

PRACTICAL VIII

DATE: 21/02/24

MongoDB with python

AIM: To learn and execute MongoDB with python.

INTRODUCTION:

Python, the top programming language for data science, and MongoDB, with its flexible and dynamic schema, are a fantastic match for building modern web applications, JSON APIs, and data processors, just to name a few. MongoDB has a native Python driver and a team of engineers dedicated to making sure MongoDB and Python work together flawlessly.

`PyMongo` is a Python driver for MongoDB, a popular NoSQL database. MongoDB is a document-oriented database that stores data in flexible, JSON-like BSON (Binary JSON) format. PyMongo allows Python applications to interact with MongoDB, enabling developers to perform various database operations using Python code.

Database Connectivity:

PyMongo facilitates the connection between Python applications and MongoDB databases. It provides a simple and efficient way to interact with MongoDB, allowing you to perform CRUD (Create, Read, Update, Delete) operations on the data.

Flexibility and Schema-less Design:

MongoDB is a schema-less database, meaning that documents in a collection do not need to have the same structure. This flexibility is advantageous in scenarios where the data model evolves over time. PyMongo supports this flexibility and allows developers to work with dynamic data structures.

Performance:

PyMongo is designed to be efficient and provides a native interface to interact with MongoDB, taking advantage of its features like indexing and sharding. This helps in building high-performance applications, especially when dealing with large datasets.

Scalability:

MongoDB is known for its horizontal scalability, and PyMongo enables developers to harness this scalability by distributing data across multiple nodes or servers. This is particularly useful in applications with growing data requirements.

Data Manipulation and Aggregation:

PyMongo provides methods and functions for performing various data manipulation tasks, including filtering, sorting, and aggregating data. It supports the MongoDB Aggregation Framework, allowing developers to perform complex data transformations directly within the database.

Geospatial Data:

MongoDB has built-in support for geospatial indexing and queries. PyMongo allows developers to work with geospatial data, making it suitable for applications that involve location-based services.

Document-Oriented Nature:

PyMongo aligns well with Python's strengths, as both MongoDB and Python use a similar document-oriented approach. Python dictionaries and MongoDB documents share a similar structure, making it easy to work with data in a natural and intuitive way.

Community and Documentation:

PyMongo is well-supported by a vibrant community, and extensive documentation is available. This makes it easy for developers to find resources, ask questions, and troubleshoot any issues they encounter while using PyMongo.

PyMongo, the Python driver for MongoDB, brings several advantages to developers working with MongoDB databases. One notable advantage is its ease of use, offering a straightforward and intuitive API that aligns seamlessly with Python's syntax. This allows developers to interact with MongoDB and perform various database operations using familiar language constructs. Another key benefit is the flexibility it provides in data modeling. MongoDB's schema-less nature is well-supported by PyMongo, enabling developers to work with dynamic and evolving data structures, crucial for applications that undergo frequent changes in their data models.

Performance optimization is another strong suit of PyMongo, leveraging MongoDB's native capabilities like indexing and sharding. This results in efficient data retrieval and manipulation, making it suitable for applications with demanding performance requirements. The scalability of MongoDB is harnessed by PyMongo, allowing developers to distribute data across multiple nodes or servers seamlessly. This is particularly advantageous for applications experiencing growth in data volume.

PyMongo's support for MongoDB's Aggregation Framework is also noteworthy. Developers can perform complex data transformations, filtering, and grouping directly within the database, reducing the need to transfer large datasets to the application for processing. Additionally, PyMongo facilitates the handling of geospatial data, offering support for MongoDB's geospatial indexing and queries. This feature is valuable for applications involving location-based services or spatial analysis.

Community and support play a crucial role in any technology, and PyMongo benefits from a large and active community of developers. This community provides assistance, resources, and a wealth of knowledge, enhancing the overall experience of working with PyMongo and MongoDB. Furthermore, PyMongo is well-documented, offering comprehensive guides, examples, and references to aid developers in learning and using the library effectively.

PyMongo's compatibility with asynchronous programming using Python's `asyncio` library is another advantage. This allows developers to build asynchronous and non-blocking applications, leveraging Python's capabilities in this area. In terms of security, PyMongo supports MongoDB's security features, including authentication and encryption, ensuring that developers can implement robust access controls and protect sensitive data.

Lastly, PyMongo is an open-source project with active maintenance and development. This ensures that developers can benefit from ongoing improvements, updates, and the introduction of new features, making PyMongo a reliable and evolving choice for Python developers working with MongoDB.

Installation:

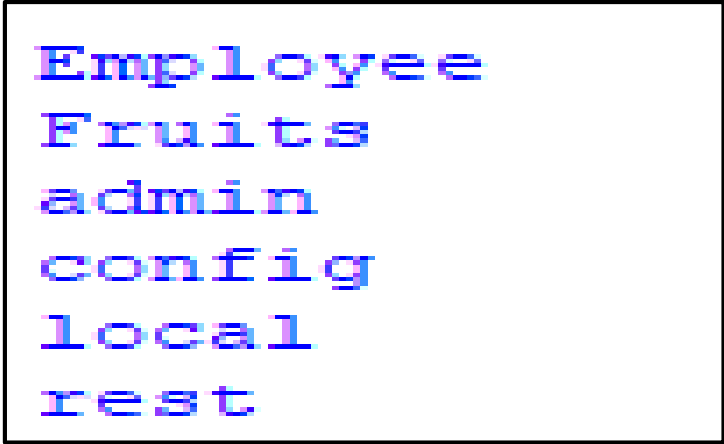
- Before interacting with MongoDB using Python, you need to install the required Python libraries.
- The primary library for MongoDB interaction in Python is pymongo.
- You can install pymongo using pip:
- Additionally, ensure that you have MongoDB installed and running on your system.



```
pip install pymongo
```

Q1. Get names of all databases present in the client system.

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
database_names = client.list_database_names()
for db_name in database_names:
    print(db_name)
```

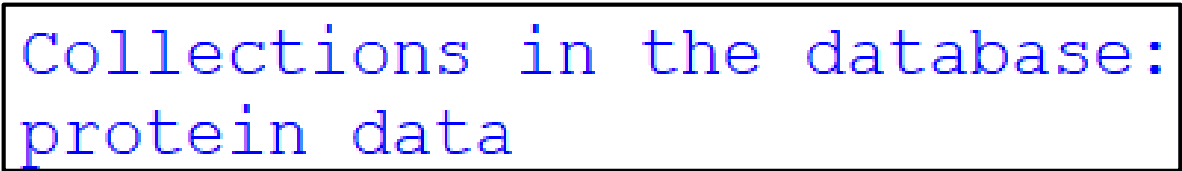


```
Employee
Fruits
admin
config
local
rest
```

Figure 1: Output of Q1

Q2. Get a list of all collection names present in the database.

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection_names = db.list_collection_names()
print("Collections in the database:")
for name in collection_names:
    print(name)
```



```
Collections in the database:
protein data
```

Figure 2: Output of Q2

Q3. Basic CRUD Operations:

1. Insert multiple documents into a collection.

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

```

collection = db["protein_data"]
protein = {
    "name": "hemoglobin",
    "sequence": "MMPLLMM",
    "molecular_weight": 5452,
    "isTransmembrane": False
}
collection.insert_one(protein)
proteins = [
    {"name": "Uglobin", "sequence": "MMMSTDTD", "molecular_weight": 15478,
    "isTransmembrane": True},
    {"name": "isnulin", "sequence": "MPPSTDTD", "molecular_weight": 85478,
    "isTransmembrane": True},
]
collection.insert_many(proteins)
collection.delete_many({})

```

```

[{'name': 'Uglobin', 'sequence': 'MMMSTDTD', 'molecular_weight': 15478, 'isTransmembrane': True, '_id': ObjectId('65e182655d49bffc420c9d97')}, {'name': 'isnulin', 'sequence': 'MPPSTDTD', 'molecular_weight': 85478, 'isTransmembrane': True, '_id': ObjectId('65e182655d49bffc420c9d98')}]

```

Figure 3: Output of Q3.1

2. Retrieve documents based on specific criteria

```

import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
protein = {
    "name": "hemoglobin",
    "sequence": "MMPLLMM",
    "molecular_weight": 5452,
    "isTransmembrane": False
}
collection.insert_one(protein)
proteins = [
    {"name": "Uglobin", "sequence": "MMMSTDTD", "molecular_weight": 15478,
    "isTransmembrane": True},
    {"name": "insulin", "sequence": "MPPSTDTD", "molecular_weight": 85478,
    "isTransmembrane": True},
]
collection.insert_many(proteins)

```

```

query = {"molecular_weight": {"$gt": 60000}}
result = collection.find(query)
for a in result:
    print(a)

```

```

{'_id': ObjectId('65d6cf363b158a54a9e98906'), 'name': 'isnulin', 'sequence': 'MPPSTDTD', 'molecular_weight': 85478, 'isTransmembrane': True}
{'_id': ObjectId('65e181f260c5a2c1990dd7c5'), 'name': 'insulin', 'sequence': 'MPPSTDTD', 'molecular_weight': 85478, 'isTransmembrane': True}
{'_id': ObjectId('65e18211c55409cbl2cdab2e'), 'name': 'insulin', 'sequence': 'MPPSTDTD', 'molecular_weight': 85478, 'isTransmembrane': True}

```

Figure 4: Output of Q3.2

3. Update a document

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
protein = {
    "name": "hemoglobin",
    "sequence": "MMPLLMM",
    "molecular_weight": 5452,
    "isTransmembrane": False
}
collection.insert_one(protein)
proteins = [
    {"name": "Uglobin", "sequence": "MMMSTDTD", "molecular_weight": 15478,
    "isTransmembrane": True},
    {"name": "insulin", "sequence": "MPPSTDTD", "molecular_weight": 85478,
    "isTransmembrane": True},
]
collection.insert_many(proteins)

filter = {"name": "hemoglobin"}
update = {"$set": {"isTransmembrane": True}}
new =
collection.update_one(filter, update)

print(protein)
```

```
{'name': 'hemoglobin', 'sequence': 'MMPLLMM', 'molecular_weight': 5452, 'isTransmembrane': False, '_id': ObjectId('65e181f260c5a2c1990dd7c3')}
```

Figure 5: Output of Q3.3

4. Delete documents from the collection

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
protein = {
    "name": "hemoglobin",
    "sequence": "MMPLLMM",
    "molecular_weight": 5452,
    "isTransmembrane": False
}
collection.insert_one(protein)
proteins = [
    {"name": "Uglobin", "sequence": "MMMSTDTD", "molecular_weight": 15478,
    "isTransmembrane": True},
```

```

    {"name": "insulin", "sequence": "MPPSTDTD", "molecular_weight": 85478,
    "isTransmembrane": True},
]
collection.insert_many(proteins)

collection.delete_many({})

for doc in collection.find():

    print(doc)

```

Q4. Indexing (Text Search):

Write a python program to use a text index to search for proteins containing specific keywords.

OR

Create a Python program that demonstrates text search functionality in MongoDB using PyMongo. Implement text indexes on a collection and perform text search queries to find documents containing specific keywords or phrases.

```

import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")

db = client["bioinformatics_db"]

collection = db["protein_data"]

collection.create_index([("name", "text")])

keyword = "uglobin"

result = collection.find({"$text": {"$search": keyword}})

for a in result:

    print(a)

```

```

{'_id': ObjectId('65d6cf363b158a54a9e98905'), 'name': 'Uglobin', 'sequence': 'MMMSTDTD', 'molecular_weight': 15478, 'isTransmembrane': True}

```

Figure 6: Output of Q4

Q5. Aggregation Framework:

- 1. Write a Python program to group documents in the "protein_data" collection by the "name" field and count the occurrences of each name.**

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
pipe = [
    {"$group": { "_id": "$name", "count": {"$sum": 1}}}
]
result = collection.aggregate(pipe)

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

```
{'_id': 'isnulin', 'count': 1}
{'_id': 'hemoglobin', 'count': 1}
{'_id': 'Uglobin', 'count': 1}
```

Figure 7: Output of Q5.1

- 2. Write a Python program to group documents in the "protein_data" collection by the isTransmembrane field and get an average of molecular weight.**

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
pipe = [
    {"$group": { "_id": "$isTransmembrane", "avg_molecular_weight": {"$avg":
"$molecular_weight"}}}
]
result = collection.aggregate(pipe)

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

```
{'_id': False, 'avg_molecular_weight': 5452.0}
{'_id': True, 'avg_molecular_weight': 50478.0}
```

Figure 8: Output of Q5.2

Q6. Error Handling:

Write a Python program that simulates various error scenarios (e.g., network errors, database errors, validation errors) when interacting with MongoDB using PyMongo. Implement error handling mechanisms to handle these errors gracefully.

```
import pymongo

try:
```



```
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
print("Connected to MongoDB successfully")
```

except:

```
print("Connection to MongoDB failed")
```

A terminal window with a black background and a white border. The text "Connected to MongoDB successfully" is displayed in a blue, monospaced font.

Figure 9: Output of Q6

Q7. Data Import/Export:

Write a Python program to import data from an external source (e.g., CSV file) into MongoDB using PyMongo. Similarly, export data from MongoDB to a JSON file format using PyMongo.

```
import pymongo
import json
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
data = list(collection.find())
for a in data:
    a["_id"] = str(a["_id"])
with open("exported_data.json", 'w') as file:
    json.dump(data, file, indent=4)
print("Data exported successfully to JSON file")
```

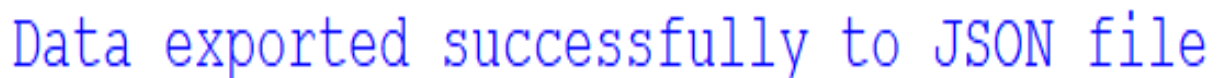
A terminal window with a black background and a white border. The text "Data exported successfully to JSON file" is displayed in a blue, monospaced font.

Figure 10: Output of Q7

Q8. Advanced Queries:

- 1. Implement a Python program that demonstrates the use of \$in in MongoDB using PyMongo.**

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
result = collection.find({"name": {"$in": ["Uglobin", "isnulin"]}})
for a in result:
    print(a)
```

```
{'_id': ObjectId('65d6cf363b158a54a9e98905'), 'name': 'Uglobin', 'sequence': 'MMMSTDTD', 'molecular_weight': 15478, 'isTransmembrane': True}
{'_id': ObjectId('65d6cf363b158a54a9e98906'), 'name': 'isnulin', 'sequence': 'MPPSTDTD', 'molecular_weight': 85478, 'isTransmembrane': True}
```

Figure 11: Output of Q8.1

2. Implement a Python program that demonstrates the use of \$nin in MongoDB using PyMongo.

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
result = collection.find({"name": {"$nin": ["Uglobin", "isnulin"]}})
for a in result:
    print(a)
```

```
{'_id': ObjectId('65e181f260c5a2c1990dd7c5'), 'name': 'insulin', 'sequence': 'MPPSTDTD', 'molecular_weight': 85478, 'isTransmembrane': True}
{'_id': ObjectId('65e18211c55409cb12cdab2c'), 'name': 'hemoglobin', 'sequence': 'MMPLLM', 'molecular_weight': 5452, 'isTransmembrane': False}
```

Figure 12: Output of Q8.2

3. Implement a Python program that demonstrates the use of \$regex in MongoDB using PyMongo.

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
result = collection.find({"sequence": {"$regex": "^MM"}})
for a in result:
    print(a)
```

```
{'_id': ObjectId('65d6cf363b158a54a9e98904'), 'name': 'hemoglobin', 'sequence': 'MMPLLM', 'molecular_weight': 5452, 'isTransmembrane': True}
{'_id': ObjectId('65d6cf363b158a54a9e98905'), 'name': 'Uglobin', 'sequence': 'MMMSTDTD', 'molecular_weight': 15478, 'isTransmembrane': True}
```

Figure 13: Output of Q8.3

4. Implement a Python program that demonstrates the use of \$exists in

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
result = collection.find({"isTransmembrane": {"$exists": True}})
for a in result:
    print(a)
```

```
'_id': ObjectId('65d6cf363b158a54a9e98904'), 'name': 'hemoglobin', 'sequence': 'MMPLLM', 'molecular_weight': 5452, 'isTransmembrane': True}
'_id': ObjectId('65d6cf363b158a54a9e98905'), 'name': 'Uglobin', 'sequence': 'MMSTDID', 'molecular_weight': 15478, 'isTransmembrane': True}
```

Figure 14: Output of Q8.4

5. Write a Python program to count all the documents where the “isTransmembrane” field is True.

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["bioinformatics_db"]
collection = db["protein_data"]
result = collection.count_documents({"isTransmembrane": True})
print("Number of documents with isTransmembrane = True: ", result)
```

```
Number of documents with isTransmembrane = True:  11
```

Figure 15: Output of Q8.5