Temasek Junior College

2023 JC2 H2 Computing

Database 7 – PyMongo

Syllabus Objectives

After completing this set of notes, 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

1 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.

Step 1: Start the MongoDB server. The MongoDB server window should remain open while you want to access the MongoDB database.

Step 2: Start your Python program by importing the pymongo package.

Example 1

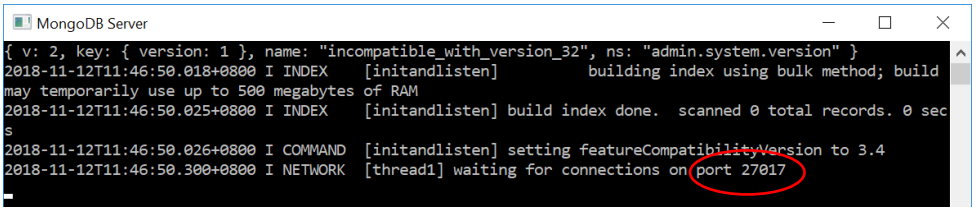
The program below connects to the MongoDB server and outputs the databases currently in the MongoDB server.

Program 1: access.py

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

8  client.close()
```

Line 2 connects to the local MongoDB database via port 27017. (You can obtain the port number when you start the MongoDB server.)



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

Line 6 closes the connection to the server.

2 Create Operations

Example 2

The program below demonstrates two ways of inserting documents into the `movies` collection of the `entertainment` database.

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

Program 2: movie.py

```
1  import pymongo
2
3  client = pymongo.MongoClient("127.0.0.1", 27017)
4  db = client.get_database("entertainment")
5  coll = db.get_collection("movies")
6
7  coll.insert_one({"_id":1, "title":"Johnny Maths", "genre":"comedy"})
8  coll.insert_one({"title":"Star Walls", "genre":"science fiction"})
9  coll.insert_one({"title":"Detection"}) #no genre
10
11 list_to_add = []
12 list_to_add.append({"title":"Badman", "genre":"adventure", year":2015})
13 list_to_add.append({"title":"Averages", "genre":["science
   fiction","adventure"], "year":2017})
14 list_to_add.append({"title":"Octopus Man", "genre":"adventure", "year":2017})
15 list_to_add.append({"title":"Fantastic Bees", "genre":"adventure",
   "year":2018})
16 list_to_add.append({"title":"Underground", "genre":"horror", "year":2014})
17
18 coll.insert_many(list_to_add)
19
20 c = db.collection_names("entertainment")
21
22 print ("Collections in entertainment database: ",c)
23
24 client.close()
```

To insert one document, you can use the `insert_one()` method shown in Lines 7, 8 and 9. Notice that not all fields are required for insertion.

To insert multiple documents, you can use the `insert_many()` method to insert a `list` of documents as shown in Line 18.

MongoDB will assign a unique `_id` to each document. You can customise the `_id` by stating it during the insertion process, as shown in Line 7. 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 7 commented out. Duplicates of the other documents will be created.

Line 20 gathers the list of collections while Line 22 prints it as a list.

Example 3 [Practice]
Write a Python program to ask for one movie title and the year (as an integer) of the movie, then insert the document into the movie collection. Assume no genre is given.

Program 3: insert_from_input.py

```
1  import pymongo
2
3  client = pymongo.MongoClient("127.0.0.1", 27017)
4  db = client.get_database("entertainment")
5  coll = db.get_collection("movie")
6
7  title =
8  year =
9
10                                                  # insert the document
11
12 client.close()
```

Can you extend the above program to include genres (where movies can have none or multiple genres)?

In fact, to do so in the manner of the above program will be tedious. For large amounts of data, it is more efficient to import from a file.

Example 4
The delimited text file `input.txt` contains data in the following format.

Text File: input.txt

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

The program below reads from `input.txt` and inserts the documents into the database.

Program 4: insert_from_file.py

```
1  import pymongo, csv
2
3  client = pymongo.MongoClient("127.0.0.1", 27017)
4  db = client.get_database("entertainment")
5  coll = db.get_collection("users")
6
7  with open('input.txt') as csv_file:
8      csv_reader = csv.reader(csv_file, delimiter=',')
9      for row in csv_reader:
10         coll.insert_one({"name":row[0], "age":row[1]})
11
12 client.close()
```

Example 5

The JSON file `input.json` contains data in the following format.

JSON File: input.json

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

The `load()` function can be used to import the data from `input.json`. The program below demonstrates how this can done.

Program 5: insert_from_json.py

```
1  import pymongo, json
2
3  client = pymongo.MongoClient("127.0.0.1", 27017)
4
5  with open('input.json') as file:
6      data = json.load(file)
7
8  client['entertainment']['moreusers'].insert_many(data)
9
10 client.close()
```

The syntax `client['entertainment']['moreusers']` in Line 8 refers to the database `entertainment` and the collection `moreusers`.

3 Read Operations

Example 6
The program below demonstrates how data can be obtained from the database.

Program 6: view.py

```
1  import pymongo
2
3  client = pymongo.MongoClient("127.0.0.1", 27017)
4  db = client.get_database("entertainment")
5  coll = db.get_collection("movies")
6
7  result = coll.find()
8
9  print("All documents in movie collection:")
10 for document in result:
11     print(document)
12 print("Number of items in movie collection:", coll.count())
13
14 result = coll.find({'genre': 'adventure'})
15
16 print("All movies with adventure genre:")
17 for document in result:
18     print(document)
19
20 query2 = {'genre': 'adventure', 'year': {'$gt': 2016}}
21 result = coll.find(query2)
22
23 print("All titles of movies with adventure genre after 2016:")
24 for document in result:
25     print(" - " + document.get('title'))
26 print("There are",result.count(),"movies in the list above.")
27
28 client.close()
```

The method `find()` in Line 7 returns a cursor of all the documents in the movie collection. The results can be printed with an iterative loop.

The `count()` method gives the number of documents in the movie collection.

Line 14 onwards demonstrates the searching of specific documents in MongoDB. The query can be formed directly as shown in Line 14, or built with variables (see Lines 20 and 21).

Each document is just a Python dictionary. Hence you can use the usual built-in methods for dictionaries. For example, Line 25 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 20 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 as follows:

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

Line 26 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.

3.1 Common 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

Example 7

The program below demonstrates the use of some of the common query operators.

Program 7: view2.py

```
1  import pymongo
2
3  client = pymongo.MongoClient("127.0.0.1", 27017)
4  db = client.get_database("entertainment")
5  coll = db.get_collection("movies")
6
7  result = coll.find()
8
9  print("All documents in movie collection:")
10 for document in result:
11     print(document)
12 print("Number of items in movie collection:", coll.count())
13
14 result = coll.find({'genre':{'$in':['adventure', 'comedy']}})
15
16 print("All movies with adventure or comedy genre inside:")
17 for document in result:
18     print(document)
19
20 query2 = {'genre': {'$exists':False}}
21 result = coll.find(query2)
22
23 print("All movies without genre:")
24 for document in result:
25     print(" - " + document.get('title'))
26
27 result = coll.find_one({'year':{'$eq':2017}})
28
29 print("One movie that was released in 2017")
30 print(result)
31
32 client.close()
```

Line 27 uses `find_one()` which returns one document that matches the condition.

Exercise 1

Modify the program with different query operators and options to perform the following:

(a) Find all movies without adventure and comedy genres.

```
result = coll.find(                                    )

for document in result:
    print(document)
```

(b) For all movies with data with year, print out the movie title and how many years ago the movie was released.

```
result = coll.find(                          )

for document in result:
    title =                          
    years_released = 2023 -                  
    print(f"Title: {title}, Released {years_released} years ago")
```

(c) Print out all movies released before 2017.

```
result = coll.find(                          )

for document in result:
    print(document)
```

4 Update Operations

To modify the content in the database, use

- `update_one()` method to modify the first document that matches the query,
- `update_many()` method to modify all documents that matches the query.

Example 8

The program below demonstrates the update process. Line 13 uses `$set` to set all the year values greater than 2016 to be 2015.

There is also the `$unset` operator to remove given fields (see Line 29). Note that even though `$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 8: update.py

```
1  import pymongo
2
3  client = pymongo.MongoClient("127.0.0.1", 27017)
4  db = client.get_database("entertainment")
5  coll = db.get_collection("movies")
6
7  result = coll.find()
8  print("All documents in movies collection:")
9  for document in result:
10     print(document)
11
12 search = {'year':{'$gt':2016}}
13 update = {'$set':{'year':2015}}
14 coll.update_one(search, update)
15
16 result = coll.find()
17 print("All documents in movies collection after update one:")
18 for document in result:
19     print(document)
20
21 coll.update_many(search, update)
22
23 result = coll.find()
24 print("All documents in movies collection after updating all:")
25 for document in result:
26     print(document)
27
28 search = {'year':{'$eq':2014}}
29 update = {'$unset':{'year':0}}
30 coll.update_many(search, update)
31
32 result = coll.find()
33 print("All documents in movies collection after unset:")
34 for document in result:
35     print(document)
36
37 client.close()
```

Exercise 2

(a) Modify Lines 12 and 13 to add comedy genre to all movies that currently have no genres.

(b) Modify lines 28 and 29 to remove the genre field to all movies that currently have adventure as its genre or one of its genre.

5 Delete Operations

To delete a collection, you can use

- `delete_one()` method to delete the first document that matches the given condition
- `delete_many()` method to delete all the documents that match the condition.

Example 9

The program below demonstrates the use of the above methods.

Program 9: delete.py

```
1  import pymongo
2
3  client = pymongo.MongoClient("127.0.0.1", 27017)
4  db = client.get_database("entertainment")
5  coll = db.get_collection("movies")
6
7  result = coll.find()
8  print("All documents in movies collection:")
9  for document in result:
10       print(document)
11
12 coll.delete_one({'year':2015})
13
14 result = coll.find()
15 print("All documents in movies collection after deleting one:")
16 for document in result:
17       print(document)
18
19 coll.delete_many({'year':2015})
20
21 result = coll.find()
22 print("All documents in movies collection after deleting all:")
23 for document in result:
24       print(document)
25
26 client.close()
```

Exercise 3

Modify Line 19 to delete all movies with adventure as its genre or one of its genre.

Example 10
The program below demonstrates how to clear a collection.

Program 10: remove.py

```
1  import pymongo
2
3  client = pymongo.MongoClient("127.0.0.1", 27017)
4  db = client.get_database("entertainment")
5  coll = db.get_collection("tv")
6
7  coll.insert_one({"title":"X Man", "genre":"science fiction"})
8  coll.insert_one({"title":"Fresh from the boat", "genre":"comedy"})
9  coll.insert_one({"title":"", "genre":"comedy"})
10 coll.insert_one({"genre":"comedy"})
11
12 result = coll.find()
13
14 print("All documents in tv collection:")
15 for document in result:
16     print(document)
17 print("Number of items in tv collection:", coll.count())
18
19 db.drop_collection("tv")
20
21 result = coll.find()
22
23 print("After tv collection is dropped:")
24 for document in result:
25     print(document)
26 print("Number of items in tv collection:", coll.count())
27
28 client.close()
```

To remove the entire entertainment database, you can use the following statement. All collections and documents within the database will be removed.

```
client.drop_database("entertainment")
```

Practice Task
You are tasked to create and store concert information on a NoSQL database, accessing them through a Python program.

Create a program that allows user 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.

Add in one more option to view all the concert information