Learn MongoDB

Objectives:

- Connect to MongoDB
- Get, list, insert, update and delete documents from MongoDB

1. Get Started

Install Library

Libary `pymongo` and `dnspython`

To work with MongoDB in Python, install library `pymongo`. If you are using MongoDB Cloud to host your database, you need to install `dnspython` to connect to MongoDB Cloud.

Run following commands on command line to install these 2 libraries.

In [1]:
```
!pip install dnspython
!pip install pymongo
```

```
Requirement already satisfied: dnspython in c:\users\isszq\anaconda3\lib\si
te-packages (2.0.0)
Requirement already satisfied: pymongo in c:\users\isszq\anaconda3\lib\site
-packages (3.11.0)
```

Import `pymongo` library, and print its version.

In [2]:
```
import pymongo
pymongo.__version__
```

Out[2]: `'3.11.0'`

Create a MongoClient

Option 1: Connect to MongoDB at Localhost

MongoClient by default will connect to `localhost` at port `27017`.

In [ ]:
```
client = pymongo.MongoClient("mongodb://localhost:27017")
client
```

Option 2: Connect to MongoDB Cloud

To connect to MongoDB Cloud, use the connection string copied from MongoDB Cloud.

- Remember to update username, password and database-name in connection string.

In [3]:
```
client = pymongo.MongoClient("mongodb+srv://root:qwer1234@cluster0.hlix
client
```

Out[3]:
```
MongoClient(host=['cluster0-shard-00-02.hlixs.mongodb.net:27017', 'cluster0
-shard-00-00.hlixs.mongodb.net:27017', 'cluster0-shard-00-01.hlixs.mongodb.
net:27017'], document_class=dict, tz_aware=False, connect=True, retrywrites
=True, w='majority', authsource='admin', replicaset='atlas-s2hqn5-shard-0',
ssl=True)
```

Check out documentation of `MongoClient`. Check out its attributes, e.g. `server_info()`, `list_database_names()`, `get_database()`

Note: If you hit a `ServerSelectionTimeoutError`, your MongoDB server may not be running. Run `mongod` on a command prompt.

Connect to a Database

Find out the list of existing databases in your MongoDB.

In [4]:
```
client.list_database_names()
```

Out[4]:
```
['demo', 'leisure', 'admin', 'local']
```

Connect to a database `demo`.

In [5]:
```
db = client.demo
db
```

Out[5]:
```
Database(MongoClient(host=['cluster0-shard-00-02.hlixs.mongodb.net:27017',
'cluster0-shard-00-00.hlixs.mongodb.net:27017', 'cluster0-shard-00-01.hlix
s.mongodb.net:27017'], document_class=dict, tz_aware=False, connect=True, r
etrywrites=True, w='majority', authsource='admin', replicaset='atlas-s2hqn5
-shard-0', ssl=True), 'demo')
```

Reference to a Collection

Check the documentation of `db` object. Database object offeres attributes like `list_collection_names()`.

In [6]:
```
db.*?
```

Drop the `students` collection, if it already exists, from the database.

In [7]:
```
result = db.drop_collection('students')
result
```

Out[7]:
```
{'nIndexesWas': 1,
 'ns': '5f58922223a2183a0bcc1647_demo.students',
 'ok': 1.0,
 '$clusterTime': {'clusterTime': Timestamp(1600364974, 1),
  'signature': {'hash': b'\xc3h\x0b\xaf\xc6\xa1\xf0\x8e\x86\xd4od\xe3L`\x8
c"\xe4\x19\xe9',
   'keyId': 6869179485573349379}},
 'operationTime': Timestamp(1600364974, 1)}
```

Create a reference to a collection using its name, e.g. `students`.

- MongoDB will create the new database if it doesn't exist.

In [8]:
```
stud_col = db.students
print(type(stud_col))
stud_col
```

```
<class 'pymongo.collection.Collection'>
```

Out[8]:
```
Collection(Database(MongoClient(host=['cluster0-shard-00-02.hlixs.mongodb.n
et:27017', 'cluster0-shard-00-00.hlixs.mongodb.net:27017', 'cluster0-shard-
00-01.hlixs.mongodb.net:27017'], document_class=dict, tz_aware=False, conne
ct=True, retrywrites=True, w='majority', authsource='admin', replicaset='at
las-s2hqn5-shard-0', ssl=True), 'demo'), 'students')
```

Check out Collection documentation. A collection offers functions `insert_one()`, `insert_many()`, `find_one()`, `find()` etc.

In [9]:
```
stud_col.*?
```

List collections in the database.

In [10]:
```
db.list_collection_names()
```

Out[10]:
```
['tv-shows', 'users', 'books']
```

find() vs findOne()

- `find_one()` - if query matches, first document is returned, otherwise null.
- `find()` - nomatter number of documents matched, a cursor is returned, never null.

In [11]:
```
1 stud_col.find_one?
```

2. Work with a Collection

Insert a Document

Insert a document with following value:

```
{
    'name':'Ah Girl',
    'age':7,
    'subjects':['English', 'Physics']
}
```

MongoDB will automatically add a `_id` field if it doesn't exists in the dictionary.

In [12]:
```
1 # create a dictionary
2 d = {
3     'name':'Ah Girl',
4     'age':7,
5     'subjects':['English', 'Physics']
6 }
7 # insert it
8 result = stud_col.insert_one(d)
9 print(result.inserted_id)
```

```
5f63a1a0f72563e0846886d8
```

Insert another document with following value:

```
{
    'name':'Ah Boy',
    'age':10,
    'subjects':['Maths', 'Chemistry']
}
```

In [13]:
```
# create a dictionary
d = {
    'name':'Ah Boy',
    'age':10,
    'subjects':['Maths', 'Chemistry']
}
# insert it
result2 = stud_col.insert_one(d)
print(result2.inserted_id)
```

```
5f63a1a0f72563e0846886d9
```

Find a Document

Let's find our first inserted document by its ID `result.inserted_id` .

- To find a document by its `_id` , use `find_one()` method with filter `{'id': xxx}` .
- The returned value is a dictionary.

In [14]:
```
stud = stud_col.find_one({'_id':result.inserted_id})
print(type(stud))
print(stud)
```

```
<class 'dict'>
{'_id': ObjectId('5f63a1a0f72563e0846886d8'), 'name': 'Ah Girl', 'age': 7,
'subjects': ['English', 'Physics']}
```

Find All Documents

To find all existing documents in the collection, use `find()` method to get a cursor.

In [15]:
```
cursor = stud_col.find()
print(type(cursor))
```

```
<class 'pymongo.cursor.Cursor'>
```

In [16]:
```
cursor.*?
```

Retrieve all records from cursor by converting it to a list.

In [17]:
```
records = list(cursor)
print(records)
```

```
[{'_id': ObjectId('5f63a1a0f72563e0846886d8'), 'name': 'Ah Girl', 'age': 7,
'subjects': ['English', 'Physics']}, {'_id': ObjectId('5f63a1a0f72563e08468
86d9'), 'name': 'Ah Boy', 'age': 10, 'subjects': ['Maths', 'Chemistry']}]
```

Must close `cursor` after use. If not, it will end up in memory leak.

In [18]:
```
1 cursor.close()
```

Count Documents

Count all documents in a collection.

In [19]:
```
1 stud_col.count_documents({})
```

Out[19]: 2

Count documents in a collection, which matches a filter.

Question: Why it returns a document count of 0?

In [20]:
```
1 stud_col.count_documents({'name': 'ah girl'})
```

Out[20]: 0

By default, the search in MongoDB is case-sensitive. To make it case insensitive, use `$regex` with `'$options': 'i'`.

In [21]:
```
1 stud_col.count_documents({'name':
2     {'$regex': '^ah girl$', '$options': 'i'} })
```

Out[21]: 1

You can also count documents whose name contains `'Girl'`.

In [22]:
```
1 stud_col.count_documents({'name': {'$regex': 'Girl'} })
```

Out[22]: 1

Find Documents by Attributes

Similarly, you can find documents by filter. Regex is supported in find-operation too.

In [23]:
```
cursor = stud_col.find({'name':'Ah Girl'})
records = list(cursor)
print(records)
cursor.close()
```

```
[{'_id': ObjectId('5f63a1a0f72563e0846886d8'), 'name': 'Ah Girl', 'age': 7,
'subjects': ['English', 'Physics']}]
```

Update a Document

To update document(s) in database, you can use `update_one()` or `update_many()`.

- Records to be found by attributes
- Attributes can be updated using `$set` parameter

Exercise: Update a student, whose `name` is `Ah girl`, by setting her age to `12`.

In [24]:
```
result = stud_col.update_one({'name': 'Ah Girl'},
                             {'$set': {'age':12}})
print('Matched =', result.matched_count)
print('Modified =', result.modified_count)
```

```
Matched = 1
Modified = 1
```

Additional attributes can be added to the document using `$set`.

In [25]:
```
result = stud_col.update_one({'name': 'Ah Girl'},
                             {'$set': {'age':12, 'grade':1}})
print('Matched =', result.matched_count)
print('Modified =', result.modified_count)
```

```
Matched = 1
Modified = 1
```

Examine the updated document.

In [26]:
```
row = stud_col.find_one({'name':'Ah Girl'})
print(row)
```

```
{'_id': ObjectId('5f63a1a0f72563e0846886d8'), 'name': 'Ah Girl', 'age': 12,
'subjects': ['English', 'Physics'], 'grade': 1}
```

Remove Attribute(s) from a Document

To remove attribute(s) from a document, use `$unset` parameter.

In [27]:
```
result = stud_col.update_one({'name':'Ah Girl'},
                             {'$unset': {'grade':0}})
print('Matched =', result.matched_count)
print('Modified =', result.modified_count)
```

```
Matched = 1
Modified = 1
```

In [28]:
```
row = stud_col.find_one({'name':'Ah Girl'})
print(row)
```

```
{'_id': ObjectId('5f63a1a0f72563e0846886d8'), 'name': 'Ah Girl', 'age': 12,
'subjects': ['English', 'Physics']}
```

Find by Range

Find all students who are above 8 years old.

In [29]:
```
cursor = stud_col.find({'age': {'$gt':8}})
```

In [30]:
```
records = list(cursor)
print(records)
```

```
[{'_id': ObjectId('5f63a1a0f72563e0846886d8'), 'name': 'Ah Girl', 'age': 1
2, 'subjects': ['English', 'Physics']}, {'_id': ObjectId('5f63a1a0f72563e08
46886d9'), 'name': 'Ah Boy', 'age': 10, 'subjects': ['Maths', 'Chemistr
y']}]
```

In [31]:
```
cursor.close()
```

Delete a Document

Delete a student whose name is `'Ah Girl'` .

In [32]:
```
result = stud_col.delete_one({'name': 'Ah Girl'})
print(result.deleted_count)
```

```
1
```

Duplicate a record whose name = `'Ah Boy'` .

- Get the document and remove its `_id` attribute
- Insert the record back and MongoDB will create a document with new `_id`

In [33]:
```
r = stud_col.find_one({'name': 'Ah Boy'})
print(r)
r.pop('_id')
print(r)
stud_col.insert_one(r)
```

```
{'_id': ObjectId('5f63a1a0f72563e0846886d9'), 'name': 'Ah Boy', 'age': 10,
'subjects': ['Maths', 'Chemistry']}
{'name': 'Ah Boy', 'age': 10, 'subjects': ['Maths', 'Chemistry']}
```

Out[33]:
```
<pymongo.results.InsertOneResult at 0x195b293cc80>
```

Delete multiple students whose name are `'Ah Boy'`.

In [34]:
```
result = stud_col.delete_many({'name': 'Ah Boy'})
print(result.deleted_count)
```

```
2
```

3. Exercise

Task: Import Data into Database

Download JSON file from https://github.com/qinjie/sample-data/blob/master/tv-shows.json (https://github.com/qinjie/sample-data/blob/master/tv-shows.json)

Use python script to read the file and insert them into a collection `tvshows` in database `demo` in MongoDB Cloud.

In [12]:
```
with open('tv-shows.json') as f:
    data = json.load(f)
```

In [10]:
```
import requests
url = 'https://raw.githubusercontent.com/qinjie/sample-data/master/tv-s
response = requests.get(url)
data = response.json()
print(len(data))

```

Out[10]:
```
<pymongo.results.InsertManyResult at 0x22c74593d00>
```

In [ ]:
```
from pymongo import MongoClient
url = 'mongodb://127.0.0.1:27017'

client = MongoClient(url)
db = client['demo']
coll = db['tvshows']

result = coll.insert_many(data)
```

Task: Find documents and Save to File ¶

Find all tv-shows whose `runtime` is greater than or equals to `90`.

Save them in csv file with columns `name`, `language`, `average rating`.

In [29]:
```
with coll.find({'runtime': {'$gte': 90}}) as cursor:
    result = [i for i in cursor]

data = [[i['name'], i['language'], i['rating']['average']]
            for i in result]
data
```

Out[29]:
```
[['The Voice', 'English', 7.3], ['Dancing with the Stars', 'English', 4.7]]
```

In [31]:
```
with open('tv-shows.csv', 'w') as f:
    import csv
    writer = csv.writer(f)
    writer.writerows(data)
```