**How to get data from MongoDB with Python**

MongoDB is one of the most popular no SQL databases used as a backend database for web and mobile applications. Data is stored in MongoDB as [BSON](https://docs.mongodb.com/manual/reference/bson-types/), which looks like JSON files. Once you understand the way MongoDB stores data, all you need to do is to connect to the database, query the data and convert it into a structured file with the same way as we do with JSON files.

Here are two useful MongoDB tutorials for beginners: [Learn Mongo DB](https://www.tutorialspoint.com/mongodb/mongodb_overview.htm) from Tutorial Point and [Getting Started with MongoDB](https://docs.mongodb.com/getting-started/shell/) from the official website. The pymongo package makes it really easy for us to use Python with MongoDB. I recommend [PyMongo documentation](https://api.mongodb.com/python/current/tutorial.html) for getting stated. If you are having trouble with connection like authentication, [Conncet to MongoDB via the Python Driver](https://docs.mongodb.com/tutorials/connect-to-mongodb-python/#introduction) from the official website has a detailed examples and explanations that help you to troubleshoot your code.

In this post, I will use the official MongoDB example dataset and show you how to grab data from MongoDB, convert it into structured tables and load them to Postgres database.

1. **Setting up MongoDB**

First of all, let’s install MongoDB, create the test database and user credential with the right privilege and upload an example data set, restaurants into the test.

**Steps**

1. Donwload the community server from [MongoDB Download Center](https://www.mongodb.com/download-center#community) and install it by following the [instruction](https://docs.mongodb.com/getting-started/shell/installation/). The key point for Windows installation is to create a data directory to set up the environment.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

mkdir c:\data\db

[/cc]

1. Once the installation is completed, start the database. In Windows, I just use the mongod command to start the server.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

mongod

[/cc]

1. There are a few good user interface available out there. My favorite is [Robo 3T](https://robomongo.org/). MongoDB Compass comes with MonogDB installation and is good enough for a starter. By default, you will have admin and local databases for Windows installation. Open your command line interface and bring up the mongo db command line interface client by typing mongo.
2. You can use various commands to do administrative tasks and run queries from the command line interface. Let’s create a new user called admin with password as Password1.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

db.createUser( { user: "admin",

pwd: "Password1",

customData: { employeeId: 12345 },

roles: [ { role: "clusterAdmin", db: "admin" },

{ role: "readAnyDatabase", db: "admin" },

"readWrite"] },

{ w: "majority" , wtimeout: 5000 } )

[/cc]

1. Create a new database called test. The ‘use’ command is used to switch to different databases. If no database exists, it creates the database. You can also use show dbs to check all the existing databases and show collections to check all the collections existing in the current database you are in.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

use test

show dbs

show collections

[/cc]

1. You can now use the command line interface with the newly created credential.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

mongo --host localhost --port 27017 -u admin -p Password1

[/cc]

1. Download the restaurant data as a json file and then upload it to the test database with the mongoimport command.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

mongoimport --db test --collection restaurants --drop --file C:\Users\test\Downloads\primer-dataset.json

[/cc]

1. Once it’s imported, try to query some data (query examples below). If it works, we are now ready to ingest the restaurant collection with Python.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

db.restaurants.find().limit(10).pretty()

db.restaurants.find({"restaurant\_id":"30191841"}).pretty()

[/cc]

1. **Quick Starter Guide to pymongo**

With pymongo, we can connect to MongoDB and query the records. Here is the quick code example for connecting to MongoDB, querying the first 10 records and printing the result to the console in the pretty format.

[cc lang="python" tab\_size="4" lines="-1" theme="GeSHi" line\_numbers="false"]

import pymongo

from pymongo import MongoClient

import pprint

client = MongoClient('mongodb://admin:Password1@localhost:27017/test')

db = client.test

restaurants = db.restaurants

print('Total Record for the collection: ' + str(restaurants.count()))

for record in restaurants.find().limit(10):

pprint.pprint(record)

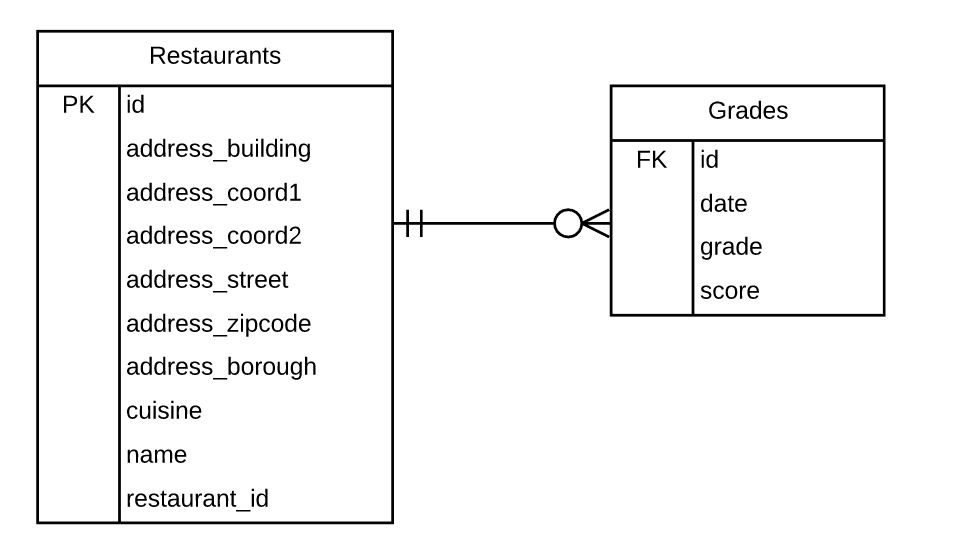
[/cc]

1. **Data Model**

Here is an example of a document (meaning a row record) from the restaurant collection (meaning table).



As you can see, the collection is denormalised as the grade fileds are nested. To convert it into a relational database table, it is better to normalise it. My suggestion is to create the main restaurant table and child grades table with id as the foreign key.



You can create table in Postgres as below. In the database, I set up a schema called mongodb.

[cc lang="sql" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

Create Table mongodb.restaurants (

id varchar(100) Primary Key,

address\_building varchar(100),

address\_coord1 varchar(100),

address\_coord2 varchar(100),

address\_street varchar(255),

address\_zipcode varchar(30),

address\_borough varchar(50),

cuisine varchar(255),

name varchar(255),

restaurant\_id varchar(100)

);

Create Table mongodb.grades (

id varchar(100) References mongodb.restaurants (id),

date timestamp,

grade varchar(100),

score varchar(100)

);

[/cc]

1. **Python Code**

It’s time for coding!

1. You need to import pymongo and psycopg2 packages.

[cc lang="python" tab\_size="4" lines="-1"]

import pymongo

from pymongo import MongoClient

import psycopg2

[/cc]

1. Grab the query results and convert it into a list that contains a list of row record. Return the record list and the header list. You need to handle the index out of bound error for corrdinates.

[cc lang="python" tab\_size="4" lines="-1"]

def create\_restaurants(results):

'''

Take the query outcome and convert to a list and return it.

It also defines headers and return it.

'''

records = []

for record in results:

tmp = []

headers = ['id','address\_building','address\_coord1','address\_coord2',\

'address\_street','address\_zipcode','address\_borough','cuisine','name',\

'restaurant\_id']

tmp.append(str(record['\_id']).split("(")[0])

address = record['address']

tmp.append(address['building'])

try:

tmp.append(str(address['coord'][0]))

except IndexError:

tmp.append('NA')

try:

tmp.append(str(address['coord'][1]))

except IndexError:

tmp.append('NA')

tmp.append(address['street'])

tmp.append(address['zipcode'])

tmp.append(record['borough'])

tmp.append(record['cuisine'])

tmp.append(record['name'])

tmp.append(record['restaurant\_id'])

records.append(tmp)

print('Preped restaurant table.')

return records, headers

def create\_grades(results):

'''

Take the query outcome and convert to a list and return it.

It also defines headers and return it.

'''

records = []

headers = ['id','date','grade','score']

for record in results:

for grade in record['grades']:

tmp = []

tmp.append(str(record['\_id']).split("(")[0])

tmp.append(str(grade['date']).split("(")[0].replace('T', ' ').replace('Z', ' '))

tmp.append(grade['grade'])

tmp.append(str(grade['score']))

records.append(tmp)

return records, headers

[/cc]

1. Create a csv file for each table with double quotes. Double quotes are important because some records contain comma.

[cc lang="python" tab\_size="4" lines="-1"]

def create\_table(records, headers, file\_path):

"Take a list of records and headers and generate csv"

f = open(file\_path, 'w', encoding='utf-8')

row\_len = len(headers)

f.write(format\_list(headers, row\_len, ',', '"'))

for record in records:

f.write(format\_list(record, row\_len, ',', '"'))

f.close()

print('CSV file successfully created: {}'.format(file\_path))

def format\_list(list, length, delimiter, quote):

counter = 1

string = ''

for record in list:

if counter == length:

string += quote + record + quote + '\n'

else:

string += quote + record + quote + delimiter

counter += 1

return string

[/cc]

1. Load csv file to postgres. I am using the copy\_expert function because it has a better control. For example, using copy\_from() will not ignore double quotes when you load data. You can check documentations [here](http://initd.org/psycopg/docs/cursor.html) for further details.

[cc lang="python" tab\_size="4" lines="-1"]

def pg\_load\_table(file\_path, table\_name):

try:

conn = psycopg2.connect("dbname='mydatahack' user='mydatahack' host='localhost' password='Password1'")

print("Connecting to Database")

cur = conn.cursor()

f = open(file\_path, "r", encoding='utf-8')

cur.execute("Truncate {} Cascade;".format(table\_name))

print("Truncated {}".format(table\_name))

cur.copy\_expert("copy {} from STDIN CSV HEADER".format(table\_name), f)

cur.execute("commit;")

print("Loaded data into {}".format(table\_name))

conn.close()

print("DB connection closed.")

except Exception as e:

print("Error: {}".format(str(e)))

[/cc]

1. Let’s put them all together and execute it.

[cc lang="python" tab\_size="4" lines="-1"]

def main():

# (1) Connect to MongoDB instance

client = MongoClient('mongodb://admin:Password1@localhost:27017/test')

# (2) Choose database and create the db object

db = client.test

# (3) Choose the collection and create Collection Object

restaurants = db.restaurants

print('Total Record for the collection: ' + str(restaurants.count()))

# (4) Retrieve all data from the restaurants collection

results = restaurants.find()

# (4-1) Transform data into a structured tables

rest\_tup = create\_restaurants(results)

rest\_records = rest\_tup[0]

rest\_headers = rest\_tup[1]

create\_table(rest\_records, rest\_headers, './restaurants.csv')

# (5) Retrive all data again for grades

results = restaurants.find()

# (5-1) Transform data for grades

grades\_tup = create\_grades(results)

grades\_records = grades\_tup[0]

grades\_headers = grades\_tup[1]

create\_table(grades\_records, grades\_headers, './grades.csv')

# (6) Load data into Postgres

pg\_load\_table('./restaurants.csv', 'mongodb.restaurants')

pg\_load\_table('./grades.csv', 'mongodb.grades')

if \_\_name\_\_ == "\_\_main\_\_":

main()

[/cc]

Once the data is loaded into the postgres, you can use it to analyse the data. For example, calculating the average grade per restaurant and rank them with SQL will be fun.

Epic!