



Northeastern  
University

# DS 5110 – Lecture 7

## MongoDB

Roi Yehoshua

# Agenda

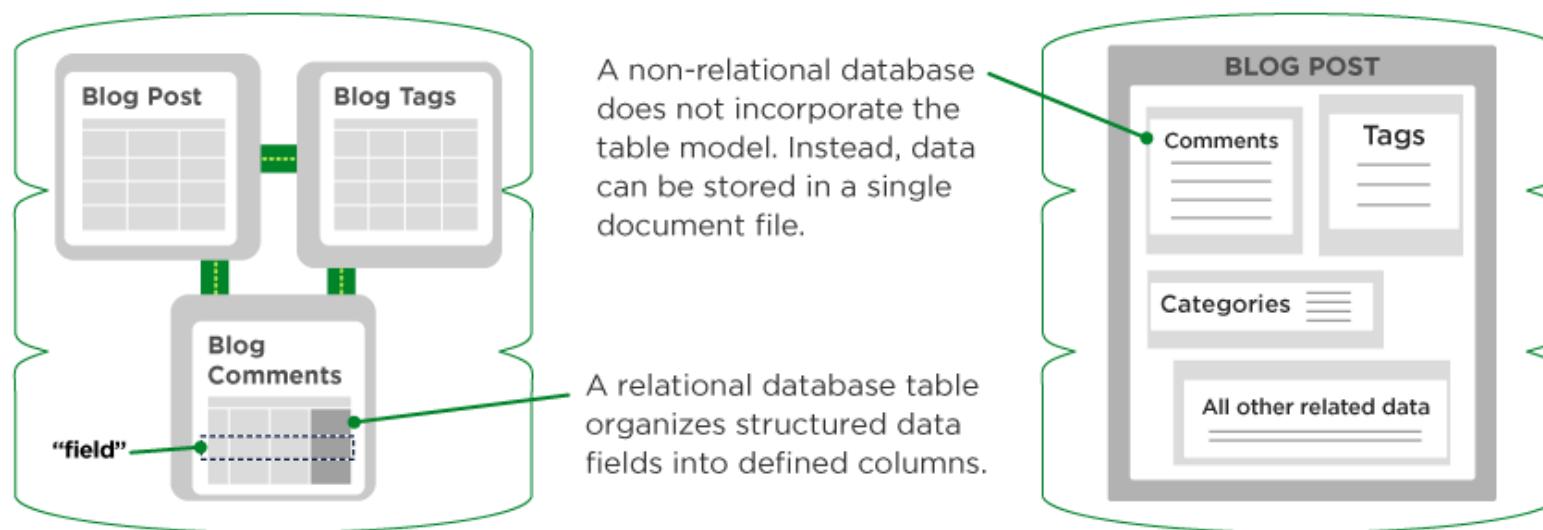
---

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



# NoSQL Databases

- ▶ **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
<b>Model</b>	Non-relational Data is stored in JSON documents, key/value pairs, wide column stores, or graphs	Relational Data is stored in tables
<b>Schema</b>	Dynamic or flexible schemas The schema can be changed on the fly	Strict schema Schema changes requires taking the entire database offline
<b>Data</b>	Various data types – structured, semi-structured and unstructured data	Only structured data
<b>Scaling</b>	Horizontal scaling Can add more machines or cloud instances	Vertical scaling Need to make the server more powerful
<b>Transactions</b>	Doesn't support the ACID properties	Supports ACID transactions
<b>Development Model</b>	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

Document Database	Graph Databases
Wide Column Stores	Key-Value Databases

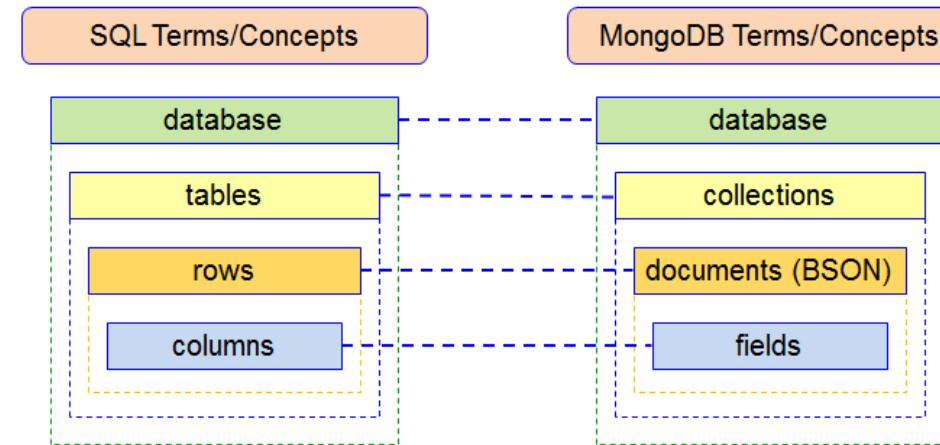
Store data tables, rows, and dynamic columns

Store data in nodes and edges

Each item contains keys and values

@cloudtxt <http://www.aryannava.com>

- ▶ 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

- ▶ 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

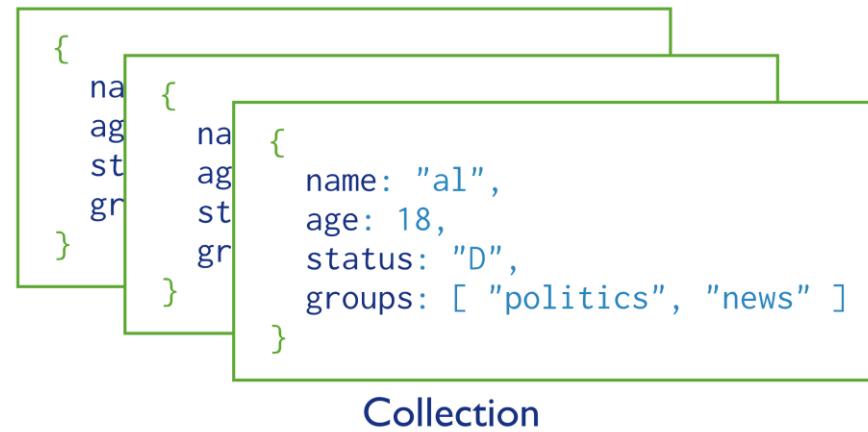
- ▶ BSON (Binary JSON) is a binary representation of JSON documents
- ▶ Designed to represent data efficiently, and allow fast encoding/decoding
- ▶ BSON supports more data types than JSON:

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

- ▶ MongoDB stores documents in collections

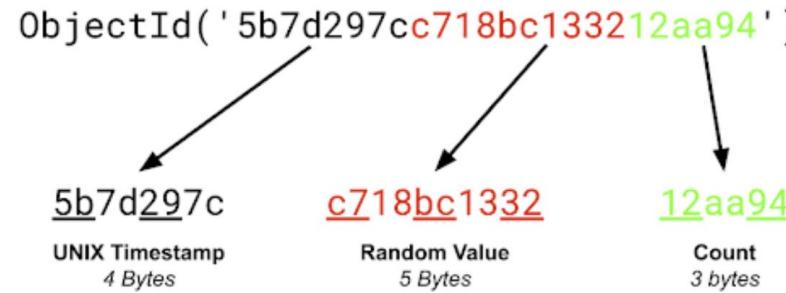


- ▶ Documents in the same collections don't need to have the same schema
  - ▶ i.e., they can have different fields and data types
- ▶ Databases and collections in MongoDB are created **lazily**
  - ▶ i.e., they are created when the first document is inserted into them



# ObjectIds

- ▶ 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

- ▶ 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/>

The screenshot shows the MongoDB website's product selection interface. On the left, a sidebar lists various MongoDB products: MongoDB Atlas, MongoDB Enterprise Advanced, MongoDB Community Edition, MongoDB Community Server (which is highlighted in blue), MongoDB Community, Kubernetes Operator, Tools, and Mobile & Edge. The main content area is titled "MONGODB COMMUNITY SERVER" and "MongoDB Community Server Download". It describes the Community version as a flexible document data model with ad-hoc queries, secondary indexing, and real-time aggregations. It also mentions MongoDB Atlas as a managed service with auto-scaling, serverless instances, full-text search, and data distribution across regions and clouds. A call-to-action button says "Give it a try with a free, highly-available 512 MB cluster." At the bottom, a dropdown menu shows the current version as "6.0.2 (current)".



# MongoDB Shell (mongosh)

- ▶ mongosh is an interactive JavaScript interface to MongoDB
- ▶ Download the shell from the [MongoDB Download Center](#)
- ▶ After installation run **mongosh** from the command line:

The screenshot shows a terminal window titled "mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000". The window displays the following text:

```
C:\Users\roi_y>mongosh
Current Mongosh Log ID: 637046823547d387201cfab3
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.6.0
Using MongoDB:    6.0.2
Using Mongosh:    1.6.0

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

-----
The server generated these startup warnings when booting
2022-11-12T20:08:03.626-05:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----

-----
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

-----
Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.

test>
```



# MongoDB Shell (mongosh)

- ▶ 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

- ▶ 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

- ▶ 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

TOOLS

## MongoDB Command Line Database Tools Download

The MongoDB Database Tools are a collection of command-line utilities for working with a MongoDB deployment. These tools release independently from the MongoDB Server schedule enabling you to receive more frequent updates and leverage new features as soon as they are available. See the [MongoDB Database Tools](#) documentation for more information.

Version  
100.6.1

Platform  
Windows x86\_64

Package  
msi

[Download](#) More Options

A screenshot of the MongoDB Command Line Database Tools download page. It shows a dropdown menu for Version (100.6.1), Platform (Windows x86\_64), and Package (msi). At the bottom, there is a green 'Download' button with a downward arrow icon and a 'More Options' button with three dots.

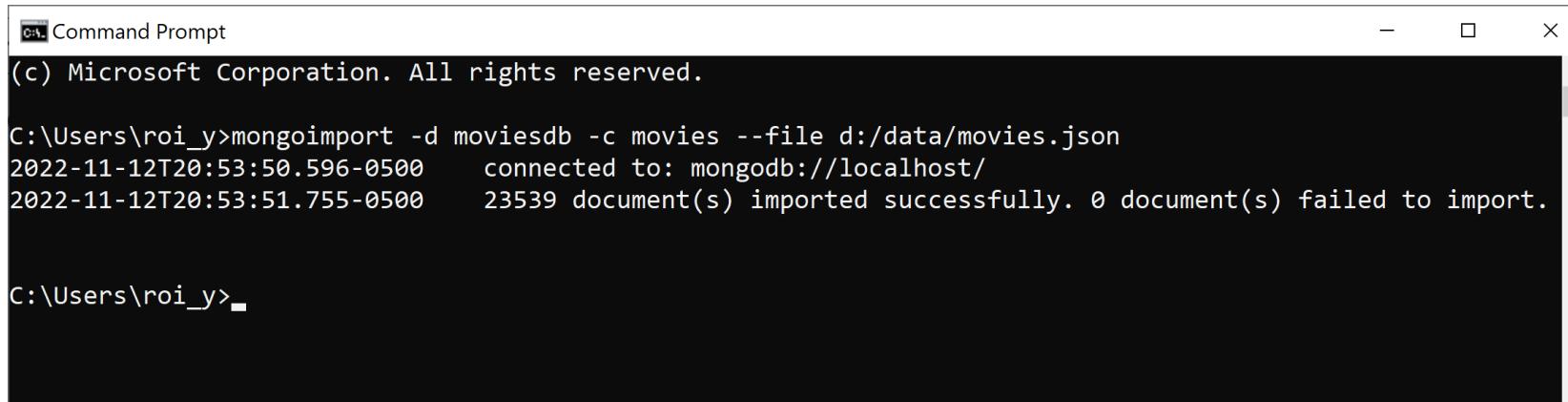


# mongoimport

- ▶ 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



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text output:

```
(c) Microsoft Corporation. All rights reserved.  
C:\Users\roi_y>mongoimport -d moviesdb -c movies --file d:/data/movies.json  
2022-11-12T20:53:50.596-0500      connected to: mongodb://localhost/  
2022-11-12T20:53:51.755-0500      23539 document(s) imported successfully. 0 document(s) failed to import.  
C:\Users\roi_y>
```



# Displaying All Documents

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
test> use moviesdb
switched to db moviesdb
moviesdb> db.movies.find()
[
  {
    _id: ObjectId("573a1390f29313caabcd4135"),
    plot: 'Three men hammer on an anvil and pass a bottle of beer around.',
    genres: [ 'Short' ],
    runtime: 1,
    cast: [ 'Charles Kayser', 'John Ott' ],
    num_mflix_comments: 1,
    title: 'Blacksmith Scene',
    fullplot: 'A stationary camera looks at a large anvil with a blacksmith behind it and one on either side. The smith in the middle draws a heated metal rod from the fire, places it on the anvil, and all three begin a rhythmic hammering. After several blows, the metal goes back in the fire. One smith pulls out a bottle of beer, and they each take a swig. Then, out comes the glowing metal and the hammering resumes.',
    countries: [ 'USA' ],
    released: ISODate("1893-05-09T00:00:00.000Z"),
    directors: [ 'William K.L. Dickson' ],
    rated: 'UNRATED',
    awards: { wins: 1, nominations: 0, text: '1 win.' },
    lastupdated: '2015-08-26 00:03:50.133000000',
    year: 1893,
    imdb: { rating: 6.2, votes: 1189, id: 5 },
    type: 'movie',
    tomatoes: {
      viewer: { rating: 3, numReviews: 184, meter: 32 },
      lastUpdated: ISODate("2015-06-28T18:34:09.000Z")
    }
  }
]
```



# MongoDB Compass

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

The screenshot shows the MongoDB Compass application interface. On the left, the sidebar displays the connection details: 'localhost:27017', '4 DBS', '4 COLLECTIONS', and the current database 'moviesdb'. Below this, there are sections for 'My Queries', 'Databases', and a 'Filter your data' search bar. The main area is titled 'moviesdb.movies' and shows the 'Documents' tab selected. A search bar at the top of the documents list contains the query '{title: "Titanic"}'. To the right, it shows '23.5k DOCUMENTS' and '1 INDEXES'. The results list displays two movie documents:

```
_id: ObjectId('573a1394f29313caabcf639')
plot: "An unhappy married couple deal with their problems on board the ill-fa..."
> genres: Array
  runtime: 98
  rated: "NOT RATED"
> cast: Array
  num_mflix_comments: 1
  poster: "https://m.media-amazon.com/images/M/MV5BMTU3NTUyMTc3N15BMl5BanBnXkFtZT..."
  title: "Titanic"
  fullplot: "Unhappily married and uncomfortable with life among the British upper ..."
> languages: Array
  released: 1953-07-13T00:00:00.000+00:00
> directors: Array
> writers: Array
> awards: Object
  lastupdated: "2015-09-16 00:00:16.593000000"
  year: 1953
> imdb: Object
> countries: Array
  type: "movie"
> tomatoes: Object

_id: ObjectId('573a139af29313caabcefbd1')
plot: "The story of the 1912 sinking of the largest luxury liner ever built, ..."
> genres: Array
  runtime: 173
  ...
```

# 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

- ▶ 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

```
mongosh mongoDB://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({year: 2010})
[
  {
    _id: ObjectId("573a139cf29313caabcf6fb1"),
    plot: 'A lonely obese nurse, working at a hospital terminal ward, is reminded of her childhood friend Adrienn Pal and wants to track her down.',
    genres: [ 'Drama' ],
    runtime: 136,
    cast: [ 'Éva Gébor', 'István Zhamenék', 'Ékos Horváth', 'Lia Pokorny' ],
    num_flix_comments: 1,
    poster: 'https://m.media-amazon.com/images/M/MV5BODNjZmM3MWYtYmUwZC00M2RjLTkyZmEtYzRhMDU1NmM3ZmIzXkEyXkFqcGdeQXVyOTAzODAwQQ@@_V1_SY1000_SX677_AL_.jpg',
    title: 'Pél Adrienn',
    fullplot: 'A lonely obese nurse, working at a hospital terminal ward, is reminded of her childhood friend Adrienn Pal and wants to track her down.',
    languages: [ 'Hungarian' ],
    released: ISODate("2011-03-17T00:00:00.000Z"),
    directors: [ 'Égnes Kocsis' ],
    writers: [ 'Égnes Kocsis (screenplay)', 'Andrea Roberti (screenplay)' ],
    awards: { wins: 4, nominations: 5, text: '4 wins & 5 nominations.' },
    lastupdated: '2015-09-01 01:00:29.427000000',
    year: 2010,
    imdb: { rating: 6.8, votes: 348, id: 146592 },
    countries: [ 'Hungary', 'France', 'Austria', 'Netherlands' ],
    type: 'movie',
    tomatoes: {
      viewer: { rating: 3.5, numReviews: 40, meter: 67 },
      lastUpdated: ISODate("2015-07-22T18:18:12.000Z")
    }
  },
]
```



# Exact Matching

- ▶ 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:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({year: 2010, rated: 'PG-13'})
[
  {
    _id: ObjectId("573a13aaaf29313caabd21654"),
    fullplot: 'Marcus is a popular massage therapist who struggles with parasomnia, a severe sleepwalking disorder that causes him to do things in his sleep that he cannot remember the next day. When he wakes up with blood on his hands and a knife at his side, he is startled to hear that a close friend has been found stabbed to death. Marcus frantically tries to put the pieces together - could he have murdered his friend while sleepwalking to hide a dark secret between them? The police close in as Marcus investigates his own nocturnal activities, desperate to figure out what happens after he goes to sleep. His journey to uncover the truth leads him to a shocking revelation.',
    imdb: { rating: 5.5, votes: 1629, id: 326965 },
    year: 2010,
    plot: 'Marcus is a popular massage therapist who struggles with parasomnia, a severe sleepwalking disorder that causes him to do things in his sleep that he cannot remember the next day. When he ...',
    genres: [ 'Drama', 'Mystery', 'Thriller' ],
    rated: 'PG-13',
    metacritic: 33,
    title: 'In My Sleep',
    lastupdated: '2015-08-18 00:37:59.593000000',
    languages: [ 'English' ],
    writers: [ 'Allen Wolf' ],
    type: 'movie',
    tomatoes: {
      website: 'http://www.inmysleep.com/',
      viewer: { rating: 2.7, numReviews: 457, meter: 33 },
      dvd: ISODate("2010-10-01T00:00:00.000Z"),
      critic: { rating: 3.6, numReviews: 17, meter: 12 },
      lastUpdated: ISODate("2015-09-15T17:08:35.000Z"),
      rotten: 15,
      production: 'Morning Star Pictures',
    }
  }
]
```



# findOne()

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.findOne({title: 'Titanic'})
{
  _id: ObjectId("573a1394f29313caabcf639"),
  plot: 'An unhappy married couple deal with their problems on board the ill-fated ship.',
  genres: [ 'Drama', 'History', 'Romance' ],
  runtime: 98,
  rated: 'NOT RATED',
  cast: [
    'Clifton Webb',
    'Barbara Stanwyck',
    'Robert Wagner',
    'Audrey Dalton'
  ],
  num_mflix_comments: 1,
  poster: 'https://m.media-amazon.com/images/M/MV5BMTU3NTUyMTc3Nl5BMl5BanBnXkFtZTgwOTA2MDE3MTE@._V1_SY1000_SX677_AL_.jpg',
  title: 'Titanic',
  fullplot: 'Unhappily married and uncomfortable with life among the British upper crust, Julia Sturges takes her two children and boards the Titanic for America. Her husband Richard also arranges passage on the doomed luxury liner in order to let him have custody of their two children. Their problems soon seem minor when the ship hits an iceberg.',
  languages: [ 'English', 'Basque', 'French', 'Spanish' ],
  released: ISODate("1953-07-13T00:00:00.000Z"),
  directors: [ 'Jean Negulesco' ],
  writers: [ 'Charles Brackett', 'Walter Reisch', 'Richard L. Breen' ],
  awards: {
    wins: 0,
    nominations: 3,
    text: 'Won 1 Oscar. Another 2 nominations.'
  },
  lastupdated: '2015-09-16 00:00:16.593000000',
```



# ObjectId

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.findOne({_id: ObjectId("573a1394f29313caabcf639")})
{
  _id: ObjectId("573a1394f29313caabcf639"),
  plot: 'An unhappy married couple deal with their problems on board the ill-fated ship.',
  genres: [ 'Drama', 'History', 'Romance' ],
  runtime: 98,
  rated: 'NOT RATED',
  cast: [
    'Clifton Webb',
    'Barbara Stanwyck',
    'Robert Wagner',
    'Audrey Dalton'
  ],
  num_mflix_comments: 1,
  poster: 'https://m.media-amazon.com/images/M/MV5BMTU3NTUyMTc3Nl5BMl5BanBnXkFtZTgwOTA2MDE3MTE@._V1_SY1000_SX677_AL_.jpg',
  title: 'Titanic',
  fullplot: 'Unhappily married and uncomfortable with life among the British upper crust, Julia Sturges takes her two children and boards the Titanic for America. Her husband Richard also arranges passage on the doomed luxury liner in order to let him have custody of their two children. Their problems soon seem minor when the ship hits an iceberg.',
  languages: [ 'English', 'Basque', 'French', 'Spanish' ],
  released: ISODate("1953-07-13T00:00:00.000Z"),
  directors: [ 'Jean Negulesco' ],
  writers: [ 'Charles Brackett', 'Walter Reisch', 'Richard L. Breen' ],
  awards: {
    wins: 0,
    nominations: 3,
    text: 'Won 1 Oscar. Another 2 nominations.'
  },
  lastupdated: '2015-09-16 00:00:16.593000000',
```



# Projection

- ▶ 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

A screenshot of a terminal window titled "mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=..." showing the execution of a MongoDB find command. The command filters the "movies" collection and projects only the "title" and "year" fields. The output shows three documents, each with an "\_id" field and the specified "title" and "year".

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS...
moviesdb> db.movies.find({}, {title: 1, year: 1})
[{
  _id: ObjectId("573a1390f29313caabcd4135"),
  title: 'Blacksmith Scene',
  year: 1893
},
{
  _id: ObjectId("573a1390f29313caabcd42e8"),
  title: 'The Great Train Robbery',
  year: 1903
},
{
  _id: ObjectId("573a1390f29313caabcd4323"),
  title: 'The Land Beyond the Sunset',
  year: 1912
}]
```



# \_id Field Projection

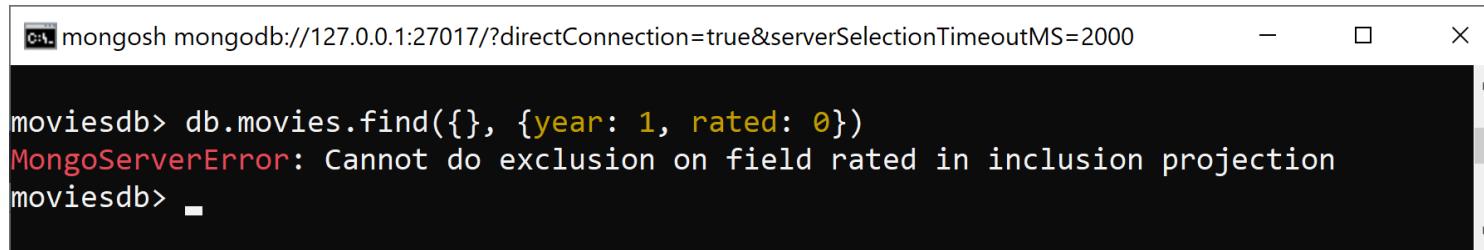
- ▶ The \_id field is included in the returned documents by default
  - ▶ unless you explicitly specify \_id: 0 in the projection to suppress the field

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({}, {title: 1, year: 1, _id: 0})
[
  { title: 'Blacksmith Scene', year: 1893 },
  { title: 'The Great Train Robbery', year: 1903 },
  { title: 'The Land Beyond the Sunset', year: 1912 },
  { title: 'A Corner in Wheat', year: 1909 },
  {
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics',
    year: 1911
  },
  { title: 'Traffic in Souls', year: 1913 },
  { title: 'Gertie the Dinosaur', year: 1914 },
  { title: 'In the Land of the Head Hunters', year: 1914 },
  { title: 'The Poor Little Rich Girl', year: 1917 },
  { title: 'The Immigrant', year: 1917 },
  { title: 'Les vampires', year: 1915 },
  { title: 'The Birth of a Nation', year: 1915 },
  { title: 'Wild and Woolly', year: 1917 },
  { title: 'The Blue Bird', year: 1918 },
  { title: 'From Hand to Mouth', year: 1919 },
  {
    title: 'Broken Blossoms or The Yellow Man and the Girl',
    year: 1919
  },
  { title: 'High and Dizzy', year: 1920 },
  moviesdb>
  { title: 'One Week', year: 1920 },
```



# Inclusion or Exclusion

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



A screenshot of a terminal window titled "mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000". The window shows a MongoDB shell session. The user runs the command `db.movies.find({}, {year: 1, rated: 0})`, which results in a `MongoServerError`: "Cannot do exclusion on field rated in inclusion projection".

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({}, {year: 1, rated: 0})
MongoServerError: Cannot do exclusion on field rated in inclusion projection
moviesdb>
```



# Projecting New Fields

- ▶ 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**:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({}, {name: '$title'})
[
  {
    _id: ObjectId("573a1390f29313caabcd4135"),
    name: 'Blacksmith Scene'
  },
  {
    _id: ObjectId("573a1390f29313caabcd42e8"),
    name: 'The Great Train Robbery'
  },
  {
    _id: ObjectId("573a1390f29313caabcd4323"),
    name: 'The Great Train Robbery'
  }
]
```



# Projecting New Fields

- ▶ 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 },
  {
    title: 'The Story of a Man', rating: 6.8
  }
]
```



# Query Operators

- ▶ 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

- ▶ Example: find all movies that were released after 2015

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({'year': {$gt: 2015}})
[
  {
    _id: ObjectId("573a13e6f29313caabdc6a9a"),
    plot: 'The journey of a professional wrestler who becomes a small town pastor and moonlights as a masked vigilante fighting injustice. While facing crises at home and at the church, the Pastor ...',
    genres: [ 'Action', 'Biography', 'Crime' ],
    runtime: 111,
    title: 'The Masked Saint',
    num_mflix_comments: 5,
    poster: 'https://m.media-amazon.com/images/M/MV5BMjI0NzU4MjkxNl5BMl5BanBnXkFtZTgwMTYxMTA1NzE@._V1_SY1000_SX677_AL_.jpg',
    countries: [ 'Canada' ],
    fullplot: 'The journey of a professional wrestler who becomes a small town pastor and moonlights as a masked vigilante fighting injustice. While facing crises at home and at the church, the Pastor must evade the police and somehow reconcile his violent secret identity with his calling as a pastor.',
    languages: [ 'English' ],
    cast: [
      'Brett Granstaff',
      'Lara Jean Chorostecki',
      'T.J. McGibbon',
      'Diahann Carroll'
    ],
    directors: [ 'Warren P. Sonoda' ],
    writers: [ 'Scott Crowell', 'Scott Crowell', 'Brett Granstaff' ],
    awards: { wins: 1, nominations: 3, text: '1 win & 3 nominations.' },
    lastupdated: '2015-09-01 01:13:10.960000000',
    year: 2016,
    imdb: { rating: '', votes: '', id: 3103166 },
    type: 'movie',
  }
]
```



# Query Operators

- ▶ 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

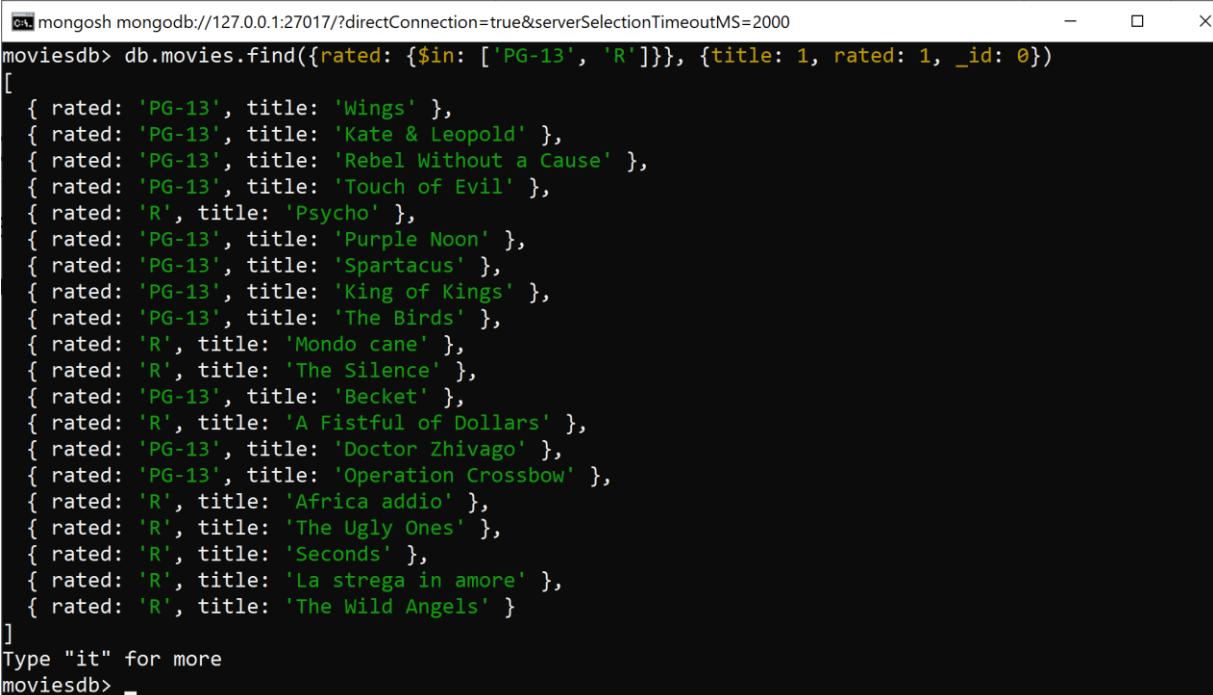
- ▶ Example: find all the movies that are not rated PG-13

```
c:\mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({rated: {$ne: 'PG-13'}}, {title: 1, rated: 1, _id: 0})
[
  { title: 'Blacksmith Scene', rated: 'UNRATED' },
  { title: 'The Great Train Robbery', rated: 'TV-G' },
  { rated: 'UNRATED', title: 'The Land Beyond the Sunset' },
  { title: 'A Corner in Wheat', rated: 'G' },
  {
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics'
  },
  { title: 'Traffic in Souls', rated: 'TV-PG' },
  { title: 'Gertie the Dinosaur' },
  { title: 'In the Land of the Head Hunters' },
  { title: 'The Poor Little Rich Girl' },
  { title: 'The Immigrant', rated: 'UNRATED' },
  { rated: 'NOT RATED', title: 'Les vampires' },
  { rated: 'NOT RATED', title: 'The Birth of a Nation' },
  { title: 'Wild and Woolly' },
  { title: 'The Blue Bird' },
  { rated: 'TV-G', title: 'From Hand to Mouth' },
  {
    rated: 'NOT RATED',
    title: 'Broken Blossoms or The Yellow Man and the Girl'
  },
  { rated: 'PASSED', title: 'High and Dizzy' },
  { rated: 'NOT RATED', title: 'The Last of the Mohicans' },
  { title: 'One Week', rated: 'TV-G' },
  { title: 'The Saphead' }
]
Type "it" for more
moviesdb> _
```



# The \$in Operator

- ▶ 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:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({$in: ['PG-13', 'R']}, {title: 1, rated: 1, _id: 0})
[
  { rated: 'PG-13', title: 'Wings' },
  { rated: 'PG-13', title: 'Kate & Leopold' },
  { rated: 'PG-13', title: 'Rebel Without a Cause' },
  { rated: 'PG-13', title: 'Touch of Evil' },
  { rated: 'R', title: 'Psycho' },
  { rated: 'PG-13', title: 'Purple Noon' },
  { rated: 'PG-13', title: 'Spartacus' },
  { rated: 'PG-13', title: 'King of Kings' },
  { rated: 'PG-13', title: 'The Birds' },
  { rated: 'R', title: 'Mondo cane' },
  { rated: 'R', title: 'The Silence' },
  { rated: 'PG-13', title: 'Becket' },
  { rated: 'R', title: 'A Fistful of Dollars' },
  { rated: 'PG-13', title: 'Doctor Zhivago' },
  { rated: 'PG-13', title: 'Operation Crossbow' },
  { rated: 'R', title: 'Africa addio' },
  { rated: 'R', title: 'The Ugly Ones' },
  { rated: 'R', title: 'Seconds' },
  { rated: 'R', title: 'La strega in amore' },
  { rated: 'R', title: 'The Wild Angels' }
]
Type "it" for more
moviesdb>
```

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



# The \$or Operator

- ▶ 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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({$or: [{year: {$gt: 2010}}, {rated: 'PG-13'}]}, {year: 1, rated: 1, _id: 0})
[
  { rated: 'PG-13', year: 1927 },
  { rated: 'PG-13', year: 2001 },
  { rated: 'PG-13', year: 1955 },
  { rated: 'PG-13', year: 1958 },
  { rated: 'PG-13', year: 1960 },
  { rated: 'PG-13', year: 1960 },
  { rated: 'PG-13', year: 1961 },
  { rated: 'PG-13', year: 1963 },
  { year: 1964, rated: 'PG-13' },
  { rated: 'PG-13', year: 1965 },
  { rated: 'PG-13', year: 1965 },
  { rated: 'PG-13', year: 1967 },
  { rated: 'PG-13', year: 1967 },
  { rated: 'PG-13', year: 1968 },
  { rated: 'PG-13', year: 1968 },
  { year: 1968, rated: 'PG-13' },
  { rated: 'PG-13', year: 1969 }
]
Type "it" for more
moviesdb>
```



# The \$not Operator

- ▶ **\$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: '1981' },
  { title: 'All Passion Spent', year: '1986' },
  { title: 'The Storyteller', year: '1987' },
  { title: "Tanner '88", year: '1988' },
  { title: 'Jack the Ripper', year: '1988' },
  { title: 'Crime and Punishment', year: 2002 },
  { title: 'Babylon 5', year: '1994e1998' },
  { title: 'The West', year: '1996' },
  { title: 'What Is It?', year: 2005 },
  { title: 'The Odyssey', year: '1997' },
  { year: 2001, title: 'Glitter' },
  { year: 2003, title: 'The Manson Family' },
  { title: 'The Dancer Upstairs', year: 2002 },
  { title: 'The Ice House', year: '1997' },
  { 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

- ▶ 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'

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({title: /^M/}, {title: 1})
[
  {
    _id: ObjectId("573a1391f29313caabcd71e3"),
    title: 'Miss Lulu Bett'
  },
  { _id: ObjectId("573a1391f29313caabcd8414"), title: 'Metropolis' },
  {
    _id: ObjectId("573a1391f29313caabcd8e45"),
    title: 'Man with a Movie Camera'
  },
  {
    _id: ObjectId("573a1391f29313caabcd93a3"),
    title: 'Men Without Women'
  },
  { _id: ObjectId("573a1391f29313caabcd93aa"), title: 'Morocco' },
  { _id: ObjectId("573a1392f29313caabcd975d"), title: 'M' },
  {
    _id: ObjectId("573a1392f29313caabcd97a7"),
    title: 'Mädchen in Uniform'
  },
]
```



# The \$exists Operator

- ▶ 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

A screenshot of a terminal window titled "mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000". The window displays a MongoDB query result. The command is "db.movies.find({rated: {\$exists: false}}, {title: 1, \_id: 0})". The output shows a list of movie titles where the 'rated' field does not exist. The titles listed are: Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics, Gertie the Dinosaur, In the Land of the Head Hunters, The Poor Little Rich Girl, Wild and Woolly, The Blue Bird, The Saphead, Within Our Gates, Miss Lulu Bett, Tol'able David, Civilization, and The Perils of Pauline.

```
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' },
]
```

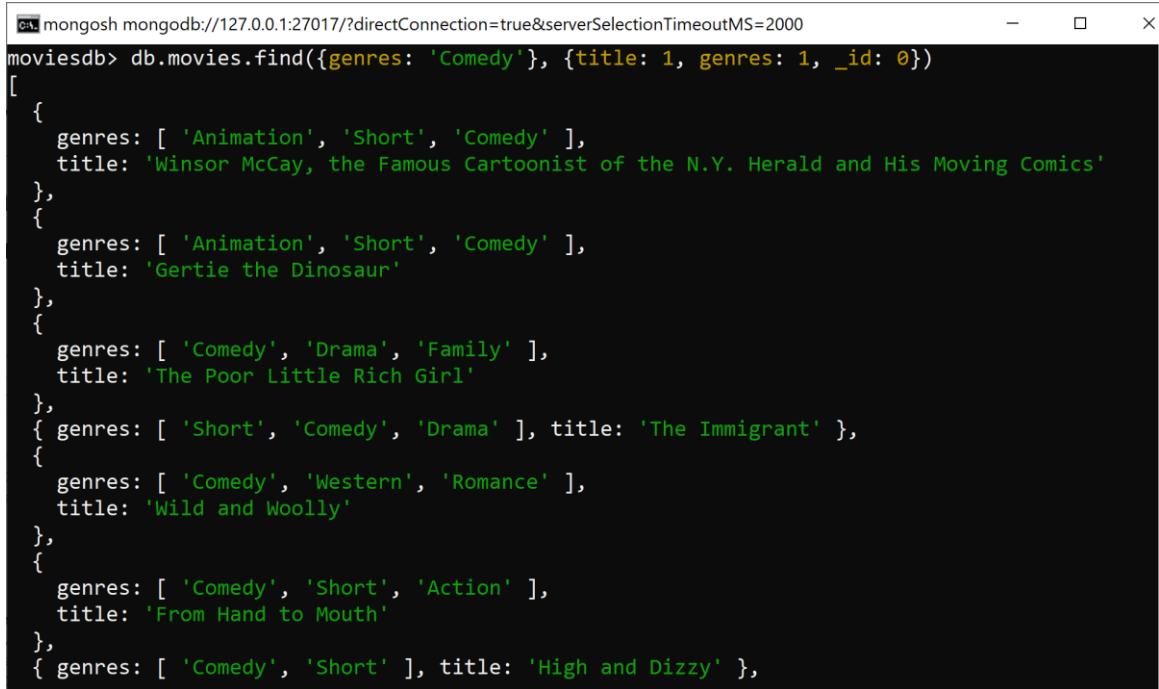


# Querying Arrays

- ▶ 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



A screenshot of a terminal window titled "mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000". The window displays the result of a MongoDB query on the "movies" collection. The query is: db.movies.find({genres: 'Comedy'}, {title: 1, genres: 1, \_id: 0}). The results are listed as an array of documents, each containing the title and genres fields. The titles of the movies are: "Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics", "Gertie the Dinosaur", "The Poor Little Rich Girl", "The Immigrant", "Wild and Woolly", "From Hand to Mouth", and "High and Dizzy".

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({genres: 'Comedy'}, {title: 1, genres: 1, _id: 0})
[
  {
    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'
  },
  {
    genres: [ 'Short', 'Comedy', 'Drama' ],
    title: 'The Immigrant'
  },
  {
    genres: [ 'Comedy', 'Western', 'Romance' ],
    title: 'Wild and Woolly'
  },
  {
    genres: [ 'Comedy', 'Short', 'Action' ],
    title: 'From Hand to Mouth'
  },
  {
    genres: [ 'Comedy', 'Short' ],
    title: 'High and Dizzy'
  }
]
```



# Querying Arrays

- 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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({cast: {$gte: 'U'}}, {title: 1, cast: 1, _id: 0})
[
  {
    cast: [
      'Martin Fuller',
      'Mrs. William Bechtel',
      'Walter Edwin',
      'Ethel Jewett'
    ],
    title: 'The Land Beyond the Sunset'
  },
  {
    cast: [ 'Winsor McCay' ],
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics'
  },
  {
    cast: [ 'Jane Gail', 'Ethel Grandin', 'William H. Turner', 'Matt Moore' ],
    title: 'Traffic in Souls'
  },
  {
    cast: [ 'Winsor McCay', 'George McManus', 'Roy L. McCardell' ],
    title: 'Gertie the Dinosaur'
  },
  {
    cast: [
      'John Barrymore',
      'Lillian Gish',
      'Walter Huston',
      'Norman Kerry',
      'Henry Krauss',
      'Fiona老虎',
      'John Wengraf',
      'John Wengraf'
    ],
    title: 'The Mystery of the Blue Room'
  }
]
```



# The \$elemMatch Operator

- 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

- ▶ 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”

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({genres: {$all: ['Comedy', 'Family']}}, {title: 1, genres: 1, _id: 0})
[
  {
    genres: [ 'Comedy', 'Drama', 'Family' ],
    title: 'The Poor Little Rich Girl'
  },
  { genres: [ 'Comedy', 'Drama', 'Family' ], title: 'The Kid' },
  { genres: [ 'Short', 'Comedy', 'Family' ], title: 'Cops' },
  { genres: [ 'Comedy', 'Family' ], title: 'Our Hospitality' },
  { genres: [ 'Comedy', 'Family', 'Fantasy' ], title: 'Sherlock Jr.' },
  { genres: [ 'Comedy', 'Family', 'Sport' ], title: 'The Freshman' },
  { genres: [ 'Comedy', 'Family', 'Romance' ], title: 'Seven Chances' },
  { genres: [ 'Comedy', 'Drama', 'Family' ], title: 'The Kid Brother' },
  { genres: [ 'Comedy', 'Romance', 'Family' ], title: 'The Cameraman' },
  {
    genres: [ 'Animation', 'Family', 'Comedy' ],
    title: 'The Poor Little Rich Girl'
  }
]
```



# Querying for an Element by the Array Index

- ▶ 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:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({`genres.1': 'Animation'}, {title: 1, genres: 1, _id: 0})
[{"genres": ["Short", "Animation"], "title": "Dots"}, {"genres": ["Documentary", "Animation", "History"], "title": "Victory Through Air Power"}, {"genres": ["Family", "Animation", "Comedy"], "title": "Duck Amuck"}, {"genres": ["Action", "Animation", "Adventure"], "title": "Asterix the Gaul"}, {"genres": ["Documentary", "Animation", "Short"], "title": "Why Man Creates"}, {"genres": ["Family", "Animation", "Fantasy"], "title": "The Little Prince"}]
```



# Querying an Array by the Array Length

- ▶ 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

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({}, {genres_num: {$size: '$genres'}})
[
  { _id: ObjectId("573a1390f29313caabcd4135"), genres_num: 1 },
  { _id: ObjectId("573a1390f29313caabcd42e8"), genres_num: 2 },
  { _id: ObjectId("573a1390f29313caabcd4323"), genres_num: 3 },
  { _id: ObjectId("573a1390f29313caabcd446f"), genres_num: 2 },
  { _id: ObjectId("573a1390f29313caabcd4803"), genres_num: 3 },
  { _id: ObjectId("573a1390f29313caabcd4eaf"), genres_num: 2 },
  { _id: ObjectId("573a1390f29313caabcd50e5"), genres_num: 3 },
  { _id: ObjectId("573a1390f29313caabcd516c"), genres_num: 2 },
  { _id: ObjectId("573a1390f29313caabcd6223"), genres_num: 3 },
  { _id: ObjectId("573a1390f29313caabcd60e4"), genres_num: 3 },
  { _id: ObjectId("573a1390f29313caabcd5967"), genres_num: 3 },
  { _id: ObjectId("573a1390f29313caabcd548c"), genres_num: 3 },
  { _id: ObjectId("573a1390f29313caabcd6377"), genres_num: 3 },
  { _id: ObjectId("573a1390f29313caabcd63d6"), genres_num: 1 },
  { _id: ObjectId("573a1391f29313caabcd68d0"), genres_num: 3 },
  { _id: ObjectId("573a1390f29313caabcd680a"), genres_num: 2 },
  { _id: ObjectId("573a1391f29313caabcd6d40"), genres_num: 2 },
  { _id: ObjectId("573a1391f29313caabcd6d90"), genres_num: 2 },
  { _id: ObjectId("573a1391f29313caabcd6e2a"), genres_num: 2 },
]
```



# Querying Embedded Documents

- ▶ 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

- ▶ To match the entire embedded document use

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

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



A screenshot of a terminal window showing MongoDB shell results. The command executed is:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({awards: {wins: 3, nominations: 0, text: '3 wins.'}}, {title: 1, awards: 1})
```

The output shows five documents, each representing a movie with its title and the awards document embedded:

```
[{"_id": ObjectId("573a1391f29313caabcd7e23"), "title": "The Big Parade", "awards": { "wins": 3, "nominations": 0, "text": "3 wins." }}, {"_id": ObjectId("573a1391f29313caabcd883d"), "title": "Napoleon", "awards": { "wins": 3, "nominations": 0, "text": "3 wins." }}, {"_id": ObjectId("573a1391f29313caabcd9600"), "title": "City Lights", "awards": { "wins": 3, "nominations": 0, "text": "3 wins." }}, {"_id": ObjectId("573a1392f29313caabcd97a7"), "title": "M\u00e4dchen in Uniform", "awards": { "wins": 3, "nominations": 0, "text": "3 wins." }}, {"_id": ObjectId("573a1392f29313caabbcda473"), "title": "Man of Aran", "awards": { "wins": 3, "nominations": 0, "text": "3 wins." }}}]
```



## Class Exercise

- ▶ 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

- ▶ 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

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

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

# Cursors



Northeastern  
University

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.find({}, {title: 1, 'awards.wins': 1, _id: 0}).sort({'awards.wins': -1}).limit(5)
[
  { title: '12 Years a Slave', awards: { wins: 267 } },
  { title: 'Gravity', awards: { wins: 231 } },
  { title: 'Gravity', awards: { wins: 231 } },
  {
    title: 'Birdman: Or (The Unexpected Virtue of Ignorance)',
    awards: { wins: 210 }
  },
  { title: 'Boyhood', awards: { wins: 185 } }
]
moviesdb>
```

# 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



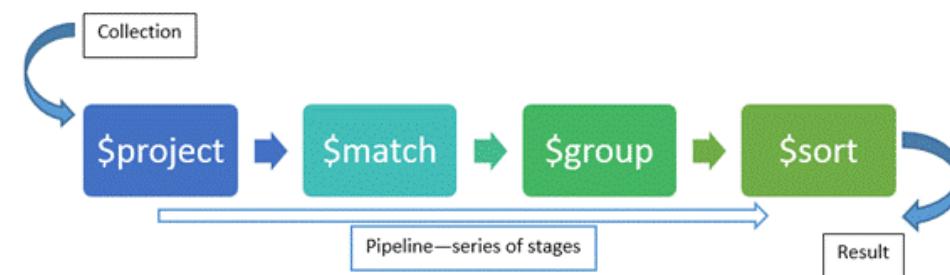
A screenshot of a terminal window titled "mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTi...". The window displays the following MongoDB command and its results:

```
moviesdb> db.movies.distinct('genres.0')
[
  'Action',      'Adventure',
  'Animation',   'Biography',
  'Comedy',       'Crime',
  'Documentary', 'Drama',
  'Family',       'Fantasy',
  'Film-Noir',    'History',
  'Horror',        'Music',
  'Musical',      'Mystery',
  'Romance',      'Sci-Fi',
  'Short',         'Sport',
  'Thriller',     'War',
  'Western'
]
```



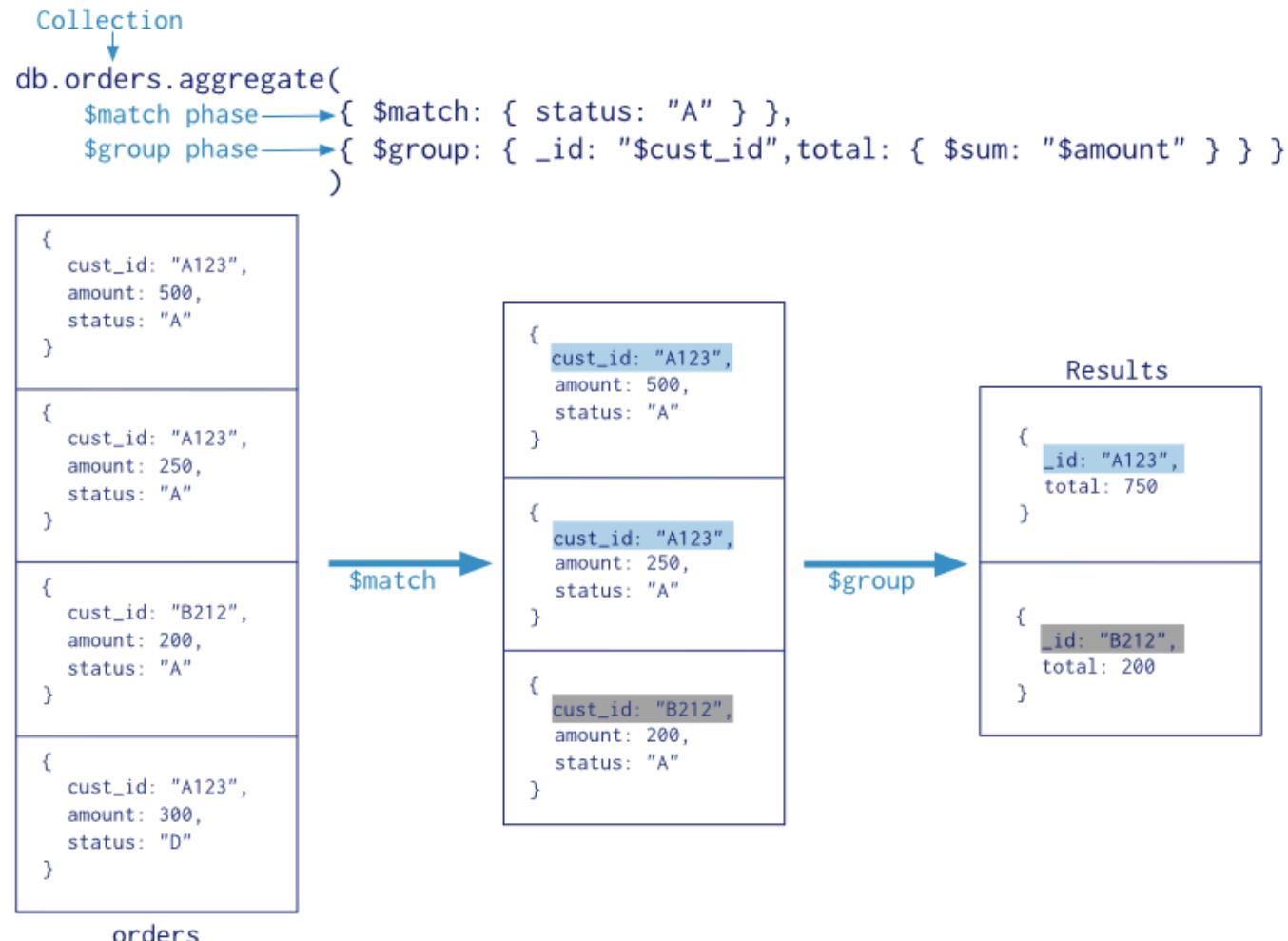
# Aggregation Pipelines

- ▶ 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 <code>_id</code> 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

- ▶ \$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':

A screenshot of a terminal window titled "mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000". The window displays the following MongoDB aggregate command and its result:

```
moviesdb> db.movies.aggregate([
... {$match: {genres: 'Sci-Fi'}}
... ])
[ {
  _id: ObjectId("573a1391f29313caabcd8414"),
  fullplot: `Sometime in the future, the city of Metropolis is home to a Utopian society where its wealthy residents live a carefree life. One of those is Freder Fr edersen. One day, he spots a beautiful woman with a group of children, she and the children who quickly disappear. Trying to follow her, he, oblivious to such, is hor
```

The terminal window has a dark background with light-colored text. The scroll bar on the right side of the terminal window is visible.

# \$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

- ▶ 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

- ▶ 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

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$project: {runtimeInHours: {$divide: ['$runtime', 60]}, title: 1, _id: 0}}
... ])
[
  { title: 'Blacksmith Scene', runtimeInHours: 0.01666666666666666 },
  {
    title: 'The Great Train Robbery',
    runtimeInHours: 0.1833333333333332
  },
  {
    title: 'The Land Beyond the Sunset',
    runtimeInHours: 0.2333333333333334
  },
  { title: 'A Corner in Wheat', runtimeInHours: 0.2333333333333334 },
  {
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Com
ics',
    runtimeInHours: 0.11666666666666667
  },
moviesdb>
  { title: 'Gertie the Dinosaur', runtimeInHours: 0.2 },
  {
```



# Comparison Operators

- ▶ 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

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$project: {greatMovie: {$gt: ['$imdb.rating', 9]}, 'imdb.rating': 1, title: 1, _id: 0}}
... ])
[
  {
    title: 'Blacksmith Scene',
    imdb: { rating: 6.2 },
    greatMovie: false
  },
  {
    title: 'The Great Train Robbery',
    imdb: { rating: 7.4 },
    greatMovie: false
  },
  {
    title: 'The Land Beyond the Sunset',
    imdb: { rating: 7.1 },
    greatMovie: false
  },
  {
    title: 'A Corner in Wheat',
    imdb: { rating: 6.6 },
```



# Boolean Operators

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

| Operator | Description                                                               |
|----------|---------------------------------------------------------------------------|
| \$and    | Returns true only when <i>all</i> 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 },
    goodMovie: true
  }
]
```



# \$ifNull

- ▶ 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 /<br>\$strLenCP  | Returns the number of UTF-8 encoded bytes / code points in a string                                                                                                                     |
| \$indexOfBytes /<br>\$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 /<br>\$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

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS... - X
moviesdb> db.movies.aggregate([
...   {$project: {shortTitle: {$substrCP: ['$title', 0, 10]}, _id: 0}}
... ])
[
  { shortTitle: 'Blacksmith' },
  { shortTitle: 'The Great' },
  { shortTitle: 'The Land B' },
  { shortTitle: 'A Corner i' },
  { shortTitle: 'Winsor McC' },
  { shortTitle: 'Traffic in' },
  { shortTitle: 'Gertie the' },
  { shortTitle: 'In the Lan' },
  { shortTitle: 'The Poor L' },
  { shortTitle: 'The Immigr' },
  { shortTitle: 'Les vampir' },
  { shortTitle: 'The Birth' },
  { shortTitle: 'Wild and W' },
  { shortTitle: 'The Blue B' },
  { shortTitle: 'From Hand' },
  { shortTitle: 'Broken Blo' },
  { shortTitle: 'High and D' },
  { shortTitle: 'The Last o' },
]
```

# 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

- ▶ 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

- ▶ 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

- ▶ \$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'
  },
  {
    firstDirectorFirstName: 'John',
    firstDirectorLastName: 'Stahl'
  }
]
```



# \$group

- ▶ 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

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS...
moviesdb> db.movies.aggregate([
...   {$group: {_id: '$year', moviesNumber: {$count: {}}}}
... ])
[
  { _id: 1911, moviesNumber: 2 },
  { _id: 1921, moviesNumber: 6 },
  { _id: 1957, moviesNumber: 89 },
  { _id: 1968, moviesNumber: 112 },
  { _id: 2008, moviesNumber: 969 },
  { _id: 2015, moviesNumber: 484 },
  { _id: 1989, moviesNumber: 244 },
  { _id: '1987è', moviesNumber: 1 },
  { _id: 1978, moviesNumber: 133 },
  { _id: 2010, moviesNumber: 970 },
  { _id: 1960, moviesNumber: 93 },
  { _id: 1953, moviesNumber: 82 },
  { _id: 1999, moviesNumber: 542 },
  { _id: 1923, moviesNumber: 6 },
  { _id: 1976, moviesNumber: 127 },
  { _id: 1987, moviesNumber: 239 },
  { _id: 1925, moviesNumber: 13 },
  { _id: 1977, moviesNumber: 134 },
  { _id: 1938, moviesNumber: 44 },
  { _id: 1962, moviesNumber: 98 }
]
Type "it" for more
moviesdb> -
```



# Accumulator Operators

- ▶ 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

- ▶ 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: 7.4166666666666667 },
  { _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: 7.5666666666666666 },
  { _id: 1911, averageRating: 7.3 },
  { _id: 1957, averageRating: 7.216853932584269 },
  { _id: 1936, averageRating: 7.25 }
]
Type "it" for more
moviesdb>
```



# Group by Having

- ▶ 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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
moviesdb> db.movies.aggregate([
...   {$group: {_id: '$year', moviesNumber: {$count: {}}}},
...   {$match: {moviesNumber: {$lt: 20}}}
... ])
[{"_id": "1911", "moviesNumber": 2}, {"_id": "1921", "moviesNumber": 6}, {"_id": "1987", "moviesNumber": 1}, {"_id": "1923", "moviesNumber": 6}, {"_id": "1925", "moviesNumber": 13}, {"_id": "1914", "moviesNumber": 3}, {"_id": "1997", "moviesNumber": 2}, {"_id": "1917", "moviesNumber": 3}, {"_id": "1893", "moviesNumber": 1}, {"_id": "1918", "moviesNumber": 1}, {"_id": "1986", "moviesNumber": 1}, {"_id": "2009", "moviesNumber": 2}, {"_id": "1915", "moviesNumber": 5}, {"_id": "1999", "moviesNumber": 1}, {"_id": "2016", "moviesNumber": 1}, {"_id": "1916", "moviesNumber": 4}, {"_id": "1920", "moviesNumber": 6}, {"_id": "2007", "moviesNumber": 3}, {"_id": "1891", "moviesNumber": 1}, {"_id": "1922", "moviesNumber": 7}]
Type "it" for more
```



# Retrieving Distinct Values

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTi...
moviesdb> db.movies.aggregate([
...   {$group: {_id: '$rated'}}
... ])
[
  { _id: 'X' },      { _id: 'NC-17' },
  { _id: 'UNRATED' }, { _id: 'TV-PG' },
  { _id: 'AO' },     { _id: 'NOT RATED' },
  { _id: 'PG-13' },   { _id: 'Approved' },
  { _id: 'R' },       { _id: 'APPROVED' },
  { _id: 'TV-MA' },   { _id: null },
  { _id: 'TV-14' },   { _id: 'PASSED' },
  { _id: 'TV-G' },    { _id: 'PG' },
  { _id: 'GP' },      { _id: 'M' },
  { _id: 'OPEN' },    { _id: 'Not Rated' }
]
Type "it" for more
moviesdb>
```



# Group by null

- ▶ 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



Northeastern  
University

- ▶ 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>}
```

{ key1: "value1",  
key2: "value2",  
key3: [ "elem1",  
"elem2",  
"elem3" ] }      ⇒ \$unwind  
                        ↓

{ key1: "value1",    { key1: "value1",    { key1: "value1",  
key2: "value2",    key2: "value2",    key2: "value2",  
key3: "elem1" }    key3: "elem2" }    key3: "elem3" }



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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS... - X
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

- ▶ 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... - X
moviesdb> db.movies.aggregate([
...   {$project: {cast: 1, _id: 0}},
...   {$unwind: '$cast'},
...   {$group: {_id: '$cast', numOfMovies: {$count: {}}}}
... ])
[
  { _id: 'Aurélien 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 Beaufort', numOfMovies: 1 },
```



# Class Exercise

- ▶ 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

- ▶ 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

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTi...
test> use productsdb
switched to db productsdb
productsdb> db.products.insertOne(
...   {
...     name: 'iPhone 14',
...     price: 799,
...     size: {width: 2.82, height: 5.78, uom: 'inch'},
...     tags: ['smartphone', 'mobile']
...   }
... )
{
  acknowledged: true,
  insertedId: ObjectId("6389f8cadbd02aa23513ee7d")
}
productsdb> _
```



# Updating Documents

- ▶ 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(  
  { age: { $lt: 18 } },  
  { $set: { status: "reject" } } )
```

collection ←  
update filter ←  
update action ←



# \$set Operator

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

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

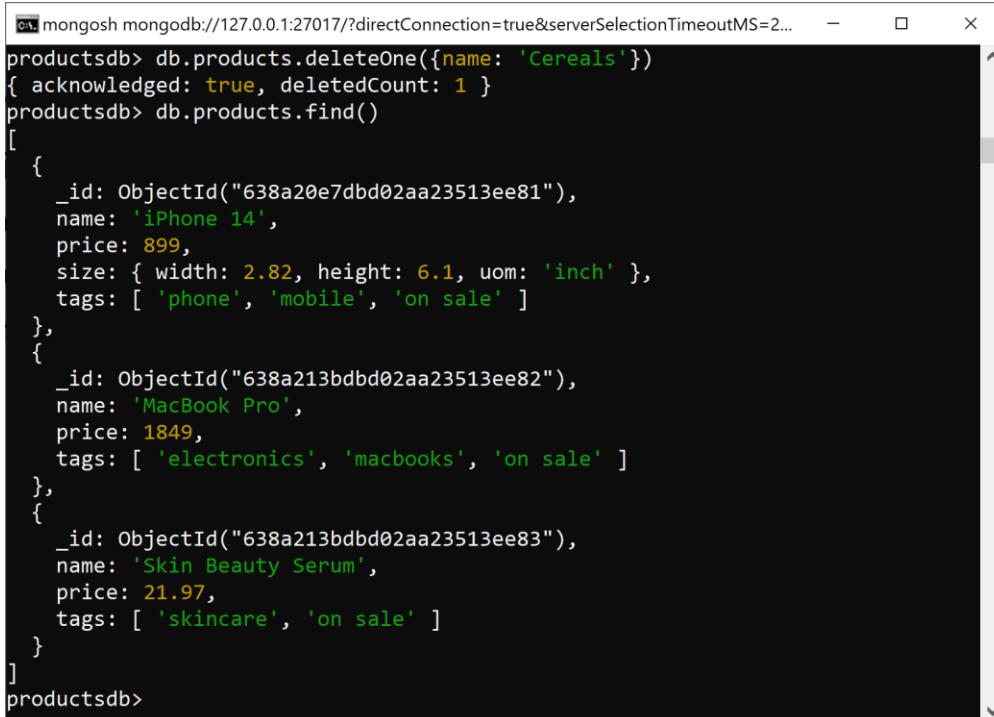
- ▶ If the field doesn't exist, it will be added with the specified value
- ▶ For example, let's change the price of iPhone 14 to \$899:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=5000
productsdb> db.products.updateOne(
...   {name: 'iPhone 14'},
...   {$set: {price: 899}}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
productsdb> db.products.findOne({name: 'iPhone 14'})
{
  _id: ObjectId("638a20e7dbd02aa23513ee81"),
  name: 'iPhone 14',
  price: 899,
  size: { width: 2.82, height: 5.78, uom: 'inch' },
  tags: [ 'smartphone', 'mobile' ]
}
productsdb>
```



# Deleting Documents

- ▶ 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

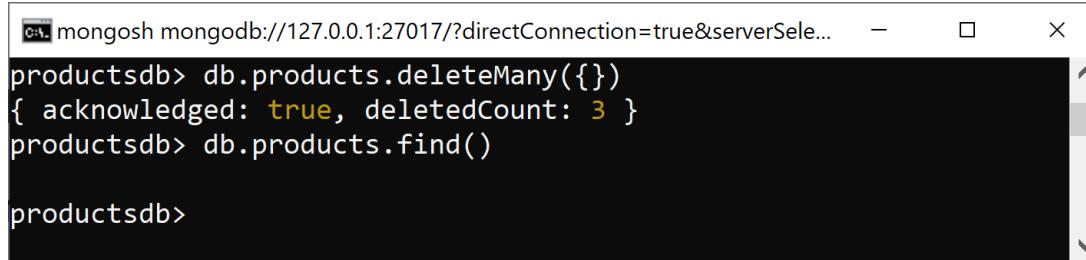


```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2...
productsdb> db.products.deleteOne({name: 'Cereals'})
{ acknowledged: true, deletedCount: 1 }
productsdb> db.products.find()
[
  {
    _id: ObjectId("638a20e7dbd02aa23513ee81"),
    name: 'iPhone 14',
    price: 899,
    size: { width: 2.82, height: 6.1, uom: 'inch' },
    tags: [ 'phone', 'mobile', 'on sale' ]
  },
  {
    _id: ObjectId("638a213bdbd02aa23513ee82"),
    name: 'MacBook Pro',
    price: 1849,
    tags: [ 'electronics', 'macbooks', 'on sale' ]
  },
  {
    _id: ObjectId("638a213bdbd02aa23513ee83"),
    name: 'Skin Beauty Serum',
    price: 21.97,
    tags: [ 'skincare', 'on sale' ]
  }
]
productsdb>
```



# Deleting All Documents

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



A screenshot of a terminal window showing a MongoDB shell session. The session starts with the command `db.products.deleteMany({})`, which returns an object indicating success with `acknowledged: true` and `deletedCount: 3`. Following this, the command `db.products.find()` is run, but its output is completely obscured by a large black rectangular redaction box. Finally, the prompt `productsdb>` is shown at the bottom.

- ▶ 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

- ▶ 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.getSiblingDB("movies"); // the equivalent of use movies

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

- ▶ 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 successfully
```



# Indexes

- ▶ 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

- ▶ 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

- 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** 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

- ▶ 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 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

- ▶ 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

- ▶ 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

- ▶ 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

- ▶ 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}))
```

```
[]
```