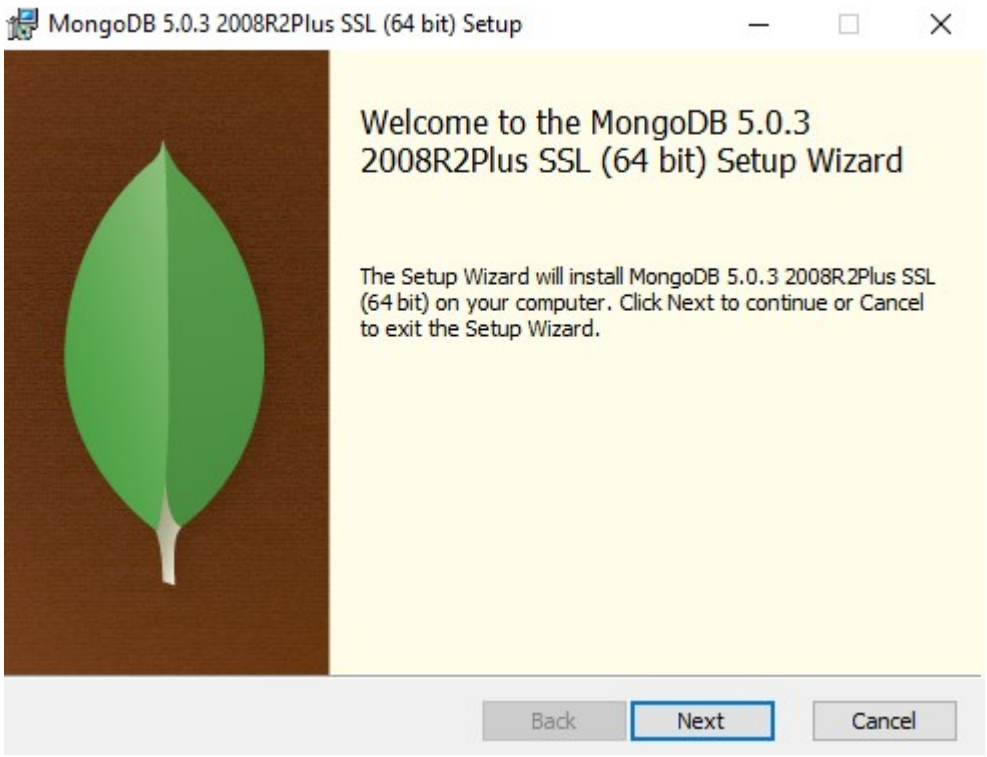
Mongo DB Lab Manual

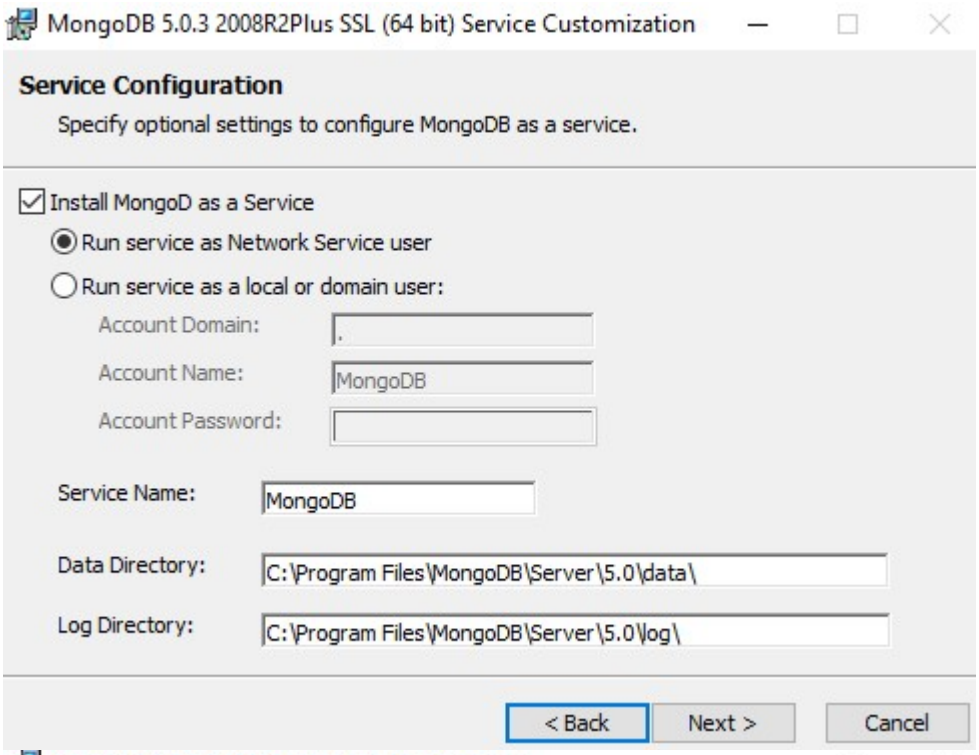
Prepared By: Selvakumar K
Reviewed By: Dr.D.Narashiman, IST Department, Anna University

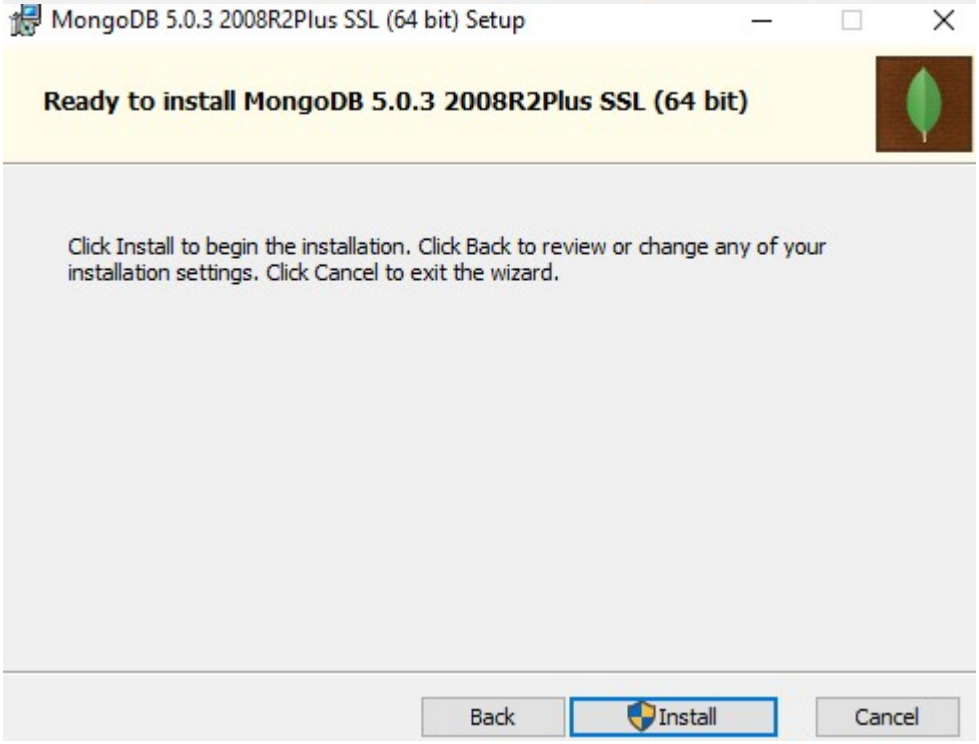
---

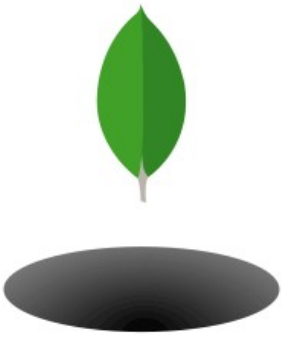
Installation:

https://www.mongodb.com/try/download/community?tck=docs_server&_ga=2.67263761.1475647540.1640838619-602879753.1632928402



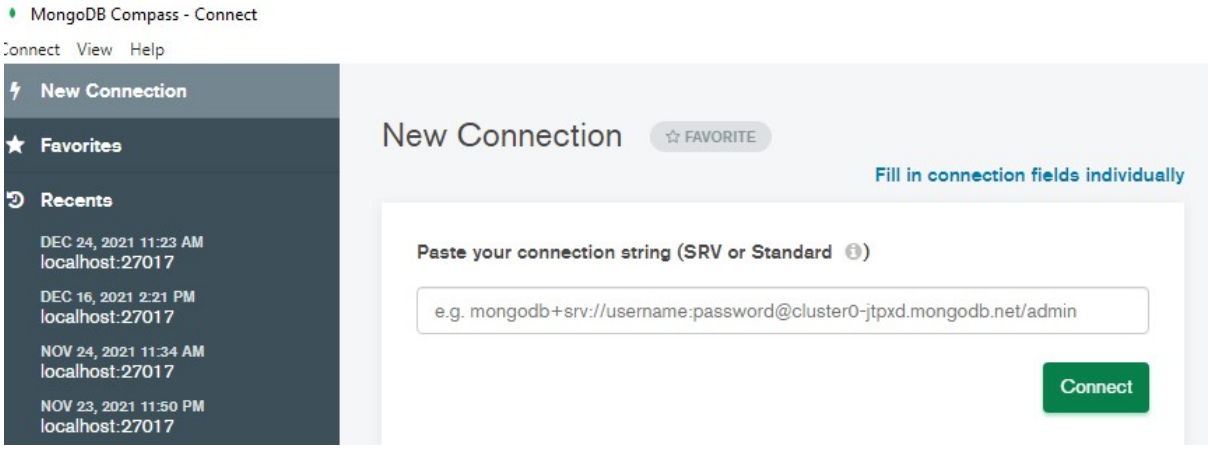

MongoDB 5.0.3 2008R2Plus SSL (64 bit) Service Customization
Service Configuration
Specify optional settings to configure MongoDB as a service.
Install MongoD as a Service
Run service as Network Service user
Run service as a local or domain user:
Account Domain:
.
Account Name:
MongoDB
Account Password:
Service Name:
MongoDB
Data Directory:
C:\Program Files\MongoDB\Server\5.0\data\
Log Directory:
C:\Program Files\MongoDB\Server\5.0\log\
< Back
Next >
Cancel


MongoDB 5.0.3 2008R2Plus SSL (64 bit) Setup
Ready to install MongoDB 5.0.3 2008R2Plus SSL (64 bit)
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel



MongoDB Compass is being installed.

It will launch once it is done.



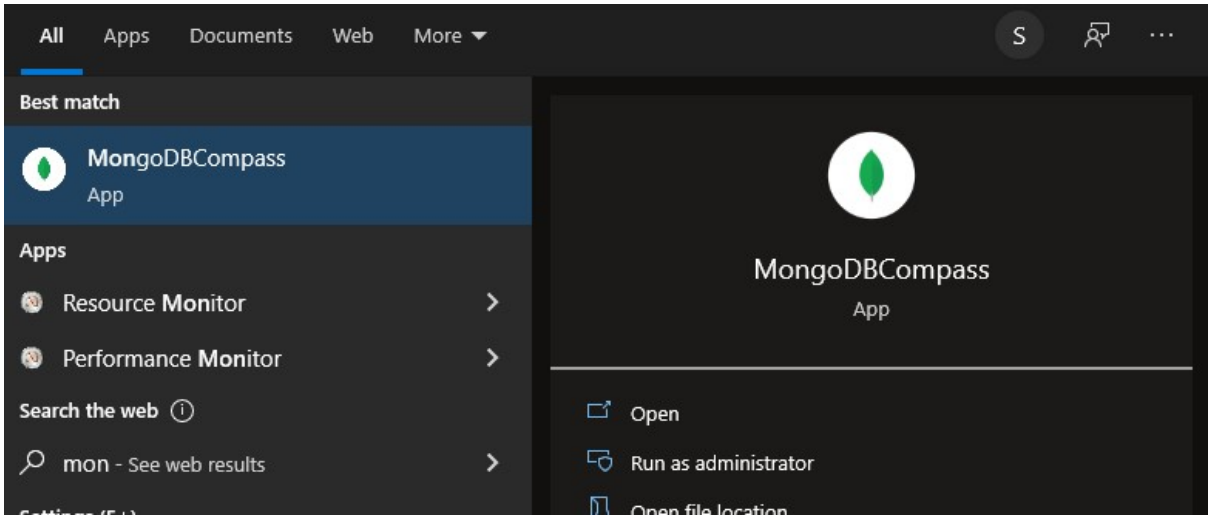
Connecting to localhost:27017

Cancel

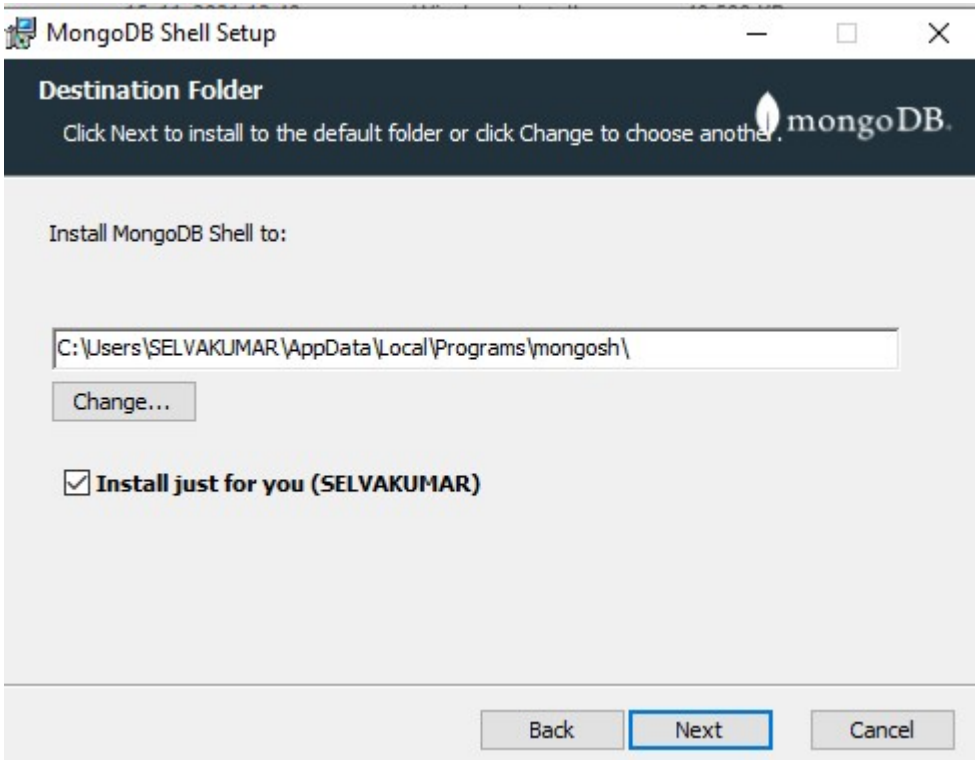
Mongo DB Tools:
https://www.mongodb.com/try/download/database-tools

Name	Date modified	Type	Size
bsondump.exe	12-10-2021 16:53	Application	18,707 KB
mongodump.exe	12-10-2021 16:53	Application	22,375 KB
mongoexport.exe	12-10-2021 16:53	Application	21,941 KB
mongofiles.exe	12-10-2021 16:54	Application	23,123 KB
mongoimport.exe	12-10-2021 16:53	Application	22,283 KB
mongorestore.exe	12-10-2021 16:53	Application	22,830 KB
mongostat.exe	12-10-2021 16:53	Application	21,603 KB
mongotop.exe	12-10-2021 16:53	Application	21,216 KB

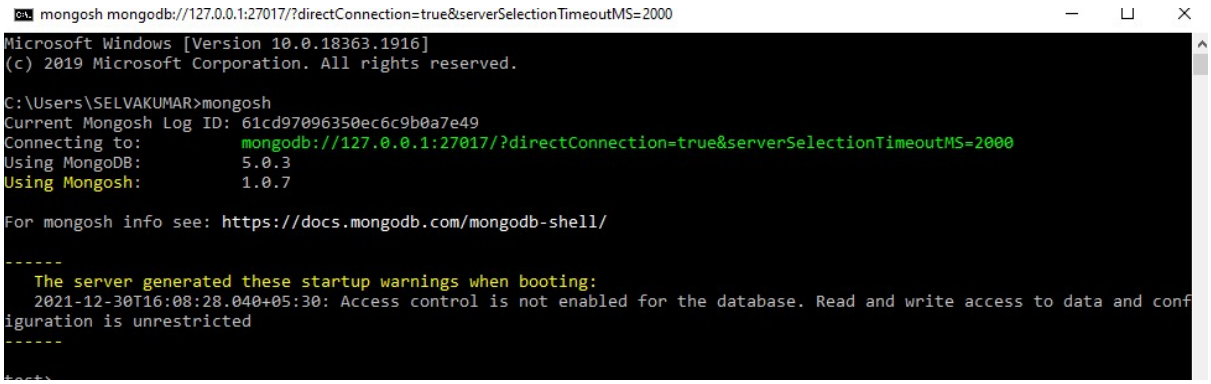
Mongo DB Compass
It will default install when installing Mono DB. You can directly download if you remote Mongo DB Machine.
https://www.mongodb.com/try/download/compass



Mongo DB Shell:
https://www.mongodb.com/try/download/shell



Connecting DB from Shell



JSON

SON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

https://www.json.org/json-en.html

https://www.w3schools.com/js/js_json_intro.asp

```
person = {name:"John", age:31, city:"Chennai"};
```

JSON Values

In **JSON**, values must be one of the following data types:

- a string
- a number
- an object
- an array
- a boolean
- null

Accessing the Values:

```
person.name;
```

person["name"];

Modified the Values:

person.name = "Gilbert";

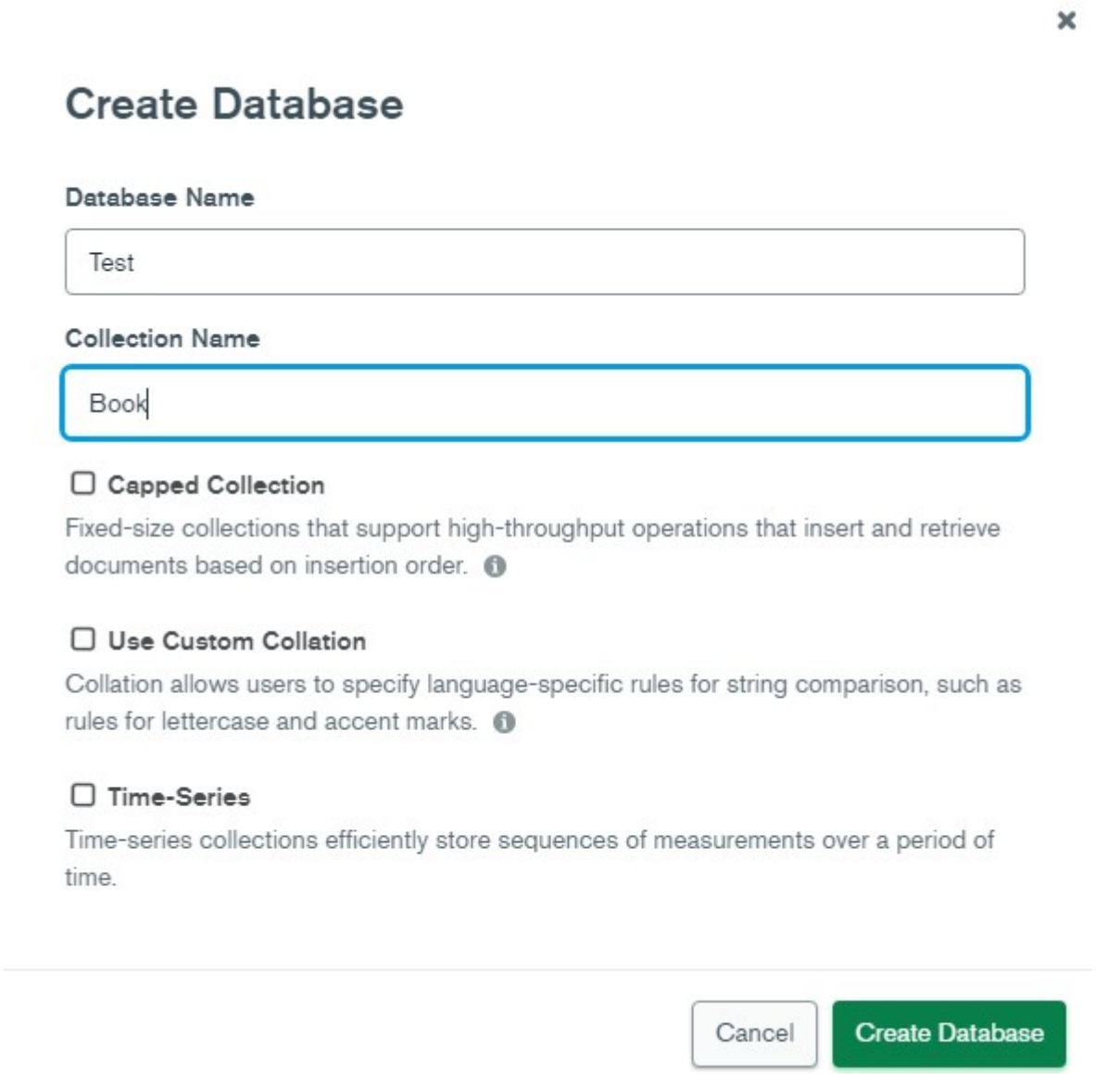
person["name"] = "Gilbert";

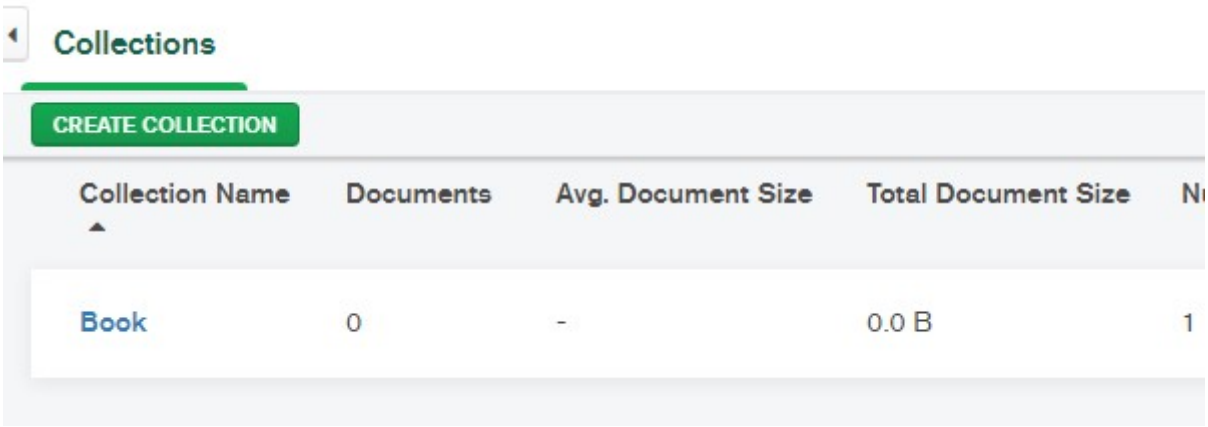
Mongo DB Query:

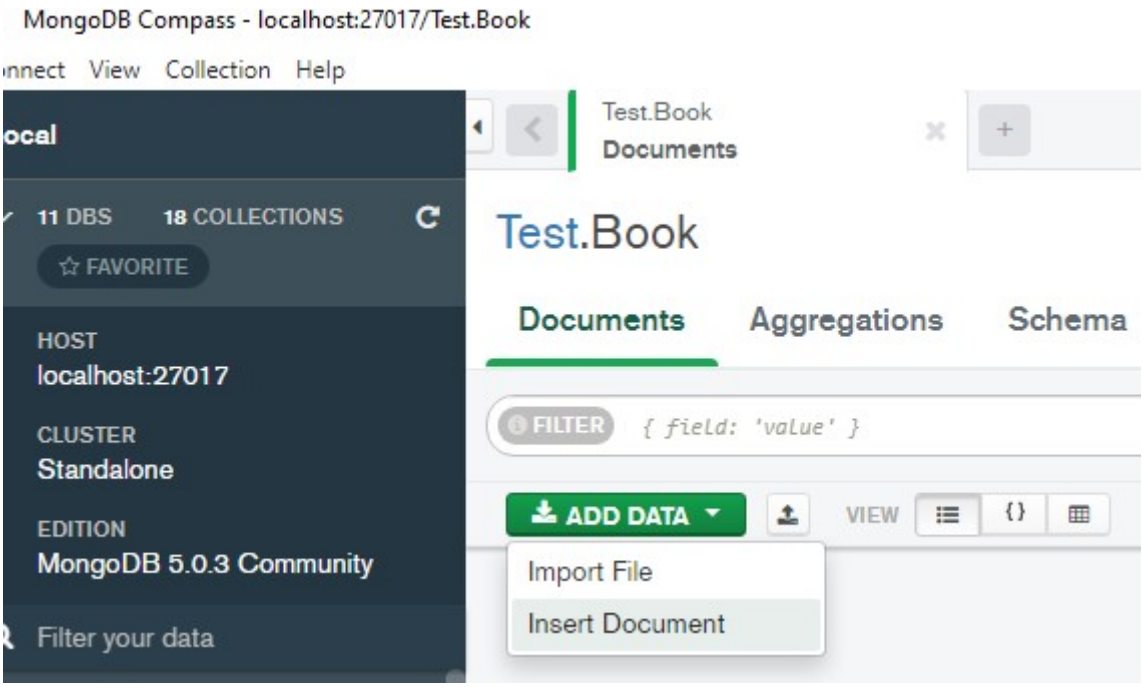
Here Collection is equivalent to table in RDBMS.

Create new Database and Collections

Use Compass Application to create the Database.

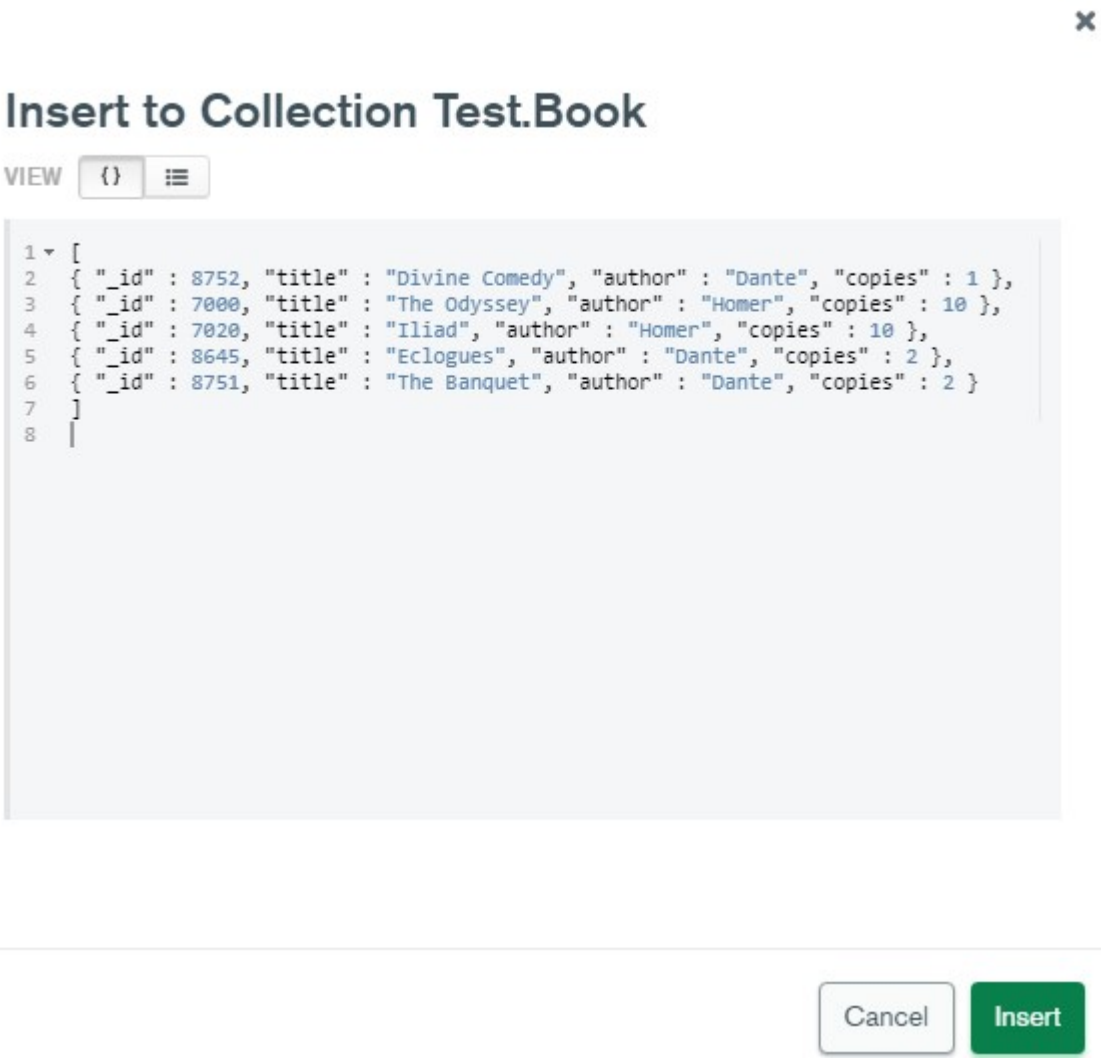




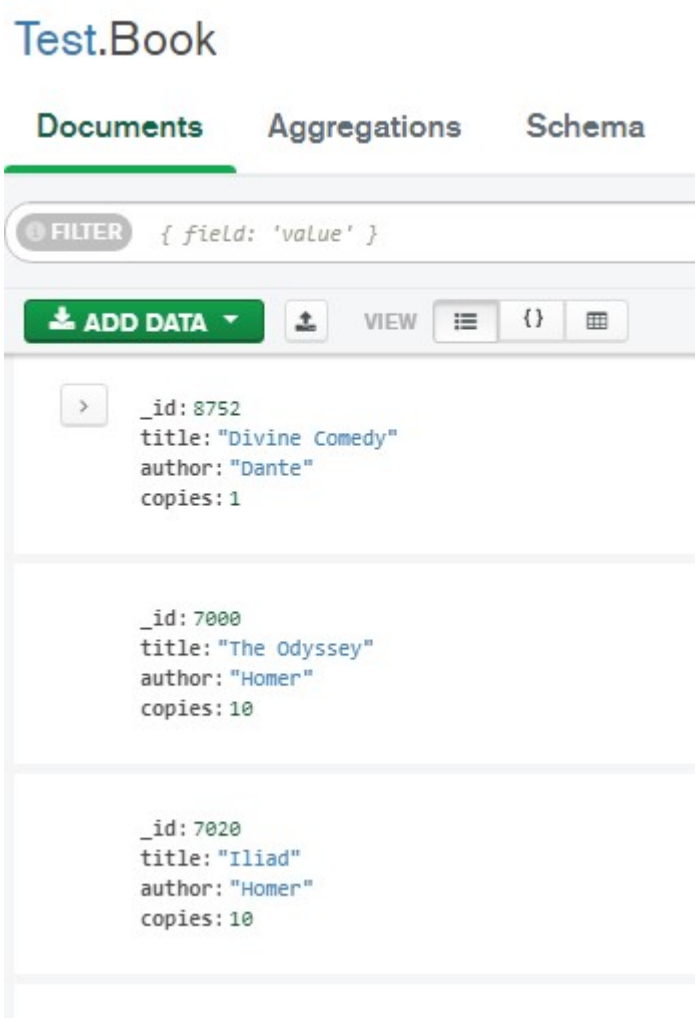


Click the insert Document, paste the following content and click insert

```
[
{ "_id" : 8752, "title" : "Divine Comedy", "author" : "Dante", "copies" : 1 },
{ "_id" : 7000, "title" : "The Odyssey", "author" : "Homer", "copies" : 10 },
{ "_id" : 7020, "title" : "Iliad", "author" : "Homer", "copies" : 10 },
{ "_id" : 8645, "title" : "Eclogues", "author" : "Dante", "copies" : 2 },
{ "_id" : 8751, "title" : "The Banquet", "author" : "Dante", "copies" : 2 }
]
```



After Insert



To Show the list of DBS

```
> show dbs
< Test                41 kB
  admin               41 kB
  airbnb            56.8 MB
  business           152 kB
```

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Microsoft Windows [Version 10.0.18363.1916]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\SELVAKUMAR>mongosh
Current Mongosh Log ID: 61cd9ad34ead561114c3f889
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:          5.0.3
Using Mongosh:          1.0.7

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting:
   2021-12-30T16:08:28.040+05:30: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

test>
```

Use Test

```
mongosh mongodb://127.0.0.1:27017/?directCo

sample> use Test
switched to db Test
Test>
```

Find the Author is Dante

db.Book.find({ "author": "Dante" }).pretty()

```
Test> show collections
Book
Test> db.Book.find({ "author": "Dante" }).pretty()
[
  { _id: 8752, title: 'Divine Comedy', author: 'Dante', copies: 1 },
  { _id: 8645, title: 'Eclogues', author: 'Dante', copies: 2 },
  { _id: 8751, title: 'The Banquet', author: 'Dante', copies: 2 }
]
Test>
```

Data Base Name is case sensitive. Following query won't return any result. "b" is small letter.

```
Test> db.book.find({ "author": "Dante" }).pretty()

Test> show collections
```

Find the Author is Dante and title is "Divine Comedy"

db.Book.find({ "author": "Dante", "title": "Divine Comedy" }).pretty()

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

Test> db.Book.find({ "author": "Dante",  "title": "Divine Comedy" }).pretty()
[ { _id: 8752, title: 'Divine Comedy', author: 'Dante', copies: 1 } ]
Test>
```

Update Document:

Update the Author to "Selva" where the Author is Dante and title is "Divine Comedy"

db.Book.updateOne({"author": "Dante","title": "Divine Comedy" }, {"$set": { "author": "Selva"} })

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

Test> db.Book.updateOne({ "author": "Dante","title": "Divine Comedy" }, { "$set": { "author": "Selva"} })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Test>
```

Find the Author with "Selva"

db.Book.find({ "author": "Selva", "title": "Divine Comedy" }).pretty()

```
}
Test> db.Book.find({ "author": "Selva",  "title": "Divine Comedy" }).pretty()
[ { _id: 8752, title: 'Divine Comedy', author: 'Selva', copies: 1 } ]
Test>
```

Delete Document

Delete the document with the Author with "Selva"

db.Book.deleteOne({"author":"Selva"})

```
Book> use Test
switched to db Test
Test> db.Book.deleteOne({ "author":"Selva"})
{ acknowledged: true, deletedCount: 1 }
Test>
```

Insert with Code:

db.Book.insert({ "_id" : 8762, "title" : "Java", "author" : "Sarumathy", "copies" : 2 })

db.Book.insert([{ "_id" : 8763, "title" : "C++", "author" : "Archana", "copies" : 2 },{ "_id" : 8764, "title" : "NLP", "author" : "Archana", "copies" : 2 }])

We can also use the insertOne for insert single document and and insertMany for inserting multiple document.

Mongo DB Dump

Download the Mongo DB data base dump from the following link for practice.

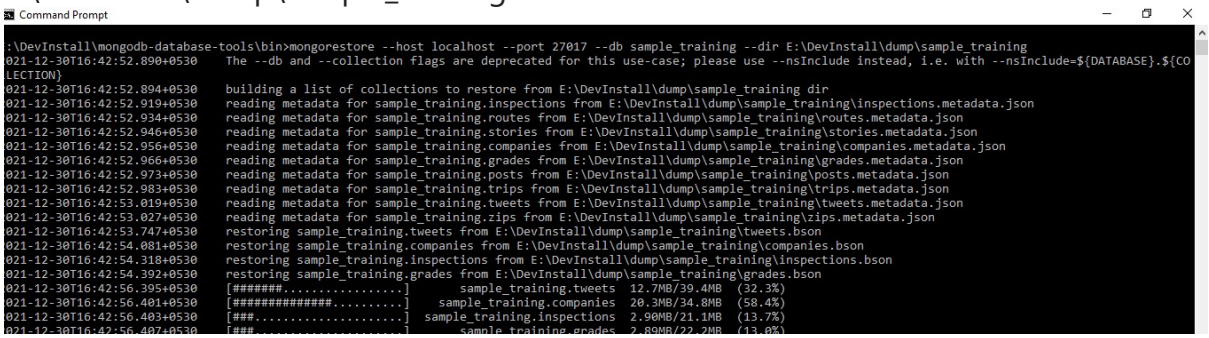
https://github.com/huynhsamha/quick-mongo-atlas-datasets

The sample_training database contains a set of realistic data.

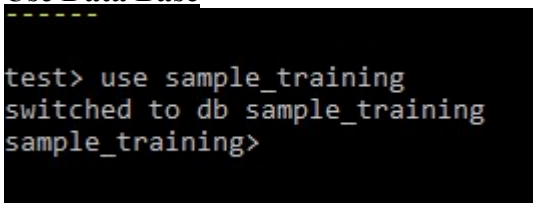
The `sample_training` database contains the following collections:

Collection Name	Description
companies	Contains a list of Crunchbase Data company information.
grades	Contains student grade information on a given class, including scores on different assessments.
inspections	Contains a list of New York City business inspections, including whether the business failed or passed the inspection.
posts	Contains randomized US Senate speeches organized as blog posts with randomly generated comments.
routes	Contains information of airline routes, with source and destination airports, the service airline and the type of airplane. This collection is used in labs that explore the $graphLookup aggregation stage.
stories	Contains Digg stories, a website for sharing and commenting on online content.
trips	Contains New York City Citibike Data trips data. This data is useful to explore the $graphLookup aggregation stage and showcase Geospatial Queries.
tweets	Contains tweet data from Twitter Decahose stream service.
zips	Contains United States general cities postal/zip code data.

mongorestore --host localhost --port 27017 --db sample_training --dir E:\DevInstall\dump\sample_training



Use Data Base



Show Collections

```
test> use sample_training
switched to db sample_training
sample_training> show collections
companies
grades
inspections
posts
routes
stories
trips
tweets
zips
sample_training>
```

1. Find with filter
db.zips.find({"state":"NY"})

```
sample_training> db.zips.find({"state": "NY"})
[
  {
    _id: ObjectId("5c8eccc1caa187d17ca72f89"),
    city: 'FISHERS ISLAND',
    zip: '06390',
    loc: { y: 41.263934, x: 72.017834 },
    pop: 329,
    state: 'NY'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca72f8a"),
    city: 'NEW YORK',
    zip: '10001',
    loc: { y: 40.74838, x: 73.996705 },
    pop: 18913,
    state: 'NY'
  },
  {
```

2. Find the Count
db.zips.find({"state":"NY"}).count()

```
sample_training> db.zips.find({"state": "NY"}).count()
1596
sample_training>
```

3. Multiple and Conditions
db.zips.find({"state": "NY", "city": "ALBANY"})

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
sample_training> db.zips.find({"state": "NY", "city": "ALBANY"})
[
  {
    _id: ObjectId("5c8eccc1caa187d17ca731d0"),
    city: 'ALBANY',
    zip: '12204',
    loc: { y: 42.684667, x: 73.735364 },
    pop: 6927,
    state: 'NY'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca731d4"),
    city: 'ALBANY',
    zip: '12206',
    loc: { y: 42.668326, x: 73.774406 },
    pop: 17230,
    state: 'NY'
  },
  {
```

2. Insert the document

We will duplicate the document and insert

db.inspections.find({"id" : "10021-2015-ENFO", "certificate_number" : 9278806}).pretty()

```
sample_training> db.inspections.find({"id" : "10021-2015-ENFO", "certificate_number" : 9278806}).pretty()
[
  {
    _id: ObjectId("56d61033a378eccde8a8354f"),
    id: '10021-2015-ENFO',
    certificate_number: 9278806,
    business_name: 'ATLIXCO DELI GROCERY INC.',
    date: 'Feb 20 2015',
    result: 'No Violation Issued',
    sector: 'Cigarette Retail Dealer - 127',
    address: {
      city: 'RIDGEWOOD',
      zip: 11385,
      street: 'MENAHAN ST',
      number: 1712
    }
  }
]
sample_training>
```

db.inspections.insert({

"_id" : ObjectId("56d61033a378eccde8a8354f"),

"id" : "10021-2015-ENFO",

"certificate_number" : 9278806,

"business_name" : "ATLIXCO DELI GROCERY INC.",

"date" : "Feb 20 2015",

"result" : "No Violation Issued",

"sector" : "Cigarette Retail Dealer - 127",

"address" : {

"city" : "RIDGEWOOD",

"zip" : 11385,

"street" : "MENAHAN ST",

```
        "number" : 1712
    }
})
```

```
sample_training> db.inspections.insert({
...     "_id" : ObjectId("56d61033a378eccde8a8354f"),
...     "id" : "10021-2015-ENFO",
...     "certificate_number" : 9278806,
...     "business_name" : "ATLIXCO DELI GROCERY INC.",
...     "date" : "Feb 20 2015",
...     "result" : "No Violation Issued",
...     "sector" : "Cigarette Retail Dealer - 127",
...     "address" : {
.....             "city" : "RIDGEWOOD",
.....             "zip" : 11385,
.....             "street" : "MENAHAN ST",
.....             "number" : 1712
.....         }
...   })
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
Uncaught:
MongoBulkWriteError: E11000 duplicate key error collection: sample_training.inspections index: _id_ dup key: { _id: Obje
ctId('56d61033a378eccde8a8354f') }
Result: BulkWriteResult {
  result: {
    ok: 1,
    writeErrors: [
      WriteError {
```

Since the ID exists already in the data base, we are getting duplicate error.

2.1. Now we remove the Id field and insert again. The id will be automatically inserted by DB.

```
db.inspections.insert({
    "id" : "10021-2015-ENFO",
    "certificate_number" : 9278806,
    "business_name" : "ATLIXCO DELI GROCERY INC.",
    "date" : "Feb 20 2015",
    "result" : "No Violation Issued",
    "sector" : "Cigarette Retail Dealer - 127",
    "address" : {
        "city" : "RIDGEWOOD",
        "zip" : 11385,
        "street" : "MENAHAN ST",
        "number" : 1712
    }
})
```

```
sample_training> db.inspections.insert({ "_id": ObjectId("56d61033a378eccde8a8354f"), "id": "10021-2015-ENFO", "certific
ate_number": 9278806, "business_name": "ATLIXCO DELI GROCERY INC.", "date": "Feb 20 2015", "result": "No Violation Issue
d", "sector": "Cigarette Retail Dealer - 127", "address": { "city": "RIDGEWOOD", "zip": 11385, "street": "MENAHAN ST", "
...     "id" : "10021-2015-ENFO",
...     "certificate_number" : 9278806,
...     "business_name" : "ATLIXCO DELI GROCERY INC.",
...     "date" : "Feb 20 2015",
...     "result" : "No Violation Issued",
...     "sector" : "Cigarette Retail Dealer - 127",
...     "address" : {
.....             "city" : "RIDGEWOOD",
.....             "zip" : 11385,
.....             "street" : "MENAHAN ST",
.....             "number" : 1712
.....         }
...   })
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("61cda07cbe33ffd8f7cc002a") }
}
sample_training>
```

Query again to verify the inserted document

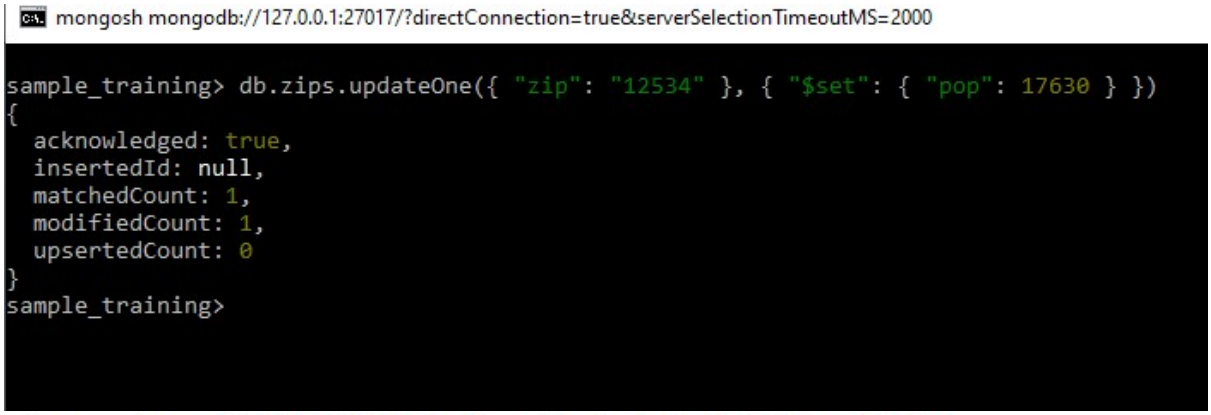
db.inspections.find({"id" : "10021-2015-ENFO", "certificate_number" : 9278806}).pretty()

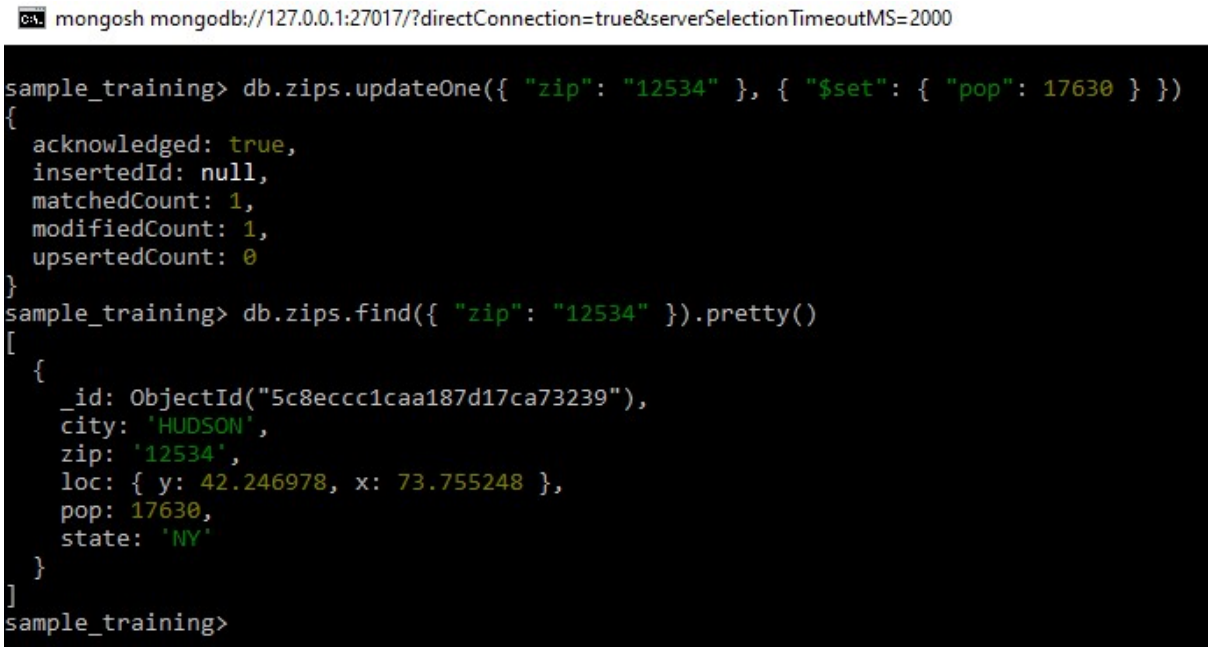
Update Document

1. Update a single document in the zips collection where the zip field is equal to "12534" by setting the value of the "pop" field to 17630.

db.zips.find({ "zip": "12534" }).pretty()

db.zips.updateOne({ "zip": "12534" }, { "$set": { "pop": 17630 } })





2. Update all documents in the zips collection where the city field is equal to "HUDSON" by adding 10 to the current value of the "pop" field.

db.zips.find({ "city": "HUDSON" }).pretty()

```
sample_training> db.zips.find({ "city": "HUDSON" }).pretty()
[
  {
    _id: ObjectId("5c8eccc1caa187d17ca6f9ff"),
    city: 'HUDSON',
    zip: '80642',
    loc: { y: 40.060555, x: 104.653208 },
    pop: 2369,
    state: 'CO'
  },
  {tched to db zips
    _id: ObjectId("5c8eccc1caa187d17ca6ff48"),
    city: 'HUDSON',
    zip: '34669',
    loc: { y: 28.350634, x: 82.628793 },
    pop: 8577,
    state: 'FL'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca6ff4c"),
    city: 'HUDSON',
```

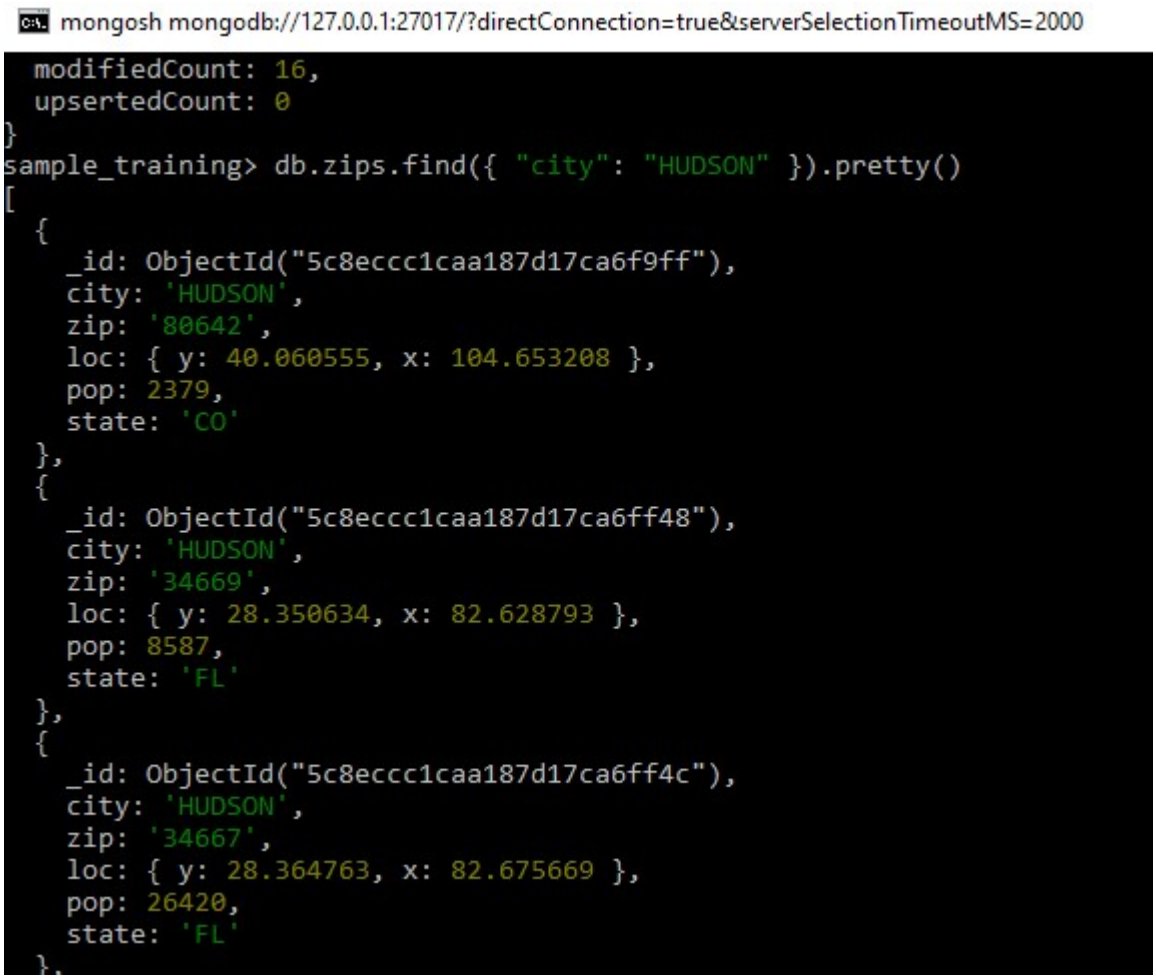
db.zips.find({ "city": "HUDSON" }).count()

```
sample_training> db.zips.find({ "city": "HUDSON" }).count()
16
sample_training>
```

db.zips.updateMany({ "city": "HUDSON" }, { "$inc": { "pop": 10 } })

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

sample_training> db.zips.updateMany({ "city": "HUDSON" }, { "$inc": { "pop": 10 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 16,
  modifiedCount: 16,
  upsertedCount: 0
}
sample_training>
```



Deleted Documents

db.inspections.insert([{ "_id": 1, "test": 1 },{ "_id": 1, "test": 2 }, { "_id": 3, "test": 3 }])

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
sample_training> db.inspections.insert([{ "_id": 1, "test": 1 },{ "_id": 1, "test": 2 }, { "_id": 3, "test": 3 }])
Uncaught:
MongoBulkWriteError: E11000 duplicate key error collection: sample_training.inspections index: _id_ dup key: { _id: 1 }
Result: BulkWriteResult {
  result: {
    ok: 1,
    writeErrors: [
      WriteError {
        err: {
          index: 1,
          code: 11000,
          errmsg: 'E11000 duplicate key error collection: sample_training.inspections index: _id_ dup key: { _id: 1 }',
          errInfo: undefined,
          op: { _id: 1, test: 2 }
        }
      }
    ],
    writeConcernErrors: [],
    insertedIds: [
      { index: 0, _id: 1 },
      { index: 1, _id: 1 },
      { index: 2, _id: 3 }
    ],
    nInserted: 1,
    nUpserted: 0,
    nMatched: 0,
    nModified: 0,
    nRemoved: 0,
    upserted: []
  }
```

db.inspections.find({ "_id": 1 })

```
> db.inspections.find({ "_id": 1 })

< { _id: 1, test: 1 }
```

Since error is happen only when insert second record.
db.inspections.find({ "_id": 2 })
No Results due to error
db.inspections.find({ "_id": 3 })
No Results due to error

```
sample_training> db.inspections.find({ "_id": 1 })
[ { _id: 1, test: 1 } ]
sample_training> db.inspections.find({ "_id": 2 })

sample_training> db.inspections.find({ "_id": 3 })

sample_training>
```

db.inspections.insert([{ "_id": 11, "test": 1 },{ "_id": 12, "test": 2 }, { "_id":13, "test": 3 }])

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
sample_training> db.inspections.insert([{ "_id": 11, "test": 1 },{ "_id": 12, "test": 2 }, { "_id":13, "test": 3 }])
{ acknowledged: true, insertedIds: { '0': 11, '1': 12, '2': 13 } }
sample_training> db.inspections.deleteOne({ "test": 1 })
{ acknowledged: true, deletedCount: 1 }
sample_training>
```

db.inspections.deleteOne({ "test": 1 })

```
sample_training> db.inspections.deleteOne({ "test": 1 })
{ acknowledged: true, deletedCount: 1 }
```

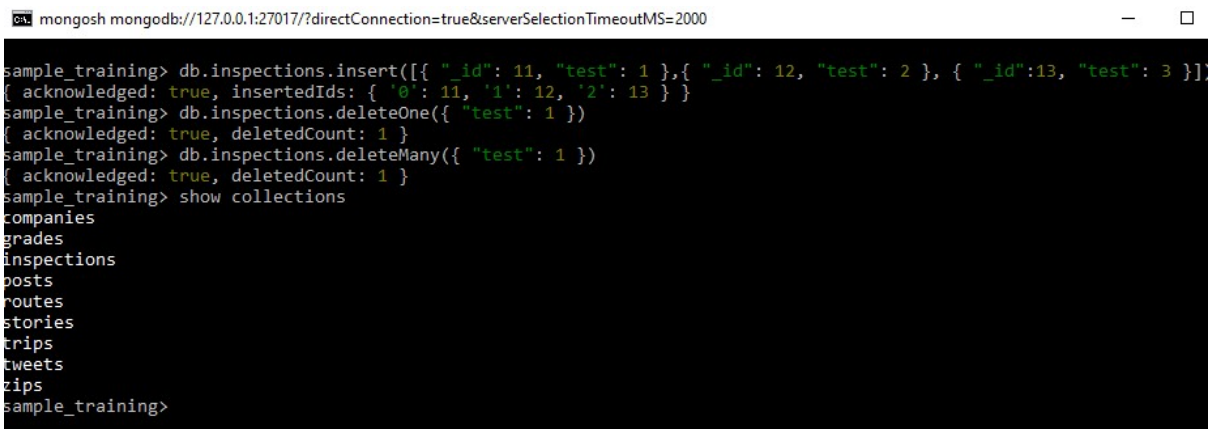
There are two records now, but it will delete only the first records.
We will insert one more records again and then call deleteMany()
db.inspections.insert([{ "_id": 21, "test": 1 }])
db.inspections.deleteMany({ "test": 1 })

```
> db.inspections.insert([{ "_id": 21, "test": 1 }])
< { acknowledged: true, insertedIds: { '0': 21 } }
> db.inspections.deleteMany({ "test": 1 })
< { acknowledged: true, deletedCount: 2 }
```

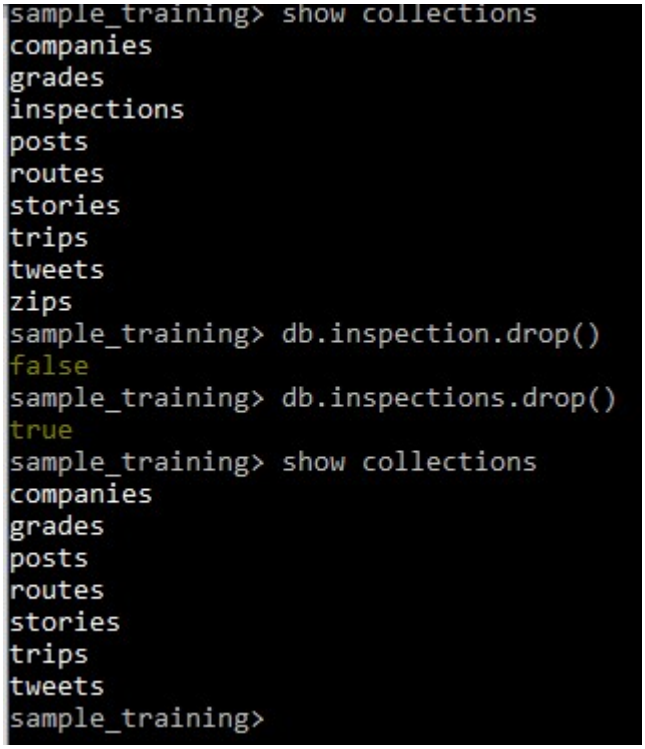
Drop Collection

show collections

```
sample_training> show collections
companies
grades
inspections
posts
routes
stories
trips
tweets
zips
sample_training>
```



db.inspections.drop()


sample_training> show collections
companies
grades
inspections
posts
routes
stories
trips
tweets
zips
sample_training> db.inspection.drop()
false
sample_training> db.inspections.drop()
true
sample_training> show collections
companies
grades
posts
routes
stories
trips
tweets
sample_training>

Advanced Topic

Comparison Operator

Use the sample_training database, trips collection for the following exercise

1. Find all documents where the tripduration **was less than or equal to 70 seconds** and the **usertype was not Subscriber**:

db.trips.find({ "tripduration": { "$lte" : 70 },"usertype": { "$ne": "Subscriber" } }).pretty()

```
Test> use sample_training
switched to db sample_training
sample_training> db.trips.find({ "tripduration": { "$lte" : 70 },"usertype": { "$ne": "Subscriber" } }).pretty()
[
  {
    _id: ObjectId("572bb8232b288919b68af7cd"),
    tripduration: 66,
    'start station id': 460,
    'start station name': 'S 4 St & Wythe Ave',
    'end station id': 460,
    'end station name': 'S 4 St & Wythe Ave',
    bikeid: 23779,
    usertype: 'Customer',
    'birth year': '',
    gender: 0,
    'start station location': { type: 'Point', coordinates: [ -73.96590294, 40.71285887 ] },
    'end station location': { type: 'Point', coordinates: [ -73.96590294, 40.71285887 ] },
    'start time': ISODate("2016-01-02T11:49:11.000Z"),
    'stop time': ISODate("2016-01-02T11:50:18.000Z")
  }
]
sample_training>
```

2. Find all documents where the tripduration was less than or **equal to 70 seconds** and the **usertype was Customer** using a redundant equality operator:

db.trips.find({ "tripduration": { "$lte" : 70 },"usertype": { "$eq": "Subscriber" } }).pretty()

```
> db.trips.find({ "tripduration": { "$lte" : 70 },"usertype": { "$eq": "Subscriber" } }).pretty()
< { _id: ObjectId("572bb8222b288919b68ac2d4"),
    tripduration: 61,
    'start station id': 3150,
    'start station name': 'E 85 St & York Ave',
    'end station id': 3150,
    'end station name': 'E 85 St & York Ave',
```

3. Find all documents where the tripduration was **less than or equal to 70 seconds** and the **usertype was Customer** using the implicit equality operator:

db.trips.find({ "tripduration": { "$lte" : 70 },"usertype": "Customer" }).pretty()

```
sample_training> db.trips.find({ "tripduration": { "$lte" : 70 },"usertype": "Customer" }).pretty()
[
  {
    _id: ObjectId("572bb8232b288919b68af7cd"),
    tripduration: 66,
    'start station id': 460,
    'start station name': 'S 4 St & Wythe Ave',
    'end station id': 460,
    'end station name': 'S 4 St & Wythe Ave',
    bikeid: 23779,
    usertype: 'Customer',
    'birth year': '',
    gender: 0,
    'start station location': { type: 'Point', coordinates: [ -73.96590294, 40.71285887 ] },
    'end station location': { type: 'Point', coordinates: [ -73.96590294, 40.71285887 ] },
    'start time': ISODate("2016-01-02T11:49:11.000Z"),
    'stop time': ISODate("2016-01-02T11:50:18.000Z")
  }
]
sample_training>
```

2. Query Operators – Logic

Use the **sample_training** database**, routes** collection for the following exercise

Find all documents where airplanes CR2 or A81 left or landed in the KZN airport:

```
db.routes.find({ "$and": [ { "$or" :[ { "dst_airport": "KZN" },
                                      { "src_airport": "KZN" }
                                    ] },
                           { "$or" :[ { "airplane": "CR2" },
                                      { "airplane": "A81" } ] }

                          ]}).pretty()
```

```
> db.routes.find({ "$and": [ { "$or" :[ { "dst_airport": "KZN" },
                                        { "src_airport": "KZN" }
                                      ] },
                             { "$or" :[ { "airplane": "CR2" },
                                        { "airplane": "A81" } ] }
                            ]}).pretty();
```

Projection

`{ $project: { <specification(s)> }`
https://www.mongodb.com/docs/manual/reference/operator/aggregation/project/

Filed can be included are excluded using the value 1 and 0 respectively.
_id fields are included by default. If don't want can exclude it.
{ $project: { "<field1>": 0, "<field2>": 0, ... } } // Return all but the specified fields
Insert the following document.

```
db.books.insertOne(
{
  "_id" : 1,
  title: "abc123",
  isbn: "0001122223334",
  author: { last: "zzz", first: "aaa" },
  copies: 5
})
```

`<field>: <1 or true>`	Specifies the inclusion of a field. Non-zero integers are also treated as `true`.
`_id: <0 or false>`	Specifies the suppression of the `_id` field. To exclude a field conditionally, use the `REMOVE` variable instead. For details, see Exclude Fields Conditionally.
`<field>: <expression>`	Adds a new field or resets the value of an existing field. If the the expression evaluates to `$$REMOVE`, the field is excluded in the output. For details, see Exclude Fields Conditionally.
`<field>:<0 or false>`	Specifies the exclusion of a field. To exclude a field conditionally, use the `REMOVE` variable instead. For details, see Exclude Fields Conditionally. If you specify the exclusion of a field other than `_id`, you **cannot** employ any other `$project` specification forms. This restriction does not apply to conditionally exclusion of a field using the `REMOVE` variable. See also the `$unset` stage to exclude fields.

Figure 1: $project specifications.

db.books.aggregate([{ $project : { title : 1 , author : 1 } }])

```
> db.books.insertOne(
  {
    "_id" : 1,
    title: "abc123",
    isbn: "0001122223334",
    author: { last: "zzz", first: "aaa" },
    copies: 5
  });
< { acknowledged: true, insertedId: 1 }
> db.books.aggregate( [ { $project : { title : 1 , author : 1 } } ] );
< { _id: 1,
    title: 'abc123',
    author: { last: 'zzz', first: 'aaa' } }
sample_training>
```

1. If want to supress the id filed it can be excluded by 0. For Example find the following query.

db.books.aggregate([{ $project : { _id: 0, title : 1 , author : 1 } }])

```
> db.books.aggregate( [ { $project : { _id: 0, title : 1 , author : 1 } } ] )
< { title: 'abc123', author: { last: 'zzz', first: 'aaa' } }
sample_training>
```

Insert the following document also.

```
db.books.insertOne(
{
  "_id" : 2,
  title: "abc123",
  isbn: "0001122223334",
  author: { last: "zzz", first: "aaa" },
  copies: 5,
  lastModified: "2016-07-28"
})
```

When retrieving both the document for consistency we will skip the lastModified filed.

db.books.aggregate([{ $project : { "lastModified": 0 } }])

2. Exclude Fields from Embedded Documents

db.books.aggregate([{ $project : { "author.first" : 0, "lastModified" : 0 } }])

```
> db.books.aggregate( [ { $project : { "author.first" : 0, "lastModified" : 0 } } ] );
< { _id: 1,
    title: 'abc123',
    isbn: '0001122223334',
    author: { last: 'zzz' },
    copies: 5 }
  { _id: 2,
    title: 'abc123',
    isbn: '0001122223334',
    author: { last: 'zzz' },
    copies: 5 }
```

3. Conditionally Exclude Fields from Embedded Documents

```
db.books.insertMany(
[
{
  "_id" : 3,
  title: "abc123",
  isbn: "0001122223334",
  author: { last: "zzz", first: "aaa" },
  copies: 5,
  lastModified: "2016-07-28"
},
{
  "_id" : 4,
  title: "Baked Goods",
  isbn: "9999999999999",
```

```
  author: { last: "xyz", first: "abc", middle: "" },
  copies: 2,
  lastModified: "2017-07-21"
},
{
  "_id" : 5,
  title: "Ice Cream Cakes",
  isbn: "8888888888888",
  author: { last: "xyz", first: "abc", middle: "mmm" },
  copies: 5,
  lastModified: "2017-07-22"
}
]);
```

Remove the middle name if it is empty.

```
db.books.aggregate( [
   {
      $project: {
         title: 1,
         "author.first": 1,
         "author.last" : 1,
         "author.middle": {
            $cond: {
               if: { $eq: [ "", "$author.middle" ] },
               then: "$$REMOVE",
               else: "$author.middle"
            }
         }
      }
   }
] )
```

```
{ _id: 1,
  title: 'abc123',
  author: { last: 'zzz', first: 'aaa' } }
{ _id: 2,
  title: 'abc123',
  author: { last: 'zzz', first: 'aaa' } }
{ _id: 3,
  title: 'abc123',
  author: { last: 'zzz', first: 'aaa' } }
{ _id: 4,
  title: 'Baked Goods',
  author: { last: 'xyz', first: 'abc' } }
{ _id: 5,
  title: 'Ice Cream Cakes',
  author: { last: 'xyz', first: 'abc', middle: 'mmm' } }
sample_training>
```

Remove the isbn when it equals to 8888888888888.

```
db.books.aggregate( [
   {
      $project: {
         title: 1,
         "author.first": 1,
              "author.last" : 1,
              "isbn" : 1,
         "isbn": {
            $cond: {
               if: { $eq: [ "8888888888888", "$isbn" ] },
               then: "$REMOVE",
               else: "$isbn"
            }
         }
      }
   }
] )
```

Array Operators and Projection

Download the Mongo DB data base dump from the following link for practice.
https://github.com/huynhsamha/quick-mongo-atlas-datasets
mongorestore --host localhost --port 27017 --db sample_training --dir E:\DevInstall\dump\sample_airbnb

Use the sample_training database and collection sample_airbnb

```
E:\DevInstall\mongodb-database-tools\bin>mongorestore --host localhost --port 27017 --db sample_training --dir E:\DevInstall\dump\sample_airbnb
2021-12-30T20:58:05.375+0530    The --db and --collection flags are deprecated for this use-case; please use --nsInclude instead, i.e. with --nsInclude=${DATABASE}.${COLLECTION}
2021-12-30T20:58:05.410+0530    building a list of collections to restore from E:\DevInstall\dump\sample_airbnb dir
2021-12-30T20:58:05.430+0530    reading metadata for sample_training.listingsAndReviews from E:\DevInstall\dump\sample_airbnb\listingsAndReviews.metadata.json
2021-12-30T20:58:06.114+0530    restoring sample_training.listingsAndReviews from E:\DevInstall\dump\sample_airbnb\listingsAndReviews.bson
2021-12-30T20:58:08.477+0530    [#####.....................]  sample_training.listingsAndReviews  20.9MB/90.0MB  (23.2%)
2021-12-30T20:58:11.364+0530    [#########.................]  sample_training.listingsAndReviews  38.1MB/90.0MB  (42.3%)
2021-12-30T20:58:14.363+0530    [############..............]  sample_training.listingsAndReviews  47.3MB/90.0MB  (52.5%)
2021-12-30T20:58:17.363+0530    [###############...........]  sample_training.listingsAndReviews  57.5MB/90.0MB  (63.9%)
2021-12-30T20:58:20.363+0530    [##################........]  sample_training.listingsAndReviews  66.0MB/90.0MB  (73.3%)
2021-12-30T20:58:23.364+0530    [######################....]  sample_training.listingsAndReviews  79.8MB/90.0MB  (88.7%)
2021-12-30T20:58:26.267+0530    [########################]  sample_training.listingsAndReviews  90.0MB/90.0MB  (100.0%)
2021-12-30T20:58:26.289+0530    finished restoring sample_training.listingsAndReviews (5555 documents, 0 failures)
2021-12-30T20:58:26.289+0530    restoring indexes for collection sample_training.listingsAndReviews from metadata
2021-12-30T20:58:26.320+0530    index: &idx.IndexDocument{Options:primitive.M{"background":true, "name":"property_type_1_room_type_1_beds_1", "ns":"sample_airbnb.listingsAndReviews", "v":2}, Key:primitive.D{primitive.E{Key:"property_type", Value:1}, primitive.E{Key:"room_type", Value:1}, primitive.E{Key:"beds", Value:1}}, PartialFilterExpression:primitive.D(nil)}
2021-12-30T20:58:26.321+0530    index: &idx.IndexDocument{Options:primitive.M{"background":true, "name":"name_1", "ns":"sample_airbnb.listingsAndReviews", "v":2}, Key:primitive.D{primitive.E{Key:"name", Value:1}}, PartialFilterExpression:primitive.D(nil)}
2021-12-30T20:58:26.322+0530    index: &idx.IndexDocument{Options:primitive.M{"2dsphereIndexVersion":3, "background":true, "name":"address.location_2dsphere", "ns":"sample_airbnb.listingsAndReviews", "v":2}, Key:primitive.D{primitive.E{Key:"address.location", Value:"2dsphere"}}, PartialFilterExpression:primitive.D(nil)}
2021-12-30T20:58:28.454+0530    5555 document(s) restored successfully. 0 document(s) failed to restore.
```

Find all documents that have Wifi as one of the amenities only include price and address in the resulting cursor:

Here the amenities is the array type.

```
  db.listingsAndReviews.find({"amenities": "Wifi"}, {"price":1, "address":1,
"_id":0})
```

```
sample_training> db.listingsAndReviews.find({ "amenities": "Wifi" },
...                         { "price": 1, "address": 1, "_id": 0 }).pretty()
[
  {
    price: Decimal128("80.00"),
    address: {
      street: 'Porto, Porto, Portugal',
      suburb: '',
      government_area: 'Cedofeita, Ildefonso, Sé, Miragaia, Nicolau, Vitória',
      market: 'Porto',
      country: 'Portugal',
      country_code: 'PT',
      location: {
        type: 'Point',
        coordinates: [ -8.61308, 41.1413 ],
        is_location_exact: false
      }
    }
  },
  {
    price: Decimal128("317.00"),
    address: {
      street: 'Rio de Janeiro, Rio de Janeiro, Brazil',
      suburb: 'Jardim Botânico',
```

Find all documents with exactly 20 amenities which include all the amenities listed in the query array, and display their price and address

```
  db.listingsAndReviews.find({"amenities": {"$size":20,
"$all":["Wifi","Iron"]}})
```

```
> db.listingsAndReviews.find({"amenities": {"$size":20, "$all":["Wifi","Iron"]}}).count()
< 152
```

Array Operators and Sub-Documents

db.trips.findOne({ "start station location.type": "Point" })

```
sample_training> db.trips.findOne({ "start station location.type": "Point" })
{
  _id: ObjectId("572bb8222b288919b68abf5a"),
  tripduration: 379,
  'start station id': 476,
  'start station name': 'E 31 St & 3 Ave',
  'end station id': 498,
  'end station name': 'Broadway & W 32 St',
  bikeid: 17827,
  usertype: 'Subscriber',
  'birth year': 1969,
  gender: 1,
  'start station location': { type: 'Point', coordinates: [ -73.97966069, 40.74394314 ] },
  'end station location': { type: 'Point', coordinates: [ -73.98808416, 40.74854862 ] },
  'start time': ISODate("2016-01-01T00:00:45.000Z"),
  'stop time': ISODate("2016-01-01T00:07:04.000Z")
}
sample_training>
```

db.companies.find({ "relationships.0.person.last_name": "Zuckerberg" },{ "name": 1 }).pretty()

```
sample_training> db.companies.find({ "relationships.0.person.last_name": "Zuckerberg" },
...                   { "name": 1 }).pretty()
[ { _id: ObjectId("52cdef7c4bab8bd675297d8e"), name: 'Facebook' } ]
sample_training>

sample_training>
```

db.companies.find({ "relationships.0.person.first_name": "Mark",
"relationships.0.title": { "$regex": "CEO" } },
{ "name": 1 }).count()

```
sample_training> db.companies.find({ "relationships.0.person.first_name": "Mark",
...                     "relationships.0.title": { "$regex": "CEO" } },
...                   { "name": 1 }).count()
52
sample_training>
```

Element Match

```
sample_training> db.companies.find({ "relationships":
...                         { "$elemMatch": { "is_past": true,
.....                                         "person.first_name": "Mark" } } },
...                     { "name": 1 }).pretty()
[
  { _id: ObjectId("52cdef7c4bab8bd675297d94"), name: 'Twitter' },
  { _id: ObjectId("52cdef7c4bab8bd675297d9e"), name: 'CBS' },
  { _id: ObjectId("52cdef7c4bab8bd675297da0"), name: 'Babelgum' },
  { _id: ObjectId("52cdef7c4bab8bd675297da2"), name: 'Cisco' },
  { _id: ObjectId("52cdef7c4bab8bd675297da3"), name: 'Yahoo!' },
  { _id: ObjectId("52cdef7c4bab8bd675297da4"), name: 'Powerset' },
  { _id: ObjectId("52cdef7c4bab8bd675297dc4"), name: 'Intel' },
  { _id: ObjectId("52cdef7c4bab8bd675297def"), name: 'KickApps' },
  { _id: ObjectId("52cdef7c4bab8bd675297df7"), name: 'Bebo' },
  { _id: ObjectId("52cdef7c4bab8bd675297e0c"), name: 'LinkedIn' },
  { _id: ObjectId("52cdef7c4bab8bd675297e6f"), name: 'Sony' },
  { _id: ObjectId("52cdef7c4bab8bd675297e79"), name: 'Wikia' },
  { _id: ObjectId("52cdef7c4bab8bd675297e89"), name: 'PayPal' },
  { _id: ObjectId("52cdef7c4bab8bd675297ed4"), name: 'Yola' },
  { _id: ObjectId("52cdef7c4bab8bd675297ed6"), name: 'BitTorrent' },
  {
    _id: ObjectId("52cdef7c4bab8bd675297ee9"),
    name: 'Sun Microsystems'
  }
```

```
db.companies.find({ "relationships":
                      { "$elemMatch": { "is_past": true,
                                        "person.first_name": "Mark" } } },
                  { "name": 1 }).count()
```

```
sample_training> db.companies.find({ "relationships":
...                         { "$elemMatch": { "is_past": true,
.....                                         "person.first_name": "Mark" } } },
...                     { "name": 1 }).count()
256
sample_training>
```

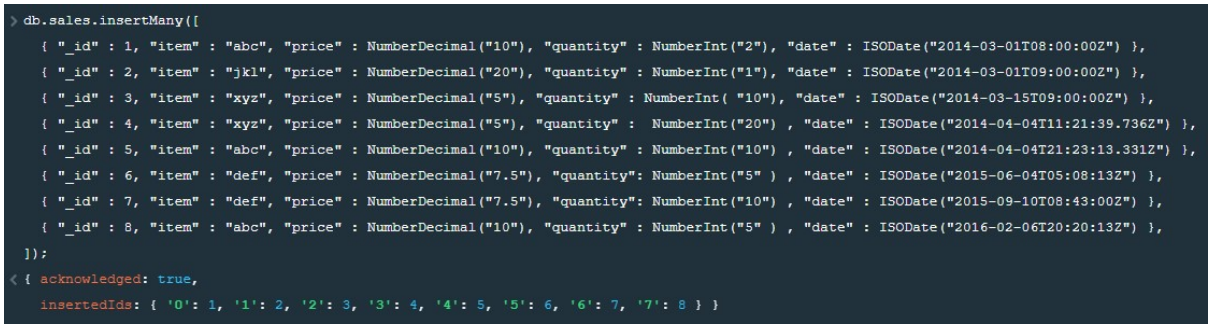
Group

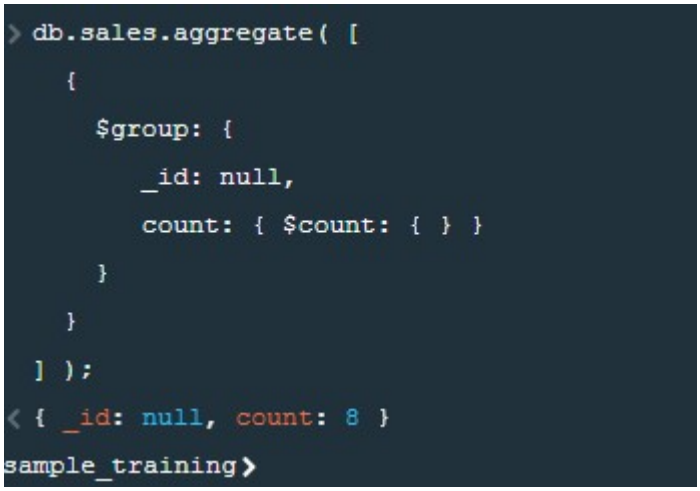
https://www.mongodb.com/docs/manual/reference/operator/aggregation/group/

```
db.sales.insertMany([
  { "_id" : 1, "item" : "abc", "price" : NumberDecimal("10"), "quantity" : NumberInt("2"), "date" : ISODate("2014-03-01T08:00:00Z") },
  { "_id" : 2, "item" : "jkl", "price" : NumberDecimal("20"), "quantity" : NumberInt("1"), "date" : ISODate("2014-03-01T09:00:00Z") },
  { "_id" : 3, "item" : "xyz", "price" : NumberDecimal("5"), "quantity" : NumberInt( "10"), "date" : ISODate("2014-03-15T09:00:00Z") },
  { "_id" : 4, "item" : "xyz", "price" : NumberDecimal("5"), "quantity" : NumberInt("20") , "date" : ISODate("2014-04-04T11:21:39.736Z") },
  { "_id" : 5, "item" : "abc", "price" : NumberDecimal("10"), "quantity" : NumberInt("10") , "date" : ISODate("2014-04-04T21:23:13.331Z") },
  { "_id" : 6, "item" : "def", "price" : NumberDecimal("7.5"), "quantity": NumberInt("5" ) , "date" : ISODate("2015-06-04T05:08:13Z") },
  { "_id" : 7, "item" : "def", "price" : NumberDecimal("7.5"), "quantity": NumberInt("10") , "date" : ISODate("2015-09-10T08:43:00Z") },
  { "_id" : 8, "item" : "abc", "price" : NumberDecimal("10"), "quantity" : NumberInt("5" ) , "date" : ISODate("2016-02-06T20:20:13Z") },

])
```

1. Count the Number of Documents in a Collection





The above query is equivalent to following SQL.

SELECT COUNT(*) AS count FROM sales

2. Retrieve Distinct Values

db.sales.aggregate([{ $group : { _id : "$item" } }])

```
> db.sales.aggregate( [ { $group : { _id : "$item" } } ] );
< { _id: 'jkl' }
  { _id: 'abc' }
  { _id: 'xyz' }
  { _id: 'def' }
sample_training>
```

3. Group by Item Having

SELECT item,

Sum((price * quantity)) AS totalSaleAmount

FROM sales

GROUP BY item

HAVING totalSaleAmount >= 100

```
db.sales.aggregate(
  [
    // First Stage
    {
      $group :
        {
          _id : "$item",
          totalSaleAmount: { $sum: { $multiply: [ "$price", "$quantity" ] } }
        }
     },
     // Second Stage
     {
       $match: { "totalSaleAmount": { $gte: 100 } }
     }
   ]

 )
```

```
{ _id: 'abc', totalSaleAmount: Decimal128("170") }
{ _id: 'xyz', totalSaleAmount: Decimal128("150") }
{ _id: 'def', totalSaleAmount: Decimal128("112.5") }
```

4. Calculate Count, Sum, and Average

SELECT date,

Sum((price * quantity)) AS totalSaleAmount,

Avg(quantity) AS averageQuantity,

Count(*) AS Count

FROM sales

GROUP BY Date(date)

ORDER BY totalSaleAmount DESC

```
db.sales.aggregate([
   // First Stage
   {
     $match : { "date": { $gte: new ISODate("2014-01-01"), $lt: new
ISODate("2015-01-01") } }
   },
   // Second Stage
   {
     $group : {
        _id : { $dateToString: { format: "%Y-%m-%d", date: "$date" } },
        totalSaleAmount: { $sum: { $multiply: [ "$price", "$quantity" ] } },
        averageQuantity: { $avg: "$quantity" },
        count: { $sum: 1 }
     }
   },
   // Third Stage
   {
     $sort : { totalSaleAmount: -1 }
   }
  ])
```

Results:

```
{ _id: '2014-04-04',
  totalSaleAmount: Decimal128("200"),
  averageQuantity: 15,
  count: 2 }
{ _id: '2014-03-15',
  totalSaleAmount: Decimal128("50"),
  averageQuantity: 10,
  count: 1 }
{ _id: '2014-03-01',
  totalSaleAmount: Decimal128("40"),
  averageQuantity: 1.5,
  count: 2 }
```

4. Group by null

SELECT Sum(price * quantity) AS totalSaleAmount,

Avg(quantity) AS averageQuantity,

Count(*) AS Count

FROM sales

```
db.sales.aggregate([
 {
   $group : {
      _id : null,
      totalSaleAmount: { $sum: { $multiply: [ "$price", "$quantity" ] } },
      averageQuantity: { $avg: "$quantity" },
      count: { $sum: 1 }
   }
 }])
```

```
> db.sales.aggregate([
    {
      $group : {
         _id : null,
         totalSaleAmount: { $sum: { $multiply: [ "$price", "$quantity" ] } },
         averageQuantity: { $avg: "$quantity" },
         count: { $sum: 1 }
      }
    }
  ]);
{ _id: null,
  totalSaleAmount: Decimal128("452.5"),
  averageQuantity: 7.875,
  count: 8 }
sample_training>
```

5. Pivot Data

Pivot and Unpivot in SQL are two relational operators that are used to convert a table expression into another. Pivot in SQL is used when we want to transfer data from row level to column level and Unpivot in SQL is used when we want to convert data from column level to row level.

db.books.insertMany([

{ "_id" : 8751, "title" : "The Banquet", "author" : "Dante", "copies" : 2 },

{ "_id" : 8752, "title" : "Divine Comedy", "author" : "Dante", "copies" : 1 },

{ "_id" : 8645, "title" : "Eclogues", "author" : "Dante", "copies" : 2 },

{ "_id" : 7000, "title" : "The Odyssey", "author" : "Homer", "copies" : 10 },

{ "_id" : 7020, "title" : "Iliad", "author" : "Homer", "copies" : 10 }

])

```
db.books.aggregate([
  { $group : { _id : "$author", booksw: { $push: "$title" } } }
])
```

```
> db.books.insertMany([
    { "_id" : 8751, "title" : "The Banquet", "author" : "Dante", "copies" : 2 },
    { "_id" : 8752, "title" : "Divine Comedy", "author" : "Dante", "copies" : 1 },
    { "_id" : 8645, "title" : "Eclogues", "author" : "Dante", "copies" : 2 },
    { "_id" : 7000, "title" : "The Odyssey", "author" : "Homer", "copies" : 10 },
    { "_id" : 7020, "title" : "Iliad", "author" : "Homer", "copies" : 10 }
  ]);
< { acknowledged: true,
    insertedIds: { '0': 8751, '1': 8752, '2': 8645, '3': 7000, '4': 7020 } }
```

```
> db.books.aggregate([
      { $group : { _id : "$author", booksw: { $push: "$title" } } }
   ]);
< { _id: 'Dante',
    booksw: [ 'The Banquet', 'Divine Comedy', 'Eclogues' ] }
  { _id: 'Homer', booksw: [ 'The Odyssey', 'Iliad' ] }
```

5.1 Group Documents by Author

Stage 1: Group the books by author.

```
>   db.books.aggregate([
      // First Stage
      {
        $group : { _id : "$author", books: { $push: "$$ROOT" } }
      }
    ]);
< { _id: 'Dante',
    books:
     [ { _id: 8751, title: 'The Banquet', author: 'Dante', copies: 2 },
       { _id: 8752, title: 'Divine Comedy', author: 'Dante', copies: 1 },
       { _id: 8645, title: 'Eclogues', author: 'Dante', copies: 2 } ] }
  { _id: 'Homer',
    books:
     [ { _id: 7000, title: 'The Odyssey', author: 'Homer', copies: 10 },
       { _id: 7020, title: 'Iliad', author: 'Homer', copies: 10 } ] }
basic>
```

$$ROOT system variable to group the entire documents by authors.

Stage 2: Add the Fields totalcopies to the each author.

```
db.books.aggregate([
  // First Stage
  {
    $group : { _id : "$author", books: { $push: "$$ROOT" } }
  },
  // Second Stage
  {
    $addFields:
      {
        totalCopies : { $sum: "$books.copies" }
      }
  }
])
```

```
> db.books.aggregate([
    // First Stage
    {
      $group : { _id : "$author", books: { $push: "$$ROOT" } }
    },
    // Second Stage
    {
      $addFields:
        {
          totalCopies : { $sum: "$books.copies" }
        }
    }
  ]);
```

```
< { _id: 'Homer',
  books:
   [ { _id: 7000, title: 'The Odyssey', author: 'Homer', copies: 10 },
     { _id: 7020, title: 'Iliad', author: 'Homer', copies: 10 } ],
  totalCopies: 20 }
{ _id: 'Dante',
  books:
   [ { _id: 8751, title: 'The Banquet', author: 'Dante', copies: 2 },
     { _id: 8752, title: 'Divine Comedy', author: 'Dante', copies: 1 },
     { _id: 8645, title: 'Eclogues', author: 'Dante', copies: 2 } ],
  totalCopies: 5 }
```

5.2 Using the Sample Training Dataset

Project only the address field value for each document, then group all documents into one document per address.country value, and count one for each document in each group.

```
db.listingsAndReviews.aggregate([ { "$project": { "address": 1, "_id": 0 }},
                        { "$group": { "_id": "$address.country" }}])
```

```
db.listingsAndReviews.aggregate([
                        { "$project": { "address": 1, "_id": 0 }},
                        { "$group": { "_id": "$address.country",
                                "count": { "$sum": 1 } } }
                       ])
```

```
sample_training> db.listingsAndReviews.aggregate([
...                                 { "$project": { "address": 1, "_id": 0 }},
...                                 { "$group": { "_id": "$address.country",
.....                                           "count": { "$sum": 1 } } }
...                                ])
[
  { _id: 'Portugal', count: 555 },
  { _id: 'Spain', count: 633 },
  { _id: 'Canada', count: 649 },
  { _id: 'China', count: 19 },
  { _id: 'Hong Kong', count: 600 },
  { _id: 'Turkey', count: 661 },
  { _id: 'United States', count: 1222 },
  { _id: 'Brazil', count: 606 },
  { _id: 'Australia', count: 610 }
]
sample_training>
```

Sorting

Value 1 indicates Ascending Order
Value -1 indicates Descending Order

Sorting: Ascending Order

db.zips.find().sort({ "pop": 1 }).limit(10)

```
sample_training> db.zips.find().sort({ "pop": 1 }).limit(10)
[
  {
    _id: ObjectId("5c8eccc1caa187d17ca6ef8f"),
    city: 'NAKNEK',
    zip: '99633',
    loc: { y: 58.885699, x: 156.705405 },
    pop: 0,
    state: 'AK'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca6ef9c"),
    city: 'RUSSIAN MISSION',
    zip: '99657',
    loc: { y: 61.591302, x: 161.558413 },
    pop: 0,
    state: 'AK'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca6efed"),
    city: 'SELAWIK',
    zip: '99770',
    loc: { y: 65.713537, x: 158.534287 },
    pop: 0,
    state: 'AK'
  },
```

Descending Order:

db.zips.find().sort({ "pop": -1 }).limit(10)

```
sample_training> db.zips.find().sort({ "pop": -1 }).limit(10)
[
  {
    _id: ObjectId("5c8eccc1caa187d17ca7044d"),
    city: 'CHICAGO',
    zip: '60623',
    loc: { y: 41.849015, x: 87.7157 },
    pop: 112047,
    state: 'IL'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca7307f"),
    city: 'BROOKLYN',
    zip: '11226',
    loc: { y: 40.646694, x: 73.956985 },
    pop: 111396,
    state: 'NY'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca72fa0"),
    city: 'NEW YORK',
    zip: '10021',
    loc: { y: 40.768476, x: 73.958805 },
```

Multiple Fields with different Order

Order By Population Ascending and City Descending.

db.zips.find().sort({ "pop": 1, "city": -1 })

```
sample_training> db.zips.find().sort({ "pop": 1, "city": -1 })
[
  {
    _id: ObjectId("5c8eccc1caa187d17ca756c9"),
    city: 'WALLOPS ISLAND',
    zip: '23337',
    loc: { y: 37.827338, x: 75.506503 },
    pop: 0,
    state: 'VA'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca6f948"),
    city: 'VINTON',
    zip: '96135',
    loc: { y: 39.720719, x: 120.204994 },
    pop: 0,
    state: 'CA'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca754b2"),
```

Sorting with Limit

Population in Descending, limit the data size to 10

```
sample_training> db.zips.find().sort({ "pop": -1 }).limit(10)
[
  {
    _id: ObjectId("5c8eccc1caa187d17ca7044d"),
    city: 'CHICAGO',
    zip: '60623',
    loc: { y: 41.849015, x: 87.7157 },
    pop: 112047,
    state: 'IL'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca7307f"),
    city: 'BROOKLYN',
    zip: '11226',
    loc: { y: 40.646694, x: 73.956985 },
    pop: 111396,
    state: 'NY'
  }
```

Indexing

1. **Find Query. Consider the following query take time to execute**

db.trips.find({ "birth year": 1989 })
db.trips.find({ "start station id": 476 }).sort({ "birth year": 1 })

```
sample_training> db.trips.find({ "start station id": 476 }).sort( { "birth year": 1 } )
[
  {
    _id: ObjectId("572bb8222b288919b68ad677"),
    tripduration: 211,
    'start station id': 476,
    'start station name': 'E 31 St & 3 Ave',
    'end station id': 167,
    'end station name': 'E 39 St & 3 Ave',
    bikeid: 23613,
    usertype: 'Subscriber',
    'birth year': 1950,
    gender: 2,
    'start station location': { type: 'Point', coordinates: [ -73.97966069, 40.74394314 ] },
    'end station location': { type: 'Point', coordinates: [ -73.97604882, 40.7489006 ] },
    'start time': ISODate("2016-01-01T14:53:06.000Z"),
```

2. **Create the index for birth year**

   db.trips.createIndex({ "birth year": 1 })

```
sample_training> db.trips.createIndex({ "birth year": 1 })
birth year_1
sample_training>
```

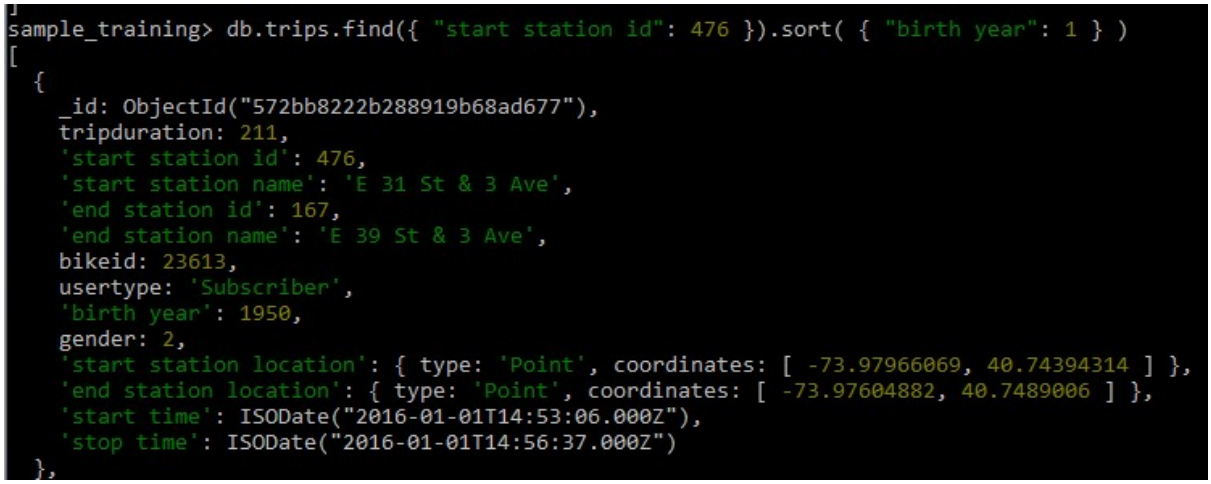
3. **Create the index for start station id and birth year**

   db.trips.createIndex({ "start station id": 1, "birth year": 1 })

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

sample_training> db.trips.createIndex({ "start station id": 1, "birth year": 1 })
start station id_1_birth year_1
sample_training>
```

4. **Run the same Query again.**
   The run time will be less now. But it can be observable only on large database.



Reference:

https://www.mongodb.com/basics/create-database
https://university.mongodb.com/courses/M001/about