Following example creates a database in MangoDB.

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['mydb']
print("Database created........")

#Verification
print("List of databases after creating new one")
print(client.list_database_names())
```

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['mydb']
print("Database created........")
```

Output

```
Database created........
```

A collection in MongoDB holds a set of documents, it is analogous to a table in relational databases.

You can create a collection using the ***createCollection()*** method. This method accepts a String value representing the name of the collection to be created and an options (optional) parameter.

Using this you can specify the following −

- The *size* of the collection.
- The *max* number of documents allowed in the capped collection.
- Whether the collection we create should be capped collection (fixed size collection).
- Whether the collection we create should be auto-indexed.

Syntax

Following is the syntax to create a collection in MongoDB.

db.createCollection("*CollectionName*")

Example

Following method creates a collection named ExampleCollection.

```
> use mydb
switched to db mydb
> db.createCollection("ExampleCollection")
{ "ok" : 1 }
>
```

Similarly, following is a query that creates a collection using the options of the createCollection() method.

```
>db.createCollection("mycol", { capped : true, autoIndexId : true, size :
6142800, max : 10000 } )
{ "ok" : 1 }
>
```

Creating a Collection Using Python

Following python example connects to a database in MongoDB (mydb) and, creates a collection in it.

Example

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['mydb']

#Creating a collection
collection = db['example']
print("Collection created........")
```

Output

```
Collection created........
```

Python MongoDB - Insert Document

You can store documents into MongoDB using the *insert()* method. This method accepts a JSON document as a parameter.

Syntax

Following is the syntax of the insert method.

```
>db.COLLECTION_NAME.insert(DOCUMENT_NAME)
```

Example

```
> use mydb
switched to db mydb
> db.createCollection("sample")
{ "ok" : 1 }
> doc1 = {"name": "Ram", "age": "26", "city": "Hyderabad"}
{ "name" : "Ram", "age" : "26", "city" : "Hyderabad" }
> db.sample.insert(doc1)
WriteResult({ "nInserted" : 1 })
>
```

Similarly, you can also insert multiple documents using the ***insert()*** method.

```
> use testDB
switched to db testDB
> db.createCollection("sample")
{ "ok" : 1 }
> data = [
   {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "1002", "name" : "Rahim", "age" : 27, "city" : "Bangalore" },
   {"_id": "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai" }
]
[
   {"_id" : "1001", "name" : "Ram", "age" : "26", "city" : "Hyderabad"},
   {"_id" : "1002", "name" : "Rahim", "age" : 27, "city" : "Bangalore"},
   {"_id" : "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai"}
]
> db.sample.insert(data)
BulkWriteResult({
   "writeErrors" : [ ],
   "writeConcernErrors" : [ ],
   "nInserted" : 3,
   "nUpserted" : 0,
   "nMatched" : 0,
   "nModified" : 0,
   "nRemoved" : 0,
   "upserted" : [ ]
})
>
```

Creating a Collection Using Python

Pymongo provides a method named insert_one() to insert a document in MangoDB. To this method, we need to pass the document in dictionary format.

Example

Following example inserts a document in the collection named example.

```python
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['mydb']

#Creating a collection
coll = db['example']

#Inserting document into a collection
doc1 = {"name": "Ram", "age": "26", "city": "Hyderabad"}
coll.insert_one(doc1)
print(coll.find_one())
```

Output

```
{'_id': ObjectId('5d63ad6ce043e2a93885858b'), 'name': 'Ram', 'age': '26',
'city': 'Hyderabad'}
```

To insert multiple documents into MongoDB using pymongo, you need to invoke the insert_many() method.

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['mydb']

#Creating a collection
coll = db['example']

#Inserting document into a collection
data = [
   {"_id": "101", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "102", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "103", "name": "Robert", "age": "28", "city": "Mumbai"}
]
res = coll.insert_many(data)
print("Data inserted ......")
print(res.inserted_ids)
```

Output

```
Data inserted ......
['101', '102', '103']
```

Python MongoDB - Find

You can read/retrieve stored documents from MongoDB using the ***find()*** method. This method retrieves and displays all the documents in MongoDB in a non-structured way.

Syntax

Following is the syntax of the ***find()*** method.

```
>db.COLLECTION_NAME.find()
```

Example

Assume we have inserted 3 documents into a database named testDB in a collection named sample using the following queries −

```
> use testDB
> db.createCollection("sample")
> data = [
   {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "1002", "name" : "Rahim", "age" : 27, "city" : "Bangalore" },
   {"_id": "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai" }
]
> db.sample.insert(data)
```

You can retrieve the inserted documents using the find() method as −

```
> use testDB
switched to db testDB
> db.sample.find()
{ "_id" : "1001", "name" : "Ram", "age" : "26", "city" : "Hyderabad" }
{ "_id" : "1002", "name" : "Rahim", "age" : 27, "city" : "Bangalore" }
{ "_id" : "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai" }
>
```

You can also retrieve first document in the collection using the findOne() method as −

```
> db.sample.findOne()
{ "_id" : "1001", "name" : "Ram", "age" : "26", "city" : "Hyderabad" }
```

Retrieving Data (find) Using Python

The ***find_One()*** method of pymongo is used to retrieve a single document based on your query, in case of no matches this method returns nothing and if you doesn't use any query it returns the first document of the collection.

This method comes handy whenever you need to retrieve only one document of a result or, if you are sure that your query returns only one document.

Example

Following python example retrieve first document of a collection −

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
```

```
db = client['mydatabase']

#Creating a collection
coll = db['example']

#Inserting document into a collection
data = [
   {"_id": "101", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "102", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "103", "name": "Robert", "age": "28", "city": "Mumbai"}
]
res = coll.insert_many(data)
print("Data inserted ......")
print(res.inserted_ids)

#Retrieving the first record using the find_one() method
print("First record of the collection: ")
print(coll.find_one())

#Retrieving a record with is 103 using the find_one() method
print("Record whose id is 103: ")
print(coll.find_one({"_id": "103"}))
```

Output

```
Data inserted ......
['101', '102', '103']
First record of the collection:
{'_id': '101', 'name': 'Ram', 'age': '26', 'city': 'Hyderabad'}
Record whose id is 103:
{'_id': '103', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
```

To get multiple documents in a single query (single call od find method), you can use the ***find()*** method of the pymongo. If haven’t passed any query, this returns all the documents of a collection and, if you have passed a query to this method, it returns all the matched documents.

Example

```
#Getting the database instance
db = client['myDB']

#Creating a collection
```

```
coll = db['example']

#Inserting document into a collection
data = [
   {"_id": "101", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "102", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "103", "name": "Robert", "age": "28", "city": "Mumbai"}
]
res = coll.insert_many(data)
print("Data inserted ......")

#Retrieving all the records using the find() method
print("Records of the collection: ")
for doc1 in coll.find():
   print(doc1)

#Retrieving records with age greater than 26 using the find() method
print("Record whose age is more than 26: ")
for doc2 in coll.find({"age":{"$gt":"26"}}):
   print(doc2)
```

Output

```
Data inserted ......
Records of the collection:
{'_id': '101', 'name': 'Ram', 'age': '26', 'city': 'Hyderabad'}
{'_id': '102', 'name': 'Rahim', 'age': '27', 'city': 'Bangalore'}
{'_id': '103', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
Record whose age is more than 26:
{'_id': '102', 'name': 'Rahim', 'age': '27', 'city': 'Bangalore'}
{'_id': '103', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
```

Python MongoDB - Query

While retrieving using **find()** method, you can filter the documents using the query object. You can pass the query specifying the condition for the required documents as a parameter to this method.

Operators

Following is the list of operators used in the queries in MongoDB.

Operation	Syntax	Example
Equality	{"key" : "value"}	db.mycol.find({"by":"tutorials poi
Less Than	{"key" :{$lt:"value"}}	db.mycol.find({"likes":{$lt:50}})
Less Than Equals	{"key" :{$lte:"value"}}	db.mycol.find({"likes":{$lte:50}})
Greater Than	{"key" :{$gt:"value"}}	db.mycol.find({"likes":{$gt:50}})
Greater Than Equals	{"key" {$gte:"value"}}	db.mycol.find({"likes":{$gte:50}})
Not Equals	{"key":{$ne: "value"}}	db.mycol.find({"likes":{$ne:50}})

Example1

Following example retrieves the document in a collection whose name is sarmista.

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['sdsegf']

#Creating a collection
coll = db['example']

#Inserting document into a collection
data = [
   {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "1002", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "1003", "name": "Robert", "age": "28", "city": "Mumbai"},
   {"_id": "1004", "name": "Romeo", "age": "25", "city": "Pune"},
   {"_id": "1005", "name": "Sarmista", "age": "23", "city": "Delhi"},
   {"_id": "1006", "name": "Rasajna", "age": "26", "city": "Chennai"}
]
res = coll.insert_many(data)
print("Data inserted ......")

#Retrieving data
print("Documents in the collection: ")
for doc1 in coll.find({"name":"Sarmista"}):
   print(doc1)
```

Output

```
Data inserted ......
Documents in the collection:
{'_id': '1005', 'name': 'Sarmista', 'age': '23', 'city': 'Delhi'}
```

Example2

Following example retrieves the document in a collection whose age value is greater than 26.

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['ghhj']

#Creating a collection
coll = db['example']

#Inserting document into a collection
data = [
   {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "1002", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "1003", "name": "Robert", "age": "28", "city": "Mumbai"},
   {"_id": "1004", "name": "Romeo", "age": "25", "city": "Pune"},
   {"_id": "1005", "name": "Sarmista", "age": "23", "city": "Delhi"},
   {"_id": "1006", "name": "Rasajna", "age": "26", "city": "Chennai"}
]
```

```
res = coll.insert_many(data)
print("Data inserted ......")

#Retrieving data
print("Documents in the collection: ")
for doc in coll.find({"age":{"$gt":"26"}}):
  print(doc)
```

Output

```
Data inserted ......
Documents in the collection:
{'_id': '1002', 'name': 'Rahim', 'age': '27', 'city': 'Bangalore'}
{'_id': '1003', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
```

Python MongoDB - Sort

While retrieving the contents of a collection, you can sort and arrange them in ascending or descending orders using the ***sort()*** method.

To this method, you can pass the field(s) and the sorting order which is 1 or -1. Where, 1 is for ascending order and -1 is descending order.

Syntax

Following is the syntax of the *sort()* method.

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

Example

Assume we have created a collection and inserted 5 documents into it as shown below −

```
> use testDB
switched to db testDB
> db.createCollection("myColl")
{ "ok" : 1 }
> data = [
  ... {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
  ... {"_id": "1002", "name": "Rahim", "age": 27, "city": "Bangalore"},
  ... {"_id": "1003", "name": "Robert", "age": 28, "city": "Mumbai"},
  ... {"_id": "1004", "name": "Romeo", "age": 25, "city": "Pune"},
  ... {"_id": "1005", "name": "Sarmista", "age": 23, "city": "Delhi"},
  ... {"_id": "1006", "name": "Rasajna", "age": 26, "city": "Chennai"}
]
> db.sample.insert(data)
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 6,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

Following line retrieves all the documents of the collection which are sorted in ascending order based on age.

```
> db.sample.find().sort({age:1})
{ "_id" : "1005", "name" : "Sarmista", "age" : 23, "city" : "Delhi" }
{ "_id" : "1004", "name" : "Romeo", "age" : 25, "city" : "Pune" }
{ "_id" : "1006", "name" : "Rasajna", "age" : 26, "city" : "Chennai" }
{ "_id" : "1002", "name" : "Rahim", "age" : 27, "city" : "Bangalore" }
{ "_id" : "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai" }
{ "_id" : "1001", "name" : "Ram", "age" : "26", "city" : "Hyderabad" }
```

Sorting the Documents Using Python

To sort the results of a query in ascending or, descending order pymongo provides the ***sort()*** method. To this method, pass a number value representing the number of documents you need in the result.

By default, this method sorts the documents in ascending order based on the specified field. If you need to sort in descending order pass -1 along with the field name −

```
coll.find().sort("age",-1)
```

Example

Following example retrieves all the documents of a collection arranged according to the age values in ascending order −

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['b_mydb']

#Creating a collection
coll = db['myColl']

#Inserting document into a collection
data = [
   {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "1002", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "1003", "name": "Robert", "age": "28", "city": "Mumbai"},
   {"_id": "1004", "name": "Romeo", "age": 25, "city": "Pune"},
   {"_id": "1005", "name": "Sarmista", "age": 23, "city": "Delhi"},
   {"_id": "1006", "name": "Rasajna", "age": 26, "city": "Chennai"}
]
res = coll.insert_many(data)
print("Data inserted ......")

#Retrieving first 3 documents using the find() and limit() methods
print("List of documents (sorted in ascending order based on age): ")
for doc1 in coll.find().sort("age"):
   print(doc1)
```

Output

```
Data inserted ......
List of documents (sorted in ascending order based on age):
{'_id': '1005', 'name': 'Sarmista', 'age': 23, 'city': 'Delhi'}
{'_id': '1004', 'name': 'Romeo', 'age': 25, 'city': 'Pune'}
{'_id': '1006', 'name': 'Rasajna', 'age': 26, 'city': 'Chennai'}
{'_id': '1001', 'name': 'Ram', 'age': '26', 'city': 'Hyderabad'}
{'_id': '1002', 'name': 'Rahim', 'age': '27', 'city': 'Bangalore'}
{'_id': '1003', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
```

Python MongoDB - Delete Document

You can delete documents in a collection using the ***remove()*** method of MongoDB. This method accepts two optional parameters −

- Deletion criteria specifying the condition to delete documents.
- Just one, if you pass true or 1 as second parameter, then only one document will be deleted.

Syntax

Following is the syntax of the remove() method −

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

Example

Assume we have created a collection and inserted 5 documents into it as shown below −

```
> use testDB
switched to db testDB
> db.createCollection("myColl")
{ "ok" : 1 }
> data = [
   ... {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
   ... {"_id": "1002", "name": "Rahim", "age": 27, "city": "Bangalore"},
   ... {"_id": "1003", "name": "Robert", "age": 28, "city": "Mumbai"},
   ... {"_id": "1004", "name": "Romeo", "age": 25, "city": "Pune"},
   ... {"_id": "1005", "name": "Sarmista", "age": 23, "city": "Delhi"},
   ... {"_id": "1006", "name": "Rasajna", "age": 26, "city": "Chennai"}
]
> db.sample.insert(data)
BulkWriteResult({
   "writeErrors" : [ ],
   "writeConcernErrors" : [ ],
   "nInserted" : 6,
   "nUpserted" : 0,
   "nMatched" : 0,
   "nModified" : 0,
   "nRemoved" : 0,
   "upserted" : [ ]
})
```

Following query deletes the document(s) of the collection which have name value as Sarmista.

```
> db.sample.remove({"name": "Sarmista"})
WriteResult({ "nRemoved" : 1 })
> db.sample.find()
{ "_id" : "1001", "name" : "Ram", "age" : "26", "city" : "Hyderabad" }
{ "_id" : "1002", "name" : "Rahim", "age" : 27, "city" : "Bangalore" }
{ "_id" : "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai" }
{ "_id" : "1004", "name" : "Romeo", "age" : 25, "city" : "Pune" }
{ "_id" : "1006", "name" : "Rasajna", "age" : 26, "city" : "Chennai" }
```

If you invoke **remove()** method without passing deletion criteria, all the documents in the collection will be deleted.

```
> db.sample.remove({})
WriteResult({ "nRemoved" : 5 })
> db.sample.find()
```

Deleting Documents Using Python

To delete documents from a collection of MangoDB, you can delete documents from a collections using the methods **delete_one()** and **delete_many()** methods.

These methods accept a query object specifying the condition for deleting documents.

The detele_one() method deletes a single document, in case of a match. If no query is specified this method deletes the first document in the collection.

Example

Following python example deletes the document in the collection which has id value as 1006.

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

```

```
#Getting the database instance
db = client['lpaksgf']

#Creating a collection
coll = db['example']

#Inserting document into a collection
data = [
   {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "1002", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "1003", "name": "Robert", "age": "28", "city": "Mumbai"},
   {"_id": "1004", "name": "Romeo", "age": 25, "city": "Pune"},
   {"_id": "1005", "name": "Sarmista", "age": 23, "city": "Delhi"},
   {"_id": "1006", "name": "Rasajna", "age": 26, "city": "Chennai"}
]
res = coll.insert_many(data)
print("Data inserted ......")

#Deleting one document
coll.delete_one({"_id" : "1006"})

#Retrieving all the records using the find() method
print("Documents in the collection after update operation: ")
for doc2 in coll.find():
   print(doc2)
```

Output

```
Data inserted ......
Documents in the collection after update operation:
{'_id': '1001', 'name': 'Ram', 'age': '26', 'city': 'Hyderabad'}
{'_id': '1002', 'name': 'Rahim', 'age': '27', 'city': 'Bangalore'}
{'_id': '1003', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
{'_id': '1004', 'name': 'Romeo', 'age': 25, 'city': 'Pune'}
{'_id': '1005', 'name': 'Sarmista', 'age': 23, 'city': 'Delhi'}
```

Similarly, the ***delete_many()*** method of pymongo deletes all the documents that satisfies the specified condition.

Example

Following example deletes all the documents in the collection whose age value is greater than 26 −

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['sampleDB']

#Creating a collection
coll = db['example']

#Inserting document into a collection
data = [
   {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "1002", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "1003", "name": "Robert", "age": "28", "city": "Mumbai"},
   {"_id": "1004", "name": "Romeo", "age": "25", "city": "Pune"},
   {"_id": "1005", "name": "Sarmista", "age": "23", "city": "Delhi"},
   {"_id": "1006", "name": "Rasajna", "age": "26", "city": "Chennai"}
]
res = coll.insert_many(data)
print("Data inserted ......")

#Deleting multiple documents
coll.delete_many({"age":{"$gt":"26"}})

#Retrieving all the records using the find() method
print("Documents in the collection after update operation: ")
for doc2 in coll.find():
   print(doc2)
```

Output

```
Data inserted ......
Documents in the collection after update operation:
{'_id': '1001', 'name': 'Ram', 'age': '26', 'city': 'Hyderabad'}
{'_id': '1004', 'name': 'Romeo', 'age': '25', 'city': 'Pune'}
{'_id': '1005', 'name': 'Sarmista', 'age': '23', 'city': 'Delhi'}
{'_id': '1006', 'name': 'Rasajna', 'age': '26', 'city': 'Chennai'}
```

If you invoke the delete_many() method without passing any query, this method deletes all the documents in the collection.

```
coll.delete_many({})
```

Python MongoDB - Drop Collection

You can delete collections using ***drop()*** method of MongoDB.

Syntax

Following is the syntax of drop() method −

```
db.COLLECTION_NAME.drop()
```

Example

Following example drops collection with name sample −

```
> show collections
myColl
sample
> db.sample.drop()
true
> show collections
myColl
```

Dropping Collection Using Python

You can drop/delete a collection from the current by invoking drop() method.

Example

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['example2']

#Creating a collection
col1 = db['collection']
col1.insert_one({"name": "Ram", "age": "26", "city": "Hyderabad"})
col2 = db['coll']
col2.insert_one({"name": "Rahim", "age": "27", "city": "Bangalore"})
col3 = db['myColl']
col3.insert_one({"name": "Robert", "age": "28", "city": "Mumbai"})
col4 = db['data']
```

```
col4.insert_one({"name": "Romeo", "age": "25", "city": "Pune"})

#List of collections
print("List of collections:")
collections = db.list_collection_names()
for coll in collections:
   print(coll)

#Dropping a collection
col1.drop()
col4.drop()
print("List of collections after dropping two of them: ")

#List of collections
collections = db.list_collection_names()
for coll in collections:
   print(coll)
```

Output

```
List of collections:
coll
data
collection
myColl
List of collections after dropping two of them:
coll
myColl
```

Python MongoDB - Update

You can update the contents of an existing documents using the ***update()*** method or ***save()*** method.

The update method modifies the existing document whereas the save method replaces the existing document with the new one.

Syntax

Following is the syntax of the update() and save() methods of MangoDB −

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA,
UPDATED_DATA)
Or,
db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

Example

Assume we have created a collection in a database and inserted 3 records in it as shown below −

```
> use testdatabase
switched to db testdatabase
> data = [
   ... {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
   ... {"_id": "1002", "name" : "Rahim", "age" : 27, "city" : "Bangalore" },
   ... {"_id": "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai" }
]
[
   {"_id" : "1001", "name" : "Ram", "age" : "26", "city" : "Hyderabad"},
   {"_id" : "1002", "name" : "Rahim", "age" : 27, "city" : "Bangalore"},
   {"_id" : "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai"}
]
> db.createCollection("sample")
{ "ok" : 1 }
> db.sample.insert(data)
```

Following method updates the city value of the document with id 1002.

```
>db.sample.update({"_id":"1002"},{"$set":{"city":"Visakhapatnam"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.sample.find()
{ "_id" : "1001", "name" : "Ram", "age" : "26", "city" : "Hyderabad" }
{ "_id" : "1002", "name" : "Rahim", "age" : 27, "city" : "Visakhapatnam" }
{ "_id" : "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai" }
```

Similarly you can replace the document with new data by saving it with same id using the save() method.

```
> db.sample.save({ "_id" : "1001", "name" : "Ram", "age" : "26", "city" :
"Vijayawada" })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.sample.find()
{ "_id" : "1001", "name" : "Ram", "age" : "26", "city" : "Vijayawada" }
{ "_id" : "1002", "name" : "Rahim", "age" : 27, "city" : "Visakhapatnam" }
{ "_id" : "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai" }
```

Updating documents using python

Similar to find_one() method which retrieves single document, the update_one() method of pymongo updates a single document.

This method accepts a query specifying which document to update and the update operation.

Example

Following python example updates the location value of a document in a collection.

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['myDB']

#Creating a collection
coll = db['example']

#Inserting document into a collection
data = [
   {"_id": "101", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "102", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "103", "name": "Robert", "age": "28", "city": "Mumbai"}
]
res = coll.insert_many(data)
print("Data inserted ......")

#Retrieving all the records using the find() method
print("Documents in the collection: ")
for doc1 in coll.find():
   print(doc1)
coll.update_one({"_id":"102"},{"$set":{"city":"Visakhapatnam"}})

#Retrieving all the records using the find() method
print("Documents in the collection after update operation: ")
for doc2 in coll.find():
   print(doc2)
```

Output

```
Data inserted ......
Documents in the collection:
{'_id': '101', 'name': 'Ram', 'age': '26', 'city': 'Hyderabad'}
{'_id': '102', 'name': 'Rahim', 'age': '27', 'city': 'Bangalore'}
{'_id': '103', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
Documents in the collection after update operation:
{'_id': '101', 'name': 'Ram', 'age': '26', 'city': 'Hyderabad'}
{'_id': '102', 'name': 'Rahim', 'age': '27', 'city': 'Visakhapatnam'}
{'_id': '103', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
```

Similarly, the ***update_many()*** method of pymongo updates all the documents that satisfies the specified condition.

Example

Following example updates the location value in all the documents in a collection (empty condition) −

```python
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['myDB']

#Creating a collection
coll = db['example']

#Inserting document into a collection
data = [
   {"_id": "101", "name": "Ram", "age": "26", "city": "Hyderabad"},
   {"_id": "102", "name": "Rahim", "age": "27", "city": "Bangalore"},
   {"_id": "103", "name": "Robert", "age": "28", "city": "Mumbai"}
]
res = coll.insert_many(data)
print("Data inserted ......")

#Retrieving all the records using the find() method
print("Documents in the collection: ")
for doc1 in coll.find():
   print(doc1)
```

```
coll.update_many({},{"$set":{"city":"Visakhapatnam"}})

#Retrieving all the records using the find() method
print("Documents in the collection after update operation: ")
for doc2 in coll.find():
  print(doc2)
```

Output

```
Data inserted ......
Documents in the collection:
{'_id': '101', 'name': 'Ram', 'age': '26', 'city': 'Hyderabad'}
{'_id': '102', 'name': 'Rahim', 'age': '27', 'city': 'Bangalore'}
{'_id': '103', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
Documents in the collection after update operation:
{'_id': '101', 'name': 'Ram', 'age': '26', 'city': 'Visakhapatnam'}
{'_id': '102', 'name': 'Rahim', 'age': '27', 'city': 'Visakhapatnam'}
{'_id': '103', 'name': 'Robert', 'age': '28', 'city': 'Visakhapatnam'}
```

Python MongoDB - Limit

While retrieving the contents of a collection you can limit the number of documents in the result using the limit() method. This method accepts a number value representing the number of documents you want in the result.

Syntax

Following is the syntax of the limit() method −

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

Example

Assume we have created a collection and inserted 5 documents into it as shown below −

```
> use testDB
switched to db testDB
> db.createCollection("sample")
{ "ok" : 1 }
> data = [
  ... {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
  ... {"_id": "1002", "name": "Rahim", "age": 27, "city": "Bangalore"},
  ... {"_id": "1003", "name": "Robert", "age": 28, "city": "Mumbai"},
  ... {"_id": "1004", "name": "Romeo", "age": 25, "city": "Pune"},
  ... {"_id": "1005", "name": "Sarmista", "age": 23, "city": "Delhi"},
  ... {"_id": "1006", "name": "Rasajna", "age": 26, "city": "Chennai"}
]
> db.sample.insert(data)
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 6,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

Following line retrieves the first 3 documents of the collection.

```
> db.sample.find().limit(3)
{ "_id" : "1001", "name" : "Ram", "age" : "26", "city" : "Hyderabad" }
{ "_id" : "1002", "name" : "Rahim", "age" : 27, "city" : "Bangalore" }
{ "_id" : "1003", "name" : "Robert", "age" : 28, "city" : "Mumbai" }
```

Limiting the Documents Using Python

To restrict the results of a query to a particular number of documents pymongo provides the ***limit()*** method. To this method pass a number value representing the number of documents you need in the result.

Example

Following example retrieves first three documents in a collection.

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

#Getting the database instance
db = client['l']

#Creating a collection
coll = db['myColl']
```

```
#Inserting document into a collection
data = [
  {"_id": "1001", "name": "Ram", "age": "26", "city": "Hyderabad"},
  {"_id": "1002", "name": "Rahim", "age": "27", "city": "Bangalore"},
  {"_id": "1003", "name": "Robert", "age": "28", "city": "Mumbai"},
  {"_id": "1004", "name": "Romeo", "age": 25, "city": "Pune"},
  {"_id": "1005", "name": "Sarmista", "age": 23, "city": "Delhi"},
  {"_id": "1006", "name": "Rasajna", "age": 26, "city": "Chennai"}
]
res = coll.insert_many(data)
print("Data inserted ......")

#Retrieving first 3 documents using the find() and limit() methods
print("First 3 documents in the collection: ")
for doc1 in coll.find().limit(3):
  print(doc1)
```

Output

```
Data inserted ......
First 3 documents in the collection:
{'_id': '1001', 'name': 'Ram', 'age': '26', 'city': 'Hyderabad'}
{'_id': '1002', 'name': 'Rahim', 'age': '27', 'city': 'Bangalore'}
{'_id': '1003', 'name': 'Robert', 'age': '28', 'city': 'Mumbai'}
```