**Mongo Db**

1. **DataBase**
   1. **Creating a 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.

**import pyngo**

**#create connection**

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

**#Create database**

**db=client["Sharydb"]**

**Important:** 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).

**1.2 Check Database**

**print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* Check databases \*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

**#check data base**

**dblist=client.list\_database\_names()**

**if "Sharydb" in dblist:**

**print("The Sharydb is exist")**

* 1. **Drop database**

**#Drop database**

**client.drop\_database('Shary')**

## Creating a Collection

* To create a collection in MongoDB, u se database object and specify the name of the collection you want to create.
* MongoDB will create the collection if it does not exist.
* A collection in MongoDB is the same as a table in SQL databases.
* **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.

import pymongo

#create connection

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

#Create database

db=client["Sharydb"]

#Create collection

collection=db["Customer"]

#insert one doc

dict={"name":"Shary","Address":"Charsadda","Marks":400}

collection.insert\_one(dict)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* Check collection \*\*\*\*\*\*\*\*\*\*\*\*\*\*")

clist=db.list\_collection\_names()

print(clist)

#Or check by name

if 'Customer' in clist:

    print("Customer is exist")

A **document** in MongoDB is the same as a **record** in SQL databases.

## Insert Into Collection

* 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.

import pymongo

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

#Access the database

db=mclient.Sharydb

#Access Collection or table

CustColection=db.Customer

#Insert a record

dict={"Id":1,"name":"Shary","Address":"Charsadda"}

x=CustColection.insert\_one(dict)

print(x)

## 3.2 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.

#Example Insert another record in the "customers" collection, and return the value of the \_id field:

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Example2 Insert one record \*\*\*\*\*\*\*\*\*\*\*")

dict1={'id':2,"Name":"Umair","Address":"Peshawer"}

x2=CustColection.insert\_one(dict1)

print(x2.inserted\_id)

## 3.3 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:

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Example2 Insert many record with specified ids \*\*\*\*\*\*\*\*\*\*\*")

record\_list=[

    {"\_id":4,"Name":"Fahad","Address":"Mardan"},

    {"\_id":5,"Name":"Ali","Address":"Swat"},

    {"\_id":6,"Name":"Shary","Address":"Umerzai"},

    {"\_id":7,"Name":"Hamza","Address":"Peshawer"},

    {"\_id":8,"Name":"Hamad","Address":"Charsadda"}

]

x3=CustColection.insert\_many(record\_list)

#print list of the all id values of the inserted doc

print(x3.inserted\_ids)

# **MongoDB Find**

In MongoDB we use the find() and find\_one() methods to find data in a collection.

Just like the **SELECT** statement is used to find data in a table in a MySQL database.

## 4.1 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.

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Find one() \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=cust\_cln.find\_one()

print(result)

## 4.2 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.

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Find() all record  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=cust\_cln.find()

for i in result:

    print(i)

## 4.3 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.

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* names and addresses \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

for data in cust\_cln.find({},{"\_id":0,"name":1,"address":1}):

    print(data)

**4.4 Comparsion Operator**

In MongoDB, comparison operators are used to compare values in query expressions and to filter documents based on the comparison results. Here is a list of some of the comparison operators available in MongoDB:

**$eq:** This operator tests for equality and returns true if the operands are equal.

for row in collection.find({"Name":{"$eq":"Shahriyar"}}):

    print(row)

**$ne:** This operator tests for inequality and returns true if the operands are not equal.

for row in collection.find({"Name":{"$ne":"Shahriyar"}}):

    print(row)

**$gt:** This operator tests for greater than and returns true if the left operand is greater than the right operand.

for row in collection.find({"Age":{"$gt":"29"}}):

    print(row)

**$gte:** This operator tests for greater than or equal to and returns true if the left operand is greater than or equal to the right operand.

for row in collection.find({"Age":{"$gte":"29"}}):

    print(row)

**$lt:** This operator tests for less than and returns true if the left operand is less than the right operand.

for row in collection.find({"Age":{"$lt":"22"}}):

    print(row)

**$lte:** This operator tests for less than or equal to and returns true if the left operand is less than or equal to the right operand.

for row in collection.find({"Age":{"$lte":"22"}}):

    print(row)

**$in:**

result2=collection.find({"Age":{"$in":["20","29"]}})

for row in result2:

    print(row)

**$nin:**

result2=collection.find({"Age":{"$nin":["20","29"]}})

for row in result2:

    print(row)

**4.5 Logical Operator**

**$and:** This operator performs a logical AND operation on an array of expressions and returns true if all the expressions are true.

result2=collection.find({"$and":[{"Name":"Ysir"},{"Age":"20"}]})

for rows in result2:

    print(rows)

**$or:** This operator performs a logical OR operation on an array of expressions and returns true if at least one of the expressions is true.

result2=collection.find({"$or":[{"Name":"Shahriyar"},{"Age":"20"}]})

for rows in result2:

    print(rows)

**$not:** This operator performs a logical NOT operation on a single expression and returns true if the expression is false and false if the expression is true.

# result2=collection.find({"$not":[{"Name":"Shahriyar"},{"Age":"20"}]})

# for rows in result2:

#     print(rows)

result2=collection.find({"$nor":[{"Age":{"$lte":22}},{"Age":{"$gte":26}}]})

for rows in result2:

    print(rows)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Operators And,OR \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result2=collection.find({"$and":[{"Name":"Shahriyar"},{"$or":[{"Age":"20"},{"City":"Swaat"}]}]})

for rows in result2:

    print(rows)

**4.5 Element Operator:**

Here's an example of how you might use the $exists operator in a find() method:

```db.collection.find({ field: { $exists: true } })```

This would retrieve all documents from the collection where the field exists. You can also use $exists: false to retrieve documents where the field does not exist.

```db.collection.find({ field: { $exists: true, $in: [value1, value2, value3] } })```

This would retrieve all documents from the collection where the field exists and has a value of value1, value2, or value3.

```db.collection.find({ field: { $type: typeCode } })```

This would retrieve all documents from the collection where the field has the data type specified by typeCode.

Here are some examples of type codes that you can use with the $type operator:

1 - double

2 - string

3 - object

4 - array

5 - binary data

6 - undefined (deprecated)

7 - object id

8 - boolean

9 - date

10 - null

11 - regular expression

13 - javascript

14 - symbol

15 - javascript (with scope)

16 - 32-bit integer

17 - timestamp

18 - 64-bit integer

from pymongo import MongoClient

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

#Access database

db=myclient.Sharydb

#Access collection

collection=db.TutorialPoint

for i in collection.find():

    print(i)

print("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\* $exist using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

x=collection.find({"City":{"$exists":True}})

for docs in x:

    print(i)

print("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\* $exist and type using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

x=collection.find({"\_id":{"$exists":True,"$type":7}})

for docs in x:

    print(i)

**4.7 Evaluation Operator**

**$expr** is a MongoDB operator that allows you to use JavaScript expressions to perform queries on the database.

**$regex** is a MongoDB operator that allows you to use regular expressions to perform queries on the database.

$mod is a MongoDB operator that allows you to use modulo arithmetic in queries.

for doc in collection.find({"Age":{"$mod":[2,0]}}):

    print(doc)

**$text** is a MongoDB operator that allows you to perform full-text searches on the database.

idx=collection.create\_index({"Bio":"text"})

for doc in collection.find({"$text":{"$search":"youtube"}}):

    print(doc)

**$jsonSchema** is a MongoDB operator that allows you to validate the structure and content of documents in a collection based on a provided JSON schema.

## 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.

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Find document(s) by address \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

finding={"Address":"Charsadda"}

result=cln.find(finding)

for records in result:

    print(records)

## 5.1 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"}:

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Find document(s) by address start letter \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

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

finding1={"Address":{"$gt":"P"}}

result1=cln.find(finding1)

for records in result1:

    print(records)

## 5.3 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"}:

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Find document(s) using Regression Exp \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

finding2={"Address":{"$regex":"^P"}}

result2=cln.find(finding2)

for record in result2:

    print(record)

## 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).

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Sorting in Ascending \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

records=collection.find().sort("name",1)

for i in records:

    print(i)

## 6.1 Sort Descending

Use the value -1 as the second parameter to sort descending.

sort("name", 1) #ascending  
sort("name", -1) #descending

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Sorting in Descending \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

records=collection.find().sort("Address",-1)

for i in records:

    print(i)

## 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.

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Delete\_one() one row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

q={"Address":"Charsadda"}

cln.delete\_one(q)

## 7.1 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.

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Delete\_many() Multiple row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

q={"Address":"Charsadda"}

cln.delete\_many(q)

for i in cln.find():

    print(i)

## 7.2 Delete All Documents in a Collection

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

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Delete\_many() Multiple row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

x=cln.delete\_many({})

print(x.deleted\_count,"Document deleted.")

## Update() Method

The update() method updates the values in the existing document

**8.1 Update\_one()**

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Update\_one() one row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

old\_q={"Address":"Charsadda"}

new\_q={"$set":{"Address":"Karachi"}}

x=cln.update\_one(old\_q,new\_q)

for rows in cln.find():

    print(rows)

**8.2 Update\_many()**

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Update\_many() one row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

old\_q={"Address":{"$regex":"Islamabad"}}

new\_q={"$set":{"Address":"Peshawer"}}

x=cln.update\_many(old\_q,new\_q)

for rows in cln.find():

    print(rows)

**8.2 Update\_findOneAndUpdate()**

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* findOneAndUpdate() using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=cln.find\_one\_and\_update(

    {"Name":"Shahriyar"},

    {"$set":{"Age":"24","Email":"shahriyarkhanpk1@gmail.com"}}

)

records=cln.find()

for rows in records:

    print(rows)

**8.B Advanced Update**

**$inc:** The $inc operator increments the value of a field by a specified amount.

**$min:** The $min operator updates the value of a field if the specified value is less than the current value of the field.

**$max:** The $max operator updates the value of a field if the specified value is greater than the current value of the field.

**$mul:** The $mul operator multiplies the value of a field by a specified amount.

**$unset:** The $unset operator removes a specific field from a document.

**$rename:** The $rename operator renames a field.

**Upsert:** An upsert is a combination of an update and an insert operation. If a document matching the update criteria does not exist, the update operation creates a new document with the specified update criteria. If a document matching the update criteria does exist, the update operation modifies the existing document.

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* $inc using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=collection.update\_many({},{"$inc":{"Age":2}})

for doc in collection.find():

    print("Increase the age by 2:",doc)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* $max using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=collection.update\_one({"Name":"Mobeen"},{"$max":{"Age":50}})

for doc in collection.find():

    print("increase age of the Shayan :",doc)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* $min using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=collection.update\_one({"Name":"Shahriyar"},{"$min":{"Age":10}})

for doc in collection.find():

    print("increase age of the Shayan :",doc)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* $min using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=collection.update\_one({"Name":"Shahriyar"},{"$min":{"Age":10}})

for doc in collection.find():

    print(doc)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* $mul using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=collection.update\_one({"Name":"Shahriyar"},{"$mul":{"Age":2}})

for doc in collection.find():

    print(doc)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* $unset using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=collection.update\_one({"Name":"Ali"},{"$unset":{"Age":22222}})

for doc in collection.find():

    print(doc)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* $rename using \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

result=collection.update\_many({},{"$rename":{"Age":"StudentAge"}})

for doc in collection.find():

    print(doc)

# MongoDB - Indexing

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require MongoDB to process a large volume of data.

Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

## The createIndex() Method

To create an index, you need to use createIndex() method of MongoDB.

### **Syntax**

import pymongo

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

#Access database

db=client.Sharydb

#Access collection

cln=db.TutorialPoint

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* create index \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

idx=cln.create\_index({"Name":1})

print("Idx is created")

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Retreive index \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

idx\_inf=cln.index\_information()

for idx\_name,idx\_spec in idx\_inf.items():

    print(f"idx\_name: {idx\_name}")

    print(f"idx\_spec: {idx\_spec}")

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Drop index \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

cln.drop\_index("Name\_1")

print("Droped")

# MongoDB - Aggregation

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(\*) and with group by is an equivalent of MongoDB aggregation.

## The aggregate() Method

For the aggregation in MongoDB, you should use **aggregate()** method.