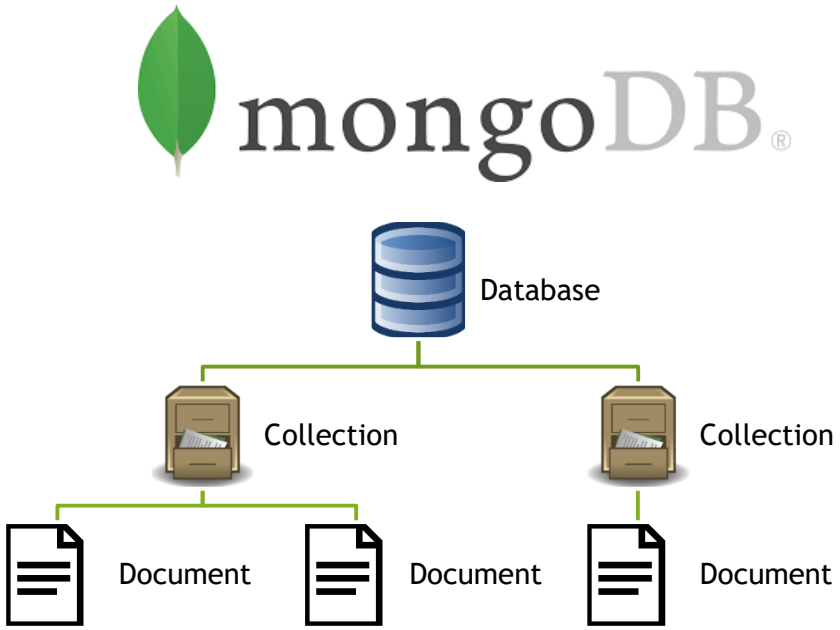
C9d- PyMongo

Using Python to interface with MongoDB

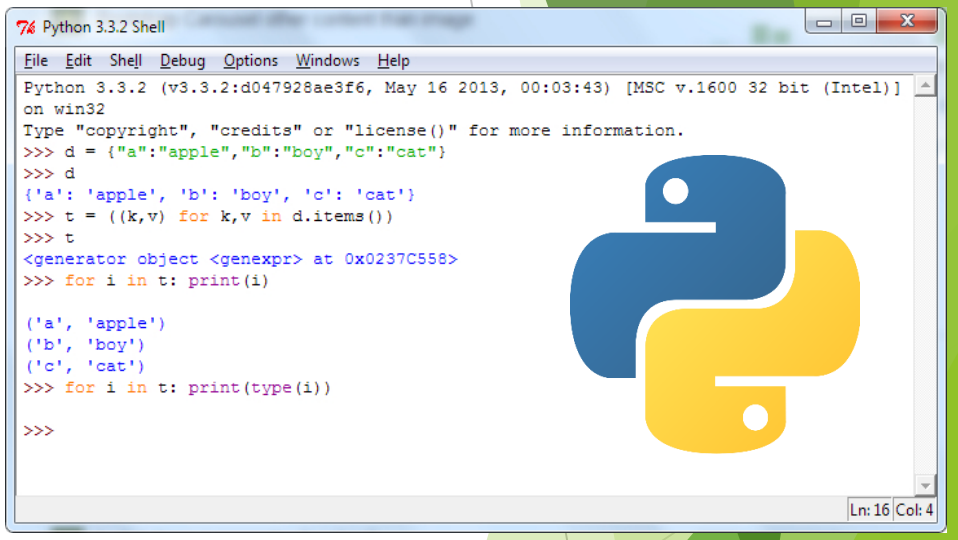
MongoDB: Document-based NoSQL Database



Database Server

PyMongo

Python driver to access MongoDB



Client

Launch MongoDB Server

- Create a running instance of MongoDB server by launching mongod.
- The MongoDB server window should always remain open when accessing MongoDB database.

mongod.exe - Shortcut

```
2020-04-02T01:31:14.245-0700 I CONTROL  [initandlisten] MongoDB starting : pid=16140 port=27017 dbpath=C:\data\db\ 64-bi
t host=N0708JADMW00109
2020-04-02T01:31:14.245-0700 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2020-04-02T01:31:14.246-0700 I CONTROL  [initandlisten] db version v3.4.9
2020-04-02T01:31:14.246-0700 I CONTROL  [initandlisten] git version: 876ebee8c7dd0e2d992f36a848ff4dc50ee6603e
2020-04-02T01:31:14.246-0700 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.1u-fips  22 Sep 2016
2020-04-02T01:31:14.246-0700 I CONTROL  [initandlisten] allocator: tcmalloc
2020-04-02T01:31:14.247-0700 I CONTROL  [initandlisten] modules: none
2020-04-02T01:31:14.247-0700 I CONTROL  [initandlisten] build environment:
2020-04-02T01:31:14.247-0700 I CONTROL  [initandlisten]     distmod: 2008plus-ssl
2020-04-02T01:31:14.247-0700 I CONTROL  [initandlisten]     distarch: x86_64
2020-04-02T01:31:14.248-0700 I CONTROL  [initandlisten]     target_arch: x86_64
2020-04-02T01:31:14.248-0700 I CONTROL  [initandlisten] options: {}
2020-04-02T01:31:14.257-0700 I -        [initandlisten] Detected data files in C:\data\db\ created by the 'wiredTiger' s
torage engine, so setting the active storage engine to 'wiredTiger'.
2020-04-02T01:31:14.259-0700 I STORAGE  [initandlisten] wiredtiger_open config: create,cache_size=3511M,session_max=2000
0,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal
,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),
2020-04-02T01:31:15.126-0700 I CONTROL  [initandlisten]
2020-04-02T01:31:15.127-0700 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-04-02T01:31:15.127-0700 I CONTROL  [initandlisten] **          Read and write access to data and configuration is u
nrestricted.
2020-04-02T01:31:15.128-0700 I CONTROL  [initandlisten]
2020-04-02T16:31:19.304+0800 I FTDC     [initandlisten] Initializing full-time diagnostic data capture with directory 'C
2020-04-02T16:31:19.307+0800 I NETWORK  [thread1] waiting for connections on port 27017
```

0

Connecting to MongoDB

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

db_client_0.py

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)

databases = client.database_names()

print("The databases in the MongoDB server are:")

print(databases)

client.close()
```

create client for MongoDB instance

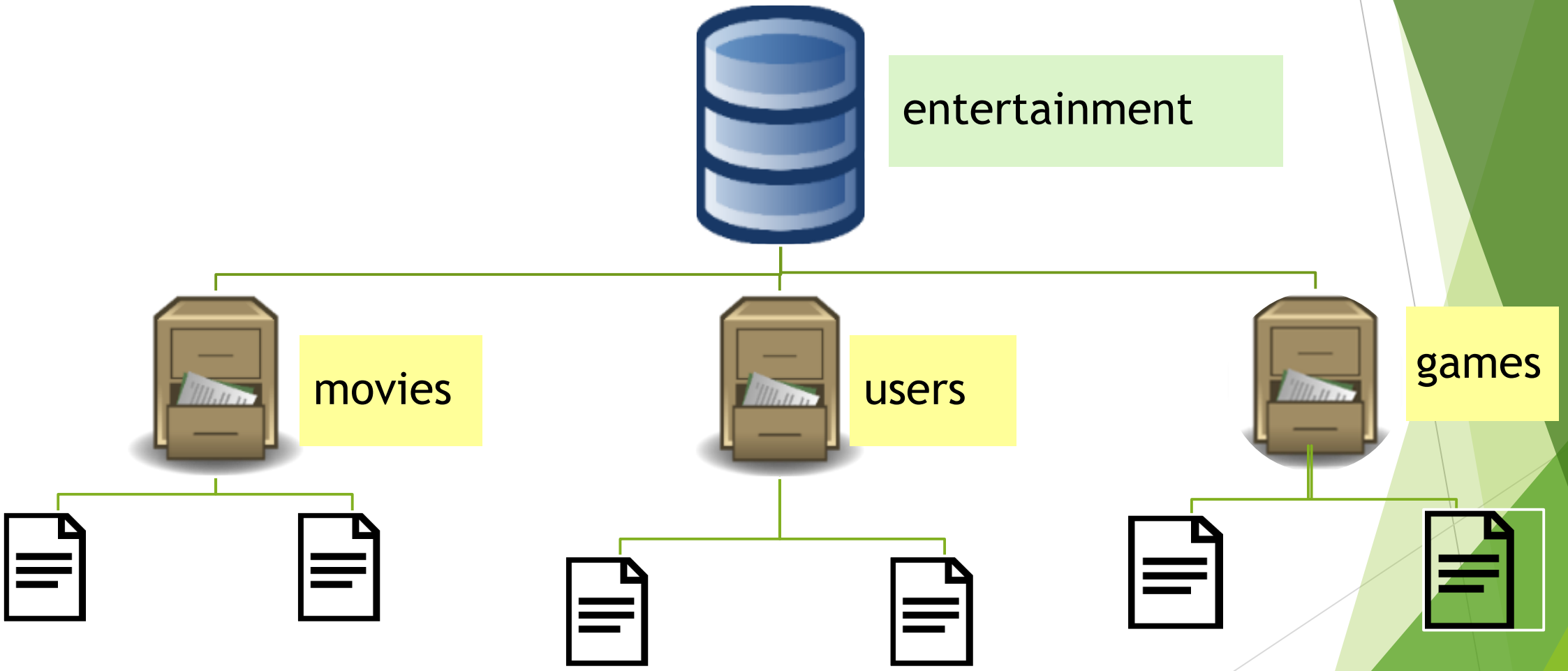
connects to the local MongoDB database which has uri “127.0.0.1” and port num 27017

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

close and disconnects from MongoDB database

```
The databases in the mongoDB server are:
['admin', 'class_info', 'local', 'petshop'
, 'pokemon', 'restaurant', 'social_media',
'test', 'testing']
```

Entertainment database: Using pymongo



Create/access database using pymongo

- 1. Using attribute style access
- Syntax: db = client.DATABASE_NAME

- 2. Using dictionary style access
- Syntax: db = client[“DATABASE_NAME”]

- 3. Using client method
- Syntax: db = client.get_database(“DATABASE_NAME”)

1

Access a database

The program connects to the MongoDB server and access a database in the MongoDB server. If the database specified is not there, it will be created.

db_client_1.py

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)

db = client.get_database("entertainment")

databases = client.database_names()
print("The databases in the MongoDB server are:")
print(databases)

client.close()
```

use <database>

access/create the database named `entertainment` using the client method

show dbs

Create/access database using pymongo

```
databases = client.database_names()
print("The databases in the MongoDB server are:")
print(databases)
```

show dbs

Output:

```
The databases in the mongoDB server are:
['admin', 'class_info', 'local', 'petshop'
, 'pokemon', 'restaurant', 'social_media',
'test', 'testing']
```

'entertainment' database not displayed

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.

Create/access collection using pymongo

- 1. Using attribute style access
- Syntax: col = db.COLLECTION_NAME

- 2. Using dictionary style access
- Syntax: col = db[“COLLECTION_NAME”]

- 3. Using database object method
- Syntax: col = db.get_collection(“COLLECTION_NAME”)

2

Access a collection in a database

db_client_2.py

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)

db = client.get_database("entertainment")

col = db.get_collection("movies")

collections = db.collection_names("entertainment")
print("The collections in the entertainment db are:")
print(collections)

client.close()
```

The program creates a `movies` collection in the `entertainment` database

access/create the collection named `movies`

show collections

Create/access collection using pymongo

```
collections = db.collection_names("entertainment")
print("The collections in the entertainment db are:")
print(collections)
```

show collections

Output:

```
The collections in the db are:
[]
```

'movies' collection not displayed

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.

Insert documents using pymongo

- 1. insert_one()
- Syntax: col.insert_one(document)

- 2. insert_many ()
- Syntax: col.insert_many(ARRAY:document)

3

Insert a document into a collection

db_client_3.py

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("entertainment")
col = db.get_collection("movies")

col.insert_one({"title":"Star Wars","genre":"Sci-fi"})
databases = client.database_names()
print("The databases in the MongoDB server are:")
print(databases)
collections = db.collection_names("entertainment")
print("The collections in the entertainment db are:")
print(collections)
client.close()
```

The program inserts a document in the `movies` collection

insert one document

show dbs

show collections

Insert a document into a collection

Output:

'entertainment' database displayed

```
The databases in the mongoDB server are:
['admin', 'class_info', 'entertainment', 'local',
'petshop', 'pokemon', 'restaurant', 'social_media'
, 'test', 'testing']
The collections in the db are:
['movies']
```

'movies' collection displayed

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.

Insert a document into a collection

db_client_4.py

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("entertainment")
col = db.get_collection("movies")

movie1 = "Titanic"
genre1 = "Romance"
year1= 1997

col.insert_one({"title": movie1 , "genre": genre1 ,\
"year": year1 })

client.close()
```

The program inserts a document in the `movies` collection

Note that the keys of the dictionary must string

here, the values are assigned as variables

Inserting many documents into a collection

The program inserts many documents in the `movies` collection

db_client_5.py

```
(codes to connect to db and access movies collection)

list_to_add=[]

list_to_add.append({"title": "Inception",\
"genre": "Thriller", "year": 2010})
list_to_add.append({"title":"Avengers", \
"genre": "Action", "year": 2012})
list_to_add.append({"title": "Jaws", "genre": "Horror",\
"year": 1975})

col.insert_many(list_to_add)

client.close()
```

insert many documents, takes in an array of documents

Display one document in a collection

The find_one() method returns only the first occurrence of a query.

By leaving the argument empty, it returns the first document in the collection.

- 1. Use find_one() method on the collection
- first=col.find_one ()

- 2. Print the document
- print(first)

Displaying just one document in a collection

db_client_6.py

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("entertainment")
col = db.get_collection("movies")

cursor=col.find_one()

print(cursor)

client.close()
```

The program displays only the first occurrence documents in the `movies` collection

creates a cursor object from the query

displays the first document in the collection

Output:

```
{'_id': ObjectId('5ea3e59b6d511f3824c5cab8'),
'title': 'Star Wars', 'genre': 'Sci-fi'}
```

Display all documents in a collection

The find( ) method returns a pymongo cursor object, which is a reference to the result set of a query.

By leaving the argument empty, it returns all document in the collection.

- 1. Create a cursor object using .find()
- cursor=col.find ()
- 2. Iterate through the cursor object to display the documents
- for doc in cursor:

  print(doc)

Displaying all documents in a collection

db_client_7.py

The program displays all documents in the `movies` collection

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("entertainment")
col = db.get_collection("movies")

cursor=col.find()

for doc in cursor:
   print(doc)

client.close()
```

creates a cursor object from the query

displays every document from the query

Displaying documents in a collection

Output:

```
{'_id': ObjectId('5ea3e59b6d511f3824c5cab8'),
'title': 'Star Wars', 'genre': 'Sci-fi'}
{'_id': ObjectId('5ea3e59b6d511f3824c5cab9'),
'title': 'Titanic', 'genre': 'Romance', 'year'
: 1997}
{'_id': ObjectId('5ea3e59b6d511f3824c5caba'),
'title': 'Inception', 'genre': 'Thriller', 'ye
ar': 2010}
{'_id': ObjectId('5ea3e59b6d511f3824c5cabb'),
'title': 'Avengers', 'genre': 'Action', 'year'
: 2012}
{'_id': ObjectId('5ea3e59b6d511f3824c5cabc'),
'title': 'Jaws', 'genre': 'Horror', 'year': 19
75}
```

Exercise 1

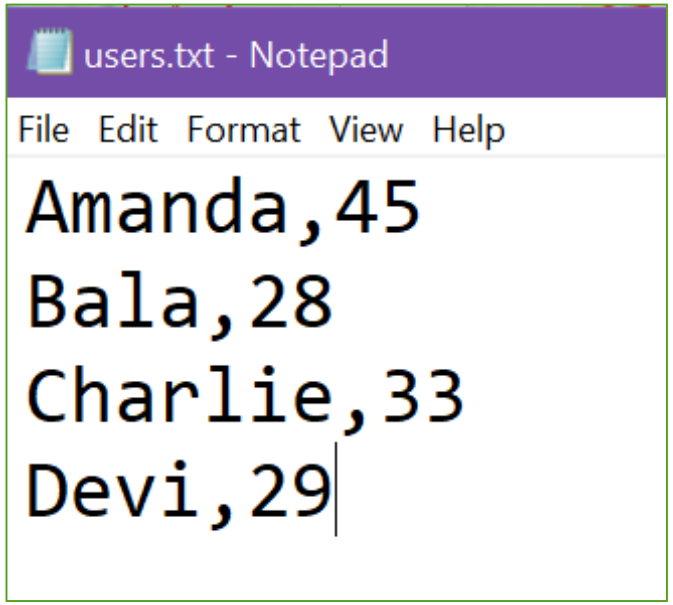
Using the entertainment db created,

- Write a python program (save the file as exercise1.py) that does the following:
  - prompts the user to input a movie title
  - prompts the user to input the year of movie
  - insert the movie data as a document in the movies collection
  - display all documents in the movies collection

Insert documents by importing data

- 1. importing data from a .txt or .csv file, using csv library
- with open(“input.txt") as csv_file:

```
    data=csv.reader(csv_file, delimiter= ‘,’)
    for row in data:
        col.insert_one({‘field1’:row[0], ‘field2’:row[1],…})
```



Importing data from .txt or .csv file

db_client_8.py

```
import pymongo
import csv

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

with open("users.txt") as csv_file:
     data=csv.reader(csv_file, delimiter= ",")
     for row in data:
        col.insert_one({"name":row[0], "age":row[1]})

(codes to display all documents in users collection)

client.close()
```

The program inserts data from "users.txt" into as documents in the `users` collection

creates a new collection, users

Displaying documents in users collection

Output:

```
{'_id': ObjectId('5ea44d466d511f6618249bf2'),
'name': 'Amanda', 'age': '45'}
{'_id': ObjectId('5ea44d466d511f6618249bf3'),
'name': 'Bala', 'age': '28'}
{'_id': ObjectId('5ea44d466d511f6618249bf4'),
'name': 'Charlie', 'age': '33'}
{'_id': ObjectId('5ea44d466d511f6618249bf5'),
'name': 'Devi', 'age': '29'}
```

Insert documents by importing data

- 2. importing data from a json file using json library
- with open(“input.json") as json_file:

```
data=json.load(json_file)
col.insert_many(data)
```

The json file here consist of an array of documents

games.json - Notepad

File Edit Format View Help

```
[
        {
        "title": "CS:GO",
        "genre": "FPS"
        },
        {
        "title": "PUBG",
        "genre": "Battle Royale"
        },
        {
        "title": "MapleStory",
        "genre": "MMORPG"
        },
```

Importing data from .json

db_client_9.py

```
import pymongo
import json

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("entertainment")
col = db.get_collection("games")

with open("games.json") as json_file:
     data=json.load(json_file)
     col.insert_many(data)

(codes to display all documents in games collection)

client.close()
```

The program inserts data from "games.json" into as documents in the `games` collection

creates a new collection, games

Displaying documents in games collection

Output:

```
{'_id': ObjectId('5ea50d666d511f4fbc3fd60e'),
'title': 'CS:GO', 'genre': 'FPS'}
{'_id': ObjectId('5ea50d666d511f4fbc3fd60f'),
'title': 'PUBG', 'genre': 'Battle Royale'}
{'_id': ObjectId('5ea50d666d511f4fbc3fd610'),
'title': 'MapleStory', 'genre': 'MMORPG'}
{'_id': ObjectId('5ea51f3b0bd389adac5f7be4'),
'title': 'Starcraft', 'genre': 'RTS'}
```

Exercise 2

Using the entertainment db created,

- Write a python program (save it as exercise2.py) that does the following:
- Creates a menu to have the following selections: (sample template on next slide)
  - 1. insert one movie (already done in exercise 1)
  - 2. insert movies using text file
  - 3. insert movies using json file
  - 4. display all movies
  - 5. quit
- Write codes for all 5 options. For options 2 and 3, prompt the users for the file name.
- use the files “movies.txt” and “movies.json” to test your menu option 2 and 3

exercise2.py

```
(codes to connect to db and access movies collection)

while True:
   print("-------MENU------------")
   print("Please choose an option")
   print("1. insert one movie")
   print("2. insert movies using text file")
   print("3. insert movies using json file")
   print("4. display all movies")
   print("5. quit")
   user_option=input("Your option:")
   if user_option == "1":
        <your code>
   elif user_option == "2":
        <your code>
   etc.
```

Querying documents in a collection

The find( ) method returns a pymongo cursor object, which is a reference to the result set of a query.

Just like in MongoDB, the criteria of query can be passed into find().

- Create a cursor object using .find(), with query criteria specified.
- query=col.find ({ }) ##returns all documents
- query=col.find (criteria) ##returns all documents that fulfil criteria
- query=col.find (criteria,field_display) ##returns all doc that fulfil criteria, displaying only certain fields
- Apply query methods (e.g. count(), sort(), limit(), etc.) on cursor object.
- query.count()

Query documents in a collection

The program displays all documents from the queries

db_client_10.py

```
(codes to connect to db and access movies collection)
field_display= {'_id':0,'title':1,'genre':1,'year':1}
query1=col.find({"genre":"Sci-fi"},field_display})
print("All Sci-fi movies:")
for doc in query1:
   print(doc)
print("There are ",query1.count(),"  Sci-fi movies")
print('\n\n\n')
criteria={"$and":[{"genre": "Sci-fi"},{"year":{"$gt":2015}}]}
query2=col.find(criteria,field_display)
print("All Sci-fi movies later than 2015:")
for doc in query2:
   print(doc)
client.close()
```

Displaying documents from query

Output:

```
All Sci-fi movies:
{'title': 'Star Wars', 'genre': 'Sci-fi'}
{'title': 'X-Men', 'genre': ['Action', 'Sci-fi'], 'year': 2000}
{'title': 'Men in Black', 'genre': ['Action', 'Sci-fi', 'Comedy'], 'year'
: 2002}
{'title': 'The Girl Who Leapt Through Time', 'genre': ['Anime', 'Sci-fi']
, 'year': 2006}
{'title': 'Tron: Legacy', 'genre': ['Action', 'Sci-fi'], 'year': 2010}
{'title': 'Source Code', 'genre': ['Thriller', 'Sci-fi'], 'year': 2011}
{'title': 'The Thing', 'genre': ['Horror', 'Sci-fi'], 'year': 2011}
{'title': 'World War Z', 'genre': ['Horror', 'Sci-fi'], 'year': 2013}
{'title': 'Jurassic World: Fallen Kingdom', 'genre': ['Action', 'Sci-fi']
, 'year': 2018}
There are  9  Sci-fi movies
```

```
All Sci-fi movies later than 2015:
{'title': 'Jurassic World: Fallen Kingdom', 'genre': ['Action', 'Sci-fi'],
'year': 2018}
```

Getters for documents objects

The get( ) method returns the value of the associated key (field name) in a dictionary object.

- 1. Using dictionary style access
- Syntax: doc[“FIELD_NAME”]
- 2. Using dictionary object method
- Syntax: doc.get(“FIELD_NAME”)

Query documents in a collection

The program displays all documents from the queries

db_client_11.py

```
(codes to connect to db and access movies collection)

criteria={"$and":[{"genre": "Horror"},\
{"year":{"$lt":2000}}]}
display_fields={'_id':0,'title':1,'genre':1,'year':1}
query3=col.find(criteria,display_fields)
print("All horror movies before 2000:")

for doc in query3:
   print("Title:", doc.get("title"))
   print("Genre:", doc.get("genre"))
   print("Year:", doc.get("year"))

print("There are "+ str(query3.count()) + "movies.")
client.close()
```

Displaying documents from query

Output:

```
All horror movies before 2000:
Title: Jaws
Genre: Horror
Year: 1975
Title: Friday the 13th
Genre: ['Thriller', 'Horror']
Year: 1980
Title: A Nightmare on Elm Street
Genre: ['Horror', 'Thriller']
Year: 1984
Title: Sleepy Hollow
Genre: ['Horror', 'Thriller']
Year: 1999
There are 4 movies.
```

Exercise 3

Using the entertainment db created,

- Write a python program (save it as exercise3.py) that does the following:
- Creates a menu to have the following selection:
  - 1. find movie based on title
  - 2. find movie based on genre
  - 3. find movie based on year
  - 4. display all movies in ascending order of title
  - 5. quit
- Write codes for all 5 options. Options 1-3 should prompt user for a value (title/genre/year). For option 2, allow users to choose presence or absence of genre. For option 3, allow users to choose greater than, equals to, or less than
- Options 1-4: display only the title, genre and year of the movies

Update documents using pymongo

- 1. update_one(): updates only the first occurrence of a query.
- Syntax: col.update_one(criteria, update)

- 2. update_many (): updates all occurrences of a query.
- Syntax: col. update_many(criteria, update)

Update documents in a collection

The program updates documents in a collection.

db_client_12.py

```
(codes to connect to db and access movies collection)

search1={"year":{"$exists":False}}
update1={"$set":{"year":"pending"}}
col.update_many(search1,update1)

search2={"year":{"$eq":1997}}
update2={"$unset":{"genre":""}}
col.update_many(search2,update2)

(codes to display all documents in movies collection)

client.close()
```

update/insert field using $set

remove the genre field using $unset

Displaying documents from query

Output:

```
{'_id': ObjectId('5ea3e59b6d511f3824c5cab8'), 'title': 'Star Wars
', 'genre': 'Sci-fi', 'year': 'pending'}
{'_id': ObjectId('5ea3e59b6d511f3824c5cab9'), 'title': 'Titanic',
'year': 1997}
{'_id': ObjectId('5ea3e59b6d511f3824c5caba'), 'title': 'Inception
', 'genre': 'Thriller', 'year': 2010}
{'_id': ObjectId('5ea3e59b6d511f3824c5cabb'), 'title': 'Avengers'
, 'genre': 'Action', 'year': 2012}
{'_id': ObjectId('5ea3e59b6d511f3824c5cabc'), 'title': 'Jaws', 'g
```

Delete documents using pymongo

- 1. delete_one(): deletes only the first occurrence of a query.
- Syntax: col.delete_one(criteria)

- 2. delete_many (): deletes all occurrences of a query.
- Syntax: col. delete_many(criteria)

13

Delete documents in a collection

The program deletes documents in a collection.

db_client_13.py

```
(codes to connect to db and access movies collection)

print("There are ", col.count(), "movies.") ##25

search1={"year":2010}
col.delete_one(search1)

search2={"genre":"Romance"}
col.delete_many(search2)

print("Now, there are ", col.count(), "movies.")

client.close()
```

delete only the first occurrence with year: 2010

remove all documents with field containing Romance

count the total number of documents in the movies collection

Output:

```
Now, there are  22 movies.
```

Exercise 4

Using the entertainment db created,

- Write a python program (save it as exercise4.py) that does the following:
- Creates a menu to have the following selection:
  - 1. update movie
  - 2. delete movie
  - 3. display all movies in ascending order of title
  - 4. quit
- Write codes for all 4 options. Options 1-2 should prompt user for a selection criteria via (title/genre/year). Allow users to choose presence or absence of genre. Allow users to choose year greater than, equals to, or less than (already done in exercise 3). Option 1 should prompt user for field to update and the update value.
- Options 3: display only the title, genre and year of the movies

Drop database/collection using pymongo

- 1. Dropping collection
- Syntax: db.drop_collection(“COLLECTION_NAME”)

- 2. Dropping database
- Syntax: client.drop_database(“DATABASE_NAME”)

14

Delete documents in a collection

The program drops collection and database.

db_client_14.py

```
(codes to connect to db and access movies collection)

db.drop_collection("movies")

(codes to show collections in a db)

client.drop_database("entertainment")

(codes to show dbs)

client.close()
```

Insert Booklet- pymongo module

pymongo module	
MongoClient()	<collection>.update_one()
<client>.database_names()	<collection>.update_many()
<client>.get_database()	<collection>.delete_one()
<client>.drop_database()	<collection>.delete_many()
<client>.close()	<collection>.count()
<database>.collection_names()	<cursor>.count()
<database>.get_collection()	
<database>.drop_collection()	
<collection>.insert_one()	
<collection>.insert_many()	
<collection>.find_one()	
<collection>.find()	

Insert Booklet-pymongo operators

4 PyMongo Operators

Comparison

$eq	$gt	$gte	$lt	$lte
$ne	$in	$nin		

Logical

$and	$not	$or

Element

$exists

Update

$set	$unset

Resources

- https://pymongo.readthedocs.io/en/stable/(official documentation for pymongo)
- https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb (pymongo)
- http://zetcode.com/python/pymongo/ (pymongo tutorial)