Week 7 – Mongo Demo

Python package pymongo

Now that we have installed the Mongo NoSQL database server and tried a few commands to insert data from the mongo command line client, we are mostly going to use MongoDB from Python. The recommended package for this is pymongo. Open a Python interpreter and try to import pymongo.

```
>>> import pymongo
```

Use Mongo from Python:

In this section, we show how to use the Mongo database directly in Python interpreter using pymongo. The commands are similar to those used in the Mongo command line interface. While the Mongo server is running, we can use pymongo to connect to the server.

```
# Connecting to the database
>>> from pymongo import MongoClient
>>> client = MongoClient('localhost', 27017)
# show existing databases
>>> client.database_names()
```

So far we just have a default local database. Now we'll show how to add data in pymongo and start by **creating a database**. If we just access a database name, it will create it if it does not already exist.

```
# Create a new database or use an existing database and show its collections
>>> db = client.peopledb
>>> db.collection_names()
[]
```

We can **create a collection** by just accessing it, and again it will be created if it does not already exist.

```
# Create a new collection or use an existing collection
>>> peoplecoll = db.people
>>> type(peoplecoll)
<class 'pymongo.collection.Collection'>
```

Note that collections don't show in the db.collection_names() list until they have data.

A collection consists of a number of documents, and each document is a dictionary corresponding to a JSON object. We add data to a collection by **inserting documents**.

Suppose that we have the people data that we used in the previous example, and suppose that we have a Python variable that has a list of the people dictionaries.

```
# list of dictionaries to be used as documents in the collection
>>> peoplelist = [{ "name": "John Smith", "age": 30 }, { "name": "Bo Bennett", "age": 23 }, { "name": "Anna Jones", "age": 25 }]
```

We can insert this data one at a time using the insert() function, or a list of data items using the insert_many() function. When documents are inserted, the default is that mongo generates a unique identifier that will function as a primary key for the collection. The insert_many() function is sometimes called bulk insertion and it only connects with the database server once.

```
#  insert one document
peoplecoll.insert({ "name": "John Smith", "age": 30 })
ObjectId('58cfeeb664a4f302cd1bcb1c')

# insert several documents
>>> peoplecoll.insert_many(peoplelist[1: ])
<pymongo.results.InsertManyResult object at 0x102ffe558>
```

We avoided inserting the first person twice. If we had inserted the same data again, pymongo would insert a duplicate element, but generate a different identifier.

To **show all data**, we can use the find() function to get all the documents in the collection. In Python, the result of the find() function is a cursor, which can be used as an iterator to go over every item.

```
>>> docs = peoplecoll.find()
>>> type(docs)
<class 'pymongo.cursor.Cursor'>
>>> for doc in docs:
...   print(doc)
...
{'age': 30, 'name': 'John Smith', '_id': ObjectId('58cfeece64a4f302cd1bcb1d')}
{'age': 23, 'name': 'Bo Bennett', '_id': ObjectId('58cfeece64a4f302cd1bcb1e')}
{'age': 25, 'name': 'Anna Jones', '_id':
ObjectId('58cfeece64a4f302cd1bcb1f')}
```

Now to illustrate that the documents do not have to conform to a fixed schema, let's suppose that we have some additional people dictionaries with an additional field. We can also insert them into the database. This time we use insert(), which inserts the items on the list one at a time.

```
>>> morepeoplelist = [{ "name": "Britney Sykes", "age": 21 ,
'position':'Guard'}, { "name": "Briana Day", "age": 21,
'position':'Center'}, { "name": "Alexis Peterson", "age": 21,
'position':'Guard' }, { "name": "Gabby Cooper", "age": 18,
'position':'Guard'}]
```

```
>>> peoplecoll.insert(morepeoplelist)
[ObjectId('58cff3e164a4f302cd1bcb20'),
ObjectId('58cff3e164a4f302cd1bcb21'),
ObjectId('58cff3e164a4f302cd1bcb22'),
ObjectId('58cff3e164a4f302cd1bcb23')]
```

Let’s fetch all the documents again:

```
>>> docs = peoplecoll.find()
>>> for doc in docs:
...   print(doc)
...
{'age': 30, 'name': 'John Smith', '_id': ObjectId('58cfeeb664a4f302cd1bcb1c')}
{'age': 30, 'name': 'John Smith', '_id': ObjectId('58cfeece64a4f302cd1bcb1d')}
{'age': 23, 'name': 'Bo Bennett', '_id': ObjectId('58cfeece64a4f302cd1bcb1e')}
{'age': 25, 'name': 'Anna Jones', '_id': ObjectId('58cfeece64a4f302cd1bcb1f')}
{'age': 21, 'name': 'Britney Sykes', '_id': ObjectId('58cff3e164a4f302cd1bcb20'),
'position': 'Guard'}
{'age': 21, 'name': 'Briana Day', '_id': ObjectId('58cff3e164a4f302cd1bcb21'),
'position': 'Center'}
{'age': 21, 'name': 'Alexis Peterson', '_id': ObjectId('58cff3e164a4f302cd1bcb22'),
```

'position': 'Guard'}

{'age': 18, 'name': 'Gabby Cooper', '_id': ObjectId('58cff3e164a4f302cd1bcb23'), 'position': 'Guard'}

Next, we may want to **query the collection** for a particular document or set of documents matching some criterion.

If we use the find_one() function with no arguments, it returns the first document.

```
>>> peoplecoll.find_one()
```

{'age': 30, 'name': 'John Smith', '_id': ObjectId('58cfeeb664a4f302cd1bcb1c')}

We can add query arguments to the find_one function to find a particular document. The arguments use a query language that, at its simplest, consists of field value(s), formatted into a dictionary. You can think of it as a partial description of a document.

find a document where the name field has value ‘Anna Jones’

```
>>> result = peoplecoll.find_one({'name':'Anna Jones'})
>>> print(result)
```

{'age': 25, 'name': 'Anna Jones', '_id': ObjectId('58cfeece64a4f302cd1bcb1f')}

If we want to find more than one document, we can use the function find() with an argument query, and all the documents matching that query will be returned as a cursor.

find all the documents where the position field has value ‘Guard’

>>> results = peoplecoll.find({'position':'Guard'})

>>> for result in results:

... print(result)

...

{'age': 21, 'name': 'Britney Sykes', '_id': ObjectId('58cff3e164a4f302cd1bcb20'), 'position': 'Guard'}

{'age': 21, 'name': 'Alexis Peterson', '_id': ObjectId('58cff3e164a4f302cd1bcb22'), 'position': 'Guard'}

{'age': 18, 'name': 'Gabby Cooper', '_id': ObjectId('58cff3e164a4f302cd1bcb23'), 'position': 'Guard'}