Data Integration in the Life Science

Report - Integration of Big Biological Data

1 February 2017

Shu SHANG
Zhao ZHANG
Zhe LV

Data&Knowledge
Paris-Saclay University

Introduction

This report serves as a presentation for our project of Data Integration course. The idea is to concretely integrate several biological datasets while following a process enhancing the reproducibility using the general methods like scripts and data set setting.

Data integration involves combining data residing in different sources and providing users with a unified view of these data. This process becomes significant in a variety of situations, which include both commercial and scientific domains especially in bioinformatics.

Bioinformatics uses the methods of mathematics, informatics, statistics, and computer science to study biology. Now, the main research directions of bioinformatics are: sequence alignment, gene recognition, gene recombination, protein structure prediction, gene expression, protein response prediction, and the establishment of evolutionary models.

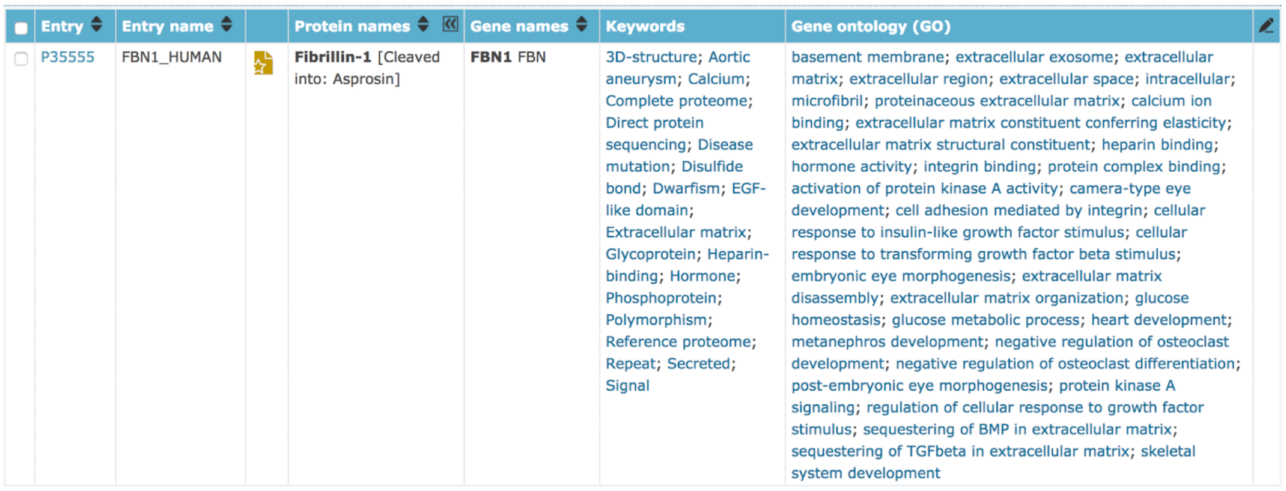
This project follows following steps: Firstly, we download the datasets from UniProtKB and NCBI. The dataset of UniProtKB can be downloaded with different formats like plain text, XML and RDF. We used XML format for this project. The datasets downloaded from NCBI are already well annotated with column names. Then, we should design a database model for store the data. For this project, we used the relational database management system MySQL. The step followed is to populate the data into the model. The database model we designed will decide how we populate the data. Finally, we will do some experiments and analyses based on the data we populated, especially relationship of the data from two different sources (UniPort and NCBI).

For the following part, we will first get a glimpse of the data, especially the data from UniProtKB which is more difficult to handle. Then, we will make a detailed description on the work we did. And in the last part, we will make a conclusion and talk about the difficulties we encountered during this project.

Datasets

UniProtKB

In this project, we used the congenital disease dataset in the form of XML downloaded from UniProtKB. To get this dataset, we first go to http://www.uniprot.org, then search *congenital disease AND reviewed:yes*, we will get the query result like this:



Entry	Entry name	Protein names	Gene names	Keywords	Gene ontology (GO)
P35555	FBN1_HUMAN	**Fibrillin-1** [Cleaved into: Asprosin]	**FBN1** FBN	3D-structure; Aortic aneurysm; Calcium; Complete proteome; Direct protein sequencing; Disease mutation; Disulfide bond; Dwarfism; EGF-like domain; Extracellular matrix; Glycoprotein; Heparin-binding; Hormone; Phosphoprotein; Polymorphism; Reference proteome; Repeat; Secreted; Signal	basement membrane; extracellular exosome; extracellular matrix; extracellular region; extracellular space; intracellular; microfibril; proteinaceous extracellular matrix; calcium ion binding; extracellular matrix constituent conferring elasticity; extracellular matrix structural constituent; heparin binding; hormone activity; integrin binding; protein complex binding; activation of protein kinase A activity; camera-type eye development; cell adhesion mediated by integrin; cellular response to insulin-like growth factor stimulus; cellular response to transforming growth factor beta stimulus; embryonic eye morphogenesis; extracellular matrix disassembly; extracellular matrix organization; glucose homeostasis; glucose metabolic process; heart development; metanephros development; negative regulation of osteoclast development; negative regulation of osteoclast differentiation; post-embryonic eye morphogenesis; protein kinase A signaling; regulation of cellular response to growth factor stimulus; sequestering of BMP in extracellular matrix; sequestering of TGFbeta in extracellular matrix; skeletal system development

Figure 1 - Query result of e congenital disease

As it is said in the subject, for this project, we are only interested in these fields:

- ID, AC, DE, GN, KW
- DR but only the lines starting with GO (to get the gene ontology annotations)

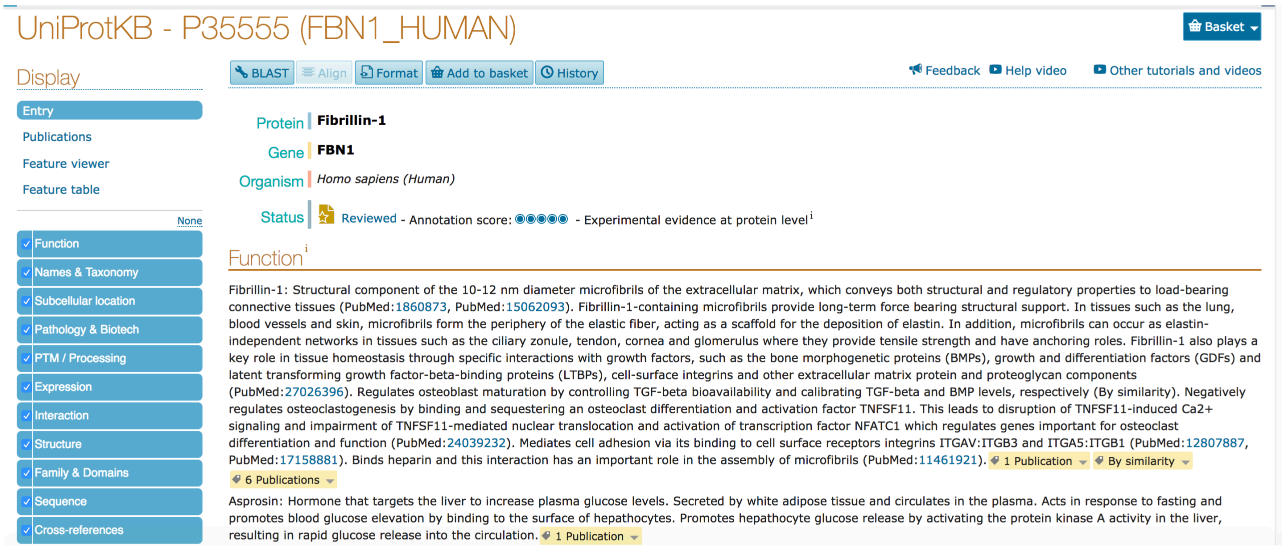
In the above of the table, we can modify the column configuration to get only the columns that we are interested. **For instance, we selected to show the columns: Entry, Entry name, Protein names, Gene names, Keywords and Gene ontology (GO).**

In fact, these are columns are exactly correspond to the fields we mentioned above. The mapping is:

ID	Entry name
AC	Entry
DE	Protein names
GN	Gene names
KW	Keywords

It should be noted that, in the figure 1, the column only shows the gene ontology term without the GO id like GO:0005604.

We can more details on every entry and the characteristics of the columns by clicking on the Entry.



Protein Fibrillin-1
Gene FBN1
Organism *Homo sapiens (Human)*
Status Reviewed - Annotation score: ●●●●● - Experimental evidence at protein level

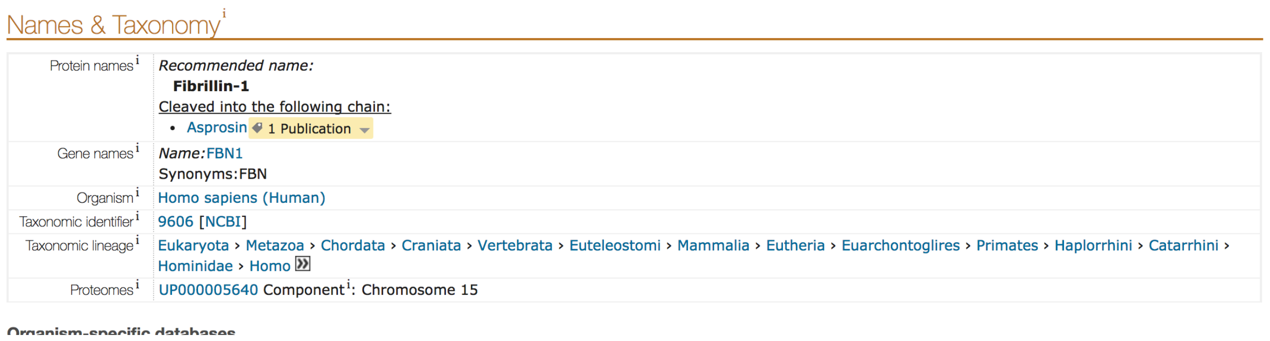
Function

Fibrillin-1: Structural component of the 10-12 nm diameter microfibrils of the extracellular matrix, which conveys both structural and regulatory properties to load-bearing connective tissues (PubMed:1860873, PubMed:15062093). Fibrillin-1-containing microfibrils provide long-term force bearing structural support. In tissues such as the lung, blood vessels and skin, microfibrils form the periphery of the elastic fiber, acting as a scaffold for the deposition of elastin. In addition, microfibrils can occur as elastin-independent networks in tissues such as the ciliary zonule, tendon, cornea and glomerulus where they provide tensile strength and have anchoring roles. Fibrillin-1 also plays a key role in tissue homeostasis through specific interactions with growth factors, such as the bone morphogenetic proteins (BMPs), growth and differentiation factors (GDFs) and latent transforming growth factor-beta-binding proteins (LTBPs), cell-surface integrins and other extracellular matrix protein and proteoglycan components (PubMed:27026396). Regulates osteoblast maturation by controlling TGF-beta bioavailability and calibrating TGF-beta and BMP levels, respectively (By similarity). Negatively regulates osteoclastogenesis by binding and sequestering an osteoclast differentiation and activation factor TNFSF11. This leads to disruption of TNFSF11-induced Ca2+ signaling and impairment of TNFSF11-mediated nuclear translocation and activation of transcription factor NFATC1 which regulates genes important for osteoclast differentiation and function (PubMed:24039232). Mediates cell adhesion via its binding to cell surface receptors integrins ITGAV:ITGB3 and ITGA5:ITGB1 (PubMed:12807887, PubMed:17158881). Binds heparin and this interaction has an important role in the assembly of microfibrils (PubMed:11461921). 1 Publication By similarity 6 Publications

Asprosin: Hormone that targets the liver to increase plasma glucose levels. Secreted by white adipose tissue and circulates in the plasma. Acts in response to fasting and promotes blood glucose elevation by binding to the surface of hepatocytes. Promotes hepatocyte glucose release by activating the protein kinase A activity in the liver, resulting in rapid glucose release into the circulation. 1 Publication

Figure 2 - Entry details

The figure 2 above shows the details of an entry. For instance, we can see the protein name, the gene name and many other details.



Names & Taxonomy

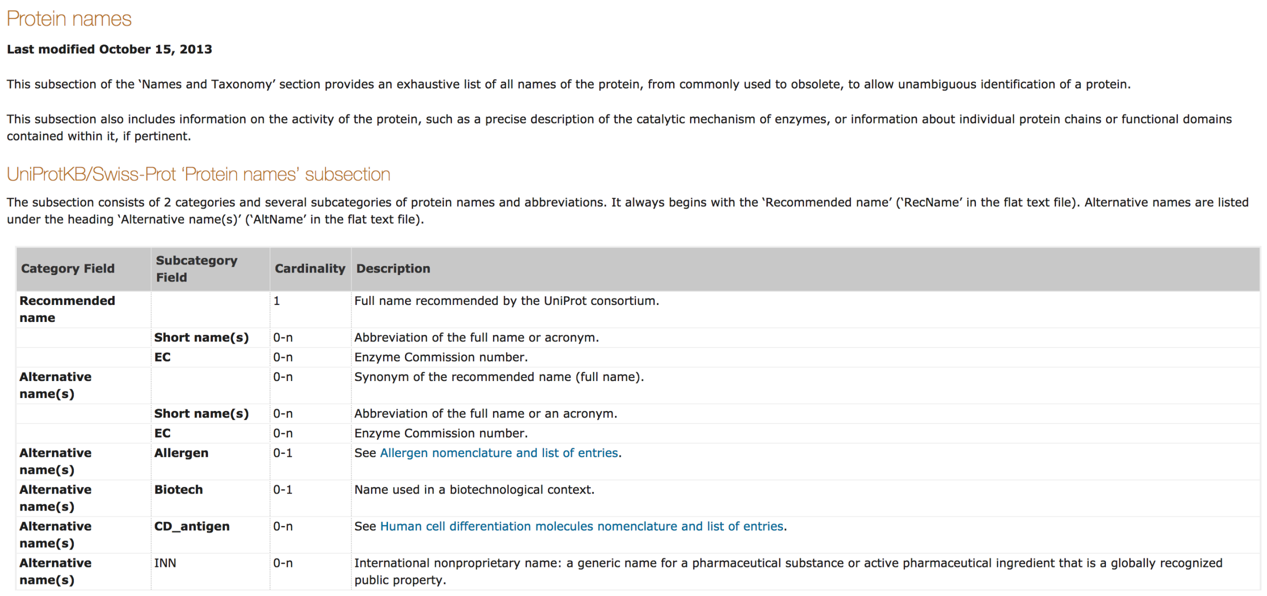
Protein names	*Recommended name:* **Fibrillin-1** Cleaved into the following chain: • Asprosin 1 Publication
Gene names	*Name:* FBN1 Synonyms:FBN
Organism	Homo sapiens (Human)
Taxonomic identifier	9606 [NCBI]
Taxonomic lineage	Eukaryota › Metazoa › Chordata › Craniata › Vertebrata › Euteleostomi › Mammalia › Eutheria › Euarchontoglires › Primates › Haplorrhini › Catarrhini › Hominidae › Homo
Proteomes	UP000005640 Component: Chromosome 15

Organism-specific databases

Figure 3 - Protein details

Besides, we can find the protein name details here. The figure 3 shows the protein details of the entry **P35555** we illustrated above. From this table information, we know that **Fibrillin-1** is the recommended name of the protein which linked to the disease entry.

Moreover, we can click on the Protein names (with a letter ‘s’ as the superscript) to find more details about this column.



Protein names

Last modified October 15, 2013

This subsection of the 'Names and Taxonomy' section provides an exhaustive list of all names of the protein, from commonly used to obsolete, to allow unambiguous identification of a protein.

This subsection also includes information on the activity of the protein, such as a precise description of the catalytic mechanism of enzymes, or information about individual protein chains or functional domains contained within it, if pertinent.

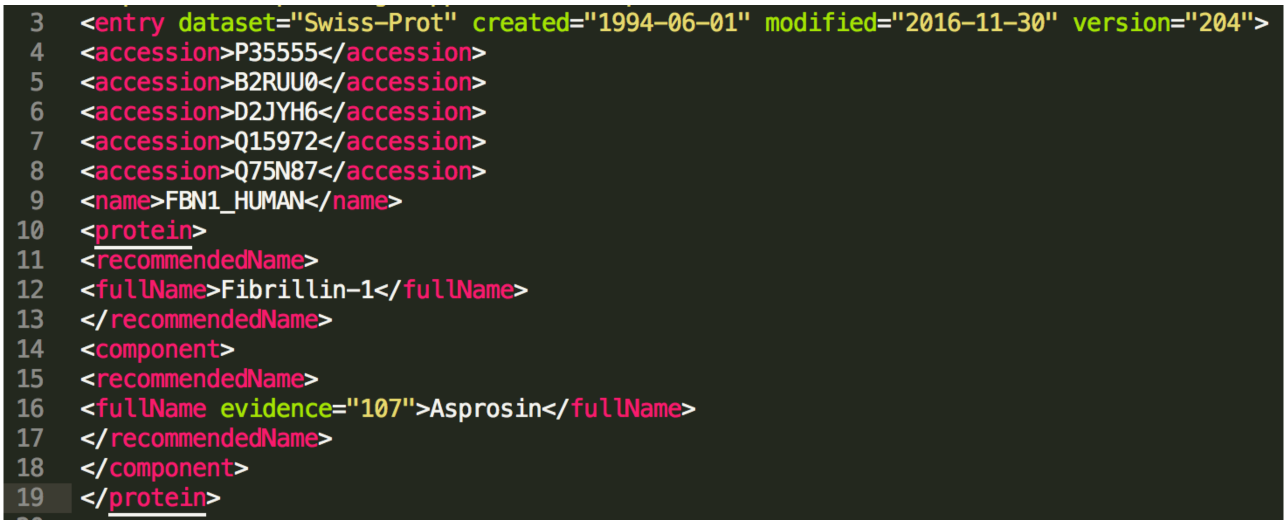
UniProtKB/Swiss-Prot 'Protein names' subsection

The subsection consists of 2 categories and several subcategories of protein names and abbreviations. It always begins with the 'Recommended name' ('RecName' in the flat text file). Alternative names are listed under the heading 'Alternative name(s)' ('AltName' in the flat text file).

Category Field	Subcategory Field	Cardinality	Description
Recommended name		1	Full name recommended by the UniProt consortium.
	Short name(s)	0-n	Abbreviation of the full name or acronym.
	EC	0-n	Enzyme Commission number.
Alternative name(s)		0-n	Synonym of the recommended name (full name).
	Short name(s)	0-n	Abbreviation of the full name or an acronym.
	EC	0-n	Enzyme Commission number.
Alternative name(s)	**Allergen**	0-1	See Allergen nomenclature and list of entries.
Alternative name(s)	**Biotech**	0-1	Name used in a biotechnological context.
Alternative name(s)	**CD_antigen**	0-n	See Human cell differentiation molecules nomenclature and list of entries.
Alternative name(s)	INN	0-n	International nonproprietary name: a generic name for a pharmaceutical substance or active pharmaceutical ingredient that is a globally recognized public property.

Figure 4 - Protein name details

The figure above shows the details of the column Protein names we talked about in the Figure 1. We can see the cardinality of every category field. For instance, the cardinality of the field Recommended name is 1. This means that in the Protein names section, it can only have one recommended name. This can be illustrated more concretely in the XML file as below.



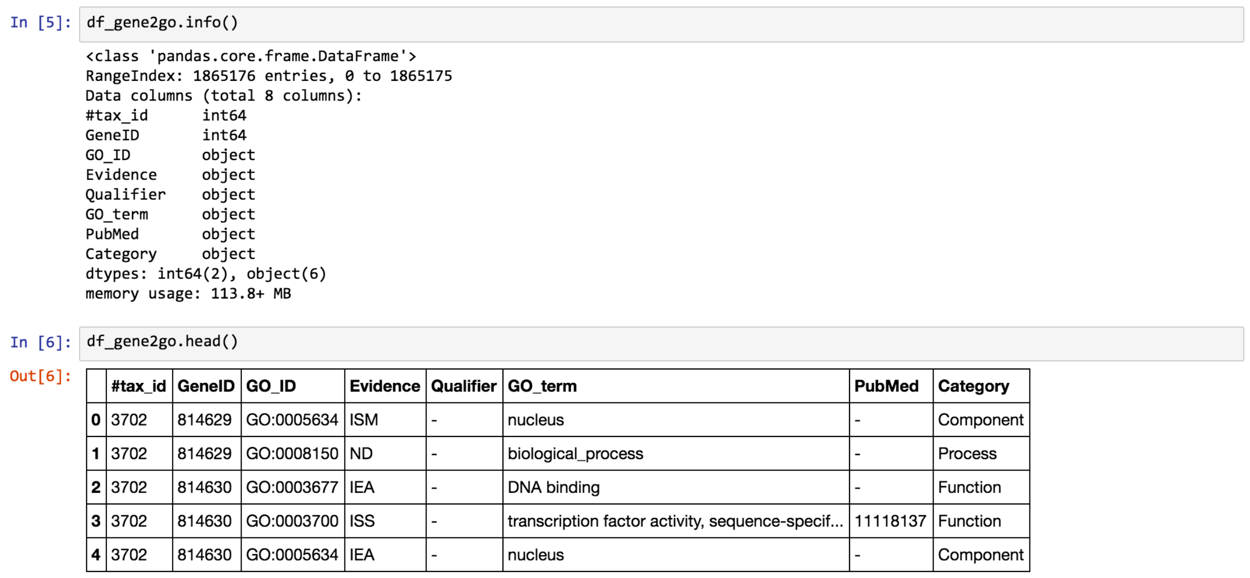
```xml
<entry dataset="Swiss-Prot" created="1994-06-01" modified="2016-11-30" version="204">
<accession>P35555</accession>
<accession>B2RUU0</accession>
<accession>D2JYH6</accession>
<accession>Q15972</accession>
<accession>Q75N87</accession>
<name>FBN1_HUMAN</name>
<protein>
<recommendedName>
<fullName>Fibrillin-1</fullName>
</recommendedName>
<component>
<recommendedName>
<fullName evidence="107">Asprosin</fullName>
</recommendedName>
</component>
</protein>
```

Figure 5 - The protein element of XML data

Here, it’s also the first entry P35555, we can see that inside the protein element, there is only one recommended name element named recommendedName. It should be noted that the second recommendedName inside the protein element clause belongs to component element. This correspond to the component of the protein, which we don't concern about here.

UCBI

As we said before, the two dataset, namely *Homo_sapiens.gene_info* and *gene2GO* are well annotated. Here is a simple exploration using Python script and Pandas library.



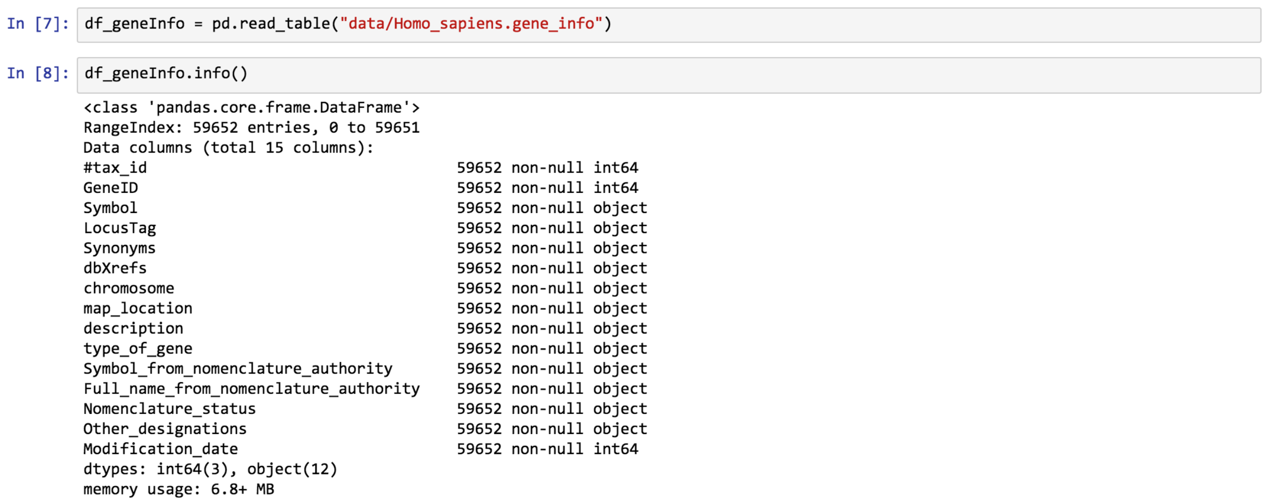
```
In [5]: df_gene2go.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1865176 entries, 0 to 1865175
Data columns (total 8 columns):
#tax_id      int64
GeneID       int64
GO_ID        object
Evidence     object
Qualifier    object
GO_term      object
PubMed       object
Category     object
dtypes: int64(2), object(6)
memory usage: 113.8+ MB
```

```
In [6]: df_gene2go.head()
Out[6]:
```

	#tax_id	GeneID	GO_ID	Evidence	Qualifier	GO_term	PubMed	Category
0	3702	814629	GO:0005634	ISM	-	nucleus	-	Component
1	3702	814629	GO:0008150	ND	-	biological_process	-	Process
2	3702	814630	GO:0003677	IEA	-	DNA binding	-	Function
3	3702	814630	GO:0003700	ISS	-	transcription factor activity, sequence-specif...	11118137	Function
4	3702	814630	GO:0005634	IEA	-	nucleus	-	Component

Figure 6 - gene2go dataset



```
In [7]: df_geneInfo = pd.read_table("data/Homo_sapiens.gene_info")

In [8]: df_geneInfo.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59652 entries, 0 to 59651
Data columns (total 15 columns):
#tax_id                                 59652 non-null int64
GeneID                                  59652 non-null int64
Symbol                                  59652 non-null object
LocusTag                                59652 non-null object
Synonyms                                59652 non-null object
dbXrefs                                 59652 non-null object
chromosome                              59652 non-null object
map_location                            59652 non-null object
description                             59652 non-null object
type_of_gene                            59652 non-null object
Symbol_from_nomenclature_authority      59652 non-null object
Full_name_from_nomenclature_authority   59652 non-null object
Nomenclature_status                     59652 non-null object
Other_designations                      59652 non-null object
Modification_date                       59652 non-null int64
dtypes: int64(3), object(12)
memory usage: 6.8+ MB
```

Figure 7 - Home_sapiens_gene_info dataset (Jupyter notebook)

As we can see, the ***gene2go*** dataset downloaded from UCBI has eight columns and the ***Home_sapiens_gene_info*** dataset has total 15 columns. We can easily import these data directly into a relational database using the embedded data structure of Pandas like dataFrame.

Description of Work

Database Model

To populate the data, we need firstly design the database model. The figure below illustrates the UML model of the database.

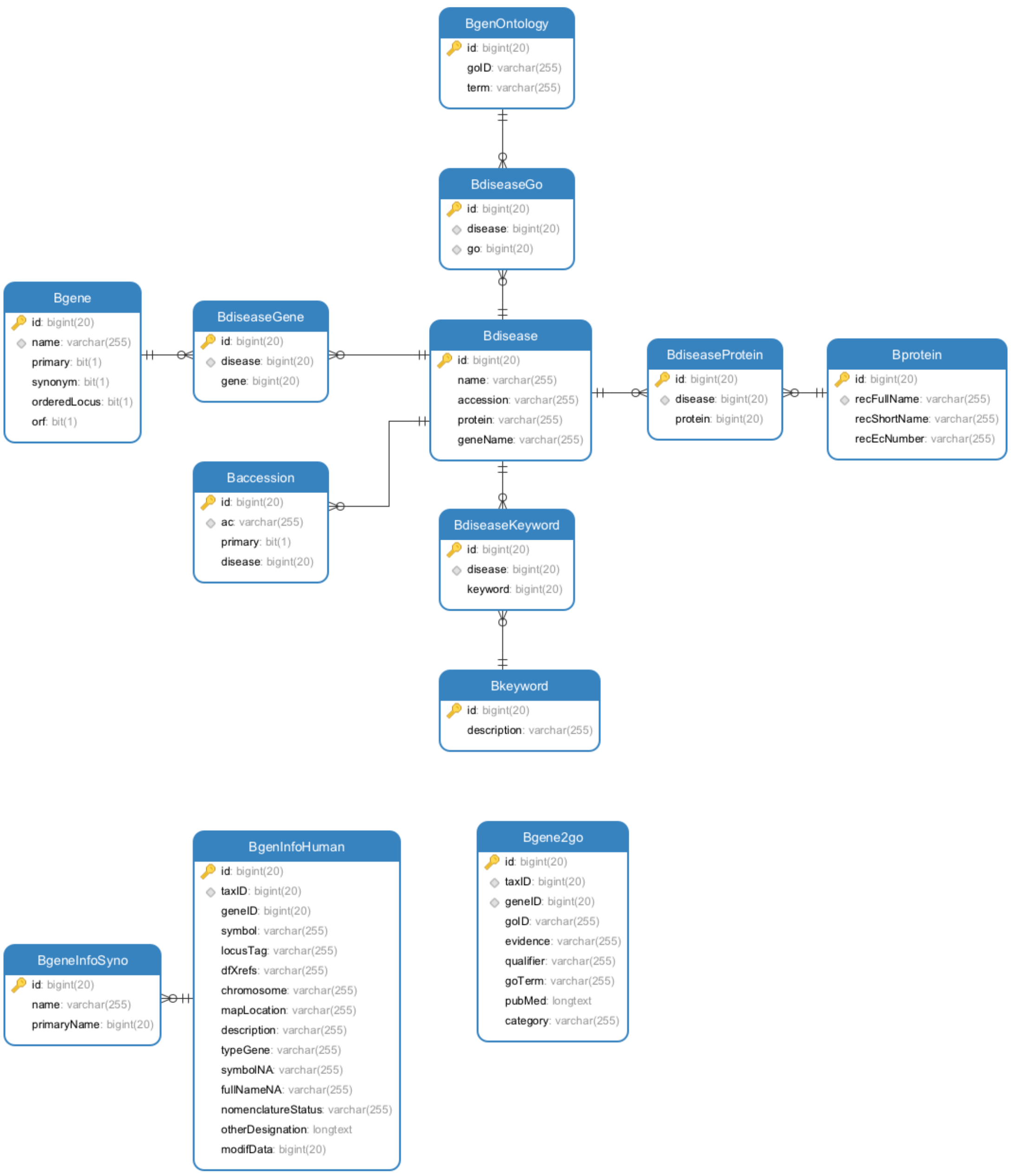


Figure 8 - Database model

The name of every table in the database starts with a capital B. Here, for instance, ***Bdisease*** represents one entry (one row) in the Figure 1. It has 5 fields: id, name, accession, protein and geneName. The filed ***id*** is the row number, the field ***name*** is the Entry name illustrated in the Figure 1, the field ***accession*** is the recommended (citable) ac number of this entry. The field ***protein*** is the recommended name of the protein and the filed ***geneName*** is the primary gene name (corresponding to the gene/name element with a primary attribut in the XML data).

As we know, every disease (i.e. every entry) has more than one accession numbers, has more than one protein names, gene names, keywords and gene ontology, vice versa. Consequently, we have to put an intermediate between the table ***Bdisease*** and each of the other four. This is the reason why there exist tables like BdiseaseProtein which contains mapping between the two tables to simplify the N-N relationship.

For the other two datasets, we integrate them directly keeping the columns structures. Here, it should be noted that the synonyms of protein in the dataset UCBI are not well formed. Every gene has several synonyms, separating by a vertical line. So we separate them using scripts and store them in the table ***BgeneInfoSyno***.

Populate the data

To populate the three datasets, we use Python Script. Firstly, to make sure the reproducibility. The version of techniques and dependencies that we used in this project are listed here:

Python	**2.7.11**
Pandas	**0.19**
mysql-python	**1.2.5**
MySQL	**5.7.12**
Jupyter notebook	**4.2.3**

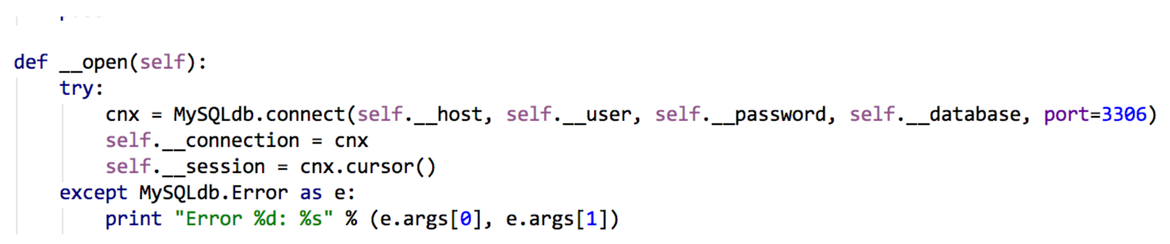
We used the package manage software Anaconda to manage these libraries and software. The Anaconda is running on a Macbook computer and is the latest version of 64bits.

Firstly, we should populate the XML data. Here we have used a very import method : **XPath**. It is an XML-based tree structure which provides the ability to find nodes in a data structure tree. We can easily find the interesting element we want using this tool.

Python already has XML package installed. To use the XPath, we should import ElementTree module. We use the parse() method to parse the XML file and then get the root element. Then, we can use the findall() method to find the interested element in the XML tree structure.

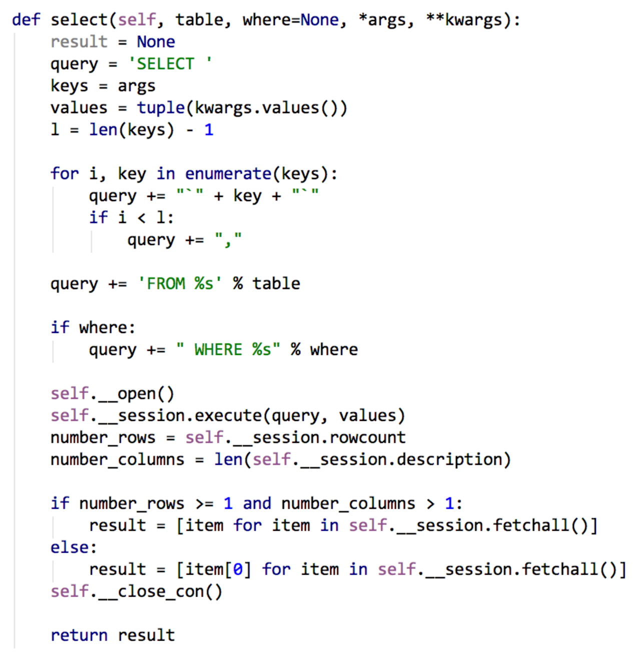
After finding the data we want, we should then populate it into the relational database. To do this, we use the mysql-python package listed before. It's a popular package in the Python Eco-system to handle the MySQL connection problem.

We have implemented a module named MySQLConnector, which is a Python class for connecting the MySQL server and accelerating the development of project. In this class, there are a set of general methods such as **select()**, **insert()**, **update()** and **delete()**. These methods take the name of table, the condition, the selected columns as parameters to form the SQL command and return a list of results. This module can be used to handle most of the iteration problems with MySQL databases.



```python
def __open(self):
    try:
        cnx = MySQLdb.connect(self.__host, self.__user, self.__password, self.__database, port=3306)
        self.__connection = cnx
        self.__session = cnx.cursor()
    except MySQLdb.Error as e:
        print "Error %d: %s" % (e.args[0], e.args[1])
```

Figure 9 - General method **open()**



```python
def select(self, table, where=None, *args, **kwargs):
    result = None
    query = 'SELECT '
    keys = args
    values = tuple(kwargs.values())
    l = len(keys) - 1

    for i, key in enumerate(keys):
        query += "`" + key + "`"
        if i < l:
            query += ","

    query += 'FROM %s' % table

    if where:
        query += " WHERE %s" % where

    self.__open()
    self.__session.execute(query, values)
    number_rows = self.__session.rowcount
    number_columns = len(self.__session.description)

    if number_rows >= 1 and number_columns > 1:
        result = [item for item in self.__session.fetchall()]
    else:
        result = [item[0] for item in self.__session.fetchall()]
    self.__close_con()

    return result
```

Figure 10 - Method **select()**

The two figures above illustrated the ideas of this module. For instance, the select method takes a table name as the first parameter, a condition, a set of selected column named (*args) and a set of conditions (**kwargs).

The whole process is a streaming method, which means we iterate the XML file from the beginning to the end only one time and we should get all we want. Concretely, the entrance of the element we want is **entry** element. An entry corresponds to one row of Figure 1, but contains all the information about this entry. For instance, to get the accession number of an entry, we use the findall() method to looking for the direct descend child the entry element whose element name is <accession>. Then we get the text data and populate to the database.

The population of the other two datasets is more simple, we populate directly into two table, except for the synonyms of protein, which we use another table to store.

Analysis/Queries

After populating the data, we should write two queries/scripts to find the genes having the same official first name UCBI and UniProtKB but with different synonyms names. Also, we want to find the genes having the same first name but with different gene ontology terms.

For the first one, we just need to find out the genes with the same official first names in the two sources. Then, we compare their corresponding synonym name one by one to find check out if they have different synonym (secondary) names. The same for the comparison of gene ontology, but we should use the gene2go dataset to get the gene ontology information.

Difficulties

The parser program can correctly parse the dataset. But there is still a problem we haven't overcome. The MySQL server can be suddenly closed during the execution of a command. We have to wait until the connection is re-established. This problem may due to the frequent close actions during the whole parsing processus.

Conclusion

This project gives us a new vision of data integration. We have experienced each step of a data integration processus including downing the source, pre-precessing, populating the data and data analysis based on the data model.

The bioinformatics data is well formed and important. We can use these data to apply to many different interesting areas like medicine, health-care industry.