**Problem 1 (Required):**

**Construct a program to import at least 2 publicly available web-based datasets into a**

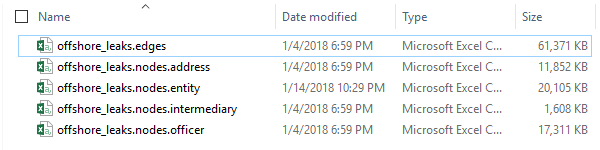
**common data structure stored in a NoSQL database. You may use any programming**

**language and any type of database.**

**Ans-**

First of all, I start with the publicly available dataset “**offshore\_leaks**” which is available on following website – https://offshoreleaks.icij.org/pages/database

The data available is in **.zip** format, so when we download it, appears as a **“csv\_offshore\_leaks.2017-12-19”** which includes following .csv files-



NOTE **-** The date in the folder name **“csv\_offshore\_leaks.2017-12-19”**, tells us when the ‘csv\_offshore\_leaks’ data updated.

After analyzing all files, what I concluded that – after converting Neo4j data to a flat format, data got converted to this many .csv files which shows the relationships between each entity. So I consider the “offshore\_leaks.nodes.entity” file, which gives the all information about leaks.

Zoom in the below image to know about the information-



From above available data, I considered only following fields –

fieldnames **=** (**"name"**, **"address"**, **"countries"**, **"incorporation\_date"**, **"type"**)

And to store in NoSQL I apply the unique index on “name” & “countries”

And while transferring CSV to NoSQL, if address is available then I converted it into Latitude & longitude and add the extra key “Country\_of\_Residence” which basically can fetch from address.

Same process followed for “US\_Consolidate\_screening\_lists” dataset form following website- <https://2016.export.gov/ecr/eg_main_023148.asp>

When we download data from above website it downloaded as a .csv file with date timestamp indicating the when it got updated. Like this –



The data in in .csv is-





I apply unique index on “name” key to remove and update the duplicate records.

So, I followed the following steps to complete my assessment-

STEP 1- Create the function which downloads/ update the .csv files form the web

**def update\_Zip**()**:  
  
 if** offshore\_leaks\_url**:** r **=** requests.**get**(offshore\_leaks\_url)  
  
 *# create beautiful-soup object* soup **= BeautifulSoup**(r.content, **'html5lib'**)  
  
 *# find all links on web-page* links **=** soup.**findAll**(href**=**re.**compile**(**"\.zip$"**))  
  
 **for** link **in** links**:** n **=** link[**'href'**].**split**(**"/"**)  
 s **=** n[**-**1].**split**(**"."**)  
 **if** s[0] **== 'csv\_offshore\_leaks':  
 if not** os.path.**isfile**(**"{}//{}.{}//offshore\_leaks.nodes.entity.csv"**.**format**(directory,s[0],s[1]))**:** file\_directory **= "{}//{}.{}"**.**format**(directory,s[0],s[1])  
 **if not** os.path.**exists**(file\_directory)**:** os.**makedirs**(file\_directory)  
  
 print(**"Dowloading and unpacking new csv- {}"**.**format**(link[**'href'**]))  
 **with urlopen**(link[**'href'**]) **as** zipresp**:  
 with ZipFile**(**BytesIO**(zipresp.**read**())) **as** zfile**:** zfile.**extractall**(file\_directory)  
 *#offshore\_file = "{}.{}".format(s[0], s[1])  
 #break* offshore\_file**= "{}.{}"**.**format**(s[0],s[1])  
 print(offshore\_file)  
 files.**append**(offshore\_file)  
 **break  
  
 if** screening\_url**:** list\_of\_files **=** glob.**glob**(**'{}US\_Consolidate\_Screening\_List//\*'**.**format**(directory)) *# \* means all if need specific format then \*.csv* latest\_file **=** max(list\_of\_files, key**=**os.path.getctime)  
 latest\_file **=** latest\_file.**split**(**"**\\**"**)[1]  
 print(latest\_file)  
  
 **with** requests.**Session**() **as** s**:** download **=** s.**get**(screening\_url)  
  
 file\_name **=** download.headers[**'Content-Disposition'**].**split**(**'/'**)[1]  
  
 **if** latest\_file **!=** file\_name**:** temp **= '{}US\_Consolidate\_Screening\_List//'**.**format**(directory)  
 **with** open(**'{}//{}'**.**format**(temp,file\_name), **'wb'**) **as** temp\_file**:** *#temp\_file.TemporaryFile(mode="w+")* temp\_file.**write**(download.content)  
  
  
 screening\_file\_path **=** file\_name  
 print(screening\_file\_path)  
 files.**append**(screening\_file\_path)  
  
  
 **else:** screening\_file\_path **=** latest\_file  
 print(screening\_file\_path)  
 files.**append**(screening\_file\_path

**return** files

STEP 2- Created the library to fetch environment variables used in main script to maintain the abstraction

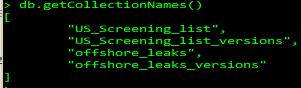
**from** pprint **import** pprint *# pprint library is used to make the output look more pretty***import** pymongo  
  
  
directory **= "C:**\\**Users**\\**Mayur**\\**Documents**\\**Assesment**\\**Sayari\_Analytics**\\**"  
  
def getInputFile**(*offshore\_file*, *screening\_file*)**:** input\_offshore **= "{}{}**\\**{}"**.**format**(directory,*offshore\_file*,**'offshore\_leaks.nodes.entity.csv'**)  
 input\_screening **= "{}**\\**US\_Consolidate\_Screening\_List**\\**{}"**.**format**(directory,*screening\_file*)  
 **return** input\_offshore, input\_screening  
  
**def getOutpuFile**()**:** output **= "{}file.json"**.**format**(directory)  
 **return** output  
  
**def getAddressFile**()**:** addrFile **= "{}address\_latlag.json"**.**format**(directory)  
 **return** addrFile  
  
**def setAddressFile**()**:** addrFile **= "{}address\_latlag.json"**.**format**(directory)  
 **return** addrFile  
  
**def getGeocodeURL**()**:** url **= "https://maps.googleapis.com/maps/api/geocode/json"  
 return** url  
  
**def getAPIKey**()**:** key **= Api\_key** *##this is secure key* **return** key  
  
**def connect\_MangoDb**()**:** *# connect to MongoDB, change the << MONGODB URL >> to reflect your own connection string* client **=** pymongo.**MongoClient**(**'mongodb+srv://USERNAME:PASSWORD@cluster0-etwif.mongodb.net/test'**)  
 db**=**client.test  
  
 *# Issue the serverStatus command and print the results* serverStatusResult**=**db.**command**(**"serverStatus"**)  
 **pprint**(serverStatusResult)  
 **return** db

STEP 3- Connect the MongoDb to store the data in NoSQL JSON format and parse each address (if exists) in to Latitude, Longitude and Country

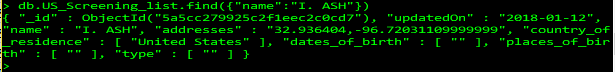
*##clean address string***def clean\_address**(*addr*)**:** x **=** re.**search**(**'\d+'**, *addr*)  
 **return** *addr*[x.**start**(0)**:**]  
  
*##Find the (Lattitude,Longitude) & Country of given address string***def find\_location**(*addr*)**:** url **=** ENV.**getGeocodeURL**()  
  
 *##Clean the address string* **if** re.**search**(**'\d+'**, *addr*)**:** x **=** re.**search**(**'\d+'**, *addr*)  
 caddr **=** *addr*[x.**start**(0)**:**]  
 *#print("clean addrs - ",caddr)* **else: return None**,**None** *## Prepare URL using parameters required to fetch location* params **=** {**'sensor': 'false'**,**'address':** caddr,**'key':**ENV.**getAPIKey**()}  
 **try:** r **=** requests.**get**(url, params**=**params)  
 results **=** r.**json**()[**'results'**]  
 location **=** results[0][**'geometry'**][**'location'**]  
 loc **=** str(location[**'lat'**]) **+ "," +** str(location[**'lng'**]) *##fetch the Lattutude and Longitude* i **=** 1  
 **while** results[0][**'address\_components'**][**-**i][**'types'**][0] **!= 'country':** i **+=** 1  
 country **=** results[0][**'address\_components'**][**-**i][**'long\_name'**] *##fetch the Country* **except** Exception **as** e**:** *#print(e)* **return None**,**None  
  
 return** loc,country

STEP 4 – Transfer each csv record to MongoDB. If record is already available then update it or insert it.

In MongoDb, I created the 4 collections which maintain the updated values and timeline values for Offshore\_leaks and Screening lists

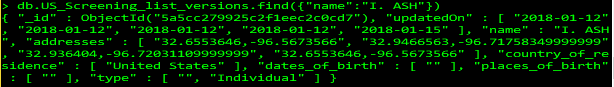


In MngoDb- “US\_Screening\_list” data structure maintain the most updated values like following JSON format-



And when it got updated then in “US\_Screening\_list\_versions” it got store in arrays of ‘updatedOn’ date and the updated value get added in the respective key’s array.

In the below example- ‘updatedOn’ get added when the address and type get changed and the changed values get added in array.



The same structure maintained for ‘offshore\_leaks’ and ‘offshore\_leaks\_versions’ to maintain the timeline of updated fields of CSV records.

**if "offshore\_leaks" not in** db.**collection\_names**()**:** *#pass  
 #db.offshore\_leaks* mdb **=** db.offshore\_leaks  
 mdb.**create\_index**([(**"name"**, 1),(**"countries"**, 1)],unique**=True**)  
*#db.offhsore\_leaks.create\_index()***if "offshore\_leaks\_versions" not in** db.**collection\_names**()**:** *#pass  
 #db.offshore\_leaks\_versions* mdb\_v **=** db.offshore\_leaks\_versions  
 mdb\_v.**create\_index**([(**"name"**, 1),(**"countries"**, 1)],unique**=True**)  
  
  
start **= True  
for** row **in** offshore\_reader**:** data**=**{**"updatedOn":** [timestr]}  
 **if** start**:** *#jsonfile.write(',\n')* start **= False  
 else:** jsonfile.**write**(**',**\n**'**)  
  
 **for** item **in** fieldnames**:  
 if** item **in** row**:** current\_addrs **=** row[item]  
 **if** item **== "address" and** current\_addrs **!= "":  
 if** current\_addrs **not in** addrs\_dict**:** loc, country **= find\_location**(row[item])  
 temp **=** {**"latlang":**loc, **"country":**country}  
 addrs\_dict[current\_addrs] **=** temp  
 **else:** loc **=** addrs\_dict[current\_addrs][**"latlang"**]  
 country **=** addrs\_dict[current\_addrs][**"country"**]  
 *#print(row[item])  
  
 #print(loc,country)* data[item],data[**"country\_of\_residence"**] **=** [loc], [country]  
  
 **else:  
 if** item **== "name" or** item **== "countries":** data[item] **=** row[item] **if** row[item] **!= "" else ""  
 else:** data[item] **=** [row[item]] **if** row[item] **!= "" else** [**""**]  
 *#data["country\_of\_residence"] = None* **if "country\_of\_residence" not in** data**:** data[**"country\_of\_residence"**] **=** [**""**]  
 *#json.dump(data, jsonfile,separators=(',',':')) ## this is the local store for json objects  
  
 ## Check if item exists otherwise update or insert into a MongoDb.collections* mdb **=** db.offshore\_leaks  
 mdb\_v **=** db.offshore\_leaks\_versions  
 mdb\_item **=** mdb.**find**({**"name":** data[**"name"**], **"countries":** data[**"countries"**]})  
 mdb\_v\_item **=** mdb\_v.**find**({**"name":** data[**"name"**], **"countries":** data[**"countries"**]})  
  
 **if** mdb\_item.**count**() **==** 0 **or** mdb\_v\_item.**count**() **==** 0**:  
 if** mdb\_item.**count**() **==** 0**:  
 try:** mdb.**insert**(data) *## inserting the each item in MongoDB collection offshore\_leaks* **except** pymongo.errors.BulkWriteError **as** e**:  
 pprint**([err **for** err **in** e.details[**'writeErrors'**] **if** err[**'code'**] **==** 11000])  
 **if** mdb\_v\_item.**count**() **==** 0**:  
 try:** mdb\_v.**insert**(data)  
 **except** pymongo.errors.BulkWriteError **as** e**:  
 pprint**([err **for** err **in** e.details[**'writeErrors'**] **if** err[**'code'**] **==** 11000])  
 **else:  
 for** result **in** mdb\_item[0**:**1]**:** set\_fields **=** {}  
 **if** result[**"address"**] **!=** data[**"address"**]**:** set\_fields[**"address"**] **=** data[**"address"**][**-**1]  
  
 **if** result[**"country\_of\_residence"**] **!=** data[**"country\_of\_residence"**]**:** set\_fields[**"country\_of\_residence"**] **=** data[**"country\_of\_residence"**][**-**1]  
  
 **if** result[**"incorporation\_date"**] **!=** data[**"incorporation\_date"**]**:** set\_fields[**"incorporation\_date"**] **=** data[**"incorporation\_date"**][**-**1]  
  
 **if** result[**"type"**] **!=** data[**"type"**]**:** set\_fields[**"type"**] **=** data[**"type"**][**-**1]  
  
 **if** any(set\_fields.**values**())**:** set\_fields[**"updatedOn"**] **=** timestr  
 mdb.**update**(  
 {**"name":** data[**"name"**], **"countries":** data[**"countries"**]},  
 {  
 **'$set':** set\_fields  
 }, upsert**=True** );  
  
 mdb\_v.**update**(  
 {**"name":** data[**"name"**], **"countries":** data[**"countries"**]},  
 {  
 **'$push':** set\_fields  
 }, upsert**=True** );

Same for “US consolidate Screening list” data –

**if "US\_Screening\_list" not in** db.**collection\_names**()**:** *# pass  
 # db.offshore\_leaks* mdb **=** db.US\_Screening\_list  
 mdb.**create\_index**([(**"name"**, 1),(**"dates\_of\_birth"**, 1)], unique**=True**)  
*# db.offhsore\_leaks.create\_index()***if "US\_Screening\_list\_versions" not in** db.**collection\_names**()**:** *# pass  
 # db.offshore\_leaks\_versions* mdb\_v **=** db.US\_Screening\_list\_versions  
 mdb\_v.**create\_index**([(**"name"**, 1),(**"dates\_of\_birth"**, 1)], unique**=True**)  
  
start **= True  
for** row **in** screening\_reader**:** data **=** {**"updatedOn":** [timestr]}  
 **if** start**:** *# jsonfile.write(',\n')* start **= False  
 else:** jsonfile.**write**(**',**\n**'**)  
  
 **for** item **in** fieldnames**:  
 if** item **in** row**:** current\_addrs **=** (row[item]).**strip**(**" "**)  
 **if** item **== "addresses" and** current\_addrs **!= "":  
 if** current\_addrs **not in** addrs\_dict**:** loc, country **= find\_location**(row[item])  
 temp **=** {**"latlang":** loc, **"country":** country}  
 addrs\_dict[current\_addrs] **=** temp  
 **else:** loc **=** addrs\_dict[current\_addrs][**"latlang"**]  
 country **=** addrs\_dict[current\_addrs][**"country"**]  
 *# print(row[item])  
  
 # print(loc,country)* data[item], data[**"country\_of\_residence"**] **=** [loc], [country]  
  
 **else:  
 if** item **== "name":** data[item] **=** (row[item]).**strip**(**" "**) **if** row[item] **!= "" else ""  
 else:** data[item] **=** (row[item]).**strip**(**" "**) **if** row[item] **!= "" else** [**""**]  
 *# data["country\_of\_residence"] = None* **if "country\_of\_residence" not in** data**:** data[**"country\_of\_residence"**] **=** [**""**]  
 *# json.dump(data, jsonfile,separators=(',',':')) ## this is the local store for json objects  
  
 ## Check if item exists otherwise update or insert into a MongoDb.collections* mdb **=** db.US\_Screening\_list  
 mdb\_v **=** db.US\_Screening\_list\_versions  
 mdb\_item **=** mdb.**find**({**"name":** data[**"name"**], **"dates\_of\_birth":** data[**"dates\_of\_birth"**]})  
 mdb\_v\_item **=** mdb\_v.**find**({**"name":** data[**"name"**], **"dates\_of\_birth":** data[**"dates\_of\_birth"**]})  
  
 **if** mdb\_item.**count**() **==** 0 **or** mdb\_v\_item.**count**() **==** 0**:  
 if** mdb\_item.**count**() **==** 0**:  
 try:** mdb.**insert**(data) *## inserting the each item in MongoDB collection offshore\_leaks* **except** pymongo.errors.BulkWriteError **as** e**:  
 pprint**([err **for** err **in** e.details[**'writeErrors'**] **if** err[**'code'**] **==** 11000])  
 **if** mdb\_v\_item.**count**() **==** 0**:  
 try:** mdb\_v.**insert**(data)  
 **except** pymongo.errors.BulkWriteError **as** e**:  
 pprint**([err **for** err **in** e.details[**'writeErrors'**] **if** err[**'code'**] **==** 11000])  
 **else:** *##fieldnames = ("name", "addresses", "dates\_of\_birth", "places\_of\_birth", "type")* **for** result **in** mdb\_item[0**:**1]**:** set\_fields **=** {}  
 **if** result[**"addresses"**] **!=** data[**"addresses"**]**:** set\_fields[**"addresses"**] **=** data[**"addresses"**][**-**1]  
  
 **if** result[**"country\_of\_residence"**] **!=** data[**"country\_of\_residence"**]**:** set\_fields[**"country\_of\_residence"**] **=** data[**"country\_of\_residence"**][**-**1]  
  
 **if** result[**"dates\_of\_birth"**] **!=** data[**"dates\_of\_birth"**]**:** set\_fields[**"dates\_of\_birth"**] **=** data[**"dates\_of\_birth"**][**-**1]  
  
 **if** result[**"places\_of\_birth"**] **!=** data[**"places\_of\_birth"**]**:** set\_fields[**"places\_of\_birth"**] **=** data[**"places\_of\_birth"**][**-**1]  
  
 **if** result[**"type"**] **!=** data[**"type"**]**:** set\_fields[**"type"**] **=** data[**"type"**][**-**1]  
  
  
 **if** any(set\_fields.**values**())**:** set\_fields[**"updatedOn"**] **=** timestr  
 mdb.**update**(  
 {**"name":** data[**"name"**], **"dates\_of\_birth":** data[**"dates\_of\_birth"**]},  
 {  
 **'$set':** set\_fields  
 }, upsert**=True** );  
  
 mdb\_v.**update**(  
 {**"name":** data[**"name"**], **"dates\_of\_birth":** data[**"dates\_of\_birth"**]},  
 {  
 **'$push':** set\_fields  
 }, upsert**=True** );

Step 5- To run this scripts for every hour, I created the task scheduler which can set by following way in Command Line –

schtasks /Create /SC HOURLY /TN offshore\_to\_MOngoDb.py /TR “C:\Python34\python.exe"

schtasks /Create /SC HOURLY /TN screening\_to\_mongoDb.py /TR “C:\Python34\python.exe "

or we can schedule the tasks using **Task Scheduler GUI**  for each hour.

I created two tasks to run the scripts in parallel, so that it will run fast.

How to improve running speed-

The running time of **offshore\_to\_MongoDb**.py is 160 minutes and for **screening\_to\_MongoDb.py** is 55 minutes at first time.

How can I minimize the execution time-

1. I am working on of machine 8 GB ram. On RAM more than 8 GB, it will run fast than current.
2. I am using could space to store the data in MondoDB. Unlimited memory on my pc to store the JSON data will extremely minimize the time.
3. I can use more dictionary data structure to store the seen address values, so that in future it won’t take time to fetch latitude and longitude using web request.
4. I already split the work by creating two python scripts to run Offshore\_leaks in one script and screening\_List in one script, which helps to run in parallel.
5. I can use the distributed framework to run the scripts by splitting big csv file to small csv files and run parallelly. It will run the whole script on an average 25% time of current time.