Using the PDB MongoDB in the Richarson Lab

Bradley J. Hintze

August 2, 2016

Contents

1 Introduction

Herin you will find various topics relating to the use of the MongoDB in the Richardson Lab including the MongoDB shell, updating the database, and python query scripts. Note the following conventions for distiguishing command line types. Command line arguments that begin with a $ denote the linux command line. e.g.

```
$ phenix.probe 1ubqH.pdb
```

Command line arguments that begin with a > denote the Mongodb command line. e.g.

```
> show databases
```

Note that within the text of this document database names are in **blue** and collection names are in **drak orange**.

2 Contents of the PDB MongdDB

There are two main databases that are relevant for most queries we are interested in running – **top8000_rota_data** and **pdb_info**. The collections within the **top8000_rota_data** database are duplicats of tables from the original Top8000 SQL database on c3po maintained by Bradley during his dissertation years. Collections within **top8000_rota_data** are listed in Table 1.

Table 1: Contents of **top8000_rota_data**

Collection	Contents
rsc	Real-space correlation data at the residue level.
top_8000_filtered_src	A list of the filtered residues which make up the Top8000 rotamer dataset.
versions_2	Homology clusters from the PDB.

The **pdb_info** database contains new data Bradley tried to maintain after in his postdoc. Some collections are not actively updated and a good project

would be to come up with a way to keep these collections up to date. Collections within **pdb_info** are listed in Table 2.

Table 2: Contents of **pdb_info**

Collection	Contents
experiment	Has expermental data taken straight from PDBe. Actively updated using the update_mongo script in lab_scripts.
summary	Has summary data taken straight from PDBe. Actively updated using the update_mongo script in lab_scripts.
pdb_residues	Comprehensive validation metrics on the residue level. Not actively updated.
residues_colkeys	Comprehensive validation metrics on the residue level. Not actively updated. I believe this is the one we want to keep up-to-date rather than **pdb_residues** – the difference being the colon separated '_id' – useful to do a quick lookup.
rscc	Real-space correlation info on the residue level. Not actively updated. Use **residues_colkeys** instead.
file_info	Various PDB entry level data from on of my scripts. Not actively updated.

3 Using the MongoDB shell

Currently the MongoDB in the richardson lab is maitained on Daneel. To log into the shell you will need to ssh onto daneel.

```
$ ssh user@daneel.research.duhd.duke.edu
```

You may need to have permissions set up to do this – Bradley can help you get that setup. Currently you need to be in the lab to ssh onto daneel (I think). In the future we may put this on muscle so you can be anywhere and when we do, this section should be upadted to reflect that. Now you

neet to enter the MongoDB shell. The MongoDB server should be running automatically as I set it up to start on startup. To see if the server is running:

```
$ ps aux | grep mongo

root 99 0.0 0.4 ...  /Users/bhintze/.../mongodb/bin/mongod
```

If the sever is running you can enter the MongoDB shell :

```
$ mongo
```

You should now be in the shell. To view the databases on the system:

```
> show databases
admin                 0.000GB
local                 0.000GB
pdb_info            168.198GB
test                  0.001GB
top8000_rota_data     0.230GB
```

Most of the data on individual PDB entries are in the **pdb_info** database. Top8000 data is in the **top8000_rota_data** database. To use a given database:

```
> use pdb_info
```

Now that you are using a particular database you can see what collections exist therein. You can think of a collection as an SQL table. Unlike SQL tables, collections can differ in what info they hold (except for the '_id' record). e.g. a PDB entry in the 'experiment' collection will have `resolution` if 'experimental_method' is 'X-ray diffraction' but not if it is 'Solution NMR'. The neuances relating to differing record-schemas within a given collection is beyond the scope of this document but you should be aware of it. To view the collections:

```
> show collections
experiment
file_info
pdb_residues
```

```
residues_colkeys
rscc
summary
```

3.1 Queries in the Shell Environment

You can do queries in the MongoDB shell. This isn't where you will do prodction type work but it is a place to get an idea of what type of data a given collection holds. It is also good for getting quick counts. e.g. How many PDB entries were solved via X-ray diffraction? So let's tackle this question. Let's look for a record that has to do with the experimental method – a good guess would be that this data is within the **experiment** collection in the **pdb_info** database. Assuming you are 'using' **pdb_info** in the shell, you can enter a query that looks for just one record :

```
> db.experiment.findOne()
```

Note that you typically do not want to do `db.experiment.find()` as this will return every document in the collection – similar to the following SQL query:

```
SELECT *
FROM experiment
```

which returns every column in the table. (Note that a 'column' in SQL space is essentially a 'document' in MongoDB space.) However, its ok if you accidently issue `db.experiment.find()` in the shell as MongoDB returns just 20 documents at a time and asks if you'd like to see more. When we issue the `db.experiment.findOne()` command we get one document and we get *every* record in that document. The returned ducument should be in JSON format and can be though of as a python dictionary – a 'record' consists of a key and value. The returned document should hold a record called `experimental_method` and the value is a string.

```
"experimental_method" :  "Solution NMR"
```

We have to know the exact string for X-ray diffraction in order to query it

but the one record we found returned "Solution NMR". We can ust `findOne` with a query. Queries in MongoDB are just like JSON documents (of Python dictionaries). Again, the different options for querying within MongoDB is beyond the scope of this document – Google is here to help! The query to find documents where "Solution NMR" is not the "experimental_method" looks like `{"experimental_method":{$ne:"X-ray diffraction"}}`. $ne stands for 'not equal'. This query document is placed within the parens of `findOne` or `find`.

```
> db.experiment.findOne(
{"experimental_method":{$ne:"X-ray diffraction"}})
```

Here we have given `findOne` one argument – the query. `findOne` actually takes two optional arguments, the second is a JSON document that tells the program what records to return. To return just the "experimental_method" issue this:

```
> db.experiment.findOne(
{"experimental_method":{$ne:"X-ray diffraction"}},
{"experimental_method":1})
```

Note that the '_id' record is returned by defaut. You can supress this by issuing:

```
> db.experiment.findOne(
{"experimental_method":{$ne:"X-ray diffraction"}},
{"experimental_method":1,"_id":0})
```

Or try the `find` function.

```
> db.experiment.find(
{"experimental_method":{$ne:"X-ray diffraction"}},
{"experimental_method":1})
```

which will print out the first 20 results an ask if you want to see more.

You can also count. To see how many entries are in **experiment**, issue :

```
> db.experiment.count()
118970
```

As of the date this was written, there are 118,9870 entries in **experiment**. To see how mant of these were soved via X-ray diffraction enter the query as we did before when using `findOne` and `find`.

```
> db.experiment.count(
{"experimental_method" :  "X-ray diffraction"})
106306
```

As of the date this was written, there are 106,306 entries in the collection that were solved via X-ray diffraction.

4 Interacting with MongoDB with Python

4.1 Installing pymongo

In order to interact with the MongoDB with Python you need to install pymongo. I reccomend installing from source so you install the dependencies with you PHENIX python. For up-to-date instructions, consult Google. The essentials are:

```
$ git clone
   git://github.com/mongodb/mongo-python-driver.git pymongo
$ cd pymongo/
$ python setup.py install
```

If you want pymongo available with PHENIX tools, ensure you source PHENIX before doing the above install.

4.2 Connecting to MongoDB on daneel

Interacting with Mongo in python is pretty intuitive. First you need to connect and set the database.

```
from pymongo import MongoClient

# connect to MongoDB on daneel
uri = "mongodb://user:psw@daneel.research.duhs.duke.edu/"
client = MongoClient(uri)

# get desired database
database = 'pdb_info'
db = getattr(client,database)
```

Now we're ready to query the database much like we did in the shell. This time queries are native Python dictionaries. A count query looks like:

```
query = {"experimental_method":"X-ray diffraction"}
n = db.experiment.count(query)
```

Now we can put it all together.

```
from pymongo import MongoClient
import getpass

# get Mongo credentials
msg = "Please enter your Mongo username on daneel:"
user = raw_input(msg)
print "Please enter the Mongo password on daneel for %s:" % user
psw = getpass.getpass()

# connect to MongoDB on daneel
uri = "mongodb://%s:%s@daneel.research.duhs.duke.edu/"
client = MongoClient(uri% (user,psw))

# get desired database
database = 'pdb_info'
db = getattr(client,database)

# Now you're ready to query the database
expmet = "X-ray diffraction"
query = {"experimental_method" : expmet}
print "Your query is:\n  %s" % query
n = db.experiment.count(query)
msg = 'There are %i entries in the expriment collection that
    were solved via %s'
print msg % (n,expmet)
```

Now let's get a list of PDBs that are above 1 Å resolution. Here I'll just list the first 10.

```
high_resolution = 1.0
query = {"experimental_method" : "X-ray diffraction"}
query['resolution'] = {'$lte':high_resolution}
projection = {"_id":1}
n = 10
pdbs = db.experiment.find(query, projection)
msg = "Here are the first %i PDBs with a resolution above %.1f"
print msg % (n,high_resolution)
for i,d in enumerate(pdbs) :
  print d['_id']['pdb_id']
  if i > n : break
```

Pretty easy. Any questions? Ask Google or Bradley. Also note that there are more examples (real ones I used to get various data for the lab) on Github under lab_scripts. These are a bit more complicated (e.g. the connection to the database is its own object) but all of the essential elements are there.