PyMongo Tutorials

Welcome to the **PyMongo Tutorials** repository! This collection of tutorials is designed to help you get started with PyMongo, the official MongoDB driver for Python. PyMongo provides the necessary tools to interact with MongoDB, a powerful NoSQL database, directly from your Python code.

Table of Contents

Introduction

PyMongo is the official MongoDB driver for Python, providing a straightforward API for interacting with MongoDB databases. This documentation aims to cover all aspects of using PyMongo, from basic setup and CRUD operations to more advanced features like transactions and aggregation.

Whether you are new to MongoDB or a seasoned developer, these tutorials will help you leverage the power of MongoDB with PyMongo in your Python projects.

Installation

To install PyMongo, use pip:

```
pip install pymongo
```

To install the latest development version, you can use:

```
pip install git+https://github.com/mongodb/mongo-python-driver.git
```

For more details on installation, refer to the official documentation.

Getting Started

Before diving into the tutorials, you need to set up a MongoDB instance. You can either run MongoDB locally or use a cloud service like MongoDB Atlas.

Connecting to MongoDB

```
from pymongo import MongoClient

# Create a connection to the MongoDB server
client = MongoClient('mongodb://localhost:27017/')

# Access a specific database
db = client['mydatabase']

# Access a collection
collection = db['mycollection']
```

CRUD Operations

CRUD stands for Create, Read, Update, and Delete—four fundamental operations for interacting with a database.

Create

Inserting documents into a collection:

```
document = {"name": "John", "age": 30, "city": "New York"}
result = collection.insert_one(document)

# Insert multiple documents
documents = [
    {"name": "Jane", "age": 25, "city": "San Francisco"},
    {"name": "Mike", "age": 35, "city": "Chicago"}
]
result = collection.insert_many(documents)
```

Read

Querying documents from a collection:

```
# Find one document
result = collection.find_one({"name": "John"})

# Find all documents matching a query
results = collection.find({"city": "New York"})
for doc in results:
    print(doc)
```

Update

Updating documents in a collection:

```
# Update one document
collection.update_one({"name": "John"}, {"$set": {"age": 31}})

# Update multiple documents
collection.update_many({"city": "New York"}, {"$set": {"city": "Los Angeles"}})
```

Delete

Removing documents from a collection:

```
# Delete one document
collection.delete_one({"name": "John"})

# Delete multiple documents
collection.delete_many({"age": {"$lt": 30}})
```

Indexing and Aggregation

Indexes in MongoDB improve query performance. PyMongo supports creating and managing indexes.

Creating an Index

```
# Create a single-field index
collection.create_index([("name", pymongo.ASCENDING)])

# Create a compound index
collection.create_index([("name", pymongo.ASCENDING), ("age", pymongo.DESCENDING)])
```

Aggregation

Aggregation allows for complex data processing and analysis.

```
pipeline = [
    {"$match": {"city": "New York"}},
    {"$group": {"_id": "$city", "average_age": {"$avg": "$age"}}}
]
results = collection.aggregate(pipeline)
for doc in results:
    print(doc)
```

Advanced Topics

Transactions

PyMongo supports multi-document transactions, allowing for ACID-compliant operations.

```
with client.start_session() as session:
    with session.start_transaction():
        collection.insert_one({"name": "Alice"}, session=session)
        collection.insert_one({"name": "Bob"}, session=session)
```

GridFS

GridFS is used to store and retrieve large files, such as images or videos, in MongoDB.

```
from gridfs import GridFS

fs = GridFS(db)

# Store a file
with open("example.txt", "rb") as f:
    fs.put(f, filename="example.txt")

# Retrieve a file
output = fs.get_last_version("example.txt")
with open("output.txt", "wb") as f:
    f.write(output.read())
```

Change Streams

Change Streams provide a way to watch changes to documents in real time.

```
with collection.watch() as stream:
    for change in stream:
        print(change)
```

Best Practices

- Use indexes to improve query performance.
- Always handle potential exceptions (e.g., network errors, timeouts).
- Use transactions for multi-document operations that require ACID guarantees.
- Regularly backup your MongoDB database.
- Ensure your database is secured, especially when exposed to the internet.

Troubleshooting

If you encounter issues while using PyMongo, here are some common troubleshooting tips:

- Ensure MongoDB is running and accessible from your Python environment.
- Check the PyMongo version compatibility with your MongoDB server version.
- Review the error messages carefully; they often provide hints about what went wrong.

For more detailed troubleshooting, refer to the official PyMongo troubleshooting guide.

Resources

- PyMongo Documentation
- MongoDB Documentation
- MongoDB University
- PyMongo GitHub Repository

Contributing

We welcome contributions to improve these tutorials! If you find a bug or have an idea for a new tutorial, feel free to open an issue or submit a pull request.

License