**Temasek Junior College**

**2024 JC2 H2 Computing**

# NoSQL Databases 2 – PyMongo

| | | |
|---|---|---|
| **Section** | 3 | Data and Information |
| **Unit** | 3.3 | Databases and Data Management |
| **Objectives** | 3.3.6 | Understand how NoSQL database management system addresses the shortcomings of relational database management system (SQL). |
| | 3.3.7 | Explain the applications of SQL and NoSQL. |

## 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`**.

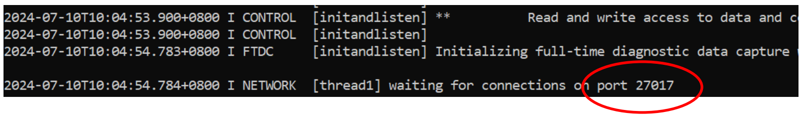
***<u>Step 1</u>***
Start the MongoDB server. The MongoDB server window **<u>MUST REMAIN OPENED</u>** while you want to access the MongoDB database.

***<u>Step 2</u>***
Start your Python program by importing **`PyMongo`** on your IDLE or Jupyter Notebook.

**<u>Example 1</u>**
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) # Connect to server
4  
5  databases = client.list_database_names() # Check available databases
6  
7  print("The databases in the MongoDB server are:")
8  print(databases)
9  
10 client.close() # Always remember to close connection
```

**<u>Establishing Connection</u>**
Line **`3`** creates a connection to the local MongoDB database via port 27017. (You can obtain the port number when you start the MongoDB server.)



```
2024-07-10T10:04:53.900+0800 I CONTROL  [initandlisten] **          Read and write access to data and c
2024-07-10T10:04:53.900+0800 I CONTROL  [initandlisten]
2024-07-10T10:04:54.783+0800 I FTDC     [initandlisten] Initializing full-time diagnostic data capture

2024-07-10T10:04:54.784+0800 I NETWORK  [thread1] waiting for connections on port 27017
```

**<u>Checking Available Databases</u>**
In Line **`5`**, **`list_database_names()`** can be used to retrieved the names of the MongoDB databases available as a Python list.

**<u>Terminating Connection</u>**
Connection to the MongoDB Server can be terminated using **`close()`**, as shown on Line **`10`**.

## 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)  # Connect to server
4   db = client.get_database("entertainment")
5   coll = db.get_collection("movies")
6
7   # Insert one document at a time
8   coll.insert_one({"_id":1, "title":"Johnny Maths", "genre":"comedy"})
9   coll.insert_one({"title":"Star Walls", "genre":"science fiction"})
10  coll.insert_one({"title":"Detection"}) # no genre
11
12  # Create and insert list of documents
13  docs = []
14  docs.append({"title":"Badman", "genre":"adventure", "year":2015})
15  docs.append({"title":"Mean", "genre":["sci-fi","adventure"], "year":2017})
16  docs.append({"title":"Octopus Man", "genre":"adventure", "year":2017})
17  docs.append({"title":"Fantastic Bees", "genre":"adventure", "year":2018})
18  docs.append({"title":"Underground", "genre":"horror", "year":2014})
19
20  coll.insert_many(list_to_add)
20
21  collections = db.list_collection_names()  # Get collections in database
22
23  print(f'Collections in entertainment database: {collections}')
24
25  client.close()  # Always remember to close connection
```

**Insert one document**

Use the `insert_one()` method as shown in Lines **8**, **9** and **10**. Notice that not all fields are required for insertion.

**Insert multiple documents**

Use the `insert_many()` method to insert a list of documents as shown in Line **20**.

**Unique document ID assignment**

MongoDB will automatically assign a unique `_id` to each document. You can customise the `_id` by stating it during the insertion process, as shown in Line **8**. However, this means that you cannot run the program again until you remove this document, otherwise the program will produce an error as the customised `_id` is already in use after the first run of the program.

You can try to run the program again with Line **8** commented out. Duplicates of the other documents will be created as new _id values will be automatically assigned by the system when the documents are added again.

## Checking Available Collections in a Database

The **`list_collection_names()`** method in Line **21** can be used to gather a list of collections in the database.

## Exercise 1

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**

```
import pymongo

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

title =                                       # ask for title
year =                                        # ask for year

                                              # insert the document

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 3

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**

```
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()
```

## Example 4

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 5**
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()
```

**Retrieving All Documents**
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.

**Retrieving Specific Documents**
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 `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

In Line **20** of **Example 5**, 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}}]}
```

The table below shows a list of common MongoDB query operators:

| | | | |
|---|---|---|---|
| **`$eq`** | Equals to | **`$in`** | In a specified **`list`** |
| **`$gt`** | Greater than | **`$nin`** | Not in a specified **`list`** |
| **`$gte`** | Greater than or equal to | **`$or`** | Logical **`OR`** |
| **`$lt`** | Less than | **`$and`** | Logical **`AND`** |
| **`$lte`** | Less than or equal to | **`$not`** | Logical **`NOT`** |
| **`$ne`** | Not equal to | **`$exists`** | Matches documents having 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(f'One movie that was released in 2017 is {result}')
30 
31 client.close()
```

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

**Exercise 2**

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 3

**(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.

**<u>Example 9</u>**
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()
```

**<u>Exercise 3</u>**
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**

```
import pymongo

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

coll.insert_one({"title":"X Man", "genre":"science fiction"})
coll.insert_one({"title":"Fresh from the boat", "genre":"comedy"})
coll.insert_one({"title":"", "genre":"comedy"})
coll.insert_one({"genre":"comedy"})

result = coll.find()

print("All documents in tv collection:")
for document in result:
    print(document)
print("Number of items in tv collection:", coll.count())

db.drop_collection("tv")

result = coll.find()

print("After tv collection is dropped:")
for document in result:
    print(document)
print("Number of items in tv collection:", coll.count())

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")
```