MongoDB & PyMongo

1. What is PyMongo?

- **PyMongo** is a Python library used to interact with **MongoDB**, a NoSQL database.
- Allows **reading, writing, updating, and deleting** data from MongoDB **programmatically**.

Why Use PyMongo?

- **Python Integration**: Enables Python programs to connect and interact with MongoDB.
- **Simplified Querying**: Provides an easy way to query JSON-like documents.
- **Flexibility**: Works well with dynamic schema data structures.

2. Understanding MongoDB

What is MongoDB?

- A **document-oriented NoSQL database** that stores data in **JSON-like BSON (Binary JSON)** format.
- Designed for **high performance, scalability, and flexibility**.
- Unlike relational databases, it does **not use tables, rows, or schemas**.

Why Use MongoDB Instead of Relational Databases?

1. **Schema Flexibility**: No fixed structure—documents can have varying fields.
2. **Scalability**: Supports horizontal scaling and distributed storage.
3. **High Performance**: Optimized for **high read/write throughput**.
4. **Handles Unstructured Data**: Can store **nested and complex objects**.
5. **Replication & Sharding**: Ensures **fault tolerance** and supports **large datasets**.

3. Connecting to MongoDB with PyMongo

- To use MongoDB with Python, you first establish a **MongoClient**.
- A **MongoClient** is an object that represents the connection to the MongoDB server.

Components of a MongoDB Connection

- **Database**: A logical container (like a schema in SQL).
- **Collection**: A grouping of related documents (similar to a table in SQL).
- **Document**: A single record (like a row in SQL) but stored in **JSON/BSON format**.

4. Data Operations in MongoDB

MongoDB supports CRUD operations:

Operation	Equivalent in SQL	Description
Create	`INSERT`	Add new documents to a collection.
Read	`SELECT`	Retrieve data from the database.
Update	`UPDATE`	Modify existing documents.
Delete	`DELETE`	Remove documents from a collection.

5. Understanding MongoDB Data Model

Document Structure

MongoDB stores data in **documents**, which are **JSON-like objects**.

Example Document:

```
{
  "author": "Mark",
  "text": "MongoDB is Cool!",
  "tags": ["mongodb", "python"]
}
```

- Each document has a unique **_id** (automatically assigned if not specified).
- Can store **arrays and nested fields**.

6. Querying Data in MongoDB

Find Operations

- Unlike SQL, which uses **SELECT queries**, MongoDB uses **find()**.
- Example: **Find all movies from the year 2000.**
- Data is returned in **BSON format**, which can be converted to JSON.

Filtering Data

- MongoDB uses **key-value pairs** to filter documents.
- Example conditions:
  - `{ "year": 2000 }` → Find all movies released in 2000.
  - `{ "author": "Mark" }` → Find all documents where **author = Mark**.

7. Using Jupyter for MongoDB

- **Jupyter Notebooks** allow interactive data exploration.
- Install **PyMongo** using `pip install pymongo`.
- Use **Jupyter Lab** for **interactive MongoDB queries** and visualization.

8. Key Takeaways

- **MongoDB is NoSQL**, meaning it doesn't use tables but instead **JSON-like documents**.
- **PyMongo provides a Python API** to interact with MongoDB.
- **CRUD operations** work similarly to SQL but with different syntax.
- **Jupyter Notebooks** can be used for MongoDB data analysis.