Lab 13: Query Modifiers and Indexing

The learning objectives of this lab are to:

- **Query Modifiers**
- **Indexing**
- **Mongo Compass**

1. MongoDB order with Sort() & Limit() Query with Examples

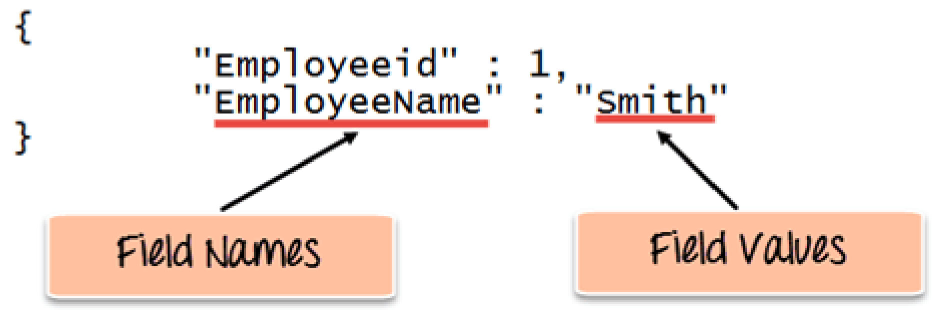
What is Query Modifications?

Mongo DB provides query modifiers such as the 'limit' and 'Orders' clause to provide more flexibility when executing queries. We will take a look at the following query modifiers.

MongoDB Limit Query Results

This modifier is used to limit the number of documents which are returned in the result set for a query. The following example shows how this can be done.

Assume you have a collection named employee having a document layout as shown below.



Use the insert command to insert the array of documents into the collection.

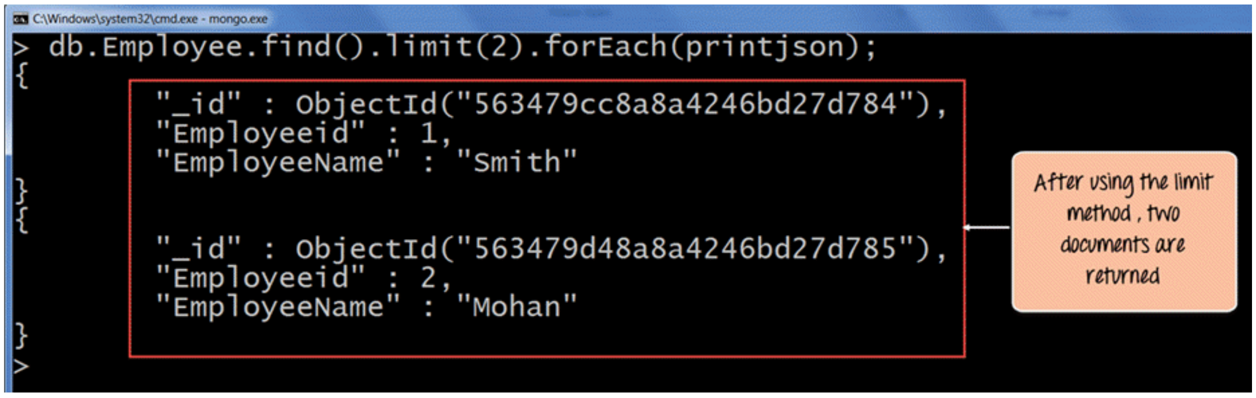
```
        db.Employee.insert([

                {
                        "Employeeid" : 1,
                        "EmployeeName" : "Smith"
                },
                {
                        "Employeeid"   : 2,
                        "EmployeeName" : "Mohan"
                },
                {
                        "Employeeid"   : 3,
                        "EmployeeName" : "Joe"
                },

        ]
);
```

```
db.Employee.find().limit(2).forEach(printjson);
```

Code Explanation:

1. The above code takes the find function which returns all of the documents in the collection but then uses the limit clause to limit the number of documents being returned to just 2.
2. The first change is to append the function called for Each() to the find() function. What this does is that it makes sure to explicitly go through each document in the collection. In this way, you have more control of what you can do with each of the documents in the collection.
3. The second change is to put the printjson command to the forEach statement. This will cause each document in the collection to be displayed in JSON format.
   Printing in JSON format
   JSON is a format called JavaScript Object Notation, and is just a way to store information in an organized, easy-to-read manner. In our further examples, we are going to use the JSON print functionality to see the output in a better format.

If the command is executed successfully, the following Output will be shown.


C:\Windows\system32\cmd.exe - mongo.exe
After using the limit method , two documents are returned

```
> db.Employee.find().limit(2).forEach(printjson);
{
        "_id" : ObjectId("563479cc8a8a4246bd27d784"),
        "Employeeid" : 1,
        "EmployeeName" : "Smith"
}
{
        "_id" : ObjectId("563479d48a8a4246bd27d785"),
        "Employeeid" : 2,
        "EmployeeName" : "Mohan"
}
>
```

The output clearly shows that since there is a limit modifier, so at most just 2 records are returned as part of the result set based on the ObjectId in ascending order.

MongoDB Sort by Descending Order

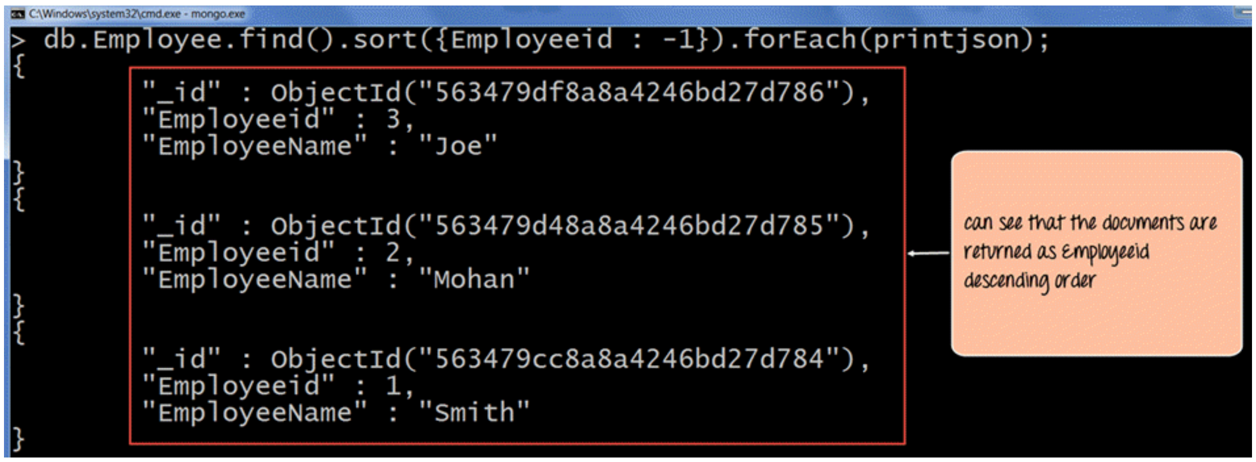
One can specify the order of documents to be returned based on ascending or descending order of any key in the collection. The following example shows how this can be done.

```
db.Employee.find().sort({Employeeid:-1}).forEach(printjson)
```

Code Explanation:

1. The above code takes the sort function which returns all of the documents in the collection but then uses the modifier to change the order in which the records are returned. Here the -1 indicates that we want to return the documents based on the descending order of Employee id.

If the command is executed successfully, the following Output will be shown



```
> db.Employee.find().sort({Employeeid : -1}).forEach(printjson);
{
        "_id" : ObjectId("563479df8a8a4246bd27d786"),
        "Employeeid" : 3,
        "EmployeeName" : "Joe"
}
{
        "_id" : ObjectId("563479d48a8a4246bd27d785"),
        "Employeeid" : 2,
        "EmployeeName" : "Mohan"
}
{
        "_id" : ObjectId("563479cc8a8a4246bd27d784"),
        "Employeeid" : 1,
        "EmployeeName" : "Smith"
}
```

C:\Windows\system32\cmd.exe - mongo.exe
can see that the documents are returned as employeeid descending order

The output clearly shows the documents being returned in descending order of the Employeeid.

Ascending order is defined by value 1.

MongoDB Count() & Remove() Functions with Examples

The concept of aggregation is to carry out a computation on the results which are returned in a query. For example, suppose you wanted to know what is the count of documents in a collection as per the query fired, then MongoDB provides the count() function.

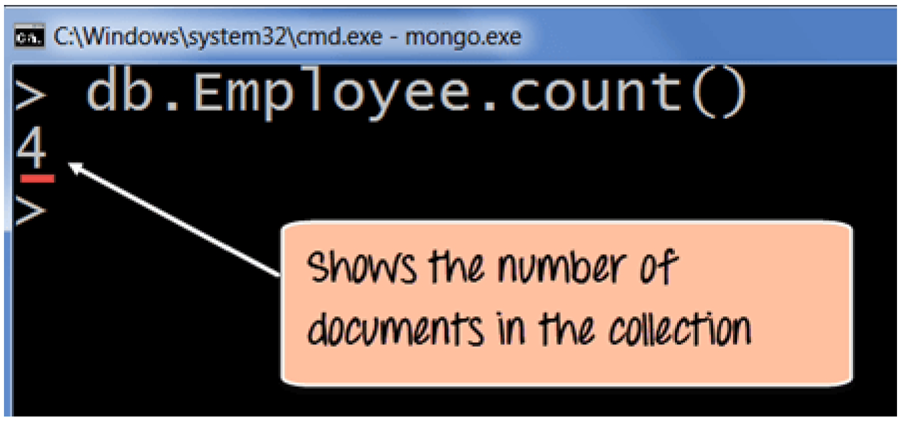
Let's look at an example of this.

```
db.Employee.count()
```

Code Explanation:

1. The above code executes the count function.

If the command is executed successfully, the following Output will be shown


C:\Windows\system32\cmd.exe - mongo.exe
Shows the number of documents in the collection

```
> db.Employee.count()
4
>
```

The output clearly shows that 4 documents are there in the collection.

Performing Modifications

The other two classes of operations in MongoDB are the update and remove statements.

The update operations allow one to modify existing data, and the remove operations allow the deletion of data from a collection.

Deleting Documents

In MongoDB, the **db.collection.remove ()** method is used to remove documents from a collection. Either all of the documents can be removed from a collection or only those which matches a specific condition.

If you just issue the remove command, all of the documents will be removed from the collection.

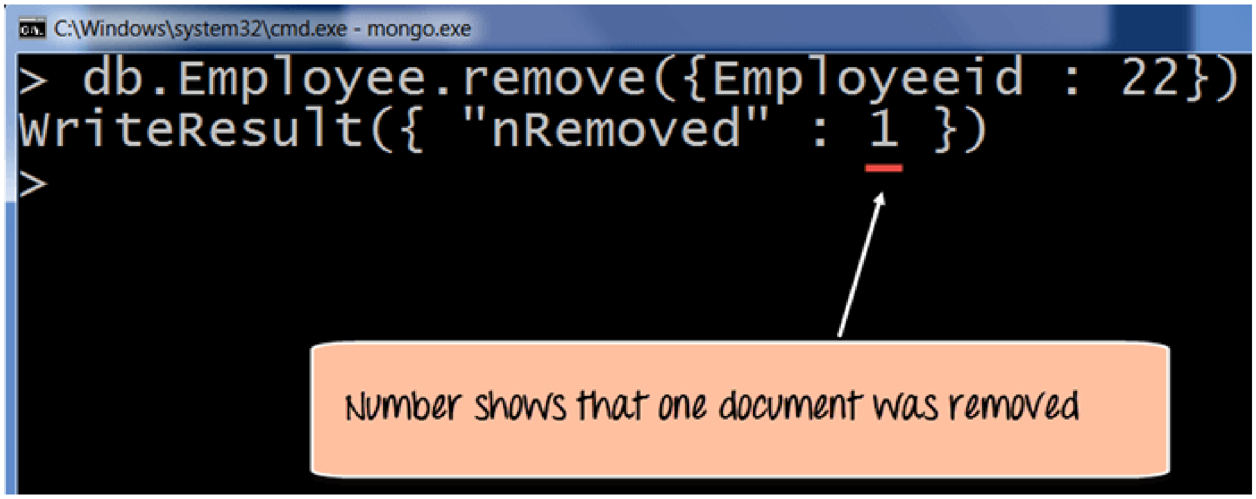
The following code example demonstrate how to remove a specific document from the collection.

```
db.Employee.remove({Employeeid:2})
```

Code Explanation:

1. The above code use the remove function and specifies the criteria which in this case is to remove the documents which have the Employee id as 2.

If the command is executed successfully, the following Output will be shown


C:\Windows\system32\cmd.exe - mongo.exe
Number shows that one document was removed

```
> db.Employee.remove({Employeeid : 22})
WriteResult({ "nRemoved" : 1 })
>
```

The output will show that 1 document was modified.

MongoDB Indexing and Performance Considerations

1. Indexes are very important in any database and can be used to improve the efficiency of search queries in MongoDB. If you are continually performing searches in your document, then it's better to add indexes on the fields of the document that are used in the search criteria.
2. Try to always limit the number of query results returned. Suppose you have 8 field names in a document, but you just want to see 2 fields from the document. Then make sure your query only targets to display the 2 fields you require and not all of the fields. Do not query for all the fields in the collection if they are not required.

MongoDB Indexing Tutorial - createIndex(), dropindex()

Indexes are very important in any database, and with MongoDB it's no different. With the use of Indexes, performing queries in MongoDB becomes more efficient.

If you had a collection with thousands of documents with no indexes, and then you query to find certain documents, then in such case MongoDB would need to scan the entire collection to find the documents. But if you had indexes, MongoDB would use these indexes to limit the number of documents that had to be searched in the collection.

Indexes are special data sets which store a partial part of the collection's data. Since the data is partial, it becomes easier to read this data. This partial set stores the value of a specific field or a set of fields ordered by the value of the field.

In this section, you will learn –

- Understanding Impact of Indexes
- How to Create Indexes: createIndex()
- How to Find Indexes: getindexes()
- How to Drop Indexes: dropindex()

Understanding Impact of Indexes

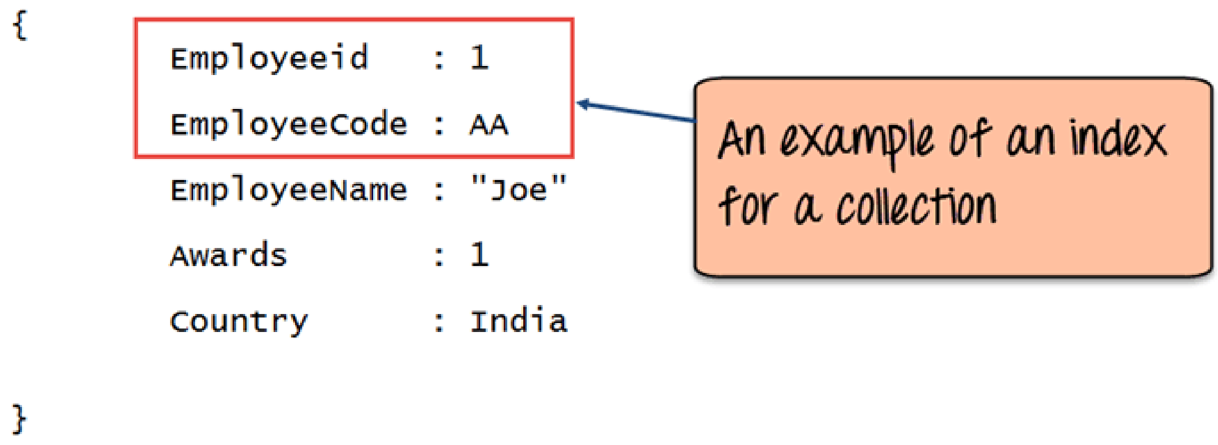
Now even though from the introduction we have seen that indexes are good for queries, but having too many indexes can slow down other operations such as the Insert, Delete and Update operation.

If there are frequent insert, delete and update operations carried out on documents, then the indexes would need to change that often, which would just be an overhead for the collection.

The below example shows an example of what field values could constitute an index in a collection. An index can either be based on just one field in the collection, or it can be based on multiple fields in the collection.

In the example below, the Employeeid "1" and EmployeeCode "AA" are used to index the documents in the collection. So when a query search is made, these indexes will be used to quickly and efficiently find the required documents in the collection.

So even if the search query is based on the EmployeeCode "AA", that document would be returned.

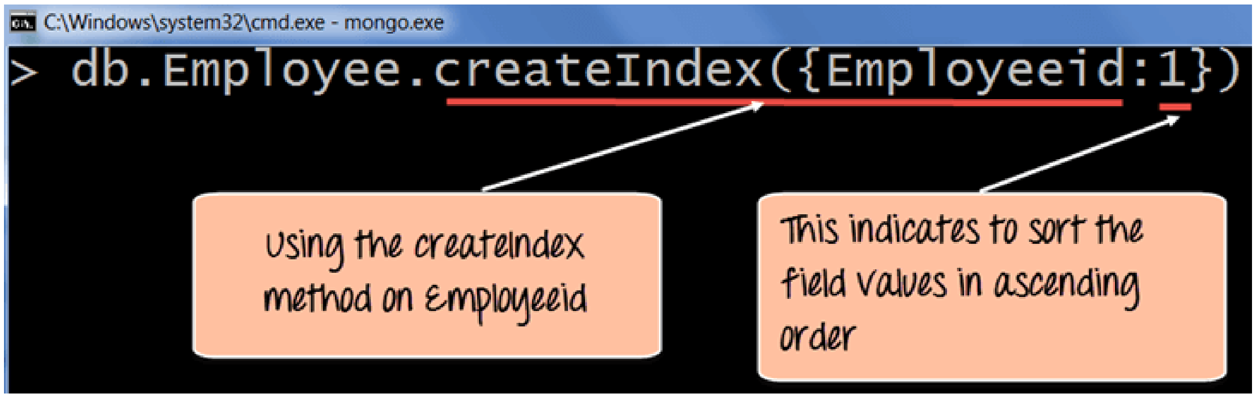


```
{
        Employeeid   : 1
        EmployeeCode : AA
        EmployeeName : "Joe"
        Awards       : 1
        Country      : India
}
```

How to Create Indexes: createIndex()

Creating an Index in MongoDB is done by using the "**createIndex**" method.

The following example shows how add index to collection. Let's assume that we have our same Employee collection which has the Field names of "Employeeid" and "EmployeeName".



```
C:\Windows\system32\cmd.exe - mongo.exe
> db.Employee.createIndex({Employeeid:1})
```

Using the createIndex method on Employeeid
This indicates to sort the field values in ascending order

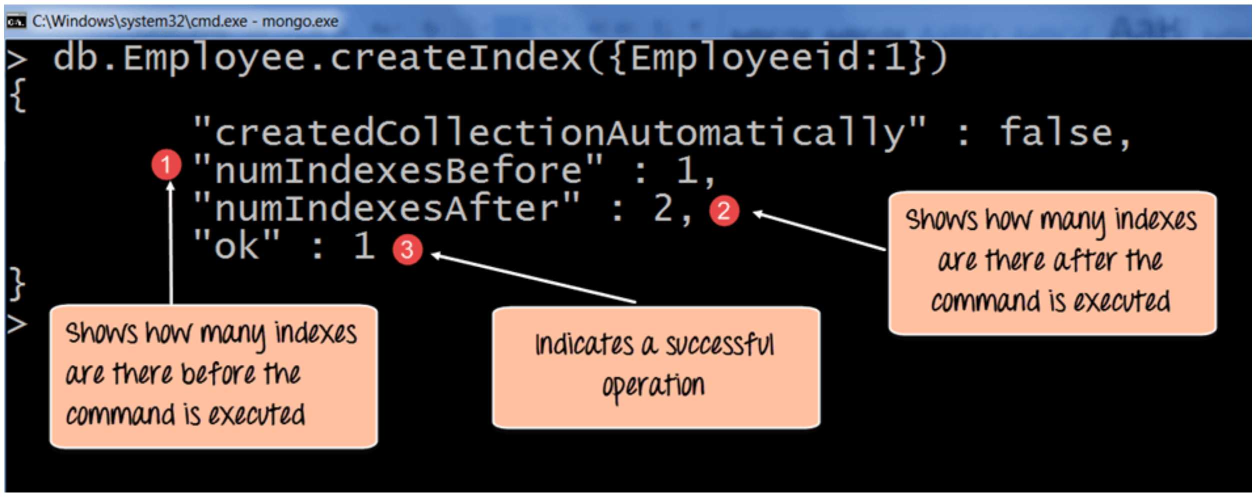
```
db.Employee.createIndex({Employeeid:1})
```

Code Explanation:

1. The **createIndex** method is used to create an index based on the "Employeeid" of the document.
2. The '1' parameter indicates that when the index is created with the "Employeeid" Field values, they should be sorted in ascending order. Please note that this is different from the _id field (The _id field is used to uniquely identify each document in the collection) which is created automatically in the collection by MongoDB.

If the command is executed successfully, the following Output will be shown:

Output:



```
C:\Windows\system32\cmd.exe - mongo.exe
> db.Employee.createIndex({Employeeid:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

1
2
3
Shows how many indexes are there after the command is executed
Shows how many indexes are there before the command is executed
Indicates a successful operation

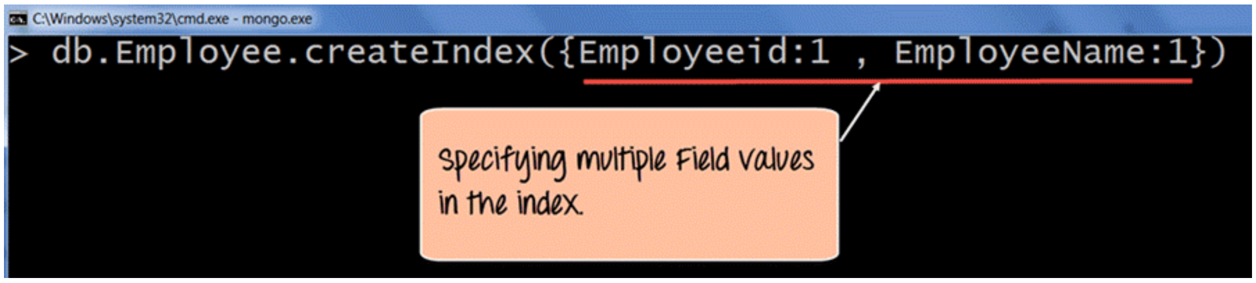
1. The numIndexesBefore: 1 indicates the number of Field values (The actual fields in the collection) which were there in the indexes before the command was run. Remember that each collection has the _id field which also counts as a Field value to the index. Since the _id index field is part of the collection when it is initially created, the value of numIndexesBefore is 1.

2. The numIndexesAfter: 2 indicates the number of Field values which were there in the indexes after the command was run.

3. Here the "ok: 1" output specifies that the operation was successful, and the new index is added to the collection.

The above code shows how to create an index based on one field value, but one can also create an index based on multiple field values.

The following example shows how this can be done;


C:\Windows\system32\cmd.exe - mongo.exe
Specifying multiple Field values in the index.

```
> db.Employee.createIndex({Employeeid:1 , EmployeeName:1})
```

```
db.Employee.createIndex({Employeeid:1, EmployeeName:1])
```

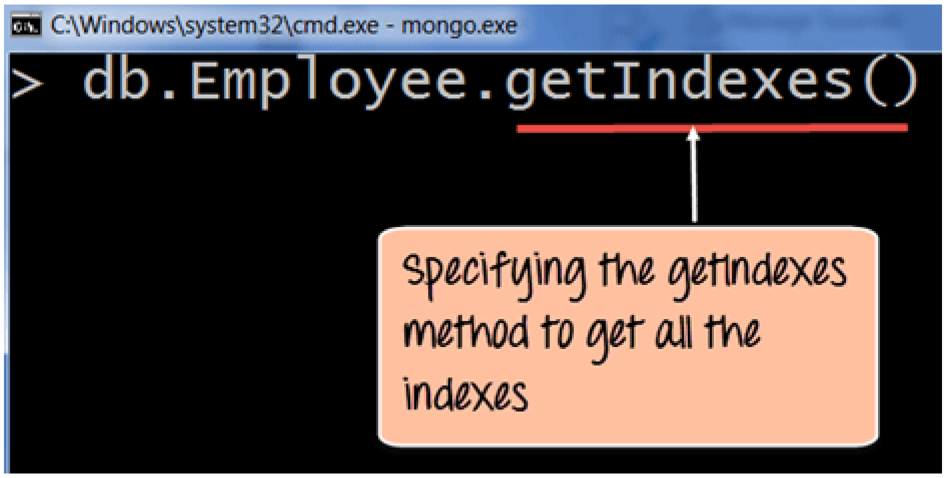
Code Explanation:

1. The createIndex method now takes into account multiple Field values which will now cause the index to be created based on the "Employeeid" and "EmployeeName". The Employeeid:1 and EmployeeName:1 indicates that the index should be created on these 2 field values with the :1 indicating that it should be in ascending order.

How to Find Indexes: getindexes()

Finding an Index in MongoDB is done by using the **"getIndexes"** method.

The following example shows how this can be done;



```
C:\Windows\system32\cmd.exe - mongo.exe
> db.Employee.getIndexes()
```

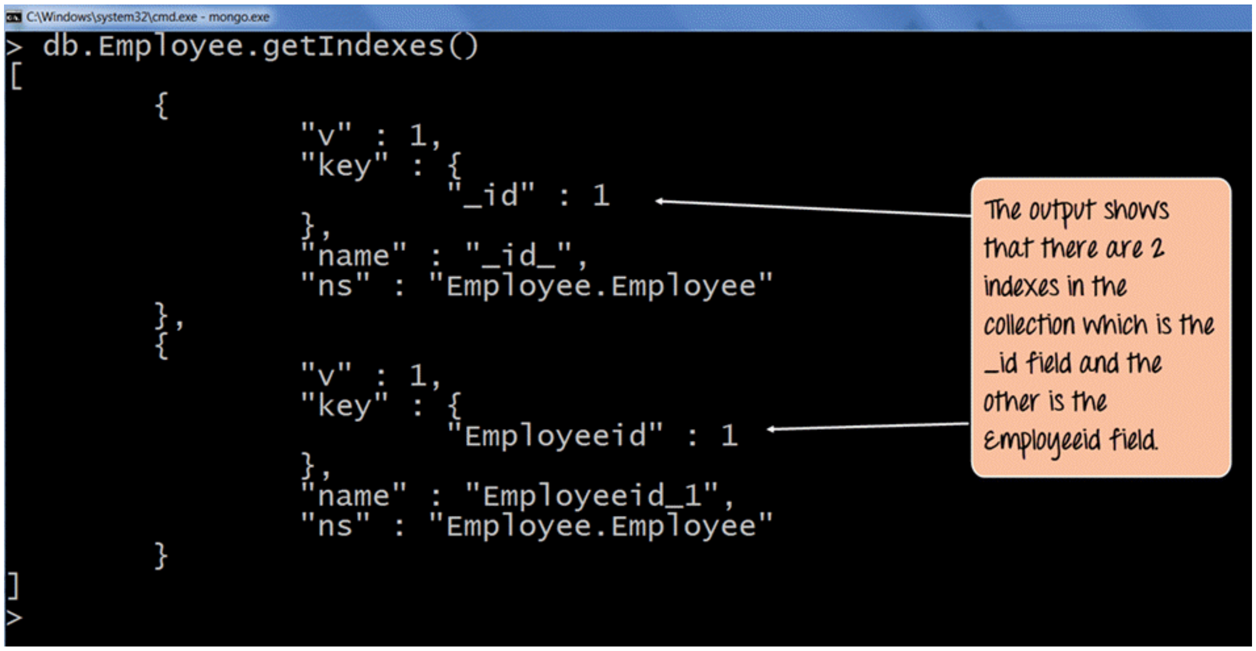
```
db.Employee.getIndexes()
```

Code Explanation:

1. The getIndexes method is used to find all of the indexes in a collection.

If the command is executed successfully, the following Output will be shown:

Output:



```
C:\Windows\system32\cmd.exe - mongo.exe
> db.Employee.getIndexes()
[
        {
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "Employee.Employee"
        },
        {
                "v" : 1,
                "key" : {
                        "Employeeid" : 1
                },
                "name" : "Employeeid_1",
                "ns" : "Employee.Employee"
        }
]
>
```

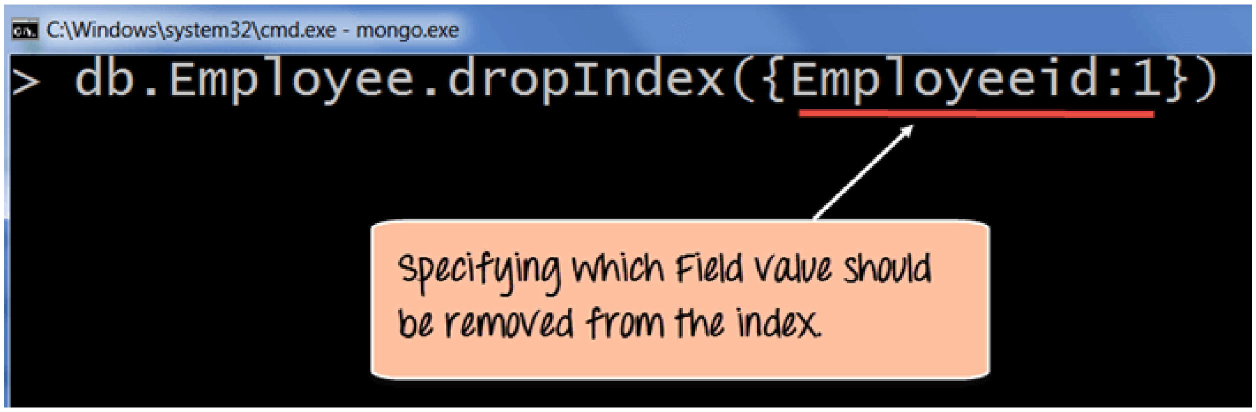
The output shows that there are 2 indexes in the collection which is the _id field and the other is the Employeeid field.

1. The output returns a document which just shows that there are 2 indexes in the collection which is the _id field, and the other is the Employee id field. The :1 indicates that the field values in the index are created in ascending order.

How to Drop Indexes: dropindex()

Removing an Index in MongoDB is done by using the dropIndex method.

The following example shows how this can be done;



```
> db.Employee.dropIndex({Employeeid:1})
```

C:\Windows\system32\cmd.exe - mongo.exe
Specifying which Field value should be removed from the index.

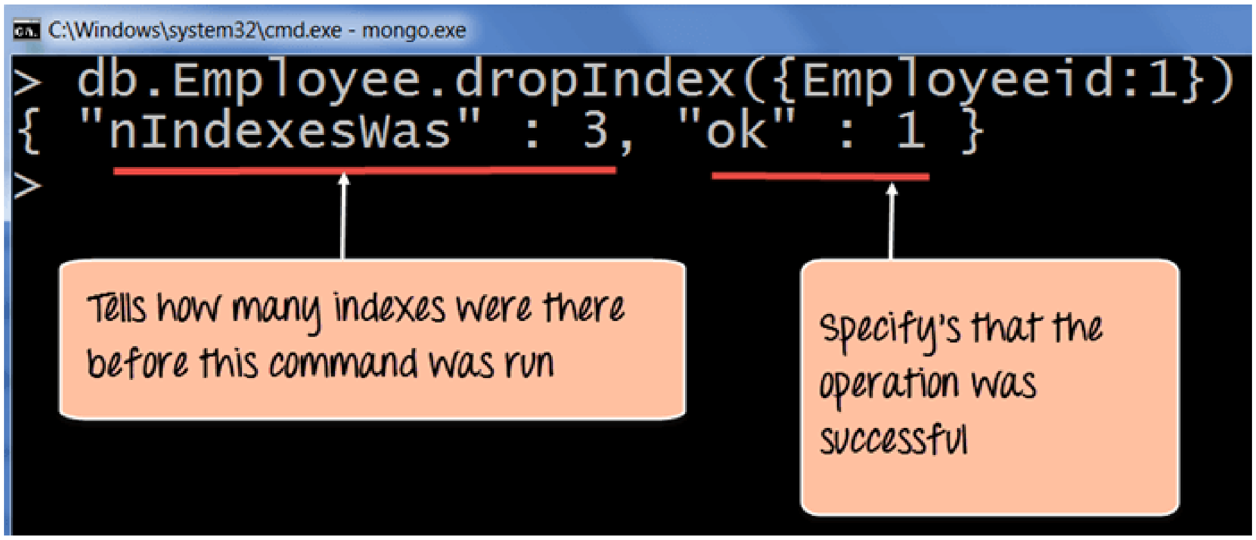
```
db.Employee.dropIndex(Employeeid:1)
```

Code Explanation:

1. The dropIndex method takes the required Field values which needs to be removed from the Index.

If the command is executed successfully, the following Output will be shown:

Output:



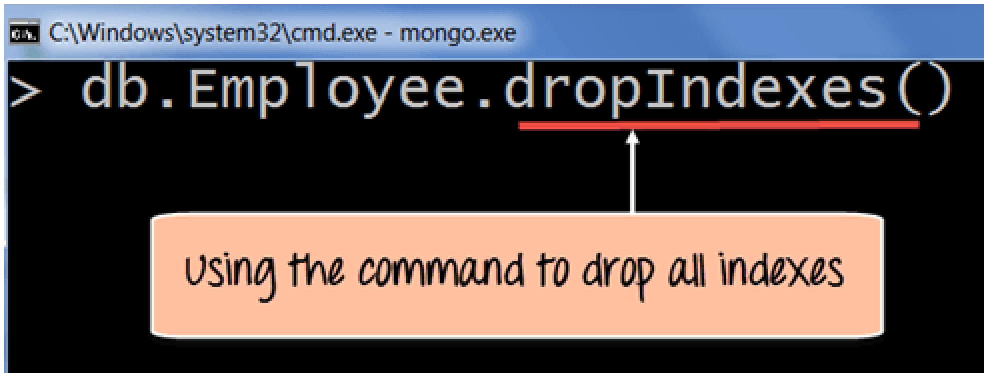
```
> db.Employee.dropIndex({Employeeid:1})
{ "nIndexesWas" : 3, "ok" : 1 }
>
```

C:\Windows\system32\cmd.exe - mongo.exe
Tells how many indexes were there before this command was run
Specify's that the operation was successful

1. The nIndexesWas: 3 indicates the number of Field values which were there in the indexes before the command was run. Remember that each collection has the _id field which also counts as a Field value to the index.
2. The ok: 1 output specifies that the operation was successful, and the "Employeeid" field is removed from the index.

To remove all of the indexes at once in the collection, one can use the dropIndexes command.

The following example shows how this can be done.


C:\Windows\system32\cmd.exe - mongo.exe
Using the command to drop all indexes

```
> db.Employee.dropIndexes()
```

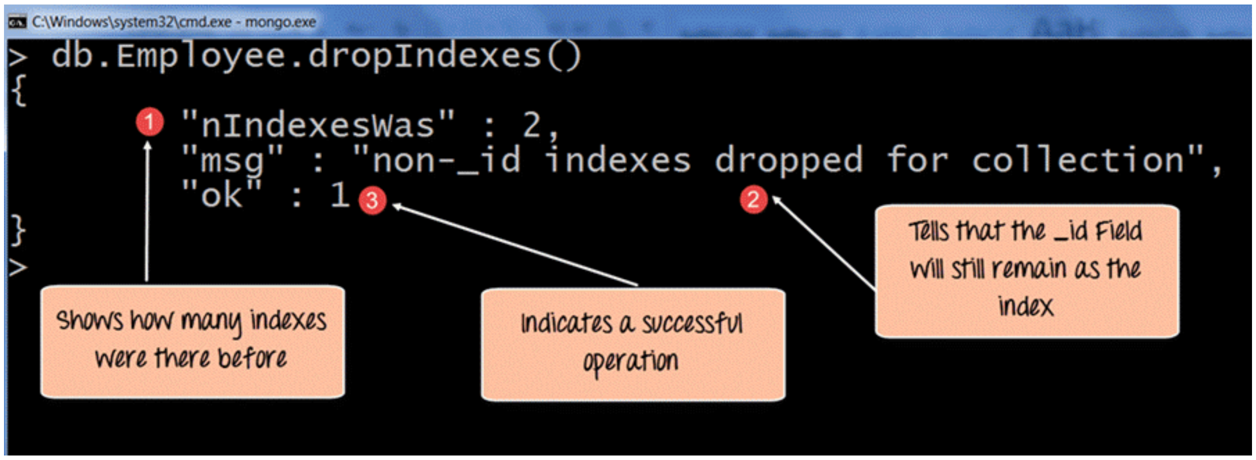
```
db.Employee.dropIndex()
```

Code Explanation:

1. The dropIndexes method will drop all of the indexes except for the _id index.

If the command is executed successfully, the following Output will be shown:

Output:


C:\Windows\system32\cmd.exe - mongo.exe
1
2
3
Shows how many indexes were there before
Indicates a successful operation
Tells that the _id Field will still remain as the index

```
> db.Employee.dropIndexes()
{
        "nIndexesWas" : 2,
        "msg" : "non-_id indexes dropped for collection",
        "ok" : 1
}
>
```

1. The nIndexesWas: 2 indicates the number of Field values which were there in the indexes before the command was run.
2. Remember again that each collection has the _id field which also counts as a Field value to the index, and that will not be removed by MongoDB and that is what this message indicates.
3. The ok: 1 output specifies that the operation was successful.

Summary

i. Defining indexes are important for faster and efficient searching of documents in a collection.
ii. Indexes can be created by using the createIndex method. Indexes can be created on just one field or multiple field values.
iii. Indexes can be found by using the getIndexes method.
iv. Indexes can be removed by using the dropIndex for single indexes or dropIndexes for dropping all indexes.

MongoDB Compass:

MongoDB Compass is the official GUI for MongoDB, maintained by MongoDB itself. MongoDB Compass helps users make clever decisions about the data structure, querying, indexing, and many more actions you can perform on the database.

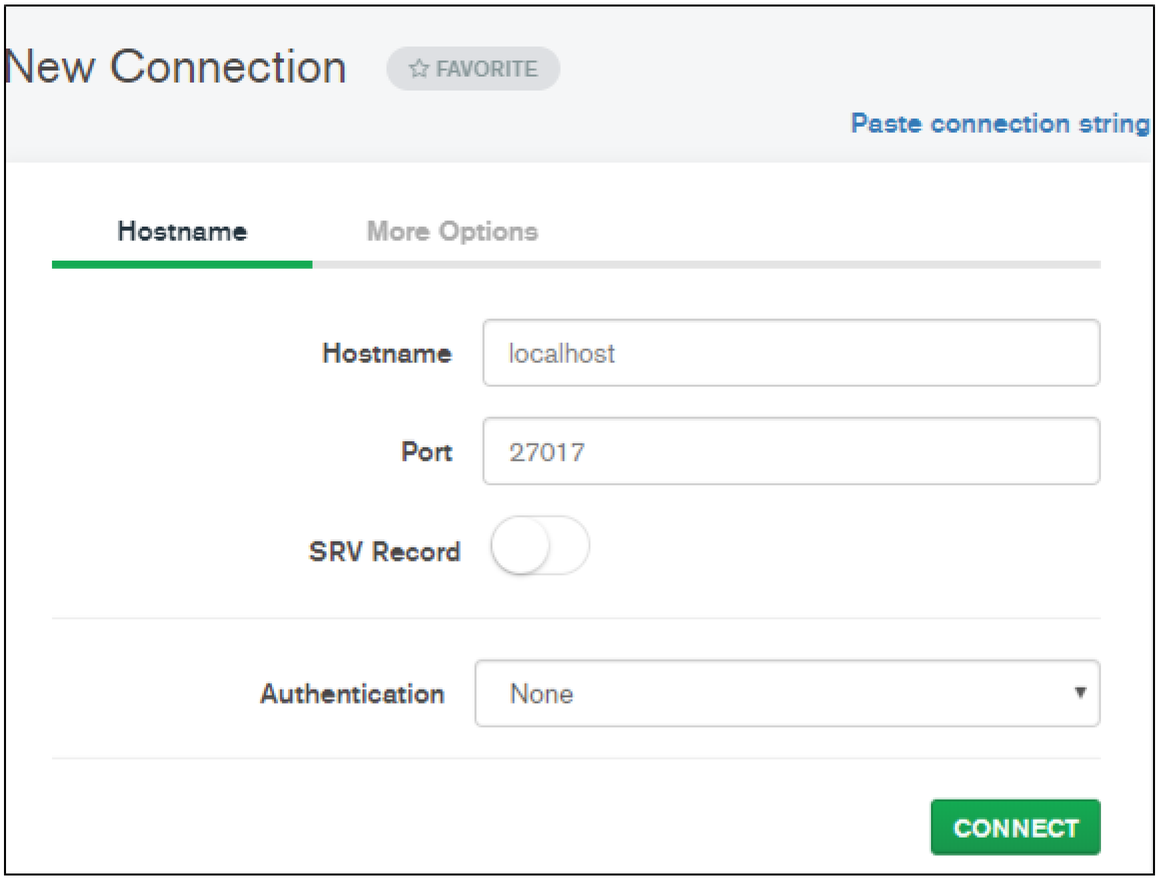
The easiest way to explore and manipulate your MongoDB data. Interact with your data with full CRUD functionality. View and optimize your query performance.

Compass can carry out all the operations that Mongo Shell does and more, including:

- Visualize and explore data stored in your database
- Create databases and Insert, update, and delete data in your database
- Get immediate real-time server statistics
- Understand performance issues with visual explain plans
- Manage your indexes
- Validate your data with JSON schema validation rules

Connecting to a database

First, open your MongoDB Compass application and click the **Fill in connection fields individually** option. Specify the Hostname and the port in which your MongoDB server is running. If you installed MongoDB on your machine with default settings, the Hostname would be the localhost, and the port is 27017. Then click **CONNECT**.



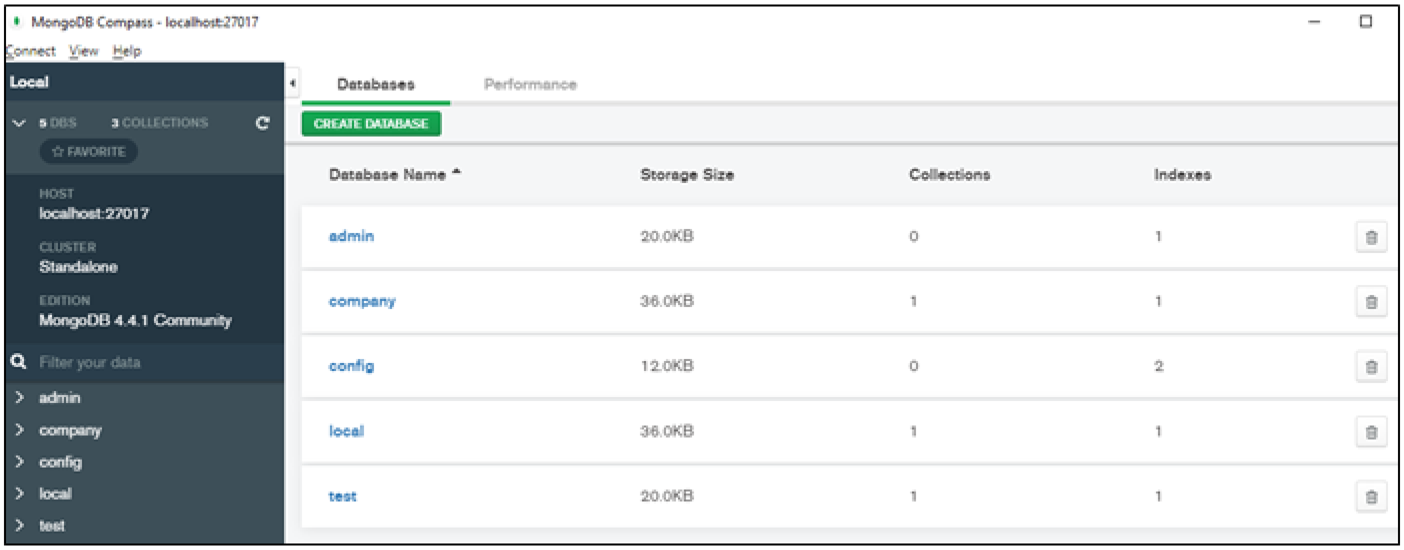
Hostname: localhost

Port: 27017

SRV Record

Authentication: None

Now, you are connected to your MongoDB server. You can see a list of databases available in the server and a set of options that you can use to create and delete databases:

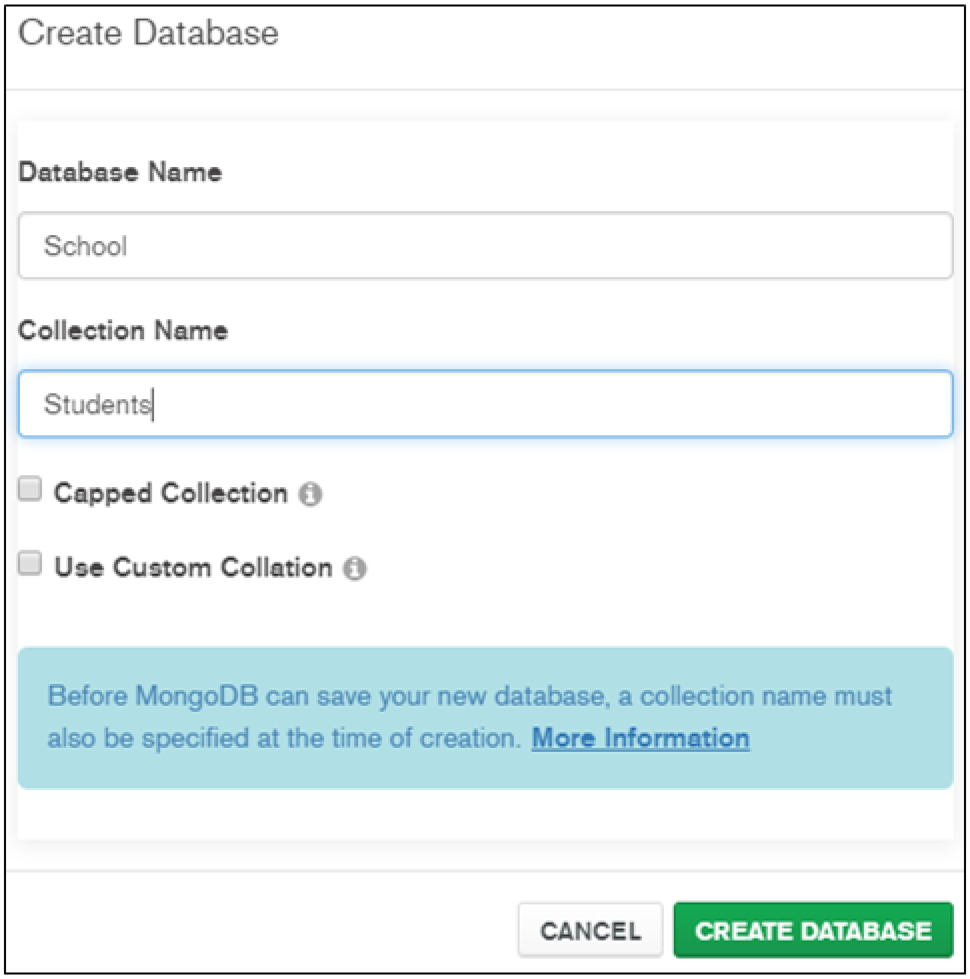


Database Name	Storage Size	Collections	Indexes
admin	20.0KB	0	1
company	36.0KB	1	1
config	12.0KB	0	2
local	36.0KB	1	1
test	20.0KB	1	1

Creating databases, collections, and inserting data

Click the **CREATE DATABASE** option to create a new database. A new window will pop up, as shown below:

- Enter the Database name (e.g., School)
- Enter a Collection name (e.g., Students).
- Click on the **CREATE DATABASE**


Create Database

Database Name

School

Collection Name

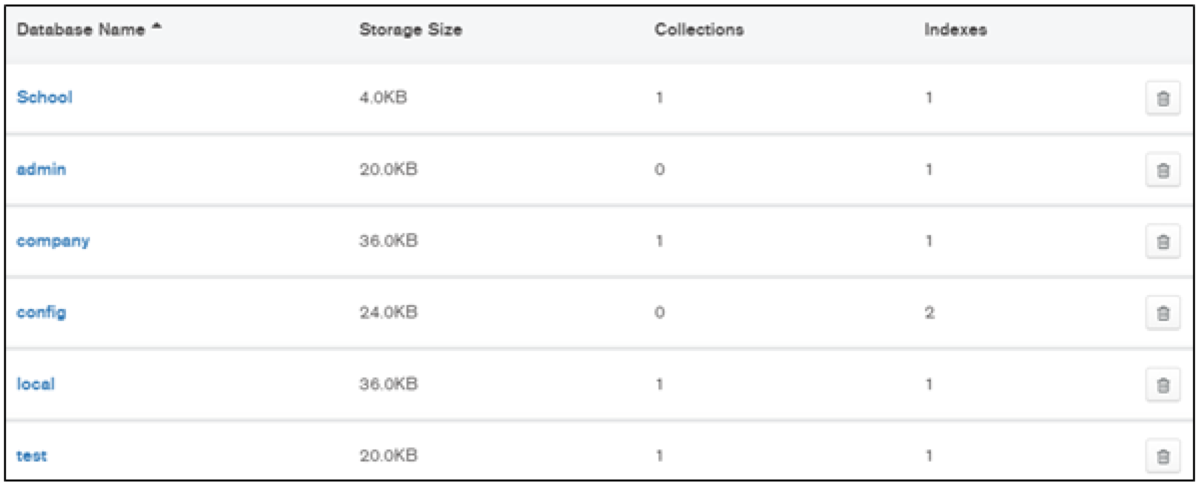
Students

☐ 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

The newly created database will appear in the dashboard, as shown below.

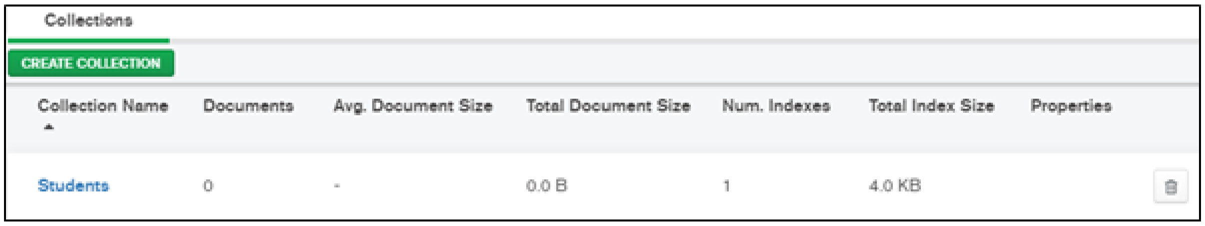


Database Name	Storage Size	Collections	Indexes
School	4.0KB	1	1
admin	20.0KB	0	1
company	36.0KB	1	1
config	24.0KB	0	2
local	36.0KB	1	1
test	20.0KB	1	1

Creating documents

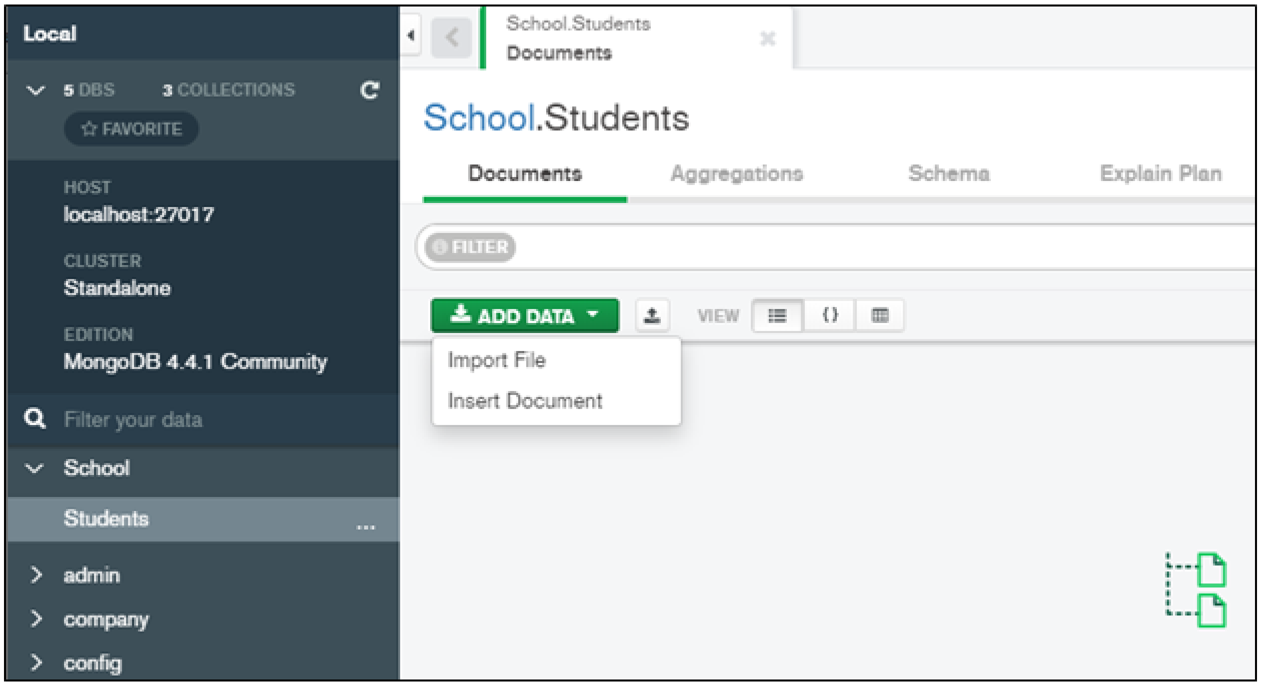
In MongoDB, data is inserted as documents. Each record in a MongoDB database is a document. Let's see how to add documents to our newly created database.

Locate the cursor on the **School** database from the left side pane of Compass and click on it. A screen will be displayed as below:



Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
Students	0	-	0.0 B	1	4.0 KB	

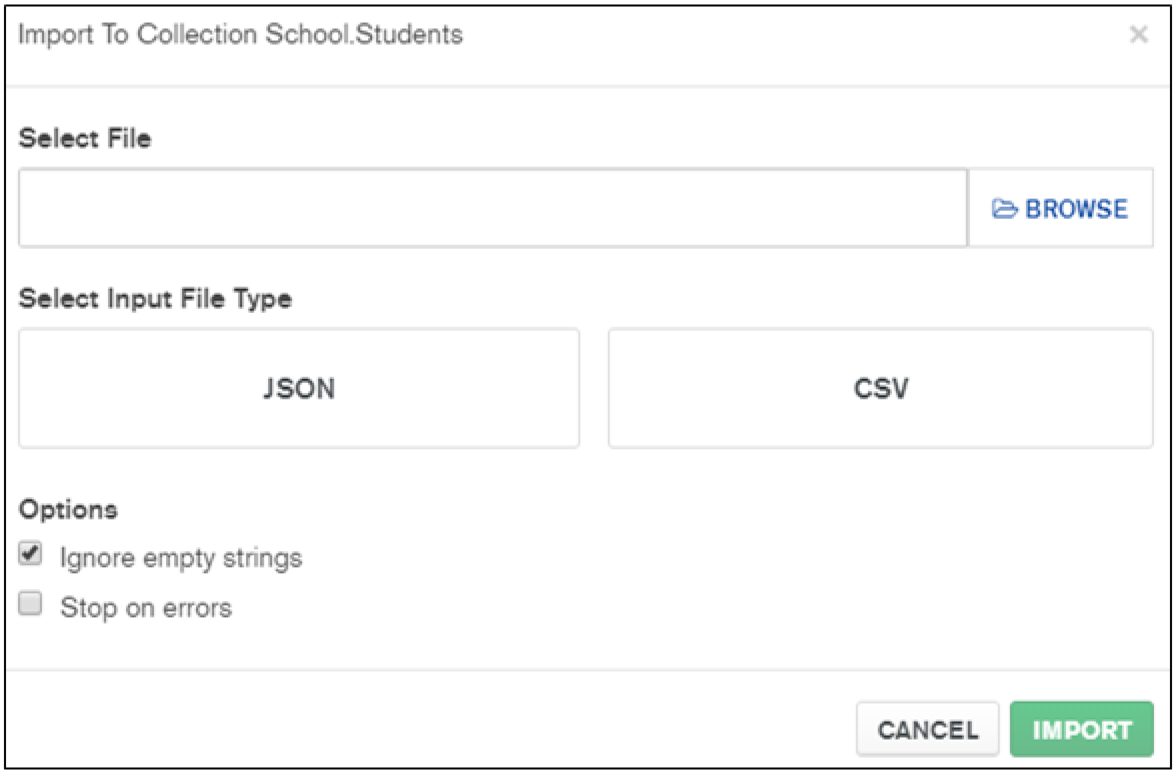
Next, click on the collection name, Students, which you created earlier, and Compass will display a new screen, like this:



Here you can find the **Add Data** drop-down that provides you two ways to insert data:

- By importing a JSON/CSV file
- By adding data manually

The first option allows you to import data as a JSON or CSV file. When you click it, a new window will open in which you can upload the file. Browse and upload the file there, then tick the relevant file type and click import.



Select File

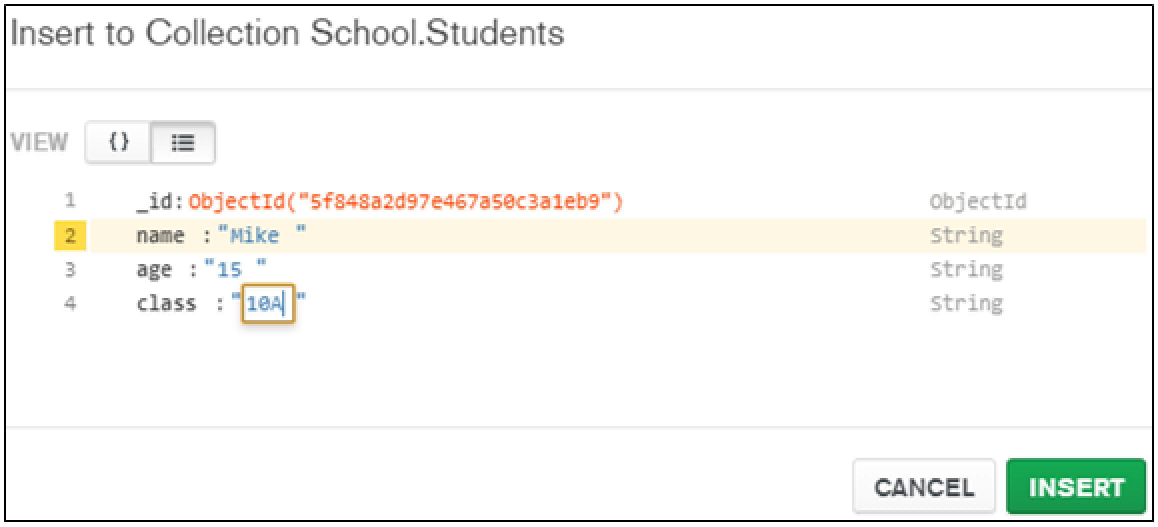
Select Input File Type

JSON CSV

Options

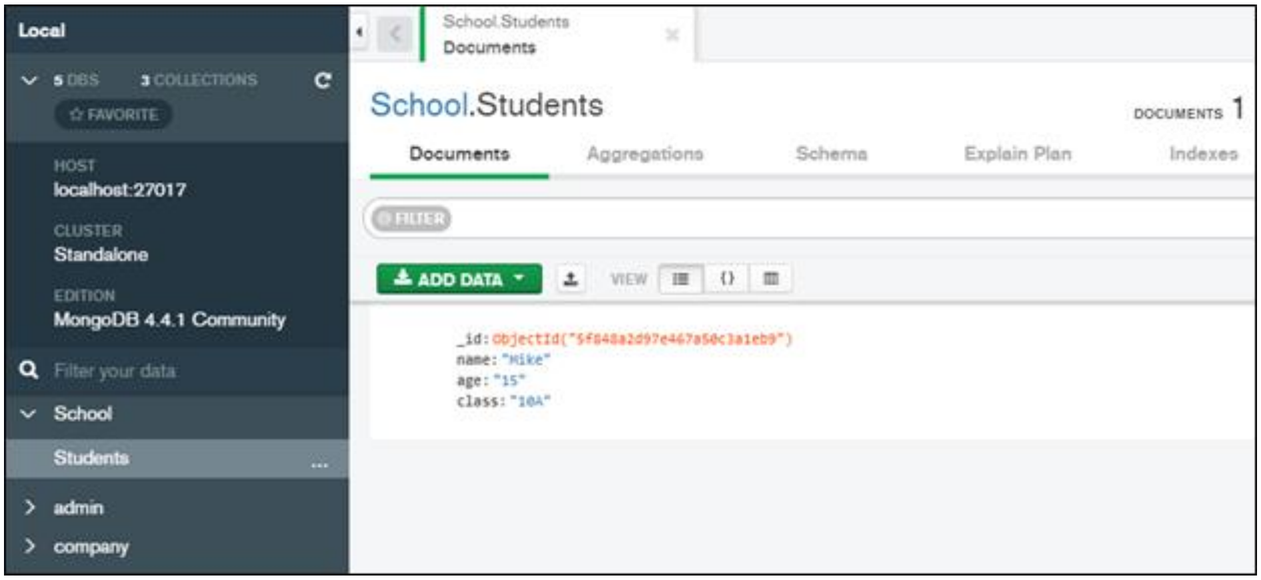
- [x] Ignore empty strings
- [ ] Stop on errors

In order to add data manually, click on the second option of the Add Data dropdown. A Helper window will pop up to insert documents. You can add values as JSON or key-value pairs in this Helper window.



```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")     ObjectId
name : "Mike "                                String
age : "15 "                                   String
class : "10A"                                 String
```

Finally, click the **Insert** button.

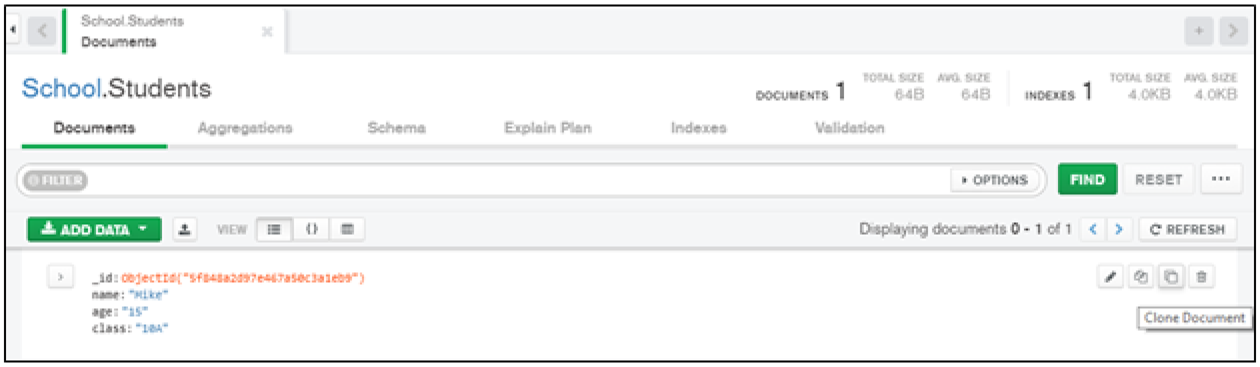


```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")
name: "Mike"
age: "15"
class: "10A"
```

Inserting more documents

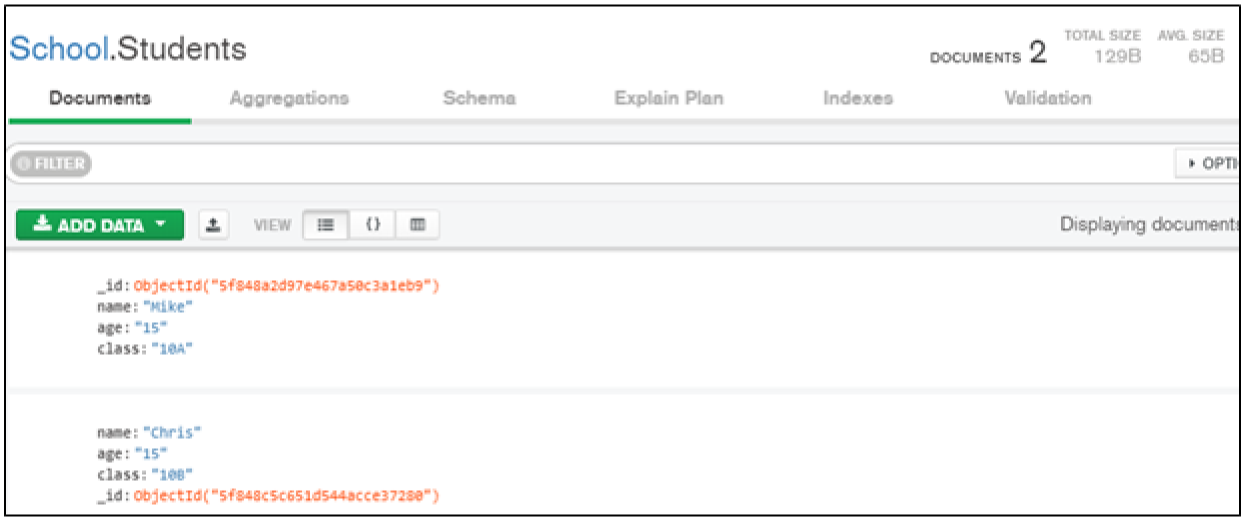
Although you can insert more documents in the same way as we just discussed, we can speed it up with the Clone operation.

Hover over the newly created data in the Compass UI and click the **Clone Document** button. This option will copy the data into a new **Insert Document** window. From there, you just have to type the next row of data, and there's no need to specify the field names or data types again.



```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")
name: "Mike"
age: "15"
class: "10A"
```

Likewise, you can keep on adding any amount of data to the collection, and those data will show up on the Compass UI.



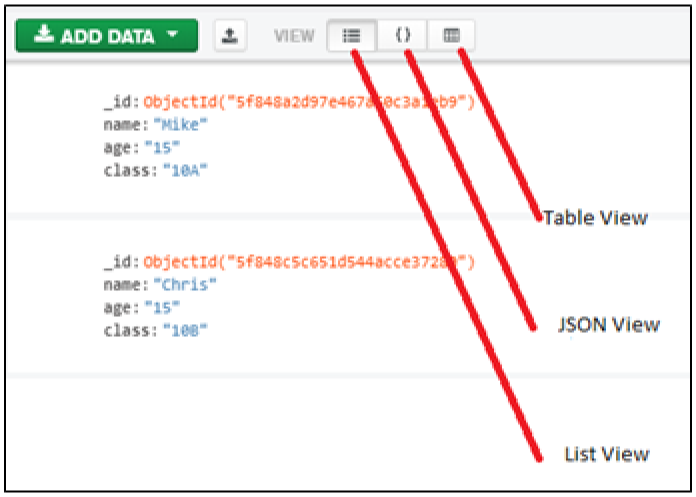
```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")
name: "Mike"
age: "15"
class: "10A"

name: "Chris"
age: "15"
class: "10B"
_id: ObjectId("5f848c5c651d544acce37280")
```

Viewing data (documents)

Compass lets you view your data in three modes. The modes are as follows:

- List view
- JSON view
- Table view

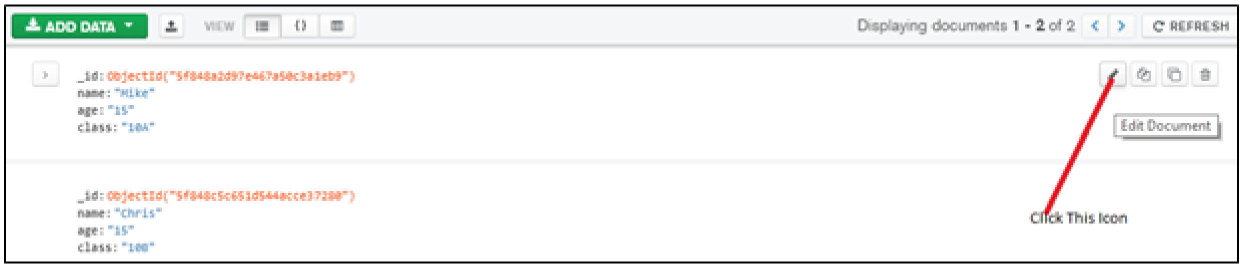
You can change the view by clicking the buttons next to the view option, as shown here:



```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")
name: "Mike"
age: "15"
class: "10A"

_id: ObjectId("5f848c5c651d544acce37280")
name: "Chris"
age: "15"
class: "10B"
```

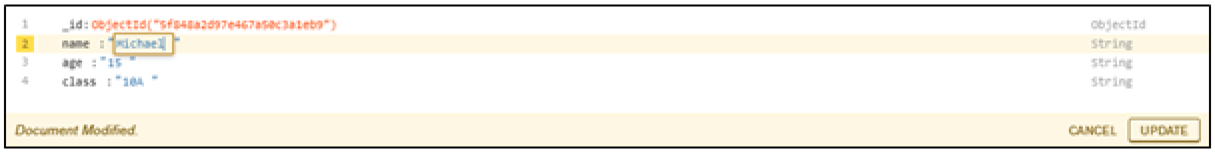
Updating documents

Updating documents through Compass is straightforward. Hover over the document you want to update in the Compass and click on the pencil icon, which appears on the right-hand side.



```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")
name: "Mike"
age: "15"
class: "10A"

_id: ObjectId("5f848c5c651d544acce37280")
name: "Chris"
age: "15"
class: "10B"
```

Click on the field you need to update, then update as required, and click the **Update** button.

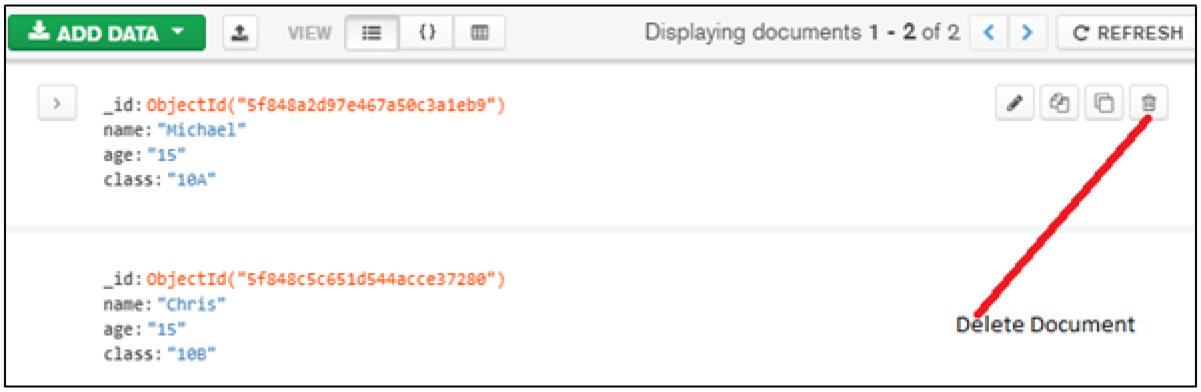


```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")     ObjectId
name : "Michael"                              String
age : "15 "                                   String
class : "10A "                                String
```

Deleting documents

Deleting data is another simple task that Compass can do with just two clicks:

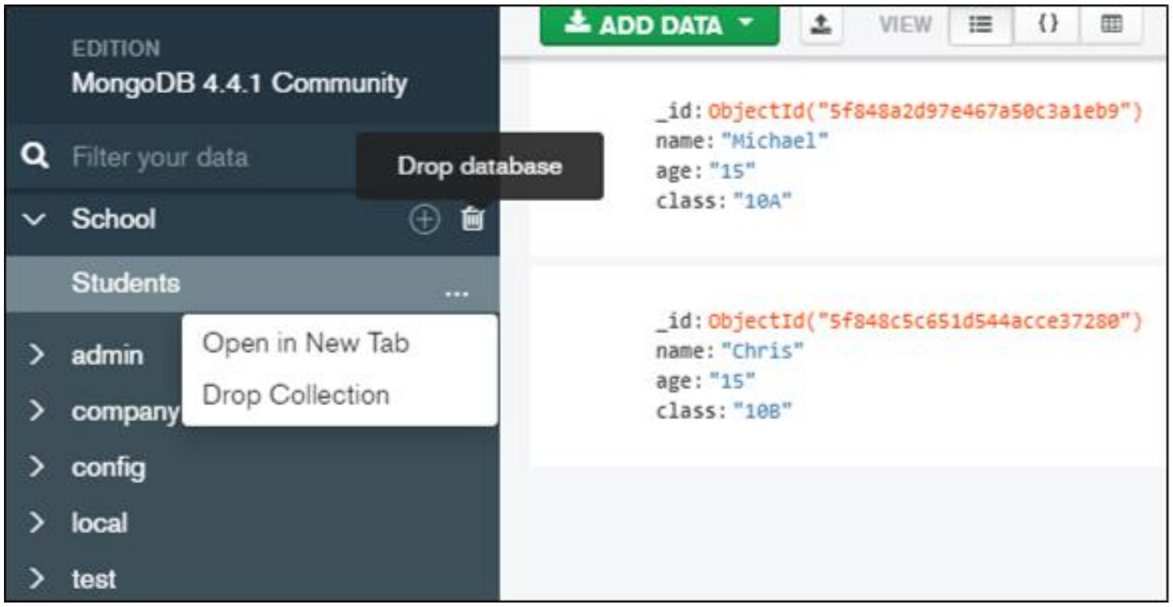
1. Hover over the document you want to delete.
2. Click on the trash icon, located on the right-hand side:



```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")
name: "Michael"
age: "15"
class: "10A"

_id: ObjectId("5f848c5c651d544acce37280")
name: "Chris"
age: "15"
class: "10B"
```

Dropping collection and database

Dropping collections and databases are as straightforward as other operations. You can find all your databases and collections appearing on the left menu of Compass. There you can find options to drop collections and databases. The image below shows both options.



```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")
name: "Michael"
age: "15"
class: "10A"

_id: ObjectId("5f848c5c651d544acce37280")
name: "Chris"
age: "15"
class: "10B"
```

References:

https://www.guru99.com/mongodb-tutorials.html

https://docs.mongodb.com/v4.2/introduction/

https://www.mongodb.com/products/compass

https://www.bmc.com/blogs/mongodb-compass/