pymongo

Petr Gregor

10. 4. 2021

What is pymongo?

- Pymongo is an official MongoDB ORM
- MongoDB stores data in JSON-like documents, which makes the database very flexible and scalable.
- Python needs a MongoDB driver to access the MongoDB database.

Instalace

- pip install pymongo – instalace knihovny do Pythonu
- mongodb: https://docs.mongodb.com/manual/installation/ - databázový server
- nastavení cesty k databázi:
  - "C:\Program Files\MongoDB\Server\4.2\bin\mongod.exe" –dbpath d:\test\mongodb\data
- spuštění serveru: "C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe"

Vytvoření databáze

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

Vytvoření databáze

```
import pymongo
# vytvoření klienta
# funkce: pymongo.MongoClient( "IPAddress" , port)
# nebo pymongo.MongoClient( "mongodb://IPAddress:port/")
myclient = pymongo.MongoClient( "127.0.0.1" , 27017)
# myclient = pymongo.MongoClient("mongodb://localhost:2701

# vytvoření databáze
# funkce: klient.nazevDatabeze nebo klient["nazevDatabaze"]
mydb = myclient
# mydb = client["test_db"]

print(mydb) # výpis informací o databázi
print(myclient) # výpis databází
# test existence databáze
dblist = myclient
if "mydatabase" in dblist:
  print("The database exists.")
```

Vytvoření kolekce

- 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.

```
import pymongo

myclient = pymongo.MongoClient( "127.0.0.1" , 27017)
mydb = myclient["mydatabase"]

mycol = mydb["customers"]
```

- Important: In MongoDB, a collection is not created until it gets content!

Insert Into Collection

- 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.
- The insert_one() method returns a InsertOneResult object, which has a property, inserted_id, that holds the id of the inserted document.
- If you do not specify an _id field, then MongoDB will add one for you and assign a unique id for each document.

Insert Into Collection

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1" , 27017)
mydb = myclient.test_db

collection = mydb.test_collection

response = collection.insert_one({ "test" : "data" , "numbe

print(response.inserted_id)
```

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:
- 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

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1" , 27017)
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "name": "Amy", "address": "Apple st 652"},
  { "name": "Hannah", "address": "Mountain 21"},
  { "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)
```

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.

Insert Multiple Documents, with Specified IDs

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1" , 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"}
]

x = mycol.insert_many(mylist)

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

Find One

- 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.
- 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.

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1" , 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.

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1" , 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.

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1" , 27017)
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

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

Return Only Some Fields

- 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:

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1" , 27017)
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

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

Test databáze

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1" , 27017)
mydb = myclient.test_db
print(mydb)

collection = mydb.test_collection
print(collection)

response = collection.insert_one({ "test" : "data" , "numbe
print(response.inserted_id)

print(collection.find_one())
print(collection.find_one({ "number" : {"$gt": 1}}))
```

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.

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 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"}:

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 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"}:

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 27017)
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

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

mydoc = mycol.find(myquery)

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

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).

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 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

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 27017)
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

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

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

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.

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 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.

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 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:

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 27017)
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.delete_many({})

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

Delete Collection

- You can delete a table, or collection as it is called in MongoDB, by using the drop() method.
- The drop() method returns true if the collection was dropped successfully, and false if the collection does not exist.

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 27017)
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mycol.drop()
```

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.

Update Collection

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 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.

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 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.")
```

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.

```
import pymongo

myclient = pymongo.MongoClient("127.0.0.1", 27017)
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

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

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

Exercise 1

- Use data in medical-data.json to create a new collection: medicaldata
- Find all rows with procedure_code equal 0F1F4ZC
- Find patient with patient_id equal 74 , print his full name
- Find a procedure performed on 2019-05-24T01:52:37.000Z and update its procedure code to 0F1F4ZC