

# MongoDB Lab



# Download Resources



- <https://goo.gl/GvjLD3>

# MongoDB

Explore Datasets using Mongo Compass

# Connect to a MongoDB Atlas Server

- Setup Mongo Compass and connect to Atlas Server using the following information:
- **Hostname:** cluster0-shard-00-00-jxeqq.mongodb.net
- **Username:** m001-student
- **Password:** m001-mongodb-basics
- **Replica Set Name:** Cluster0-shard-0
- **Read Preference:** Primary Preferred

# Explore Data Types From Schema Screen

Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.10 Enterprise

**video.movies**

Documents Aggregations **Schema** Explain Plan Indexes Validation

FILTER OPTIONS ANALYZE RESET ...

Query returned 1029440 documents. This report is based on a sample of 1000 documents (0.10%). ⓘ

Field	Type	Description
<code>_id</code>	objectid	ObjectID
<code>cast</code>	array	Cast array
<code>director</code>	string	Director string
<code>genre</code>	string	Genre string
<code>imdbId</code>	string	IMDb ID string
<code>language</code>	string	Language string

**Schema** ANALYZE

**Array lengths**  
min: 1 average: 3.4 max: 4

**imdbId**

- tt2254131 tt1651920 tt2877352 tt1943008 tt2483898 tt0778634 tt0075609 tt0443423 tt0188229 tt3560102
- tt2399497 tt1342408 tt0084938 tt0418148 tt0000952 tt0045917 tt1031226 tt0423489 tt0248254 tt3543636

58% 29%

first: 2017-03-12 19:07:22 last: 2017-03-12 19:09:19



# Documents: Aggregate Value Types

Cluster0-shard-0 REPLICA SET (3 NODES) MongoDB 3.6.10 Enterprise

**100YWeatherSmall.data**

Documents Aggregations **Schema** Explain Plan Indexes Validation

**DOCUMENTS 250.0k** TOTAL SIZE 403.0MB AVG. SIZE 1.7KB **INDEXES 5** TOTAL SIZE 11.2MB AVG. SIZE 2.2MB

**FILTER** OPTIONS ANALYZE RESET ...

Query returned 250000 documents. This report is based on a sample of 1000 documents (0.40%). ⓘ

**\_id**  
objectid

S M T W T F S 0:00 6:00 12:00 18:00 23:00  
first: 2015-05-13 19:44:12 last: 2015-05-13 19:44:26

**airTemperature**  
document

Document with 2 nested fields.

**quality**  
string

1 9

**value**  
double

93% 46.5% 0% min: -34.8 max: 999.9

**atmosphericPressureChange**  
document undefined

Document with 3 nested fields.

**atmosphericPressureObservation**  
undefined

Document with 2 nested fields.

**callLetters**

5%



# Documents: Fields with Complex Values

Cluster0-shard-0 REPLICA SET 3 NODES      MongoDB 3.6.10 Enterprise

**100YWeatherSmall.data**

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER    OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE    Displaying documents 1 - 20 of 250000 < > C

```

_id: ObjectId("5553a98ce4b02cf7150dee2a")
st: "x+59500+001500"
ts: 1984-01-01 00:00:00.000
> position: Object
  elevation: 9999
  callLetters: "PLAT"
  qualityControlProcess: "V020"
  dataSource: "4"
  type: "FM-13"
> airTemperature: Object
> dewPoint: Object
> pressure: Object
> wind: Object
> visibility: Object
> skyCondition: Object
< sections: Array
  0: "AG1"
  1: "MD1" ←
  2: "OA1"
> precipitationEstimatedObservation: Object
> atmosphericPressureChange: Object

```

```

_id: ObjectId("5553a98ce4b02cf7150dee23")
st: "x+61300-015800"
ts: 1984-01-01 00:00:00.000
> position: Object
  elevation: 9999
  callLetters: "TFLZ"
  qualityControlProcess: "V020"
  dataSource: "4"
  type: "FM-13"
> airTemperature: Object
> dewPoint: Object
> pressure: Object
> wind: Object
> visibility: Object
> skyCondition: Object
< sections: Array
> precipitationEstimatedObservation: Object
> pastWeatherObservationManual: Array
> skyCoverLayer: Array
  < 0: Object
    < coverage: Object
      value: "09"
      quality: "1"
    < baseHeight: Object
      value: 99999
      quality: "9"
    < cloudType: Object
      value: "99"
      quality: "9"
  > skyConditionObservation: Object
  > presentWeatherObservationManual: Array

```

```

_id: ObjectId("5553a98ce4b02cf7150dee20")
st: "x+53100+002200"
ts: 1984-01-01 00:00:00.000
> position: Object
  type: "Point"
  coordinates: Array
    0: 2.2
    1: 53.1
  elevation: 9999
  callLetters: "PLAT"
  qualityControlProcess: "V020"
  dataSource: "4"
  type: "FM-13"
> airTemperature: Object
> dewPoint: Object
> pressure: Object
> wind: Object
> visibility: Object
> skyCondition: Object
< sections: Array
> precipitationEstimatedObservation: Object
> atmosphericPressureChange: Object

```

+

# Geospatial Fields

- Geospatial is not a field type per se, however, MongoDB is able to use fields like Position and do various calculations
  - Various shapes
  - Distance
  - Within Radius

position  
coordinates



# Filtering Collections with Queries

**citibike.trips**

Documents Aggregations **Schema** Explain Plan Indexes Validate

**FILTER**  { 'end station name': 'MacDougal St & Prince St'}

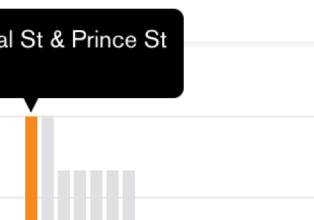
Query returned **1990273** documents. This report is based on a sample of **1000** documents (0.05%). **i**



MacDougal St & Prince St  
3%

end station name

string



# Filtering Collections with Queries

**citibike.trips**

Documents Aggregations **Schema** Explain Plan Indexes Validation

**FILTER** `{'birth year': {$gte: 1985,$lt: 1990}}` **OPTIONS** **ANALYZE** **RESET**

Query returned **1990273** documents. This report is based on a sample of **1000** documents (0.05%). ⓘ

Field	Type	Description
<code>_id</code>	objectid	Object ID
<code>bikeid</code>	int32	Bike ID
<code>birth year</code>	string	Birth Year
<code>end station id</code>	string	End Station ID

`_id`  
objectid

`bikeid`  
int32

`birth year`  
string

`end station id`  
string

DOCUMENTS **2.0m** TOTAL SIZE 834.8MB AVG. SIZE 440B | INDEXES **13** TOTAL SIZE 122.5MB AVG. 9.4

# Filtering Collections with Queries

**citibike.trips**

Documents Aggregations **Schema** Explain Plan Indexes Validation

**FILTER** `{'birth year': {$gte: 1985,$lt: 1990},'end station name': 'MacDougal St & Prince St'}` **OPTIONS** **ANALYZE**

Query returned **1990273** documents. This report is based on a sample of **1000** documents (0.05%). i

+ Drag to Build a Query

© Mapbox © OpenStreetMap Improve this

**end station name**

string

# Filtering on Geospatial Coordinates

**ships.shipwrecks**

Documents Aggregations **Schema** Explain Plan Indexes Validation

DOCUMENTS 11.1k TOTAL SIZE 3.6MB AVG. SIZE 344B | INDEXES 2 TOTAL SIZE 272.0KB AVG. SIZE 136.0KB

**FILTER** {coordinates: {\$geoWithin: { \$centerSphere: [ [-64.80419786699281, 18.347148998736586], 0.009076137892705297 ] }}}

Query returned 11095 documents. This report is based on a sample of 1000 documents (9.01%).

**\_id**  
objectid

S M T W T F S 0:00 6:00 12:00 18:00 23:00  
Inserted: 2016-07-20 12:33:39

**chart**  
string

0% 1.5% 3%  
0% 1.5% 3%

**coordinates**  
coordinates

San Juan, Arecibo, Carolina, Luquillo, Ceiba, Culebra, Ponce, Guayama, Saba Bank National Park

mapbox © Mapbox © OpenStreetMap Improve this map

# MongoDB

Introduction to CRUD  
(Create, Read, Update, Delete)

# MongoDB

Create

# Install Enterprise MongoDB

- Follow the instructions here for your OS version
  - <https://docs.mongodb.com/manual/installation/>
- We use SSL so make sure you get version 3.6 or higher
- Once installed, issue the following command:

```
mongo "mongodb://cluster0-shard-00-00-jxeqq.mongodb.net:27017,cluster0-shard-00-01-jxeqq.mongodb.net:27017,cluster0-shard-00-02-jxeqq.mongodb.net:27017/test?replicaSet=Cluster0-shard-0" --authenticationDatabase admin --ssl --username m001-student --password m001-mongodb-basics
```

- Test the following commands:

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> use movies
MongoDB Enterprise Cluster0-shard-0:PRIMARY> show collections
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.movies.find().pretty()
```

# Sign-up for free Atlas Account



- Sign up for a Atlas account and create your first cluster using the free-tier servers



### Create a cluster

Choose your cloud provider, region, and specs.

[Build a Cluster](#)

Once your cluster is up and running, live migrate an existing MongoDB database into Atlas with our [Live Migration Service](#).

★ recommended region ⓘ

NORTH AMERICA	EUROPE	AUSTRALIA
N. Virginia (us-east-1) ★ <small>FREE TIER AVAILABLE</small>	Stockholm (eu-north-1) ★	Sydney (ap-southeast-2) ★
Ohio (us-east-2) ★	Ireland (eu-west-1) ★	Tokyo (ap-northeast-1) ★
N. California (us-west-1)	London (eu-west-2) ★	Seoul (ap-northeast-2)
Oregon (us-west-2) ★	Paris (eu-west-3) ★	Singapore (ap-southeast-1) ★ <small>FREE TIER AVAILABLE</small>
Montreal (ca-central-1)	Frankfurt (eu-central-1) ★ <small>FREE TIER AVAILABLE</small>	Mumbai (ap-south-1) <small>FREE TIER AVAILABLE</small>

**Cluster Tier**

MO (Shared RAM, 512 MB Storage) Encrypted

Base hourly rate is for a MongoDB replica set with 3 data bearing servers.

Shared Sandbox Clusters for getting started with MongoDB

Tier	RAM	Storage	vCPU	Base Price
MO	Shared	512 MB	Shared	<b>Free forever</b>
M0 clusters are best for getting started, and are not suitable for production environments.				
100 max connections   Low network performance   100 max databases   500 max collections				
M2	Shared	2 GB	Shared	\$9 / MONTH
M5	Shared	5 GB	Shared	\$25 / MONTH

**Connect to Henry001**

✓ Setup connection security ✓ Choose a connection method Connect

If you are not running MongoDB 4.0.6+ with TLS/SSL, click below to download the latest Mongo Shell

[Windows](#) [Mac OS X](#) Other Operating Systems ▾

**Download**

Or fetch with this URL:  
[https://downloads.mongodb.org/osx/mongodb-shell-osx-x86\\_64-4.0.6.tgz](https://downloads.mongodb.org/osx/mongodb-shell-osx-x86_64-4.0.6.tgz) [COPY](#)

Or install with Homebrew (includes TLS/SSL support):  
**brew install mongodb --with-openssl** [COPY](#)

Connect via the Mongo Shell  
[View detailed instructions](#)

Short SRV connection string (shell 3.6+) [COPY](#) Standard connection string (shell 3.4+)

```
mongo "mongodb+srv://henry001-a7ojk.mongodb.net/test" --username hspark
```

# Login to your Cluster and Import Data

- Download the sample data script from the following URL
  - <https://goo.gl/3VpyuN>
- Load the javascript to create the video database

```
HenryMac:M001 hwpark$ mongo "mongodb+srv://henry001-a7ojk.mongodb.net/test" --username hwpark
...Successfully connected to henry001-shard-00-00-a7ojk.mongodb.net:27017
MongoDB Enterprise Henry001-shard-0:PRIMARY> show dbs
admin 0.000GB
local 4.374GB
MongoDB Enterprise Henry001-shard-0:PRIMARY> load("loadMovieDetailsDataset.js")
true
MongoDB Enterprise Henry001-shard-0:PRIMARY> show dbs
admin 0.000GB
local 4.374GB
video 0.000GB
MongoDB Enterprise Henry001-shard-0:PRIMARY> use video
switched to db video
MongoDB Enterprise Henry001-shard-0:PRIMARY> show collections
movieDetails
```

# Connect to your Cluster from Compass

- Obtain the Compass connection string by following the steps below and then copy the string to the clipboard
- Start Compass
  - Compass will automatically detect the connection string in your clipboard and fill in the necessary details

The screenshot shows the MongoDB Compass interface. On the left, under the 'Clusters' tab, there is a 'Clusters' section with tabs for 'Overview' (which is selected) and 'Security'. Below this is a search bar labeled 'Find a cluster...' and a 'SANDBOX' section containing a cluster named 'Henry001' (Version 4.0.6). A red box highlights the 'CONNECT' button in the 'SANDBOX' section. To the right, a modal window titled 'Connect to Henry001' is open. It has a 'Setup connection security' step completed, a 'Choose a connection method' step, and a 'Connect' button. Below these are two options: 'Connect with the Mongo Shell' (Mongo Shell with TLS/SSL support is required) and 'Connect Your Application' (Get a connection string and view driver connection examples). At the bottom of this modal is another 'CONNECT' button, which is also highlighted with a red box. To the right of the modal, a separate panel titled 'Copy the URI Connection String' provides instructions and two buttons: 'I am using Compass 1.12 or later' (highlighted with a green border) and 'I am using Compass 1.11 or earlier'. It shows a connection string: `mongodb+srv://hwpark:<PASSWORD>@henry001-a7ojk.mongodb.net/admin`. A red box highlights the 'COPY' button next to the string. Below this panel, text says 'Replace **PASSWORD** with the password for the *hwpark* user.' and 'When you open Compass, it should detect the URI string from your clipboard and auto-populate the form.'

# Create New Collection and Insert Documents

- Create a new collection “moviesScratch” from Compass
- Add the following documents from Compass
  - {title : “Rocky”, year : 1976, imdb : “tt0075148”}
  - {title : “Creed”, year : 2015, imdb : “tt3076658”}
- Add the following document from Mongo Shell
  - We will use insertOne api

```
MongoDB Enterprise Henry001-shard-0:PRIMARY> show collections
```

```
movieDetails
```

```
movieScratch
```

```
MongoDB Enterprise Henry001-shard-0:PRIMARY> db.moviesScratch.insertOne({title:  
"Star Trek II: The Wrath of Khan", year: 1982, imdb: "tt0084726"})
```

```
{
```

```
    "acknowledged" : true,
```

```
    "insertedId" : ObjectId("5c716f37c5d874198072740a")
```

```
}
```

# insertMany – Insert Multiple Documents

- There are two types of insertMany() operations
  - Ordered – Stops when an error is encountered
  - Unordered – Keeps going with the rest even after an error

```
db.moviesScratch.insertMany(
  [
    {
      "_id" : "tt0084726",
      "title" : "Star Trek II: The Wrath of Khan",
      "year" : 1982,
      "type" : "movie"
    },
    {
      "_id" : "tt0796366",
      "title" : "Star Trek",
      "year" : 2009,
      "type" : "movie"
    },
    {
      "_id" : "tt0084726",
      "title" : "Star Trek II: The Wrath of Khan",
      "year" : 1982,
      "type" : "movie"
    },
    {
      "_id" : "tt1408101",
      "title" : "Star Trek Into Darkness",
      "year" : 2013,
      "type" : "movie"
    },
    {
      "_id" : "tt0117731",
      "title" : "Star Trek: First Contact",
      "year" : 1996,
      "type" : "movie"
    }
  ]
);
```

```
db.moviesScratch.insertMany(
  [
    {
      "_id" : "tt0084726",
      "title" : "Star Trek II: The Wrath of Khan",
      "year" : 1982,
      "type" : "movie"
    },
    {
      "_id" : "tt0796366",
      "title" : "Star Trek",
      "year" : 2009,
      "type" : "movie"
    },
    {
      "_id" : "tt0084726",
      "title" : "Star Trek II: The Wrath of Khan",
      "year" : 1982,
      "type" : "movie"
    },
    {
      "_id" : "tt1408101",
      "title" : "Star Trek Into Darkness",
      "year" : 2013,
      "type" : "movie"
    },
    {
      "_id" : "tt0117731",
      "title" : "Star Trek: First Contact",
      "year" : 1996,
      "type" : "movie"
    }
  ],
  {
    "ordered": false
  }
);
```

# MongoDB

Read

# Using `find({query})` to Read from Dataset

- Go to Compass and filter the `video.movies` collection and observe the filter condition
  - Choose to view movies with `mpaaRating` of PG-13
  - Add the condition that the year of the movie is 2009
- Now from the shell, use the `find()` command to display the same query from above

```
db.movies.find({mpaaRating: 'PG-13', year: 2009}).pretty()
```

# Query on Nested Fields

- Go to Compass and choose the 100YWeatherSmall.data collection and observe the filter for nested fields
  - Filter on the wind -> type -> "C" or "N"
  - Observe the dot notation and quotes around the key

```
{'wind.type': 'N'}
```

- This type, choose wind -> direction -> 290
  - Observe the filter notation

```
{'wind.direction.angle': 290}
```

- Now go to Mongo Shell and try the above queries

```
use 100YWeatherSmall
db.data.find({"wind.direction.angle": 290}).pretty()
```

## Try this Query On Your Own

- Explore the movieDetails collection that you loaded into your Atlas sandbox cluster and then issue a query to answer the following question.
  - How many movies in the movieDetails collection have exactly 2 award wins and 2 award nominations?
  - How about rated PG movies with 10 award nominations?
    - You will find the [count\(\)](#) method useful in answering this question using the mongo shell

# Querying Array Fields

- Try the following query from movies collection

```
use video
db.movies.find({cast: ["Jeff Bridges", "Tim Robbins"]}).pretty()
```

- Notice that this selects a document with an exact match
  - Even the order of the cast must match
  - Even the actual movie does not match because there are extra casts

```
db.movies.find({title: "Arlington Road"}).pretty()
```

- What if I just want to choose a movie where Jeff Bridges is a cast

```
db.movies.find({cast: "Jeff Bridges"}).pretty()
```

- Or perhaps in a movie where he is the star

```
db.movies.find({'cast.0': "Jeff Bridges"}).pretty()
```

## Try this Query On Your Own

- Explore the movieDetails collection that you loaded into your Atlas sandbox cluster and then issue a query to answer the following question.
  - How many documents list just two writers: "Ethan Coen" and "Joel Coen", in that order?
  - How many movies are Family oriented movies?
  - How many movies are listed as Western as the second entry?
    - You will find the [count\(\)](#) method useful in answering this question using the mongo shell.

# Introduction to Cursors

- The `find()` operator returns a cursor
- In the shell, 20 documents are printed automatically
- From the shell, we can type `it` to get the next 20

```
"actors" : [
    "Peter Cushing",
    "Bernard Cribbins",
    "Ray Brooks",
    "Andrew Keir"
],
"plot" : "The Daleks' fiendish plot in 2150 against Earth and its people is foiled when Dr. Who and friends arrive from the 20th century and
figure it out.",
"poster" : "http://ia.media-imdb.com/images/M/MV5BMTg0OTc0MDc0M15BMl5BanBnXkFtZTgwMTg5ODYxMTE@._V1_SX300.jpg",
"imdb" : {
    "id" : "tt0060278",
    "rating" : 6,
    "votes" : 2236
},
"awards" : {
    "wins" : 0,
    "nominations" : 0,
    "text" : ""
},
"type" : "movie"
}
Type "it" for more
MongoDB Enterprise >
```

- Select only certain fields to be returned from a `find()` query
- Second argument to the `find()` operator
  - Use 1 if you want to explicitly select that field
  - Use 0 if you want to explicitly remove that field
- Try the following command from the class cluster

```
db.movies.find({genre: "Action, Adventure"}, {title: 1})
```

- Notice that the primary field `_id` is included automatically
  - How might we explicitly exclude this field

```
db.movies.find({genre: "Action, Adventure"}, {title: 1, _id: 0})
```

- Try explicitly removing some of the fields

# MongoDB

## Update

# Dynamic and Flexible Schema

- One of the core advantages of MongoDB is that the schema is dynamic and flexible
  - Not all documents are required to have the same fields
  - This means that we do not need to have fields with NULL values
- Let's look at movieDetails – notice that many of the documents are missing fields such as awards and posters
- From movieDetails, locate the movie title “The Martian”

```
use video
db.movieDetails.find({title: "The Martian"}).pretty()
```

- Notice that this document does not have a link to a poster picture

# Updating a document with updateOne()

- We will use the updateOne() operator to add a poster field to the movie – The Martian
- The update operator allows us to specify how fields should be modified in matching documents
- The first parameter specifies the filter
- The second parameter specifies the type of update
- Here we will use the \$set operator
  - Think of it as write or replace
  - If the field does not exist, it will write
  - If it exists, \$set will overwrite with the new data

```
db.movieDetails.updateOne({  
    title: "The Martian"  
}, {  
    $set: {  
        poster: "https://www.imdb.com/title/tt3659388/mediaviewer/rm1391324160"  
    }  
})
```

# MongoDB Update Operators - \$inc

- Please review the update operators
  - <https://docs.mongodb.com/manual/reference/operator/update/>
- Let's try one - \$inc
  - While we can also use \$set to just change, it may be prone to errors

```
db.movieDetails.updateOne({  
    title: "The Martian"  
, {  
    $inc: {  
        "tomato.reviews": 3,  
        "tomato.userReviews": 25  
    }  
})
```

# MongoDB Update Operators - \$push

- Let's try another one that works on Arrays

```
let reviewText1 = [
  "The Martian could have been a sad drama film, instead it was a",
  "hilarious film with a little bit of drama added to it. The Martian is what ",
  "everybody wants from a space adventure. Ridley Scott can still make great ",
  "movies and this is one of the best"
].join()

db.movieDetails.updateOne({
  title: "The Martian"
}, {
  $push: {
    reviews: {
      rating: 4.5,
      date: ISODate("2016-01-12T09:00:00Z"),
      reviewer: "Spencer H.",
      text: reviewText1
    }
  }
})
```

# Updating Multiple Documents with updateMany()



- With updateMany(), you can update multiple documents that match your filter criteris
- Observe that many of the documents contain a null rated field
- Let's remove them

```
db.movieDetails.updateMany({  
    rated: null  
}, {  
    $unset: {  
        rated: ""  
    }  
})
```

```
db.movieDetails.updateMany({  
    poster: null  
}, {  
    $unset: {  
        poster: ""  
    }  
})
```

- You try: Remove all empty poster fields

# Replace Document with replaceOne()

- Use findOne() to get the document into memory
  - find() returns a cursor, findOne() returns the first matching document
- Update the document in memory
- Use replaceOne() to substitute with my in-memory document

```
detailDoc = db.movieDetails.findOne({"imdb.id": "tt4368814"});
detailDoc.poster;
detailDoc.poster =
"https://www.imdb.com/title/tt4368814/mediaviewer/rm2926634240";
detailDoc.genres;
detailDoc.genres.push("Documentary");
db.movieDetails.replaceOne({
    "imdb.id": detailDoc.imdb.id
},
detailDoc
);
```

# MongoDB

## Delete

# MongoDB deleteOne() operation

- First copy the loadReviewDataset.js script to the directory where you are running your shell for your personal cluster
- Run the script with `load("loadReviewDataset.js")`
- This will create a review collection in the video database
- Run the following to delete a document:

```
db.reviews.deleteOne({_id: ObjectId("5c754f064ad41406ab794dfa")})  
db.reviews.deleteMany({reviewer_id: 759723314})
```