Topic: Mongo primer

A short primer on Mongo using PyMongo

Dr. Daniel Yellin

Mongo, a NoSQL database

NoSQL databases are a good fit for programs that handle diverse data types.

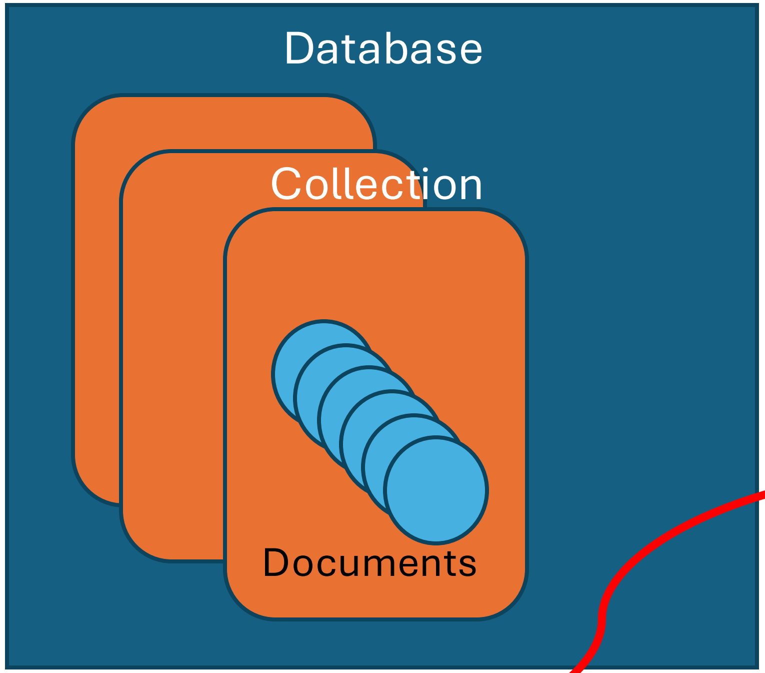
- Because the data is stored in a JSON-like format, and has a flexible schema, it is easy for developers to use, to integrate with and evolve their programs.
- It is highly scalable.

Assignment #2 requires running a database in a container

- We illustrate using the pymongo package that provides a python driver for MongoDB. Drivers are available for other programming languages.
- However, it is not a prerequisite for assignment #2 – you can use any DB you want.

MongoDB

- MongoDB is organized around:
  - **Documents**: a BSON document, very similar to JSON. A document is simply a set of field-and value pairs.
  - **Collections**: a set of documents. Similar to a table in relational DBs.
    - Unlike a relational table, each document in a collection can have different fields.
  - **Databases**: A database holds one or more collections of documents.



```
{
 "_id": 1,
 "name": "AC3 Phone",
"colors" : ["black", "silver"],
 "price" : 200
}
```

Every document must have an ”_id” field

Example document

Mongo _id field

Every Mongo document must have an **_id** field. The value of this field is either provided by the programmer when inserting the record into a Mongo DB collection. If it is not provided by the programmer, it is automatically assigned a unique value by Mongo.

- If the programmer assigns the value, it can be any unique identifier, including a number.
- If Mongo assigns the value, it is a unique id of type ObjectId. **You can convert to and from string and ObjectId representations.**
- The _id field is always the first field in the document.
- See this article for an explanation of ObjectIds and how likely it is that Mongo generates 2 Ids that are the same.

IDs and assignment #2

IDs in REST requests for assignment #2 (like #1) must be strings.

- If you use default Mongo ObjectIds for the “_id” field, then you convert between ObjectIds and REST string IDs. Benefit: Mongo manages the IDs for you.
- If you use your own string IDs, then you must manage the IDs. Benefit: you need not convert between ObjectIds and strings.
- You can also use, say integers, for IDs. Then you must manage the IDs and convert between the string representation of the ID its integer representation.
- If you manage your own IDs, make sure you can continue to create unique IDs even if the system crashes and then comes back up.

Setting up the client to use Mongo

The python code on the right uses the pymongo package.

It assumes mongo is running in a container and the name “mongo” is the one assigned by docker-compose.yml to the mongo service and that mongo is listening on the port 27017.

1. The first line creates a mongo client connected to the mongo service running in a container.
2. The second line creates a mongo DB called *myDB*.
3. The third line creates a collection *inventory* in *myDB*. Methods on that collection are done via the python variable `inv`.

```
client =
pymongo.MongoClient(

"mongodb://mongo:27017/")


db = client[“myDB”]


inv = db[“inventory”]
```

Warning. You cannot use “`localhost`” in place of “`mongo`” in the example above. You must use the name given in the docker-compose.yml, in my case “`mongo`”.

Note: we are hardcoding the default Mongo port 27017 into our code. Not a good practice. One should set this an an environment variable outside of the code.

The inventory collection

Each document in the *inventory* collection will have the form:

{

”_id”: _id,

“name”: *name*,

“info”: *description*

}

name is a string. *description* is itself a JSON record of the form:

{

“size”: *size*,

“color”: *color*

}

where *size* is an integer and *color* is a string.

Inserting and deleting documents

1. To insert a document into the collection, use the `insert_one` method on a collection object. You give it the JSON for the document to be inserted.
2. To delete a document from the collection, use the delete_one method. You provide the value of field (or fields) identifying the document to be deleted. You can check from the response that the document was successfully deleted.

```
newdoc = {
              “_id”: 1,
              “name”:
“slacks”,
              “info”: {
           “size”: 18,
           “color”: ”blue”}
            }
response =
inv.insert_one(newdoc)


resp = inv.delete_one({"_id":
8})
if resp.deletedCount==0:
     print(“document not
deleted”)
```

In this example, the programmer provides the _id field with integer values.

Finding documents

To find a the document into the collection, use the `find_one` method on a collection object.

1. Find a document whose “_id” field equals 3. You can check if a document was found or not by checking if the return value is None.
2. Find a document whose “info” field is a JSON document with a field “size” whose value is 18 and then retrieve the value of its “name” & “size” fields.
3. The `find({})` method returns all items in the collection

```
doc = inv.find_one({“_id”:3})
if doc is None:
 ...


doc =
inv.find_one({“info.size”:18}
)
name = doc[“name”]
size = doc[“info”][“size”]
…


cursor = inv.find({})
items = list(cursor)
for item in items:

      size =
item[“info”][“size”]

      …
```

Finding and updating documents (cont)

1. Find those document whose “_id” field is great than or equal to 3.

2. Update a document whose “_id” field is equal to 8, and update its name field to be “shirt”.

```
cursor = inv.find({"_id":
{"$gte": 3}})
```

```
result =
inv.update_one({"_id": 8},
{"$set": {”name": “shirt”}})
```

Further reading

- https://www.mongodb.com/docs/
- https://pymongo.readthedocs.io/en/stable/