

# Introduction to



## Hands-On Workshop

### Lab 2 - Compass

## Overview

In Lab 1 of this workshop you've set the foundation by creating a MongoDB cluster using Atlas, MongoDB's cloud-based database-as-a-service. You will now learn how to load data into this cluster and interact with the data using the MongoDB query language. To do this, you will use a tool called **MongoDB Compass**, a desktop application that acts as front-end GUI for MongoDB.

## Prerequisites

You've completed Lab 1, in which you used **MongoDB Atlas** to deploy a MongoDB cluster. You should also have installed **Compass** and downloaded the sample dataset of restaurant reviews. If you haven't already done that, those instructions are repeated here as *Lab Prep*. Otherwise, you can skip ahead to Exercise 1.

## Hands-On Exercises

### Lab Prep: Install Compass & Download Sample Dataset

If you haven't done so already, **download and install Compass**, the GUI for MongoDB. Go to the [download](#) page and select the latest Version and appropriate Platform:

*Note, don't use the Compass download link in Atlas as it downloads the Community Edition of Compass.*

Version: 1.19.12 (Stable) Platforms: Windows 64-bit (7+)

Download

Make sure you select the “Stable” version, which contains the enterprise features we’ll use in a moment:

Version

- ✓ 1.19.12 (Stable)
- 1.19.12 (Community Edition Stable)
- 1.19.12 (Readonly Edition Stable)
- 1.19.12 (Isolated Edition Stable)

Also, if you lack the administrative privileges to install software on your Windows laptop, select the **Windows 64-bit (7+)** option, which is the executable:

- OS X 64-bit (10.10+)
- ✓ Windows 64-bit (7+)
- Windows 64-bit (7+) (Zip)
- Windows 64-bit (7+) (MSI)
- Ubuntu 64-bit (14.04+)
- RedHat 64-bit (7+)

Once downloaded, install Compass on your local workstation.

Next, **download the sample dataset** from GitHub. For this workshop, we’re going to load a Yelp-like collection of New York City restaurants. If you have the *wget* utility, you can get the dataset as follows:

```
wget https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json
```

Or, more simply:

```
wget http://bit.ly/MongoWorkshopData
```

If you don’t have *wget*, just open the link in your browser **and wait for the page load to complete**. Then save the file (in Chrome, for example, select File > Save Page As).

The dataset is 11.9 MB and has 25K restaurants.

## Exercise 1 - Connect to the Cluster

*We aren't using the optional sample data for this workshop, so you can skip that step. However, these sample data sets provided by Atlas are a great way to gain familiarity with MongoDB. To load them at a later time, navigate to your cluster, click the button with the three ellipses, and then select **Load Sample Dataset**.*

### Connect to Atlas

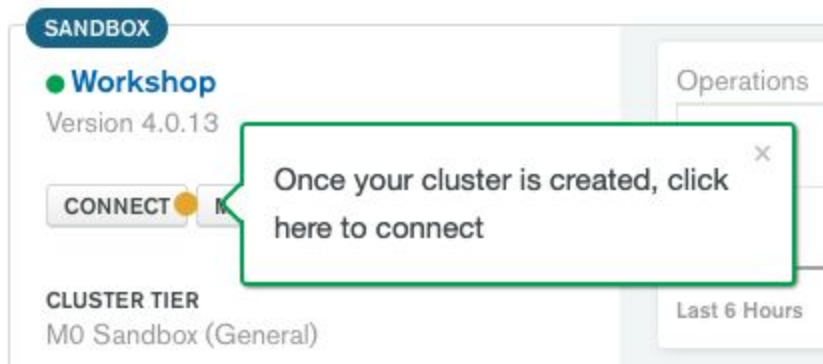
Follow this checklist to get started.

60%

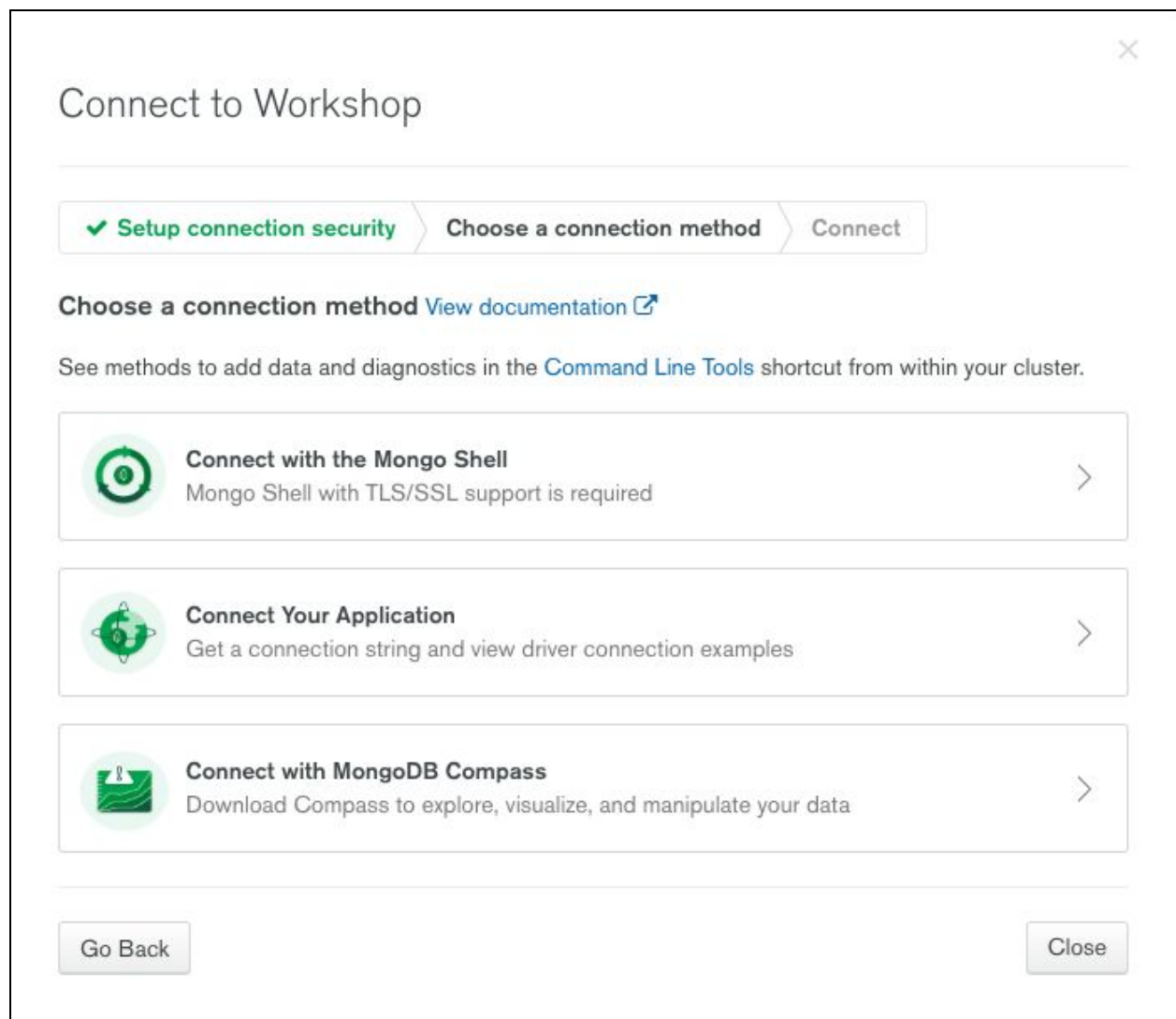
- ☒ Build your first cluster
- ☒ Create your first database user
- ☒ Whitelist your IP address
- ☐ Load Sample Data (Optional)
- ☐ Connect to your cluster

No thanks

Click the **CONNECT** button for your Workshop cluster:



There are several methods to connect to MongoDB, notably from the Shell (not used in this workshop), from your application code (also not used in this workshop) and from Compass: the GUI for MongoDB:



Select **Connect with MongoDB Compass**. Then select **I have Compass** and **COPY** the connection string presented:

Connect to Workshop

✓ Setup connection security > ✓ Choose a connection method > Connect

I do not have Compass I have Compass

1 Choose your version of Compass:

1.12 or later

See your Compass version in "About Compass"

2 Copy the connection string below, and open Compass:

mongodb+srv://workshop:<password>@workshop-6umr3.mongodb.net/te

Copy

Replace <password> with the password for the workshop user.  
When entering your password, make sure that any special characters are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back Close

## Connect Compass

Start Compass and paste your connection string into the New Connection window. Replace the username and password, ensuring that you remove all sets of brackets in the process:

# New Connection

☆ FAVORITE

Fill in connection fields individually

Paste your connection string (SRV or Standard ⓘ)

mongodb+srv://worshop:workshop@workshop-quel1.mongodb.net/test

CONNECT

Before clicking **CONNECT**, click the **FAVORITE** button to create a favorite named **Workshop**. This will allow us to quickly connect to the cluster in the future.

# New Connection

☆ FAVORITE

Now click **CONNECT**. If successful, you'll see some internal databases used by MongoDB:

MongoDB Compass - workshop-ghuzr.mongodb.net:27017

Workshop-shard-0 REPLICA SET 3 NODES

MongoDB 4.0.4 Enterprise

Workshop 3 DBS 7 COLLECTIONS

filter

> admin

> config

> local

CREATE DATABASE

Database Name	Storage Size	Collections	Indexes	
admin	0.0B	0	0	
config	0.0B	1	0	
local	0.0B	6	0	

## Exercise 2 - Load Data

For this workshop we're going to load a Yelp-like collection of New York City restaurants. Download the dataset from Github. If you have the wget utility, you can get the dataset as follows:

wget

<https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json>

Otherwise, just open the link in your browser **and wait for the page load to complete** Then save the file (File > Save Page As in Chrome)

The dataset is 11.9 MB and has 25K restaurants.

### Create a Database and Collection

In Compass, click the **CREATE DATABASE** button and create a database named **Workshop** with a collection named **Restaurants**:

### Create Database

Database Name

Workshop

Collection Name

Restaurants

☐ Capped Collection ⓘ

☐ Use Custom Collation ⓘ

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

CANCEL

CREATE DATABASE

Navigate to the Restaurants collection and then either select **Import Data** from the menu (Collection > Import Data) or click on the Import Data button as shown below:




Workshop.Restaurants

DOCUMENTS	TOTAL SIZE	AVG. SIZE	INDEXES	TOTAL SIZE	AVG. SIZE
0	0B	0B	1	4.0KB	4.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

Displaying documents 0 - 0 of N/A



## This collection has no data

It only takes a few seconds to import data from a JSON or CSV file

Then **BROWSE** to the primer-dataset.json file you downloaded previously:

Import To Collection Workshop.Restaurants

**Select Input File Type**

**Select File**

Now click **IMPORT**. You've just imported 25K documents!

*Note, sometimes in these workshop environments, the import will timeout before it can successfully complete. For the purpose of this workshop, it's not a problem.*

## Exercise 3 - Browse the Documents

Click the down arrow that appears when you hover over a document to fully expand the document. Notice how the restaurant documents have a nested subdocument (address, type Object) and an array of subdocuments (grades, type Array):

```

  ▾ _id: ObjectId("5dadf4042f9547ef12603f12")
    ▾ address: Object
      building: "1007"
      ▾ coord: Array
        0: -73.856077
        1: 40.848447
      street: "Morris Park Ave"
      zipcode: "10462"
      borough: "Bronx"
      cuisine: "Bakery"
    ▾ grades: Array
      ▾ 0: Object
        date: 2014-03-03T00:00:00.000+00:00
        grade: "A"
        score: 2
      ▾ 1: Object
        date: 2013-09-11T00:00:00.000+00:00
        grade: "A"
        score: 6
      ▾ 2: Object
        date: 2013-01-24T00:00:00.000+00:00
        grade: "A"
        score: 10
      ▾ 3: Object
        date: 2011-11-23T00:00:00.000+00:00
        grade: "A"
        score: 9
      ▾ 4: Object
        date: 2011-03-10T00:00:00.000+00:00
        grade: "B"
        score: 14
    name: "Morris Park Bake Shop"
    restaurant_id: "30075445"
```

In a relational database, these fields would most likely be separate tables, but MongoDB allows us to embed this information. Working with data in this natural way is much **easier** than decomposing and composing your business objects from relational tables.

## Exercise 4 - View the Schema

Wait, I thought MongoDB is a NoSQL database, and hence, didn't have a schema? While that's technically true, no dataset would be of any use without a schema. So while MongoDB doesn't enforce a schema, your collections of documents will still always have one. The key difference with MongoDB is that the schema can be **flexible**.

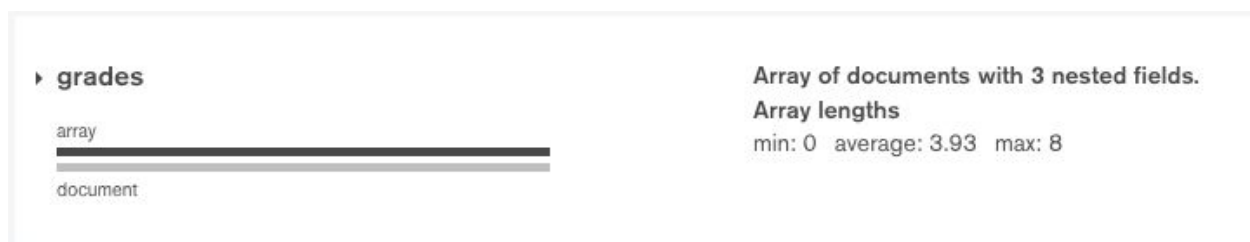
Select the **Schema** tab and select **Analyze Schema**.

*Note, if you don't have a **Schema** tab, you're running the Community Edition of Compass. Revist Lab 2 to get the correct edition.*

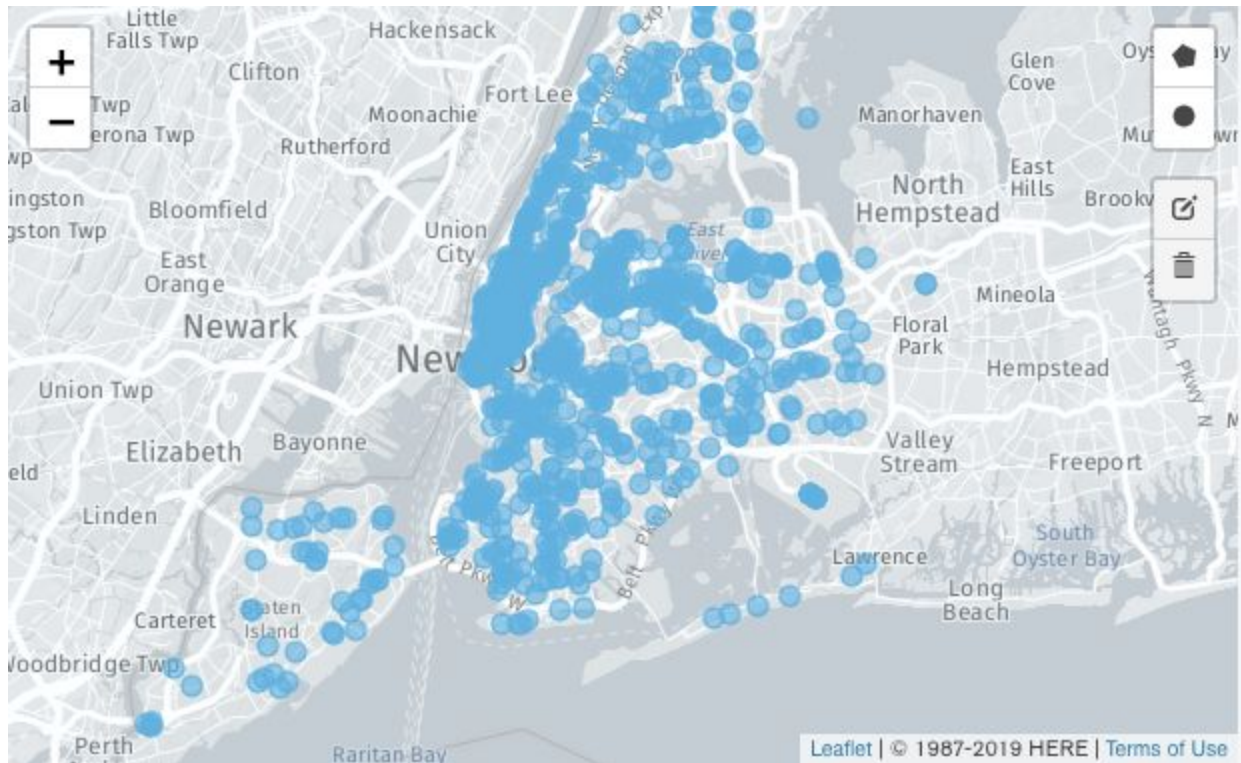
Compass will sample the documents in the collection to derive a schema. In addition to providing field names and types, Compass will also provide a summary of the data values. For example, for cuisine, we can see that Chinese is the 2nd most common at 12% (your results may differ slightly based on the sample that was taken):



For the array of grades, you'll find information on the number of fields in the sub-documents as well as the minimum, average and maximum number of elements in the array:

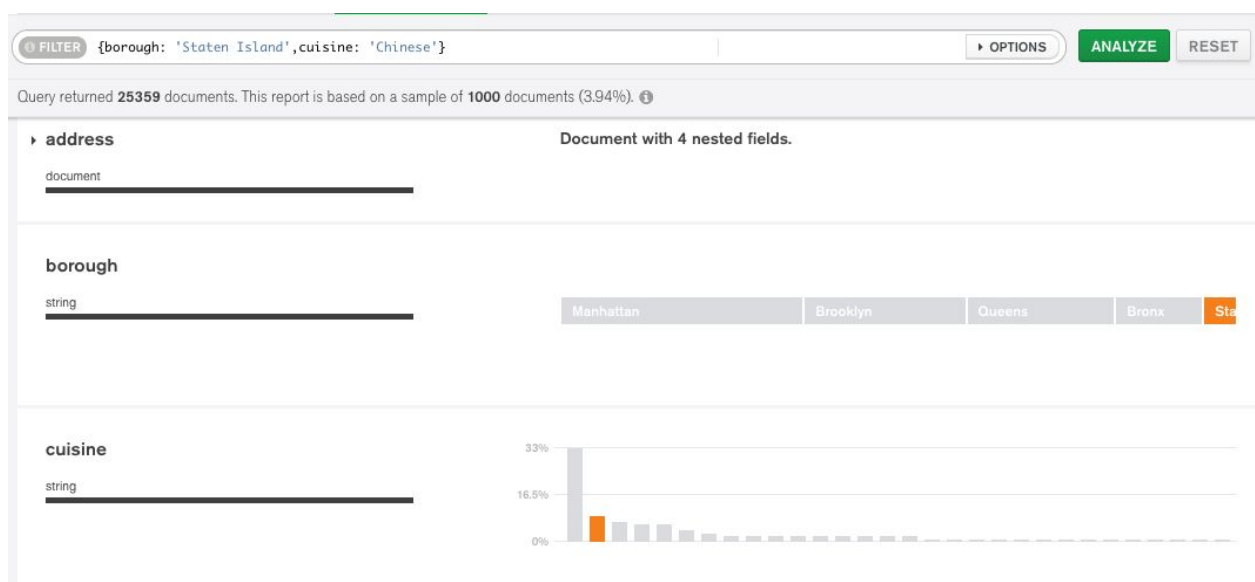


Expand the **address** field to discover MongoDB's excellent support for [Geospatial Queries](#). As the collection consists of restaurants in New York City, zoom the map to NYC:



## Exercise 5 - Query the Data

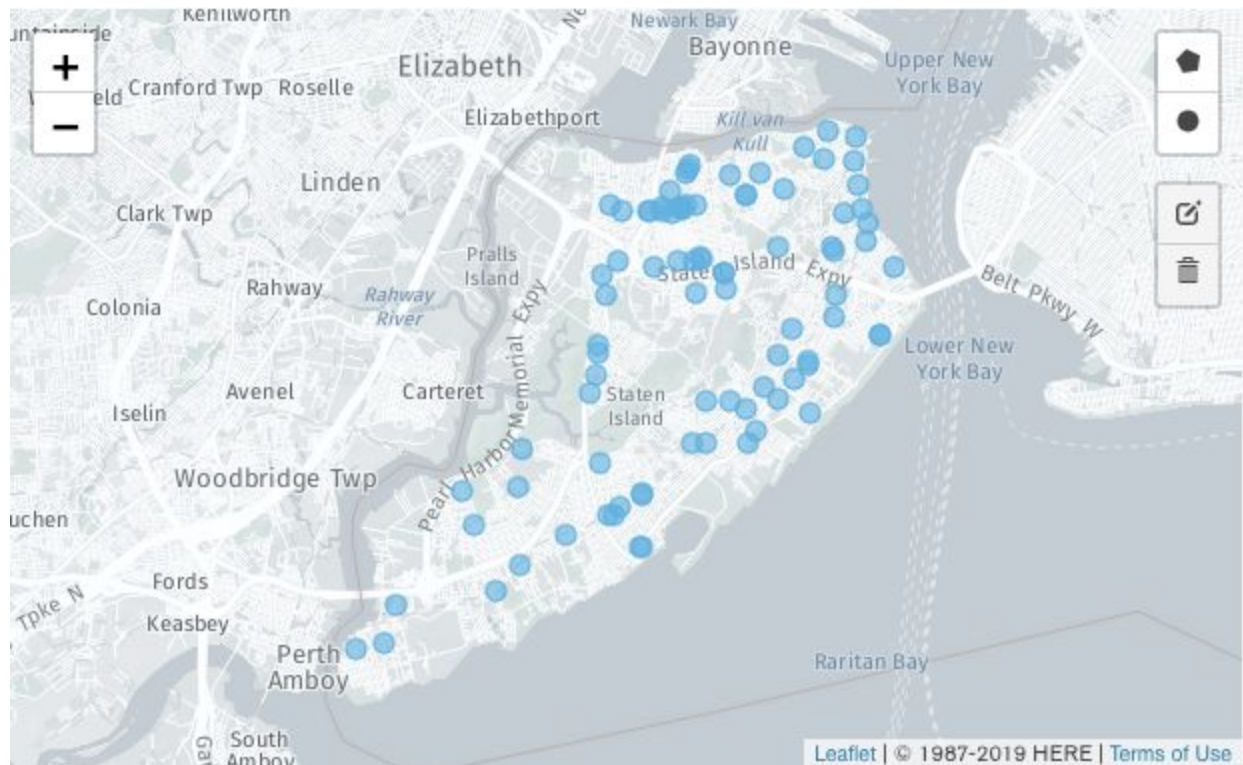
Unsurprisingly, the MongoDB Query Language (MQL) is also based on JSON. The Schema Analyzer in Compass provides an easy way to learn the language. For example, select **Staten Island** from the borough field (only **Sta** may be showing) and **Chinese** from the cuisine field. Notice as you make selections the FILTER field at the top of the window gets populated:



Click the **ANALYZE** button to filter for Chinese restaurants in Staten Island, of which there are 88:

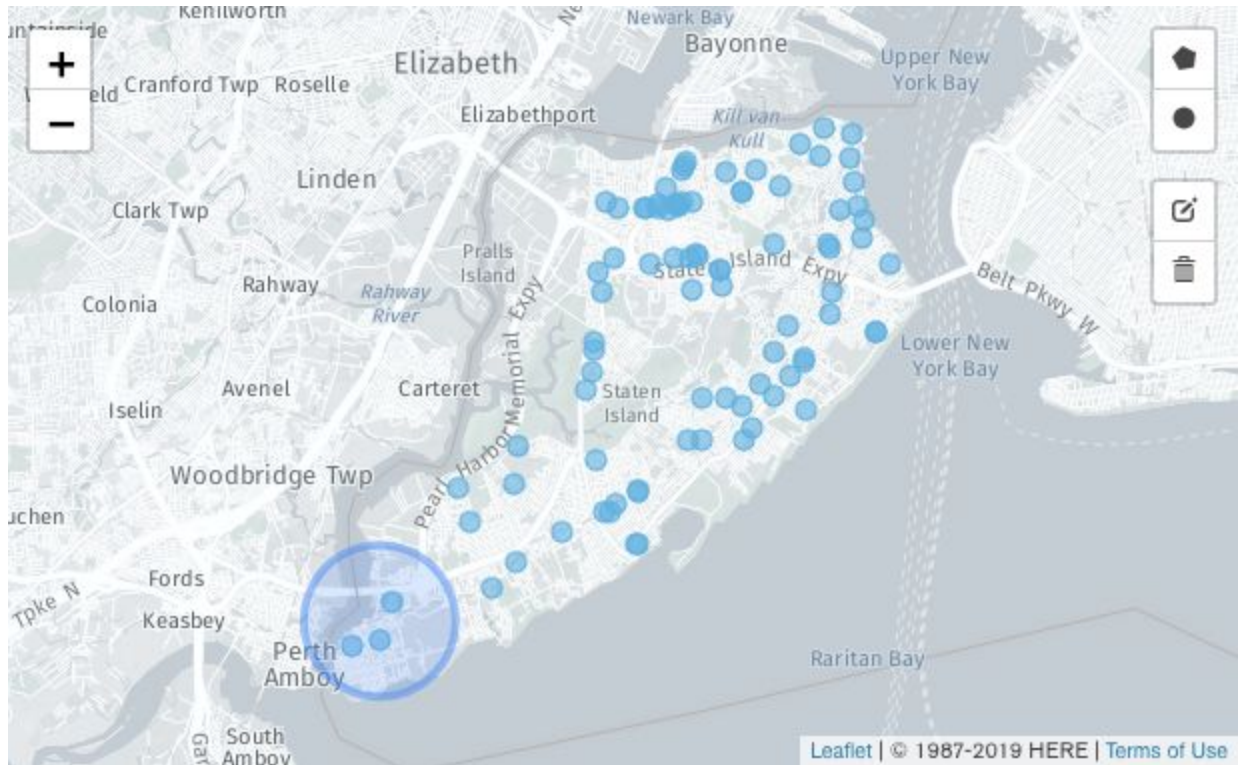
Query returned **88** documents. This report is based on a sample of **88** documents (100.00%). ⓘ

And you can now see this reflected on our map (more dots now appear on Staten Island because our sample now includes all 88 restaurants):



To perform a geospatial query, click the “Draw a circle” button in the top right of the map and then click and drag to draw your selection circle:





And notice the [\\$geoWithin](#) filter that was added to our query:



Finally, click **ANALYZE** again and click the **Documents** tab to view the Chinese restaurants in our selected radius in Staten Island:

Documents
Aggregations
Schema

FILTER
{borough: 'Staten Island',cuisine: 'Chinese','addr

INSERT DOCUMENT
VIEW
LIST
TABLE

```

_id: ObjectId("5beb3a21af37deb50165abb9")
> address: Object
  borough: "Staten Island"
  cuisine: "Chinese"
> grades: Array
  name: "Kim'S Island Chinese Restaurant"
  restaurant_id: "41436344"

```

```

_id: ObjectId("5beb3a25af37deb50165caaf")
> address: Object
  borough: "Staten Island"
  cuisine: "Chinese"
> grades: Array
  name: "Loon Chuan Restaurant"
  restaurant_id: "41717895"

```

```

_id: ObjectId("5beb3a27af37deb50165e0cb")
> address: Object
  borough: "Staten Island"
  cuisine: "Chinese"
> grades: Array
  name: "Chef Hong"
  restaurant_id: "50015617"

```

## Embedded Documents

Querying on embedded documents is simply achieved by using dot notation. For example, to find all documents on Park Ave, your filter would be:


```
{'address.street': 'Park Ave'}
```

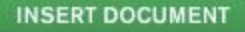


*Note, when using dot notation, your query field, 'address.street', must be in quotes (single or double).*

Switch to the **Documents** tab and click  to the right of the filter field. Then enter the query filter above to find all restaurants on Park Ave (don't forget to click **FIND**):

demo.restaurants

Documents Aggregations Schema

 FILTER `{'address.street': 'Park Ave'}`

 VIEW  

```
> {
  "_id": ObjectId("5a739601c23e89010201f4d8"),
  "address": {
    "building": "1259",
    "coord": {
      "street": "Park Ave",
      "zipcode": "10029"
    },
    "borough": "Manhattan",
    "cuisine": "Armenian"
  },
  "grades": {
    "name": "Earl'S Beer & Cheese",
    "restaurant_id": "50001622"
  }
}
```

```
> {
  "_id": ObjectId("5a739601c23e89010201fb34"),
  "address": {
    "building": "100",
    "coord": {
      "street": "Park Ave",
      "zipcode": "10017"
    },
    "borough": "Manhattan",
    "cuisine": "Sandwiches/Salads/Mixed Buffet"
  },
  "grades": {
    "name": "Potbelly Sandwich Shop",
    "restaurant_id": "50005560"
  }
}
```

## Querying Arrays

Understanding how to work with arrays of embedded documents is critical to understanding how to work with MongoDB, as they are a core component behind the power of the document model.

In our sample dataset, we have an array of grades. To find all the restaurants that have ever earned a C grade is as simple as:



```
{ 'grades.grade': 'C' }
```

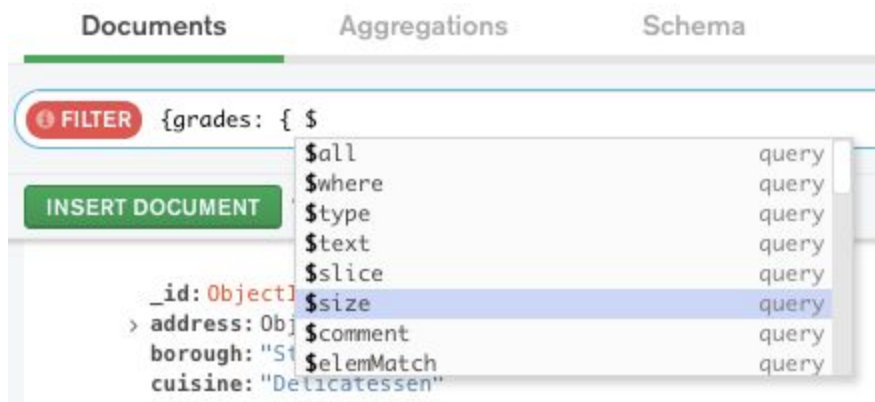
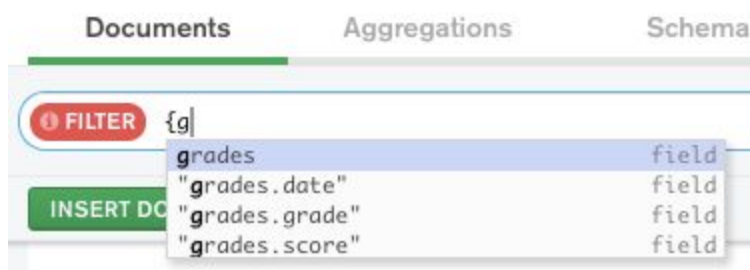
To find all the restaurants that have earned both a B and C rating, we can include the [\\$all](#) operator, which returns only the documents containing all the elements you want to match:

```
{ 'grades.grade': { $all : ['B', 'C'] } }
```

To find all the restaurants that have never earned an A or B grade, we can use the [\\$nin](#) operator, which returns only the documents where the values are not in the specified array (note, include 'Not Yet Graded' in the filter to exclude those restaurants as well):

```
{ 'grades.grade': { $nin: ['A', 'B', 'Not Yet Graded'] } }
```

To find all the restaurants that have 5 grades, we'll use the [\\$size](#) operator. As you type this query, notice the autocomplete support provided by Compass:

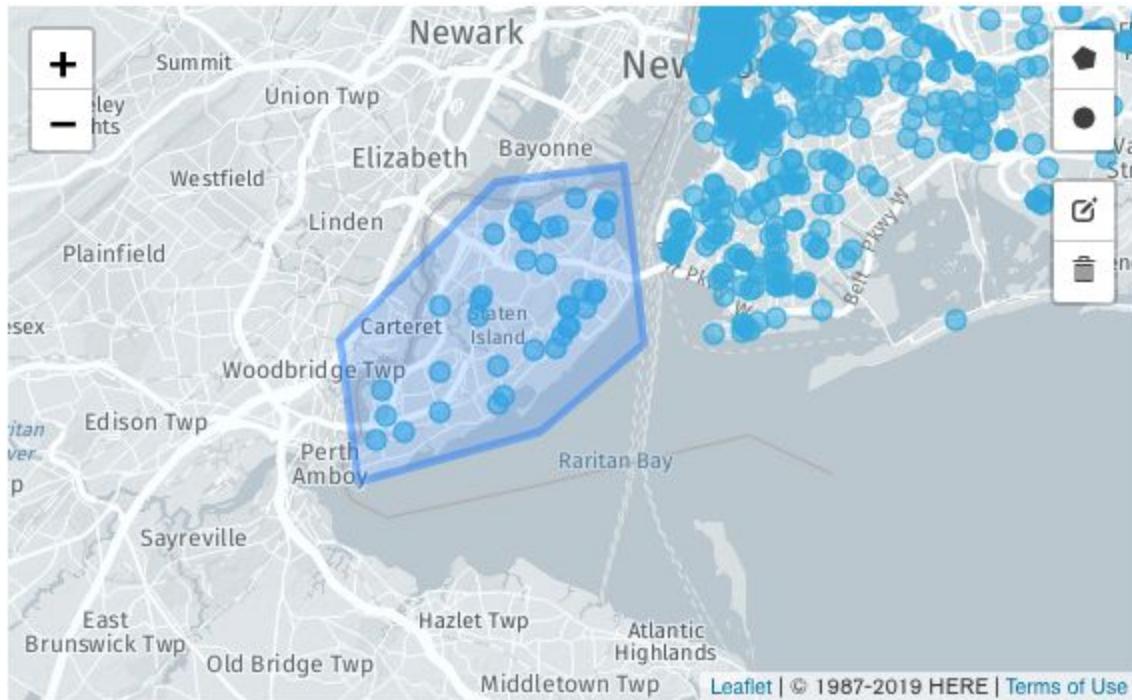


```
{grades: { $size: 5 } }
```

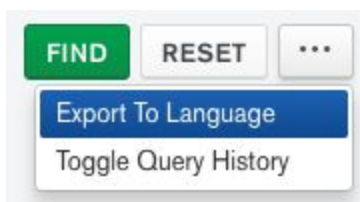
## Exercise 6 - Export to Language

Compass is a great tool for helping you learn the MongoDB Query Language (MQL). At some point, however, you want to call the query from your application code, and Compass supports this as well via the Export to Language feature.

As mentioned previously, using the Schema tab is a great way to visually build your queries. Return to the Schema tab and click the **RESET** and **ANALYZE** buttons. Then expand the address field to get back to the map view. Zoom in on NY again and click the Draw a polygon to draw a polygon of your choice. For example:



The filter field is populated with the associated MongoDB query. Click the ellipses to the right of the FIND button and select **Export to Language**:



You'll find your query alongside how you would use it in one of 4 associated languages: Java, Node, C# and Python 3:

Export Query To Language

My Query:

Export Query To: PYTHON 3

```

1 {
2   'address.coord': {
3     $geoWithin: {
4       $geometry: {
5         type: 'Polygon',
6         coordinates: [
7           [
8             [-74.06618177890779, 40.6
9             [-74.05211091041566, 40.5
10            [-74.1239243745804, 40.52
11            [-74.24874365329744, 40.4
12            [-74.26015377044679, 40.5
13            [-74.15383636951448, 40.6
14            [-74.06618177890779, 40.6
15          ]
16        ]
17      }
18    }
19  }
20 }

```

```

1 {
2   'address.coord': {
3     $geoWithin: {
4       $geometry: {
5         type: 'Polygon',
6         coordinates: [
7           [
8             [-74.06618177890779, 4
9             [-74.05211091041566, 4
10            [-74.1239243745804, 40
11            [-74.24874365329744, 4
12            [-74.26015377044679, 4
13            [-74.15383636951448, 4
14            [-74.06618177890779, 4
15          ]
16        ]
17      }
18    }
19  }
20 }

```

☐ Include Import Statements

CLOSE

## Exercise 7 - Controlling Query Output

### Projection

Thus far, every query has returned the full document, which usually isn't needed or desired, especially when working with larger documents. The MongoDB query language allows you to [project fields to return from a query](#). For example, say we only want the names of the

restaurants on Broadway. In the Documents tab, click the **OPTIONS** button at the end of the FILTER to expand the query options and add the following to FILTER and PROJECT and click FIND:

Documents

Aggregations

Schema

FILTER

{'address.street': 'Broadway'}

PROJECT

{name:1}

SORT

COLLATION

VIEW

LIST

TABLE

\_id: ObjectId("5ce45e6aeb1e230856ca13bf")

name: "Bully'S Deli"

\_id: ObjectId("5ce45e6aeb1e230856ca13e3")

name: "Peter Luger Steakhouse"

\_id: ObjectId("5ce45e6beb1e230856ca14b2")

name: "Dan Tempura"

To exclude fields, set the field value for the projection to 0. For example, the `_id` field is always included, and it can be excluded with:

Documents Aggregations Sc

**FILTER** `{'address.street': 'Broadway'}`

**PROJECT** `{name:1, _id:0}`

**SORT**

**COLLATION**

VIEW **LIST** **TABLE**

`name: "Bully'S Deli"`

`name: "Peter Luger Steakhouse"`

`name: "Dan Tempura"`

## Sorting

The results can be sorted on one or more fields in ascending or descending order. Simply specify the field to sort on and the order using 1 for ascending for -1 for descending:

Documents
Aggregations
Schemas

FILTER
{'address.street': 'Broadway'}

PROJECT
{name:1, \_id:0}

SORT
{name:1}

COLLATION

VIEW
LIST
TABLE

name: ""

name: ""

name: ""

Note, we can use the following to filter out those restaurants with empty names:

```
{'address.street': 'Broadway', name : {$ne: ""} }
```

## Skip and Limit

Finally, it's very common to want to limit the amount of documents returned. There are 922 restaurants on Broadway, but assume we only want to display two at a time. To display the first two, set LIMIT to 2:

The screenshot shows the MongoDB Compass interface with the 'Documents' tab selected. The query builder shows a FILTER stage with the query `{'address.street': 'Broadway', name : {'$ne': ''} }`, a PROJECT stage with `{name:1, _id:0}`, and a SORT stage with `{name:1}`. The COLLATION stage is empty. The SKIP value is 0 and the LIMIT value is 2. The 'VIEW' section shows 'LIST' and 'TABLE' buttons, with 'LIST' selected. The text 'Displaying documents' is visible. The results pane shows two documents:

```
name: "'U' Like Chinese Restaurant"
```

```
name: "/ L'Ecole"
```

To display the next two, set the SKIP to 2:

The screenshot shows the MongoDB Compass interface with the 'Documents' tab selected. The query builder shows the same FILTER, PROJECT, and SORT stages as the previous image. The COLLATION stage is empty. The SKIP value is now 2 and the LIMIT value is 2. The 'VIEW' section shows 'LIST' and 'TABLE' buttons, with 'LIST' selected. The text 'Displaying documents' is visible. The results pane shows two documents:

```
name: "107 West Restaurant"
```

```
name: "137 Bar & Grill"
```

Then just keep incrementing the SKIP by 2:

The screenshot shows the MongoDB Compass interface with the Documents tab selected. The query bar contains a filter: `{'address.street': 'Broadway', name: {'$ne': ''}}`. Below the query bar, there are sections for PROJECT, SORT, and COLLATION. The SKIP value is set to 4 and the LIMIT is set to 2. The view is set to LIST. The results show two documents: `{'name': '16 Handles'}` and `{'name': '21 Bar'}`.

You can envision how easily you can use skip and limit to support pagination in your application.

## Exercise 8 - Query Performance

We've been querying a relatively small dataset of 25,000 records, so performance hasn't really been an issue. But as your data sets grow, it's imperative you know how to run your queries most efficiently. Not surprisingly, one of the easiest things you can do to improve query performance is to define an index.

Click the **RESET** button. Switch to the **Explain Plan** tab of the Compass and enter the following filter:

```
{name: 'Starbucks Coffee'}
```

Then click **EXPLAIN**:



The screenshot shows the MongoDB Compass interface with the 'Explain Plan' tab selected. The query filter is `{name: 'Starbucks Coffee'}`. The 'Query Performance Summary' section displays the following metrics:

Metric	Value
Documents Returned	223
Index Keys Examined	0
Documents Examined	25370
Actual Query Execution Time (ms)	13
Sorted in Memory	no
No index available for this query.	

A callout box for the 'COLLSCAN' operation provides more details:

- nReturned: 223
- Execution Time: 2 ms
- Documents Examined: 25370

A 'DETAILS' button is located at the bottom of the callout box.

The query executed in 2ms, which is lightning fast. However, it did have to scan the entire collection of 25,370 documents to find the 223 Starbucks Coffees in NYC, which is not ideal.

## Indexes

Switch to the Indexes tab of Compass and you'll see that the collection has an existing index on the `_id` field. This unique index is created by default on every collection and cannot be deleted (likewise, every collection must have an `_id` field).

To improve the performance of our index above, click [CREATE INDEX](#). Then Configure the index definition by selecting **name** and **1 (asc)**. You can leave the index name field blank, as one will be generated for you.

## Create Index

Choose an index name

Configure the index definition

name

1 (asc)

—

ADD ANOTHER FIELD

> Options

CANCEL

CREATE INDEX

Click **Create Index**:

Documents	Aggregations	Schema	Explain Plan	Indexes	Validation
CREATE INDEX					
Name and Definition ^	Type	Size	Usage	Properties	Drop
<div>_id</div> <div>_id</div>	REGULAR ⓘ	266.2 KB	2 since Wed Jul 31 2019	UNIQUE ⓘ	
<div>name_1</div> <div>name</div>	REGULAR ⓘ	491.5 KB	1 since Fri Aug 09 2019		

Now return to the **Explain Plan** tab and **EXPLAIN** the query again. You'll notice only 223 documents were examined and returned:

Documents

Aggregations

Schema

Explain Plan

Indexes

FILTER {name: 'Starbucks Coffee'}

PROJECT

SORT

COLLATION

SKIP 0

VIEW DETAILS AS

VISUAL TREE

RAW JSON

## Query Performance Summary

Documents Returned: 223

Index Keys Examined: 223

Documents Examined: 223

Actual Query Execution Time (ms): 0

Sorted in Memory: no

Query used the following index:

name ↑

## FETCH

nReturned: 223

Execution Time:

0  
ms

DETAILS

## IXSCAN

nReturned: 223

Execution Time:

0  
ms

Index Name: name\_1

Multi Key Index: no

DETAILS

## Congratulations!

You've completed this lab on using MongoDB Compass! You should now feel comfortable installing Compass, importing data sets, exploring and querying the data using the MongoDB Query Language, examining query performance, and improving query performance through the creation of indexes.