Connecting to MongoDB using PyMongo

Student Name: Dipak Bange

To connect to MongoDB and work with data using Python, we will be installing Pymongo driver.

The easiest way to install the driver is through the pip package management system. Execute the following on a command line or Anaconda Prompt :

Command line:
python -m pip install pymongo

Anaconda Prompt:

pip install pymongo

pip3 install 'pymongo[srv]'

Once PyMongo is installed we can write our first application that will return information about the MongoDB server. In your Python development environment or from a text editor enter the following code.

```
import warnings
warnings.filterwarnings('ignore')
```

```
import pymongo
from pprint import pprint
# connect to MongoDB, change the << MONGODB URL >> to reflect your own connection string
client = pymongo.MongoClient("mongodb+srv://username:<pass>@cluster0.653psma.mongodb.net/?retryWrites=true&w=majority")
```

Exploring Collections and Documents

A collection in MongoDB is a container for documents. A database is the container for collections. Some of the advantages of storing data in documents are dynamic & flexible schema and the ability to store arrays can be seen from our simple Python scripts.

Connecting to a specific Database in our cluster.

```
# alternative
#db = client.test
#collection = db.video # or db['video']
```

```
#connecting to database "video"
db = client.video  # or client['video']
# Issue the serverStatus command and print the results
serverStatusResult=db.command("serverStatus")
pprint(serverStatusResult)
```

```
      'process': 'mongod',
      'repl': {'electionId': ObjectId('7fffffff0000000000000094'),
               'hosts': ['ac-isdtpns-shard-00-00.653psma.mongodb.net:27017',
                         'ac-isdtpns-shard-00-01.653psma.mongodb.net:27017',
                         'ac-isdtpns-shard-00-02.653psma.mongodb.net:27017'],
               'isWritablePrimary': True,
               'lastWrite': {'lastWriteDate': datetime.datetime(2023, 3, 3, 2, 28, 11),
                             'majorityOpTime': {'t': 148,
                                                'ts': Timestamp(1677810491, 10)},
                             'majorityWriteDate': datetime.datetime(2023, 3, 3, 2, 28, 11),
                             'opTime': {'t': 148, 'ts': Timestamp(1677810491, 10)}},
               'me': 'ac-isdtpns-shard-00-01.653psma.mongodb.net:27017',
               'primary': 'ac-isdtpns-shard-00-01.653psma.mongodb.net:27017',
               'primaryOnlyServices': {'TenantMigrationDonorService': {'numInstances': 0,
                                                                       'state': 'running'},
                                       'TenantMigrationRecipientService': {'numInstances': 0,
                                                                           'state': 'running'}},
               'rbid': 3,
               'secondary': False,
               'setName': 'atlas-ar4ty8-shard-0',
               'setVersion': 10,
               'tags': {'nodeType': 'ELECTABLE',
                        'provider': 'AWS',
                        'region': 'US_EAST_1',
                        'workloadType': 'OPERATIONAL'},
               'topologyVersion': {'counter': 6,
                                   'processId': ObjectId('64011098c33a7edef8163748')}},
      'storageEngine': {'backupCursorOpen': False,
                        'dropPendingIdents': 0,
                        'name': 'wiredTiger',
                        'oldestRequiredTimestampForCrashRecovery': Timestamp(1677810450, 52),
                        'persistent': True,
                        'readOnly': False,
                        'supportsCommittedReads': True,
                        'supportsPendingDrops': True,
                        'supportsResumableIndexBuilds': True,
                        'supportsSnapshotReadConcern': True},
      'uptime': 19107.0,
```

Exploring different Collection in Database

With collection_names(), we get list available collections in the database.

```
with client:

   db = client.video
   print(db.list_collection_names())
```

```
['movies']
```

Connecting to a specific Database in our cluster.

We need to run this step again as pymongo auto-closes the connection after previous step

```
# alternative
#db = client.test
#collection = db.video # or db['video']
```

```
#connecting to database "video"
#insert your mongodb connection string here
client = pymongo.MongoClient("mongodb+srv://username:<pass>@cluster0.653psma.mongodb.net/?retryWrites=true&w=majority")

db = client.video  # or client['video']
# Issue the serverStatus command and print the results
serverStatusResult=db.command("serverStatus")
pprint(serverStatusResult)
```

```
{'$clusterTime': {'clusterTime': Timestamp(1677810522, 24),
                  'signature': {'hash': b'\xe4\x8f\xc3j\x909(v\xceH\xec%'
                                        b'\xa8N\x9c\xa8\xe8S\xf8\xe2',
                                'keyId': 7164370397394108457}},
 'asserts': {'msg': 0, 'regular': 0, 'rollovers': 0, 'user': 0, 'warning': 0},
 'atlasVersion': {'gitVersion': '4591fd75e0047e1fc10dc2b17529b650b2798af7',
                  'version': '20230215.0.0.1676489897'},
 'connections': {'available': 495, 'current': 5, 'totalCreated': 66},
 'extra_info': {'note': 'fields vary by platform', 'page_faults': 0},
 'host': 'ac-isdtpns-shard-00-01.653psma.mongodb.net:27017',
 'localTime': datetime.datetime(2023, 3, 3, 2, 28, 42, 339000),
 'mem': {'bits': 64,
         'mapped': 0,
         'mappedWithJournal': 0,
         'resident': 0,
         'supported': True,
         'virtual': 0},
 'metrics': {'aggStageCounters': {'search': 0,
                                  'searchBeta': 0,
                                  'searchMeta': 0},
             'atlas': {'connectionPool': {'totalCreated': 7590}},
             'operatorCounters': {'match': {'regex': 0, 'text': 0}}},
 'network': {'bytesIn': 319311071, 'bytesOut': 1760036, 'numRequests': 1295},
 'ok': 1.0,
 'opLatencies': {'commands': {'latency': 7360983469, 'ops': 831},
                 'reads': {'latency': 6007626, 'ops': 8},
                 'writes': {'latency': 901363687, 'ops': 439}},
 'opcounters': {'command': 843,
                'delete': 0,
                'deprecated': {'getmore': 0, 'query': 0},
                'getmore': 0,
                'insert': 424065,
                'query': 4,
                'update': 0},
 'opcountersRepl': {'command': 0,
                    'delete': 0,
                    'deprecated': {'getmore': 0, 'query': 0},
                    'getmore': 0,
                    'insert': 0,
                    'query': 0,
                    'update': 0},
 'operationTime': Timestamp(1677810522, 24),
 'pid': 1929,
 'process': 'mongod',
 'repl': {'electionId': ObjectId('7fffffff0000000000000094'),
          'hosts': ['ac-isdtpns-shard-00-00.653psma.mongodb.net:27017',
                    'ac-isdtpns-shard-00-01.653psma.mongodb.net:27017',
                    'ac-isdtpns-shard-00-02.653psma.mongodb.net:27017'],
          'isWritablePrimary': True,
          'lastWrite': {'lastWriteDate': datetime.datetime(2023, 3, 3, 2, 28, 42),
                        'majorityOpTime': {'t': 148,
                                           'ts': Timestamp(1677810522, 23)},
                        'majorityWriteDate': datetime.datetime(2023, 3, 3, 2, 28, 42),
                        'opTime': {'t': 148, 'ts': Timestamp(1677810522, 24)}},
          'me': 'ac-isdtpns-shard-00-01.653psma.mongodb.net:27017',
          'primary': 'ac-isdtpns-shard-00-01.653psma.mongodb.net:27017',
          'primaryOnlyServices': {'TenantMigrationDonorService': {'numInstances': 0,
                                                                  'state': 'running'},
```

Reading Data in Sorted Order

```python
from pymongo import  DESCENDING
for x in db.myMovies.find().sort("year", DESCENDING):
    print(x)
```

Creating Sample Data in Movies Collection

```python
#Step 1: Create sample data
_id = ['tt0084740','tt0084741','tt0084742','tt0084743','tt0084744','tt0084745','tt0084746','tt0084747','tt0084748','tt0084749',
      'tt0084750','tt0084751']
title = ['Avengers','Batman Begins','Dark Knight', 'Dark Knight Rises',
         'Wonder Women','Iron Man','Ant Man', 'Thor','Dr Strange', 'Captain America','X- Men:First Class','Superman', 'Hulk']
year = [2012,2012,2013,2014,2018,2012,2018,2013,2013,2012,2013,2018] #Dont go on real release dates :P

for x in range(0, 12):
    movie = {
        'id': _id[x],
        'title' : title[x],
        'year' : year[x],
        'type' : 'movies'
    }
    #Step 2: Insert movies object directly into MongoDB via isnert_one
    result=db.myMovies.insert_one(movie)
    #Step 3: Print to the console the ObjectID of the new document
    print('Created {0} of 12 as {1}'.format(x,result.inserted_id))
#Step 4: Tell us that you are done
print('finished creating 12 movies')
```

```
Created 0 of 12 as 64015b65854bb6455904b86f
Created 1 of 12 as 64015b65854bb6455904b870
Created 2 of 12 as 64015b65854bb6455904b871
Created 3 of 12 as 64015b65854bb6455904b872
Created 4 of 12 as 64015b65854bb6455904b873
Created 5 of 12 as 64015b65854bb6455904b874
Created 6 of 12 as 64015b65854bb6455904b875
Created 7 of 12 as 64015b65854bb6455904b876
Created 8 of 12 as 64015b65854bb6455904b877
Created 9 of 12 as 64015b65854bb6455904b878
Created 10 of 12 as 64015b65854bb6455904b879
Created 11 of 12 as 64015b65854bb6455904b87a
finished creating 12 movies
```

Accessing collection myMovies

In MongoDB the find_one command is used to query for a single document much like select statements are used in relational databases. To use the find_one command in PyMongo we pass a Python dictionary that specifies the search criteria.

```
result = db.myMovies.find_one({'year': 2013}) #please feel free to try different year incase you are getting a 'None'
print(result)
```

```
{'_id': ObjectId('64015b65854bb6455904b871'), 'id': 'tt0084742', 'title': 'Dark Knight', 'year': 2013, 'type': 'movies'}
```

The function "find” will return all documents that match the search criteria. These cursors also support methods like count() which returns the number of results in the query.

```
movies_2013 = db.myMovies.count_documents({'year': 2013})
print(movies_2013)
```

```
4
```

Consider the scenario where you want to sum the occurrence of each year across the entire data set. You could simply issue a single query using the MongoDB aggregation.

```
stargroup=db.myMovies.aggregate(
# The Aggregation Pipeline is defined as an array of different operations
[
# The first stage in this pipe is to group data
{ '$group':
    { '_id': "$year",
     "count" :
                 { '$sum' :1 }
    }
},
# The second stage in this pipe is to sort the data
{"$sort":  { "_id":1}
}
# Close the array with the ] tag
] )
# Print the result
for group in stargroup:
    print(group)
```

```
{'_id': 2012, 'count': 4}
{'_id': 2013, 'count': 4}
{'_id': 2014, 'count': 1}
{'_id': 2018, 'count': 3}
```

Update data

There exists functions to help you update your MongoDB data including update_one, update_many and replace_one. The update_one method will update a single document based on a query that matches a document.

Following code updates a document with this new “star_rating” field.

```

SampleRecord = db.myMovies.find_one({'year': 2013})
print('A sample document:')
pprint(SampleRecord)

result = db.myMovies.update_one({'_id' : SampleRecord.get('_id') }, {'$inc': {'star_rating': 5}})
print('Number of documents modified : ' + str(result.modified_count))

UpdatedDocument = db.myMovies.find_one({'_id':SampleRecord.get('_id')})
print('The updated document:')
pprint(UpdatedDocument)
```

```
A sample document:
{'_id': ObjectId('64015b65854bb6455904b871'),
 'id': 'tt0084742',
 'title': 'Dark Knight',
 'type': 'movies',
 'year': 2013}
Number of documents modified : 1
The updated document:
{'_id': ObjectId('64015b65854bb6455904b871'),
 'id': 'tt0084742',
 'star_rating': 5,
 'title': 'Dark Knight',
 'type': 'movies',
 'year': 2013}
```

Notice that the original document did not have the “star_rating” field and an update allowed us to easily add the field to the document. This ability to dynamically add keys without the hassle of costly Alter_Table statements is the power of MongoDB’s flexible data model. It makes rapid application development a reality.

If you wanted to update all the fields of the document and keep the same ObjectID you will want to use the replace_one function.

Deleting documents

Functions such as delete_one and delete_many take a query that matches the document to delete as the first parameter.

```
#check data you want to delete

result = db.myMovies.find_one({'year': 2018})
print(result)
```

```
{'_id': ObjectId('64015b65854bb6455904b873'), 'id': 'tt0084744', 'title': 'Wonder Women', 'year': 2018, 'type': 'movies'}
```

```
# Delete the records
result = db.myMovies.delete_many({"year": 2018})
```

```
#check if delete or not

result = db.myMovies.find_one({'year': 2018})
print(result)
```

```
None
```

If you are deleting a large number of documents it may be more efficient to drop the collection instead of deleting all the documents.

```
client.close()
```

✓ 0s completed at 9:29 PM