# BIG DATA

SECTION D

Fall’2023

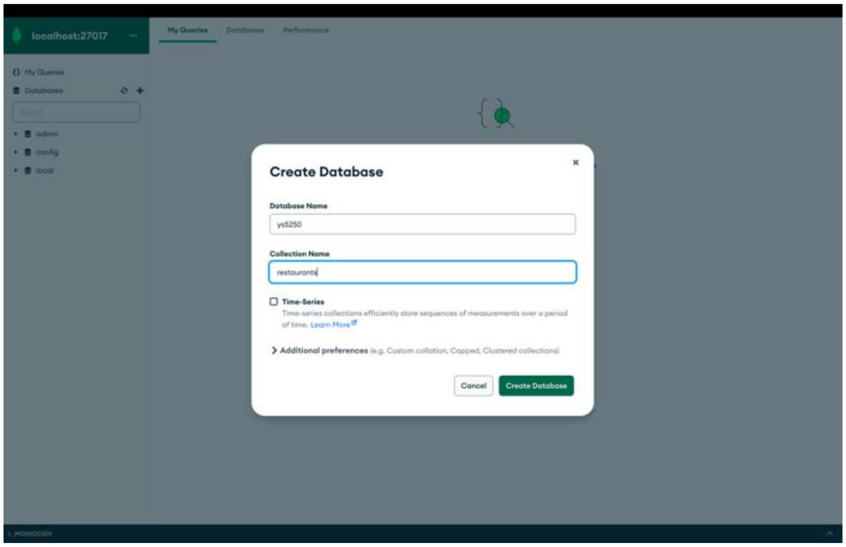
Yogya Sharma

ys5250

GHW#4

1. **Using the restaurants.json file, answer the following mongoDB questions.**

i) Load the file into mongoDB, either using the Compass or through the command line.

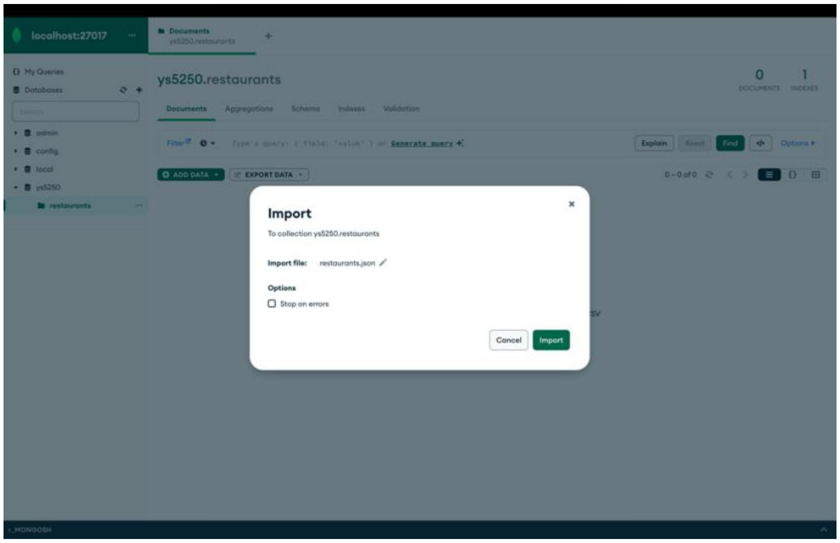


Create Database

Database Name: ys5250

Collection Name: restaurants

□ Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More



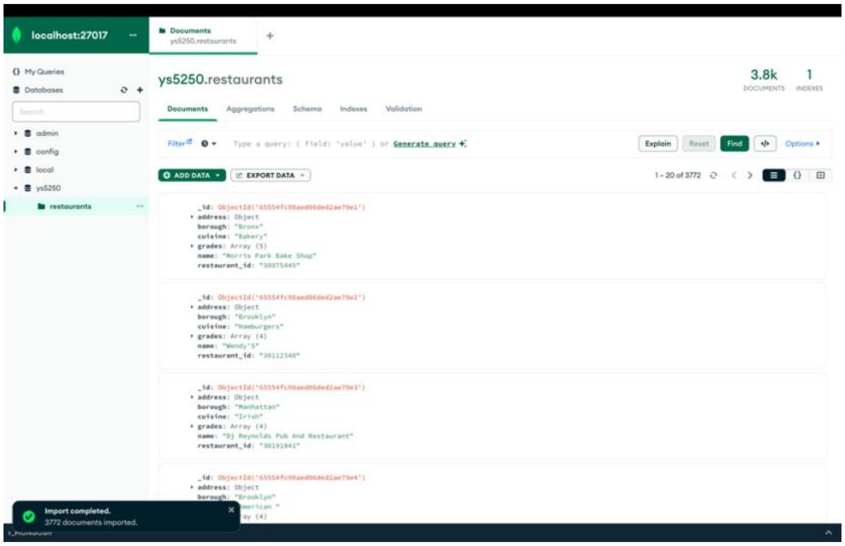
Import

To collection ys5250.restaurants

Import file: restaurants.json

Options

□ Stop on errors



ys5250.restaurants

3.8k DOCUMENTS 1 INDEXES

1 – 20 of 3772

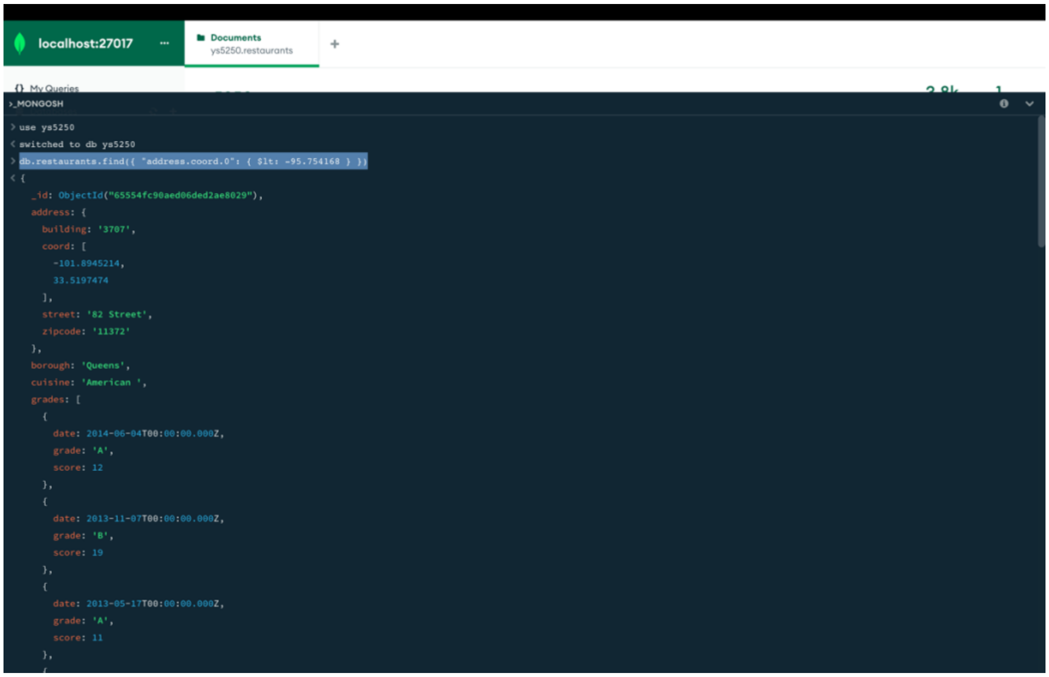
```
_id: ObjectId('65554fc90aed06ded2ae79e1')
▸ address: Object
borough: "Bronx"
cuisine: "Bakery"
▸ grades: Array (5)
name: "Morris Park Bake Shop"
restaurant_id: "30075445"

_id: ObjectId('65554fc90aed06ded2ae79e2')
▸ address: Object
borough: "Brooklyn"
cuisine: "Hamburgers"
▸ grades: Array (4)
name: "Wendy'S"
restaurant_id: "30112340"

_id: ObjectId('65554fc90aed06ded2ae79e3')
▸ address: Object
borough: "Manhattan"
cuisine: "Irish"
▸ grades: Array (4)
name: "Dj Reynolds Pub And Restaurant"
restaurant_id: "30191841"

_id: ObjectId('65554fc90aed06ded2ae79e4')
▸ address: Object
borough: "Brooklyn"
American "
ay (4)
```

Import completed.
3772 documents imported.

ii) Find the restaurants which are located in a longitude value less than -95.754168 (the coord array is of the form [longitude,latitude].

To just get all details of the restaurants which are located in a longitude value less than -95.754168:

Query: db.restaurants.find({ "address.coord.0": { $lt: -95.754168 } })

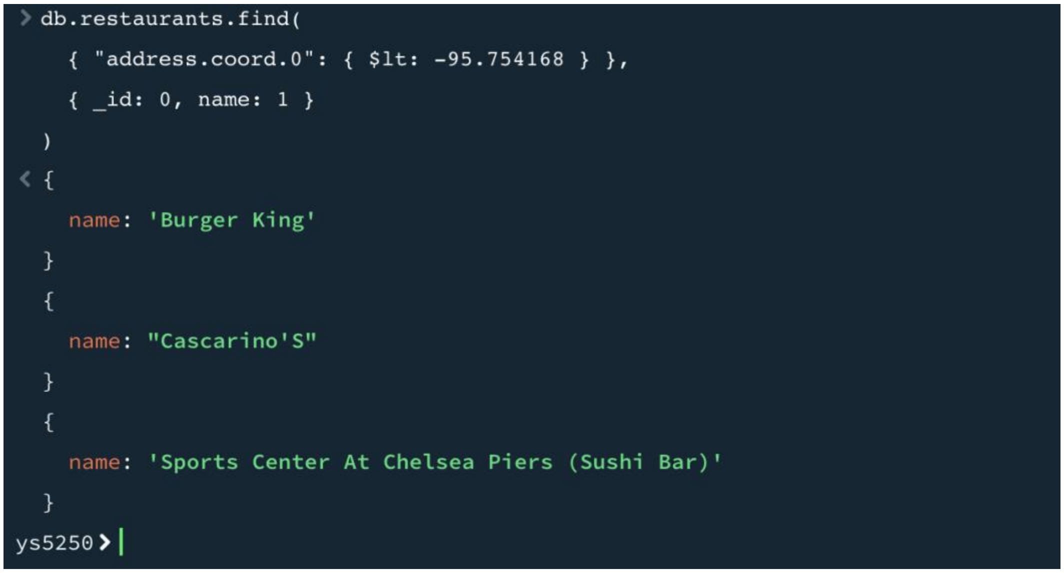


```
> use ys5250
< switched to db ys5250
> db.restaurants.find({ "address.coord.0": { $lt: -95.754168 } })
< {
    _id: ObjectId("65554fc90aed06ded2ae8029"),
    address: {
      building: '3707',
      coord: [
        -101.8945214,
        33.5197474
      ],
      street: '82 Street',
      zipcode: '11372'
    },
    borough: 'Queens',
    cuisine: 'American ',
    grades: [
      {
        date: 2014-06-04T00:00:00.000Z,
        grade: 'A',
        score: 12
      },
      {
        date: 2013-11-07T00:00:00.000Z,
        grade: 'B',
        score: 19
      },
      {
        date: 2013-05-17T00:00:00.000Z,
        grade: 'A',
        score: 11
      },
```

To just get the names of these restaurants:

Query:

```
db. restaurants.find(
  { "address.coord.0": { $lt: -95.754168 } },
  { _id: 0, name: 1 }
)
```

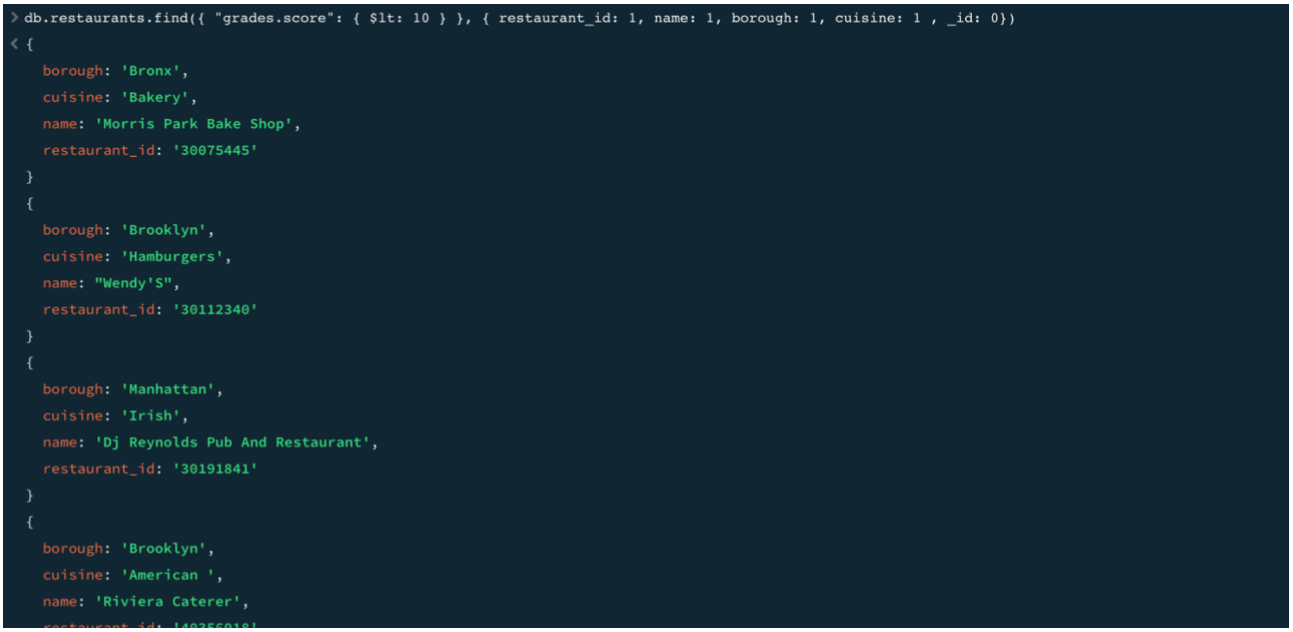


```
> db.restaurants.find(
    { "address.coord.0": { $lt: -95.754168 } },
    { _id: 0, name: 1 }
  )
< {
    name: 'Burger King'
  }
  {
    name: "Cascarino'S"
  }
  {
    name: 'Sports Center At Chelsea Piers (Sushi Bar)'
  }
ys5250>
```

iii) Find the restaurant Id, name, borough and cuisine for those restaurants which achieved a score below 10.

Query:

db.restaurants.find({ "grades.score": { $lt: 10 } }, { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 , _id: 0})

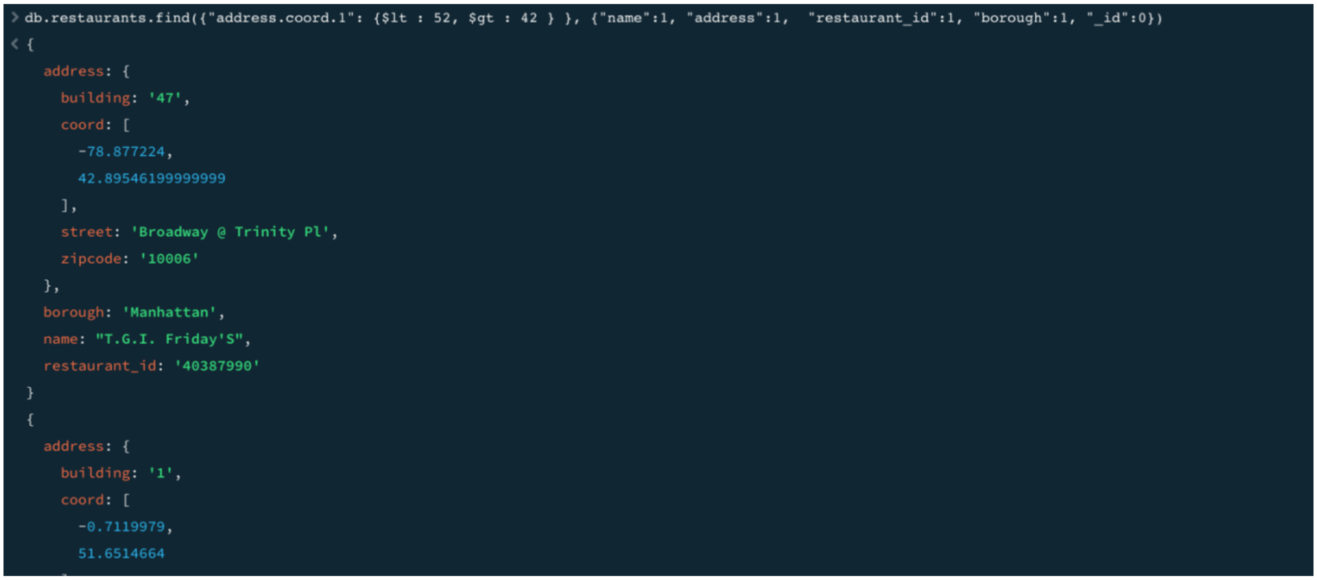


```
> db.restaurants.find({ "grades.score": { $lt: 10 } }, { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 , _id: 0})
< {
    borough: 'Bronx',
    cuisine: 'Bakery',
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
  {
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
    name: "Wendy'S",
    restaurant_id: '30112340'
  }
  {
    borough: 'Manhattan',
    cuisine: 'Irish',
    name: 'Dj Reynolds Pub And Restaurant',
    restaurant_id: '30191841'
  }
  {
    borough: 'Brooklyn',
    cuisine: 'American ',
    name: 'Riviera Caterer',
    restaurant_id: '40356018'
```

iv) Find the restaurant Id, name, address and geographical location for those restaurants where the 2nd element of the coord array contains a value which is more than 42 and less than 52.

Query:

db.restaurants.find({"address.coord.1": {$lt : 52, $gt : 42 } }, {"name":1, "address":1, "restaurant_id":1, "borough":1, "_id":0})

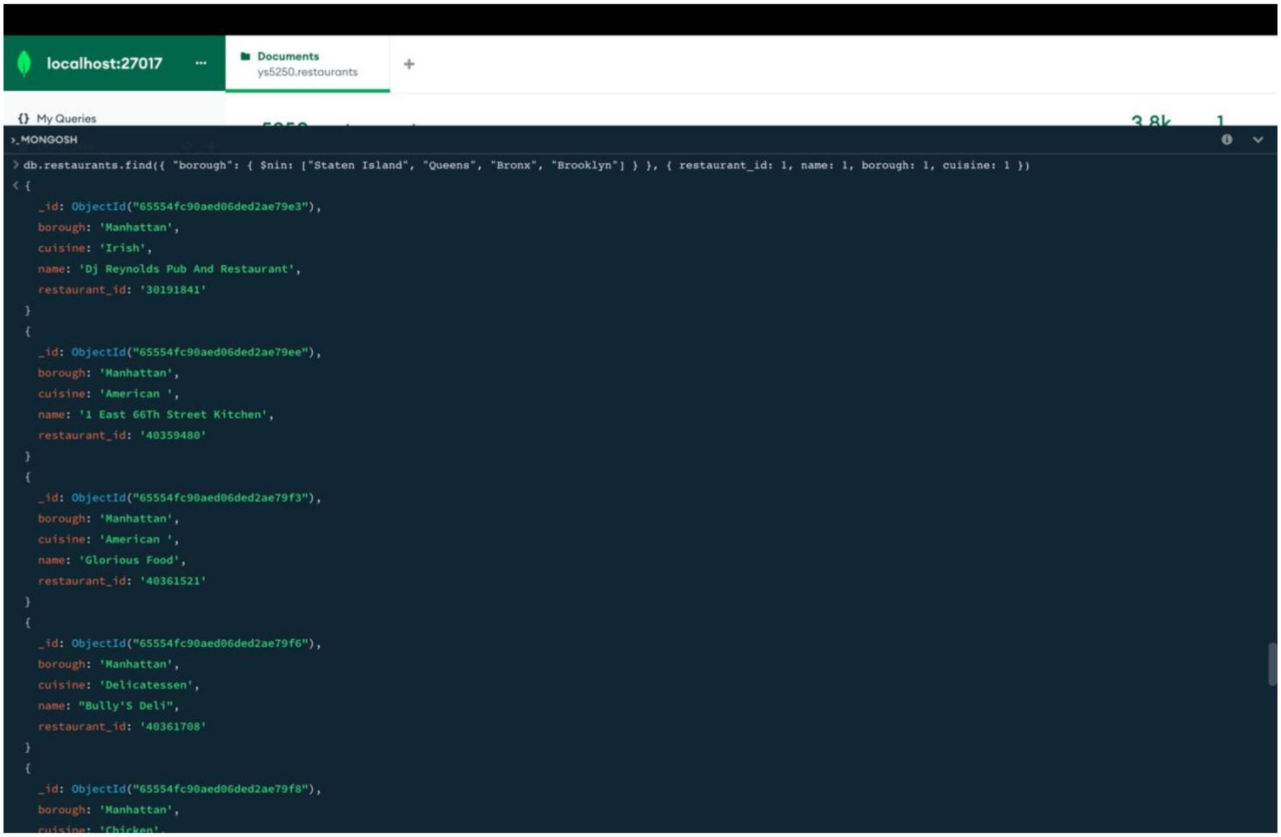


```
> db.restaurants.find({"address.coord.1": {$lt : 52, $gt : 42 } }, {"name":1, "address":1,  "restaurant_id":1, "borough":1, "_id":0})
< {
    address: {
      building: '47',
      coord: [
        -78.877224,
        42.89546199999999
      ],
      street: 'Broadway @ Trinity Pl',
      zipcode: '10006'
    },
    borough: 'Manhattan',
    name: "T.G.I. Friday'S",
    restaurant_id: '40387990'
  }
  {
    address: {
      building: '1',
      coord: [
        -0.7119979,
        51.6514664
```

v) Find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronx or Brooklyn.

Query:

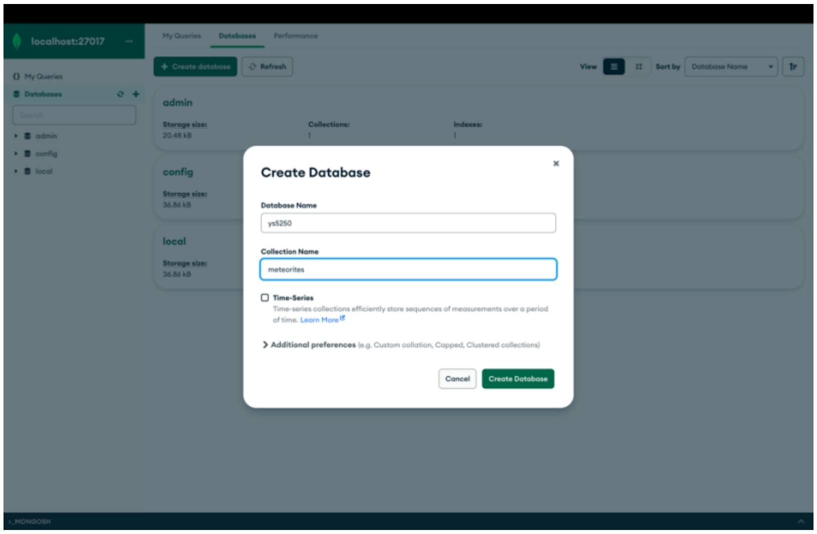
db.restaurants.find({ "borough": { $nin: ["Staten Island", "Queens", "Bronx", "Brooklyn"] } }, { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 })



```
> db.restaurants.find({ "borough": { $nin: ["Staten Island", "Queens", "Bronx", "Brooklyn"] } }, { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 })
< {
    _id: ObjectId("65554fc90aed06ded2ae79e3"),
    borough: 'Manhattan',
    cuisine: 'Irish',
    name: 'Dj Reynolds Pub And Restaurant',
    restaurant_id: '30191841'
  }
  {
    _id: ObjectId("65554fc90aed06ded2ae79ee"),
    borough: 'Manhattan',
    cuisine: 'American ',
    name: '1 East 66Th Street Kitchen',
    restaurant_id: '40359480'
  }
  {
    _id: ObjectId("65554fc90aed06ded2ae79f3"),
    borough: 'Manhattan',
    cuisine: 'American ',
    name: 'Glorious Food',
    restaurant_id: '40361521'
  }
  {
    _id: ObjectId("65554fc90aed06ded2ae79f6"),
    borough: 'Manhattan',
    cuisine: 'Delicatessen',
    name: "Bully'S Deli",
    restaurant_id: '40361708'
  }
  {
    _id: ObjectId("65554fc90aed06ded2ae79f8"),
    borough: 'Manhattan',
    cuisine: 'Chicken',
```

2. MongoDB geospatial

<u>Create a database, create collections and upload data:</u>



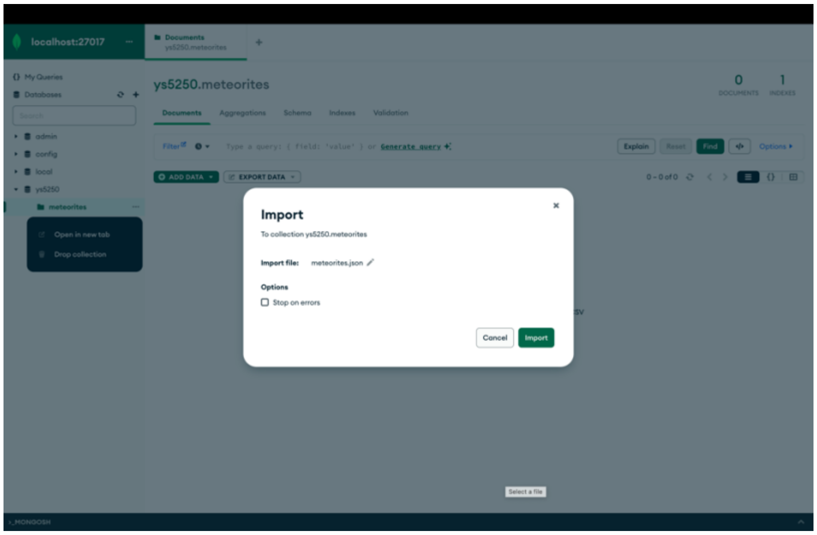
Create Database

Database Name: ys5250

Collection Name: meteorites

□ 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)



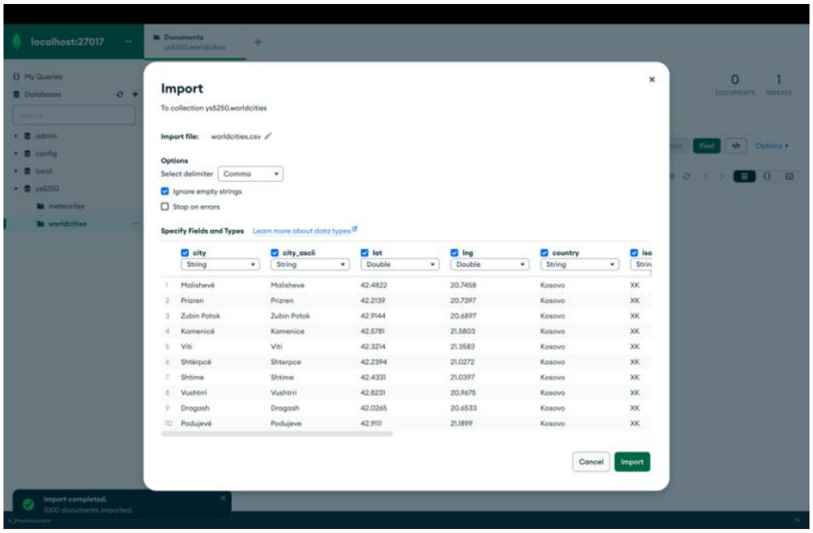
Import

To collection ys5250.meteorites

Import file: meteorites.json

Options

□ Stop on errors



Import

To collection ys5250.worldcities

Import file: worldcities.csv

Options

Select delimiter: Comma

☑ Ignore empty strings

□ Stop on errors

Specify Fields and Types

| | ☑ city | ☑ city_ascii | ☑ lat | ☑ lng | ☑ country | ☑ iso |
|---|---|---|---|---|---|---|
| | String | String | Double | Double | String | Strin |
| 1 | Malishevë | Malisheve | 42.4822 | 20.7458 | Kosovo | XK |
| 2 | Prizren | Prizren | 42.2139 | 20.7397 | Kosovo | XK |
| 3 | Zubin Potok | Zubin Potok | 42.9144 | 20.6897 | Kosovo | XK |
| 4 | Kamenicë | Kamenice | 42.5781 | 21.5803 | Kosovo | XK |
| 5 | Viti | Viti | 42.3214 | 21.3583 | Kosovo | XK |
| 6 | Shtërpcë | Shterpce | 42.2394 | 21.0272 | Kosovo | XK |
| 7 | Shtime | Shtime | 42.4331 | 21.0397 | Kosovo | XK |
| 8 | Vushtrri | Vushtrri | 42.8231 | 20.9675 | Kosovo | XK |
| 9 | Dragash | Dragash | 42.0265 | 20.6533 | Kosovo | XK |
| 10 | Podujevë | Podujeve | 42.9111 | 21.1899 | Kosovo | XK |

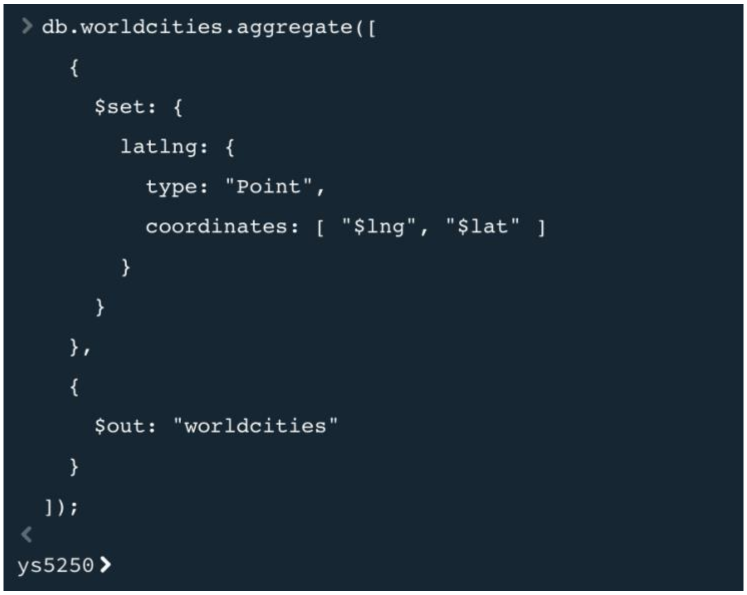
Import completed.
1000 documents imported.

## Create indexes for geospatial queries in MongoDB:

i) The createIndex command expects the geo keys to be in an array or object format, but the lat and lng fields are separate in the worldcities document. To resolve this, we create a new field in worldcities document that combines the lat and lng into a GeoJSON object,  using the createIndex command with a $jsonSchema and $set stage in an aggregation pipeline.

Command:

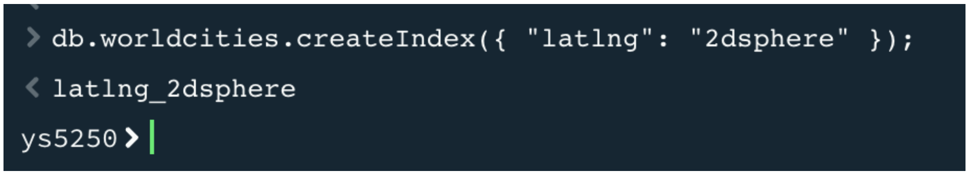
```
db.worldcities.aggregate([
  {
    $set: {
      latlng: {
        type: "Point",
        coordinates: [ "$lng", "$lat" ]
      }
    }
  },
  {
    $out: "worldcities"
  }
]);
```



```
> db.worldcities.aggregate([
    {
      $set: {
        latlng: {
          type: "Point",
          coordinates: [ "$lng", "$lat" ]
        }
      }
    },
    {
      $out: "worldcities"
    }
  ]);
<
ys5250 >
```

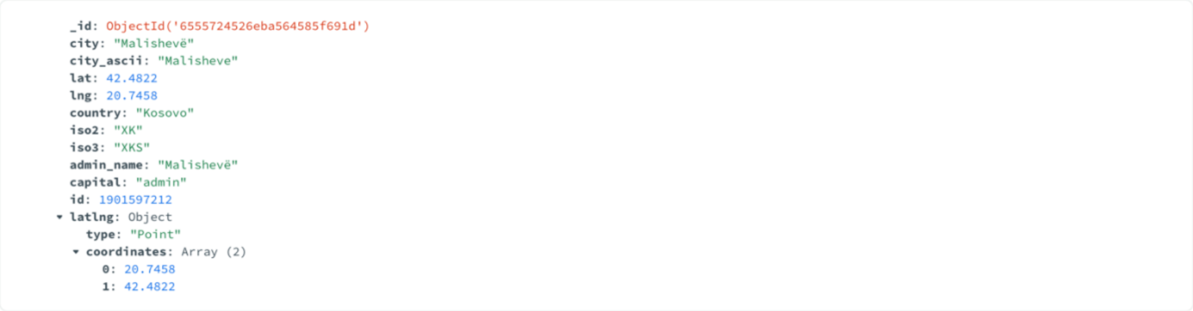
Creating the indexes as a 2dsphere:

Command: db.worldcities.createIndex({ "latlng": "2dsphere" });



```
> db.worldcities.createIndex({ "latlng": "2dsphere" });
< latlng_2dsphere
ys5250 >
```

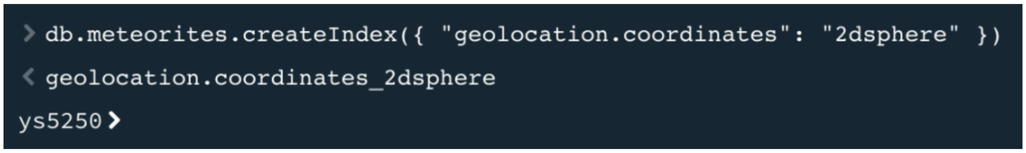
We can see the latlng field here:



```
   _id: ObjectId('6555724526eba564585f691d')
   city: "Malishevë"
   city_ascii: "Malisheve"
   lat: 42.4822
   lng: 20.7458
   country: "Kosovo"
   iso2: "XK"
   iso3: "XKS"
   admin_name: "Malishevë"
   capital: "admin"
   id: 1901597212
 ▾ latlng: Object
     type: "Point"
   ▾ coordinates: Array (2)
       0: 20.7458
       1: 42.4822
```

ii) Next, create indexes for meteorites collection.

Command: db.meteorites.createIndex({ "geolocation.coordinates": "2dsphere" })

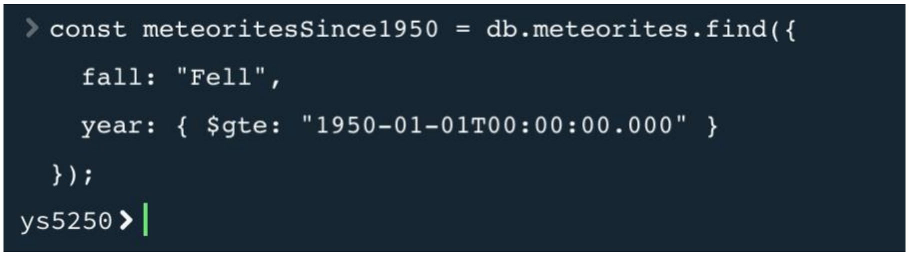


```
> db.meteorites.createIndex({ "geolocation.coordinates": "2dsphere" })
< geolocation.coordinates_2dsphere
ys5250 >
```

<u>Filter meteorites since 1950:</u>

Command:

```
const meteoritesSince1950 = db.meteorites.find({
  fall: "Fell",
  year: { $gte: "1950-01-01T00:00:00.000" }
});
```

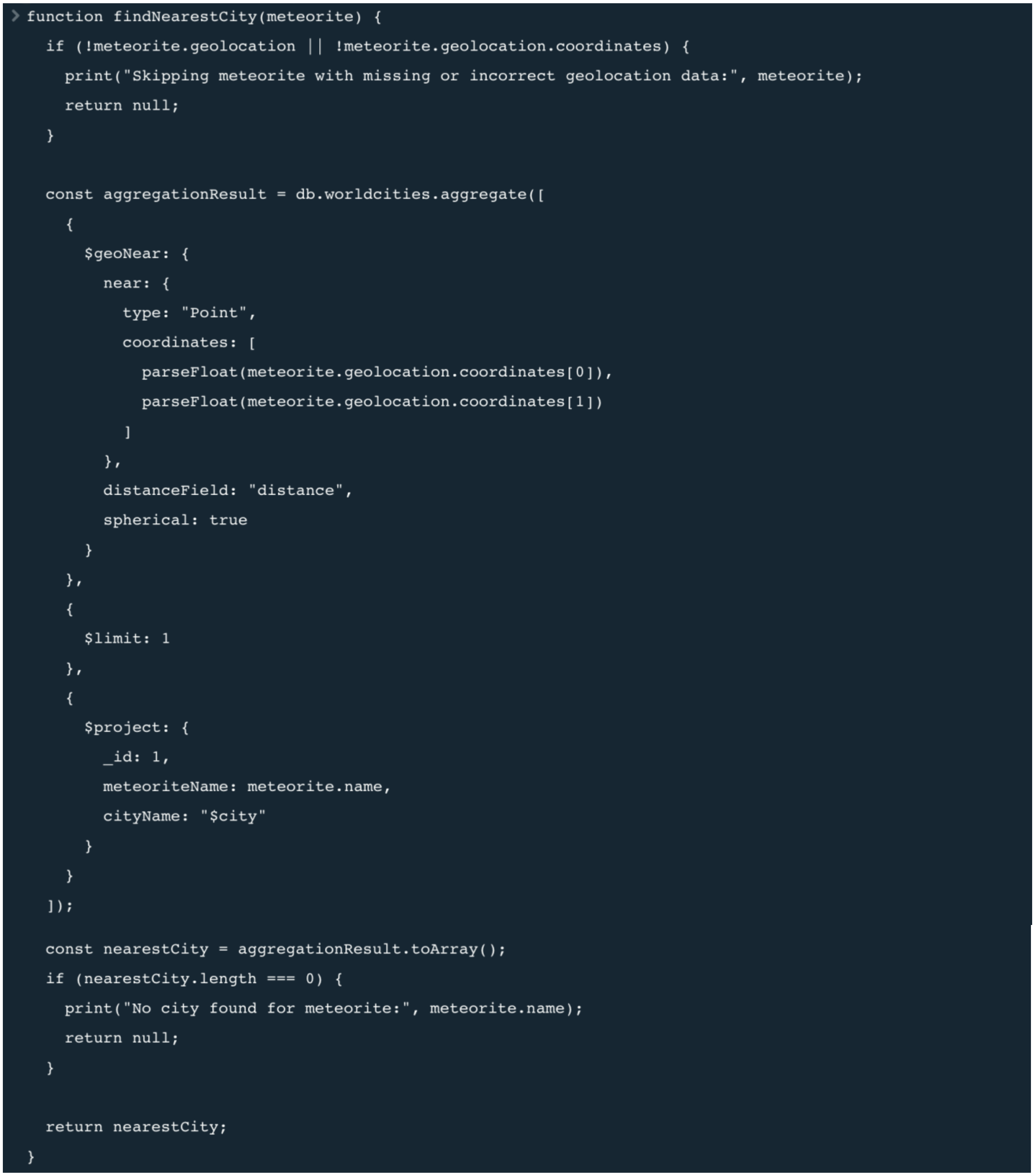


```
> const meteoritesSince1950 = db.meteorites.find({
    fall: "Fell",
    year: { $gte: "1950-01-01T00:00:00.000" }
  });
ys5250 >
```

## Creating a function to find the nearest city:

Command:

```
function findNearestCity(meteorite) {
  if (!meteorite.geolocation || !meteorite.geolocation.coordinates) {
    print("Skipping meteorite with missing or incorrect geolocation data:", meteorite);
    return null;
  }

  const aggregationResult = db.worldcities.aggregate([
    {
      $geoNear: {
        near: {
          type: "Point",
          coordinates: [
            parseFloat(meteorite.geolocation.coordinates[0]),
            parseFloat(meteorite.geolocation.coordinates[1])
          ]
        },
        distanceField: "distance",
        spherical: true
      }
    },
    {
      $limit: 1
    },
    {
      $project: {
        _id: 1,
        meteoriteName: meteorite.name,
        cityName: "$city"
      }
    }
  ]);

  const nearestCity = aggregationResult.toArray();
  if (nearestCity.length === 0) {
    print("No city found for meteorite:", meteorite.name);
    return null;
  }

  return nearestCity;
}
```

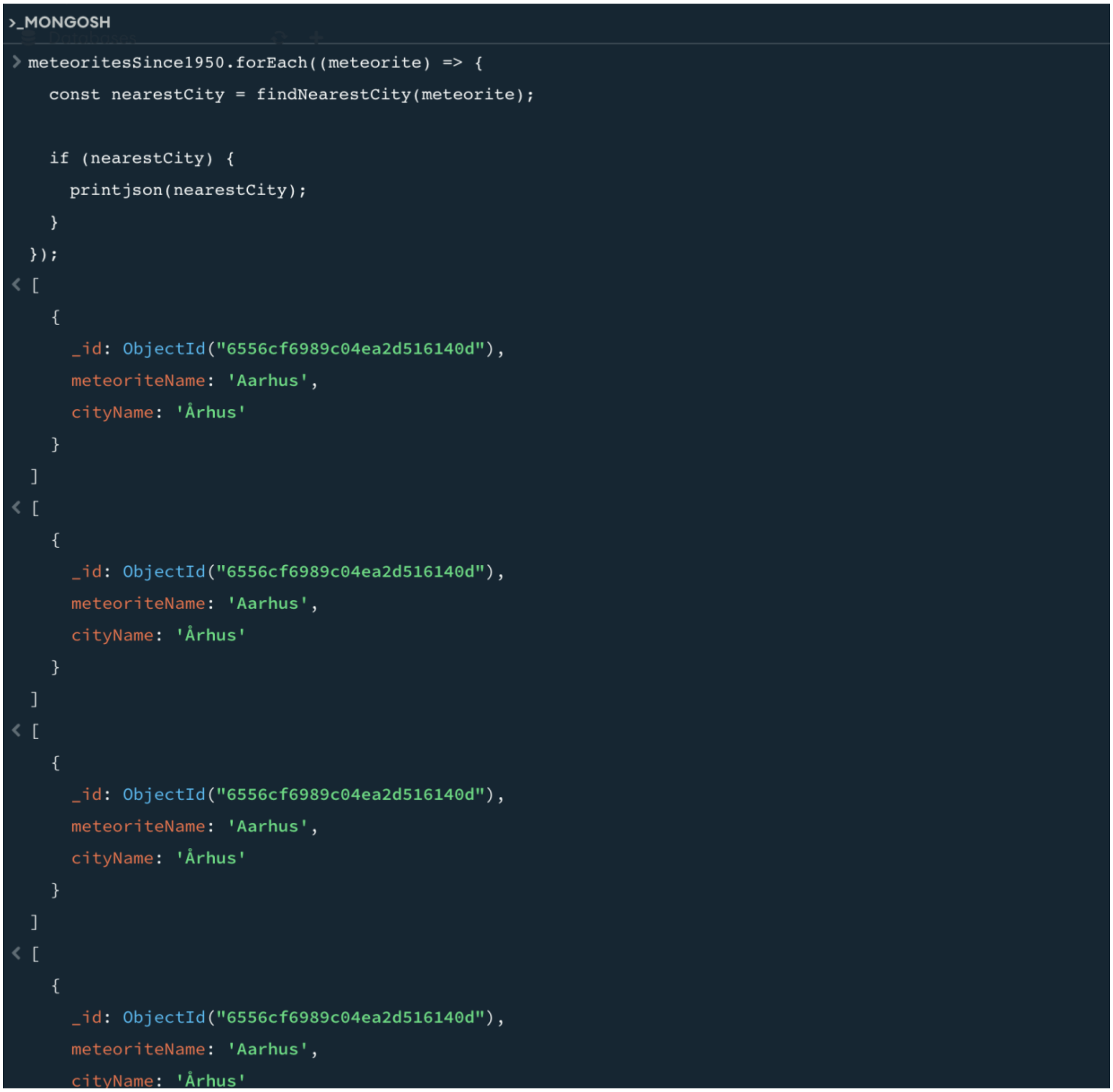


```
function findNearestCity(meteorite) {
  if (!meteorite.geolocation || !meteorite.geolocation.coordinates) {
    print("Skipping meteorite with missing or incorrect geolocation data:", meteorite);
    return null;
  }

  const aggregationResult = db.worldcities.aggregate([
    {
      $geoNear: {
        near: {
          type: "Point",
          coordinates: [
            parseFloat(meteorite.geolocation.coordinates[0]),
            parseFloat(meteorite.geolocation.coordinates[1])
          ]
        },
        distanceField: "distance",
        spherical: true
      }
    },
    {
      $limit: 1
    },
    {
      $project: {
        _id: 1,
        meteoriteName: meteorite.name,
        cityName: "$city"
      }
    }
  ]);

  const nearestCity = aggregationResult.toArray();
  if (nearestCity.length === 0) {
    print("No city found for meteorite:", meteorite.name);
    return null;
  }

  return nearestCity;
}
```

Using the findNearestCity function in a loop to find the nearest city for each meteorite:

Command:

```
meteoritesSince1950.forEach((meteorite) => {
  const nearestCity = findNearestCity(meteorite);

  if (nearestCity) {
    printjson(nearestCity);
  }
});
```



```
> meteoritesSince1950.forEach((meteorite) => {
    const nearestCity = findNearestCity(meteorite);

    if (nearestCity) {
      printjson(nearestCity);
    }
  });
< [
    {
      _id: ObjectId("6556cf6989c04ea2d516140d"),
      meteoriteName: 'Aarhus',
      cityName: 'Århus'
    }
  ]
< [
    {
      _id: ObjectId("6556cf6989c04ea2d516140d"),
      meteoriteName: 'Aarhus',
      cityName: 'Århus'
    }
  ]
< [
    {
      _id: ObjectId("6556cf6989c04ea2d516140d"),
      meteoriteName: 'Aarhus',
      cityName: 'Århus'
    }
  ]
< [
    {
      _id: ObjectId("6556cf6989c04ea2d516140d"),
      meteoriteName: 'Aarhus',
      cityName: 'Århus'
```