# MongoDB in Python

- MongoDB is a popular unstructured database (NoSQL database).
- This guide is to get started with MongoDB using Python.
- MongoDB is one of the best suited for managing big data.

## Major Disadvantages of Structured Databases

- Scalability: It is very difficult to scale as the database grows larger.
- Elasticity: Structured databases need data in a predefined format. If the data is not following the predefined format, relational databases do not store it.

## What is NoSQL database?

- NoSQL, also referred to as “not only SQL” or “non-SQL,” is an approach to database design that enables the storage and querying of data outside the traditional structures found in relational databases.
- So, NoSQL databases are different than relational (structured) databases like MQSql.
- Instead of the typical tabular structure of a relational database, NoSQL databases house data within one data structure, such as JSON document. Since this non-relational database design does not require a schema, it offers rapid scalability to manage large and typically unstructured data sets.
- Thus, one of the advantage of NoSQL database is that they are really easy to scale and they are much faster in most types of operations that we perform on database.
- There are certain situations where you would prefer relational database over NoSQL, however when you are dealing with huge amount of data then NoSQL database is your best choice.

## What is MongoDB?

- MongoDB is an unstructured database/NoSQL database.
- MongoDB is an open source, document oriented database that stores data in form of documents (key and value pairs).
- MongoDB stores data in schemaless and flexible `JSON` -like documents. Here, *schemaless* means that you can have documents with a different set of fields in the same collection, without the need for satisfying a rigid table schema.
- MongoDB stores data in `JSON` -like documents, which makes the database very flexible and scalable.
- That means, MongoDB is able to handle huge volumes of data very efficiently and is the most widely used NoSQL database as it offers rich query language and flexible and fast access to data.

## Architecture of MongoDB Database

- The information in MongoDB is stored in *documents*. Here, a *document* is analogous to *rows* in structured databases.
  - Each document is a collection of key-value pairs
  - Each key-value pair is called a *field*
  - Every document has an `_id` field, which uniquely identifies the documents
  - A document may also contain nested documents
  - Documents may have a varying number of fields (they can be blank as well)
- These documents are stored in a *collection*. A *collection* is literally a collection of documents in MongoDB. This is analogous to *tables* in traditional databases.

## Install MongoDB on Windows

1. Go to MongoDB download page
2. Download the installer (under MongoDB Community Server)
3. Run the installer and follow the on-screen instructions on the installation wizard.

- Detailed installation instructions are found at the official MongoDB documentation.

## Install MongoDB on Mac using Homebrew

- Installing MongoDB on Mac can be done using *The Official MongoDB Software Homebrew Tap* (a custom Homebrew tap for official MongoDB software).
- Detailed installation informaion are found at The Official MongoDB Software Homebrew Tap.

## What is PyMongo?

- Python needs a MongoDB driver to access the MongoDB database.
- MongoDB provides an official Python driver called **PyMongo**.
- In general, PyMongo provides a rich set of tools that you can use to communicate with a MongoDB server. It provides functionality to query, retrieve results, write and delete data, and run database commands.

## Install PyMongo

- To start using PyMongo, you first need to install it in your Python environment. To install PyMongo,
  - you can use a virtual environment, (such as, `$ pipenv install pymongo` )or
  - you can use your system-wide Python installation, or
  - you can use the quickest way to install it is with `pip` ( `$ pip install pymongo` ).

## Establish Connection in MongoDB

- To retrieve the data from a MongoDB database, we will first connect to it.
- The first thing we need to do is `import pymongo` . The `import` should run without any errors to signify that we've done our installation well.

In [24]:
```
import pymongo
```

- To establish a connection to a database, you need to create a MongoClient instance. This class provides a client for a MongoDB instance or server.

In [25]:
```python
from pymongo import MongoClient
client = MongoClient()

# Let's print the `client`
print(client)
```

```
MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True)
```

- The code above establishes a connection to the default host ( `localhost` ) and port ( `27017` ).
- `MongoClient` takes a set of arguments that allows you to specify custom host, port, and other connection parameters.

In [26]:
```python
# For example, to provide a custom host and port:
client = MongoClient(host="localhost", port=27017)
```

- This is handy when you need to provide a `host` and `port` that differ from MongoDB's default setup.
- We can also use the MongoDB URI format to do the same.

In [27]:
```python
client = MongoClient("mongodb://localhost:27017")
```

- All these instances of `MongoClient` provide the same client setup to connect your current MongoDB instance.

## Create Database

- To create a database in MongoDB, we use the MongoClient instance and specify a database name. MongoDB will create a database if it doesn't exist and connect to it.
- It is important to note that databases and collections are created lazily in MongoDB. This means, MongoDB waits until we have created a collection (table), with at least one document (record) before it actually creates the database (and collection).
- So, in MongoDB, the collections and databases are created when the first document is inserted into them.

In [28]:
```
# Create a database named "blogdb"
# Assign the database into a variable
db_blog = client['blogdb']
```

- Note that, when we use PyMongo, we can assign the database to a variable (in this case it is `db_blog`).

In [29]:
```
# let's just print the database object
print(db_blog)
```

```
Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'blogdb')
```

In [30]:
```
# Return a list of database names
print(client.list_database_names())
```

```
['admin', 'config', 'local']
```

## Create Collection (Table)

- A collection in MongoDB is the same as a table in SQL databases.
- To create a collection in MongoDB, use database object and specify the name of the collection you want to create.
- MongoDB will create the collection if it does not exist.
- However, remember that in MongoDB, a collection is not created until it gets content! That means, MongoDB waits until you have inserted a document before it actually creates the collection.

In [31]:
```
# Create a collection (table)
# Assign the collection into a variable
coll_posts = db_blog["posts"]
```

In [32]:
```
# let's just print the collection object
print(coll_posts)
```

```
Collection(Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'blo
gdb'), 'posts')
```

In [33]:
```
# Return a list of collection names
print(db_blog.list_collection_names())
```

```
[]
```

## Insert a Single Document (Record)

- A *document* in MongoDB is the same as a *record* or a *row* of a table in SQL databases.
- To insert a record, or document into a collection, we use the `insert_one()` method.
- Remember that data in MongoDB is represented and stored using JSON-Style documents. In PyMongo we use dictionaries to represent documents.

In [34]:

```
# First create a document
post_1 = {
    "author": "Jack",
    "title": "Introduction to MongoDB with Python",
    "tags": ["mongodb", "python", "pymongo"]
}
```

In [35]:

```
# Now, insert the document into collection
insert_one_result = coll_posts.insert_one(post_1)
```

- Here, `insert_one()` takes the document and insert into the collection and returns an `InsertOneResult` object. This object provides information on the inserted document.
- Remember that when the document is inserted, a special key `_id` is generated and its unique to this document.
- The `InsertOneResult` object has a property, `inserted_id`, that holds the `_id` of the inserted document.

In [36]:

```
# Let's print the id of first document (post_1)
print(f"First document ID: {insert_one_result.inserted_id}")
```

```
First document ID: 6103cc8ebbd79911caae033f
```

- As we have one document inserted in the collection of the database, now let's check the list of database names, and the list of collection names currently available.

In [37]:

```
# Return a list of database names
print(client.list_database_names())
```

```
['admin', 'blogdb', 'config', 'local']
```

In [38]:

```
# Return a list of collection names
print(db_blog.list_collection_names())
```

```
['posts']
```

- Now, we can see that our database `blogdb` and collection `posts` are created.
- That means, database and collected are created after inserting the first document.

## Insert Multiple Documents

- To insert multiple documents into a collection in MongoDB, we use the `insert_many()` method.

In [39]:
```
# Create second document
post_2 = {
    "author": "Jill",
    "title": "Data visualization in Python",
    "tags": ["matplotlib", "python", "seaborn"]
}

# Create third document
post_3 = {
    "author": "Masha",
    "title": "Webapp development in Python",
    "tags": ["django", "python", "flask", "framework"]
}
```

In [40]:
```
# Now, insert multiple document into collection
insert_many_result = coll_posts.insert_many([post_2, post_3])
```

- This is faster and more straightforward than calling `insert_one()` multiple times.
- The call to `insert_many()` takes an iterable of documents and inserts them into the `posts` collection in our `blogdb` database.
- The method returns an instance of `InsertManyResult`, which provides information on the inserted documents.
- The `InsertManyResult` object has a property, `inserted_ids`, that holds the `ids` of the inserted documents.

In [41]:
```python
# Let's print the ids of the inserted documents (post_2, and post_3)
print(f"Second and third documents IDs: {insert_many_result.inserted_ids}")
```

```
Second and third documents IDs: [ObjectId('6103cc8ebbd79911caae0340'), ObjectId('6103cc8ebbd79911caae0341')]
```

- However, if you want you can also specify the `_id` field when you insert the document.
- Remember that the values has to be unique. Two documents cannot have the same `_id`.

In [42]:
```python
# Create fourth document
post_4 = {
    "_id": "44a33b55c77d99e13f444",
    "author": "Angela",
    "title": "A discussion on SQL vs NoSQL Database",
    "tags": ["sql", "nosql", "database"]
}

# Create fifth document
post_5 = {
    "_id": "55a44b66c88d24e68f555",
    "author": "Jorge",
    "title": "Essential tools for Data Science in Python",
    "tags": ["pyhton", "numpy", "pandas", "matplotlib", "seaborn"]
}
```

In [43]:
```python
# Now, insert multiple document into collection with specified IDs
insert_many_result_with_ids = coll_posts.insert_many([post_4, post_5])
```

In [44]:
```python
# Let's print the ids of the inserted documents (post_4, and post_5)
print(f"Fourth and fifth documents IDs: {insert_many_result_with_ids.inserted_ids}")
```

```
Fourth and fifth documents IDs: ['44a33b55c77d99e13f444', '55a44b66c88d24e68f555']
```

## Retrieve a Single Document

- To select a single document from a collection in MongoDB, we can use the `find_one()` method.
- The `find_one()` method returns the first occurrence that it comes across.

In [45]:
```python
# Return the first we inserted into our collection
print(coll_posts.find_one())
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'), 'author': 'Jack', 'title': 'Introduction to MongoDB with Python', 't
ags': ['mongodb', 'python', 'pymongo']}
```

## Retrieve All Document

- To retrieve all documents in a collection in MongoDB, we can use the `find()` method.
- The `find()` method without parameter returns all documents in the collection.
- No parameters in the `find()` method gives you the same result as `SELECT *` in SQL.

In [46]:
```python
# Retrieve all documents in our collection
for post in coll_posts.find():
    print(post)
    print("-------")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'), 'author': 'Jack', 'title': 'Introduction to MongoDB with Python', 't
ags': ['mongodb', 'python', 'pymongo']}
------
{'_id': ObjectId('6103cc8ebbd79911caae0340'), 'author': 'Jill', 'title': 'Data visualization in Python', 'tags': [
'matplotlib', 'python', 'seaborn']}
------
{'_id': ObjectId('6103cc8ebbd79911caae0341'), 'author': 'Masha', 'title': 'Webapp development in Python', 'tags':
['django', 'python', 'flask', 'framework']}
------
{'_id': '44a33b55c77d99e13f444', 'author': 'Angela', 'title': 'A discussion on SQL vs NoSQL Database', 'tags': ['s
ql', 'nosql', 'database']}
------
{'_id': '55a44b66c88d24e68f555', 'author': 'Jorge', 'title': 'Essential tools for Data Science in Python', 'tags':
['pyhton', 'numpy', 'pandas', 'matplotlib', 'seaborn']}
------
```

In [47]:

```python
# Print document using "pprint.pprint()", provide a user-friendly output format.
import pprint
for post in coll_posts.find():
    pprint.pprint(post)
    print("------")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
------
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
------
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python'}
------
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
------
{'_id': '55a44b66c88d24e68f555',
 'author': 'Jorge',
 'tags': ['pyhton', 'numpy', 'pandas', 'matplotlib', 'seaborn'],
 'title': 'Essential tools for Data Science in Python'}
------
```

## Retrieve Document with Desired Fields

- The `find()` method without parameter returns all fields of the document by default. But by providing `projection` parameter in `find()` method we can retrieve document with our desired fields as well.
- The `projection` is a second parameter of the `find()` method.
- The `projection` parameter is an object describing which fields to include in the result, or which field to exclude from the result.
- To include specify the field with `1`, or to exclude specify the field with `0` in the `projection` parameter.
- Note that the `projection` parameter contains either include or exclude specifications, not both, unless the exclude is for the `_id` field (because `_id` field is returned by default if it is not explicitly specified with `0`).

In [48]:

```
# We want to retrieve 'author' and `title` fields of the document
# But it will return `_id` field also (by default).
for post in coll_posts.find({}, {"author": 1, "title": 1}):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'title': 'Data visualization in Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'title': 'Webapp development in Python'}
-----
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
{'_id': '55a44b66c88d24e68f555',
 'author': 'Jorge',
 'title': 'Essential tools for Data Science in Python'}
-----
```

In [49]:

```
# Returns 'author' and `title` fields of the document
for post in coll_posts.find({}, {"_id": 0, "author": 1, "title": 1}):
    pprint.pprint(post)
```

```
{'author': 'Jack', 'title': 'Introduction to MongoDB with Python'}
{'author': 'Jill', 'title': 'Data visualization in Python'}
{'author': 'Masha', 'title': 'Webapp development in Python'}
{'author': 'Angela', 'title': 'A discussion on SQL vs NoSQL Database'}
{'author': 'Jorge', 'title': 'Essential tools for Data Science in Python'}
```

## Filter the Results

- The first argument of the `find()` method is a query object, and is used to limit the search (i.e., to filter the result).

In [50]:

```
# Return document(s) where "author" is "Jill"
for post in coll_posts.find({"author": "Jill"}):
    pprint.pprint(post)
```

```
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
```

## Filter With Regular Expressions

- We can also use regular expressions to query string.

In [51]:
```
#To find the documents where the "author" field starts with the letter "J"
for post in coll_posts.find({"author": {"$regex": "^J"} }):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
-----
{'_id': '55a44b66c88d24e68f555',
 'author': 'Jorge',
 'tags': ['pyhton', 'numpy', 'pandas', 'matplotlib', 'seaborn'],
 'title': 'Essential tools for Data Science in Python'}
-----
```

In [52]:
```
#To find the documents where the "author" field ends with the letter "k"
for post in coll_posts.find({"author": {"$regex": "a$"} }):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python'}
-----
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
```

In [53]:

```
#To find the documents where the "author" is "Masha" or "Jack"
for post in coll_posts.find({"author": {"$regex": "Masha|Jack"} }):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python'}
-----
```

In [54]:

```
#To find the documents where the "author" names
# contain one of the specified characters [abcd]
for post in coll_posts.find({"author": {"$regex": "[abcd]"} }):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python'}
-----
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
```

In [55]:

```python
#To find the documents where the "author" names
# contains "J" followed by 0 or more "a" characters
for post in coll_posts.find({"author": {"$regex": "Ja*"} }):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
-----
{'_id': '55a44b66c88d24e68f555',
 'author': 'Jorge',
 'tags': ['pyhton', 'numpy', 'pandas', 'matplotlib', 'seaborn'],
 'title': 'Essential tools for Data Science in Python'}
-----
```

```
In [56]:
#To find the documents where the "author" names
# contains "J" followed by 1 or more "a" characters
for post in coll_posts.find({"author": {"$regex": "Ja+"} }):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
```

```
In [57]:
#To find the documents where the "author" names
# start with "M" character followed by two (any) characters, and an "a"
for post in coll_posts.find({"author": {"$regex": "A....a"} }):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
```

## Sort the Result

- We can use the `sort()` method to sort the result in ascending or descending order (default order is ascending).
- The `sort()` method takes one parameter for "fieldname" and one parameter for "direction" (ascending or descending).
- Use `1` to signify ascending and `-1` to signify descending.

```
In [58]:
# Returns all documents in ascending order by "author" names
for post in coll_posts.find().sort("author"):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
-----
{'_id': '55a44b66c88d24e68f555',
 'author': 'Jorge',
 'tags': ['pyhton', 'numpy', 'pandas', 'matplotlib', 'seaborn'],
 'title': 'Essential tools for Data Science in Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python'}
-----
```

In [59]:

```python
# Returns all documents in descending order by "author" names
for post in coll_posts.find().sort("author", -1):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python'}
-----
{'_id': '55a44b66c88d24e68f555',
 'author': 'Jorge',
 'tags': ['pyhton', 'numpy', 'pandas', 'matplotlib', 'seaborn'],
 'title': 'Essential tools for Data Science in Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
```

## Update a Single Document

- We can update a document using the `update_one()` method.
- The first parameter taken by the `update_one()` method is a **query** object defining the document to be updated. If the method finds more than one document, it will only update the first one.
- The second parameter of the `update_one()` method is an object defining the new values of the document.

In [60]:

```python
# Let's update "author" name "Jorge" by "Micky"

# Define the query object
query = { "author": "Jorge" }

# Define the object to set new value
new_author = { "$set": { "author": "Micky" } }

# Perform the update operation
coll_posts.update_one(query, new_author)

# Now see all the documents
for post in coll_posts.find():
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python'}
-----
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
{'_id': '55a44b66c88d24e68f555',
 'author': 'Micky',
 'tags': ['pyhton', 'numpy', 'pandas', 'matplotlib', 'seaborn'],
 'title': 'Essential tools for Data Science in Python'}
-----
```

## Update Multiple Document

- We can use the `update_many()` method to update all documents that meets the criteria of the query.

In [61]:

```
# Let's update all documents where "author" names start with "M"

# Set the query object
query = { "author": { "$regex": "^M" } }

# Define the object to set new value
new_field = { "$set": { "type": "Cartoon" } }

# Perform the update operation
coll_posts.update_many(query, new_field)

# Now see all the documents
for post in coll_posts.find():
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python',
 'type': 'Cartoon'}
-----
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
{'_id': '55a44b66c88d24e68f555',
 'author': 'Micky',
 'tags': ['pyhton', 'numpy', 'pandas', 'matplotlib', 'seaborn'],
 'title': 'Essential tools for Data Science in Python',
 'type': 'Cartoon'}
-----
```

## Limit the Result

- We use the `limit()` method to limit the result in MongoDB.
- The `limit()` method takes one parameter, a number defining how many documents to return.

In [65]:

```python
# Let's return 3 documents from the collection
for post in coll_posts.find().limit(3):
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python',
 'type': 'Cartoon'}
-----
```

## Delete a Single Document

- We use the `delete_one()` method in MongoDB to delete a single document from a collection.
- The first parameter of the `delete_one()` method is a query object defining which document to delete.
- If this method finds more than one document, it deletes only the first one found.

In [66]:
```python
# Let's delete the document with the "author" name is "Micky"
coll_posts.delete_one({"author": "Micky"})

# Now see all the documents currently available in the collection
for post in coll_posts.find():
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae033f'),
 'author': 'Jack',
 'tags': ['mongodb', 'python', 'pymongo'],
 'title': 'Introduction to MongoDB with Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0340'),
 'author': 'Jill',
 'tags': ['matplotlib', 'python', 'seaborn'],
 'title': 'Data visualization in Python'}
-----
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python',
 'type': 'Cartoon'}
-----
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
```

## Delete Multiple Documents

- We can use the `delete_many()` method in MongoDB to delete more than one document.
- The first parameter of the `delete_many()` method is a query object defining which documents to delete.

In [67]:
```python
# Let's delete all the document were "author" name start with "J"
coll_posts.delete_many({ "author": {"$regex": "^J"} })

# Now see all the documents currently available in the collection
for post in coll_posts.find():
    pprint.pprint(post)
    print("-----")
```

```
{'_id': ObjectId('6103cc8ebbd79911caae0341'),
 'author': 'Masha',
 'tags': ['django', 'python', 'flask', 'framework'],
 'title': 'Webapp development in Python',
 'type': 'Cartoon'}
-----
{'_id': '44a33b55c77d99e13f444',
 'author': 'Angela',
 'tags': ['sql', 'nosql', 'database'],
 'title': 'A discussion on SQL vs NoSQL Database'}
-----
```

## Delete All Documents

- We can use `delete_many()` method with an empty query string to delete all documents in a collection.

In [69]:

```python
# Let's delete all the document in the collection
coll_posts.delete_many({})

# Now see all the documents currently available in the collection
for post in coll_posts.find():
    pprint.pprint(post)
    print("-----")
else:
    print("Empty Collection")
```

```
Empty Collection
```

## Delete Collection

- We can use `drop()` method to delete a collection in MongoDB.

In [70]:

```python
# Delete the "posts" collection.
coll_posts.drop()
```

In [71]:
```
# Let's confirm that the collection has been deleted
db_blog.list_collection_names()
```
Out[71]:
```
[]
```

## Delete Database

- We can use `drop_database()` method to delete a database from MongoDB.

In [73]:
```
# Delete the "blogdb" database
client.drop_database("blogdb")
```

In [74]:
```
# Let's confirm that the database has been deleted
client.list_database_names()
```
Out[74]:
```
['admin', 'config', 'local']
```

---

Course: Programming in Python
Faculty: **Dr Akinul Islam Jony**, Assistant Professor of Computer Science, AIUB

---