Southern New Hampshire University

CS340 Python Driver README

Walker Martin

Student, SNHU

CS-340-T5530 Client/Server Dev

Professor Tad Kellogg

June 1, 2023

Southern New Hampshire University

CS340 Python Driver README

About the Project/Project Title

The name of this python module is CRUD. It uses encapsulation to store methods and attributes within a class called AnimalShelter. The class manipulates the 'animals' collection within 'AAC', a specific MongoDB database storing information about an animal shelter. The python module includes a constructor to establish communications with the database through a premade user account. Other than the constructor, the module includes functions for each of the CRUD operations such as create, read, update and delete. In addition to the CRUD functions it includes an explain function to analyze query performance and a createIndex to create simple and complex indices.

Motivation

The motivation behind this python driver to gain experience manipulating a database with an external language. This creates the glue to attach the database to a user interface. Allowing us to query the database and perform operations without having to use the mongo shell or the terminal. It also created reusability by abstracting the CRUD operations to make their execution more straightforward. The explain and createIndex functions are for optimizing the queries.

Getting Started

In order to get started using this specific driver which is hardcoded for the AAC database with predetermined user credentials based on a preexisting account one would have to import a

database into mongoDB locally, name the db AAC and collections animals. Then they would have to use an admin account to create a user account with matching credentials USER aacuser and PASS SNHU1234 with read and write permissions then probably adjust the host and port accordingly. For more detail look at the constructor provided towards the beginning of the usage section.

Installation

The tools needed for using this python driver are straight forward, you need to have pymongo installed which you can install with the command 'pip install pymongo'. Other than that you'll need python 3 installed at least version 3.9. You'll also need mongoDB installed on your machine in order to import the CSV or json to create the database that the driver will be manipulating. To test the module you'll need access to jupyter notebook before using this to create a frontend.

Usage

To use the python driver you will first have to ensure that your testing script and the python module are located within the same directory, otherwise you'll have to specify which directory the module is located in for the import statement to work. Here is an example of importing the python driver module from the same directory as the testing script.

```
In [1]: # Here I import my CRUD python module that has read and create functions
        from CRUD import AnimalShelter
```

Southern New Hampshire University

Now that you have imported the class AnimalShelter from the CRUD python module you can initialize an instance of the class AnimalShelter by creating an object. By doing this you'll be establishing a connection to the database with the user account credentials.

```
In [2]: # This initiallized an object AC with the animalShelter constructor
        AC = AnimalShelter()
```

This object initialization calls the constructor and sets AC to be used with the function, here is an image of the constructor.

```
def __init__(self): #Default constructor established connection to the DB using the user account information previously set
    # Local Variables for the method
    USER = 'aacuser' # user account name
    PASS = 'SNHU1234' # user account password
    HOST = 'nv-desktop-services.apporto.com' # mongo specified host
    PORT = 31237 # mongo specified port
    DB = 'AAC' # database name
    COL = 'animals' # database collection name

    # Class Instance Variables to initialize connection to DB
    self.client = MongoClient('mongodb://%s:%s@%s:%d' % (USER, PASS, HOST, PORT))
    self.database = self.client['%s' % (DB)]
    self.collection = self.database['%s' % (COL)] # this allows us to avoid specifying the collections name in the query
```

From here you now have connection with the database specified in the constructor of the python driver module. Through the object AC you may use the dot operator to access any of the methods that the py module includes such as create, read, update, delete, explain or createIndex.

Here is an example of how to use the create function with the newly initialized object. Keep in mind that all of the AnimalShelter methods require dictionaries as their parameters for proper queries, otherwise they will not execute. So here is an example of how to make a dictionary in python.

```
In [8]: # creating a data dictionary for insertion of a document, using my dog info as reference
        myDogsData = {
            'name': 'Nyx',
            'breed': 'Pug',
            'date_of_birth': '02-28-2023',
            'color': 'Black',
            'animal_type': 'Dog'
        }
```

Note that the database does have many more fields than what is specified through this dictionary, however for the queries to work you don't need to use every field and they do not need to be in any specific order. For this instance of creating/inserting a new document, any field not specified will be set to null. Here is how to use this dictionary with the create function.

```
In [9]: # Here I create/insert the document into the DB
        # I reran the code here and accidentily made two documents, delete would come in handy here
        AC.create(myDogsData)
Out[9]: True
```

Note that after a successful insertion or creation of a document the boolean returned will be True and if an error occurs then False will be returned. If there is no parameter passed neither true or false will be returned and 'Parameter is empty' will be printed instead. Here is the image of the code for the create method.

```
def create(self, data): # data is a dictionary with all major fields from AAC animals collection
    if data is not None:
        data['_id'] = ObjectId() # sets the primary key to a unique id using bson.objectid module to ensure proper queries
        result = self.collection.insert_one(data) # executes the insert function
        return True if result.inserted_id else False
        # checks that the result variable has a inserted_id, bool auto sets with successful insert function
    else:

        raise Exception("Parameter is empty")
```

Moving onto the read function, this also requires an input of a dictionary. You can either pass a variable storing a dictionary or a dictionary itself. To specify the query results for more accuracy you can pass multiple fields into the read function. Here is an example of how to pass a simple dictionary into the method. This dictionary happens to use a field which already has an index which will be elaborated on later.

Southern New Hampshire University

```
In [6]: # Creating a second dict for what I wwould through aggregation to be a popular breed in the db
        d2 = {"breed": "Domestic Shorthair Mix"}

In [7]: # executing the read with a popular breed yeilds vast results as expected
        AC.read(d2)
```

```
Out[7]: [{'_id': ObjectId('6469039e134e8848ea1b6b41'),
          'rec_num': 2,
          'age_upon_outcome': '1 year',
          'animal_id': 'A725717',
          'animal_type': 'Cat',
          'breed': 'Domestic Shorthair Mix',
          'color': 'Silver Tabby',
          'date_of_birth': '2015-05-02',
          'datetime': '2016-05-06 10:49:00',
          'monthyear': '2016-05-06T10:49:00',
          'name': '',
          'outcome_subtype': 'SCRP',
          'outcome_type': 'Transfer',
          'sex_upon_outcome': 'Spayed Female',
          'location_lat': 30.6525984560228,
          'location_long': -97.7419963476444,
          'age_upon_outcome_in_weeks': 52.9215277777778},
         {'_id': ObjectId('6469039e134e8848ea1b6b47'),
          'rec_num': 10,
```

One thing to note is the first return value is the primary key _id. It doesn't matter if you pass that field and set that key because the python driver with create method inserts a new objectID to ensure that the primary keys are unique. Refer back to the create function example above and notice how the myDogData dictionary does not include a _id field. Then look at the code where ObjectID sets _id for the query. Now look at what happens when we query for that document, despite not including the primary key in our parameter it includes _id because the driver sets it for us in the create function.

```
In [10]: # Here I searched the db for the inserted doc based on name feild
         AC.read({'name': 'Nyx'})
```

```
Out[10]: [{'_id': ObjectId('6472e9c5f7020aa06457747b'),
           'name': 'Nyx',
           'breed': 'Pug',
           'date_of_birth': '02-28-2023',
           'color': 'Black',
           'animal_type': 'Dog'},
```

Now, if you want to pass multiple parameters into the read function to get specific returns you can do so as such.

Southern New Hampshire University

```
In [26]: # here i used a complex field to limit the results, it worked, but there are still alot that fit the query parameters
         d3 = {"breed": "Domestic Shorthair Mix", "outcome_type": "Adoption"}
         AC.read(d3)
```

Multiple parameters create more specific queries to limit the amount of documents returned. Here is the code for the read function.

```
def read(self, inspect): # inspect should be a dictionary, can include multiple fields for query specificity
    if inspect is not None:
        docs = self.collection.find(inspect) # performs the query, note self.collection is used, its set in the constructor
        result = list(docs)
        return result# if successful return query results as a list
    else:
        return [] # else return an empty list
```

Another method included in the Animal Shelter class is explain. This is similar to read but rather than returning the document it returns the execution plan for the document. This informs us of what index if any is being used and the performance aspects of the query. This is how you use the explain function with a simple index.

```
In [3]: # here i used the explain fucntion I made to verify that the index breed_1 is beinig used, which it is
        AC.explain({"breed": "Persian"})
```

```
Out[3]: {'explainVersion': '1',
         'queryPlanner': {'namespace': 'AAC.animals',
          'indexFilterSet': False,
          'parsedQuery': {'breed': {'$eq': 'Persian'}},
          'queryHash': '17252B62',
          'planCacheKey': 'D894D82F',
          'maxIndexedOrSolutionsReached': False,
          'maxIndexedAndSolutionsReached': False,
          'maxScansToExplodeReached': False,
          'winningPlan': {'stage': 'FETCH',
           'inputStage': {'stage': 'IXSCAN',
            'keyPattern': {'breed': 1},
            'indexName': 'breed_1',
            'isMultiKey': False,
            'multiKeyPaths': {'breed': []},
            'isUnique': False,
            'isSparse': False,
            'isPartial': False,
```

Similar to the other functions you can pass multiple fields into the query. Here is an example, in this case it uses a complex index because the current database has a single index for breed and a complex index for breed and outcome_type.

```
In [4]: # Here im also using explain but this time to validate that the complex index is being used including both fields.
        AC.explain({"breed": "Domestic Shorthair Mix", "outcome_type": "Adoption"})
```

Here is the code which is very similar to read, but while read turns the result cursor into a list explain turns the cursor into mongodb's explain method.

```
def explain(self, qu): # extra function to determine if the index is being used
    if qu is not None:
        res = self.collection.find(qu)
        res2 = res.explain() # here it specifies to return the execution plan, verifys index in use
        return res2
    else:
        return []
```

Moving onto the next method from the AnimalShelter class we have update. Update and delete differ from create and insert because they implement a boolean variable as a parameter to specify whether the user's intent is to work with one or multiple documents. Here is an example of the update function used working with the document we inserted earlier.

```
In [19]: # PART 2: Update & Delete functions have been added
         # testing update one functio by not passing boolean third paramater thus is set to false
         cur_name = {'name': 'Nyx'}
         updated_name = {'name': 'Nyxy'}
         AC.update(cur_name, updated_name)
Out[19]: 'The number of documents updated: 1'
```

When the function has been completed it returns the number of documents modified. Like the other functions, if no parameter is passed it will raise an exception error. For the implementation of this method you can see that the old field value and new value have been passed, no boolean has been passed. The code is written to assume that if no boolean variable is passed then it will be set to False meaning only one document will be updated. To work with multiple documents of the same field you would then have to pass True into a parameter as such.

```
In [20]: # testing update many by passing True as the boolean parameter
         AC.update(updated_name, {'name': 'dog_007'}, True)
Out[20]: 'The number of documents updated: 5'
```

That is how you use the update function to update one or multiple objects. Here is the update methods code.

```
46  def update(self, query, update_dict, update_mult=False):# update_mult if not passed anything only one update will occur
47      if query is not None and update_dict is not None: # confirms necessary varibales are not null
48          update_dict = {"$set": update_dict} #this line adds $set to the update dictionary, so when passed it doesn't need it
49          if update_mult: # branch for multiple updates
50              result = self.collection.update_many(query, update_dict) # function for mutiple documents uodated
51              return_str = f"The number of documents updated: {result.modified_count}" # returns number of modified documents
52              return return_str
53          else: # branch for singular update
54              result = self.collection.update_one(query, update_dict) # function for one document updated
55              return_str = f"The number of documents updated: {result.modified_count}"
56              return return_str
57      else:
58          raise Exception("USER ERROR: One or more parameter is empty.")
```

The delete method is similar such that you also need to pass True as the boolean variable in order to work with multiple documents. Here is an example of the delete method in use for both single and multiple documents being deleted.

```
In [3]: # testing the delete one function using name Doby
        AC.delete({'name': 'Doby'})

Out[3]: 'The number of documents removed: 1'

In [4]: # the delete one function seems to work, now we test delete many with dog_007
        k = {'name': 'dog_007'}
        AC.delete(k, True)

Out[4]: 'The number of documents removed: 4'
```

Note that if you use the delete multiple documents method but only one doc is provided by the query then it will only delete that one document. Rather than passing the integer number of documents deleted or modified by the previous method I used a string to let the user know what the integer stands for. In the update function the returned object required a modified count function and for delete object its deleted count function. Here is the code for the delete method.

```
60      def delete(self, query, delete_mult=False): # similar sytax to update function
61          if query is not None:
62              if delete_mult:
63                  result = self.collection.delete_many(query)
64                  return_str = f"The number of documents removed: {result.deleted_count}" # returns number of deleted documents
65                  return return_str
66              else:
67                  result = self.collection.delete_one(query)
68                  return_str = f"The number of documents removed: {result.deleted_count}"
69                  return return_str
70          else:
71              raise Exception("USER ERROR: Parameter is empty.")
```

The last but not least method from the module is createIndex. This method creates a new simple or complex index for the database to use during queries. Here is the implementation for a simple index.

```
In [5]: # now I'll check that the additional function createIndex works by maaking a index for name
        # note that at this point the only indexes are a simple ascending for breed and a complex for breed & outcome_type as
        # ideally this creates an alphabetical sort based on name
        AC.createIndex({'name': 1}, -1)

Out[5]: 'name_-1'
```

Note that for a simple index it's the last integer outside of the field dictionary that determines ascending or descending order, in this example it uses descending. A minor inconvenience with this method is that for complex indices it also refers to that last integer so you can't have one field ascending and one descending.

Southern New Hampshire University

```
In [9]: AC.createIndex({'name': 1, 'breed': -1}, 1)
Out[9]: 'name_1_breed_1'
```

Despite the fact that the integer next to the second field is negative it's that external integer that it passed into the order parameter of the method to determine the order of the index as a whole. Here is the code for it.

```
73      def createIndex(self, query, order):
74          if query is not None:
75              key = [(field, order) for field in query.keys()]
76              new_index = self.collection.create_index(key)
77              return new_index
78          else:
79              raise Exception("USER ERROR: Parameter is empty.")
80
```

That's how you import and use the CRUD python module with a mongoDB database.

Southern New Hampshire University

References for help if needed

Python installation -

https://www.python.org/downloads/

Mongodb installation -

https://www.mongodb.com/docs/manual/installation/

Python mongo driver -

https://pymongo.readthedocs.io/en/stable/tutorial.html

MongoDB function cheat sheet -

https://www.opentechguides.com/how-to/article/mongodb/118/mongodb-cheatsheat.html

Import a database to MongoDB -

https://www.mongodb.com/developer/products/mongodb/mongoimport-guide/

Create a user account in MongoDB-

https://www.mongodb.com/docs/manual/tutorial/create-users/

Contact Information

{ 'Name': 'Walker Martin', 'Email': 'walker.martin@snhu.edu' }