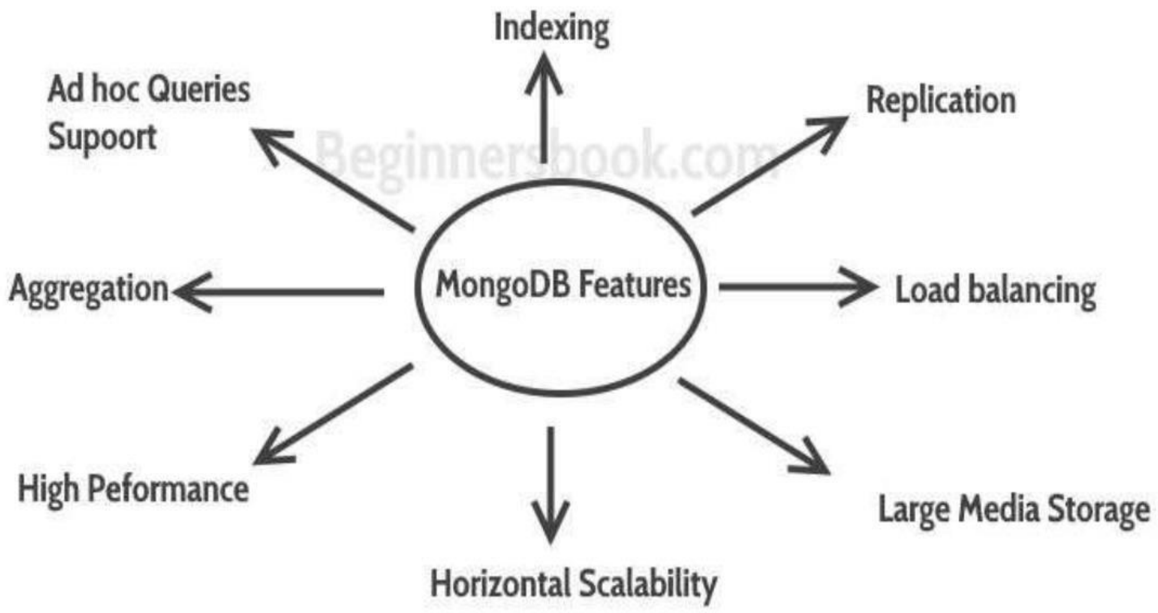
- MongoDB is a opensource,crossplatform,document-oriented database program.
- Classified as a NoSQL Database program,MongoDB uses JSON-Like documents.

• MongoDB is a learning NoSQL database.

Mongo DB – Document Oriented NoSQL Database Approach

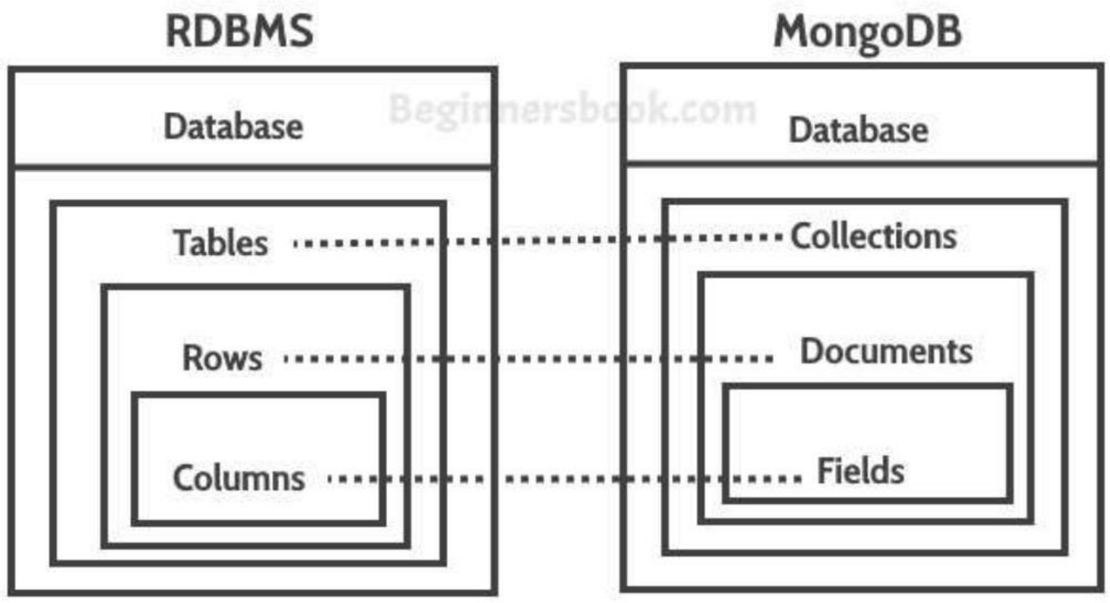
In mongo DB, Querying on this model is easy, since the schema is de-normalized. No joins are required. So we plan to use the Mongo DB based solution

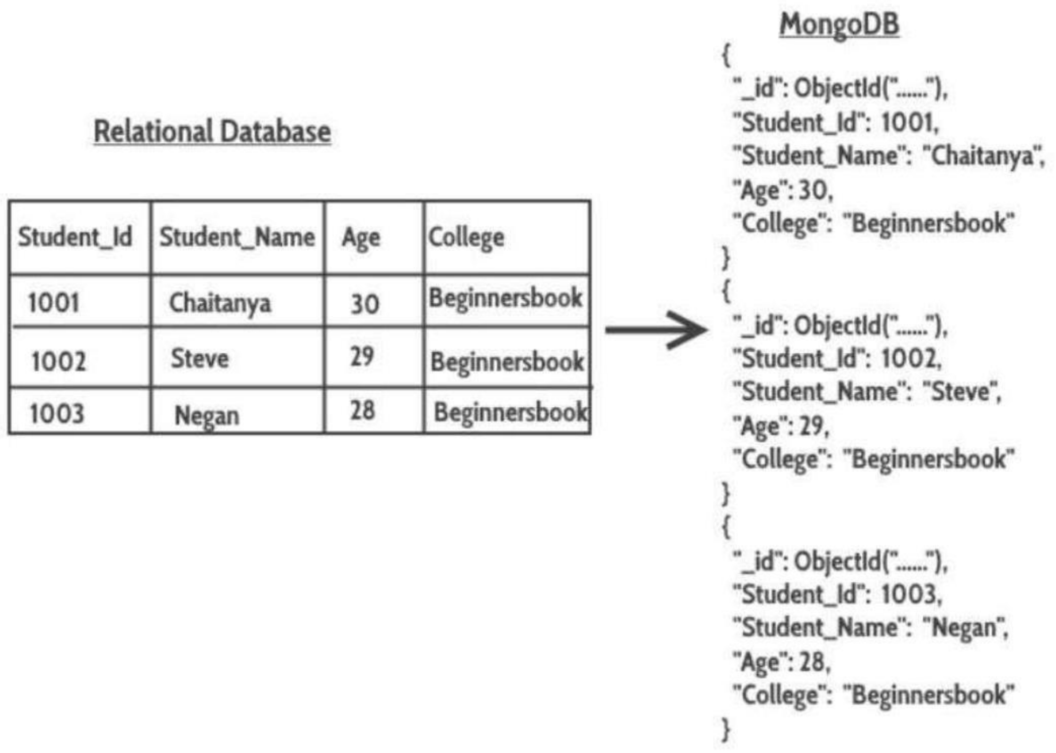
What is MongoDB ?



What is MongoDB ?

Mapping relational database to MongoDB





Relational Database

Student_Id	Student_Name	Age	College
1001	Chaitanya	30	Beginnersbook
1002	Steve	29	Beginnersbook
1003	Negan	28	Beginnersbook

MongoDB

```
{
 "_id": ObjectId("......"),
 "Student_Id": 1001,
 "Student_Name": "Chaitanya",
 "Age": 30,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("......"),
 "Student_Id": 1002,
 "Student_Name": "Steve",
 "Age": 29,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("......"),
 "Student_Id": 1003,
 "Student_Name": "Negan",
 "Age": 28,
 "College": "Beginnersbook"
}
```

Mongo DB Introduction

- What is database?
- What is document?

- What is Collection?
  - **What is database?**

    Database is collection of data.

    In MongoDB context
- Database can also be described as physical container of collections.
- A database can have any number of collections.
- Each database gets its own set of files on the file system.
- A MongoDB server can hosts multiple database inside it.

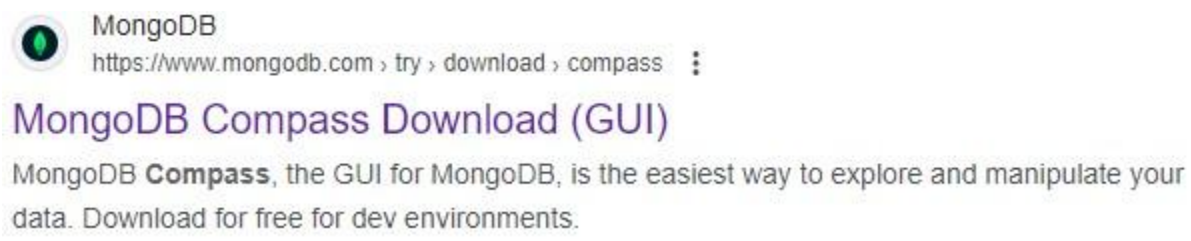
  - **What is collection?**

    Collection is a group of MongoDB documents.
- You can relate collection as a "Table"
- . Not literally but hypothetically
- Unlike tables, Collections does not have any schema definition.
- Unlike RDBMS database – the collections DO NOT have any concept of JOINS.
- However, we can achieve joins functionality using Aggregation in MongoDB.

  - **What is document?**
- Document in MongoDB is a set of key-value pairs.
- Every document in MongoDB has a unique value via Key "_id".
- Documents have flexible and dynamic schema.
- Document schema is user defined and is not Fixed or Static.
- Documents can hold any data as long as they are valid data types in MongoDB.
- Documents within a collection can have different schemas or fields.

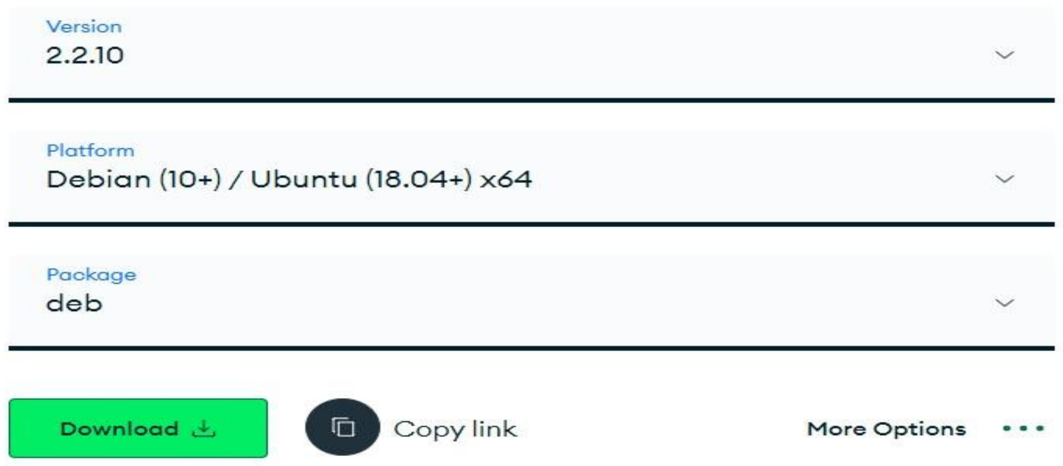
- Documents within a collection are related data belonging to a particular subject**.**

How To Install MongoDB Compass In Ubuntu


MongoDB
https://www.mongodb.com › try › download › compass
MongoDB Compass Download (GUI)
MongoDB **Compass**, the GUI for MongoDB, is the easiest way to explore and manipulate your data. Download for free for dev environments.

MongoDB Shell Download

MongoDB Shell is the quickest way to connect to (and work with) MongoDB. Easily query data, configure settings, and execute other actions with this modern, extensible command-line interface — replete with syntax highlighting, intelligent autocomplete, contextual help, and error messages.


Version
2.2.10

Platform
Debian (10+) / Ubuntu (18.04+) x64

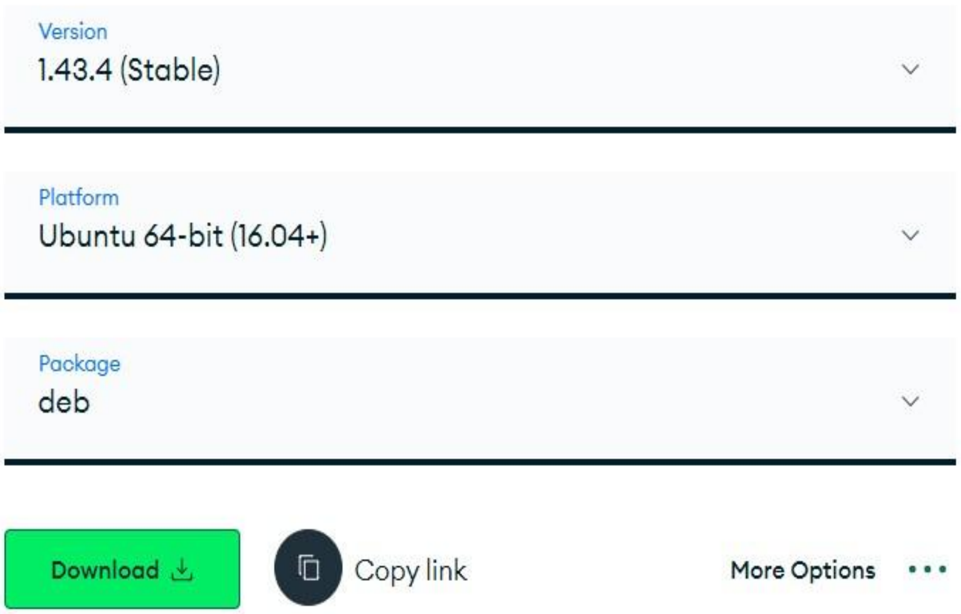
Package
deb

MongoDB Compass Download (GUI)

Easily explore and manipulate your database with Compass, the GUI for MongoDB. Intuitive and flexible, Compass provides detailed schema visualizations, real-time performance metrics, sophisticated querying abilities, and much more

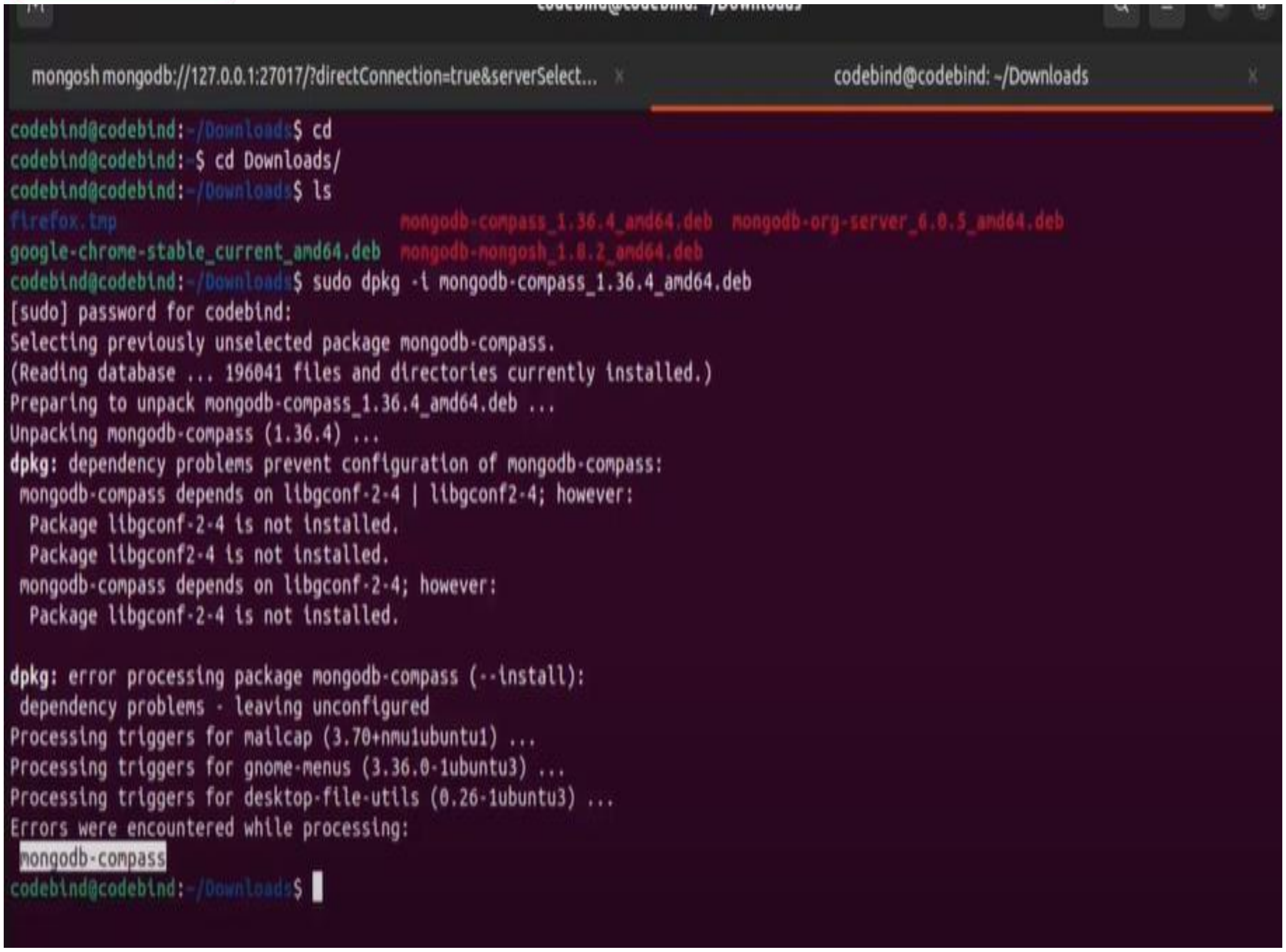
Compass

The full version of MongoDB Compass, with all features and capabilities


Version
1.43.4 (Stable)

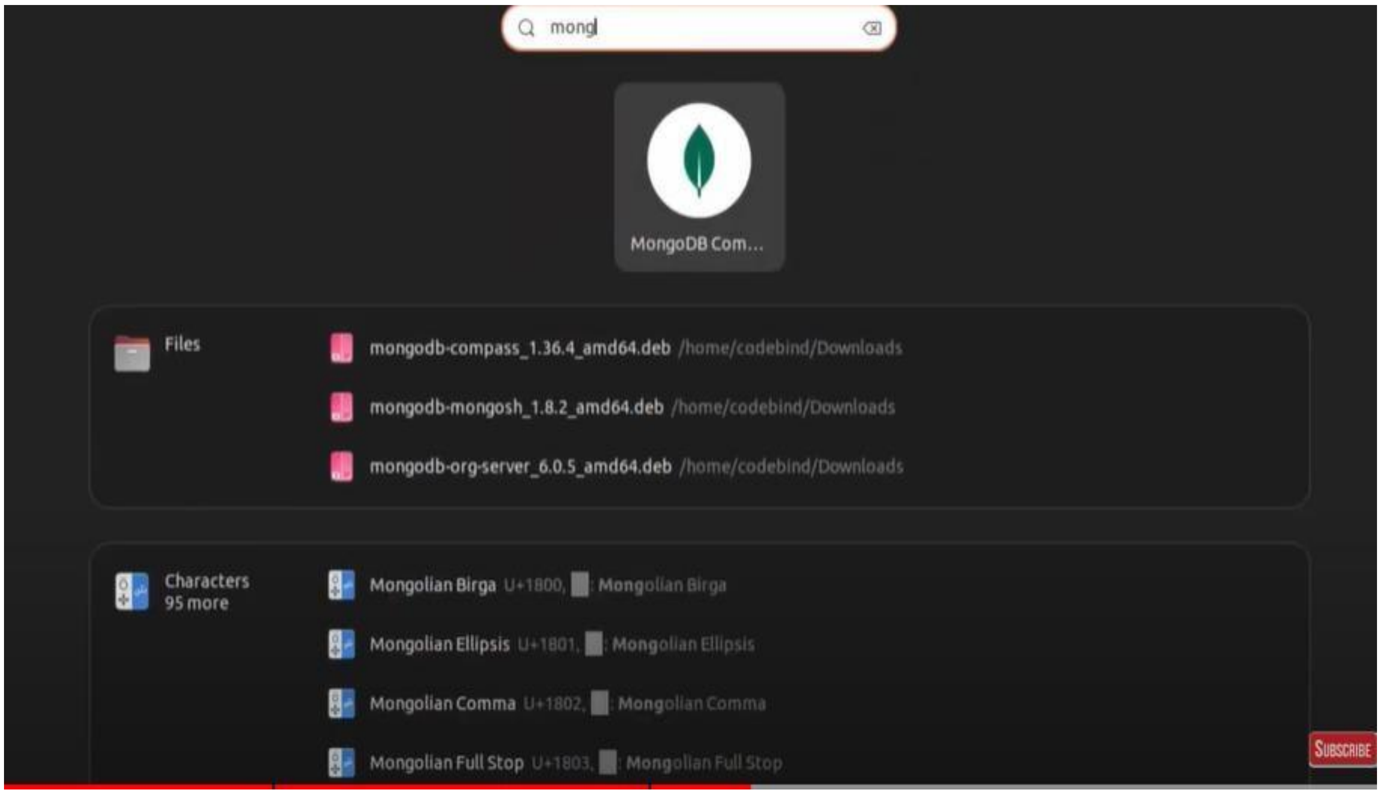
Platform
Ubuntu 64-bit (16.04+)

Package
deb


mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelect...
codebind@codebind: ~/Downloads

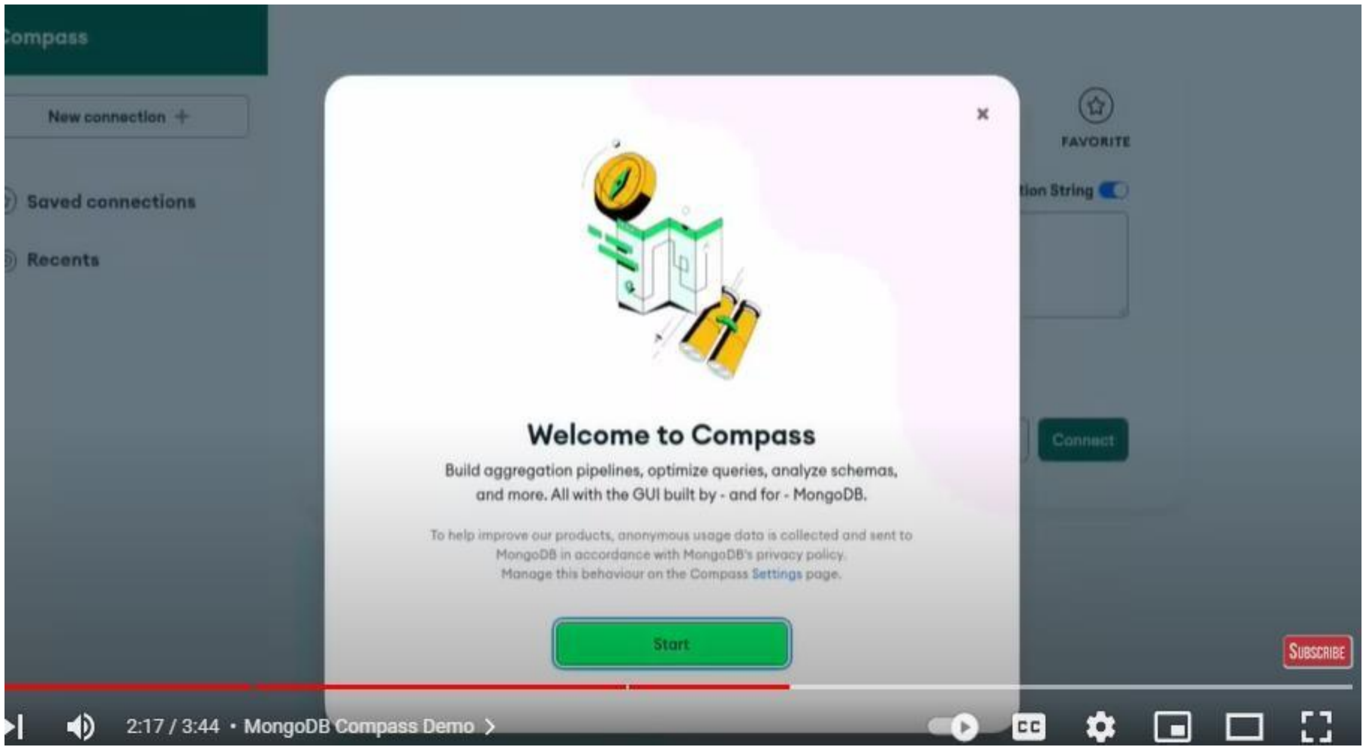
```
codebind@codebind:~/Downloads$ cd
codebind@codebind:~$ cd Downloads/
codebind@codebind:~/Downloads$ ls
firefox.tmp                            mongodb-compass_1.36.4_amd64.deb  mongodb-org-server_6.0.5_amd64.deb
google-chrome-stable_current_amd64.deb mongodb-mongosh_1.8.2_amd64.deb
codebind@codebind:~/Downloads$ sudo dpkg -i mongodb-compass_1.36.4_amd64.deb
[sudo] password for codebind:
Selecting previously unselected package mongodb-compass.
(Reading database ... 196041 files and directories currently installed.)
Preparing to unpack mongodb-compass_1.36.4_amd64.deb ...
Unpacking mongodb-compass (1.36.4) ...
dpkg: dependency problems prevent configuration of mongodb-compass:
 mongodb-compass depends on libgconf-2-4 | libgconf2-4; however:
  Package libgconf-2-4 is not installed.
  Package libgconf2-4 is not installed.
 mongodb-compass depends on libgconf-2-4; however:
  Package libgconf-2-4 is not installed.

dpkg: error processing package mongodb-compass (--install):
 dependency problems - leaving unconfigured
Processing triggers for mailcap (3.70+nmu1ubuntu1) ...
Processing triggers for gnome-menus (3.36.0-1ubuntu3) ...
Processing triggers for desktop-file-utils (0.26-1ubuntu3) ...
Errors were encountered while processing:
 mongodb-compass
codebind@codebind:~/Downloads$
```

>After go to terminal
>Then type MongoDB
>icon Display



mongodb-compass_1.36.4_amd64.deb /home/codebind/Downloads
mongodb-mongosh_1.8.2_amd64.deb /home/codebind/Downloads
mongodb-org-server_6.0.5_amd64.deb /home/codebind/Downloads

Mongolian Birga U+1800, : Mongolian Birga
Mongolian Ellipsis U+1801, : Mongolian Ellipsis
Mongolian Comma U+1802, : Mongolian Comma
Mongolian Full Stop U+1803, : Mongolian Full Stop



Welcome to Compass

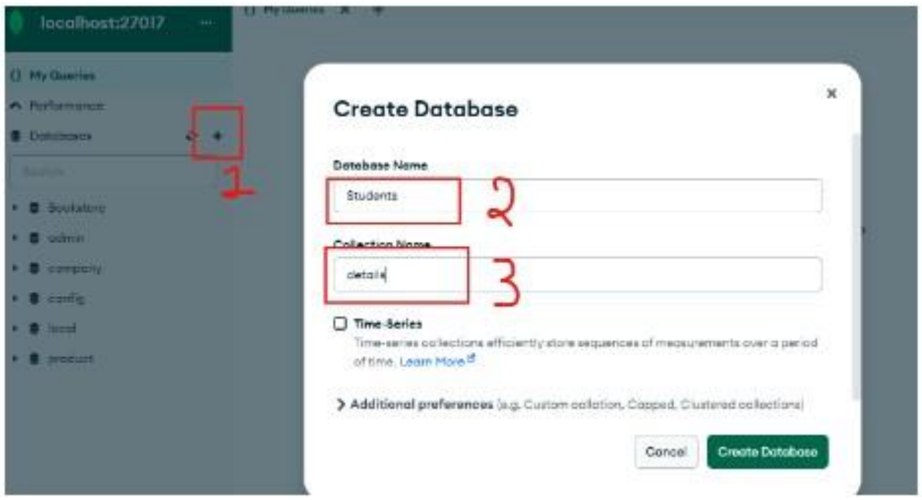
Build aggregation pipelines, optimize queries, analyze schemas, and more. All with the GUI built by - and for - MongoDB.

To help improve our products, anonymous usage data is collected and sent to MongoDB in accordance with MongoDB's privacy policy. Manage this behaviour on the Compass Settings page.

Program 1

a. Illustration of Where Clause, AND, OR operations in MongoDB.

Create a database **Students** and collection **details** in Mongo DB IDE**.**



Create Database

Database Name: Students

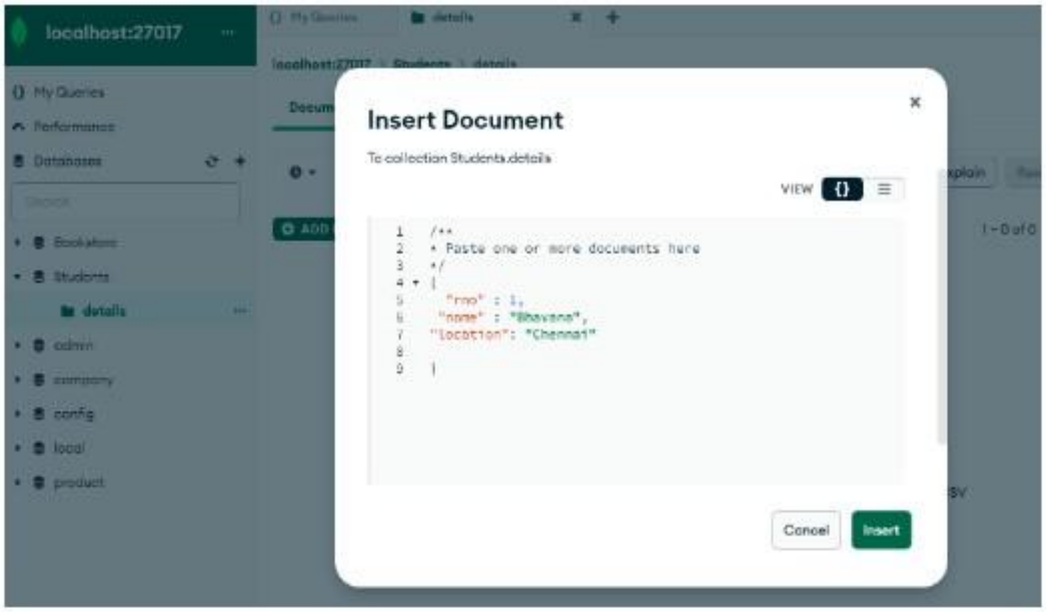
Collection Name: details

Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

Additional preferences (e.g. Custom collation, Capped, Clustered collections)

Add the following documents in the **details collection** in MongoDB IDE.

{
 "rno" : 1,
 "name" : "Bhavana",
"location": "Chennai"
}

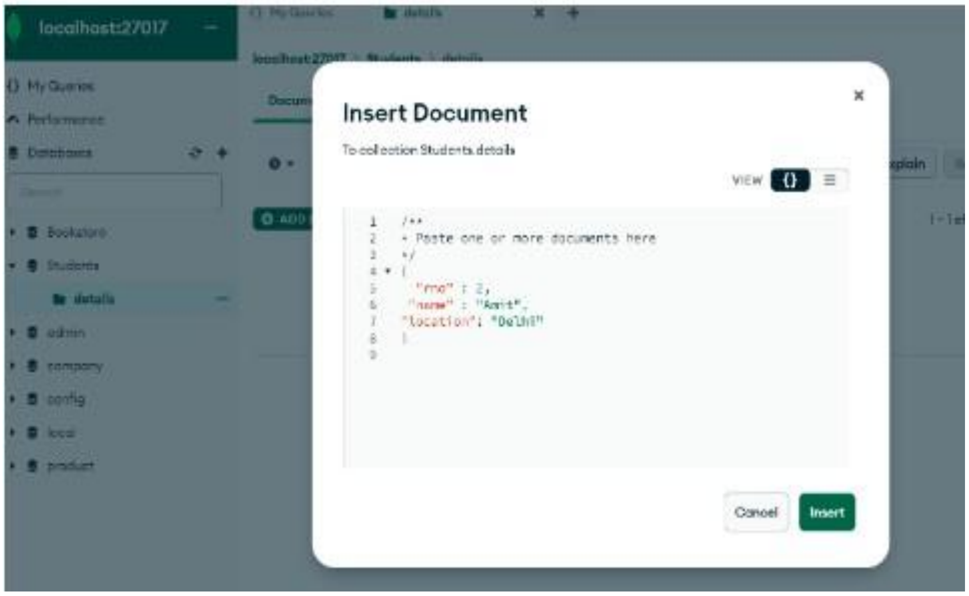


Insert Document

To collection Students.details

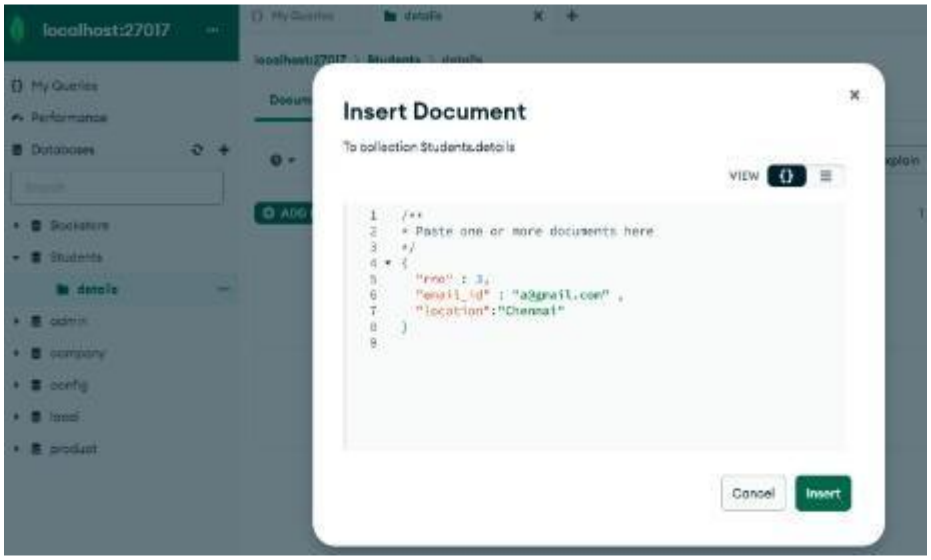
```
/**
* Paste one or more documents here
*/
{
  "rno" : 1,
 "name" : "Bhavana",
"location": "Chennai"

}
```

```
{
  "rno" : 2,
 "name" : "Amit",
"location": "Delhi"
}
```



```
/**
* Paste one or more documents here
*/
{
  "rno" : 2,
 "name" : "Amit",
"location": "Delhi"
}
```

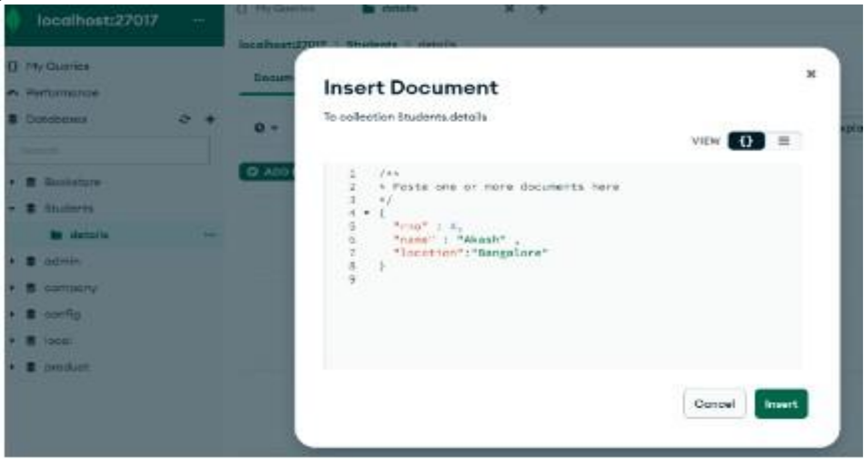
```
{
  "rno" : 3,
  "email_id" : "a@gmail.com" ,
  "location":"Chennai"
}
```



```
/**
* Paste one or more documents here
*/
{
  "rno" : 3,
  "email_id" : "a@gmail.com" ,
  "location":"Chennai"
}
```

```
        {
  "rno" : 4,
```

"name" : "Akash" ,
"location":"Bangalore"
}



```
/**
* Paste one or more documents here
*/
{
  "rno" : 4,
  "name" : "Akash" ,
  "location":"Bangalore"
}
```

1. **Where Clause in MongoDB:** In MongoDB, the find() method is used to query documents from a collection. The find() method can accept a query document as a parameter which acts as a "WHERE" clause.

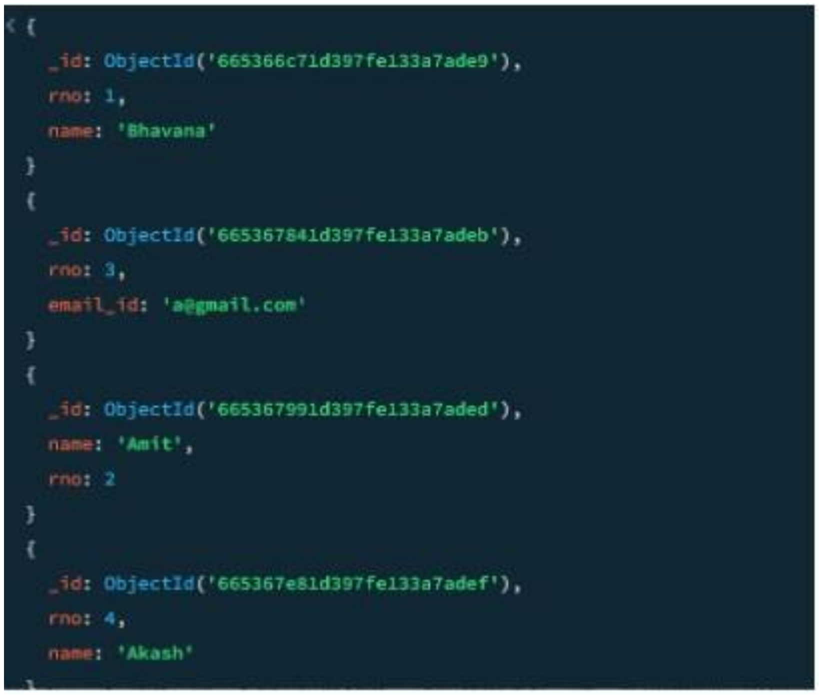
Syntax: db.collection.find({ field: value })

In Mongodb shell, execute the following code:

> use Students

> db.details.find()

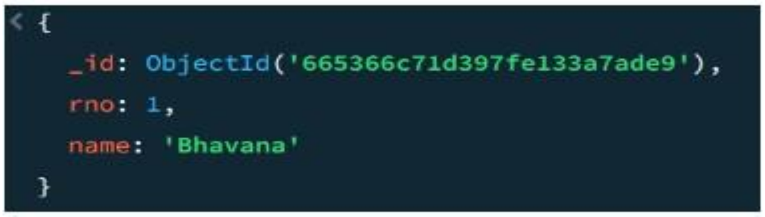
Output:



```
< {
    _id: ObjectId('665366c71d397fe133a7ade9'),
    rno: 1,
    name: 'Bhavana'
  }
  {
    _id: ObjectId('665367841d397fe133a7adeb'),
    rno: 3,
    email_id: 'a@gmail.com'
  }
  {
    _id: ObjectId('665367991d397fe133a7aded'),
    name: 'Amit',
    rno: 2
  }
  {
    _id: ObjectId('665367e81d397fe133a7adef'),
    rno: 4,
    name: 'Akash'
```

//findOne to show only first record
> db. details.findOne()

Output:



```
< {
    _id: ObjectId('665366c71d397fe133a7ade9'),
    rno: 1,
    name: 'Bhavana'
  }
```

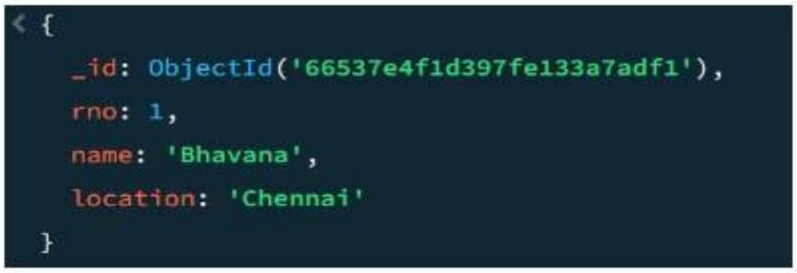
----

2. AND Operation in MongoDB: MongoDB provides the $and operator to perform logical AND operation between multiple conditions in a query.

Syntax: db.collection.find({ $and: [{ field1: value1 }, { field2: value2 }] })

>db.details.find({$and: [{"location": "Chennai"},{rno:1}] })

Output:



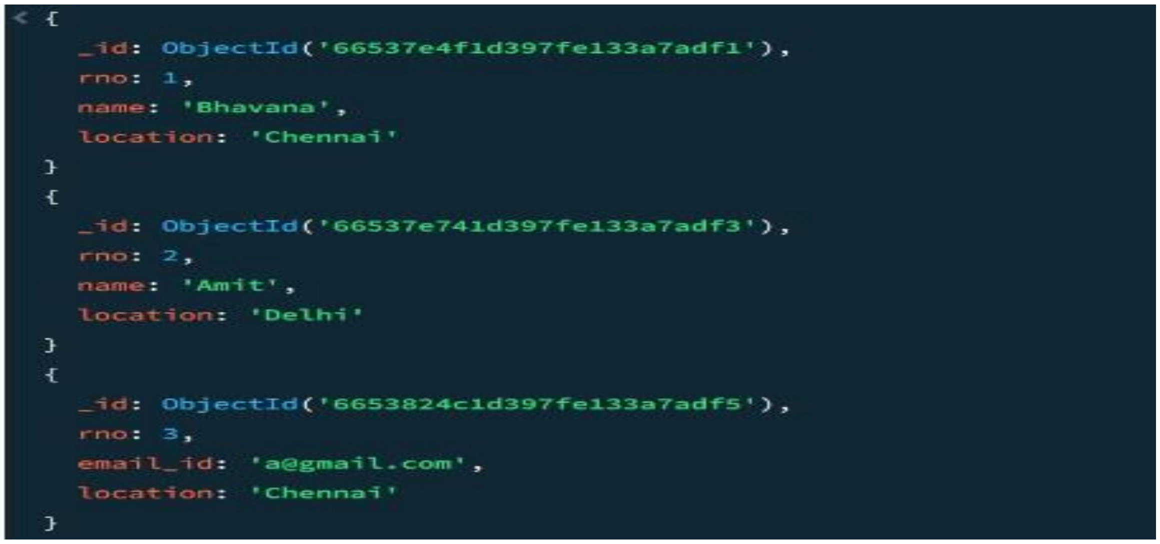
```
< {
    _id: ObjectId('66537e4f1d397fe133a7adf1'),
    rno: 1,
    name: 'Bhavana',
    location: 'Chennai'
  }
```

----

3. OR Operation in MongoDB: Similarly, MongoDB provides the $or operator to perform logical OR operation between multiple conditions in a query.

Syntax: db.collection.find({ $or: [{ field1: value1 }, { field2: value2 }] })

>db.details.find({$or: [{"location": "Chennai"}, {"location": "Delhi"}] })
Output:



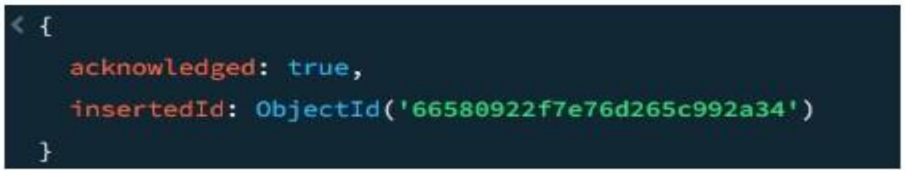
```
< {
    _id: ObjectId('66537e4f1d397fe133a7adf1'),
    rno: 1,
    name: 'Bhavana',
    location: 'Chennai'
  }
  {
    _id: ObjectId('66537e741d397fe133a7adf3'),
    rno: 2,
    name: 'Amit',
    location: 'Delhi'
  }
  {
    _id: ObjectId('6653824c1d397fe133a7adf5'),
    rno: 3,
    email_id: 'a@gmail.com',
    location: 'Chennai'
  }
```

b. Execute the Commands of MongoDB and operations in MongoDB: Insert, Query, Update, Delete and Projection. (Note: use any collection).

1. Insert Operation: Use the insertOne() method to insert a single document into a collection. **Syntax:** db.collection.insertOne({ field1: value1, field2: value2, field3: value3 })

Every row/document can be different than other
> db.details.insertOne({name:'Amar',rno:5},{name:’Ajay’,rno:10})

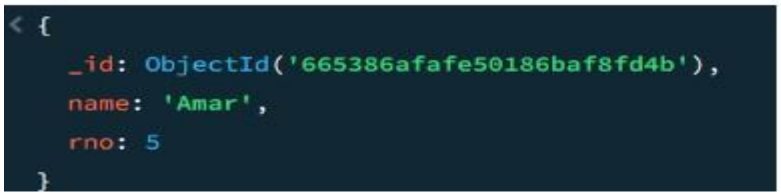
Output:



```
< {
    acknowledged: true,
    insertedId: ObjectId('66580922f7e76d265c992a34')
  }
```

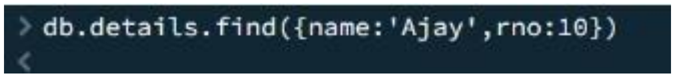
Verification Code:

>db.details.find({name:'Amar',rno:5})



```
< {
    _id: ObjectId('665386afafe50186baf8fd4b'),
    name: 'Amar',
    rno: 5
  }
```

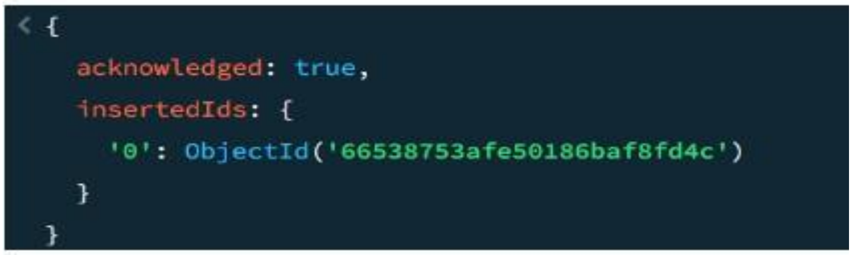
>db.details.find({name:'Ajay',rno:10})



```
> db.details.find({name:'Ajay',rno:10})
<
```

> db.details.insert({rno:6, email_id:'d@gmail.com'})

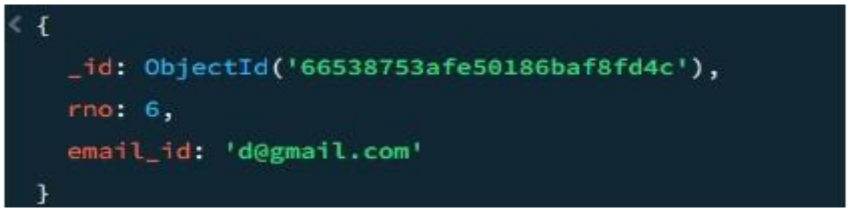
Output:



```
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('66538753afe50186baf8fd4c')
    }
  }
```

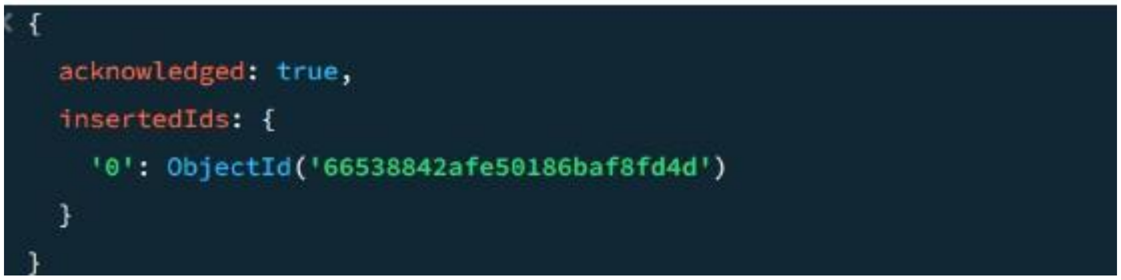
Verification Code:

>db.details.find({rno:6, email_id:'d@gmail.com'})



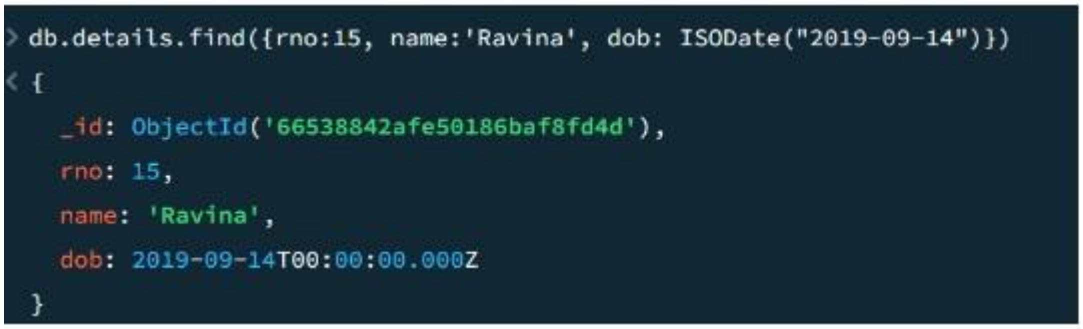
```
< {
    _id: ObjectId('66538753afe50186baf8fd4c'),
    rno: 6,
    email_id: 'd@gmail.com'
  }
```

// To insert date use ISODate function
> db.details.insert({rno:15, name:'Ravina', dob: ISODate("2019-09-14")})



```
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('66538842afe50186baf8fd4d')
    }
  }
```

Verification Code:

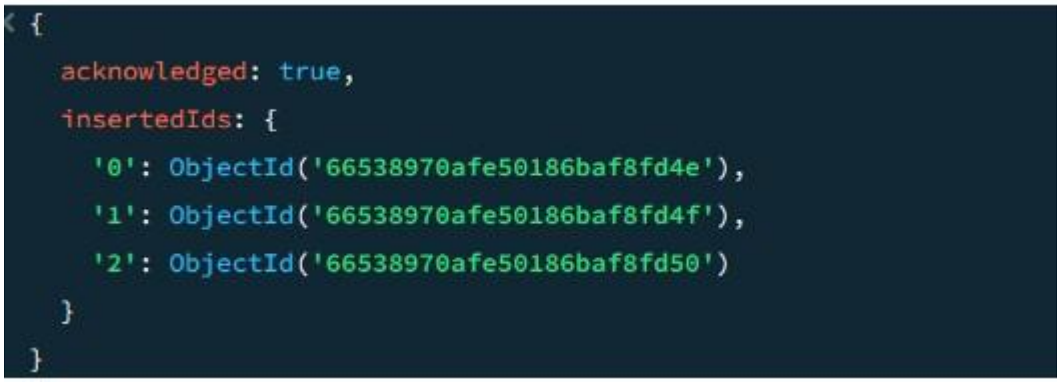


```
> db.details.find({rno:15, name:'Ravina', dob: ISODate("2019-09-14")})
< {
    _id: ObjectId('66538842afe50186baf8fd4d'),
    rno: 15,
    name: 'Ravina',
    dob: 2019-09-14T00:00:00.000Z
  }
```

//Insert multiple documents at once

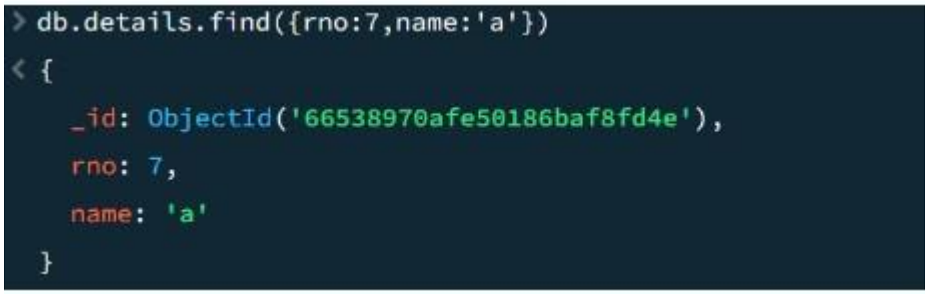
> db.details.insert([{rno:7,name:'a'},{rno:8,name:'b'},{rno:8,name:'c'}])

Output:



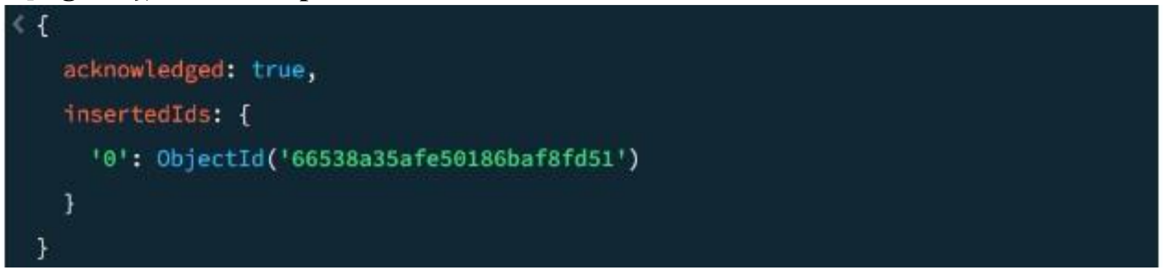
```
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('66538970afe50186baf8fd4e'),
      '1': ObjectId('66538970afe50186baf8fd4f'),
      '2': ObjectId('66538970afe50186baf8fd50')
    }
  }
```

Verification Code:



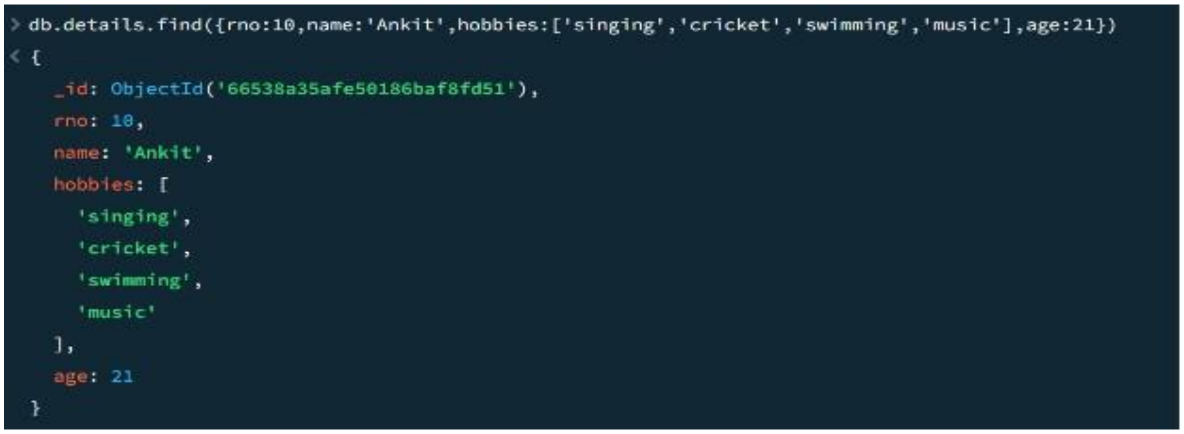
```
> db.details.find({rno:7,name:'a'})
< {
    _id: ObjectId('66538970afe50186baf8fd4e'),
    rno: 7,
    name: 'a'
  }
```

// to insert multiple values for one key using []
>db.details.insert({rno:10,name:'Ankit',hobbies:['singing','cricket','swimming', 'music'],age:21}) **Output:**



```
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('66538a35afe50186baf8fd51')
    }
  }
```

Verification Code:

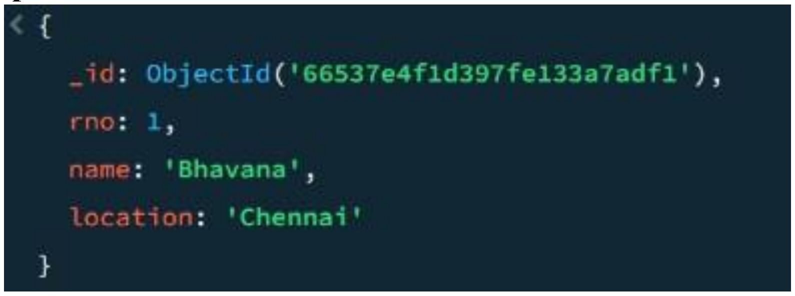


```
> db.details.find({rno:10,name:'Ankit',hobbies:['singing','cricket','swimming','music'],age:21})
< {
    _id: ObjectId('66538a35afe50186baf8fd51'),
    rno: 10,
    name: 'Ankit',
    hobbies: [
      'singing',
      'cricket',
      'swimming',
      'music'
    ],
    age: 21
  }
```

--- 2.
Query Operation: Use the find() method to query documents from a collection.

Syntax: db.collection.find({ field: value })

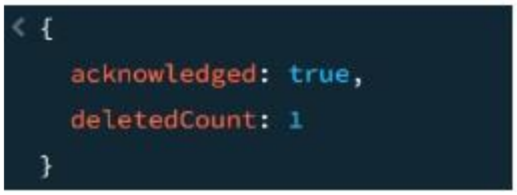
>db.details.find({rno:1})
Output:



```
< {
    _id: ObjectId('66537e4f1d397fe133a7adf1'),
    rno: 1,
    name: 'Bhavana',
    location: 'Chennai'
  }
```

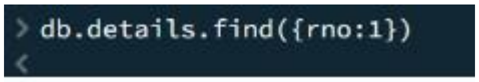
3.Delete Operation: Use the deleteOne() method to delete a single document from a collection.
Syntax: db.collection.deleteOne({ field: value })

>db.details.deleteOne({rno:1}) Output:



```
< {
    acknowledged: true,
    deletedCount: 1
  }
```

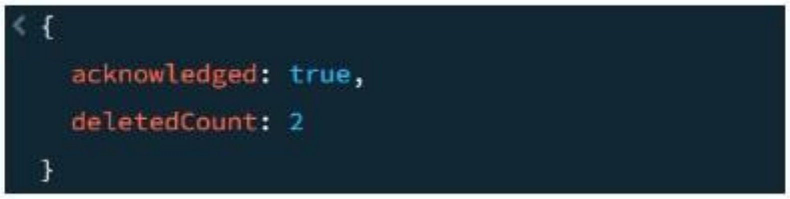
Verification Code:



```
> db.details.find({rno:1})
<
```

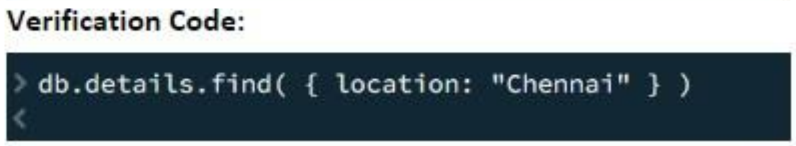
>db. details.deleteMany({ location: "Chennai" })

Output



```
< {
    acknowledged: true,
    deletedCount: 2
  }
```

:



Verification Code:

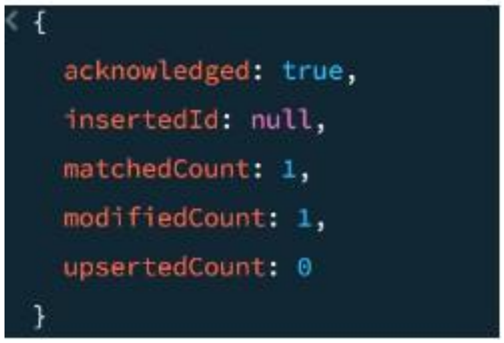
```
> db.details.find( { location: "Chennai" } )
<
```

Update Operation: Use the updateOne() method to update a single document in a collection.

Syntax: db.collection.updateOne({ field: value }, { $set: { fieldToUpdate: newValue } })

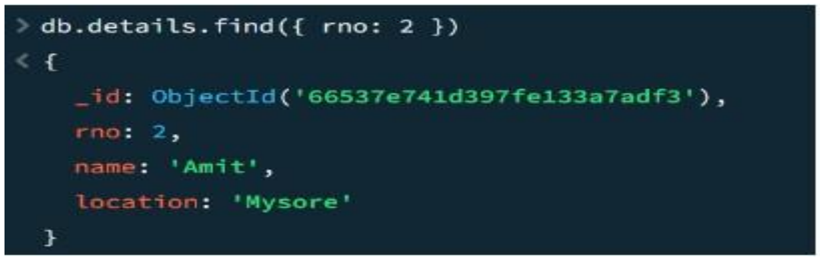
> db.details.updateOne({ rno: 2 }, {$set: { location: "Mysore" } })

Output:



```
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

Verification Code:



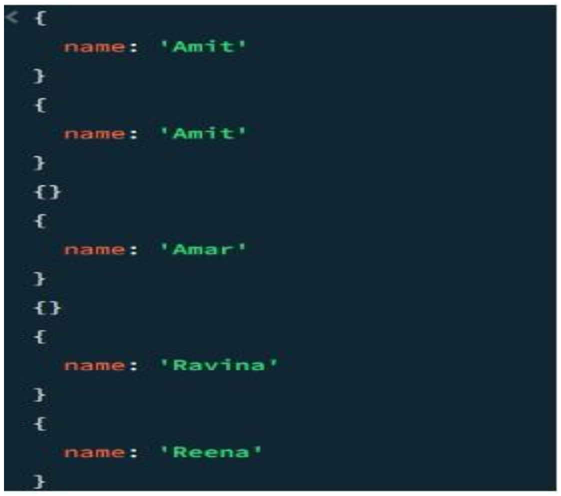
```
> db.details.find({ rno: 2 })
< {
    _id: ObjectId('66537e741d397fe133a7adf3'),
    rno: 2,
    name: 'Amit',
    location: 'Mysore'
  }
```

4.Projection Operation: Use the second parameter of the find() method to specify which fields to include or exclude in the query result.

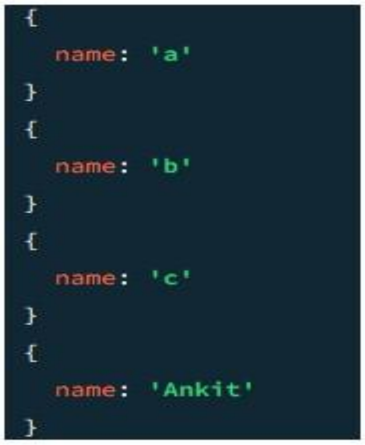
Syntax: db.collection.find({}, { field1: 1, field2: 1, _id: 0 })

// Find command to show only names without condition

> db. details.find({},{name:1,_id:0})



```
< {
    name: 'Amit'
  }
  {
    name: 'Amit'
  }
  {}
  {
    name: 'Amar'
  }
  {}
  {
    name: 'Ravina'
  }
  {
    name: 'Reena'
  }
```



```
{
  name: 'a'
}
{
  name: 'b'
}
{
  name: 'c'
}
{
  name: 'Ankit'
}
```

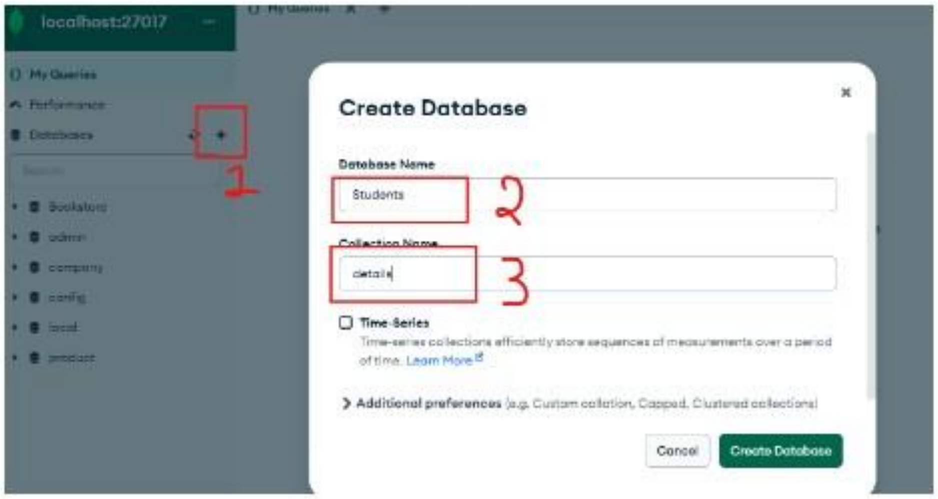
Program 2

a. Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection.

Syntax: db.collection.find({}, { field1: 1, field2: 1, _id: 0 })

- db.collection.find({}) is used to retrieve all documents from the collection.
- { field1: 1, field2: 1, _id: 0 } specifies the projection document where:
- field1: 1 and field2: 1 indicates that these fields will be included in the result.
- _id: 0 indicates that the _id field will be excluded from the result.

Create a database **Students** and collection **details** in Mongo DB IDE**.**



Create Database

Database Name: Students

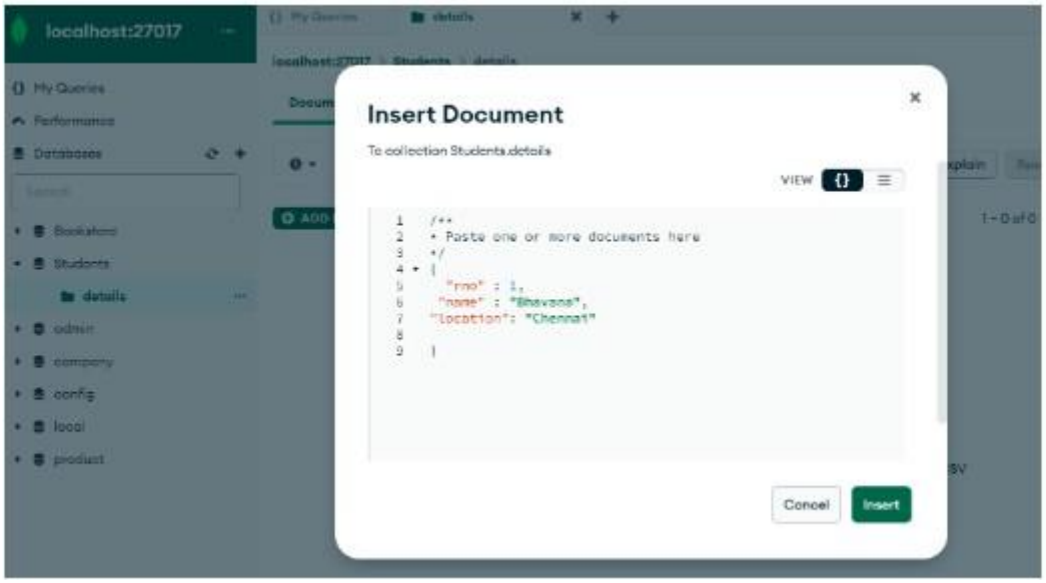
Collection Name: details

Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

Additional preferences (e.g. Custom collation, Capped, Clustered collections)

Add the following documents in the **details collection** in MongoDB IDE.

```
{
  "rno" : 1,
 "name" : "Bhavana",
"location": "Chennai"
}
```

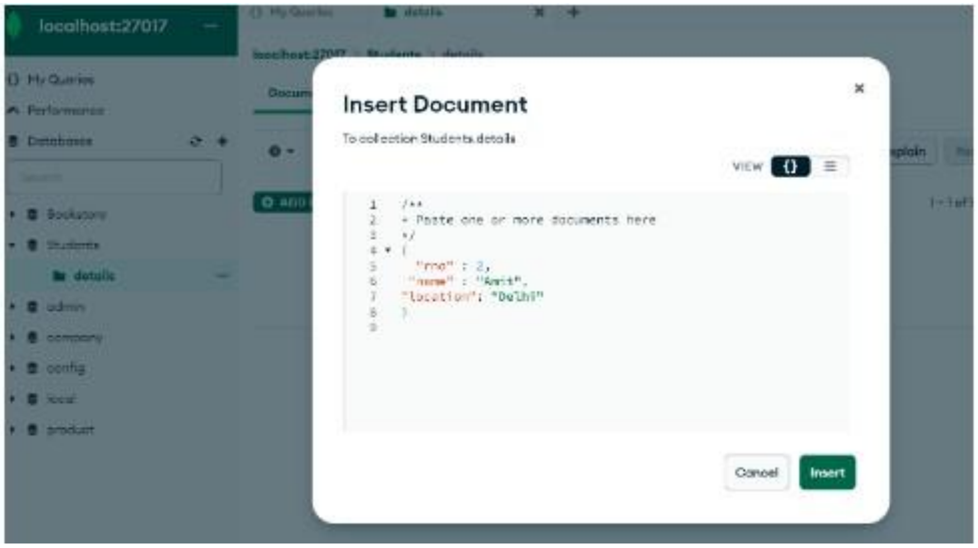


Insert Document

To collection Students.details

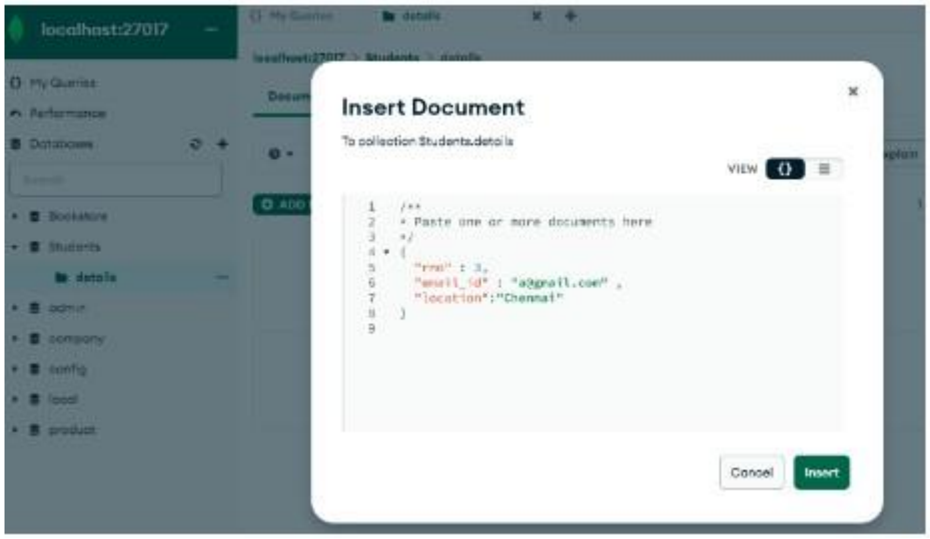
```
/**
* Paste one or more documents here
*/
{
  "rno" : 1,
 "name" : "Bhavana",
"location": "Chennai"

}
```

```
{
  "rno" : 2,
 "name" : "Amit",
"location": "Delhi"
}
```



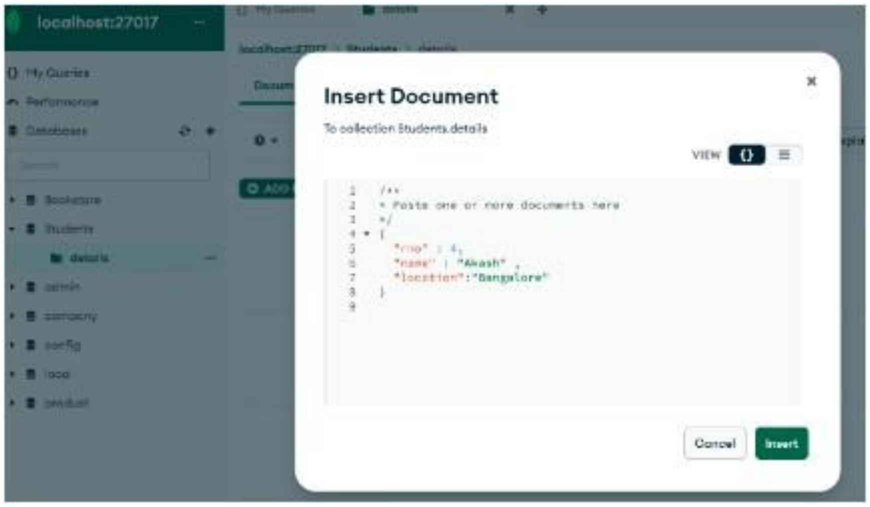
```
/**
* Paste one or more documents here
*/
{
  "rno" : 2,
  "name" : "Amit",
"location": "Delhi"
}
```

{
"rno" : 3,
"email_id" : "a@gmail.com" ,
"location":"Chennai"
}



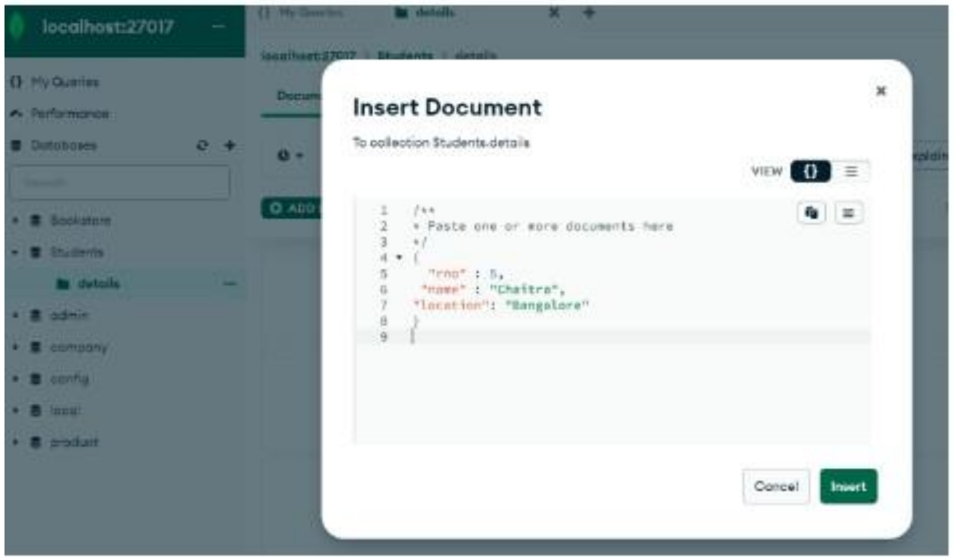
```
/**
* Paste one or more documents here
*/
{
  "rno" : 3,
  "email_id" : "a@gmail.com" ,
  "location":"Chennai"
}
```

{
"rno" : 4,
"name" : "Akash" ,
"location":"Bangalore"
}



```
/**
* Paste one or more documents here
*/
{
  "rno" : 4,
  "name" : "Akash" ,
  "location":"Bangalore"
}
```

```
{
  "rno" : 5,
 "name" : "Chaitra",
"location": "Bangalore"
}
```

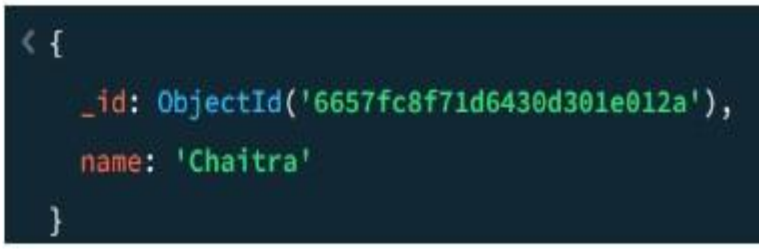


```
/**
* Paste one or more documents here
*/
{
  "rno" : 5,
 "name" : "Chaitra",
"location": "Bangalore"
}
```

//Find command with condition with giving name field only to show

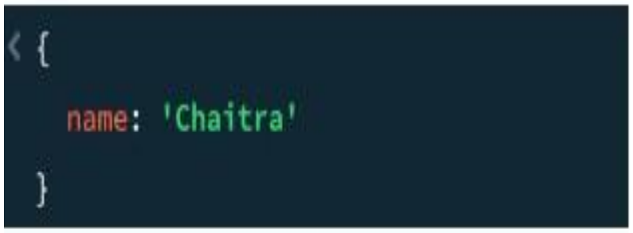
```
> db. details.find({rno:5},{name:1})
```

Output:



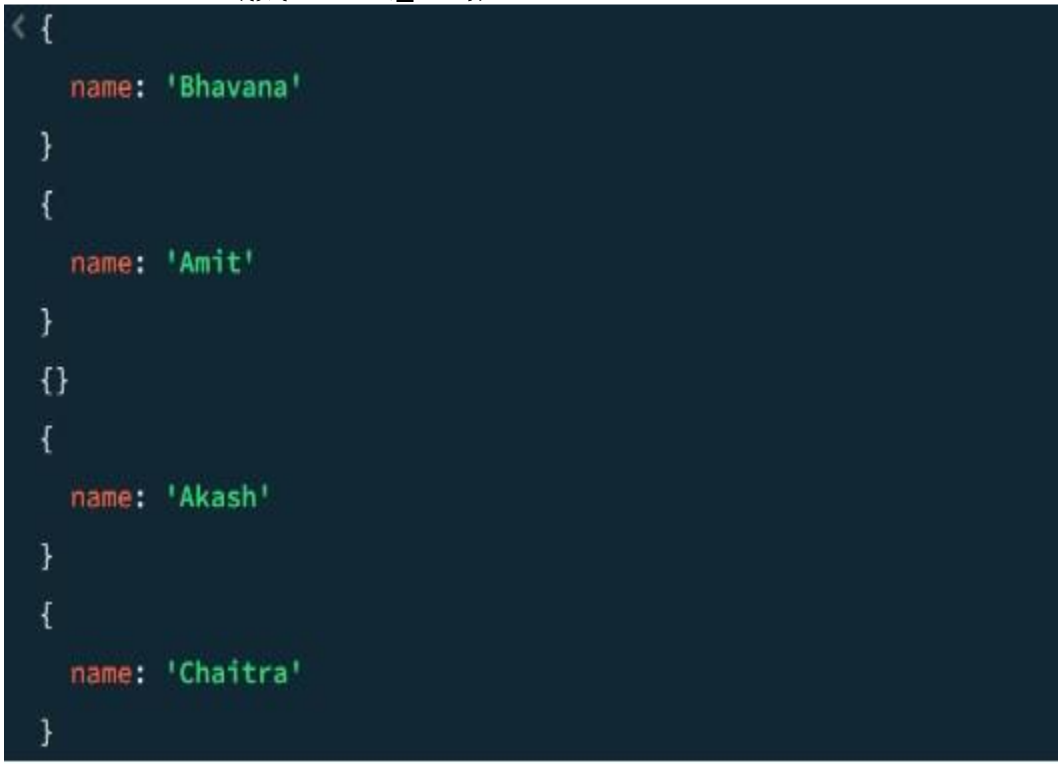
```
< {
    _id: ObjectId('6657fc8f71d6430d301e012a'),
    name: 'Chaitra'
  }
```

//Find command with condition with giving name field only to show and _id to hide
>db. details.find({rno:5},{name:1,_id:0}) Output:



```
< {
    name: 'Chaitra'
  }
```

// Find command to show only names without condition
> db. details.find({},{name:1,_id:0})



```
< {
    name: 'Bhavana'
  }
  {
    name: 'Amit'
  }
  {}
  {
    name: 'Akash'
  }
  {
    name: 'Chaitra'
  }
```

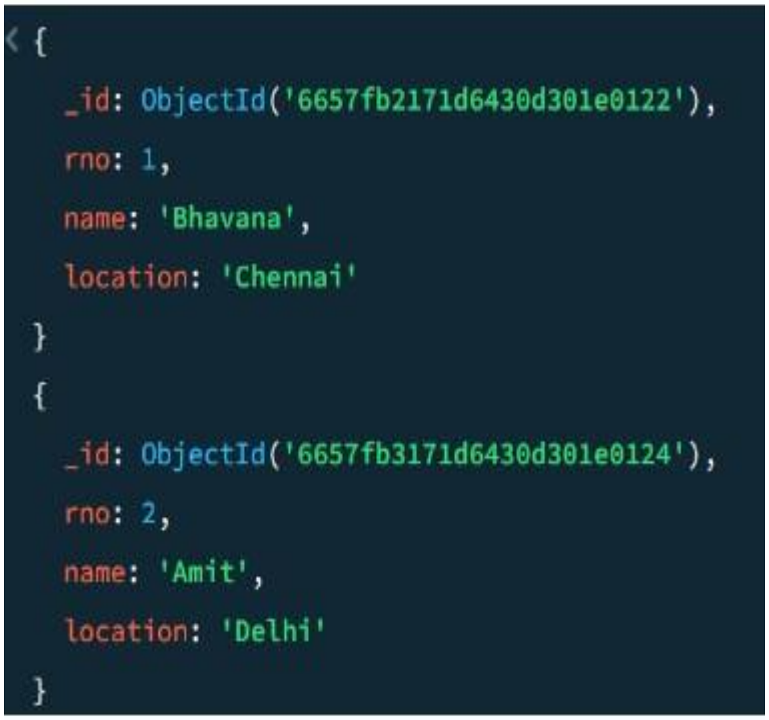
b. Develop a MongoDB query to display the first 5 documents from the results obtained in a. [use of limit and find]

Limit Operation: Used to restrict the number of documents returned by a query. This is particularly useful when you're dealing with large datasets and you only need a subset of documents. **Syntax:** db.collection.find({}, { field1: 1, field2: 1, _id: 0 }).limit(5) Limit (5) limits the number of documents returned to 5.

 // Limit use to show only some records from starting- following command shows only first 2 records from collection >

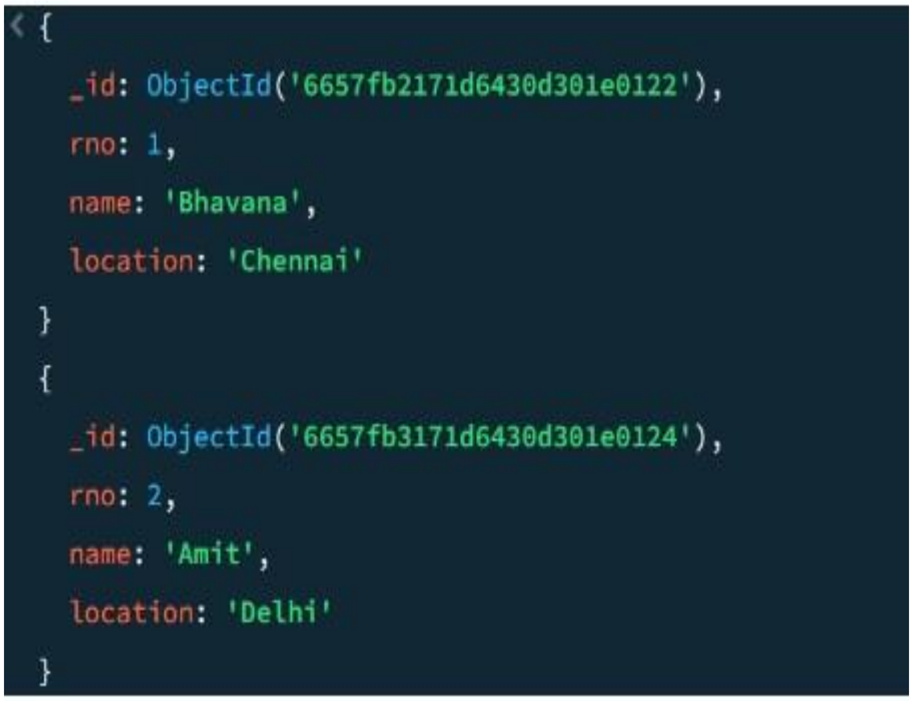
db. details.find().limit(2)

Output:

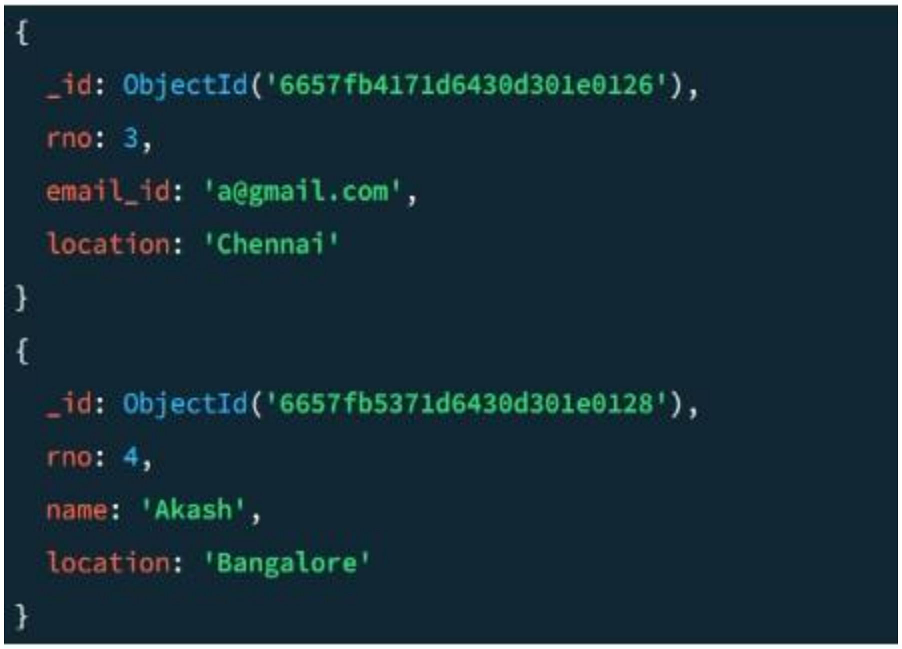


```
< {
    _id: ObjectId('6657fb2171d6430d301e0122'),
    rno: 1,
    name: 'Bhavana',
    location: 'Chennai'
  }
  {
    _id: ObjectId('6657fb3171d6430d301e0124'),
    rno: 2,
    name: 'Amit',
    location: 'Delhi'
  }
```

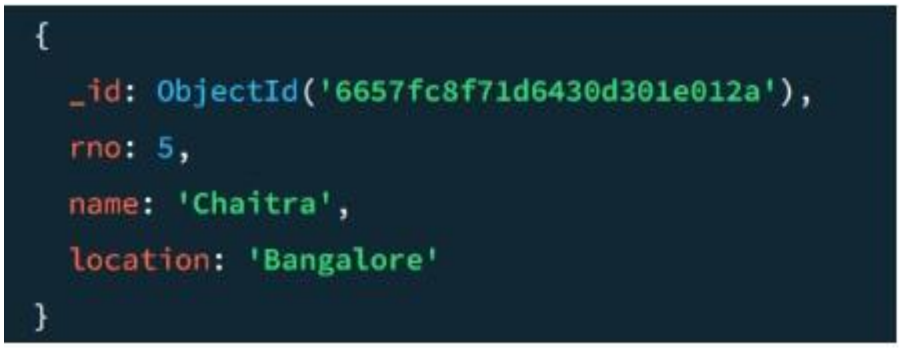
>db. details.find().limit(5) Output:



```
< {
    _id: ObjectId('6657fb2171d6430d301e0122'),
    rno: 1,
    name: 'Bhavana',
    location: 'Chennai'
  }
  {
    _id: ObjectId('6657fb3171d6430d301e0124'),
    rno: 2,
    name: 'Amit',
    location: 'Delhi'
  }
```



```
{
  _id: ObjectId('6657fb4171d6430d301e0126'),
  rno: 3,
  email_id: 'a@gmail.com',
  location: 'Chennai'
}
{
  _id: ObjectId('6657fb5371d6430d301e0128'),
  rno: 4,
  name: 'Akash',
  location: 'Bangalore'
}
```



```
{
  _id: ObjectId('6657fc8f71d6430d301e012a'),
  rno: 5,
  name: 'Chaitra',
  location: 'Bangalore'
}
```

Program 3

a. Execute query selectors (comparison selectors, logical selectors) and list out the results on any collection

b. Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection

Comparison Selectors: Comparison selectors are used to compare fields against specific values or other fields. Here are some common comparison selectors:

$eq - Matches values that are equal to a specified value.

$ne - Matches all values that are not equal to a specified value.

$gt - Matches values that are greater than a specified value.

$gte - Matches values that are greater than or equal to a specified value.

$lt - Matches values that are less than a specified value.

$lte - Matches values that are less than or equal to a specified value.

$in - Matches any of the values specified in an array.

$nin - Matches none of the values specified in an array.

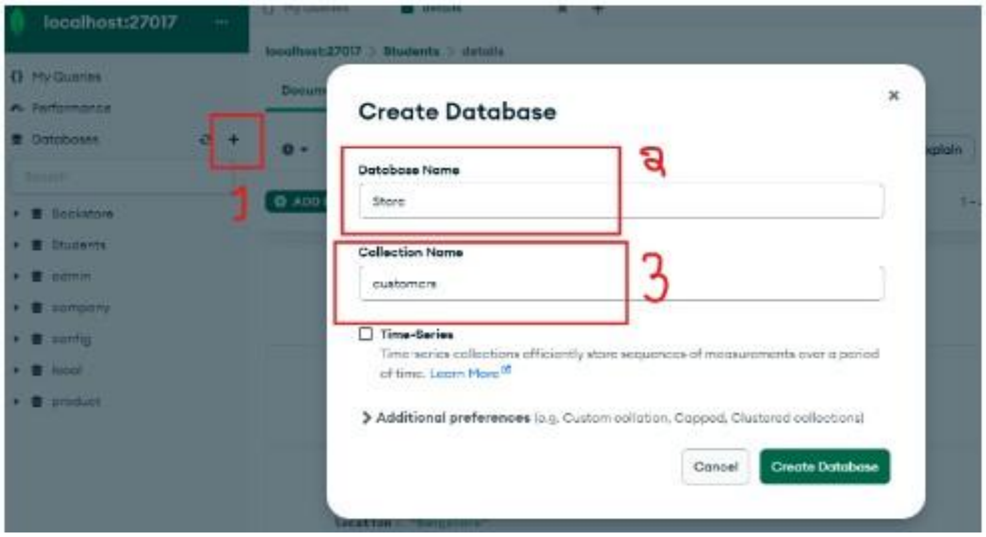
Logical Selectors: Logical selectors are used to combine multiple conditions in a query. Here are some common logical selectors:

$and - Joins query clauses with a logical AND and requires that all conditions be true.

$or - Joins query clauses with a logical OR and requires that at least one condition be true. **$not** - Inverts the effect of a query expression and returns documents that do not match the query expression.

$nor - Joins query clauses with a logical NOR and requires that none of the conditions be true.

Create a database **Store** and collection **customers** in Mongo DB IDE**.**



In MongoDB Shell:

>use Store

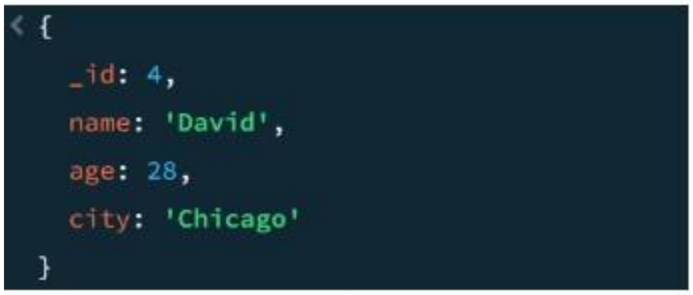
```
> db.customers.insertMany([ { _id: 1, name: "Alice", age: 30, city: "New York" },
        { _id: 2, name: "Bob", age: 25, city: "San Francisco" },
                                          { _id: 3, name: "Charlie", age: 35, city: "Los Angeles" },
                                          { _id: 4, name: "David", age: 28, city: "Chicago" },
                                           {_id: 5, name: "Eve", age: 32, city: "Miami" } ])
```

a. Execute query selectors (comparison selectors, logical selectors) and list out the results on any collection.

Using Comparison Selectors

1.Find customers aged 28:

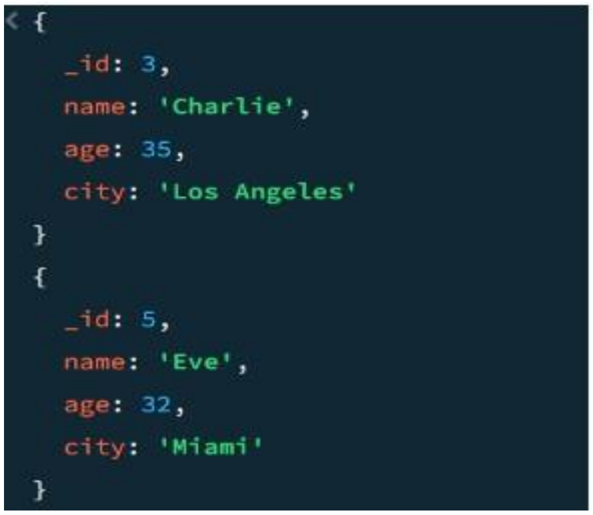
>db.customers.find({ "age": { "$eq": 28 } }) Output:



```
< {
    _id: 4,
    name: 'David',
    age: 28,
    city: 'Chicago'
  }
```

2.Find customers older than 30:

>db.customers.find({ "age": { "$gt": 30 } }) Output:



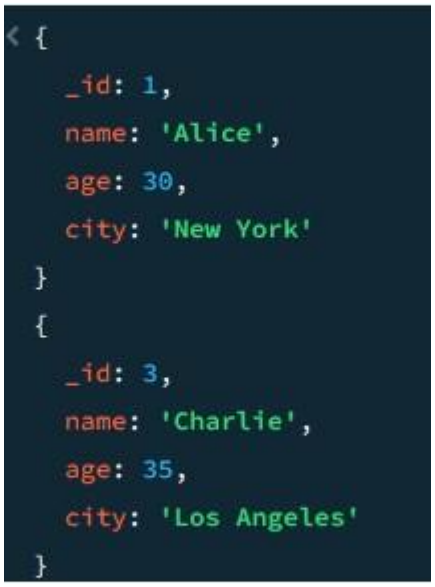
```
{
  _id: 3,
  name: 'Charlie',
  age: 35,
  city: 'Los Angeles'
}
{
  _id: 5,
  name: 'Eve',
  age: 32,
  city: 'Miami'
}
```

Using Logical Selectors

3. Find customers in city is New York OR city is Los Angeles:

```
>db.customers.find({
   $or: [
      { city: "New York" },
      { city: " Los Angeles"  }
   ] })
```

Output:

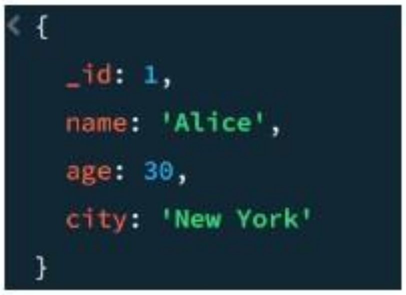


```
{
  _id: 1,
  name: 'Alice',
  age: 30,
  city: 'New York'
}
{
  _id: 3,
  name: 'Charlie',
  age: 35,
  city: 'Los Angeles'
}
```

4. Find customers age 30 and city New York

```
>db.customers.find({
   $and: [
      { age: 30 },
    { city:"New York"  }
  ] })
```

Output:



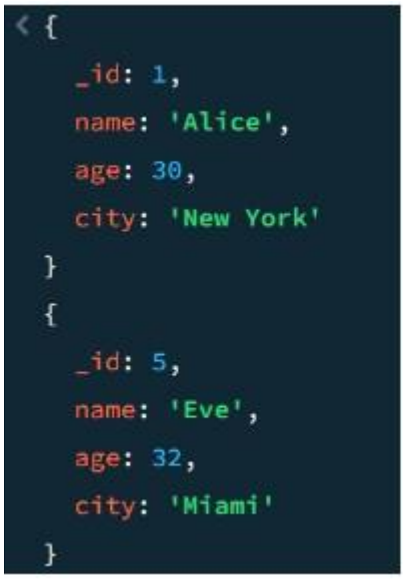
```
< {
    _id: 1,
    name: 'Alice',
    age: 30,
    city: 'New York'
  }
```

Using Both Comparison and Logical Selectors

5. Find customers greater than or equal to 18, less than 35, in city New York or Miami

```
>db.customers.find({$and: [
    { age: { $gte: 18 } },        // age greater than or equal to 18
     { age: { $lt: 35 } },         // age less than 35
     { city: { $in: ["New York", "Miami"] } }   // city is either "New York" or "Miami"
  ]})
```

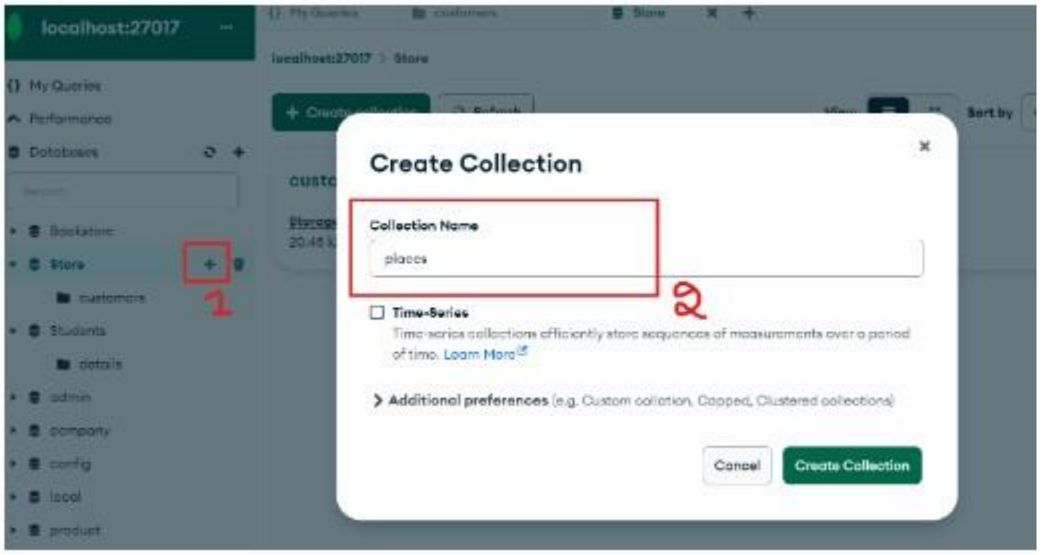
Output:



```
< {
    _id: 1,
    name: 'Alice',
    age: 30,
    city: 'New York'
  }
  {
    _id: 5,
    name: 'Eve',
    age: 32,
    city: 'Miami'
  }
```

b. Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection

Under database **Store**, create a collection **places** in Mongo DB IDE**.**



Geospatial Selectors: MongoDB supports geospatial queries for geospatial data. It provides two types of geospatial indexes: 2d indexes and 2d sphere indexes.

Add the following documents in the **places collection** in MongoDB Shell.

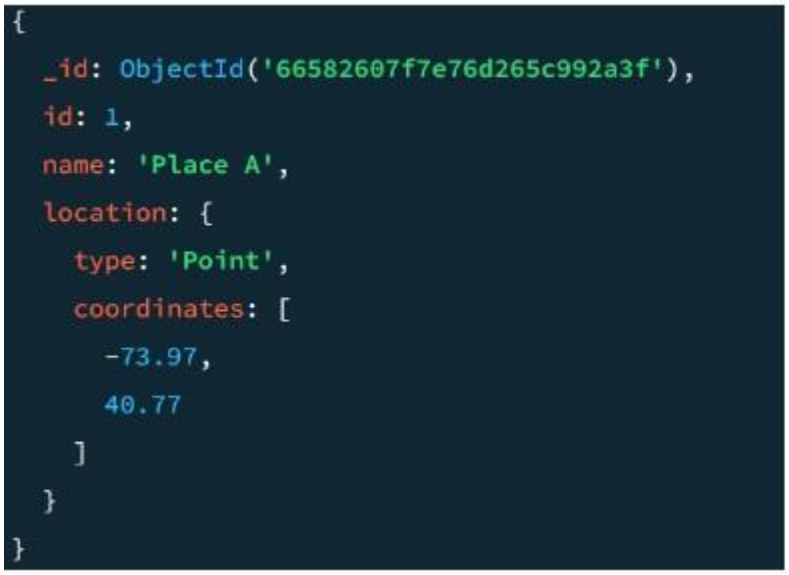
```
  >db.places.insertMany([
{ id: 1, name: "Place A", location: { type: "Point", coordinates: [ -73.97, 40.77 ] } }, // New York
 { id: 2, name: "Place B", location: { type: "Point", coordinates: [ -122.43, 37.77 ] } }, // San Francisco
   { id: 3, name: "Place C", location: { type: "Point", coordinates: [ -118.25, 34.05 ] } }, // Los Angeles
   { id: 4, name: "Place D", location: { type: "Point", coordinates: [ -87.63, 41.88 ] } }, // Chicago
   { id: 5, name: "Place E", location: { type: "Point", coordinates: [ -80.19, 25.77 ] } }  // Miami
])
```

Create a 2dsphere Index on location:

```
>db.places.createIndex({ location: "2dsphere" })
```

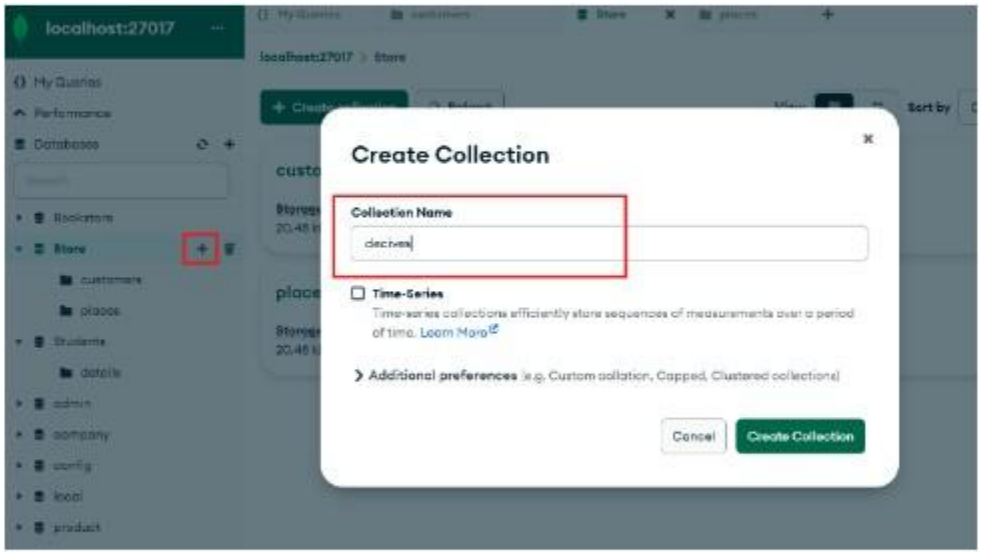
Geospatial Query (Find places within 10km of a given point):

```
>db.places.find({ location: {
          $near: {     $geometry: {
type: "Point",              coordinates:
[ -73.97, 40.77 ]
          }, $maxDistance: 10000 // 10km in meters
             }}})
```


```
{
  _id: ObjectId('66582607f7e76d265c992a3f'),
  id: 1,
  name: 'Place A',
  location: {
    type: 'Point',
    coordinates: [
      -73.97,
      40.77
    ]
  }
}
```

2. Bitwise Selectors

Under database **Store**, create a collection **devices** in Mongo DB IDE**.**


Create Collection

Collection Name: devices

Time-Series

Time-series collections efficiently store sequences of measurements over a period of time. Learn More

Additional preferences (e.g. Custom collation, Capped, Clustered collections)

We'll use a collection devices with fields id, name, and status (where status is a bitwise flag).

>db.devices.insertMany([

{ id: 1, name: "Device A", status: 5 }, // 0101 in binary

{ id: 2, name: "Device B", status: 3 }, // 0011 in binary

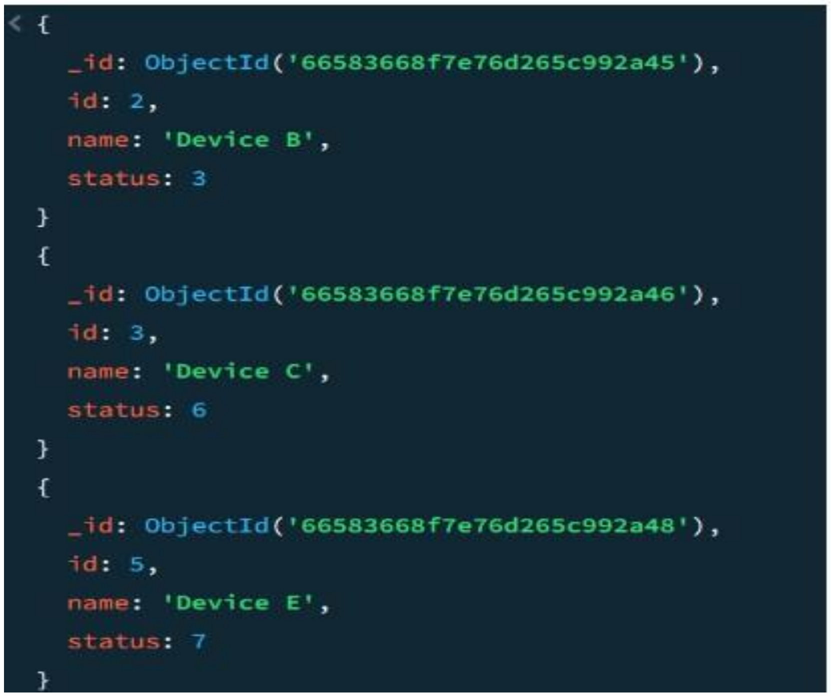
{ id: 3, name: "Device C", status: 6 }, // 0110 in binary

{ id: 4, name: "Device D", status: 12 }, // 1100 in binary

{ id: 5, name: "Device E", status: 7 } // 0111 in binary])

Bitwise AND Query (Find devices where the 2nd bit is set):

>db.devices.find({ status: { $bitsAllSet: 2 } })

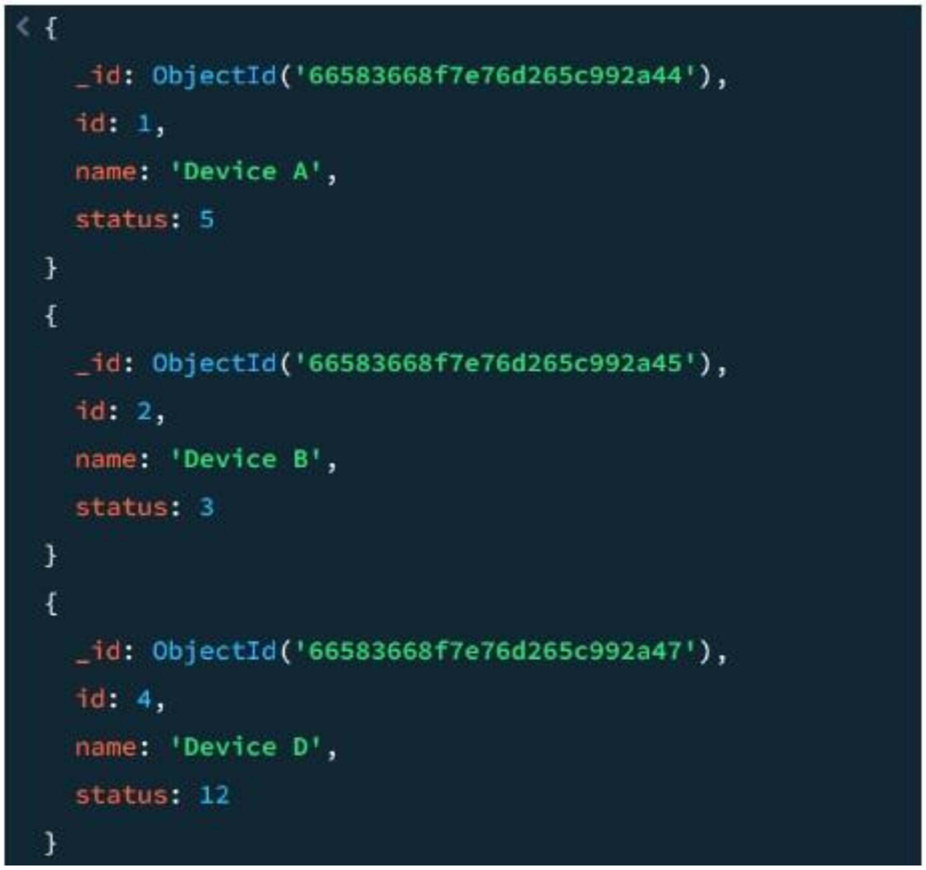


```
< {
    _id: ObjectId('66583668f7e76d265c992a45'),
    id: 2,
    name: 'Device B',
    status: 3
  }
  {
    _id: ObjectId('66583668f7e76d265c992a46'),
    id: 3,
    name: 'Device C',
    status: 6
  }
  {
    _id: ObjectId('66583668f7e76d265c992a48'),
    id: 5,
    name: 'Device E',
    status: 7
  }
```

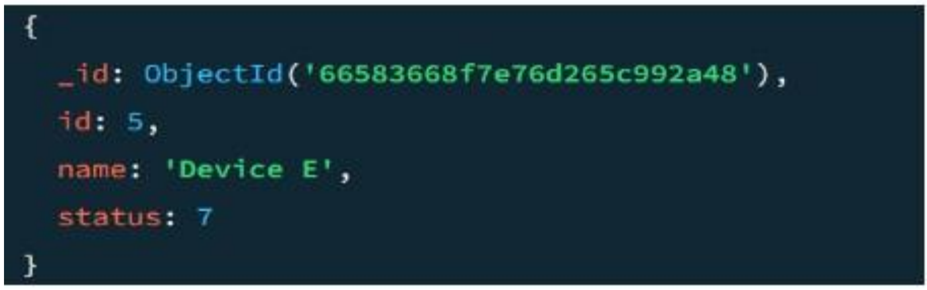
Bitwise OR Query (Find devices where any bit in 0101 is set):

To find all devices where any of the bits at positions 0 or 3 are set (i.e., either the least significant bit or the fourth bit is set), you can use the `$bitsAnySet` operator as follows:

> **db.devices.find({ "status": { "$bitsAnySet": [0, 3] } })**



```
< {
    _id: ObjectId('66583668f7e76d265c992a44'),
    id: 1,
    name: 'Device A',
    status: 5
  }
  {
    _id: ObjectId('66583668f7e76d265c992a45'),
    id: 2,
    name: 'Device B',
    status: 3
  }
  {
    _id: ObjectId('66583668f7e76d265c992a47'),
    id: 4,
    name: 'Device D',
    status: 12
  }
```



```
{
  _id: ObjectId('66583668f7e76d265c992a48'),
  id: 5,
  name: 'Device E',
  status: 7
}
```

In MongoDB, the main geospatial query operators include:

1. **$geoWithin**: Finds documents within a specified geometry (e.g., a polygon).
2. **$geoIntersects**: Finds documents that intersect with a specified geometry.
3. **$near**: Finds documents near a specified point, using a 2dsphere index.
4. **$nearSphere**: Similar to $near, but calculates distances using spherical geometry.
5. **$center**: Finds documents within a circular area (used with legacy coordinate pairs).
6. **$centerSphere**: Finds documents within a circular area on a sphere (used with legacy coordinate pairs).
7. **$box**: Finds documents within a rectangular area (used with legacy coordinate pairs).
8. **$polygon**: Finds documents within a polygon defined by multiple points (used with legacy coordinate pairs).

In MongoDB, the main bitwise query operators include:

1. **$bitsAllClear**: Matches documents where all of the given bit positions are clear (i.e., 0).
2. **$bitsAllSet**: Matches documents where all of the given bit positions are set (i.e., 1).
3. **$bitsAnyClear**: Matches documents where any of the given bit positions are clear (i.e., 0).
4. **$bitsAnySet**: Matches documents where any of the given bit positions are set (i.e., 1).

Explanation

- **Geospatial Selector**: ○ **$near**: Finds documents near a specified point. Requires a `2dsphere` index on the location field. ○ **$geometry**: Specifies the reference point as a GeoJSON object. ○ **$maxDistance**: Limits the distance from the reference point (in meters).
- **Bitwise Selector**: ○ **$bitsAllSet**: Matches documents where all of the given bit positions are 1. ○ **$bitsAnySet**: Matches documents where any of the given bit positions are 1.

By executing these queries, you can filter documents based on geospatial proximity and bitwise conditions.

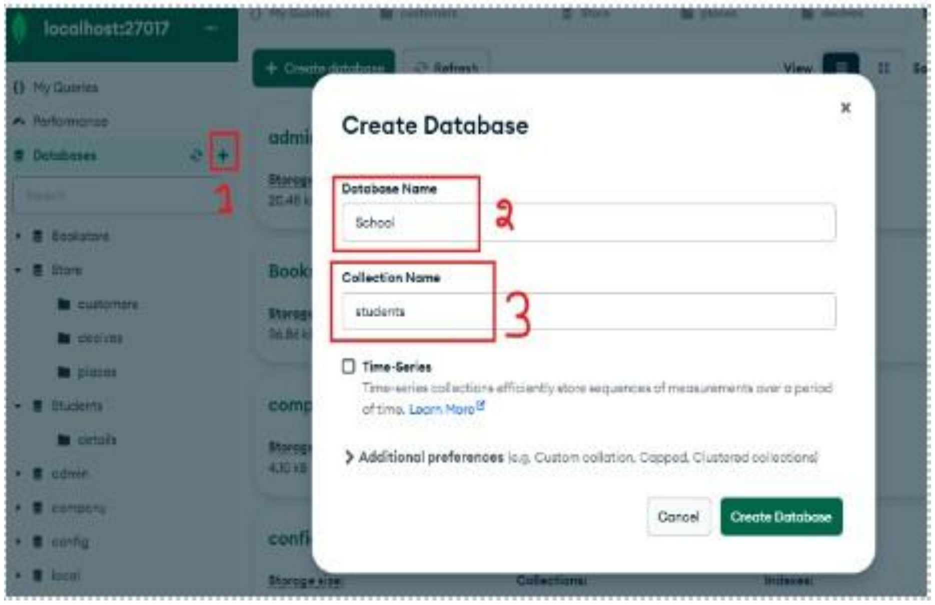
Program 4

Create and demonstrate how projection operators ($, $elematch and $slice) would be used in the MongoDB.

$elemMatch: The $elemMatch operator is used to match documents that contain an array field with at least one element that matches all the specified query criteria.

$slice: The $slice projection operator is used within the projection document to limit the number of elements returned from an array field.

Create a database **School** and collection **students** in Mongo DB IDE**.**



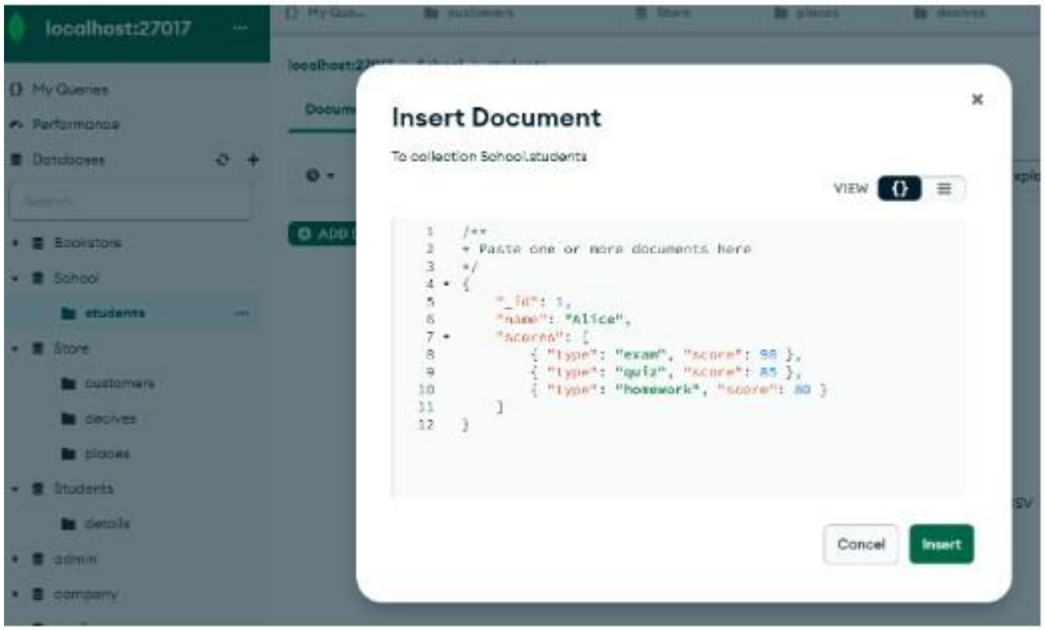
Create Database

Database Name: School

Collection Name: students

Add the following documents in the **details collection** in MongoDB IDE.

```
{
 "_id": 1,
 "name": "Alice",
 "scores": [ { "type": "exam", "score": 90 }, { "type": "quiz", "score": 85 }, { "type": "homework", "score": 80 }
 ]
 }
```



Insert Document

To collection School.students

```
/**
* Paste one or more documents here
*/
{
    "_id": 1,
    "name": "Alice",
    "scores": [
        { "type": "exam", "score": 90 },
        { "type": "quiz", "score": 85 },
        { "type": "homework", "score": 80 }
    ]
}
```

{

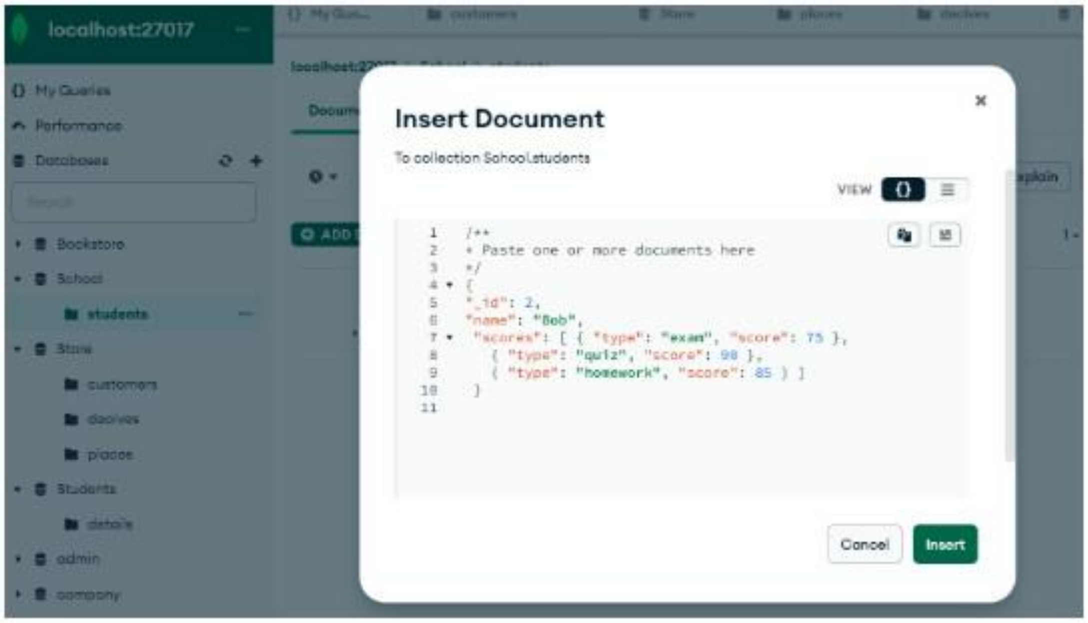
"_id": 2,

"name": "Bob",

"scores": [{ "type": "exam", "score": 75 }, { "type": "quiz", "score": 90 }, { "type": "homework", "score": 85 }

]

}



Insert Document

To collection School.students

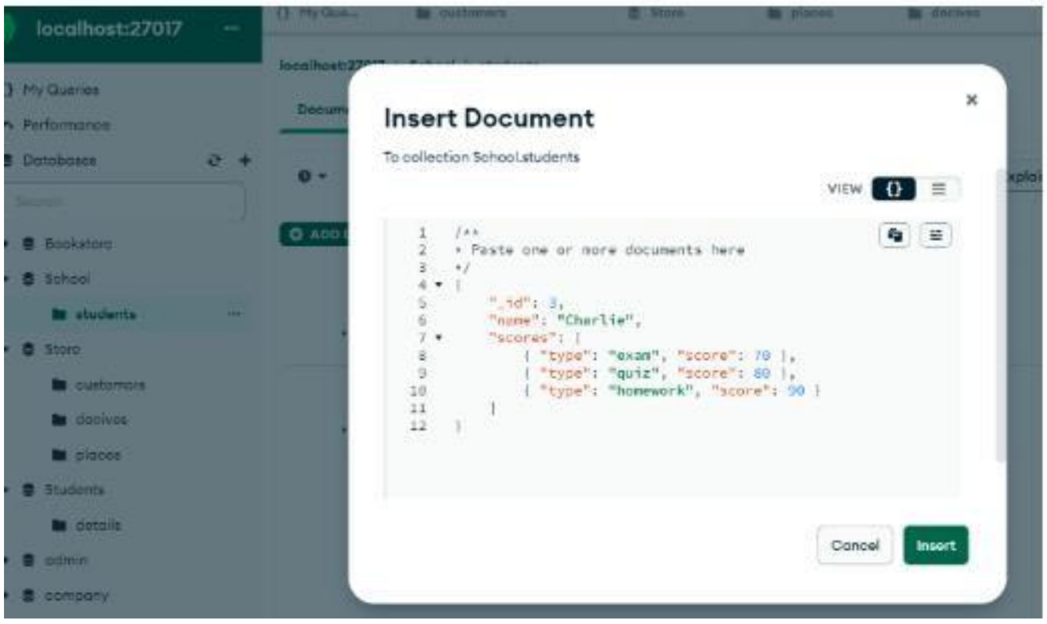
```
/**
* Paste one or more documents here
*/
{
"_id": 2,
"name": "Bob",
 "scores": [ { "type": "exam", "score": 75 },
   { "type": "quiz", "score": 90 },
   { "type": "homework", "score": 85 } ]
 }
```

{

"_id": 3,

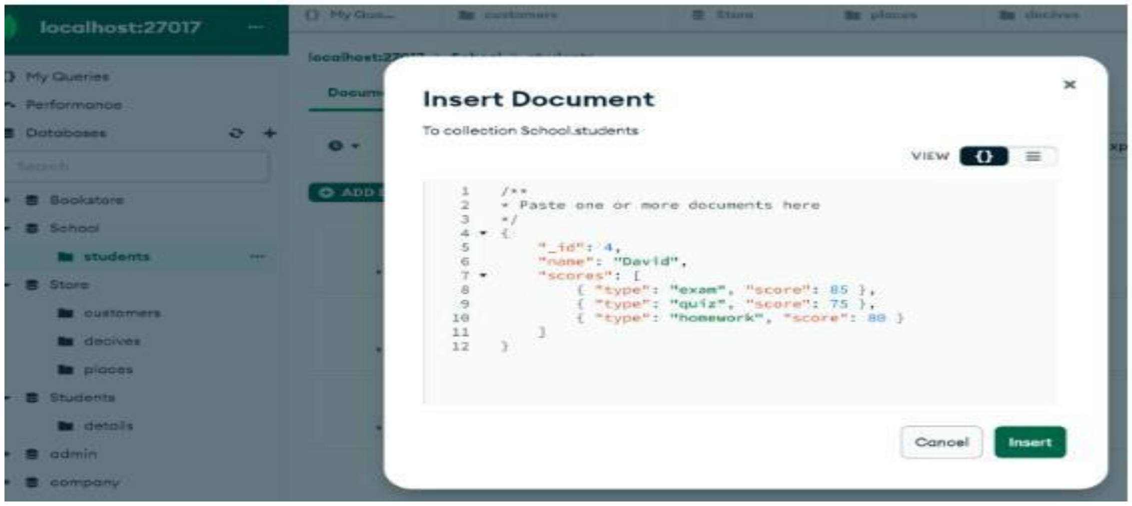
"name": "Charlie",

```
  "scores": [
    { "type": "exam", "score": 70 },
    { "type": "quiz", "score": 80 },
    { "type": "homework", "score": 90 }
  ]
}
```



```
/**
 * Paste one or more documents here
 */
{
    "_id": 3,
    "name": "Charlie",
    "scores": [
        { "type": "exam", "score": 70 },
        { "type": "quiz", "score": 80 },
        { "type": "homework", "score": 90 }
    ]
}
```

```
{
  "_id": 4,
  "name": "David",
  "scores": [
    { "type": "exam", "score": 85 },
    { "type": "quiz", "score": 75 },
    { "type": "homework", "score": 80 }
  ]
}
```

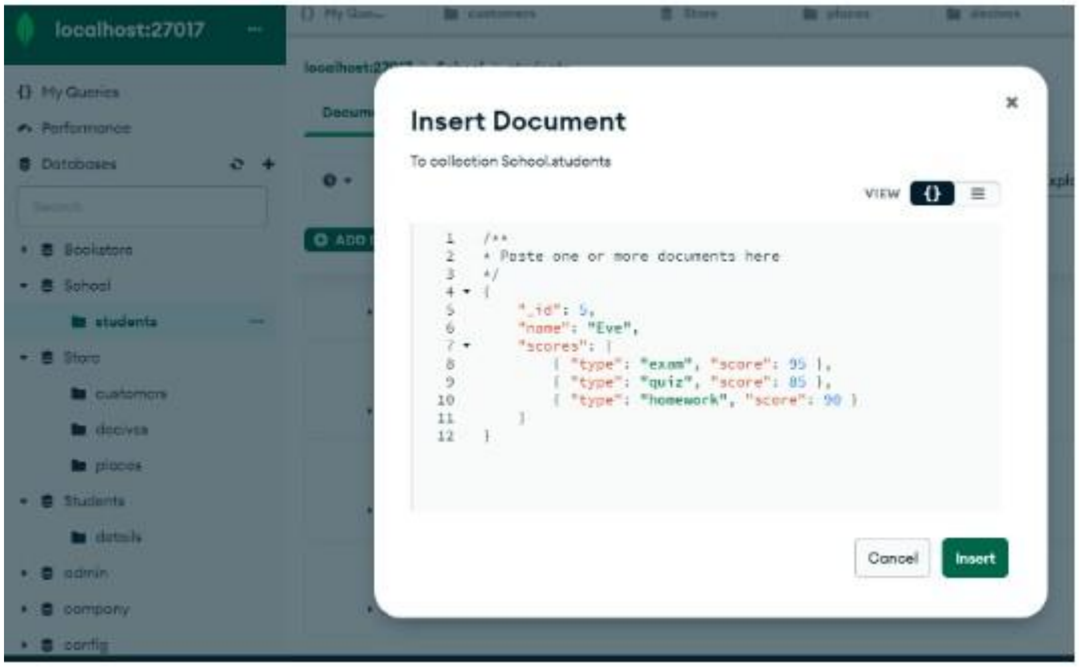


Insert Document

To collection School.students

```
/**
 * Paste one or more documents here
 */
{
    "_id": 4,
    "name": "David",
    "scores": [
        { "type": "exam", "score": 85 },
        { "type": "quiz", "score": 75 },
        { "type": "homework", "score": 88 }
    ]
}
```

```
{
  "_id": 5,
  "name": "Eve",
  "scores": [
    { "type": "exam", "score": 95 },
    { "type": "quiz", "score": 85 },
    { "type": "homework", "score": 90 }
  ]
}
```



Insert Document

To collection School.students

```
/**
 * Paste one or more documents here
 */
{
    "_id": 5,
    "name": "Eve",
    "scores": [
        { "type": "exam", "score": 95 },
        { "type": "quiz", "score": 85 },
        { "type": "homework", "score": 90 }
    ]
}
```

In Mongo DB Shell

>use School

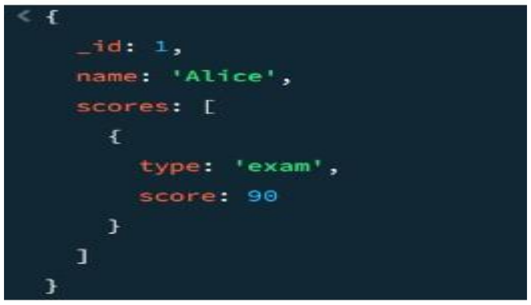
1. $ Operator

The $ operator is used to project a single element from an array that matches a specified condition. For instance, to find the exam score of Alice, you would use:

// To project only the first element in the grades array that is greater than or equal to 85, we can use the following query:

```
db.students.find(
  { "name": "Alice", "scores.type": "exam" },
  { "name": 1, "scores.$": 1 }
)
```

Output:



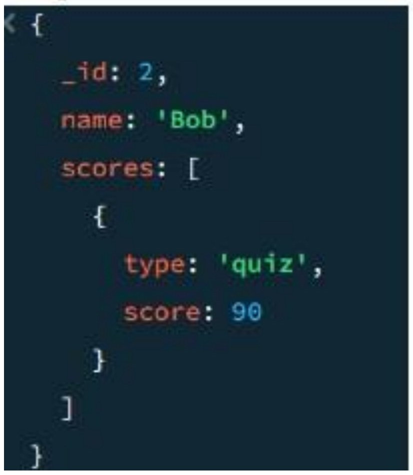
```
< {
    _id: 1,
    name: 'Alice',
    scores: [
      {
        type: 'exam',
        score: 90
      }
    ]
  }
```

2. $elemMatch Operator

The **$elemMatch** operator is used to project the first matching element from an array. To get the quiz score of Bob, you would use:

```
>db.students.find(
    { "name": "Bob" },
    { "name": 1, "scores": { $elemMatch: { "type": "quiz" } } }
)
```

Output:



```
< {
    _id: 2,
    name: 'Bob',
    scores: [
      {
        type: 'quiz',
        score: 90
      }
    ]
  }
```

3. $slice Operator

The $slice operator limits the number of array elements included in the query result. Example

Query to find students with the first two score entries:

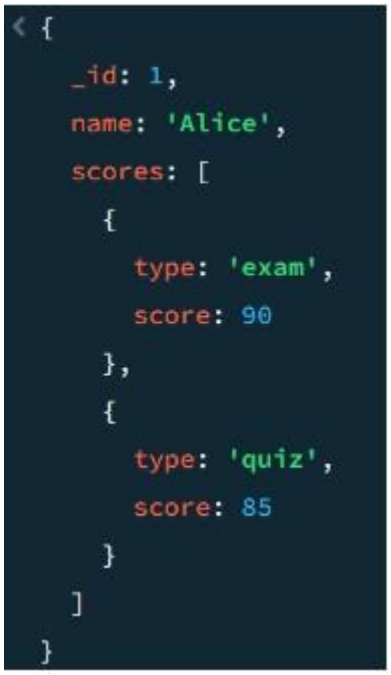
> **db.students.find(**

{},

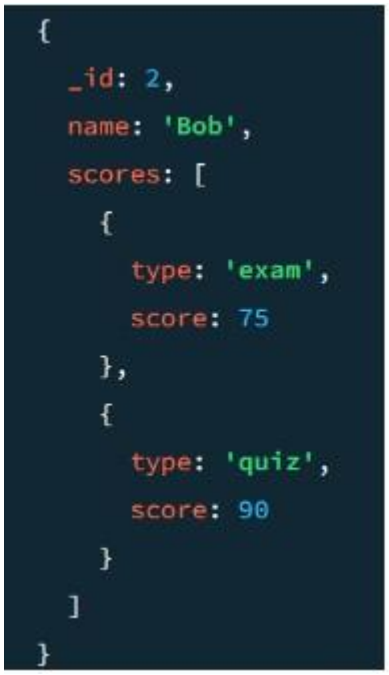
{ "name": 1, "scores": { $slice: 2 } }

)

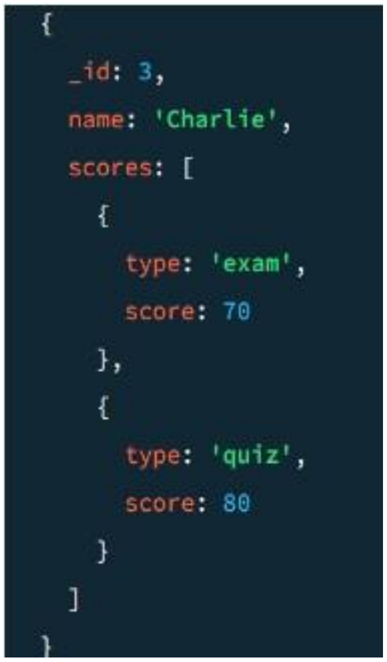
Output:



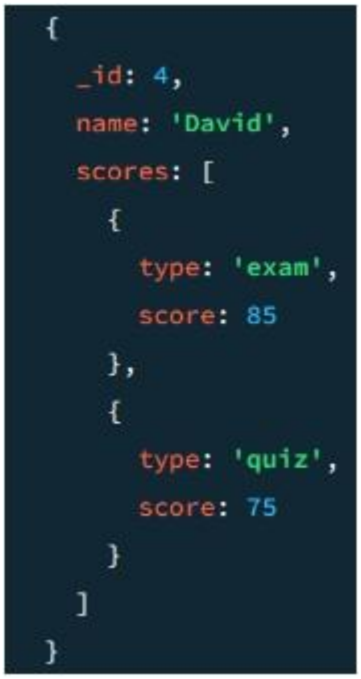
```
< {
    _id: 1,
    name: 'Alice',
    scores: [
      {
        type: 'exam',
        score: 90
      },
      {
        type: 'quiz',
        score: 85
      }
    ]
  }
```



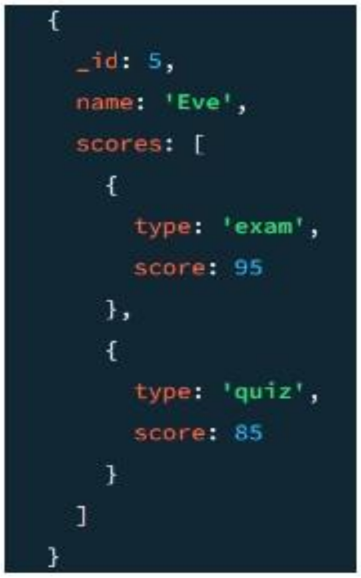
```
{
  _id: 2,
  name: 'Bob',
  scores: [
    {
      type: 'exam',
      score: 75
    },
    {
      type: 'quiz',
      score: 90
    }
  ]
}
```



```
{
  _id: 3,
  name: 'Charlie',
  scores: [
    {
      type: 'exam',
      score: 70
    },
    {
      type: 'quiz',
      score: 80
    }
  ]
}
```



```
{
  _id: 4,
  name: 'David',
  scores: [
    {
      type: 'exam',
      score: 85
    },
    {
      type: 'quiz',
      score: 75
    }
  ]
}
```



```
{
  _id: 5,
  name: 'Eve',
  scores: [
    {
      type: 'exam',
      score: 95
    },
    {
      type: 'quiz',
      score: 85
    }
  ]
}
```

Alternatively, you can use negative values with **$slice** to get elements from the end of the array.

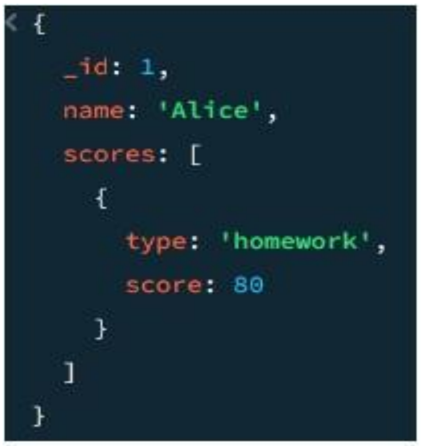
Query to find students with the last score entry: **db.students.find(**

{},

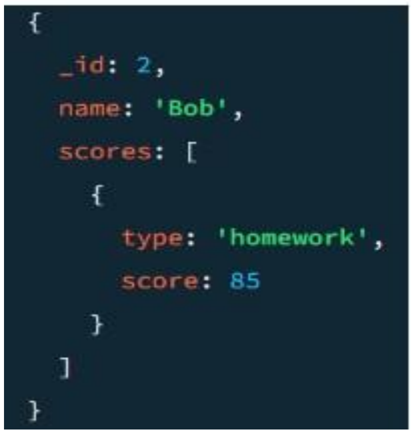
{ "name": 1, "scores": { $slice: -1 } }

)

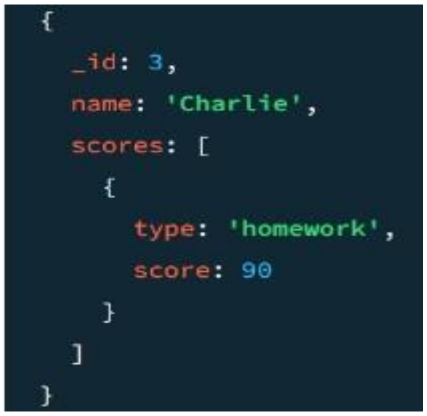
Output:



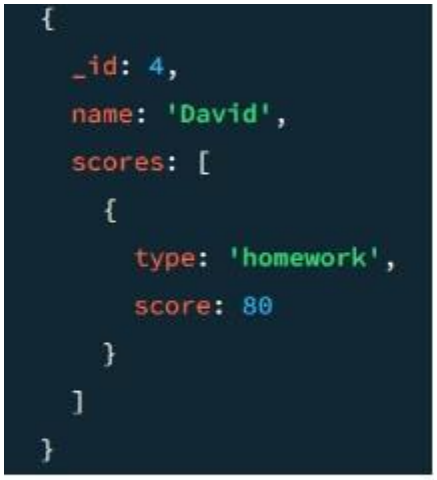
```
< {
    _id: 1,
    name: 'Alice',
    scores: [
      {
        type: 'homework',
        score: 80
      }
    ]
  }
```



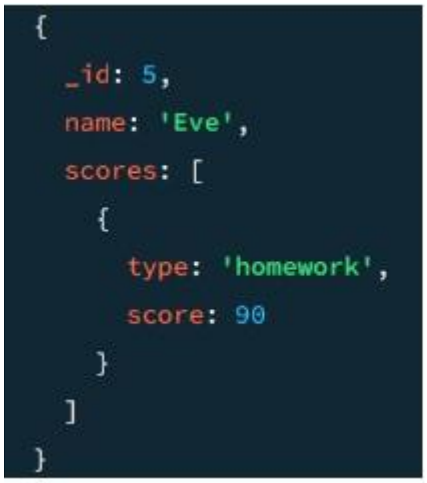
```
{
  _id: 2,
  name: 'Bob',
  scores: [
    {
      type: 'homework',
      score: 85
    }
  ]
}
```



```
{
  _id: 3,
  name: 'Charlie',
  scores: [
    {
      type: 'homework',
      score: 90
    }
  ]
}
```



```
{
  _id: 4,
  name: 'David',
  scores: [
    {
      type: 'homework',
      score: 80
    }
  ]
}
```



```
{
  _id: 5,
  name: 'Eve',
  scores: [
    {
      type: 'homework',
      score: 90
    }
  ]
}
```

Explanation

• **Geospatial Selector**: ○ **$near**: Finds documents near a specified point. Requires a 2dsphere index on the location field. ○ **$geometry**: Specifies the reference point as a GeoJSON object. ○ **$maxDistance**: Limits the distance from the reference point (in meters).

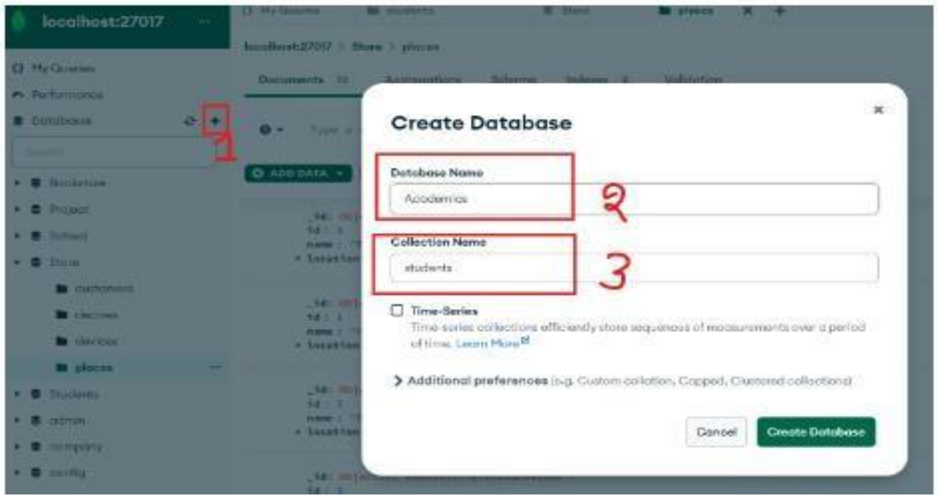
• **Bitwise Selector**: ○ **$bitsAllSet**: Matches documents where all of the given bit positions are 1. ○ **$bitsAnySet**: Matches documents where any of the given bit positions are 1.

By executing these queries, you can filter documents based on geospatial proximity and bitwise conditions.

Program 5

Execute Aggregation operations ($avg, $min, $max, $push, $addToSet etc.). Encourage students to execute several queries to demonstrate various aggregation operators.

Create a database **Academics** and collection **students** in Mongo IDE.



Create Database

Database Name: Academics

Collection Name: students

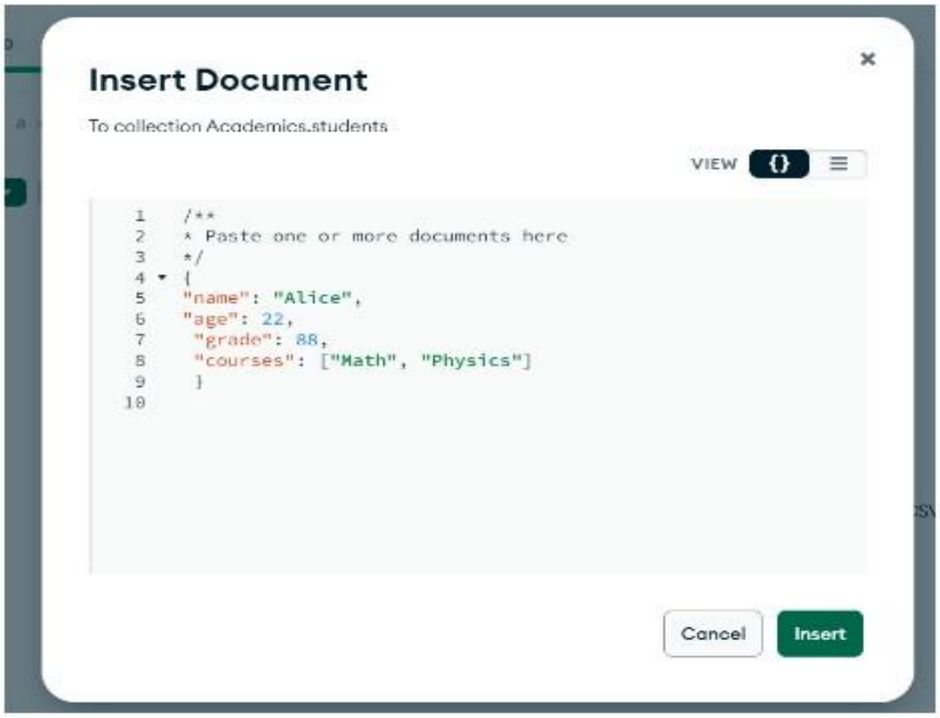
Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

Additional preferences (e.g. Custom collation, Capped, Clustered collections)

Cancel | Create Database

Add the following documents in the **students collection** in MongoDB IDE.

```
{
"name": "Alice",
"age": 22,
 "grade": 88,
 "courses": ["Math", "Physics"]
 }
```



```
/**
* Paste one or more documents here
*/
{
"name": "Alice",
"age": 22,
 "grade": 88,
 "courses": ["Math", "Physics"]
 }
```

{

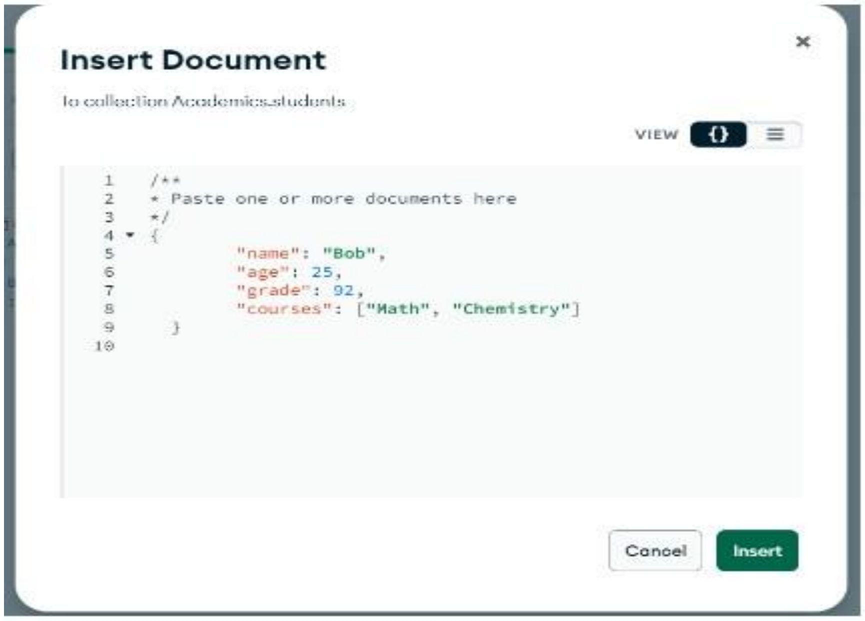
"name": "Bob",

"age": 25,

"grade": 92,

"courses": ["Math", "Chemistry"]

}



```
/**
* Paste one or more documents here
*/
{
        "name": "Bob",
        "age": 25,
        "grade": 92,
        "courses": ["Math", "Chemistry"]
  }
```

{

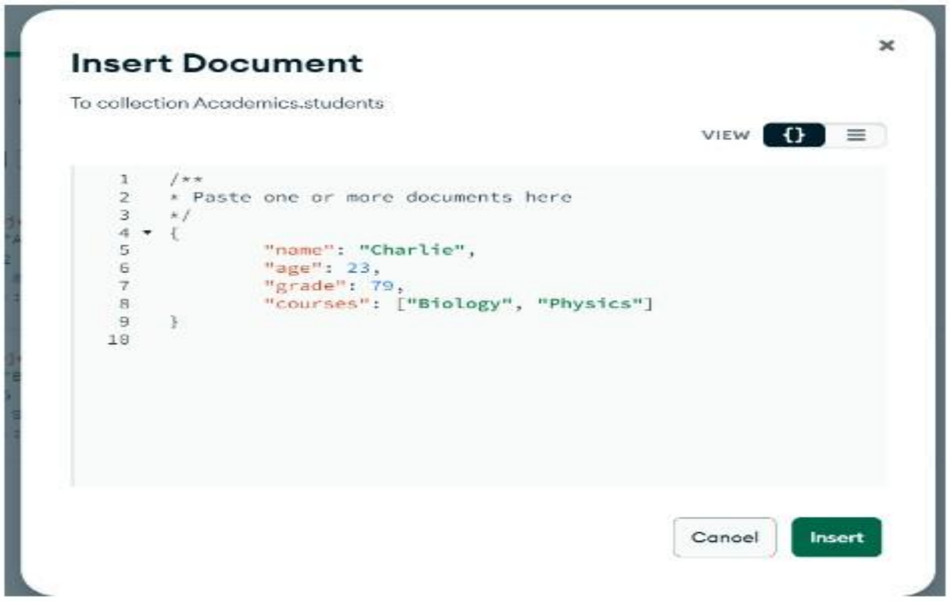
"name": "Charlie",

"age": 23,

"grade": 79,

"courses": ["Biology", "Physics"]

}



Insert Document

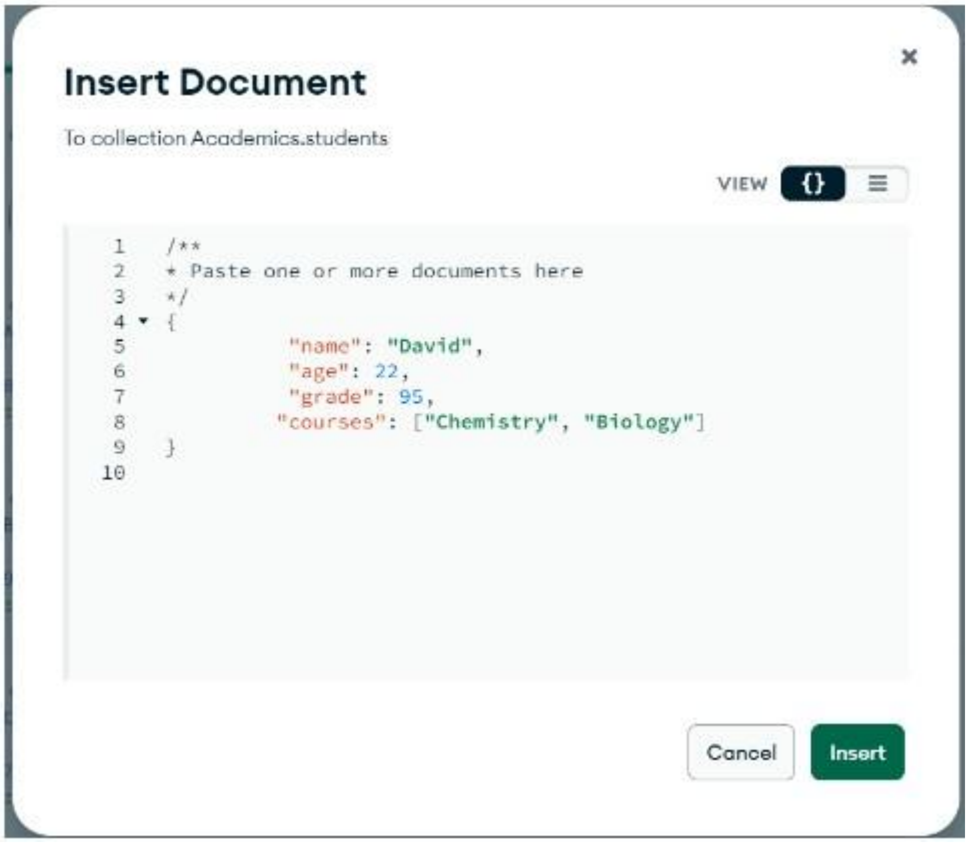
To collection Academics.students

VIEW

```
/**
* Paste one or more documents here
*/
{
        "name": "Charlie",
        "age": 23,
        "grade": 79,
        "courses": ["Biology", "Physics"]
}
```

Cancel Insert

```
{
    "name": "David",
    "age": 22,
    "grade": 95,
   "courses": ["Chemistry", "Biology"]
 }
```

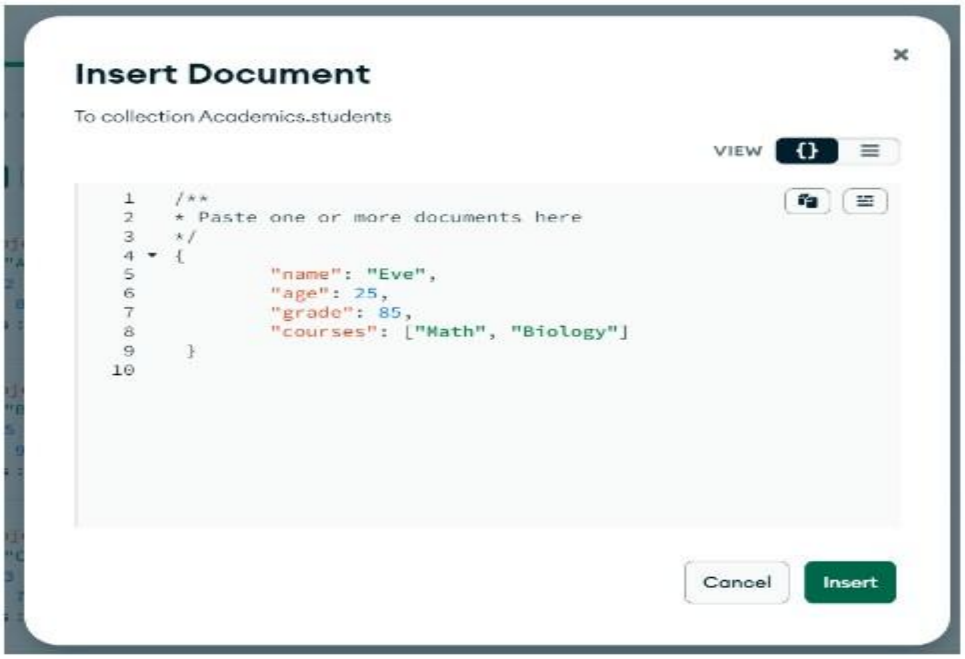


Insert Document

To collection Academics.students

```
/**
* Paste one or more documents here
*/
{
            "name": "David",
            "age": 22,
            "grade": 95,
           "courses": ["Chemistry", "Biology"]
}
```

```
{
      "name": "Eve",
      "age": 25,
      "grade": 85,
      "courses": ["Math", "Biology"]
 }
```



```
/**
* Paste one or more documents here
*/
{
        "name": "Eve",
        "age": 25,
        "grade": 85,
        "courses": ["Math", "Biology"]
 }
```

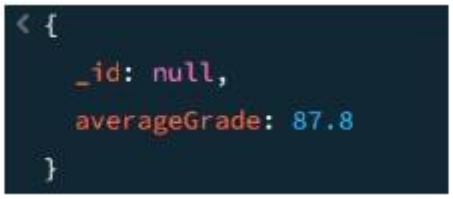
In MongoShell

>use Academics

1. **$avg** - Calculate the average grade of all students

```
> db.students.aggregate([
   {
      $group: {
         _id: null,
         averageGrade: { $avg: "$grade" }
      }
   }
])
```

Output:

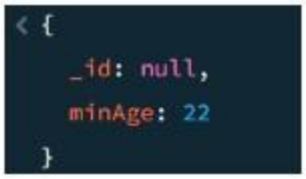


```
< {
    _id: null,
    averageGrade: 87.8
  }
```

2. **$min** - Find the minimum age of students

```
>db.students.aggregate([
   {
      $group: {
         _id: null,
            minAge: { $min: "$age" }
        }
    }
])
```

Output:

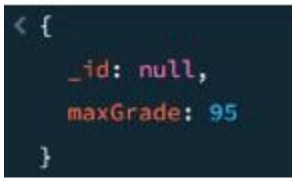


```
< {
    _id: null,
    minAge: 22
  }
```

3. **$max** - Find the maximum grade among students

```
>db.students.aggregate([
    {
        $group: {
            _id: null,
            maxGrade: { $max: "$grade" }
        }
    }  ])
```

Output:

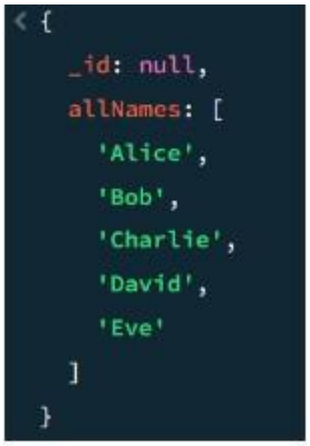


```
< {
    _id: null,
    maxGrade: 95
  }
```

4. **$push** - List all student names in an array **db.students.aggregate([**

```
  {
     $group: {
        _id: null,
        allNames: { $push: "$name" }
     }
  }
])
```

Output:

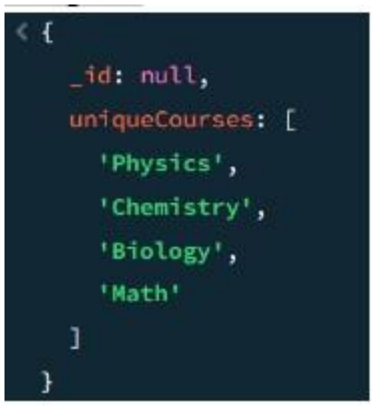


```
{
  _id: null,
  allNames: [
    'Alice',
    'Bob',
    'Charlie',
    'David',
    'Eve'
  ]
}
```

5.$addToSet - List all unique courses taken by students

```
>db.students.aggregate([

   {

      $unwind: "$courses"

   },

   {

      $group: {
         _id: null,

         uniqueCourses: { $addToSet: "$courses" }

      }

   }

])
```

Output:

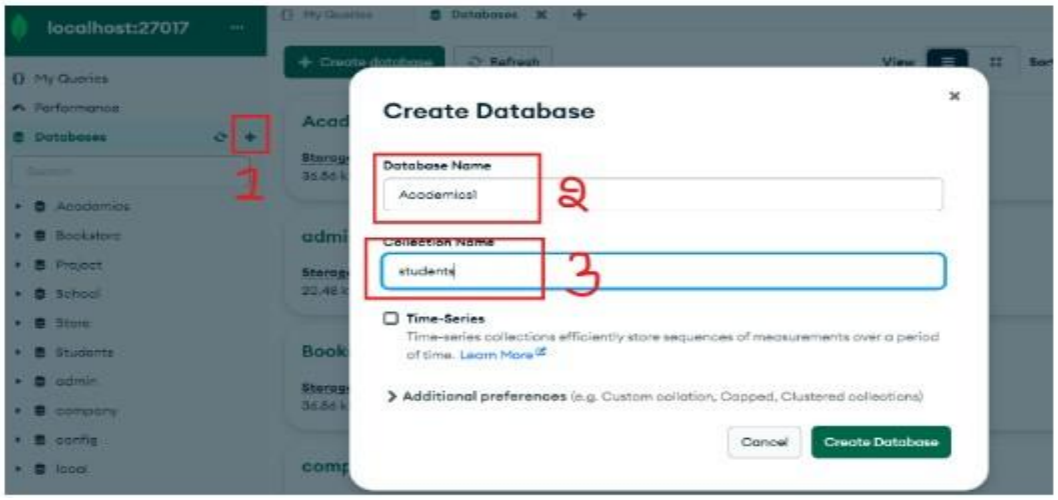


```
{
  _id: null,
  uniqueCourses: [
    'Physics',
    'Chemistry',
    'Biology',
    'Math'
  ]
}
```

Program 6

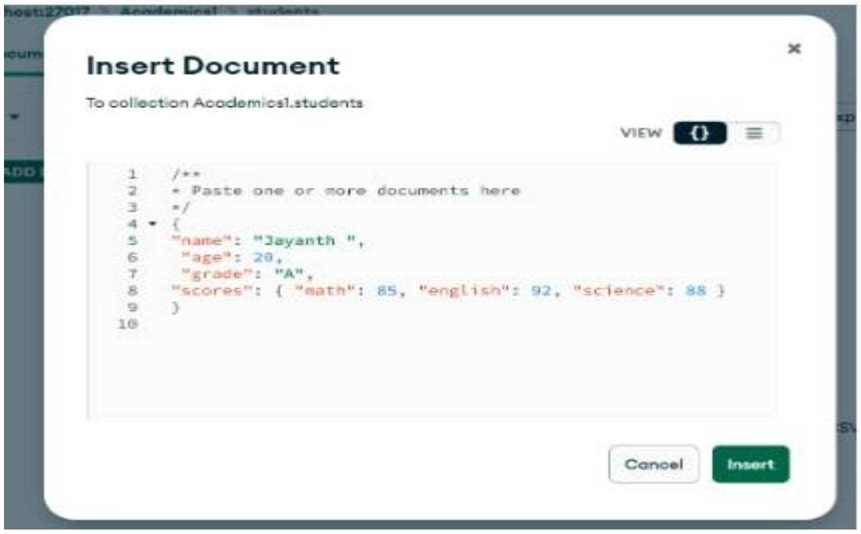
Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project, $skip etc. students encourage to execute several queries to demonstrate various aggregation operators)

Create a database **Academics1** and collection **students** in Mongo IDE.



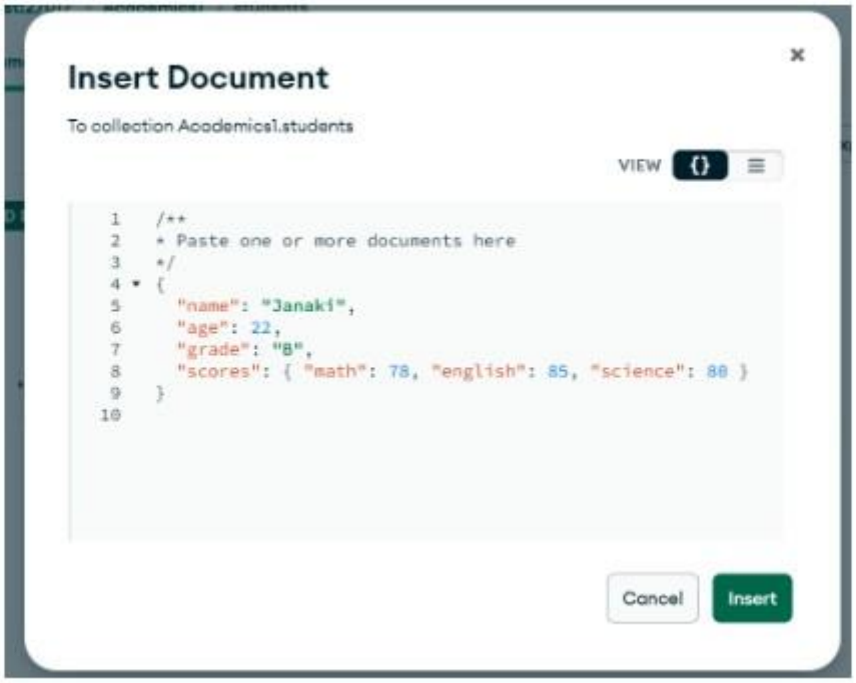
Add the following documents in the **students collection** in MongoDB IDE.

```
{
    "name": "Jayanth ",
            "age": 20,
            "grade": "A",
           "scores": { "math": 85, "english": 92, "science": 88 }
}
```



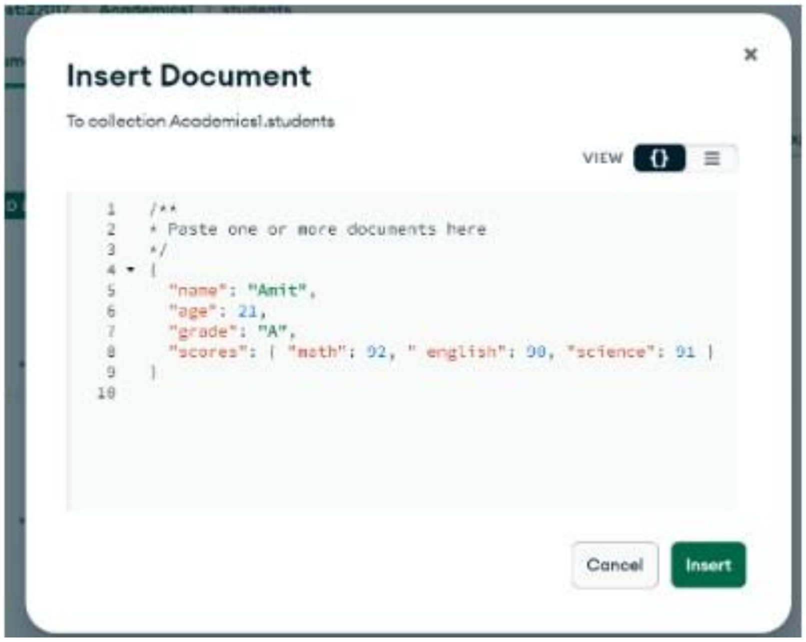
```
/**
* Paste one or more documents here
*/
{
"name": "Jayanth ",
 "age": 20,
 "grade": "A",
"scores": { "math": 85, "english": 92, "science": 88 }
}
```

```
 {
  "name": "Janaki",
        "age": 22,
        "grade": "B",
        "scores": { "math": 78, "english": 85, "science": 80 }
 }
```



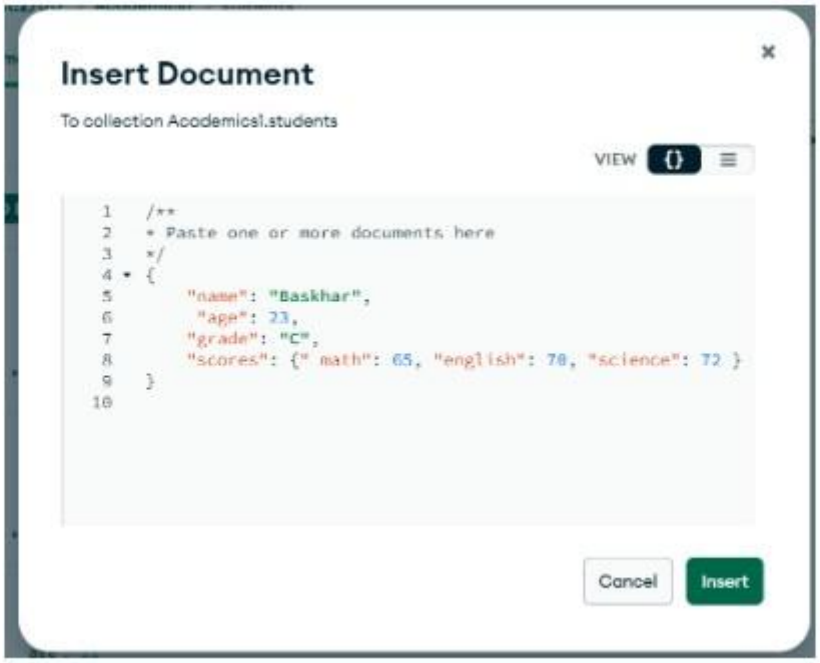
```
/**
* Paste one or more documents here
*/
{
  "name": "Janaki",
  "age": 22,
  "grade": "B",
  "scores": { "math": 78, "english": 85, "science": 80 }
}
```

```
{
  "name": "Amit",
          "age": 21,
"grade": "A",
          "scores": { "math": 92, " english": 90, "science": 91 }
}
```



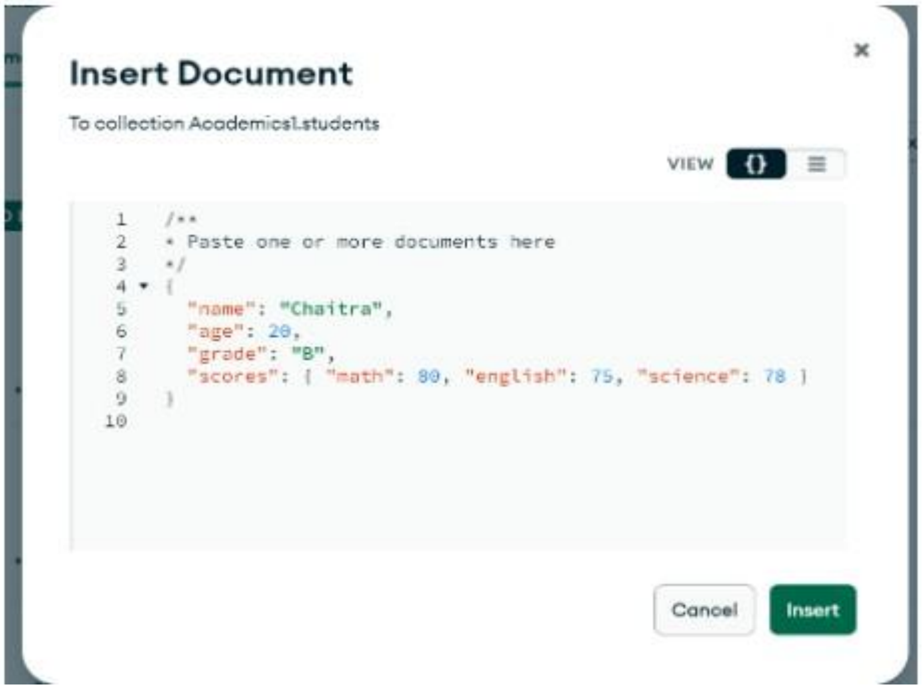
```
/**
* Paste one or more documents here
*/
{
  "name": "Amit",
  "age": 21,
  "grade": "A",
  "scores": { "math": 92, " english": 90, "science": 91 }
}
```

```
{
   "name": "Baskhar",
  "age": 23,
  "grade": "C",
  "scores": {" math": 65, "english": 70, "science": 72 }
}
```



```
/**
* Paste one or more documents here
*/
{
    "name": "Baskhar",
     "age": 23,
    "grade": "C",
    "scores": {" math": 65, "english": 70, "science": 72 }
}
```

```
{
"name": "Chaitra",
"age": 20,
"grade": "B",
"scores": { "math": 80, "english": 75, "science": 78 }
}
```



```
/**
* Paste one or more documents here
*/
{
  "name": "Chaitra",
  "age": 20,
  "grade": "B",
  "scores": { "math": 80, "english": 75, "science": 78 }
}
```

In MongoDB Shell

>use Academics1

Now, let's execute an aggregation pipeline with several stages:

1.$match: Filter students who are 21 years or older.

2.$group: Group by grade and calculate the average age.

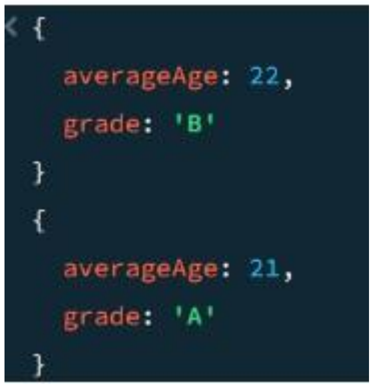
3.$sort: Sort by average age in descending order.

4.$project: Project the grade and average age.

5.$skip: Skip the first result.

```
db.students.aggregate([
  {
         $match: { age: { $gte: 21 } }
  },
   {
    $group: {      _id: "$grade",
averageAge: { $avg: "$age" }
}

},
{
  $sort: { averageAge: -1 }
},
{
  $project: {
_id: 0,      grade:
"$_id",
averageAge: 1
  }
},
{
  $skip: 1
}])
```

Output:



```
< {
    averageAge: 22,
    grade: 'B'
  }
  {
    averageAge: 21,
    grade: 'A'
  }
```

Let's break down each stage:

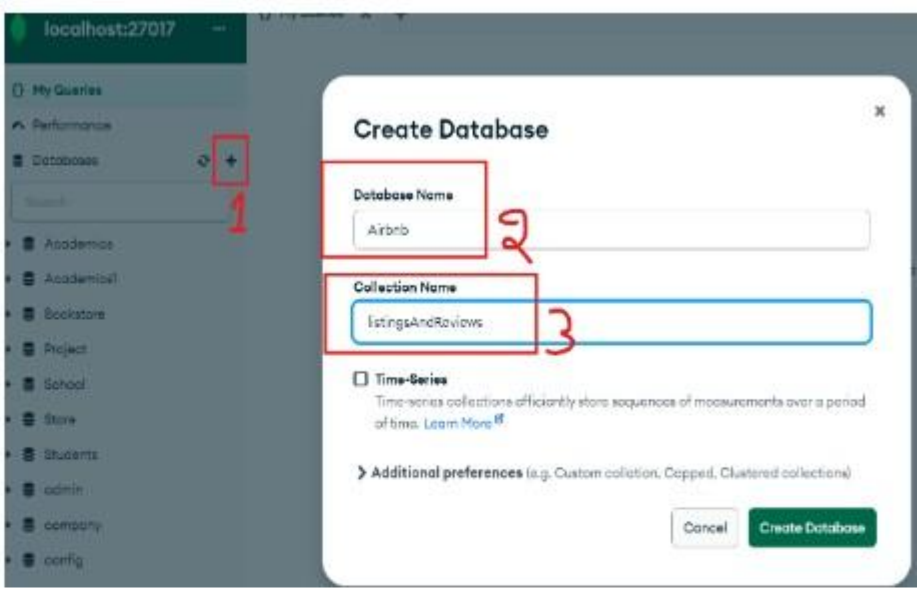
1.**$match**: Filters documents to include only those where age is greater than or equal to 21.
2.**$group**: Groups the documents by grade and computes the average age for each grade.
3.**$sort**: Sorts the resulting documents by average Age in descending order.
4.**$project**: Projects the fields grade and average Age, excluding the _id field.
5.**$skip**: Skips the first document in the sorted results.

When you execute this pipeline, you will get a result that first filters students by age, groups them by grade, calculates the average age, sorts by this average age in descending order, and finally skips the first result.

Program 7

a. Find all listings with listing_url, name, address, host_picture_url in the listings and Reviews collection that have a host with a picture url.

Create a database **Airbnb** and collection **listingsAndReviews** in Mongo IDE.



Create Database

Database Name: Airbnb

Collection Name: listingsAndReviews

Time-series collections efficiently store sequences of measurements over a period of time.

Additional preferences (e.g. Custom collation, Capped, Clustered collections)

Add the following documents in the **listingsAndReviews collection** in MongoDB IDE.

{

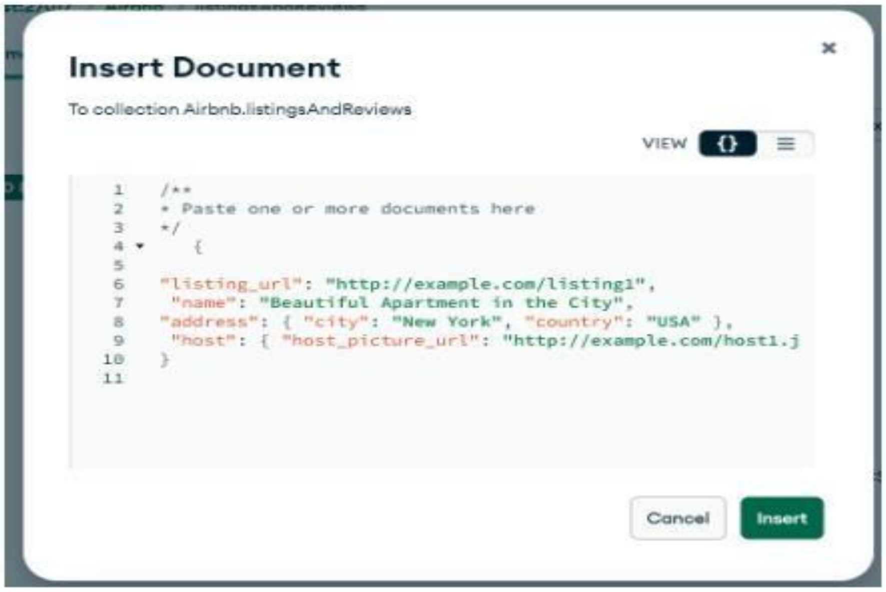
"listing_url": "http://example.com/listing1",

"name": "Beautiful Apartment in the City",

"address": { "city": "New York", "country": "USA" },

"host": { "host_picture_url": "http://example.com/host1.jpg" }

}

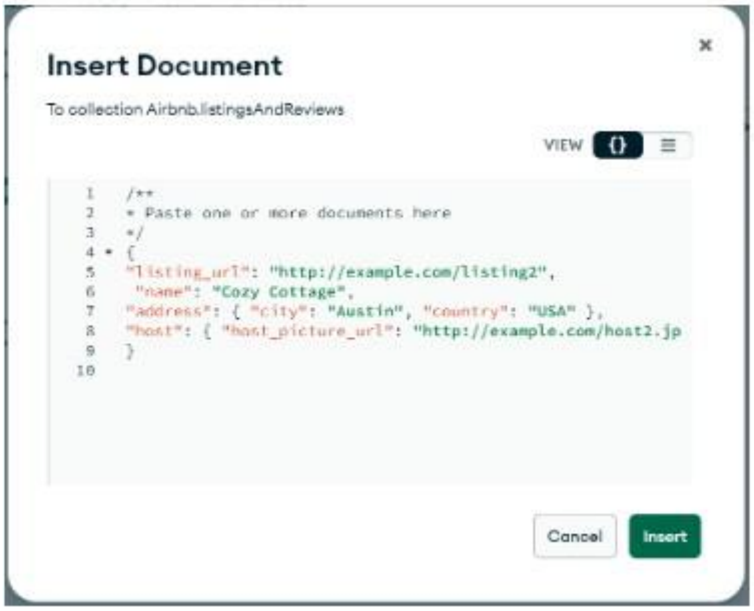


Insert Document

To collection Airbnb.listingsAndReviews

```
/**
* Paste one or more documents here
*/
    {

"listing_url": "http://example.com/listing1",
 "name": "Beautiful Apartment in the City",
"address": { "city": "New York", "country": "USA" },
 "host": { "host_picture_url": "http://example.com/host1.j
}

```

```
{
"listing_url": "http://example.com/listing2",
 "name": "Cozy Cottage",
"address": { "city": "Austin", "country": "USA" },
"host": { "host_picture_url": "http://example.com/host2.jpg" }
}
```

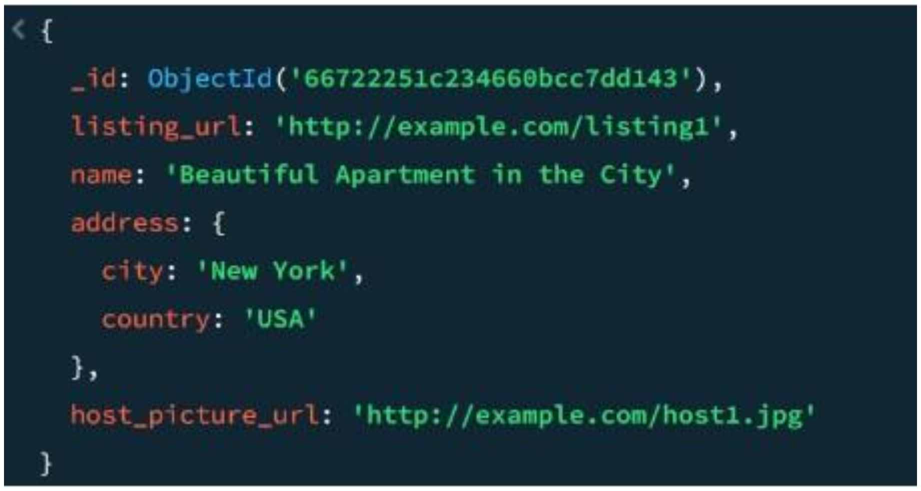


```
/**
* Paste one or more documents here
*/
{
"listing_url": "http://example.com/listing2",
 "name": "Cozy Cottage",
"address": { "city": "Austin", "country": "USA" },
"host": { "host_picture_url": "http://example.com/host2.jp
}
```

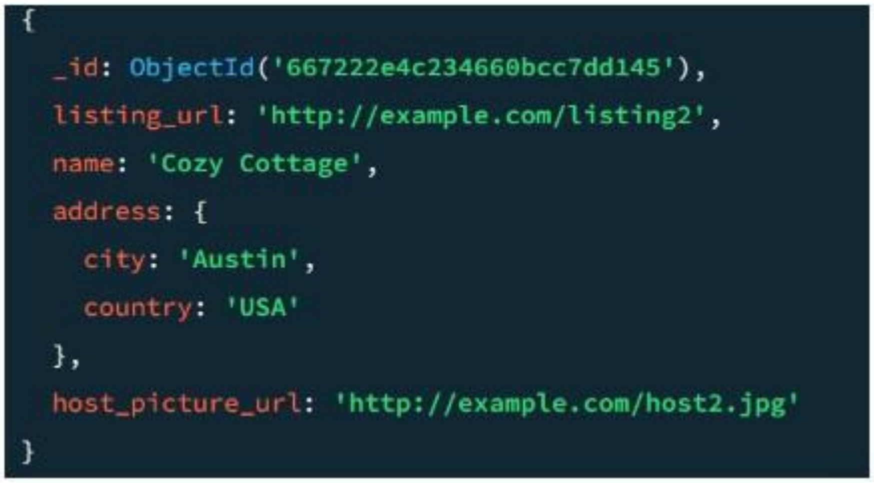
In MongoDB Shell >use Airbnb

```
db.listingsAndReviews.aggregate(
[
  { $match: { "host.host_picture_url": { $exists: true, $ne: null } } },
    {
    $project: {        listing_url: 1,        name: 1,
address:    1,                          host_picture_url:
"$host.host_picture_url" }
  }])
```

Output:



```
< {
    _id: ObjectId('66722251c234660bcc7dd143'),
    listing_url: 'http://example.com/listing1',
    name: 'Beautiful Apartment in the City',
    address: {
      city: 'New York',
      country: 'USA'
    },
    host_picture_url: 'http://example.com/host1.jpg'
  }
```

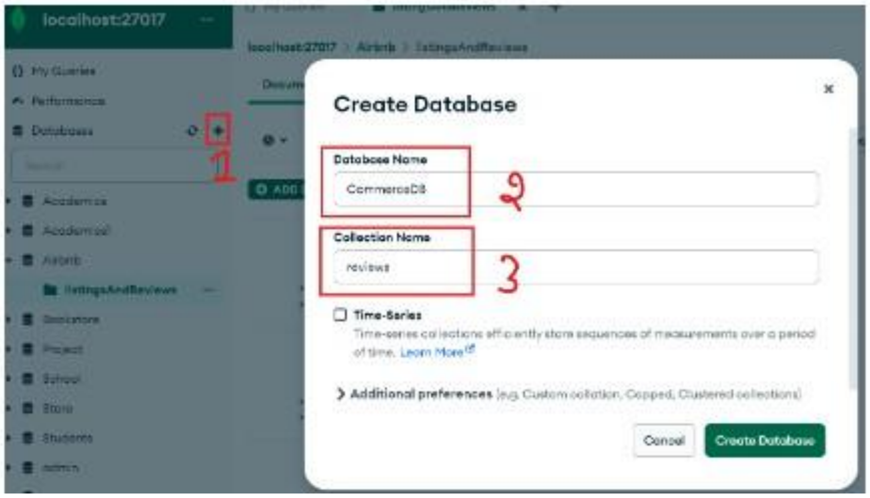


```
{
  _id: ObjectId('667222e4c234660bcc7dd145'),
  listing_url: 'http://example.com/listing2',
  name: 'Cozy Cottage',
  address: {
    city: 'Austin',
    country: 'USA'
  },
  host_picture_url: 'http://example.com/host2.jpg'
}
```

---

b. Using E-commerce collection write a query to display reviews summary.

Create a database **CommerceDB** and collection **reviews** in Mongo IDE.



Create Database

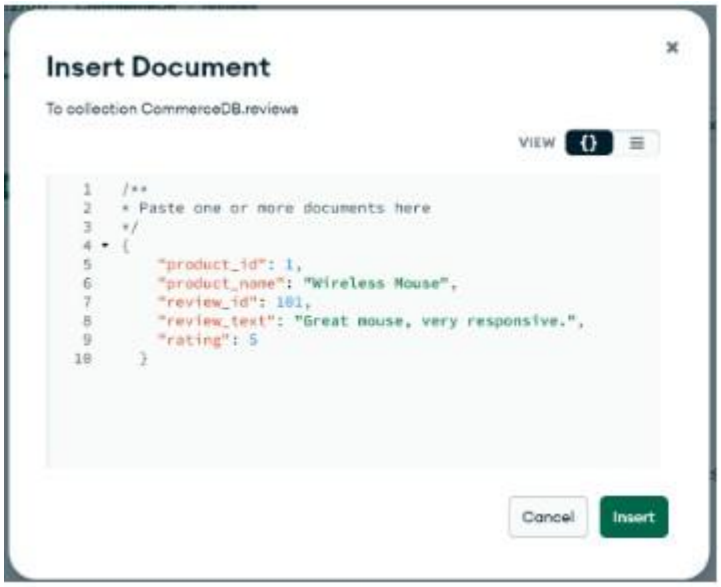
Database Name: CommerceDB

Collection Name: reviews

Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

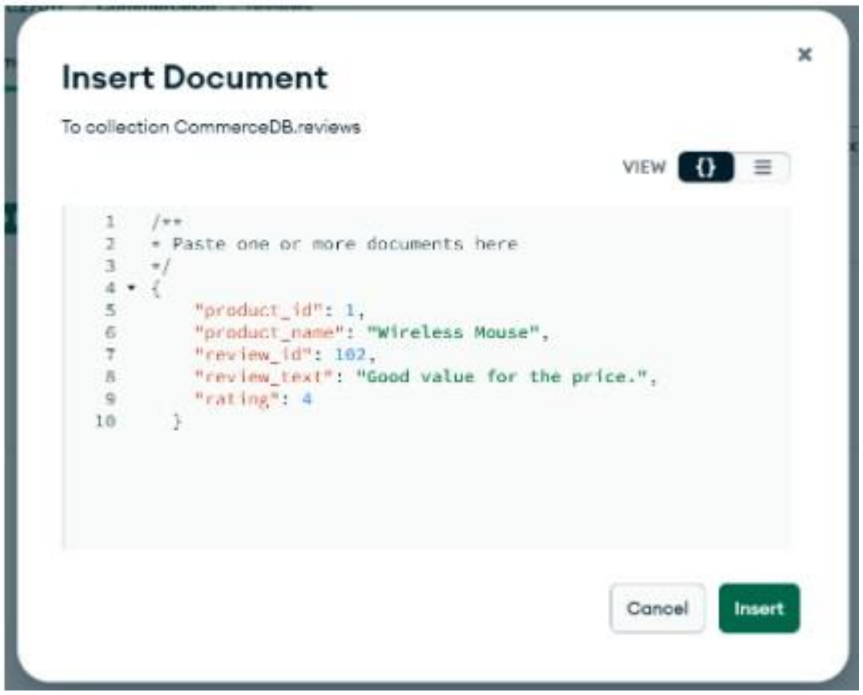
Add the following documents in the **reviews collection** in MongoDB IDE.

```
{
  "product_id": 1,
  "product_name": "Wireless Mouse",
  "review_id": 101,
  "review_text": "Great mouse, very responsive.",
  "rating": 5
}
```



```
/**
* Paste one or more documents here
*/
{
    "product_id": 1,
    "product_name": "Wireless Mouse",
    "review_id": 101,
    "review_text": "Great mouse, very responsive.",
    "rating": 5
  }
```

```
{
  "product_id": 1,
 "product_name": "Wireless Mouse",
  "review_id": 102,
  "review_text": "Good value for the price.",
  "rating": 4
}
```

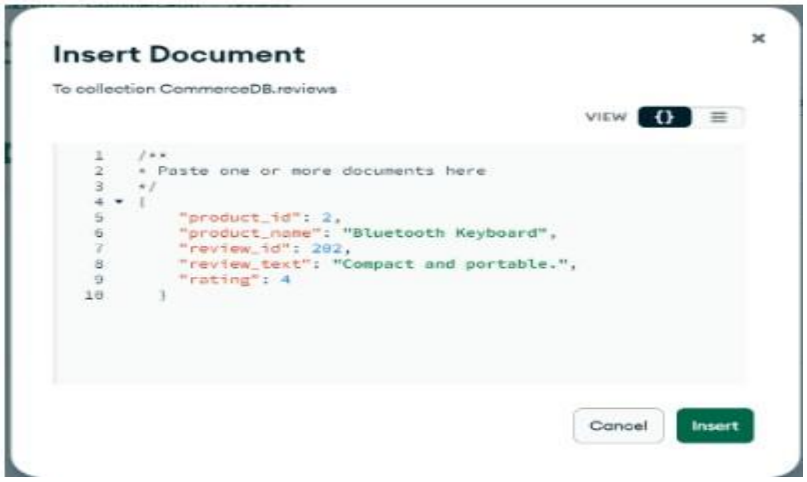


Insert Document

To collection CommerceDB.reviews

```
/**
 * Paste one or more documents here
 */
{
    "product_id": 1,
    "product_name": "Wireless Mouse",
    "review_id": 102,
    "review_text": "Good value for the price.",
    "rating": 4
  }
```

```
{
   "product_id": 2,
   "product_name": "Bluetooth Keyboard",
   "review_id": 202,
   "review_text": "Compact and portable.",
   "rating": 4
 }
```

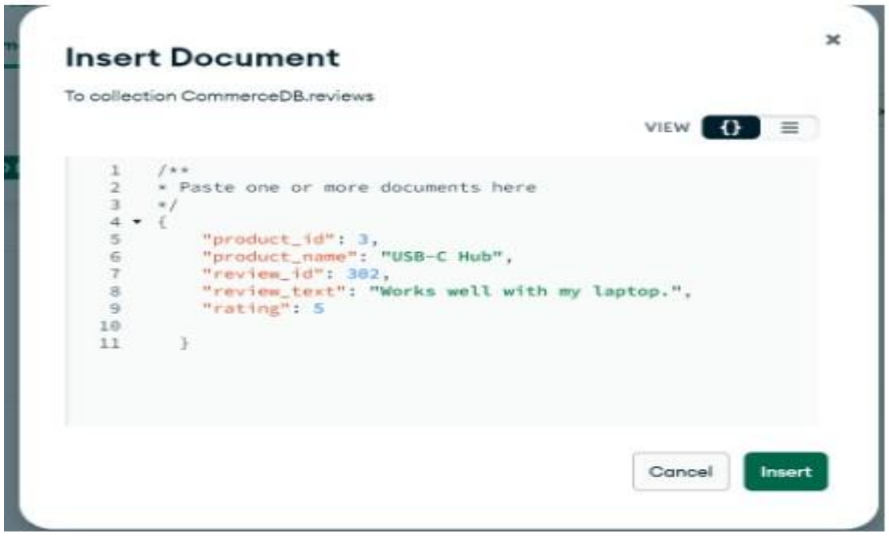


Insert Document

To collection CommerceDB.reviews

```
/**
 * Paste one or more documents here
 */
{
    "product_id": 2,
    "product_name": "Bluetooth Keyboard",
    "review_id": 202,
    "review_text": "Compact and portable.",
    "rating": 4
  }
```

```
{
   "product_id": 3,
   "product_name": "USB-C Hub",
   "review_id": 302,
   "review_text": "Works well with my laptop.",
   "rating": 5
}
```



```
/**
* Paste one or more documents here
*/
{
     "product_id": 3,
     "product_name": "USB-C Hub",
     "review_id": 302,
     "review_text": "Works well with my laptop.",
     "rating": 5

   }
```

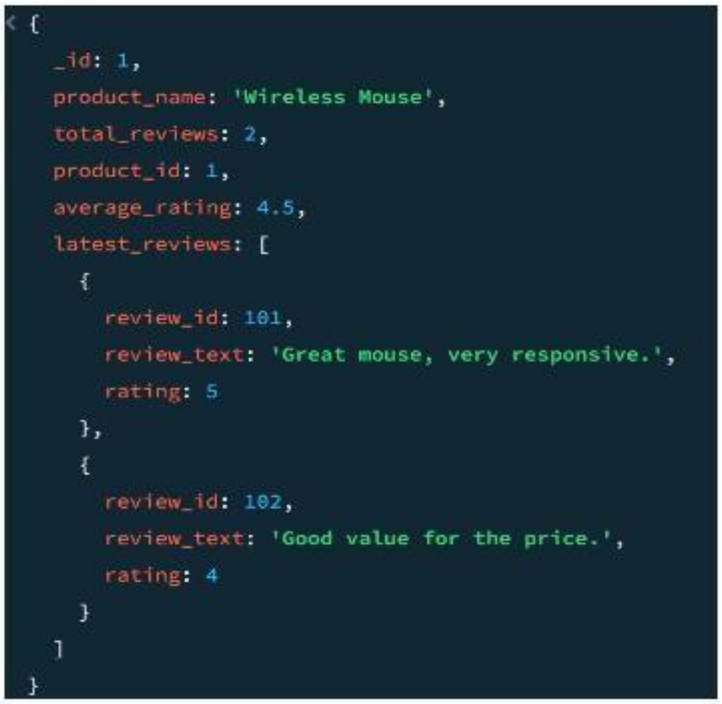
In MongoShell

>use CommerceDB

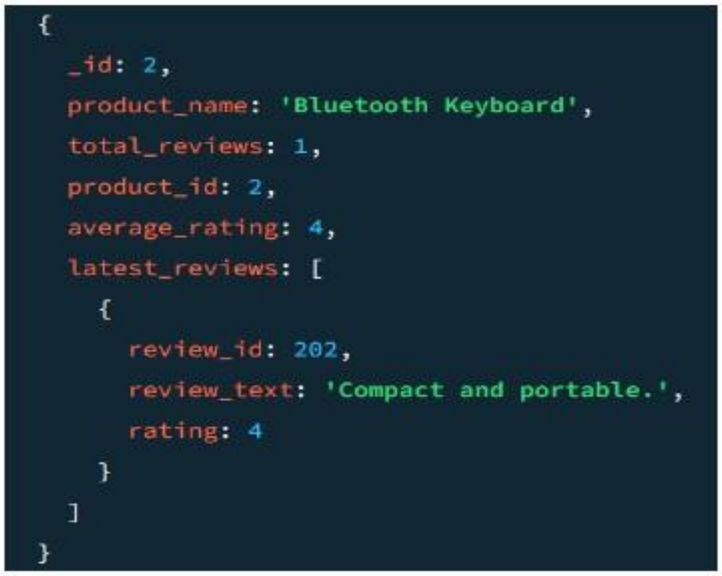
Query to Display Reviews Summary

```
db.reviews.aggregate([
  {
    $group: {        _id: "$product_id",
product_name: { $first: "$product_name" },
total_reviews: { $sum: 1 },       average_rating:
{ $avg: "$rating" },       latest_reviews: {
$push: {           review_id: "$review_id",
        review_text: "$review_text",
rating: "$rating",
        }
      }
    }
  },
  {
    $project: {       product_id: "$_id",
product_name: 1,       total_reviews: 1,
average_rating: "$average_rating",
latest_reviews: {
        $slice: ["$latest_reviews", -3] // Adjust the number of latest reviews as needed
      }
    }
  },
  {
    $sort: { total_reviews: -1 } // Sort by total reviews in descending order
  }
])
```

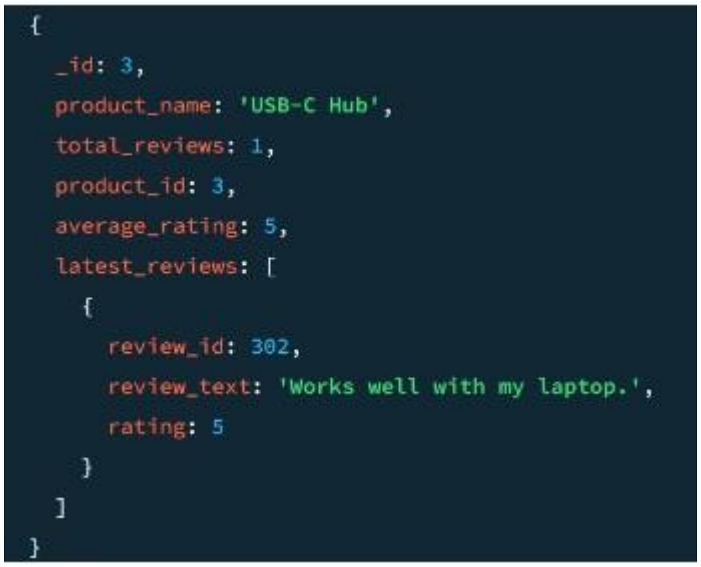
Output:



```
< {
    _id: 1,
    product_name: 'Wireless Mouse',
    total_reviews: 2,
    product_id: 1,
    average_rating: 4.5,
    latest_reviews: [
      {
        review_id: 101,
        review_text: 'Great mouse, very responsive.',
        rating: 5
      },
      {
        review_id: 102,
        review_text: 'Good value for the price.',
        rating: 4
      }
    ]
  }
```



```
  {
    _id: 2,
    product_name: 'Bluetooth Keyboard',
    total_reviews: 1,
    product_id: 2,
    average_rating: 4,
    latest_reviews: [
      {
        review_id: 202,
        review_text: 'Compact and portable.',
        rating: 4
      }
    ]
  }
```

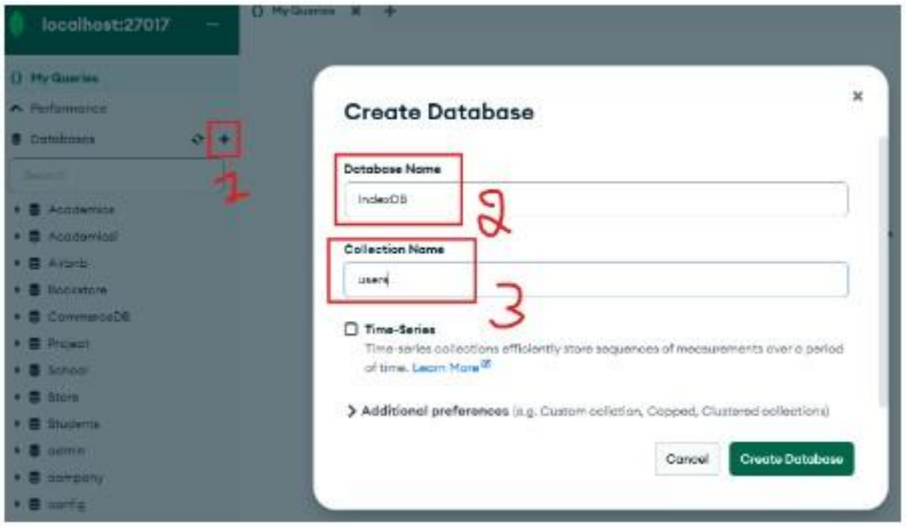


```
{
  _id: 3,
  product_name: 'USB-C Hub',
  total_reviews: 1,
  product_id: 3,
  average_rating: 5,
  latest_reviews: [
    {
      review_id: 302,
      review_text: 'Works well with my laptop.',
      rating: 5
    }
  ]
}
```

Program 8

a. Demonstrate creation of different types of indexes on collection (unique, sparse, compound and multikey indexes)

Create a database **IndexDB** and collection **users** in Mongo IDE.



Create Database

Database Name: IndexDB

Collection Name: users

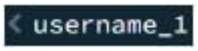
Add the following documents in the **users collection** in MongoDB Shell.

```
db.users.insert({
  username: "John",   age:
  30,   city:"Chennai",
  "interests":["music","garden"],
  Description:["good","avg","excellent"],
  "hashedField": "hashedValue1",
  "location": { "type": "Point", "coordinates": [ -73.97, 40.77 ] },
  "createdAt": ISODate("2023-01-01T00:00:00Z") })
```

1.Unique index:

A unique index ensures that the indexed field(s) do not have duplicate values

db.users.createIndex({ "username": 1 }, { unique: true }) **Output:**



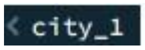
```
< username_1
```

2. Sparse Index:

A sparse index only includes documents that have the indexed field.

db.users.createIndex({ "city": 1 }, { sparse: true })

Output:

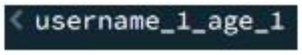


```
city_1
```

3. Compound Index:

A compound index includes multiple fields within a single index.

db.users.createIndex({ “username”: 1, “age”: 1 }) Output:

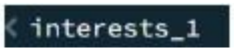


```
username_1_age_1
```

4. Multikey Index

A multikey index is created on an array field, indexing each value of the array.

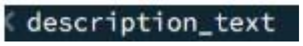
db.users.createIndex({ "interests": 1 })



```
interests_1
```

Assuming interests is an array field in the user documents.

db.users.createIndex({ description: "text" }) Output:



```
description_text
```

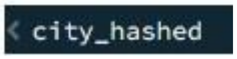
MongoDB provides text indexes to support text search queries on string content. text indexes can include any field whose value is a string or an array of string elements. Remember that you can have only one text index per collection so after creating one if you create another text index, you will get an error.

Hashed Index: Indexes where MongoDB hashes the index keys to create a more even distribution of keys.

db. users.createIndex({ city: "hashed" })

Hashed indexes are beneficial for sharding collections in MongoDB.

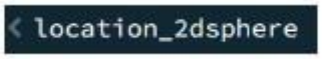
They distribute data across shards based on the hash value of the indexed field, improving query performance for filtering based on that field **Output:**



```
city_hashed
```

Geo-spatial Index: Indexes used for geo-spatial queries.

db. users.createIndex({ location: "2dsphere" }) Output:

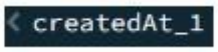


```
location_2dsphere
```

TTL (Time-To-Live) Index: Indexes that automatically expire documents after a certain amount of time.

db. users.createIndex({ createdAt: 1 }, { expireAfterSeconds: 3600 })

Output:



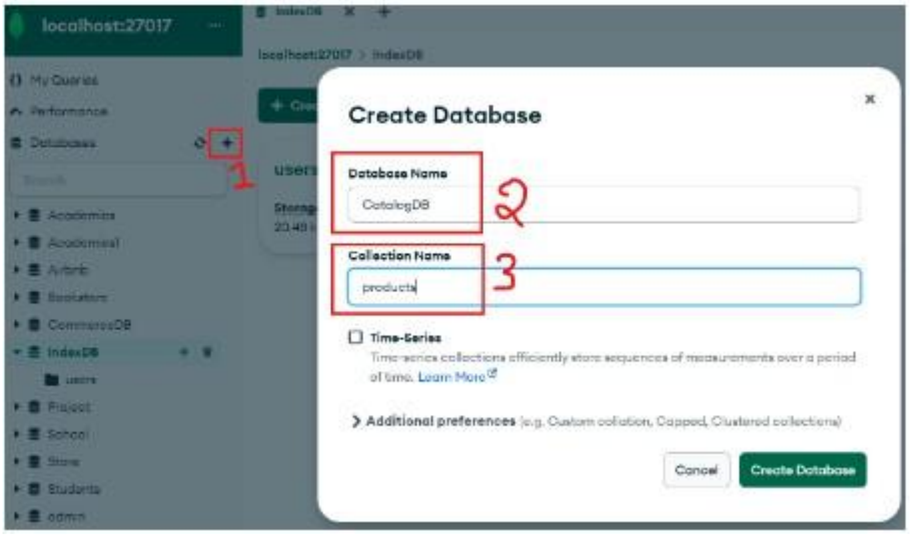
```
< createdAt_1
```

Program 9

a. Develop a query to demonstrate Text search using catalog data collection for a given word

b. Develop queries to illustrate excluding documents with certain words and phrases.

Create a database **CatalogDB** and collection **products** in Mongo IDE.


localhost:27017
My Queries
Performance
Databases
1
Create Database
Database Name
CatalogDB
2
Collection Name
products
3
Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More
Additional preferences (e.g. Custom collation, Capped, Clustered collections)
Cancel
Create Database

Add the following documents in the **collection products** in MongoDB IDE.

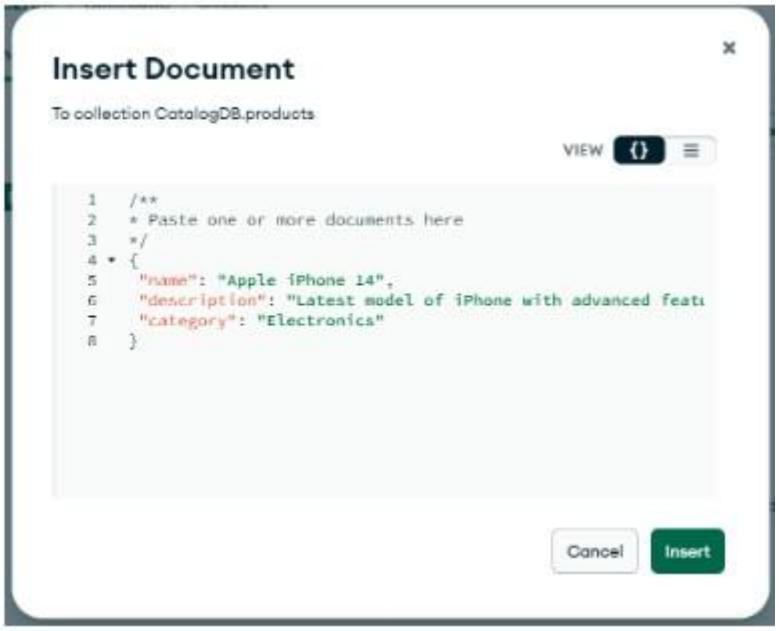
{

"name": "Apple iPhone 14",

"description": "Latest model of iPhone with advanced features",

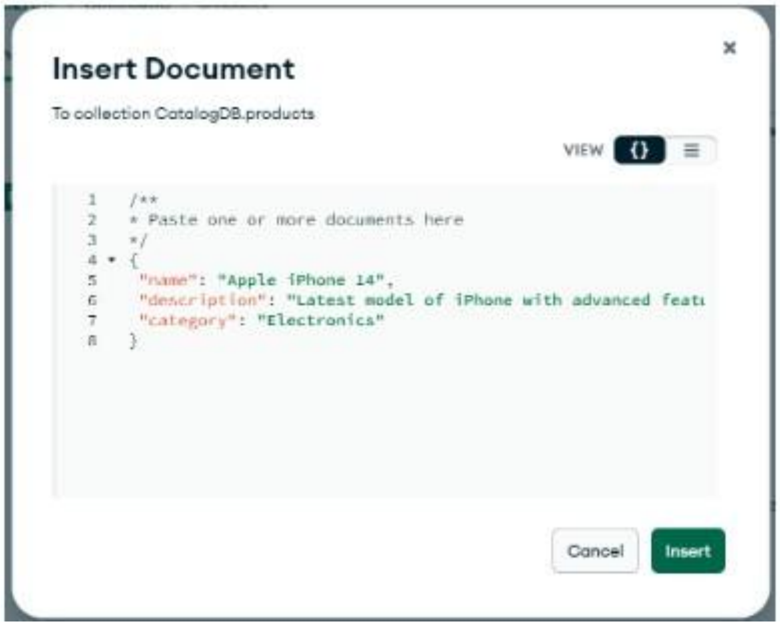
"category": "Electronics"

}



```
/**
* Paste one or more documents here
*/
{
 "name": "Apple iPhone 14",
 "description": "Latest model of iPhone with advanced featu
 "category": "Electronics"
}
```

```
{
    "name": "Samsung Galaxy S21",
  "description": "Newest Samsung smartphone with great camera",
 "category": "Electronics"
}
```



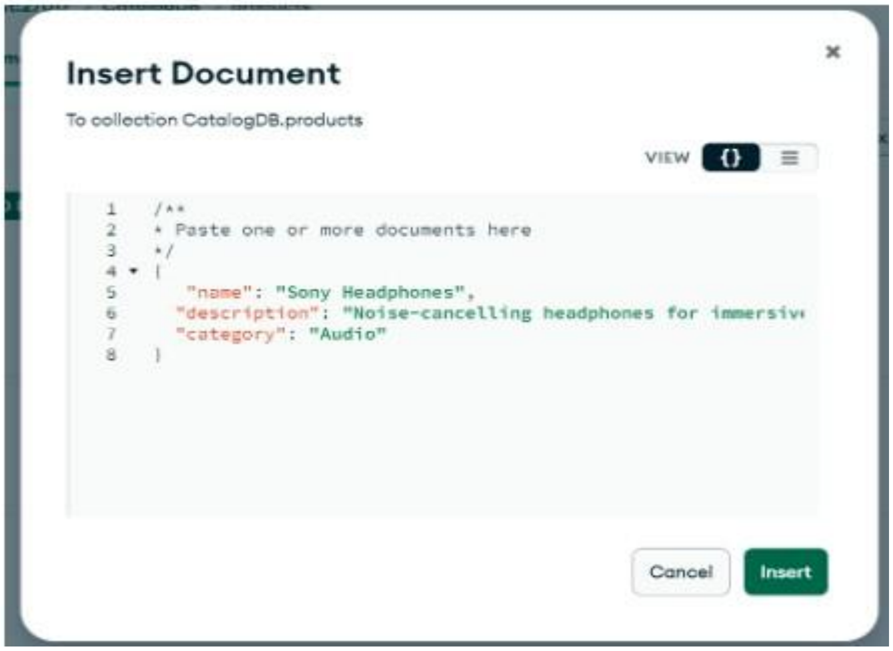
Insert Document

To collection CatalogDB.products

VIEW

```
/**
* Paste one or more documents here
*/
{
 "name": "Apple iPhone 14",
 "description": "Latest model of iPhone with advanced featu
 "category": "Electronics"
}
```

```
{
  "name": "Sony Headphones",
 "description": "Noise-cancelling headphones for immersive sound",
 "category": "Audio"
}
```



```
/**
* Paste one or more documents here
*/
{
  "name": "Sony Headphones",
  "description": "Noise-cancelling headphones for immersiv
  "category": "Audio"
}
```

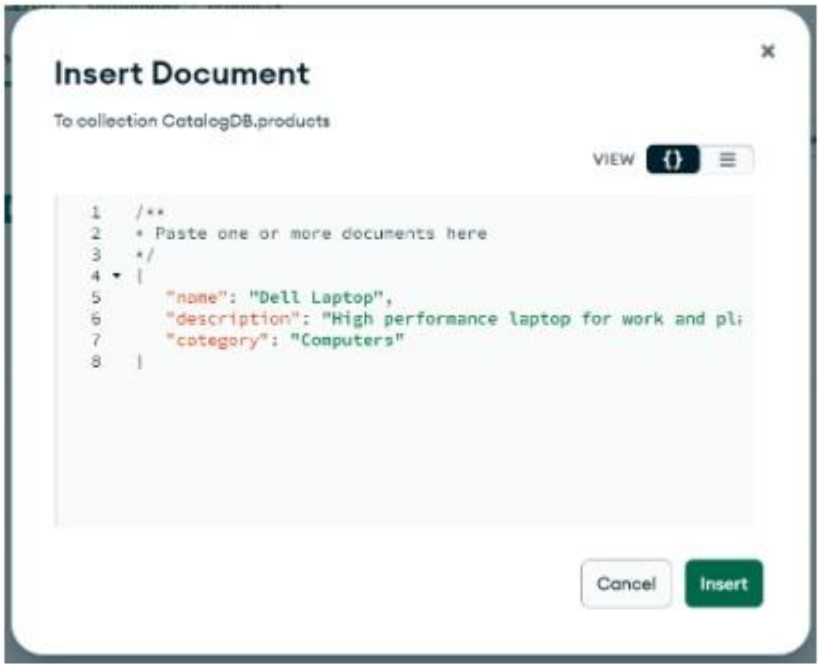
{

"name": "Dell Laptop",

"description": "High performance laptop for work and play",

"category": "Computers"

}



```
/**
* Paste one or more documents here
*/
{
  "name": "Dell Laptop",
  "description": "High performance laptop for work and pl
  "category": "Computers"
}
```

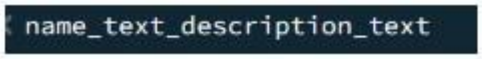
In MongoShell

>use CatalogDB

a. 1. Create a Text Index

To enable text search, you need to create a text index on the fields you want to search. Here, we'll create a text index on the name and description fields:

db.products.createIndex({ name: "text", description: "text" }) Output:



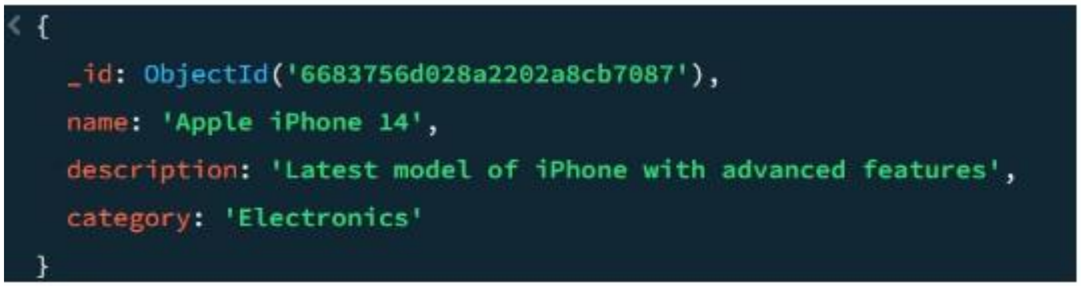
```
name_text_description_text
```

2. **Perform a Text Search**

Now, let's perform a text search. Suppose you want to search for products related to the word "latest":

db.products.find({ $text: { $search: "latest" } })

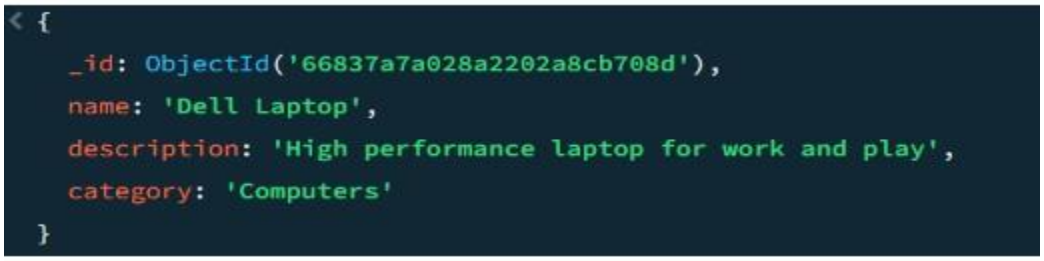
Output:



```
< {
    _id: ObjectId('6683756d028a2202a8cb7087'),
    name: 'Apple iPhone 14',
    description: 'Latest model of iPhone with advanced features',
    category: 'Electronics'
  }
```

db.products.find({ $text: { $search: "High performance" } })

Output:

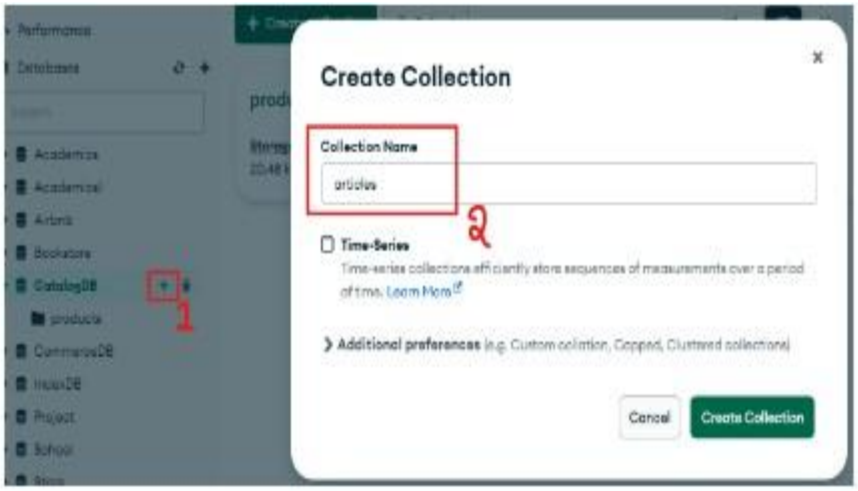


```
< {
    _id: ObjectId('66837a7a028a2202a8cb708d'),
    name: 'Dell Laptop',
    description: 'High performance laptop for work and play',
    category: 'Computers'
  }
```

b. Develop queries to illustrate excluding documents with certain words and phrases.

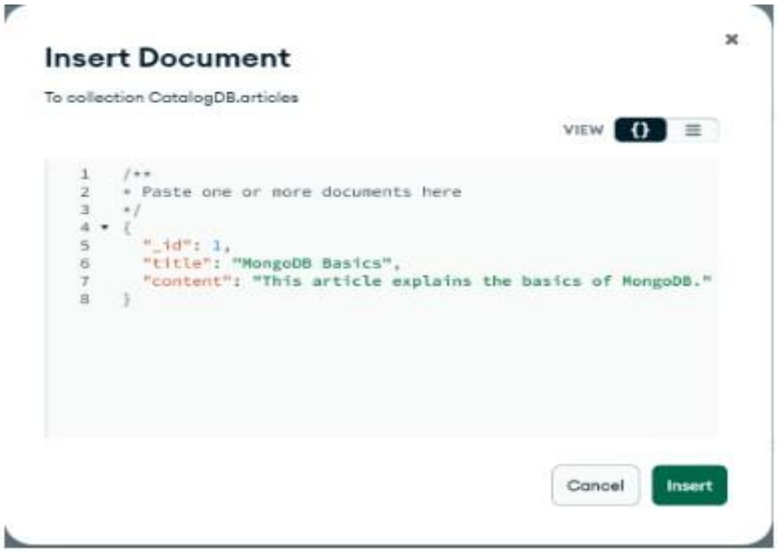
In MongoDB, you can use the $not operator combined with the $regex operator to exclude documents that contain certain words or phrases. Below are some examples of queries to illustrate this.

Add the following documents in the **collection articles** in MongoDB IDE.


Performance
Databases
Academics
Academics2
Airbnb
Bookstore
CatalogDB
products
CommerceDB
InsiteDB
Project
School
1
2
Create Collection
Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More
Additional preferences (e.g. Custom collation, Capped, Clustered collections)
Cancel
Create Collection

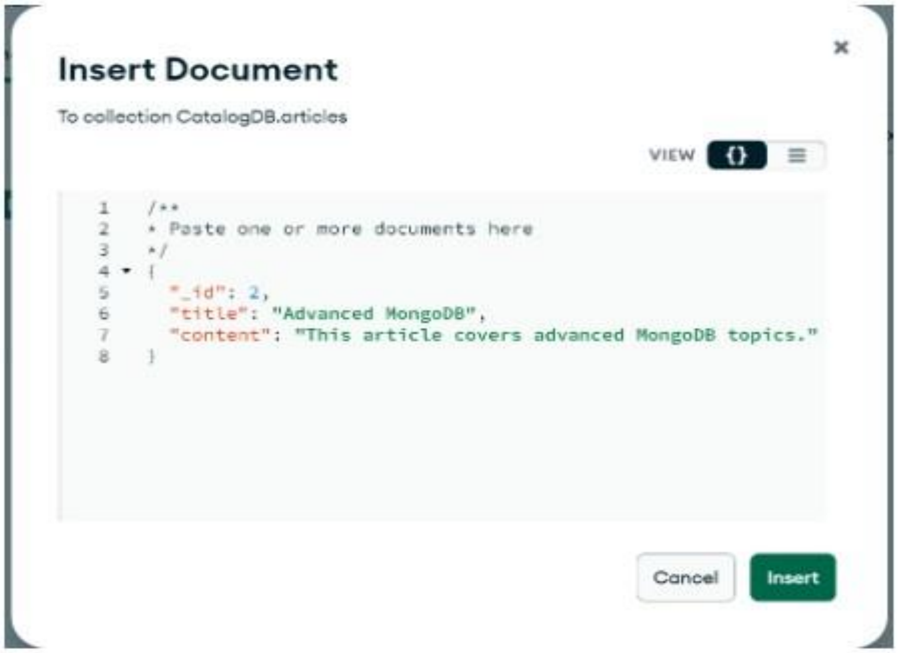
Collection Name: articles

{
  "_id": 1,
  "title": "MongoDB Basics",
  "content": "This article explains the basics of MongoDB."
}



```
/**
 * Paste one or more documents here
 */
{
  "_id": 1,
  "title": "MongoDB Basics",
  "content": "This article explains the basics of MongoDB."
}
```

{
  "_id": 2,
  "title": "Advanced MongoDB",
  "content": "This article covers advanced MongoDB topics."
}



```
/**
* Paste one or more documents here
*/
{
  "_id": 2,
  "title": "Advanced MongoDB",
  "content": "This article covers advanced MongoDB topics."
}
```

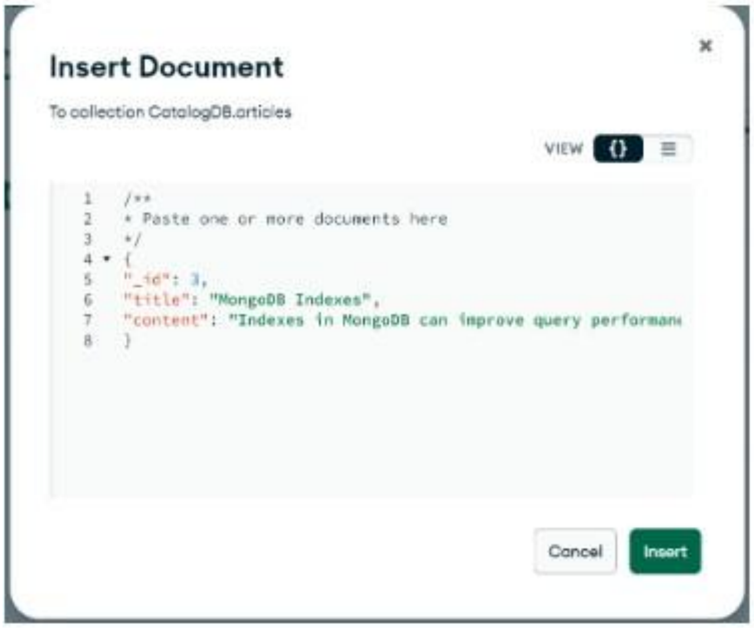
{

"_id": 3,

"title": "MongoDB Indexes",

"content": "Indexes in MongoDB can improve query performance."

}

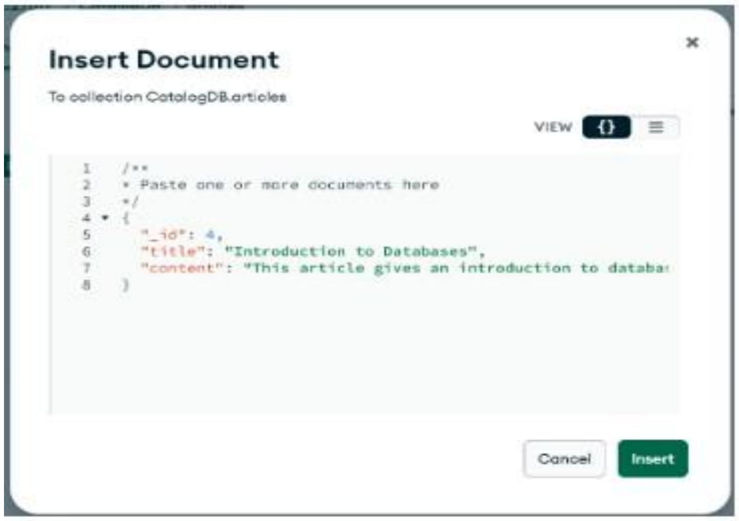


```
/**
* Paste one or more documents here
*/
{
"_id": 3,
"title": "MongoDB Indexes",
"content": "Indexes in MongoDB can improve query performan
}
```

{

"_id": 4,

```
  "title": "Introduction to Databases",
  "content": "This article gives an introduction to databases in general."
}
```



Insert Document

To collection CatalogDB.articles

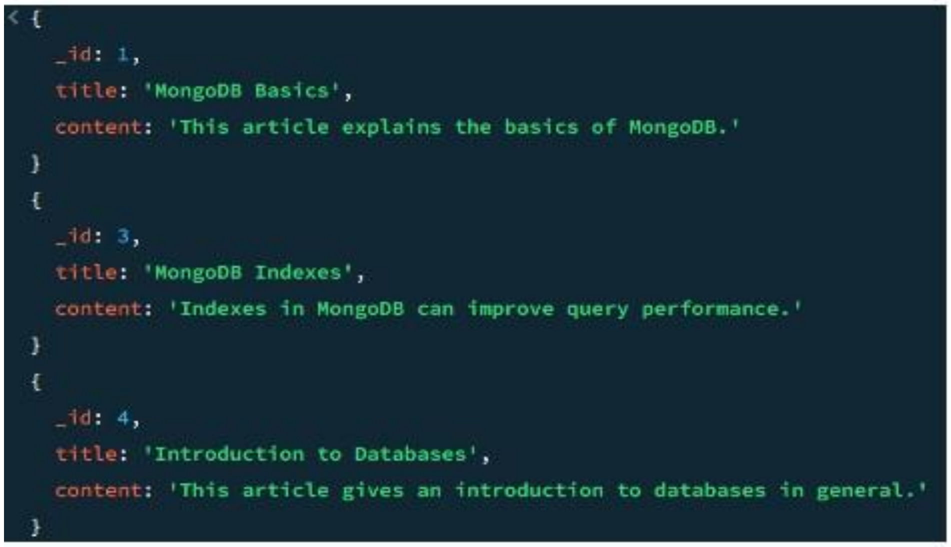
```
/**
 * Paste one or more documents here
 */
{
  "_id": 4,
  "title": "Introduction to Databases",
  "content": "This article gives an introduction to databa
}
```

1. Exclude Documents Containing a Specific Word

To exclude documents that contain the word "advanced" in the ‘content’ field:

```
db.articles.find({
"content": {
        $not: /advanced/
   }
})
```

Output:



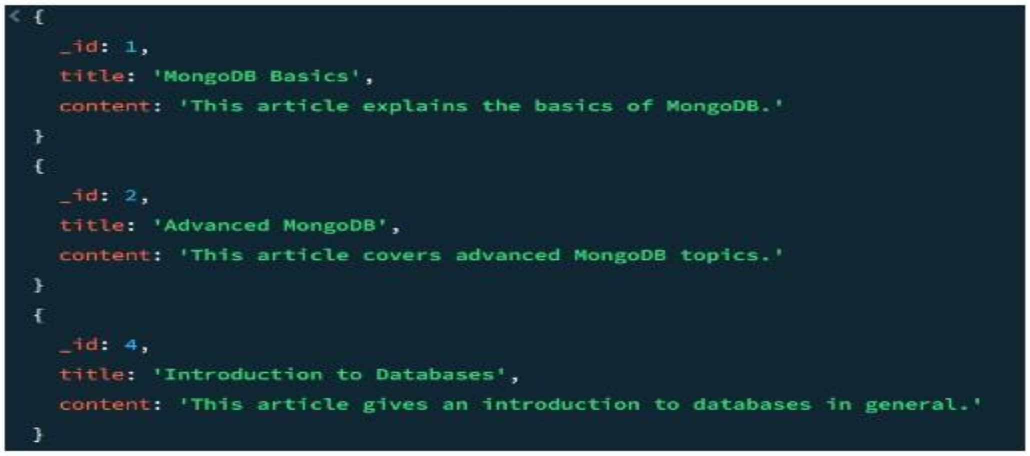
```
< {
    _id: 1,
    title: 'MongoDB Basics',
    content: 'This article explains the basics of MongoDB.'
  }
  {
    _id: 3,
    title: 'MongoDB Indexes',
    content: 'Indexes in MongoDB can improve query performance.'
  }
  {
    _id: 4,
    title: 'Introduction to Databases',
    content: 'This article gives an introduction to databases in general.'
  }
```

2. Exclude Documents Containing Any of Multiple Words

To exclude documents that contain either "improve" or "performance" in the ‘content’ field:

```
db.articles.find({

   "content": {
      $not: /(improve|performance)/
   }
})
```

Output:



```
< {
    _id: 1,
    title: 'MongoDB Basics',
    content: 'This article explains the basics of MongoDB.'
  }
  {
    _id: 2,
    title: 'Advanced MongoDB',
    content: 'This article covers advanced MongoDB topics.'
  }
  {
    _id: 4,
    title: 'Introduction to Databases',
    content: 'This article gives an introduction to databases in general.'
  }
```

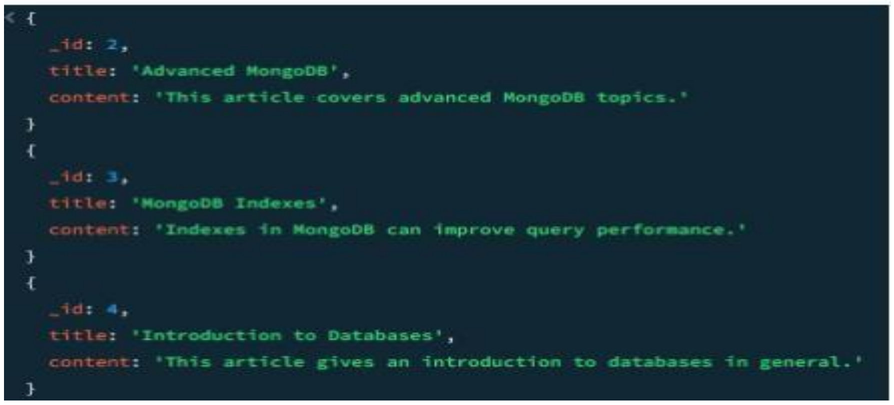
3. Exclude Documents Containing a Specific Phrase

To exclude documents that contain the phrase "MongoDB Basics" in the ‘title’ field:

```
db.articles.find({
   "title": {

      $not: /MongoDB Basics/
```

```
    }
})
```

Output:



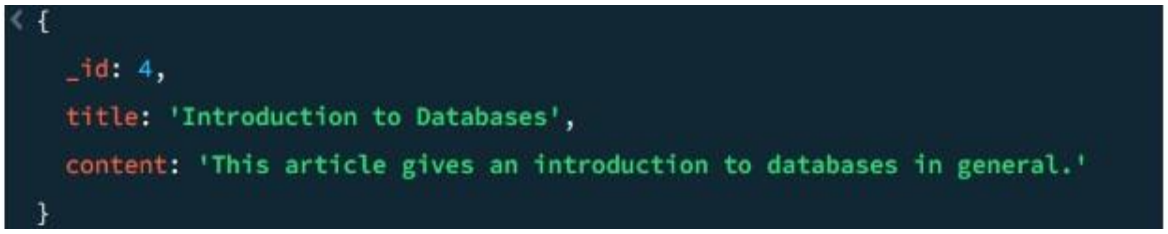
```
< {
    _id: 2,
    title: 'Advanced MongoDB',
    content: 'This article covers advanced MongoDB topics.'
  }
  {
    _id: 3,
    title: 'MongoDB Indexes',
    content: 'Indexes in MongoDB can improve query performance.'
  }
  {
    _id: 4,
    title: 'Introduction to Databases',
    content: 'This article gives an introduction to databases in general.'
  }
```

4. Exclude Documents Based on Multiple Fields

To exclude documents that contain "MongoDB" in the ‘title’ or "advanced" in the ‘content’:

```
db.articles.find({
   $and: [
      { "title": { $not: /MongoDB/ } },
      { "content": { $not: /advanced/ } }
   ]
})
```

Output:

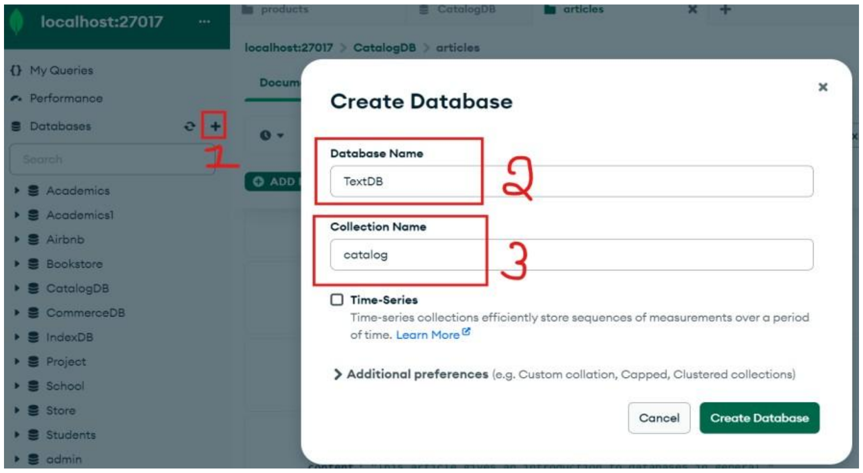


```
< {
    _id: 4,
    title: 'Introduction to Databases',
    content: 'This article gives an introduction to databases in general.'
  }
```

Program 10

Develop an aggregation pipeline to illustrate Text search on Catalog data collection.

Create a database **TextDB** and collection **catalog** in Mongo IDE.



Database Name: TextDB

Collection Name: catalog

Time-series collections efficiently store sequences of measurements over a period of time.

Add the following documents in the **catalog collection** in MongoDB Shell.

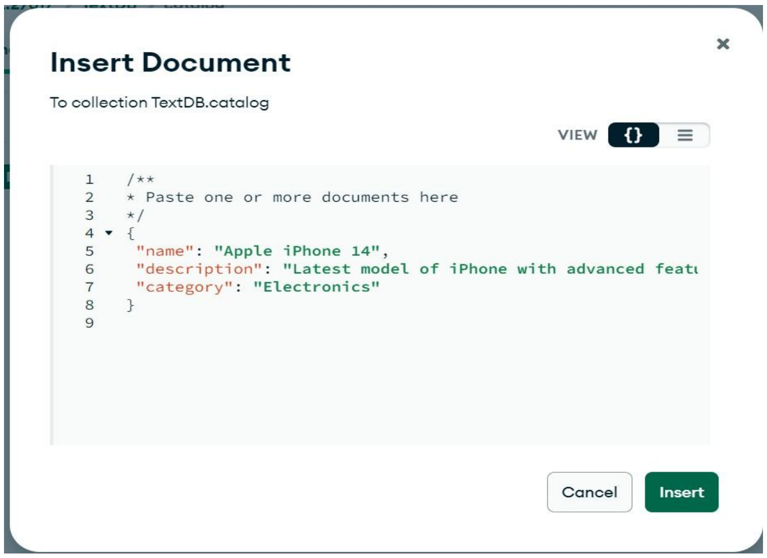
{

"name": "Apple iPhone 14",

"description": "Latest model of iPhone with advanced features",

"category": "Electronics"

}



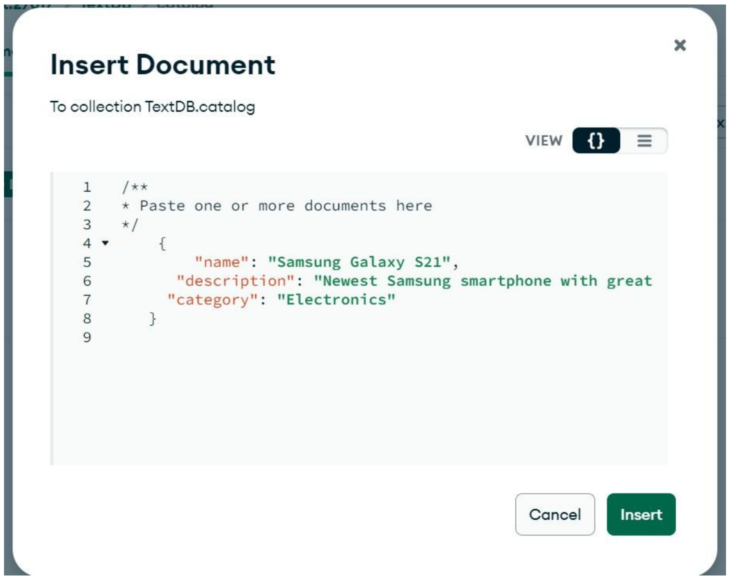
To collection TextDB.catalog

```
/**
* Paste one or more documents here
*/
{
 "name": "Apple iPhone 14",
 "description": "Latest model of iPhone with advanced featu
 "category": "Electronics"
}
```

{

"name": "Samsung Galaxy S21",

```
    "description": "Newest Samsung smartphone with great camera",

   "category": "Electronics"

 }
```

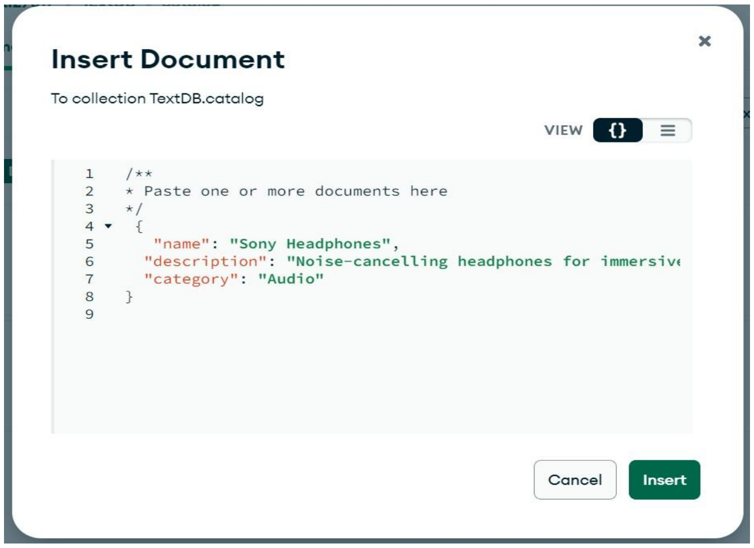


Insert Document

To collection TextDB.catalog

```
/**
* Paste one or more documents here
*/
    {
        "name": "Samsung Galaxy S21",
      "description": "Newest Samsung smartphone with great
     "category": "Electronics"
    }
```

```
{
  "name": "Sony Headphones",
 "description": "Noise-cancelling headphones for immersive sound",
 "category": "Audio"
}
```



Insert Document

To collection TextDB.catalog

```
/**
* Paste one or more documents here
*/
 {
   "name": "Sony Headphones",
  "description": "Noise-cancelling headphones for immersive
  "category": "Audio"
}
```

In MongoShell

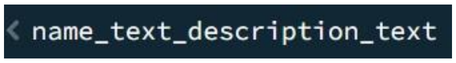
```
>use TextDB
```

Create a Text Index

First, create a text index on the '`name`' and '`description`' fields.

db.catalog.createIndex({ name: "text", description: "text" });

Output:



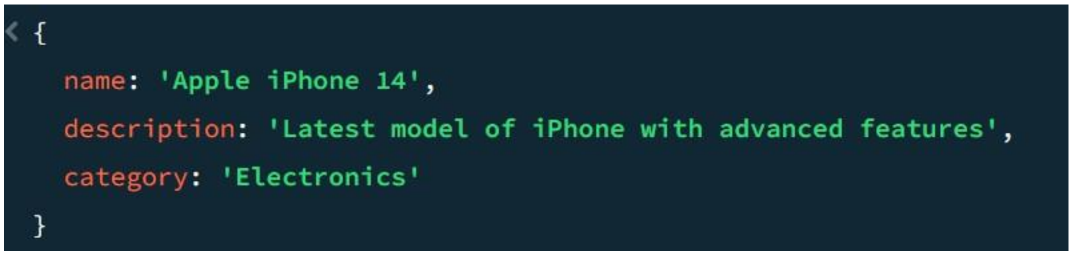
```
< name_text_description_text
```

Define the Aggregation Pipeline

Now, create an aggregation pipeline to perform the text search and process the results. Below is an example pipeline:

```
db.catalog.aggregate([
 // Stage 1: Match documents containing the search term
 {
   $match: {
     $text: { $search: " Apple iPhone 14" }
   }
 },
 // Stage 2: Project only required fields
 {
   $project:        {
_id: 0,        name:
1,      description:
1,
     category: 1

     // Add more fields as needed
   }
 }
])
```

Output:



```
< {
    name: 'Apple iPhone 14',
    description: 'Latest model of iPhone with advanced features',
    category: 'Electronics'
  }
```

VIVA QUESTIONS WITH ANSWERS

1.What makes MongoDB the best?
Answer: MongoDB is considered to be the best NoSQL database because of its following features:

- Document-oriented (DO)
- High performance (HP)
- High availability (HA)
- Easy scalability
- Rich query language

2.When do we use a namespace in MongoDB?

Answer: During the sequencing of the names of the database and the collection, the namespace is used.

3. If you remove an object attribute, is it deleted from the database?
Answer: Yes, it is deleted. Hence, it is better to eliminate the attribute and then save the object again.

4. How does MongoDB provide consistency?
Answer: MongoDB uses the reader–writer locks, allowing simultaneous readers to access any supply like a database or a collection but always offering private access to single writes.

5.Define MongoDB.
Answer: It is a document-oriented database that is used for high availability, easy scalability, and high performance. It supports the dynamic schema design.

6. What are the key features of MongoDB?
Answer: There are three main features of MongoDB:
Automatic scaling
High performance
High availability

6. What is CRUD?
Answer: MongoDB provides CRUD operations:
Create
Read
Update
Delete

7. What is Sharding?
Answer: In MongoDB, sharding means to store data on multiple machines.

8. What is Aggregation in MongoDB?
Answer: In MongoDB, aggregations are operations that process data records and return computed results.

9.Which syntax is used to create a Collection in MongoDB?
Answer: We can create a collection in MongoDB using the following syntax:

```
db.createCollection(name,options)
```

10. Which syntax is used to drop a Collection in MongoDB?
Answer: We can use the following syntax to drop a collection in MongoDB:

```
db.collection.drop()
```

11. What is the use of an Index in MongoDB?
Answer: In MongoDB, indexes provide high-performance read operations for frequently used queries.

12. Which command is used for inserting a document in MongoDB? Answer: The following command is used for inserting a document in MongoDB:

```
database.collection.insert (document)
```

13. What type of data is stored by MongoDB?
Answer: MongoDB stores data in the form of documents, which are JSON-like field and value pairs.

14. Which method is used to create an index?
Answer: The createIndex() method is used to create an index.

15. Which command is used to create a database?
Answer: To create a database, we can use the Database_Name command.

16. Which command is used to drop a database?
Answer: The db.dropDatabse() command is used to drop a database.

17. What is the use of the pretty() method?
Answer: The pretty() method is used to show the results in a formatted way.

18. Which method is used to remove a document from a collection?
Answer: The remove() method is used to remove a document from a collection.

19. Define MongoDB Projection.
Answer: Projection is used to select only the necessary data. It does not select the whole data of a document.

20. What is the use of the limit() method?
Answer: The limit() method is used to limit the records in the database.

21. What is the syntax of the limit() method?

Answer: The syntax of the limit() method is as follows:

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

22. What is the syntax of the sort() method?
Answer: In MongoDB, the following syntax is used for sorting documents:

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

23. What is a Collection in MongoDB?
Answer: In MongoDB, a collection is a group of MongoDB documents.

24. What is the use of the db command?
Answer: The db command gives the name of the currently selected database.

25. Which method is used to update documents into a collection?
Answer: The update() and save() methods are used to update documents into a collection.

26. What is the syntax of the skip() method?
Answer: The syntax of the skip() methopd is as follows:

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER
```

27. Which command is used to restore the backup?
Answer: The mongo restore command is used to restore the backup.

28. Define the Aggregation pipeline.
Answer: The aggregation pipeline is a framework for performing aggregation tasks. The pipeline is used to transform documents into aggregated results.

29.What is MongoDB?
Answer: MongoDB is a NoSQL, document-oriented database designed for scalability and flexibility. It stores data in JSON-like documents with dynamic schemas.

30.What is a NoSQL database?
Answer: NoSQL databases are non-relational databases that provide a mechanism for storage and retrieval of data. They are designed for large-scale data storage and for massively-parallel, highperformance applications.

31.What is a document in MongoDB?
Answer: A document in MongoDB is a set of key-value pairs. Documents are analogous to rows in relational databases, but they are more flexible, allowing embedded documents and arrays

32.What is a collection in MongoDB?
Answer: A collection in MongoDB is a group of documents. Collections are analogous to tables in relational databases.
33.How do you insert a document into a collection in MongoDB?

Answer: You can insert a document into a collection using the insertOne() or insertMany() methods.

34.What is an index in MongoDB?
Answer: An index in MongoDB is a special data structure that improves the speed of data retrieval operations on a collection.

35.What is the difference between findOne() and find() in MongoDB?
Answer: findOne() returns the first document that matches the query, while find() returns a cursor to all documents that match the query.

36.What is the purpose of the _id field in MongoDB?
Answer: The _id field is a unique identifier for a document in a MongoDB collection. It is automatically generated if not provided by the user.

37.Can you explain what an aggregation pipeline is in MongoDB?
Answer: An aggregation pipeline is a framework for data aggregation in MongoDB, allowing users to process data through a sequence of stages, transforming and filtering the data as it passes through each stage**.**

38.How do you remove all documents from a collection in MongoDB?
Answer: You can remove all documents from a collection using the deleteMany() method with an empty filter

39.What is the purpose of the $set operator in MongoDB?
Answer: The $set operator is used to update the value of a field in a document. If the field does not exist, $set adds it to the document.

40.What are the main components of MongoDB architecture?
Answer: The main components of MongoDB architecture are the database, collections, documents, and indexes.

41.How do you list all collections in a MongoDB database?
Answer: You can list all collections in a MongoDB database using the show collections command in the MongoDB shell**.**

42.How do you limit the number of documents returned by a query in MongoDB?
Answer: You can limit the number of documents returned by a query using the **limit()** method.

43.What is the purpose of the $or operator in MongoDB?
Answer: The $or operator is used to perform a logical OR operation on an array of conditions and select documents that match at least one of the conditions.

44.How do you sort the results of a query in MongoDB?

Answer: You can sort the results of a query using the **sort()** method.

45.What is the purpose of the $push operator in MongoDB?
Answer: The $push operator is used to append a value to an array field in a document**.**

46.What is a primary key in MongoDB?
Answer: In MongoDB, the _id field is the primary key for a document. It uniquely identifies each document in a collection**.**

47.How do you find the number of documents in a collection?
Answer: You can find the number of documents in a collection using the countDocuments() method.

48.What is the difference between drop() and dropDatabase() in MongoDB?
Answer: The drop() method is used to remove a collection from a database, while the **dropDatabase()** method is used to remove an entire database.

49.What is the purpose of the $pull operator in MongoDB?
Answer: The $pull operator is used to remove all instances of a value from an array field in a document.

50.What is the purpose of the $ne operator in MongoDB?
Answer: The $ne (not equal) operator is used to select documents where the value of a field is not equal to the specified value**.**
51.How do you create a text index in MongoDB?

Answer: You can create a text index using the createIndex() method with the field set to text.
52.What is the purpose of the $match stage in the aggregation pipeline?

Answer: The $match stage in the aggregation pipeline is used to filter documents based on specified criteria, similar to the find() method.