ASSIGNMENT NO: 7

AIM : Execute NoSQL queries on the sample collections using MongoDB and User Interface in Python.

INDEX TERMS: MongoDB, User Interface, Python, Database Connectivity,Pymongo.

THEORY

Python is a powerful programming language used for many different types of applications within the development community. Many know it as a flexible language that can handle just about any task.

MongoDB is a NoSQL database which has become pretty popular throughout the industry in recent years. NoSQL databases provide features of retrieval and storage of data in a much different way than their relational database counterparts.

PyMongo:

PyMongo is a Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python.

Installation

To build the optional C extensions on Linux or another non-macOS Unix you must have the GNU C compiler (gcc) installed. Depending on your flavor of Unix (or Linux distribution) you may also need a python development package that provides the necessary header files for your version of Python. Ubuntu users should issue the following command:

$ sudo apt-get install build-essential python-dev

Connection with MongoClient

The first step when working with PyMongo is to create a MongoClient to the running mongod instance.:

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

The above code will connect on the default host and port. We can also specify the host and port explicitly, as follows:

```
>>> client = MongoClient('localhost', 27017)
```

Or use the MongoDB URI format:

```
>>> client = MongoClient('mongodb://localhost:27017/')
```

Getting a Database

A single instance of MongoDB can support multiple independent databases. When working with PyMongo you access databases using attribute style access on MongoClient instances:

```
>>> db = client.test_database
```

If your database name is such that using attribute style access won't work (like test-database), you can use dictionary style access instead:

```
>>> db = client['test-database']
```

Getting a Collection

A collection is a group of documents stored in MongoDB, and can be thought of as roughly the equivalent of a table in a relational database. Getting a collection in PyMongo works the same as getting a database:

```
>>> collection = db.test_collection
```
or (using dictionary style access):

```
>>> collection = db['test-collection']
```

An important note about collections (and databases) in MongoDB is that they are created lazily - none of the above commands have actually performed any operations on the MongoDB server. Collections and databases are created when the first document is inserted into them.

Documents

Data in MongoDB is represented (and stored) using JSON-style documents. In PyMongo we use dictionaries to represent documents. As an example, the following dictionary might be used to represent a blog post:

```
>>> import datetime
>>> post = {"author": "Mike",
...         "text": "My first blog post!",
...         "tags": ["mongodb", "python", "pymongo"],
...         "date": datetime.datetime.utcnow()}
```

Note that documents can contain native Python types (like datetime.datetime instances) which will be automatically converted to and from the appropriate BSON types.

Inserting a Document

To insert a document into a collection we can use the insert_one() method:

```
>>> posts = db.posts
>>> post_id = posts.insert_one(post).inserted_id
>>> post_id
ObjectId('...')
```

When a document is inserted a special key, "_id", is automatically added if the document doesn't already contain an "_id" key. The value of "_id" must be unique across the collection. insert_one() returns an instance of InsertOneResult. For more information on "_id", see the documentation on _id.

After inserting the first document, the posts collection has actually been created on the server. We can verify this by listing all of the collections in our database:

```
>>> db.collection_names(include_system_collections=False)
```

Getting a Single Document With find_one()

The most basic type of query that can be performed in MongoDB is find_one(). This method returns a single document matching a query (or None if there are no matches). It is useful when you know there is only one matching document, or are only interested in the first match. Here we use find_one() to get the first document from the posts collection:

```
>>> import pprint
>>> pprint.pprint(posts.find_one())
{u'_id': ObjectId('...'),
 u'author': u'Mike',
 u'date': datetime.datetime(...),
 u'tags': [u'mongodb', u'python', u'pymongo'],
 u'text': u'My first blog post!'}
```

The result is a dictionary matching the one that we inserted previously.

FAQS:

Q.1. What is Pymongo?

Q2. Write code to retrieve record from mongdb using pyMongo?

Conclusion:

Outcome of the experiment is students are able to understand and implement MongoDB User interface in Python.