11 MongoDB

Lab Objective: *Relational databases, including those managed with SQL or pandas, require data to be organized into tables. However, many data sets have an inherently dynamic structure that cannot be efficiently represented as tables. MongoDB is a non-relational database management system that is well-suited to large, fast-changing datasets. In this lab we introduce the Python interface to MongoDB, including common commands and practices.*

Database Initialization

Suppose the manager of a general store has all sorts of inventory: food, clothing, tools, toys, etc. There are some common attributes shared by all items: name, price, and producer. However, other attributes are unique to certain items: sale price, number of wheels, or whether or not the product is gluten-free. A relational database housing this data would be full of mostly-blank rows, which is extremely inefficient. In addition, adding new items to the inventory requires adding new columns, causing the size of the database to rapidly increase. To efficiently store the data, the whole database would have to be restructured and rebuilt often.

To avoid this problem, NoSQL databases like MongoDB avoid using relational tables. Instead, each item is a JSON-like object, and thus can contain whatever attributes are relevant to the specific item, without including any meaningless attribute columns.

> **Note**
>
> MongoDB is a database management system (DBMS) that runs on a server, which should be running in its own dedicated terminal. Refer to the Additional Material section for installation instructions.

The Python interface to MongoDB is called `pymongo`. After installing `pymongo` **and** with the MongoDB server running, use the following code to connect to the server.

```
>>> from pymongo import MongoClient
# Create an instance of a client connected to a database running
# at the default host IP and port of the MongoDB service on your machine.
>>> client = MongoClient()
```

Creating Collections and Documents

A MongoDB database stores *collections*, and a collection stores *documents*. The syntax for creating databases and collections is a little unorthodox, as it is done through attributes instead of methods.

```
# Create a new database.
>>> db = client.db1

# Create a new collection in the db database.
>>> col = db.collection1
```

Documents in MongoDB are represented as JSON-like objects, and therefore do not adhere to a set schema. Each document can have its own *fields*, which are completely independent of the fields in other documents.

```
# Insert one document with fields 'name' and 'age' into the collection.
>>> col.insert_one({'name': 'Jack', 'age': 23})

# Insert another document. Notice that the value of a field can be a string,
# integer, truth value, or even an array.
>>> col.insert_one({'name': 'Jack', 'age': 22, 'student': True,
...                 'classes': ['Math', 'Geography', 'English']})

# Insert many documents simultaneously into the collection.
>>> col.insert_many([
...     {'name': 'Jill', 'age': 24, 'student': False},
...     {'name': 'John', 'nickname': 'Johnny Boy', 'soldier': True},
...     {'name': 'Jeremy', 'student': True, 'occupation': 'waiter'}  ])
```

Note

Once information has been added to the database it will remain there, even if the python environment you are working with is shut down. It can be reaccessed anytime using the same commands as before.

```
>>> client = MongoClient()
>>> db = client.db1
>>> col = db.collection1
```

To delete a collection, use the database's `drop_collection()` method. To delete a database, use the client's `drop_database()` method.

Problem 1. The file `trump.json` contains posts from `http://www.twitter.com` (tweets) over the course of an hour that have the key word "trump".[a] Each line in the file is a single JSON message that can be loaded with `json.loads()`.

Create a MongoDB database and initialize a collection in the database. Use the collection's `delete_many()` method with an empty set as input to clear existing contents of the collection, then fill the collection one line at a time with the data from `trump.json`. Check that your collection has 95,643 entries with its `count()` method.

[a] See the Additional Materials section for an example of using the Twitter API.

Querying a Collection

MongoDB uses a *query by example* pattern for querying. This means that to query a database, an example must be provided for the database to use in matching other documents.

```
# Find all the documents that have a 'name' field containing the value 'Jack'.
>>> data = col.find({'name': 'Jack'})

# Find the FIRST document with a 'name' field containing the value 'Jack'.
>>> data = col.find_one({'name': 'Jack'})
```

The `find_one()` method returns the first matching document as a dictionary. The `find()` query may find any number of objects, so it will return a `Cursor`, a Python object that is used to iterate over the query results. There are many useful functions that can be called on a `Cursor`, for more information see `http://api.mongodb.com/python/current/api/pymongo/cursor.html`.

```
# Search for documents containing True in the 'student' field.
>>> students = col.find({'student': True})
>>> students.count()                # There are 2 matching documents.
2

# List the first student's data.
# Notice that each document is automatically assigned an ID number as '_id'.
>>> students[0]
{'_id': ObjectId('59260028617410748cc7b8c7'),
 'age': 22,
 'classes': ['Math', 'Geography', 'English'],
 'name': 'Jack',
 'student': True}

# Get the age of the first student.
>>> students[0]['age']
22

# List the data for every student.
>>> list(students)
[{'_id': ObjectId('59260028617410748cc7b8c7'),
```

```
  'age': 22,
  'classes': ['Math', 'Geography', 'English'],
  'name': 'Jack',
  'student': True},
 {'_id': ObjectId('59260028617410748cc7b8ca'),
  'name': 'Jeremy',
  'occupation': 'waiter',
  'student': True}]
```

The Logical operators listed in the following table can be used to do more complex queries.

Operator	Description
$lt, $gt	<, >
$lte,$gte	<=, >=
$eq, $ne	==, !=
$in, $nin	in, not in
$or, $and, $not	or, and, not
$exists	Match documents with a specific field
$type	Match documents with values of a specific type
$all	Match arrays that contain all queried elements
$size	Match arrays with a specified number of elements
$regex	Search documents with a regular expression

Table 11.1: MongoDB Query Operators

```
# Query for everyone that is either above the age of 23 or a soldier.
>>> results = col.find({'$or':[{'age':{'$gt': 23}},{'soldier': True}]})

# Query for everyone that is a student (those that have a 'student' attribute
# and haven't been expelled).
>>> results = col.find({'student': {'$not': {'$in': [False, 'Expelled']}}})

# Query for everyone that has a student attribute.
>>> results = col.find({'student': {'$exists': True}})

# Query for people whose name contains a the letter 'e'.
>>> import re
>>> results = col.find({'name': {'$regex': re.compile('e')}})
```

It is likely that a database will hold more complex JSON entries then these, with many nested attributes and arrays. For example, an entry in a database for a school might look like this.

```
{'name': 'Jason', 'age': 16,
 'student': {'year':'senior', 'grades': ['A','C','A','B'],'flunking': False},
 'jobs':['waiter', 'custodian']}
```

To query the nested attributes and arrays use a dot, as in the following examples.

```
# Query for student that are seniors
>>> results = col.find({'student.year': 'senior'})

# Query for students that have an A in their first class.
>>> results = col.find({'student.grades.0': 'A'})
```

The Twitter JSON files are large and complex. To see what they look like, either look at the JSON file used to populate the `collection` or print any tweet from the database. The following website also contains useful information about the fields in the JSON file `https://dev.twitter.com/overview/api/tweets`.

The `distinct` function is also useful in seeing what the possible values are for a given field.

```
# Find all the values in the names field.
>>> col.distinct("name")
['Jack', 'Jill', 'John', 'Jeremy']
```

Problem 2. Query the Twitter collection from Problem 1 for the following information.

- How many tweets include the word Russia? Use `re.IGNORECASE`.
- How many tweets came from one of the main continental US time zones? These are listed as `"Central Time (US & Canada)"`, `"Pacific Time (US & Canada)"`, `"Eastern Time (US & Canada)"`, and `"Mountain Time (US & Canada)"`.
- How often did each language occur? Construct a dictionary with each language and it's frequency count.
  (Hint: use `distinct()` to get the language options.)

Deleting and Sorting Documents

Items can be deleted from a database using the same syntax that is used to find them. Use `delete_one` to delete just the first item that matches your search, or `delete_many` to delete all items that match your search.

```
# Delete the first person from the database whose name is Jack.
>>> col.delete_one({'name':'Jack'})

# Delete everyone from the database whose name is Jack.
>>> col.delete_many({'name':'Jack'})

# Clear the entire collection.
>>> col.delete_many({})
```

Another useful function is the `sort` function, which can sort the data by some attribute. It takes in the attribute by which the data will be sorted, and then the direction (1 for ascending and -1 for descending). Ascending is the default. The following code is an example of sorting.

```
# Sort the students by name in alphabetic order.
>>> results = col.find().sort('name', 1)
>>> for person in results:
...     print(person['name'])
...
Jack
Jack
Jeremy
Jill
John

# Sort the students oldest to youngest, ignoring those whose age is not listed.
>>> results = col.find({'age': {'$exists': True}}).sort('age', -1)
>>> for person in results:
...    print(person['name'])
...
Jill
Jack
Jack
```

Problem 3. Query the Twitter collection from Problem 1 for the following information.

- What are the usernames of the 5 most popular (defined as having the most followers) tweeters? Don't include repeats.
- Of the tweets containing at least 5 hashtags, sort the tweets by how early the 5th hashtag appears in the text. What is the earliest spot (character count) it appears?
- What are the coordinates of the tweet that came from the northernmost location? Use the latitude and longitude point in `"coordinates"`.

Updating Documents

Another useful attribute of MongoDB is that data in the database can be updated. It is possible to change values in existing fields, rename fields, delete fields, or create new fields with new values. This gives much more flexibility than a relational database, in which the structure of the databse must stay the same. To update a database, use either `update_one` or `update_many`, depending on whether one or more documents should be changed (the same as with `delete`). Both of these take two parameters; a find query, which finds documents to change, and the update parameters, telling these things what to update. The syntax is `update_many({find query},{update parameters})`.

The update parameters must contain update operators. Each update operator is followed by the field it is changing and the value to change it. The syntax is the same as with query operators. The operators are shown in the table below.

Operator	Description
`$inc` , `$mul`	`+=`, `*=`
`$min`, `$max`	`min()`, `max()`
`$rename`	Rename a specified field to the given new name
`$set`	Assign a value to a specified field (creating the field if necessary)
`$unset`	Remove a specified field
`$currentDate`	Set the value of the field to the current date. With `"$type": "date"`, use a `datetime` format; with `"$type": "timestamp:`, use a timestamp.

Table 11.2: MongoDB Update Operators

```
# Update the first person from the database whose name is Jack to include a
# new field 'lastModified' containing the current date.
>>> col.update_one({'name':'Jack'},
...                {'$currentDate': {'lastModified': {'$type': 'date'}}})

# Increment everyones age by 1, if they already have an age field.
>>> col.update_many({'age': {'$exists': True}}, {'$inc': {'age': 1}})

# Give the first John a new field 'best_friend' that is set to True.
>>> col.update_one({'name':'John'}, {'$set': {'best_friend': True}})
```

Problem 4. Clean the twitter collection in the following ways.

- Get rid of the `"retweeted_status"` field in each tweet.
- Update every tweet from someone with at least 1000 followers to include a `popular` field whose value is `True`. Report the number of popular tweets.
- (OPTIONAL) The geographical coordinates used before in `coordinates.coordinates` are turned off for most tweets. But many more have a bounding box around the coordinates in the `place` field. Update every tweet without coordinates that contains a bounding box so that the coordinates contains the average value of the points that form the bounding box. Make the structure of `coordinates` the same as the others, so it contains `coordinates` with a longitude, latitude array and a `type`, the value of which should be 'Point'.
  (Hint: Iterate through each tweet in with a bounding box but no coordinates. Then for each tweet, grab it's id and the bounding box coordinates. Find the average, and then update the tweet. To update it search for it's id and then give the needed update parameters. First unset coordinates, and then set coordinates.coordinates and coordinates.type to the needed values.)

Additional Material

Installation of MongoDB

MongoDB runs as an isolated program with a path directed to its database storage. To run a practice MongoDB server on your machine, complete the following steps:

Create Database Directory

To begin, navigate to an appropriate directory on your machine and create a folder called `data`. Within that folder, create another folder called `db`. Make sure that you have read, write, and execute permissions for both folders.

Retrieve Shell Files

To run a server on your machine, you will need the proper executable files from MongoDB. The following instructions are individualized by operating system. For all of them, download your binary files from `https://www.mongodb.com/download-center?jmp=nav#community`.

1. For Linux/Mac:

   Extract the necessary files from the downloaded package. In the terminal, navigate into the `bin` directory of the extracted folder. You may then start a Mongo server by running in a terminal: `./mongod --dbpath /pathtoyourdatafolder`.

2. For Windows:

   Go into your Downloads folder and run the Mongo `.msi` file. Follow the installation instructions. You may install the program at any location on your machine, but do not forget where you have installed it. You may then start a Mongo server by running in command prompt: C:\locationofmongoprogram\mongod.exe –dbpath C:\pathtodatafolder\data\db.

MongoDB servers are set by default to run at address:port `127.0.0.1:27107` on your machine.

You can also run Mongo commands through a mongo terminal shell. More information on this can be found at `https://docs.mongodb.com/getting-started/shell/introduction/`.

Twitter API

Pulling information from the Twitter API is simple. First you must get a Twitter account and register your app with them on `apps.twitter.com`. This will enable you to have a consumer key, consumer secret, access token, and access secret, all required by the Twitter API.

You will also need to install `tweepy`, an open source library that allows python to easily work with the Twitter API. This can be installed with pip by running from the command line

```
$pip install tweepy
```

The data for this lab was then pulled using the following code on May 26, 2017.

```
import tweepy
from tweepy import OAuthHandler
from tweepy import Stream
```

```
from tweepy.streaming import StreamListener
from pymongo import MongoClient
import json

#Set up the databse
client = MongoClient()
mydb = client.db1
twitter = mydb.collection1

f = open('trump.txt','w') #If you want to write to a file

consumer_key = #Your Consumer Key
consumer_secret = #Your Consumer Secret
access_token = #Your Access Token
access_secret = #Your Access Secret

my_auth = OAuthHandler(consumer_key, consumer_secret)
my_auth.set_access_token(access_token, access_secret)

class StreamListener(tweepy.StreamListener):
    def on_status(self, status):
        print(status.text)

    def on_data(self, data):
        try:
            twitter.insert_one(json.loads(data)) #Puts the data into your ↩
                MongoDB
            f.write(str(data)) #Writes the data to an output file
            return True
        except BaseException as e:
            print(str(e))
            print("Error")
        return True

    def on_error(self, status):
        print(status)
        if status_code == 420: #This means twitter has blocked us temporarily, ↩
            so we want to stop or they will get mad. Wait 30 minutes or so and ↩
            try again. Running this code often in a short period of time will ↩
            cause twitter to block you. But you can stream tweets for as long ↩
            as you want without any problems.
            return False
        else:
            return True

stream_listener = StreamListener()
stream = tweepy.Stream(auth=my_auth, listener=stream_listener)
stream.filter(track=["trump"]) #This pulls all tweets that include the keyword ↩
    "trump". Any number of keywords can be searched for.
```