A REPORT
ON

Graph Data Digest Document Format (GDF)

BY

Vipin Baswan	2017A7PS0429P
Suyash Raj	2017A7PS0191P
Yashdeep Gupta	2017A7PS0114P
Abhinava Arsada	2017A7PS0028P
Sreyas Ravichandran	2017A7PS0275P

AT

Homi Bhabha Centre for Science Education (HBCSE)

A Practice School-1 station of

Birla Institute of Technology & Science, Pilani

(June 2019)

A REPORT
ON

Graph Data Digest Document Format (GDF)

BY

Vipin Baswan	2017A7PS0429P	Computer Science
Suyash Raj	2017A7PS0191P	Computer Science
Yashdeep Gupta	2017A7PS0114P	Computer Science
Abhinava Arsada	2017A7PS0028P	Computer Science
Sreyas Ravichandran	2017A7PS0275P	Computer Science

Prepared in the partial fulfilment of the

Practice School-I (BITS F221)

AT

Homi Bhabha Centre for Science Education (HBCSE)

A Practice School-1 station of

Birla Institute of Technology & Science, Pilani

(June 2019)

ACKNOWLEDGEMENTS

We would like to thank **Dr K Subramaniam**, Director, **Homi Bhabha Centre for Science Education** for providing this opportunity to us, the students of BITS Pilani, to work with the organization towards the fulfilment of its purpose of promoting science education throughout the country. We'd also like to thank **Nagarjuna G.** to allow us the chance to work on this project under his able guidance. We also extend our thanks to **Prof. Mukesh Kumar Rohil**, our Practice School Coordinator for his constant support and guidance. Additionally, we express our heartfelt gratitude to **Mr. J. B. Waghmare** for his constant support at the PS station in non-academic matters.

Finally, our humblest apologies to all others who helped us but whose names could not be mentioned in the list.

Birla Institute of Technology and Science

Pilani (Rajasthan)

Practice School Division

Station: Homi Bhabha Centre for Science Education **Centre:** Mumbai

Duration: 21 days (till the submission of the Mid-Semester report)

Date of Start: 21st May, 2019

Date of Submission: 10th June, 2019

Title of Project: Graph Data Digest Document Format (GDF)

Submitted By:

Vipin Baswan	2017A7PS0429P	Computer Science
Suyash Raj	2017A7PS0191P	Computer Science
Yashdeep Gupta	2017A7PS0114P	Computer Science
Abhinava Arsada	2017A7PS0028P	Computer Science
Sreyas Ravichandran	2017A7PS0275P	Computer Science

Name(s) of expert(s):

Nagarjuna G., faculty at Homi Bhabha Centre for Science Education

Name of the PS Faculty:

Prof. Mukesh Kumar Rohil

Key Words: GDF, Graph Databases, RDF, GraphQL, Data Digest Format, SPARQL, Renderer, 7-column format, MetaData

Project Areas: Graph Databases and querying languages

Abstract

Graph databases have always been a promising tool in increasing the querying efficiency on datasets. Hence, the prospect of data digest document format such as GDF seems very promising in today's world where datasets interact in a complex manner and quick information retrieval is of prime import.

Our project deals with developing a format called GDF and the method to convert any document format into GDF. This will assist us in quick merging of different files as graphs can be merged easily. On completion of our project, we will be able to convert any file format into GDF and also view any file data in the form of graphs (only nodes and edges)! This format is essentially NoSQL type format since there are no tables.

_____	_____
Signature of the Student	Signature of PS Faculty
_____	_____
Date	Date

TABLE OF CONTENTS

1. Acknowledgements 3
2. Abstract 4-5
3. Introduction 7
4. Main Text 8
   - 4.1. Background 8
     - 4.1.1. Tools used in the Project 8
   - 4.2. 7-Column Format 8-9
   - 4.3. Meta-Data Format 9-10
   - 4.4. Text to Graph Conversion 10-11
   - 4.5. Our Progress 11
   - 4.6. Future Prospects 11
   - 4.7. Skills Acquired 11
5. Conclusion 12
6. Appendix 13
   - 6.1. Code (Python) to generate sample input data 13-15
   - 6.2. Code (Shell script) to convert text to GDF file 15-16
   - 6.3. ReadMe file 16-17
   - 6.4. Code (Shell script) of about() function 18
   - 6.5. Code (Shell script) of separateBySub() function 19
7. References 20
8. Glossary 21

INTRODUCTION

The aim of our project is to develop a Data Digest Format which can be used to convert and represent information of any format. The format is graph based, hence the name.

The scope of our project is to:

a) Decide the format of GDF
b) Write methods to convert a text file into GDF
c) Create meta-data in the GDF format from the meta-data in the text format
d) Develop a query language (based on SPARQL) for information retrieval

Since the idea of GDF is pretty innovative and unique in itself (Credit: Nagarjuna G.), not much literature is available to us for this exact format. But a similar data format called *Resource Description Format* (RDF) already exists. It is also graph based data format. Hence, we have gone through the literatures regarding RDF (links of the online resources have been given in the References section). Also, we have generated the sample data (for testing our code) using a python code but later we will collect more data from *DBPedia* (an online platform to get 3-column formatted data for various Wikipedia pages). Also, the query language for our format is based on *SPARQL*. We are referring to the official literature available on *SPARQL* (link for the same has been provided in References Section) for building our querying engine.

We wanted to limit the dependency of our code on various platforms. Hence, we have used BASH Scripting to write our code.

Due to time constraints, we will not be able to create our own renderer but we will be using already existing D3.js renderer.

The report initially gives a basic background information on Graph based database formats. Then, the *7-column format* has been discussed in detail. Since, the format is graph based, we also need to define nodes and edges of the graph. This is done through generating the meta-data file. The format of meta-data file is also discussed in detail. After this, the report explains the flow of the project (from text to Graph) through a flow chart. Since we have limited time to complete the project, we have also stated the current progress and what we have to achieve in two separate sections. The relevant code and readMe files have been attached in the appendix. The links of various resources we referred to have been given in the references section. Few important jargons have been defined in the glossary at the end.

MAIN TEXT

1. BACKGROUND

This data format can be said to be loosely inspired from the RDF format, additionally making use of the seven-column format, which has been described below. In both of these formats, *data is stored in the form of graphs* i.e. nodes as well as edges for easier and more efficient querying of data. We also decided to additionally generate unique IDs for each one of the tuples generated as well as each entity uniquely specified by the edges and nodes. The last addition to this data format is that we shall implement a constantly self-updating metadata section of our data which cannot be accessed by the users and contains information about the type of entities stored in our data which reduces our query time to a very large extent despite requiring a very large amount of storage space.

Tools used in this project are:

a) **Shell Scripting:** *to write most of the code*
b) **RDF:** *to understand how text can be shown as graph*
c) **D3.js:** *Renderer to get graph SGV*
d) **SPARQL:** *to design querying engine for GDF*

2. 7-COLUMN FORMAT

Each tuple in the input file (can be in any format) can be viewed as an entity with a Subject, Object and a Predicate. For instance, in "*Yashdeep likes to eat ice-cream*", 'Yashdeep' is the Subject, 'ice-cream' is the Object and 'likes to eat' is the Predicate.
Let's see the graph given below:

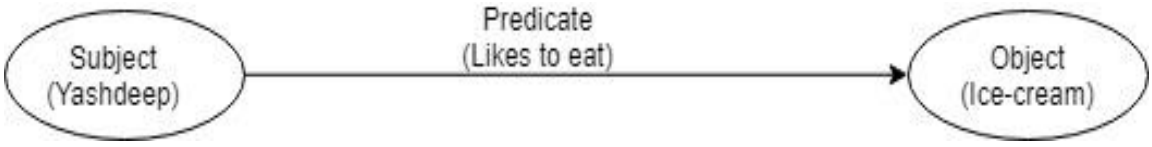

Fig.1: A graph representing the sentence "Yashdeep like to eat ice-cream"

Here are the observations:

a) Both subject and object are represented by the nodes of the graph
b) Predicate is represented by the edge of the graph
c) Both nodes and edges have some text associated with them (like 'Yashdeep' and 'Ice-cream' associated with nodes and 'Likes to eat' associated with edge)
d) The edge originates from Subject and terminates at Object. Thus, *the graph we get is always a directed graph*

e) Subject, Objects and Predicates can also have '*qualifiers*' associated with them. For example, in above graph, Subject Qualifier can be 'Person', Object Qualifier can be 'Dessert' and Predicate Qualifier can be 'Preferences'. *In short, a qualifier gives more information about the subject/object/predicate.*

The above graph corresponds to a single tuple of the GDF file. This graph is represented by a 7-column format in our GDF File. The format is:

UID | Subject | Subject_Qualifier | Predicate | Predicate_Qualifier | Object | Object_Qualifier

Hence, above graph will be represented by the following tuple in our GDF File:

1242353353|Yashdeep|Person|LikesToEat|Preferences|Ice-cream|Dessert

The UID is generated by taking the hash of Subject, Object and Predicate (appended and then delimited by space). We have used built-in 'md5sum' hash of the bash. For each tuple in the input file, a tuple is generated in the GDF file. Finally, the output will contain a file with .gdf extension.

The code for the conversion of text into GDF file is attached in the appendices section along with the ReadMe file for the code. For complete detailed explanation on how the code works and what 7-column format is, please refer to the ReadMe file.

3. META-DATA FORMAT

Along with the GDF file a meta-data GDF file is also created. For creating this file, the user has to give the meta-data text file as the input. The meta-data file will also have a Graph based file format. For instance, see the graph below:

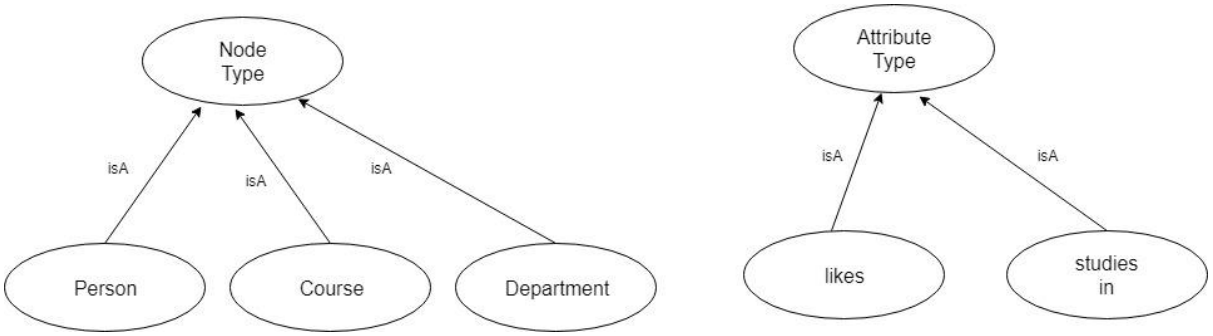

Fig. 2: An example Node and Attribute types for meta-data.gdf file

The meta-data contains two basic entities: Node type and Attribute/Predicate type. This is used to define the nodes and edges within the graph.

In the example discussed previously, "*Yashdeep likes to eat ice-cream*", Yashdeep is a Person and hence Yashdeep is a Member of Person Node type. This information can also be conveyed through the 7-column format:

UID | Subject | Subject_Qualifier | memberOf | | Node_Type

UID | Predicate | Predicate_Qualifier | memberOf | | Attribute_Type

UID | Object | Object_Qualifier | memberOf | | Node_Type

In our example,

1243453453 | Yashdeep | | memberOf | | Person

1453456564 | Ice-cream | | memberOf | | Food-items

2434536456 | Likes to eat | | memberOf | | likes

Hence, for each subject/object/predicate, our meta-data.gdf file will contain a tuple in it.

4. TEXT TO GRAPH CONVERSION

The conversion of input text file into Graph database can be represented by the following chart:

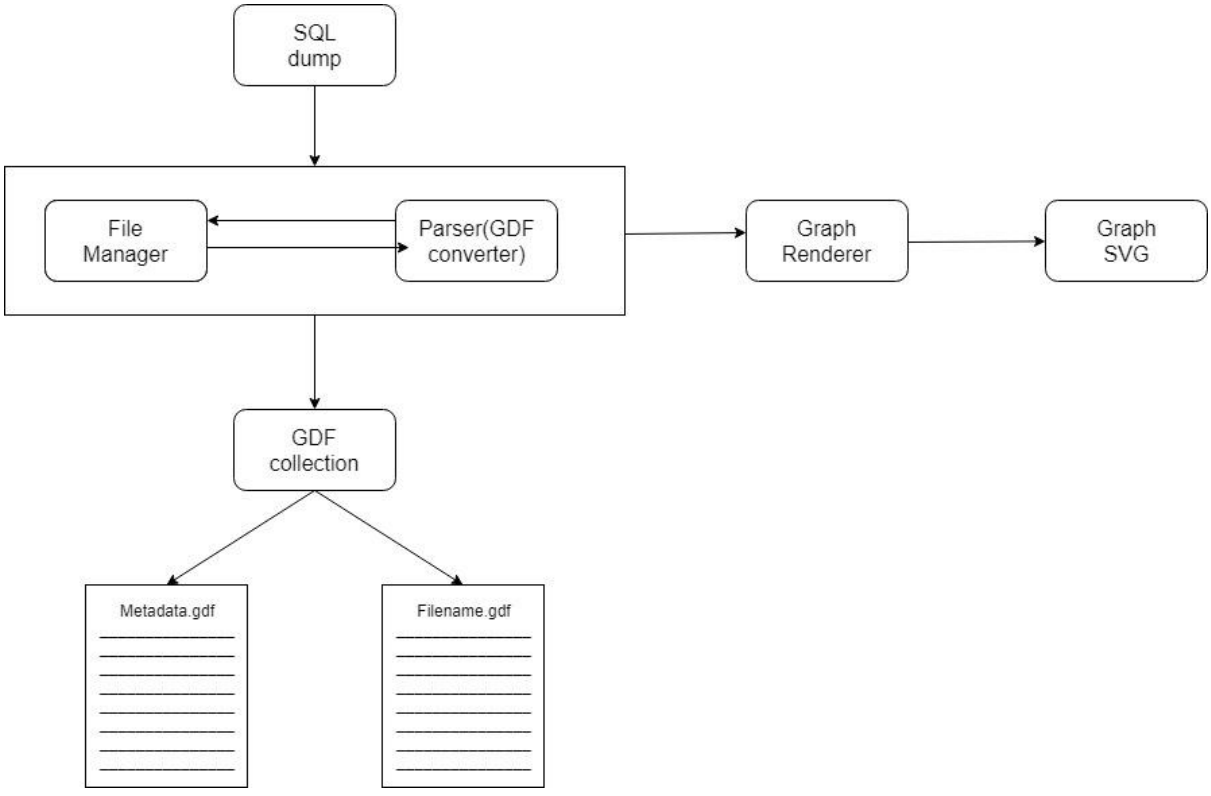

Fig. 3: Processing of input file to get an interactive Graph SVG

The sequence of steps has been explained below:

a) We will receive input from the user in the form of a table (called SQL Dump in the above chart). Mostly, user will also provide the corresponding meta-data file as another input file.
b) The parser will parse the input file and generate the GDF file along with the meta-data.gdf file. The availability of meta-data.gdf rests upon the availability of meta-data file as the input from user.
c) We need to convert the GDF file into the JSON objects and then render these JSON objects to Graph SVG. This will be done by the Graph Renderer. It will read our GDF file and generate the corresponding graph database.

5. OUR PROGRESS

Till now, we have completed the following:

a) Creation of GDF Convertor to convert a text file into 7-column format
b) Creation of meta-data.gdf file from meta-data.txt file
c) Reading SPARQL

We will also start working on the querying engine within this week.

Everything that we have done has been committed to the GIT remote repository (https://github.com/Sreyas-108/GDF). Also, the code and readMe files have been attached in the Appendix section.

6. FUTURE PROSPECTS

We plan to complete the following objectives:

a) Conversion of our GDF file into JSON objects
b) Understand the working of D3.js renderer and use it to generate graph SVGs from the JSON objects generated from (a)
c) Complete the building of querying engine (based on SPARQL)

7. SKILLS ACQUIRED

We have learned the following:

a) Bash and Shell Scripting
b) RDF and SPARQL
c) GitHub
d) JSON

CONCLUSION

To sum up, our project is to convert any file into GDF format and to render the file using D3.js renderer. We are using 7-column format for making GDF file. The same format is also followed to produce meta-data.gdf file. Later, we will also write code to get JSON objects corresponding to tuples in the GDF file. We will use D3.js to get graph SVGs from JSON objects.

APPENDIX

NOTE: The code attached here is already present on GitHub (https://github.com/Sreyas-108/GDF). Also, for sample input data and corresponding GDF file, please refer to the GitHub link.

Python code to generate sample input data:

```
#import numpy as np
import random
students=['Yashdeep','Vipin','Suyash','Sreyas','Abhinava','Ayush','Sid','An
irudh','Arvind','Ravi','Kavi','Isha','Ritu','Jaya','Kapil','Divya','Pankaj'
,'Sashan','Sushakt','Pratik','Saksham','Rachit','Shivam','Shalvi','Akriti',
'Bhoomi','Kavya','Sahil','Sargun','Swadesh','Kshitij','Anshuman','Samir']
teachers=['Ashish','Shan','Geeta','Ashu','Nago','Pogo','Sanjay','Sunita','K
alpana','GN','Mukesh','Suresh','Rohil','Ramesh','Manoj','Raman','Kannan','S
undar','Vishal','Amit','Kamlesh','Jenny']
fooditems=['Dosa','Pavbhaji','VadaPav','AlooParatha','BreadButter','BreadJa
m','BhelPuri','PaneerTikka','ShahiPaneer','Kadhi','Biryani','FriedRice','Pi
zza','Burger','Sandwich']
people=students+teachers
courses=['DSA','DBS','MuP','DD','POE','POM','DISCO','OOP','PAVA','CP','ES',
'TRW','MeOW','Thermo','M1','M2','M3','PnS','Bio','PPL','CompArch','DAA']


def createCrossData(list1,list2):
        cross=[]
        for l1 in list1:
                for l2 in list2:
                        if random.randint(0,1)==1:
                                cross.append((l1,l2))
        return cross

def createTripletStrings(crossTable,predName):
        tripletList=[]
        for l1,l2 in crossTable:
                tripletList.append(l1+'|'+predName+'|'+l2)
        return tripletList

def printStringListToFile(stringList,filename):
        file = open(filename,'w')
        for string in stringList:
                file.write(string)
                file.write('\n')


def isDataThere(text):
    fhandle=open('metadata.txt','r')

    lines=fhandle.readlines()

    for line in lines:
        line=line.rstrip()
        items=line.split('|')

        compare=items[0]
        if compare==text:
            fhandle.close()
            return True
    fhandle.close()
```

```
    return False


def printStudentMetadata(crossTable1,crossTable2,crossTable3):
    f=open('metadata.txt','a+')
    string='|memberOf|student'

    for t,s in crossTable1:
        if f.closed==False:
            f.close()
        if isDataThere(s)==False:
            f=open('metadata.txt','a+')
            f.write(s+string+'\n')

    for s,fo in crossTable2:
        if f.closed==False:
            f.close()
        if students.count(s)!=0 and isDataThere(s)==False:
            f=open('metadata.txt','a+')
            f.write(s+string+'\n')

    for s,co in crossTable3:
        if f.closed==False:
            f.close()
        if isDataThere(s)==False:
            f=open('metadata.txt','a+')
            f.write(s+string+'\n')

def printTeacherMetadata(crossTable1,crossTable2):
    f=open('metadata.txt','a+')
    string='|memberOf|teacher'

    for t,s in crossTable1:
        if f.closed==False:
            f.close()
        if isDataThere(t)==False:
            f=open('metadata.txt','a+')
            f.write(t+string+'\n')

    for s,fo in crossTable2:
        if f.closed==False:
            f.close()
        if teachers.count(s)!=0 and isDataThere(s)==False:
            f=open('metadata.txt','a+')
            f.write(s+string+'\n')

def printFoodMetadata(crossTable1):
    f=open('metadata.txt','a+')
    string='|memberOf|foodItem'

    for s,fo in crossTable1:
        if f.closed==False:
            f.close()
        if isDataThere(fo)==False:
            f=open('metadata.txt','a+')
            f.write(fo+string+'\n')

def printCourseMetadata(crossTable1):
    f=open('metadata.txt','a+')
    string='|memberOf|course'
```

```
    for s,co in crossTable1:
        if f.closed==False:
            f.close()
        if isDataThere(co)==False:
            f=open('metadata.txt','a+')
            f.write(co+string+'\n')

teaches=createCrossData(teachers,students)
eats=createCrossData(people,fooditems)
registeredIn=createCrossData(students,courses)

tripTeaches=createTripletStrings(teaches,'teaches')
tripEats=createTripletStrings(eats,'likesToEat')
tripReg=createTripletStrings(registeredIn,'registeredIn')

tripAll=tripTeaches+tripReg+tripEats
printStringListToFile(tripAll,filename='sampleData.txt')

printStudentMetadata(teaches,eats,registeredIn)
printTeacherMetadata(teaches,eats)
printFoodMetadata(eats)
printCourseMetadata(registeredIn)
```

Code (Shell script) to convert text file to GDF file

```
#!/bin/bash

function xtoGDF {
    filename=$1

    outname=''

    udi=''

    #code to extract the primary name of the file
    IFS='.'
    fileArr=()
    read -ra fileArr <<< "$filename"
    outname="${fileArr[0]}.gdf"

    IFS='|'

    count=0
    while IFS= read -r line
    do
        array=()
        uidArr=()

        ############ Code to create 7 column format ##############
        read -ra array <<< "$line"

        cum="${array[0]} ${array[1]} ${array[2]}"
        uid=$(echo "$cum" | md5sum)

        IFS=' '
        read -ra uidArr <<< "$uid"
        IFS='|'

        uid=${uidArr[0]}
```

```
        ############# Code to append data to GDF file #############

toWrite="$uid|${array[0]}|${array[3]}|${array[1]}|${array[4]}|${array[2]}|$
{array[5]}"
        if [[ $count -eq 0 ]]
        then
            echo "$toWrite" > "$outname"
        else
            echo "$toWrite" >> "$outname"

        fi

        let count=$count+1
    done < "$1"

    IFS='|'
    let count=0
    while IFS= read -r line
    do
        read -ra array <<< "$line"

        cum="${array[0]} ${array[1]} ${array[2]}"
        uid=$(echo "$cum" | md5sum)
        id=$(echo "${array[0]}" | md5sum)

        IFS=' '
        read -ra uidArr <<< "$uid"
        read -ra idArr <<< "$id"
        IFS='|'

        uid=${uidArr[0]}
        id=${idArr[0]}

        toWrite="$uid|${array[0]}|$id|${array[1]}||${array[2]}"

        if [[ $count -eq 0 ]]
        then
            echo "$toWrite" > "metadata.gdf"
        else
            echo "$toWrite" >> "metadata.gdf"
        fi
        let count=$count+1
    done < "$2"

    #set the IFS back to default
    IFS=' '
}

xtoGDF $@
```

ReadMe file for the above code

```
SUBJECT: Text to GDF Convertor
--------------------------------------------------------------------------
RELATED FUNCTIONS:
                    xtoGDF()    :  Requires filename of the text file as
the only command line argument
--------------------------------------------------------------------------
```

```
OUTPUT:
                    The command is a silent command. A new file with the
same name as the input file (but with .gdf extension) will be generated
containing the GDF format
--------------------------------------------------------------------------
SYNOPSIS:

Section 1: TEXT FILE FORMAT

    The text file must be formtted by the following rules:

    a) Each line must have only one entry
    b) Multiple enteries (i.e. lines) must be seperated by a newline (\n)
character
    c) Each line should have the following format:


Subject_Name|Predicate_Name|Object_Name|Subject_Qualifier|Predicate_Qualifi
er|Object_Qualifier

        NOTE:
        c.1) All or any enteries can be omitted

        c.2) If an entry from the middle of the format is omitted, even
then the delimiters (i.e. |) must be kept
             Eg: Yash|likes|IceCream|Intelligent||Chocolate

            But if the entry omitted is not in the middle of the format,
then successive enteries can be omitted too
            Eg: Yash|likes|IceCream

        c.3) If a line doesn't contain any entry, GDF format will be :
UID||||||
--------------------------------------------------------------------------
Section 2: THE GDF FORMAT

    The output file has the following format:

    a) Corresponding to each entry(i.e. line) of the text file, an entry is
written in .gdf file
    b) Multiple enteries are separated by newline character
    c) Each line has the following format:


UID|Subject_Name|Subject_Qualifier|Predicate_Name|Predicate_Qualifier|Objec
t_Name|Object_Qualifier

        NOTE:
        c.1) UID is generated as the md5sum hash of (SubjectName'
'Predicate_Name' 'ObjectName). Here ' ' indicates a space

        c.2) Based on the input, each entry can have any or all enteries
omitted. But delimiter '|' would still be present
--------------------------------------------------------------------------
```

About( ) : Function to create a file that contains the names of all the unique subjects

```
#!/bin/bash

function about {
    filename=$1
    IFS='.'

    fileArr=()
    read -ra fileArr <<< "$filename"

    tmpname1=${fileArr[0]}_tmp1.gdf

    count=0

    while IFS= read -r line
    do
        if [[ $count -eq 0 ]]
        then
            echo "$line" > "$tmpname1"
        else
            echo "$line" >> "$tmpname1"
        fi
        let count=$count+1
    done < "$1"


    tmpname2=${fileArr[0]}_tmp2.gdf
    sort -k 2,2 --field-separator='|' "$tmpname1" > "$tmpname2"

    IFS='|'

    tmpname3=${fileArr[0]}_tmp3.gdf

    count=0
    while IFS= read -r line
    do
        IFS='|'
        segment=()
        read -ra segment <<< "$line"

        if [[ $count -eq 0 ]]
        then
            echo "${segment[1]}" > $tmpname3
        else
            echo "${segment[1]}" >> $tmpname3
        fi
        let count=$count+1
    done < "$tmpname2"

    aboutFile=${fileArr[0]}_about.gdf
    uniq "$tmpname3" > "$aboutFile"
    rm $tmpname1 $tmpname2 $tmpname3
    IFS=' '
}

about $@
```

separateBySub( ): Function to create a separate file for each subject having all information of that subject

```
#!/bin/bash
function separateBySub
{
        filename=$1

        outname=''
        rel=(NaN NaN NaN NaN NaN NaN NaN)
        IFS=$'\n'
        while read line
                        #Loop to go through each relation in a gdf file
        do
                IFS='|' read -r -a rel<<<"$line"
        #Reading the line into array 'rel' separated by '|'
                local outfile=${rel[1]}.gdf
                flag=$(ls | grep $outfile)
                #Flag to check for existence of the filename
                if [[ -z $flag ]]
                then
                        touch $outfile
                #Creating the file from subject's name if it doesn't exist
                fi
                echo $line >> $outfile
                #Appending the relation
                IFS=$'\n'
        done < $filename
}

separateBySub $@               #calling the function
```

REFERENCES

1. https://github.com/Sreyas-108/GDF : Link to the GIT repository having all our work

2. https://www.w3.org/TR/rdf-concepts/ : Official documentation for RDF

3. https://www.w3.org/TR/rdf-sparql-query/ : Official documentation for SPARQL (Query language for RDF)

4. https://d3js.org/ : Official website of D3.js, the renderer we are using

5. https://wiki.dbpedia.org/ : Link to DBPedia – the site from where we will collect 3-column formatted data

GLOSSARY

RDF: Stands for Resource Description Framework. It is a graph-based database system (NoSQL)

SPARQL: The query language for RDF

Meta-Data: The data about data

JSON: Stands for Java Script Object Notation. It is a lightweight format for storing and transporting data.

Renderer: It is a software to read a data file. In context of our project, renderer will read the JSON objects generated from the GDF file and generate Graph SVG.

SVG: Stands for Scalable Vector Graphics. It is an XML based vector image format that supports interactivity and animation.

D3.js: JavaScript library for manipulating documents based on data. This is the renderer we are using in our project.