MongoDB With Python

- Python can be used in database applications.
- One of the most popular NoSQL database is MongoDB.

In This Lab, you will learn the MongoDB

1.Create Database
2.Create Collection
3.Insert
4.Find
5.Query
6. Sort
7.Delete
8.Drop Collection
9.Update
10.Limit

MongoDB

- MongoDB stores data in JSON-like documents, which makes the database very flexible and scalable.
- To be able to experiment with the code examples in this Notebook, you will need access to a MongoDB database.
- You can download a free MongoDB database at https://www.mongodb.com.
- Or get started right away with a MongoDB cloud service at https://www.mongodb.com/cloud/atlas.

PyMongo (Download and install "PyMongo")

- Python needs a MongoDB driver to access the MongoDB database.
- In this tutorial we will use the MongoDB driver "PyMongo".
- We recommend that you use PIP to install "PyMongo". PIP is most likely already installed in your Python environment.
- Navigate your command line to the location of PIP, and type the following:

```
pip install pymongo
```

- If you are using anaconda, you can also install using the anaconda shell with the following

```
conda install pymongo
```

Test PyMongo

- To test if the installation was successful, or if you already have "pymongo" installed, create a Python page with the following content:

```
# demo_mongodb_test.py:
import pymongo
```

- If the above code was executed with no errors, "pymongo" is installed and ready to be used.

1.Create Database

- To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

- MongoDB will create the database if it does not exist, and make a connection to it.

```
# Create a database called "mydatabase":
import pymongo

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

mydb = myclient["mydatabase"]
```

Note:

- In MongoDB, a database is not created until it gets content!

- MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

2. Check if Database Exists

Remember:

- In MongoDB, a database is not created until it gets content, so if this is your first time creating a database, you should complete the next two chapters (create collection and create document) before you check if the database exists!

- You can check if a database exist by listing all databases in you system:

```
# Return a list of your system's databases:
print(myclient.list_database_names())

# Or you can check a specific database by name:
# Check if "mydatabase" exists:

dblist = myclient.list_database_names()
if "mydatabase" in dblist:
    print("The database exists.")
```

3. Create Collection

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.

```
# Create a collection called "customers":
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]

mycol = mydb["customers"]
```

- Important: In MongoDB, a collection is not created until it gets content!
- MongoDB waits until you have inserted a document before it actually creates the collection.

Remember:

- In MongoDB, a collection is not created until it gets content, so if this is your first time creating a collection, you should complete the next chapter (create document) before you check if the collection exists!
- You can check if a collection exist in a database by listing all collections:

```
# Return a list of all collections in your database:
print(mydb.list_collection_names())

# Or you can check a specific collection by name:
# Check if the "customers" collection exists:
collist = mydb.list_collection_names()
if "customers" in collist:
    print("The collection exists.")
```

4.Insert Document

- A document in MongoDB is the same as a record in SQL databases.
- To insert a record, or document as it is called in MongoDB, into a collection, we use the insert_one() method.
- The first parameter of the insert_one() method is a dictionary containing the name(s) and value(s) of each field in the document you want to insert.

```
# Insert a record in the "customers" collection:
import pymongo

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

```
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydict = { "name": "John", "address": "Highway 37" }

x = mycol.insert_one(mydict)
```

5.Return the _id Field

- The insert_one() method returns a InsertOneResult object, which has a property, inserted_id, that holds the id of the inserted document.
- Insert another record in the "customers" collection, and return the value of the _id field:

```
mydict = { "name": "Peter", "address": "Lowstreet 27" }

x = mycol.insert_one(mydict)

print(x.inserted_id)
```

- If you do not specify an _id field, then MongoDB will add one for you and assign a unique id for each document.
- In the example above no _id field was specified, so MongoDB assigned a unique _id for the record (document).

6.Insert Multiple Documents

- To insert multiple documents into a collection in MongoDB, we use the insert_many() method.
- The first parameter of the insert_many() method is a list containing dictionaries with the data you want to insert:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "name": "Amy", "address": "Apple st 652"},
  { "name": "Hannah", "address": "Mountain 21"},
  { "name": "Michael", "address": "Valley 345"},
  { "name": "Sandy", "address": "Ocean blvd 2"},
  { "name": "Betty", "address": "Green Grass 1"},
  { "name": "Richard", "address": "Sky st 331"},
  { "name": "Susan", "address": "One way 98"},
  { "name": "Vicky", "address": "Yellow Garden 2"},
  { "name": "Ben", "address": "Park Lane 38"},
  { "name": "William", "address": "Central st 954"},
  { "name": "Chuck", "address": "Main Road 989"},
  { "name": "Viola", "address": "Sideway 1633"}
]
```

```
x = mycol.insert_many(mylist)

#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

- The insert_many() method returns a InsertManyResult object, which has a property, inserted_ids, that holds the ids of the inserted documents.

- Insert Multiple Documents, with Specified IDs
- If you do not want MongoDB to assign unique ids for you document, you can specify the _id field when you insert the document(s).
- Remember that the values has to be unique. Two documents cannot have the same _id.

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "_id": 1, "name": "John", "address": "Highway 37"},
  { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
  { "_id": 3, "name": "Amy", "address": "Apple st 652"},
  { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
  { "_id": 5, "name": "Michael", "address": "Valley 345"},
  { "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"},
  { "_id": 7, "name": "Betty", "address": "Green Grass 1"},
  { "_id": 8, "name": "Richard", "address": "Sky st 331"},
  { "_id": 9, "name": "Susan", "address": "One way 98"},
  { "_id": 10, "name": "Vicky", "address": "Yellow Garden 2"},
  { "_id": 11, "name": "Ben", "address": "Park Lane 38"},
  { "_id": 12, "name": "William", "address": "Central st 954"},
  { "_id": 13, "name": "Chuck", "address": "Main Road 989"},
  { "_id": 14, "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)

#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

- The insert_many() method returns a InsertManyResult object, which has a property, inserted_ids, that holds the ids of the inserted documents.

7.Insert Multiple Documents, with Specified IDs

- If you do not want MongoDB to assign unique ids for you document, you can specify the _id field when you insert the document(s).

- Remember that the values has to be unique. Two documents cannot have the same _id.

```
import pymongo

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

```
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "_id": 1, "name": "John", "address": "Highway 37"},
  { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
  { "_id": 3, "name": "Amy", "address": "Apple st 652"},
  { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
  { "_id": 5, "name": "Michael", "address": "Valley 345"},
  { "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"},
  { "_id": 7, "name": "Betty", "address": "Green Grass 1"},
  { "_id": 8, "name": "Richard", "address": "Sky st 331"},
  { "_id": 9, "name": "Susan", "address": "One way 98"},
  { "_id": 10, "name": "Vicky", "address": "Yellow Garden 2"},
  { "_id": 11, "name": "Ben", "address": "Park Lane 38"},
  { "_id": 12, "name": "William", "address": "Central st 954"},
  { "_id": 13, "name": "Chuck", "address": "Main Road 989"},
  { "_id": 14, "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)

#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

8.Find

- In MongoDB we use the find and findOne methods to find data in a collection.
- Just like the SELECT statement is used to find data in a table in a MySQL database

Find One

- To select data from a collection in MongoDB, we can use the find_one() method.
- The find_one() method returns the first occurrence in the selection.

```
# Find the first document in the customers collection:

import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.find_one()

print(x)
```

Find All

- To select data from a table in MongoDB, we can also use the find() method.
- The find() method returns all occurrences in the selection.
- The first parameter of the find() method is a query object. In this example we use an empty query object, which selects all documents in the collection.
- No parameters in the find() method gives you the same result as SELECT * in MySQL.

```
# Return all documents in the "customers" collection, and print each docume
nt:
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find():
    print(x)
```

Return Only Some Fields

- The second parameter of the find() method is an object describing which fields to include in the result.
- This parameter is optional, and if omitted, all fields will be included in the result.

```
# Return only the names and addresses, not the _ids:
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find({},{ "_id": 0, "name": 1, "address": 1 }):
    print(x)
```

- You are not allowed to specify both 0 and 1 values in the same object (except if one of the fields is the _id field). If you specify a field with the value 0, all other fields get the value 1, and vice versa:

```
# This example will exclude "address" from the result:

import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find({},{ "address": 0 }):
    print(x)
# You get an error if you specify both 0 and 1 values in the same object (e
xcept if one of the fields is the _id field):
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find({},{ "name": 1, "address": 0 }):
    print(x)
```

9.Query

Filter the Result

- When finding documents in a collection, you can filter the result by using a query object.
- The first argument of the find() method is a query object, and is used to limit the search.

```
# Find document(s) with the address "Park Lane 38":
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Park Lane 38" }

mydoc = mycol.find(myquery)

for x in mydoc:
    print(x)
```

Advanced Query

- To make advanced queries you can use modifiers as values in the query object.
- E.g. to find the documents where the "address" field starts with the letter "S" or higher (alphabetically), use the greater than modifier: {"$gt": "S"}:

```
# Find documents where the address starts with the letter "S" or higher:

import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$gt": "S" } }

mydoc = mycol.find(myquery)

for x in mydoc:
    print(x)
```

Filter With Regular Expressions

- You can also use regular expressions as a modifier.
- Regular expressions can only be used to query strings.
- To find only the documents where the "address" field starts with the letter "S", use the regular expression {"$regex": "^S"}:

```
# Find documents where the address starts with the letter "S":
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$regex": "^S" } }

mydoc = mycol.find(myquery)

for x in mydoc:
    print(x)
```

10.Sort

Sort the Result

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

```
# Sort the result alphabetically by name:
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name")

for x in mydoc:
    print(x)
```

Sort Descending

- Use the value -1 as the second parameter to sort descending.
- sort("name", 1) #ascending
- sort("name", -1) #descending

```
# Sort the result reverse alphabetically by name:
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name", -1)

for x in mydoc:
    print(x)
```

11.Delete Document

- To delete one document, we use the delete_one() method.
- The first parameter of the delete_one() method is a query object defining which document to delete.
- Note: If the query finds more than one document, only the first occurrence is deleted.

```
# Delete the document with the address "Mountain 21":
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Mountain 21" }

mycol.delete_one(myquery)
```

Delete Many Documents

- To delete more than one document, use the delete_many() method.
- The first parameter of the delete_many() method is a query object defining which documents to delete.

```
# Delete all documents were the address starts with the letter S:
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": {"$regex": "^S"} }

x = mycol.delete_many(myquery)

print(x.deleted_count, " documents deleted.")
```

Delete All Documents in a Collection

- To delete all documents in a collection, pass an empty query object to the delete_many() method:

```
# Delete all documents in the "customers" collection:
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.delete_many({})

print(x.deleted_count, " documents deleted.")
```

12. Drop Collection

Delete Collection

- You can delete a table, or collection as it is called in MongoDB, by using the drop() method.

```
# Delete the "customers" collection:
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mycol.drop()
```

- The drop() method returns true if the collection was dropped successfully, and false if the collection does not exist.

13.Update

Update Collection

- You can update a record, or document as it is called in MongoDB, by using the update_one() method.
- The first parameter of the update_one() method is a query object defining which document to update.
- Note: If the query finds more than one record, only the first occurrence is updated.
- The second parameter is an object defining the new values of the document.

```
# Change the address from "Valley 345" to "Canyon 123":
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Valley 345" }
```

```
newvalues = { "$set": { "address": "Canyon 123" } }

mycol.update_one(myquery, newvalues)

#print "customers" after the update:
for x in mycol.find():
    print(x)
```

Update Many

- To update all documents that meets the criteria of the query, use the update_many() method.

```
# Update all documents where the address starts with the letter "S":
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$regex": "^S" } }
newvalues = { "$set": { "name": "Minnie" } }

x = mycol.update_many(myquery, newvalues)

print(x.modified_count, "documents updated.")
```

14.Limit

Limit the Result

- To limit the result in MongoDB, we use the limit() method.
- The limit() method takes one parameter, a number defining how many documents to return.
- Consider you have a "customers" collection:

```
# Limit the result to only return 5 documents:
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myresult = mycol.find().limit(5)

#print the result:
for x in myresult:
    print(x)
```