MongoDB + PyMongo Example Queries

- Make sure your MongoDB container is running
- Make sure you have pymongo installed before running cells in this notebook. If not, use `pip install pymongo`.

```
import pymongo
from bson.json_util import dumps

#  -->  Update the URI with your username and password <--

uri = "mongodb://sawyerss:Northeastern1!!@localhost:27017"
client = pymongo.MongoClient(uri)
mflixdb = client.mflix
demodb = client.demodb

# Setup DemoDB with 2 collections
demodb.customers.drop()
demodb.orders.drop()

customers = [
  {"custid": "C13", "name": "T. Cruise", "address": { "street": "201 Main St.", "city": "St. Louis, MO", "zipcode": "63101" }, "rating": 750 },
  {"custid": "C25", "name": "M. Streep", "address": { "street": "690 River St.", "city": "Hanover, MA", "zipcode": "02340" }, "rating": 690 },
  {"custid": "C31", "name": "B. Pitt", "address": { "street": "360 Mountain Ave.", "city": "St. Louis, MO", "zipcode": "63101" } },
  {"custid": "C35", "name": "J. Roberts", "address": { "street": "420 Green St.", "city": "Boston, MA", "zipcode": "02115" }, "rating": 565 },
  {"custid": "C37", "name": "T. Hanks", "address": { "street": "120 Harbor Blvd.", "city": "Boston, MA", "zipcode": "02115" }, "rating": 750 },
  {"custid": "C41", "name": "R. Duvall", "address": { "street": "150 Market St.", "city": "St. Louis, MO", "zipcode": "63101" }, "rating": 640 },
  {"custid": "C47", "name": "S. Loren", "address": { "street": "Via del Corso", "city": "Rome, Italy" }, "rating": 625 }
]

orders = [
  { "orderno": 1001, "custid": "C41", "order_date": "2017-04-29", "ship_date": "2017-05-03", "items": [ { "itemno": 347, "qty": 5, "price": 19.99 }, { "itemno": 193, "qty": 2, "price": 28.89 } ] },
```

```
  { "orderno": 1002, "custid": "C13", "order_date": "2017-05-01", "ship_date": "2017-05-03",
"items": [ { "itemno": 460, "qty": 95, "price": 100.99 }, { "itemno": 680, "qty": 150, "price": 8.75 } ]
},
  { "orderno": 1003, "custid": "C31", "order_date": "2017-06-15", "ship_date": "2017-06-16",
"items": [ { "itemno": 120, "qty": 2, "price": 88.99 }, { "itemno": 460, "qty": 3, "price": 99.99 } ] },
  { "orderno": 1004, "custid": "C35", "order_date": "2017-07-10", "ship_date": "2017-07-15",
"items": [ { "itemno": 680, "qty": 6, "price": 9.99 }, { "itemno": 195, "qty": 4, "price": 35.00 } ] },
  { "orderno": 1005, "custid": "C37", "order_date": "2017-08-30", "items": [ { "itemno": 460, "qty":
2, "price": 99.98 }, { "itemno": 347, "qty": 120, "price": 22.00 }, { "itemno": 780, "qty": 1, "price":
1500.00  }, { "itemno": 375, "qty": 2, "price": 149.98 } ] },
  { "orderno": 1006, "custid": "C41", "order_date": "2017-09-02", "ship_date": "2017-09-04",
"items": [ { "itemno": 680, "qty": 51, "price": 25.98 }, { "itemno": 120, "qty": 65, "price": 85.00 }, {
"itemno": 460, "qty": 120, "price": 99.98 } ] },
  { "orderno": 1007, "custid": "C13", "order_date": "2017-09-13", "ship_date": "2017-09-20",
"items": [ { "itemno": 185, "qty": 5, "price": 21.99 }, { "itemno": 680, "qty": 1, "price": 20.50 } ] },
  { "orderno": 1008, "custid": "C13", "order_date": "2017-10-13", "items": [ { "itemno": 460, "qty":
20, "price": 99.99 } ] }
]

demodb.customers.insert_many(customers)
demodb.orders.insert_many(orders)

numCustomers = demodb.customers.count_documents({})
numOrders = demodb.orders.count_documents({})

print(f'There are {numCustomers} customers and {numOrders} orders')

# The key (_id) attribute is automatically returned unless you explicitly say to remove it.

# SELECT name, rating FROM customers
data = demodb.customers.find({}, {"name":1, "rating":1})
print(dumps(data, indent=2))

# Now without the _id field.

# SELECT name, rating FROM customers
data = demodb.customers.find({}, {"name":1, "rating":1, "_id":0})
print(dumps(data, indent=2))
```

All fields EXCEPT specific ones returned

```
# For every customer, return all fields except _id and address.

data = demodb.customers.find({}, {"_id": 0, "address": 0})
```

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

Equivalent to SQL LIKE operator

```
# SELECT name, rating FROM customers WHERE name LIKE 'T%'

# Regular Expression Explanation:
  # ^ - match beginning of line
  # T - match literal character T (at the beginning of the line in this case)
  # . - match any single character except newline
  # * - match zero or more occurrences of the previous character (the . in this case)

data = demodb.customers.find({"name": {"$regex": "^T.*"}}, {"_id": 0, "name": 1, "rating":1}) #^T.*
equivalent to LIKE 'T%' (anything that starts with T.)
print(dumps(data, indent=2)) # have to use dumps
```

Sorting and limiting

```
# SELECT name, rating FROM customers ORDER BY rating LIMIT 2

data = demodb.customers.find( { }, {"_id": 0, "name": 1, "rating":1} ).sort("rating").limit(2) # no
document filters so { }, second { } is filter for attributes
print(dumps(data, indent=2))

# Same as above, but sorting in DESC order

# SELECT name, rating FROM customers ORDER BY rating DESC LIMIT 2

data = demodb.customers.find( { }, {"_id": 0, "name": 1, "rating":1} ).sort("rating", -1).limit(2)
print(dumps(data, indent=2))

# Providing 2 sort keys...

data = demodb.customers.find( { }, {"_id": 0, "name": 1, "rating":1} ).sort({"rating": -1, "name":
1}).limit(2) # rating descending order, name ascending order
print(dumps(data, indent=2))
```

Your Turn with mflix DB

Question 1

```
# How many Users are there in the mflix database? How many movies?
numUsers = mflixdb.users.count_documents({})
numMovies = mflixdb.movies.count_documents({})

print(f'There are {numUsers} users and {numMovies} movies')
```

Question 2

```
# Which movies have a rating of “TV-G”? Only return the Title and Year.
data = mflixdb.movies.find({"rated": "TV-G"}, {"_id": 0, "title": 1, "year": 1})
print(dumps(data, indent=2))
```

Question 3

```
# Which movies have a runtime of less than 20 minutes?  Only return the title and runtime of each movie.
data = mflixdb.movies.find({"runtime": {"$lt": 20}}, {"_id": 0, "title": 1, "runtime": 1})
print(dumps(data, indent=2))
```

Question 4

```
# How many theaters are in MN or MA?
count = mflixdb.theaters.count_documents({"location.address.state": {"$in": ["MN", "MA"]}})
print(f'There are {count} theaters in MN or MA')
```

Question 5

```
# Give the names of all movies that have no comments yet. Make sure the names are in alphabetical order.
commented_movies = mflixdb.comments.distinct("movie_id")

movies_no_comments = mflixdb.movies.find({"_id": {"$nin": commented_movies}}, {"_id": 0, "title": 1}).sort("title", 1)
print(dumps(movies_no_comments, indent=2))
```

Question 6

```
# Return a list of movie titles and all actors from any movie with a title that contains the word 'Four'.
# Sort the list by title.
data = mflixdb.movies.find({"title": {"$regex": "Four", "$options": "i"}}, {"_id": 0, "title": 1, "cast": 1}).sort("title", 1)
print(dumps(data, indent=2, ensure_ascii = False)) # ensure_ascii will properly render letters that have non-ASCII marks
```