DSC 333 - Cloud Services for Data Science
Spring 2025

•••

Intro to NoSQL databases

Imad Antonios

Key features of relational databases (SQL)

- Data is stored in structured tables with rows and columns that follows a predefined schema and a related using keys
- Query language is fairly standardized (SQL)
- Emphasis on transaction control that ensures data integrity (e.g. atomicity, isolation).
- Designed for vertical scalability (add more powerful server to support increased capacity)

Key features of NoSQL databases

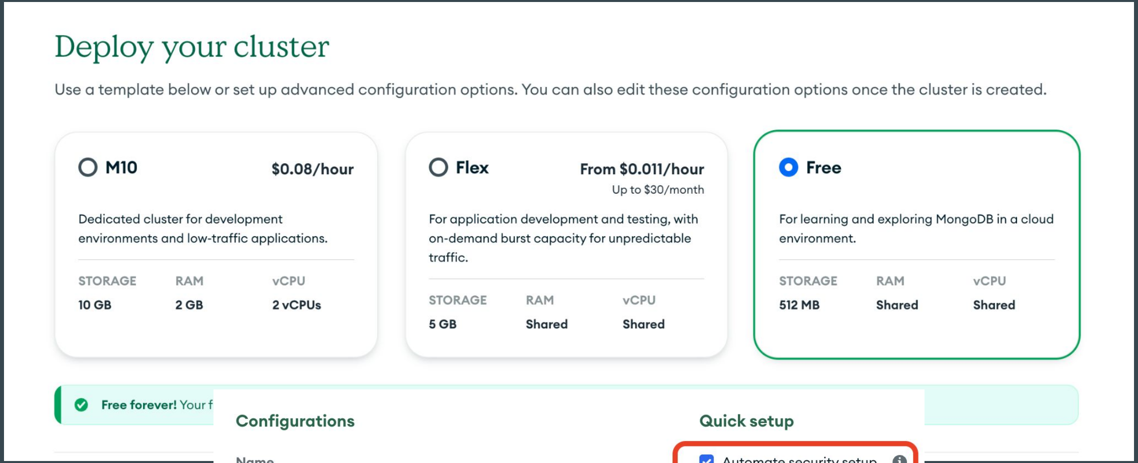
- Various data models supported (e.g. document-oriented (JSON), graph databases).
- Schema-less or flexible schema
- Prioritizes data availability and performance over integrity
- Designed for **horizontal scalability** (add more servers to increase capacity)
- Querying interface is dependent on DB system (sometimes SQL like).

Common NoSQL databases

- MongoDB: open-source document DB (JSON-like)
- Amazon DocumentDB: fully managed service in AWS
- Azure Cosmos DB: multi-model DB.
- Neo4j: Graph DB
- Elasticsearch: text analytics, REST based
- Google Cloud Firestore

MongoDB Atlas (Cloud-based)

Sign up for free tier: https://www.mongodb.com/pricing



Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

○ M10 — $0.08/hour

Dedicated cluster for development environments and low-traffic applications.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

○ Flex — From $0.011/hour, Up to $30/month

For application development and testing, with on-demand burst capacity for unpredictable traffic.

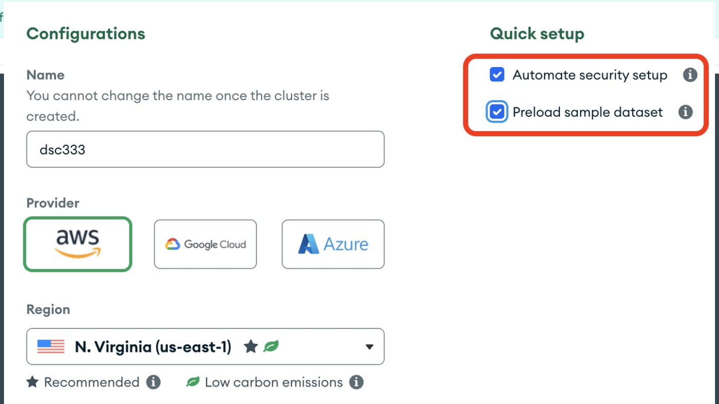
STORAGE	RAM	vCPU
5 GB	Shared	Shared

● Free

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

Free forever! Your f



Configurations

Name

You cannot change the name once the cluster is created.

dsc333

Provider

Region

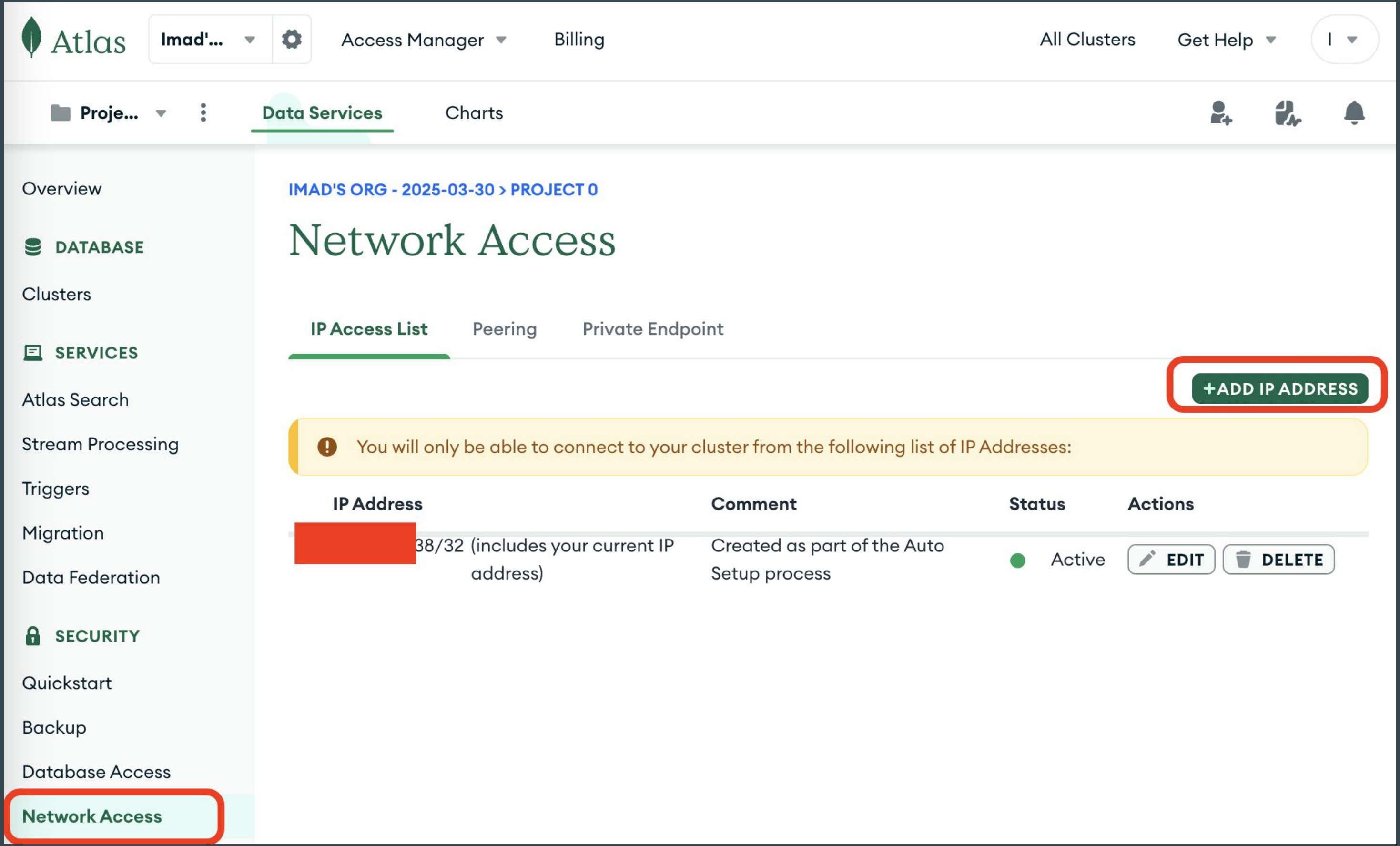
N. Virginia (us-east-1)

Recommended · Low carbon emissions

Quick setup

- [✓] Automate security setup
- [✓] Preload sample dataset

Add additional IP addresses to access list



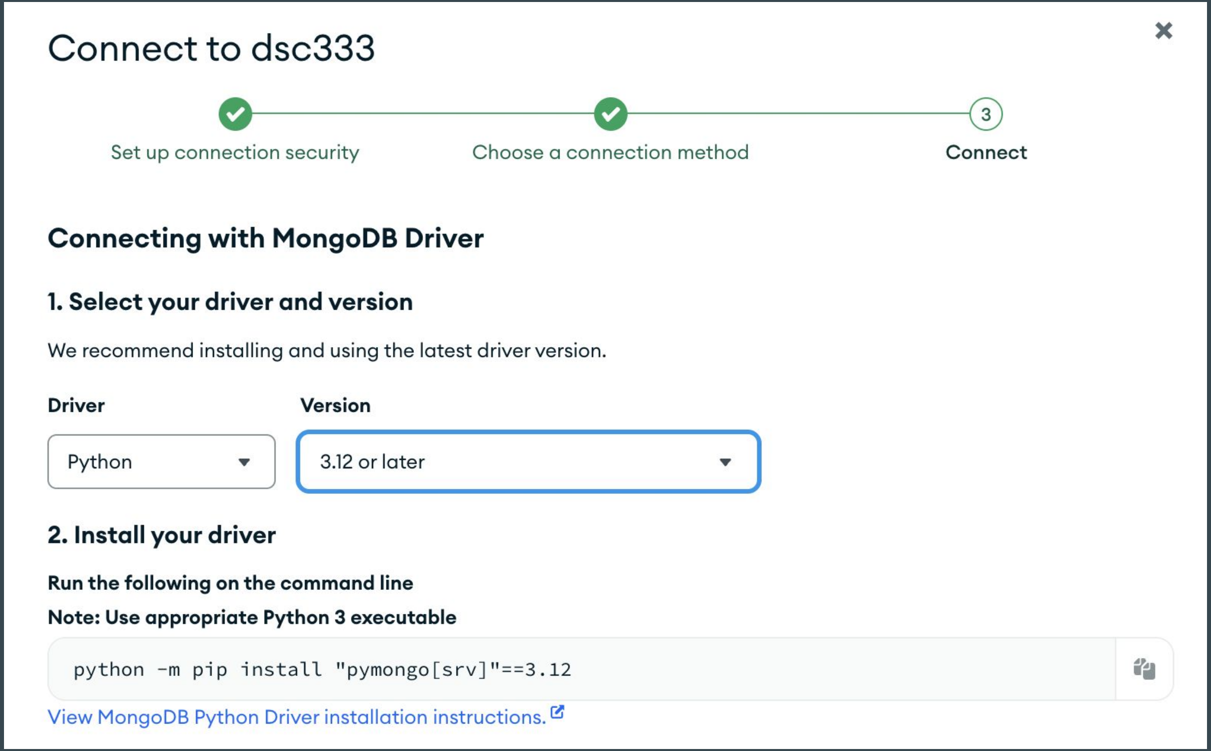
IMAD'S ORG - 2025-03-30 > PROJECT 0

Network Access

You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
[redacted]38/32 (includes your current IP address)	Created as part of the Auto Setup process	● Active	EDIT DELETE

MongoDB Python driver



Connect to dsc333

Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver: Python

Version: 3.12 or later

2. Install your driver

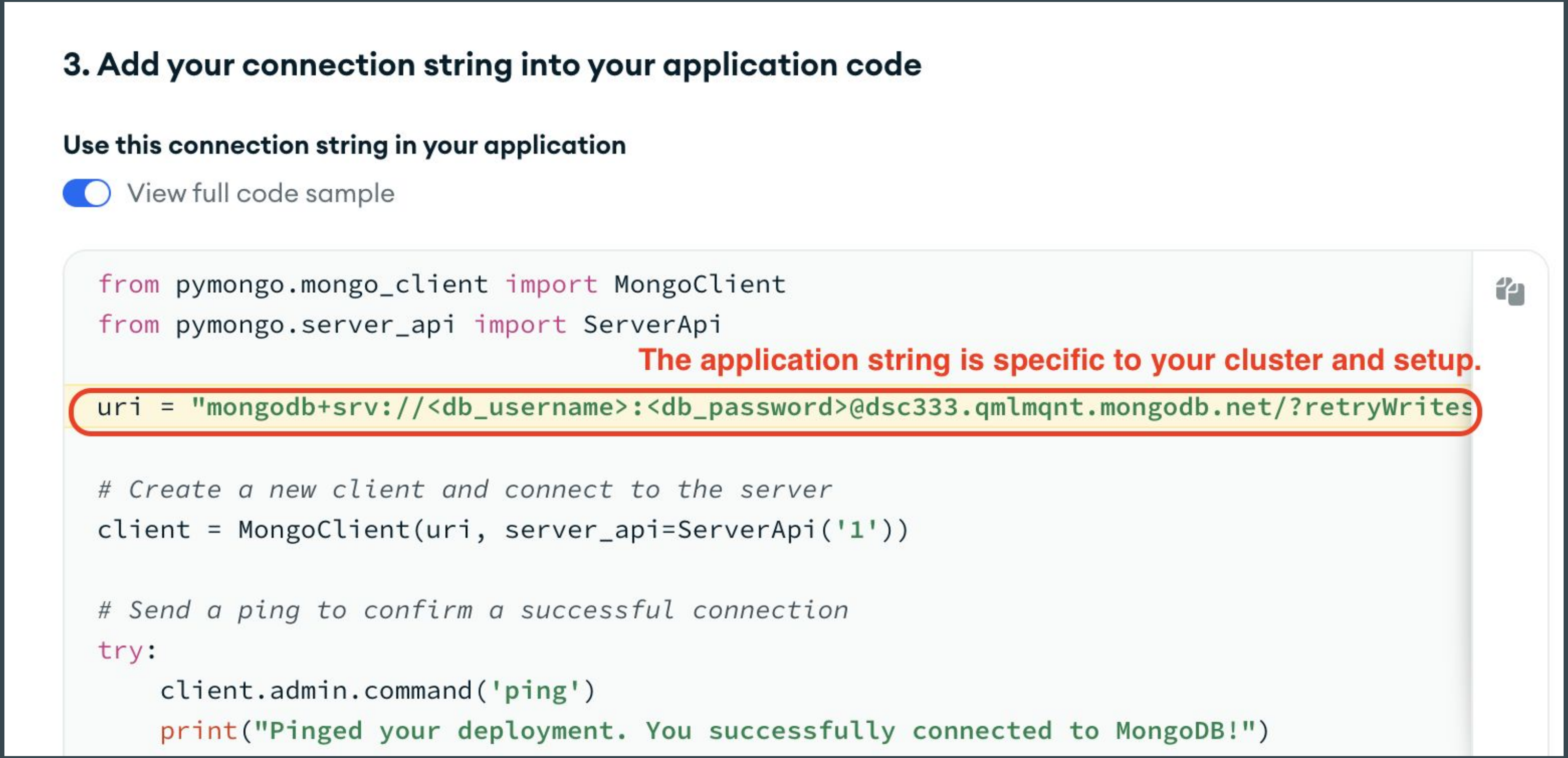
Run the following on the command line

Note: Use appropriate Python 3 executable

```
python -m pip install "pymongo[srv]"==3.12
```

View MongoDB Python Driver installation instructions.

Connection string



3. Add your connection string into your application code

Use this connection string in your application

View full code sample

The application string is specific to your cluster and setup.

```
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi

uri = "mongodb+srv://<db_username>:<db_password>@dsc333.qmlmqnt.mongodb.net/?retryWrites

# Create a new client and connect to the server
client = MongoClient(uri, server_api=ServerApi('1'))

# Send a ping to confirm a successful connection
try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
```

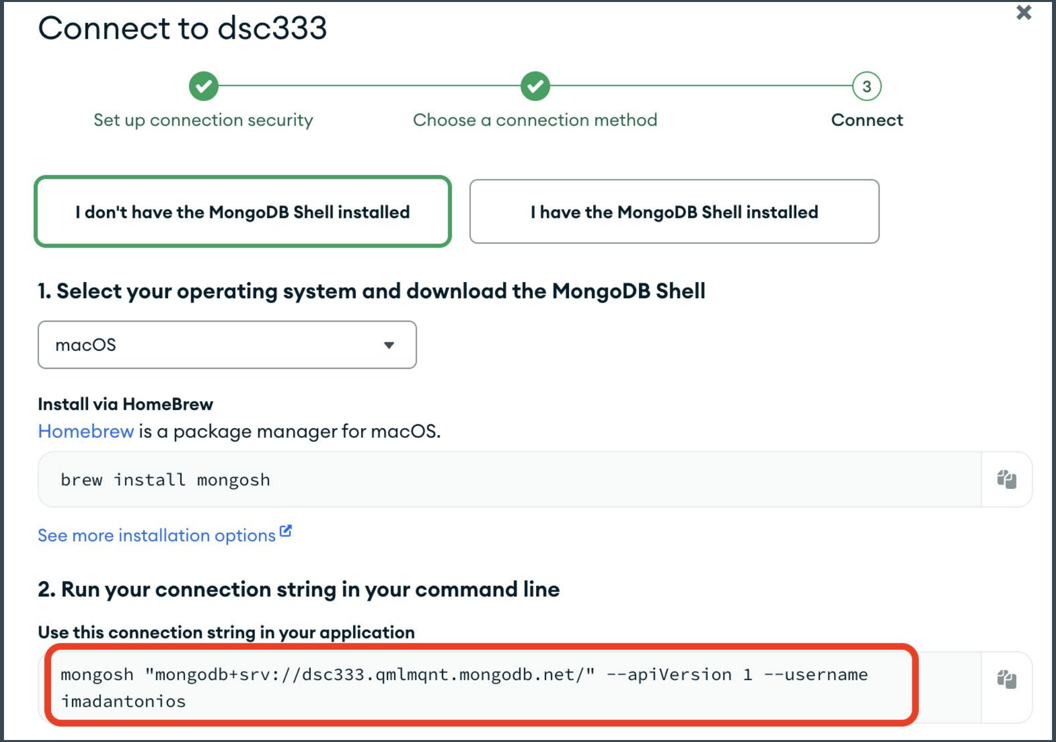
Test connection to Atlas

Repo: https://github.com/dsc333/mongo

- Clone repo into VSCode and create virtual environment using requirements.txt
- Create .env file and initialize MONGO_USER and MONGO_PASS (your username and password on Atlas MongoDB)
- **Modify the connection string in the mongo-conn-test.py according to your connection string (check the domain)**

Interacting with Mongodb using mongosh

Mongosh is the command-line interface for MongoDB.
Install mongosh and then connect using your connection string.


Connect to dsc333

Set up connection security
Choose a connection method
3
Connect
I don't have the MongoDB Shell installed
I have the MongoDB Shell installed

1. Select your operating system and download the MongoDB Shell

macOS

Install via HomeBrew
Homebrew is a package manager for macOS.

```
brew install mongosh
```

See more installation options

2. Run your connection string in your command line

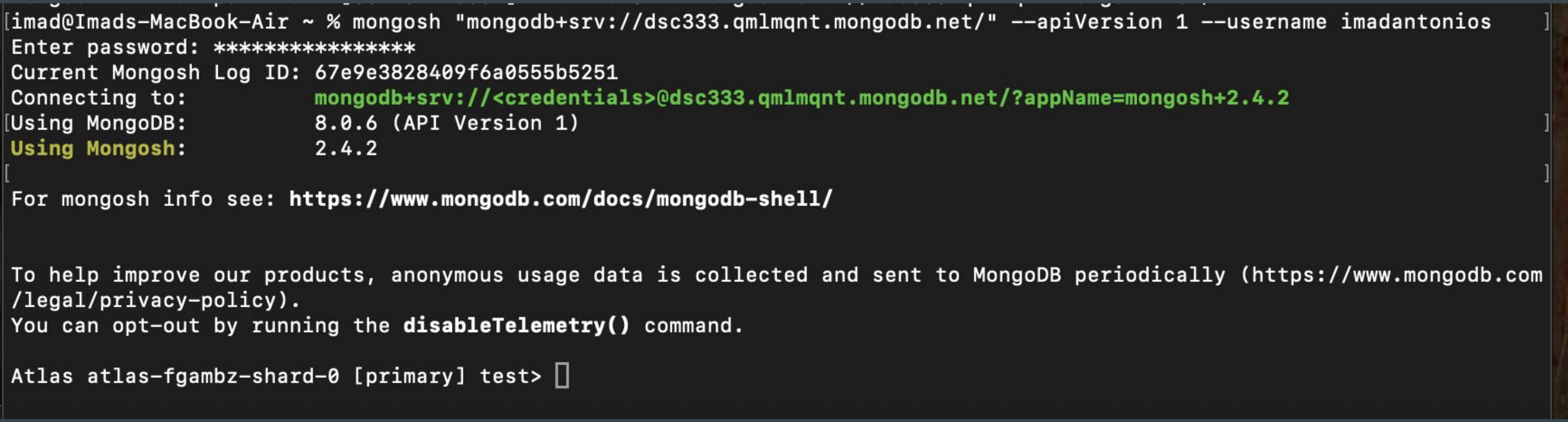
Use this connection string in your application

```
mongosh "mongodb+srv://dsc333.qmlmqnt.mongodb.net/" --apiVersion 1 --username
imadantonios
```

Connect to Atlas using mongosh

Connection string (replace the variables in < > with your values):

```
mongosh "mongodb+srv://<your mongodb domain>"
--apiVersion 1 --username <your username>
```



```
[imad@Imads-MacBook-Air ~ % mongosh "mongodb+srv://dsc333.qmlmqnt.mongodb.net/" --apiVersion 1 --username imadantonios  ]
 Enter password: ****************
 Current Mongosh Log ID: 67e9e3828409f6a0555b5251
 Connecting to:          mongodb+srv://<credentials>@dsc333.qmlmqnt.mongodb.net/?appName=mongosh+2.4.2
[Using MongoDB:          8.0.6 (API Version 1)                                                                    ]
 Using Mongosh:          2.4.2
[                                                                                                                   ]
 For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/


 To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com
 /legal/privacy-policy).
 You can opt-out by running the disableTelemetry() command.

 Atlas atlas-fgambz-shard-0 [primary] test> 
```

MongoDB data model

Data is stored in BSON (B for Binary), JSON-like documents that naturally map to object-oriented representation (B for Binary).

MongoDB is "*schema-less*" but allows for rule definition for schema enforcement

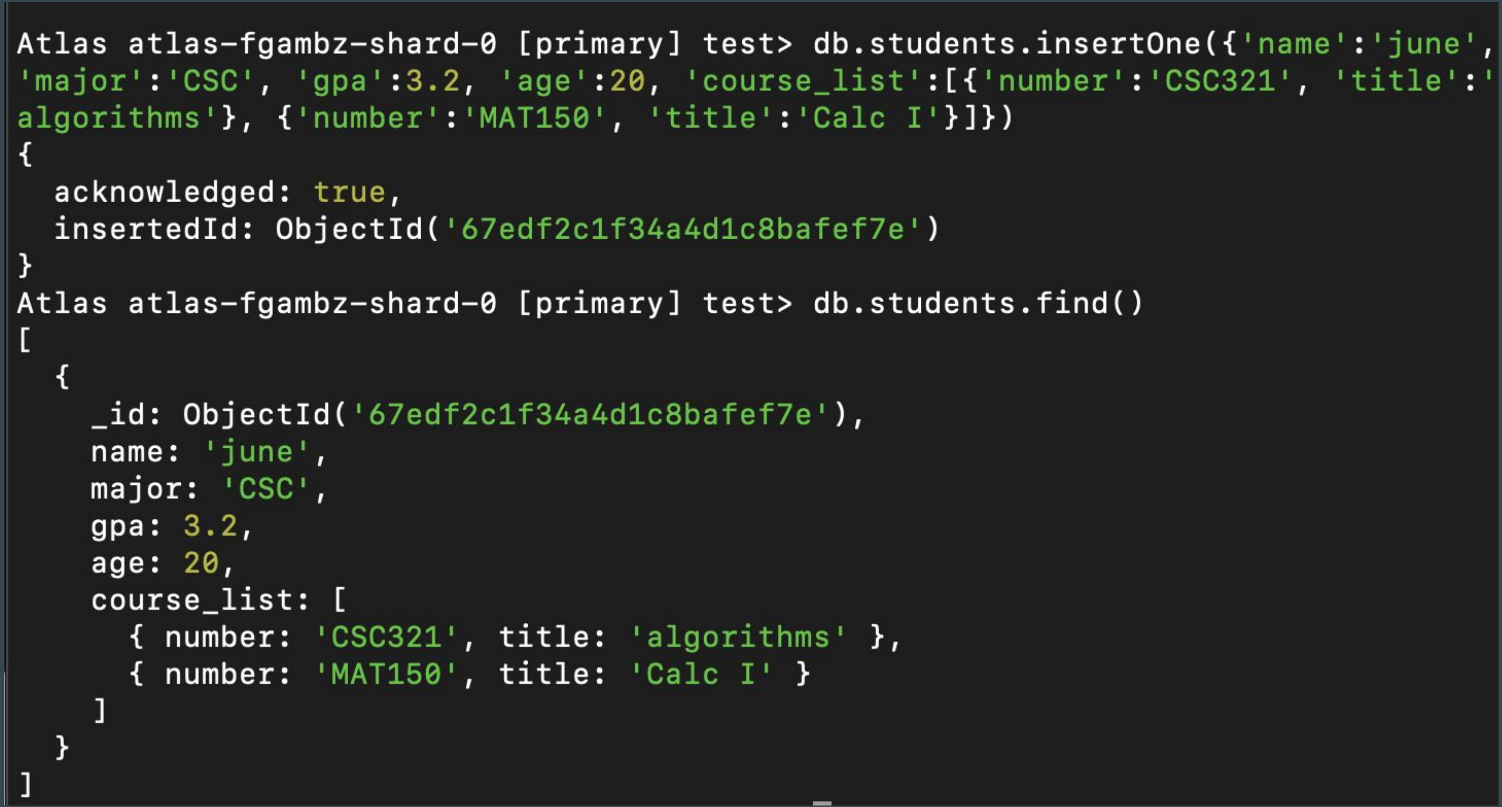
Collections and documents

Collections in MongoDB are what tables are in MySQL

Documents in MongoDB are what rows are in MySQL

CRUD operations in MongoDB

Creating a collection



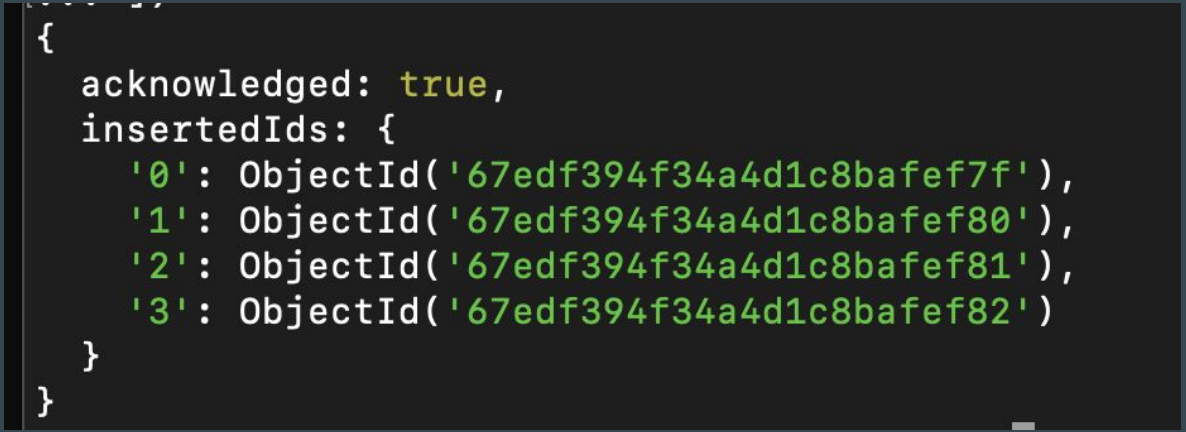
```
Atlas atlas-fgambz-shard-0 [primary] test> db.students.insertOne({'name':'june',
'major':'CSC', 'gpa':3.2, 'age':20, 'course_list':[{'number':'CSC321', 'title':'
algorithms'}, {'number':'MAT150', 'title':'Calc I'}]})
{
  acknowledged: true,
  insertedId: ObjectId('67edf2c1f34a4d1c8bafef7e')
}
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find()
[
  {
    _id: ObjectId('67edf2c1f34a4d1c8bafef7e'),
    name: 'june',
    major: 'CSC',
    gpa: 3.2,
    age: 20,
    course_list: [
      { number: 'CSC321', title: 'algorithms' },
      { number: 'MAT150', title: 'Calc I' }
    ]
  }
]
```

insertMany: insert json objects from file

https://raw.githubusercontent.com/dsc333/json-data/refs/heads/main/data.json

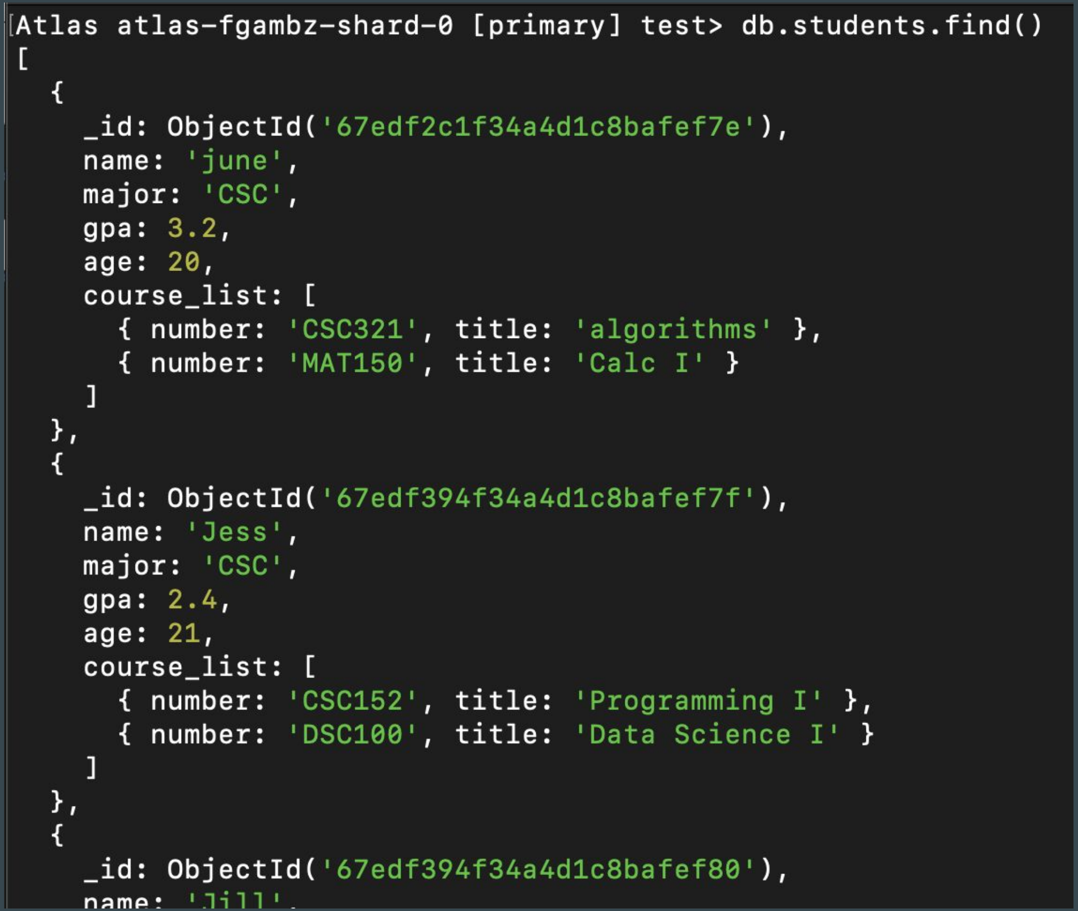
Try this:
db.students.insertMany(<copy and paste contents of file>)

4 objects are inserted into the collection.



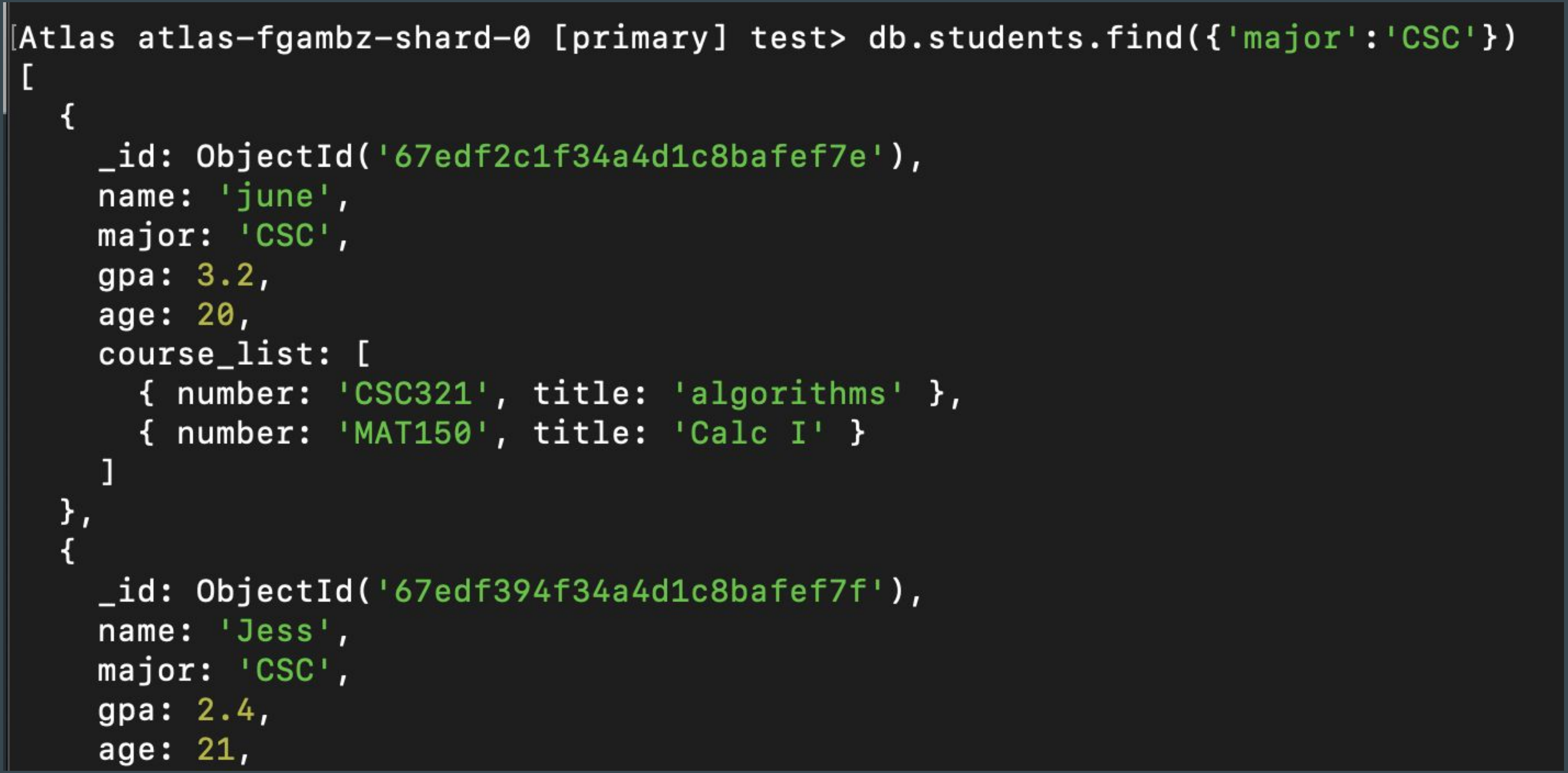
```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67edf394f34a4d1c8bafef7f'),
    '1': ObjectId('67edf394f34a4d1c8bafef80'),
    '2': ObjectId('67edf394f34a4d1c8bafef81'),
    '3': ObjectId('67edf394f34a4d1c8bafef82')
  }
}
```

View the collection



```
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find()
[
  {
    _id: ObjectId('67edf2c1f34a4d1c8bafef7e'),
    name: 'june',
    major: 'CSC',
    gpa: 3.2,
    age: 20,
    course_list: [
      { number: 'CSC321', title: 'algorithms' },
      { number: 'MAT150', title: 'Calc I' }
    ]
  },
  {
    _id: ObjectId('67edf394f34a4d1c8bafef7f'),
    name: 'Jess',
    major: 'CSC',
    gpa: 2.4,
    age: 21,
    course_list: [
      { number: 'CSC152', title: 'Programming I' },
      { number: 'DSC100', title: 'Data Science I' }
    ]
  },
  {
    _id: ObjectId('67edf394f34a4d1c8bafef80'),
    name: 'Jill',
```

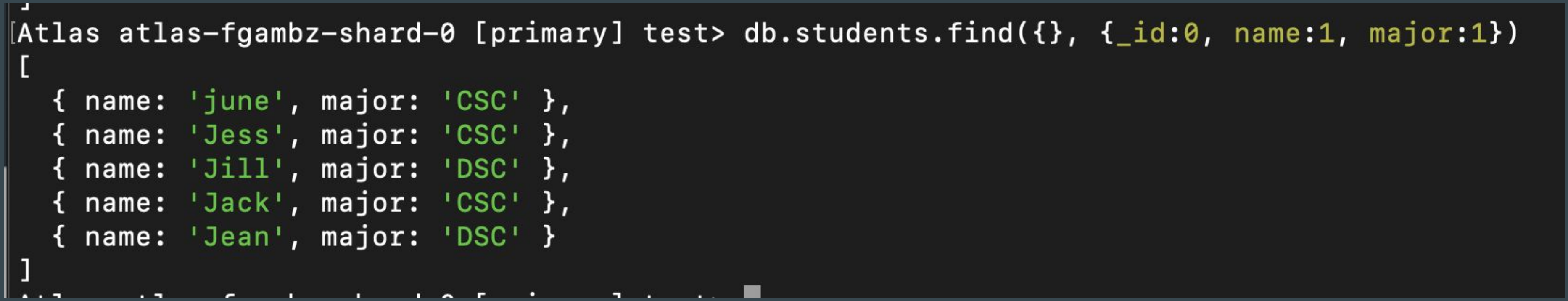
Search by field



```
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({'major':'CSC'})
[
  {
    _id: ObjectId('67edf2c1f34a4d1c8bafef7e'),
    name: 'june',
    major: 'CSC',
    gpa: 3.2,
    age: 20,
    course_list: [
      { number: 'CSC321', title: 'algorithms' },
      { number: 'MAT150', title: 'Calc I' }
    ]
  },
  {
    _id: ObjectId('67edf394f34a4d1c8bafef7f'),
    name: 'Jess',
    major: 'CSC',
    gpa: 2.4,
    age: 21,
```

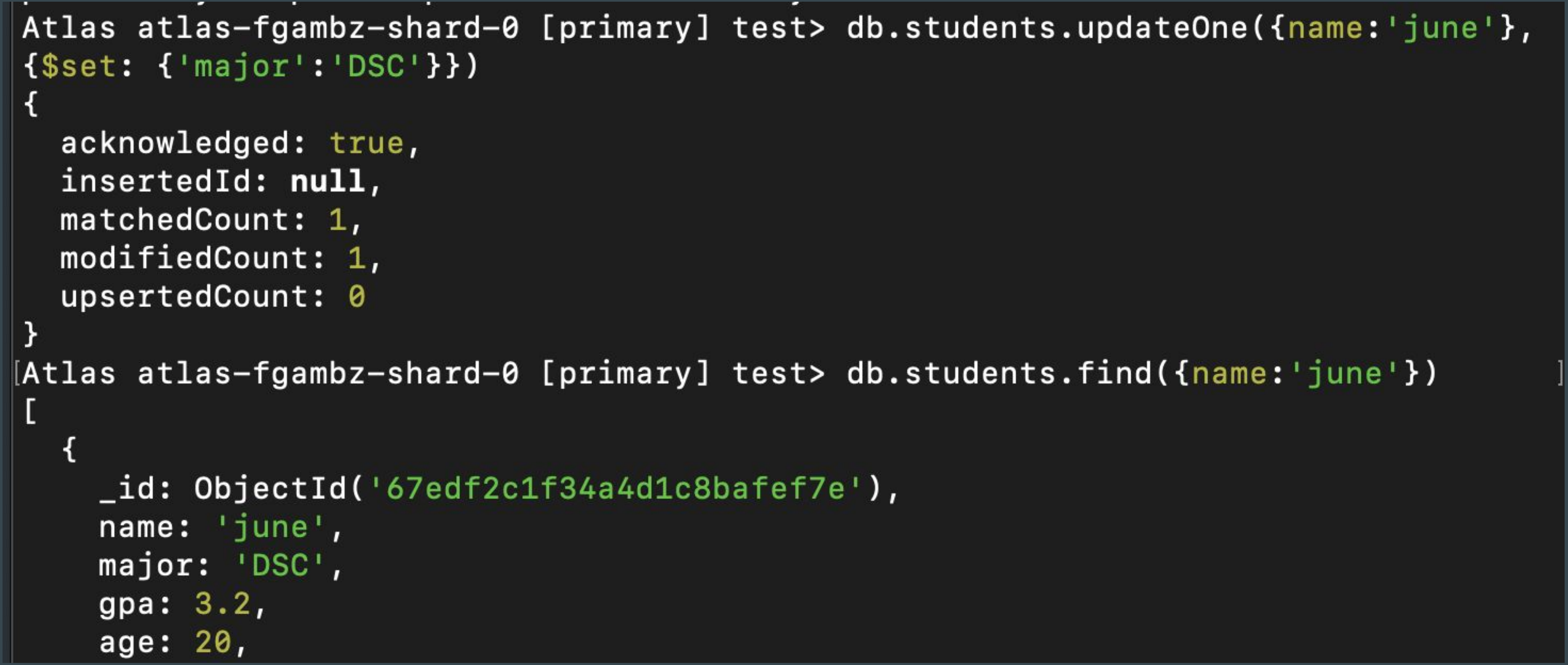
Search projections: select fields to return

Projections are equivalent to SELECT in SQL



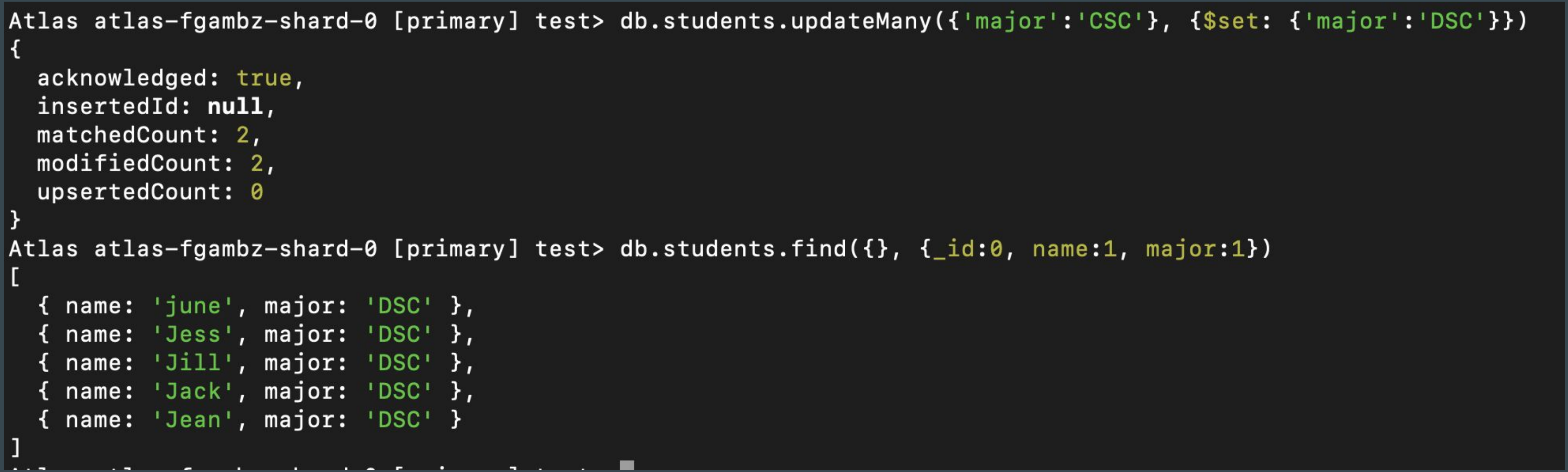
```
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({}, {_id:0, name:1, major:1})
[
  { name: 'june', major: 'CSC' },
  { name: 'Jess', major: 'CSC' },
  { name: 'Jill', major: 'DSC' },
  { name: 'Jack', major: 'CSC' },
  { name: 'Jean', major: 'DSC' }
]
```

Document update with $set operator



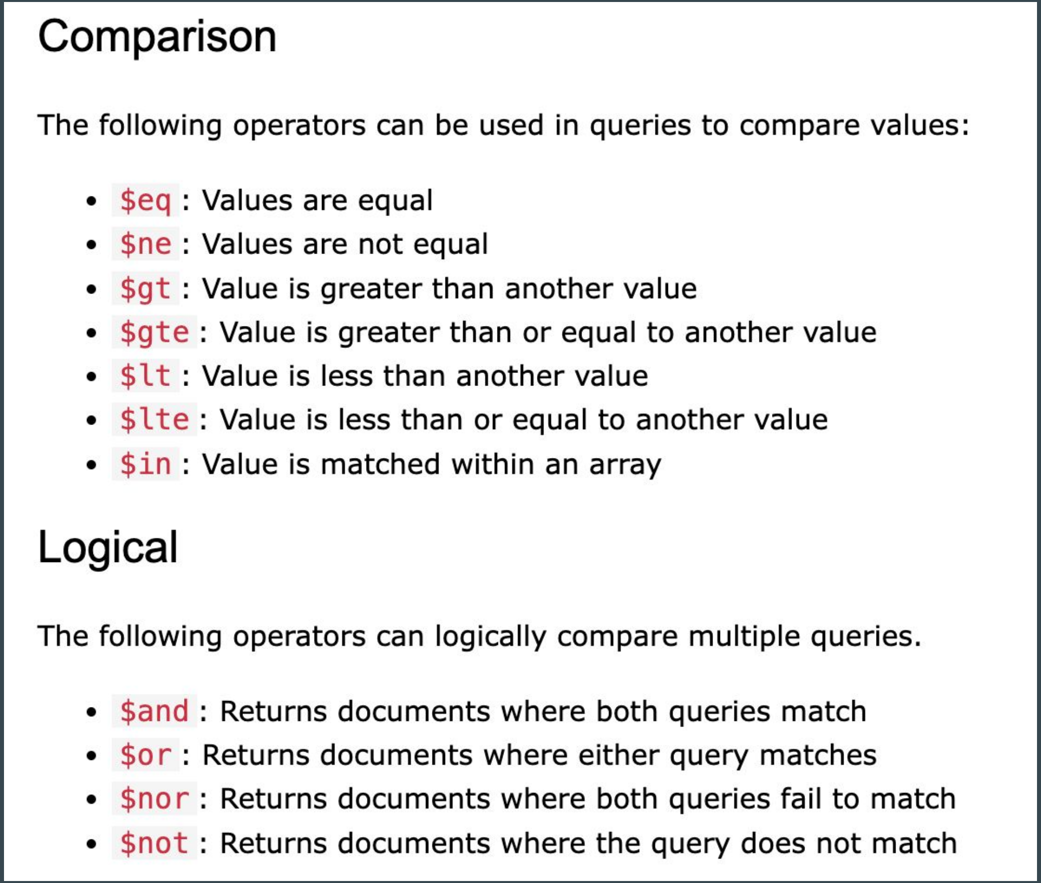
```
Atlas atlas-fgambz-shard-0 [primary] test> db.students.updateOne({name:'june'},
{$set: {'major':'DSC'}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({name:'june'})
[
  {
    _id: ObjectId('67edf2c1f34a4d1c8bafef7e'),
    name: 'june',
    major: 'DSC',
    gpa: 3.2,
    age: 20,
```

updateMany: change all student majors to DSC



```
Atlas atlas-fgambz-shard-0 [primary] test> db.students.updateMany({'major':'CSC'}, {$set: {'major':'DSC'}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({}, {_id:0, name:1, major:1})
[
  { name: 'june', major: 'DSC' },
  { name: 'Jess', major: 'DSC' },
  { name: 'Jill', major: 'DSC' },
  { name: 'Jack', major: 'DSC' },
  { name: 'Jean', major: 'DSC' }
]
```

Operators to filter document fields



Comparison

The following operators can be used in queries to compare values:

- `$eq` : Values are equal
- `$ne` : Values are not equal
- `$gt` : Value is greater than another value
- `$gte` : Value is greater than or equal to another value
- `$lt` : Value is less than another value
- `$lte` : Value is less than or equal to another value
- `$in` : Value is matched within an array

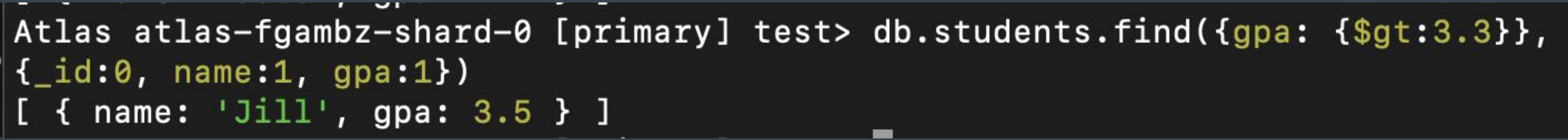
Logical

The following operators can logically compare multiple queries.

- `$and` : Returns documents where both queries match
- `$or` : Returns documents where either query matches
- `$nor` : Returns documents where both queries fail to match
- `$not` : Returns documents where the query does not match

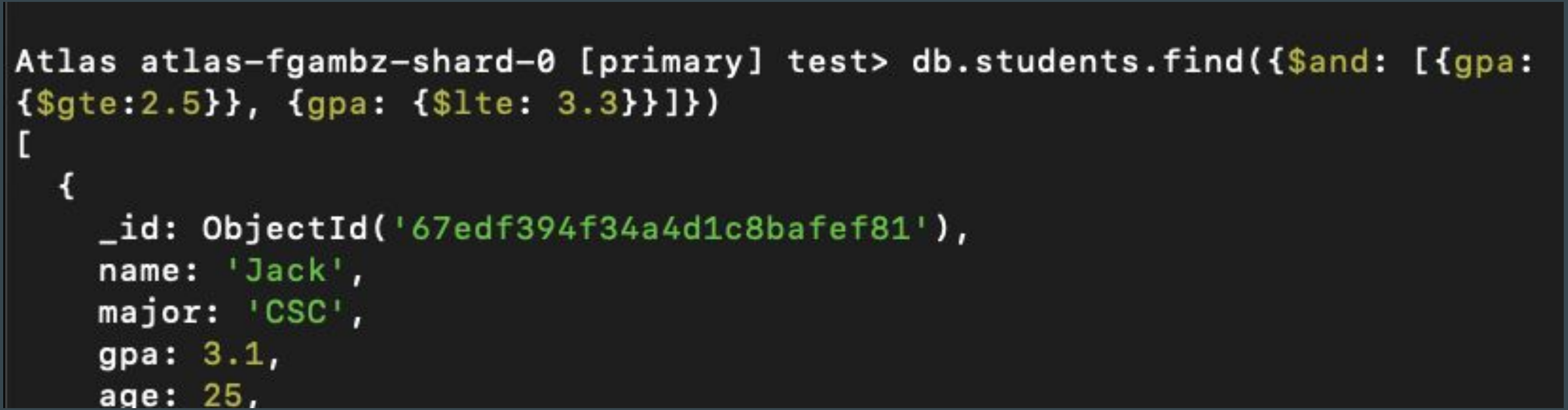
Comparison operator examples

students with gpa > 3.3



```
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({gpa: {$gt:3.3}},
{_id:0, name:1, gpa:1})
[ { name: 'Jill', gpa: 3.5 } ]
```

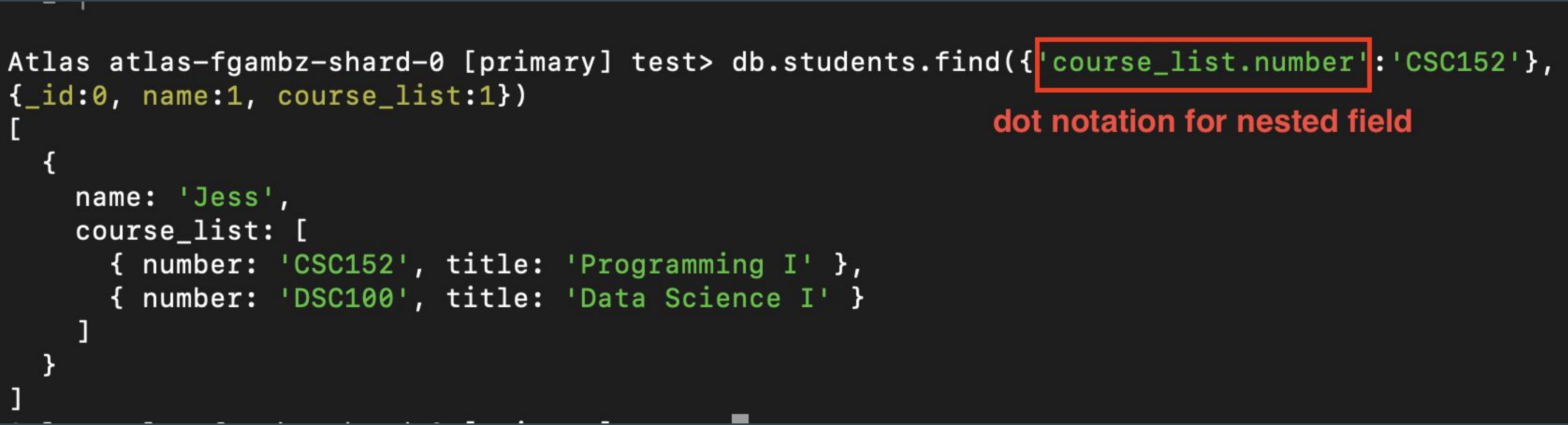
students with 2.5 <= gpa <= 3.3



```
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({$and: [{gpa:
{$gte:2.5}}, {gpa: {$lte: 3.3}}]})
[
  {
    _id: ObjectId('67edf394f34a4d1c8bafef81'),
    name: 'Jack',
    major: 'CSC',
    gpa: 3.1,
    age: 25,
```

Search within arrays of nested documents

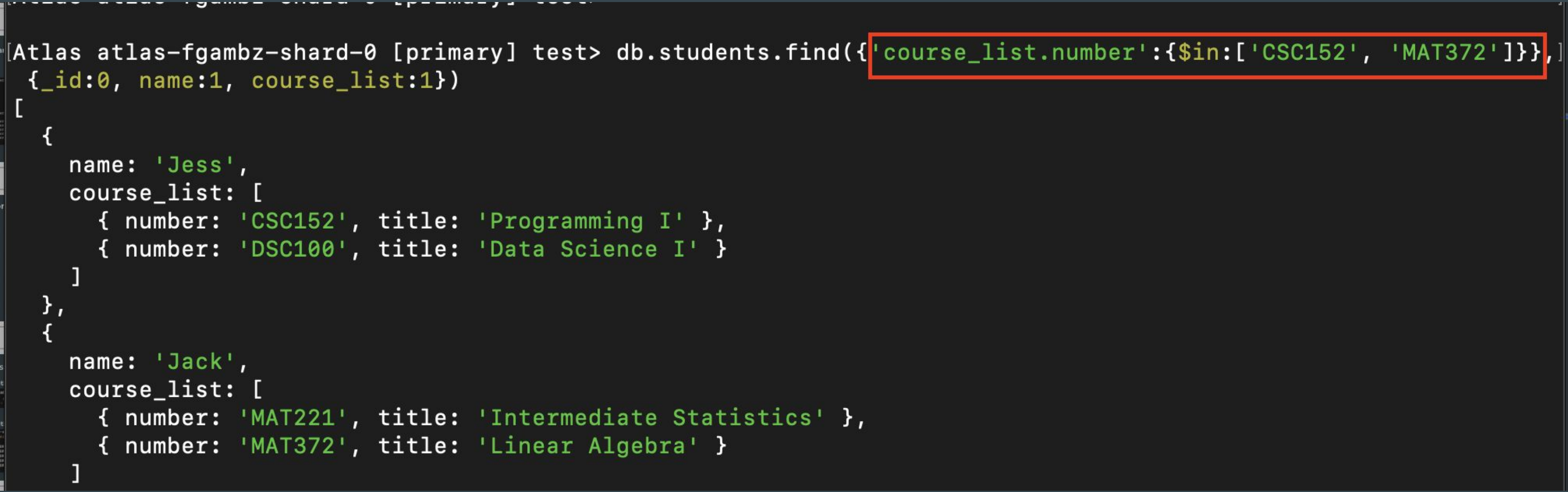
Students with 'CSC152' on their course_list



```
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({'course_list.number':'CSC152'},
{_id:0, name:1, course_list:1})
[
  {
    name: 'Jess',
    course_list: [
      { number: 'CSC152', title: 'Programming I' },
      { number: 'DSC100', title: 'Data Science I' }
    ]
  }
]
```

$in operator

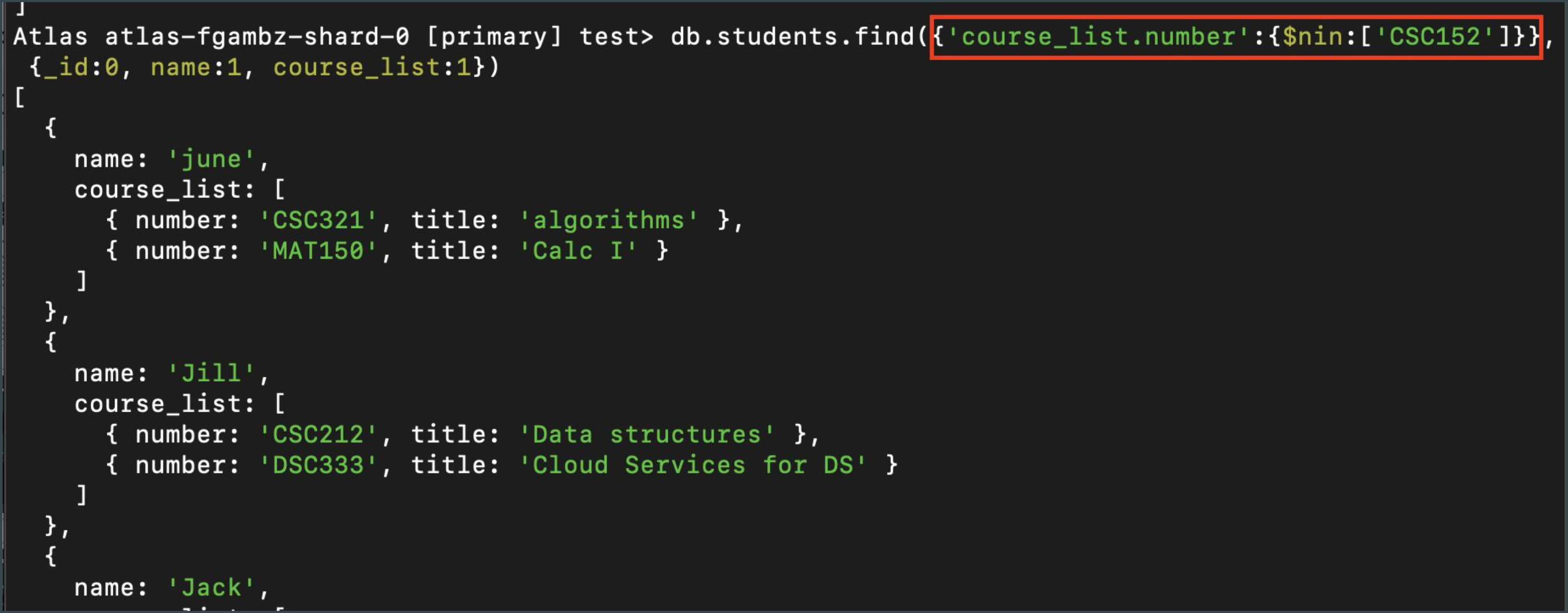
Students taking CSC152 or MAT372



```
[Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({'course_list.number':{$in:['CSC152', 'MAT372']}},
 {_id:0, name:1, course_list:1})
[
  {
    name: 'Jess',
    course_list: [
      { number: 'CSC152', title: 'Programming I' },
      { number: 'DSC100', title: 'Data Science I' }
    ]
  },
  {
    name: 'Jack',
    course_list: [
      { number: 'MAT221', title: 'Intermediate Statistics' },
      { number: 'MAT372', title: 'Linear Algebra' }
    ]
```

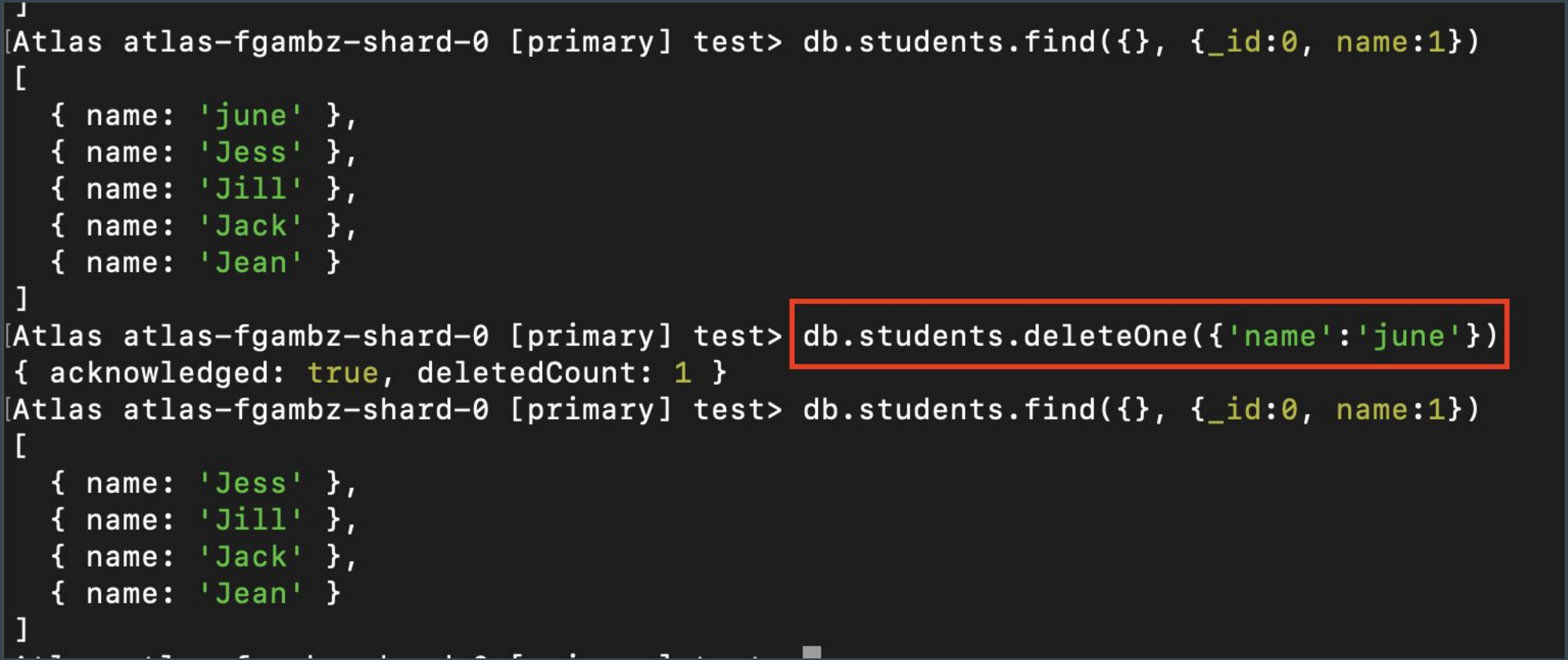
$nin operator

Students NOT taking CSC152



```
]
Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({'course_list.number':{$nin:['CSC152']}},
 {_id:0, name:1, course_list:1})
[
  {
    name: 'june',
    course_list: [
      { number: 'CSC321', title: 'algorithms' },
      { number: 'MAT150', title: 'Calc I' }
    ]
  },
  {
    name: 'Jill',
    course_list: [
      { number: 'CSC212', title: 'Data structures' },
      { number: 'DSC333', title: 'Cloud Services for DS' }
    ]
  },
  {
    name: 'Jack',
```

delete a document



```
]
[Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({}, {_id:0, name:1})
[
  { name: 'june' },
  { name: 'Jess' },
  { name: 'Jill' },
  { name: 'Jack' },
  { name: 'Jean' }
]
[Atlas atlas-fgambz-shard-0 [primary] test> db.students.deleteOne({'name':'june'})
{ acknowledged: true, deletedCount: 1 }
[Atlas atlas-fgambz-shard-0 [primary] test> db.students.find({}, {_id:0, name:1})
[
  { name: 'Jess' },
  { name: 'Jill' },
  { name: 'Jack' },
  { name: 'Jean' }
]
```

Operation atomicity

A single update (updateOne) is guaranteed to be atomic but not a series of updates (updateMany).

How do we recover from a failed `updateMany`?

updateMany scenario

Given an employee DB as follows:

```
db.employees.insertMany( [
    { "_id" : 1, "name" : "Rob", "salary" : 37000 },
    { "_id" : 2, "name" : "Trish", "salary" : 65000 },
    { "_id" : 3, "name" : "Zeke", "salary" : 99999 },
    { "_id" : 4, "name" : "Mary", "salary" : 200000 }
] )
```

We want to give everyone with salary < $100k a raise of $1k.

updateMany scenario

Possible approach ($inc operator to increment):

```
    db.employees.updateMany(
        { salary: { $lt: 100000 }},
        { $inc: { salary: 1000 } }
    )
```

What happens if `updateMany` fails?

How will we know which employees ended up getting a raise?

updateMany scenario

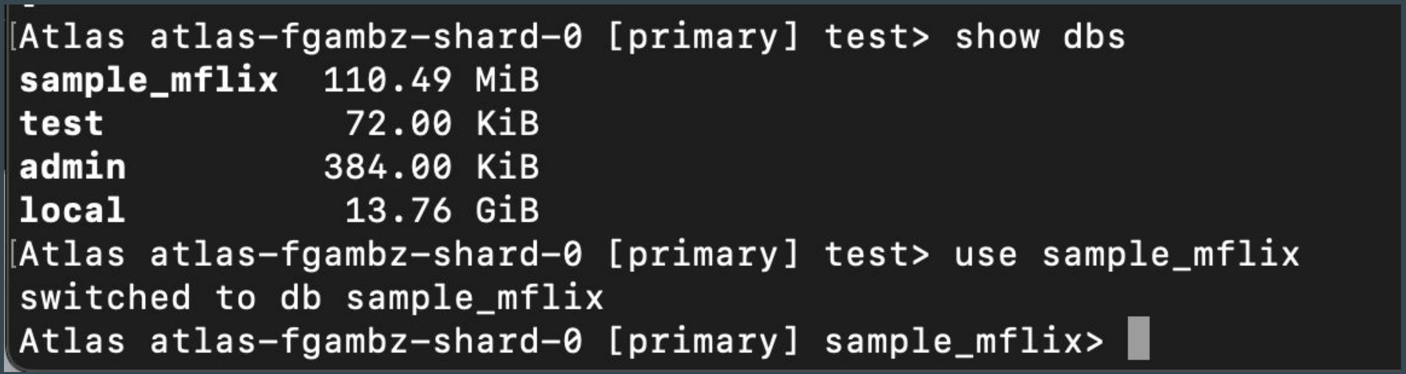
Better approach: mark records that have been updated.

```
db.employees.updateMany(
    { salary: { $lt: 100000 }, raiseApplied: { $ne: true } },
    { $inc: { salary: 1000 }, $set: { raiseApplied: true } }
)
```

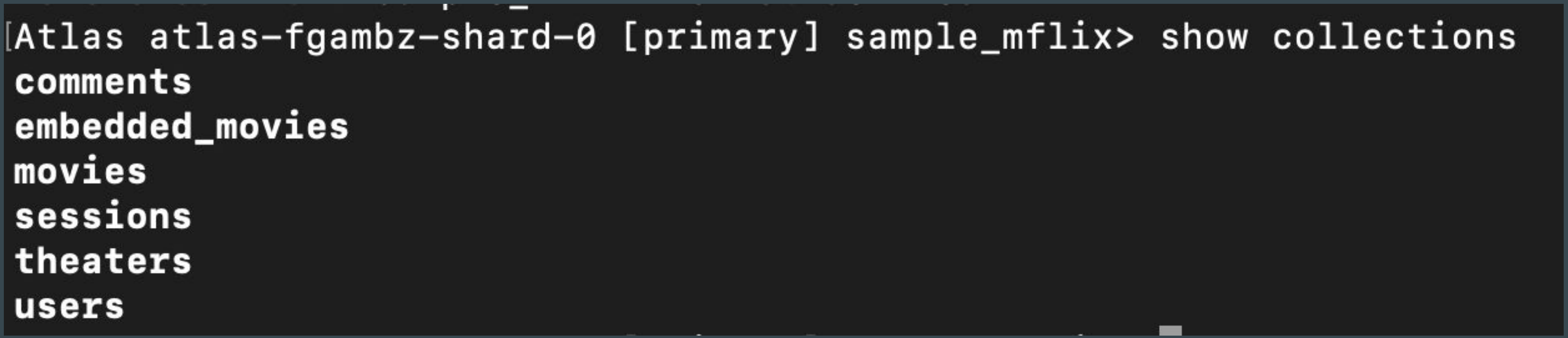
Running the operation again ensures that only those records that didn't get updated will be updated. Operation is idempotent.

Sample database: movies

Movies databases in Atlas



```
[Atlas atlas-fgambz-shard-0 [primary] test> show dbs
sample_mflix  110.49 MiB
test           72.00 KiB
admin         384.00 KiB
local          13.76 GiB
[Atlas atlas-fgambz-shard-0 [primary] test> use sample_mflix
switched to db sample_mflix
Atlas atlas-fgambz-shard-0 [primary] sample_mflix>
```

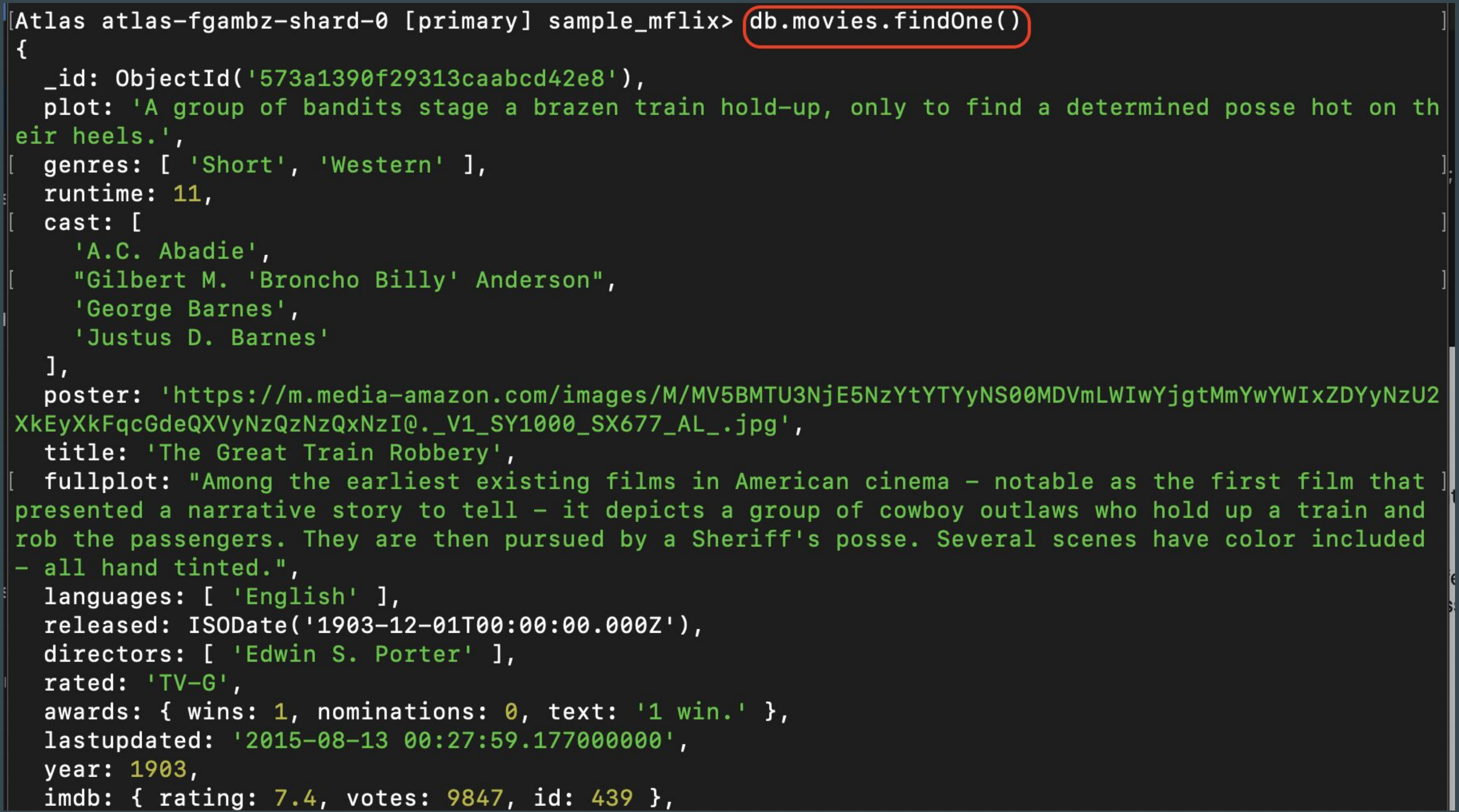


```
[Atlas atlas-fgambz-shard-0 [primary] sample_mflix> show collections
comments
embedded_movies
movies
sessions
theaters
users
```

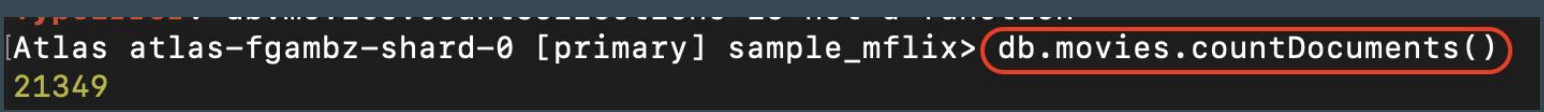
Use the "use" command to switch databases.

Note: "countries" collection is in database "test".

movies collection



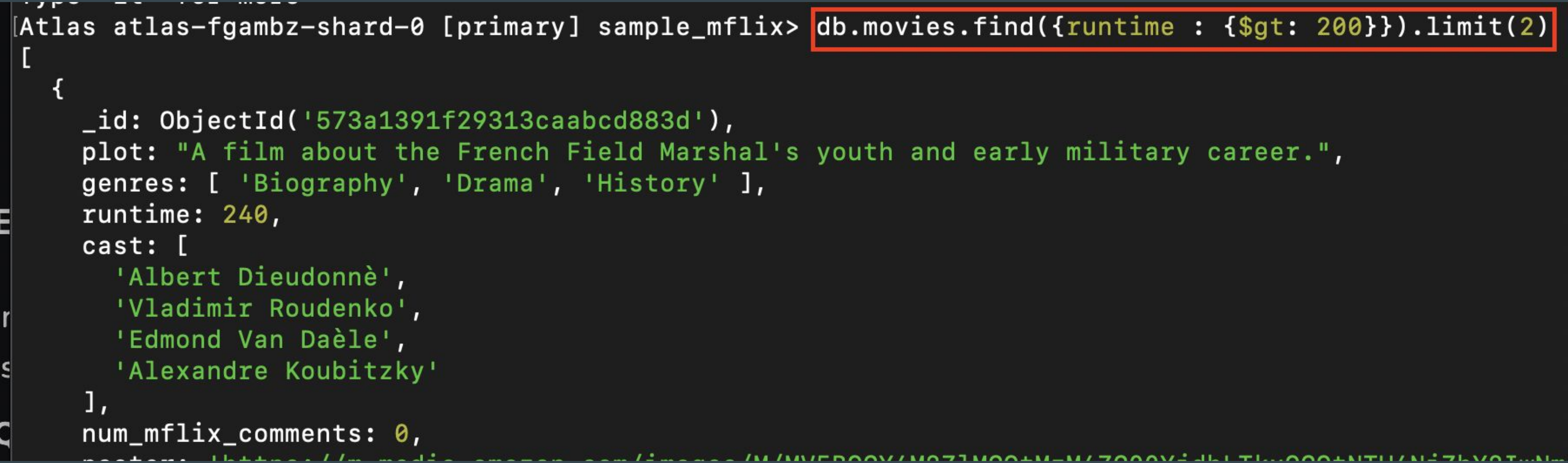
```
Atlas atlas-fgambz-shard-0 [primary] sample_mflix> db.movies.findOne()
{
  _id: ObjectId('573a1390f29313caabcd42e8'),
  plot: 'A group of bandits stage a brazen train hold-up, only to find a determined posse hot on th
eir heels.',
  genres: [ 'Short', 'Western' ],
  runtime: 11,
  cast: [
    'A.C. Abadie',
    "Gilbert M. 'Broncho Billy' Anderson",
    'George Barnes',
    'Justus D. Barnes'
  ],
  poster: 'https://m.media-amazon.com/images/M/MV5BMTU3NjE5NzYtYTYyNS00MDVmLWIwYjgtMmYwYWIxZDYyNzU2
XkEyXkFqcGdeQXVyNzQzNzQxNzI@._V1_SY1000_SX677_AL_.jpg',
  title: 'The Great Train Robbery',
  fullplot: "Among the earliest existing films in American cinema - notable as the first film that
presented a narrative story to tell - it depicts a group of cowboy outlaws who hold up a train and
rob the passengers. They are then pursued by a Sheriff's posse. Several scenes have color included
- all hand tinted.",
  languages: [ 'English' ],
  released: ISODate('1903-12-01T00:00:00.000Z'),
  directors: [ 'Edwin S. Porter' ],
  rated: 'TV-G',
  awards: { wins: 1, nominations: 0, text: '1 win.' },
  lastupdated: '2015-08-13 00:27:59.177000000',
  year: 1903,
  imdb: { rating: 7.4, votes: 9847, id: 439 },
```



```
Atlas atlas-fgambz-shard-0 [primary] sample_mflix> db.movies.countDocuments()
21349
```

Operator example: one filter

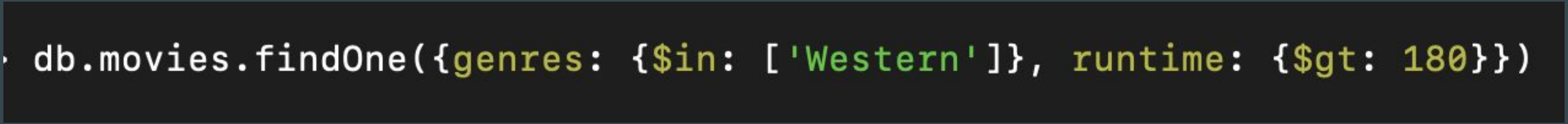
Find movies with runtime > 200 minutes and limit results to 2



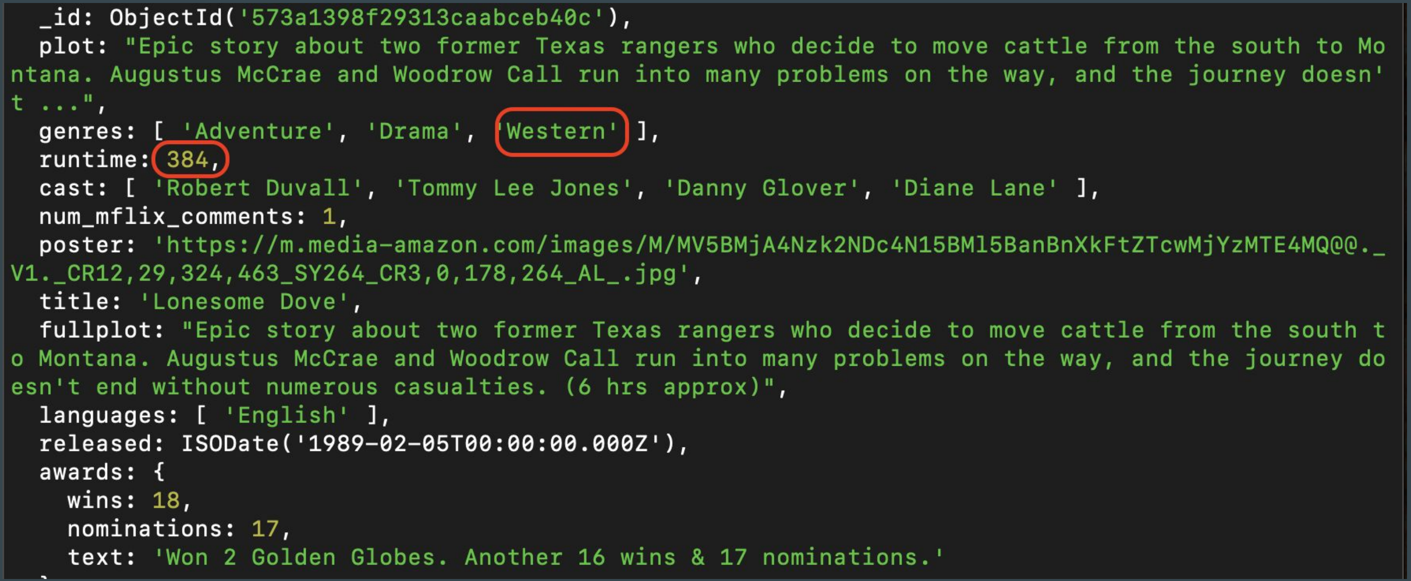
```
Atlas atlas-fgambz-shard-0 [primary] sample_mflix> db.movies.find({runtime : {$gt: 200}}).limit(2)
[
  {
    _id: ObjectId('573a1391f29313caabcd883d'),
    plot: "A film about the French Field Marshal's youth and early military career.",
    genres: [ 'Biography', 'Drama', 'History' ],
    runtime: 240,
    cast: [
      'Albert Dieudonnè',
      'Vladimir Roudenko',
      'Edmond Van Daèle',
      'Alexandre Koubitzky'
    ],
    num_mflix_comments: 0,
```

Operator example: two filters

Find the first Western movie with runtime > 180 minutes



```
db.movies.findOne({genres: {$in: ['Western']}, runtime: {$gt: 180}})
```



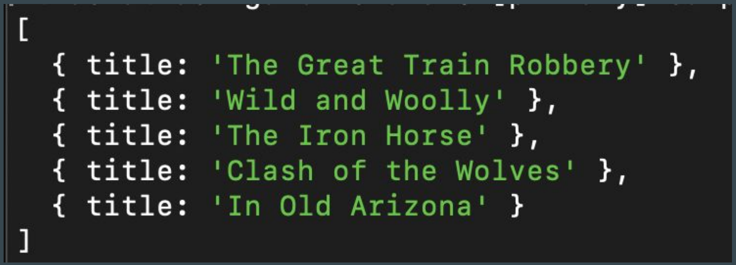
```
  _id: ObjectId('573a1398f29313caabceb40c'),
  plot: "Epic story about two former Texas rangers who decide to move cattle from the south to Mo
ntana. Augustus McCrae and Woodrow Call run into many problems on the way, and the journey doesn'
t ...",
  genres: [ 'Adventure', 'Drama', 'Western' ],
  runtime: 384,
  cast: [ 'Robert Duvall', 'Tommy Lee Jones', 'Danny Glover', 'Diane Lane' ],
  num_mflix_comments: 1,
  poster: 'https://m.media-amazon.com/images/M/MV5BMjA4Nzk2NDc4N15BMl5BanBnXkFtZTcwMjYzMTE4MQ@@._
V1._CR12,29,324,463_SY264_CR3,0,178,264_AL_.jpg',
  title: 'Lonesome Dove',
  fullplot: "Epic story about two former Texas rangers who decide to move cattle from the south t
o Montana. Augustus McCrae and Woodrow Call run into many problems on the way, and the journey do
esn't end without numerous casualties. (6 hrs approx)",
  languages: [ 'English' ],
  released: ISODate('1989-02-05T00:00:00.000Z'),
  awards: {
    wins: 18,
    nominations: 17,
    text: 'Won 2 Golden Globes. Another 16 wins & 17 nominations.'
```

Sort example

Find Western movies and sort them by title, limit results to 5

```
db.movies.find({genres: {$in: ['Western']}}, {_id:0, title: 1}).limit(5)
```

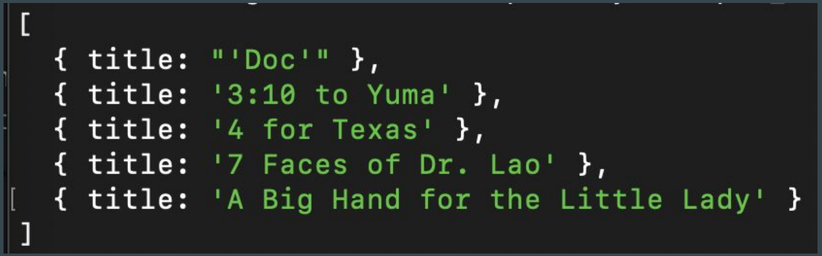
Unsorted



```
[
  { title: 'The Great Train Robbery' },
  { title: 'Wild and Woolly' },
  { title: 'The Iron Horse' },
  { title: 'Clash of the Wolves' },
  { title: 'In Old Arizona' }
]
```

```
db.movies.find({genres: {$in: ['Western']}}, {_id:0, title: 1}).limit(5).sort({title: 1})
```

Sort by title

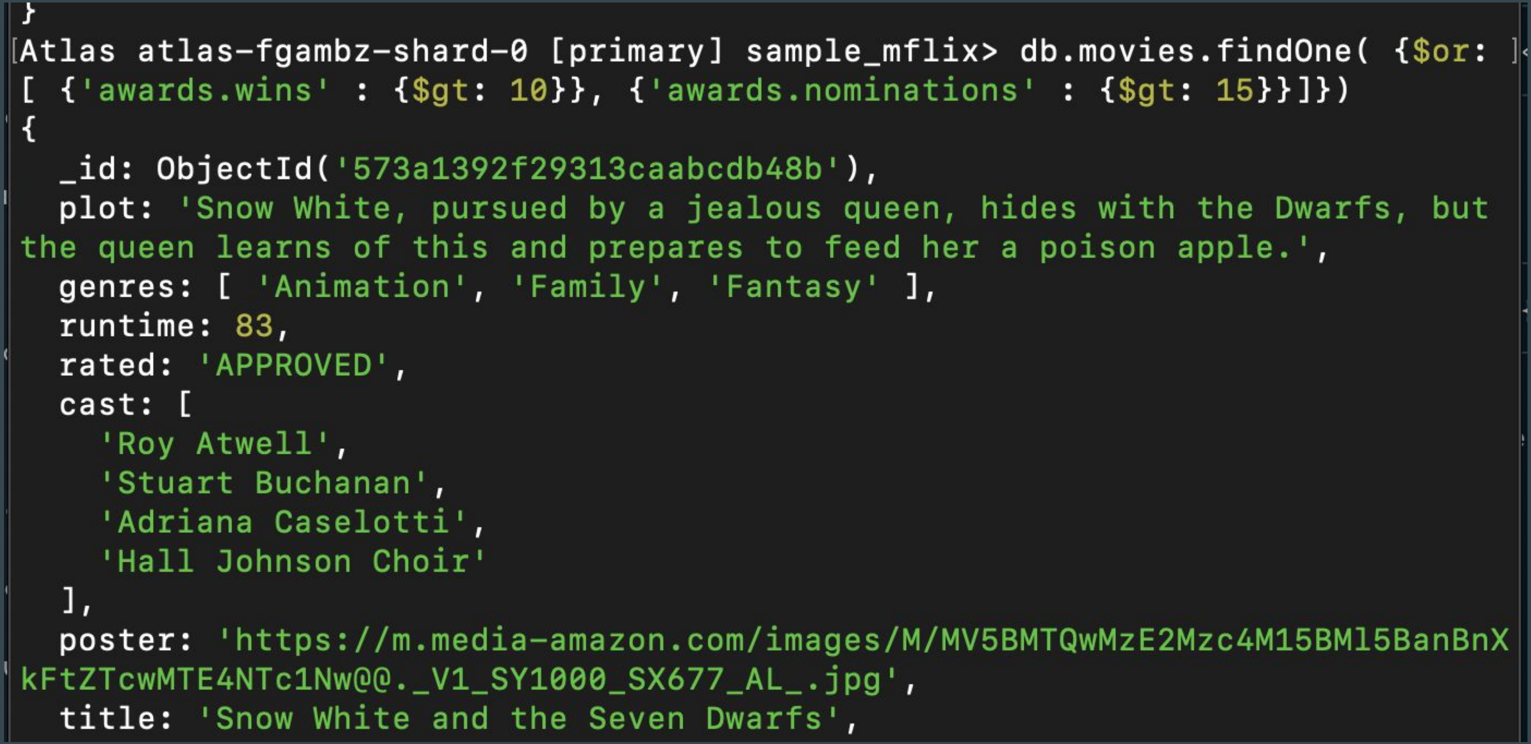


```
[
  { title: "'Doc'" },
  { title: '3:10 to Yuma' },
  { title: '4 for Texas' },
  { title: '7 Faces of Dr. Lao' },
  { title: 'A Big Hand for the Little Lady' }
]
```

Operator example: $or operator

Find movies with > 15 nominations or > 10 wins.

$or operator matches any filter in a list.

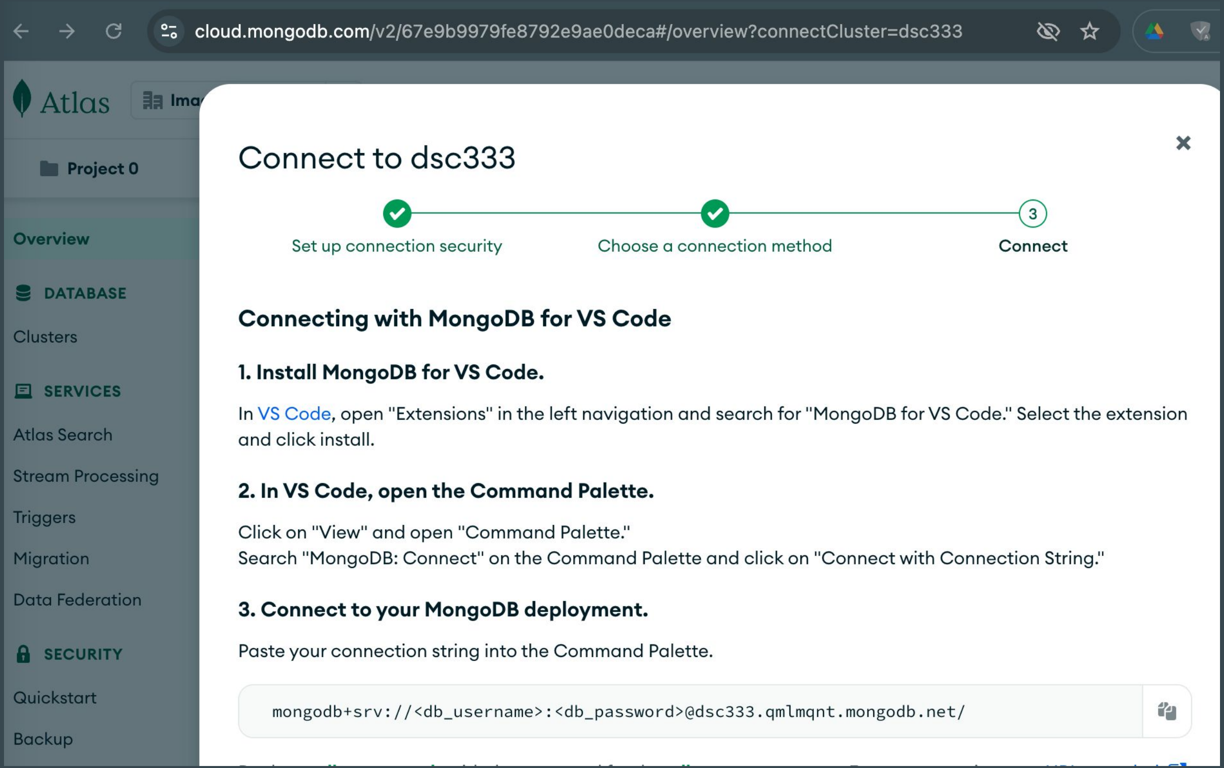


```
}
Atlas atlas-fgambz-shard-0 [primary] sample_mflix> db.movies.findOne( {$or:
[ {'awards.wins' : {$gt: 10}}, {'awards.nominations' : {$gt: 15}}]})
{
  _id: ObjectId('573a1392f29313caabcdb48b'),
  plot: 'Snow White, pursued by a jealous queen, hides with the Dwarfs, but
the queen learns of this and prepares to feed her a poison apple.',
  genres: [ 'Animation', 'Family', 'Fantasy' ],
  runtime: 83,
  rated: 'APPROVED',
  cast: [
    'Roy Atwell',
    'Stuart Buchanan',
    'Adriana Caselotti',
    'Hall Johnson Choir'
  ],
  poster: 'https://m.media-amazon.com/images/M/MV5BMTQwMzE2Mzc4M15BMl5BanBnX
kFtZTcwMTE4NTc1Nw@@._V1_SY1000_SX677_AL_.jpg',
  title: 'Snow White and the Seven Dwarfs',
```

MongoDB VSCode extension

Install MongoDB extension for VS Code

Initialize using your connection string (Connect → MongoDB for VS Code).



Connect to dsc333

Connecting with MongoDB for VS Code

1. Install MongoDB for VS Code.

In VS Code, open "Extensions" in the left navigation and search for "MongoDB for VS Code." Select the extension and click install.

2. In VS Code, open the Command Palette.

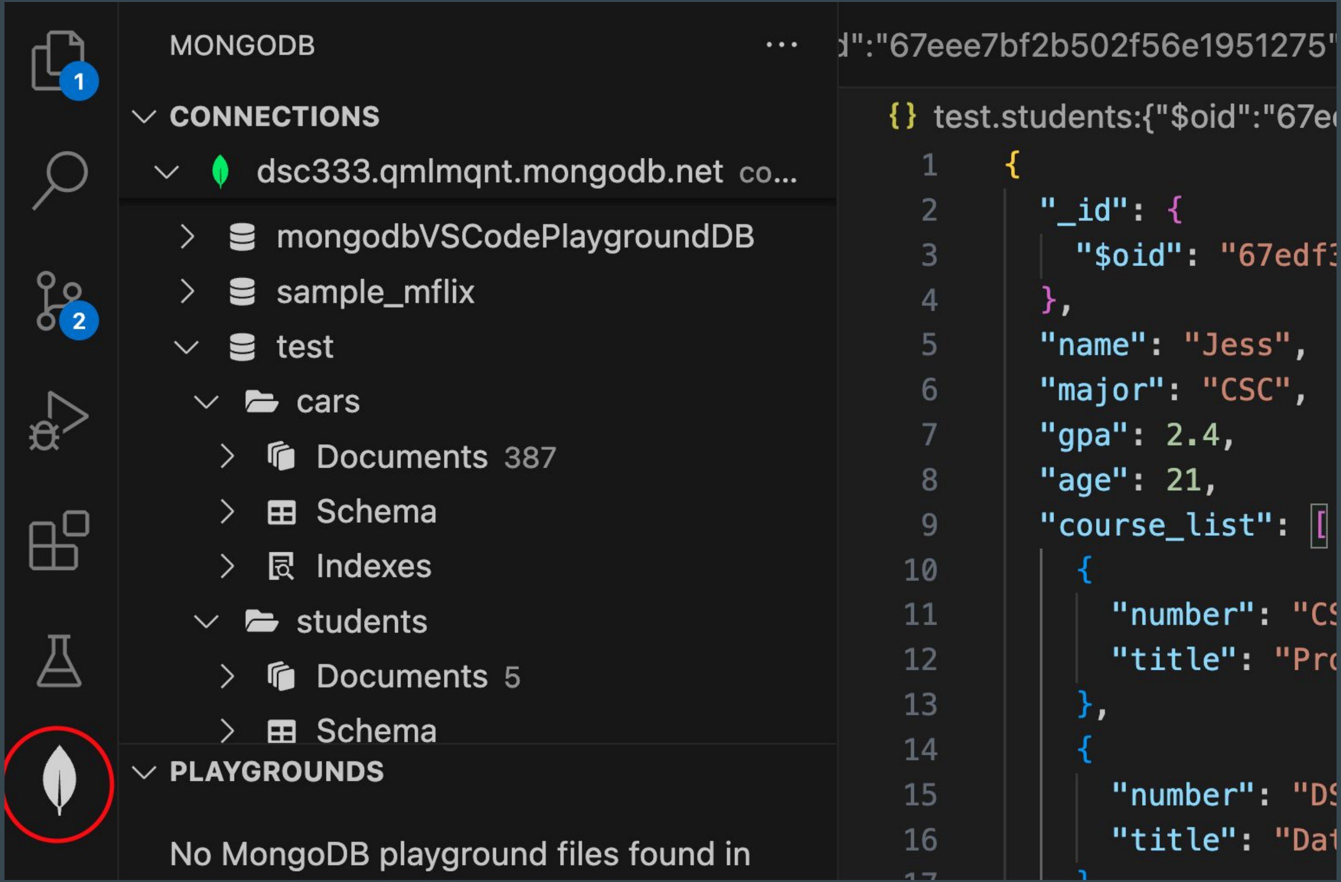
Click on "View" and open "Command Palette."
Search "MongoDB: Connect" on the Command Palette and click on "Connect with Connection String."

3. Connect to your MongoDB deployment.

Paste your connection string into the Command Palette.

```
mongodb+srv://<db_username>:<db_password>@dsc333.qmlmqnt.mongodb.net/
```

Browse Collections from VSCode



MONGODB ...

CONNECTIONS

dsc333.qmlmqnt.mongodb.net co...

mongodbVSCodePlaygroundDB

sample_mflix

test

cars

Documents 387

Schema

Indexes

students

Documents 5

Schema

PLAYGROUNDS

No MongoDB playground files found in

```
d":"67eee7bf2b502f56e1951275"
{} test.students:{"$oid":"67e
{
  "_id": {
    "$oid": "67edf3
  },
  "name": "Jess",
  "major": "CSC",
  "gpa": 2.4,
  "age": 21,
  "course_list": [
    {
      "number": "CS
      "title": "Pr
    },
    {
      "number": "DS
      "title": "Dat
```

MongoDB Python Driver

MongoDB Python Driver

pymongo is the Python driver, similar syntax to mongosh.

Test code: https://github.com/dsc333/mongo

- Pull (refresh) your code in VSCode
- Modify the connection string in crud-test.py
- Recreate the virtual environment (new libraries are added)
- Run the code

Aggregation Pipelines

Pipelines

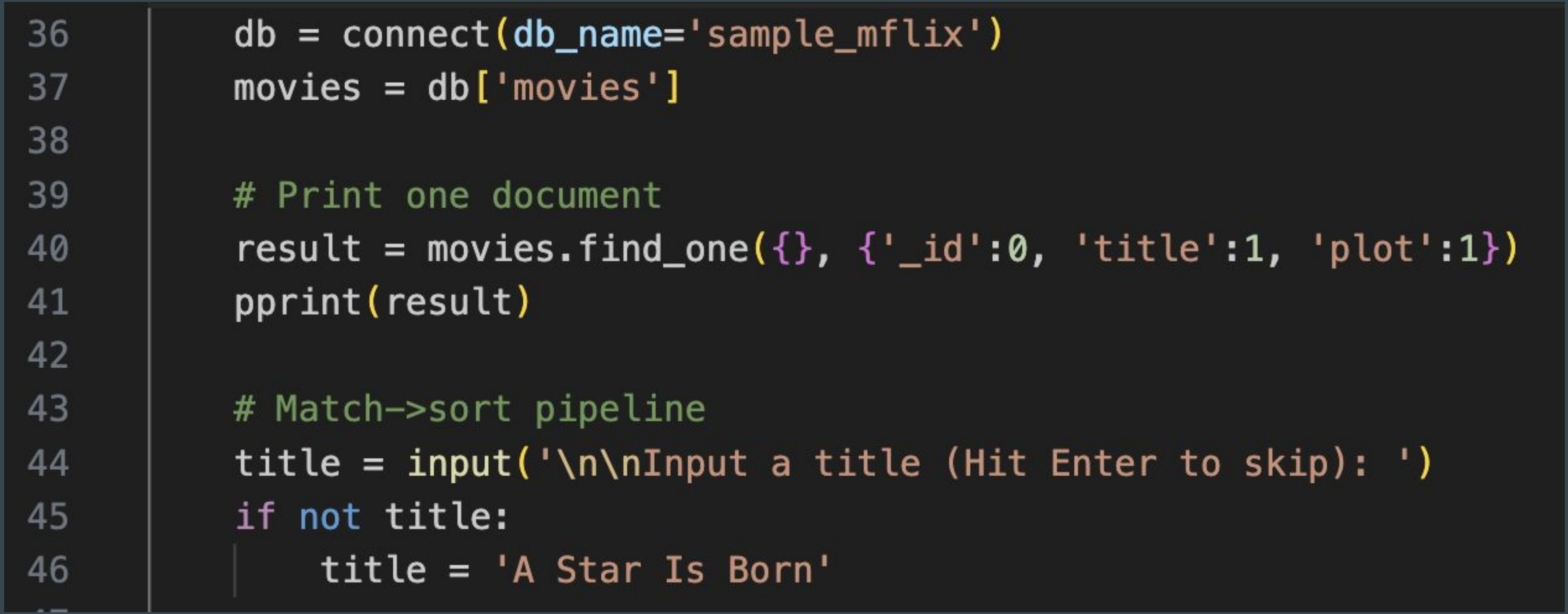
A pipeline defines a list of chained operations (pipeline stages) that are executed on a collection in sequence.

A pipeline stage can act as a filter or modify documents as they pass through a pipeline.

Pipelines are executed using the `aggregate` method.

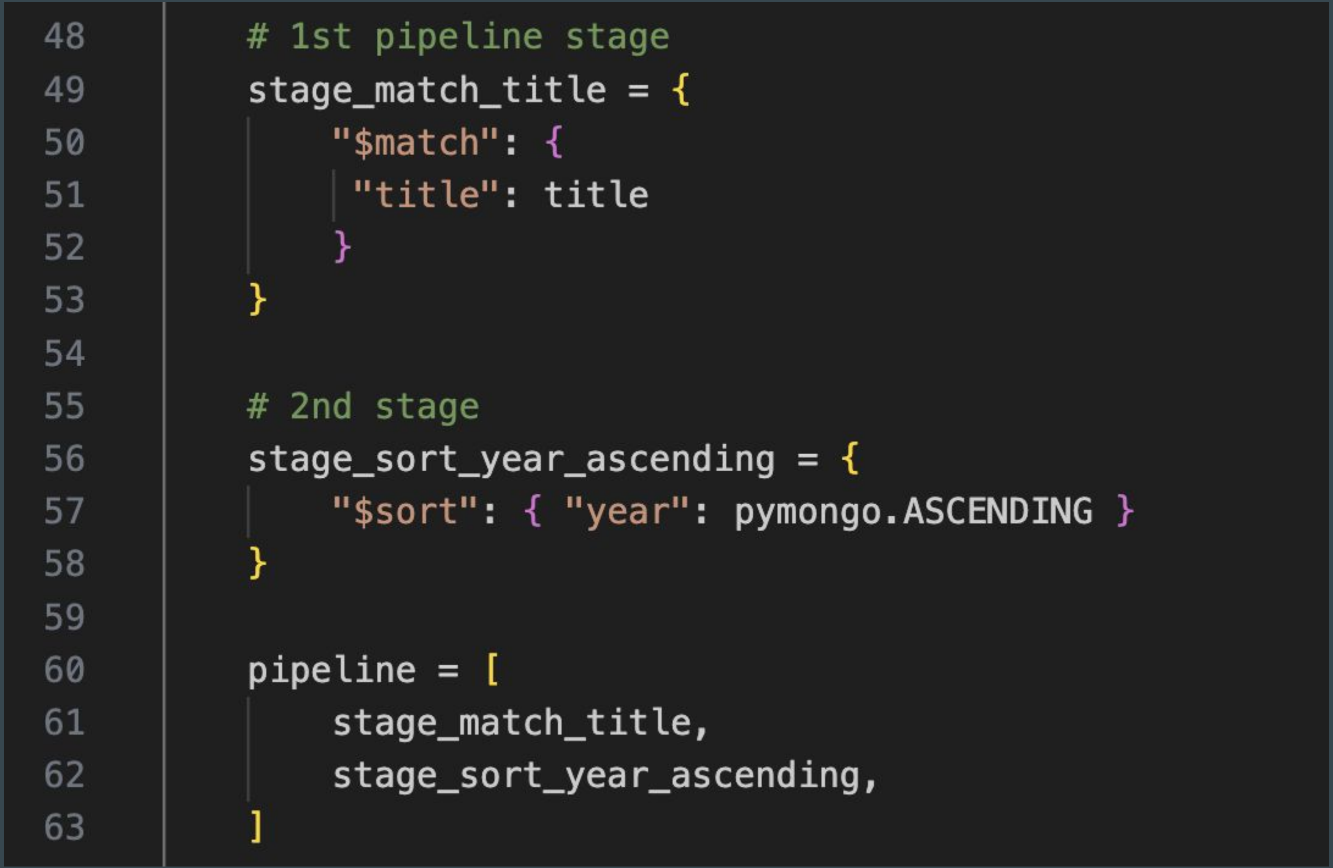
Example: pipeline-test.py

Uses the sample DB sample_mflix (provided by MongoDB)



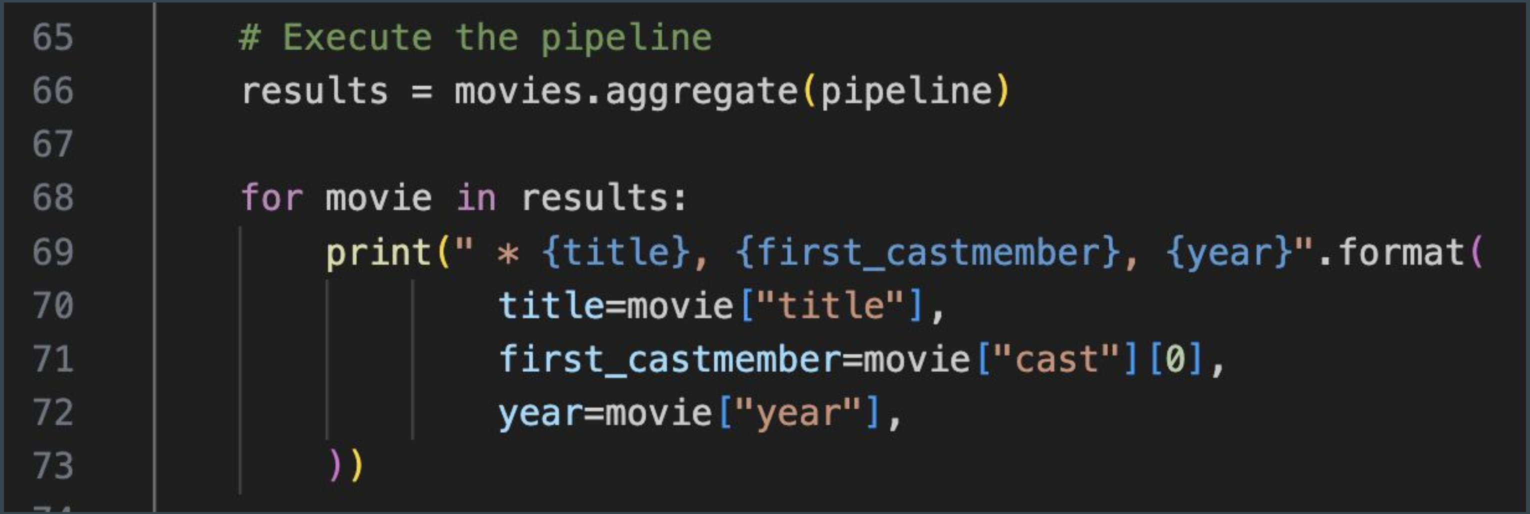
```python
    db = connect(db_name='sample_mflix')
    movies = db['movies']

    # Print one document
    result = movies.find_one({}, {'_id':0, 'title':1, 'plot':1})
    pprint(result)

    # Match->sort pipeline
    title = input('\n\nInput a title (Hit Enter to skip): ')
    if not title:
        title = 'A Star Is Born'
```

Define pipeline

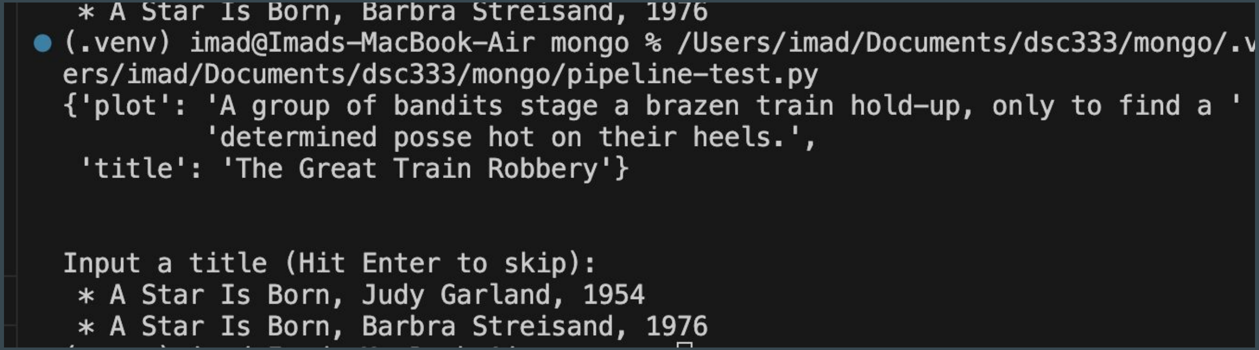


```
    # 1st pipeline stage
    stage_match_title = {
        "$match": {
         "title": title
        }
    }

    # 2nd stage
    stage_sort_year_ascending = {
        "$sort": { "year": pymongo.ASCENDING }
    }

    pipeline = [
        stage_match_title,
        stage_sort_year_ascending,
    ]
```

Execute pipeline and show results



```python
    # Execute the pipeline
    results = movies.aggregate(pipeline)

    for movie in results:
        print(" * {title}, {first_castmember}, {year}".format(
                title=movie["title"],
                first_castmember=movie["cast"][0],
                year=movie["year"],
        ))
```



```
 * A Star Is Born, Barbra Streisand, 1976
● (.venv) imad@Imads-MacBook-Air mongo % /Users/imad/Documents/dsc333/mongo/.v
ers/imad/Documents/dsc333/mongo/pipeline-test.py
{'plot': 'A group of bandits stage a brazen train hold-up, only to find a '
         'determined posse hot on their heels.',
 'title': 'The Great Train Robbery'}


Input a title (Hit Enter to skip):
 * A Star Is Born, Judy Garland, 1954
 * A Star Is Born, Barbra Streisand, 1976
```

Look up related data: A "join"

Comments collection

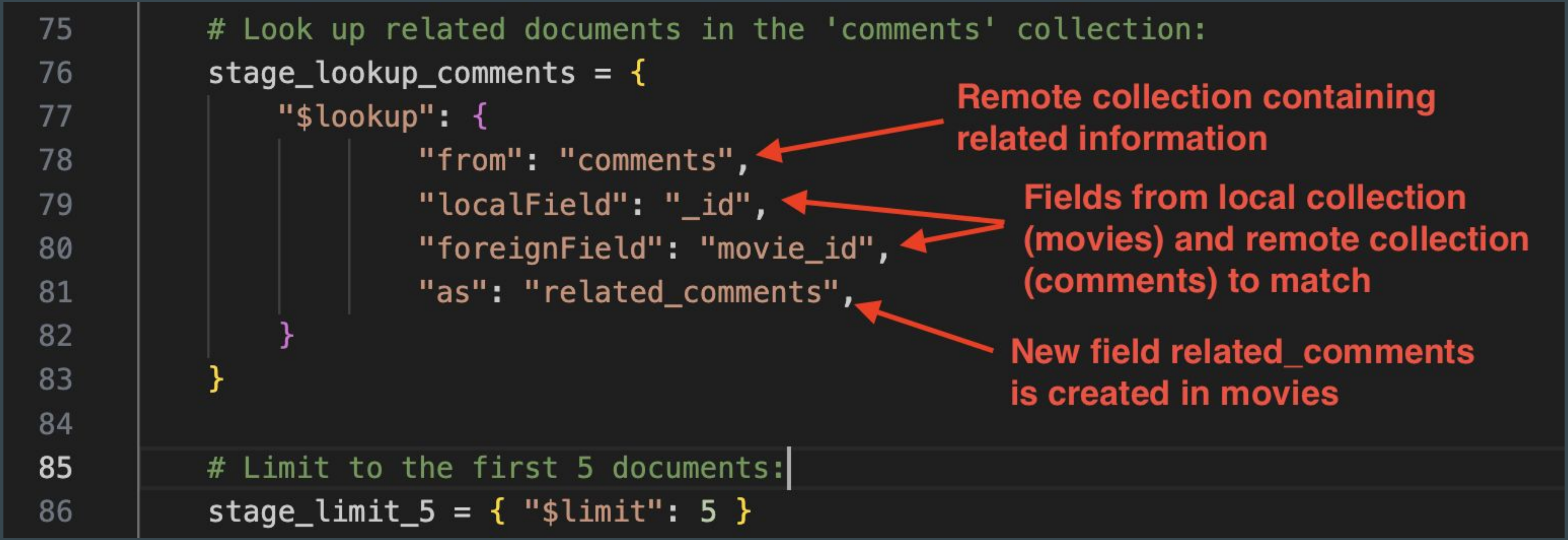
Movie comments are stored in a comments collection that references movie_id (acts as a foreign key)

Movie document

```
{
    '_id': ObjectId('5a9427648b0beebeb69579d3'),
    'movie_id': ObjectId('573a1390f29313caabcd4217'),
    'date': datetime.datetime(1983, 4, 27, 20, 39, 15),
    'email': 'cameron_duran@fakegmail.com',
    'name': 'Cameron Duran',
    'text': 'Quasi dicta culpa asperiores quaerat perferendis neque. Est animi '
             'pariatur impedit itaque exercitationem.'}
```

Goal: extract comments for a movie

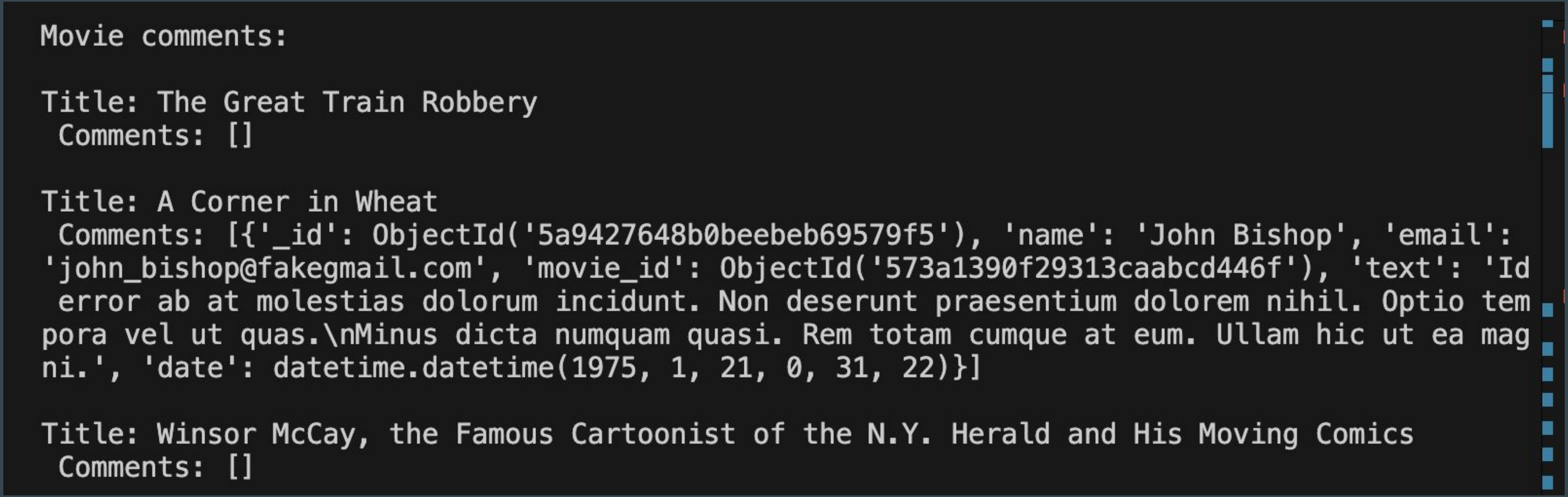
Pipeline of $lookup and $limit operations



```
    # Look up related documents in the 'comments' collection:
    stage_lookup_comments = {
        "$lookup": {
                "from": "comments",
                "localField": "_id",
                "foreignField": "movie_id",
                "as": "related_comments",
        }
    }

    # Limit to the first 5 documents:
    stage_limit_5 = { "$limit": 5 }
```

Remote collection containing related information
Fields from local collection (movies) and remote collection (comments) to match
New field related_comments is created in movies

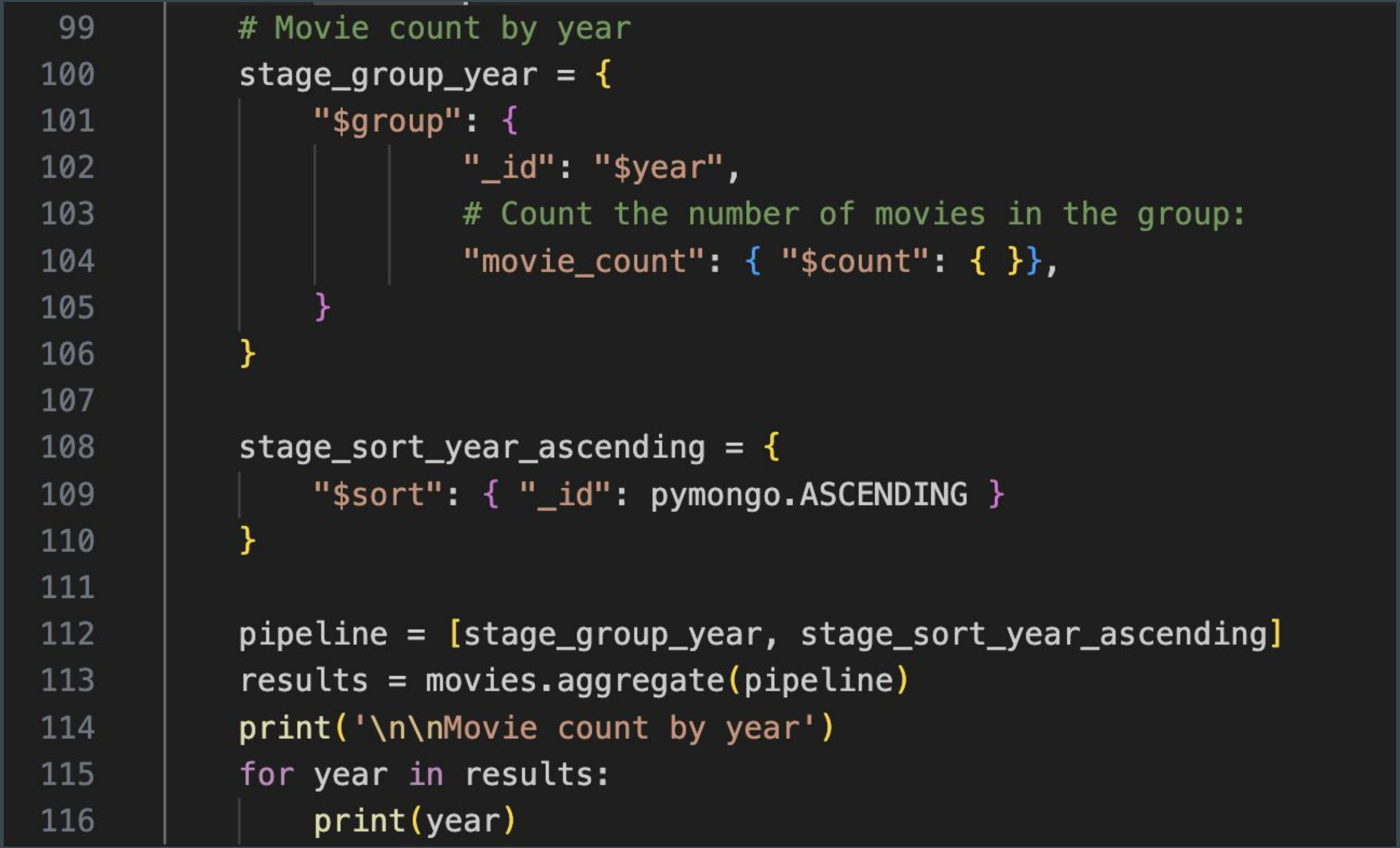
Results



```
Movie comments:

Title: The Great Train Robbery
 Comments: []

Title: A Corner in Wheat
 Comments: [{'_id': ObjectId('5a9427648b0beebeb69579f5'), 'name': 'John Bishop', 'email':
'john_bishop@fakegmail.com', 'movie_id': ObjectId('573a1390f29313caabcd446f'), 'text': 'Id
 error ab at molestias dolorum incidunt. Non deserunt praesentium dolorem nihil. Optio tem
pora vel ut quas.\nMinus dicta numquam quasi. Rem totam cumque at eum. Ullam hic ut ea mag
ni.', 'date': datetime.datetime(1975, 1, 21, 0, 31, 22)}]

Title: Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics
 Comments: []
```

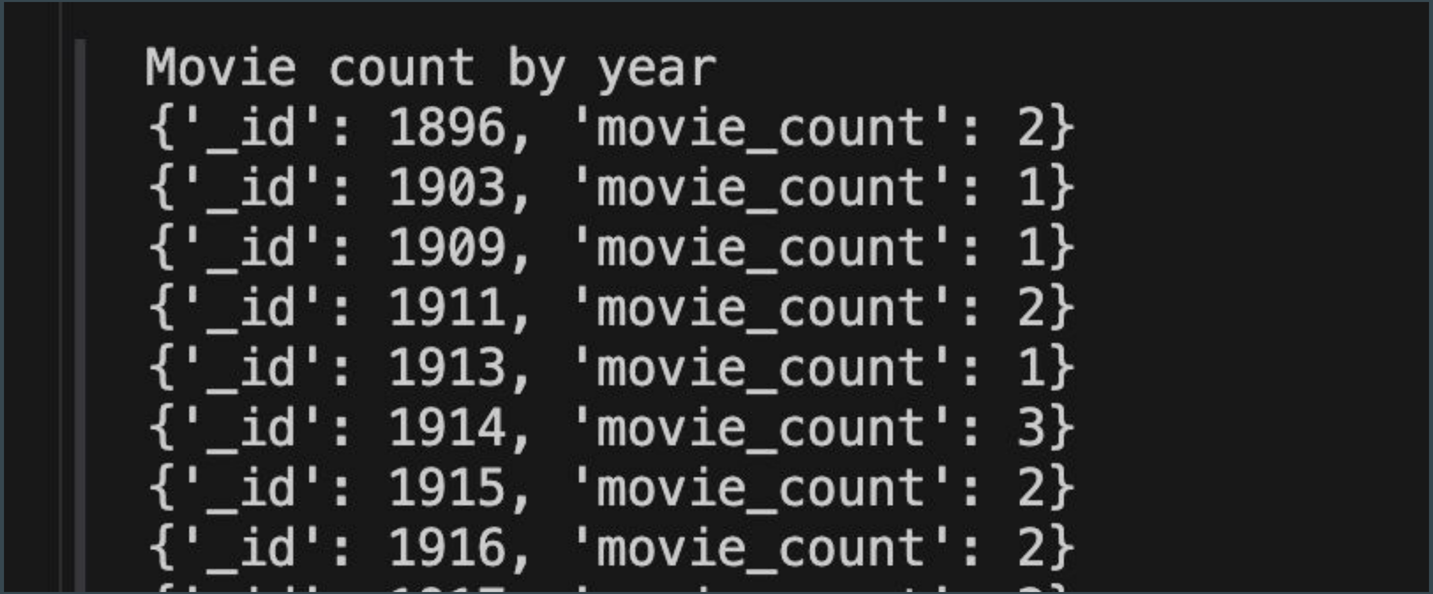
Grouping

Count movies by year



```
    # Movie count by year
    stage_group_year = {
        "$group": {
                "_id": "$year",
                # Count the number of movies in the group:
                "movie_count": { "$count": { }},
        }
    }

    stage_sort_year_ascending = {
        "$sort": { "_id": pymongo.ASCENDING }
    }

    pipeline = [stage_group_year, stage_sort_year_ascending]
    results = movies.aggregate(pipeline)
    print('\n\nMovie count by year')
    for year in results:
        print(year)
```

Results



```
Movie count by year
{'_id': 1896, 'movie_count': 2}
{'_id': 1903, 'movie_count': 1}
{'_id': 1909, 'movie_count': 1}
{'_id': 1911, 'movie_count': 2}
{'_id': 1913, 'movie_count': 1}
{'_id': 1914, 'movie_count': 3}
{'_id': 1915, 'movie_count': 2}
{'_id': 1916, 'movie_count': 2}
```

References

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

https://www.mongodb.com/developer/languages/python/python-quickstart-aggregation/

https://www.mongodb.com/docs/languages/python/pymongo-driver/current/crud/

https://www.mongodb.com/docs/manual/reference/sql-comparison/