assignment02_muley_tushar_week2

December 10, 2021

Name: Muley, Tushar Assignment: Assignment02-Week2 Date: Dec 12, 2021

Assignment 2.1 - KVDB

[44]:
```
import json
from pathlib import Path
import os

import pandas as pd
import s3fs
from operator import itemgetter

#def read_cluster_csv(file_path, endpoint_url='https://storage.budsc.
↪midwest-datascience.com'):
#    s3 = s3fs.S3FileSystem(
#        anon=True,
#        client_kwargs={
#            'endpoint_url': endpoint_url
#        }
#    )
#    return pd.read_csv(s3.open(file_path, mode='rb'))

#site file
site = 'site.csv'
people = 'person.csv'
visited = 'visited.csv'
measurements = 'measurements.csv'

current_dir = Path(os.getcwd()).absolute()
results_dir = current_dir.joinpath('results')
kv_data_dir = results_dir.joinpath('kvdb')
kv_data_dir.mkdir(parents=True, exist_ok=True)

people_json = kv_data_dir.joinpath('people.json')
visited_json = kv_data_dir.joinpath('visited.json')
sites_json = kv_data_dir.joinpath('sites.json')
measurements_json = kv_data_dir.joinpath('measurements.json')
```

[2]:
```
class KVDB(object):
    def __init__(self, db_path):
        self._db_path = Path(db_path)
        self._db = {}
        self._load_db()

    def _load_db(self):
        if self._db_path.exists():
            with open(self._db_path) as f:
                self._db = json.load(f)

    def get_value(self, key):
        return self._db.get(key)

    def set_value(self, key, value):
        self._db[key] = value

    def save(self):
        with open(self._db_path, 'w') as f:
            json.dump(self._db, f, indent=2)
```

[5]:
```
def create_sites_kvdb():
    db = KVDB(sites_json)
    #df = read_cluster_csv('data/external/tidynomicon/site.csv')
    df = pd.read_csv(site) #read the file and move to dataframe
    for site_id, group_df in df.groupby('site_id'):
        db.set_value(site_id, group_df.to_dict(orient='records')[0])
    db.save()

def create_people_kvdb():
    db = KVDB(people_json)
    ## TODO: Implement code
    df = pd.read_csv(people)
    for person_id, group_df in df.groupby('person_id'):
        db.set_value(person_id, group_df.to_dict(orient='records')[0])
    db.save()

def create_visits_kvdb():
    db = KVDB(visited_json)
    df=pd.read_csv(visited)
    for visit_id, group_df in df.groupby('visit_id'):
        db.set_value(visit_id, group_df.to_dict(orient='records')[0])
    db.save()

def create_visits_kvdb():
    db = KVDB(visited_json)
    ## TODO: Implement code
    df = pd.read_csv(visited)
    for ids, group_df in df.groupby(['visit_id','site_id']):
        db.set_value(str(ids), group_df.to_dict(orient='records')[0])
    db.save()


def create_measurements_kvdb():
    db = KVDB(measurements_json)
    ## TODO: Implement code
    df = pd.read_csv(measurements)
    for ids, group_df in df.groupby(['visit_id','person_id','quantity']):
        db.set_value(str(ids),group_df.to_dict(orient='records')[0])
    db.save()
```

[8]:
```
create_sites_kvdb()
create_people_kvdb()
create_visits_kvdb()
create_measurements_kvdb()
```

Assignment 2.2 0 TinyDB

[10]:
```
from pathlib import Path
import json
import os

from tinydb import TinyDB

current_dir = Path(os.getcwd()).absolute()
results_dir = current_dir.joinpath('results')
kv_data_dir = results_dir.joinpath('kvdb')
kv_data_dir.mkdir(parents=True, exist_ok=True)
```

[13]:
```
def _load_json(json_path):
    with open(json_path) as f:
        return json.load(f)

class DocumentDB(object):
    def __init__(self, db_path):
        ## You can use the code from the previous exmaple if you would like
        people_json = kv_data_dir.joinpath('people.json')
        visited_json = kv_data_dir.joinpath('visited.json')
        sites_json = kv_data_dir.joinpath('sites.json')
        measurements_json = kv_data_dir.joinpath('measurements.json')
        self._db_path = Path(db_path)
        self._db = None
        ## TODO: Implement code
        # load JSON files
        self._person_find = _load_json(people_json)
        self._visit_find = _load_json(visited_json)
        self._site_find = _load_json(sites_json)
        self._measurements_find = _load_json(measurements_json)

        self._load_db()

        # get needed ids for site
    def _get_site(self, site_id):
        return self._site_find[site_id]
        # get person id from measurement
    def _get_measurements(self, person_id):
        measurements = []
        for values in self._measurements_find.values():
            if str(values['person_id']) == str(person_id):
                measurements.extend([values])
        return measurements
        # get visit id
    def _get_visit(self, visit_id):
        visits = self._visit_find.values()
        for value in visits:
            visit = value
            if value['visit_id'] == visit_id:
                site_id = str(value['site_id'])
                site = self._get_site(site_id)
                visit['site'] = site
                break
        return visit

    # load the tinyDb with the data
    def _load_db(self):
        self._db = TinyDB(self._db_path)
        ## TODO: Implement code
        persons = self._person_find.items() #find person 1st
        for person_id, record in persons:
            measurements = self._get_measurements(person_id) #find person in␣
↪measurement to find visit
            visit_ids = set([measurement['visit_id'] for measurement in␣
↪measurements])
            visits = [] #load visit
            for visit_id in visit_ids:
                visit = self._get_visit(visit_id) #find visit in visit
                visit['measurements'] = [
                    measurement for measurement in measurements
                    if visit_id == measurement['visit_id'] #do the match
                ]
                visits.append(visit) #add visit
            record['visits'] = visits
            self._db.insert(record) #insert in toe TinyDB
```

[14]:
```
db_path = results_dir.joinpath('patient-info.json')
if db_path.exists():
    os.remove(db_path)

db = DocumentDB(db_path)
```

[ ]:
```
##Notes:
pj=open(people)
pj=json.load(pj)
vj=open(visited)
vj=json.load(vj)
sj=open(sites)
sj=json.load(sj)
mj=open(measurements)
msmts=json.load(mj)

for key in mtms.keys():
     pkeys=key.split(', ')
     pval=pj.get(pkeys[1])
     print(pval)
     visval=vj.get(pkeys[0])
     print(visval)
     sitval=sj.get(visval[1].get('site_id'))
     print(sitval)
     mval=msmts.get(key)
     print(mval)
```

Assignment 2.3 - SQLite

[75]:
```
from pathlib import Path
import os
import sqlite3

import s3fs
import pandas as pd

current_dir = Path(os.getcwd()).absolute()
results_dir = current_dir.joinpath('results')
kv_data_dir = results_dir.joinpath('kvdb')
kv_data_dir.mkdir(parents=True, exist_ok=True)

#def read_cluster_csv(file_path, endpoint_url='https://storage.budsc.
 ↪midwest-datascience.com'):
```

```
#    s3 = s3fs.S3FileSystem(
#        anon=True,
#      client_kwargs={
#             'endpoint_url': endpoint_url
#        }
#    )
#    return pd.read_csv(s3.open(file_path, mode='rb'))

#site file
fsite = 'site.csv'
fpeople = 'person.csv'
fvisited = 'visited.csv'
fmeasurements = 'measurements.csv'

#current_dir = Path(os.getcwd()).absolute()
#results_dir = current_dir.joinpath('results')

# dsc650/assignments/assignment02/results/patient-info.db for SQL
```

0.1 Create and Load Measurements Table

[76]:
```
def create_measurements_table(conn):
    sql = """
    CREATE TABLE IF NOT EXISTS measurements (
        visit_id integer NOT NULL,
        person_id text NOT NULL,
        quantity text,
        reading real,
        FOREIGN KEY (visit_id) REFERENCES visits (visit_id),
        FOREIGN KEY (person_id) REFERENCES people (people_id)
        );
    """

    c = conn.cursor()
    c.execute(sql)

def load_measurements_table(conn):
    create_measurements_table(conn)
    df = pd.read_csv(fmeasurements) #read the file and move to dataframe
    measurements = df.values
    c = conn.cursor()
    c.execute('DELETE FROM measurements;') # Delete data if exists
    c.executemany('INSERT INTO measurements VALUES (?,?,?,?)', measurements)
```

0.2 Create and Load People Table

[77]:
```
def create_people_table(conn):
    sql = """
    CREATE TABLE IF NOT EXISTS people (
        person_id text PRIMARY KEY,
        personal_name text,
        family_name text
        );
    """
    ## TODO: Complete SQL
    c = conn.cursor()
    c.execute(sql)

def load_people_table(conn):
    create_people_table(conn)
    df = pd.read_csv(fpeople)
    person = df.values
    c = conn.cursor()
    c.execute('DELETE FROM people;')
    c.executemany('INSERT INTO people VALUES (?,?,?)', person)
    ## TODO: Complete code
```

0.3 Create and Load Sites Table

[87]:
```
def create_sites_table(conn):
    sql = """
    CREATE TABLE IF NOT EXISTS sites (
        site_id text PRIMARY KEY,
        latitude double NOT NULL,
        longitude double NOT NULL
        );
    """

    c = conn.cursor()
    c.execute(sql)

def load_sites_table(conn):
    create_sites_table(conn)
    df = pd.read_csv(fsite)
    sites = df.values
    c = conn.cursor()
    c.execute('DELETE FROM sites;')
    c.executemany('INSERT INTO sites VALUES (?,?,?)', sites)
    ## TODO: Complete code
```

0.4 Create and Load Visits Table

[79]:
```
def create_visits_table(conn):
    sql = """
    CREATE TABLE IF NOT EXISTS visits (
        visit_id integer PRIMARY KEY,
        site_id text NOT NULL,
        visit_date text,
        FOREIGN KEY (site_id) REFERENCES sites (site_id)
        );
    """

    c = conn.cursor()
    c.execute(sql)

def load_visits_table(conn):
    create_visits_table(conn)
    df = pd.read_csv(fvisited)
    visits = df.values
    c = conn.cursor()
    c.execute('DELETE FROM visits;')
    c.executemany('INSERT INTO visits VALUES (?,?,?)', visits)
    ## TODO: Complete code
```

0.5 Create DB and Load Tables

[88]:
```
db_path = results_dir.joinpath('patient-info.db')
conn = sqlite3.connect(str(db_path))
# TODO: Uncomment once functions completed
load_people_table(conn)
load_sites_table(conn)
load_visits_table(conn)
load_measurements_table(conn)

conn.commit()
conn.close()
```

0.5.1 Assignment 2.4

Modify the query so that the column order is date, event, and eventLabel instead of event, eventLabel, and date. Download the results as a JSON file and copy the results to dsc650/assignments/assignment02/results/wikidata-query.json.

[ ]:

[ ]: