**PyMongo**

*PyMongo* is a Python module for working with MongoDB in Python.

**Installing PyMongo**

The following command is used to install PyMongo.

$ sudo pip install pymongo

We install PyMongo with pip.

**Creating a MongoDB database**

The mongo tool is an interactive JavaScript shell interface to MongoDB, which provides an interface for systems administrators as well as a way for developers to test queries and operations directly with the database.

$ mongo testdb

MongoDB shell version: 2.6.10

connecting to: testdb

> show dbs

admin (empty)

local 0.078GB

test 0.078GB

testdb 0.078GB

We create a testdb database.

**PyMongo create collection**

In the first example, we create a new collection. MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

create\_collection.py

#!/usr/bin/python3

from pymongo import MongoClient

cars = [ {'name': 'Audi', 'price': 52642},

{'name': 'Mercedes', 'price': 57127},

{'name': 'Skoda', 'price': 9000},

{'name': 'Volvo', 'price': 29000},

{'name': 'Bentley', 'price': 350000},

{'name': 'Citroen', 'price': 21000},

{'name': 'Hummer', 'price': 41400},

{'name': 'Volkswagen', 'price': 21600} ]

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

db.cars.insert\_many(cars)

The example creates a new cars collection. It contains eight documents.

cars = [ {'name': 'Audi', 'price': 52642},

{'name': 'Mercedes', 'price': 57127},

{'name': 'Skoda', 'price': 9000},

{'name': 'Volvo', 'price': 29000},

{'name': 'Bentley', 'price': 350000},

{'name': 'Citroen', 'price': 21000},

{'name': 'Hummer', 'price': 41400},

{'name': 'Volkswagen', 'price': 21600} ]

This Python dictionary stores eight records to be inserted into the MongoDB collection.

client = MongoClient('mongodb://localhost:27017/')

MongoClient is used to communicate with MongoDB. We pass MongoClient a host name and a port number.

db = client.testdb

We get a reference to the testdb database.

db.cars.insert\_many(cars)

With insert\_many() method, we insert eight documents into the cars collection, which is automatically created as well.

> db.cars.find()

{ "\_id" : ObjectId("5b41eb21b9c5d915989d48a8"), "price" : 52642, "name" : "Audi" }

{ "\_id" : ObjectId("5b41eb21b9c5d915989d48a9"), "price" : 57127, "name" : "Mercedes" }

{ "\_id" : ObjectId("5b41eb21b9c5d915989d48aa"), "price" : 9000, "name" : "Skoda" }

{ "\_id" : ObjectId("5b41eb21b9c5d915989d48ab"), "price" : 29000, "name" : "Volvo" }

{ "\_id" : ObjectId("5b41eb21b9c5d915989d48ac"), "price" : 350000, "name" : "Bentley" }

{ "\_id" : ObjectId("5b41eb21b9c5d915989d48ad"), "price" : 21000, "name" : "Citroen" }

{ "\_id" : ObjectId("5b41eb21b9c5d915989d48ae"), "price" : 41400, "name" : "Hummer" }

{ "\_id" : ObjectId("5b41eb21b9c5d915989d48af"), "price" : 21600, "name" : "Volkswagen" }

We verify the data with mongo tool.

**PyMongo list collections**

With collection\_names(), we get list available collections in the database.

list\_collections.py

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

print(db.collection\_names())

The example prints collections in the testdb database.

**PyMongo drop collection**

The drop() method removes a collection from the database.

drop\_collection.py

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

db.cars.drop()

The example removes the cars collection from the testdb database.

**PyMongo running commands**

We can issue commnads to MongoDB with command(). The serverStatus command returns the status of the MongoDB server.

server\_status.py

#!/usr/bin/python3

from pymongo import MongoClient

from pprint import pprint

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

status = db.command("serverStatus")

pprint(status)

The example prints a lengthy servers status.

The dbstats command returns statistics that reflect the use state of a single database.

db\_stats.py

#!/usr/bin/python3

from pymongo import MongoClient

from pprint import pprint

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

print(db.collection\_names())

status = db.command("dbstats")

pprint(status)

The example prints the database statistics of testdb.

**PyMongo cursor**

The find methods return a PyMongo cursor, which is a reference to the result set of a query.

cursor.py

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

cars = db.cars.find()

print(cars.next())

print(cars.next())

print(cars.next())

cars.rewind()

print(cars.next())

print(cars.next())

print(cars.next())

print(list(cars))

In the example, we work with a cursor.

cars = db.cars.find()

The find() method returns a PyMongo cursor.

print(cars.next())

With the next() method, we get the next document from the result set.

cars.rewind()

The rewind() method rewinds the cursor to its unevaluated state.

print(list(cars))

With the list() method, we can transform the cursor to a Python list. It loads all data into the memory.

**PyMongo read all data**

In the following example, we read all records from the collection. We use Python for loop to traverse the returned cursor.

all\_cars.py

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

cars = db.cars.find()

for car in cars:

print('{0} {1}'.format(car['name'],

car['price']))

The example prints all car names and their prices from the collection.

cars = db.cars.find()

The find() method selects documents in a collection or view and returns a cursor to the selected documents. A cursor is a reference to the result set of a query.

for car in cars:

print('{0} {1}'.format(car['name'],

car['price']))

With the Python for loop, we iterate over the result set.

$ ./all\_cars.py

Audi 52642

Mercedes 57127

Skoda 9000

Volvo 29000

Bentley 350000

Citroen 21000

Hummer 41400

Volkswagen 21600

This is the output.

**PyMongo count documents**

The number of documents is retrieved with the count() method.

count\_cars.py

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

n\_cars = db.cars.find().count()

print("There are {} cars".format(n\_cars))

The example counts the number of cars in the collection with count().

$ ./count\_cars.py

There are 8 cars

There are eight cars in the collection.

**PyMongo filters**

The first parameter of find() and find\_one() is a filter. The filter is a condition that all documents must match.

filtering.py

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

expensive\_cars = db.cars.find({'price': {'$gt': 50000}})

for ecar in expensive\_cars:

print(ecar['name'])

The example prints the names of cars whose price is greater than 50000.

expensive\_cars = db.cars.find({'price': {'$gt': 50000}})

The first parameter of the find() method is the filter that all returned records must match. The filter uses the $gt operator to return only expensive cars.

$ ./filtering.py

Audi

Mercedes

Bentley

This is the output.

**PyMongo projections**

With projections, we can select specific fields from the returned documents. The projections are passed in the second argument of the find() method.

projection.py

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

cars = db.cars.find({}, {'\_id': 1, 'name':1})

for car in cars:

print(car)

The example prints the \_id and name fields of the documents.

cars = db.cars.find({}, {'\_id': 1, 'name':1})

We can specify either including or excluding projections, not both at the same time.

$ ./projection.py

{'name': 'Audi', '\_id': ObjectId('5b41eb21b9c5d915989d48a8')}

{'name': 'Mercedes', '\_id': ObjectId('5b41eb21b9c5d915989d48a9')}

{'name': 'Skoda', '\_id': ObjectId('5b41eb21b9c5d915989d48aa')}

{'name': 'Volvo', '\_id': ObjectId('5b41eb21b9c5d915989d48ab')}

{'name': 'Bentley', '\_id': ObjectId('5b41eb21b9c5d915989d48ac')}

{'name': 'Citroen', '\_id': ObjectId('5b41eb21b9c5d915989d48ad')}

{'name': 'Hummer', '\_id': ObjectId('5b41eb21b9c5d915989d48ae')}

{'name': 'Volkswagen', '\_id': ObjectId('5b41eb21b9c5d915989d48af')}

This is the output.

**PyMongo sorting documents**

We can sort documents with sort().

sorting.py

#!/usr/bin/python3

from pymongo import MongoClient, DESCENDING

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

cars = db.cars.find().sort("price", DESCENDING)

for car in cars:

print('{0} {1}'.format(car['name'],

car['price']))

The example sorts records by price in descending order.

$ ./sorting.py

Bentley 350000

Mercedes 57127

Audi 52642

Hummer 41400

Volvo 29000

Volkswagen 21600

Citroen 21000

Skoda 9000

This is the output.

**PyMongo aggregations**

Aggregations calculate aggregate values for the data in a collection.

aggregate\_sum.py

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

agr = [ {'$group': {'\_id': 1, 'all': { '$sum': '$price' } } } ]

val = list(db.cars.aggregate(agr))

print('The sum of prices is {}'.format(val[0]['all']))

The example calculates the sum of all car prices.

agr = [ {'$group': {'\_id': 1, 'all': { '$sum': '$price' } } } ]

The $sum operator calculates and returns the sum of numeric values. The $group operator groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group.

val = list(db.cars.aggregate(agr))

The aggregate() method applies the aggregation operation on the cars collection.

$ ./aggregate\_sum.py

The sum of prices is 581769

The sum of all values is 581769.

We can use the $match operator to select specific cars to aggregate.

sum\_two\_cars.py

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

agr = [{ '$match': {'$or': [ { 'name': "Audi" }, { 'name': "Volvo" }] }},

{ '$group': {'\_id': 1, 'sum2cars': { '$sum': "$price" } }}]

val = list(db.cars.aggregate(agr))

print('The sum of prices of two cars is {}'.format(val[0]['sum2cars']))

The example calculates the sum of prices of Audi and Volvo cars.

agr = [{ '$match': {'$or': [ { 'name': "Audi" }, { 'name': "Volvo" }] }},

{ '$group': {'\_id': 1, 'sum2cars': { '$sum': "$price" } }}]

The expression uses $match, $or, $group, and $sum operators to do the task.

$ ./sum\_two\_cars.py

The sum of prices of two cars is 81642

The sum of prices of two cars is 81642.

**PyMongo limit data output**

The limit query option specifies the number of documents to be returned and the skip() option some documents.

MongoSkipLimit.java

#!/usr/bin/python3

from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

db = client.testdb

cars = db.cars.find().skip(2).limit(3)

for car in cars:

print('{0}: {1}'.format(car['name'], car['price']))

The example reads from the cars collection, skips the first two documents, and limits the output to three documents.

cars = db.cars.find().skip(2).limit(3)

The skip() method skips the first two documents and the limit() method limits the output to three documents.

$ ./limit\_documents.py

Skoda: 9000

Volvo: 29000

Bentley: 350000

This is the output of the example.

update

query = { "author": "Derrick Mwiti" }

new\_author = { "$set": { "author": "John David" } }

articles.update\_one(query, new\_author)

for article in articles.find():

print(article)

**MongoDB Delete Document**

We use the delete\_one() method to delete a document in MongoDB. The first parameter for this method is the query object of the document we want to delete. If this method finds more than one document, it deletes only the first one found. Let's delete the article with the id 5ba4cbe42e8ca029163417ce.

db.articles.delete\_one({"\_id":ObjectId("5ba4d00e2e8ca029163417d4")})

**Deleting Many Documents**

In order to delete many documents, we use the delete\_many() method. Passing an empty query object will delete all the documents.

delete\_articles = articles.delete\_many({})

print(delete\_articles.deleted\_count, " articles deleted.")