Installing MongoDB & Preparing to Run

You can download the MongoDB installer from https://www.mongodb.com/download-center?jmp=nav#community You should then run it, following default installation settings.

My default installation location was:

C:\Program Files\MongoDB\Server\3.4\bin

If your installation is not there, you can search using your files looking for "MongoDB". This is different from other programs you have probably run because there is no clear GUI. When the database program is run, it quietly runs on the computer it was launched on and can be connected to using a few different tools.

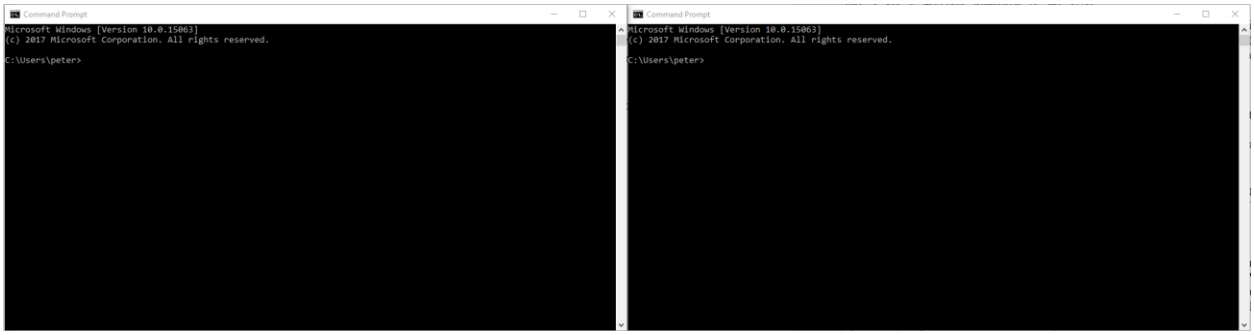
The "MongoDB\Server\3.4\bin" folder is important because in it exists all the programs that allow us to interact with the database. The two important program names in here are "mongo" and "mongod".

"mongod" will run the database server. The system requires that the folder "C:\data\db" exists on your computer, which you can do by opening "File Explorer" and creating those files. You could also type

mkdir C:\data\db

in your command prompt and it will create them. I am using some Linux/Unix conventions here. If you are unfamiliar, these are text commands that run programs on a computer. The above one creates a file in the location you tell it to(mkdir = make directory).

Running Local Instance for Windows



To run MongoDB locally, I like to open two command prompt windows side by side. One will be used to run the database. The second will be used to connect to it.

To run the database in one of the command prompt windows use the command:

"C:\Program Files\MongoDB\Server\3.4\bin\mongod"

It should ask you for access – Click ALLOW. What this command is doing is saying "run the mongod program in this directory". The mongod command is the startup command for the server. You will notice that it outputs a bunch of text about starting up and doesn't return to its normal command prompt

structure. That is because this command prompt window is now showing a connection feed to your database.

Let's use the other command prompt window to connect to this local database. To do this, we are going to use the "mongo" program, but not in the same exact way as "mongod". "mongo" is the program used to connect to running databases, and since databases are usually run on separate computers, the "mongo" command wants to know where it is connecting to. We ran the database on our local computer without feeding any information into it. That means it defaults to normal local connection information. For locally run databases the connection is usually 127.0.0.1:27017 or more easily written as localhost:27017. "localhost" and "127.0.0.1" are two ways to say "connecting to things being run on the computer you are currently connected to". If you don't fully understand why this is happening, don't worry. Many technologies leverage network/web technologies to run and this local IP is just how some programs run when you want to locally run them. The programs still use the web technologies by running a connectable service, but the programs don't open themselves up to public networks.

For class we use MySQL Workbench, which is very nice and clean. For MongoDB a similar free product exists. It is here : https://robomongo.org/

Feel free to use this from here on out. I just wanted you to use the command line for the first connection so you understand how this is running.

To connect in our other window, lets run this command:

"C:\Program Files\MongoDB\Server\3.4\bin\mongo" mongodb://localhost:27017/

What this command is doing is saying "run the mongo command in this directory" and give it the following argument, which is that "mongodb://localhost:27017/" piece. This argument piece is the location of the database. You connect to running mongodb servers using the "mongo" command and structuring the connection argument as **mongodb:// IP.ADDRESS : PORT# /.** If you look at the command prompt window that we ran the server on you can see that it has accepted a connection from 127.0.0.1:64332. That's one port on our computer connecting to the database being run on the other! Wow so exciting!

At this point, the command prompt window we ran the "mongo" command from is a shell connected to the database and the "mongod" one is a feed of what is happening to the database. We can shut down the program by pressing "Ctrl-C" in. If we did this, our connection in the other window would disconnect, and we would have to re-run the "mongod" command to start it back up.

In the "mongo" window you will see a ">" which is the computer saying "waiting for user input". We could pass through arguments to create, upload, and query data like we would with SQL, although the commands are slightly different. Don't worry about that too much. We will be looking at easier ways to write commands like this in the next section.

A bit about MongoDB compared to what we learned about MySQL:

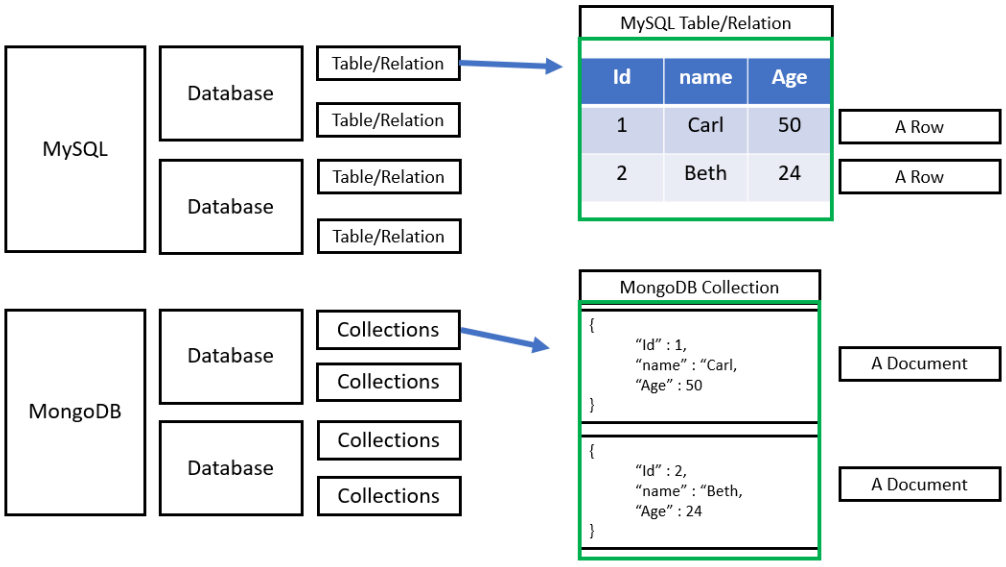
MongoDB is called a NoSQL database. It stores data in a different way than Relational Databases, but a lot of the same features exist. Some items we already understand have new names, but they do the same thing.

MySQL	MongoDB
Database	Database
Relation / Table	Collection
Row	Document

The thing that can seem the scariest at a glance is that data is stored not in "rows" but in "documents". These documents look exactly like json or python dictionaries for those familiar with that type of formatting. It basically means that each row is stored in brackers "{}" and each "field" has a "value" separated by a colon "**:"** which looks like this:

{ "field" : "value"}

See the following diagram for a side-by-side comparison.



Much of what we learned has not gone to waste thankfully. While MongoDB has its own Syntax that is different from SQL, most things are done in similar ways. This means that we can "Google" how to do things with the simple structure of "How to do ***this SQL action*** in MongoDB**"** where we replace the bolded part with our action. We will consider querying and reviewing data in the next section.

Basics with MongoDB

It's helpful to re-state that MongoDB is a computer program that's goal is to provide accuracy, speed, and efficiency when storing and retrieving a lot of data at once. We have our server running, but we

haven't created a database yet. Now is a great time to introduce the mongodb documentation, which is very clear when it comes to this tasks:

https://docs.mongodb.com/manual/mongo/

To summarize this documentation, you will always be working @ some level of the a database.

One you connect to a database by using "use database_name" you are connected to that database. Let's say our database name is "test". So, we can type

use test

If we want to create a collection (remember that this is the equivalent to creating a table in SQL) we just need to add data and it will create the collection and the document in the collection. We could do this by typing

db.testCollection.insertOne({id : 1});

If you are unfamiliar with Object Oriented notation, this command above has some intuitive things going on. When using a "." in many coding languages it means "reference the previous item and do something. Here we can break it down as "We are connected to the "test" database which is always called "db" and in that database, we want to connect to the "testCollection" collection. Insert 1 document (same as inserting a new row in SQL) with the following information.

If we create this above entry MongoDB will give it a primary key, which is always names _id. We will discuss this in the Python / R data sections, but it is good to know that one thing MongoDB forces you to have is a primary key.

```
> db.testCollection.insertOne( {id : 1} )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5945bdef74ff30475fb1c573")
}
> db.testCollection.find()
{ "_id" : ObjectId("5945bdef74ff30475fb1c573"), "id" : 1 }
```

This code highlights something very important. MongoDB does not inherently require us to dictate Primary Key's, Foreign Keys, Data Types, and Character Lengths, like in SQL. In fact, this is why many companies love MongoDB. MongoDB has been called the best solution for "unstructured data". Our entries can vary and be inconsistent in their formatting, but the database will still perform well. Some entries in a collection could have a "last name" field while others do not, and things would still work well. This may sound messy, but can be a benefit in some circumstances.

Querying in MongoDB

To query a collection we simply need to run a .find() command that collection. Remember what I said about "." notation above? That exists here too. If we want to return all the items from the "testCollection" we can write

db["testCollection"].find()

This is saying “find() everything in the testCollection of the test database”. This is very familiar to python Pandas & R dataframe notation, which will be something you use a lot as a data scientist if you haven’t already.

In SQL we usually define our query by writing a query statement. In MongoDB we do this by passing an argument in the parentheses of the .find() command in the code above. This is clearly documented and compared to SQL in the MongoDB documentation https://docs.mongodb.com/manual/tutorial/query-documents/

Python & R and MongoDB

Knowing what we do above we can perform quite a lot. Even better we can import this data directly from a database using Python or R. Using R and Python libraries (RMongo and pymongo respectively) we can connect, query, and insert documents from our database. These libraries are a second interface to this database so for now lets ignore them. If you choose R or Python for the project, I will provide code examples to this.