Brandon Koskie

Ashley Alfaro

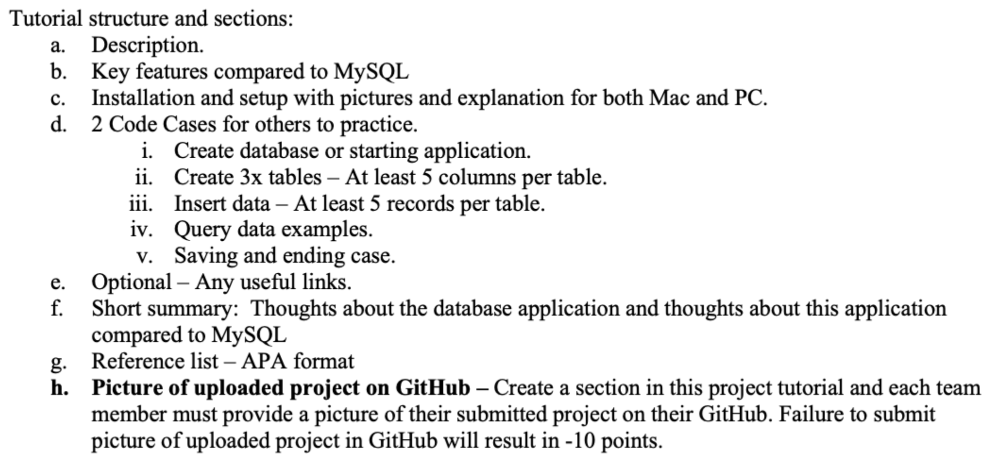
Cade Garcia

CS-200

Rylan Chong

11/15/23

MongoDB Project



Tutorial structure and sections:

a. Description.
b. Key features compared to MySQL
c. Installation and setup with pictures and explanation for both Mac and PC.
d. 2 Code Cases for others to practice.
    i. Create database or starting application.
    ii. Create 3x tables – At least 5 columns per table.
    iii. Insert data – At least 5 records per table.
    iv. Query data examples.
    v. Saving and ending case.
e. Optional – Any useful links.
f. Short summary: Thoughts about the database application and thoughts about this application compared to MySQL
g. Reference list – APA format
h. **Picture of uploaded project on GitHub –** Create a section in this project tutorial and each team member must provide a picture of their submitted project on their GitHub. Failure to submit picture of uploaded project in GitHub will result in -10 points.

What is MongoDB?

MongoDB is a non-relational database made for making applications easily and growing them bigger (*What Is MongoDB? — MongoDB Manual*, n.d.). These databases are more about storing documents in a JSON format than tables of data. It has three parts: MongoDB Atlas, MongoDB Enterprise, and MongoDB Community. Atlas helps manage MongoDB in the cloud, making it easier to use. Enterprise is a paid version for businesses, and Community is a free version available to everyone. With MongoDB, it is all about making it simple to build and expand applications, whether you're a big company or just starting out.

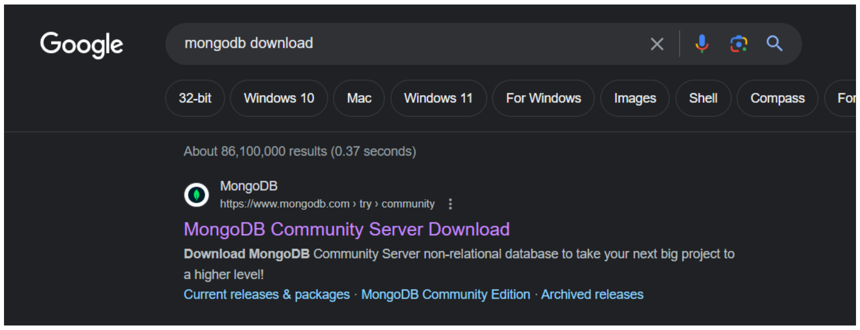
Key Features: MongoDB vs MySQL

MongoDB	MySQL
● Non-relational database ● Stores data as JSON-like documents ● Uses Java for coding	● Relational database ● Stores data using tables and rows

How to Install and Setup MongoDB on your PC

Step 1:

Search "mongodb download" on Google, and go to *MongoDB Community Server Download*.



mongodb download

About 86,100,000 results (0.37 seconds)

MongoDB
https://www.mongodb.com › try › community

MongoDB Community Server Download

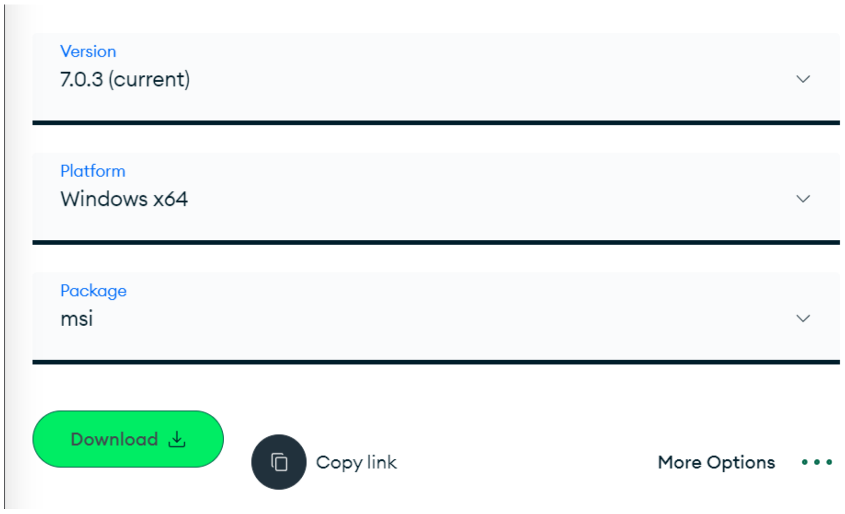
Download MongoDB Community Server non-relational database to take your next big project to a higher level!

Current releases & packages · MongoDB Community Edition · Archived releases

–

Here, you can select any Version, Platform (depends on your PC), and Package.

We chose:
- Version: 7.0.3 (Current)
- Platform: Windows x64
- Package: msi

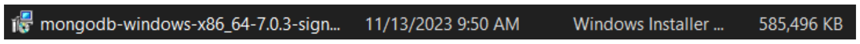


Version: 7.0.3 (current)

Platform: Windows x64

Package: msi

This is how the file should look like.

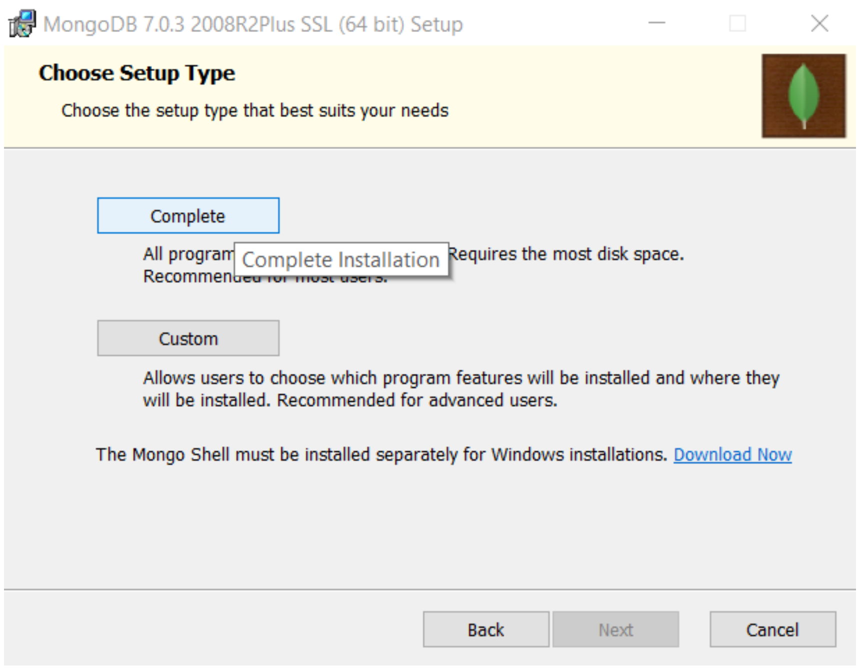


```
mongodb-windows-x86_64-7.0.3-sign...   11/13/2023 9:50 AM   Windows Installer ...   585,496 KB
```

Step 2:

Select the *Complete* option to install all the program features.

*If you want to install only selected program features and select the location of the installation, then select the *Custom* option.


MongoDB 7.0.3 2008R2Plus SSL (64 bit) Setup

Choose Setup Type

Choose the setup type that best suits your needs

Complete

All program Complete Installation Requires the most disk space.
Recommended for most users.

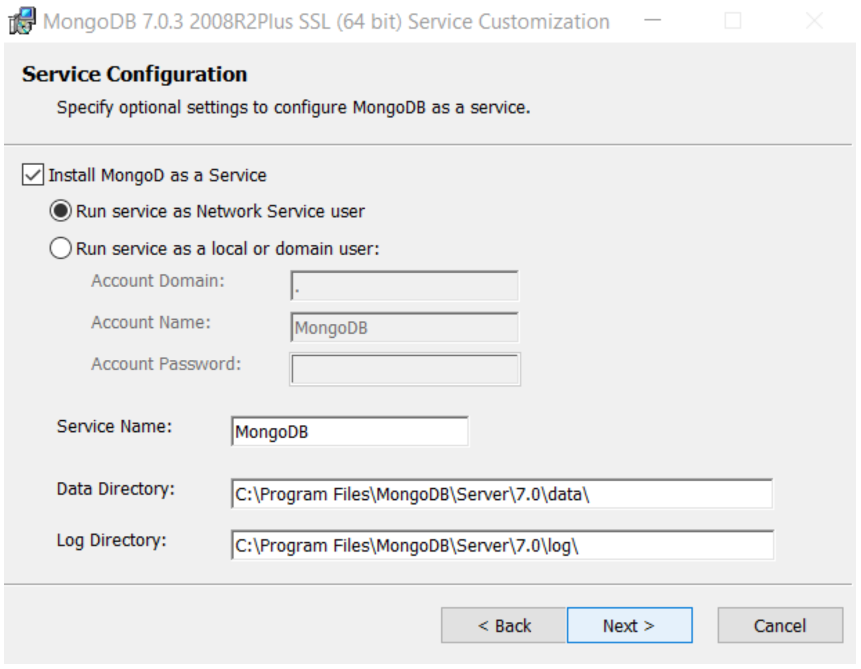
Custom

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

The Mongo Shell must be installed separately for Windows installations. Download Now

Back | Next | Cancel

Step 3:
Select *Run service as Network Service user* then click *Next*.


MongoDB 7.0.3 2008R2Plus SSL (64 bit) Service Customization

Service Configuration

Specify optional settings to configure MongoDB as a service.

☑ Install MongoD as a Service

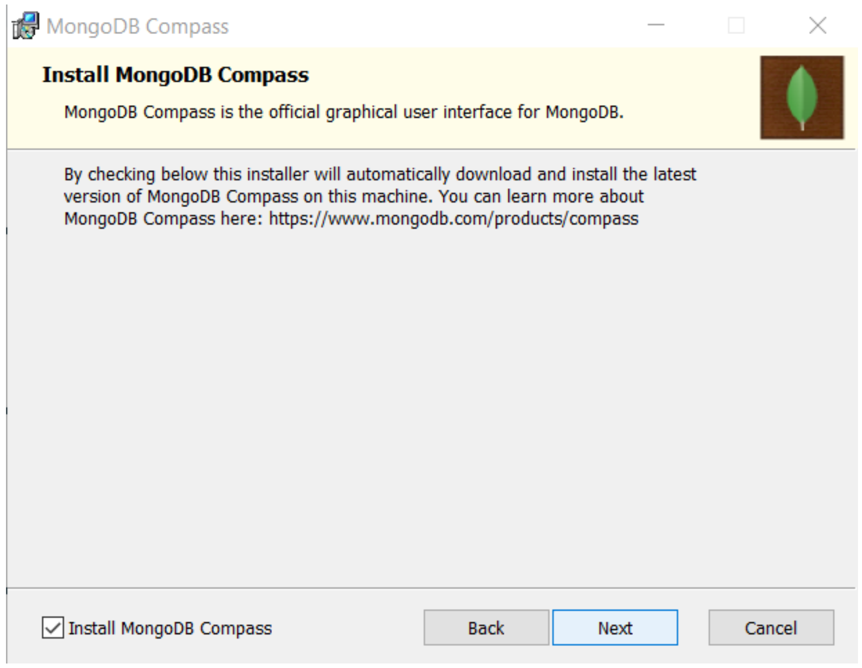
● Run service as Network Service user

○ Run service as a local or domain user:

Account Domain:	.
Account Name:	MongoDB
Account Password:	
Service Name:	MongoDB
Data Directory:	C:\Program Files\MongoDB\Server\7.0\data\
Log Directory:	C:\Program Files\MongoDB\Server\7.0\log\

< Back | Next > | Cancel

Step 4:
On the bottom left, check *Install MongoDB Compass* to automatically download MongoDB Compass to your PC


MongoDB Compass

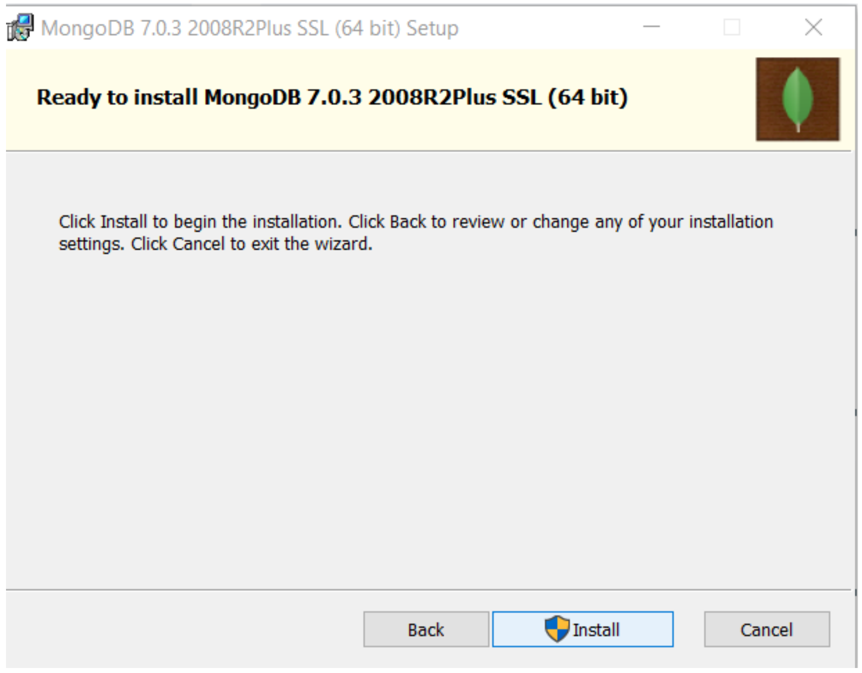
Install MongoDB Compass

MongoDB Compass is the official graphical user interface for MongoDB.

By checking below this installer will automatically download and install the latest version of MongoDB Compass on this machine. You can learn more about MongoDB Compass here: https://www.mongodb.com/products/compass

☑ Install MongoDB Compass

Step 5:
Click *Install*


MongoDB 7.0.3 2008R2Plus SSL (64 bit) Setup

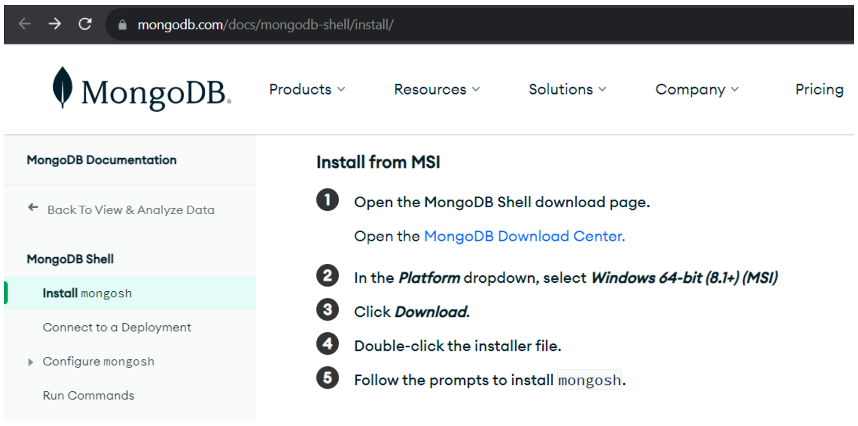
Ready to install MongoDB 7.0.3 2008R2Plus SSL (64 bit)

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Step 6:
Download MongoDB Shell.

With MongoDB Shell, you can query and update data as well as perform administrative operations.

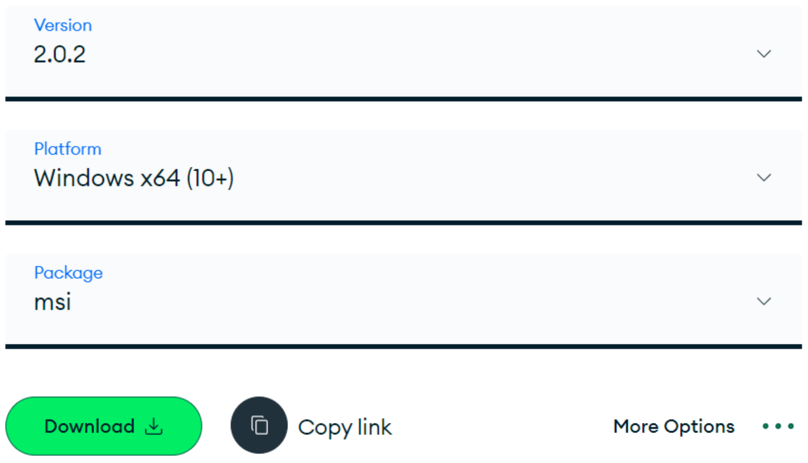
It is a JavaScript and Node.js environment.



Install from MSI

1. Open the MongoDB Shell download page.

   Open the MongoDB Download Center.
2. In the *Platform* dropdown, select *Windows 64-bit (8.1+) (MSI)*
3. Click *Download.*
4. Double-click the installer file.
5. Follow the prompts to install `mongosh`.



Version: 2.0.2

Platform: Windows x64 (10+)

Package: msi

Basic Code Commands

QUERYING

*Queries are meant to help you find and work with your data. A query can be a request for data results from your database, or for action on the data.

InsertONE = one object (can be with many columns)
InsertMANY = more than one object (can be with many columns)

And underscore (**_**) means **where**. Ex: _id is asking which ID

String is anything between the quotation marks " "
Numbers are the same as SQL

When querying, setting the column equal to **zero** removes it from the result.

COMPLEX QUERIES

Command	Definition	Notes
$eq	equal (finds objects with that exact string)	
$ne	Not equal	
$gt	Greater than	
$lt	Less than	
$gte	Greater than or equal to	
$lte	Less than or equal to	
$in	In	
$nin	Not in	
$exists	exists	set to true or false to show objects that contain the column, even if their value is null)
Command	Definition	Notes

$and	And	(personally, doesn’t seem all that useful since it already does it)
$or	Or	
$not	Not	when put in front of a query value, it negates it.

Putting a $ in front of a value makes it ask for a column***

UPDATING

_Id requires the object ID which is those jumble of letters.

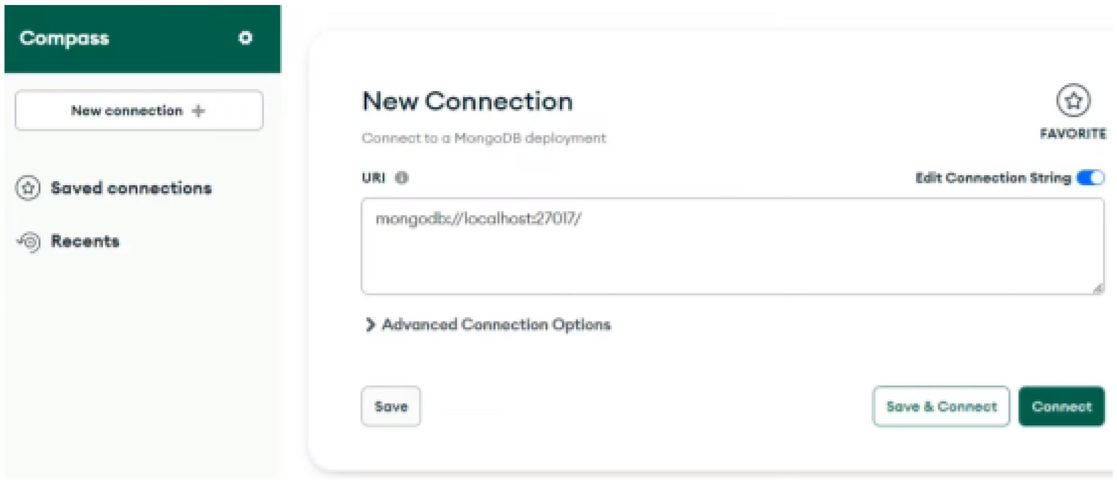
Command	Definition	Notes
$set	Set	(personally, doesn’t seem all that useful since it already does it)
$rename	Rename	Renames a column
$unset	Unset	Removes a value from a column. Which also removes it from the query.
$push	Push	Adds the value to the end of the array
$pull	Pull	Puts the value to the start of the array
db.users.repla ce	Replace	Replaces an entire objec’s field – typically, we would rather use db.users.update

Coding
If you are interested in using MongoDB, we provided 2 Code Cases for you to follow along and practice with. Case 1 is classes. We will create three classes that consist of five students.

For Case 2, we are doing geospatial locations. We chose three countries and took five tourist spots from each.

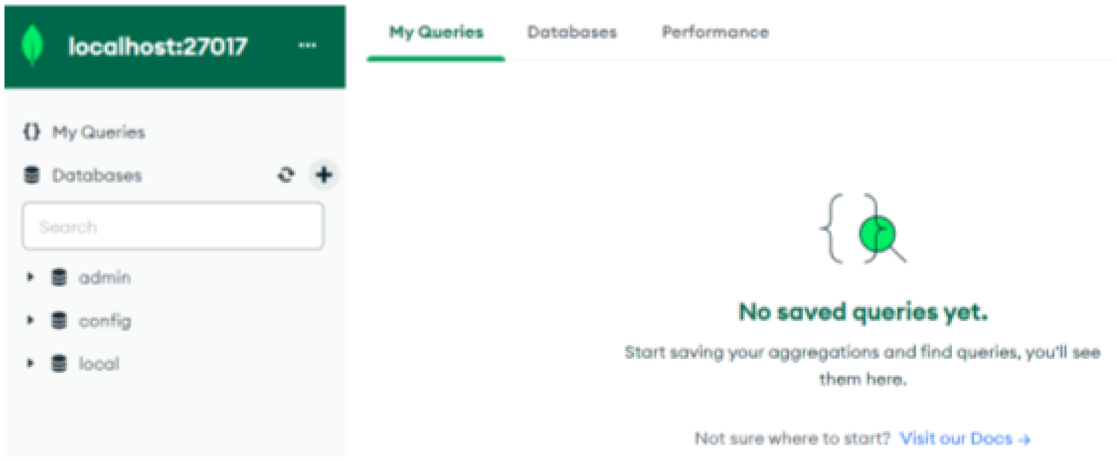
For both Code Cases, you will have to:

Open *MongoDB Compass* and select Connect.



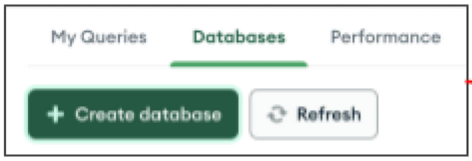
```
mongodb://localhost:27017/
```

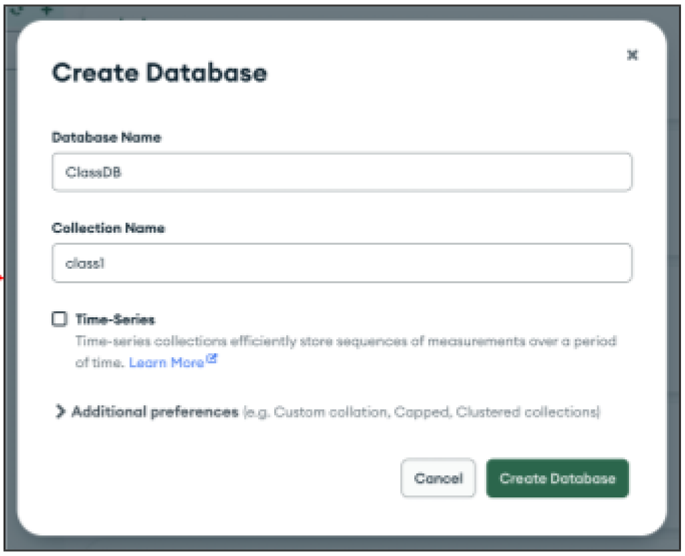
After you do that, it should bring you to this page. This is where you can create your databases.


localhost:27017
My Queries
Databases
Search
admin
config
local
My Queries
Databases
Performance
No saved queries yet.
Start saving your aggregations and find queries, you'll see them here.
Not sure where to start? Visit our Docs

Case 1: Classes

Create your Database





Database Name
ClassDB

Collection Name
class1

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

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

Database Name: ClassDB
Collection Name: class1

A **database** is a container for collections of data, and each database gets its own set of files.

A **collection** is a group of documents.

If you had downloaded MongoDB Shell, you should see this at the bottom of your MongoDB Compass window. This is where you can add data to your database, interact with your data, and test queries.

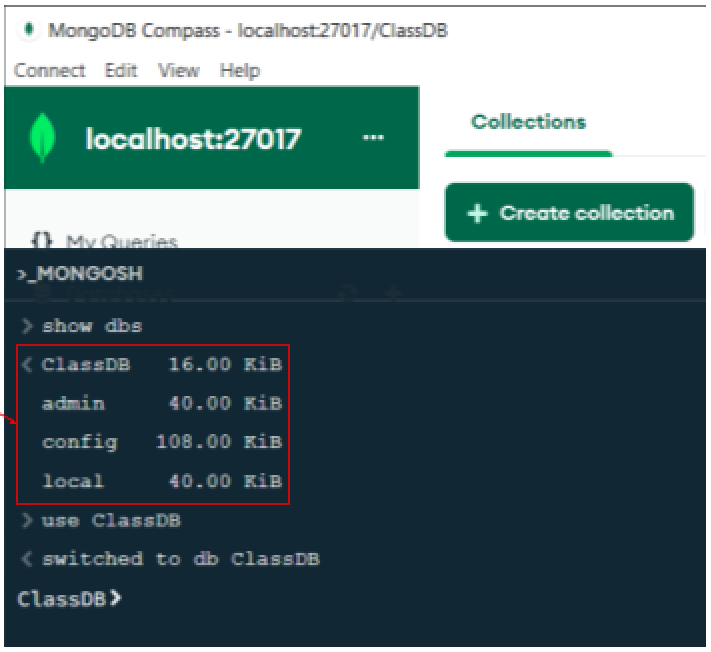
```
> show dbs
```

The `show dbs` command shows all of the databases you currently have.

```
> use ClassDB
```

To switch to the database you want to use, type `use <database_name>` or, in this case, `use ClassDB`.

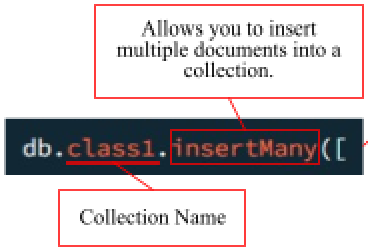
*dbs stands for Database



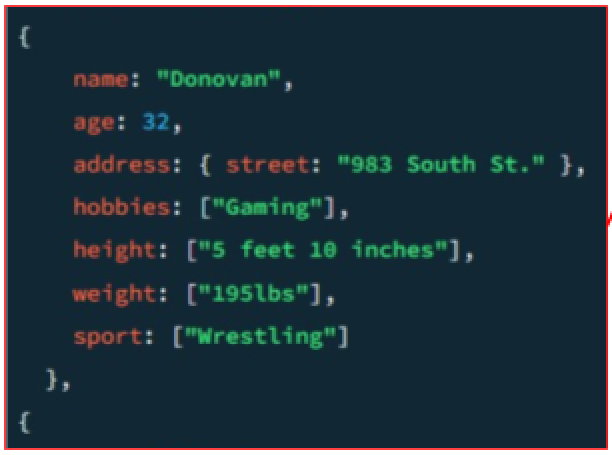
```
> show dbs
< ClassDB    16.00 KiB
  admin      40.00 KiB
  config    108.00 KiB
  local      40.00 KiB
> use ClassDB
< switched to db ClassDB
ClassDB>
```

Case 1: Inserting Data

Inserting Data into *class1* Collection

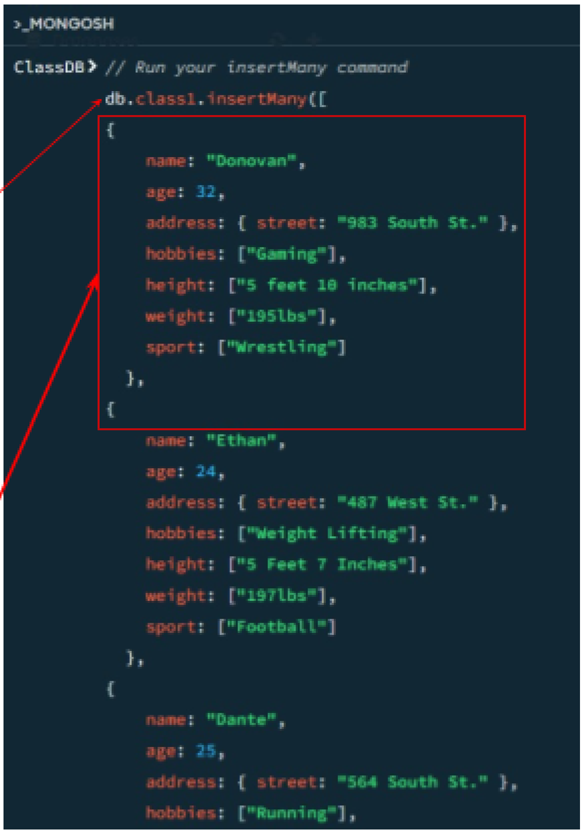


```
db.class1.insertMany([
```



```
{
    name: "Donovan",
    age: 32,
    address: { street: "983 South St." },
    hobbies: ["Gaming"],
    height: ["5 feet 10 inches"],
    weight: ["195lbs"],
    sport: ["Wrestling"]
  },
{
```

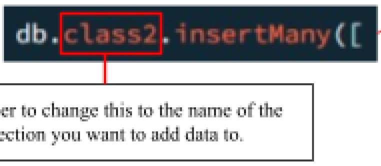
Red text are fields, green text are strings, and blue text are integers.



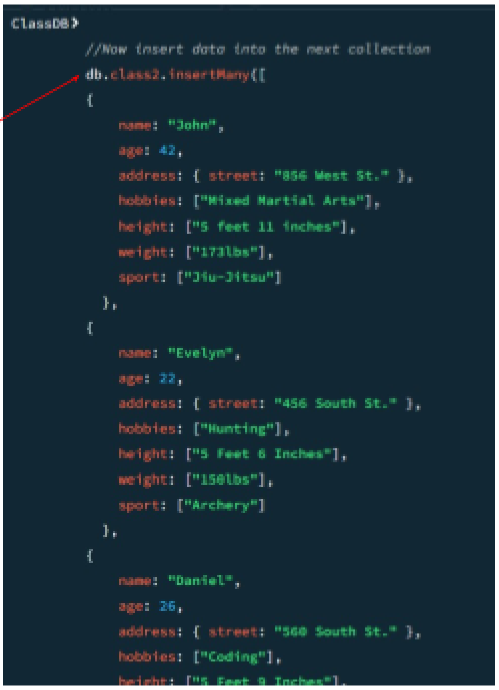
```
>_MONGOSH

ClassDB> // Run your insertMany command
         db.class1.insertMany([
         {
             name: "Donovan",
             age: 32,
             address: { street: "983 South St." },
             hobbies: ["Gaming"],
             height: ["5 feet 10 inches"],
             weight: ["195lbs"],
             sport: ["Wrestling"]
           },
         {
             name: "Ethan",
             age: 24,
             address: { street: "487 West St." },
             hobbies: ["Weight Lifting"],
             height: ["5 Feet 7 Inches"],
             weight: ["197lbs"],
             sport: ["Football"]
           },
         {
             name: "Dante",
             age: 25,
             address: { street: "564 South St." },
             hobbies: ["Running"],
```

Curly braces { } are used to group code blocks or statements.
Brackets [ ] are to access or modify the properties/elements of an object or array.

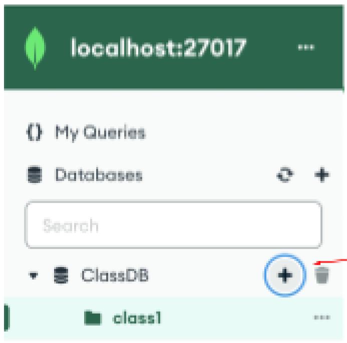
Now that you know how to insert data, we will be doing that for the rest of our collections: *class2* and *class3*.


Remember to change this to the name of the collection you want to add data to.

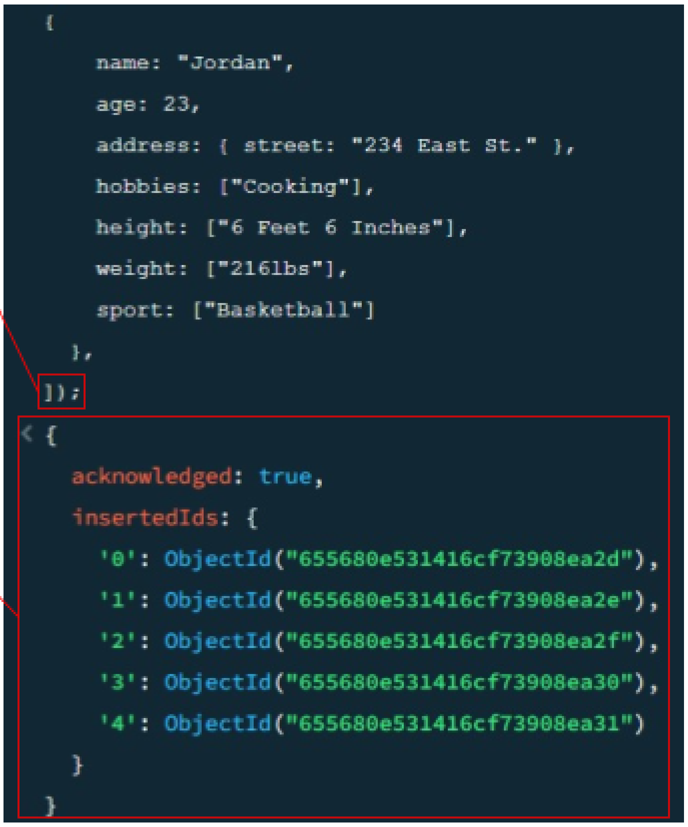
```
db.class2.insertMany([
```



```
ClassDB>
         //Now insert data into the next collection
         db.class2.insertMany([
         {
             name: "John",
             age: 42,
             address: { street: "856 West St." },
             hobbies: ["Mixed Martial Arts"],
             height: ["5 feet 11 inches"],
             weight: ["173lbs"],
             sport: ["Jiu-Jitsu"]
           },
         {
             name: "Evelyn",
             age: 22,
             address: { street: "456 South St." },
             hobbies: ["Hunting"],
             height: ["5 Feet 6 Inches"],
             weight: ["150lbs"],
             sport: ["Archery"]
           },
         {
             name: "Daniel",
             age: 26,
             address: { street: "560 South St." },
             hobbies: ["Coding"],
             height: ["5 Feet 9 Inches"],
```



Click the '+' to add more collections to your database.



```
{
    name: "Jordan",
    age: 23,
    address: { street: "234 East St." },
    hobbies: ["Cooking"],
    height: ["6 Feet 6 Inches"],
    weight: ["216lbs"],
    sport: ["Basketball"]
  },
]);
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId("655680e531416cf73908ea2d"),
      '1': ObjectId("655680e531416cf73908ea2e"),
      '2': ObjectId("655680e531416cf73908ea2f"),
      '3': ObjectId("655680e531416cf73908ea30"),
      '4': ObjectId("655680e531416cf73908ea31")
    }
  }
```

Make sure your code ends with this.

After running the code, this should pop up. This shows that you did it right.

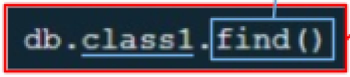
It generates an Object ID, which acts as a unique identifier for each document.

Case 1: Basic Query Commands

Find All Data in the Collection

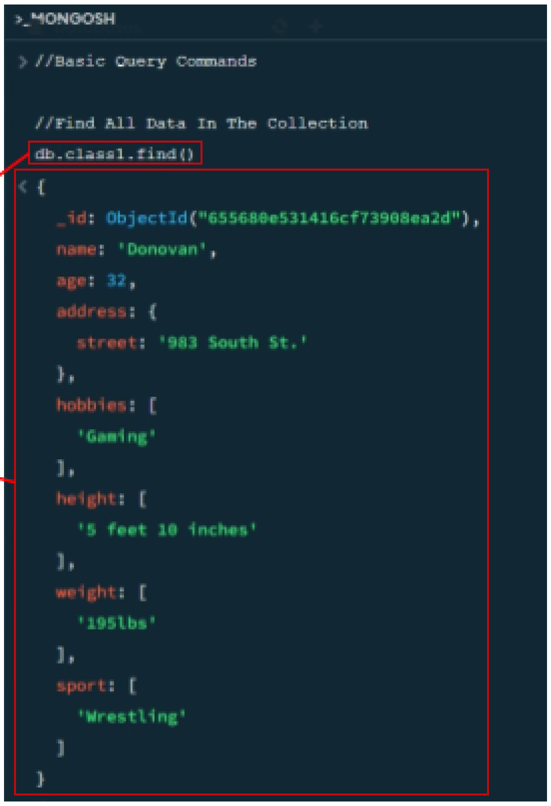
This will retrieve all the data from *class1*.

This returns documents/records



```
db.class1.find()
```

These are the results. So, from this point forward, the colored text you see in the rest of the screenshots are the results of the query.

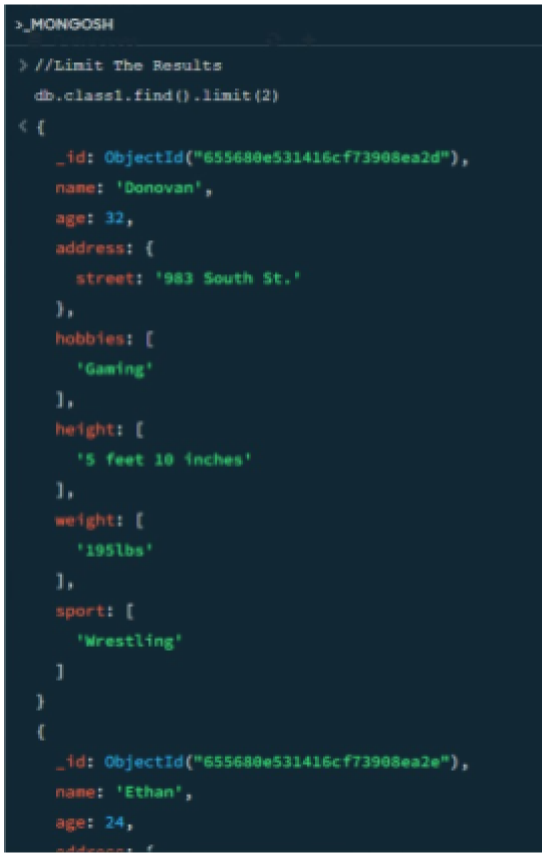


```
> //Basic Query Commands

  //Find All Data In The Collection
  db.class1.find()
< {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
    address: {
      street: '983 South St.'
    },
    hobbies: [
      'Gaming'
    ],
    height: [
      '5 feet 10 inches'
    ],
    weight: [
      '195lbs'
    ],
    sport: [
      'Wrestling'
    ]
  }
```

Limit the Results

```
db.class1.find().limit(2)
```

Defines the max limit of records/documents you want.

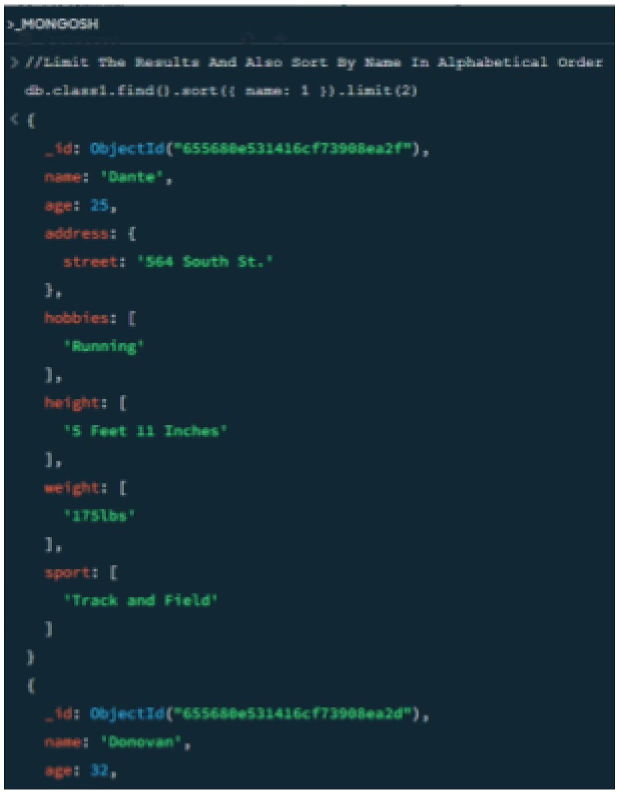


```
>_MONGOSH
> //Limit The Results
  db.class1.find().limit(2)
< {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
    address: {
      street: '983 South St.'
    },
    hobbies: [
      'Gaming'
    ],
    height: [
      '5 feet 10 inches'
    ],
    weight: [
      '195lbs'
    ],
    sport: [
      'Wrestling'
    ]
  }
  {
    _id: ObjectId("655680e531416cf73908ea2e"),
    name: 'Ethan',
    age: 24,
```

Limit Results and *Sort by Name* in Alphabetical Order

```
db.class1.find().sort({ name: 1 }).limit(2)
```

This specifies the sorting order. **1** is ascending.



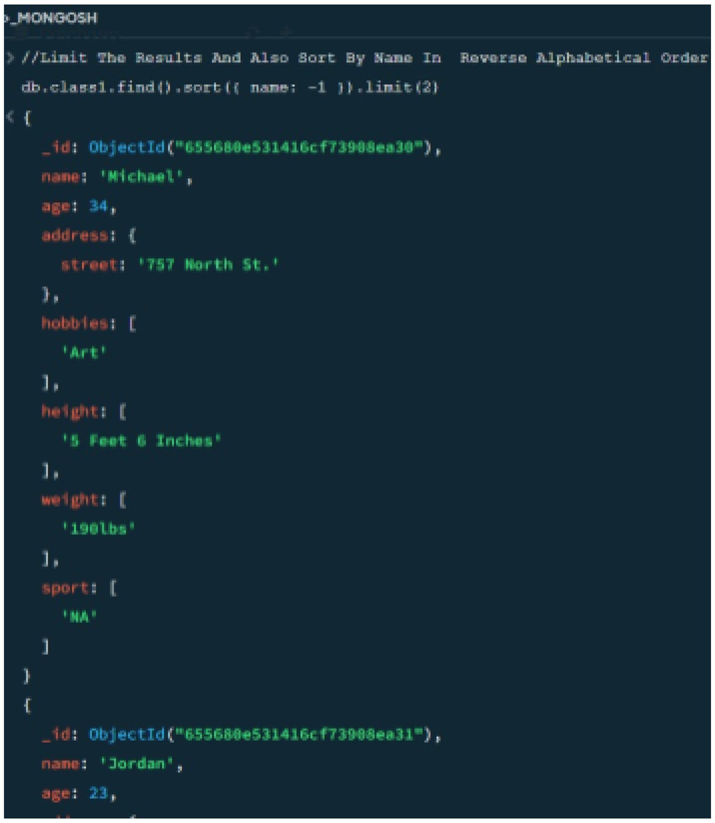
```
>_MONGOSH
> //Limit The Results And Also Sort By Name In Alphabetical Order
  db.class1.find().sort({ name: 1 }).limit(2)
< {
    _id: ObjectId("655680e531416cf73908ea2f"),
    name: 'Dante',
    age: 25,
    address: {
      street: '564 South St.'
    },
    hobbies: [
      'Running'
    ],
    height: [
      '5 Feet 11 Inches'
    ],
    weight: [
      '175lbs'
    ],
    sport: [
      'Track and Field'
    ]
  }
  {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
```

Limit Results and Sort by Name in *Reverse* Alphabetical Order

```
db.class1.find().sort({ name: -1 }).limit(2)
```

This specifies the sorting order. –1 is descending.

Text

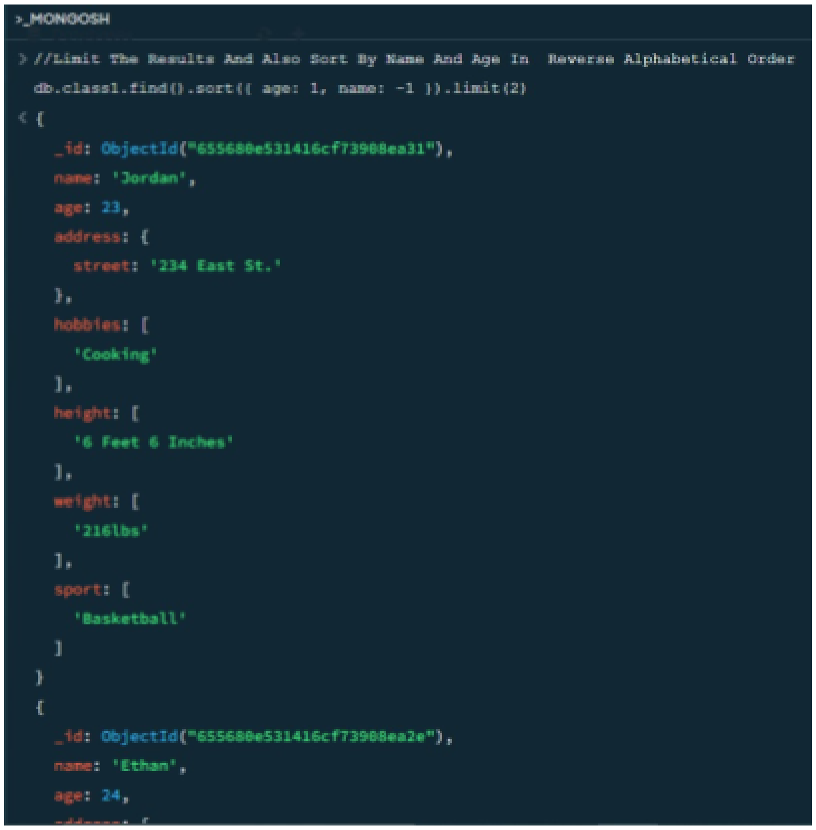


```
>_MONGOSH
> //Limit The Results And Also Sort By Name In  Reverse Alphabetical Order
  db.class1.find().sort({ name: -1 }).limit(2)
< {
    _id: ObjectId("655680e531416cf73908ea30"),
    name: 'Michael',
    age: 34,
    address: {
      street: '757 North St.'
    },
    hobbies: [
      'Art'
    ],
    height: [
      '5 Feet 6 Inches'
    ],
    weight: [
      '190lbs'
    ],
    sport: [
      'NA'
    ]
  }
  {
    _id: ObjectId("655680e531416cf73908ea31"),
    name: 'Jordan',
    age: 23,
```

Limit Results and Sort by Name and *Age* in Reverse Alphabetical Order

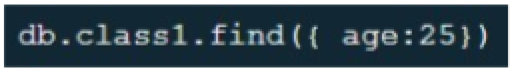
```
db.class1.find().sort({ age: 1,
name: -1 }).limit(2)
```

Text



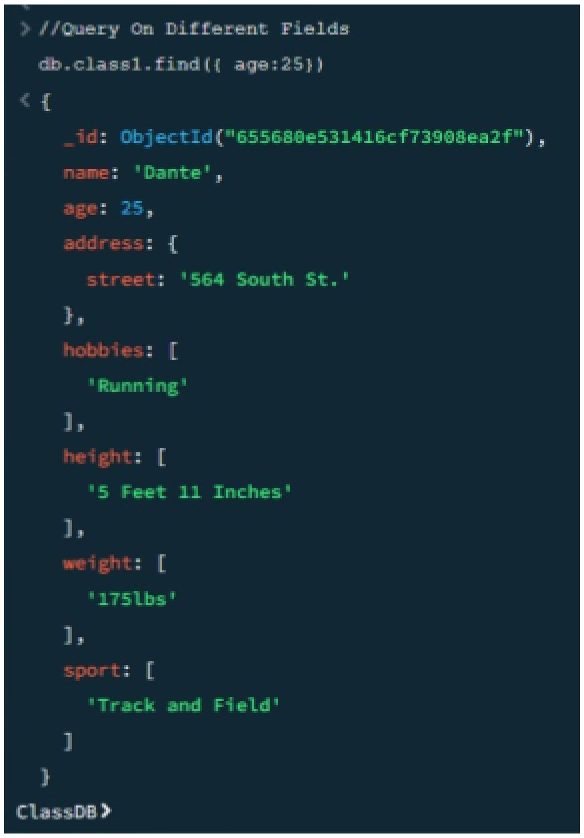
```
>_MONGOSH
> //Limit The Results And Also Sort By Name And Age In  Reverse Alphabetical Order
  db.class1.find().sort({ age: 1, name: -1 }).limit(2)
< {
    _id: ObjectId("655680e531416cf73908ea31"),
    name: 'Jordan',
    age: 23,
    address: {
      street: '234 East St.'
    },
    hobbies: [
      'Cooking'
    ],
    height: [
      '6 Feet 6 Inches'
    ],
    weight: [
      '216lbs'
    ],
    sport: [
      'Basketball'
    ]
  }
  {
    _id: ObjectId("655680e531416cf73908ea2e"),
    name: 'Ethan',
    age: 24,
```

Query on Different Fields



```
db.class1.find({ age:25})
```

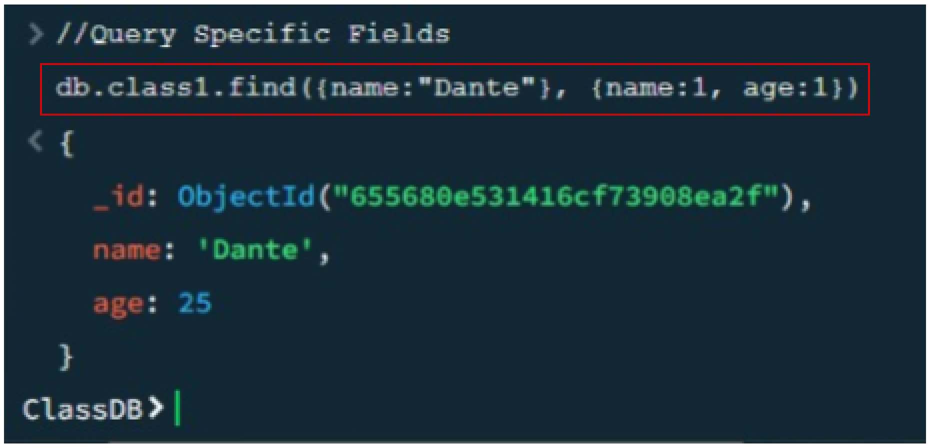
You can play around with this query and search for different ages that you know is in your database.



```
> //Query On Different Fields
  db.class1.find({ age:25})
< {
    _id: ObjectId("655680e531416cf73908ea2f"),
    name: 'Dante',
    age: 25,
    address: {
      street: '564 South St.'
    },
    hobbies: [
      'Running'
    ],
    height: [
      '5 Feet 11 Inches'
    ],
    weight: [
      '175lbs'
    ],
    sport: [
      'Track and Field'
    ]
  }
ClassDB>
```

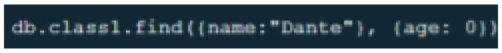
Query Specific Fields

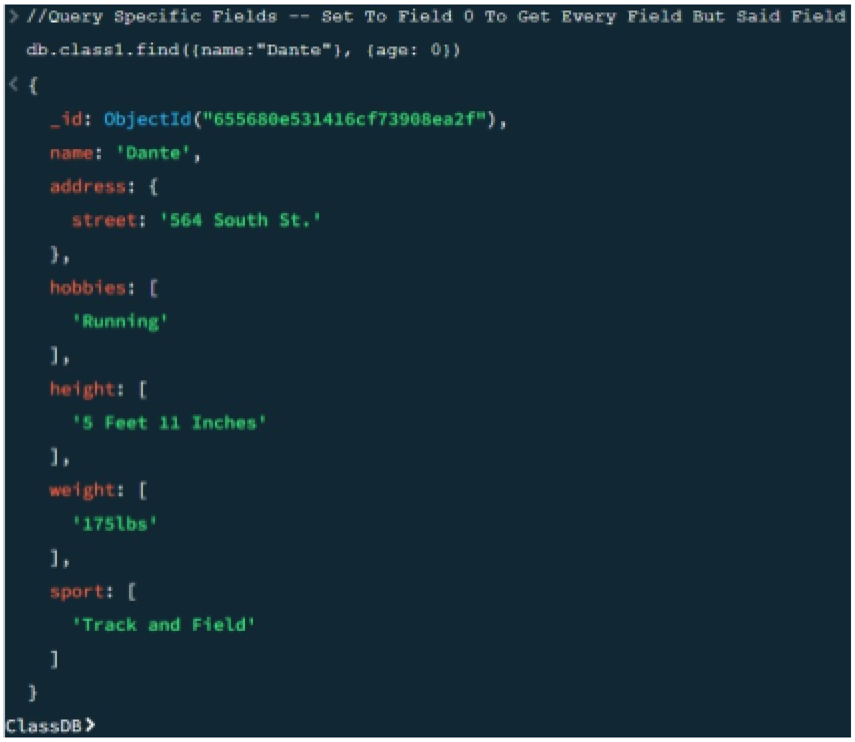
In this example, the results showed only Dante's name and age.



```
> //Query Specific Fields
  db.class1.find({name:"Dante"}, {name:1, age:1})
< {
    _id: ObjectId("655680e531416cf73908ea2f"),
    name: 'Dante',
    age: 25
  }
ClassDB>
```

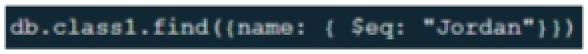
Set Field to 0 to get Every Field but Said Field

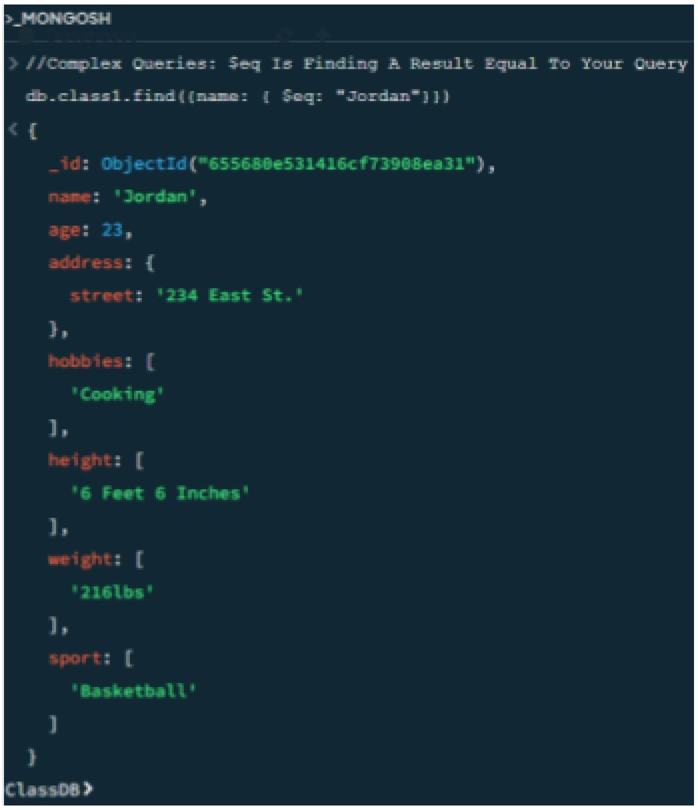

```
db.class1.find({name:"Dante"}, {age: 0})
```


```
> //Query Specific Fields -- Set To Field 0 To Get Every Field But Said Field
  db.class1.find({name:"Dante"}, {age: 0})
< {
    _id: ObjectId("655680e531416cf73908ea2f"),
    name: 'Dante',
    address: {
      street: '564 South St.'
    },
    hobbies: [
      'Running'
    ],
    height: [
      '5 Feet 11 Inches'
    ],
    weight: [
      '175lbs'
    ],
    sport: [
      'Track and Field'
    ]
  }
ClassDB>
```

Case 1: Complex Queries

$eq Is Finding a Results Equal to your Query


```
db.class1.find({name: { $eq: "Jordan"}})
```

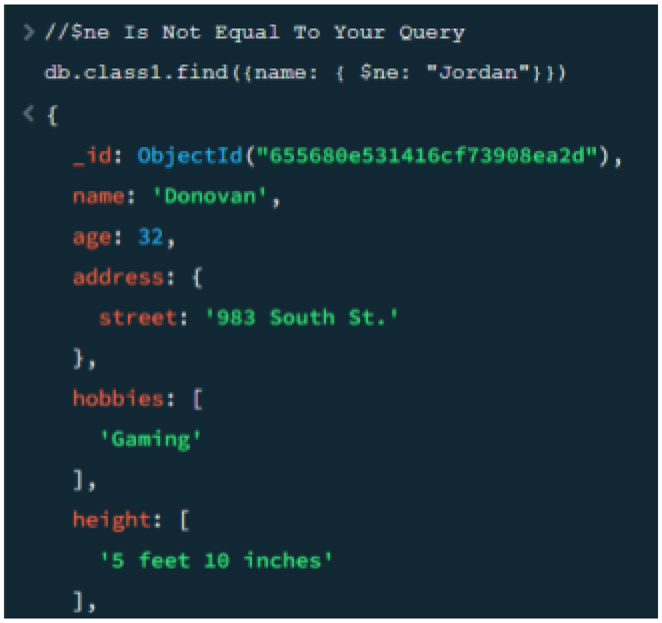

```
>_MONGOSH
> //Complex Queries: $eq Is Finding A Result Equal To Your Query
  db.class1.find({name: { $eq: "Jordan"}})
< {
    _id: ObjectId("655680e531416cf73908ea31"),
    name: 'Jordan',
    age: 23,
    address: {
      street: '234 East St.'
    },
    hobbies: [
      'Cooking'
    ],
    height: [
      '6 Feet 6 Inches'
    ],
    weight: [
      '216lbs'
    ],
    sport: [
      'Basketball'
    ]
  }
ClassDB>
```

$ne is Not Equal to your Query

```
db.class1.find({name: { $ne: "Jordan"}})
```

The results given are the documents where the value of the specified field (name) is not equal to whatever the value is.

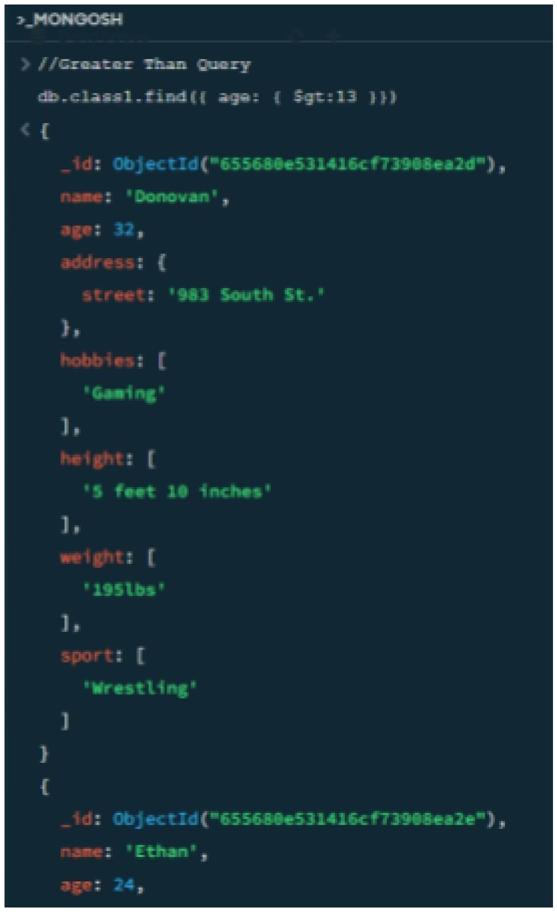
In the screenshot, the results will list everyone in *class1* whose name is not "Jordan".



```
> //$ne Is Not Equal To Your Query
  db.class1.find({name: { $ne: "Jordan"}})
< {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
    address: {
      street: '983 South St.'
    },
    hobbies: [
      'Gaming'
    ],
    height: [
      '5 feet 10 inches'
    ],
```

Greater Than Query

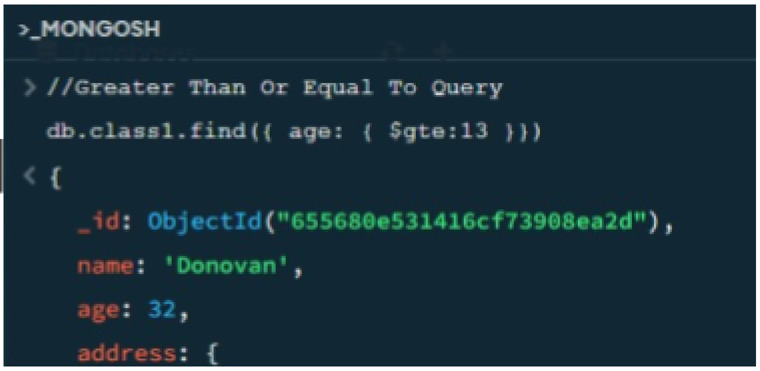
```
db.class1.find({ age: { $gt:13 }})
```



```
>_MONGOSH
> //Greater Than Query
  db.class1.find({ age: { $gt:13 }})
< {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
    address: {
      street: '983 South St.'
    },
    hobbies: [
      'Gaming'
    ],
    height: [
      '5 feet 10 inches'
    ],
    weight: [
      '195lbs'
    ],
    sport: [
      'Wrestling'
    ]
  }
  {
    _id: ObjectId("655680e531416cf73908ea2e"),
    name: 'Ethan',
    age: 24,
```

Query – limits results to objects with a value *greater than or equal to* 13

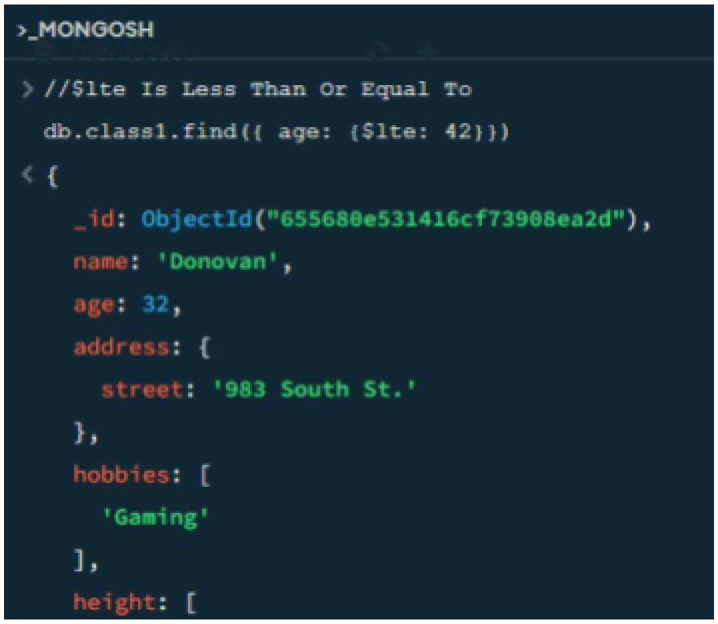
```
db.class1.find({ age: { $gte:13 }})
```



```
>_MONGOSH
> //Greater Than Or Equal To Query
  db.class1.find({ age: { $gte:13 }})
< {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
    address: {
```

Query – limits the results to objects with a value *less than or equal to* 42

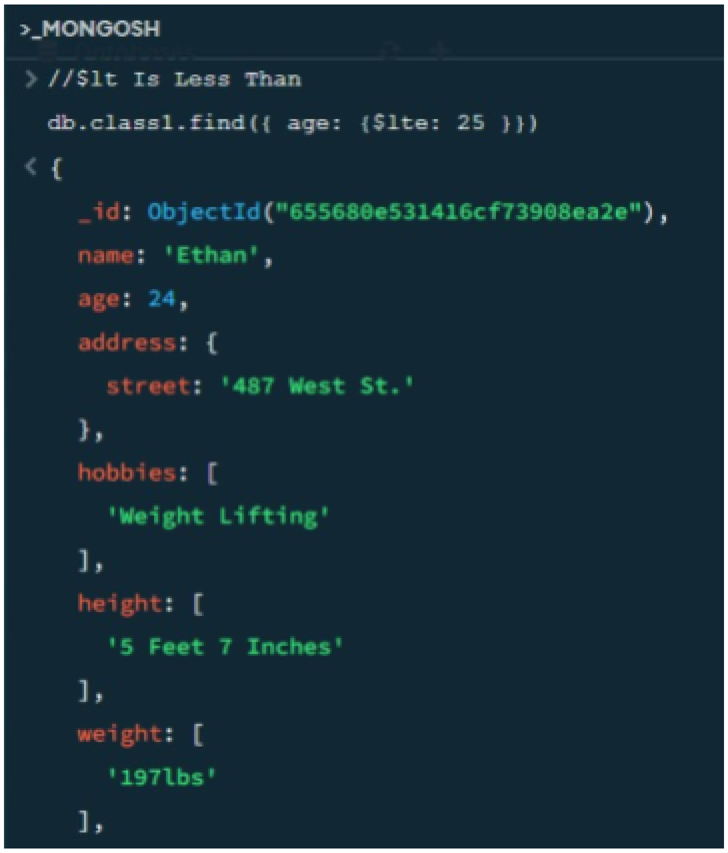
```
db.class1.find({ age: {$lte: 42}})
```



```
>_MONGOSH
> //$lte Is Less Than Or Equal To
  db.class1.find({ age: {$lte: 42}})
< {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
    address: {
      street: '983 South St.'
    },
    hobbies: [
      'Gaming'
    ],
    height: [
```

Query – limits the results to objects with the age *less than* 25

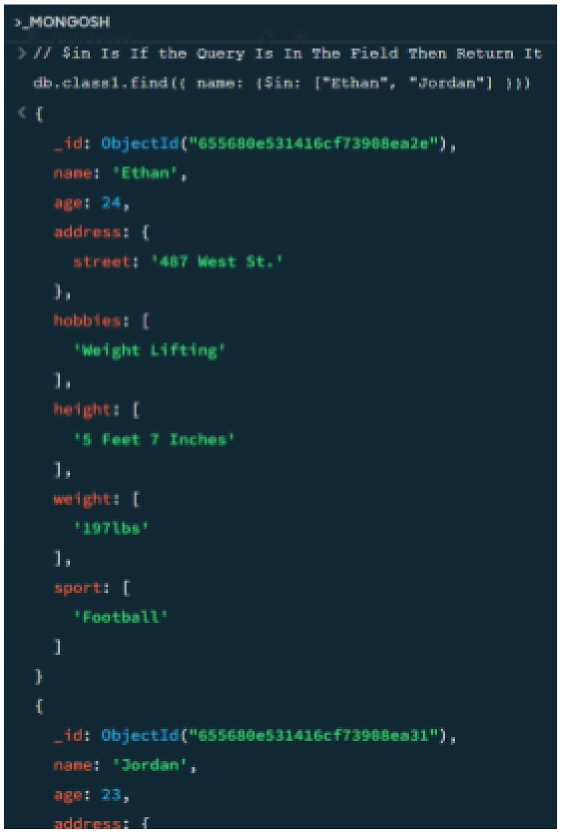
```
db.class1.find({ age: {$lte: 25 }})
```



```
>_MONGOSH
> //$lt Is Less Than
  db.class1.find({ age: {$lte: 25 }})
< {
    _id: ObjectId("655680e531416cf73908ea2e"),
    name: 'Ethan',
    age: 24,
    address: {
      street: '487 West St.'
    },
    hobbies: [
      'Weight Lifting'
    ],
    height: [
      '5 Feet 7 Inches'
    ],
    weight: [
      '197lbs'
    ],
```

Query – Uses the *$in* function to find only the objects with the specified data

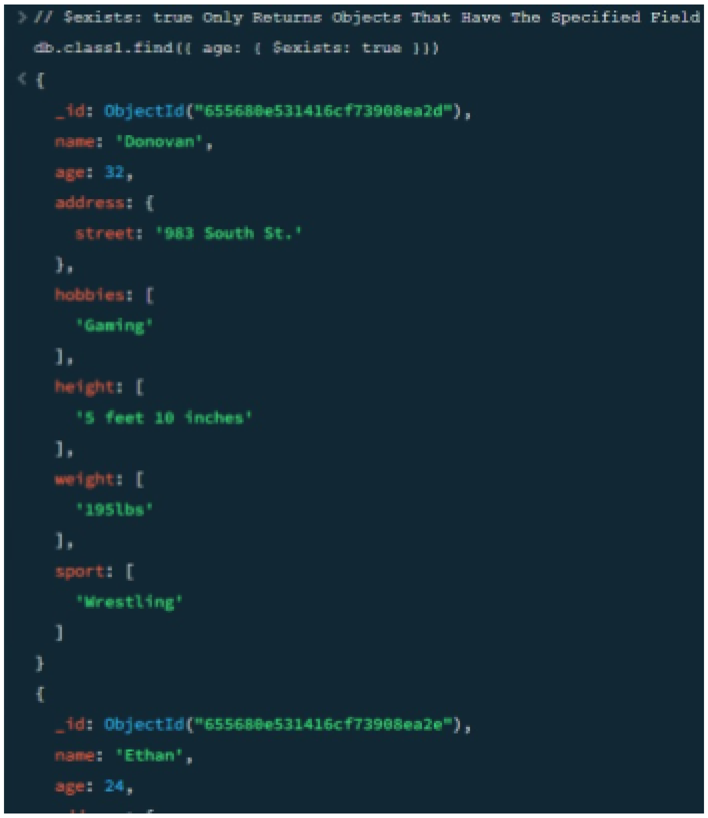
```
db.class1.find({ name: {$in: ["Ethan", "Jordan"] }})
```



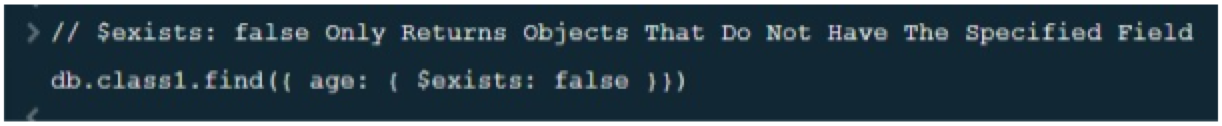
```
>_MONGOSH
> // $in Is If the Query Is In The Field Then Return It
  db.class1.find({ name: {$in: ["Ethan", "Jordan"] }})
< {
    _id: ObjectId("655680e531416cf73908ea2e"),
    name: 'Ethan',
    age: 24,
    address: {
      street: '487 West St.'
    },
    hobbies: [
      'Weight Lifting'
    ],
    height: [
      '5 Feet 7 Inches'
    ],
    weight: [
      '197lbs'
    ],
    sport: [
      'Football'
    ]
  }
  {
    _id: ObjectId("655680e531416cf73908ea31"),
    name: 'Jordan',
    age: 23,
    address: {
```

Query – limit the results to objects that have a value in that specified column

```
db.class1.find({ age: { $exists: true }})
```



```
> // $exists: true Only Returns Objects That Have The Specified Field
  db.class1.find({ age: { $exists: true }})
< {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
    address: {
      street: '983 South St.'
    },
    hobbies: [
      'Gaming'
    ],
    height: [
      '5 feet 10 inches'
    ],
    weight: [
      '195lbs'
    ],
    sport: [
      'Wrestling'
    ]
  }
  {
    _id: ObjectId("655680e531416cf73908ea2e"),
    name: 'Ethan',
    age: 24,
```

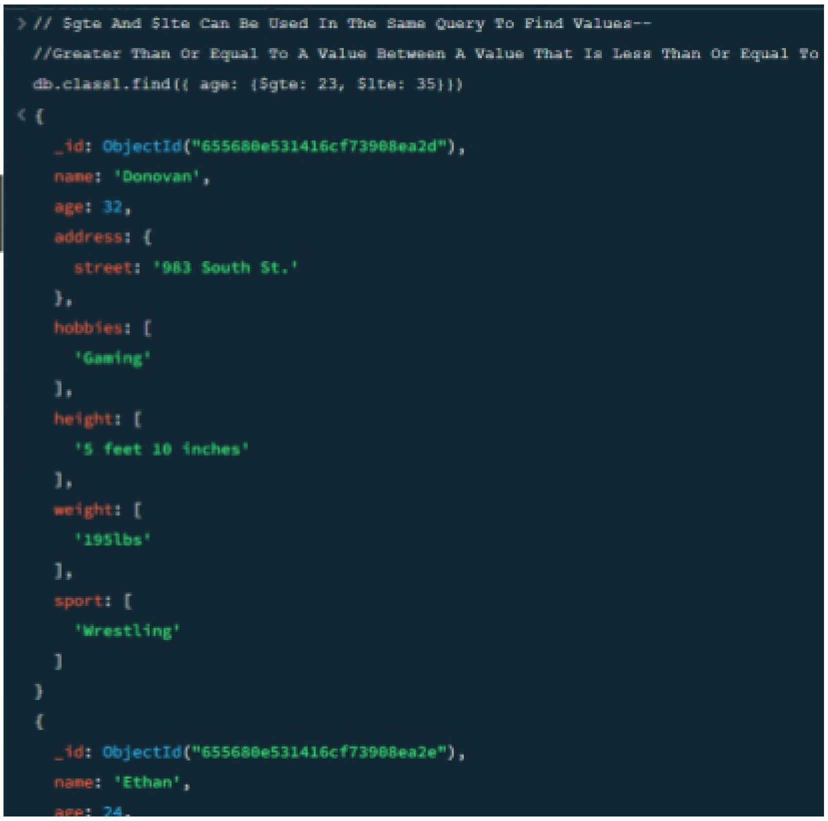


```
> // $exists: false Only Returns Objects That Do Not Have The Specified Field
  db.class1.find({ age: { $exists: false }})
<
```

//Nothing should be given as a result because all entries should have ages.

Query – Limiting the results to objects between the age of 23 and 35

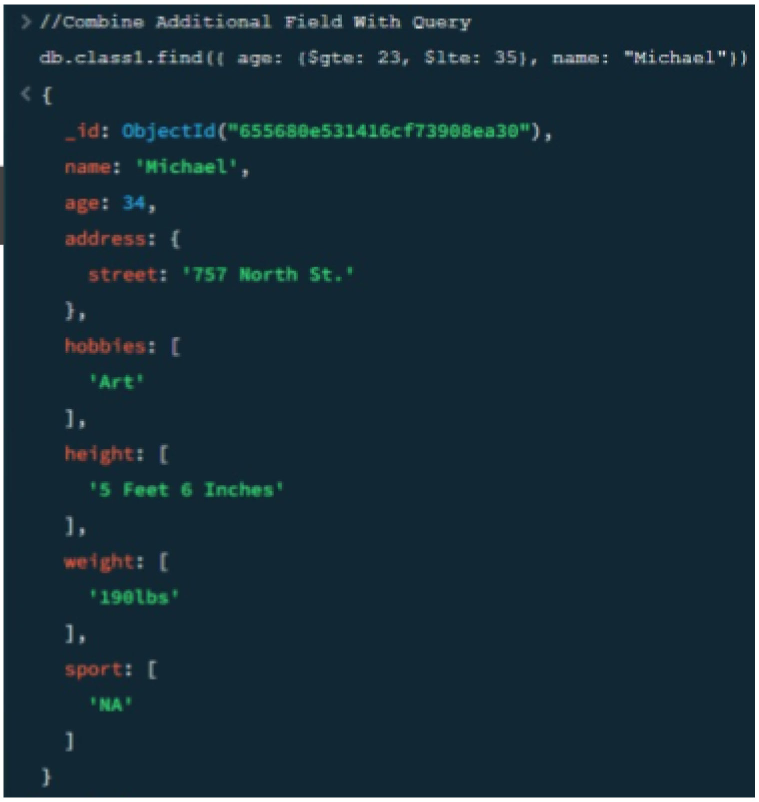
```
db.class1.find({ age: {$gte: 23,
        $lte: 35}})
```



```
> // $gte And $lte Can Be Used In The Same Query To Find Values--
  //Greater Than Or Equal To A Value Between A Value That Is Less Than Or Equal To
  db.class1.find({ age: {$gte: 23, $lte: 35}})
< {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
    address: {
      street: '983 South St.'
    },
    hobbies: [
      'Gaming'
    ],
    height: [
      '5 feet 10 inches'
    ],
    weight: [
      '195lbs'
    ],
    sport: [
      'Wrestling'
    ]
  }
  {
    _id: ObjectId("655680e531416cf73908ea2e"),
    name: 'Ethan',
    age: 24,
```

Query — Limit results to objects between the ages of 23 and 35 with the name of Michael

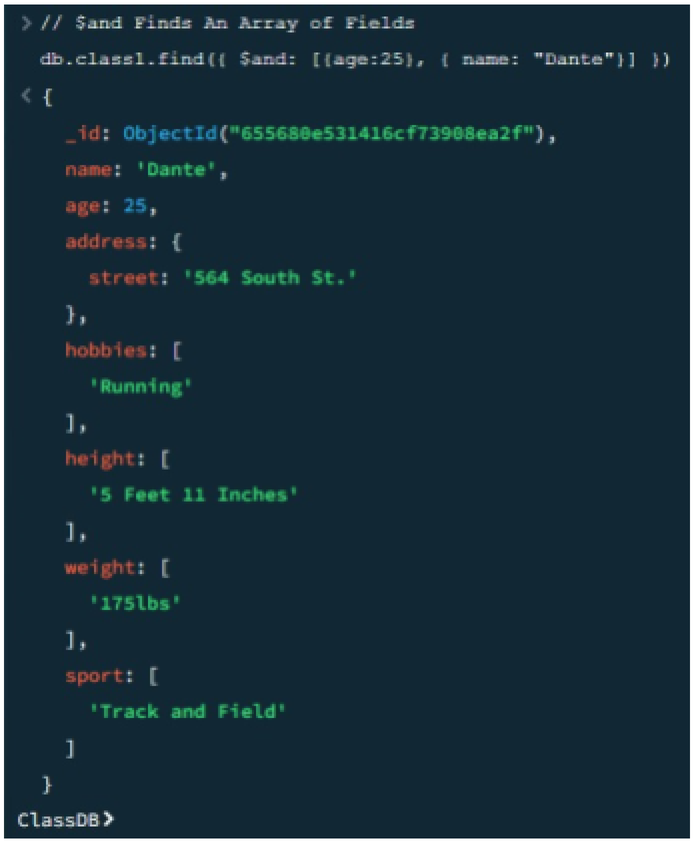
```
db.class1.find({ age: {$gte: 23, $lte:
      35}, name: "Michael"})
```



```
> //Combine Additional Field With Query
  db.class1.find({ age: {$gte: 23, $lte: 35}, name: "Michael"})
< {
    _id: ObjectId("655680e531416cf73908ea30"),
    name: 'Michael',
    age: 34,
    address: {
      street: '757 North St.'
    },
    hobbies: [
      'Art'
    ],
    height: [
      '5 Feet 6 Inches'
    ],
    weight: [
      '190lbs'
    ],
    sport: [
      'NA'
    ]
  }
```

Query — Limiting the results to just the age and name of these values

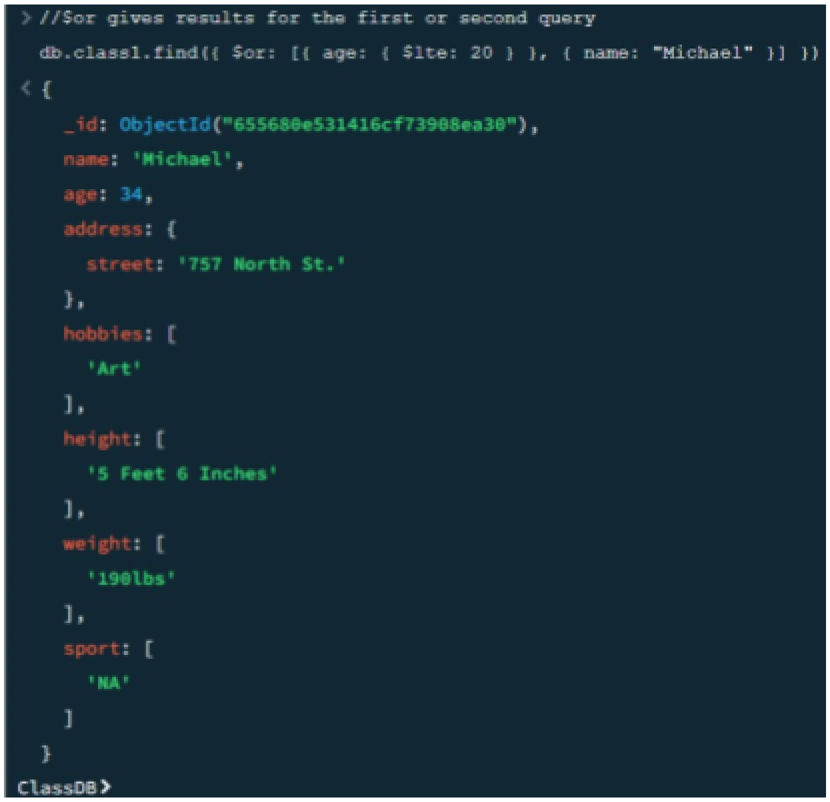
```
db.class1.find({ $and: [{age:25}, { name:
            "Dante"}] })
```



```
> // $and Finds An Array of Fields
  db.class1.find({ $and: [{age:25}, { name: "Dante"}] })
< {
    _id: ObjectId("655680e531416cf73908ea2f"),
    name: 'Dante',
    age: 25,
    address: {
      street: '564 South St.'
    },
    hobbies: [
      'Running'
    ],
    height: [
      '5 Feet 11 Inches'
    ],
    weight: [
      '175lbs'
    ],
    sport: [
      'Track and Field'
    ]
  }
ClassDB>
```

Query – Searching for *two* values in separate columns

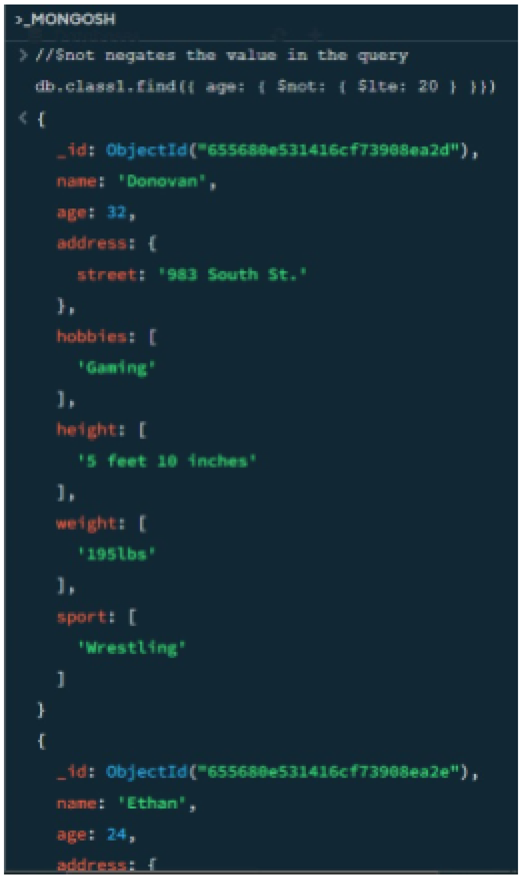
```
db.class1.find({ $or: [{ age: {
$lte: 20 }}, {name: "Michael" }]
})
```



```
> //$or gives results for the first or second query
  db.class1.find({ $or: [{ age: { $lte: 20 } }, { name: "Michael" }] })
< {
    _id: ObjectId("655680e531416cf73908ea30"),
    name: 'Michael',
    age: 34,
    address: {
      street: '757 North St.'
    },
    hobbies: [
      'Art'
    ],
    height: [
      '5 Feet 6 Inches'
    ],
    weight: [
      '190lbs'
    ],
    sport: [
      'NA'
    ]
  }
ClassDB>
```

Query – Objects that are *not* less than 20 in the column for age

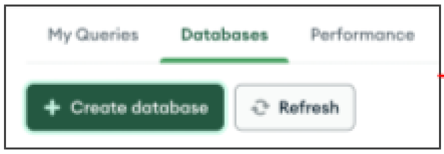
```
db.class1.find({ age: { $not: {$lte: 20} }})
```



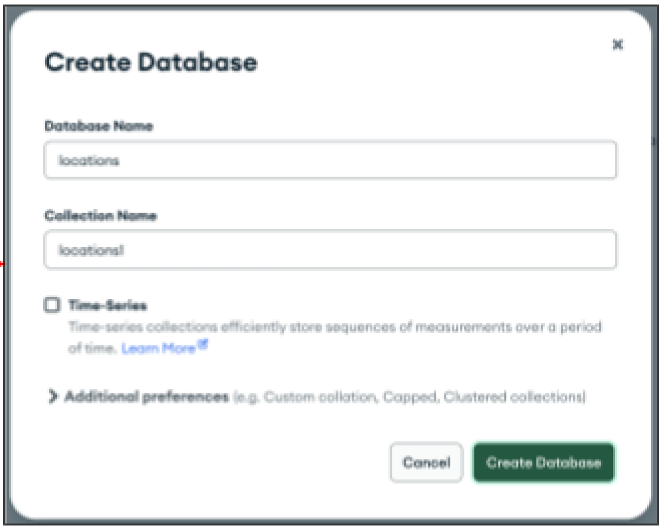
```
>_MONGOSH
> //$not negates the value in the query
  db.class1.find({ age: { $not: { $lte: 20 } }})
< {
    _id: ObjectId("655680e531416cf73908ea2d"),
    name: 'Donovan',
    age: 32,
    address: {
      street: '983 South St.'
    },
    hobbies: [
      'Gaming'
    ],
    height: [
      '5 feet 10 inches'
    ],
    weight: [
      '195lbs'
    ],
    sport: [
      'Wrestling'
    ]
  }
  {
    _id: ObjectId("655680e531416cf73908ea2e"),
    name: 'Ethan',
    age: 24,
    address: {
```

Case 2: Geospatial Locations

Create your Database

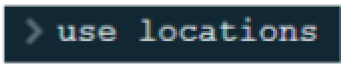


Database Name: locations
Collection Name: locations1



Database Name: locations
Collection Name: locations1
☐ 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)

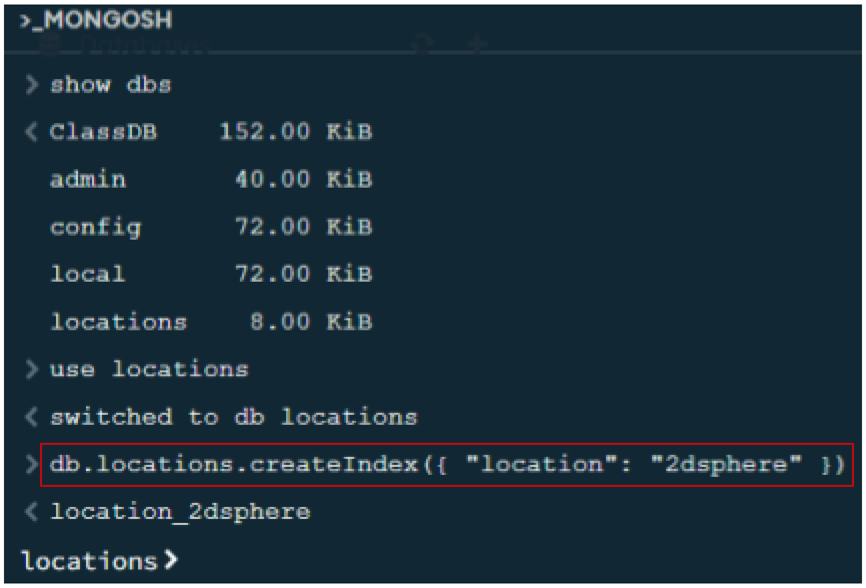
Make sure to switch to your *locations* database.



```
> use locations
```

After that, use this command to create a 2dsphere index.

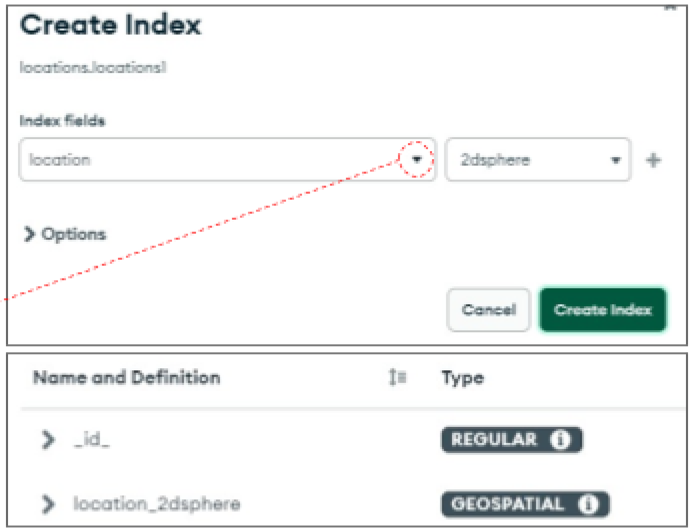
A 2dsphere index supports queries that calculate geometries on an earth-like sphere.



```
>_MONGOSH
> show dbs
< ClassDB     152.00 KiB
  admin        40.00 KiB
  config       72.00 KiB
  local        72.00 KiB
  locations     8.00 KiB
> use locations
< switched to db locations
> db.locations.createIndex({ "location": "2dsphere" })
< location_2dsphere
locations>
```

Another way to create an index is by going to the *Index* tab in your selected collection.

Clicking this would show a dropdown menu. Select *location*.



locations.locations1

Index fields: location | 2dsphere

Name and Definition	Type
_id_	REGULAR
location_2dsphere	GEOSPATIAL

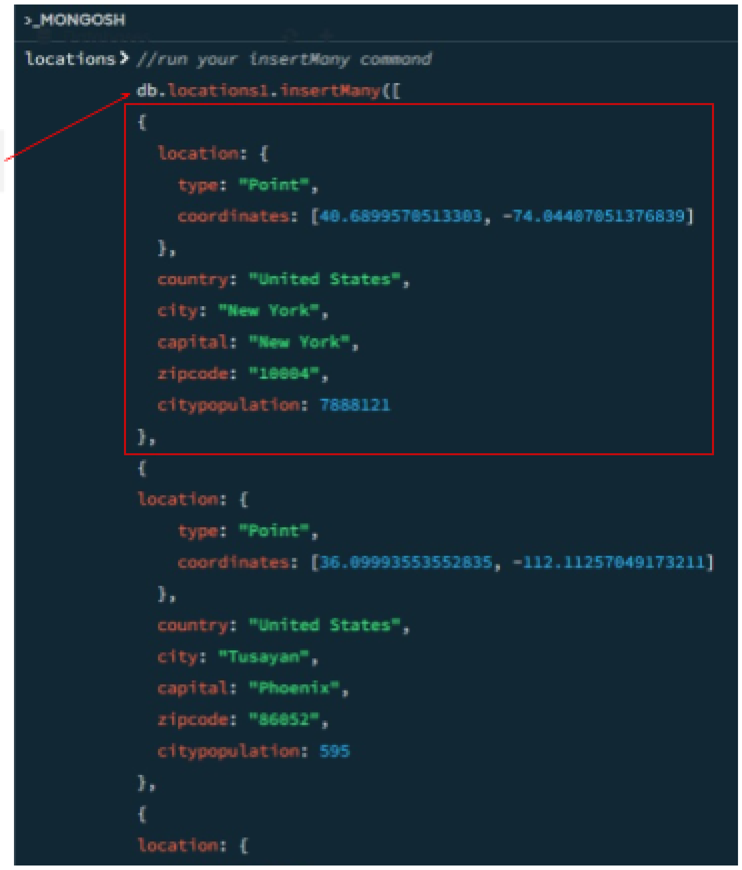
Case 2: Inserting Data

```
db.collection_name.insertMany([
```

You can get coordinates from Google Maps by right clicking.

If an error occurs when you run your code saying that the Longitude/Latitude is out of bounds, you may have to switch the coordinates around.

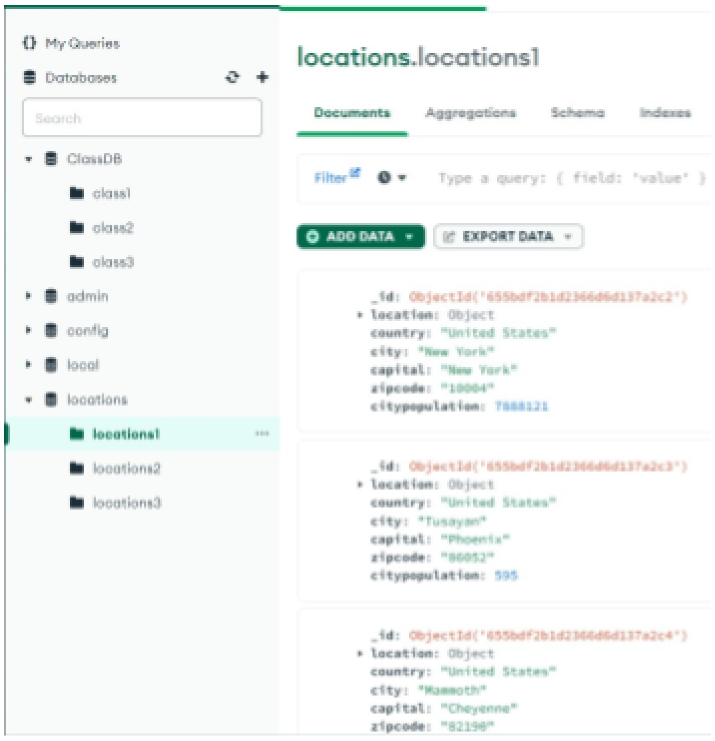
Instead of `[lat, lng]` try `[lng, lat]`



```
>_MONGOSH
locations> //run your insertMany command
           db.locations1.insertMany([
           {
             location: {
               type: "Point",
               coordinates: [40.6899570513303, -74.04407051376839]
             },
             country: "United States",
             city: "New York",
             capital: "New York",
             zipcode: "10004",
             citypopulation: 7888121
           },
           {
           location: {
               type: "Point",
               coordinates: [36.09993553552835, -112.11257049173211]
             },
             country: "United States",
             city: "Tusayan",
             capital: "Phoenix",
             zipcode: "86052",
             citypopulation: 595
           },
           {
           location: {
```

Tourist spots in the United States.

This screenshot shows the documents in the *locations1* collection.

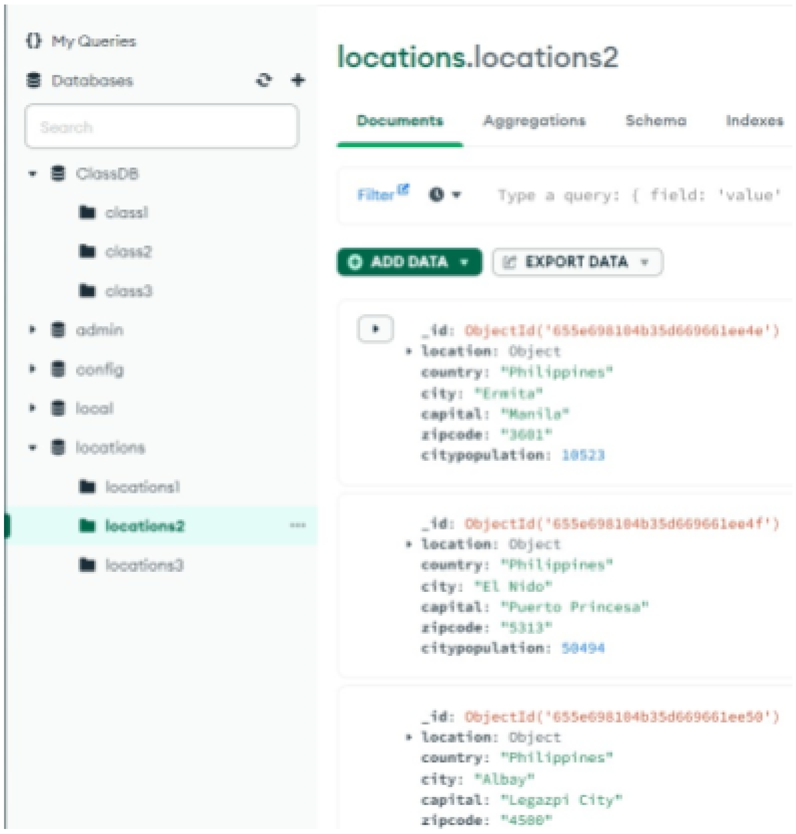


locations.locations1

```
Type a query: { field: 'value' }

_id: ObjectId('655bdf2b1d2366d6d137a2c2')
▸ location: Object
country: "United States"
city: "New York"
capital: "New York"
zipcode: "10004"
citypopulation: 7888121

_id: ObjectId('655bdf2b1d2366d6d137a2c3')
▸ location: Object
country: "United States"
city: "Tusayan"
capital: "Phoenix"
zipcode: "86052"
citypopulation: 595

_id: ObjectId('655bdf2b1d2366d6d137a2c4')
▸ location: Object
country: "United States"
city: "Mammoth"
capital: "Cheyenne"
zipcode: "82190"
```

Tourist spots in the Philippines.

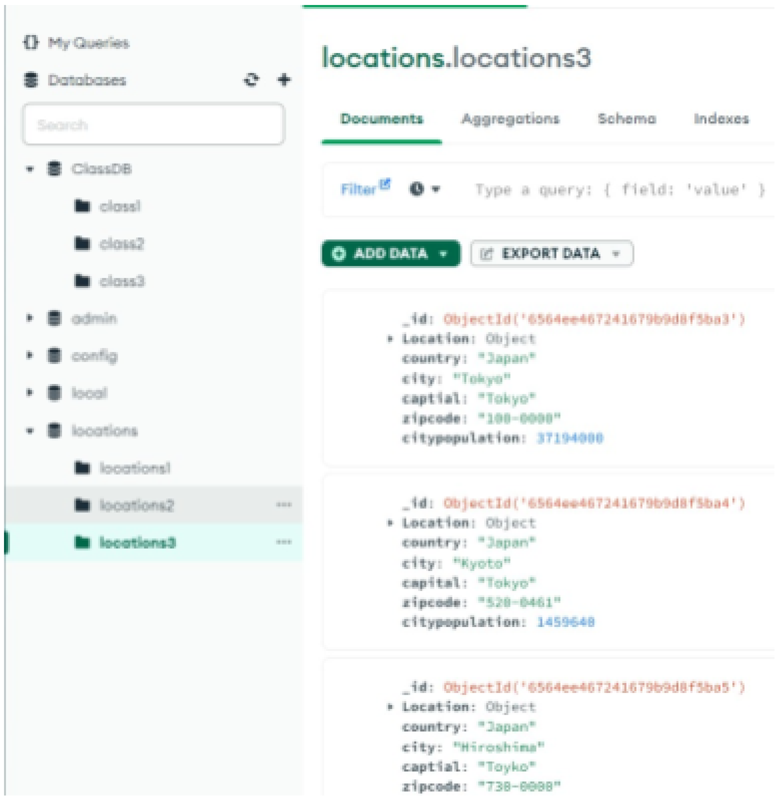
This screenshot shows the documents in the *locations2* collection.



```
_id: ObjectId('655e698104b35d669661ee4e')
▸ location: Object
  country: "Philippines"
  city: "Ermita"
  capital: "Manila"
  zipcode: "3601"
  citypopulation: 10523

_id: ObjectId('655e698104b35d669661ee4f')
▸ location: Object
  country: "Philippines"
  city: "El Nido"
  capital: "Puerto Princesa"
  zipcode: "5313"
  citypopulation: 50494

_id: ObjectId('655e698104b35d669661ee50')
▸ location: Object
  country: "Philippines"
  city: "Albay"
  capital: "Legazpi City"
  zipcode: "4500"
```

Tourist spots in the Japan.

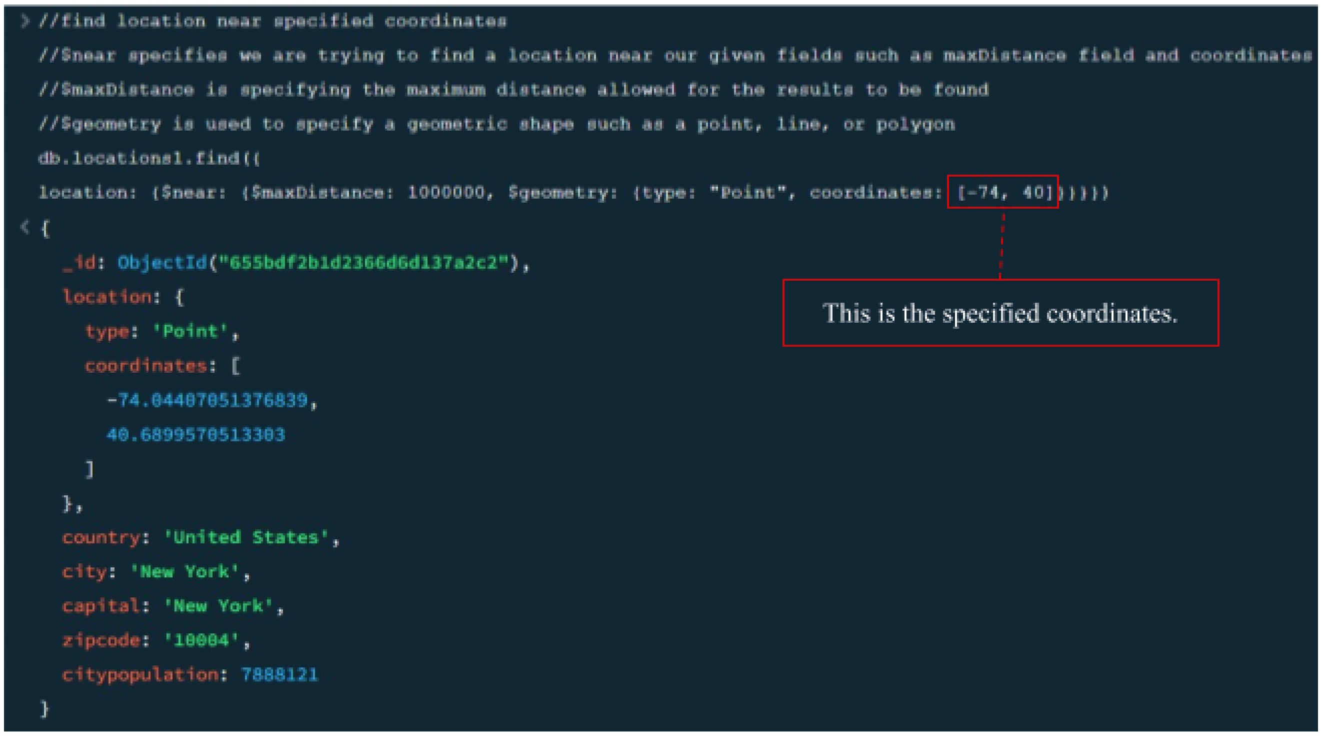
This screenshot shows the documents in the *locations3* collection.



```
_id: ObjectId('6564ee467241679b9d8f5ba3')
▸ Location: Object
  country: "Japan"
  city: "Tokyo"
  captial: "Tokyo"
  zipcode: "100-0000"
  citypopulation: 37194000

_id: ObjectId('6564ee467241679b9d8f5ba4')
▸ Location: Object
  country: "Japan"
  city: "Kyoto"
  capital: "Tokyo"
  zipcode: "520-0461"
  citypopulation: 1459640

_id: ObjectId('6564ee467241679b9d8f5ba5')
▸ Location: Object
  country: "Japan"
  city: "Hiroshima"
  captial: "Toyko"
  zipcode: "730-0000"
```

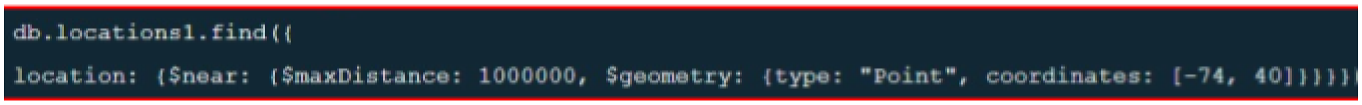
Case 2: Query Commands

Query — Sorting locations from nearest to farthest from specified coordinates.

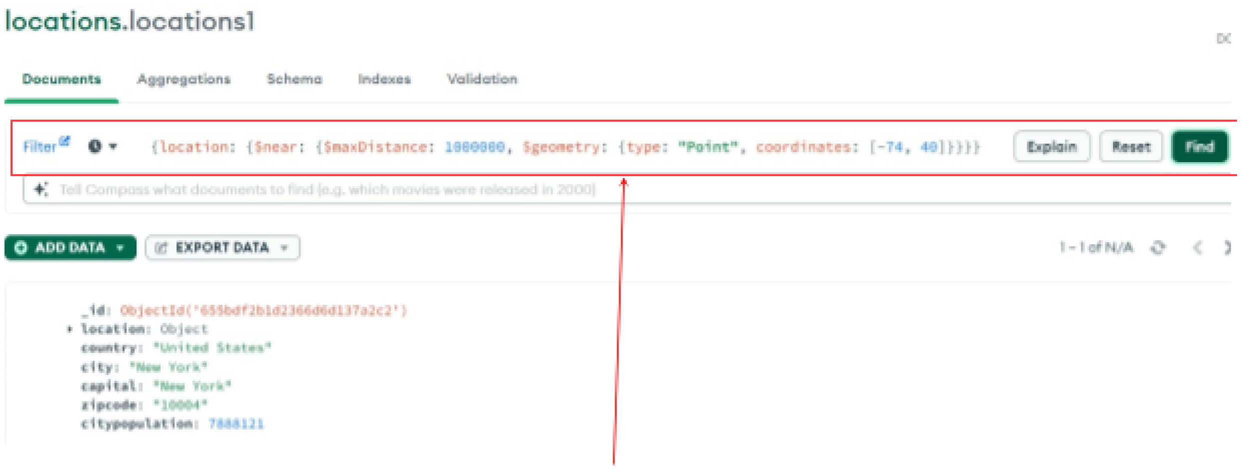


```
> //find location near specified coordinates
  //$near specifies we are trying to find a location near our given fields such as maxDistance field and coordinates
  //$maxDistance is specifying the maximum distance allowed for the results to be found
  //$geometry is used to specify a geometric shape such as a point, line, or polygon
  db.locations1.find({
  location: {$near: {$maxDistance: 1000000, $geometry: {type: "Point", coordinates: [-74, 40]}}}})
< {
    _id: ObjectId("655bdf2b1d2366d6d137a2c2"),
    location: {
      type: 'Point',
      coordinates: [
        -74.04407051376839,
        40.6899570513303
      ]
    },
    country: 'United States',
    city: 'New York',
    capital: 'New York',
    zipcode: '10004',
    citypopulation: 7888121
  }
```

This is the specified coordinates.



```
db.locations1.find({
location: {$near: {$maxDistance: 1000000, $geometry: {type: "Point", coordinates: [-74, 40]}}}})
```



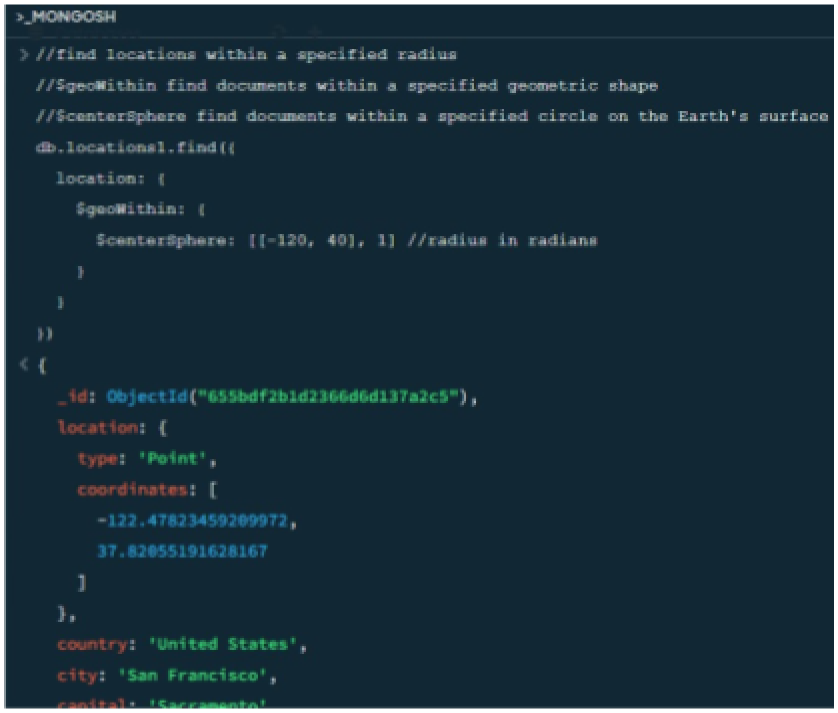
locations.locations1

```
{location: {$near: {$maxDistance: 1000000, $geometry: {type: "Point", coordinates: [-74, 40]}}}}
```

```
_id: ObjectId('655bdf2b1d2366d6d137a2c2')
▸ location: Object
  country: "United States"
  city: "New York"
  capital: "New York"
  zipcode: "10004"
  citypopulation: 7888121
```

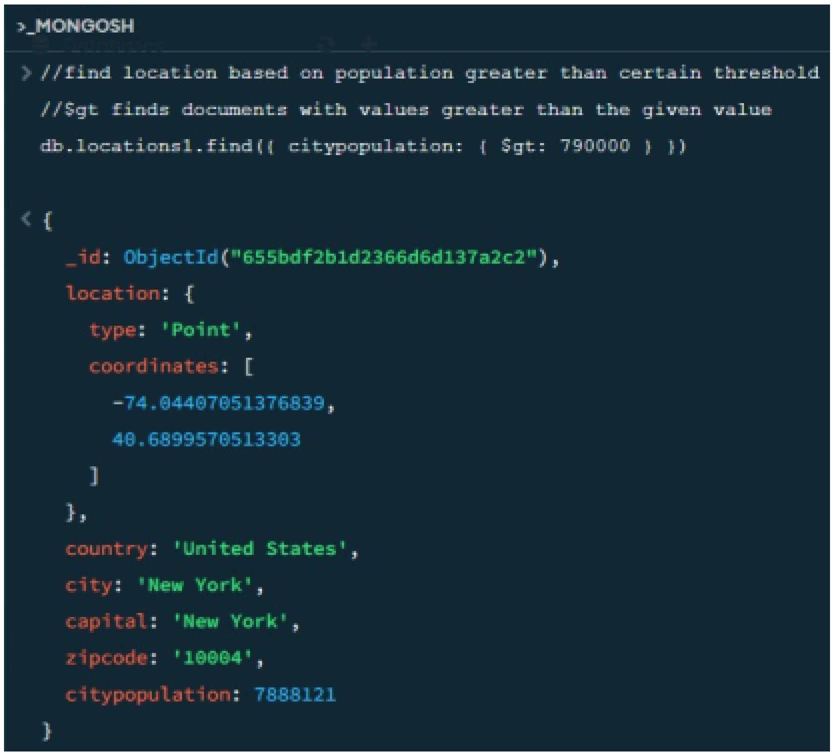
In Compass, code can be typed into the command line as well instead of typing in the shell. When finished with typing click "Find"

Query – searching for objects with matching coordinate values within the radius of the circle



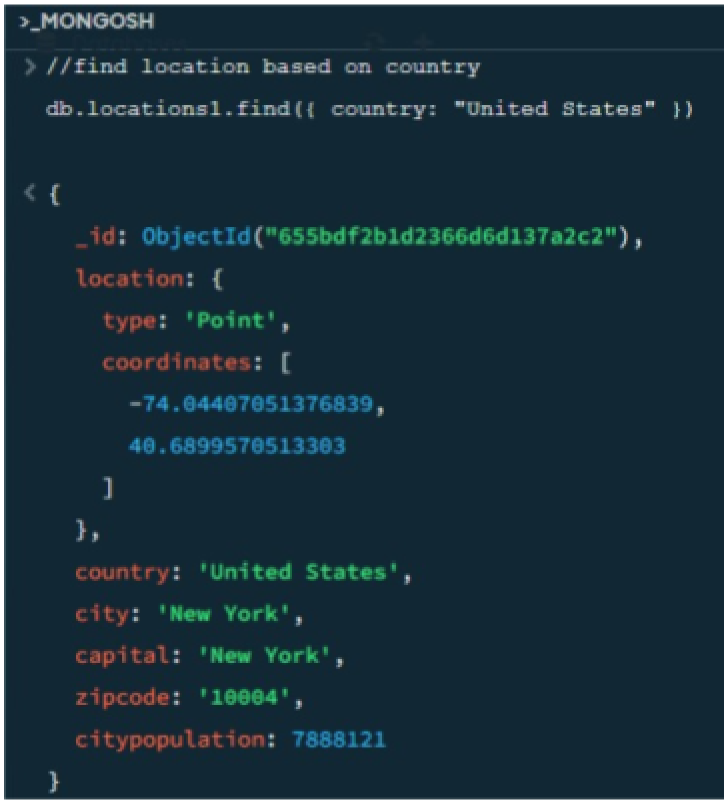
```
>_MONGOSH
> //find locations within a specified radius
  //$geoWithin find documents within a specified geometric shape
  //$centerSphere find documents within a specified circle on the Earth's surface
  db.locations1.find({
    location: {
      $geoWithin: {
        $centerSphere: [[-120, 40], 1] //radius in radians
      }
    }
  })
< {
    _id: ObjectId("655bdf2b1d2366d6d137a2c5"),
    location: {
      type: 'Point',
      coordinates: [
        -122.47823459209972,
        37.82055191628167
      ]
    },
    country: 'United States',
    city: 'San Francisco',
    capital: 'Sacramento'
```

Query – uses the *$gt* to find objects that have values greater than the inputted value of 790000 in the "city population" column



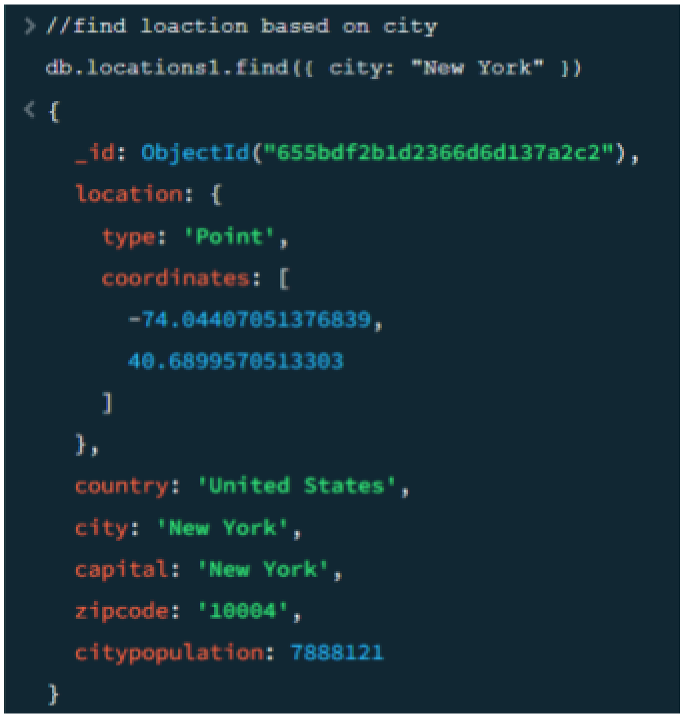
```
>_MONGOSH
> //find location based on population greater than certain threshold
  //$gt finds documents with values greater than the given value
  db.locations1.find({ citypopulation: { $gt: 790000 } })

< {
    _id: ObjectId("655bdf2b1d2366d6d137a2c2"),
    location: {
      type: 'Point',
      coordinates: [
        -74.04407051376839,
        40.6899570513303
      ]
    },
    country: 'United States',
    city: 'New York',
    capital: 'New York',
    zipcode: '10004',
    citypopulation: 7888121
  }
```

Query – limits the results to objects with the string "United States" in the country column



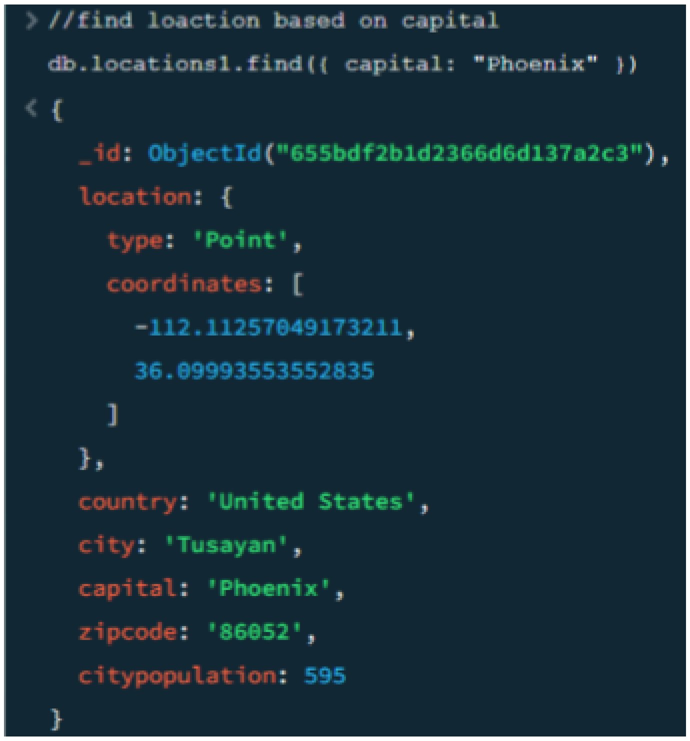
```
>_MONGOSH
> //find location based on country
  db.locations1.find({ country: "United States" })

< {
    _id: ObjectId("655bdf2b1d2366d6d137a2c2"),
    location: {
      type: 'Point',
      coordinates: [
        -74.04407051376839,
        40.6899570513303
      ]
    },
    country: 'United States',
    city: 'New York',
    capital: 'New York',
    zipcode: '10004',
    citypopulation: 7888121
  }
```

Query – limit the results to objects with the string "New York" in the city column



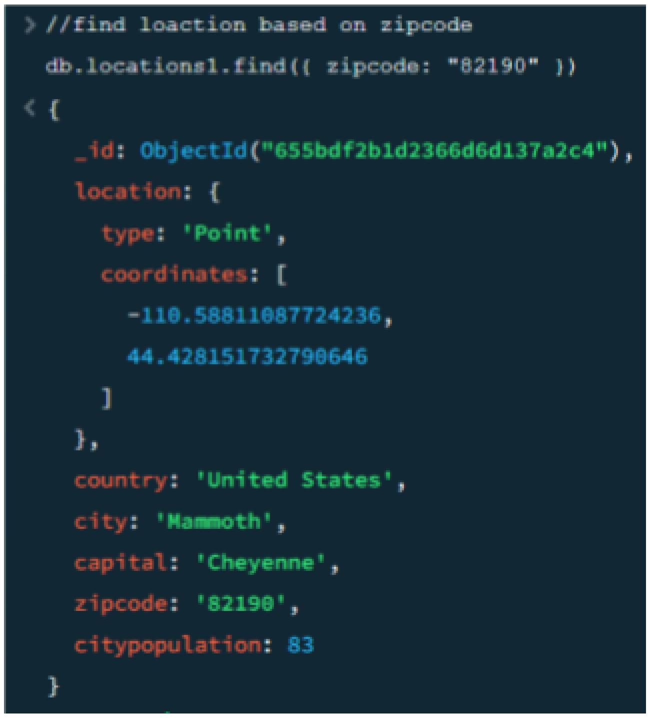
```
> //find loaction based on city
  db.locations1.find({ city: "New York" })
< {
    _id: ObjectId("655bdf2b1d2366d6d137a2c2"),
    location: {
      type: 'Point',
      coordinates: [
        -74.04407051376839,
        40.6899570513303
      ]
    },
    country: 'United States',
    city: 'New York',
    capital: 'New York',
    zipcode: '10004',
    citypopulation: 7888121
  }
```

Query – limits the results to objects with the string "Phoenix" in the capital column



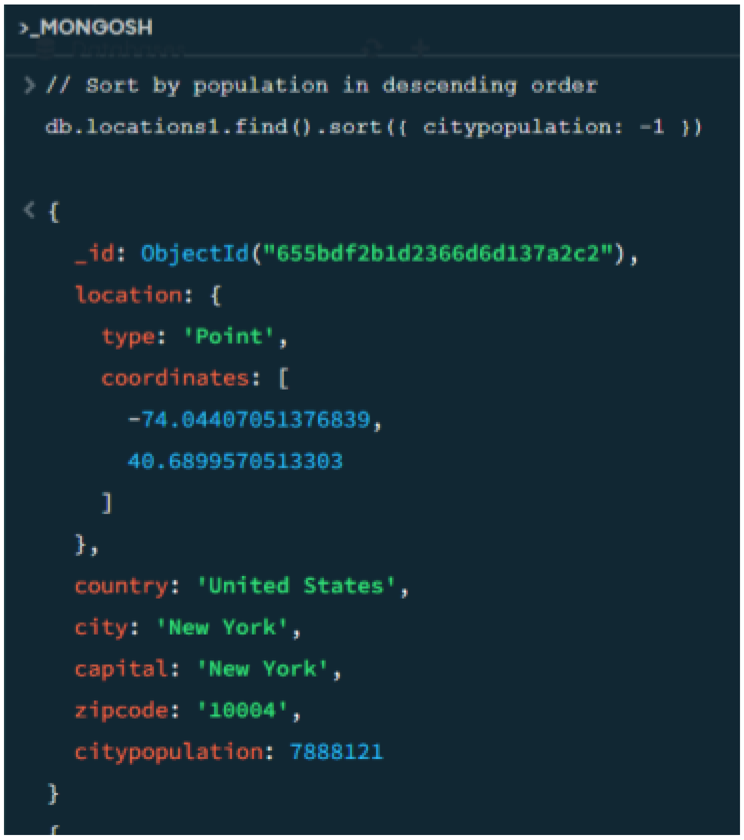
```
> //find loaction based on capital
  db.locations1.find({ capital: "Phoenix" })
< {
    _id: ObjectId("655bdf2b1d2366d6d137a2c3"),
    location: {
      type: 'Point',
      coordinates: [
        -112.11257049173211,
        36.09993553552835
      ]
    },
    country: 'United States',
    city: 'Tusayan',
    capital: 'Phoenix',
    zipcode: '86052',
    citypopulation: 595
  }
```

Query – limits the results to find objects with the value of "82190" in the zipcode column



```
> //find loaction based on zipcode
  db.locations1.find({ zipcode: "82190" })
< {
    _id: ObjectId("655bdf2b1d2366d6d137a2c4"),
    location: {
      type: 'Point',
      coordinates: [
        -110.58811087724236,
        44.428151732790646
      ]
    },
    country: 'United States',
    city: 'Mammoth',
    capital: 'Cheyenne',
    zipcode: '82190',
    citypopulation: 83
  }
```

Query – reorganizes all objects in *descending* order of city population



```
>_MONGOSH
> // Sort by population in descending order
  db.locations1.find().sort({ citypopulation: -1 })

< {
    _id: ObjectId("655bdf2b1d2366d6d137a2c2"),
    location: {
      type: 'Point',
      coordinates: [
        -74.04407051376839,
        40.6899570513303
      ]
    },
    country: 'United States',
    city: 'New York',
    capital: 'New York',
    zipcode: '10004',
    citypopulation: 7888121
  }
```

Useful Links

MongoDB Crash Course:
https://www.youtube.com/watch?v=ofme2o29ngU

MongoDB download for Mac (youtube tutorial):
https://youtu.be/MyIiM7z_j_Y?si=Prbb65z1gM3l00_M

Geospatial Queries — MongoDB Manual:
https://www.mongodb.com/docs/manual/geospatial-queries/

Short Summary:

MongoDB is written in Java language which has a little bit of a learning curve especially because it is case sensitive. However, once we got used to it, we found MongoDB to be very convenient and easy to use. The shell followed what we've learned in MySQL and automatically generated columns and tables for us. The application provided a very conventional method of querying and seeing databases and objects laid out was easy on the eyes. Overall, MongoDB was very useful compared to MySQL.

References

(n.d.). MongoDB: The Developer Data Platform | MongoDB. Retrieved November 15, 2023, from https://www.mongodb.com/

Comparing The Differences - MongoDB Vs MySQL. (n.d.). MongoDB. Retrieved November 15, 2023, from https://www.mongodb.com/compare/mongodb-mysql

Download MongoDB Community Server. (n.d.). MongoDB. Retrieved November 15, 2023, from https://www.mongodb.com/try/download/community

The mongo Shell — MongoDB Manual. (n.d.). MongoDB. Retrieved November 15, 2023, from https://www.mongodb.com/docs/v4.4/mongo/

What is MongoDB? — MongoDB Manual. (n.d.). MongoDB. Retrieved November 15, 2023, from https://www.mongodb.com/docs/v4.4/

GitHub

Brandon Koskie

Ashley Alfaro

Cade Garcia