Using MongoDB to Retrieve Information ¶

In this notebook, we will look at the `PyMongo` library, and perform some common tasks in MongoDB with it. We will be making use of data from food.gov.uk (http://ratings.food.gov.uk/), which gives information about the hygiene ratings of all food establishments in the country. The tool we use to access a MongoDB database is **'MongoClient'**, further information about this tool can be found at Mongo_Client (http://api.mongodb.com/python/current/api/pymongo/mongo_client.html).

The code required to connect to the database is as follows:

In [1]:

```
# You don't need to write anything here
from pymongo import MongoClient

client = MongoClient('mongodb://cpduser:M13pV5woDW@mongodb/health_data')
db = client.health_data
```

The MongoDB querying language is very similar to JavaScript, and in Python we make use of dictionaries to get the appropriate name/value pairs.

***WARNING!** Make sure you are careful when you run your queries. If you try and get all 500,000 records displaying on the page it will take a while and could well crash your browser!*

Just like using the native Mongo client, you can run functions or access a collection from the query object by using dot notation, so the `uk` collection would be `db.uk`. You can also use `db['uk']`, which can be more useful, e.g., if you are using variable names to access the different collections.

There is a function called `collection_names` which can be performed on the database. Run this function to see the names of the collections in the database.

In [2]:

```
# YOUR CODE HERE
db.collection_names()
#특정 데이터베이스에 있는 모든 컬렉션 네임을 반환합니다.
```

Out[2]:

```
['derbyshire_dales',
 'north_kesteven',
 'causeway_coast_and_glens',
 'slough',
 'east_renfrewshire',
 'taunton_deane',
 'warrington',
 'south_lanarkshire',
 'torfaen',
 'south_ribble',
 'wycombe',
 'forest_heath',
 'erewash',
 'nuneaton_and_bedworth',
 'rotherham',
 'south_norfolk',
 'wirral',
 'torbay',
```

Querying

Querying is done on collection objects. Start with using the `find_one` function on any collection to investigate the structure of the data.

In [3]:

```
# You don't need to write anything here
db['uk'].find_one()
#find_one은 특정한 인스턴스를 찾기위한 코드
#필드에 저장된 데이터의 예시를 볼 수 있다.
#NoSQL : 구조, 필드, 테이블이 필요없는 관계형 데이터 베이스이다.
```

Out[3]:

```
{'AddressLine2': '16a Adelphi Street',
 'AddressLine3': 'Preston',
 'BusinessName': '3 Monkeys Sandwich Bar',
 'BusinessType': 'Restaurant/Cafe/Canteen',
 'BusinessTypeID': '1',
 'ConfidenceInManagement': 10,
 'FHRSID': '90105',
 'Geocode': {'coordinates': [-2.706293, 53.763151], 'type': 'Point'},
 'Hygiene': 10,
 'Lat': 53.763151,
 'Lng': -2.706293,
 'LocalAuthorityBusinessID': '244',
 'LocalAuthorityCode': '202',
 'LocalAuthorityEmailAddress': 'info@preston.gov.uk',
 'LocalAuthorityName': 'Preston',
 'LocalAuthorityWebSite': 'http://www.preston.gov.uk',
 'NewRatingPending': 'False',
 'PostCode': 'PR1 7BE',
 'RatingDate': datetime.datetime(2015, 12, 8, 0, 0),
 'RatingKey': 'fhrs_3_en-GB',
 'RatingValue': 3,
 'Region': 'north_west',
 'SchemeType': 'FHRS',
 'Scores': {'ConfidenceInManagement': 10, 'Hygiene': 10, 'Structural':
10},
 'Structural': 10,
 '_id': ObjectId('5be545d6c4cc3a0001c61064')}
```

It can be useful to run the `find_one` function when you are trying a certain set of search conditions, to check that you are getting the results you expect. To add conditions to a query, the first parameter of the function is a dictionary in the format `{'field': 'value'}`. Search for the first document which has a `Region` value of `'london'`

In [9]:

```
# YOUR CODE HERE
print(db.uk.find_one({'Region':'london'}))
#괄호안에 사전을 넣어야함
```

```
{'NewRatingPending': 'False', 'RatingValue': 3, 'FHRSID': '847841', '_
id': ObjectId('5be54656c4cc3a0001cc564c'), 'SchemeType': 'FHRS', 'Loca
lAuthorityName': 'Bromley', 'Scores': {'Structural': 5, 'Hygiene': 10,
'ConfidenceInManagement': 10}, 'Region': 'london', 'LocalAuthorityWebS
ite': 'http://www.bromley.gov.uk', 'Lng': 0.028132, 'RatingKey': 'fhrs
_3_en-GB', 'PostCode': 'BR1 3BE', 'LocalAuthorityCode': '505', 'Busine
ssName': '118 Widmore Road', 'LocalAuthorityEmailAddress': 'food@broml
ey.gov.uk', 'Hygiene': 10, 'RatingDate': datetime.datetime(2016, 2, 1
5, 0, 0), 'LocalAuthorityBusinessID': '15/00272/MIXED', 'BusinessTyp
e': 'Hospitals/Childcare/Caring Premises', 'AddressLine1': 'Bromley',
'Lat': 51.406561, 'BusinessTypeID': '5', 'Structural': 5, 'Geocode':
{'coordinates': [0.028132, 51.406561], 'type': 'Point'}, 'ConfidenceIn
Management': 10}
```

In [16]:

```
# YOUR CODE HERE
print(db.southampton.find_one({'RatingValue':{'$lt':5}}))
#쏘튼에서 5점미만 평가를 받는 사업체의 첫번째 사례 출력
```

```
{'NewRatingPending': 'False', 'RatingValue': 5, 'FHRSID': '219539', '_
id': ObjectId('5be5463dc4cc3a0001cb266c'), 'SchemeType': 'FHRS', 'Loca
lAuthorityName': 'Southampton', 'Scores': {'Structural': 5, 'Hygiene':
5, 'ConfidenceInManagement': 5}, 'Region': 'south_east', 'LocalAuthori
tyWebSite': 'http://www.southampton.gov.uk', 'Structural': 5, 'Lng': -
1.403594, 'RatingKey': 'fhrs_5_en-GB', 'PostCode': 'SO15 2AH', 'LocalA
uthorityCode': '877', 'BusinessName': '88', 'LocalAuthorityEmailAddres
s': 'hygiene.rating@southampton.gov.uk', 'Hygiene': 5, 'RatingDate': d
atetime.datetime(2015, 1, 21, 0, 0), 'LocalAuthorityBusinessID': '1117
0/0044/0/000', 'BusinessType': 'Restaurant/Cafe/Canteen', 'AddressLine
1': '44 London Road', 'Lat': 50.911947, 'BusinessTypeID': '1', 'Addres
sLine2': 'Southampton', 'Geocode': {'coordinates': [-1.403594, 50.9119
47], 'type': 'Point'}, 'ConfidenceInManagement': 5}
```

Returning Part of a Document

By default, all values in a document will be returned from a query. This is not always the desired outcome, so it is possible to modify which parts of the document are returned. This is done by the optional second parameter to a `find` or `find_one` query as a dictionary in the format `{"keep_this_field": 1, "ignore_this_field": 0}`.

If this parameter exists, then any field name which is not specified will not be returned unless specifically requested. For example, consider the code below, which returns the name of the first business from Aberdeenshire:

In [17]:

```
db.aberdeenshire.find_one({}, {'BusinessName': 1})
#aberdeenshire가 컬렉션 네임, 검색(선택) 기준이 없는 경우에는 빈사전, 전달할 다음 매겨변수는 사업체 이름
```

Out[17]:

```
{'BusinessName': '2nd Dimensions', '_id': ObjectId('5be54625c4cc3a0001
c9f943')}
```

There are three things to notice about this query.

1. Firstly, the dictionary as the first parameter is empty, meaning that there are no criteria for the search result.
2. The `BusinessName` field is returned as expected
3. The `_id` field is also returned without our asking for it! This is an exception to the rule of requiring to request a field specifically. In order to avoid having this field (and you will need to do this for the visualisation exercise, because having it causes problems for the `Bokeh` library), you simply request that it is not there, as in the code below:

In [18]:

```
db.aberdeenshire.find_one({}, {'BusinessName': 1, '_id': 0})
```

Out[18]:

```
{'BusinessName': '2nd Dimensions'}
```

Test Yourself

Write a query to return the `BusinessType` of the first business in Swansea with a `RatingValue` of 1, excluding the `_id`

In [33]:

```
# YOUR CODE HERE


db.swansea.find_one({'RatingValue': 1}, {'BusinessType': 1, '_id': 0})
```

Out[33]:

```
{'BusinessType': 'Pub/bar/nightclub'}
```

Cursors

Whereas `find_one` returns a single record, the `find` method returns a `Cursor` (http://api.mongodb.com/python/current/api/pymongo/cursor.html) object. These can also have operations performed on them such as `count` to get the amount of records or [`distinct(distinct_field)` (https://docs.mongodb.com/manual/reference/method/db.collection.distinct/ (https://docs.mongodb.com/manual/reference/method/db.collection.distinct/)) to get unique records according to that particular field.

The useful part of a `Cursor`, however, is that it can be iterated over like a Python `list`. Each item in the cursor is an object from which fields can be accessed. For example, to get the `RatingValue` of each establishment in Swansea, the following code would be used:

In [37]:

```
# You don't need to write anything here
for c in db.swansea.find({'RatingValue': 5}):
    print(c['RatingValue'])
    # We don't want to print out all of them so break out of the loop now
    break
```

4

Write a query which gets each different type of business in the Southampton collection.

In [35]:

```
# YOUR CODE HERE
db.southampton.distinct('BusinessType')
#각기 다른 사우스햄튼 컬렉션에 포함되어있는 데이터 필드의 이름
```

Out[35]:

```
['Restaurant/Cafe/Canteen',
 'Retailers - other',
 'Hotel/bed & breakfast/guest house',
 'Hospitals/Childcare/Caring Premises',
 'Other catering premises',
 'Retailers - supermarkets/hypermarkets',
 'Mobile caterer',
 'Takeaway/sandwich shop',
 'Pub/bar/nightclub',
 'School/college/university',
 'Manufacturers/packers']
```

MongoDB Aggregation Framework

For performing SQL `GROUP BY` operations such as `MIN` or `MAX` on objects, the MongoDB Aggregation framework is what you'll need to use. It is more complicated than the simple `find` queries, as it has a pipeline (https://docs.mongodb.com/manual/core/aggregation-pipeline/) of different operations. For our purposes, the one we wish to concentrate on is the `$group` pipeline.

To use it, we call `db.collection.aggregate`, and pass a list as the first parameter. Within the list, there are a series of `dict` objects representing a stage in the pipeline as `{"$stage": {"key": "value} }"`.

For grouping then, we would have key `"$group" with a value of a dict.  In the dict, we have the pairs`"output_field": {"$operator": "field_name"}`

A simple example can be seen below, which gives the sum of each different business type in York. Note the following things about it:

- The list parameter, with the nested objects inside it.
- The `_id` of $BusinessType - this is the field we're grouping on. In this case, the `$` sign means that we are getting the value of the field.
- The output field `count` has the `"$sum"`, with each instance being given a weighting of 1. To double the value of this field, we could simply use `{"$sum": 2}` instead.

In [38]:

```
# You don't need to write anything here
coll = db.york.aggregate(
    [ #리스트 괄호 !
        {"$group": { "_id": "$BusinessType", "count": {"$sum": 1} } } #중첩 딕셔너리 $는
    ]
)

#432개는 레스토랑 카페 식당
for dot in coll:
    print(dot)
```

```
{'_id': 'Mobile caterer', 'count': 50}
{'_id': 'Retailers - supermarkets/hypermarkets', 'count': 53}
{'_id': 'Distributors/Transporters', 'count': 15}
{'_id': 'Importers/Exporters', 'count': 1}
{'_id': 'Takeaway/sandwich shop', 'count': 183}
{'_id': 'Farmers/growers', 'count': 10}
{'_id': 'Other catering premises', 'count': 273}
{'_id': 'Retailers - other', 'count': 340}
{'_id': 'School/college/university', 'count': 93}
{'_id': 'Hotel/bed & breakfast/guest house', 'count': 184}
{'_id': 'Manufacturers/packers', 'count': 25}
{'_id': 'Pub/bar/nightclub', 'count': 232}
{'_id': 'Hospitals/Childcare/Caring Premises', 'count': 144}
{'_id': 'Restaurant/Cafe/Canteen', 'count': 432}
```

Write a function which gives a count of the different `RatingValue` in `db.uk`.

In [ ]:

```
# YOUR CODE HERE
```