Lab 3.3:

JSON Data Processing

By Christen Ford

March 18th, 2019

Platform Setup

This lab utilizes the following software:

* Python 3.7.0
* MongoDB 4.0.6
* Microsoft SQL Server 2017

Python is utilized by the parser to generate TSV files representing the JSON data in a format that SQL server can work with. MongoDB holds the raw JSON data, which the parser connects to so it can retrieve the JSON data and format it into the appropriate TSV schema. Microsoft SQL Server holds the data in a relational format.

Note that the dataset for this lab is too large to include in this lab report, however it can be found in the ‘./data/business100ValidForm.json’ file.

Uploading JSON Data to MongDB

The MongoDB instance is hosted locally on my machine. In the root directory of this project, there is are two Python files: ‘parser.py’ and ‘upload.py’.

‘upload.py’ is responsible for extracting the JSON data from the dataset and uploading it to the MongoDB instance. It is a very small file, and very easily understood.

This file should be the first run if one is running the parser for the first time. Otherwise, the parser will fail.

The source code for ‘upload.py’ is as follows

#! /usr/bin/python3

'''

Uploads Yelp! review data to a MongoDB instance.

'''

import json

import os

import pymongo

\_datapath = './data/business100ValidForm.json'

# get a mongodb connection

client = pymongo.MongoClient('mongodb://localhost:27017/')

# error checking to make sure we got a connection

if not client:

print('Error: Unable to acquire MongoDB connection on localhost! Are you sure it is running?')

sys.exit(1)

# get the database

db = client.yelp\_dataset

# error checking for the database

if not db:

print('Error: Unable to connect to the yelp\_dataset database on MongoDB localhost instance!')

sys.exit(1)

# get the reviews collection

reviews = db.reviews

# error checking for the collection

if not reviews:

print('Error: Unable to connect to the reviews collection in the yelp\_dataset collection!')

sys.exit(1)

# read in the data from file

if not os.path.exists(\_datapath):

raise OSError

with open(\_datapath, 'r') as f:

json\_data = json.load(f)['Business']

# try to bulk insert the database into mongodb

if reviews.insert\_many(json\_data):

print('Successfully inserted records into the reviews collection.')

Pulling and Parsing the Yelp! Review Data from MongoDB

The other file, ‘parser.py’ is responsible for pulling the Yelp! review data from the local MongoDB instance. After the data is successfully pulled from the database, it is parsed into nine different TSV files. TSV was chosen instead of CSV as I recently found out that the Bulk Insert operation provided by SQL Server does not truly support CSV files. That is, it cannot handle quoted values (which are needed by the CSV standard to differentiate between commas that appear as actual data and commas that act as delineators of the data). The parser contains methods for reading from MongoDB as well as writing out the TSV table files.

The source code for ‘parser.py’ is as follows:

#! /usr/bin/python3

import csv

import json

import os

import pymongo

import sys

\_csvpath = './TSV/'

if not os.path.exists(\_csvpath):

os.mkdir(\_csvpath)

# taken from: https://stackoverflow.com/questions/2556108/rreplace-how-to-replace-the-last-occurrence-of-an-expression-in-a-string

def rreplace(s, old, new, occurrence):

li = s.rsplit(old, occurrence)

return new.join(li)

def write\_header(path, header):

with open(path, 'w', newline='') as f:

writer = csv.writer(f, delimiter='\t', quoting=csv.QUOTE\_MINIMAL)

writer.writerow(header)

def write\_attributes(path, business\_id, attributes):

def write\_dict(sub\_path, dictionary):

if not os.path.exists(sub\_path):

write\_header(sub\_path, ['business\_id', 'key', 'value'])

with open(sub\_path, 'a', newline='') as f:

sub\_writer = csv.writer(f, delimiter='\t', quoting=csv.QUOTE\_MINIMAL)

for key, value in dictionary.items():

sub\_writer.writerow([business\_id, key, value])

if not os.path.exists(path):

write\_header(path, ['business\_id', 'attribute', 'value'])

with open(path, 'a', newline='') as f:

writer = csv.writer(f, delimiter='\t', quoting=csv.QUOTE\_MINIMAL)

for attribute, value in attributes.items():

if type(value) == dict:

parts = '{0}'.format(attribute.lower()).split(' ')

attrib\_id = ''

for part in parts:

attrib\_id = attrib\_id + part + '\_'

attrib\_id = rreplace(attrib\_id, '\_', '', 1)

# get the sub path

sub\_path = '{0}{1}.tsv'.format(\_csvpath, attrib\_id)

write\_dict(sub\_path, value)

else:

writer.writerow([business\_id, attribute, value])

def write\_hours(path, business\_id, hours):

if not os.path.exists(path):

write\_header(path, ['business\_id', 'day', 'close', 'open'])

with open(path, 'a', newline='') as f:

writer = csv.writer(f, delimiter='\t', quoting=csv.QUOTE\_MINIMAL)

for day, hour in hours.items():

writer.writerow([business\_id, day, hour['close'], hour['open']])

def write\_list(path, business\_id, items, item\_name):

if not os.path.exists(path):

write\_header(path, ['business\_id', item\_name])

with open(path, 'a', newline='') as f:

writer = csv.writer(f, delimiter='\t', quoting=csv.QUOTE\_MINIMAL)

for item in items:

writer.writerow([business\_id, item])

def read\_from\_mongo():

# get a mongodb connection

client = pymongo.MongoClient('mongodb://localhost:27017/')

# error checking to make sure we got a connection

if not client:

print('Error: Unable to acquire MongoDB connection on localhost! Are you sure it is running?')

sys.exit(1)

# get the database

db = client.yelp\_dataset

# error checking for the database

if not db:

print('Error: Unable to connect to the yelp\_dataset database on MongoDB localhost instance!')

sys.exit(1)

# get the reviews collection

reviews = db.reviews

# error checking for the collection

if not reviews:

print('Error: Unable to connect to the reviews collection in the yelp\_dataset collection!')

sys.exit(1)

yelp\_data = reviews.find({}, {"\_id":0})

if not yelp\_data:

print('Error: Unable to retrieve Yelp! review data from MongoDB!')

sys.exit(1)

#print(len(yelp\_data))

mongo\_data = []

for review in yelp\_data:

mongo\_data.append(review)

return mongo\_data

def write\_to\_tsv(mongo\_data):

if not os.path.exists(\_csvpath + 'businesses.tsv'):

with open(\_csvpath + 'businesses.tsv', 'w', newline='') as f:

writer = csv.writer(f, delimiter='\t', quoting=csv.QUOTE\_MINIMAL)

writer.writerow(['business\_id', 'full\_address', 'open', 'city', 'review\_count',

'name', 'longitude', 'state', 'stars', 'latitude', 'type'])

with open(\_csvpath + 'businesses.tsv', 'a', newline='') as f:

writer = csv.writer(f, delimiter='\t', quoting=csv.QUOTE\_MINIMAL)

for item in mongo\_data:

values = []

business\_id = item['business\_id']

for key, value in item.items():

if type(value) == dict:

# build the path

path = "{0}{1}.tsv".format(\_csvpath, key)

if key == 'hours':

write\_hours(path, business\_id, value)

elif key == 'attributes':

write\_attributes(path, business\_id, value)

elif type(value) == list:

if key == 'categories':

path = \_csvpath + 'categories.tsv'

item\_name = 'category'

elif key == 'neighborhoods':

path = \_csvpath + 'neighborhoods.tsv'

item\_name = 'neighborhood'

write\_list(path, business\_id, value, item\_name)

else:

if type(value) == str:

values.append(value.replace('\n', ' '))

else:

values.append(value)

writer.writerow([\*values])

if \_\_name\_\_ == '\_\_main\_\_':

mongo\_data = read\_from\_mongo()

write\_to\_tsv(mongo\_data)

Generate TSV Files

The parser creates nine tab-separated-value (TSV) files from the JSON data. They are found in the ‘./TSV/’ directory. The TSV files are as follows (as they are too large, their actual content is not shown here):

* ambience.tsv: Contains ambience information on each business. Fields include business\_id, key, and value.
* attributes.tsv: Contains atomic attribute information on each business. Fields include business\_id, attribute, and value.
* businesses.tsv: Contains atomic information on each business. Fields include business\_id, ful\_address, open, city, review\_count, name, longitude, state, stars, latitiude, and type.
* categories.tsv: Contains category information on each business. Fields include business\_id and category.
* good\_for.tsv: Contains good for information on each business. Fields include business\_id, key, and value.
* hours.tsv: Contains hours information on each business. Fields include business\_id, day, close, and open.
* music.tsv: Contains music information on each business. Fields include business\_id, key, and value.
* neighborhoods.tsv: Contains neighborhood information on each business. Fields include business\_id and neighborhood.
* parking.tsv: Contains parking information on each business. Fields include business\_id, key, and value.

The parser automatically breaks each JSON business record into various tables that contain atomic information. During analysis, if any field in a JSON record is not atomic (meaning it contains an array or object) it is broken down into another table. This process is then applied to each field in the array or object until the entire JSON business record has been broken down to completely atomic units. There are no null values in any of the above tables. Each record contains data.

The primary table is the businesses table with a primary key of business\_id. Every other table is related to the primary table through a foreign key constraint on the business\_id field. This allows us to easily reconstruct the original JSON data structure without any data loss.

Microsoft SQL Server SQL Source Code

The source code for creating the SQL database is as follows:

USE YELP\_REVIEWS

GO

-- create the main table

CREATE TABLE Businesses (

business\_id VARCHAR(24) NOT NULL,

full\_address VARCHAR(128) NOT NULL,

are\_open VARCHAR(10) NOT NULL,

address\_city VARCHAR(64) NOT NULL,

review\_count INT NOT NULL,

business\_name VARCHAR(128) NOT NULL,

longitude FLOAT NOT NULL,

address\_state CHAR(2) NOT NULL,

stars FLOAT NOT NULL,

latitude FLOAT NOT NULL,

business\_type VARCHAR(64) NOT NULL,

PRIMARY KEY (business\_id)

)

-- create the categories tables

CREATE TABLE Categories (

business\_id VARCHAR(24) NOT NULL,

category VARCHAR(64) NOT NULL,

PRIMARY KEY (business\_id, category),

FOREIGN KEY (business\_id) REFERENCES Businesses(business\_id)

)

-- create the good for table

CREATE TABLE GoodFor (

business\_id VARCHAR(24) NOT NULL,

gf\_key VARCHAR(24) NOT NULL,

gf\_value VARCHAR(10) NOT NULL,

PRIMARY KEY (business\_id, gf\_key),

FOREIGN KEY (business\_id) REFERENCES Businesses(business\_id)

)

-- create the hours table

CREATE TABLE BusinessHours (

business\_id VARCHAR(24) NOT NULL,

hr\_day VARCHAR(16) NOT NULL,

hr\_close TIME NOT NULL,

hr\_open TIME NOT NULL,

PRIMARY KEY (business\_id, hr\_day),

FOREIGN KEY (business\_id) REFERENCES Businesses(business\_id)

)

-- create the music table

CREATE TABLE Music (

business\_id VARCHAR(24) NOT NULL,

mus\_key VARCHAR(64) NOT NULL,

mus\_value VARCHAR(10) NOT NULL,

PRIMARY KEY (business\_id, mus\_key),

FOREIGN KEY (business\_id) REFERENCES Businesses(business\_id)

)

-- create the neighborhoods table

CREATE TABLE Neighborhoods (

business\_id VARCHAR(24) NOT NULL,

neighborhood VARCHAR(32) NOT NULL,

PRIMARY KEY (business\_id),

FOREIGN KEY (business\_id) REFERENCES Businesses(business\_id)

)

-- create the parking table

CREATE TABLE Parking (

business\_id VARCHAR(24) NOT NULL,

park\_key VARCHAR(32) NOT NULL,

park\_value VARCHAR(10) NOT NULL,

PRIMARY KEY (business\_id, park\_key),

FOREIGN KEY (business\_id) REFERENCES Businesses(business\_id)

)

-- create the ambience table

CREATE TABLE Ambience (

business\_id VARCHAR(24) NOT NULL,

amb\_key VARCHAR(32) NOT NULL,

amb\_value VARCHAR(10) NOT NULL,

PRIMARY KEY (business\_id, amb\_key),

FOREIGN KEY (business\_id) REFERENCES Businesses(business\_id)

)

-- create the attributes table

CREATE TABLE Attributes (

business\_id VARCHAR(24) NOT NULL,

attribute VARCHAR(64) NOT NULL,

attr\_value VARCHAR(32) NOT NULL,

PRIMARY KEY (business\_id, attribute),

FOREIGN KEY (business\_id) REFERENCES Businesses(business\_id)

)

GO

The source code for creating the stored procedures to import the data from the generated TSV files is as follows:

USE YELP\_REVIEWS

GO

CREATE PROCEDURE ImportFromTSV

AS

-- import the businesses data

BULK INSERT dbo.Businesses

FROM 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\businesses.tsv'

WITH (

CODEPAGE = 65001,

FIRSTROW = 2,

FIELDTERMINATOR = '\t',

ROWTERMINATOR = '\n',

ERRORFILE = 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\businesses\_errors.tsv',

TABLOCK

)

-- import the ambience data

BULK INSERT dbo.Ambience

FROM 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\ambience.tsv'

WITH (

CODEPAGE = 65001,

FIRSTROW = 2,

FIELDTERMINATOR = '\t',

ROWTERMINATOR = '\n',

ERRORFILE = 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\ambience\_errors.tsv',

TABLOCK

)

-- import the attributes data

BULK INSERT dbo.Attributes

FROM 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\attributes.tsv'

WITH (

CODEPAGE = 65001,

FIRSTROW = 2,

FIELDTERMINATOR = '\t',

ROWTERMINATOR = '\n',

ERRORFILE = 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\atributes\_errors.tsv',

TABLOCK

)

-- import the categories data

BULK INSERT dbo.Categories

FROM 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\categories.tsv'

WITH (

CODEPAGE = 65001,

FIRSTROW = 2,

FIELDTERMINATOR = '\t',

ROWTERMINATOR = '\n',

ERRORFILE = 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\categories\_errors.tsv',

TABLOCK

)

-- import the good\_for data

BULK INSERT dbo.GoodFor

FROM 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\good\_for.tsv'

WITH (

CODEPAGE = 65001,

FIRSTROW = 2,

FIELDTERMINATOR = '\t',

ROWTERMINATOR = '\n',

ERRORFILE = 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\good\_for\_errors.tsv',

TABLOCK

)

-- import the hours data

BULK INSERT dbo.BusinessHours

FROM 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\hours.tsv'

WITH (

CODEPAGE = 65001,

FIRSTROW = 2,

FIELDTERMINATOR = '\t',

ROWTERMINATOR = '\n',

ERRORFILE = 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\hours\_errors.tsv',

TABLOCK

)

-- import the music data

BULK INSERT dbo.Music

FROM 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\music.tsv'

WITH (

CODEPAGE = 65001,

FIRSTROW = 2,

FIELDTERMINATOR = '\t',

ROWTERMINATOR = '\n',

ERRORFILE = 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\music\_errors.tsv',

TABLOCK

)

-- import the neighborhoods data

BULK INSERT dbo.Neighborhoods

FROM 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\neighborhoods.tsv'

WITH (

CODEPAGE = 65001,

FIRSTROW = 2,

FIELDTERMINATOR = '\t',

ROWTERMINATOR = '\n',

ERRORFILE = 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\neighborhoods\_errors.tsv',

TABLOCK

)

-- import the parking data

BULK INSERT dbo.Parking

FROM 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\parking.tsv'

WITH (

CODEPAGE = 65001,

FIRSTROW = 2,

FIELDTERMINATOR = '\t',

ROWTERMINATOR = '\n',

ERRORFILE = 'D:\School\Big Data & Parallel Database Processing Systems\Labs\Lab3\_3\TSV\parking\_errors.tsv',

TABLOCK

)

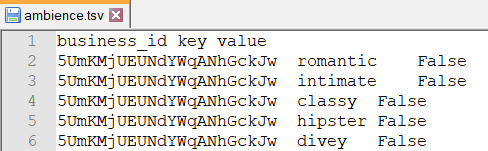
The SQL source code for importing the data into the database is as follows:

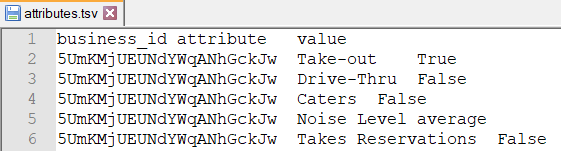
USE YELP\_REVIEWS

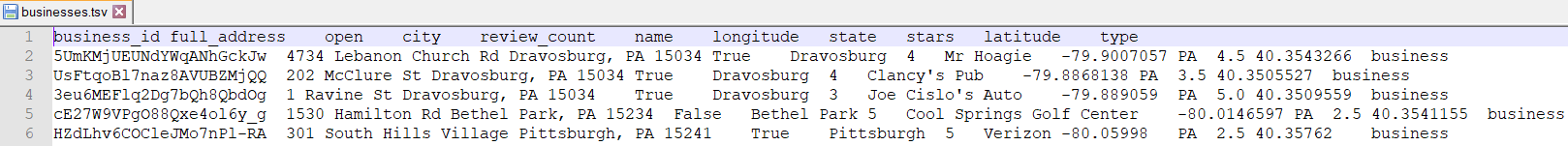
GO

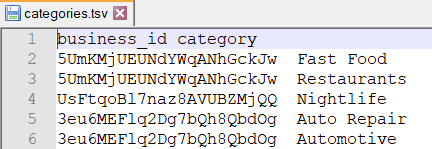
EXEC dbo.ImportFromTS

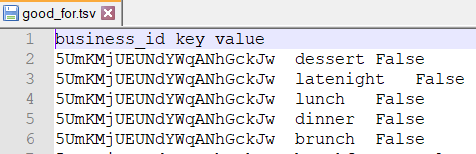
Abridged Outputs from the TSV Files

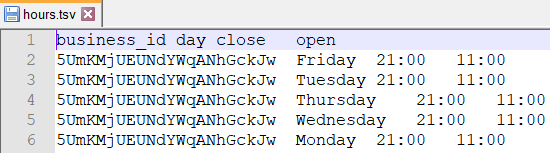


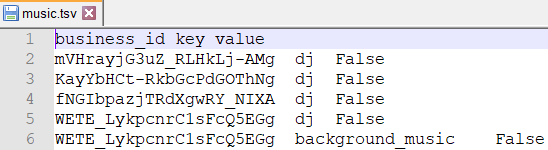


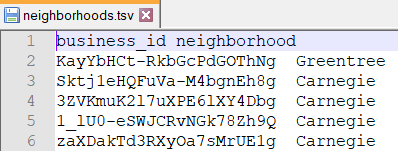


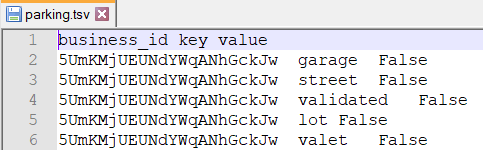












Abridged Outputs in SQL Server

