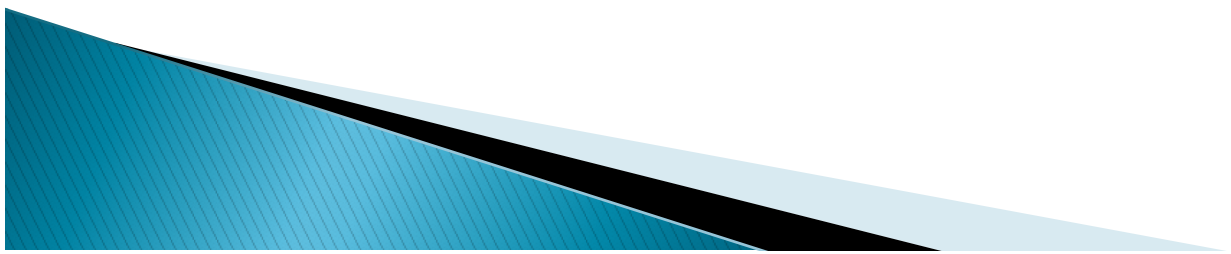
CSCI-UA.60-1 Database Design and Implementation

An Introduction to MongoDB

Prof Deena Engel
Department of Computer Science
deena.engel@nyu.edu

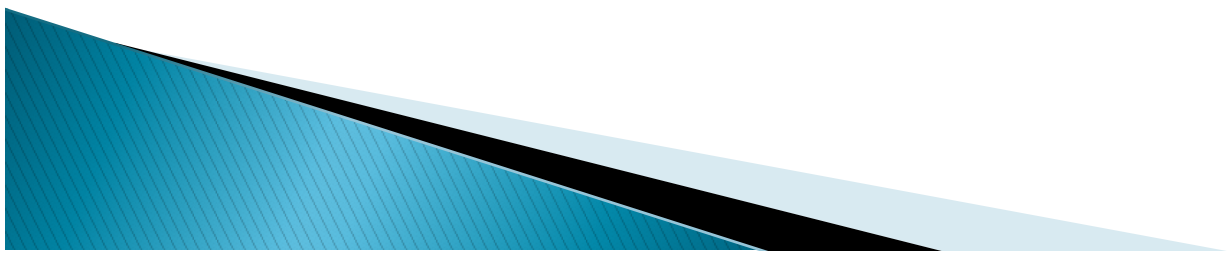
Getting started!

- Principles and concepts to remember with MongoDB:
  - A document is the basic unit of data
  - A collection can be thought of … as a dynamic table
  - One instance of MongoDB can host multiple independent databases (and then each database can hold its own collections)
  - Every document has a special key (_id) which is unique within its collection
  - MongoDB comes with a simple but powerful JavaScript shell (but you do not need to know JavaScript for the work we will do in this class).



Getting started!

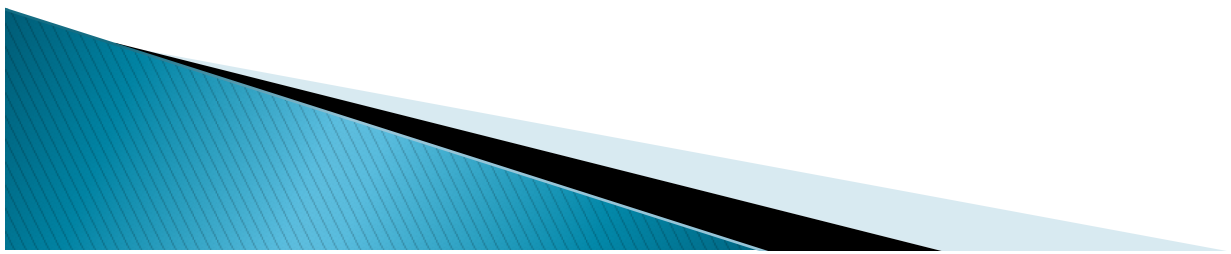
- MongoDB comes with a simple but powerful JavaScript shell (but you do not need to know JavaScript for the work we will do in this class).
- See https://docs.mongodb.com/manual/reference/method/ for built-in JavaScript methods.



“Hello, world!”

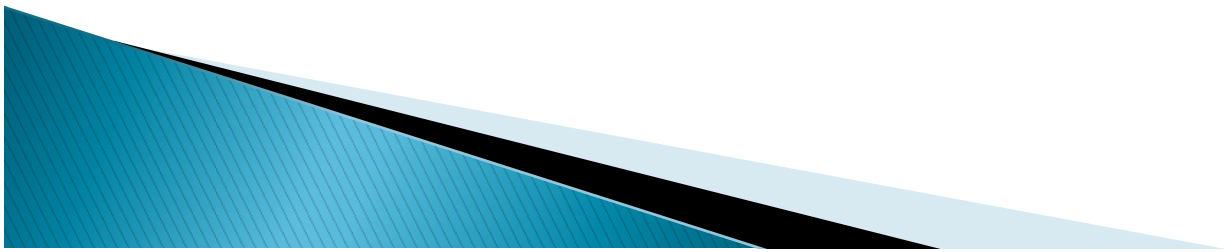
{ “greeting” : “Hello, world!” }

This simple document contains a single key (“greeting”) with the value of “Hello, world!”.



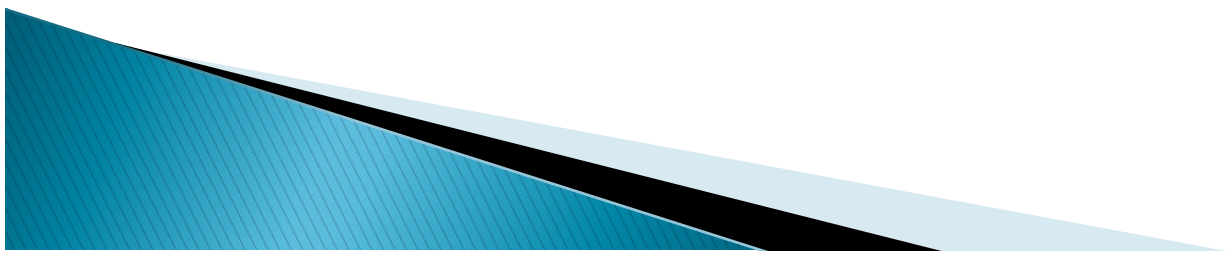
Rules about keys

- Keys must not contain the \0 (or null) character (because that is used to signify the end of a key).
- The period (.) and $ characters have special properties and should be considered *reserved*.
- Note that a single document may not contain duplicate keys.



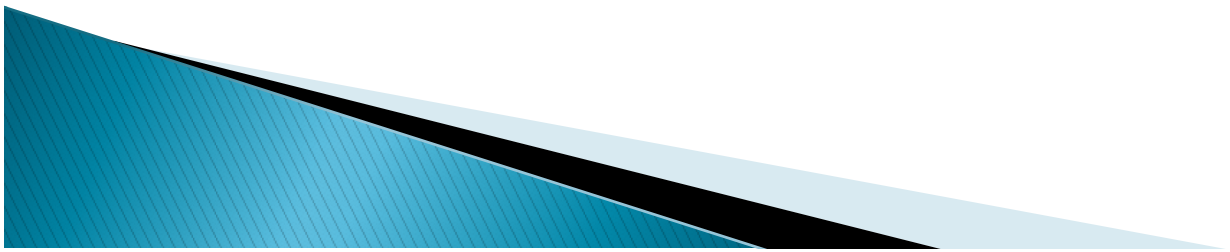
Collections

- Collections have *dynamic schemas* …
- However, in reality, it is important to keep different kinds of documents in different collections.
- This will allow for …
  - Faster searches
  - … and other considerations.



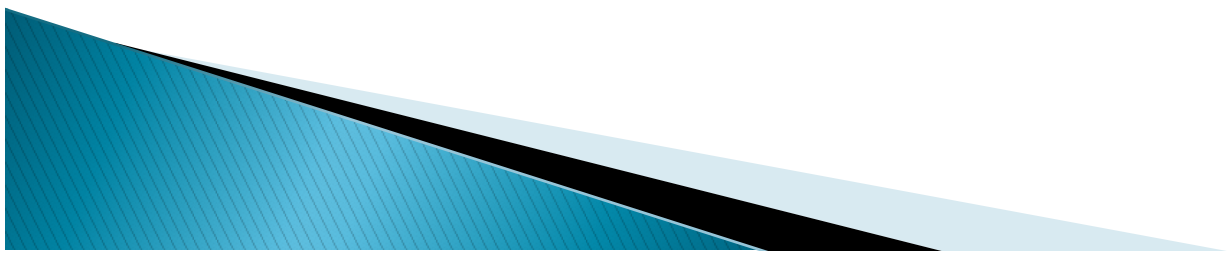
Collection Names

- Collection names can use any valid UTF-8 string.
  - UTF-8 can represent any character in the Unicode standard and is backwardsly compatible with ASCII
    - http://www.w3schools.com/tags/ref_charactersets.asp
- Collection names may not contain the null character (\0) and should not use $ in the name



Databases

- Collections are grouped into databases.
- Unlike with MySQL, you may run one database through i6.cims.nyu.edu within your account.
- Your account name and database name are the same.



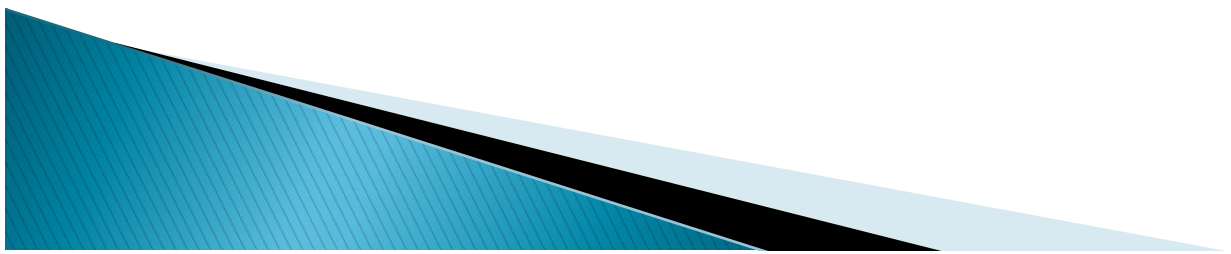
Some commands to get you started …

- Once you have opened your mongoDB session:
- To enter your database:

```
use <database name>
```

- Note: The following does not work for us, as everyone has one database:

```
show dbs
```



Class notes

- As with our other databases, I will post both class transcripts (records of the sessions) and the data files that we use to the course home pages under the “Class Notes” tab.

Getting help!

- To get documentation on the methods you can use with your database:

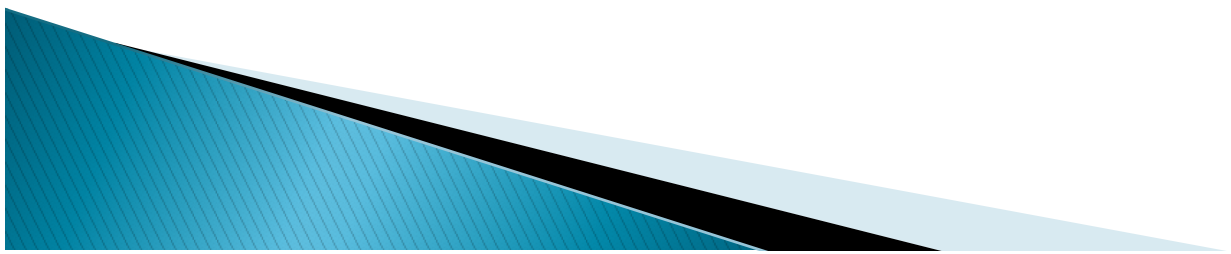
```
db.help()
```

- To get documentation on the methods you can use with your collection:

```
db.users.help()
```

- For example:

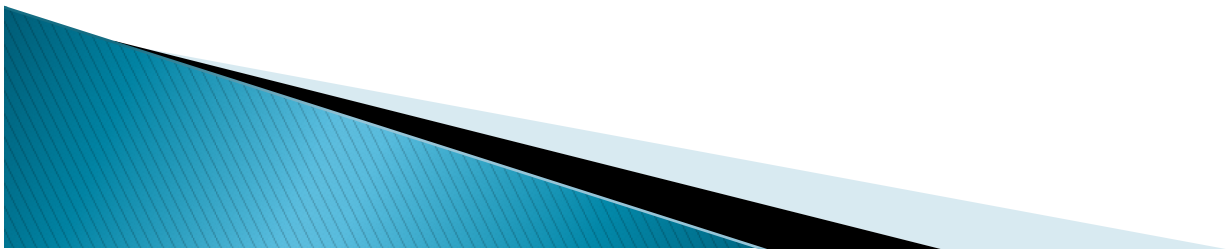
```
> db.users.help()
```



Working with external data files

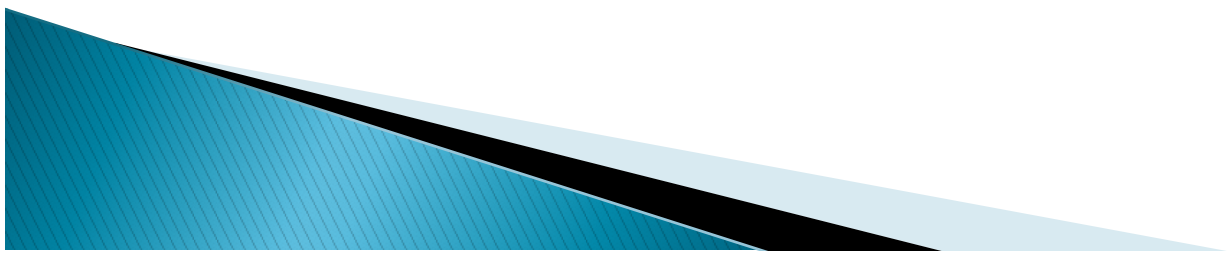
- We will import data using examples with both .json and .csv files
- For example, to import a .json file called users.json into a collection called users within a database called test through a port numbered 1815:

```
mongoimport --db your_database_name --collection
your_collection_name --host class-
mongodb.cims.nyu.edu --username your_username --
password your_password --type json --file
your_json_file_name
```



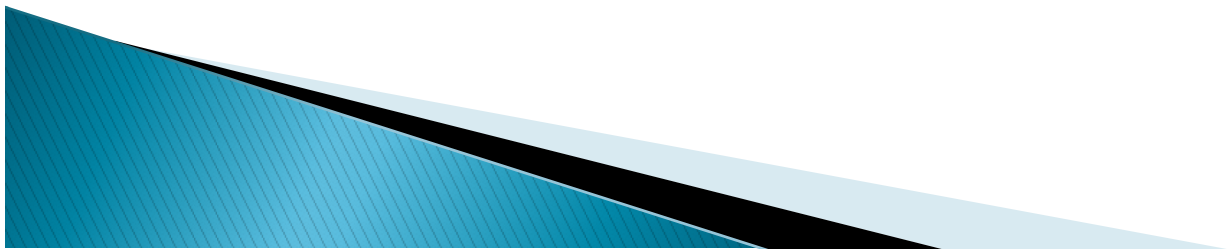
find() method

- The find() method takes two arguments:
  1. criteria
  2. projection
- Some examples are:
  - `db.books.find()`
    - – *no criteria, no selection!*
  - `db.books.find( {"AUTHOR":"Tolstoy, "Leo" })`
    - will find books by Leo Tolstoy
  - `db.books.find( { "PRICE": { $gt: 25 } } )`
    - will find books over $25



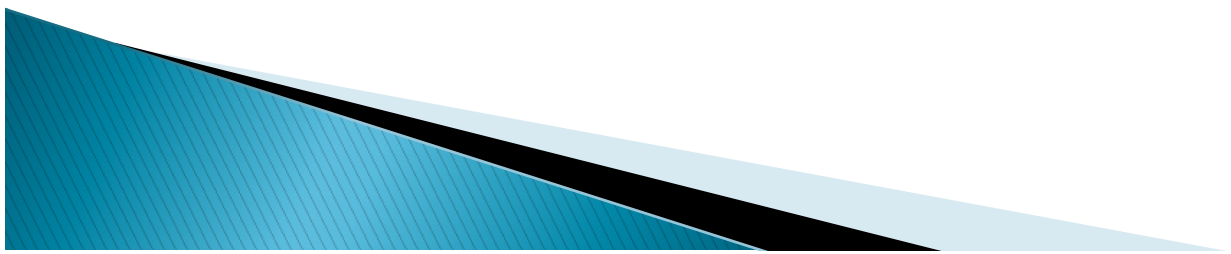
count()

- Note that these two statements are equivalent:
  - `db.books.find().count()`
  - `db.books.count()`
- To find all of the books published by Penguin in our collection:
  - `db.books.find({"EDITION":"Penguin"}).count()`
- You can also include the criteria in *count():*
  - `db.books.count({"EDITION":"Penguin"})`



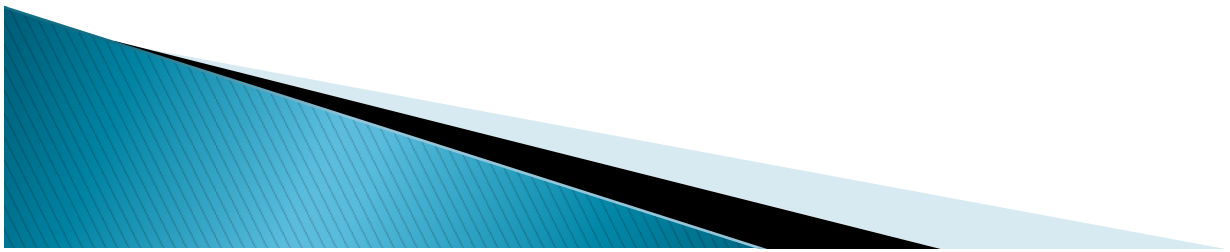
mongoDB Query operators

- $gt greater than query.
- $gte equal to or greater than
- $in Matches any of the values that exist in an array specified in the query.
- $lt less than
- $lte less than or equal to
- $ne not equal query.
- $nin Matches values that **are not** in an array specified to the query.



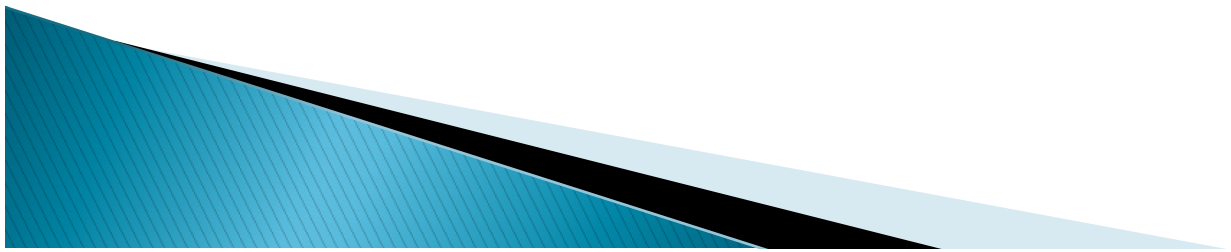
Using $or

- The mongo $or takes an array of conditions as its argument.
  - http://docs.mongodb.org/manual/reference/operator/query/or/



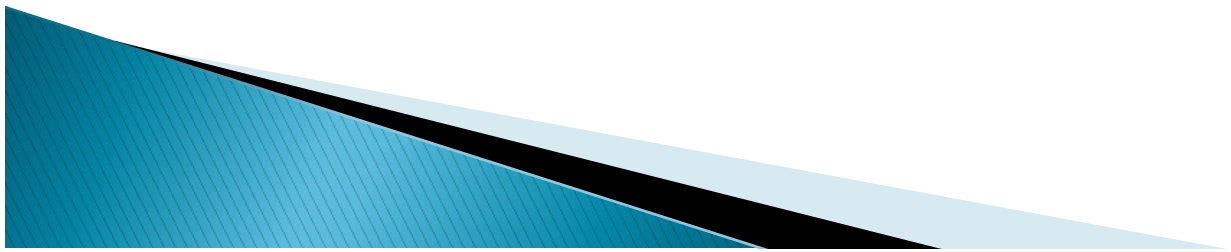
$in and $nin

- To use $in, create an array and then test the values of the data:
  - `db.books.find({ "YEAR_WRITTEN":{ $in: [ 1865, 1925 ]}})`
- $nin works in the same way, but to exclude with “not in”.



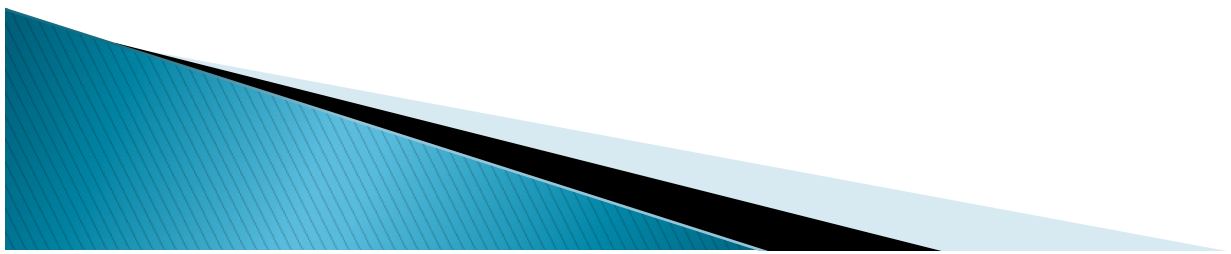
MongoDB's Aggregation Framework

- MongoDB offers a powerful aggregation framework.
- These queries are somewhat similar to the GROUP BY statements that we used in MySQL.
- The aggregation framework is only available in versions 2.4+
- See http://docs.mongodb.org/manual/aggregation/



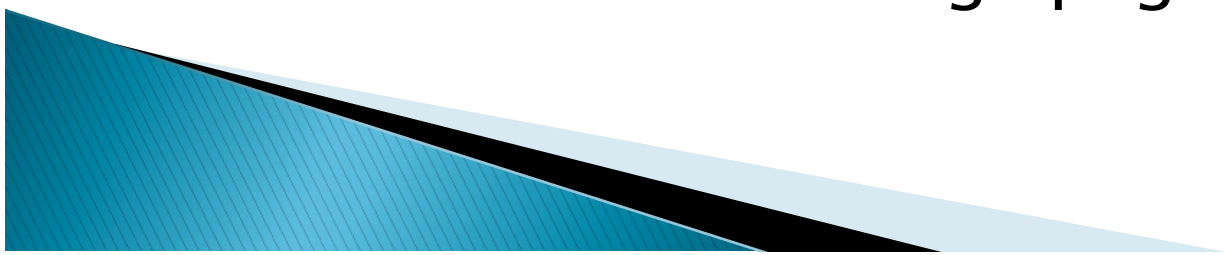
Class Notes

- Examples will be included in class discussion along with the data files so that you can practice as well.
- Some of the aggregation framework $group operators are:
  - $min
  - $max
  - $sum
  - $avg
- There are many more; see reference at http://docs.mongodb.org/manual/reference/operator/aggregation/



GeoSpatial Data in MongoDB

- MongoDB natively supports geo-spatial data (e.g. latitutde & longitude) and offers many ways to work with these data.
- This is an example that corresponds to our discussion in class about a case where MongoDB requires a built-in index to work with the data.
- In our brief example, we will use the “2d” index. MongoDB offers a “2dsphere” index as well.
- For example, here is documentation on $near: http://docs.mongodb.org/manual/reference/operator/query/near/
- The data under discussion in class are available on the course Readings pages.



Sample queries using geo-spatial data

- Latitude and longitude are stored in an array of 2 elements. For example, the Courant Institute location would be represented as follows:

  [40.7296,-73.995305]

- Assuming the location of a place in our “wifi” collection is represented by the values associated with the key “Shape” in a document, one could create the following index:

  db.wifi.ensureIndex({ "Shape":"2d"})

Finding the wifi locations closest to the Courant Institute!

- Using the $near operator:

```
db.wifi.find( { "Shape" :
                { $near : [ 40.7296,-73.995305 ]      } } )
```

- Using the $near operator and only displaying the name and address of the resulting data:

```
db.wifi.find( { "Shape" :  { $near : [ 40.7296,-73.995305 ]   } },
 {_id:0,"NAME":1,"ADDRESS":1} )
```

- Using the $near operator and only displaying the name and address of the three closest wifi hotspots to the Courant Institute:

```
db.wifi.find( { "Shape" :  { $near : [ 40.7296,-73.995305 ]   } },
  {_id:0,"NAME":1,"ADDRESS":1} ).limit(3)
```

- We will look at more examples in class!