Add Cover Add Subtitle

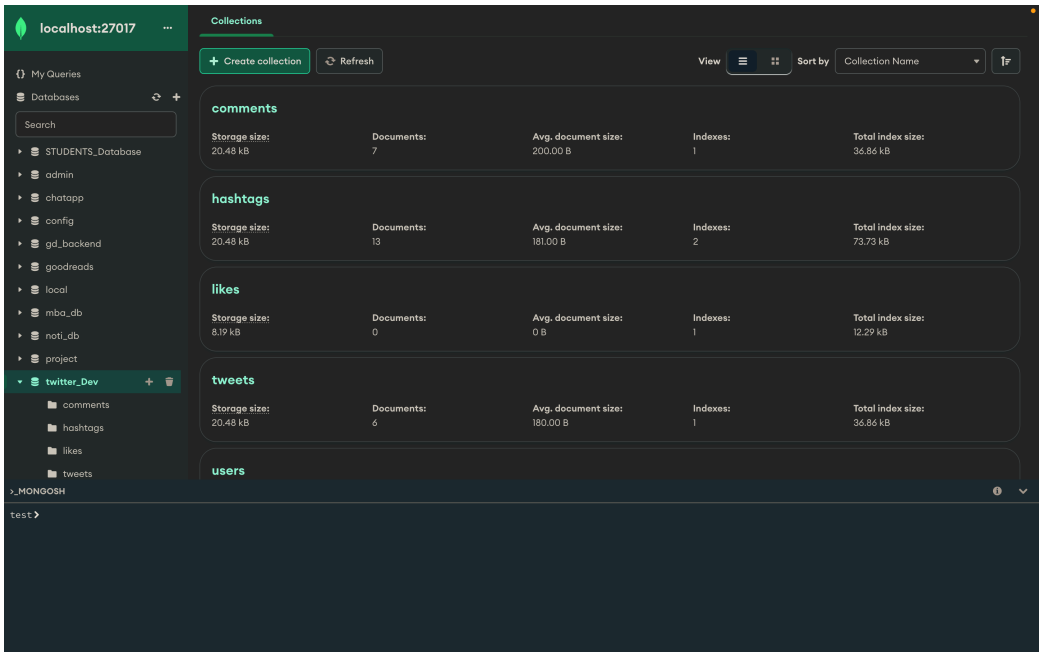
MongoDB

Using MongoDB

Open MongoDB in the terminal:

Write `mongosh` in your terminal, and it will open the MongoDB console.

You can also use MongoDB Compass software, to open the same terminal directly.



List all databases in mongodb

To list all the databases stored in your mongodb we can use the below commands:

JavaScript

```
show databases;
```

We can also use the below command:

JavaScript

```
show dbs;
```

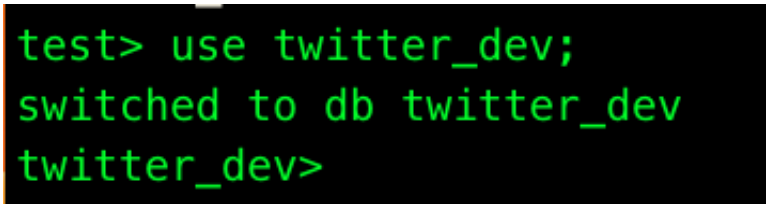
```
test> show databases;
STUDENTS_Database   88.00 KiB
admin               40.00 KiB
chatapp             72.00 KiB
config             108.00 KiB
gd_backend         416.00 KiB
goodreads          416.00 KiB
local              104.00 KiB
mba_db             504.00 KiB
noti_db             72.00 KiB
project             76.00 KiB
twitter_Dev        360.00 KiB
test> show dbs;
STUDENTS_Database   88.00 KiB
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
noti_db             72.00 KiB
project             76.00 KiB
twitter_Dev        360.00 KiB
test>
```

How to select a particular DB to work on?

To select a particular db and start querying on it we can use

JavaScript

```
use name_of_database;
```



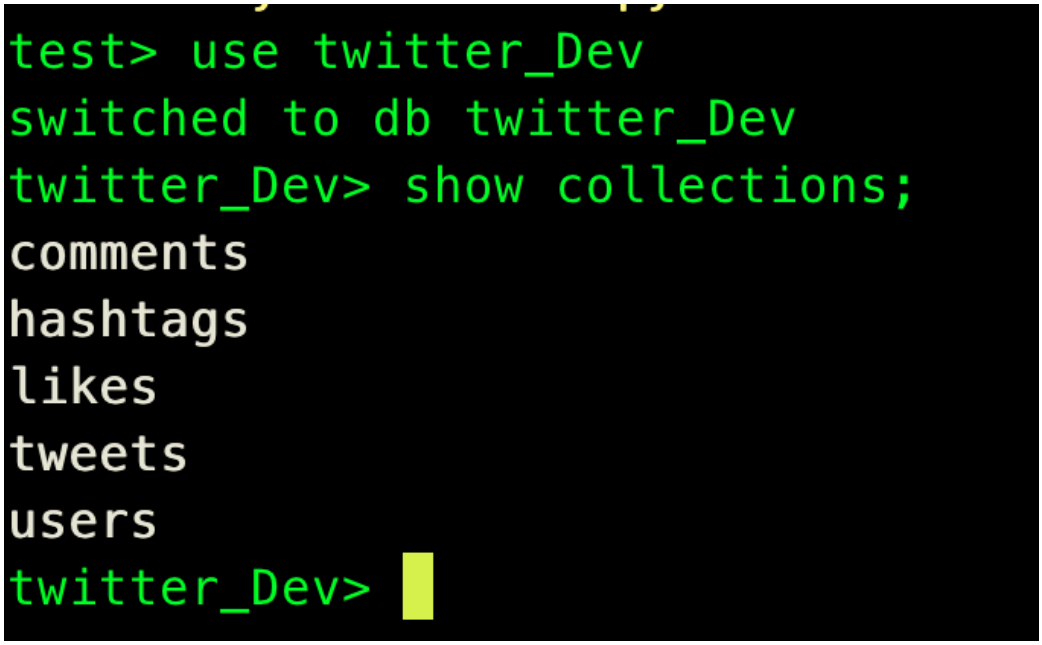
How to print all the collections stored in a database ?

To print all the collections we can use:

JavaScript

```
show collections;
```

Make sure, we run this command after executing `use some_db` otherwise we wont be having db selected.

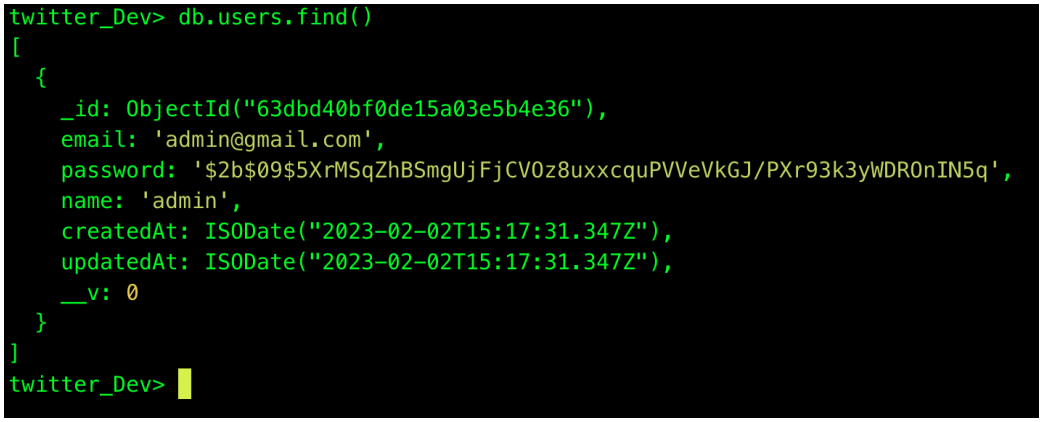


How to print all the documents of a collection ?

To print all the documents of a collection we can use:

JavaScript

```
db.collectionname.find();
```



How to create a new database ?

To create a new database we can do:

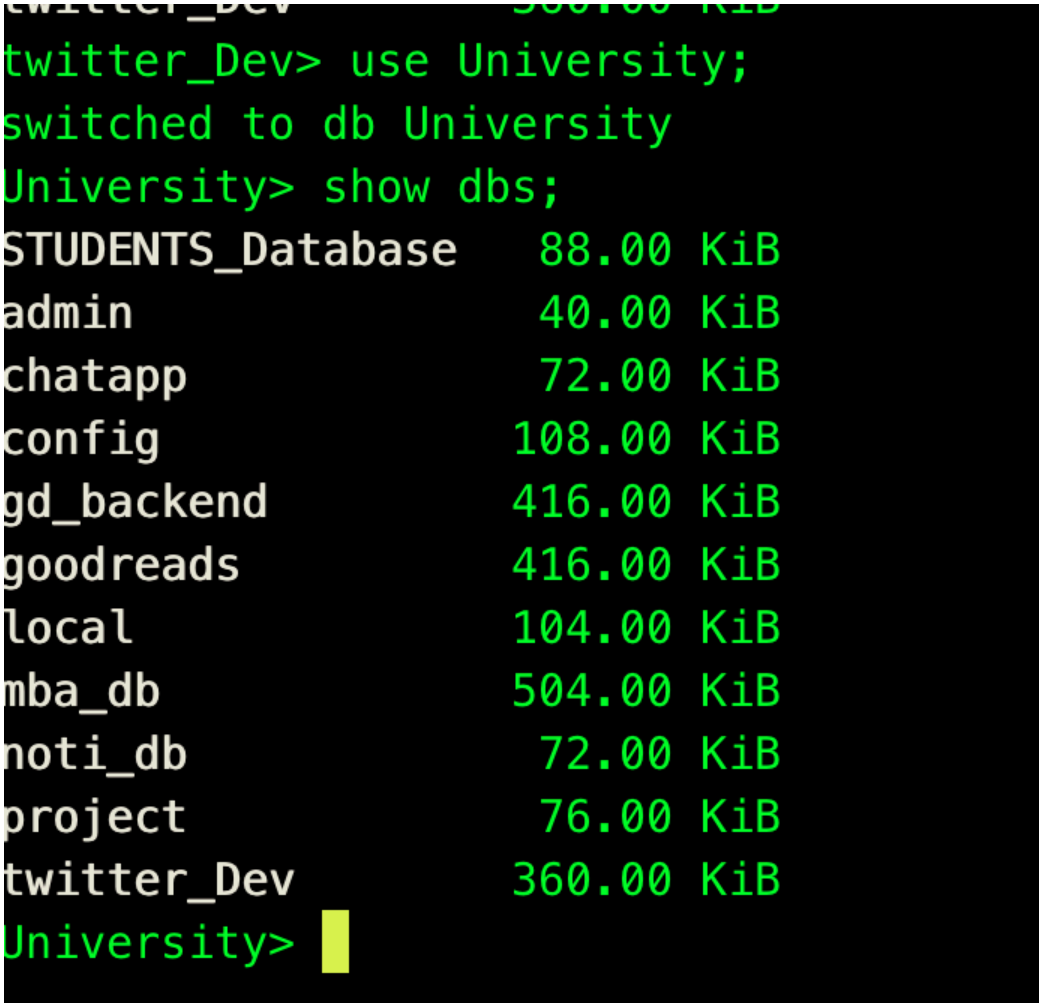
JavaScript

```
use new_database_name;
```

The `use` command creates a new database if there is no already present db with the same name, otherwise if there is a db with the same name, it just selects it.

```
twitter_Dev> show dbs;
STUDENTS_Database   88.00 KiB
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
noti_db             72.00 KiB
project             76.00 KiB
twitter_Dev        360.00 KiB
twitter_Dev> use University;
switched to db University
University>
```

Now what will happen is, after creating a DB, if we try to do `show dbs;` then it will not list our newly created database. Because, if mondodb sees that there is no valid collection added in the database, and the db is empty, it doesn't list it.



So how to add a new collection?

How to add new collections?

To create a new collection we can do:

JavaScript

```
db.createCollection("name_of_the_collection")
```

Make sure we execute this command after `use some_db_name`

```
University> db.createCollection("students")
{ ok: 1 }
University> show dbs;
STUDENTS_Database   88.00 KiB
University           8.00 KiB
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
noti_db             72.00 KiB
project             76.00 KiB
twitter_Dev        360.00 KiB
University>
```

In normal RDBMS, while creating a table, we have to define what will be the column of the table. Why we are not defining the properties of a collection on Mongodb?

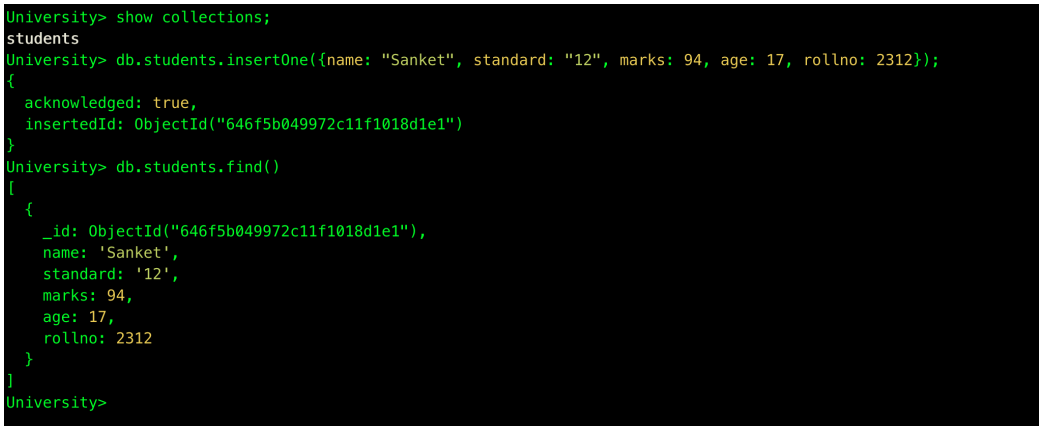
This is because, MongoDB doesn't restrict us by any means for defining documents of a collection. Two documents of the same collection can possess different types of properties.

How to add a new record to a collection?

To add a new record we can do:

JavaScript

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



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?

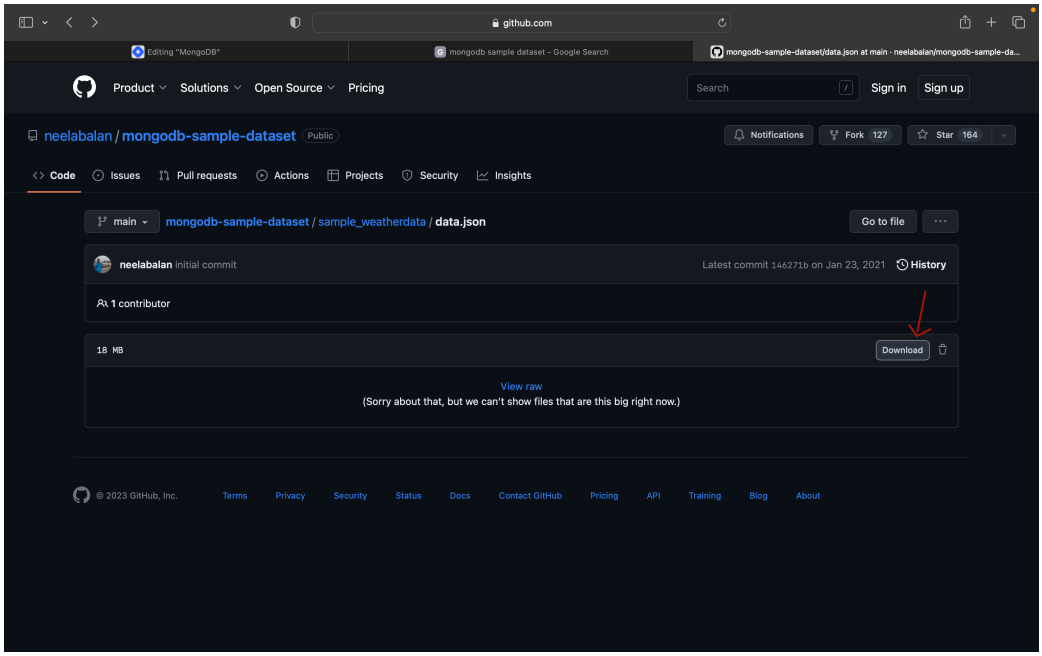
We can simply use `[]` to add an array.

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

How to import sample DB in MongoDB?

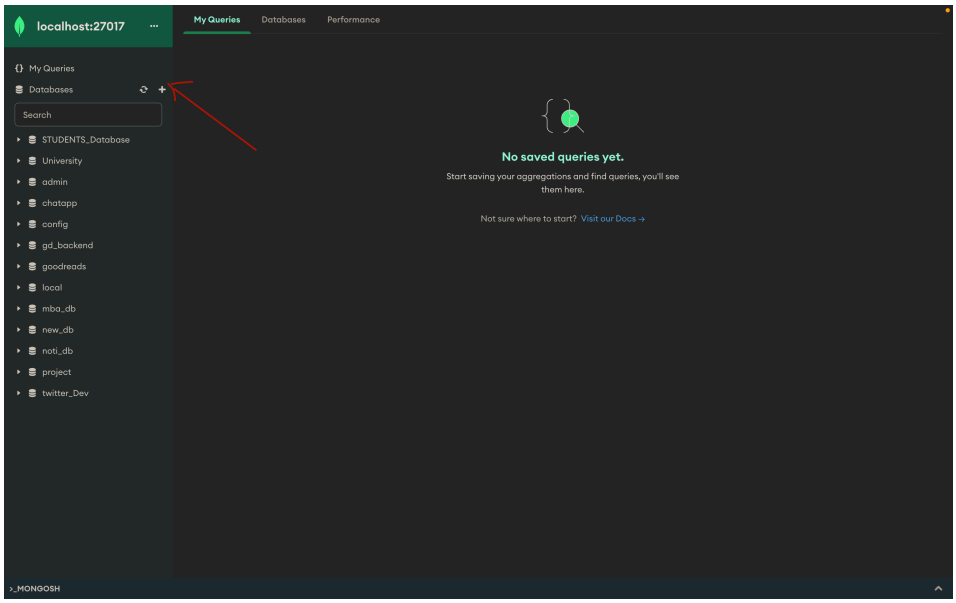
To get some free sample datasets you can check out this Github link: https://github.com/neelabalan/mongodb-sample-dataset.

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



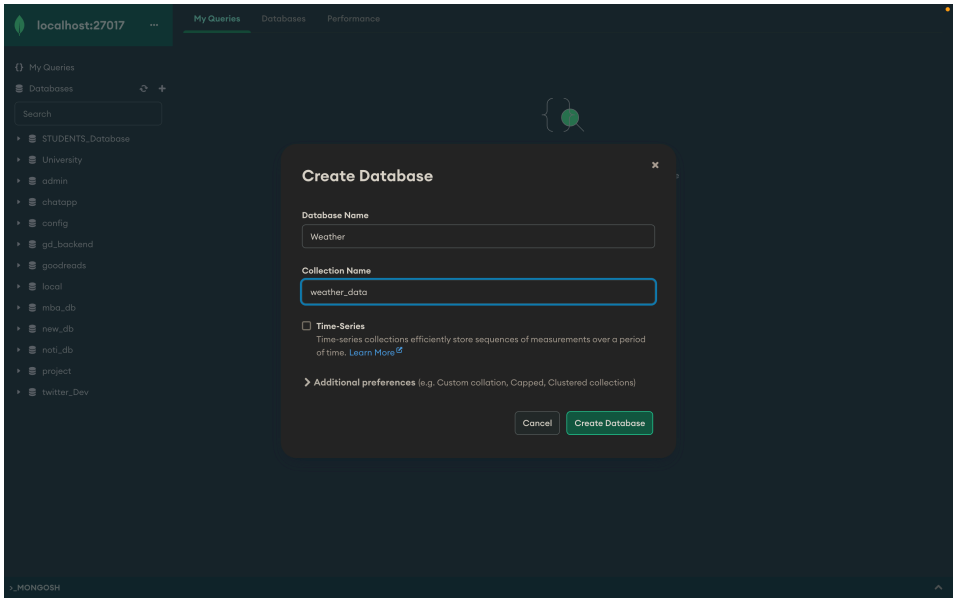
After the file is downloaded you can import this data using MongoDB Compass.

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

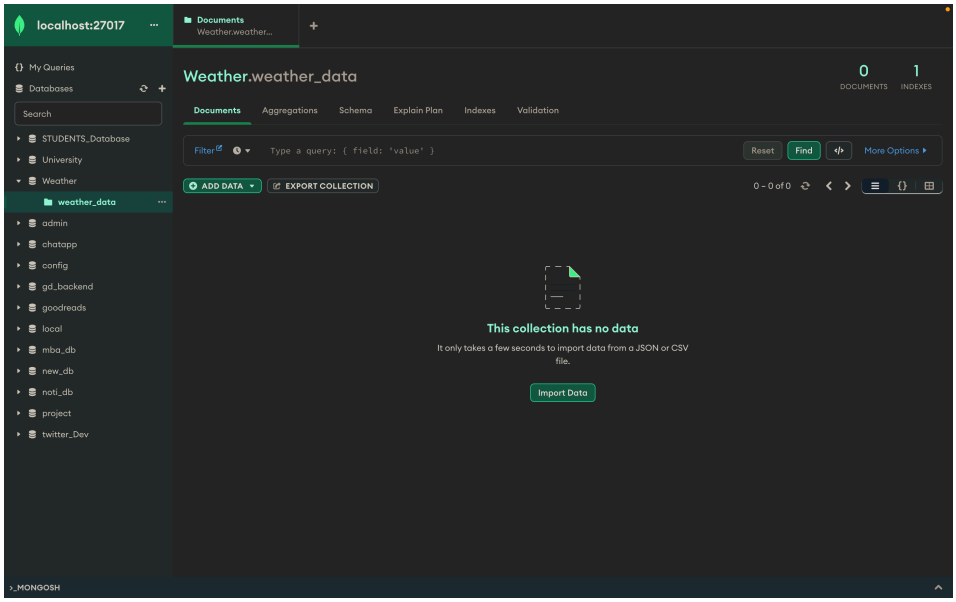


- We can give a name to the database and give

the first collection name also directly using MongoDB Compass.



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

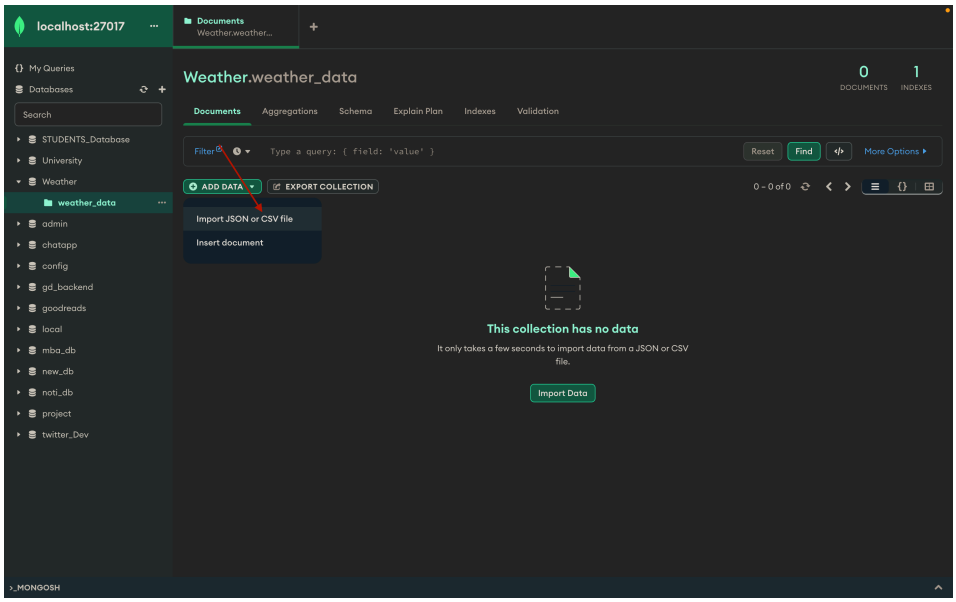


- Now click on Add Data, which will give you

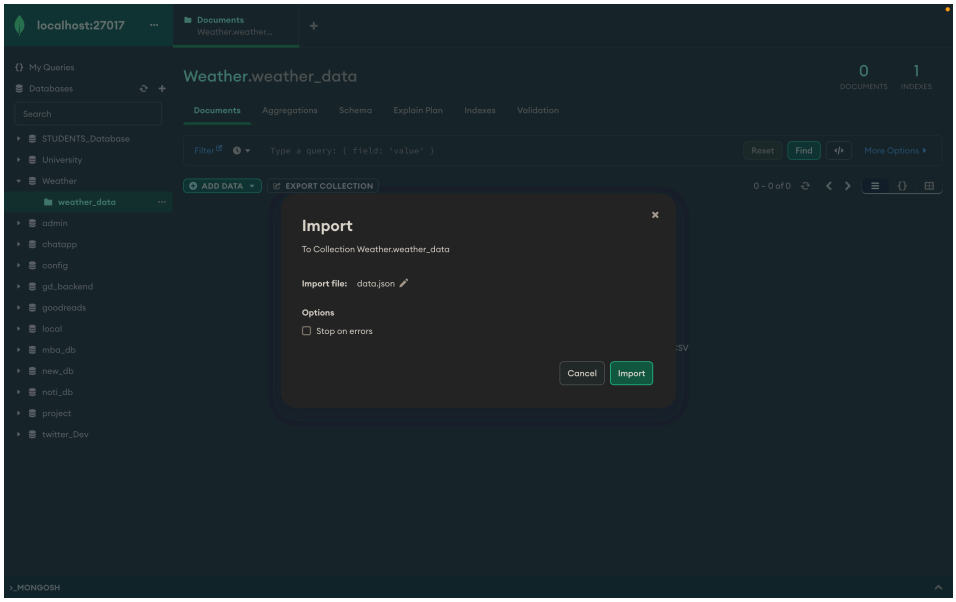
two options:

- Either you can import from JSON or CSV
- Or you can manually insert the document.

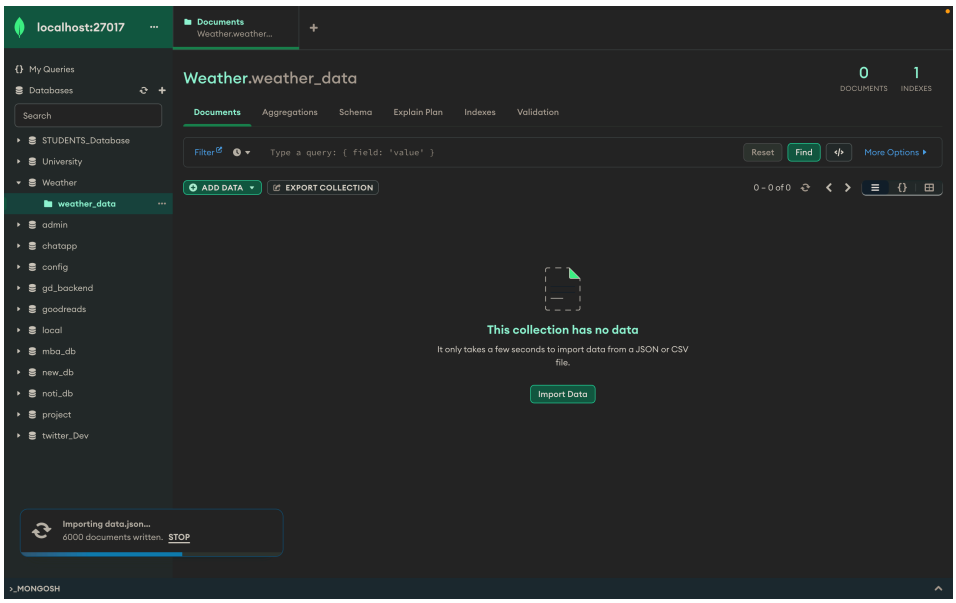
- We will go for Option 1.



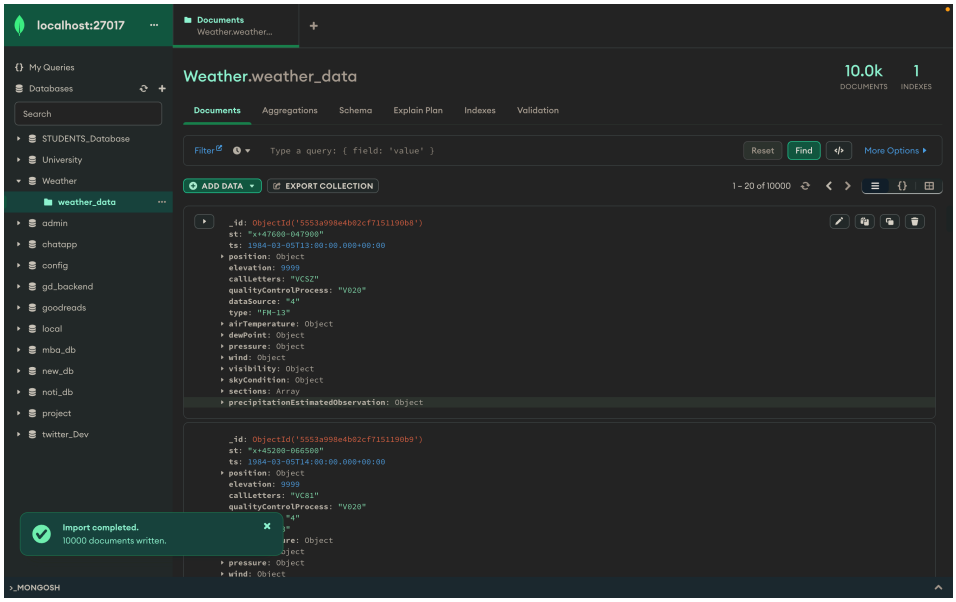
- It will open the file explorer where you can select the JSON we just downloaded.



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

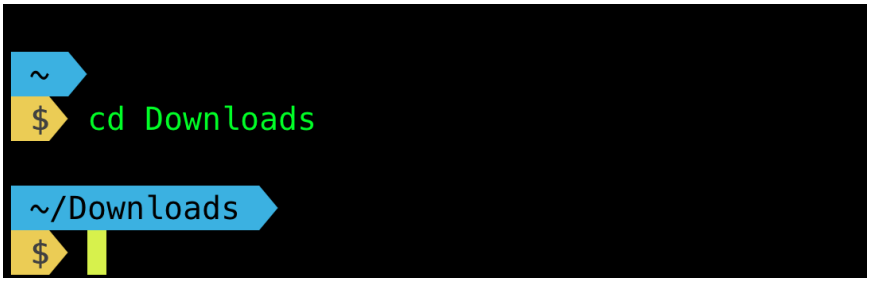


- After it finishes importing, it should look like this:



How to import without MongoDB Compass?

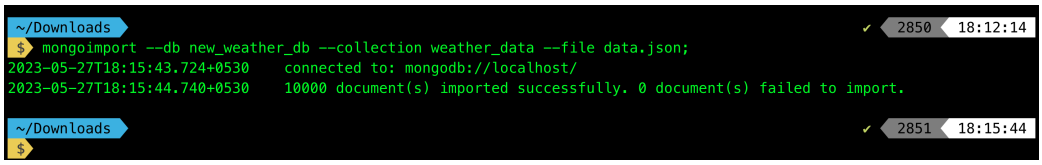
- We can use a terminal or CMD to import data without a compass.
- Open your terminal or CMD.
- Change the directory to the one where you have downloaded the JSON. If you are using Windows refer to this link to understand how to change the directory from cmd.



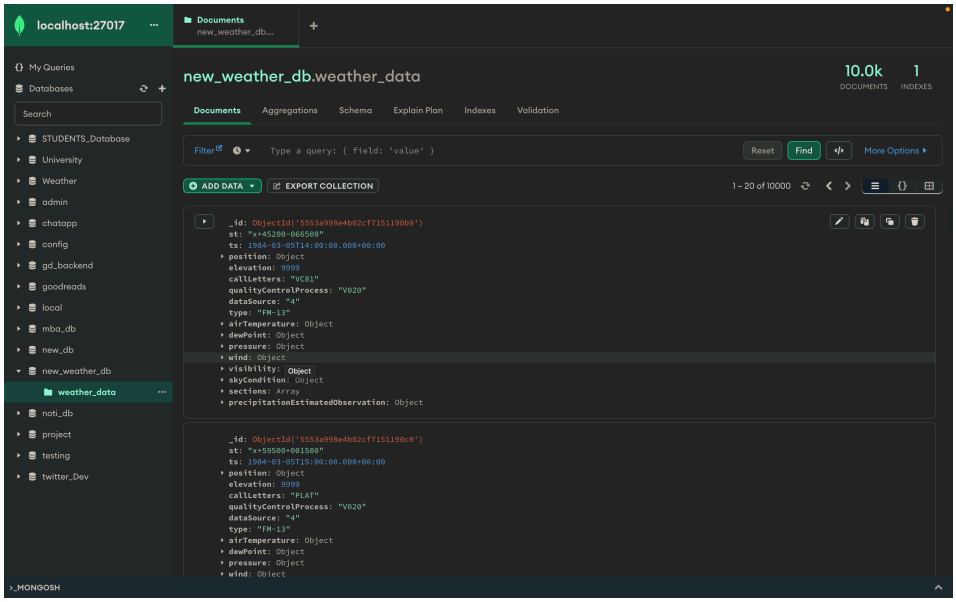
- Now we can use a MongoDB tool called as `mongoimport` (generally by default installed with MongoDB). We can use the following command:

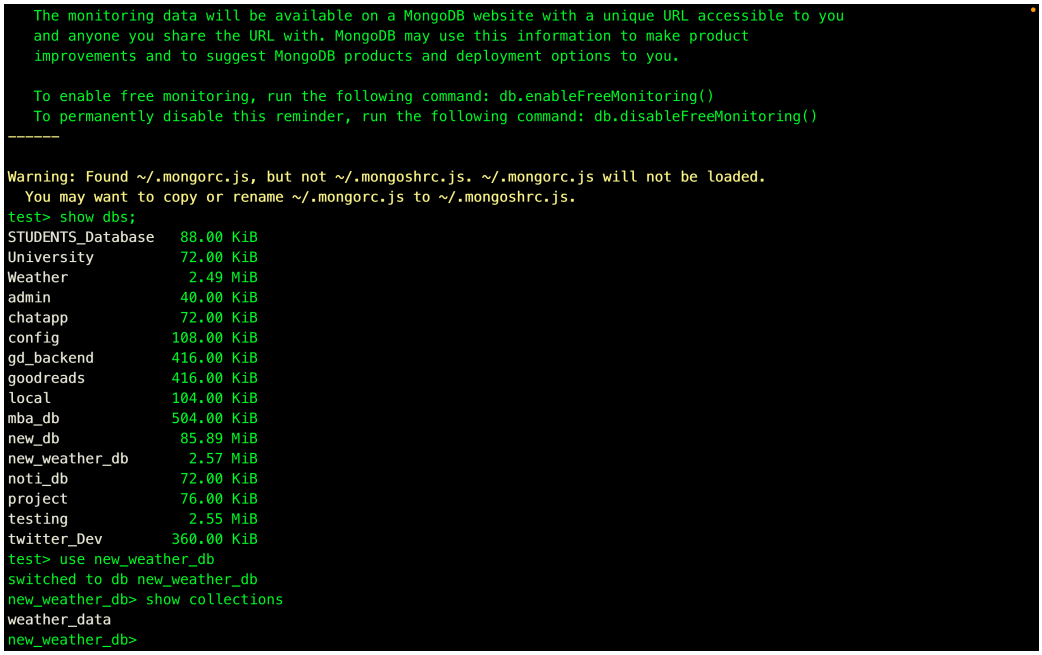
JavaScript

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



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





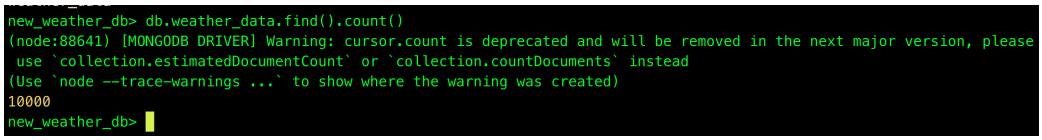
Let's explore the DB we imported

How to see the count of documents in a collection?

There is a `count` function that we can use

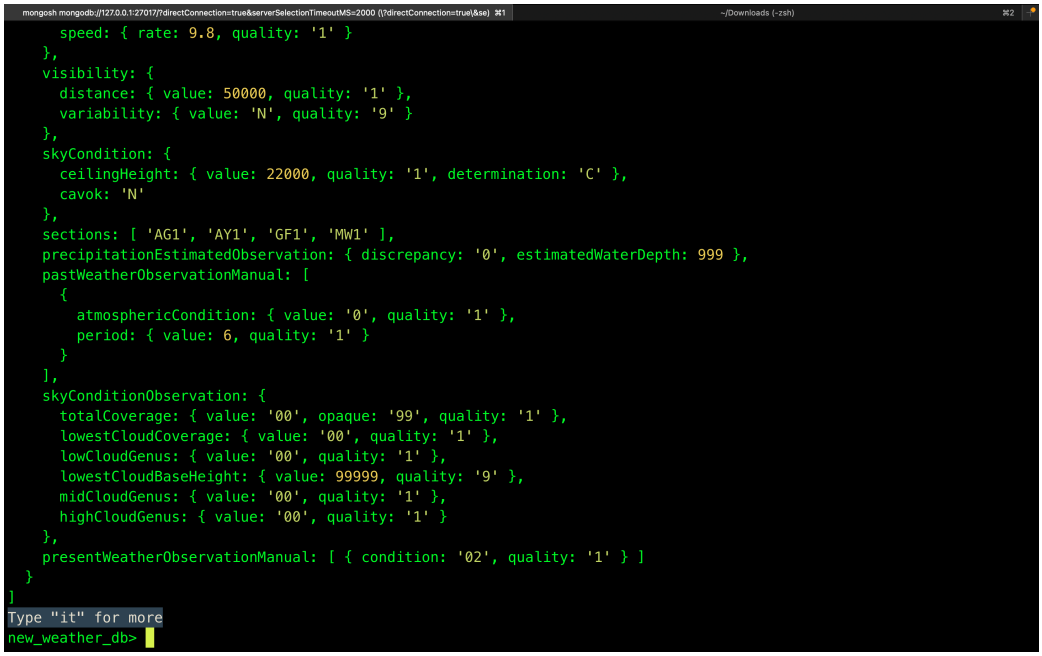
JavaScript

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

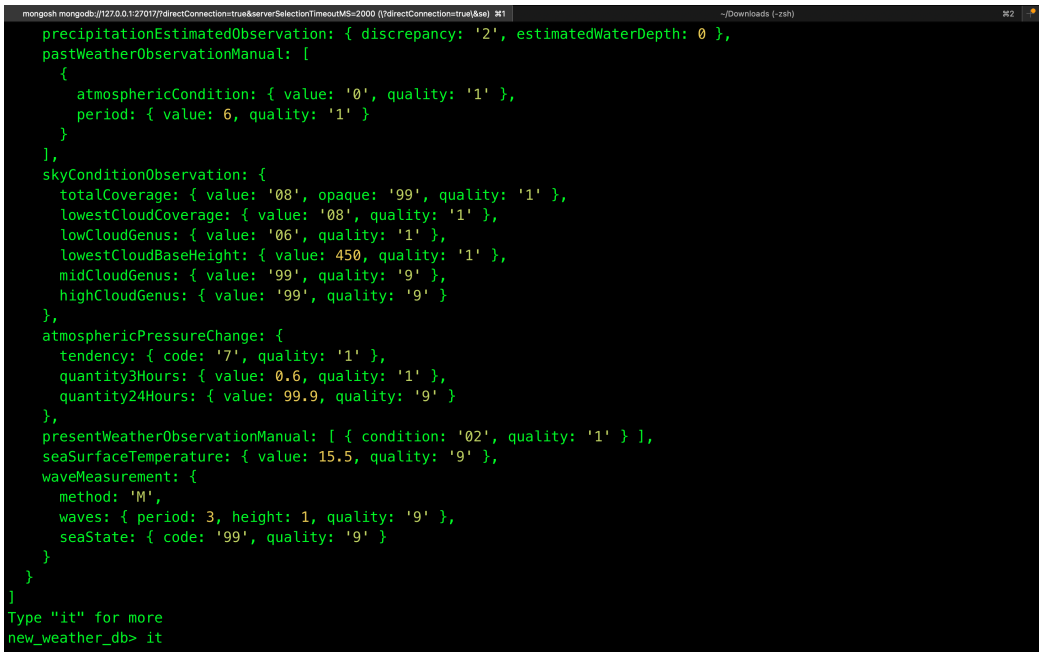


How to handle the printing of huge amounts of data?

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, because 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.

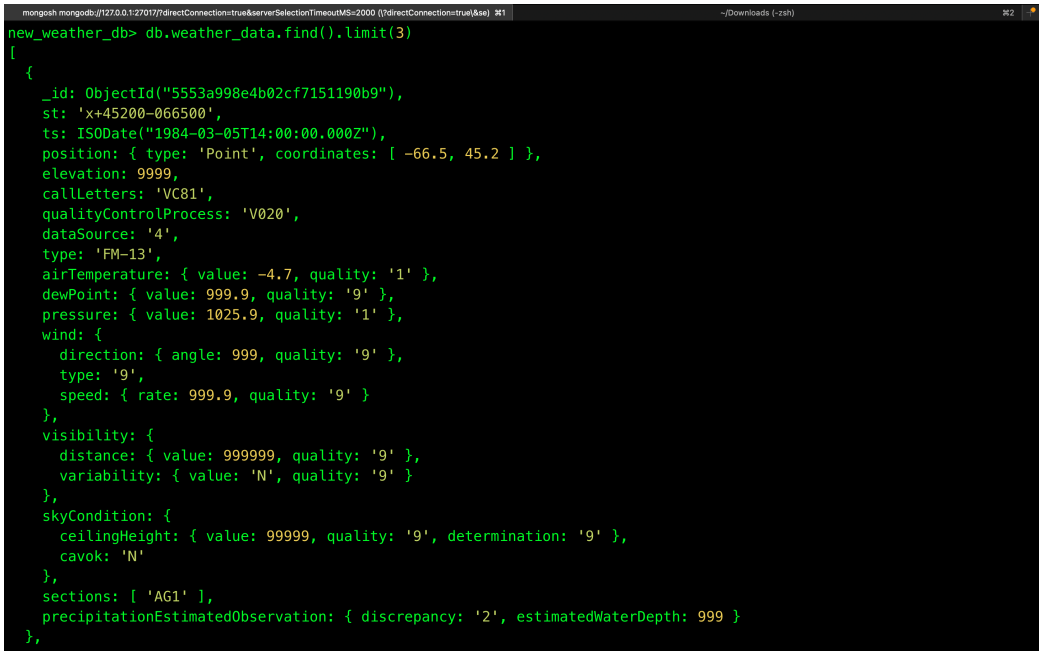


How to get a certain number of documents?

We can use a function called as `limit`

JavaScript

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

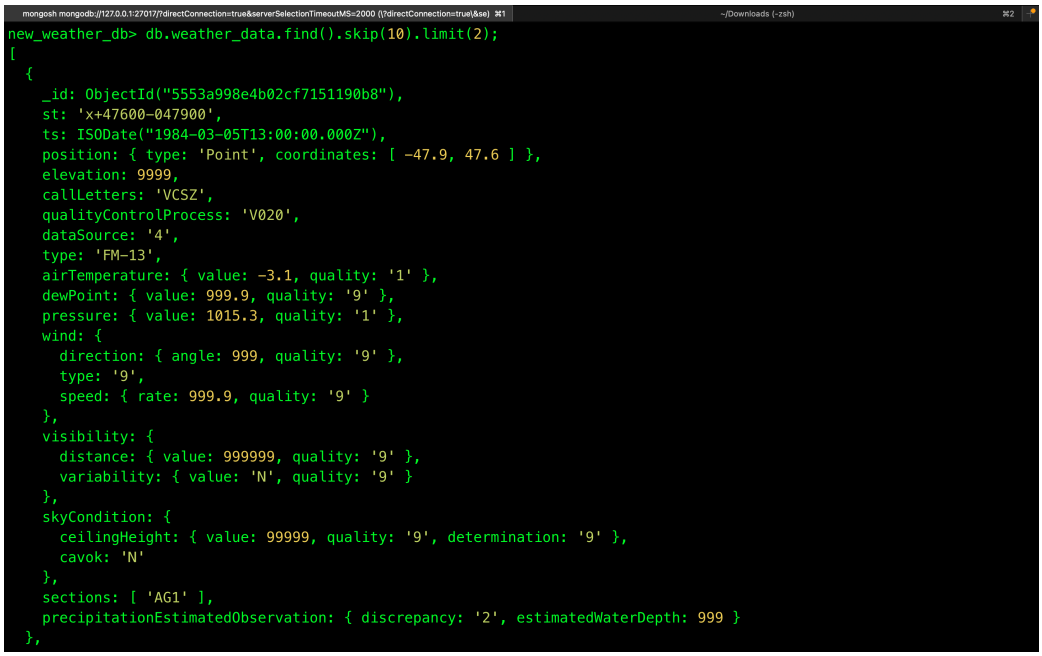


How to set an offset while querying data?

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)
```



How do filter records based on a condition?

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 with 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")}, {$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 with the filtration criteria will be updated.

How to get distinct values of a particular key?

We can use the `distinct` function in order to get all distinct values of a particular key.

JavaScript

```
db.collectionName.distinct("key");
```

```
Weather> db.weather_data.distinct("type")
[ 'FM-13', 'SAO' ]
Weather> db.weather_data.distinct("callLetters")
[
  '0BVZ', '0JSV', '2100', '3DRO', '3DRV', '3EBA', '3EDA',
  '3EEJ', '3EEK', '3EFI', '3EFQ', '3EGC', '3EGU', '3EHC',
  '3EHG', '3EHH', '3EIZ', '3EJF', '3EKJ', '3ELE', '3ENZ',
  '3EOS', '3EOU', '3EPA', '3EQN', '3ERJ', '3ERZ', '3ETZ',
  '3EWQ', '3EXD', '3EYN', '3EZL', '3FAR', '3FBD', '3FDL',
  '3FEV', '3FGE', '3FJR', '3FJV', '3FKA', '3FMV', '3FQU',
  '3FTJ', '3FXH', '3FXI', '3FXS', '3FYO', '3HZL', '4602',
  '4PKY', '4XIC', '4XII', '4XIL', '4XLU', '4XLZ', '4XMX',
  '5LFX', '5LJT', '5LLP', '5LOT', '5LOZ', '5LTW', '5LUB',
  '5LWE', '5LWG', '5LWQ', '5MCB', '5MCN', '5MGC', '5MQI',
  '5MTS', '5MZH', '5PVN', '6ZAM', '6ZAS', '6ZBM', '6ZCM',
  '6ZDN', '6ZFM', '6ZHX', '6ZJP', '6ZLH', '6ZRT', '6ZUU',
  '7FFC', '7JBJ', '7JFY', '7JKS', '7JKY', '7JOB', '7JPI',
  '7JQI', '7JQX', '7JRZ', '7JSO', '7JSP', '7JTJ', '7JTN',
  '7JUI', '7JVM',
  ... 2549 more items
]
```

Operators in MongoDB

MongoDB provides different comparison, logical, bitwise etc operators that we can use very easily.

To use these operators for comparison and logical operations, instead of directly assigning a key-value pair for your filters etc, put your key and assign it an object, inside the assigned object, put the key as your operator and value as the value to put.

JavaScript

```
db.collectionName.find({property:
{$operator: value}})
```

```
Weather> db.weather_data.find({type: {$ne: 'FM-13'}}).count()
6
Weather>
```

Commonly used operators:

- $ne -> not equals
- $eq -> equals
- $lt -> less than
- $lte -> less than or equal to
- $gt -> greater than
- $gte -> greater than or equal to
- $and -> logical and
- $or -> logical or
- $not -> logical not
- $nor -> logical nor
- $in -> for checking in the array

- $nin -> not in the array

```
Weather> db.weather_data.find({"dewPoint.value": {$gt: 999}}).count()
5116
Weather> db.weather_data.find({"dewPoint.value": {$lt: 999}}).count()
4884
Weather>
```

```
Weather> db.weather_data.find({$and: [{elevation: {$lt: 8000}}, {elevation: {$gt: 1000}}]}).count()
6
Weather>
```

If you want to find all the records based on a property and compare that with a bunch of values in an array such that the property should be equal to any one of them then we can use `$in`

```
Weather> db.weather_data.find({callLetters: {$in: ['3EHH', '3EIZ', '3EJF']}}).count()
17
Weather>
```

Analysing queries

We can analyse our queries using the `explain` function. Inside this function, we can pass an argument `executionStats` and it will provide us with the details

Bash

```
db.weather_data.find({type:
'SAO'}).explain("executionStats")
```

JSON

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'Weather.weather_data',
    indexFilterSet: false,
    parsedQuery: { type: { '$eq': 'SAO'
} },
    maxIndexedOrSolutionsReached:
false,
    maxIndexedAndSolutionsReached:
false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { type: { '$eq': 'SAO' }
},
      direction: 'forward'
    },
    rejectedPlans: []
  },
```

```
    executionStats: {
      executionSuccess: true,
      nReturned: 6,
      executionTimeMillis: 31,
      totalKeysExamined: 0,
      totalDocsExamined: 10000,
      executionStages: {
        stage: 'COLLSCAN',
        filter: { type: { '$eq': 'SAO' }
},
        nReturned: 6,
        executionTimeMillisEstimate: 0,
        works: 10002,
        advanced: 6,
        needTime: 9995,
        needYield: 0,
        saveState: 10,
        restoreState: 10,
        isEOF: 1,
        direction: 'forward',
        docsExamined: 10000
      }
    },
    command: { find: 'weather_data',
filter: { type: 'SAO' }, '$db':
'Weather' },
    serverInfo: {
      host: 'Sankets-MacBook-Pro.local',
```

```
    port: 27017,
    version: '5.0.7',
    gitVersion:
'b977129dc70eed766cbee7e412d901ee213acb
da'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes:
104857600,

internalQueryFacetMaxOutputDocSizeBytes
: 104857600,

internalLookupStageIntermediateDocument
MaxSizeBytes: 104857600,

internalDocumentSourceGroupMaxMemoryByt
es: 104857600,

internalQueryMaxBlockingSortMemoryUsage
Bytes: 104857600,

internalQueryProhibitBlockingMergeOnMon
goS: 0,
    internalQueryMaxAddToSetBytes:
104857600,

internalDocumentSourceSetWindowFieldsMa
```

```
    xMemoryBytes: 104857600
      },
      ok: 1
    }
```

Indexing

Indexing is a mechanism using which a database prepared more data structures to store data in a particular order based on particular keys, for faster search.

List all indexes present in the collection

```
Weather> db.weather_data.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { elevation: 1 }, name: 'elevation_1' },
  {
    v: 2,
    key: { elevation: 1, 'airTemperature.value': 1 },
    name: 'elevation_1_airTemperature.value_1'
  },
  { v: 2, key: { 'dewPoint.value': 1 }, name: 'dewPoint.value_1' }
]
```

To remove an index from the collection

```
Weather> db.weather_data.dropIndex('elevation_1')
{ nIndexesWas: 4, ok: 1 }
Weather> db.weather_data.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  {
    v: 2,
    key: { elevation: 1, 'airTemperature.value': 1 },
    name: 'elevation_1_airTemperature.value_1'
  },
  { v: 2, key: { 'dewPoint.value': 1 }, name: 'dewPoint.value_1' }
]
```

To create an index for a collection

```
new_weather_db> db.weather_data.createIndex({elevation: 1})
elevation_1
```

Analysing indexes

Before we created our index on the `elevation` property we were reading all the docs i.e. approx 9991 docs, to fetch all the docs with elevation less than 5000.

JSON

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace:
'new_weather_db.weather_data',
    indexFilterSet: false,
```

```
    parsedQuery: { elevation: { '$lt':
8000 } },
    maxIndexedOrSolutionsReached:
false,
    maxIndexedAndSolutionsReached:
false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { elevation: { '$lt':
8000 } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 6,
    executionTimeMillis: 27,
    totalKeysExamined: 0,
    totalDocsExamined: 9991,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { elevation: { '$lt':
8000 } },
      nReturned: 6,
      executionTimeMillisEstimate: 5,
      works: 9993,
```

```
        advanced: 6,
        needTime: 9986,
        needYield: 0,
        saveState: 9,
        restoreState: 9,
        isEOF: 1,
        direction: 'forward',
        docsExamined: 9991
      }
    },
    command: {
      find: 'weather_data',
      filter: { elevation: { '$lt': 8000
} },
      '$db': 'new_weather_db'
    },
    serverInfo: {
      host: 'Sankets-MacBook-Pro.local',
      port: 27017,
      version: '5.0.7',
      gitVersion:
'b977129dc70eed766cbee7e412d901ee213acb
da'
    },
    serverParameters: {
      internalQueryFacetBufferSizeBytes:
104857600,
```

```
    internalQueryFacetMaxOutputDocSizeBytes
    : 104857600,

    internalLookupStageIntermediateDocument
    MaxSizeBytes: 104857600,

    internalDocumentSourceGroupMaxMemoryByt
    es: 104857600,

    internalQueryMaxBlockingSortMemoryUsage
    Bytes: 104857600,

    internalQueryProhibitBlockingMergeOnMon
    goS: 0,
        internalQueryMaxAddToSetBytes:
    104857600,

    internalDocumentSourceSetWindowFieldsMa
    xMemoryBytes: 104857600
      },
      ok: 1
    }
```

But after creating an index this number will decrease drastically.

JSON

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace:
'new_weather_db.weather_data',
    indexFilterSet: false,
    parsedQuery: { elevation: { '$lt':
8000 } },
    maxIndexedOrSolutionsReached:
false,
    maxIndexedAndSolutionsReached:
false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { elevation: 1 },
        indexName: 'elevation_1',
        isMultiKey: false,
        multiKeyPaths: { elevation: []
},
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { elevation: [ '[-
```

```
inf.0, 8000)' ] }
        }
      },
      rejectedPlans: []
    },
    executionStats: {
      executionSuccess: true,
      nReturned: 6,
      executionTimeMillis: 2,
      totalKeysExamined: 6,
      totalDocsExamined: 6,
      executionStages: {
        stage: 'FETCH',
        nReturned: 6,
        executionTimeMillisEstimate: 0,
        works: 7,
        advanced: 6,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        docsExamined: 6,
        alreadyHasObj: 0,
        inputStage: {
          stage: 'IXSCAN',
          nReturned: 6,
          executionTimeMillisEstimate: 0,
```

```
            works: 7,
            advanced: 6,
            needTime: 0,
            needYield: 0,
            saveState: 0,
            restoreState: 0,
            isEOF: 1,
            keyPattern: { elevation: 1 },
            indexName: 'elevation_1',
            isMultiKey: false,
            multiKeyPaths: { elevation: []
},
            isUnique: false,
            isSparse: false,
            isPartial: false,
            indexVersion: 2,
            direction: 'forward',
            indexBounds: { elevation: [ '[-
inf.0, 8000)' ] },
            keysExamined: 6,
            seeks: 1,
            dupsTested: 0,
            dupsDropped: 0
          }
        }
      },
      command: {
        find: 'weather_data',
```

```
    filter: { elevation: { '$lt': 8000
} },
    '$db': 'new_weather_db'
  },
  serverInfo: {
    host: 'Sankets-MacBook-Pro.local',
    port: 27017,
    version: '5.0.7',
    gitVersion:
'b977129dc70eed766cbee7e412d901ee213acb
da'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes:
104857600,

internalQueryFacetMaxOutputDocSizeBytes
: 104857600,

internalLookupStageIntermediateDocument
MaxSizeBytes: 104857600,

internalDocumentSourceGroupMaxMemoryByt
es: 104857600,

internalQueryMaxBlockingSortMemoryUsage
Bytes: 104857600,
```

```
    internalQueryProhibitBlockingMergeOnMon
    goS: 0,
        internalQueryMaxAddToSetBytes:
    104857600,

    internalDocumentSourceSetWindowFieldsMa
    xMemoryBytes: 104857600
      },
      ok: 1
    }
```

If you will check the `docsExamined` property of the above 2 JSONs, you can see a significant difference.