Add Cover Add Subtitle

MongoDB

How to add multiple records together?

To add multiple records at once in MongoDB we can use:

JavaScript

```
db.collectionName.insertMany([
    {
        key1: value1,
        key2: value2
    },
    {
        key1: value1,
        key2: value2,
        key3: value3,
        .
        .
        .
    },
    {
        key5: value5,
```

```
            key7: value7,
            .
            .
        }
    ]);
```

```
University> db.students.insertMany([ {name: 'Sarthak', standard: 11, marks: 80}, {name: 'JD', marks: 90, age: 18}, {nam
e: 'Tanmay', marks: 75, rollno: 2112, phone: 98998899}]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6471f4c5072f5055404eb40e"),
    '1': ObjectId("6471f4c5072f5055404eb40f"),
    '2': ObjectId("6471f4c5072f5055404eb410")
  }
}
```

Can we add an array in the JSON in MongoDB?

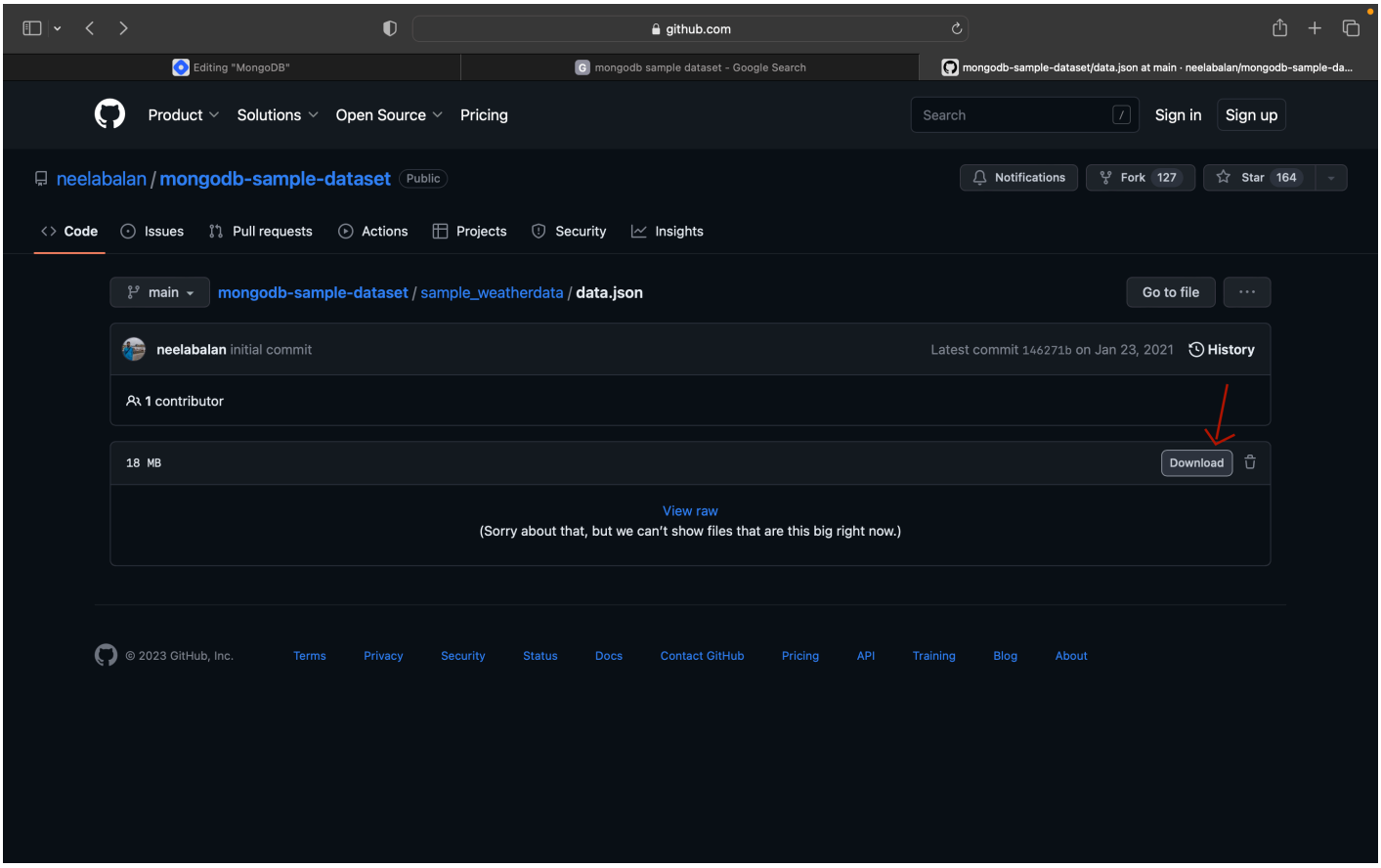
We can simply use `[]` to add an array.

```
University> db.students.insertOne({name: 'Anurag', subjects: ["Web", "Java"]});
{
  acknowledged: true,
  insertedId: ObjectId("6471f5a2072f5055404eb411")
}
```

How to import sample DB in MongoDB?

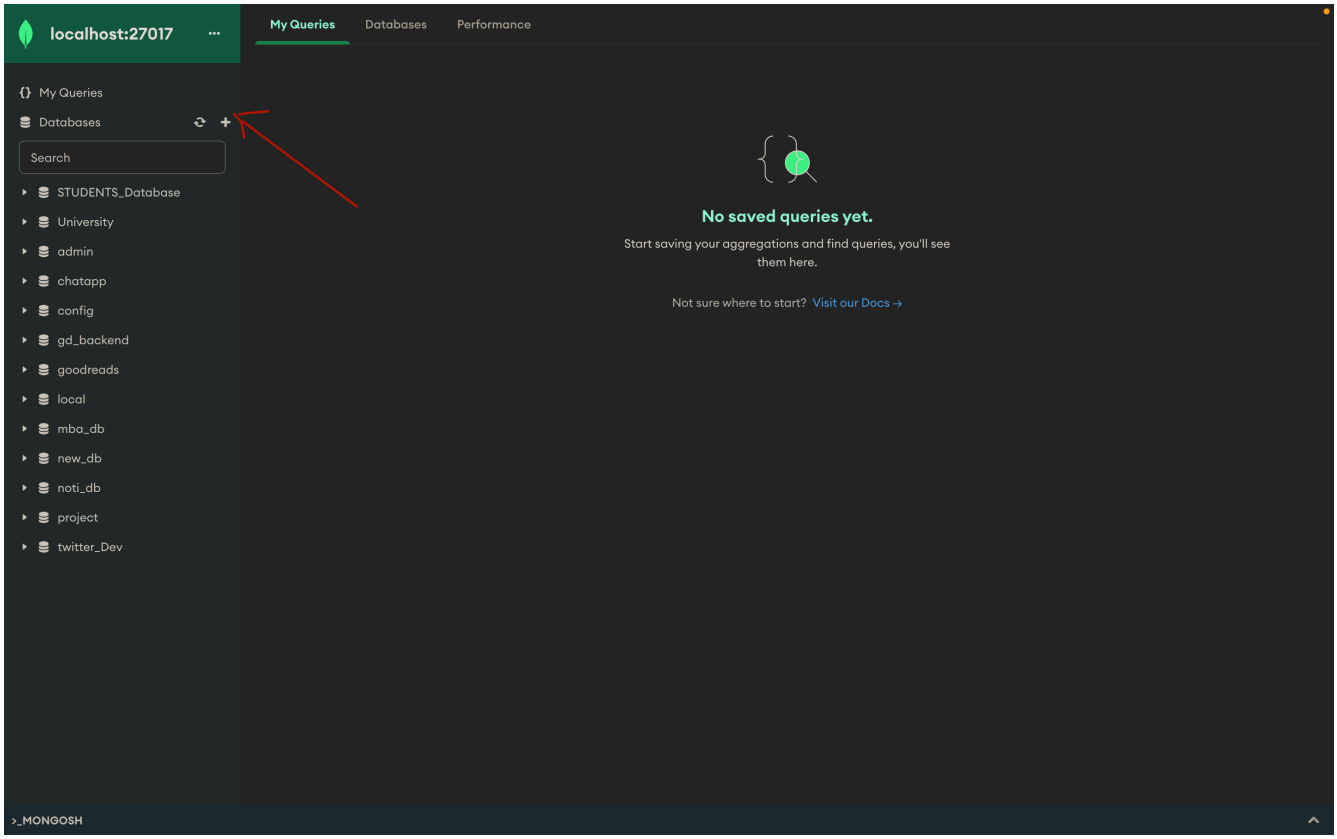
To get some free sample datasets you can check out this Github link: https://github.com/neelabalan/mongodb-sample-dataset.

You can pick any folder and download the dataset which is present in JSON format.

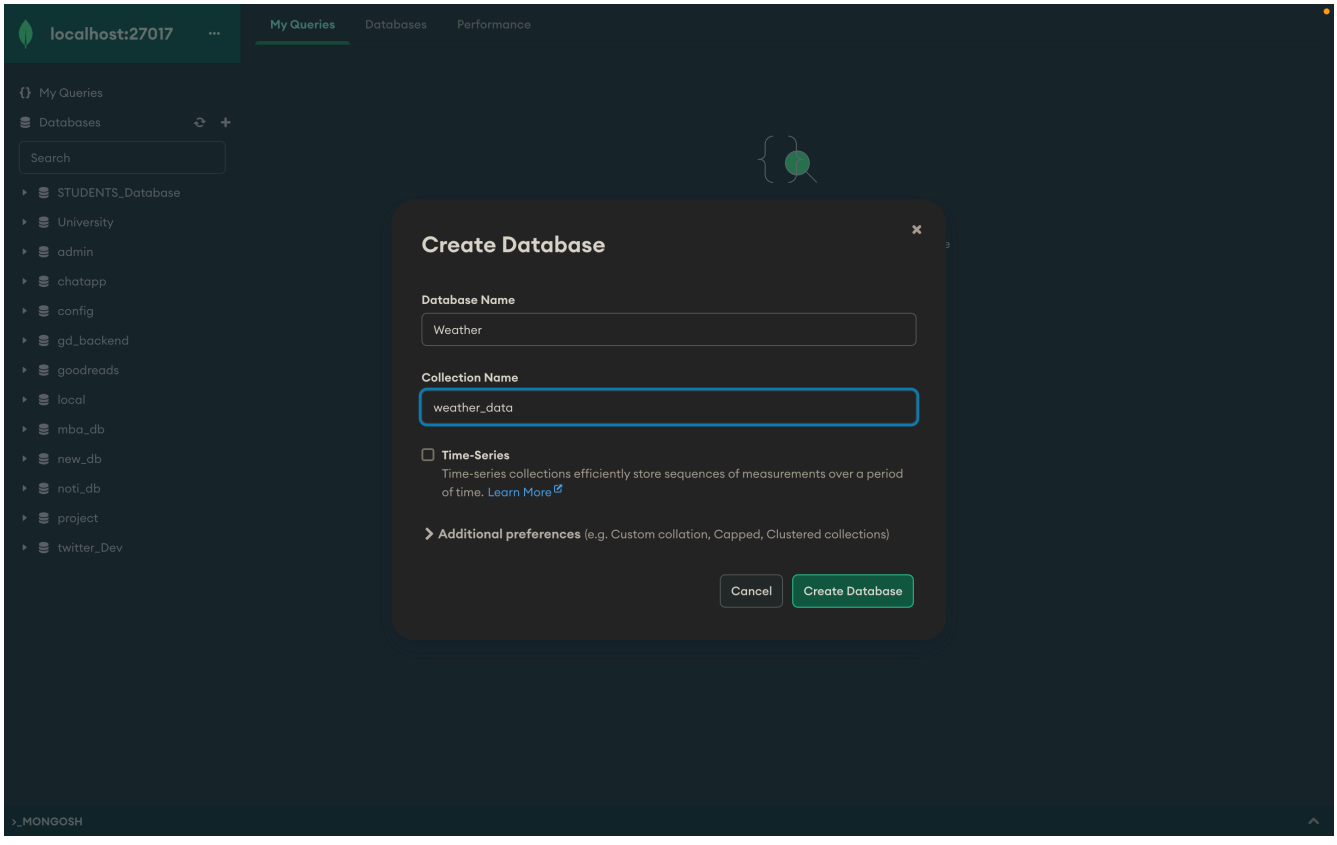


After the file is downloaded you can import this data using MongoDB Compass.

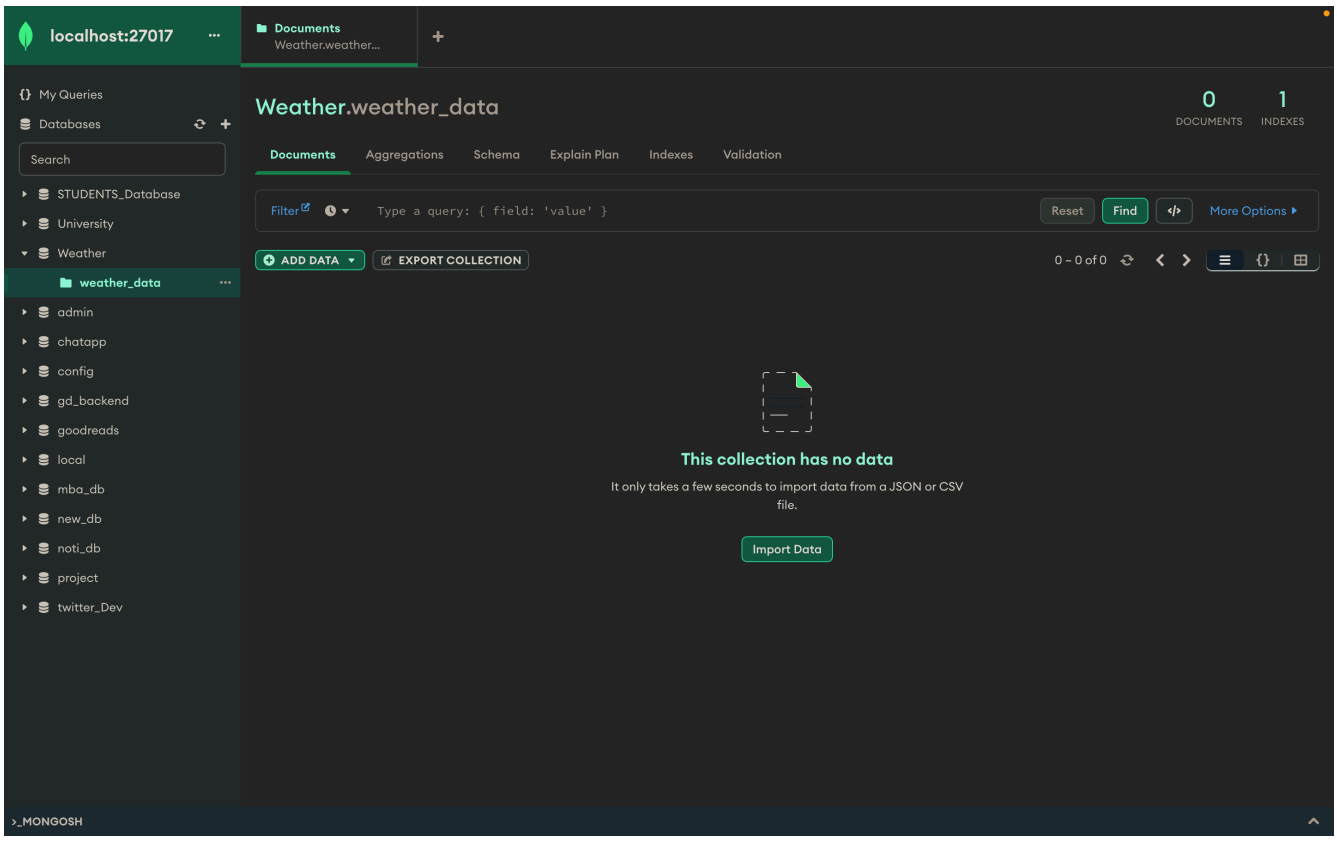
- Create a new Database (we can directly create it using MongoDB Compass)



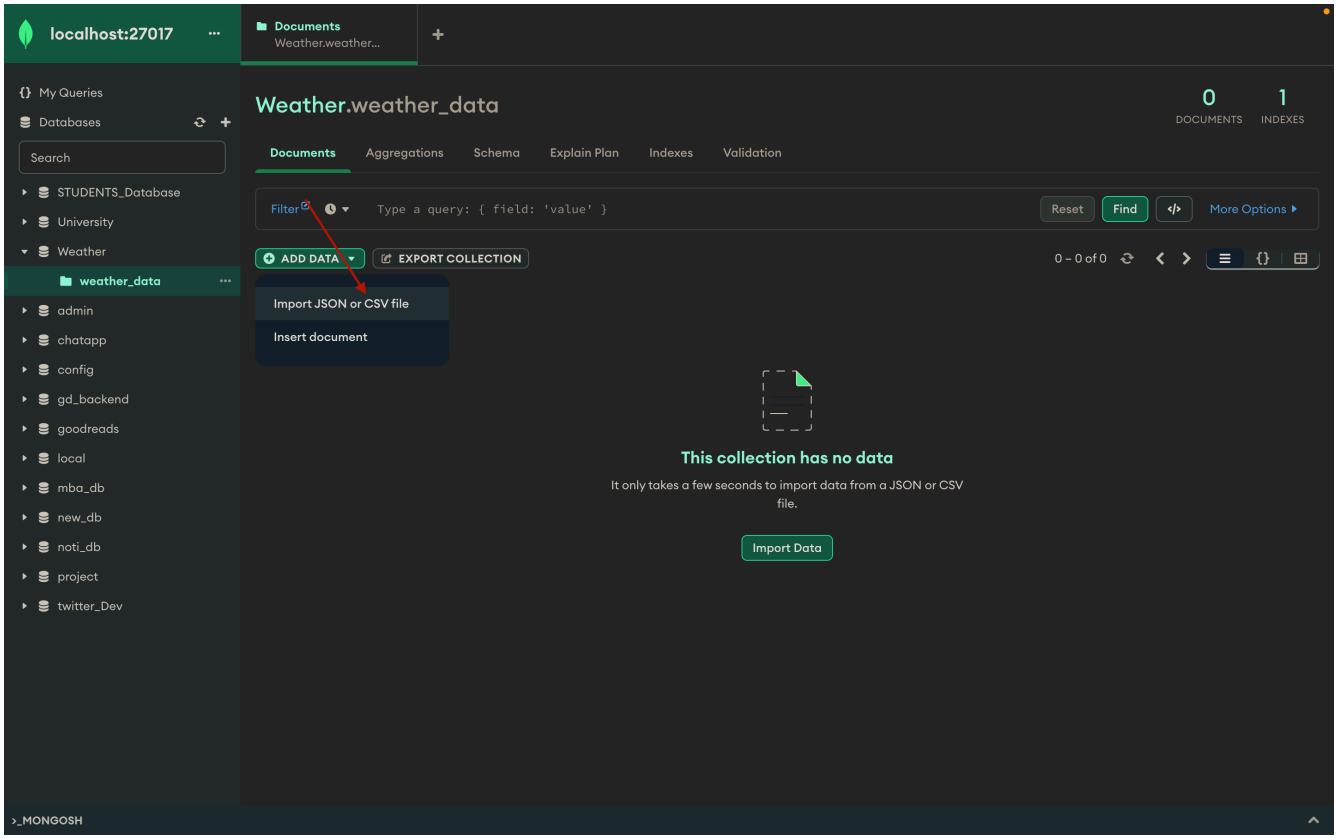
- We can give a name to the database and give the first collection name also directly using MongoDB Compass.



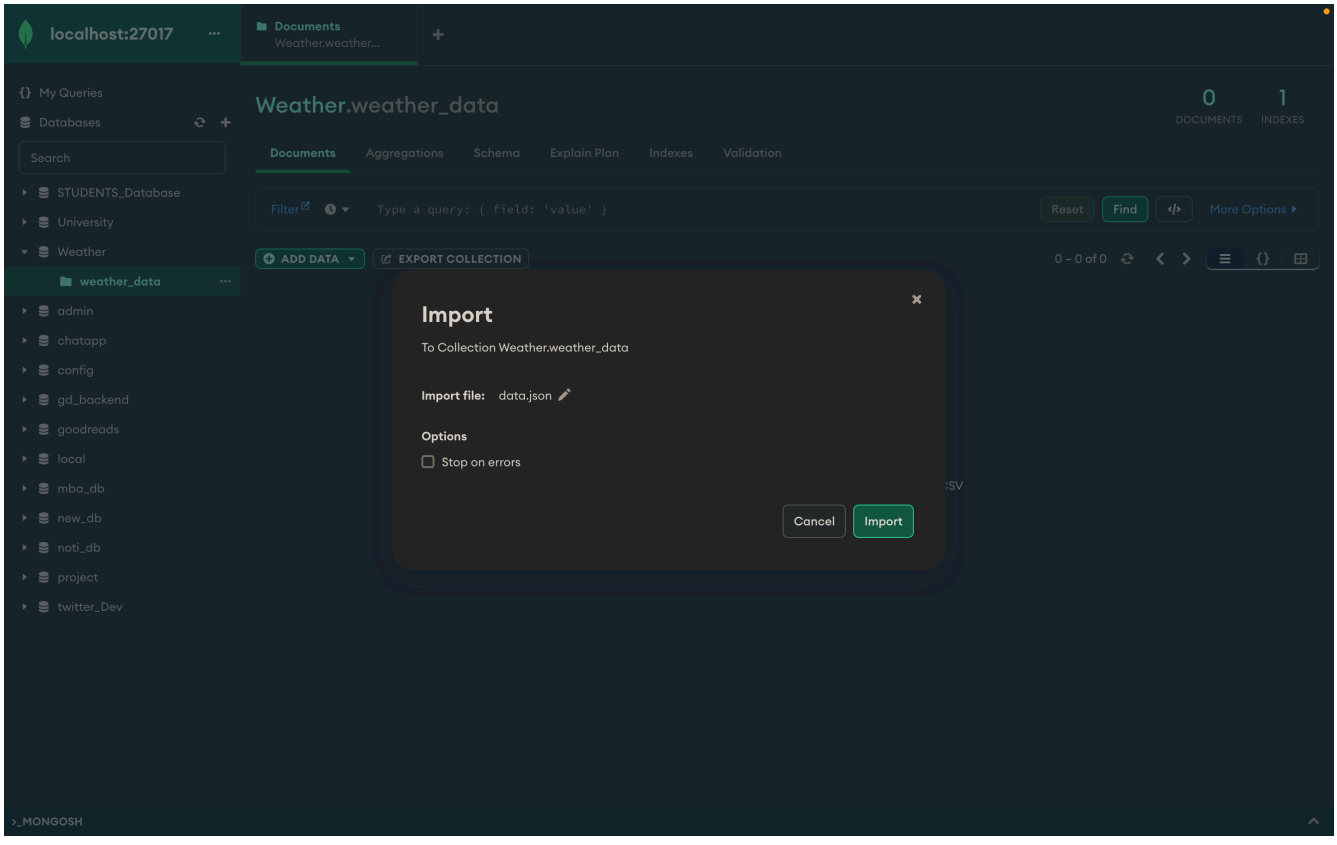
- Once you click on create a database, a brand new empty db will be created.



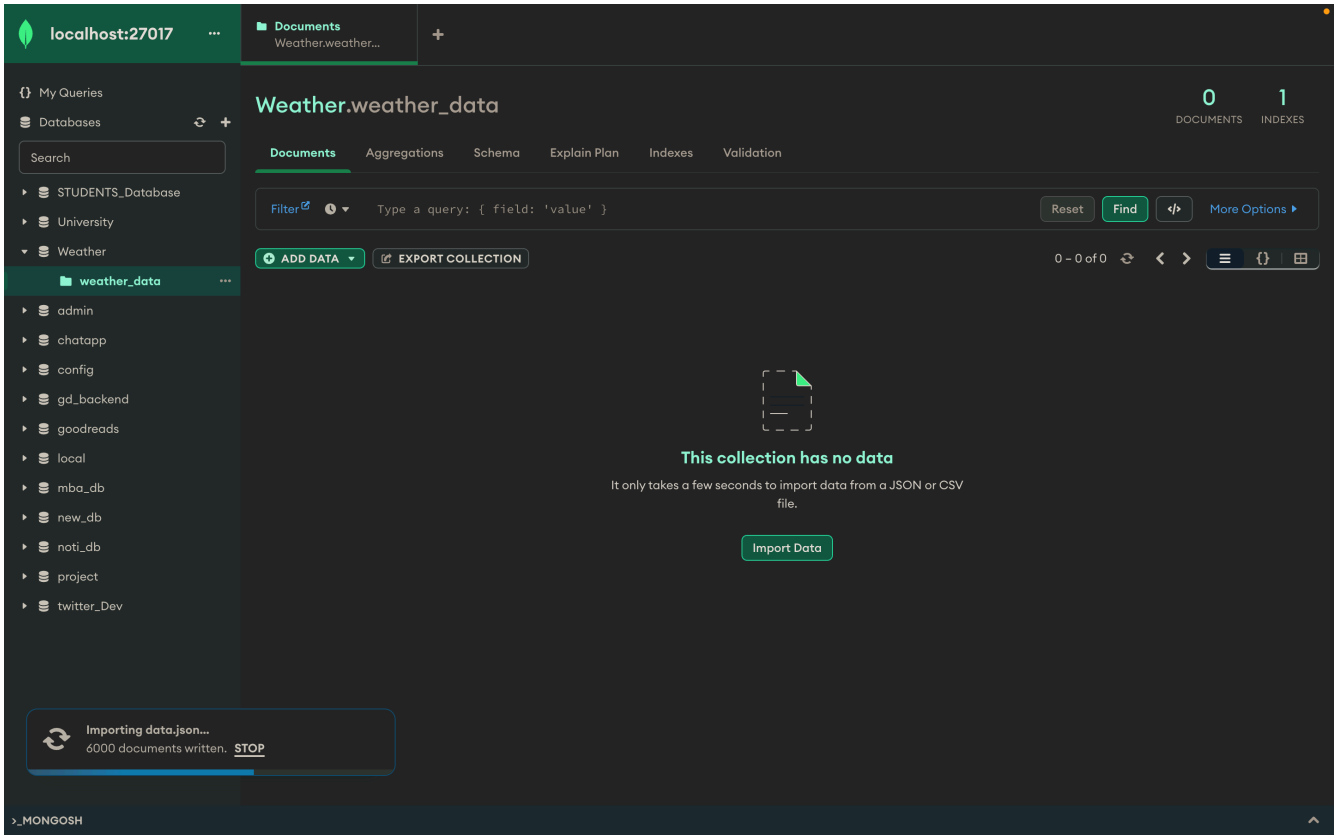
- Now click on Add Data, which will give you two options:
  - Either you can import from JSON or CSV
  - Or you can manually insert the document.
- We will go for Option 1.



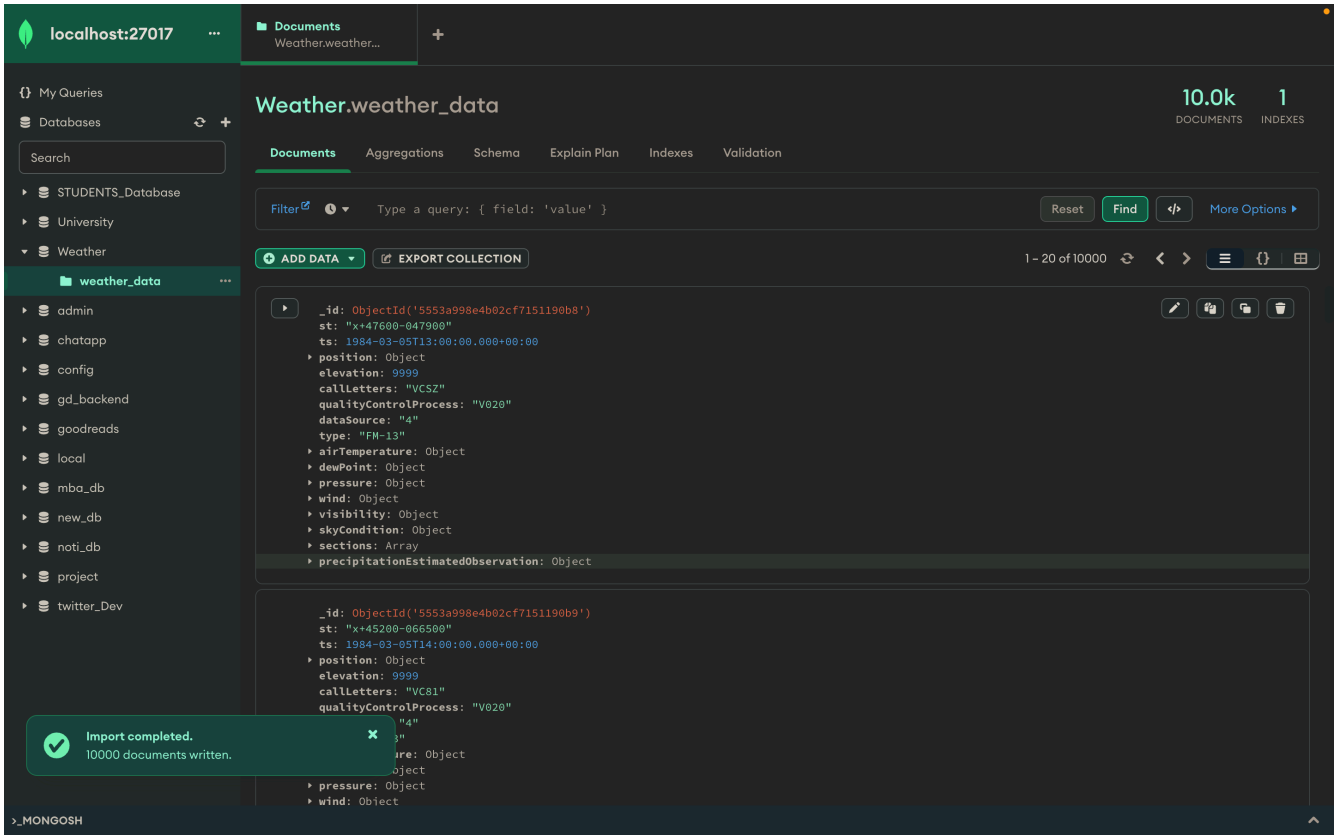
- It will open the file explorer where you can select the JSON we just downloaded.



- Click on import and it will automatically import the data from the JSON

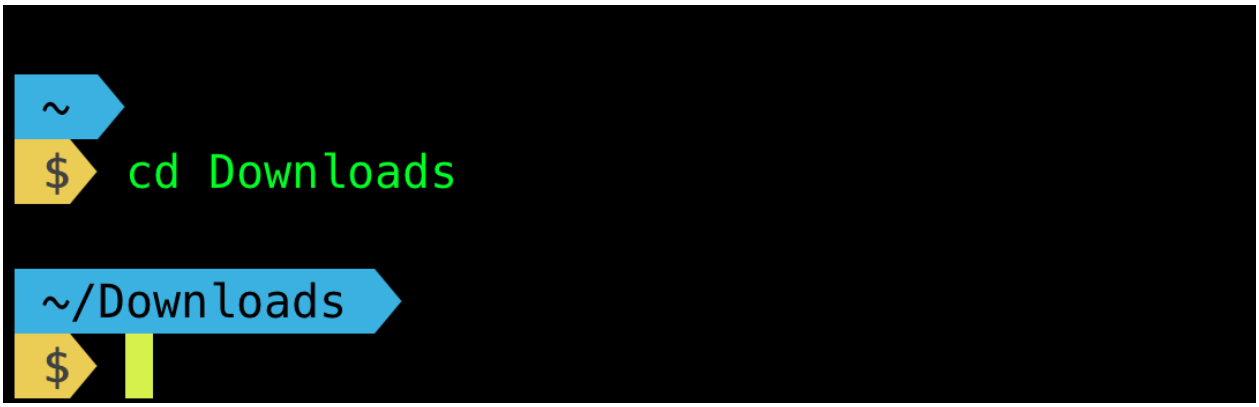


- After it finishes importing, it should look like this:



How to import without MongoDB Compass?

- We can use a terminal or CMD to import data without a compass.
- Open your terminal or CMD.
- Change the directory to the one where you have downloaded the JSON. If you are using Windows refer to this link to understand how to change the directory from cmd.



- Now we can use a MongoDB tool called as `mongoimport` (generally by default installed with MongoDB). We can use the following command:

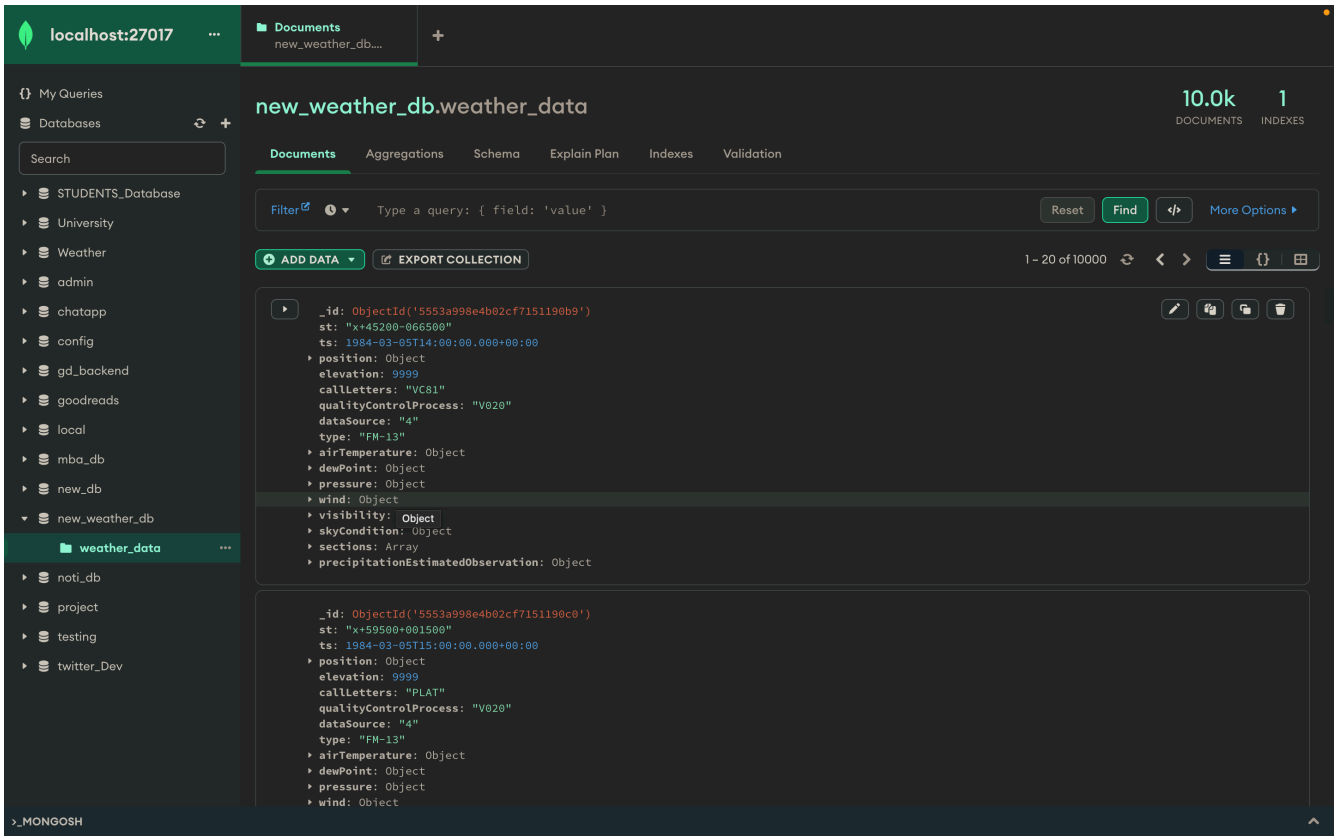
JavaScript

```
mongoimport --db db_name --collection collection_name --
file data.json;
```

```
~/Downloads                                                    ✔ 2850  18:12:14
$ mongoimport --db new_weather_db --collection weather_data --file data.json;
2023-05-27T18:15:43.724+0530    connected to: mongodb://localhost/
2023-05-27T18:15:44.740+0530    10000 document(s) imported successfully. 0 document(s) failed to import.

~/Downloads                                                    ✔ 2851  18:15:44
$
```

- You can check the import directly from `mongosh` or compass



```
   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

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

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
  You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test> show dbs;
STUDENTS_Database   88.00 KiB
University          72.00 KiB
Weather              2.49 MiB
admin               40.00 KiB
chatapp             72.00 KiB
config             108.00 KiB
gd_backend         416.00 KiB
goodreads          416.00 KiB
local              104.00 KiB
mba_db             504.00 KiB
new_db              85.89 MiB
new_weather_db       2.57 MiB
noti_db             72.00 KiB
project             76.00 KiB
testing              2.55 MiB
twitter_Dev        360.00 KiB
test> use new_weather_db
switched to db new_weather_db
new_weather_db> show collections
weather_data
new_weather_db>
```

Let's explore the DB we imported

How to see the count of documents in a collection?

There is a `count` function that we can use

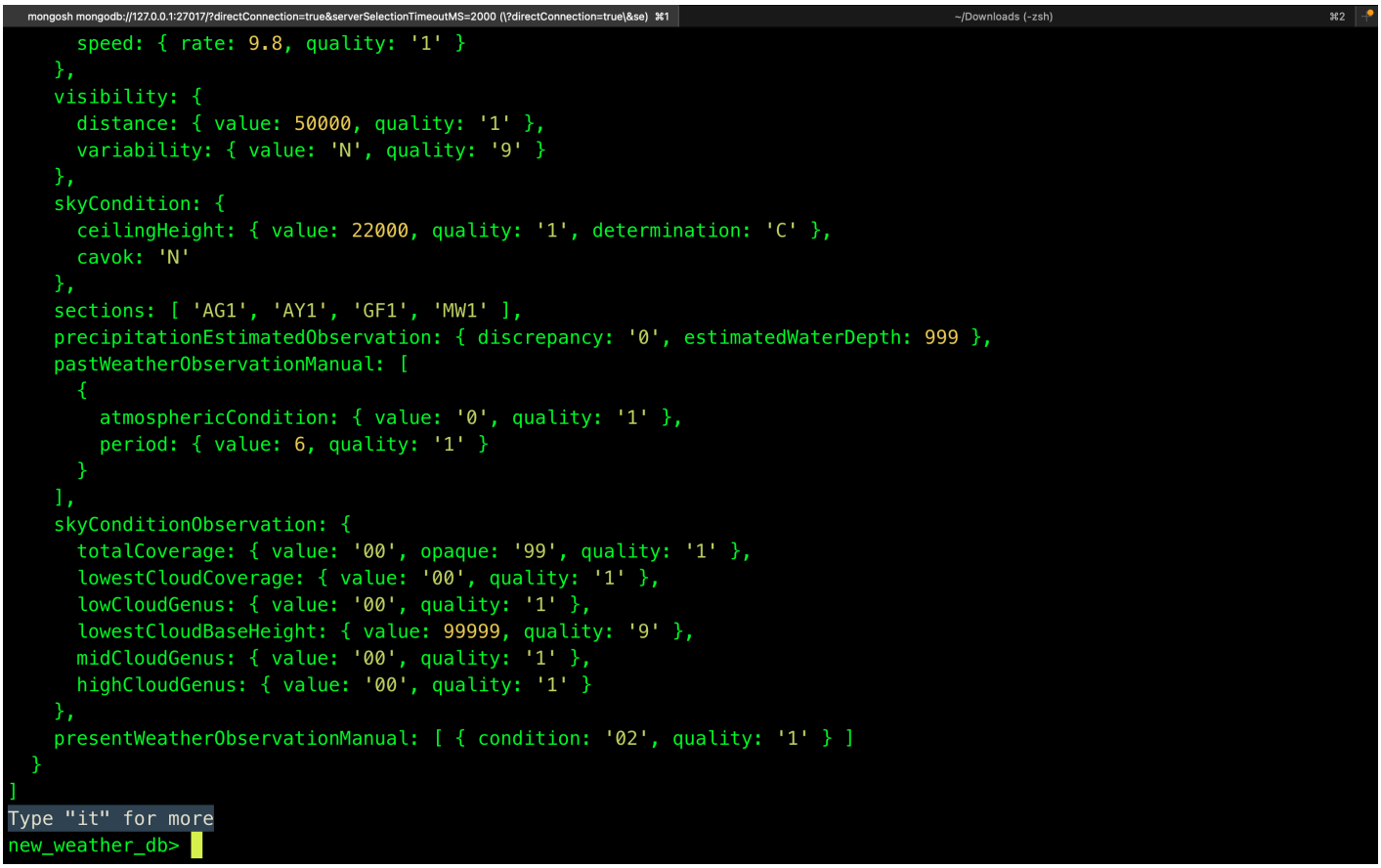
JavaScript

```
db.collectionName.find().count();
```

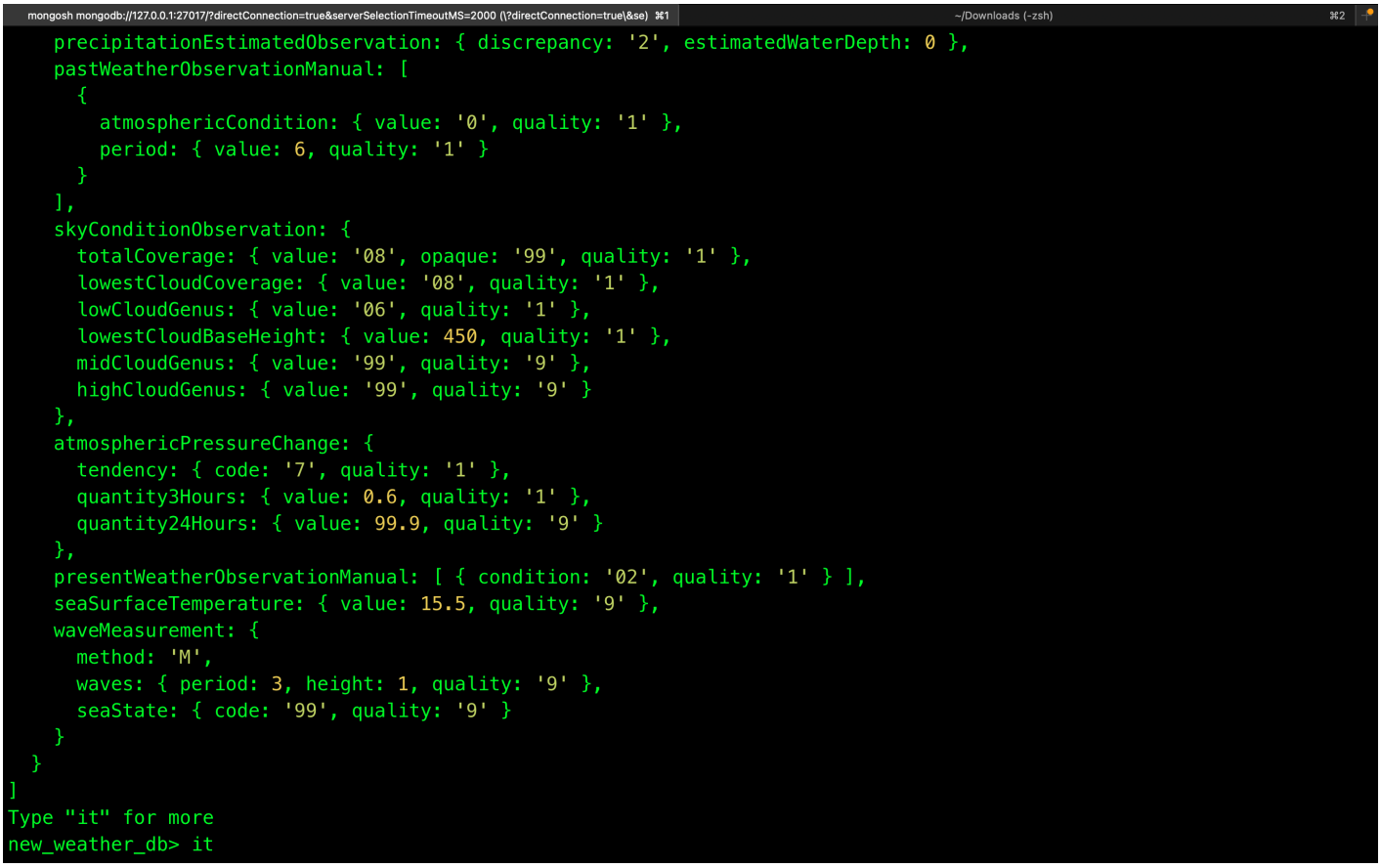
```
new_weather_db> db.weather_data.find().count()
(node:88641) [MONGODB DRIVER] Warning: cursor.count is deprecated and will be removed in the next major version, please
 use `collection.estimatedDocumentCount` or `collection.countDocuments` instead
(Use `node --trace-warnings ...` to show where the warning was created)
10000
new_weather_db>
```

How to handle the printing of huge amounts of data?

If you have a huge amount of data, just like we have here approx 10,000 documents, if you try to execute `db.collectionName.find` in your `mongosh` then will not print all the 10,000 records, becuase it cannot handle that in the shell. If you try it then in the console, it will print some data and then gives you a prompt of `Try "it" for more`



If you write `it` and press `enter` then you will get next group of data.

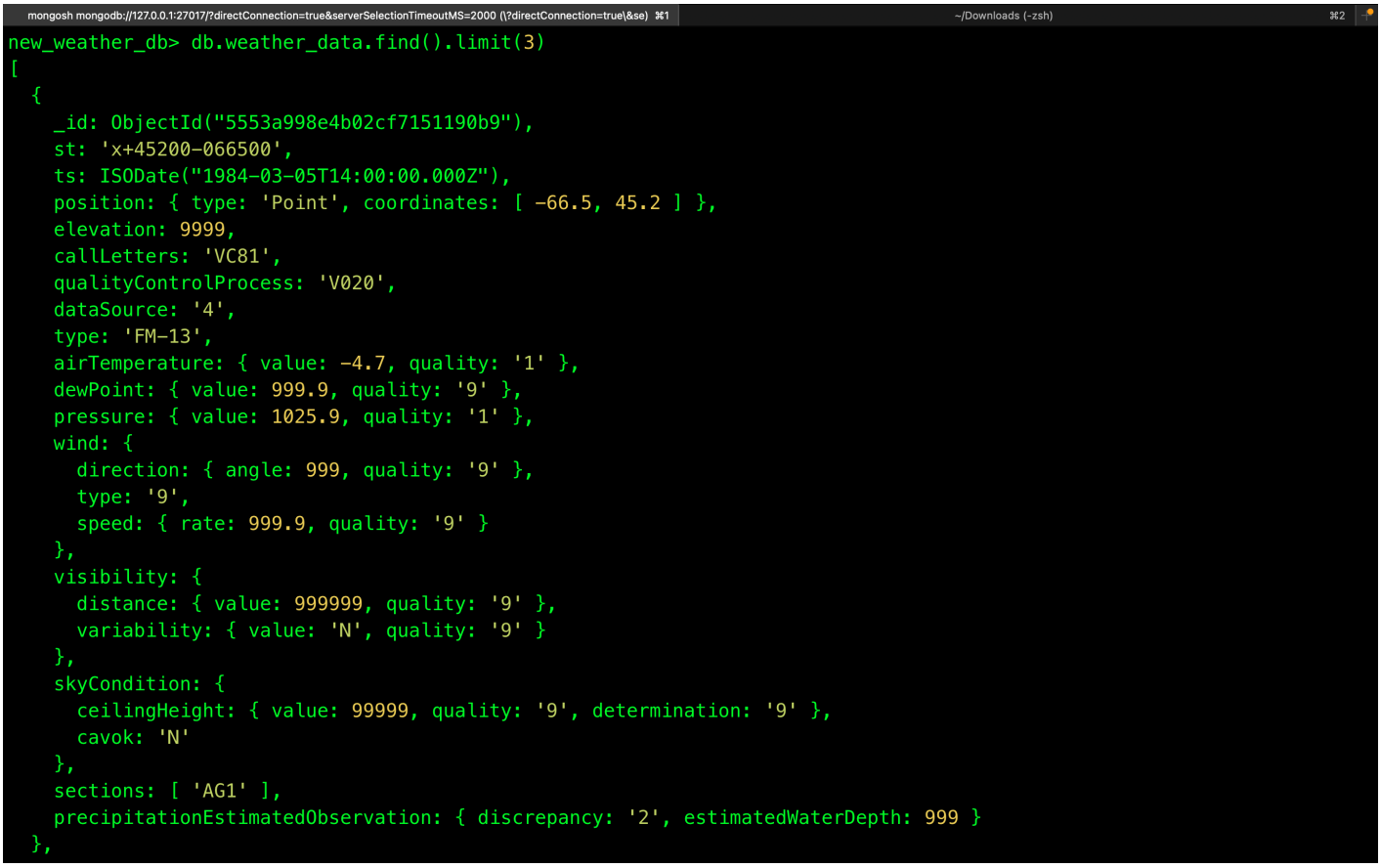


How to get a certain number of documents?

We can use a function called as `limit`

JavaScript

```
db.collectionName.find().limit(no_of_records_to_fetch);
```



How to set an offset while querying data?

We can set an offset using the `skip` function. Using the skip function we can skip some records and then start fetching records post the skip.

JavaScript

```
db.collectionName.find().skip(5).limit(3)
```

```
new_weather_db> db.weather_data.find().skip(10).limit(2);
[
  {
    _id: ObjectId("5553a998e4b02cf7151190b8"),
    st: 'x+47600-047900',
    ts: ISODate("1984-03-05T13:00:00.000Z"),
    position: { type: 'Point', coordinates: [ -47.9, 47.6 ] },
    elevation: 9999,
    callLetters: 'VCSZ',
    qualityControlProcess: 'V020',
    dataSource: '4',
    type: 'FM-13',
    airTemperature: { value: -3.1, quality: '1' },
    dewPoint: { value: 999.9, quality: '9' },
    pressure: { value: 1015.3, quality: '1' },
    wind: {
      direction: { angle: 999, quality: '9' },
      type: '9',
      speed: { rate: 999.9, quality: '9' }
    },
    visibility: {
      distance: { value: 999999, quality: '9' },
      variability: { value: 'N', quality: '9' }
    },
    skyCondition: {
      ceilingHeight: { value: 99999, quality: '9', determination: '9' },
      cavok: 'N'
    },
    sections: [ 'AG1' ],
    precipitationEstimatedObservation: { discrepancy: '2', estimatedWaterDepth: 999 }
  },
```

To filter the records, we can pass an object in the arguments of the `find` function, where we can add out conditions.

JavaScript

```
db.collectionName.find({key1: value1, key2: value2})
```

```
new_weather_db> db.weather_data.find({type: 'FM-13'}).count()
9994
new_weather_db>
```

Projections

If we want to not get all the properties of the JSON, and instead get some specific key-value pairs, this process is called Projection. In the world of SQL, if you do `SELECT * FROM TABLE;` then you get all the columns but if you do `SELECT name, address FROM TABLE;` you only get name and address. This same thing is achieved in Projections.

How to do projections?

So the first argument of the `find` function is an object which takes filtration criteria. It can accept another argument as an object, where we can write whatever properties we have to include and assign them a value true.

JavaScript

```
db.collectionName.find({filter1: value1...}, {property1: true, property2: true....});
```

```
new_weather_db> db.weather_data.find({type: 'FM-13'}, {position: true, visibility: true});
```

You can also pass the first argument as an empty JSON object.

```
new_weather_db> db.weather_data.find({}, {position: true, visibility: true});
```

If we want to manually exclude specific properties, you can write their names with false value allocated:

```
new_weather_db> db.weather_data.find({}, {skyConditionObservation: false, pastWeatherObservationManual: false});
```

Now this will bring everything apart from `pastWeatherObservationManual` and `skyConditionObservation`

How to delete a document?

If we want to delete some documents we can use functions like `deleteOne`, `deleteMany` and `findByIdAndDelete`.

JavaScript

```
db.collectionName.deleteOne({filter1: value1, filter2: value2..});
```

```
new_weather_db> db.weather_data.deleteOne({st: 'x-19300+060300'})
{ acknowledged: true, deletedCount: 1 }
new_weather_db> db.weather_data.find().count()
9999
new_weather_db>
```

JavaScript

```
db.collectionName.deleteMany({filter1: value1, filter2: value2..});
```

```
new_weather_db> db.weather_data.find({callLetters: 'FNPG'}).count()
7
new_weather_db> db.weather_data.deleteMany({callLetters: 'FNPG'})
{ acknowledged: true, deletedCount: 7 }
new_weather_db> db.weather_data.find().count()
9992
new_weather_db>
```

```
new_weather_db> db.weather_data.deleteOne({_id: ObjectId("5553a998e4b02cf7151190b9")})
{ acknowledged: true, deletedCount: 1 }
new_weather_db> db.weather_data.find({_id: ObjectId("5553a998e4b02cf7151190b9")})

new_weather_db>
```

We can similarly use `findOneAndDelete` to filter the data and then delete one record it. Docs

How to update a record?

To Update records we can use `updateOne` , `updateMany` and a few similar functions. These functions take two arguments,

1. Filtration criteria viz. how to filter what data to update
2. With what value we should update

JavaScript

```
db.collectionName.updateOne({filter1: value1}, {$operator: {key1: value1, key2: value2...}})
```

Now MongoDB provides us some operators for these updates for example:

- $set -> This will allocate the value to the key directly passed in the object
- $inc -> This will increment the value in the key

```
new_weather_db> db.weather_data.updateOne({_id: ObjectId("5553a998e4b02cf7151190c0")}, {$set: {elevation: 9998}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
new_weather_db> db.weather_data.updateOne({_id: ObjectId("5553a998e4b02cf7151190c0")}, {$inc: {elevation: 1}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

To decrease we can do `$inc` with a negative value.

```
new_weather_db> db.weather_data.updateOne({_id: ObjectId("5553a998e4b02cf7151190c0")}, {$inc: {elevation: -1}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

If you will use `updateMany` then all the records which are complying to the filteration criteria will be update.