# NoSQL Databases

Name: _____ (  ) Class: _____ Date: _____

## Lesson 2: Using PyMongo

**Instructional Objectives:**

By the end of this task, you should be able to:

- Use PyMongo to connect to MongoDB server
- Create MongoDB databases using PyMongo
- Access MongoDB databases using PyMongo
- Obtain and modify MongoDB documents with PyMongo
- Use query operators in PyMongo

---

**What is PyMongo?**

MongoDB databases can be accessed using different programming languages like C, Java and Python. To access MongoDB databases using Python, we use the Python driver for MongoDB, PyMongo.

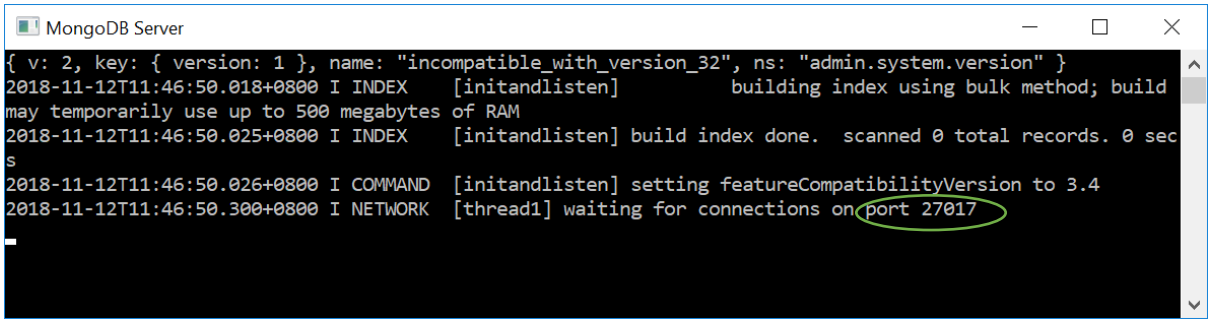
To use PyMongo, start your Python program by importing the pymongo package.

Try typing and running program 1 below. (Remember to start the MongoDB server before you run the program.) The program connects to the MongoDB server and outputs the databases currently in the MongoDB server.

**Program 1:** `access.py`

```
1  import pymongo
2  client = pymongo.MongoClient("127.0.0.1", 27017)
3  databases = client.database_names()
4  print("The databases in the MongoDB server are:")
5  print(databases)
6  client.close()
```

Line 2 connects to the local MongoDB database which is usually at port 27017. You can see the port number when you start the MongoDB server. Line 6 closes the connection to the server. The MongoDB server window should remain open while you want to access the MongoDB database.




Line 3 of the code retrieves the names of the databases, stored as a Python `list`.

As an example, let's create a database to store details on movie information.

Please note that MongoDB waits until you have inserted at least one document before it actually creates the database and collection.

**Program 2:** `insert.py`

```
1   import pymongo
2   client = pymongo.MongoClient("127.0.0.1", 27017)
3   db = client.get_database("entertainment")
4   coll = db.get_collection("movies")
5   coll.insert_one({"_id":1, "title":"Johnny Maths",
    "genre":"comedy"})
6   coll.insert_one({"title":"Star Walls", "genre":"science
    fiction"})
7   coll.insert_one({"title":"Detection"}) #no genre
8   list_to_add = []
9   list_to_add.append({"title":"Badman", "genre":"adventure",
    "year":2015})
10  list_to_add.append({"title":"Averages", "genre":["science
    fiction","adventure"], "year":2017})
11  list_to_add.append({"title":"Octopus Man",
    "genre":"adventure", "year":2017})
12  list_to_add.append({"title":"Fantastic Bees",
    "genre":"adventure", "year":2018})
13  list_to_add.append({"title":"Underground", "genre":"horror",
    "year":2014})
14  coll.insert_many(list_to_add)
15  c = db.collection_names("entertainment")
16  print ("Collections in entertainment database: ",c)
17  client.close()
```

Program 2 demonstrates two ways of inserting documents into collection `entertainment`. To insert one document, you can use the `insert_one()` method shown in lines 5 and 6. Notice that all not fields are required for insertion, as shown

in lines 5 to 7. To insert multiple documents, you can use the `insert_many()` method to insert a `list` of documents as shown in line 13.

MongoDB will assign a unique `_id` to each document. You can customise the `_id` by stating it during the insertion process, as shown on line 5. However, this means that you cannot run program 2 again until you remove this document, otherwise the program will produce an error. You can try to run the program again with line 5 commented out. Duplicates of the other documents will be created.

Line 15 gathered the list of collections while line 16 prints it as a list.

1. Write a Python program to ask for one movie title and the year of movie, then insert the document into the `movie` collection. Assume no genre is given.

**Q1 Program:** `q1.py`

```
import pymongo
title = input("Enter movie title")
year = input("Enter year of movie")
client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("entertainment")
coll = db.get_collection("movie")
coll.insert_one({"title":title, "year":year})
client.close()
```

**Go further!** Can you extend the program to include genres (where movies can have none or multiple genres)?

Of course, for large amount of data, it is more efficient to import from a file.

2. The program below reads from a delimited text file and insert the documents into the database. Parts of the input file and the program are given below. Fill in the blanks.

**Input File:** `input.txt`

```
Amanda,45
Bala,28
Charlie,33
Devi,29
...
```

**Q2 Program:** `q2.py`

```
import pymongo, csv

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("entertainment")
coll = db.get_collection("users")

with open('input.txt') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        coll.insert_one({"name":row[0], "age":row[1]})
client.close()
```

If the file is in JSON (JavaScript Object Notation), the data can also be imported using the `load()` function. A sample JSON file and program is shown below.

**JSON file:** input.`json`

```
[
      {
      "name": "Amanda",
      "age": "45"
      },
      {
      "name": "Bala",
      "age": "28"
      },
      {
      "name": "Charlie",
      "age": "33"
      },
      {
      "name": "Devi",
      "age": "29"
      }
]
```

**Program 3:** `loadjson.py`

```
import pymongo, json
client = pymongo.MongoClient("127.0.0.1", 27017)
with open('data.json') as file:
    data = json.load(file)
client['entertainment']['moreusers'].insert_many(data)
client.close()
```

Let's now try to get the data from the database.

**Program 4:** `view.py`

```
1   import pymongo
2   client = pymongo.MongoClient("127.0.0.1", 27017)
3   db = client.get_database("entertainment")
4   coll = db.get_collection("movies")
5   result = coll.find()
6   print("All documents in movie collection:")
7   for document in result:
8       print(document)
9   print("Number of items in movie collection:", coll.count())
10
11  result = coll.find({'genre': 'adventure'})
12  print("All movies with adventure genre:")
13  for document in result:
14      print(document)
15
16  query2 = {'genre': 'adventure', 'year': {'$gt': 2016}}
17  result = coll.find(query2)
18  print("All titles of movies with adventure genre after
    2016:")
19  for document in result:
20      print(" - " + document.get('title'))
21  print("There are",result.count(),"movies in the list above.")
22  client.close()
```

The method `find()` in line 5 returns a `Cursor` of all the documents in the `movie` collection. The results can be printed with a loop. The `count()` method gives the number of documents in the `movie` collection.

Line 11 onwards demonstrates the searching of specific documents in MongoDB. The query can be formed directly as shown in line 11, or built with variables (see lines 16 and 17). Each document is just a Python `dict`, so you can use the usual built-in methods for `dict`. For example, line 20 uses the `get()` method to retrieve the value of `title`. This allows you to extract the value for a particular field in the document.

Line 16 of the code creates the query to find the documents with adventure genre **and** year greater than 2016. It can be rewritten using the `$and` operator:

```
query2 = {'$and':[{'genre': 'adventure'}, {'year': {'$gt': 2016}}]}
```

Line 21 shows how to obtain the number of documents in the search results. Using the `count()` method, it gives the number of titles of movies with adventure genre after 2016.

The following is a list of commonly used query operators.

| | |
|---|---|
| `$eq` | Equals to |
| `$gt` | Greater than |
| `$gte` | Greater than or equal to |
| `$lt` | Less than |
| `$lte` | Less than or equal to |
| `$ne` | Not equal to |
| `$in` | In a specified `list` |
| `$nin` | Not in a specified `list` |
| `$or` | Logical OR |
| `$and` | Logical AND |
| `$not` | Logical NOT |
| `$exists` | Matches documents which has the named field |

Program 5 demonstrates the use of some of these query operators.

**Program 5:** `view2.py`

```
1   import pymongo
2   client = pymongo.MongoClient("127.0.0.1", 27017)
3   db = client.get_database("entertainment")
4   coll = db.get_collection("movies")
5
6   result = coll.find()
7   print("All documents in movie collection:")
8   for document in result:
9       print(document)
10  print("Number of items in movie collection:", coll.count())
11
12  result = coll.find({'genre':{'$in':['adventure', 'comedy']}})
13  print("All movies with adventure or comedy genre inside:")
14  for document in result:
15      print(document)
16
17  query2 = {'genre': {'$exists':False}}
18  result = coll.find(query2)
19  print("All movies without genre:")
20  for document in result:
21      print(" - " + document.get('title'))
22
23  result = coll.find_one({'year':{'$eq':2017}})
24  print("One movie that was released in 2017")
25  print(result)
26  client.close()
```

Line 23 uses `find_one()` which returns one document that matches the condition. Run the program. Modify the program with different query operators and options.

3. Modify lines 12 and 13 to find all movies without adventure and comedy genres.

```
result = coll.find({'genre':{ '$nin':['adventure', 'comedy']}})

print("All movies without adventure and comedy genres:")
```

.....

4. Modify lines 17 to 21 such that for all movies with year, print out the movie title and how many years ago the movie was released.

```
query2 = {'year': {'$exists':True}}
result = coll.find(query2)
print("All movies with year given:")
for document in result:
    print(" - Title:", document.get('title'), ", No. of year(s) since release:",2018-document.get('year'))
```

.....

5. Modify lines 23 to 25 to print out all movies released before 2017.

```
result = coll.find({'year':{'$lt':2017}})
print("All movies that were released before 2017")
for document in result:
    print(document)
```

.....

To modify the content in the database, use the `update_one()` method to modify the first document that matches the query, or the `update_many()` method to modify all documents that matches the query. Program 6 demonstrates the update process. Line 12 uses `$set` to set all the year values greater than 2016 to be 2015. There is also the `$unset` operator to remove given fields (see line 28). Note that even though `$unset` operator removes the given fields, there is still a requirement to have a second argument, thus 0 is placed even though it won't be updated.

**Program 6:** `update.py`

```
1   import pymongo
2   client = pymongo.MongoClient("127.0.0.1", 27017)
3   db = client.get_database("entertainment")
4   coll = db.get_collection("movies")
5
6   result = coll.find()
7   print("All documents in movies collection:")
8   for document in result:
9       print(document)
10
11  search = {'year':{'$gt':2016}}
12  update = {'$set':{'year':2015}}
13  coll.update_one(search, update)
14
15  result = coll.find()
16  print("All documents in movies collection after update one:")
17  for document in result:
```

```
18      print(document)
19
20  coll.update_many(search, update)
21
22  result = coll.find()
23  print("All documents in movies collection after updating
    all:")
24  for document in result:
25      print(document)
26
27  search = {'year':{'$eq':2018}}
28  update = {'$unset':{'year':0}}
29  coll.update_many(search, update)
30
31  result = coll.find()
32  print("All documents in movies collection after unset:")
33  for document in result:
34      print(document)
35
36  client.close()
```

6. Modify lines 11 and 12 to add comedy genre to all movies that currently have no genres.
   `search = {'genre':{'$exists':False}}`
   `update = {'$set':{'genre':'Comedy'}}`
   .....

7. Modify lines 27 and 28 to remove the genre field to all movies that currently have adventure as its genre or one of its genre.
   `search = {'genre':{'$in':'adventure'}}`
   `update = {'$unset':{'genre':0}}`
   .....

To delete a collection, you can use the `delete_one()` method to delete the first document that matches the given condition, or `delete_many()` method to delete all the documents that match the condition. This is demonstrated by program 7 below.

**Program 7:** `delete.py`

```
1   import pymongo
2   client = pymongo.MongoClient("127.0.0.1", 27017)
3   db = client.get_database("entertainment")
4   coll = db.get_collection("movies")
5
6   result = coll.find()
7   print("All documents in movies collection:")
8   for document in result:
9       print(document)
```

```
10
11  coll.delete_one({'year':2015})
12
13  result = coll.find()
14  print("All documents in movies collection after deleting
    one:")
15  for document in result:
16        print(document)
17
18  coll.delete_many({'year':2015})
19
20  result = coll.find()
21  print("All documents in movies collection after deleting
    all:")
22  for document in result:
23        print(document)
24
25  client.close()
```

**8.** Modify line 18 to delete all movies with adventure as its genre or one of its genre.

```
coll.delete_many({'genre':{'$eq':'adventure'}})
# coll.delete_many({'genre':'adventure'}) is correct too.
```

.....

To clear the collection, you can write a program similar to program 8 below.

**Program 8:** `remove.py`

```
1   import pymongo
2   client = pymongo.MongoClient("127.0.0.1", 27017)
3   db = client.get_database("entertainment")
4   coll = db.get_collection("tv")
5   coll.insert_one({"title":"X Man", "genre":"science fiction"})
6   coll.insert_one({"title":"Fresh from the boat",
    "genre":"comedy"})
7   coll.insert_one({"title":"", "genre":"comedy"})
8   coll.insert_one({"genre":"comedy"})
9   result = coll.find()
10  print("All documents in tv collection:")
11  for document in result:
12      print(document)
13  print("Number of items in tv collection:", coll.count())
14  db.drop_collection("tv")
15  result = coll.find()
16  print("After tv collection is dropped:")
17  for document in result:
18      print(document)
19  print("Number of items in tv collection:", coll.count())
```

| | |
|---|---|
| 20 | `client.close()` |

To remove the entire `entertainment` database, you can use the following statement. That will remove all the collections and the documents within it.
`client.drop_database("entertainment")`

**9.** You are tasked to create and store concert information on a NoSQL database, accessing them through a Python program.
Create a program to insert concert information (e.g. concert title, date, time, venue, price of tickets), search for information on a concert using concert title and delete the entire concert by concert title (assuming that all concerts have unique titles). You should have a menu to allow the user to select the option, and an option to end the program.

**Q9 Program:** `q9.py`

```
# possible solution given below
import pymongo

def menu_options():
    print("------------Menu------------")
    print("1: Add a concert")
    print("2: Search for concert")
    print("3: Display all concerts")
    print("4: Delete concert")
    print("5: Exit program")
    try:
        option = int(input("Enter program option (1, 2, 3, 4
or 5):"))
        return option
    except ValueError:
        print("Please enter a number.")
    except:
        print("Error occurred")
    return -1

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("entertainment")
coll = db.get_collection("concerts")
menuoption = -1
while (menuoption != 5):
    if (menuoption == 1):
        #add
        title = input("Enter concert title:")
        date = input("Enter concert date:")
        time = input("Enter concert time:")
        venue = input("Enter concert venue:")
        price = input("Enter the price of tickets (e.g. $200,
$100, $50)")
```

```
        try:
            coll.insert_one({"title":title, "date":date,
"time":time, "venue":venue, "price": price})
            print("Concert entry entered")
        except:
            print("Error occurred while trying to insert.")
    elif (menuoption == 2):
        #search
        title = input("Enter concert title for searching:")
        try:
            result = coll.find({'title':title})
            print("Search result")
            for document in result:
                print(document)
        except:
            print("Error occurred while trying to search.")
    elif (menuoption == 3):
        #display all
        result = coll.find()
        print("List of all concerts with given title")
        for document in result:
            print(document)
    elif (menuoption == 4):
        #delete
        to_delete = input("Enter title of concert to delete")
        try:
            coll.delete_one({'title':title})
            print("Deleted      first      concert      with
title",to_delete)
        except:
            print("Error occurred when deleting.")
    else:
        #menuoption is -1 or invalid option
        print("Please select a valid program option. (1, 2, 3,
4 or 5)")
    menuoption = menu_options()
# end program
client.close()
print("End of program")
```

---

## References

| Content | Link |
|---|---|
| NoSQL Databases | https://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/ |

# NoSQL Databases

| | |
|---|---|
| | https://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases<br><br>https://www.mongodb.com/scale/types-of-nosql-databases |
| MongoDB/PyMongo | https://www.mongodb.com/what-is-mongodb<br><br>http://api.mongodb.com/python/current/tutorial.html |