Northeastern University

DS 5110 – Lecture 7
MongoDB

Roi Yehoshua

Agenda

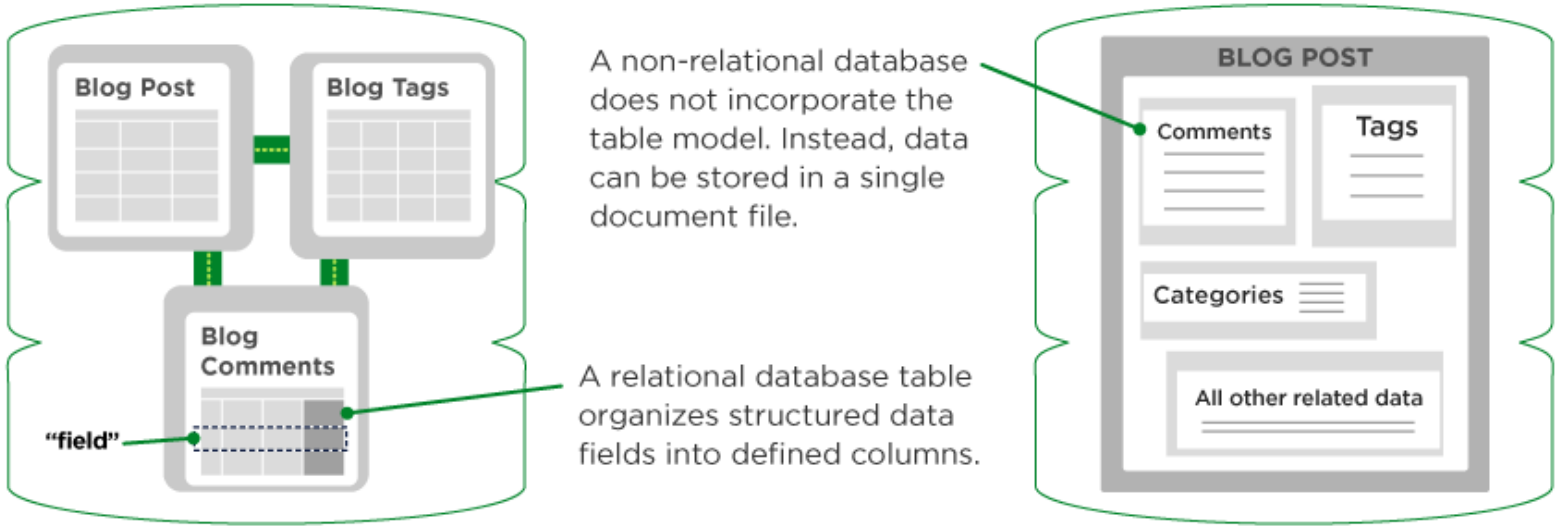
Northeastern University

- NoSQL databases
- MongoDB
- Querying documents
- Cursors
- The aggregation framework
- CRUD operations
- PyMongo

NoSQL Databases

Northeastern University

- **NoSQL** databases (aka "not only SQL") are non-tabular databases that store data differently than relational tables.
- Provide flexible schemas and scale easily with large amounts of data and high loads
- NoSQL data models allow related data to be nested within a single data structure



SQL vs. NoSQL

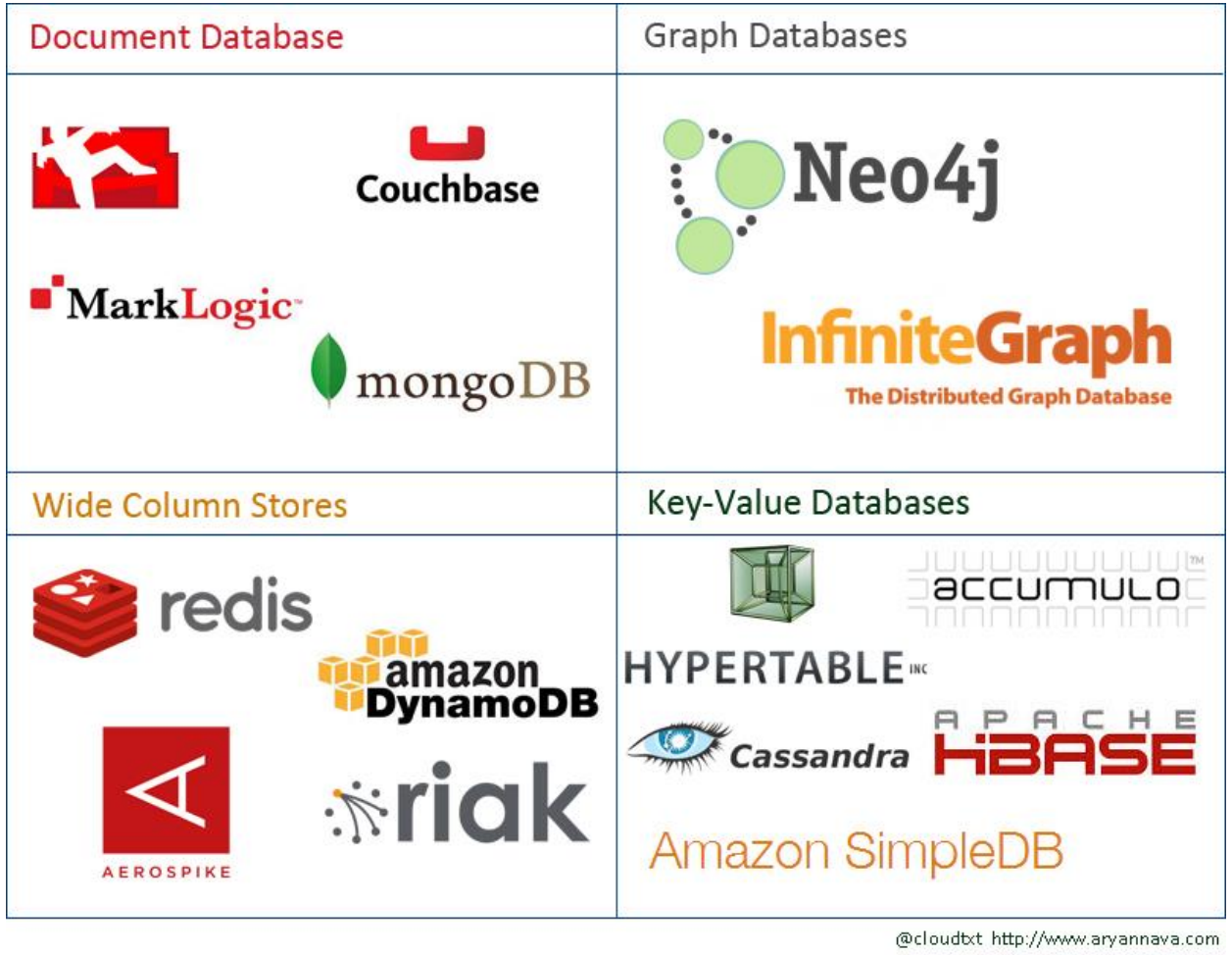
Northeastern University

	NoSQL	SQL
Model	Non-relational Data is stored in JSON documents, key/value pairs, wide column stores, or graphs	Relational Data is stored in tables
Schema	Dynamic or flexible schemas The schema can be changed on the fly	Strict schema Schema changes requires taking the entire database offline
Data	Various data types – structured, semi-structured and unstructured data	Only structured data
Scaling	Horizontal scaling Can add more machines or cloud instances	Vertical scaling Need to make the server more powerful
Transactions	Doesn't support the ACID properties	Supports ACID transactions
Development Model	Open-source	Mix of open-source (e.g., MySQL, Postgres), and closed source (e.g., Oracle, SQL server)

NoSQL Database Types

Northeastern University

Store data in documents similar to JSON



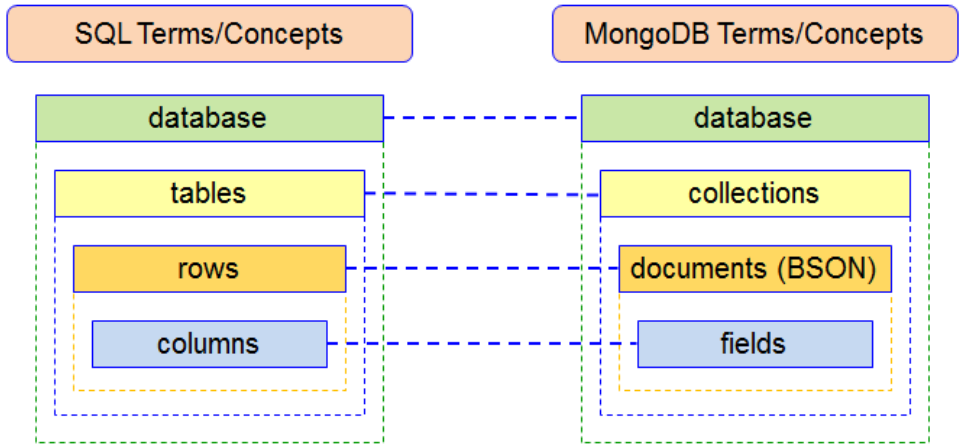
Store data in nodes and edges

Store data tables, rows, and dynamic columns

Each item contains keys and values

MongoDB

- An open-source, cross-platform **document database**
- Data is stored in JSON-like documents
- Provides high performance, high availability, and automatic scaling
- Provides a rich query language
- Supports multiple storage engines, including in-memory engines
- Offers a cloud database called MongoDB Atlas



Documents

Northeastern University

- MongoDB stores data records as BSON (Binary JSON) documents
- Documents are composed of field-and-value pairs:

```
{
  name: "sue",                      <-- field: value
  age: 26,                          <-- field: value
  status: "A",                      <-- field: value
  groups: [ "news", "sports" ]      <-- field: value
}
```

  - The keys (field names) are strings and must be unique
  - The value of a field can be any of the BSON data types, including other documents and arrays
- MongoDB is type-sensitive and case-sensitive
  - e.g., {"count": 5} is different from {"Count": "5"}

BSON

Northeastern University

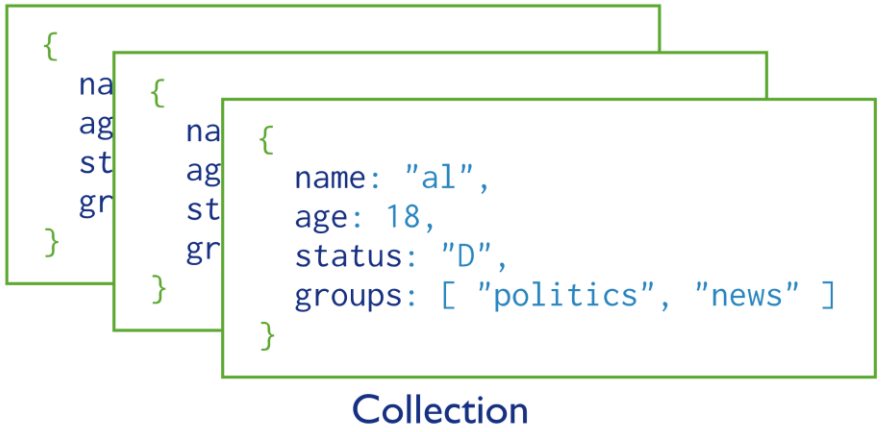
- BSON (Binary JSON) is a binary representation of JSON documents
- Designed to represent data efficiently, and allow fast encoding/decoding
- BSON supports more data types than JSON:

Type	Number	Alias	Notes
Double	1	“double”	
String	2	“string”	
Object	3	“object”	
Array	4	“array”	
Binary Data	5	“binData”	
Undefined	6	“undefined”	Deprecated
ObjectId	7	“objectId”	
Boolean	8	“bool”	
Date	9	“date”	
Null	10	“null”	
Regular Expression	11	“regex”	
DBPointer	12	“dbPointer”	Deprecated
JavaScript	13	“javascript”	
Symbol	14	“symbol”	Deprecated
Javascript (with scope)	15	“javascriptWithScope”	
32-bit integer	16	“int”	
Timestamp	17	“timestamp”	
64-bit Integer	18	“long”	
Decimal128	19	“decimal	New in Version 3.4
Min Key	-1	“minKey”	
Max Key	127	“maxKey”	

Collections

Northeastern University

- MongoDB stores documents in collections

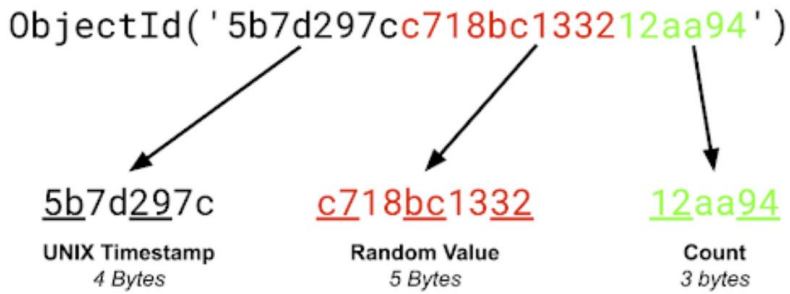


- Documents in the same collections don't need to have the same schema
  - i.e., they can have different fields and data types
- Databases and collections in MongoDB are created **lazily**
  - i.e., they are created when the first document is inserted into them

ObjectIds

Northeastern University

- Every document in MongoDB requires a unique **_id** field that acts as a primary key
- If not specified, MongoDB generates it automatically by calling ObjectId()
- This function returns a globally unique id across different machines and processes

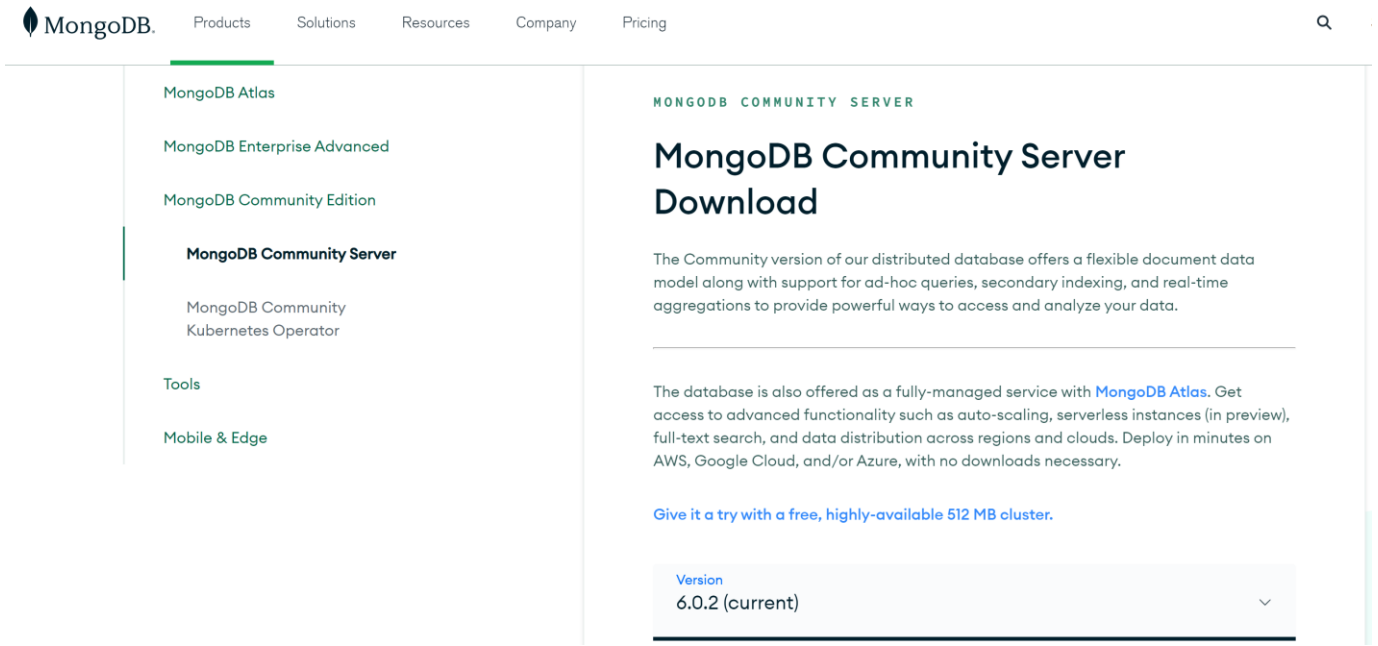


- Consists of 3 fields
  - Timestamp of the ObjectId's creation time, measured in seconds since the Unix epoch
  - Random value generated once per process, unique to machine and process
  - An incrementing counter, initialized to a random value

Installing MongoDB Community Edition

Northeastern University

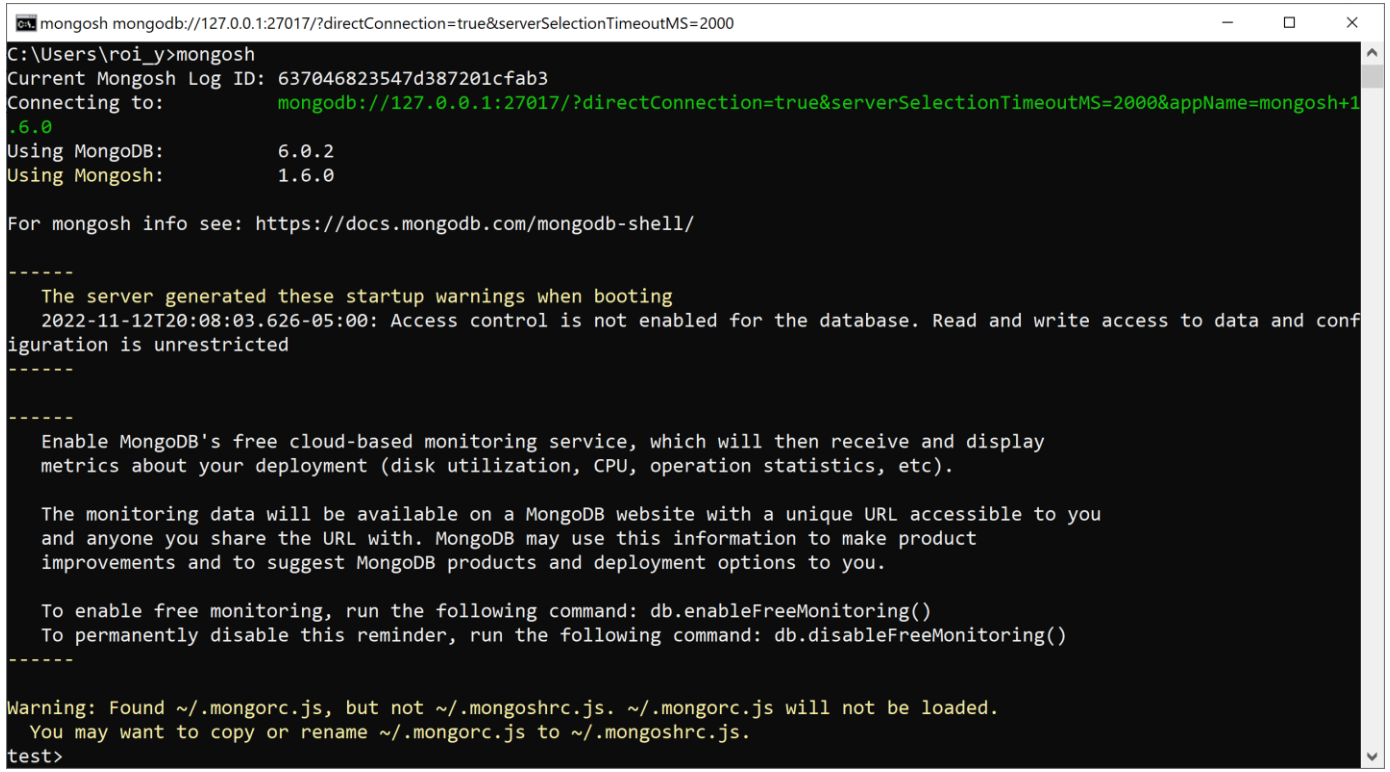
- Download MongoDB Community Server from https://www.mongodb.com/try/download/community
- For Mac OS, follow the instructions in https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/



MongoDB Shell (mongosh)

Northeastern University

- mongosh is an interactive JavaScript interface to MongoDB
- Download the shell from the MongoDB Download Center
- After installation run **mongosh** from the command line:



MongoDB Shell (mongosh)

Northeastern University

- You can run in the shell any JavaScript command:
- For example:

```
test> x = 10
10
test> x / 5
2
test> "Hello world!".replace('world', 'MongoDB')
Hello MongoDB!
test> for (let i = 0; i < 5; i++) {
... print(i);
... }
0
1
2
3
4

test> _
```

Basic Commands

Northeastern University

- The **db** variable points the database you are currently connected to
  - This variable is the primary access point to the MongoDB server
  - On startup, the shell connects to the test database

```
test> db
test
```

- To see all the available databases type **show dbs**:
  - By default, your data files are stored in C:\Program Files\MongoDB\Server\6.0\data

```
test> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
```

- The **use** command allows you change the database you are connected to:

```
test> use admin
switched to db admin
admin> _
```

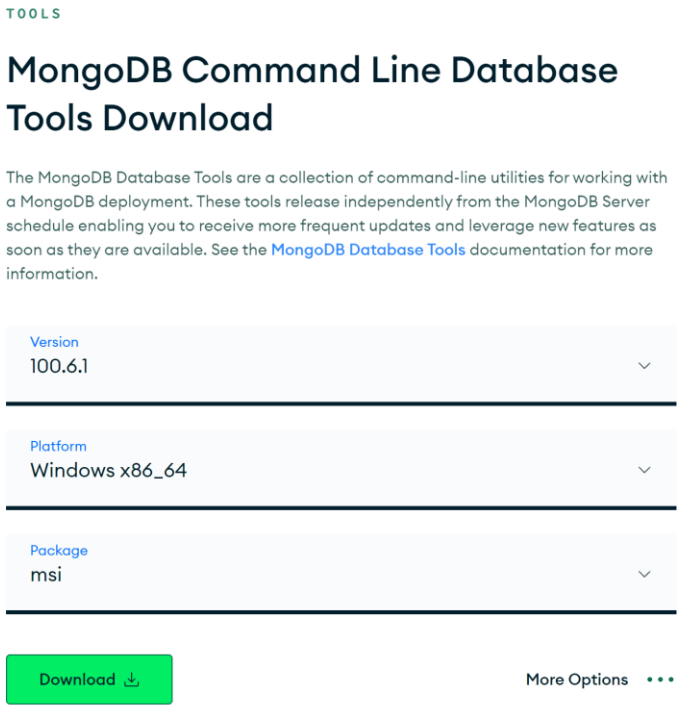
- Use **show collections** to see all the available collections in the current database

```
admin> show collections
system.version
```

MongoDB Command Line Database Tools

Northeastern University

- A collection of command-line utilities for working with a MongoDB deployment
- Download them from https://www.mongodb.com/try/download/database-tools
- Choose package type msi instead of zip



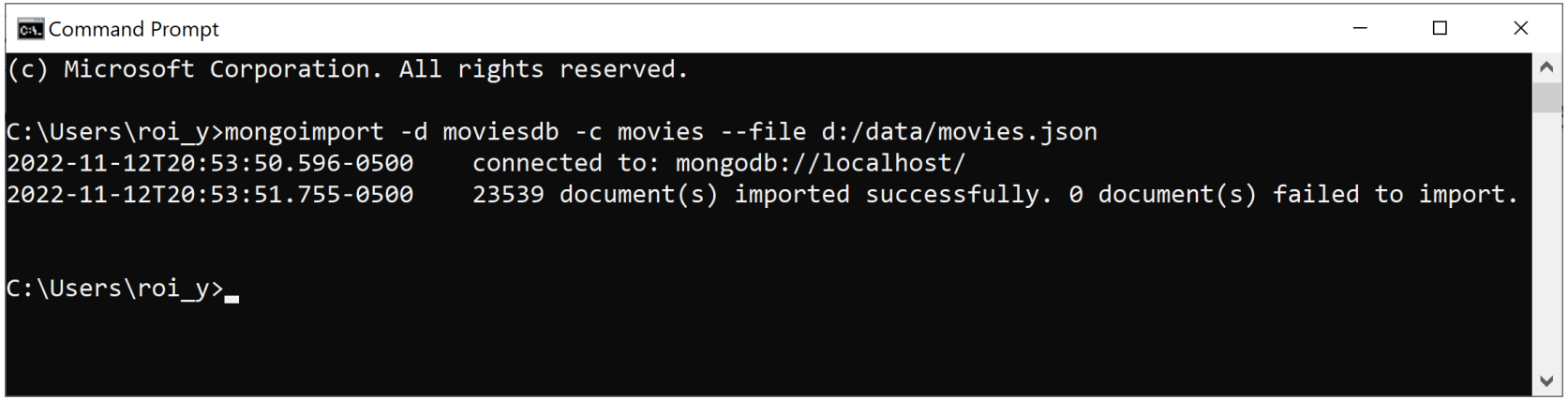
mongoimport

Northeastern University

- A command line utility that can import data from JSON, CSV and TSV files
- Run mongoimport from the system command line (not the mongo shell)

```
mongoimport -d DB_NAME -c COLLECTION_NAME --file FILE-TO-IMPORT
```

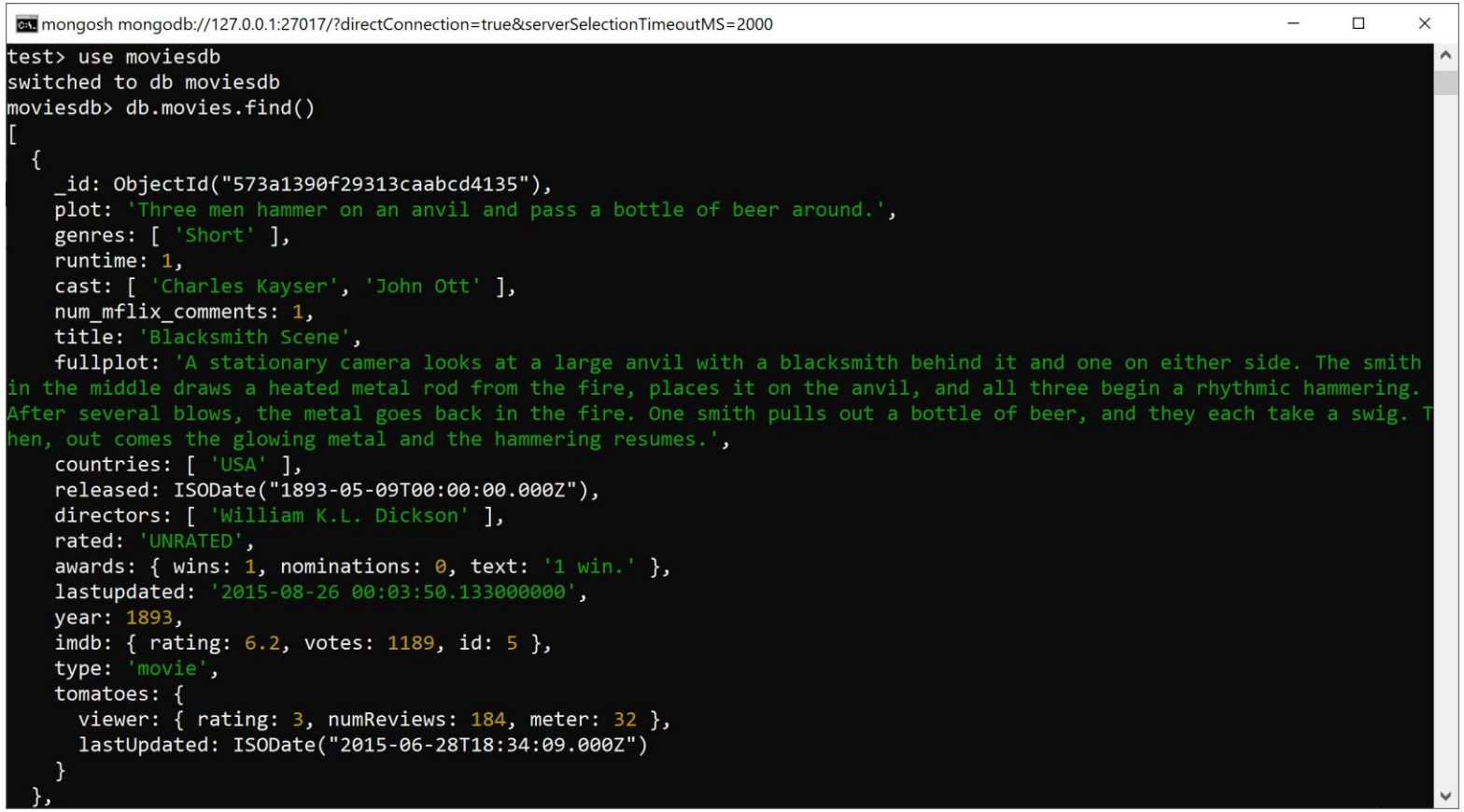
- For example, to import the file movies.json into a movies collection under moviesdb



Displaying All Documents

Northeastern University

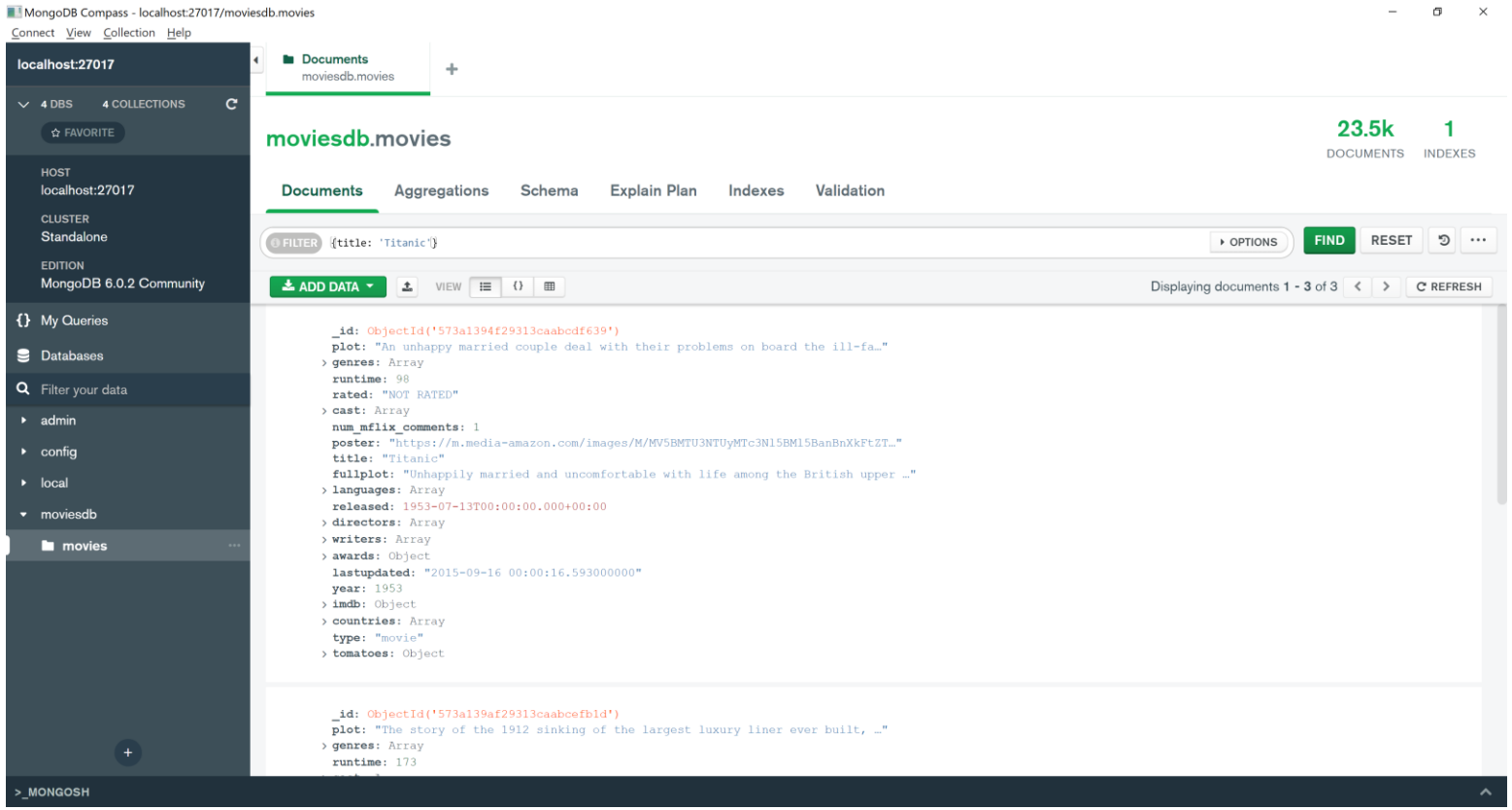
- The command db.[collection].find() will print all the documents to the console
  - 10 documents at a time



MongoDB Compass

Northeastern University

- Compass is an interactive tool for querying and analyzing your MongoDB data
  - Installed as part of your MongoDB community edition



Querying Documents

Northeastern University

- **db.collection.find**(query, projection, options) selects documents in a collection
  - returns a **cursor** to the selected documents that match the query criteria

Parameter	Type	Description
query	document	Specifies query selection criteria using query operators. To return all documents in a collection, omit this parameter or pass an empty document ({}).
projection	document	Specifies the fields to return in the documents that match the query filter. To return all fields in the matching documents, omit this parameter.
options	document	Specifies additional options for the query (optional)

```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

Exact Matching

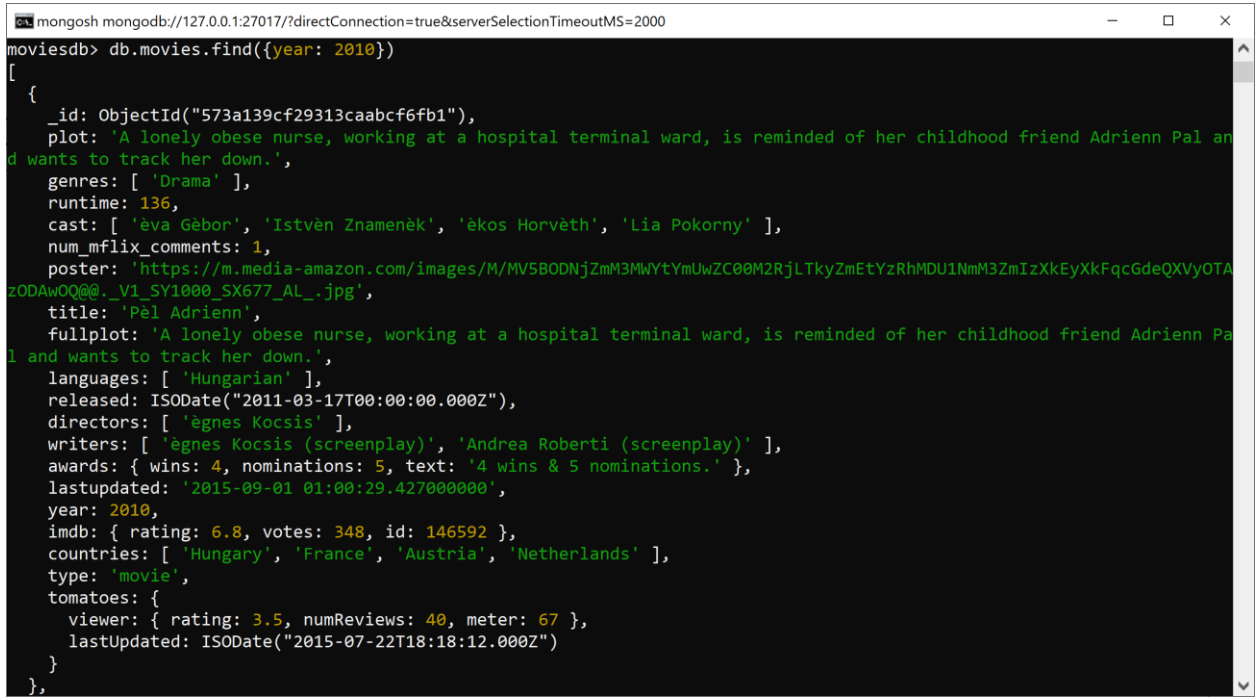
Northeastern University

- To specify equality conditions, use <field>:<value> expressions in the query filter:

```
{<field1>: <value1>, <field2>: <value2>, ... }
```

  - Field names don't have to be surrounded by single or double quotes (but they can)

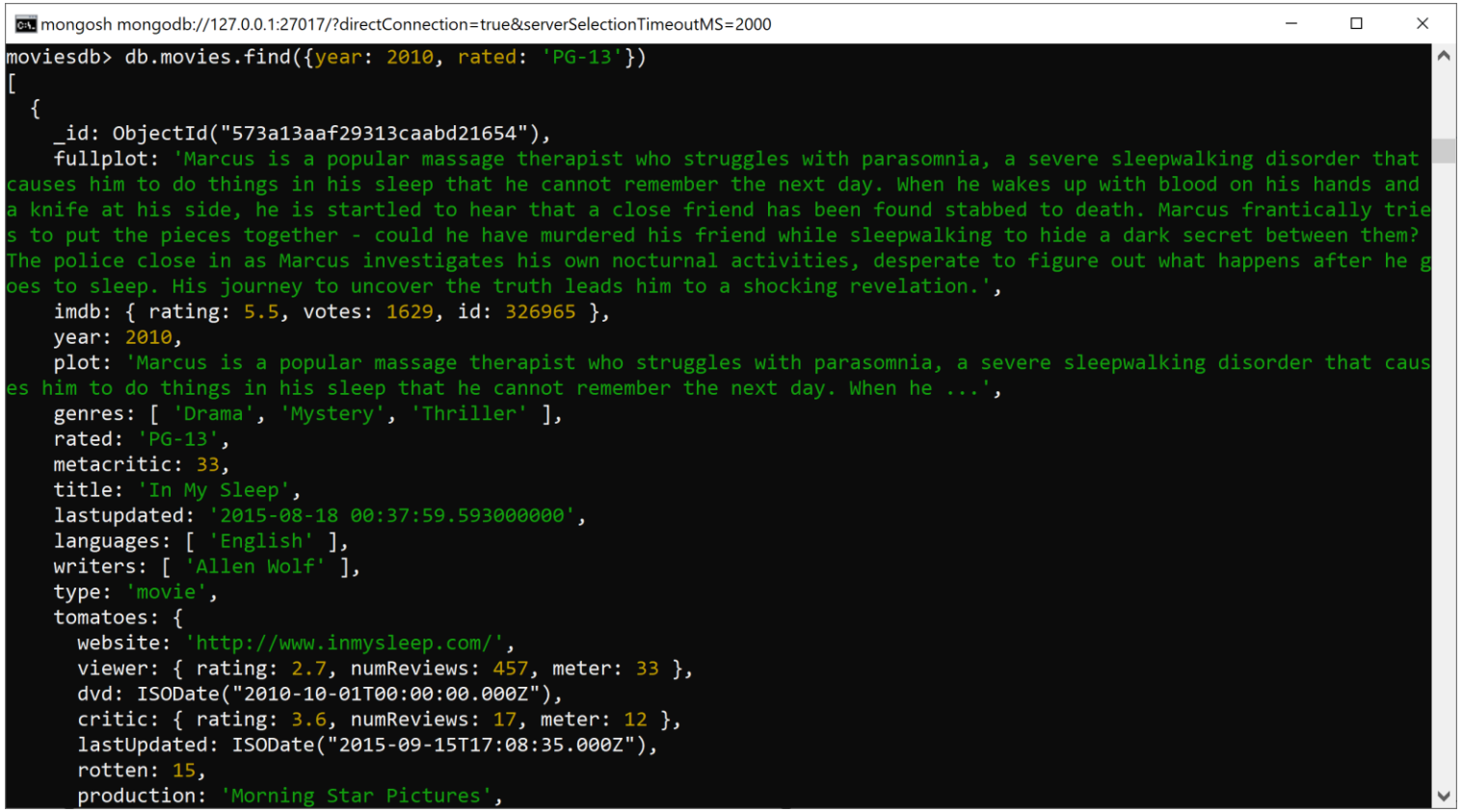
- Example: find all the movies that were released in the year 2010



Exact Matching

Northeastern University

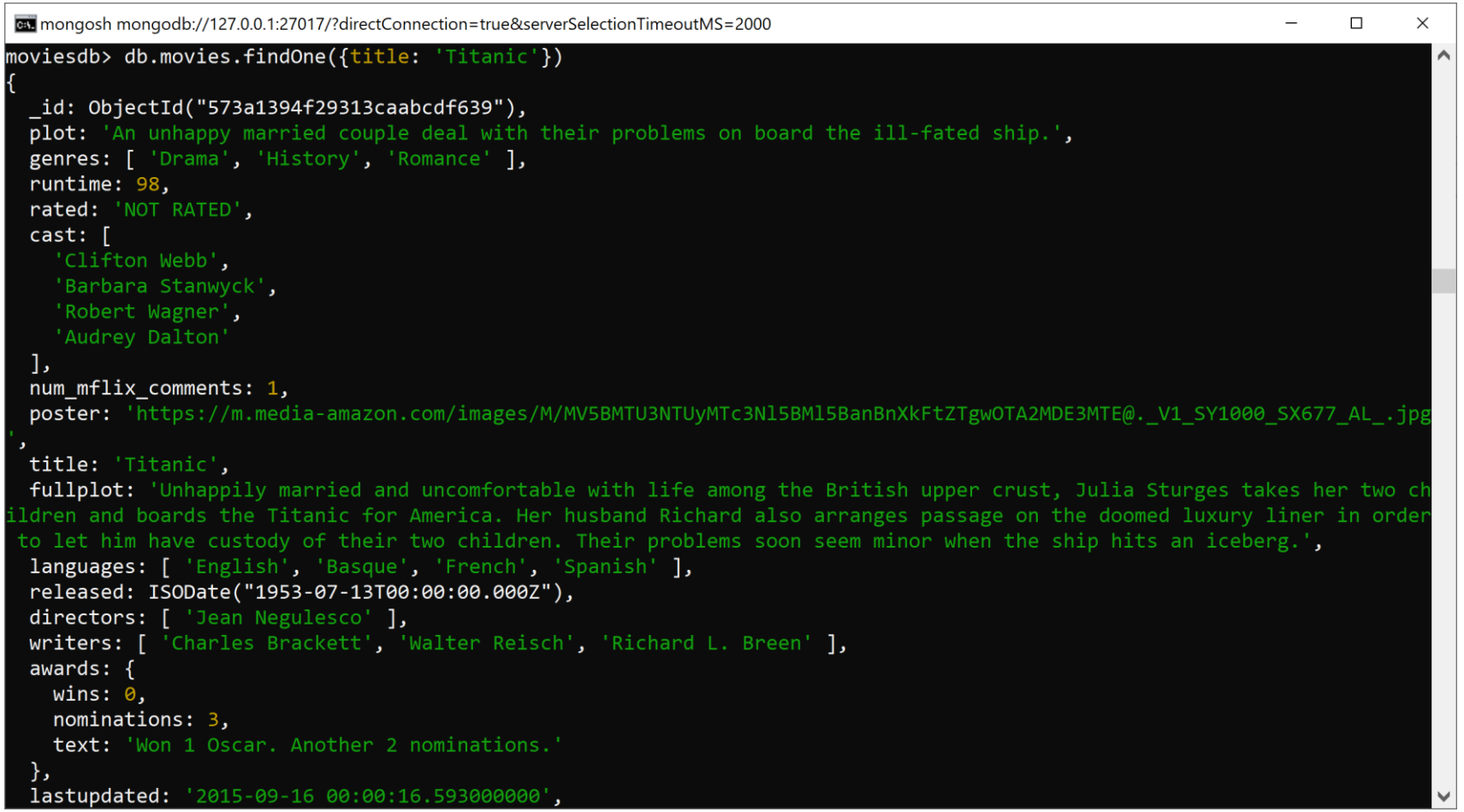
- Multiple conditions can be specified by adding more key/value pairs to the query
- Example: find all the movies released in the year 2010 **and** were rated PG-13:



findOne()

Northeastern University

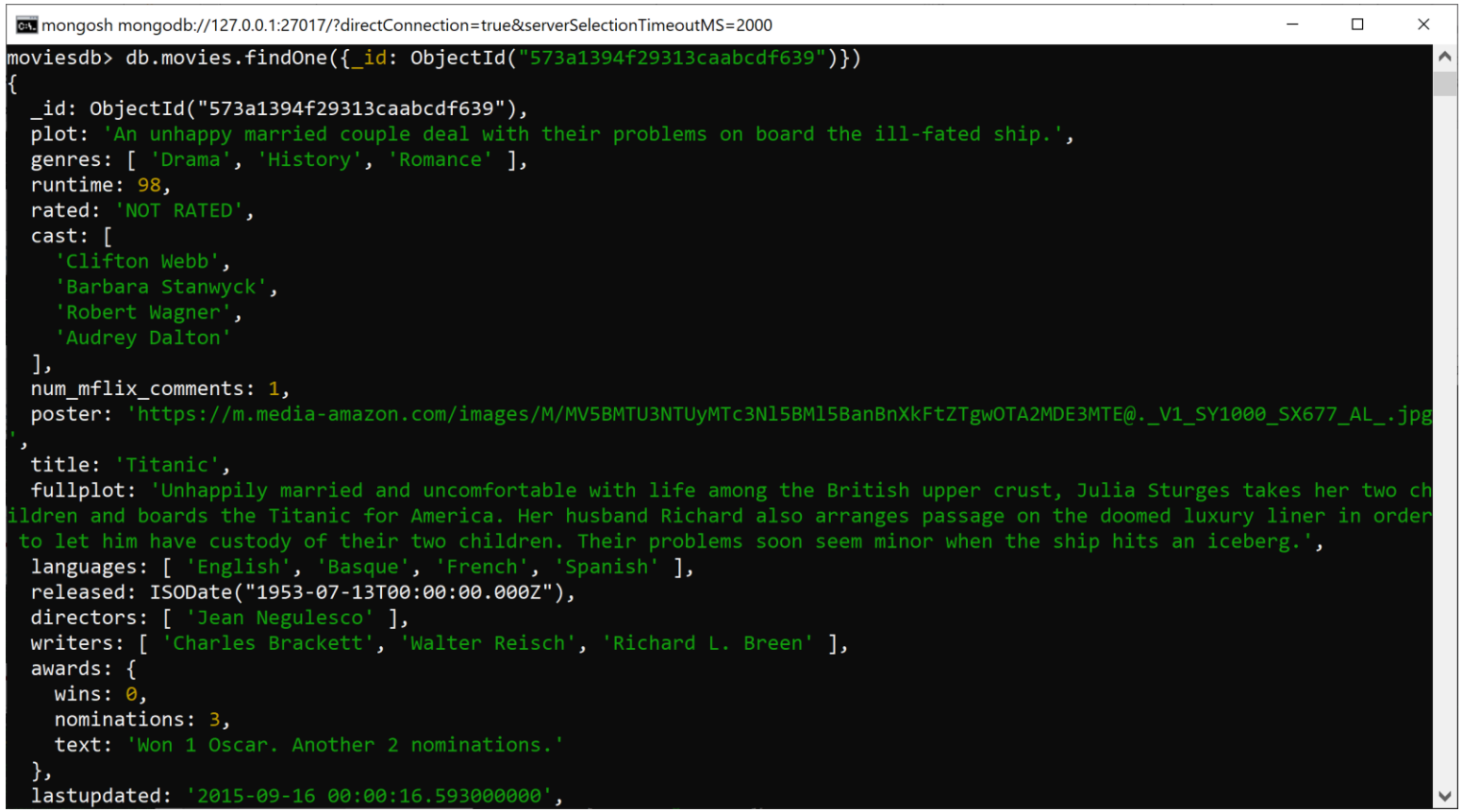
- If we just want to get only one document from the collection, you can use **findOne()**:



ObjectId

Northeastern University

- To find a document with a specific object id, you can use the ObjectId() function



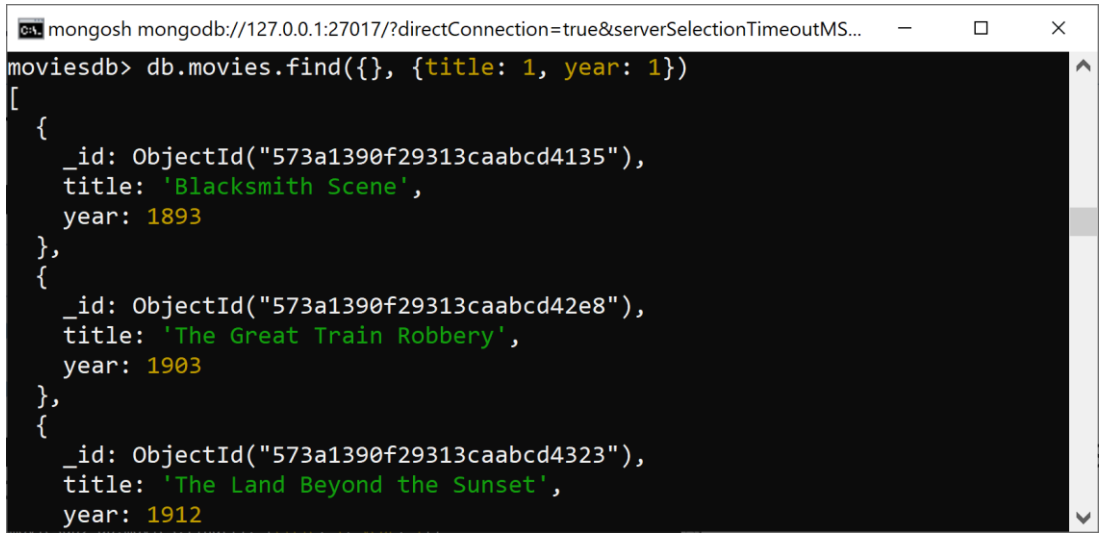
Projection

Northeastern University

- The projection parameter defines which fields to return in the matched documents
- It also takes a document of the following form:

```
{<field1>: <value1>, <field2>: <value2>, ... }
```

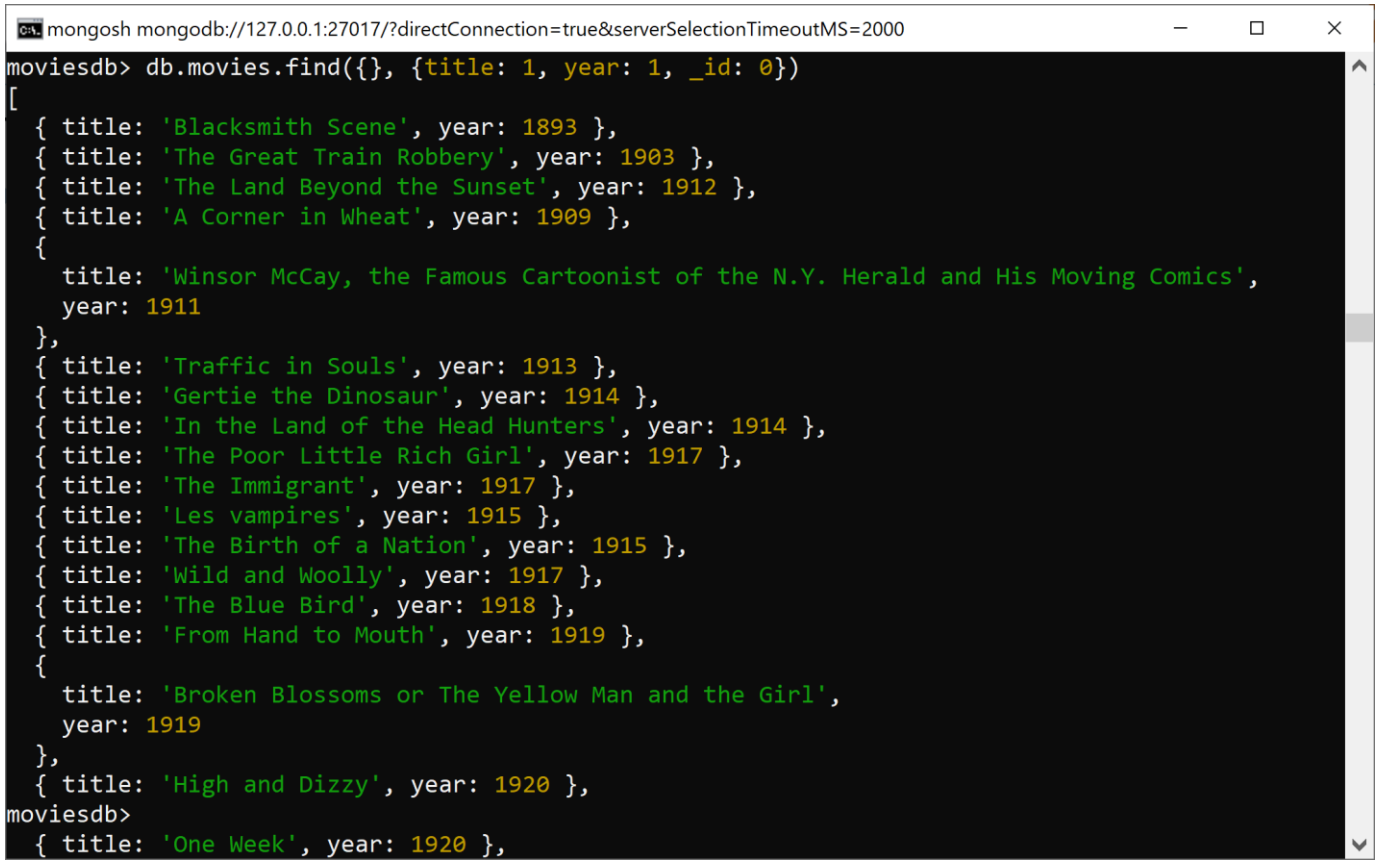
  - A value of 1 or true specifies the inclusion of a field
  - A value of 0 or false specifies the exclusion of a field
- For example, the following returns only the title and release year of the movies



_id Field Projection

Northeastern University

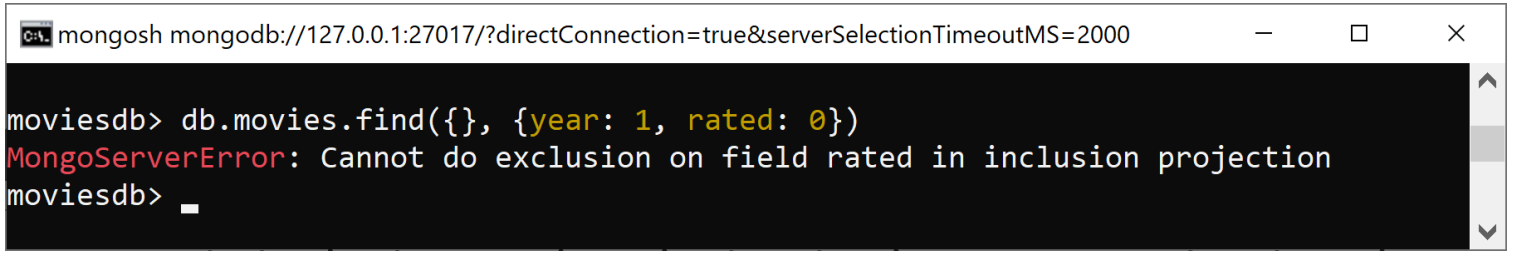
- The _id field is included in the returned documents by default
  - unless you explicitly specify _id: 0 in the projection to suppress the field



Inclusion or Exclusion

Northeastern University

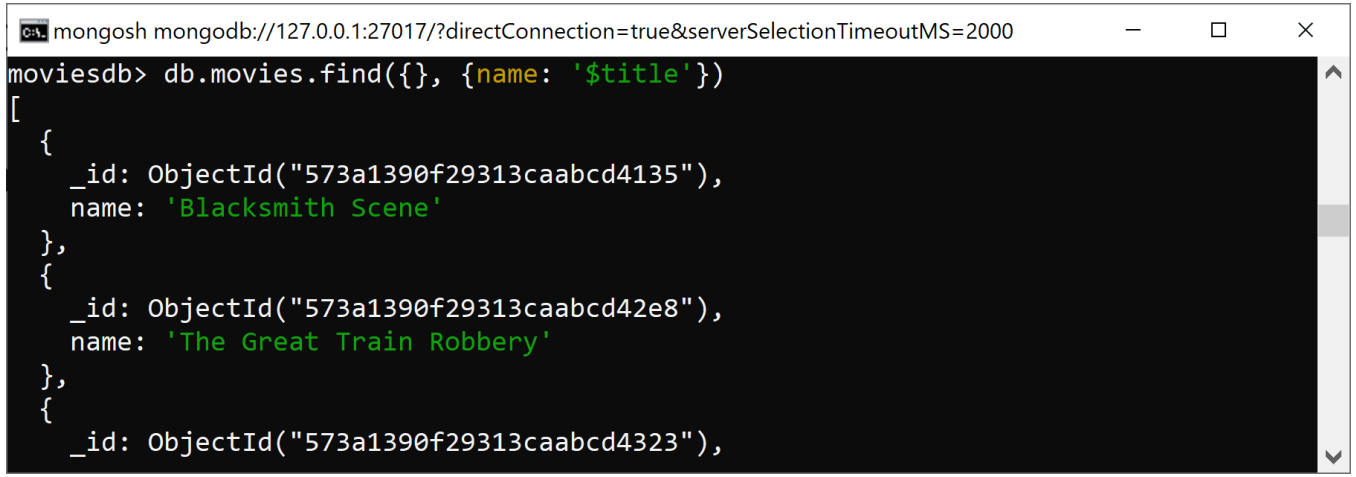
- A projection cannot contain both include and exclude specifications
  - with the exception of the _id field



Projecting New Fields

Northeastern University

- You can project new fields that are based on existing fields
  - To refer to the name of an existing field prefix it with a dollar sign $
- For example, to rename the **title** field to **name**:



Projecting New Fields

Northeastern University

- You can also use projection to promote nested fields
- Example: show the IMDB rating of each movie

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({}, {title: 1, rating: '$imdb.rating', _id: 0})
[
  { title: 'Blacksmith Scene', rating: 6.2 },
  { title: 'The Great Train Robbery', rating: 7.4 },
  { title: 'The Land Beyond the Sunset', rating: 7.1 },
  { title: 'A Corner in Wheat', rating: 6.6 },
  {
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving
Comics',
    rating: 7.3
  },
  { title: 'Traffic in Souls', rating: 6 },
  { title: 'Gertie the Dinosaur', rating: 7.3 },
  { title: 'In the Land of the Head Hunters', rating: 5.8 },
  { title: 'The Poor Little Rich Girl', rating: 6.9 },
  { title: 'The Immigrant', rating: 7.8 },
  { title: 'Les vampires', rating: 6.8 },
  { title: 'The Birth of a Nation', rating: 6.8 },
  { title: 'Wild and Woolly', rating: 6.9 },
  { title: 'The Blue Bird', rating: 6.6 },
  { title: 'From Hand to Mouth', rating: 7 },
  {
```

Query Operators

Northeastern University

- Query operators allow you to specify more complex conditions in the query filter:

```
{<field1>: {<operator1>: <value1>}, ... }
```

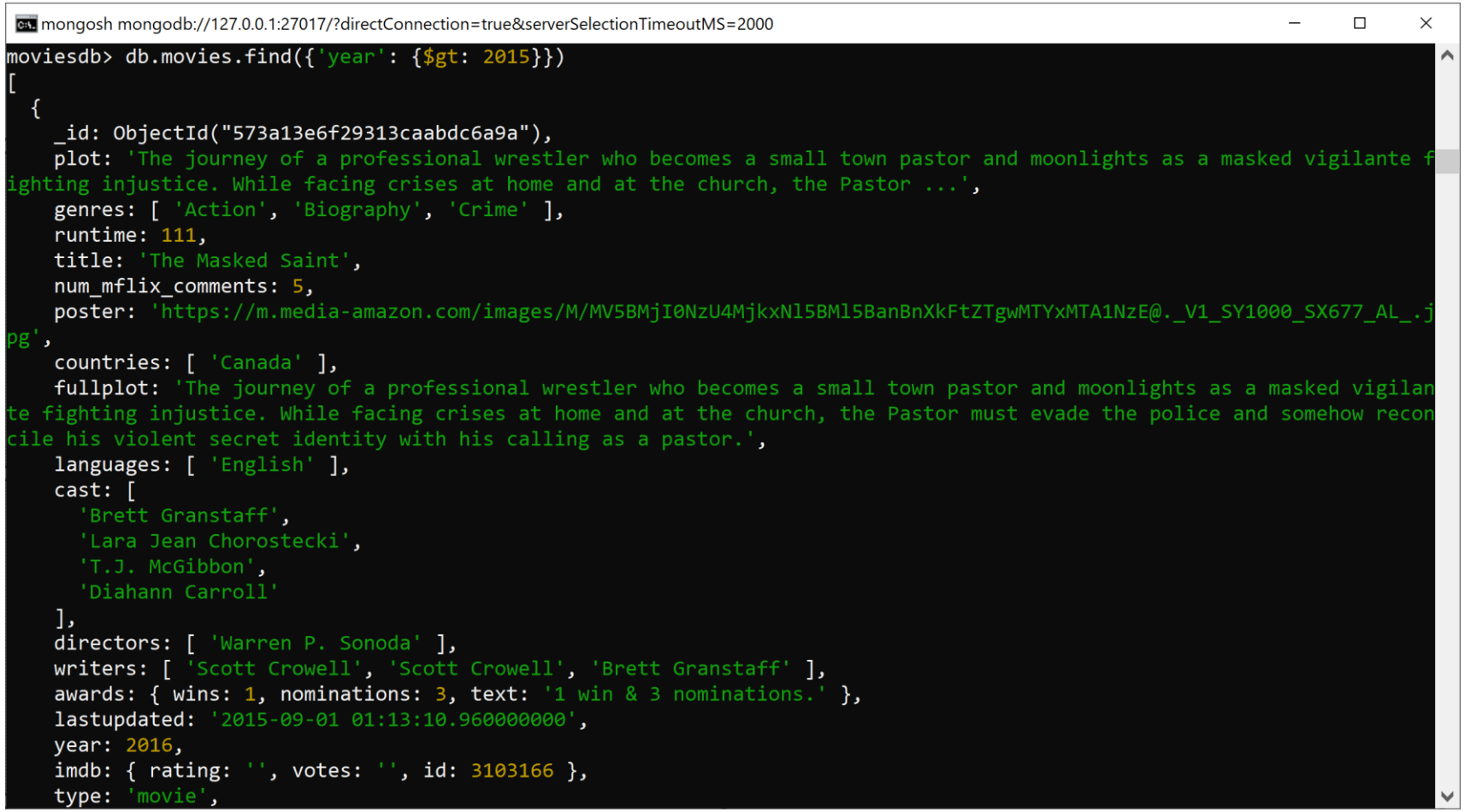
- You can use the following query comparison operators:

Operator	Description
$eq	Matches values that are equal to a specified value
$ne	Matches all values that are not equal to a specified value
$gt	Matches values that are greater than a specified value
$gte	Matches values that are greater than or equal to a specified value
$lt	Matches values that are less than a specified value
$lte	Matches values that are less than or equal to a specified value
$in	Matches any of the values specified in an array
$nin	Matches none of the values specified in an array

Query Operators

Northeastern University

- Example: find all movies that were released after 2015



Query Operators

Northeastern University

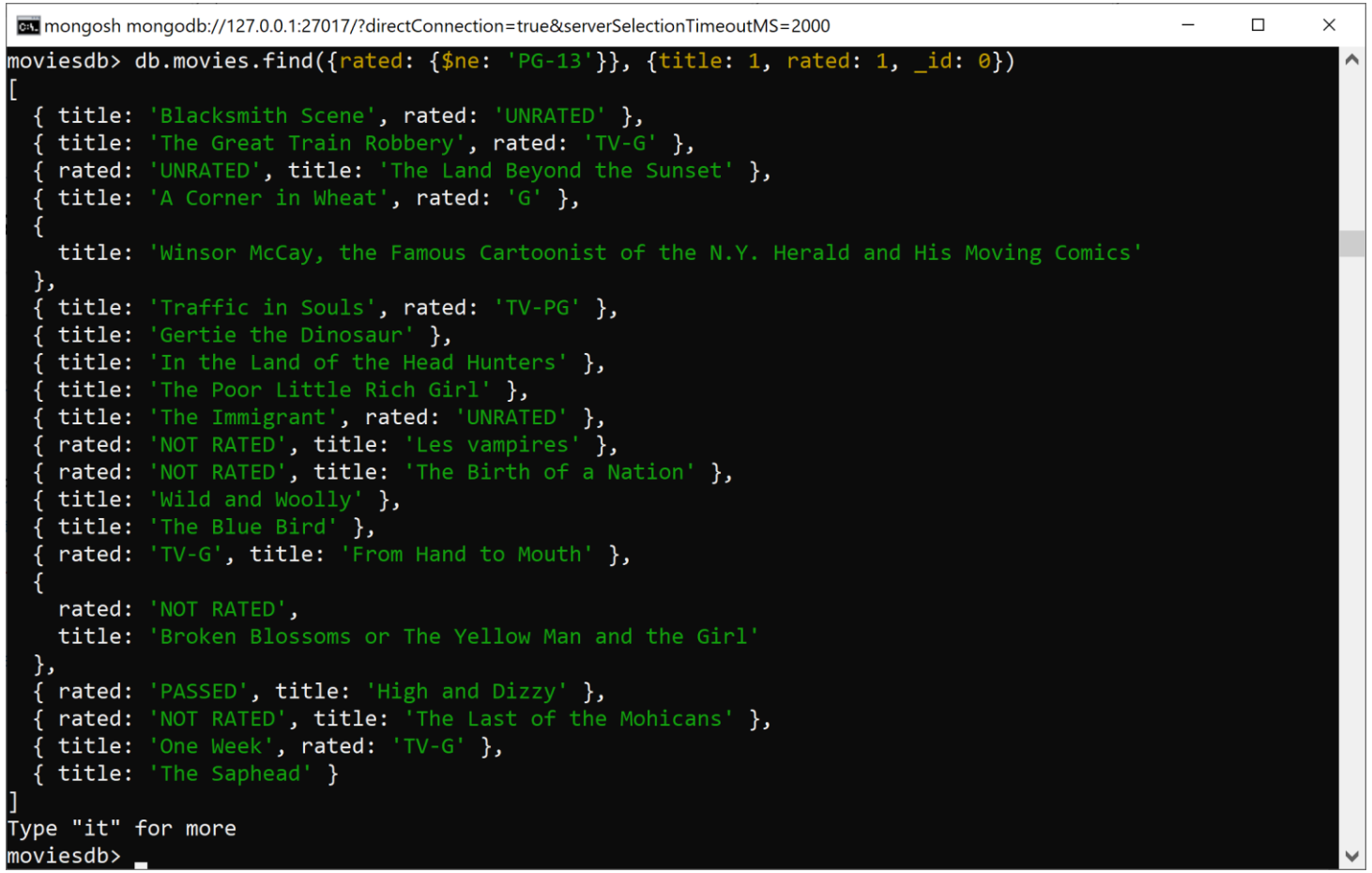
- Example: find all movies that were released between 2000 and 2005 (inclusive)

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({'year': {$gte: 2000, $lte: 2005}}, {'year': 1, 'title': 1, _id: 0})
[
  { title: 'Kate & Leopold', year: 2001 },
  { title: 'Crime and Punishment', year: 2002 },
  { title: 'What Is It?', year: 2005 },
  { year: 2001, title: 'Glitter' },
  { year: 2000, title: 'In the Mood for Love' },
  { year: 2003, title: 'The Manson Family' },
  { title: 'The Dancer Upstairs', year: 2002 },
  { year: 2000, title: 'State and Main' },
  { year: 2000, title: 'Songs from the Second Floor' },
  { title: 'April Captains', year: 2000 },
  { year: 2000, title: 'Chicken Run' },
  { year: 2005, title: 'Fantastic Four' },
  { year: 2002, title: 'Frida' },
  { year: 2001, title: 'From Hell' },
  { year: 2001, title: 'Kate & Leopold' },
  { title: 'Mission: Impossible II', year: 2000 },
  { year: 2000, title: 'The Million Dollar Hotel' },
  {
    year: 2001,
    title: 'The Lord of the Rings: The Fellowship of the Ring'
  },
  { year: 2002, title: 'Resident Evil' },
  { year: 2001, title: 'The Shipping News' }
]
Type "it" for more
moviesdb>
```

Query Operators

Northeastern University

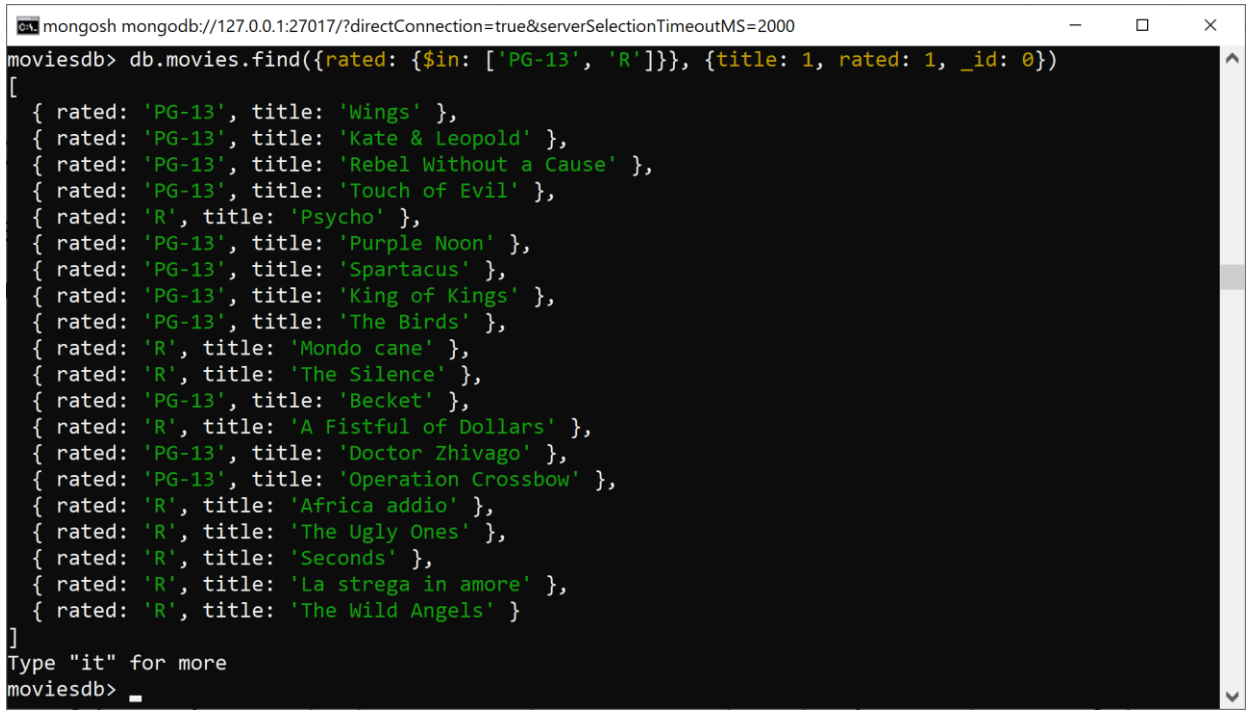
- Example: find all the movies that are not rated PG-13



The $in Operator

Northeastern University

- The $in operator can be used to perform an OR between values of the **same field**
- For example, to find all movies whose rating is PG-13 or R:



- The opposite of $in is **$nin** (returns document that don't match any of the values)

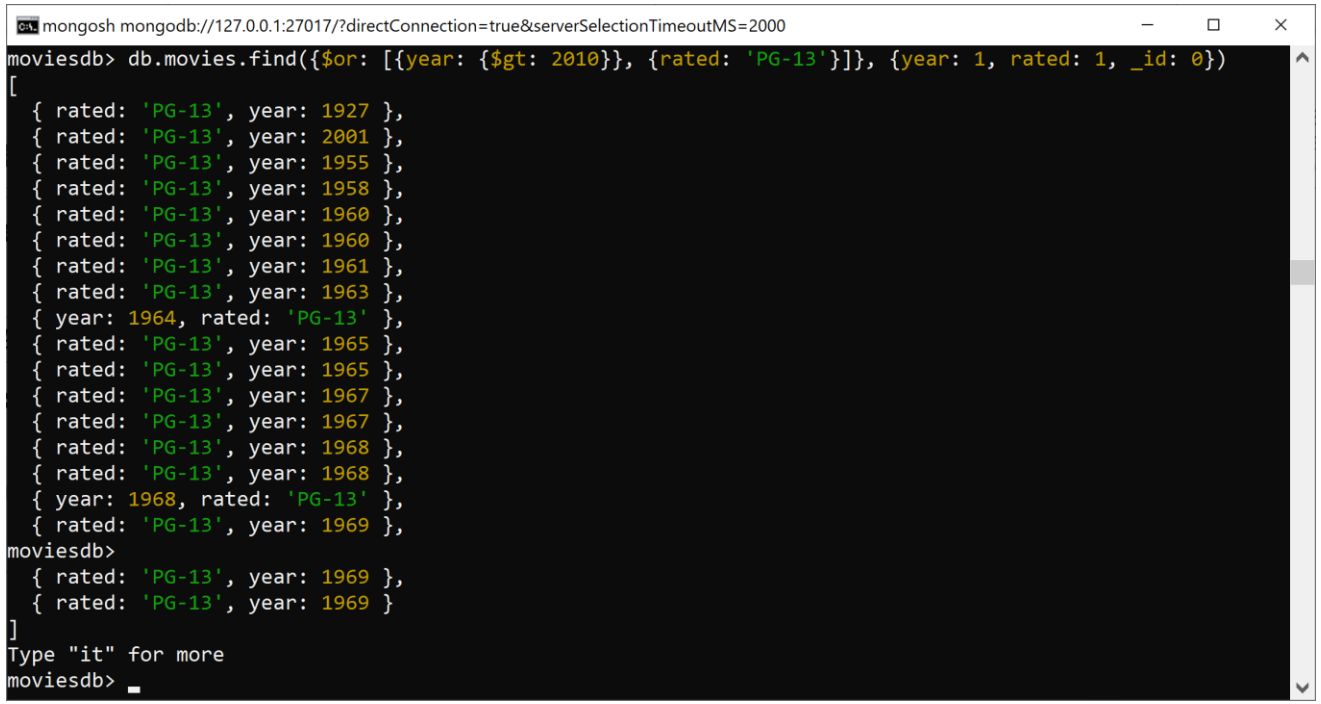
The $or Operator

Northeastern University

- Performs a logical OR operation on an array of one or more expressions
  - and selects the documents that satisfy **at least** one of the expressions

```
{$or: [{<expression1>}, {<expression2>}, ... , {<expressionN>}]}
```

- Example: find all movies that were released after 2010 or their rating is PG-13



The $not Operator

Northeastern University

- **$not** performs a logical NOT operation on the specified operator-expression
  - and selects the documents that don't match the operator-expression

```
{<field>: {$not: {<operator-expression>}}}
```

- Example: find all movies that were released not between 1900 and 2000

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({year: {$not: {$gte: 1900, $lte: 2000}}}, {title: 1, year: 1, _id: 0})
[
  { title: 'Blacksmith Scene', year: 1893 },
  { title: 'Kate & Leopold', year: 2001 },
  { title: 'The Hitch Hikers Guide to the Galaxy', year: '1981è' },
  { title: 'All Passion Spent', year: '1986è' },
  { title: 'The Storyteller', year: '1987è' },
  { title: "Tanner '88", year: '1988è' },
  { title: 'Jack the Ripper', year: '1988è' },
  { title: 'Crime and Punishment', year: 2002 },
  { title: 'Babylon 5', year: '1994è1998' },
  { title: 'The West', year: '1996è' },
  { title: 'What Is It?', year: 2005 },
  { title: 'The Odyssey', year: '1997è' },
  { year: 2001, title: 'Glitter' },
  { year: 2003, title: 'The Manson Family' },
  { title: 'The Dancer Upstairs', year: 2002 },
  { title: 'The Ice House', year: '1997è' },
  { year: 2005, title: 'Fantastic Four' },
  { year: 2002, title: 'Frida' },
  { year: 2001, title: 'From Hell' },
  { year: 2001, title: 'Kate & Leopold' }
]
Type "it" for more
moviesdb>
```

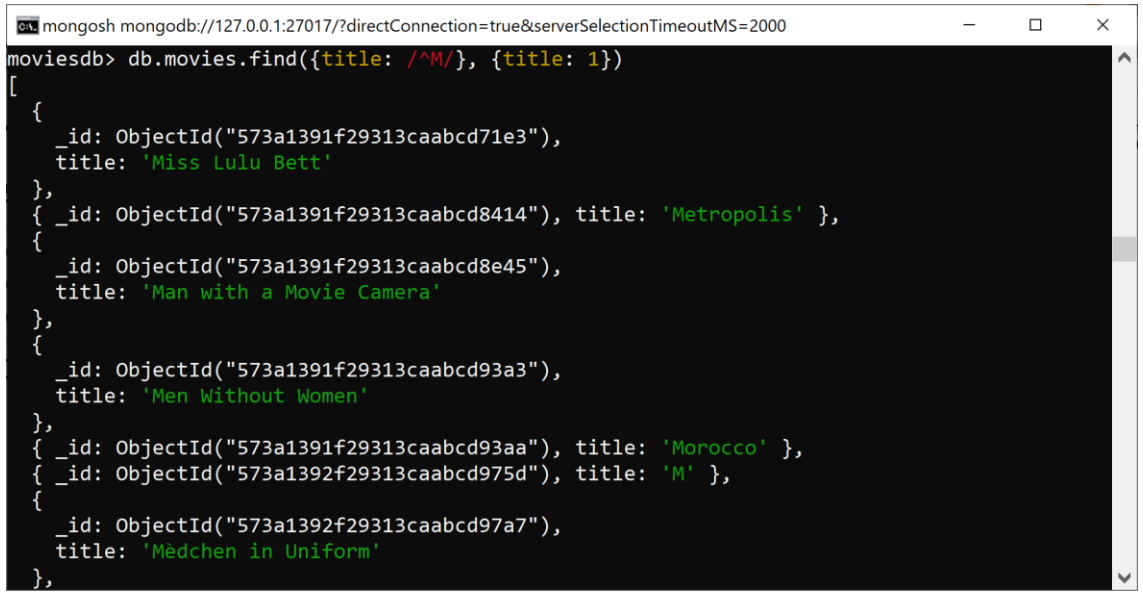
Regular Expressions

Northeastern University

- The **$regex** operator allows to use regular expressions for pattern matching in strings
- To use a regular expression in MongoDB, use one of the following syntaxes:

```
{<field>: /pattern/<options>}
{<field>: {$regex: /pattern/, $options: '<options>'}}
{<field>: {$regex: 'pattern', $options: '<options>'}}
```

- Example: find all movies whose title begins with ‘M’



The $exists Operator

Northeastern University

- The operator **$exists** allows you to check for existence of fields

```
{field: {$exists: <boolean>}}
```

  - This is different from checking if the field value is null
- Example: find movies that have no rating

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({rated: {$exists: false}}, {title: 1, _id: 0})
[
  {
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics'
  },
  { title: 'Gertie the Dinosaur' },
  { title: 'In the Land of the Head Hunters' },
  { title: 'The Poor Little Rich Girl' },
  { title: 'Wild and Woolly' },
  { title: 'The Blue Bird' },
  { title: 'The Saphead' },
  { title: 'Within Our Gates' },
  { title: 'Miss Lulu Bett' },
  { title: "Tol'able David" },
  { title: 'Civilization' },
  { title: 'The Perils of Pauline' },
```

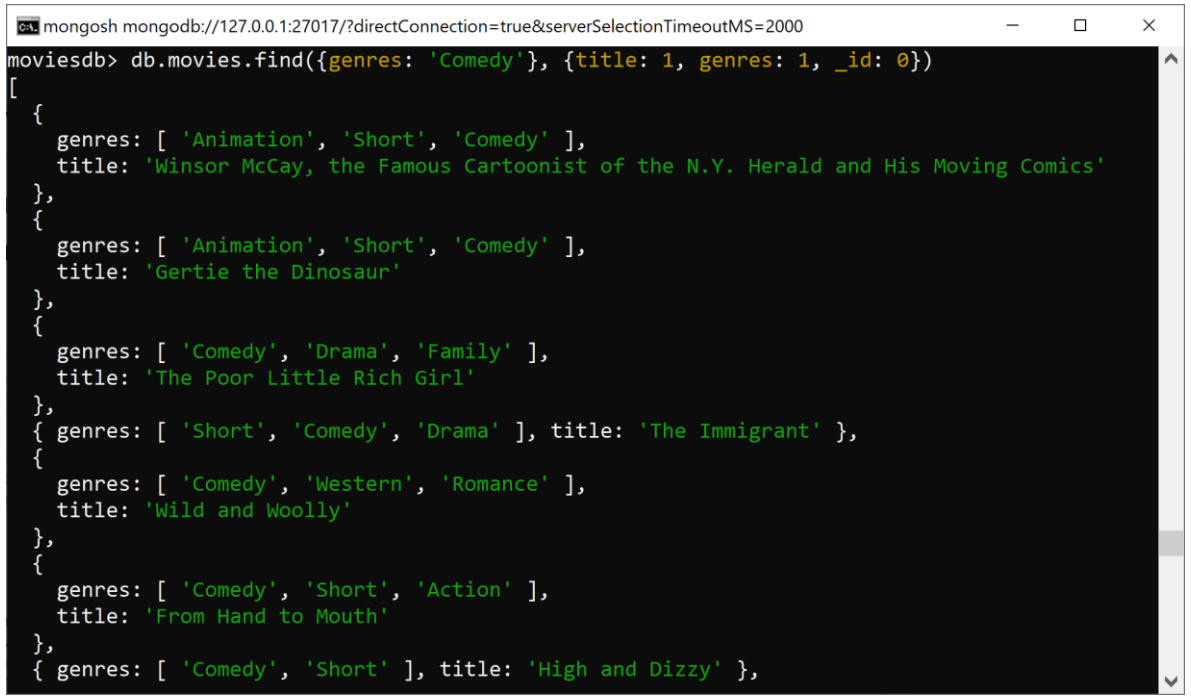
Querying Arrays

Northeastern University

- To check if at least one element in the array has a specified value:

```
{<array field>: <value>}
```

- Example: find all movies the belong to the Comedy genre



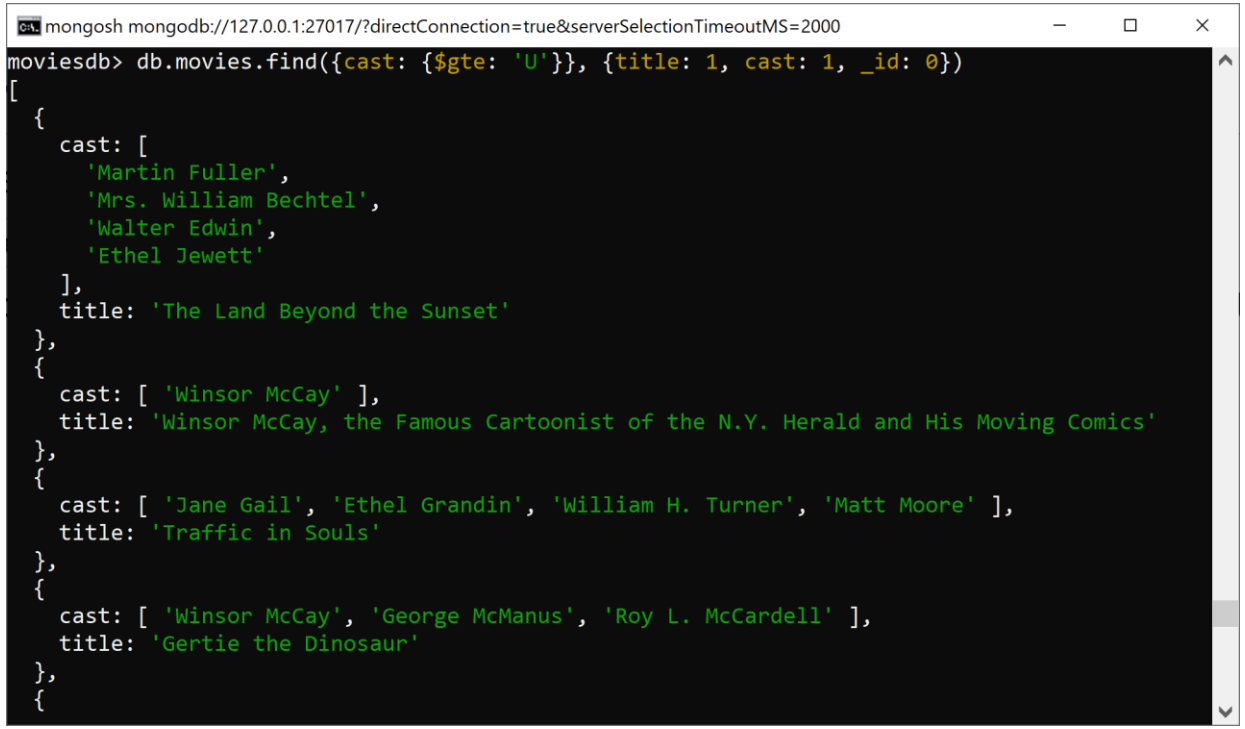
Querying Arrays

Northeastern University

- To specify conditions on the elements in the array field, use query operators:

```
{<array field>: {<operator1>: <value1>, ...}}
```

- Example: find all movies with an actor name that starts with ‘U’ or later in alphabet



The $elemMatch Operator

Northeastern University

- Checks that the array has at least one element that matches **all** the specified criteria

```
{<array field>: {$elemMatch: {<query1>, <query2>, ... }}}
```

- Example: find all movies with an actor whose name is between ‘P’ and ‘R’

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({cast: {$gte: 'P', $lte: 'R'}}, {title: 1, cast: 1, _id: 0})
[
  {
    cast: [
      'Martin Fuller',
      'Mrs. William Bechtel',
      'Walter Edwin',
      'Ethel Jewett'
    ],
    title: 'The Land Beyond the Sunset'
  },
moviesdb>
```

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({cast: {$elemMatch: {$gte: 'P', $lte: 'R'}}}, {title: 1, cast: 1, _id: 0})
[
  {
    cast: [
      'Stanley Hunt',
      'Sarah Constance Smith Hunt',
      'Mrs. George Walkus',
      "Paddy 'Malid"
    ],
    title: 'In the Land of the Head Hunters'
  },
  {
    cast: [
      'Harold Lloyd',
      'Mildred Davis',
      "'Snub' Pollard",
      'Peggy Cartwright'
    ],
    title: 'From Hand to Mouth'
  },
  {
    cast: [
      'Pomeroy Cannon',
```

The $all Operator

Northeastern University

- Checks that the array contains all the specified elements

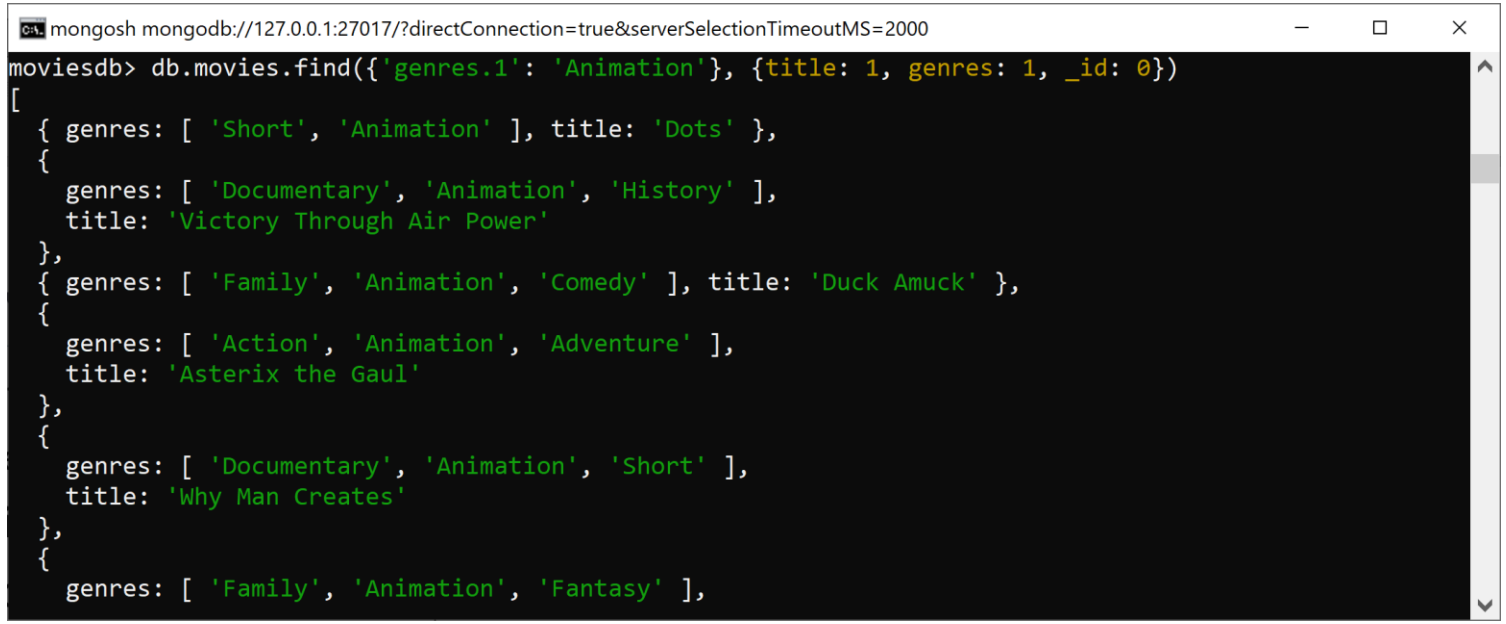
```
{<array field>: {$all: [<value1>, <value2>, ...]}}
```

- Example: find all movies that belong to the genres “Comedy” and “Family”

Querying for an Element by the Array Index

Northeastern University

- Using dot notation, you can query for an element at a specific index of the array
  - The indices start from 0
  - The field must be inside quotation marks
- Example: find all movies whose second genre is Animation:



Querying an Array by the Array Length

Northeastern University

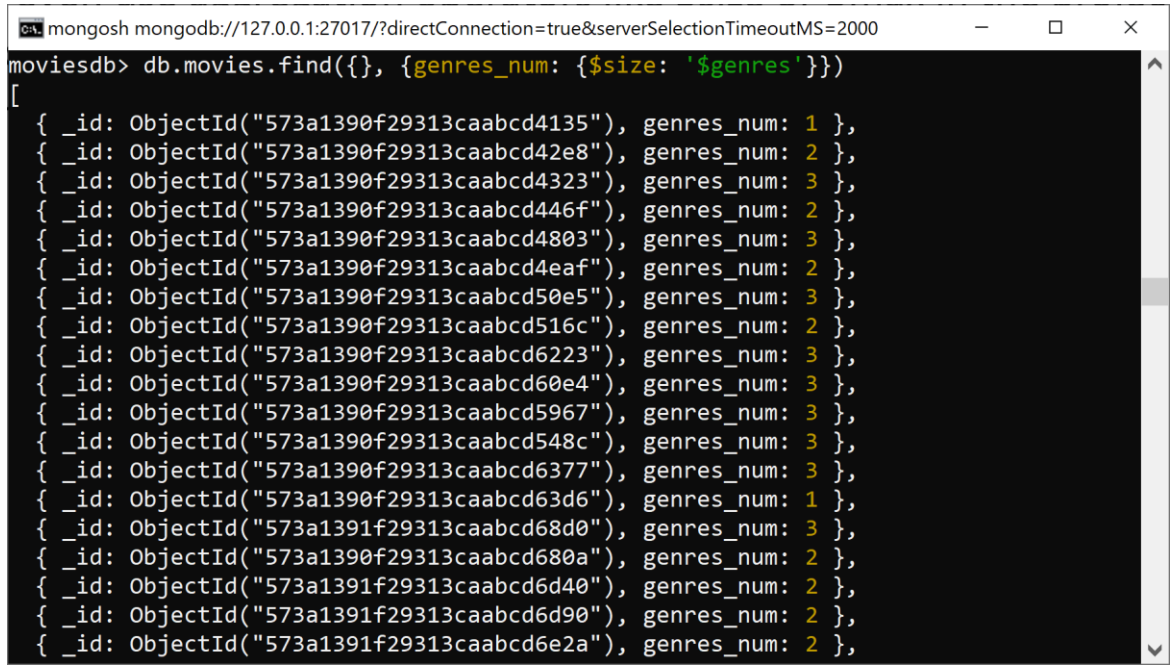
- Use the **$size** operator to query for arrays by number of elements
- Example: find movies that have 3 genres:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({genres: {$size: 3}}, {title: 1, genres: 1, _id: 0})
[
  {
    genres: [ 'Short', 'Drama', 'Fantasy' ],
    title: 'The Land Beyond the Sunset'
  },
  {
    genres: [ 'Animation', 'Short', 'Comedy' ],
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics'
  },
  {
    genres: [ 'Animation', 'Short', 'Comedy' ],
    title: 'Gertie the Dinosaur'
  },
  {
    genres: [ 'Comedy', 'Drama', 'Family' ],
    title: 'The Poor Little Rich Girl'
  },
```

Aggregation Operators

Northeastern University

- Projection can use aggregation operators such as $size or $max
- Example: display the number of genres of each movie



Querying Embedded Documents

Northeastern University

- You can use dot notation to access fields in an embedded document
- The nested field must be written inside quotation marks
- Example: find all movies whose IMDB rating is greater than 9.2:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({'imdb.rating': {$gt: 9.2}}, {title: 1, 'imdb.rating': 1, _id: 0})
[
  { title: 'Cosmos', imdb: { rating: 9.3 } },
  { title: 'The Civil War', imdb: { rating: 9.4 } },
  { imdb: { rating: 9.3 }, title: 'The Shawshank Redemption' },
  { title: 'Band of Brothers', imdb: { rating: 9.6 } },
  { imdb: { rating: 9.3 }, title: 'The Shawshank Redemption' },
  { title: 'The Civil War', imdb: { rating: 9.4 } },
  { title: 'Planet Earth', imdb: { rating: 9.5 } },
  { title: 'The Real Miyagi', imdb: { rating: 9.3 } },
  {
    title: 'A Brave Heart: The Lizzie Velasquez Story',
    imdb: { rating: 9.4 }
  }
]
moviesdb>
```

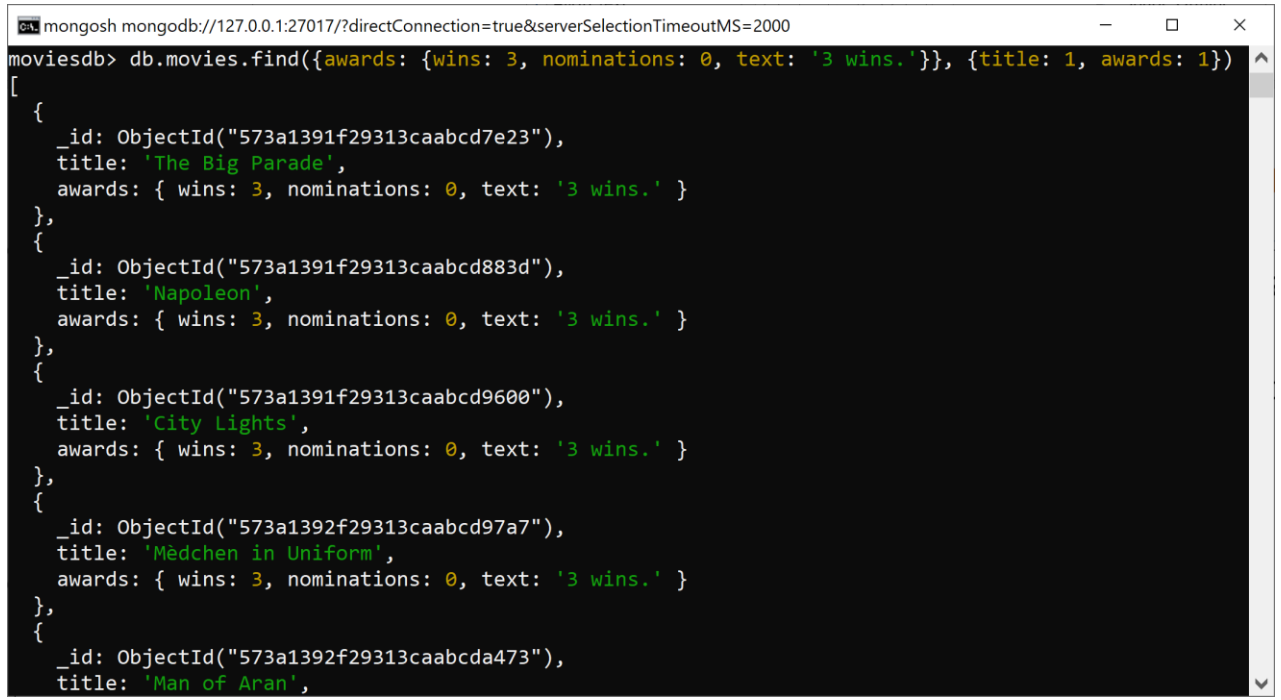
Exact Matches on Embedded Documents

Northeastern University

- To match the entire embedded document use

```
{<field>: {<document>}
```

- Example: find all the movies who won 3 awards and had 0 nominations



Class Exercise

Northeastern University

- Suppose we are storing blog posts with the following structure:

```
> db.blog.find()
{
    "content": "...",
    "comments": [
        {
            "author": "Joe",
            "score": 3,
            "comment": "nice post"
        },
        {
            "author": "Mary",
            "score": 6,
            "comment": "terrible post"
        }
    ]
}
```

- Write a query to find comments by Joe that were scored at least a 5

Solution

Northeastern University

- The following query doesn’t work

```
db.blog.find({comments: {author: "Joe", score: {$gte: 5}}})
```

  - Embedded document matches have to match the whole document
- It also wouldn’t work to do:

```
db.blog.find({"comments.author": "Joe", "comments.score": {$gte: 5}})
```

  - this would match "author": "Joe" in the first comment and "score": 6 in the second comment
- To write the query correctly we need to use $elemMatch:

```
db.blog.find({comments: {$elemMatch: {author: "joe", score: {$gte: 5}}}})
```

Cursors

Northeastern University

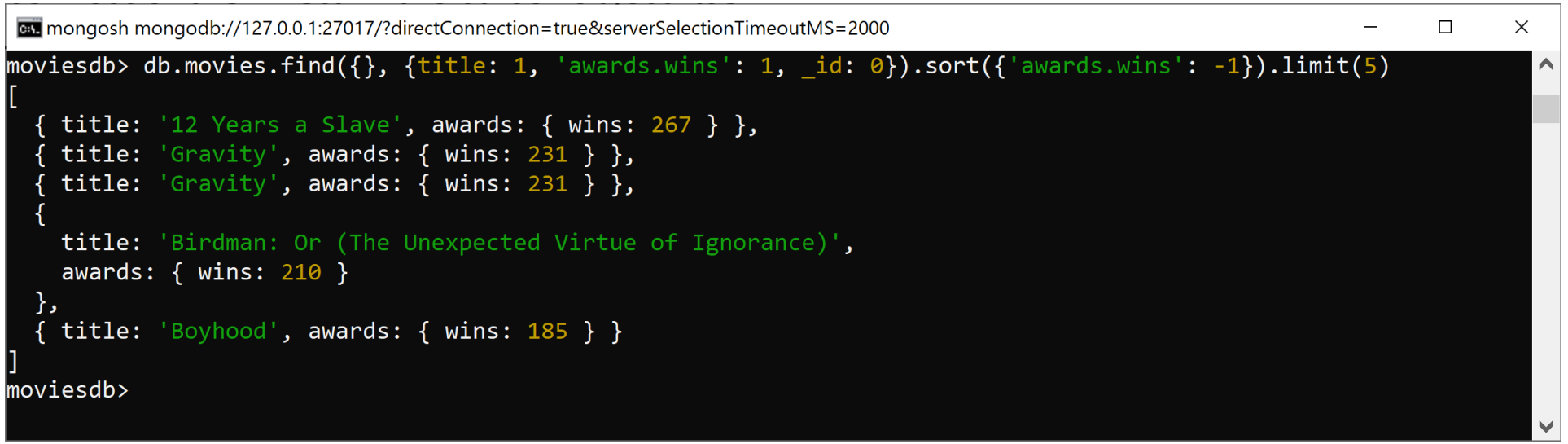
- The db.collection.find() method returns a cursor to the results
- The cursor object provides various methods that modify its behavior:

Method	Description
cursor.sort()	Sorts the documents in the result set
cursor.limit()	Limits the number of documents in the result set
cursor.skip()	Controls the starting point of the result set
cursor.count()	Returns the number of documents in the result set
cursor.map()	Applies a function to each document visited by the cursor
cursor.explain()	Provides information on the query execution plan for the find() method

Cursors

Northeastern University

- Example: find the top 5 movies that won the highest number of awards



Distinct

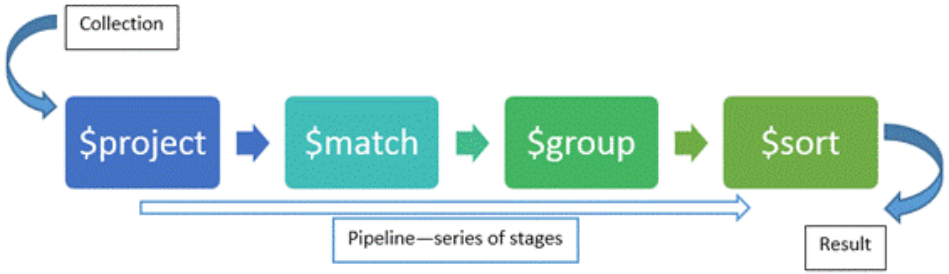
Northeastern University

- **db.collection.distinct**(field, query) returns the distinct values for a specified field
  - An optional query specifies the documents from which to retrieve the distinct values
- Example: find the distinct genres for a movie

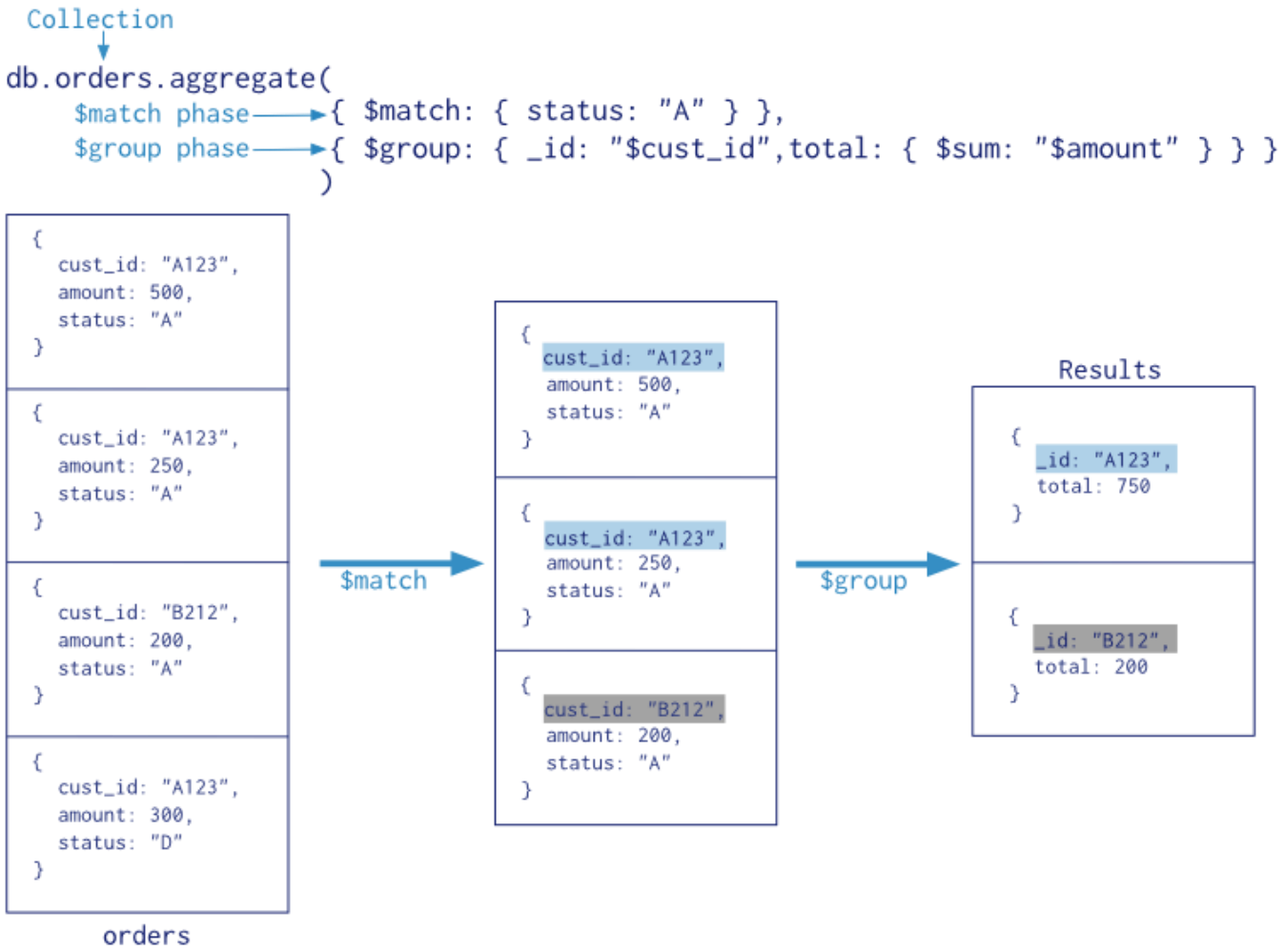
Aggregation Pipelines

Northeastern University

- An aggregation pipeline consists of one or more stages that process documents:
  - Each stage performs an operation on the input documents
    - For example, a stage can filter documents, group documents, and calculate values
  - The documents that are output from a stage are passed to the next stage
- An aggregation pipeline is defined using the **db.[collection].aggregate()** method
- This method gets as an array of pipeline stages
  - Each stage is defined using a stage operator (such as $match or $group)
  - Stage operators can be combined in any order and repeated as many times as necessary



Aggregation Pipeline Example



Pipeline Stage Operators

Northeastern University

Operator	Description
$match	Pass only the documents that match the specified condition(s) to the next pipeline stage.
$project	Reshapes each document in the stream, such as by adding new fields or removing existing fields.
$group	Groups documents by the specified _id expression and applies the accumulator expression (if specified) to each group.
$unwind	Deconstructs an array field from the input documents to output a document for each element.
$sort	Reorders the document stream by a specified sort key.
$limit	Passes the first *n* documents unmodified to the pipeline where *n* is the specified limit.
$skip	Skips the first *n* documents where *n* is the specified skip number and passes the remaining documents unmodified to the pipeline.
$count	Returns a count of the number of documents at this stage of the aggregation pipeline.
$lookup	Performs a left outer join to another collection in the same database to filter in documents from the "joined" collection for processing.

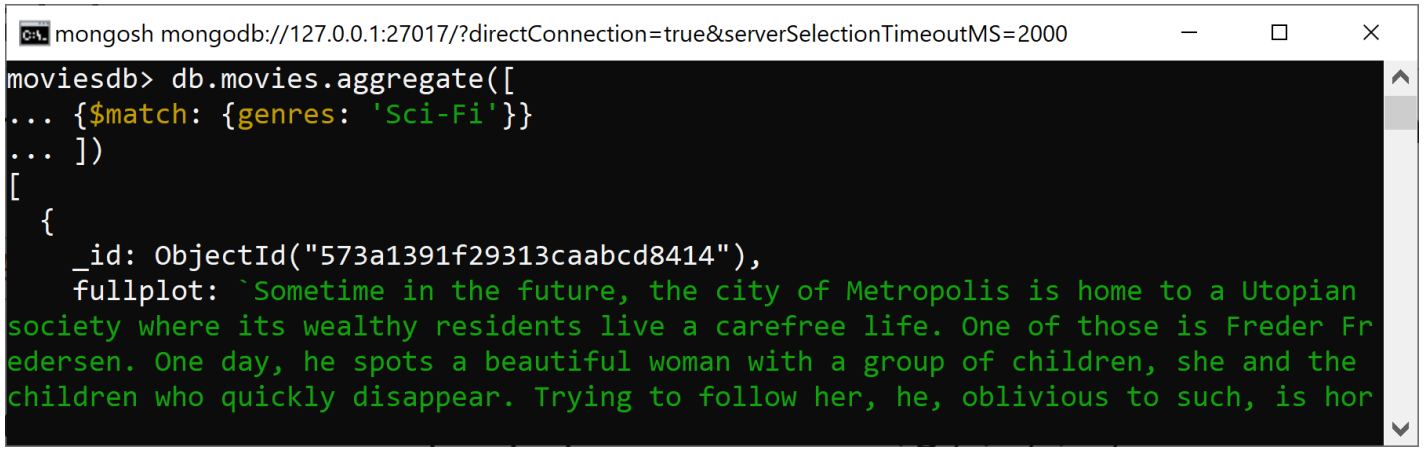
$match

Northeastern University

- $match filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage

```
{$match: {<query>}}
```

- The query syntax is identical to the query filter in find()
  - Can use all the usual query operators such as $gt, $lt, $in, etc.
- Example: find all the movies that belong to 'Sci-Fi':



$project

Northeastern University

- Passes the documents with the requested fields to the next stage in the pipeline
  - The specified fields can be existing fields in the documents or newly computed fields

```
{$project: {<specification(s)>}}
```

- The $project specifications have the following forms:

Form	Description
<field>: <1 or true>	Specifies the inclusion of a field.
<field>: <0 or false>	Specifies the exclusion of a field.
<field>: <expression>	Adds a new field or resets the value of an existing field.

$project

Northeastern University

- Example: show only the titles of the Sci-Fi movies

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
... {$match: {genres: 'Sci-Fi'}},
... {$project: {title: 1, _id: 0}}
... ])
[
  { title: 'Metropolis' },
  { title: 'Frankenstein' },
  { title: 'Dr. Jekyll and Mr. Hyde' },
  { title: 'The Invisible Man' },
  { title: 'Bride of Frankenstein' },
  { title: 'Flash Gordon' },
  { title: 'The Invisible Ray' },
  { title: 'Black Friday' },
  { title: 'House of Dracula' },
  { title: 'Bud Abbott Lou Costello Meet Frankenstein' },
  { title: 'It Happens Every Spring' },
  { title: 'Destination Moon' },
  { title: 'Rocketship X-M' },
  { title: 'The Day the Earth Stood Still' },
  { title: 'Five' },
  { title: 'The Thing from Another World' },
  { title: 'When Worlds Collide' },
```

Aggregation Pipeline Operators

Northeastern University

- Projection is much more powerful in the pipeline than it is in a “normal” query
- It can use various operators that are available only in the aggregation pipeline
  - e.g., arithmetic, comparison, conditional, string, array, date, operators
- Operator expressions are similar to functions that take arguments
- In general, these expressions take an array of arguments:

```
{<operator>: [<argument1>, <argument2>, ...]}
```

- If the operator accepts a single argument, you can omit the outer array:

```
{<operator>: <argument>}
```

- For a full list of operators see
  - https://docs.mongodb.com/manual/reference/operator/aggregation/

Arithmetic Operators

Northeastern University

- Perform arithmetic operations on numbers

Operator	Description
$abs	Returns the absolute value of a number
$add	Adds numbers to return the sum, or adds numbers and a date to return a new date
$ceil	Returns the smallest integer greater than or equal to the specified number
$divide	Returns the result of dividing the first number by the second
$exp	Raises e to the specified exponent
$floor	Returns the largest integer less than or equal to the specified number
$log	Calculates the log of a number in the specified base
$mod	Returns the remainder of the first number divided by the second
$multiply	Multiplies numbers to return the product
$pow	Raises a number to the specified exponent
$sqrt	Calculates the square root
$subtract	Returns the result of subtracting the second value from the first

Arithmetic Operators

Northeastern University

- Example: show the runtime of each movie in hours (instead of minutes)

Comparison Operators

Northeastern University

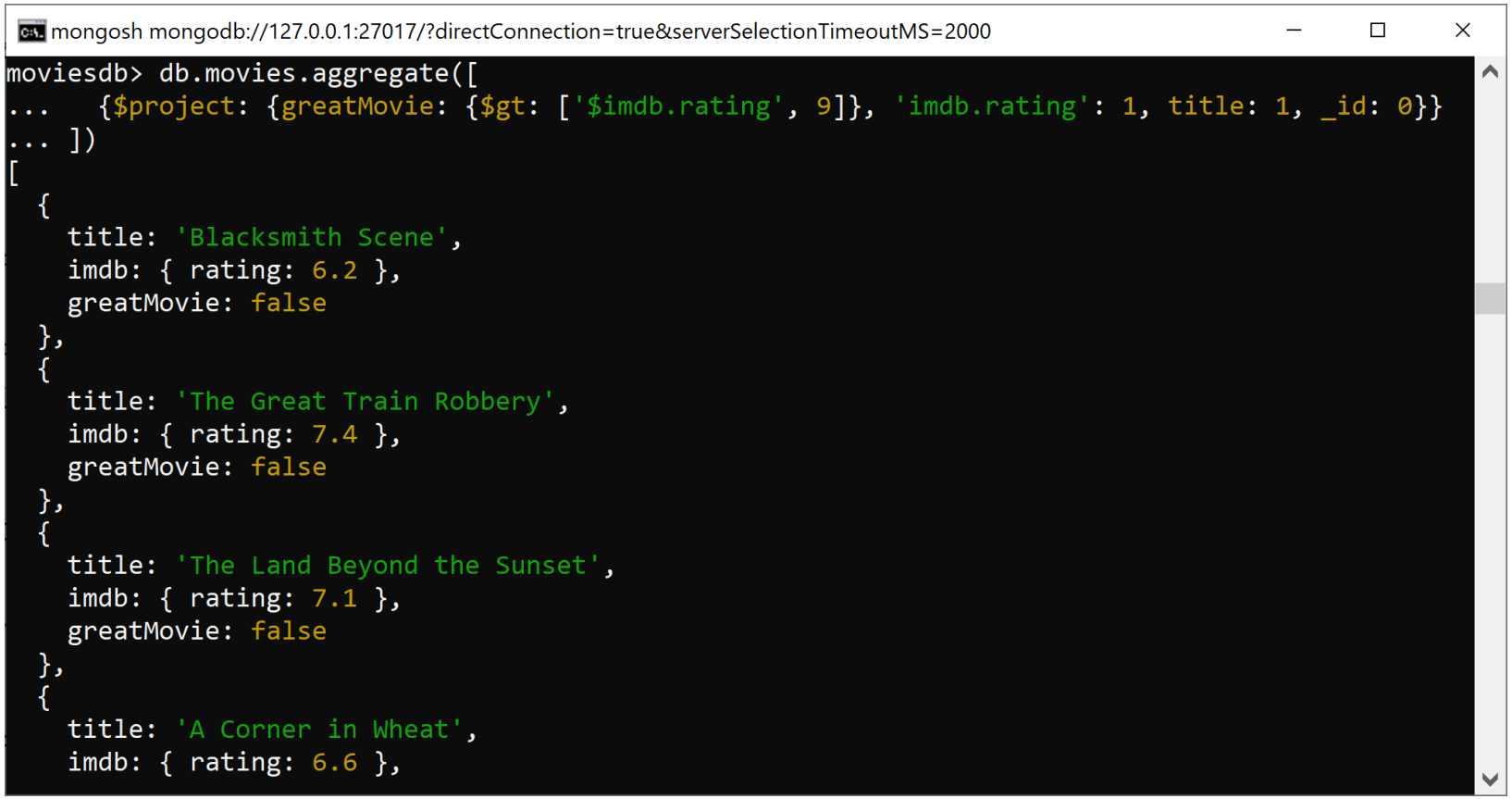
- Compare the values of two arguments and return a Boolean

Operator	Description
$eq	Returns true if the values are equivalent.
$gt	Returns true if the first value is greater than the second.
$gte	Returns true if the first value is greater than or equal to the second.
$lt	Returns true if the first value is less than the second.
$lte	Returns true if the first value is less than or equal to the second.
$ne	Returns true if the values are not equivalent.
$cmp	Returns 0 if the two values are equivalent, 1 if the first value is greater than the second, and -1 if the first value is less than the second.

Comparison Operators

Northeastern University

- Example: for each movie add {greatMovie: true} if its rating is greater than 9



Boolean Operators

Northeastern University

- Evaluate their argument expressions as Booleans and return a Boolean as the result

Operator	Description
$and	Returns true only when *all* its expressions evaluate to true
$or	Returns true when any of its expressions evaluates to true
$not	Returns the Boolean value that is the opposite of its argument expression

- Example: for each movie add {goodMovie: true} if its rating is between 8 and 9

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([ { $project: { goodMovie: { $and: [{ $gte: ['$imdb.rating', 8] },
 { $lte: ['$imdb.rating', 9] }] }, 'imdb.rating': 1, title: 1, _id: 0 } }])
[
  {
    title: 'Blacksmith Scene',
    imdb: { rating: 6.2 },
    goodMovie: false
  },
  {
    title: 'The Great Train Robbery',
    imdb: { rating: 7.4 },
    goodMovie: false
  },
  {
    title: 'The Land Beyond the Sunset',
    imdb: { rating: 7.1 },
```

$ifNull

Northeastern University

- Returns the first expression if it is not null, otherwise the second expression
- Example: add a field {directors: ‘unknown’} if there is no directors field in the movie

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$project: {title: 1, directors: {$ifNull: ['$directors', 'Unknown']}, _id: 0}},
... ])
[
  { title: 'Blacksmith Scene', directors: [ 'William K.L. Dickson' ] },
  {
    title: 'The Great Train Robbery',
    directors: [ 'Edwin S. Porter' ]
  },
  {
    title: 'The Land Beyond the Sunset',
    directors: [ 'Harold M. Shaw' ]
  },
  { title: 'A Corner in Wheat', directors: [ 'D.W. Griffith' ] },
  {
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics',
    directors: [ 'Winsor McCay', 'J. Stuart Blackton' ]
  },
  { title: 'Traffic in Souls', directors: [ 'George Loane Tucker' ] },
  { title: 'Gertie the Dinosaur', directors: [ 'Winsor McCay' ] },
  {
    title: 'In the Land of the Head Hunters',
    directors: [ 'Edward S. Curtis' ]
  },
  {
    title: 'The Poor Little Rich Girl',
```

String Operators

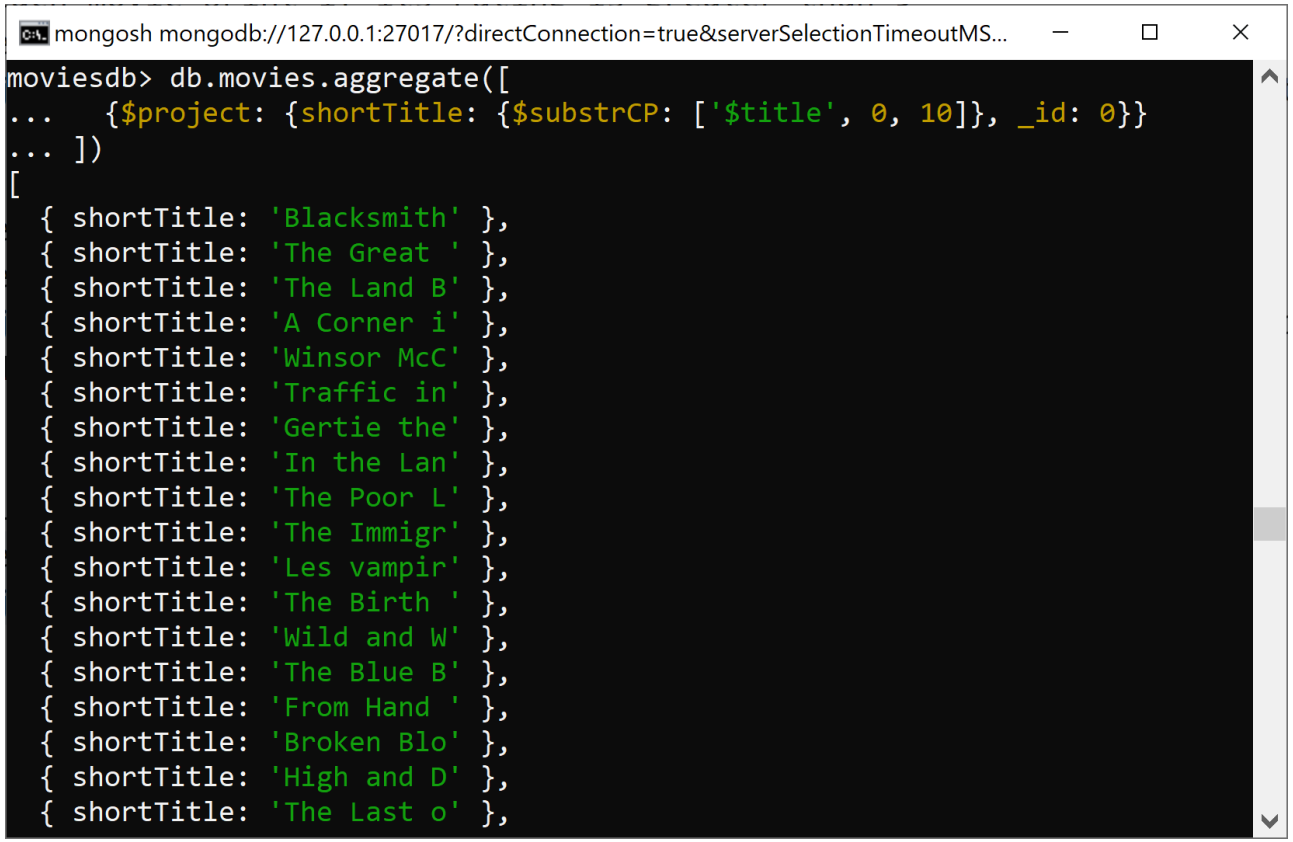
Northeastern University

Operator	Description
$concat	Concatenates any number of strings
$strLenBytes / $strLenCP	Returns the number of UTF-8 encoded bytes / code points in a string
$indexOfBytes / $indeOfCP	Searches a string for an occurrence of a substring and returns the UTF-8 byte / code point index of the first occurrence. If the substring is not found, returns -1.
$substrBytes / $substrCP	Returns the substring of a string. Starts with the character at the specified UTF-8 byte index / cpde point (zero-based) in the string and continues for the specified number of bytes.
$split	Splits a string into substrings based on a delimiter. Returns an array of substrings.
$strcasecmp	Performs case-insensitive string comparison and returns: 0, 1 or -1
$toLower	Converts a string to lowercase
$toUpper	Converts a string to uppercase
$regexFind	Applies a regex to a string and returns information on the first matched substring.
$regexFindAll	Applies a regex to a string and returns information on the all matched substrings.
$regexMatch	Applies a to a string and returns a Boolean that indicates if a match is found or not.

String Operators

Northeastern University

- Example: display the first 10 letters of each movie's title



Array Operators

Northeastern University

Operator	Description
$size	Returns the number of elements in the array
$arrayElemAt	Returns the element at the specified array index
$first	Returns the first array element
$last	Returns the last array element
$slice	Returns a subset of an array
$in	Returns a Boolean indicating whether a specified value is in an array
$indexOfArray	Searches an array for a specified value and returns the index of its first occurrence (or -1 if not found)
$filter	Selects a subset of the array to return an array with only the elements that match the filter condition
$map	Applies a subexpression to each element of an array and returns the array of resulting values in order
$reduce	Applies an expression to each element in an array and combines them into a single value

Array Operators

Northeastern University

- Example: Display the names of the first and second actor of each movie

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$project: {
...     firstActor: {$first: '$cast'},
...     secondActor: {$arrayElemAt: ['$cast', 1]},
...     _id: 0
...   }}
... ])
[
  { firstActor: 'Charles Kayser', secondActor: 'John Ott' },
  {
    firstActor: 'A.C. Abadie',
    secondActor: "Gilbert M. 'Broncho Billy' Anderson"
  },
  { firstActor: 'Martin Fuller', secondActor: 'Mrs. William Bechtel' },
  { firstActor: 'Frank Powell', secondActor: 'Grace Henderson' },
  { firstActor: 'Winsor McCay' },
  { firstActor: 'Jane Gail', secondActor: 'Ethel Grandin' },
  { firstActor: 'Winsor McCay', secondActor: 'George McManus' },
  {
    firstActor: 'Stanley Hunt',
    secondActor: 'Sarah Constance Smith Hunt'
  },
```

Array Operators

Northeastern University

- Example: Find all the movies with at least 3 genres

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$project: {
...     numOfGenres: {$size: {$ifNull: ['$genres', []]}},
...     title: 1,
...     _id: 0
...   }},
...   {$match: {numOfGenres: {$gte: 3}}}
... ])
[
  { title: 'The Land Beyond the Sunset', numOfGenres: 3 },
  {
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics',
    numOfGenres: 3
  },
  { title: 'Gertie the Dinosaur', numOfGenres: 3 },
  { title: 'The Poor Little Rich Girl', numOfGenres: 3 },
  { title: 'The Immigrant', numOfGenres: 3 },
  { title: 'Les vampires', numOfGenres: 3 },
  { title: 'The Birth of a Nation', numOfGenres: 3 },
  { title: 'Wild and Woolly', numOfGenres: 3 },
  { title: 'From Hand to Mouth', numOfGenres: 3 },
  { title: 'The Four Horsemen of the Apocalypse', numOfGenres: 3 },
  { title: 'The Ace of Hearts', numOfGenres: 3 },
  { title: 'The Kid', numOfGenres: 3 },
  { title: 'Regeneration', numOfGenres: 3 },
  { title: 'Cops', numOfGenres: 3 },
```

$split

Northeastern University

- $split divides a string into an array of substrings based on a delimiter

```
{$split: [<string expression>, <delimiter>]}
```

- Example: display the first and last name of the first director of each movie

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$project: {firstDirector: {$split: [{$first: '$directors'}, ' ']}}},
...   {$project: {
...     firstDirectorFirstName: {$first: '$firstDirector'},
...     firstDirectorLastName: {$arrayElemAt: ['$firstDirector', 1]},
...     _id: 0
...   }}
... ])
[
  { firstDirectorFirstName: 'William', firstDirectorLastName: 'K.L.' },
  { firstDirectorFirstName: 'Edwin', firstDirectorLastName: 'S.' },
  { firstDirectorFirstName: 'Harold', firstDirectorLastName: 'M.' },
  { firstDirectorFirstName: 'D.W.', firstDirectorLastName: 'Griffith' },
  { firstDirectorFirstName: 'Winsor', firstDirectorLastName: 'McCay' },
  { firstDirectorFirstName: 'George', firstDirectorLastName: 'Loane' },
  { firstDirectorFirstName: 'Winsor', firstDirectorLastName: 'McCay' },
  { firstDirectorFirstName: 'Edward', firstDirectorLastName: 'S.' },
  {
    firstDirectorFirstName: 'Maurice',
    firstDirectorLastName: 'Tourneur'
  },
  {
```

$group

Northeastern University

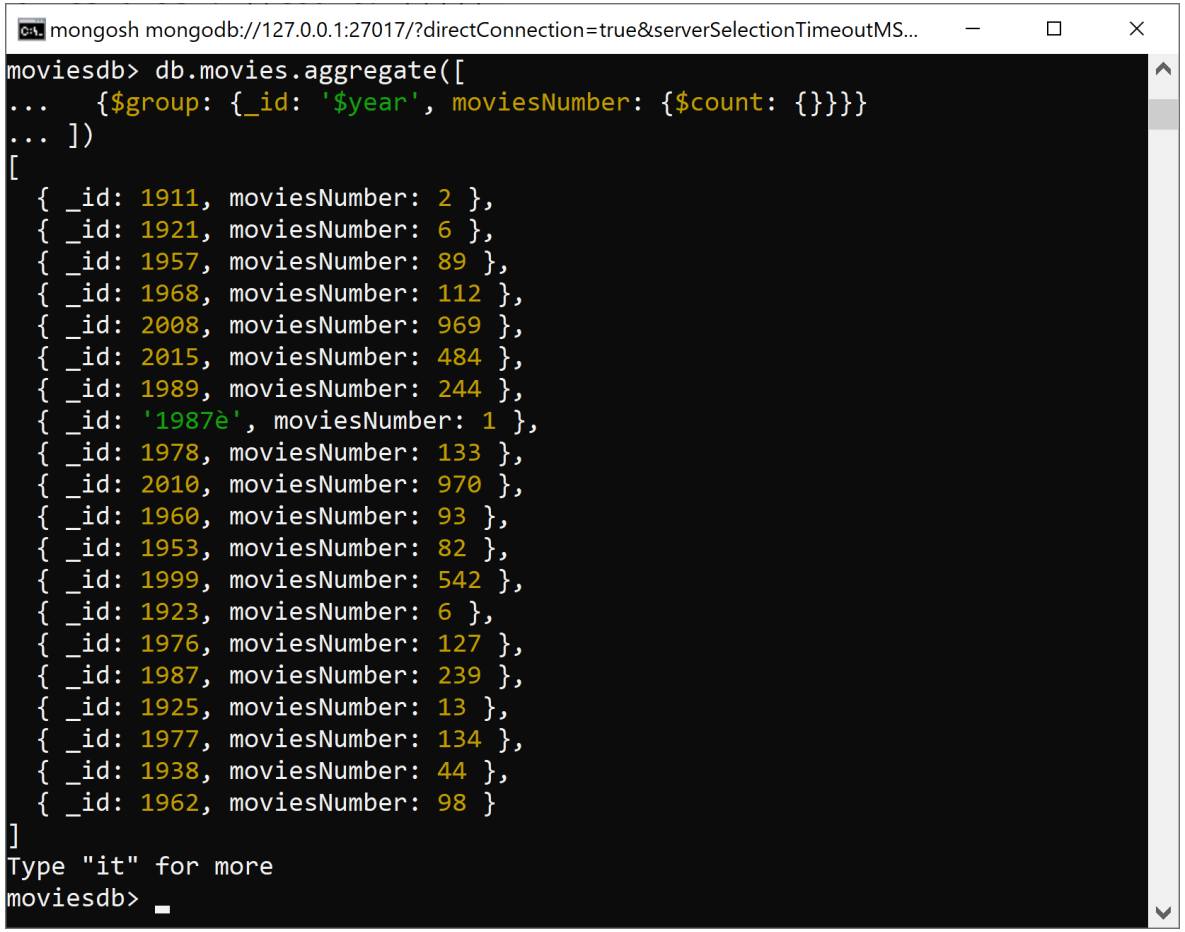
- Separate documents into groups based on a **group key**
- Use the **_id** field to set the group key
  - The group key can be a field, a set of fields or the result of an expression
- The output is one document for each unique group key
  - The _id field in this document is set to the group key of that document
- The output can contain additional accumulator expressions (such as $count or $max)

```
{$group: {
    _id: <field path or expression>, // Group key
    <field1>: {<accumulator1> : <expression1>},
    ...
  }
}
```

$group

Northeastern University

- Example: Display the number of movies released in every year



Accumulator Operators

Northeastern University

- The <accumulator> operator can be one of the following operators:

Operator	Description
$count	Returns the number of documents in a group
$sum	Returns a sum of numerical values. Ignores non-numeric values.
$avg	Returns an average of numerical values. Ignores non-numeric values.
$max	Returns the highest expression value for each group
$min	Returns the lowest expression value for each group
$first	Returns a value from the first document for each group
$last	Returns a value from the last document for each group
$top	Returns the top element within a group according to the specified sort order
$push	Returns an array of expression values for each group
$addToSet	Returns an array of unique expression values for each group
$accumulator	Returns the result of a user-defined accumulator function

Accumulator Operators

Northeastern University

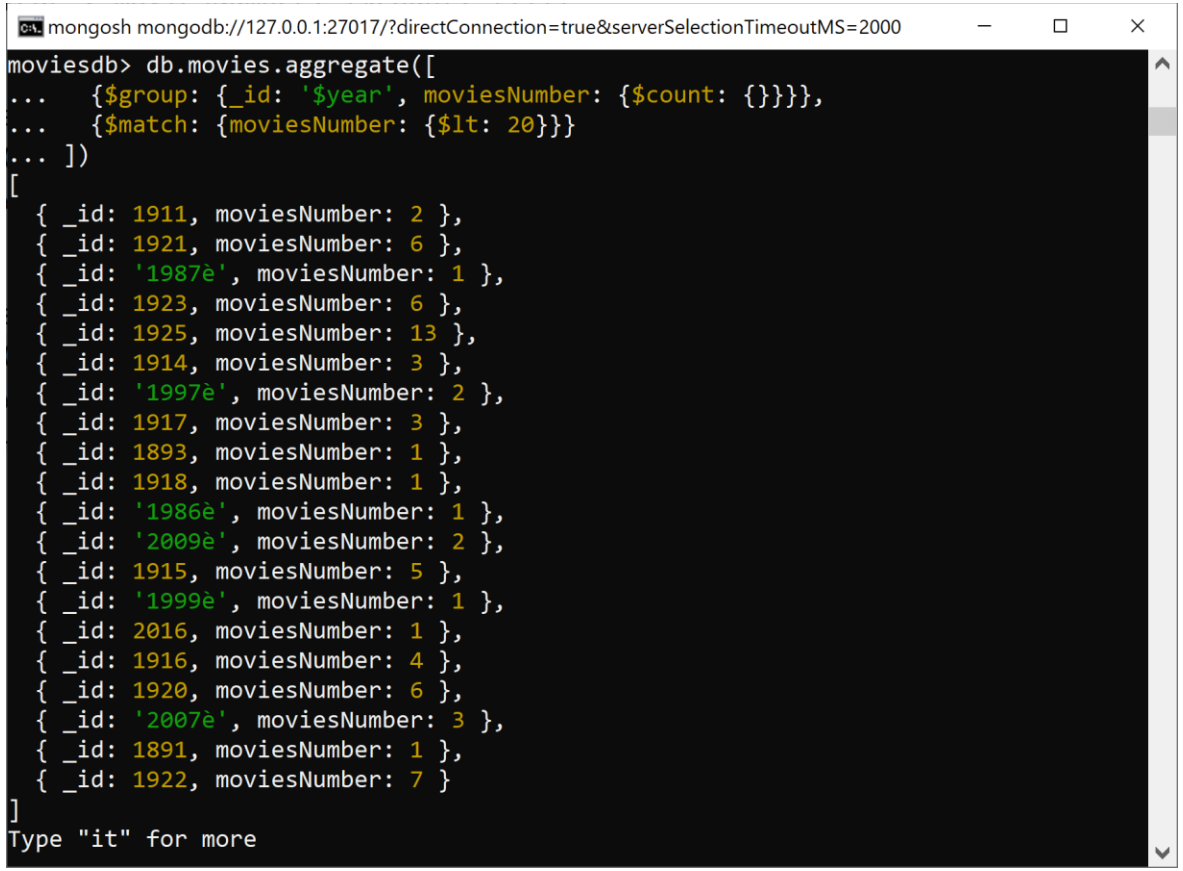
- Example: Calculate the average IMDB rating of movies in each year

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$group: {_id: '$year', averageRating: {$avg: '$imdb.rating'}}}
... ])
[
  { _id: 1925, averageRating: 7.684615384615385 },
  { _id: 2002, averageRating: 6.561679389312976 },
  { _id: 1977, averageRating: 7.079699248120301 },
  { _id: 1996, averageRating: 6.5879350348027845 },
  { _id: 1987, averageRating: 6.658577405857741 },
  { _id: 1953, averageRating: 7.182926829268292 },
  { _id: 1976, averageRating: 6.939370078740157 },
  { _id: 1923, averageRating: 7.416666666666667 },
  { _id: 1999, averageRating: 6.619003690036901 },
  { _id: 1960, averageRating: 7.238709677419355 },
  { _id: 2010, averageRating: 6.5279669762641905 },
  { _id: 1978, averageRating: 6.786466165413534 },
  { _id: 1989, averageRating: 6.722950819672132 },
  { _id: '1987è', averageRating: 8.9 },
  { _id: 2008, averageRating: 6.573243801652892 },
  { _id: 2015, averageRating: 6.94197247706422 },
  { _id: 1921, averageRating: 7.566666666666666 },
  { _id: 1911, averageRating: 7.3 },
  { _id: 1957, averageRating: 7.216853932584269 },
  { _id: 1936, averageRating: 7.25 }
]
Type "it" for more
moviesdb>
```

Group by Having

Northeastern University

- The equivalent of a SQL having clause is to add a $match stage after $group
- Example: Display the years in which less than 20 movies were released



Retrieving Distinct Values

Northeastern University

- You can use $group without any operator to retrieve the distinct field values
- Example: Display all the possible values of the rated field

Group by null

Northeastern University

- If you specify an _id value of null, or any other constant value, $group returns a single document that aggregates values across all of the input documents
- Example: Display the average IMDB rating of all the movies

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$group: {_id: null, averageRating: {$avg: '$imdb.rating'}}}
... ])
[ { _id: null, averageRating: 6.693466223698782 } ]
moviesdb>
```

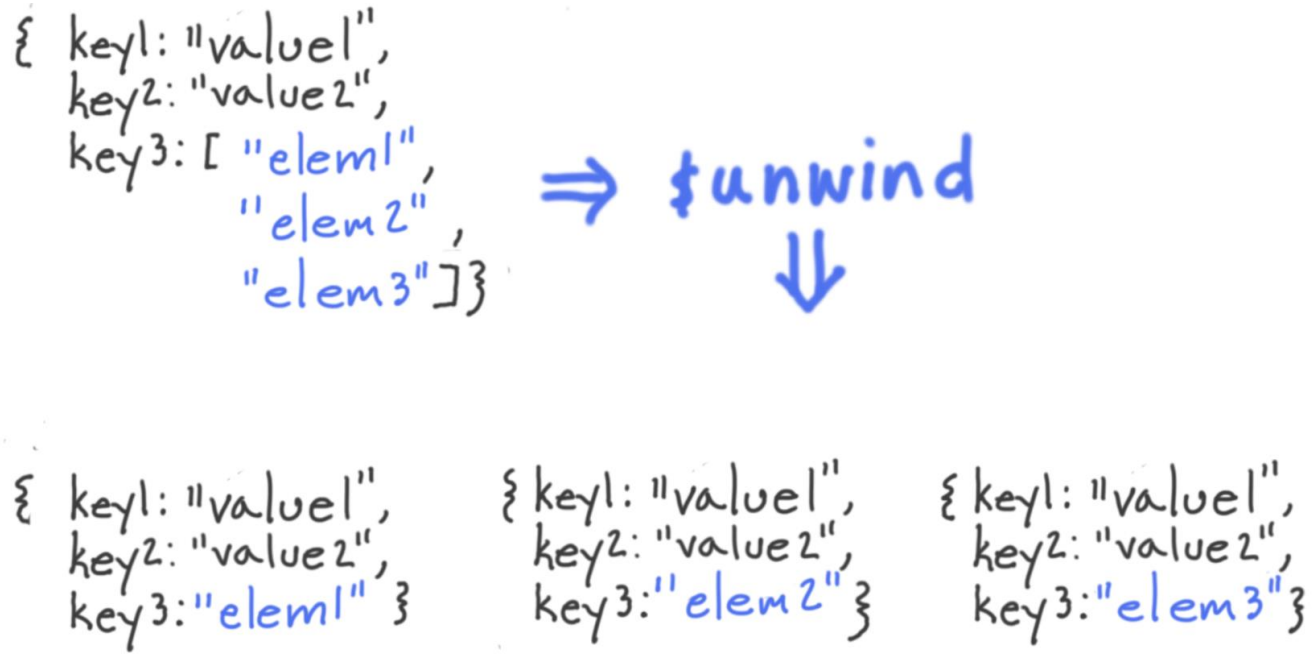
- Example: Display the average number of actors per movie

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$project: {numOfActors: {$size: {$ifNull: ['$cast', []]}}}},
...   {$group: {_id: null, averageNumOfActors: {$avg: '$numOfActors'}}}
... ])
[ { _id: null, averageNumOfActors: 3.864055397425549 } ]
moviesdb>
```

$unwind

- The $unwind stage deconstructs an array field from the input documents to output a document for **each** element
  - Each output document is the input document with the array field replaced by the element

```
{$unwind: <field path>}
```



$unwind

Northeastern University

- Example: Display the actors of each movie in separate documents

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS...
moviesdb> db.movies.aggregate([
...   {$project: {title: 1, cast: 1, _id: 0}},
...   {$unwind: '$cast'}
... ])
[
  { cast: 'Charles Kayser', title: 'Blacksmith Scene' },
  { cast: 'John Ott', title: 'Blacksmith Scene' },
  { cast: 'A.C. Abadie', title: 'The Great Train Robbery' },
  {
    cast: "Gilbert M. 'Broncho Billy' Anderson",
    title: 'The Great Train Robbery'
  },
  { cast: 'George Barnes', title: 'The Great Train Robbery' },
  { cast: 'Justus D. Barnes', title: 'The Great Train Robbery' },
  { cast: 'Martin Fuller', title: 'The Land Beyond the Sunset' },
  { cast: 'Mrs. William Bechtel', title: 'The Land Beyond the Sunset' },
  { cast: 'Walter Edwin', title: 'The Land Beyond the Sunset' },
  { cast: 'Ethel Jewett', title: 'The Land Beyond the Sunset' },
  { cast: 'Frank Powell', title: 'A Corner in Wheat' },
  { cast: 'Grace Henderson', title: 'A Corner in Wheat' },
  { cast: 'James Kirkwood', title: 'A Corner in Wheat' },
  { cast: 'Linda Arvidson', title: 'A Corner in Wheat' },
```

Group by Unwound Values

Northeastern University

- You can combine $unwind with $group to group by unwound values
- Example: Display the number of movies each actor participated in

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS...
moviesdb> db.movies.aggregate([
...   {$project: {cast: 1, _id: 0}},
...   {$unwind: '$cast'},
...   {$group: {_id: '$cast', numOfMovies: {$count: {}}}}
... ])
[
  { _id: 'Aurèlien Recoing', numOfMovies: 10 },
  { _id: 'Kazuyoshi Kushida', numOfMovies: 1 },
  { _id: 'Norma Argentina', numOfMovies: 2 },
  { _id: 'Peter Apostolopoulos', numOfMovies: 1 },
  { _id: 'Josh Gad', numOfMovies: 4 },
  { _id: 'Travis Schafer', numOfMovies: 1 },
  { _id: 'Sergey Kolesov', numOfMovies: 1 },
  { _id: 'Anatoli Petrov', numOfMovies: 1 },
  { _id: 'Andrew Dice Clay', numOfMovies: 3 },
  { _id: 'Riz Abbasi', numOfMovies: 1 },
  { _id: 'Louise Lemoine Torres', numOfMovies: 1 },
  { _id: 'Jackson Bond', numOfMovies: 1 },
  { _id: 'Martin Bashir', numOfMovies: 1 },
  { _id: 'Dong-gun Jang', numOfMovies: 8 },
  { _id: 'DJ Q-Bert', numOfMovies: 1 },
  { _id: 'Alexandra Beaujard', numOfMovies: 1 },
```

Class Exercise

Northeastern University

- Given the books collection
- Find out the 5 most prolific authors
  - authors who wrote the highest number of books


```
{
  "_id": 3,
  "title": "Specification by Example",
  "isbn": "1617290084",
  "pageCount": 0,
  "publishedDate": {
    "$date": "2011-06-03T07:00:00Z"
  },
  "thumbnailUrl":
"https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-
thumb-images/adzic.jpg",
  "status": "PUBLISH",
  "authors": [
    "Gojko Adzic"
  ],
  "categories": [
    "Software Engineering"
  ]
}
```

Solution

Northeastern University

- Create a pipeline with the following steps:
  - Project the authors out of each book
  - Deconstruct the authors array to output a document for each author
  - Group the authors by name, counting the number of occurrences
  - Sort the authors by the occurrence count, descending
  - Limit results to the first 5

```
booksdb> db.books.aggregate([
... {$project: {'authors': 1}},
... {$unwind: '$authors'},
... {$group: {'_id': '$authors', 'count': {$sum: 1}}},
... {$sort: {'count': -1}},
... {$limit: 5}
... ])
[
  { _id: '', count: 59 },
  { _id: 'Vikram Goyal', count: 12 },
  { _id: 'Richard Siddaway', count: 6 },
  { _id: 'Don Jones', count: 6 },
  { _id: 'Christian Bauer', count: 5 }
]
```

Solution

Northeastern University

- To get rid of the empty authors, we can add a $match expression after $unwind:

```
booksdb> db.books.aggregate([
... {$project: {'authors': 1}},
... {$unwind: '$authors'},
... {$match: {'authors': {$ne: ''}}},
... {$group: {'_id': '$authors', 'count': {$sum: 1}}},
... {$sort: {'count': -1}},
... {$limit: 5}
... ])
[
  { _id: 'Vikram Goyal', count: 12 },
  { _id: 'Don Jones', count: 6 },
  { _id: 'Richard Siddaway', count: 6 },
  { _id: 'Jon Skeet', count: 5 },
  { _id: 'Yehuda Katz', count: 5 }
]
```

Inserting Documents

Northeastern University

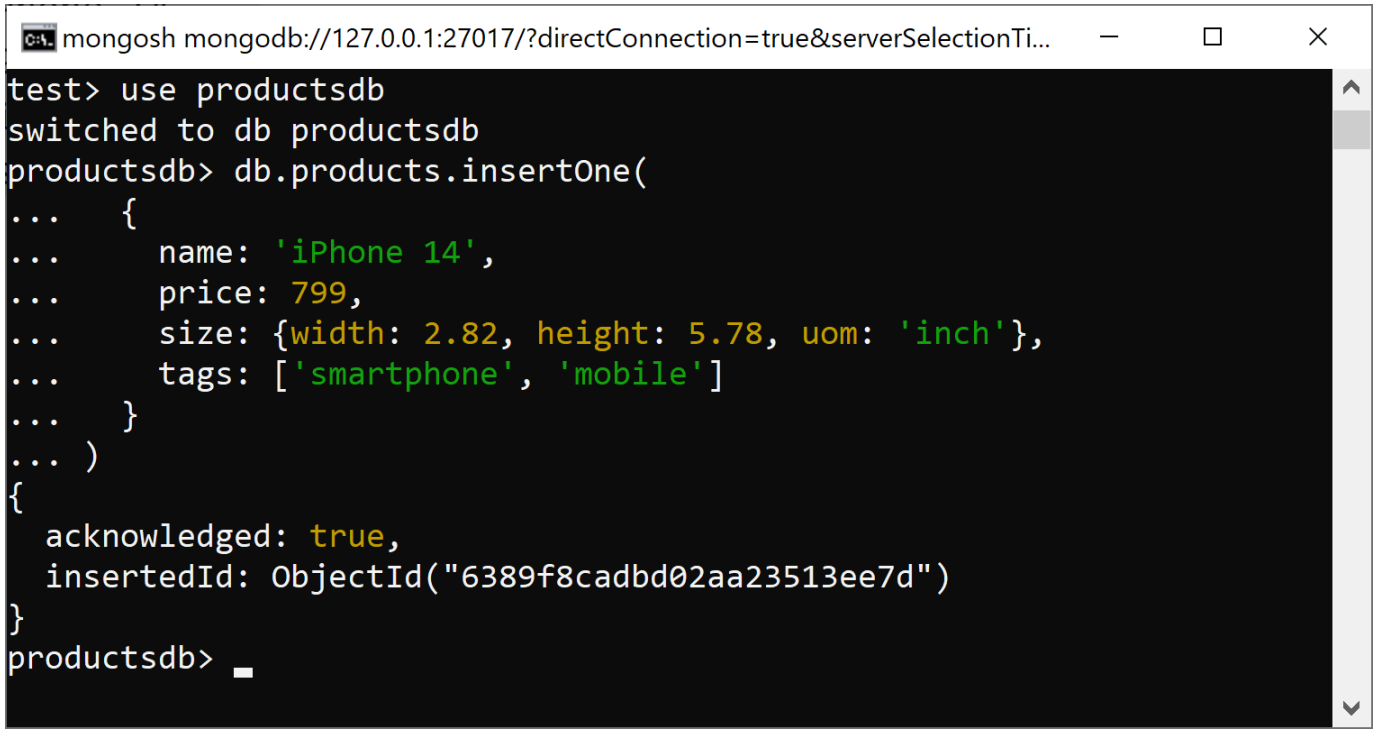
- MongoDB provides two methods for inserting documents
  - **db.collection.insertOne()** inserts a single document into a collection
  - **db.collection.insertMany()** inserts multiple documents into a collection
- If the collection doesn't currently exist, insert operations will create the collection

```
db.users.insertOne(          <-- collection
   {
      name: "sue",           <-- field: value
      age: 26,               <-- field: value
      status: "pending"      <-- field: value
   }                         } document
)
```

Inserting Documents

Northeastern University

- For example, let's insert a document into a collection called products in productsdb



Updating Documents

Northeastern University

- MongoDB provides the following methods for updates:
  - **db.collection.updateOne**(<filter>, <update>, <options>)
    - Updates the first document that matches the specified filter
  - **db.collection.updateMany**(<filter>, <update>, <options>)
    - Updates all documents that match the specified filter
  - **db.collection.replaceOne**(<filter>, <update>, <options>)
    - Replaces a single document in the collection based on the filter
- The <filter> parameter specifies the selection criteria for the update
- The <update> parameter specifies the modifications to apply

```
db.users.updateMany(                    ← collection
   { age: { $lt: 18 } },                ← update filter
   { $set: { status: "reject" } }       ← update action
)
```

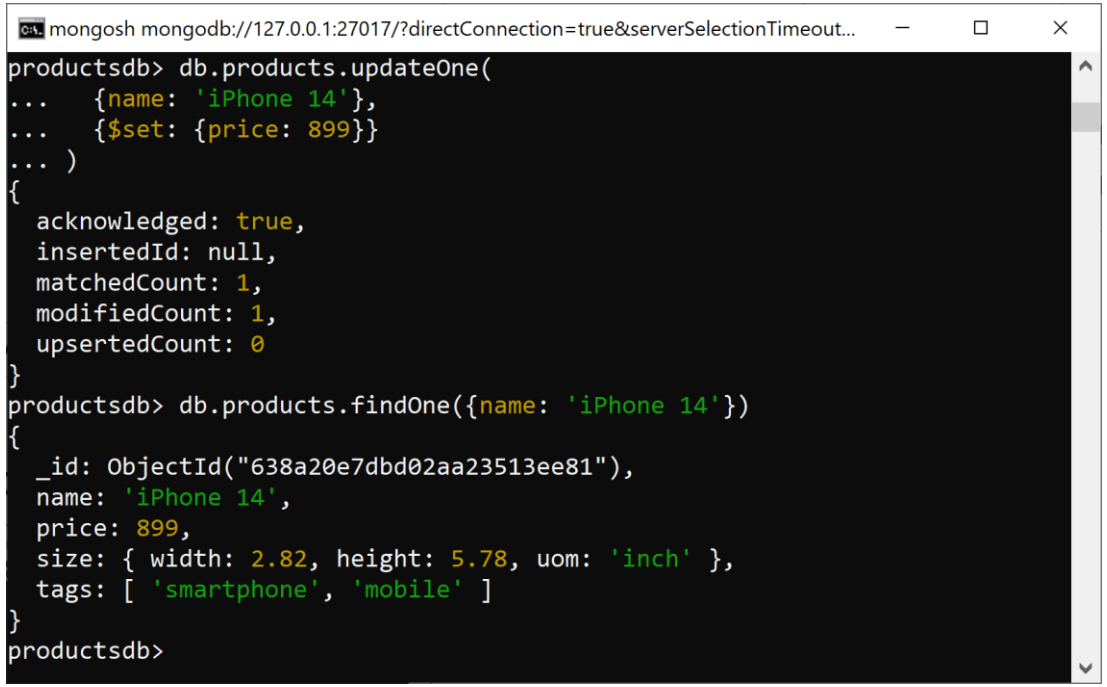
$set Operator

Northeastern University

- The **$set** operator replaces the value of a field with the specified value

```
{$set: {<field1>: <value1>, ...}}
```

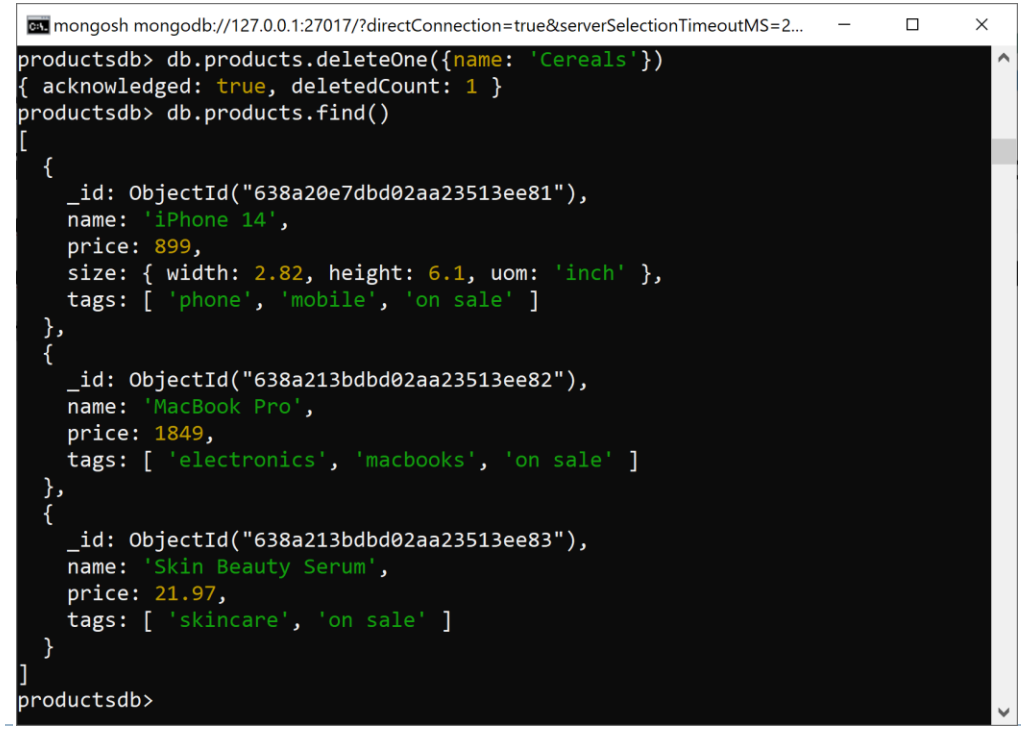
  - If the field doesn't exist, it will be added with the specified value
- For example, let's change the price of iPhone 14 to $899:



Deleting Documents

Northeastern University

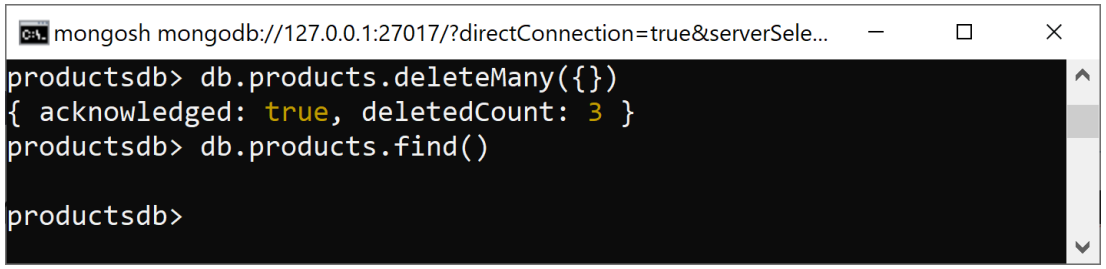
- MongoDB provides two methods for deleting documents from a collection:
  - db.collection.deleteOne(<filter>) - deletes a the first document that matches the filter
  - db.collection.deleteMany(<filter>) - deletes all documents that match the specified filter
- For example, let's delete the Cereals product from the collection



Deleting All Documents

Northeastern University

- To delete all documents from a collection, pass an empty filter {} to deleteMany()



- To delete an entire collection, it is faster to call **db.collection.drop()**
- To drop the current database call **db.dropDatabase()**
- Note: There is no way to undo a delete or drop operation

Running Scripts with the Shell

Northeastern University

- In addition to using the shell interactively, you can also run JavaScript files in the shell
- For example, the following script adds a new movie to our movies collection:

```
// script1.js
db = db.getSisterDB("movies"); // the equivalent of use movies

db.movies.insertOne({
    "title": "Johnny English Strikes Again",
    "director": "David Kerr",
    "year": 2018
});
print("Movie added succesfully");
```

- Then simply pass in the script at the command line:

```
c:\data>mongo script1.js
MongoDB shell version v4.0.0
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.0.0
Movie added succesfully
```

Indexes

Northeastern University

- Indexes allow efficient execution of queries in MongoDB
- However, indexes also make write operations (e.g., inserts and updates) slower
- An index stores the values of a field or a set of fields in a tree-based data structure
- MongoDB creates a default unique index on the _id field of every collection
- For example, let's create a collection with 1 million documents

```
usersdb> for (var i = 0; i < 1000000; i++) {
... db.users.insertOne({
..... 'username': 'user_' + i,
..... 'created': new Date()
..... });
... }
{
  acknowledged: true,
  insertedId: ObjectId("60fda0c704a69d217bb3ca3a")
}
```

Index Example

Northeastern University

- Let's search for a specific user in the collection
- We can use the **explain()** cursor method to see MongoDB's query execution plan
  - The **executionStats** mode gives you statistics about the execution time

```
usersdb> db.users.find({'username': 'user_123'}).explain('executionStats')
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'usersdb.users',
    indexFilterSet: false,
    parsedQuery: { username: { '$eq': 'user_123' } },
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { username: { '$eq': 'user_123' } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
```

```
executionStats: {
  executionSuccess: true,
  nReturned: 1,
  executionTimeMillis: 398,
  totalKeysExamined: 0,
  totalDocsExamined: 1000000,
  executionStages: {
    stage: 'COLLSCAN',
    filter: { username: { '$eq': 'user_123' } },
    nReturned: 1,
    executionTimeMillisEstimate: 13,
    works: 1000002,
    advanced: 1,
    needTime: 1000000,
    needYield: 0,
    saveState: 1000,
    restoreState: 1000,
    isEOF: 1,
    direction: 'forward',
    docsExamined: 1000000
  }
},
```

Creating an Index

Northeastern University

- Let's now create an index on the username field

```
usersdb> db.users.createIndex({'username': 1})
username_1
```

Specifies the sorting direction of the index

- Let's now repeat the same query:

```
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 6,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
    executionStages: {
      stage: 'FETCH',
      nReturned: 1,
      executionTimeMillisEstimate: 0,
      works: 2,
      advanced: 1,
      needTime: 0,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      docsExamined: 1,
      alreadyHasObj: 0,
```

Execution time dropped to 6ms!

Only one document was examined

PyMongo

Northeastern University

- **PyMongo** is the official MongoDB Python driver for MongoDB
  - You can install it using pip install pymongo
- To connect to MongoDB, you first need to create a MongoClient instance:

```
from pymongo import MongoClient
```

```
client = MongoClient()
```

  - By default, it will connect to localhost and port 27017
- You can access a specific database by using the dot notation on the client object:

```
db = client.moviesdb
```

- To get a specific collection, use the dot notation on the database object:

```
movies = db.movies
```

Getting a Single Document

Northeastern University

- In PyMongo documents are represented as dictionaries
- To get a single document matching a query use the method **find_one()**
  - Note the difference in spelling from the MongoDB shell command findOne()
- For example, to display the display the title and plot of the movie Titanic:

```
db.movies.find_one(
    {'title': 'Titanic'},
    {'title': 1, 'plot': 1, '_id': 0}
)
```

```
{'plot': 'An unhappy married couple deal with their problems on board
the ill-fated ship.',
 'title': 'Titanic'}
```

Querying for More Than One Document

Northeastern University

- To query for more than one document, you can use the **find()** method
- find() returns a **Cursor** object, which is used to iterate over all matching documents
- For example, print the title and release year directed by Martin Scorsese

```
cursor = db.movies.find({'directors': 'Martin Scorsese'})
for movie in cursor:
    print(f"{movie['title']} ({movie['year']})")
```

```
The Big Shave (1968)
Who's That Knocking at My Door (1967)
Mean Streets (1973)
Alice Doesn't Live Here Anymore (1974)
Taxi Driver (1976)
New York, New York (1977)
The Last Waltz (1978)
Raging Bull (1980)
The King of Comedy (1982)
After Hours (1985)
```

Query Operators

Northeastern University

- You can use any of the query operators supported by MongoDB
- Example: print all the movies directed by Martin Scorsese after 2000, sorted by year

```
cursor = db.movies.find(
    {'directors': 'Martin Scorsese', 'year': {'$gte': 2000}}
).sort('year')
for movie in cursor:
    print(f"{movie['title']} ({movie['year']})")
```

```
My Voyage to Italy (2001)
My Voyage to Italy (2001)
Gangs of New York (2002)
The Aviator (2004)
The Departed (2006)
Shine a Light (2008)
A Letter to Elia (2010)
Shutter Island (2010)
Public Speaking (2010)
A Letter to Elia (2010)
Hugo (2011)
George Harrison: Living in the Material World (2011)
The Wolf of Wall Street (2013)
```

Inserting a Document

Northeastern University

- To insert a document into a collection you can use the **insert_one()** method
- For example, let's add the following movie to the movies collection

```
new_movie = {
    'title': 'Black Widow',
    'year': 2021,
    'rated': 'PG-13',
    'director': 'Cate Shortland',
    'cast': ['Scarlett Johansson', 'Florence Pugh', 'David Harbour']
}

movie_id = db.movies.insert_one(new_movie)
movie_id.inserted_id
```

```
ObjectId('638b19f2ac1ffeedc04d38d3')
```

- Python native data types (like datetime.datetime) will be automatically converted to the appropriate BSON types

Deleting Documents

Northeastern University

- To delete one document, use the **delete_one()** method
- To delete more than one document, use the **delete_many()** method
- For example, let's delete the movie 'Black Widow' from the collection:

```
db.movies.delete_one({'title': 'Black Widow'})
```

```
<pymongo.results.DeleteResult at 0x1ca19e37730>
```

```
list(db.movies.find({'year': 2021}))
```

```
[]
```