MongoDB

MongoDB is a NoSQL data store that uses the document model. An instance of MongoDB runs multiple databases and each database can contain multiple collections. A collection is the equivalent of a relation (or more familiarly "table") in the relational model.

In [1]:

```
import pymongo
```

In [2]:

```
from pymongo import MongoClient
client = MongoClient()
```

We will connect to the database called `stud3` (you can pick your favorite name here)

In [3]:

```
db = client['stud3']
```

In [4]:

```
pymongo.version
```

Out[4]:

```
'3.7.2'
```

Now we check what collections are currently available in the database:

In [11]:

```
print db.list_collection_names()
```

```
[u'books']
```

In [9]:

```
collection = db['restaurants']
collection = db.restaurants
```

In [12]:

```
#clean up the collection before starting the exercise
db.restaurants.drop()
db.restaurants.drop_index("grades.score")
```

Manipulating data in collections

A collection contains documents. A document is similar in concept to a python dictionary: it contains keys (or attributes) that are associated with a value. The value of an attribute can be a simple string, or an object (like a date), or even an array.

In [13]:

```
import datetime
res = {
      "address" : {
         "street" : "2 Avenue",
         "zipcode" : "10075",
         "building" : "1480",
         "coord" : [ -73.9557413, 40.7720266 ]
      },
      "borough" : "Manhattan",
      "cuisine" : "Indian",
      "grades" : [
         {
            "date" : datetime.datetime.strptime("2014-10-01T00:00:00Z", "%Y-%m-%dT%H:%
M:%SZ"),
            "grade" : "A",
            "score" : 11
         },
         {
            "date" : datetime.datetime.strptime("2014-01-16T00:00:00Z", "%Y-%m-%dT%H:%
M:%SZ"),
            "grade" : "B",
            "score" : 17
         }
      ],
      "name" : "Chakra",
      "restaurant_id" : "41704620"
   }
```

Insert

We can now insert the document `res` to the collection `restaurants`:

In [14]:

```
restaurants = db.restaurants
# equivalent to:
# restaurants = db['restaurants']
print restaurants.insert_one(res)
```

```
<pymongo.results.InsertOneResult object at 0x7f78ec123b00>
```

We can also query the collection to retrieve a document. We use the function `find_one()` to retrieve a single document that satisfied the condition we pass as argument. In our case the condition is that the attribute `name` of the document should be equal to "Chakra".

Query

In [15]:

```
print restaurants.find_one({"name": "Chakra"})
```

```
{u'cuisine': u'Indian', u'borough': u'Manhattan', u'name': u'Chakra', u're
staurant_id': u'41704620', u'grades': [{u'date': datetime.datetime(2014, 1
0, 1, 0, 0), u'grade': u'A', u'score': 11}, {u'date': datetime.datetime(20
14, 1, 16, 0, 0), u'grade': u'B', u'score': 17}], u'address': {u'buildin
g': u'1480', u'street': u'2 Avenue', u'zipcode': u'10075', u'coord': [-73.
9557413, 40.7720266]}, u'_id': ObjectId('5eae9e438ec1436b9f534052')}
```

In [16]:

```
print restaurants.find_one({"grades.score" : 11})
```

```
{u'cuisine': u'Indian', u'borough': u'Manhattan', u'name': u'Chakra', u're
staurant_id': u'41704620', u'grades': [{u'date': datetime.datetime(2014, 1
0, 1, 0, 0), u'grade': u'A', u'score': 11}, {u'date': datetime.datetime(20
14, 1, 16, 0, 0), u'grade': u'B', u'score': 17}], u'address': {u'buildin
g': u'1480', u'street': u'2 Avenue', u'zipcode': u'10075', u'coord': [-73.
9557413, 40.7720266]}, u'_id': ObjectId('5eae9e438ec1436b9f534052')}
```

In [17]:

```
print restaurants.find_one({"name": "Zorba"})
```

```
None
```

We can instead use the function `find()` to retrieve all the documents that satisfy the condition. Then we are returned a list of documents and we need to iterate through the list if we want to print them.

In [23]:

```
for res in restaurants.find():
    print res
```

```
{u'cuisine': u'Indian', u'borough': u'Manhattan', u'name': u'Chakra', u're
staurant_id': u'41704620', u'grades': [{u'date': datetime.datetime(2014, 1
0, 1, 0, 0), u'grade': u'A', u'score': 11}, {u'date': datetime.datetime(20
14, 1, 16, 0, 0), u'grade': u'B', u'score': 17}], u'address': {u'buildin
g': u'1480', u'street': u'2 Avenue', u'zipcode': u'10075', u'coord': [-73.
9557413, 40.7720266]}, u'_id': ObjectId('5eae9e438ec1436b9f534052')}
```

We can manipulate the result of a `find` as if it was a python dictionary, so we can access single attributes, like 'name'.

In [24]:

```
[r['name'] for r in restaurants.find( { "grades.grade": "A" } )]
```

Out[24]:

```
[u'Chakra']
```

In [26]:

```
results = db.restaurants.find( { "grades.score": { "$gte": 15 } } )
print [r['name'] for r in results]
```

```
[u'Chakra']
```

For more operators: https://docs.mongodb.com/manual/reference/operator/query/ (https://docs.mongodb.com/manual/reference/operator/query/)

Update

Previously, we saw how to insert a new document in a collection. Now we want to update documents that are already in the collection. We can use the function `update_many()` to update all the documents of a collection that satisfy a condition (in our example, all the documents that have name Chakra).

Note that it is important to include in the update the `$set` attribute, to indicate the attribute that we want to update and what is the value that we want to assign to that attribute.

Another attribute that we might want to check is the `$currentDate` one. In this example we use it to assign the current date to the attribute `lastModified`.

In [27]:

```
r = restaurants.find({"name": "Chakra"})
print r[0]['cuisine']
```

```
Indian
```

In [28]:

```
result = restaurants.update_many(
    {"name": "Chakra"},
    {
        "$set": {
            "cuisine": "Indian (hot)"
        },
        "$currentDate": {"lastModified": True}
    }
)
```

In [30]:

```
r = restaurants.find({"name": "Chakra"})
print r[0]['cuisine']
```

```
Indian (hot)
```

In [31]:

```
r[0]['lastModified']
```

Out[31]:

```
datetime.datetime(2020, 5, 3, 10, 51, 10, 179000)
```

We can check how many documents have been affected by our update:

In [32]:

```
print 'n. matched documents:', result.matched_count
print 'n. modified documents:', result.modified_count
```

```
n. matched documents: 1
n. modified documents: 1
```

Now we will import data from an external json file. We do it through a mongo utility called `mongoimport`. This utility is ran in the command line (terminal in unix, command prompts in Windows). But thanks to ipython's magic, we can execute it from the notebook.

We call the bash magic to execute command line commands. The `mongoimport` utility wants as arguments the database to connect to (in our case, `lab1`), the collection where to import the data to (in our case, `restaurants`), and the file from which read the data.

In [33]:

```
%%bash
mongoimport --db stud3 --collection restaurants --file /home/stud3/test/lab1/primer-dat
aset.json
```

```
2020-05-03T13:54:55.118+0300    connected to: localhost
2020-05-03T13:54:56.162+0300    imported 25359 documents
```

Now that we have a lot of documents in our collection, we can do some more interesting query. For example, we can retrieve the names of all the restaurants in Manhattan. In this example, we show only 5 of them, because they are just too many...

In [34]:

```
manhattan_res = [r['name'] for r in restaurants.find( { "borough": "Manhattan" } )]
print 'n. restaurants in Manhattan:', len(manhattan_res)
print 'examples (5):\n', '\n'.join([str(r) for r in manhattan_res[:5]])
```

```
n. restaurants in Manhattan: 10260
examples (5):
Chakra
Glorious Food
P & S Deli Grocery
Dj Reynolds Pub And Restaurant
Cafe Metro
```

Exercise

You want to create a catalogue of books for your personal library. You know that for sure you will need to search books by title, but also by author(s) and keywords. Implement in MongoDB the catalogue, insert the following books, and provide the answer to following queries.

Books:

- "The world according to Garp" by John Irving (1978) - Novel, love, feminism
- "A tale of love and darkness" by Amos Oz (2002) - Autobiography, Jerusalem, Israel
- "If on a winter's night a traveler" by Italo Calvino (1979) - Novel, writing, postmodernism

Queries

- books written before 2000
- books that are novels
- books by author with first name 'Amos'

In [41]:

```
#Write your answer here:)
#Insert 3 documnets
to_add = []
res = {
    "title": "The world according to Garp",
    "author_name": "John",
    "author_last_name": "Irving",
    "year": 1978,
    "keywords": ['novel','love','feminism']

}

to_add.append(res)

res = {
    "title": "A tale of love and darknes",
    "author_name": "Amos",
    "author_last_name": "Oz",
    "year": 2002,
    "keywords": ['autobiography','jerusalem','israel']

}

to_add.append(res)

res = {
    "title": "If on a winter's night a traveler",
    "author_name": "Italo",
    "author_last_name": "Calvino",
    "year": 1979,
    "keywords": ['novel','writing','postmodernism']

}

to_add.append(res)
```

In [42]:

```
db.books.insert_many(to_add)
```

Out[42]:

```
<pymongo.results.InsertManyResult at 0x7f78ccc07e18>
```

In [43]:

```
#Query 1: books written before 2000
books = db.books
list(books.find({"year": {"$lt": 2000}}))
```

Out[43]:

```
[{u'_id': ObjectId('5eaea5218ec1436b9f534056'),
  u'author_last_name': u'Irving',
  u'author_name': u'John',
  u'keywords': [u'novel', u'love', u'feminism'],
  u'title': u'The world according to Garp',
  u'year': 1978},
 {u'_id': ObjectId('5eaea5218ec1436b9f534058'),
  u'author_last_name': u'Calvino',
  u'author_name': u'Italo',
  u'keywords': [u'novel', u'writing', u'postmodernism'],
  u'title': u"If on a winter's night a traveler",
  u'year': 1979}]
```

In [44]:

```
#Query 2: books that are novels
list(books.find({"keywords": {"$in":["novel"]}}))
```

Out[44]:

```
[{u'_id': ObjectId('5eaea5218ec1436b9f534056'),
  u'author_last_name': u'Irving',
  u'author_name': u'John',
  u'keywords': [u'novel', u'love', u'feminism'],
  u'title': u'The world according to Garp',
  u'year': 1978},
 {u'_id': ObjectId('5eaea5218ec1436b9f534058'),
  u'author_last_name': u'Calvino',
  u'author_name': u'Italo',
  u'keywords': [u'novel', u'writing', u'postmodernism'],
  u'title': u"If on a winter's night a traveler",
  u'year': 1979}]
```

In [45]:

```
#Query 3: books by author with first name 'Amos'
list(books.find({ "author_name":"Amos"  }))
```

Out[45]:

```
[{u'_id': ObjectId('5eaea5218ec1436b9f534057'),
  u'author_last_name': u'Oz',
  u'author_name': u'Amos',
  u'keywords': [u'autobiography', u'jerusalem', u'israel'],
  u'title': u'A tale of love and darknes',
  u'year': 2002}]
```

For more info: https://docs.mongodb.com/manual/reference/operator/query/ (https://docs.mongodb.com/manual/reference/operator/query/)

Aggregation on a collection and indexing

In this last part, we will look at how to execute an aggregation of the documents in a collection, and how to build indexes to speed up the computation (if the number of records and complexity of operation would allow us to appreciate the improvement).

We want to know how many restaurants are in each borough of New York. So we build an aggregation function that will divide the collection in groups according to the value of the attribute `borough` (`"_id":"$borough"`) and it will create an attribute `num_restaurants` for each group. The value of `num_restaurants` is given by the aggregation funtion `$sum` that will be applied on the value `1` for each document.

For SQL-like minds, this is equivalent to a group by clause. We are grouping on the attribute `borough` and using `count(*)` as aggregation function.

For a list of available aggregate functions (accumulators) see this link (https://docs.mongodb.org/manual/reference/operator/aggregation/#accumulators)

In [46]:

```
aggregation_function = [
    {"$group": {"_id": "$borough", "num_restaurants": {"$sum": 1}}}
    ]
agg_result = db.restaurants.aggregate(aggregation_function)
print '\n'.join(str(x) for x in list(agg_result))
```

```
{u'num_restaurants': 51, u'_id': u'Missing'}
{u'num_restaurants': 969, u'_id': u'Staten Island'}
{u'num_restaurants': 10260, u'_id': u'Manhattan'}
{u'num_restaurants': 6086, u'_id': u'Brooklyn'}
{u'num_restaurants': 5656, u'_id': u'Queens'}
{u'num_restaurants': 2338, u'_id': u'Bronx'}
```

In [47]:

```
#another way to display
import pandas as pd
aggregation_function = [
    {"$group": {"_id": "$borough", "num_restaurants": {"$sum": 1}}}
    ]
agg_result = db.restaurants.aggregate(aggregation_function)
df = pd.DataFrame(list(agg_result))
df
```

Out[47]:

	_id	num_restaurants
0	Missing	51
1	Staten Island	969
2	Manhattan	10260
3	Brooklyn	6086
4	Queens	5656
5	Bronx	2338

In [50]:

```
#pipeline
import pandas as pd
aggregation_function = [
    {"$match": {"name": "Chakra"}},
    {"$group": {"_id": "$name", "num_restaurants": { "$sum" :  1}}}
    ]
agg_result = db.restaurants.aggregate(aggregation_function)
df = pd.DataFrame(list(agg_result))
df.head()
```

Out[50]:

	_id	num_restaurants
0	Chakra	1

More info about stages: https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/ (https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/)

Indexing

Now we look at how to build an index on an attribute of documents, to improve performance of retrieval.

This is the query we want to perform: retrieve all the restaurants that got a score in at least one of their grades between 90 and 100 (both edges included).

In [51]:

```
results = db.restaurants.find( { "grades.score": { "$gte": 90, "$lte": 100 } } )
print '\n'.join([r['name'] for r in list(results)])
```

```
Murals On 54/Randolphs'S
Gandhi
Bella Napoli
Concrete Restaurant
Baluchi'S Indian Food
```

The query returns 5 restaurants among 25,000 or more.

Now we ask Mongo to tell us how he is executing the previous query. We can do this by using the function `explain`. For the SQL-inclined, this is equivalent to an explain plan.

In [52]:

```
execution_plan = db.restaurants.find( { "grades.score": { "$gte": 90, "$lte": 100 } } )
.explain()
print execution_plan
```

```
{u'executionStats': {u'executionTimeMillis': 91, u'nReturned': 5, u'totalK
eysExamined': 0, u'allPlansExecution': [], u'executionSuccess': True, u'ex
ecutionStages': {u'needYield': 0, u'direction': u'forward', u'saveState':
198, u'restoreState': 198, u'isEOF': 1, u'docsExamined': 25360, u'nReturne
d': 5, u'needTime': 25356, u'filter': {u'$and': [{u'grades.score': {u'$lt
e': 100}}, {u'grades.score': {u'$gte': 90}}]}, u'executionTimeMillisEstima
te': 81, u'invalidates': 0, u'works': 25362, u'advanced': 5, u'stage': u'C
OLLSCAN'}, u'totalDocsExamined': 25360}, u'queryPlanner': {u'parsedQuery':
{u'$and': [{u'grades.score': {u'$lte': 100}}, {u'grades.score': {u'$gte':
90}}]}, u'rejectedPlans': [], u'namespace': u'stud3.restaurants', u'winnin
gPlan': {u'filter': {u'$and': [{u'grades.score': {u'$lte': 100}}, {u'grade
s.score': {u'$gte': 90}}]}, u'direction': u'forward', u'stage': u'COLLSCA
N'}, u'indexFilterSet': False, u'plannerVersion': 1}, u'ok': 1.0, u'server
Info': {u'host': u'bdl1.eng.tau.ac.il', u'version': u'3.4.18', u'port': 27
017, u'gitVersion': u'4410706bef6463369ea2f42399e9843903b31923'}}
```

Let's look at the values of some of the attributes in the plan, like the number of documents examined (here more than 25,000).

In [53]:

```
print execution_plan['executionStats']['executionStages']
print 'totalDocsExamined:', execution_plan['executionStats']['totalDocsExamined']
print 'nReturned:', execution_plan['executionStats']['nReturned']
```

```
{u'needYield': 0, u'direction': u'forward', u'saveState': 198, u'restoreSt
ate': 198, u'isEOF': 1, u'docsExamined': 25360, u'nReturned': 5, u'needTim
e': 25356, u'filter': {u'$and': [{u'grades.score': {u'$lte': 100}}, {u'gra
des.score': {u'$gte': 90}}]}, u'executionTimeMillisEstimate': 81, u'invali
dates': 0, u'works': 25362, u'advanced': 5, u'stage': u'COLLSCAN'}
totalDocsExamined: 25360
nReturned: 5
```

Now we create and index on the attribute that we are using in our query.

In [54]:

```
restaurants.create_index([("grades.score", pymongo.ASCENDING)])   #mymongo.DESCENDING
```

Out[54]:

```
u'grades.score_1'
```

In [55]:

```
exec_plan_index = db.restaurants.find( { "grades.score": { "$gte": 90, "$lte": 100 } }
).explain()
```

In [57]:

```
print exec_plan_index['executionStats']['executionStages']
print 'totalDocsExamined:', exec_plan_index['executionStats']['totalDocsExamined']
print 'nReturned:',exec_plan_index['executionStats']['nReturned']
```

```
{u'needYield': 0, u'docsExamined': 5, u'saveState': 0, u'restoreState': 0,
u'isEOF': 1, u'inputStage': {u'saveState': 0, u'isEOF': 1, u'seenInvalidat
ed': 0, u'keysExamined': 5, u'nReturned': 5, u'invalidates': 0, u'keyPatte
rn': {u'grades.score': 1}, u'isUnique': False, u'needTime': 0, u'isMultiKe
y': True, u'executionTimeMillisEstimate': 0, u'dupsTested': 5, u'restoreSt
ate': 0, u'direction': u'forward', u'indexName': u'grades.score_1', u'isSp
arse': False, u'advanced': 5, u'stage': u'IXSCAN', u'dupsDropped': 0, u'mu
ltiKeyPaths': {u'grades.score': [u'grades']}, u'needYield': 0, u'isPartia
l': False, u'indexBounds': {u'grades.score': [u'[90, inf.0]']}, u'seeks':
1, u'works': 6, u'indexVersion': 2}, u'nReturned': 5, u'needTime': 0, u'fi
lter': {u'grades.score': {u'$lte': 100}}, u'executionTimeMillisEstimate':
0, u'alreadyHasObj': 0, u'invalidates': 0, u'works': 7, u'advanced': 5,
u'stage': u'FETCH'}
totalDocsExamined: 5
nReturned: 5
```

We can see that the number of documents examined went down to 5. Success! :)

Reference

https://docs.mongodb.com/manual/reference/operator
(https://docs.mongodb.com/manual/reference/operator)