Few Commands to test after connections:

Command	Expected Output	Notes
show dbs	admin 40.00 KiB config 72.00 KiB db 128.00 KiB local 40.00 KiB	All Databases are shown
Use db	switched to db db	Connect and use db
show collections	Students	Show all tables
db.foo.insert({"bar" : "baz"})		Insert a record to collection. Create Collection if not exists
db.foo.batchInsert([{"_id" : 0}, {"_id" : 1}, {"_id" : 2}])		Insert more than one document
db.foo.find()		Print all rows
db.foo.remove()		Remove foo table

Documents:

At the heart of MongoDB is the document: an ordered set of keys with associated values.

The representation of a document varies by programming language, but most languages have a data structure that is a natural fit, such as a map, hash, or dictionary.

{"greeting": "Hello, world!"}

```
{
    "name" : "John Doe",
    "address" : {
        "street" : "123 Park Street",
        "city" : "Anytown",
        "state" : "NY"
    }
}
```

Collections:

Collections A collection is a group of documents. If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.

Database:

MongoDB stores data records as documents (specifically BSON documents) which are gathered together in collections. A database stores one or more collections of documents. You can manage MongoDB databases and collections in the UI for deployments hosted in MongoDB Atlas.

Datatype:

More details link
Where each attribute inside can be of multiple data types.

Load the document:

Download the student csv from this link

Import the data to the collection created link

ADD ,UPDATE AND DELETE THE DATA :

Creating a Database:

On the server running MongoDB, type `MONGO` to open up a connection to the database:

Our first command `use userdb` creates a new database with the name of “userdb” you can put whatever name you want in the format `use <databasename>.`

```
use userdb
```

```
Output
userdb
```

Execute the following command to insert some data into your database:

```
db.people.insert({ name: "Andrew", age: 33, hobbies: ["Coding", "Gaming", "Cooking"], hungry: false})
```

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

Retrieving Data:

Once you have data in your collection, you can start to search and filter that data out using `.find(<parameters>)`
To verify that your data has been added to the “people” document, use the `find()` syntax. Execute this command in the MongoDB console:

```
db.people.find()
```

```
Output
{ "_id" : ObjectId("5c08c98f3d828385a2162d94"), "name" : "Andrew", "age" : 33, "hobbies" : [ "Coding", "Gaming", "Cooking" ], "hungry" : false }
```

If you want to turn this into pretty JSON format, use `.pretty()` after `.find()` :

```
db.people.find().pretty()
Output
{
        "_id" : ObjectId("5c08c98f3d828385a2162d94"),
        "name" : "Andrew",
        "age" : 33,
        "hobbies" : [
                "Coding",
                "Gaming",
                "Cooking"
        ],
        "hungry" : false
}
```

Try to add more data and then we'll work on modifying and searching the data.

```
db.people.insert({ name: "Riley", age: 3, hobbies: ["Sleeping",
"Barking", "Snuggles"], hungry: true})

db.people.insert({ name: "You", age: 30, hobbies: ["Coding", "Reading
DigitalOcean Articles", "Creating Droplets"], hungry: true})
```

Updating Data:

To modify your data, use the `.update()` function. but first let's look at our data to see what we want to change:

```
db.people.update({ name: "You" }, {$set: { name: "Sammy" }})
Output
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

If we now check that record with our newly set name:

```
db.people.find({ name: "Sammy" }).pretty()
Output
{
        "_id" : ObjectId("5c08cc2e3d828385a2162d96"),
        "name" : "Sammy",
        "age" : 30,
        "hobbies" : [
                "Coding",
                "Reading DigitalOcean Articles",
                "Creating Droplets"
        ],
        "hungry" : true
}
```

The `name` key value has been set to its new value of `Sammy`.

Deleting Data (D):

Remove data using the `.remove()` function. You can remove data in a couple of ways, but the safest way is to locate a record to delete by using the unique `_id` so that, for instance, you have multiple `"Sammy"` entries, removing by the `name: "Sammy"` would remove all of them.

```
db.people.remove({ _id: ObjectId("5c08cc2e3d828385a2162d96")})
Output
WriteResult({ "nRemoved" : 1 })
db.people.find().pretty()
Output
{
        "_id" : ObjectId("5c08c98f3d828385a2162d94"),
        "name" : "Andrew",
        "age" : 33,
        "hobbies" : [
                "Coding",
                "Gaming",
                "Cooking"
        ],
        "hungry" : false
}
{
        "_id" : ObjectId("5c08cbea3d828385a2162d95"),
        "name" : "Riley",
        "age" : 3,
        "hobbies" : [
                "Sleeping",
                "Barking",
                "Snuggles"
        ],
        "hungry" : true
}
```

The `"Sammy"` entry has been removed safely, without affecting any other possible `"Sammy"` records if they were to exist.