DB notes:

The database we worked with: mflix
The collections in the database:

- Comments
- Movies
- Theaters
- Users

The different queries used to answer questions on how to explore this MongoDB:

How to set up your connections to the Mongo DB (database):

```
# Set up your connection to Mongo DB here.
import pymongo
from bson.json_util import dumps

uri = "mongodb://mimi:mimi123@localhost:27017"
client = pymongo.MongoClient(uri)
mflixdb = client.mflix
```

Give the street, city, and zipcode of all theaters in Massachusetts.

```
# select street, city, and zipcode from theaters in Massachusetts
data = mflixdb.theaters.find({"location.address.state": "MA"},
                             {"location.address.street": 1,
                              "location.address.city": 1,
                              "location.address.zipcode": 1})
print(dumps(data, indent=2))
```

How many theaters are there in each state? Order the output in alphabetical order by 2-character state code.

```
# count of theaters order by location state

data = mflixdb.theaters.aggregate([
   # count theaters per state
   {"$group": {"_id": "$location.address.state", "count": {"$sum": 1}}},
   # sort alphabetically
   {"$sort": {"_id": 1}},
   # rename id to state
   {"$project": {"_id": 0, "state": "$_id", "count": 1}}
])

print(dumps(data, indent=2))
```

How many movies are in the Comedy genre?

```
# counting comedy movies genre
data = mflixdb.movies.count_documents({"genres": "Comedy"})
print(dumps(data, indent=2))
```

What movie has the longest run time? Give the movie's title and genre(s).

```
# find match and sort in descending order
data = mflixdb.movies.aggregate([
   # sort in descending order
   {"$sort": {"runtime": -1}},
   # get first result
   {"$limit": 1},
   # only keep title and genres
   {"$project": {"_id": 0, "title": 1, "genres": 1}}
])

print(dumps(data, indent=2))
```

Which movies released after 2010 have a Rotten Tomatoes viewer rating of 3 or higher? Give the title of the movies along with their Rotten Tomatoes viewer rating score. The viewer rating score should become a top-level attribute of the returned documents. Return the matching movies in descending order by viewer rating.

```
data = mflixdb.movies.aggregate([# movies released after 2010 with rating higehr than 3
   {"$match": {"year": {"$gt": 2010}, "tomatoes.viewer.rating": {"$gte": 3}}},
   # descending order by viewer rating
   {"$sort": {"tomatoes.viewer.rating" : -1}},
   # make viewer rating a top level field
   {"$project": {"_id": 0, "title": 1, "viewer_rating": "$tomatoes.viewer.rating"}}
   ])

print(dumps(data, indent = 4))
```

How many movies released each year have a plot that contains some type of police activity (i.e., plot contains the word "police")? The returned data should be in ascending order by year.

```
data = mflixdb.movies.aggregate([
   # find movies with plot including police
   {"$match": {"plot": {"$regex": "police"}}},
   # count movies per year
   {"$group": {"_id": "$year", "count": {"$sum": 1}}},
   # ascending order
```

```
    {"$sort": {"_id": 1}},
    # rename id to year to show by year
    {"$project": {"_id": 0, "year": "$_id", "count": 1}}


])
print(dumps(data, indent=2))
```

What is the average number of imdb votes per year for movies released between 1970 and 2000 (inclusive)? Make sure the results are order by year.

```
# averaging
data = mflixdb.movies.aggregate([
    # between 1999 and 2000
    {"$match": {"year": {"$gte": 1970, "$lte": 2000}}},
    {"$group": {"_id": {"release year": "$year"}, "Avg Votes": {"$avg": "$imdb.votes"}}},
    {"$sort": {"_id": 1}},
    {"$project": {"_id": 0, "year": "$_id", "average votes": {"$round": ["$Avg Votes", 2]}}},

])

print(dumps(data, indent=2))
```

What distinct movie languages are represented in the database? You only need to provide the list of languages.

```
# retrieve distinct languages from the movies collection
languages_db = set()

languages_db.update(mflixdb.movies.distinct("languages"))

# print the distinct languages as a list
print(list(languages_db))
```

The $nin operator in MongoDB is used to filter documents where the value of a field is not in the specified array. It is the opposite of the $in operator, which checks if a field's value is within an array.