MONGO DB

LAB MANUAL

EXPERIMENT 1:

Introduction to NoSQL Databases

A database Management System provides the mechanism to store and retrieve the data. There are different kinds of database Management Systems:

1. RDBMS (Relational Database Management Systems)
2. OLAP (Online Analytical Processing)
3. NoSQL (Not only SQL)

In this guide, We will discuss NoSQL. **NoSQL databases** were created to overcome the limitations of relational databases.

What is a NoSQL database?

NoSQL databases are different than relational databases like MQSql. In relational database you need to create the table, define schema, set the data types of fields etc before you can actually insert the data. In NoSQL you don't have to worry about that, you can insert, update data on the fly.

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. There are certain situations where you would prefer relational database over NoSQL, however when you are dealing with huge amount of data then NoSQL database is your best choice.

Limitations of Relational databases

1. In relational database we need to define structure and schema of data first and then only we can process the data.
2. Relational database systems provides consistency and integrity of data by enforcing ACID properties (Atomicity, Consistency, Isolation and Durability). There are some scenarios where this is useful like banking system. However in most of the other cases these properties are significant performance overhead and can make your database response very slow.
3. Most of the applications store their data in JSON format and RDBMS don't provide you a better way of performing operations such as create, insert, update, delete etc on this data. On the

other hand NoSQL store their data in JSON format, which is compatible with most of the today's world application.

What are the advantages of NoSQL

There are several advantages of working with NoSQL databases such as MongoDB and Cassandra. The main advantages are high scalability and high availability.

High scalability: NoSQL database such as MongoDB uses sharding for horizontal scaling. Sharding is partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved. Vertical scaling means adding more resources to the existing machine while horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement, on the other hand horizontal scaling is easy to implement. Horizontal scaling database examples: MongoDB, Cassandra etc. Because of this feature NoSQL can handle huge amount of data, as the data grows NoSQL scale itself to handle that data in efficient manner.

High Availability: Auto replication feature in MongoDB makes it highly available because in case of any failure data replicates itself to the previous consistent state.

Types of NoSQL database

Here are the types of NoSQL databases and the name of the databases system that falls in that category. MongoDB falls in the category of NoSQL document based database.

Key Value Store: Memcached, Redis, Coherence

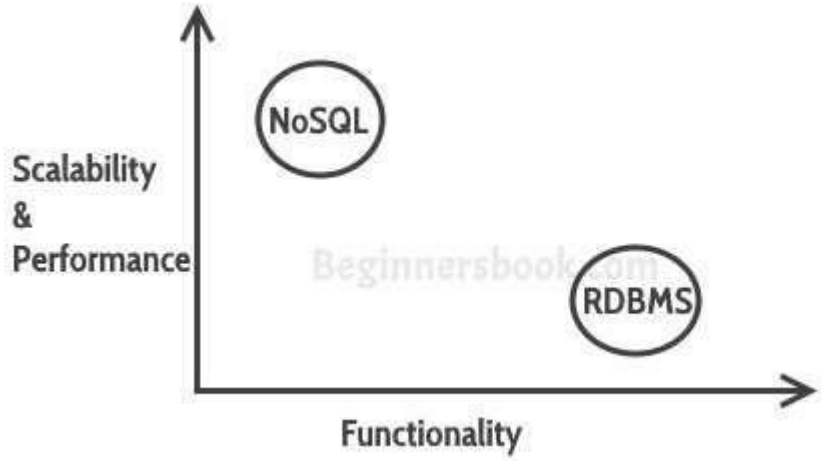
Tabular: Hbase, Big Table, Accumulo

Document based: MongoDB, CouchDB, Cloudant

RDBMS Vs NoSQL

RDBMS: It is a structured data that provides more functionality but gives less performance.

NoSQL: Structured or semi structured data, less functionality and high performance.



So when I say less functionality in NoSQL what's missing:

1. You can't have constraints in NoSQL
2. Joins are not supported in NoSQL

These supports actually hinders the scalability of a database, so while using NoSQL database like MongoDB, you can implements these functionalities at the application level.

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.

Differences between MongoDB & SQL:

SQL Database	**NoSQL Database (MongoDB)**
Is Relational database	Is Non-relational database
It supports SQL query language	It supports JSON query language
It is generally Table-based structure	It is mostly Collection-based and key-value pair structure
It follows Row-based structure	It follows Document-based structure
It follows Column-based structure	It follows Field-based structure
It supports foreign keys	It does not support foreign keys
It supports triggers	It does not support triggers
Schema is predefined	Schema is dynamic
Not good to use for hierarchical data storage	Good to use for hierarchical data storage
Due to Vertically scalability –user can increase RAM	Due to Horizontally scalability – user can add more servers
Highlights on ACID properties (Atomicity, Consistency, Isolation and Durability)	Highlights on CAP theorem (Consistency, Availability and Partition tolerance)
Tools used are Microsoft SQL Server**,** PostgreSQL**,** MySQL, Oracle, etc	Tools used are MongoDB, Cassandra, CouchDB, Bigtable, FlockDB, ArangoDB, etc

Performance comparision between MongoDB and SQL

- **SQL databases** performance tuning consists of making queries of a relational database run as fast as possible. Indexing in data structure improves the speed of data retrieval operations on a database table by providing rapid random lookups and efficient access of ordered records. It's high-performing for complex queries.
- **MongoDB Performance** As you develop and operate applications with MongoDB, you may need to analyse the performance of the application and its database. When you come across tarnished performance, it is mostly due to database access strategies or hardware availability or the number of open database connections. It's high-performing for simple querry

MongoDB : which type of DB?

- **Mongo DB – Document Oriented NoSQL Database Approach**

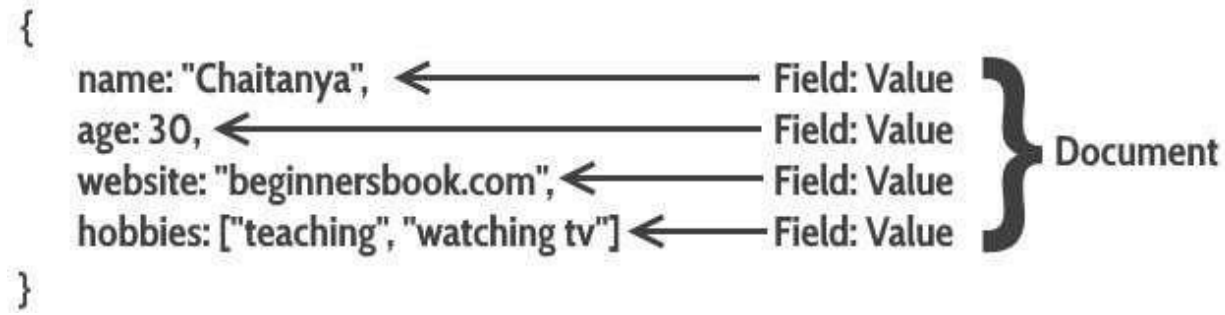
  In mongo DB ,Querying on this model is easy, since the schema is de-normalized. No joins are required. So we plan to use the Mongo DB based solution

Introduction to MongoDB

MongoDB is an open source, document oriented database that stores data in form of documents (key and value pairs). As discussed in our last tutorial (NoSQL introduction) that document based databases are one of types of NoSQL databases.

What is a document?

If you came from a relational database background then you can think of them as rows in RDBMS. The mapping between relational database and MongoDB is covered in the next tutorial so if you want to know the equivalent of rows, tables, columns in MongoDB, you should definitely check it: Mapping Relational database to MongoDB.




```
{
  name: "Chaitanya",
  age: 30,
  website: "beginnersbook.com",
  hobbies: ["Teaching", "Watching TV"]
}
```

This is a JSON like structure. Where data is stored in form of key and value pairs.

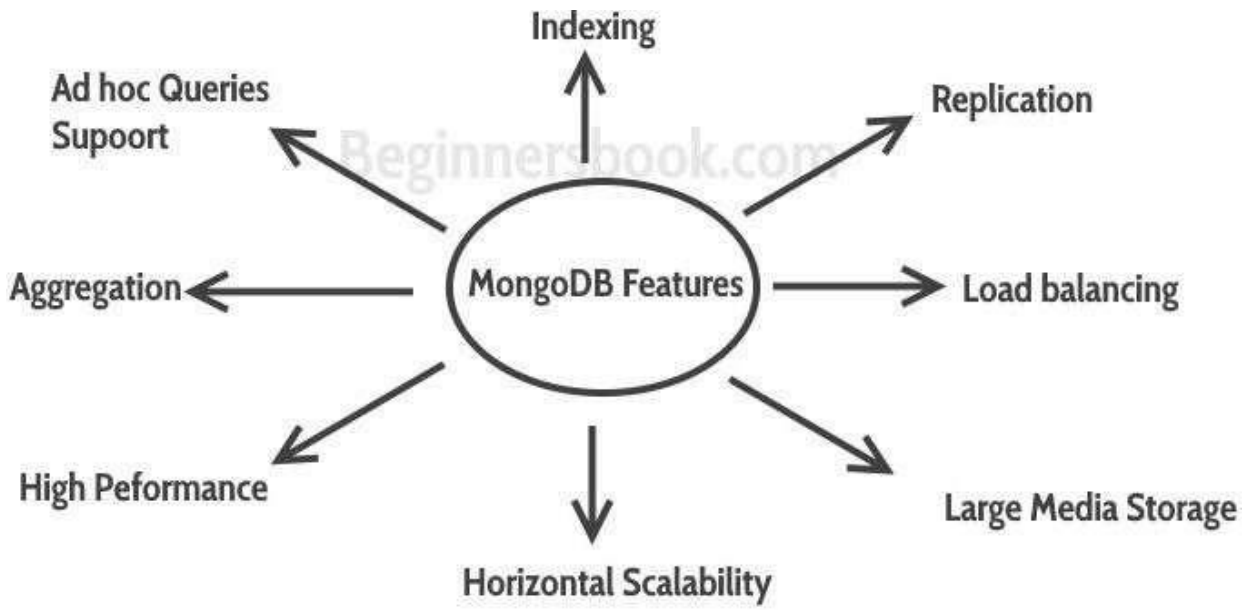
History of MongoDB

MongoDB was created by Eliot and Dwight (founders of DoubleClick) in 2007, when they faced scalability issues while working with relational database. The organization that developed MongoDB was originally known as 10gen.

In Feb 2009, they changed their business model and released MongoDB as an open source Project. The organization changed its name in 2013 and now known as MongoDB Inc.

Features of MongoDB

1. MongoDB provides **high performance**. Most of the operations in the MongoDB are faster compared to relational databases.
2. MongoDB provides **auto replication** feature that allows you to quickly recover data in case of a failure.
3. Horizontal scaling is possible in MongoDB because of sharing. Sharding is partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved.



Horizontal scaling vs vertical scaling:

Vertical scaling means adding more resources to the existing machine while horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement, on the other hand horizontal scaling is easy to implement. Horizontal scaling database examples: MongoDB, Cassandra etc.

4. Load balancing: Horizontal scaling allows MongoDB to balanace the load.

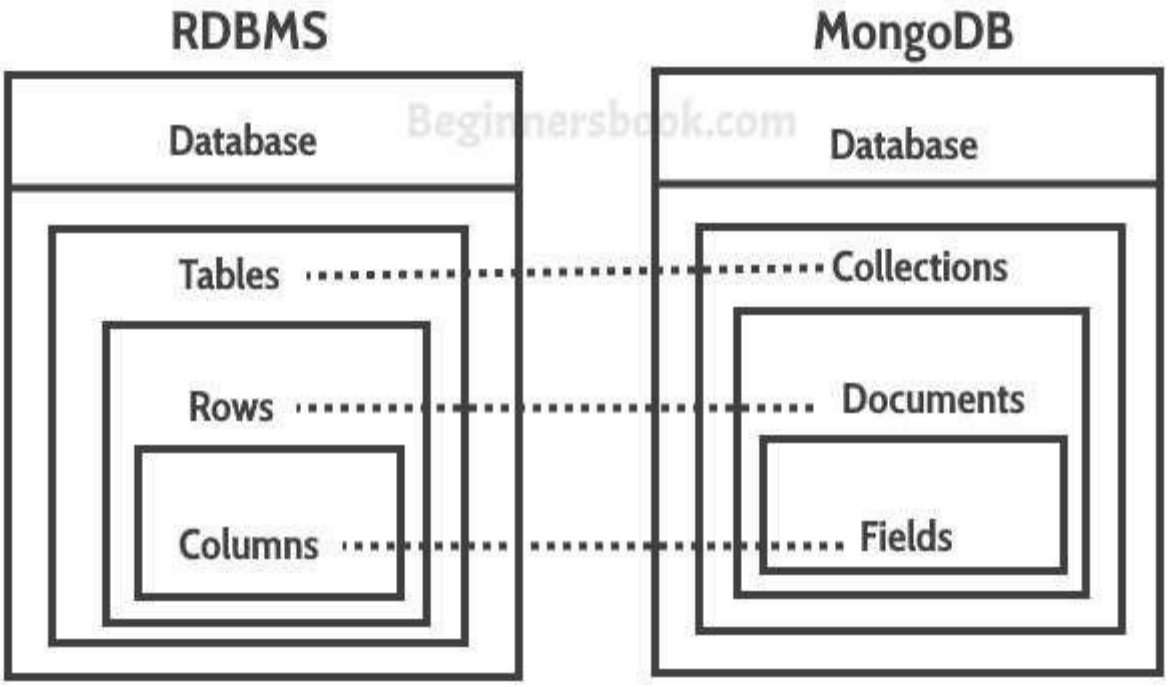
5. High Availabilty: Auto Replication improves the availability of MongoDB database.

6. Indexing: Index is a single field within the document. Indexes are used to quickly locate data without having to search every document in a MongoDB database. This improves the performance of operations performed on the MongoDB database.

Mapping Relational Databases to MongoDB

If you are coming from a relational database background then it might be difficult for you to relate the RDBMS terms with MongoDB. In this guide, we will see the mapping between relational database and MongoDB.

Mapping relational database to MongoDB



Collections in MongoDB is equivalent to the tables in RDBMS.

Documents in MongoDB is equivalent to the rows in RDBMS.

Fields in MongoDB is equivalent to the columns in RDBMS.

Fields (key and value pairs) are stored in document, documents are stored in collection and collections are stored in database.

This is how a document looks in MongoDB: As you can see this is similar to the row in RDBMS. The only difference is that they are in JSON format.

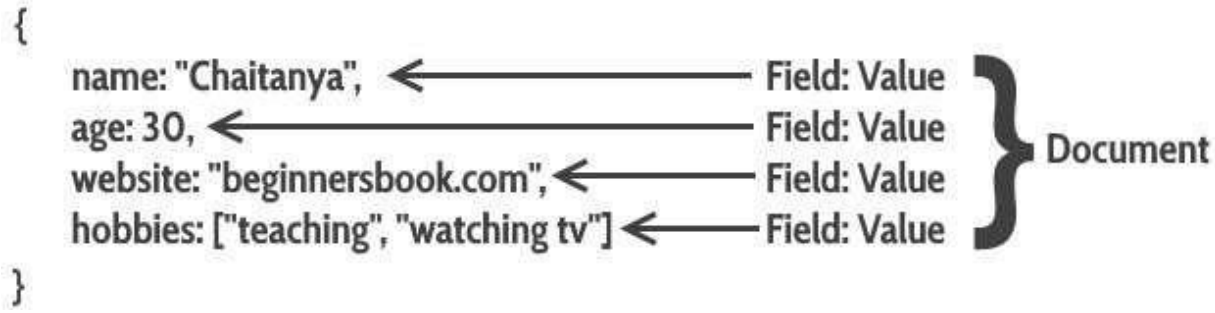
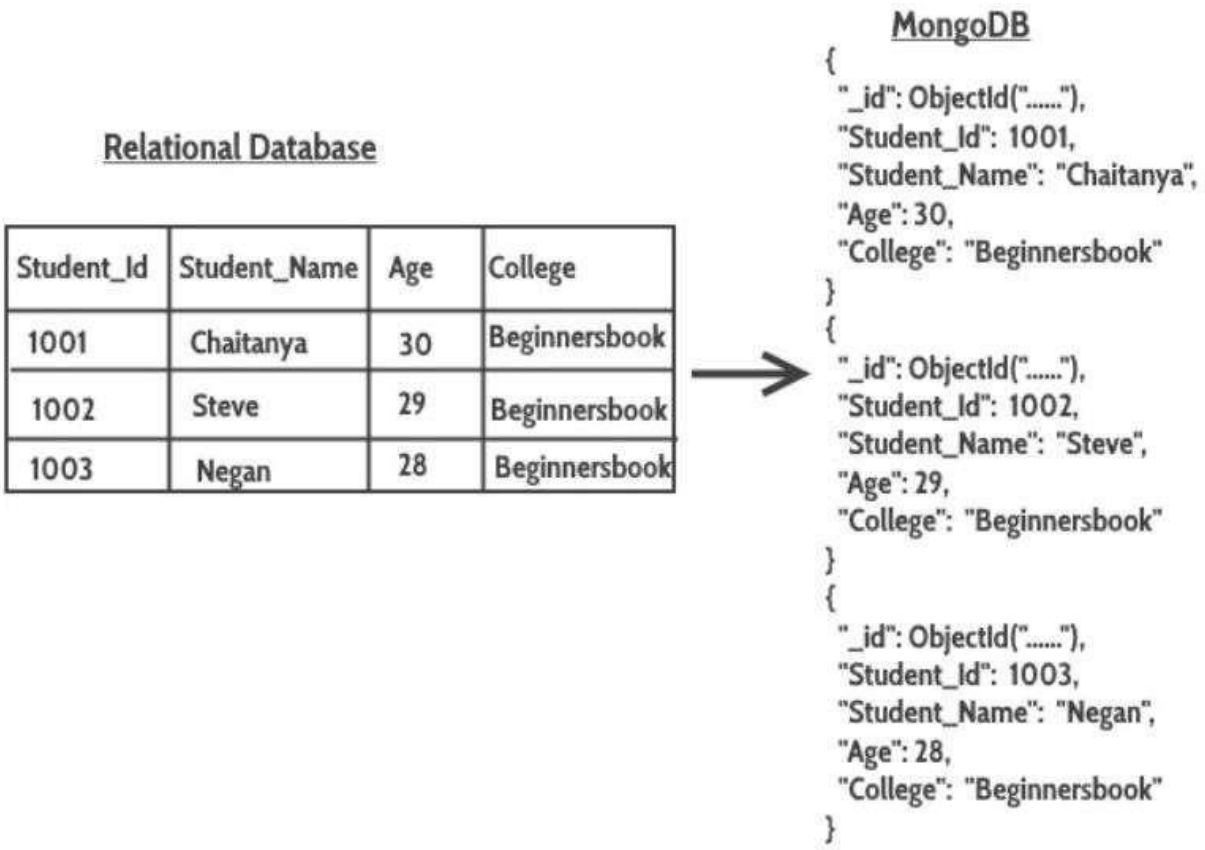


Table vs Collection

Here we will see how a table in relational database looks in MongoDB. As you see columns are represented as key-value pairs(JSON Format), rows are represented as documents. MongoDB automatically inserts a unique _id(12-byte field) field in every document, this serves as primary key for each document.

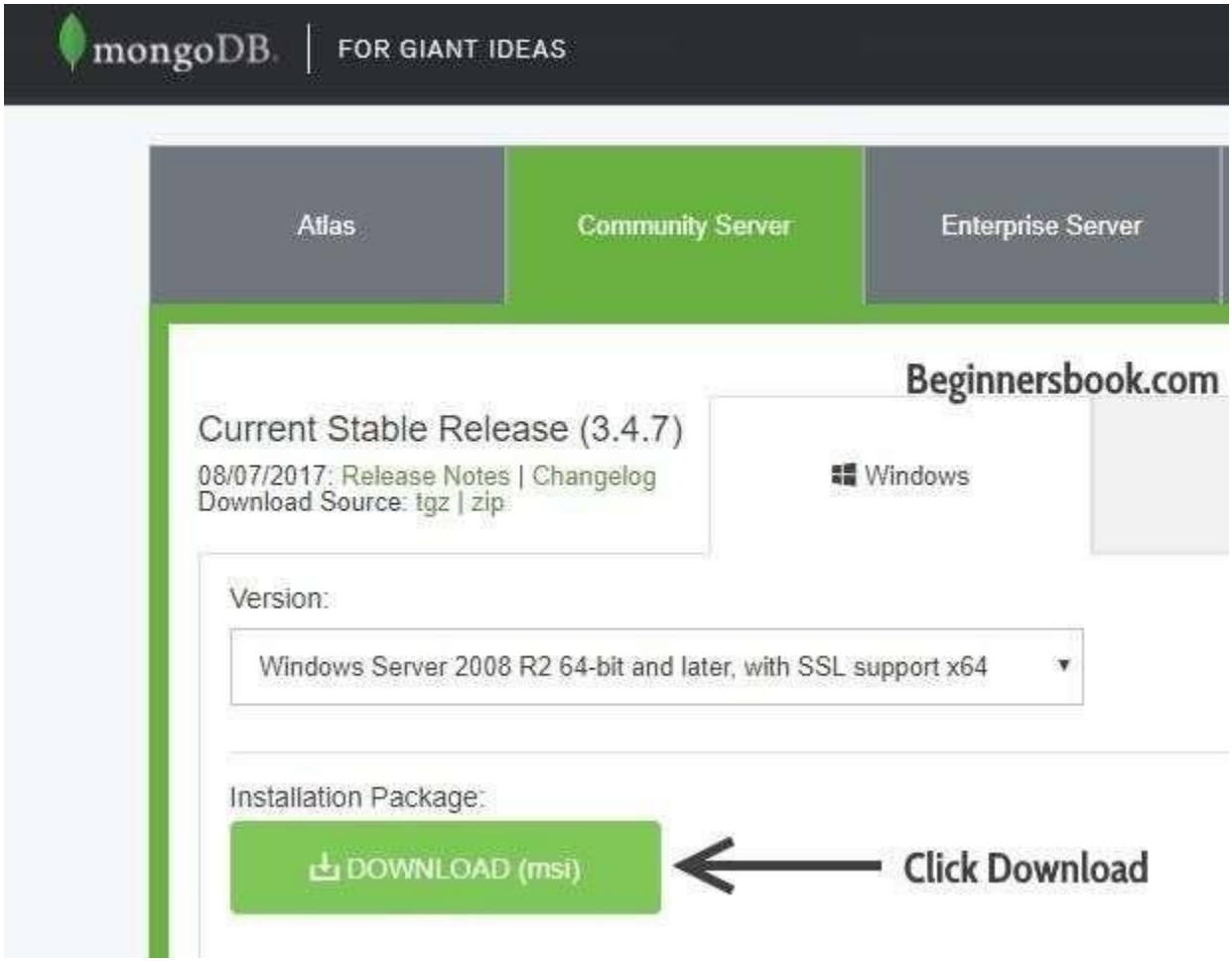


Another cool thing about MongoDB is that it supports dynamic schema which means one document of a collection can have 4 fields while the other document has only 3 fields. This is not possible in relational database.

How to install and Configure MongoDB for Windows

Install MongoDB on Windows

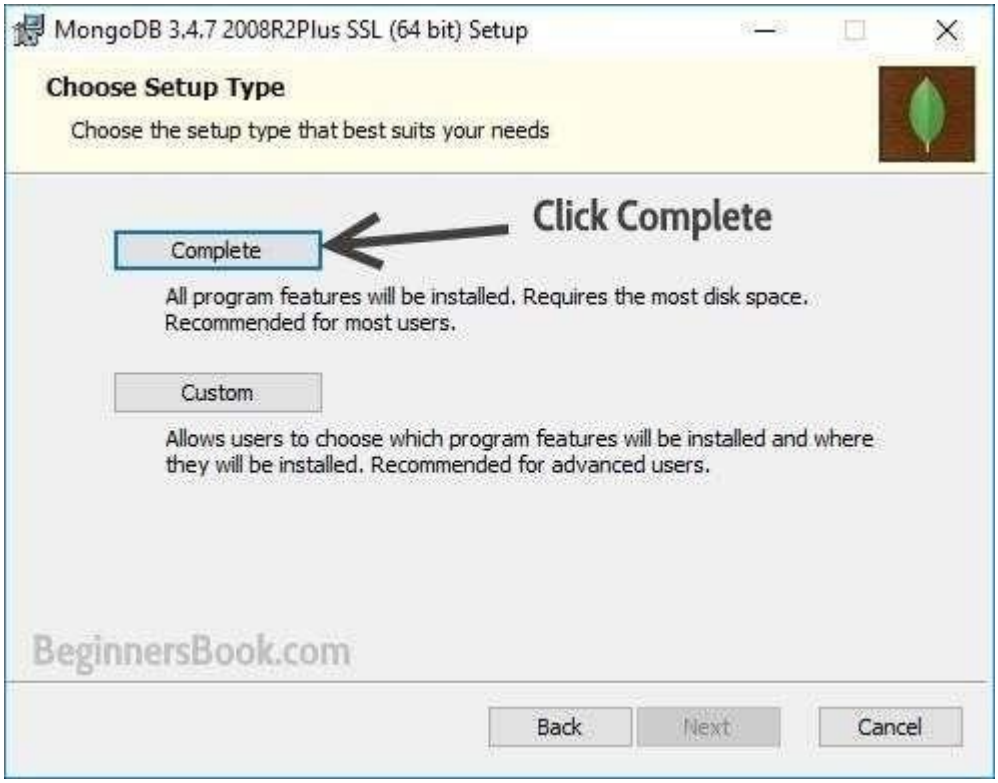
Step 1: Go to MongoDB download Page and click download as shown in the screenshot. A .msi file like this **mongodb-win32-x86_64-2008plus-ssl-3.4.7-signed** will be downloaded in your system. Double click on the file to run the installer.



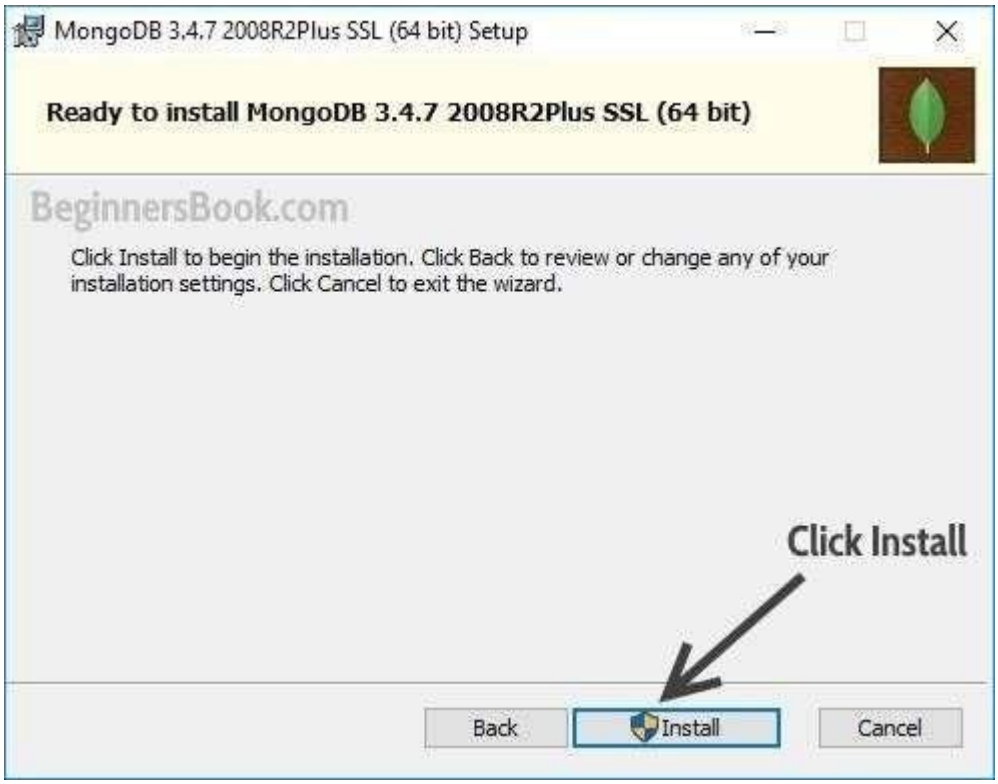
Step 2: Click Next when the MongoDB installation windows pops up.

Step 3: Accept the MongoDB user Agreement and click Next.

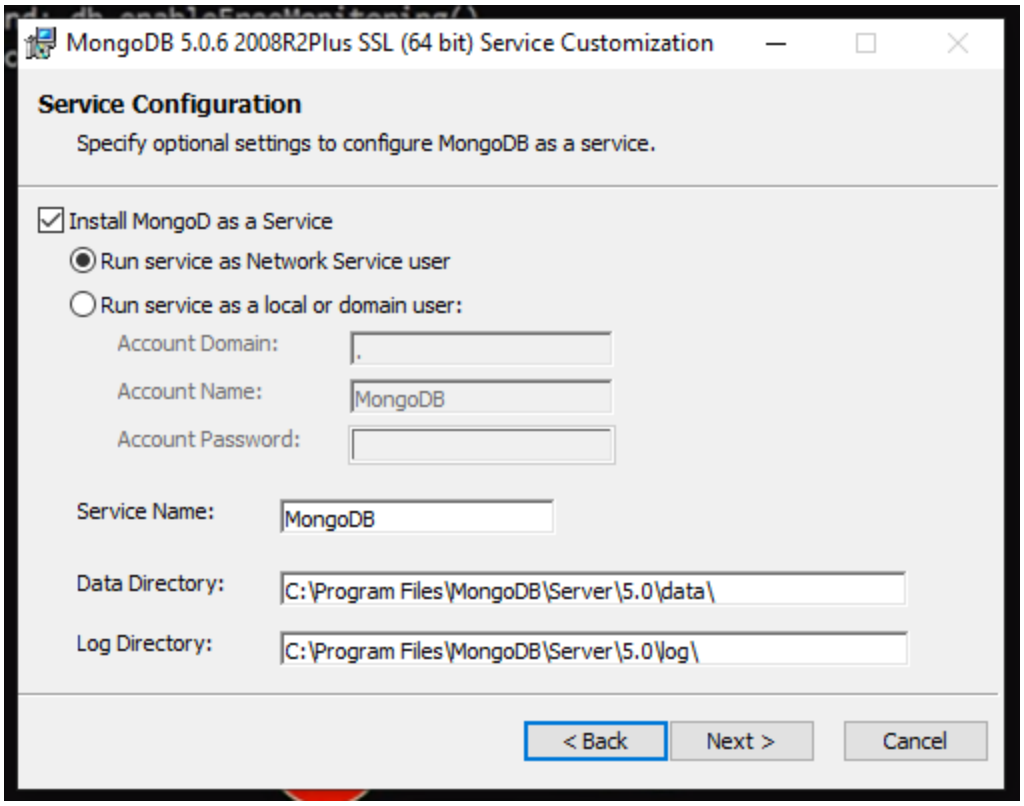
Step 4: When the setup asks you to choose the Setup type, choose Complete.

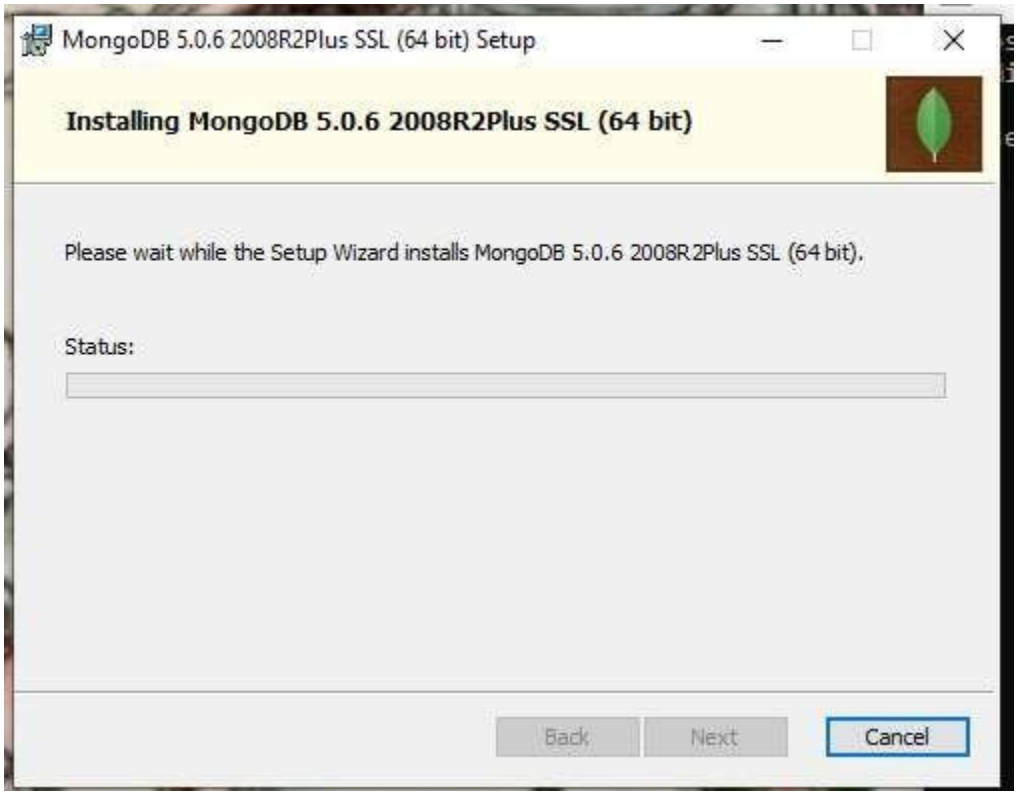


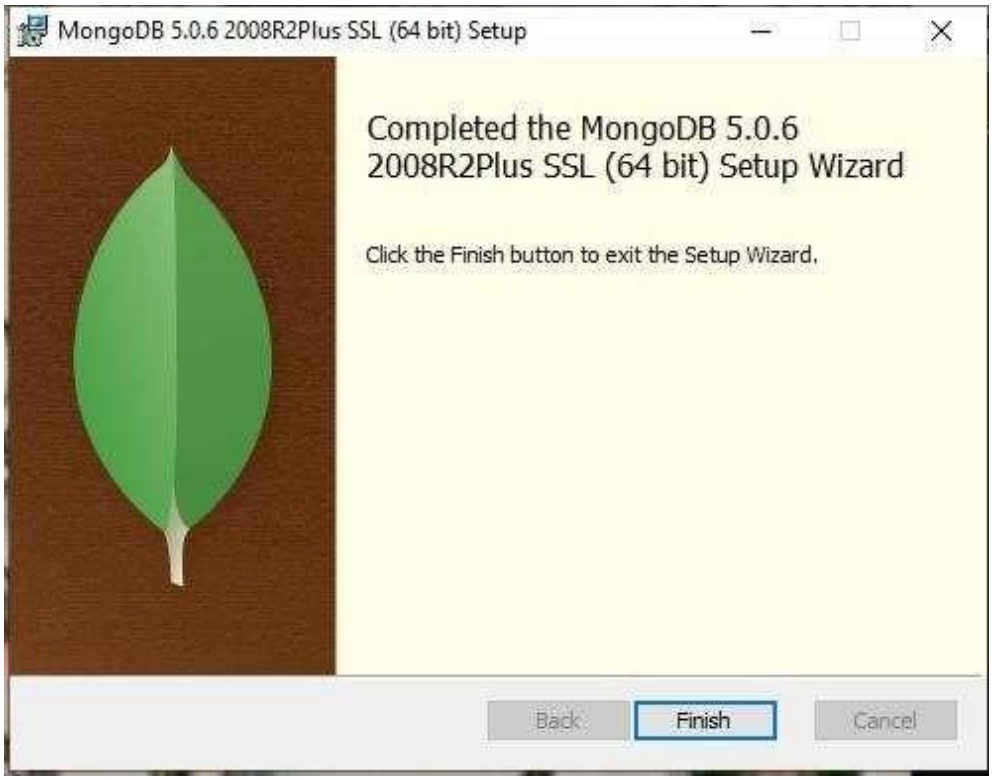
Step 5: Click Install to begin the installation.



Step 6: That's it. Click Finish once the MongoDB installation is complete.


MongoDB 5.0.6 2008R2Plus SSL (64 bit) Service Customization
Service Configuration
Specify optional settings to configure MongoDB as a service.
Install MongoD as a Service
Run service as Network Service user
Run service as a local or domain user:
Account Domain:
.
Account Name:
MongoDB
Account Password:
Service Name:
MongoDB
Data Directory:
C:\Program Files\MongoDB\Server\5.0\data\
Log Directory:
C:\Program Files\MongoDB\Server\5.0\log\
< Back
Next >
Cancel


MongoDB 5.0.6 2008R2Plus SSL (64 bit) Setup
Installing MongoDB 5.0.6 2008R2Plus SSL (64 bit)
Please wait while the Setup Wizard installs MongoDB 5.0.6 2008R2Plus SSL (64 bit).
Status:
Back
Next
Cancel

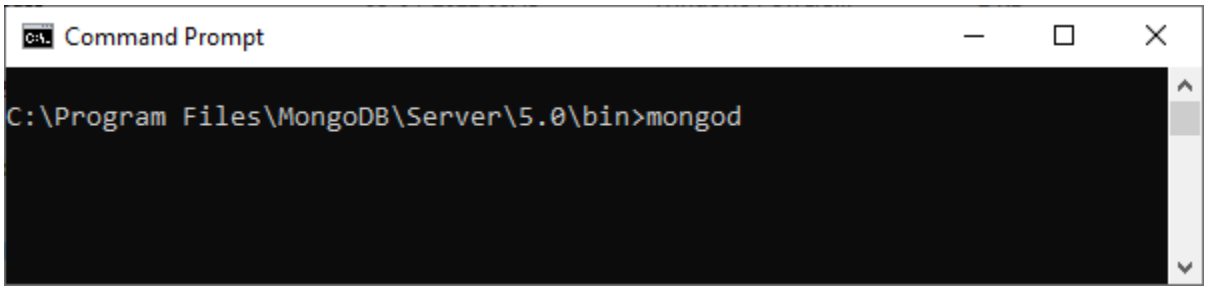

MongoDB 5.0.6 2008R2Plus SSL (64 bit) Setup
Completed the MongoDB 5.0.6 2008R2Plus SSL (64 bit) Setup Wizard
Click the Finish button to exit the Setup Wizard.
Back
Finish
Cancel

EXPERIMENT 2

Demonstrate how to create and drop a database in MongoDB

- MongoDB collection ≡ My SQL Tables
- MongoDB document ≡ My SQL rows

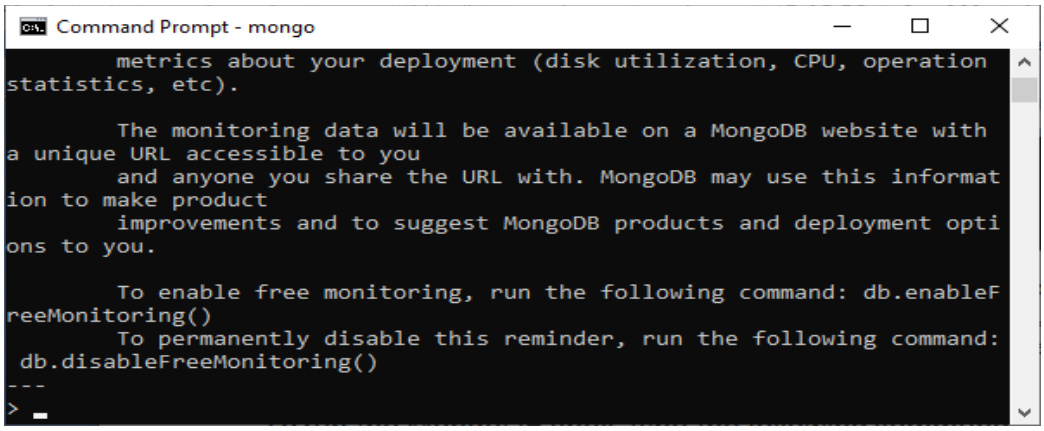
Start the MongoDB service by typing this command:



mongod is **a background process used by MongoDB**. The main purpose of mongod is to manage all the MongoDB server tasks. For instance, accepting requests, responding to client, and memory management. mongo is a command line shell that can interact with the client (for example, system administrators and developers).

To run the MongoDB shell, type the following command:

After pressing enter we are at the MongoDB shell as shown in below figure



Once you are in the MongoDB shell, create the database in MongoDB by typing this command:

>**use database_name** // to create a new DB or to connect to already existed DB

>**db** // to check currently connected DB

>**show dbs** // to list all the DBs

The DB **madavi** is created; is not present in the list of all the databases. This is because a database is not created until you save a document in it.

Note: If the database name you mentioned is already present then this command will connect you to the database. However if the database doesn't exist then this will create the database with the given name and connect you to it.

- Now we are creating a collection **Student** and inserting a document in it.

  >**db.student.insert({name: "sree", age: 30, address:"vijayawada"})**
- You can now see that the database "madavi" is created.

To Drop the DataBase ,

> show dbs // list all Dbs

> use databse_name //switch to the DB that needs to be dropped

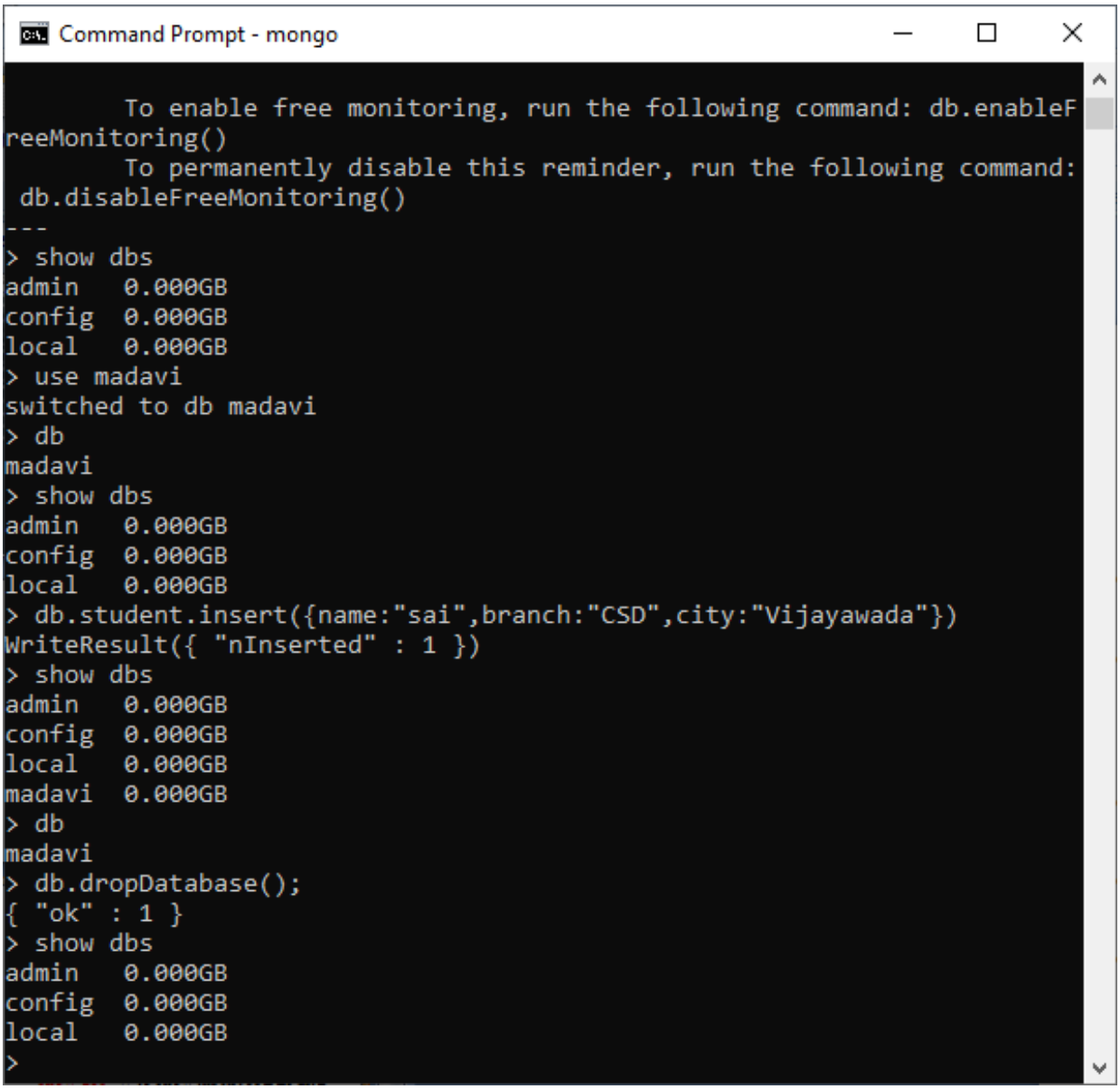
- Example:

  >use madavi

  >db.dropDatabase()

  > show dbs // to show the list of DBs after deletion.

OUTPUT:


Command Prompt - mongo
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
> use madavi
switched to db madavi
> db
madavi
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
> db.student.insert({name:"sai",branch:"CSD",city:"Vijayawada"})
WriteResult({ "nInserted" : 1 })
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
madavi 0.000GB
> db
madavi
> db.dropDatabase();
{ "ok" : 1 }
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
>

Experiment3

Creating the Collection in MongoDB on the fly.

- The cool thing about MongoDB is that you need not to create collection before you insert document in it. With a single command you can insert a document in the collection and the MongoDB creates that collection on the fly.
- **SYNTAX:**

  db.collection_name.insert({key:value, key:value…})
- **EXAMPLE:**

  db.student.insert({rollno:”20X41A0441”,name:”durga”,age:18,city:“Vijayawada”})
- **SYNTAX: db.collection_name.find()**
- To check whether the collection is created successfully, use the following command.

> show collections // This command shows the list of all the collections in the currently selected database.

OUTPUT:

SYNOPSIS:

```
>db.collection_name.drop()              // to drop the collection
>db.dropDatabase()                      // to drop the current Database
>db                                     //  to show the current Database name
> show dbs                              // to show all the list of DBs
>show collections                       // to show the collection name
>db.collection_name.find()              // to find the documents in the collection
> use databasename                      // to create the new DB
>db.createCollection(name,options)  // to create new collection
```

Experiment - 4

Creating collection with options before inserting the documents and Drop the collection created.

- We can also create collection before we actually insert data in it. This method provides you the options that you can set while creating a collection.

  SYNTAX:

  db.createCollection(name, options)

  - name is the collection name
  - options is an optional field that we can use to specify certain parameters such as size, max number of documents etc. in the collection.

  db.collection_name.drop()

  - **Note:** Once you drop a collection all the documents and the indexes associated with them will also be dropped. To preserve the indexes we use remove() function that only removes the documents in the collection but doesn't remove the collection itself and the indexes created on it. We will learn about indexes and remove() function in the later tutorials.

  EXAMPLE:

  ```
  > db.createCollection("students")
    { "ok" : 1 }
  > db.students.drop()
     true
  ```

OPTIONS field in the above syntax:

- options that we can provide while creating a collection:
  capped: type: boolean.
  This parameter takes only true and false. This specifies a cap on the max entries a collection can have. Once the collection reaches that limit, it starts overwriting old entries.
  The point to note here is that when you set the capped option to true you also have to specify the size parameter.

- **size: type: number.**
  This specifies the max size of collection (capped collection) in bytes.

- **max: type: number.**
  This specifies the max number of documents a collection can hold.

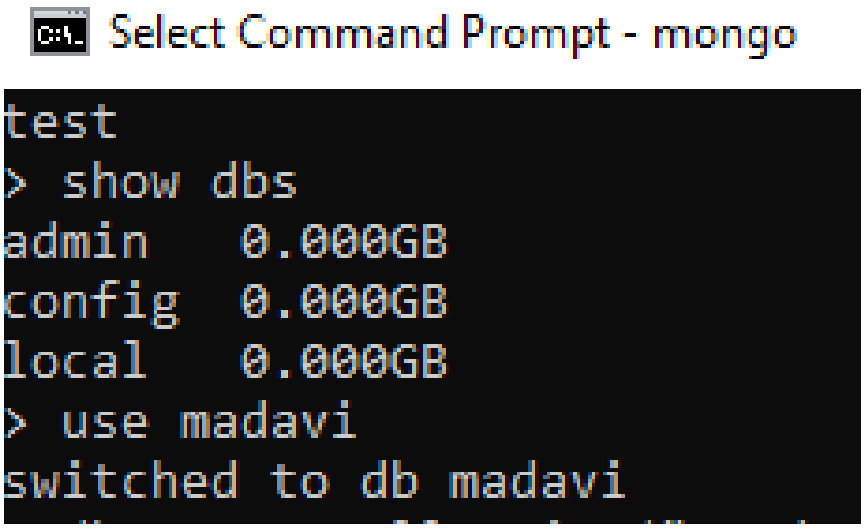
- **autoIndexId: type: boolean**
  The default value of this parameter is false. If you set it true then it automatically creates index field _id for each document. We will learn about index in the MongoDB indexing tutorial.

EXAMPLE of capped collection:

- >db.createCollection("teachers", { capped : true, size : 9232768})

  { "ok" : 1 }

- This command will create a collection named “teachers” with the max size of 9232768 bytes. Once this collection reaches that limit it will start overwriting old entries.

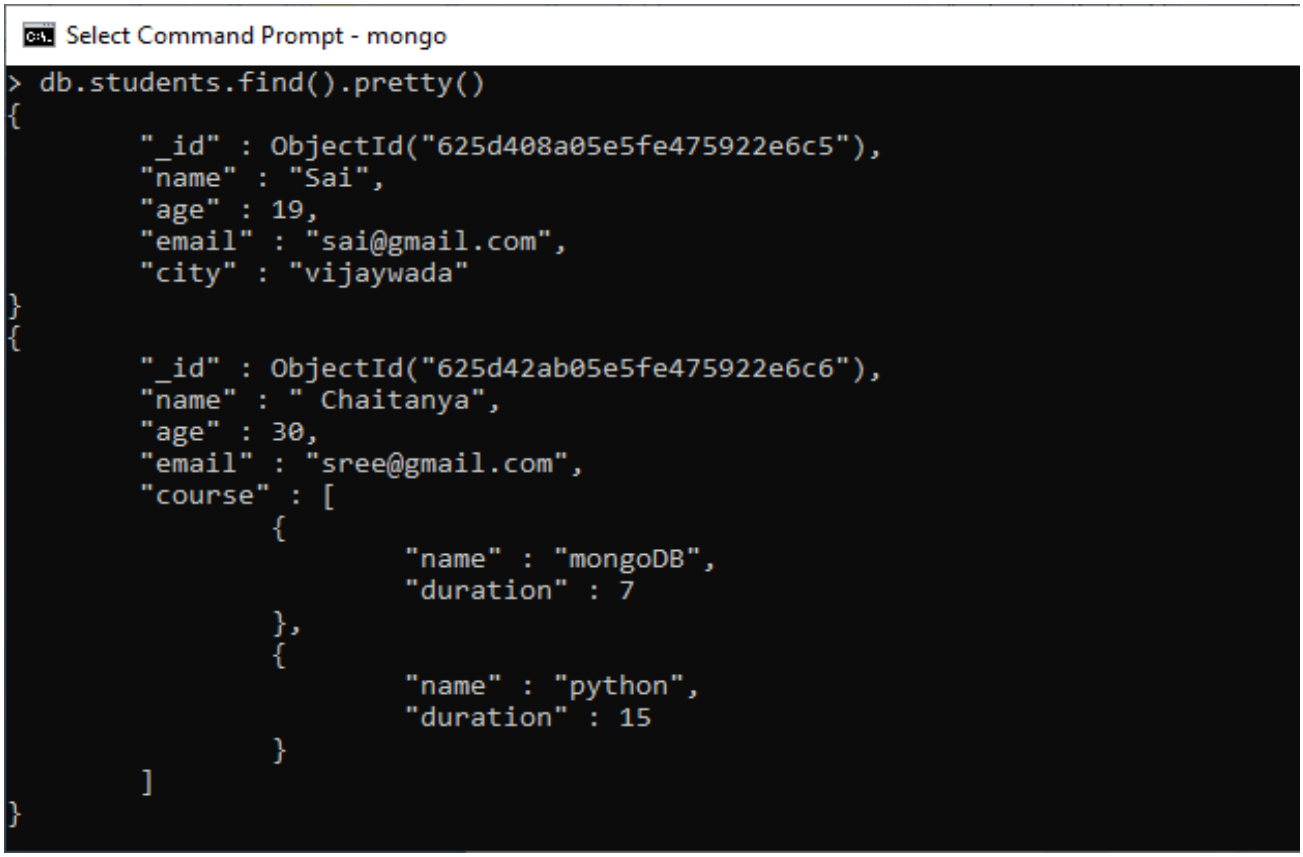


Select Command Prompt - mongo

```
> db.createCollection("students",{capped:true,size:9232768,max:3,autiIndexId:true})
{
        "ok" : 0,
        "errmsg" : "BSON field 'create.autiIndexId' is an unknown field.",
        "code" : 40415,
        "codeName" : "Location40415"
}
> show collections
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Select Command Prompt - mongo

```
> db.studnets.find()
> db.createCollection("students",{capped:true,size:9232768,max:3,autiIndexId:true})
{
        "ok" : 0,
        "errmsg" : "BSON field 'create.autiIndexId' is an unknown field.",
        "code" : 40415,
        "codeName" : "Location40415"
}
> db.students.find()
> db.students.insert({name: "Sai",age:19,email:"sai@gmail.com",city:"vijaywada"})
WriteResult({ "nInserted" : 1 })
> db.students.insert(
... {
...     name:" Chaitanya",
...     age:30,
...     email: "sree@gmail.com",
...     course: [{ name: "mongoDB", duration:7},{name:"python",duration:15}]
... }
... )
WriteResult({ "nInserted" : 1 })
```


Select Command Prompt - mongo
> db.students.find().pretty()
{
"_id" : ObjectId("625d408a05e5fe475922e6c5"),
"name" : "Sai",
"age" : 19,
"email" : "sai@gmail.com",
"city" : "vijaywada"
}
{
"_id" : ObjectId("625d42ab05e5fe475922e6c6"),
"name" : " Chaitanya",
"age" : 30,
"email" : "sree@gmail.com",
"course" : [
{
"name" : "mongoDB",
"duration" : 7
},
{
"name" : "python",
"duration" : 15
}
]
}

EXPERIMENT 5

MongoDB insert document

a. Insert single document

b. Insert multiple documents in collection

a. Insert single document

SYNTAX to insert a document into the collection:

- **>db.collection_name.insert()**
- The insert() method creates the collection if it doesn't exist but if the collection is present then it inserts the document into it

EXAMPLE:

- **The field "course" in the example below is an array that holds the several key-value pairs.**

```
>db.students.insert(
  {
    name: "Chaitanya",
    age: 20,
    email: "chaitu@gmail.co.in",
    course: [ { name: "MongoDB", duration: 7 }, { name: "Java", duration: 30 } ]
  }
)
```

Output:

```
WriteResult({ "nInserted" : 1 })
```

b. Insert multiple documents in collection

To insert multiple documents in collection, we define an array of documents and later we use the insert() method on the array variable as shown in the example below. Here we are inserting three documents in the collection named "students". This command will insert the data in "students" collection, if the collection is not present then it will create the collection and insert these documents.

EXAMPLE:

```
>var beginners =
 [
   {
        "StudentId" : 1001,
        "StudentName" : "Steve",
      "age": 30
   },
   {
        "StudentId" : 1002,
        "StudentName" : "Negan",
      "age": 42
   },
   {
        "StudentId" : 3333,
        "StudentName" : "Rick",
      "age": 35
   },
];
db.students.insert(beginners);
```

output:

```
BulkWriteResult({
      "writeErrors" : [ ],
      "writeConcernErrors" : [ ],
      "nInserted" : 3,
      "nUpserted" : 0,
      "nMatched" : 0,
      "nModified" : 0,
      "nRemoved" : 0,
      "upserted" : [ ]
})
```

As you can see that it shows number 3 in front of nInserted. this means that the 3 documents have been inserted by this command.

To verify that the documents are there in collection. Run this command:

db.students.find()

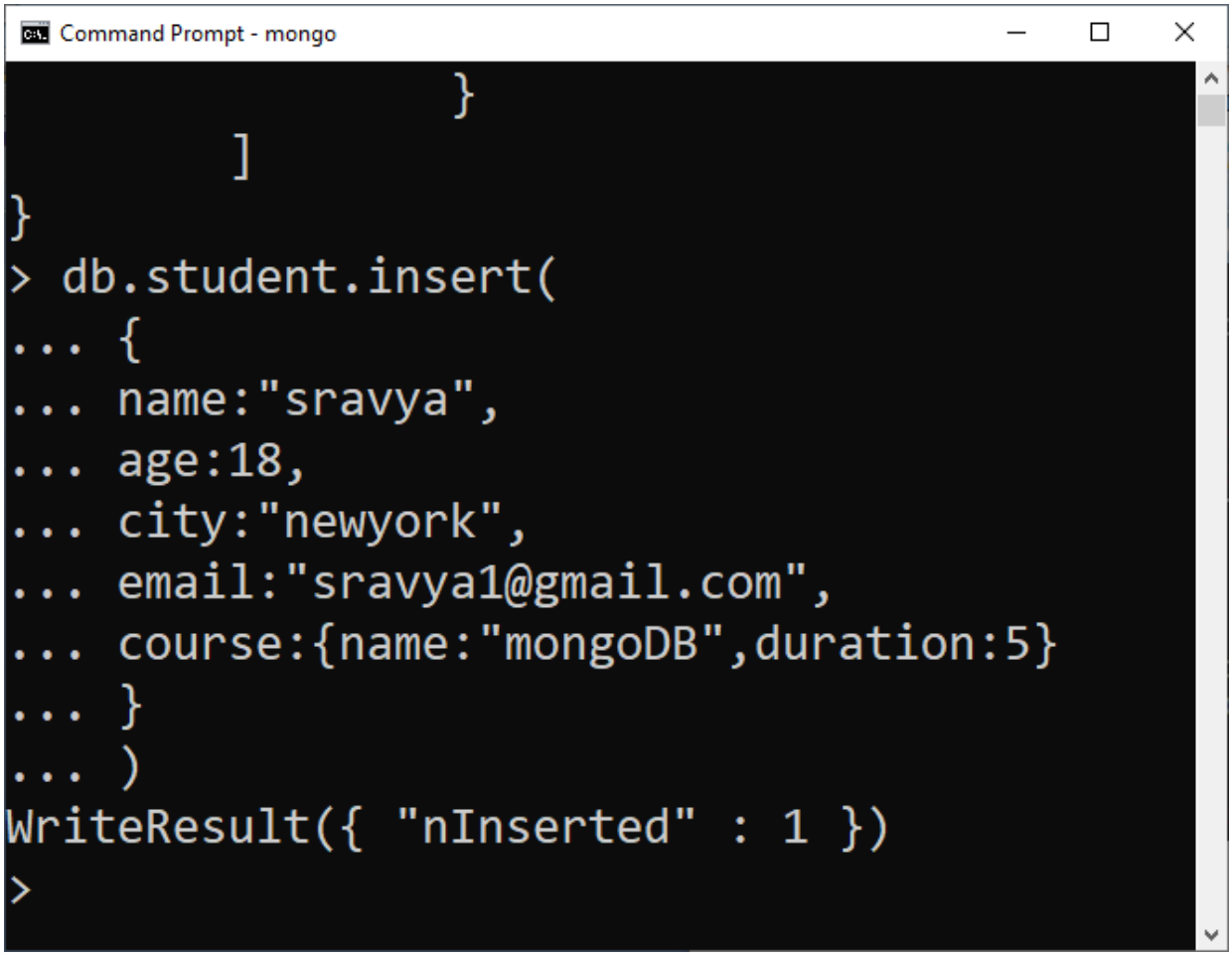
print the output data in a JSON format so that you can read it easily. To print the data in JSON format run the command

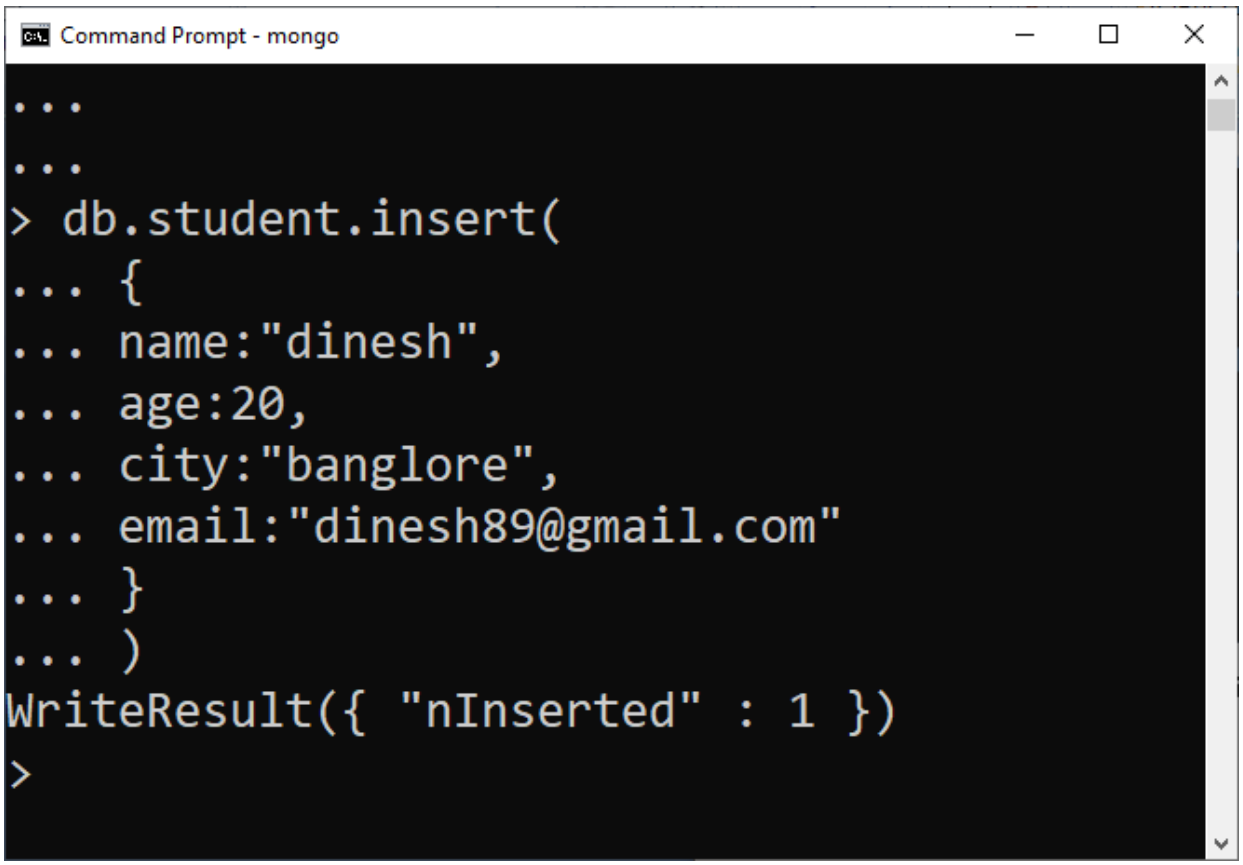
db.collection_name.find().forEach(printjson)

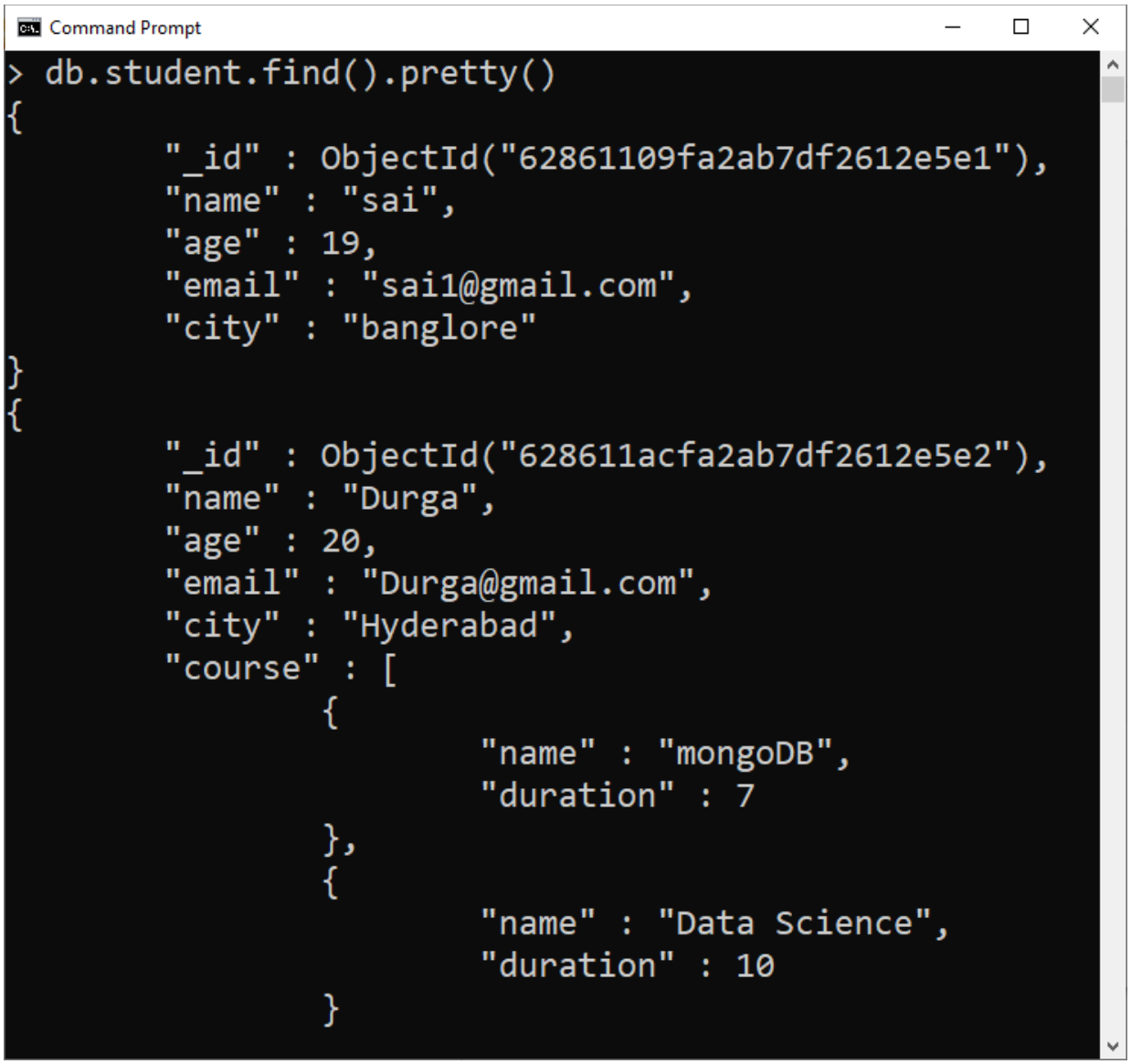
In the screenshot below, you can see the difference. First we have printed the documents using normal find() method and then we printed the documents of same collection using JSON format. The documents in JSON format are neat and easy to read.

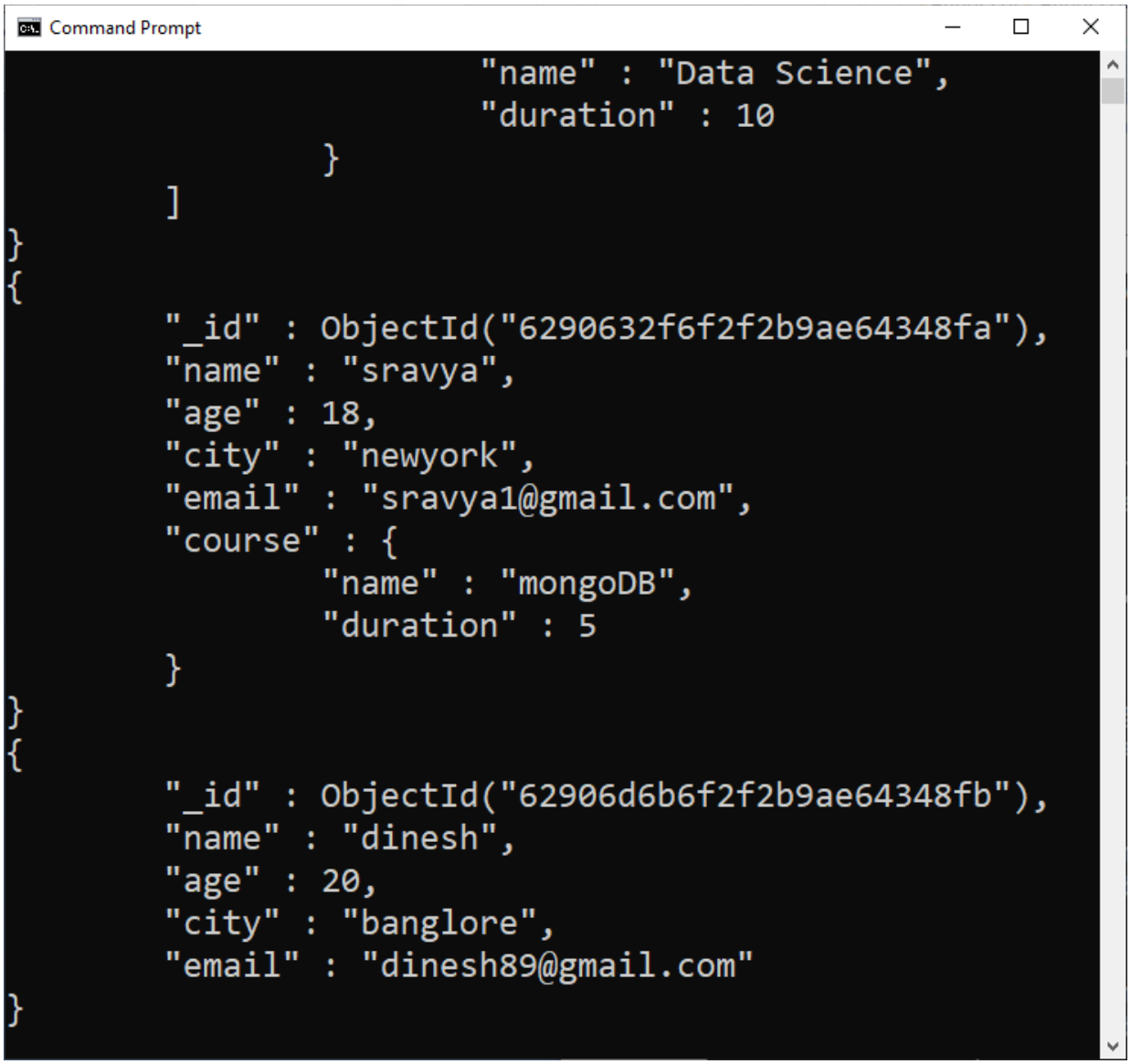
OUTPUT:

a. Insert single document




Command Prompt - mongo
...
...
> db.student.insert(
... {
... name:"dinesh",
... age:20,
... city:"banglore",
... email:"dinesh89@gmail.com"
... }
...)
WriteResult({ "nInserted" : 1 })
>


Command Prompt
> db.student.find().pretty()
{
"_id" : ObjectId("62861109fa2ab7df2612e5e1"),
"name" : "sai",
"age" : 19,
"email" : "sai1@gmail.com",
"city" : "banglore"
}
{
"_id" : ObjectId("628611acfa2ab7df2612e5e2"),
"name" : "Durga",
"age" : 20,
"email" : "Durga@gmail.com",
"city" : "Hyderabad",
"course" : [
{
"name" : "mongoDB",
"duration" : 7
},
{
"name" : "Data Science",
"duration" : 10
}


Command Prompt
"name" : "Data Science",
"duration" : 10
}
]
}
{
"_id" : ObjectId("6290632f6f2f2b9ae64348fa"),
"name" : "sravya",
"age" : 18,
"city" : "newyork",
"email" : "sravya1@gmail.com",
"course" : {
"name" : "mongoDB",
"duration" : 5
}
}
{
"_id" : ObjectId("62906d6b6f2f2b9ae64348fb"),
"name" : "dinesh",
"age" : 20,
"city" : "banglore",
"email" : "dinesh89@gmail.com"
}

b. Insert multiple documents in collection


Command Prompt - mongo
> db.student.find().pretty()
{
"_id" : ObjectId("62906d6b6f2f2b9ae64348fb"),
"name" : "dinesh",
"age" : 20,
"city" : "banglore",
"email" : "dinesh89@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b88349f"),
"roll number" : "20XA51223",
"name" : "aswarya",
"age" : 21
}
{
"_id" : ObjectId("62908a21d42774ae3b8834a0"),
"roll number" : "20X451227",
"name" : "madhuri",
"age" : 20,
"mailid" : "madhuri@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b8834a1"),
"roll number" : "20X451220",
"name" : "sonali",
"age" : 22
}
>

EXPERIMENT 6

Querying all the documents in JSON format & based on the criteria.

- Instead of fetching all the documents from collection, we can fetch selected documents based on a criteria.
    - Equality Criteria:
    - Greater Than Criteria:
    - Less than Criteria:
    - Not Equals Criteria:
    - Greater than equals Criteria:
    - Less than equals Criteria:

Equality Criteria:

Greater Than Criteria

Definition

- **$gt**

SYNTAX: { field: { $gt: value } }

- $gt selects those documents where the value of the field is greater than (i.e. >) the specified value.
- For most data types, comparison operators only perform comparisons on fields where the BSON type matches the query value's type. MongoDB supports limited cross-BSON comparison through Type Bracketing.

JSON Vs BSON

- BSON is just binary JSON(a Superset of JSON with some more data types, most importantly binary byte array).
- It is a serialization format used in MongoDB.

Less than Criteria:

$lt

SYNTAX: **{ field: { $lt: value } }**

(or)

- db.collection_name.find({"field_name":{$lt:criteria_value}}).pretty()

Not Equals Criteria:

$ne

SYNTAX: **{ field: { $ne: value } }**

(Or)

- db.collection_name.find({"field_name":{$ne:criteria_value}}).pretty()

Greater than equals Criteria:

$gte

SYNTAX: { field: {$gte: value } }

(Or)

- **db.collection_name.find({"field_name":{$gte:criteria_value}}).pretty()**

Less than equals Criteria:

$lte

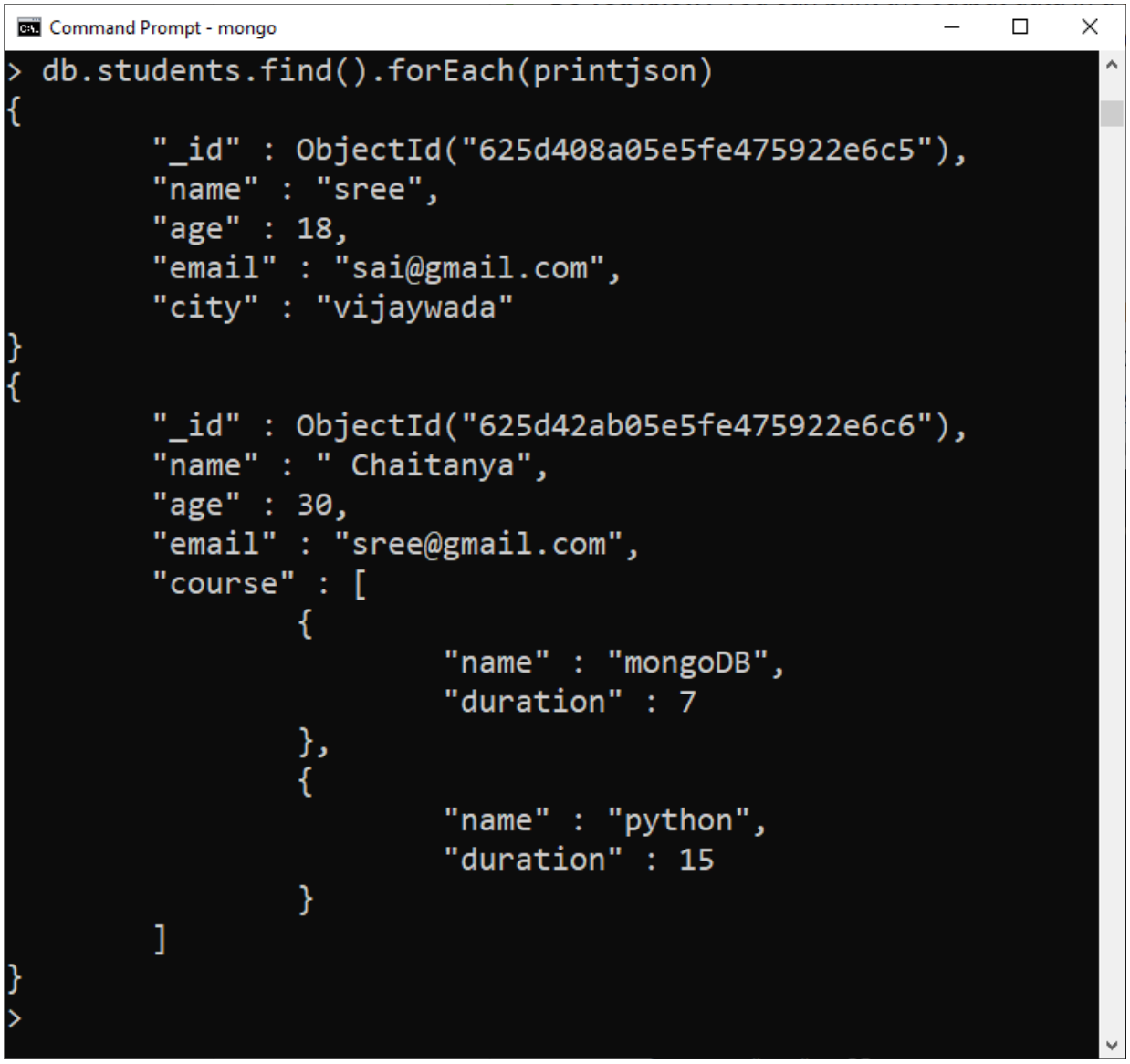
SYNTAX: { field: { $lte: value } }

(Or)

- **db.collection_name.find({"field_name":{$lte:criteria_value}}).pretty()**

OUTPUT:

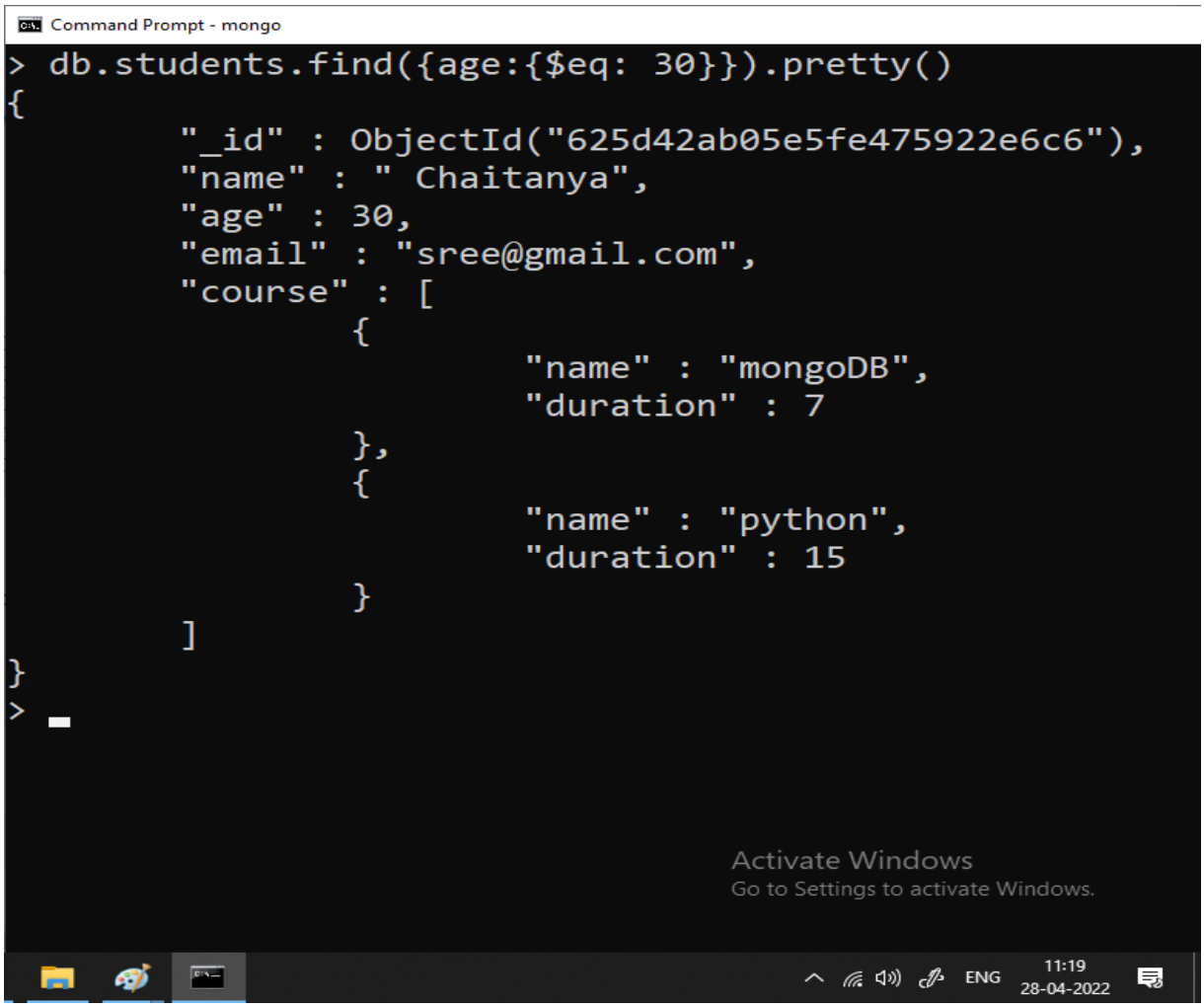
Printing of all documents based on JSON format




Command Prompt - mongo
> db.student.find().pretty()
{
"_id" : ObjectId("62861109fa2ab7df2612e5e1"),
"name" : "sai",
"age" : 19,
"email" : "sai1@gmail.com",
"city" : "banglore"
}
{
"_id" : ObjectId("628611acfa2ab7df2612e5e2"),
"name" : "Durga",
"age" : 20,
"email" : "Durga@gmail.com",
"city" : "Hyderabad",
"course" : [
{
"name" : "mongoDB",
"duration" : 7
},
{
"name" : "Data Science",
"duration" : 10
}
]
}
{
"_id" : ObjectId("6290632f6f2f2b9ae64348fa"),
"name" : "sravya",
"age" : 18,
"city" : "newyork",
"email" : "sravya1@gmail.com",


Command Prompt - mongo
"email" : "sravya1@gmail.com",
"course" : {
"name" : "mongoDB",
"duration" : 5
}
}
{
"_id" : ObjectId("62906d6b6f2f2b9ae64348fb"),
"name" : "dinesh",
"age" : 20,
"city" : "banglore",
"email" : "dinesh89@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b88349f"),
"roll number" : "20XA51223",
"name" : "aswarya",
"age" : 21
}
{
"_id" : ObjectId("62908a21d42774ae3b8834a0"),
"roll number" : "20X451227",
"name" : "madhuri",
"age" : 20,
"mailid" : "madhuri@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b8834a1"),
"roll number" : "20X451220",
"name" : "sonali",
"age" : 22
}
>

Printing all documents based on Querry



EXPERIMENT 7

MongoDB Update document

a) **using update()method.**
b) **using save() method.**

SYNTAX:

> db.collection_name.update(criteria, update_data)

EXAMPLE:

>db.studnets.update({“name”:”sai”},{$set:{“name”:”sree”}})

a) To update multiple documents with the update() method:

SYNTAX:

> db.collection_name.update(criteria, update_data, {multi:true})

EXAMPLE:

>db.studnets.update({“name”:”sai”},{$set:{“name”:”sree”}},{multi:true})

b) To Update a document using save() method:

SYNTAX:

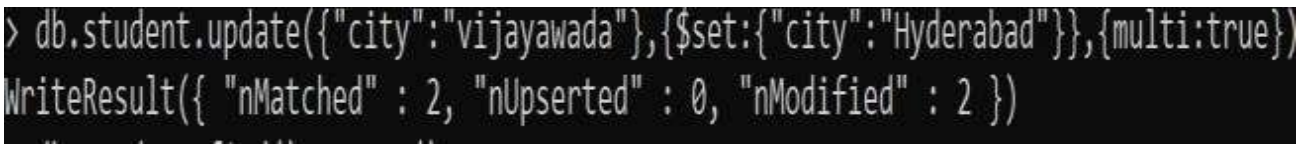
> db.collection_name.save({_id:ObjectId(), new_document})

- To work with save() method you should know the unique _id field of that document.
- A very important point to note is that when you do not provide the _id field while using save()
  method, it calls insert() method and the passed document is inserted into the collection as a new document
- To get the _id of a document, you can either type this command:
- db.students.find().pretty()

OUTPUT:

a) Update a single document using update()

To update multiple documents with the update() method:



Retrieving a document using name field

```
> db.student.find({"Name":"Maha Lakshmi"}).pretty()
{
        "_id" : ObjectId("6288a71f113dfe1c4623c260"),
        "Name" : "Maha Lakshmi",
        "age" : 17,
        "city" : "newyork"
}
```

b) Update a document using save() method:

```
> db.student.save({"_id":ObjectId("123456712345678901234567"),"name":"d
ivya","age":18})
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("123456712345678901234567")
})
```

EXPERIMENT 8

MongoDB Delete Document from a Collection

a) **Delete Document using remove() method**
b) **Remove only one document matching your criteria**
c) **Remove all documents**

a) Delete Document using remove() method

The remove() method is used for removing the documents from a collection in MongoDB.

SYNTAX of remove() method:

>db.collection_name.remove(delete_criteria)

EXAMPLE:

```
> db.students.find().pretty()
{
        "_id" : ObjectId("59bcecc7668dcce02aaa6fed"),
        "StudentId" : 1001,
        "StudentName" : "Steve",
        "age" : 30
}
{
        "_id" : ObjectId("59bcecc7668dcce02aaa6fef"),
        "StudentId" : 3333,
        "StudentName" : "Rick",
        "age" : 35
}
```

To remove the student from this collection who has a student id equal to 3333. To do this write a command using remove() method like this:

db.students.remove({"StudentId": 3333})

Output:

WriteResult({ "nRemoved" : 1 })

b) Remove only one document matching your criteria

When there are more than one documents present in collection that matches the criteria then all those documents will be deleted if you run the remove command. However there is a way to limit the deletion to only one document so that even if there are more documents matching the deletion criteria, only one document will be deleted.

SYNTAX:

>db.collection_name.remove(delete_criteria, justOne)

Here justOne is a Boolean parameter that takes only 1 and 0, if you give 1 then it will limit the the document deletion to only 1 document. This is an optional parameters as we have seen above that we have used the remove() method without using this parameter.

EXAMPLE: The following are records in collection.

```
> db.walkingdead.find().pretty()
{
        "_id" : ObjectId("59bf280cb8e797a22c654229"),
        "name" : "Rick Grimes",
        "age" : 32,
        "rname" : "Andrew Lincoln"
}
{
        "_id" : ObjectId("59bf2851b8e797a22c65422a"),
        "name" : "Negan",
```

"age" : 35,

"rname" : "Jeffrey Dean Morgan"

}

{

"_id" : ObjectId("59bf28a5b8e797a22c65422b"),

"name" : "Daryl Dixon",

"age" : 32,

"rname" : "Norman Reedus"

}

To remove the document that has age equal to 32. There are two documents in this collection that are matching this criteria. However to limit the deletion to one we are setting justOne parameter to true.

EXAMPLE:

db.walkingdead.remove({"age": 32}, 1)

Output: As you can see only one document got deleted.

WriteResult({ "nRemoved" : 1 })

c) Remove all documents

To remove all the documents from a collection but does not want to remove the collection itself then you can use remove() method like this:

SYNTAX:

>db.collection_name.remove({})

Drop collection in MongoDB:

To drop a collection , first connect to the database in which you want to delete collection and then type the following command to delete the collection:

>db.collection_name.drop()

Note: Once you drop a collection all the documents and the indexes associated with them will also be dropped. To preserve the indexes we use remove() function that only removes the documents in the collection but doesn't remove the collection itself and the indexes created on it.

EXAMPLE:

> use madavi

switched to db madavi

> show collections

admin

students

teachers

> db.teachers.drop()

true

> show collections

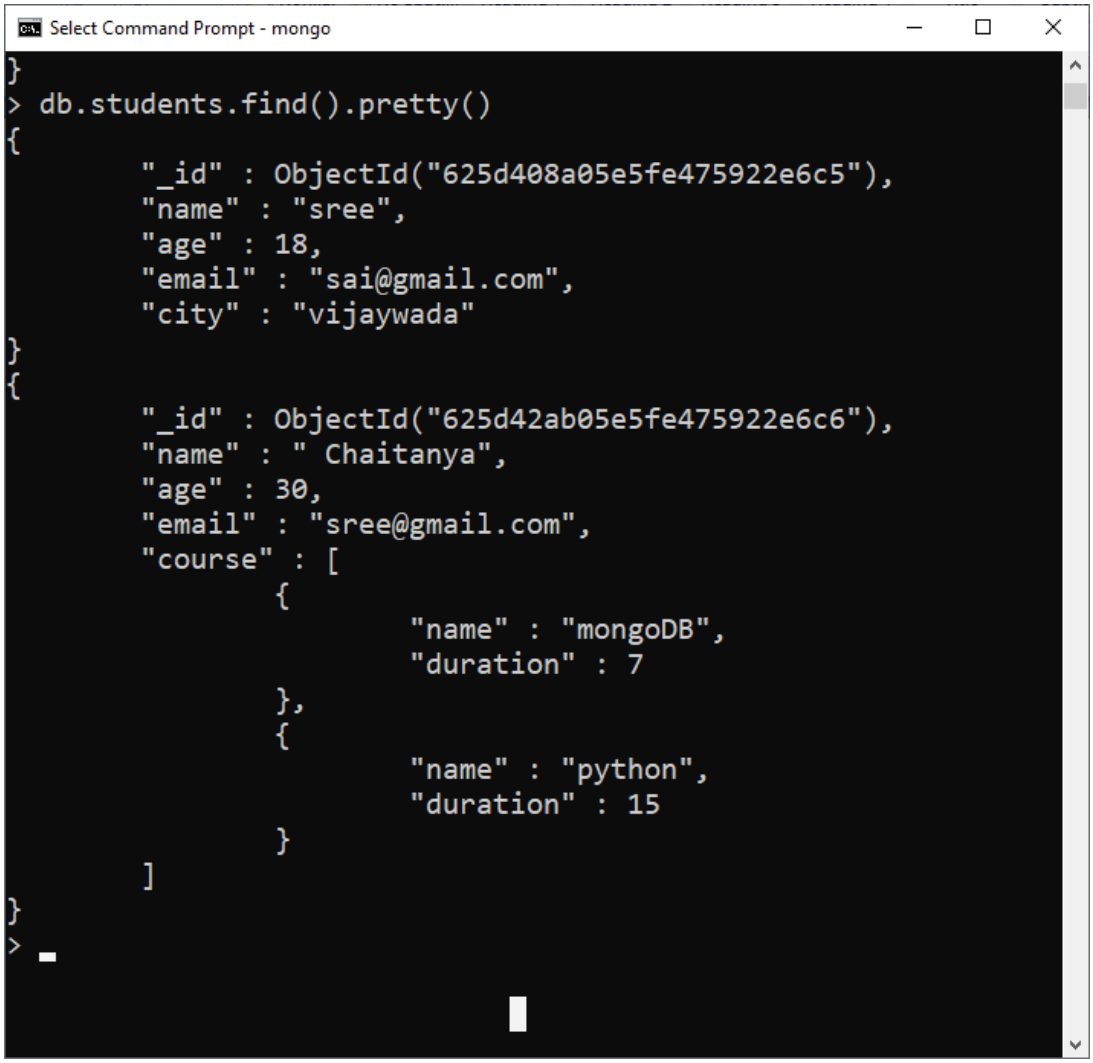
admin

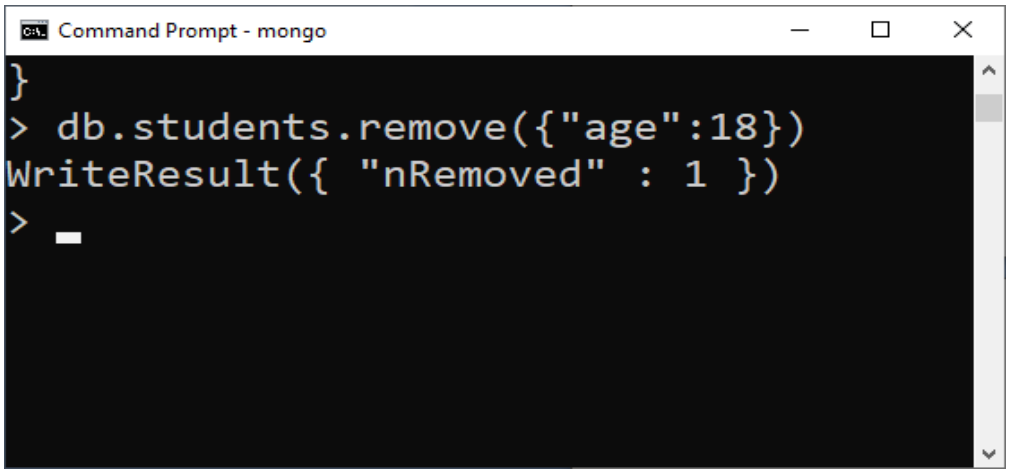
students

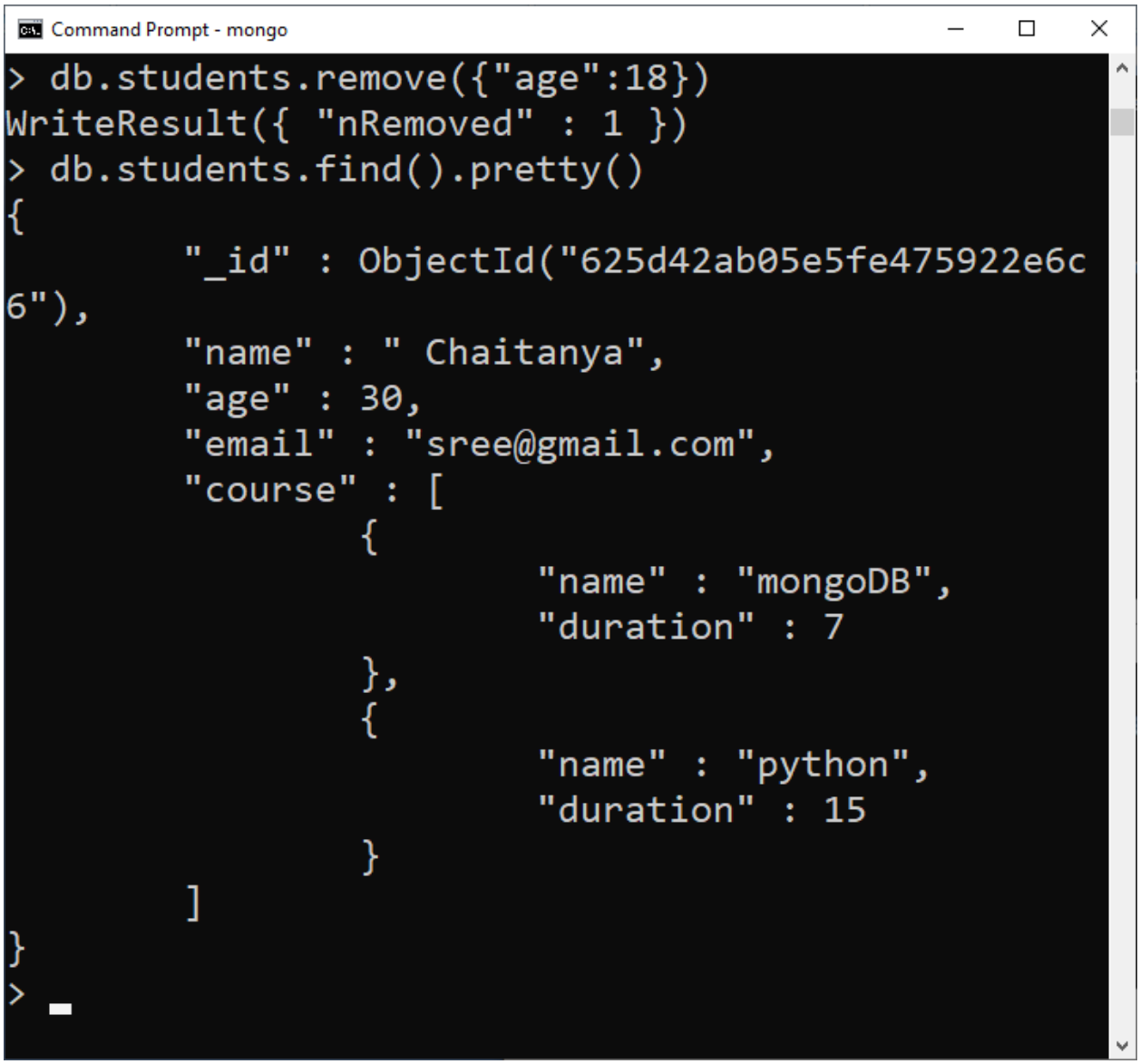
The command db.teachers.drop() returned true which means that the collection is deleted successfully. The same thing we have verified using the show collections command after deletion as shown above.

OUTPUT:

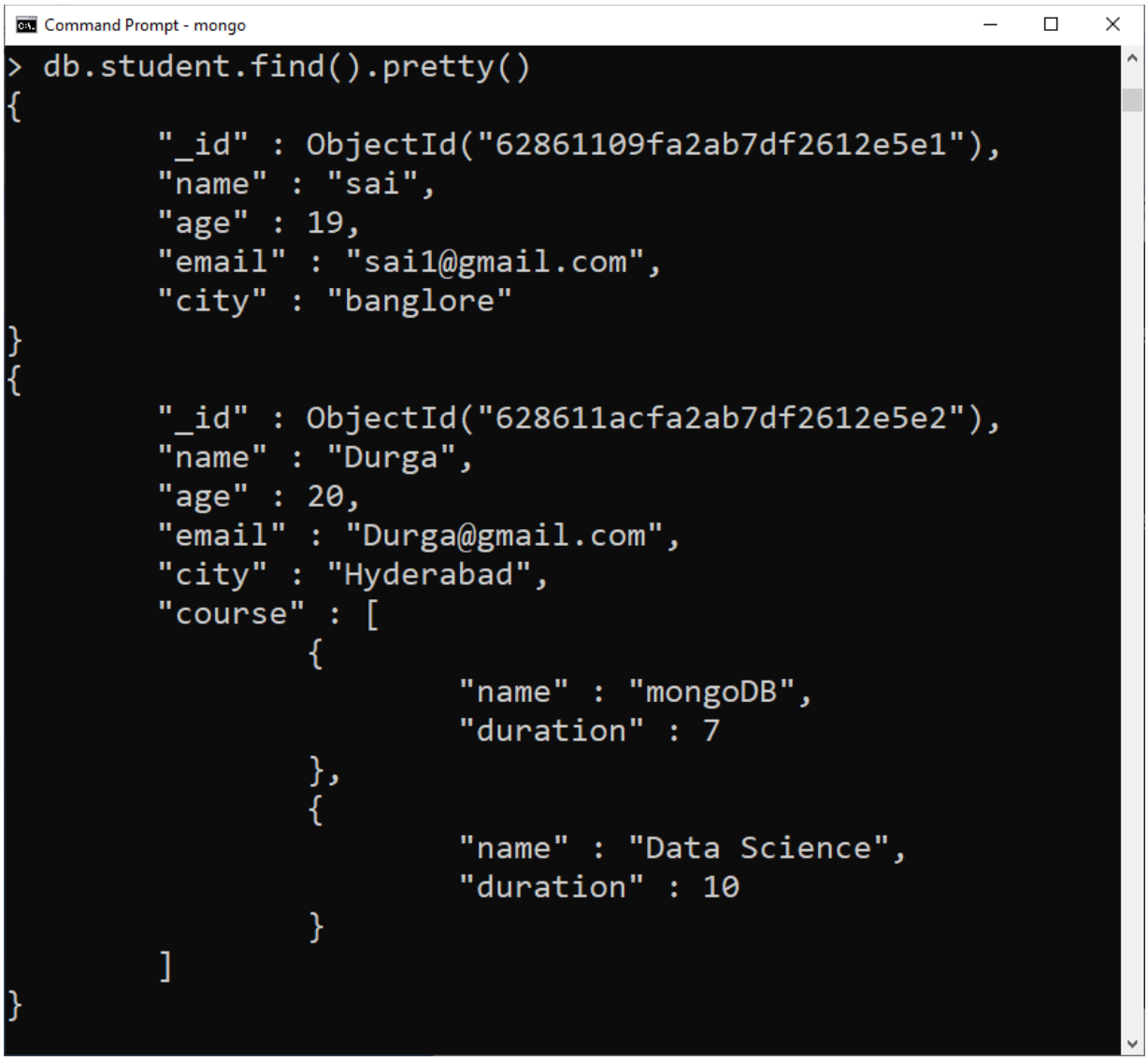
a) Delete Document using remove() method



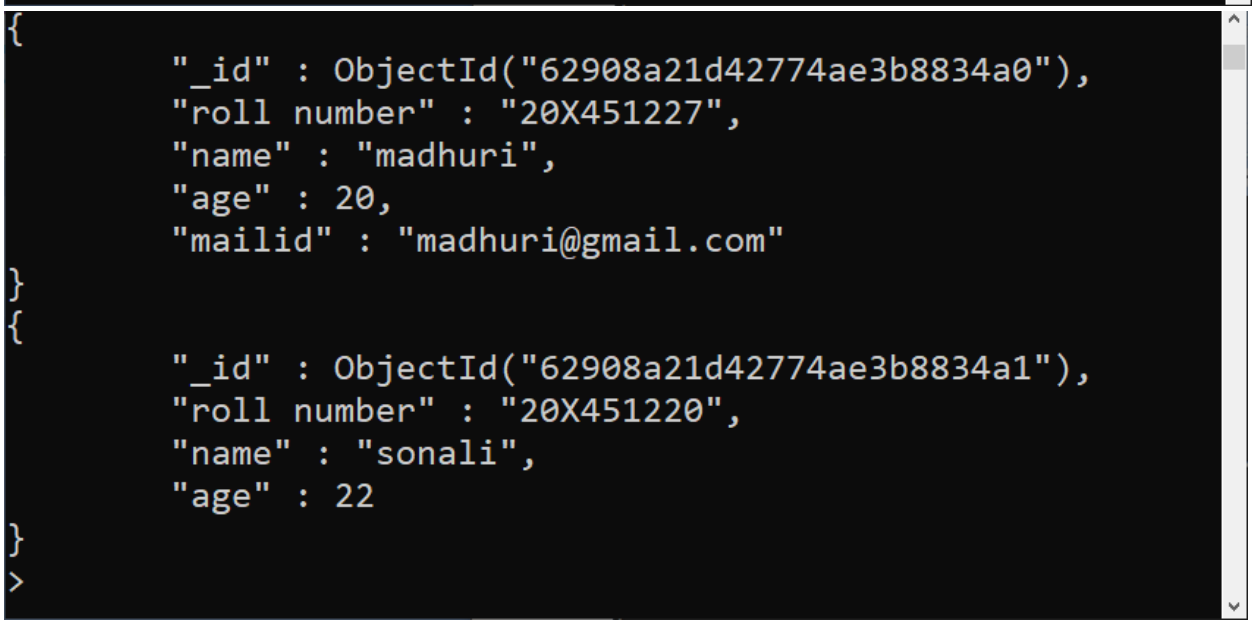


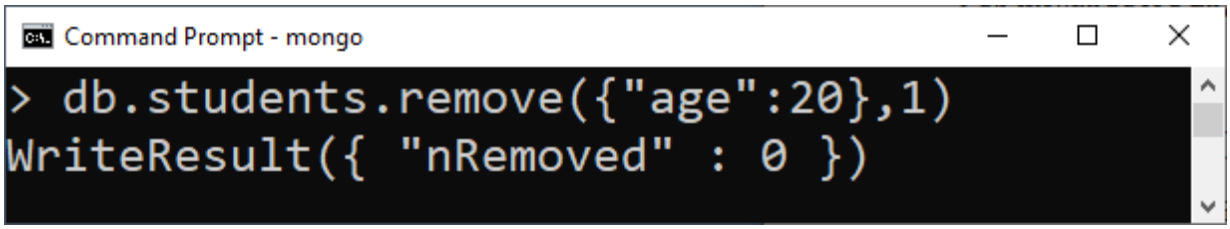


b) Remove only one document matching your criteria

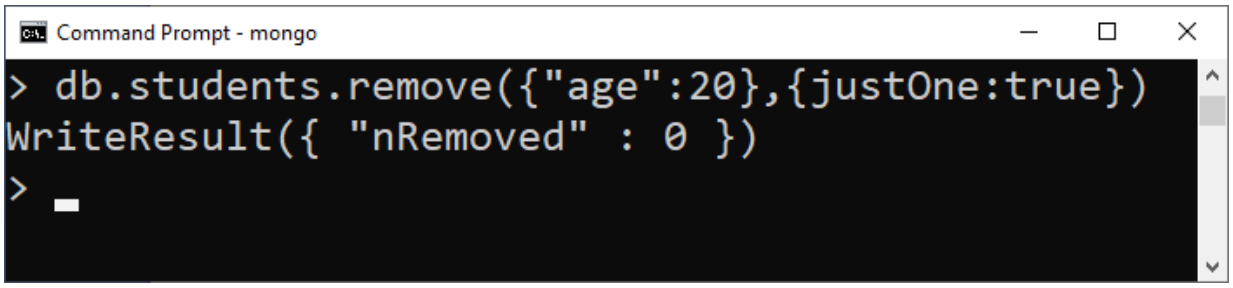

Command Prompt - mongo
> db.student.find().pretty()
{
"_id" : ObjectId("62861109fa2ab7df2612e5e1"),
"name" : "sai",
"age" : 19,
"email" : "sai1@gmail.com",
"city" : "banglore"
}
{
"_id" : ObjectId("628611acfa2ab7df2612e5e2"),
"name" : "Durga",
"age" : 20,
"email" : "Durga@gmail.com",
"city" : "Hyderabad",
"course" : [
{
"name" : "mongoDB",
"duration" : 7
},
{
"name" : "Data Science",
"duration" : 10
}
]
}

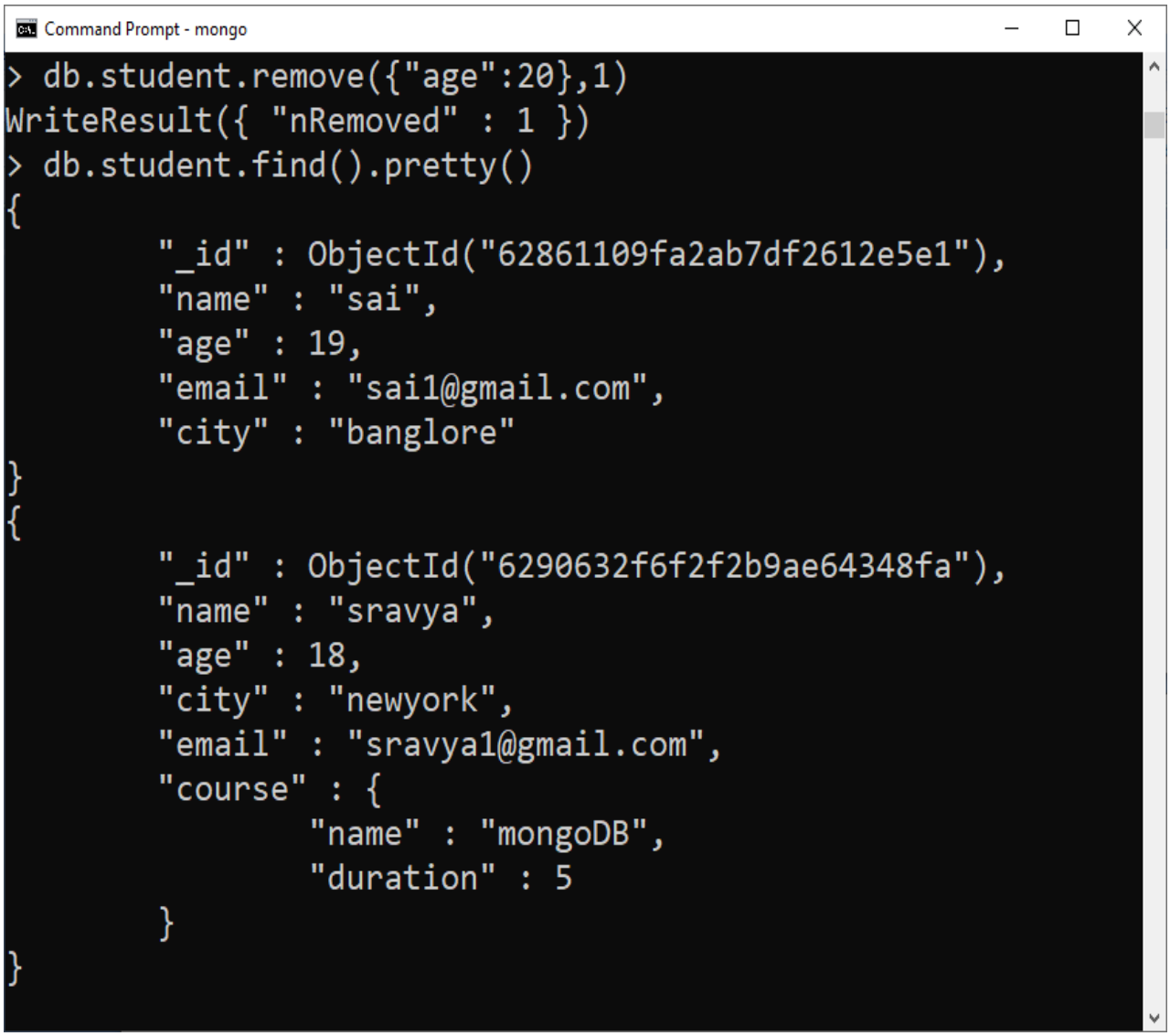

Command Prompt - mongo
{
"_id" : ObjectId("6290632f6f2f2b9ae64348fa"),
"name" : "sravya",
"age" : 18,
"city" : "newyork",
"email" : "sravya1@gmail.com",
"course" : {
"name" : "mongoDB",
"duration" : 5
}
}
{
"_id" : ObjectId("62906d6b6f2f2b9ae64348fb"),
"name" : "dinesh",
"age" : 20,
"city" : "banglore",
"email" : "dinesh89@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b88349f"),
"roll number" : "20XA51223",
"name" : "aswarya",
"age" : 21
}
{

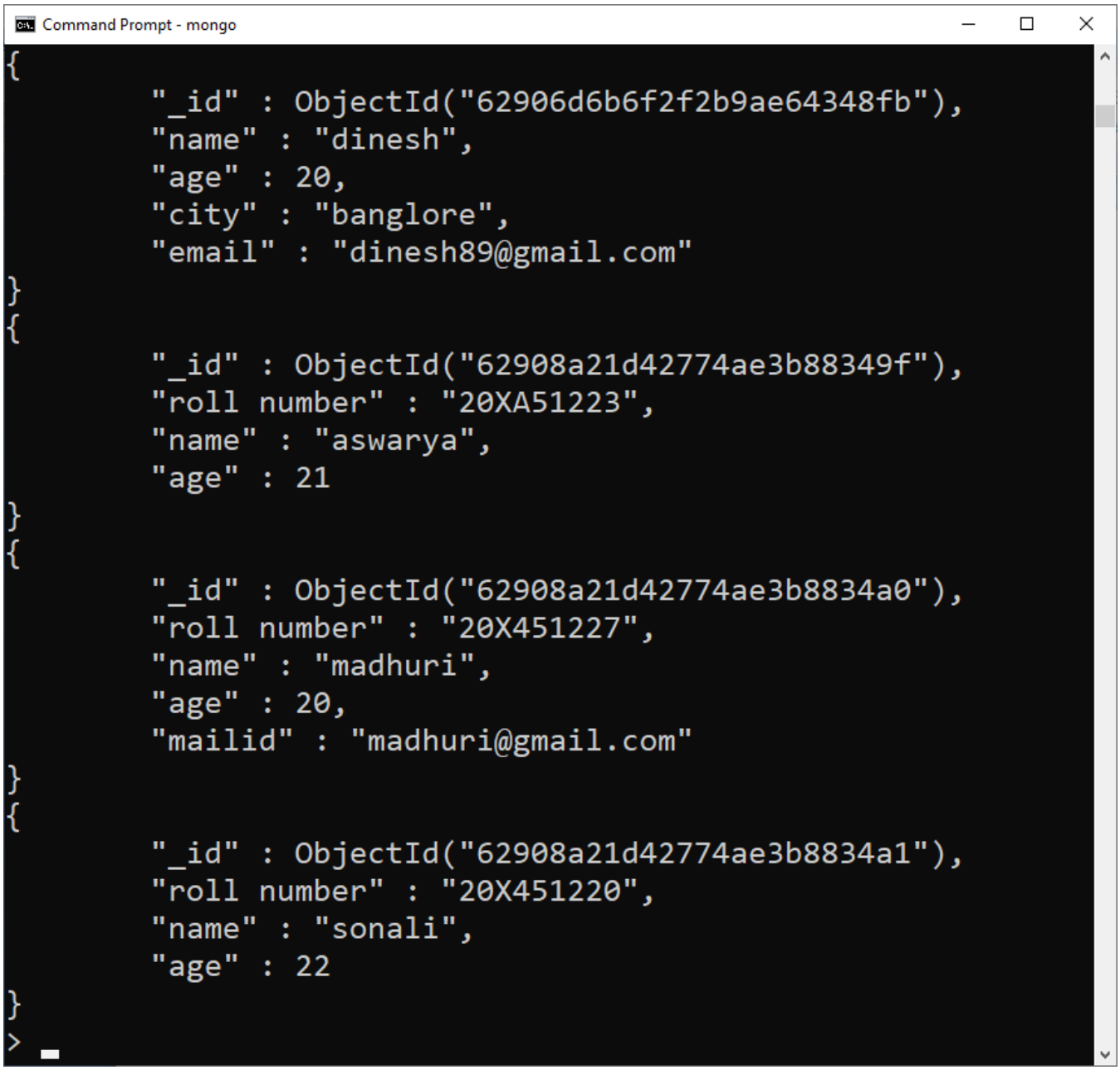

{
"_id" : ObjectId("62908a21d42774ae3b8834a0"),
"roll number" : "20X451227",
"name" : "madhuri",
"age" : 20,
"mailid" : "madhuri@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b8834a1"),
"roll number" : "20X451220",
"name" : "sonali",
"age" : 22
}
>


Command Prompt - mongo
> db.students.remove({"age":20},1)
WriteResult({ "nRemoved" : 0 })

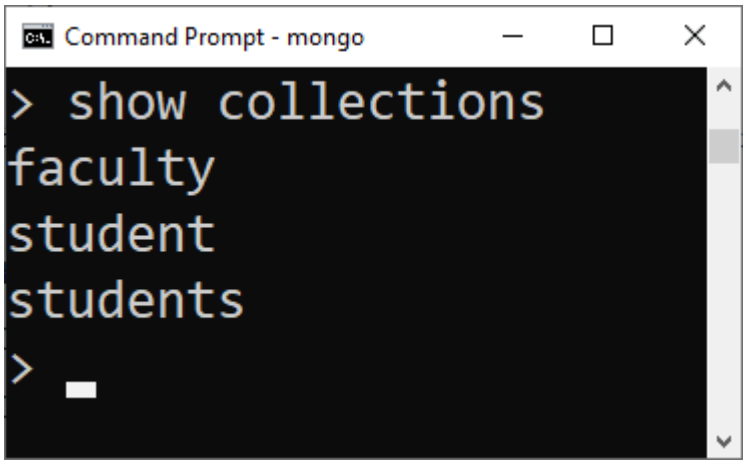
(Or)

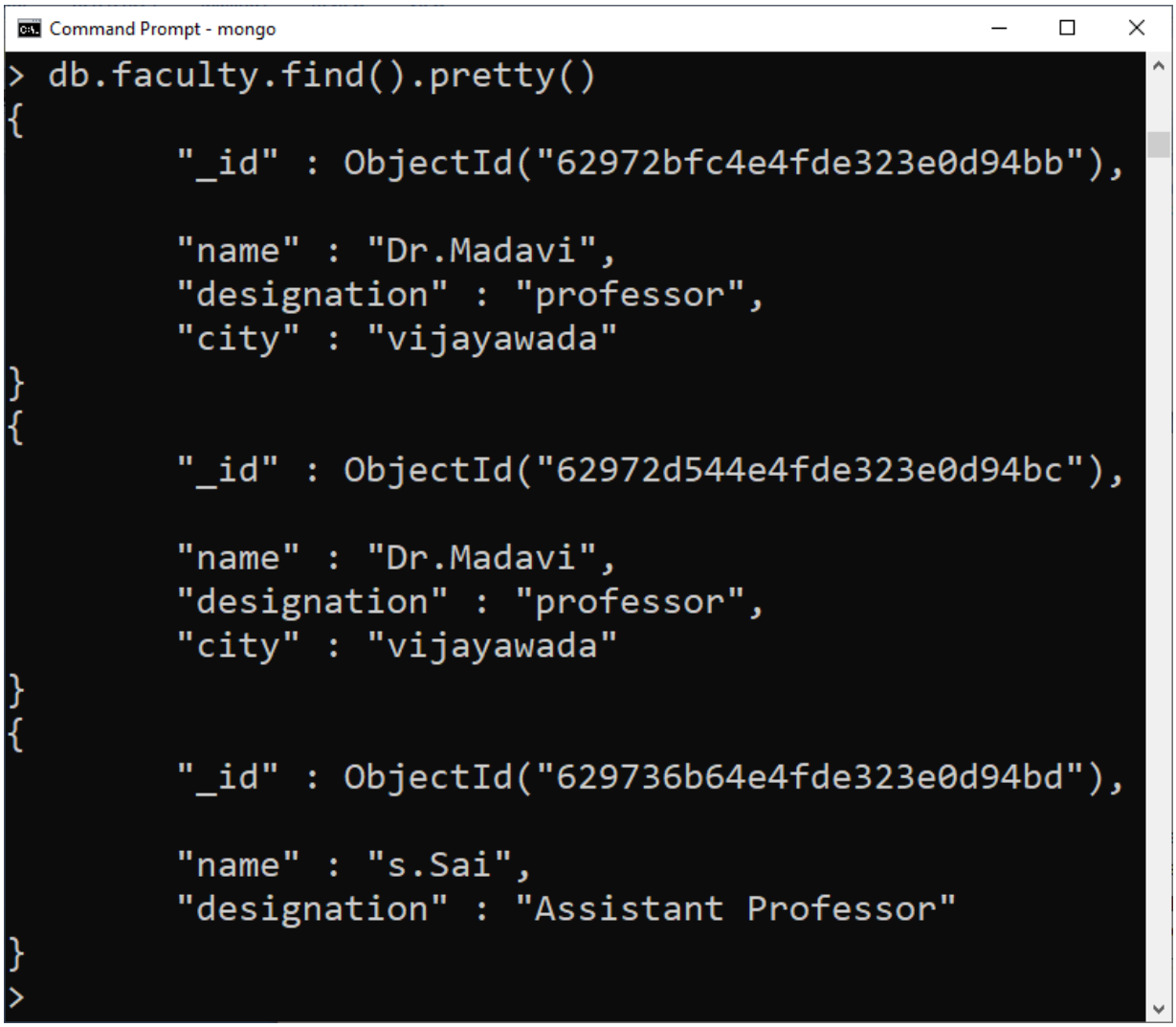

Command Prompt - mongo
> db.students.remove({"age":20},{justOne:true})
WriteResult({ "nRemoved" : 0 })
>

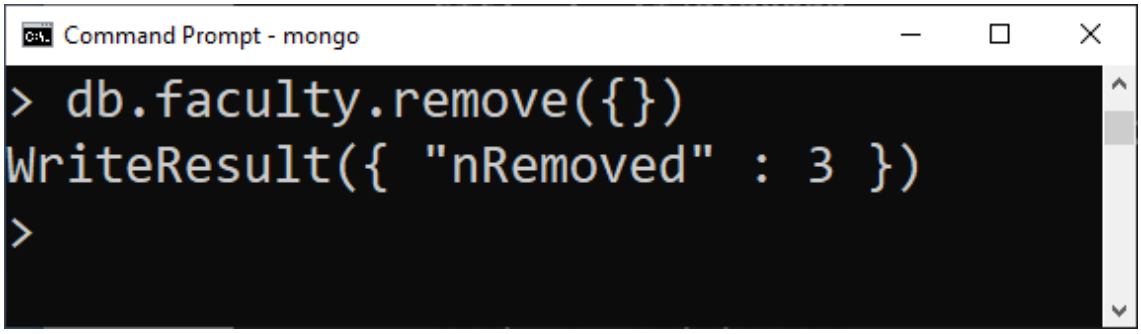

Command Prompt - mongo
> db.student.remove({"age":20},1)
WriteResult({ "nRemoved" : 1 })
> db.student.find().pretty()
{
"_id" : ObjectId("62861109fa2ab7df2612e5e1"),
"name" : "sai",
"age" : 19,
"email" : "sai1@gmail.com",
"city" : "banglore"
}
{
"_id" : ObjectId("6290632f6f2f2b9ae64348fa"),
"name" : "sravya",
"age" : 18,
"city" : "newyork",
"email" : "sravya1@gmail.com",
"course" : {
"name" : "mongoDB",
"duration" : 5
}
}


Command Prompt - mongo
{
"_id" : ObjectId("62906d6b6f2f2b9ae64348fb"),
"name" : "dinesh",
"age" : 20,
"city" : "banglore",
"email" : "dinesh89@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b88349f"),
"roll number" : "20XA51223",
"name" : "aswarya",
"age" : 21
}
{
"_id" : ObjectId("62908a21d42774ae3b8834a0"),
"roll number" : "20X451227",
"name" : "madhuri",
"age" : 20,
"mailid" : "madhuri@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b8834a1"),
"roll number" : "20X451220",
"name" : "sonali",
"age" : 22
}
>

c) Remove all documents






Command Prompt - mongo
> db.faculty.remove({})
WriteResult({ "nRemoved" : 3 })
>

EXPERIMENT 9

MongoDB Projection

MongoDB Projection is used when we want to get the selected fields of the documents rather than all fields.

For example, we have a collection where we have stored documents that have the fields: student_name, student_id, student_age but we want to see only the student_id of all the students then in that case we can use projection to get only the student_id.

SYNTAX:

>db.collection_name.find({},{field_key:1 or 0})

EXAMPLE:

> db.studentdata.find().pretty()

{

"_id" : ObjectId("59bf63380be1d7770c3982af"),

"student_name" : "Steve",

"student_id" : 2002,

"student_age" : 22

}

{

"_id" : ObjectId("59bf63500be1d7770c3982b0"),
"student_name" : "Carol",

"student_id" : 2003,

"student_age" : 22

}

{

"_id" : ObjectId("59bf63650be1d7770c3982b1"),

"student_name" : "Tim",

"student_id" : 2004,

"student_age" : 23

}

To get only the student_id for all the documents, we will use the Projection like this:

> db.studentdata.find({}, {"_id": 0, "student_id": 1})

{ "student_id" : 2002 }

{ "student_id" : 2003 }

{ "student_id" : 2004 }

Value 1 means show that field and 0 means do not show that field. When we set a field to 1 in Projection other fields are automatically set to 0, except _id, so to avoid the _id we need to specifically set it to 0 in projection. The vice versa is also true when we set few fields to 0, other fields set to 1 automatically.

Another way of doing the same thing:

> db.studentdata.find({}, {"_id": 0, "student_name": 0, "student_age": 0})

{ "student_id" : 2002 }

{ "student_id" : 2003 }

{ "student_id" : 2004 }

IMPORTANT NOTE:

Some of you may get this error while using Projection in Query:

Error: error: {

"ok" : 0,

"errmsg" : "Projection cannot have a mix of inclusion and exclusion.",

"code" : 2,

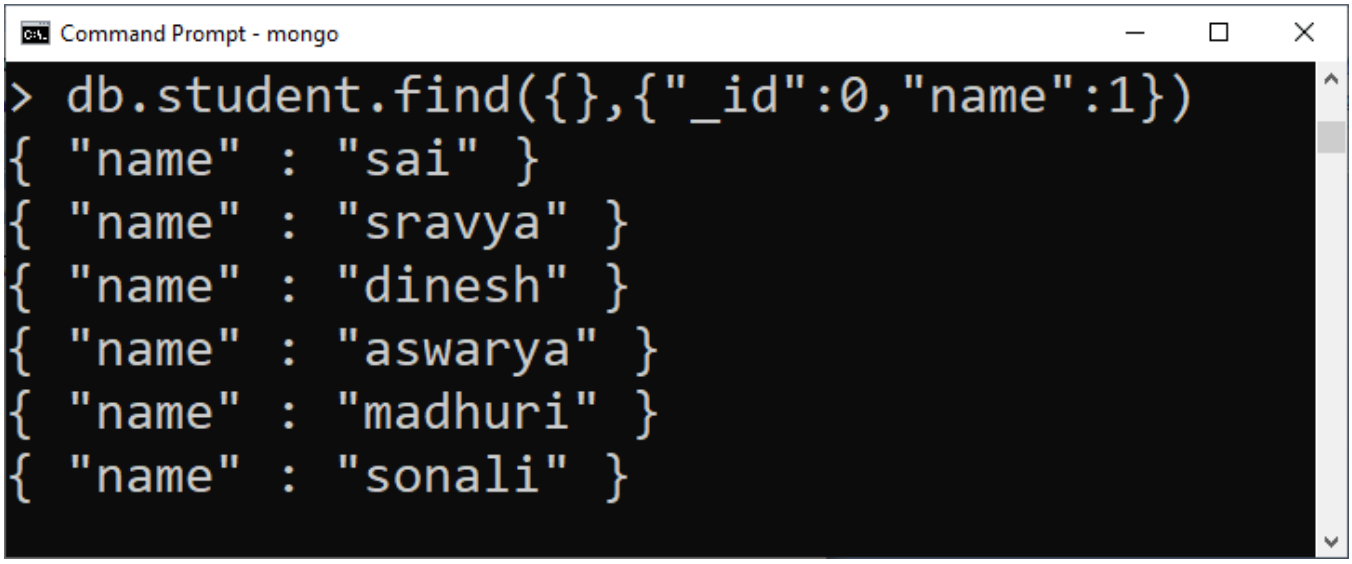
"codeName" : "BadValue"

}

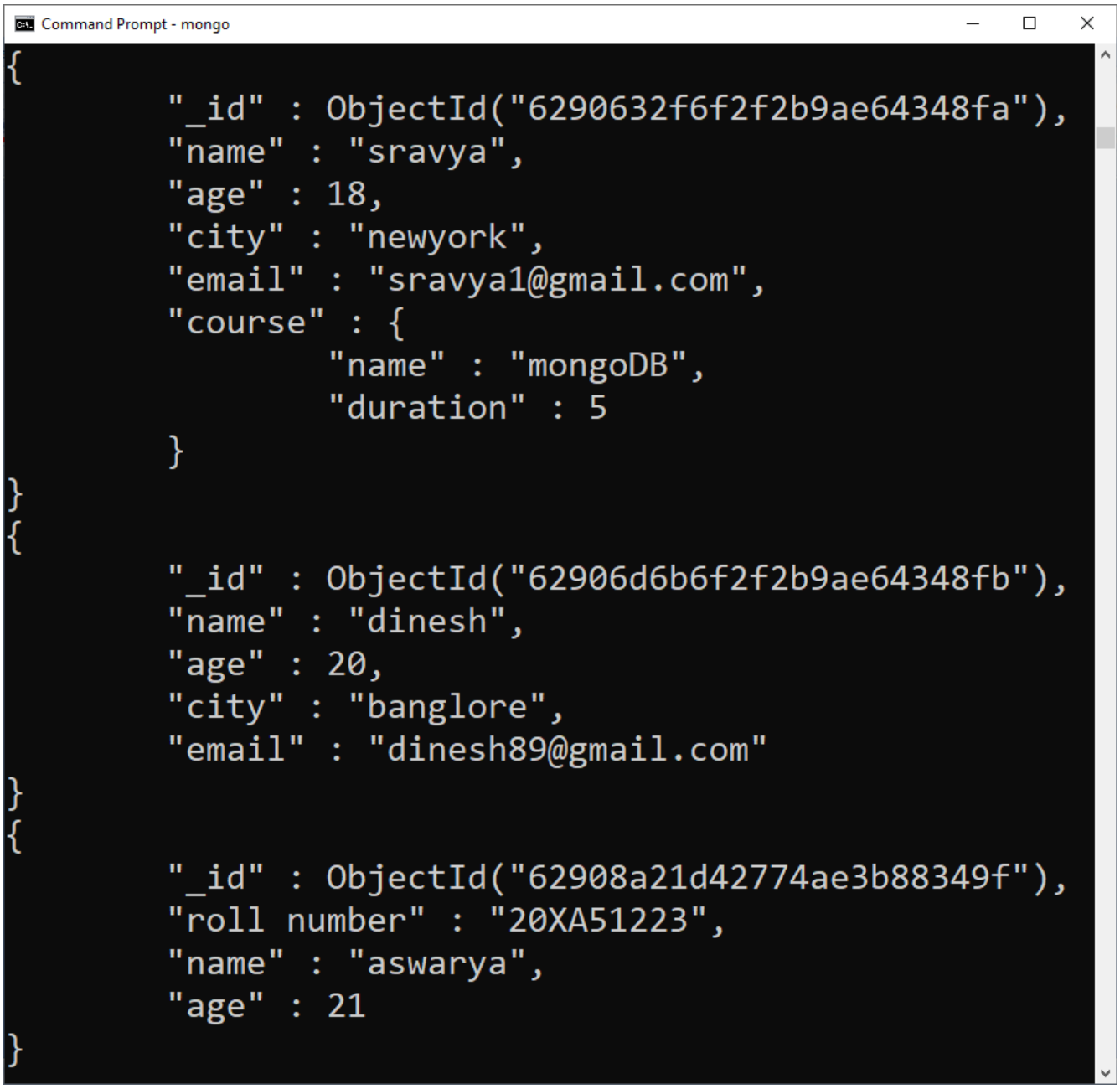
This happens when you set some fields to 0 and other to 1, in other words you mix inclusion and exclusion, the only exception is the _id field. for example: The following Query would produce this error:
db.studentdata.find({}, {"_id": 0, "student_name": 0, "student_age": 1})

This is because we have set student_name to 0 and other field student_age to 1. We can't mix these. You either set those fields that you don't want to display to 0 or set the fields to 1 that you want to display.

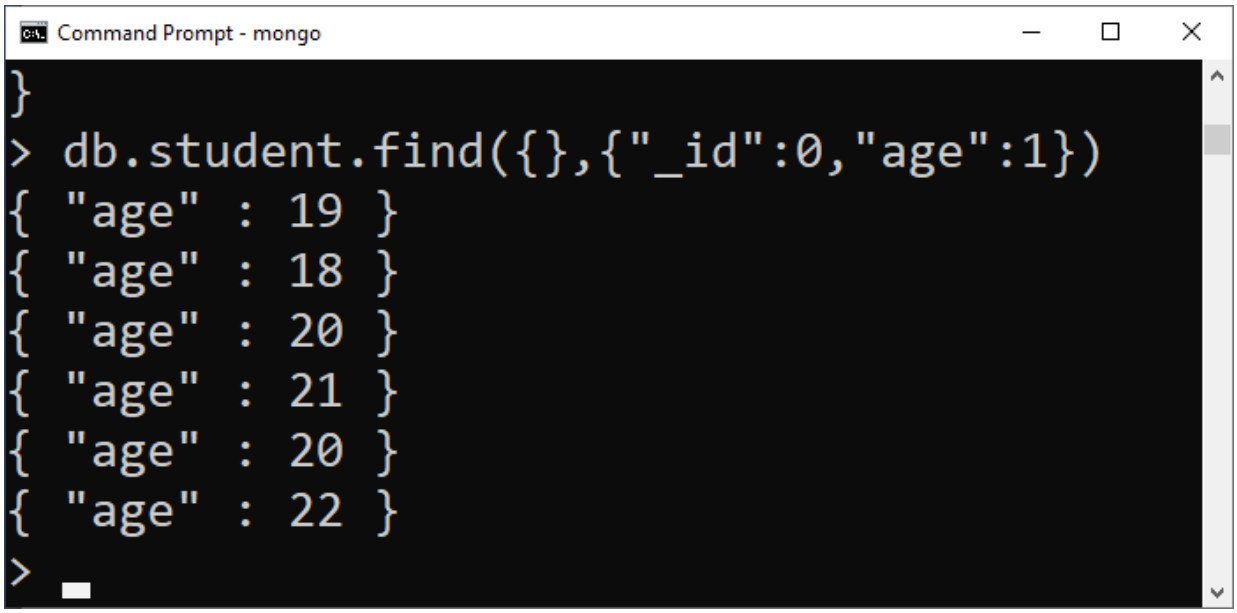
OUTPUT:


Command Prompt - mongo
> db.student.find({},{"_id":0,"name":1})
{ "name" : "sai" }
{ "name" : "sravya" }
{ "name" : "dinesh" }
{ "name" : "aswarya" }
{ "name" : "madhuri" }
{ "name" : "sonali" }


Command Prompt - mongo
> db.student.find({},{"_id":0,"roll number":1})
{ }
{ }
{ }
{ "roll number" : "20XA51223" }
{ "roll number" : "20X451227" }
{ "roll number" : "20X451220" }
> db.student.find().pretty()
{
"_id" : ObjectId("62861109fa2ab7df2612e5e1"),
"name" : "sai",
"age" : 19,
"email" : "sai1@gmail.com",
"city" : "banglore"
}


Command Prompt - mongo
{
"_id" : ObjectId("6290632f6f2f2b9ae64348fa"),
"name" : "sravya",
"age" : 18,
"city" : "newyork",
"email" : "sravya1@gmail.com",
"course" : {
"name" : "mongoDB",
"duration" : 5
}
}
{
"_id" : ObjectId("62906d6b6f2f2b9ae64348fb"),
"name" : "dinesh",
"age" : 20,
"city" : "banglore",
"email" : "dinesh89@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b88349f"),
"roll number" : "20XA51223",
"name" : "aswarya",
"age" : 21
}


Command Prompt - mongo
{
"_id" : ObjectId("62908a21d42774ae3b8834a0"),
"roll number" : "20X451227",
"name" : "madhuri",
"age" : 20,
"mailid" : "madhuri@gmail.com"
}
{
"_id" : ObjectId("62908a21d42774ae3b8834a1"),
"roll number" : "20X451220",
"name" : "sonali",
"age" : 22
}
>


Command Prompt - mongo
}
> db.student.find({},{"_id":0,"age":1})
{ "age" : 19 }
{ "age" : 18 }
{ "age" : 20 }
{ "age" : 21 }
{ "age" : 20 }
{ "age" : 22 }
>

Exercise-10

limit(), skip(), sort() methods in Mongo DB

AIM:

The limit () Method

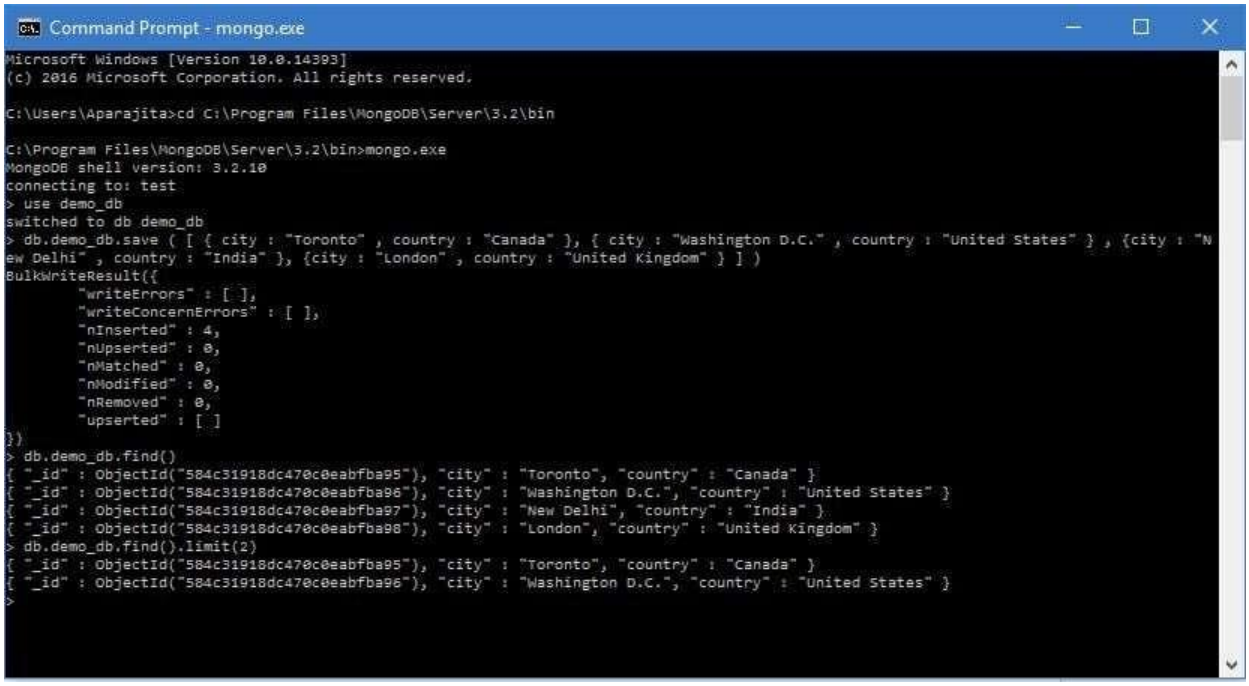
In MongoDB, the 'limit ()' method is used to limit the records or documents present inside a collection. It accepts only one argument which is of number type. Depending on the value of the number, we can limit the number of documents to be displayed. This argument is an optional field inside the 'limit ()' method, and when not specified then by default, it will display all the documents from the collection.

The following is the basic syntax for 'limit ()' method in MongoDB.

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

Program:

```
> use  demo_db
switched to db demo_db
> db.demo_db.save ( [ { city : "Toronto" , country : "Canada" }, { city : "Washington D.C." , country : "United States" } , {city : "New Delhi" , country : "India" }, {city : "London" , country : "United Kingdom" } ] )
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 4,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
> db.demo_db.find()
{ "_id" : ObjectId("584c31918dc470c0eabfba95"), "city" : "Toronto", "country" : "Canada" }
{ "_id" : ObjectId("584c31918dc470c0eabfba96"), "city" : "Washington D.C.", "country" : "United States" }
{ "_id" : ObjectId("584c31918dc470c0eabfba97"), "city" : "New Delhi", "country" : "India" }
{ "_id" : ObjectId("584c31918dc470c0eabfba98"), "city" : "London", "country" : "United Kingdom" }
> db.demo_db.find().limit(2)
{ "_id" : ObjectId("584c31918dc470c0eabfba95"), "city" : "Toronto", "country" : "Canada" }
{ "_id" : ObjectId("584c31918dc470c0eabfba96"), "city" : "Washington D.C.", "country" : "United States" }
>
```



The skip () Method

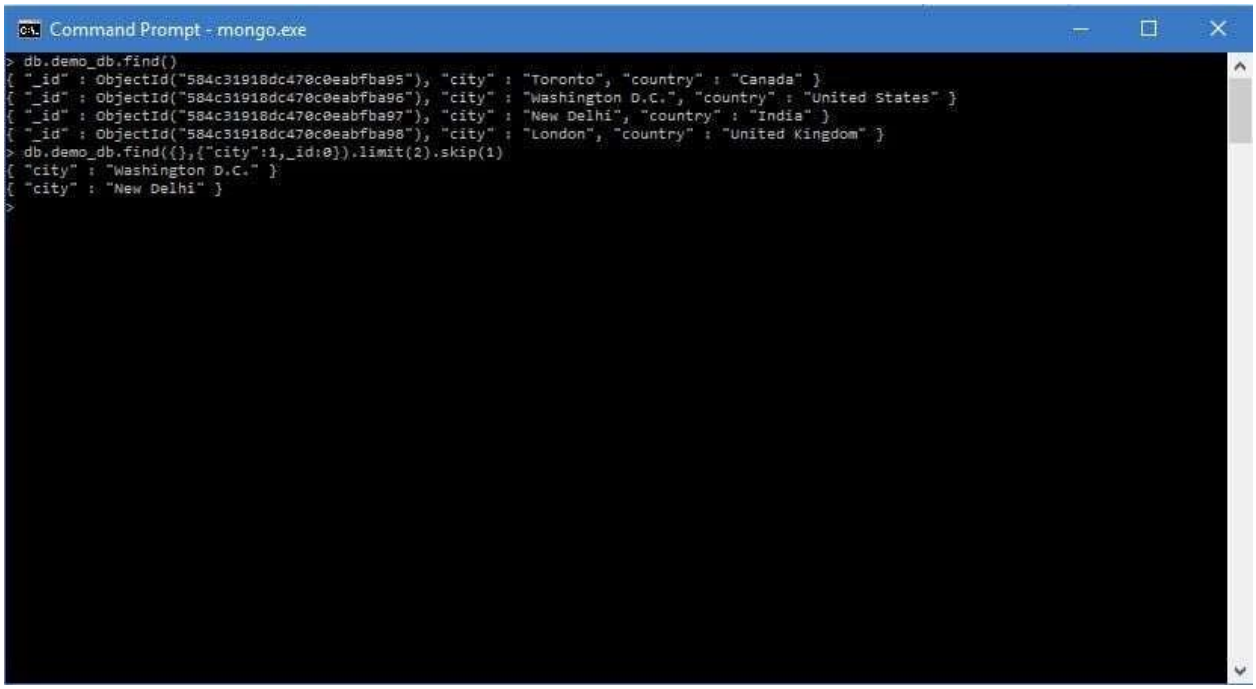
In MongoDB, the 'skip ()' method is used to skip the number of documents. Like the 'limit ()' method, it accepts only one argument which is of number type. Depending on the value of the number, we can skip the number of documents to be displayed. This argument is an optional field inside the 'skip ()' method, and when not specified, then it will display all documents from the collection as the default value in 'skip ()' method is 0.

The following is the basic syntax for 'skip ()' method in the MongoDB.

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

Program

```
> use  demo_db
switched to db demo_db
> db.demo_db.find()
{ "_id" : ObjectId("584c31918dc470c0eabfba95"), "city" : "Toronto", "country" : "Canada" }
{ "_id" : ObjectId("584c31918dc470c0eabfba96"), "city" : "Washington D.C.", "country" : "United States" }
{ "_id" : ObjectId("584c31918dc470c0eabfba97"), "city" : "New Delhi", "country" : "India" }
{ "_id" : ObjectId("584c31918dc470c0eabfba98"), "city" : "London", "country" : "United Kingdom" }
> db.demo_db.find({},{"city":1,_id:0}).limit(2).skip(1)
{ "city" : "Washington D.C." }
{ "city" : "New Delhi" }
>
```



The sort () Method

In MongoDB, the 'sort ()' method is used to sort the documents inside a collection. It accepts a document which contains a list of fields along with their sorting order. We use 1 and -1 in order to specify the sorting order where 1 corresponds to the ascending order and -1 corresponds to the descending order. Also, it should be noted that if we do not specify any sorting preference, then 'sort ()' method will display the documents in the ascending order.

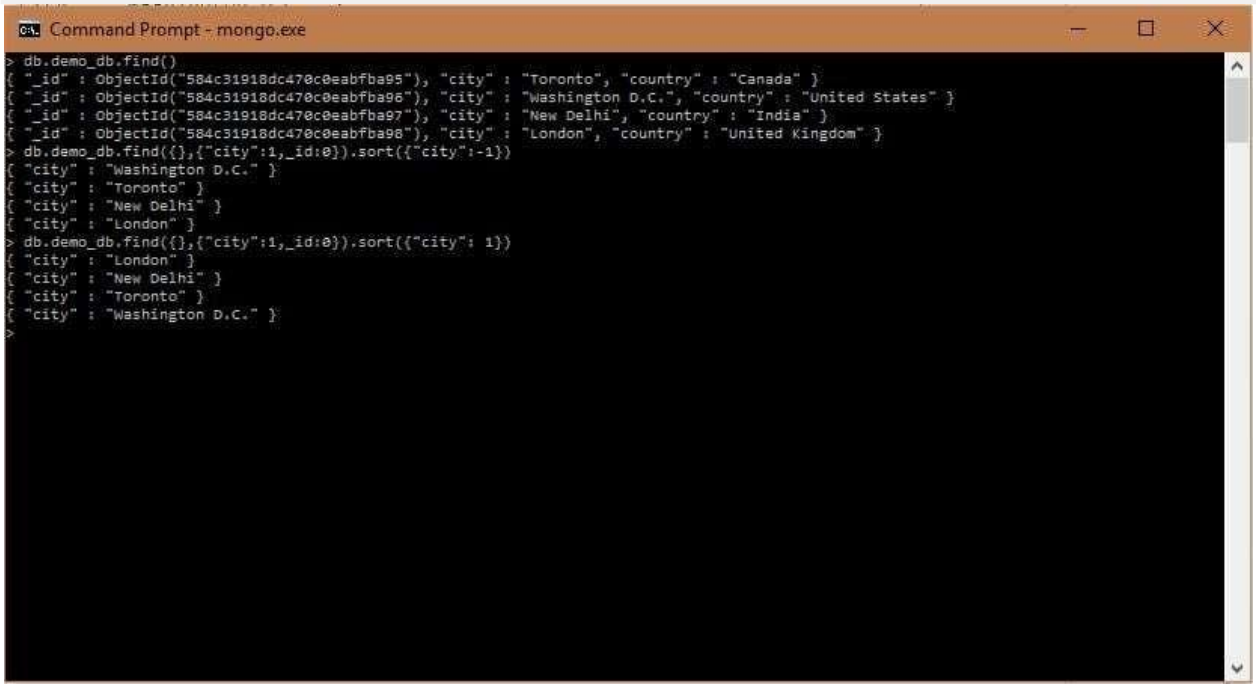
The following is the basic syntax for 'sort ()' method in the MongoDB.

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

Program:

```
> use  demo_db
switched to db demo_db
> db.demo_db.find()
{ "_id" : ObjectId("584c31918dc470c0eabfba95"), "city" : "Toronto", "country" : "Canada" }
{ "_id" : ObjectId("584c31918dc470c0eabfba96"), "city" : "Washington D.C.", "country" : "United States" }
{ "_id" : ObjectId("584c31918dc470c0eabfba97"), "city" : "New Delhi", "country" : "India" }
{ "_id" : ObjectId("584c31918dc470c0eabfba98"), "city" : "London", "country" : "United Kingdom" }
> db.demo_db.find({},{"city":1,_id:0}).sort({"city":-1})
{ "city" : "Washington D.C." }
```

{ “city” : “Toronto” }
{ “city” : “New Delhi” }
{ “city” : “London” }
> db.demo_db.find({},{“city”:1,_id:0}).sort({“city”: 1})
{ “city” : “London” }
{ “city” : “New Delhi” }
{ “city” : “Toronto” }
{ “city” : “Washington D.C.” }
>



Exercise-11

Mongo DB indexing

a. **Create index in Mongo DB**
b. **Finding the indexes in a collection**
c. **Drop indexes in a collection**
d. **Drop all the indexes**

AIM:

Indexes are special data structures that store a small part of the Collection's data in a way that can be queried easily.
indexes store the values of the indexed fields outside the table or collection and keep track of their location in the disk. These values are used to order the indexed fields. This ordering helps to perform equality matches and range-based query operations efficiently. In MongoDB, indexes are defined in the collection level and indexes on any field or subfield of the documents in a collection are supported.

```
use students
db.createCollection("studentgrades")
db.studentgrades.insertMany(
   [
      {name: "Barry", subject: "Maths", score: 92},
      {name: "Kent", subject: "Physics", score: 87},
      {name: "Harry", subject: "Maths", score: 99, notes: "Exceptional Performance"},
      {name: "Alex", subject: "Literature", score: 78},
      {name: "Tom", subject: "History", score: 65, notes: "Adequate"}
   ]
)
db.studentgrades.find({},{_id:0})
```

Result

```
> db.studentgrades.find({},{_id:0})
{ "name" : "Barry", "subject" : "Maths", "score" : 92 }
{ "name" : "Kent", "subject" : "Physics", "score" : 87 }
{ "name" : "Harry", "subject" : "Maths", "score" : 99, "notes" : "Exceptional Performance" }
{ "name" : "Alex", "subject" : "Literature", "score" : 78 }
{ "name" : "Tom", "subject" : "History", "score" : 65, "notes" : "Adequate" }
>
```

Creating indexes

When creating documents in a collection, MongoDB creates a unique index using the _id field. MongoDB refers to this as the **Default _id Index**. This default index cannot be dropped from the collection.

When querying the test data set, you can see the _id field which will be utilized as the default index:

```
db.studentgrades.find().pretty()
```

Result:

```
> db.studentgrades.find().pretty()
{
        "_id" : ObjectId("6026a176de0e65dbecbef031"),
        "name" : "Barry",
        "subject" : "Maths",
        "score" : 92
}
{
        "_id" : ObjectId("6026a176de0e65dbecbef032"),
        "name" : "Kent",
        "subject" : "Physics",
        "score" : 87
}
{
        "_id" : ObjectId("6026a176de0e65dbecbef033"),
        "name" : "Harry",
        "subject" : "Maths",
        "score" : 99,
        "notes" : "Exceptional Performance"
}
{
        "_id" : ObjectId("6026a176de0e65dbecbef034"),
        "name" : "Alex",
        "subject" : "Literature",
        "score" : 78
}
{
        "_id" : ObjectId("6026a176de0e65dbecbef035"),
        "name" : "Tom",
        "subject" : "History",
        "score" : 65,
        "notes" : "Adequate"
}
>
```

Now let's create an index. To do that, you can use the **createIndex** method using the following syntax:

```
db.<collection>.createIndex(<Key and Index Type>, <Options>)
```

When creating an index, you need to define the field to be indexed and the direction of the key (1 or -1) to indicate ascending or descending order.
Another thing to keep in mind is the index names. By default, MongoDB will generate index names by concatenating the indexed keys with the direction of each key in the index using an underscore as the separator. For example: {name: 1} will be created as name_1.

Let's create an index using the name field in the studentgrades collection and name it as **student name index**.

```
db.studentgrades.createIndex(
{name: 1},
{name: "student name index"}
)
```

Result:

```
> db.studentgrades.createIndex(
...     {name: 1},
...     {name: "student name index"}
... )
{
        "createdCollectionAutomatically" : true,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Finding indexes

You can find all the available indexes in a MongoDB collection by using the **getIndexes** method. This will return all the indexes in a specific collection.

```
db.<collection>.getIndexes()
```

Let's view all the indexes in the studentgrades collection using the following command:

```
db.studentgrades.getIndexes()
```

Result:

```
> db.studentgrades.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "name" : 1
                },
                "name" : "student name index"
        }
]
>
```

The output contains the **default _id index** and the user-created index **student name index**.

Dropping indexes

To delete an index from a collection, use the **dropIndex** method while specifying the index name to be dropped.

```
db.<collection>.dropIndex(<Index Name / Field Name>)
```

Let's remove the user-created index with the index name **student name index**, as shown below.

```
db.studentgrades.dropIndex("student name index")
```

Result:

```
> db.studentgrades.dropIndex("student name index")
{ "nIndexesWas" : 2, "ok" : 1 }
>
> db.studentgrades.getIndexes()
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
>
```

You can also use the index field value for removing an index without a defined name:

```
db.studentgrades.dropIndex({name:1})
```

Result:

```
> db.studentgrades.dropIndex({name:1})
{ "nIndexesWas" : 2, "ok" : 1 }
>
>
```

The **dropIndexes** command can also drop all the indexes excluding the default _id index.

```
db.studentgrades.dropIndexes()
```

Result:

```
> db.studentgrades.dropIndexes()
{
        "nIndexesWas" : 1,
        "msg" : "non-_id indexes dropped for collection",
        "ok" : 1
}
>
```

Exercise-12

Mongo DB with java and PHP

a. **Create a simple application that uses Mongo DB with Java**
b. **Create a simple application that uses Mongo DB with PHP**

AIM:

Establishing connections to database

For making the connection, you have to mention the database name. MongoDB creates a database by default if no name is mentioned.

1. Firstly, import the required libraries for establishing the connection.
2. Here, "**MongoClient**" is used to create the client for the database.
3. "**MongoCredential**" is used for creating the credentials.
4. And finally, to access the database "**MongoDatabase**" is used.
5. Username will be: "**GFGUser**" and the database name will be "**mongoDb**".
6. The function ".toCharArray()" is used to convert the password into a character array.
7. The function ".getDatabase()" is used for getting the database.

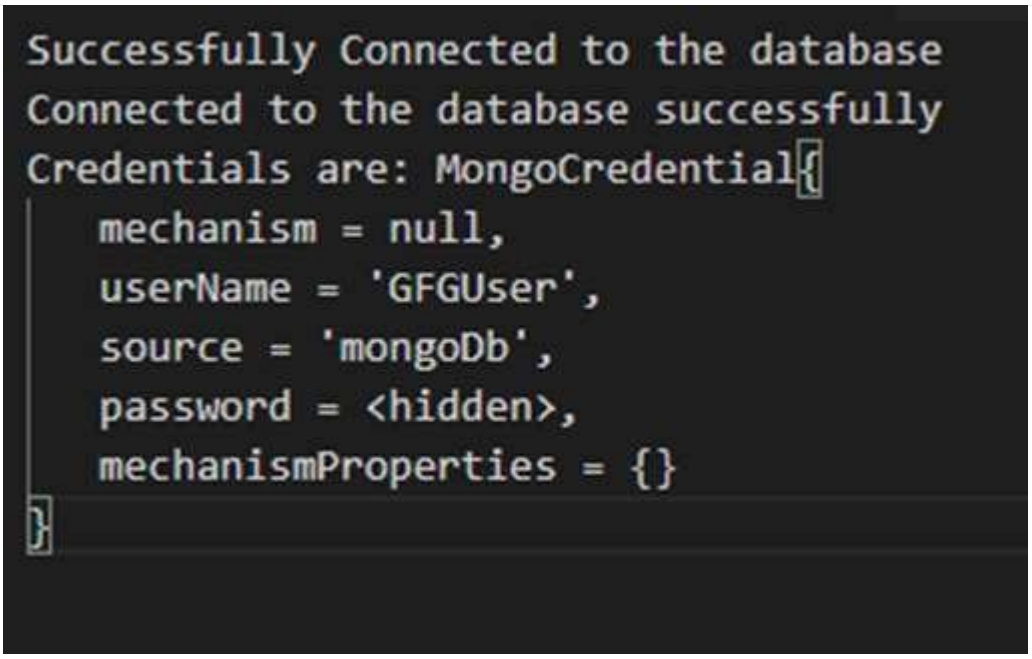
The following code establishes a connection to MongoDB ->

- Java

```
// Java program for establishing connections
// to MongoDb

import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class ConnectionDB {
    public static void establishConnections()
    {

        try {
            MongoClient db
                = new MongoClient("localhost", 27017);

            MongoCredential credential;
            credential
                = MongoCredential
                      .createCredential(
                          "GFGUser", "mongoDb",
                          "password".toCharArray());
            System.out.println(
                "Successfully Connected"
                + " to the database");
```

```
            MongoDatabase database
                = db.getDatabase("mongoDb");
            System.out.println("Credentials are: "
                                + credential);
        }
        catch (Exception e) {
            System.out.println(
                "Connection establishment failed");
            System.out.println(e);
        }
    }
}
```

Output:



To use MongoDB with PHP, you need to use MongoDB PHP driver. Download the driver from the url Download PHP Driver. Make sure to download the latest release of it. Now unzip the archive and put php_mongo.dll in your PHP extension directory ("ext" by default) and add the following line to your php.ini file −

```
extension = php_mongo.dll
```

Make a Connection and Select a Database

To make a connection, you need to specify the database name, if the database doesn't exist then MongoDB creates it automatically.

Following is the code snippet to connect to the database −

```
<?php
  // connect to mongodb
  $m = new MongoClient();
```

```
   echo "Connection to database successfully";
   // select a database
   $db = $m->mydb;

   echo "Database mydb selected";
?>
```

When the program is executed, it will produce the following result −

```
Connection to database successfully
Database mydb selected
```

****** THE END ******