Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Building a Semantic Graph Database for Bioinformatics

A. Messina

Rapporto Tecnico N.:
RT-ICAR-PA-18-05 **Maggio 2018**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sede di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Building a Semantic Graph Database for Bioinformatics

A. Messina[1]

Rapporto Tecnico N.:
RT-ICAR-PA-18-05 **Maggio 2018**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Palermo, Via Ugo La Malfa n. 153, 90146 Palermo.

Index

1 Introduction

Nowadays, the amount of biological data available online has proliferated, but this has been accompanied by enormous challenges arising from the need to integrate and connect related information from different sources [1].

Common problems include locating resources, differing data formats, ambiguity and duplication, relationships between data and the sheer volume and granularity of the information. As yet, there is no standard memorization and query format for this kind of data, so each resource usually requires a different approach to be properly handled.

Several classes of bio-molecular data, such as transcriptional regulatory networks and protein-protein interaction networks, interact as complex networks. They can usually be modeled as graphs, where nodes (and their attributes) model biological entities and edges contain relationships between these entities. Since query languages play a key role in the success of databases, in order to allow for efficient queries, these graphs can be stored either in relational or graph databases [2], where the latter by their nature seem to be a *natural* choice.

In this work, we illustrate how to build *BioGrakn*, a semantic graph database for bioinformatics [3] based on GRAKN.AI [4], which is a deductive database in the form of a knowledge graph, allowing complex data modelling, verification, scaling, querying and analysis.

The database behind GRAKN.AI uses an ontology to facilitate the modelling of extremely complex datasets, functioning as a data schema constraint to guarantee information consistency. GRAKN.AI stores data in a way that allows machines to understand the meaning of information in the complete context of their relationships. Consequently, the semantic layer of Grakn allows computers to process complex information more intelligently, with less human intervention.

2 GRAKN.AI

2.1 Introduction

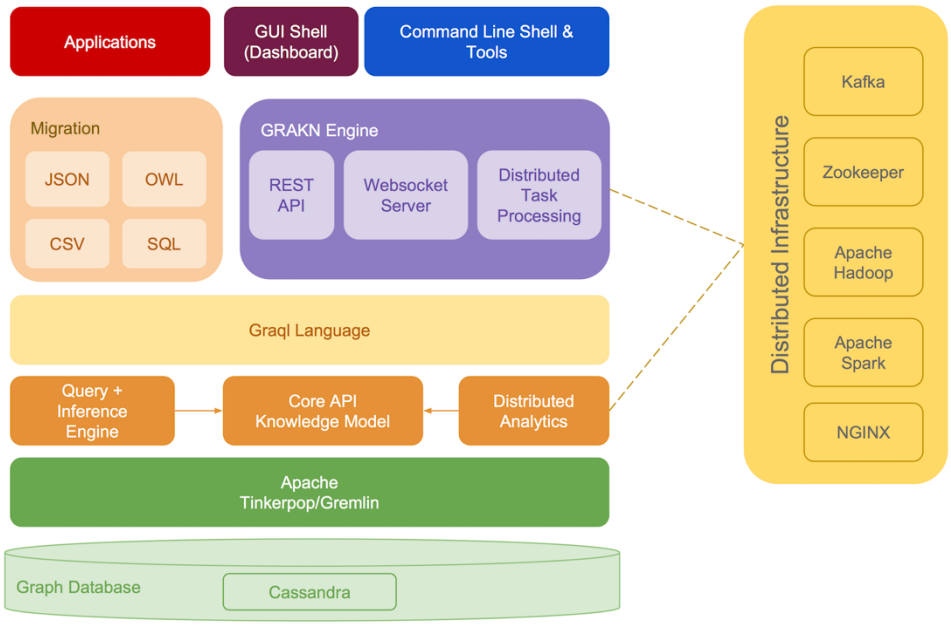
GRAKN.AI is a deductive database in the form of a knowledge graph, allowing complex data modelling, verification, scaling, querying and analysis.

The database behind GRAKN.AI uses an ontology to facilitate the modelling of extremely complex datasets, functioning as a data schema constraint to guarantee information consistency. GRAKN.AI stores data in a way that allows machines to understand the meaning of information in the complete context of their relationships. Consequently, the semantic layer of Grakn allows computers to process complex information more intelligently, with less human intervention.

GRAKN.AI is composed of two parts: Grakn (the storage), and Graql (a declarative query language).

2.2 Grakn

Grakn is built using several graph computing and distributed computing platforms, such as Apache TinkerPop and Apache Spark. Grakn is designed to be sharded and replicated over a network of distributed machines. The underlying data structure of Grakn is that of a labelled, directed hypergraph.



Grakn exposes a high-level knowledge model, allowing developers to represent their application domain as an ontology, specifying it in terms of entities, resources, relations, and roles.

Grakns ontology modelling constructs include, but are not limited to, data type hierarchy, relation type hierarchy, bi-directional relationships, multi-type relationships, N-ary relationships, relationships in relationships, and so on.

Therefore, Grakn can model the real world and all the hierarchies and hyper-relationships contained within it.

2.3 Graql

Graql is a declarative, knowledge-oriented graph query language that uses machine reasoning to retrieve explicitly stored and implicitly derived knowledge from Grakn.

When using legacy systems, database queries have to define explicitly the data patterns they are looking for. Graql, on the other hand, will translate a query pattern into all its logical equivalents and evaluate them against the database. This includes, but is not limited to, the inference of types, relation- ships, context, and pattern combination. This way, Graql can derive implicit information with concise and intuitive statements, reducing the complexity of expressing intelligent questions.

In Graql, there are different types of queries available: for matching patterns in the graph, inserting or deleting types and instances, and for computing useful information about the graph, such as statistics or shortest path between nodes.

Two inference mechanisms are supported: *type inference*, based on the semantics defined in the ontology, and *rule-based inference*, that involves rules defined by expressions of the form

when G1 then G2

where G1 and G2 are a pair of Graql patterns.

Whenever the left-hand-side (*when*) pattern G1 is found in the data, the right-hand-side (*then*) pattern G2 can be assumed to exist and optionally materialized (inserted).

3 Data Sources

3.1 Introduction

The data sources selected for database population allow us to build an integrated database containing resources related to genes, proteins, miRNAs, and metabolic pathways.

Getting into the details, we have considered the following:

- NCBI Entrez Gene [5]: provides a lot of genes data, such as interactions with other genes, genomic context, annotated pathways, and so on.
- Gene Ontology (GO) [6]: provides annotations for gene products in biological processes, cellular components and molecular functions.
- UniProt KnowledgeBase (UniprotKB) [7]: the largest public collection of annotated functional information on proteins.
- Reactome [8]: contains validated metabolic pathways, each annotated as a set of biological events, dealing with genes and proteins.
- miRBase [9]: provides all the known miRNAs sequences and annotations, associated with names, keywords, genomic locations, and references.
- mirCancer [10]: contains associations between miRNAs and human cancers.
- miRNASNP [11]: aims to provide a resource of the miRNA-related mutations (SNPs) for human and other species.
- mirTarBase [12]: list of experimentally validated miRNA-target interactions.
- HGNC [13]: the HUGO Gene Nomenclature Committee database contains, for each gene symbol, a list of synonyms and a list of corresponding entries in the most popular genes databases.

Many of the above are supplied in *tab-separated values* (TSV) format, a simple text format for storing data in a tabular structure where each record in the table is one line of the text file, and each field value of a record is separated from the next by a tab character. By contrast, miRBase, GO, and UniprotKB are distributed as EMBL text file format and XML format, respectively.

3.2 NCBI Entrez Gene

The NCBI Entrez Gene is a searchable database of genes, focusing on genomes that have been completely sequenced and that have an active research community to

contribute gene-specific data.

Information includes nomenclature, chromosomal localization, gene products and their attributes (e.g., protein interactions), associated markers, phenotypes, interactions, and links to citations, sequences, variation details, maps, expression reports, homologs, protein domain content, and external databases.

The database is available for download from the address ftp://ftp.ncbi.nih.gov/gene/ and it is splitted in several compressed files updated daily within the *DATA* directory. In this work, only the file *gene_info.gz* is considered.

3.3 Gene Ontology

The Gene Ontology (GO) is the most complete and daily updated public resource for genes and proteins annotation. It provides annotations for gene products in biological processes, cellular components and molecular functions.

The GO main features include ontologies, definitions and mappings to other databases, such as UniProtKB, and manual and automated generated annotations.

3.4 UniProt

The UniProt Knowledgebase (UniProtKB) is the largest public collection of annotated functional information on proteins and it is updated every four weeks.

It stores both computationally analyzed and manually annotated records, including classifications, cross-references and quality indications available to scientific researchers.

UniProtKB is actually composed of two sections: UniProtKB/Swiss-Prot and UniProtKB/TrEMBL.

UniProtKB/Swiss-Prot is the reviewed section of the UniProt Knowledgebase.

The TrEMBL section of UniProtKB was introduced in 1996 in response to the increased dataflow resulting from genome projects. It was already recognized at that time that the traditional time- and labour-intensive manual curation process which is the hallmark of Swiss-Prot could not be broadened to encompass all available protein sequences. UniProtKB/TrEMBL contains high quality computationally analyzed records that are enriched with automatic annotation and classification.

These UniProtKB/TrEMBL unreviewed entries are kept separated from the UniProtKB/Swiss-Prot manually reviewed entries so that the high-quality data of the latter is not diluted in any way. Automatic processing of the data enables the records to be made available to the public quickly.

UniProtKB is available for download from the URL ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/ and it is released in three different file formats: XML, fasta, and text.

Thanks to the availability of well documented and powerful XML API in the Java language, in this work the XML version of UniProKB/Swiss-Prot was utilized. It is accompanied by an XML schema file, which describes the structure of the XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD). The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

3.5 Reactome

Reactome is a database containing validated metabolic pathways in human biology and computationally inferred pathways for 20 non-human species.

Each pathway is annotated as a set of biological events, dealing with genes and proteins.

In order to support the scientific community, it provides a mapping among these entities and the main source database identifiers, such as UniprotKB and miRBase.

The download section of Reactome site at the URL http://www.reactome.org/pages/download-data/ contains many downloadable resources, such as:

- Identifier mapping files, which link the source database identifier to the lowest level pathway diagram or subset of the pathway or to all levels of the pathway hierarchy;
- Pathway information, that is the complete list of pathways and the pathway hierarchy relationship;
- Interactions derived from Reactome pathways, such as the human protein-

protein interaction pairs in tab-delimited format;
- MySQL dumps of Reactome database.

By executing some specifically custom SQL views, data of interest have been directly extracted from the SQL dumps of the database, which were previously imported into a dedicated MySQL server instance. Examples of such data are: pathway to disease relations, pathways summations, literature references.

3.6 miRBase

The miRBase database is a searchable database of published miRNA sequences and annotation.

Each entry in the miRBase Sequence database represents a predicted hairpin portion of a miRNA transcript (termed *mir* in the database), with information on the location and sequence of the mature miRNA sequence (termed *miR*).

Both hairpin and mature sequences are available for searching and browsing, and entries can also be retrieved by name, keyword, references and annotation.

The file at the URL ftp://mirbase.org/pub/mirbase/CURRENT/miRNA.dat.gz contains all the current published miRNA data in EMBL format.

Latest available database is version 21 released on June 2014.

3.7 mirCancer

miRCancer provides comprehensive collection of microRNA (miRNA) expression profiles in various human cancers which are automatically extracted from published literatures in PubMed. It utilizes text mining techniques for information collection. Manual revision is applied after auto-extraction to provide 100% precision.

User can search the database by miRNA and/or cancer names in the miRCancer Search page.

The website also provides two sequence analysis tools: clustering and chi-square analysis which can perform analysis on all or selected pool of miRNA sequences.

The latest miRCancer database is usually available for download upon request. At time of writing, the latest was updated on Dec. 3rd, 2015. In this work, the version http://mircancer.ecu.edu/downloads/miRCancerSeptember2015.txt was used.

3.8 mirTarBase

This resource is a collection of publicly available miRNA-target interactions databases and contains experimentally validated interactions, which are collected by manually surveying pertinent literature after data mining of the text systematically to filter research articles related to functional studies of miRNAs.

Generally, the collected MTIs are validated experimentally by reporter assay, western blot, microarray and next-generation sequencing experiments. While containing the largest amount of validated MTIs, miRTarBase provides the most updated collection by comparing with other similar, previously developed databases.

The URL http://mirtarbase.mbc.nctu.edu.tw/php/download.php provides all published miRNA target interaction data in Excel format.

3.9 HGNC

The HUGO Gene Nomenclature Committee (HGNC) is the authority responsible for the gene nomenclatures (also known as *gene symbols*) for the human species.

This authority also provides the HGNC database, that contains, for each gene symbol, a list of synonyms and a list of corresponding entries in the most popular gene databases.

The web page at URL http://www.genenames.org/cgi-bin/statistics shows a table that contains the number of genes associated to locus groups and types. Within the table there also are links to download the data for each locus group or type. The first source data file used in this work has been downloaded from the URL ftp://ftp.ebi.ac.uk/pub/databases/genenames/new/tsv/locus_groups/protein-coding_gene.txt and it has been used to load all the symbol synonyms of a given gene and to create associations of type coding between genes and proteins. Another data file, found at the URL ftp://ftp.ebi.ac.uk/pub/databases/genenames/new/tsv/locus_groups/non-coding_RNA.txt, has been instead used only to load other symbol synonyms for genes.

4 Building the database

4.1 Introduction

To efficiently manage the complexity and the extreme abundance of available data and external references, a modular Extract-Transform-Load (ETL) tool processes source data. A precise order of execution of ETLs sub-modules, derived from the ones developed in [14] [15], guarantees data consistency and proper relations between entities. This way, when a data source which refers to others is imported, the database already contains all the depending resources.

You can find a ready-to-use binary version of the software at the URL:

https://github.com/xMAnton/BioGrakn/releases/tag/v1.2.0

Choose a work directory and be sure to download the .jar and the two .gql files containing the ontology and the inference rules.

4.2 Notes on the ETL tool

Many of the data files are supplied in textual tab-separated values format, where each line of the text file is a record, and each field value of a record is separated from the next by a tab character.

It is almost trivial to develop sub-modules to handle such files. For example, the main code to import NCBI Entrez Gene is something like this:

```
…
System.out.print("Importing NCBI Gene ");

while ((line = reader.readLine()) != null) {
    String datavalue[] = line.split("\t");

    if (!datavalue[0].equals("9606"))
        continue;

    String symbol = datavalue[10].equals("-") ? datavalue[2]: datavalue[10];

    InsertQuery gene = insert(
       var("g")
               .isa("gene")
               .has("geneId", datavalue[1])
               .has("locusTag", datavalue[3])
               .has("chromosome", datavalue[6])
               .has("location", datavalue[7])
               .has("description", datavalue[8])
               .has("type", datavalue[9])
```

```
                .has("symbol", symbol)
                .has("fullName", datavalue[11])
    );

    loader.add(gene, keyspace);

    entryCounter++;

    if (entryCounter % 2500 == 0) {
         System.out.print(".");
    }

}

System.out.println(" done");
…
```

By contrast, miRBase, GO, and UniprotKB are available in EMBL text file format [16] and XML format, therefore different approaches are needed.

4.2.1 miRBase and GO

The entries in a database in EMBL format are structured so as to be usable by human readers as well as by computer programs. The explanations, descriptions, classifications and other comments are in ordinary English, and the symbols and formatting employed for the base sequences themselves have been chosen for readability. Wherever possible, symbols familiar to molecular biologists have been used. At the same time, the structure is systematic enough to allow computer programs easily to read, identify, and manipulate the various types of data included.

Each entry in the database is composed of lines. Different types of lines, each with its own format, are used to record the various types of data which make up the entry. In general, fixed format items have been kept to a minimum, and a more syntax-oriented structure adopted for the lines.

Each line begins with a two-character line code, which indicates the type of information contained in the line. The currently used line types, along with their respective line codes, are listed below:

```
ID - identification             (begins each entry; 1 per entry)
AC - accession number           (>=1 per entry)
PR - project identifier         (0 or 1 per entry)
DT - date                       (2 per entry)
DE - description                (>=1 per entry)
KW - keyword                    (>=1 per entry)
OS - organism species           (>=1 per entry)
OC - organism classification    (>=1 per entry)
OG - organelle                  (0 or 1 per entry)
RN - reference number           (>=1 per entry)
RC - reference comment          (>=0 per entry)
RP - reference positions        (>=1 per entry)
```

```
RX - reference cross-reference  (>=0 per entry)
RG - reference group            (>=0 per entry)
RA - reference author(s)        (>=0 per entry)
RT - reference title            (>=1 per entry)
RL - reference location         (>=1 per entry)
DR - database cross-reference   (>=0 per entry)
CC - comments or notes          (>=0 per entry)
AH - assembly header            (0 or 1 per entry)
AS - assembly information       (0 or >=1 per entry)
FH - feature table header       (2 per entry)
FT - feature table data         (>=2 per entry)
XX - spacer line                (many per entry)
SQ - sequence header            (1 per entry)
CO - contig/construct line      (0 or >=1 per entry)
bb - (blanks) sequence data     (>=1 per entry)
// - termination line           (ends each entry; 1 per entry)
```

Details on line types cited above can be found in [16] and [14].

The processing of biological data in EMBL format has been developed using *BioJava* [17], an open-source framework that enables rapid bioinformatics application development in the Java programming language. BioJava contains powerful analysis and statistical routines, tools for parsing common file formats and packages for manipulating sequences and 3D structures.

Let's consider miRBase's ETL. Its structure is similar to the one used to read textual tab-delimited files, but now the main loop iterates through the sequences found in the source file.

For each miRNA, the ETL extracts all the data and, by the reading of the features, it also extracts all the related mature miRNAs. This way, it gives values to two class of vertices, *miRNA* and *miRNAmature*, and creates edges of type *precursorOf* between them.

```
…
BufferedReader br = new BufferedReader(new FileReader(fileName));
Namespace ns = RichObjectFactory.getDefaultNamespace();
RichSequenceIterator seqs = RichSequence.IOTools.readEMBLRNA(br, ns);

System.out.print("Importing miRBase ");

while (seqs.hasNext()) {
  RichSequence entry = seqs.nextRichSequence();

  String accession = entry.getAccession();
  String name = entry.getName();
  String description = entry.getDescription();
  Vector<String> comments = new Vector<String>();

  for (Comment comment : entry.getComments()) {
    String cmt = comment.getComment().replaceAll("\n", " ");
    comments.add(cmt);
  }
  String comment = "";
  if (comments.size() > 0)
    comment = comments.get(0);
```

```
  String sequence = entry.getInternalSymbolList().seqString();

  InsertQuery mirna = insert(
                      var("m")
                      .isa("mirna")
                      .has("accession", accession)
                      .has("name", name)
                      .has("description", description)
                      .has("comment", comment)
                      .has("sequence", sequence));

  loader.add(mirna, keyspace);

  entryCounter++;

  Iterator<Feature> itf = entry.getFeatureSet().iterator();

  int cnt = 1;
  while (itf.hasNext()) {
    Feature f = itf.next();

    String location = f.getLocation().toString();
    String subSequence =
          sequence.substring(f.getLocation().getMin()-1, f.getLocation().getMax());
    String matAccession = "";
    String matProduct = "";

    @SuppressWarnings("unchecked")
    Map<Object, ?> map = f.getAnnotation().asMap();
    Set<Object> keys = map.keySet();
    for (Object key : keys) {
        String keyString = key.toString();
        String value = (String) map.get(key);

        if (keyString.substring(keyString.lastIndexOf(":")+1).equals("accession"))
          matAccession = value;

        if (keyString.substring(keyString.lastIndexOf(":")+1).equals("product"))
          matProduct = value;
    }

    InsertQuery mature = insert(
                               var("mat" + cnt)
                               .isa("mirnaMature")
                               .has("accession", matAccession)
                               .has("product", matProduct)
                               .has("sequence", subSequence)
                               .has("location", location));

    loader.add(mature, keyspace);

    entryCounter++;

    Query<?> rel = match(
                var("m1").isa("mirna").has("accession", accession),
                var("m2").isa("mirnaMature").has("accession", matAccession)).
              insert(
                var("p"+cnt).isa("precursorOf").
                            rel("precursor", "m1").rel("mature", "m2")
              );

    loader.add(rel, keyspace);

    cnt++;
  }
```

```
  if (entryCounter % 1000 == 0) {
    System.out.print(".");
  }

}

System.out.println(" done");

…
```

4.2.2 UniprotKB

The XML version of UniProKB/Swiss-Prot is accompanied by an XML schema file, which describes the structure of the XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD). The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

In order to get a Java representation of the XML schema cited above, it is necessary to map the elements of the schema to members of a (or more) java class. This transformation can be done using the data binder JAXB [18].

JAXB generates classes and groups them in Java packages. A package consists of a Java class name and an *ObjectFactory* class. The latter is a factory that is used to return instances of a bound Java class.

After data bindings exist, the JAXB binding runtime API can be used to convert XML instance documents to and from Java objects. Data stored in an XML document is accessible without the need to understand the data structure. JAXB annotated classes and artifacts contains all the information that the JAXB runtime API needs to process XML instance documents. The JAXB runtime API enables marshaling of JAXB objects to XML and unmarshaling the XML document back to JAXB class instances.

In this case, the ETL reads the XML source document by using the Streaming API for XML (StAX) [19]. It provides the interface *XMLStreamReader*, which gives a low-level but very efficient cursor-like API for reading XML documents. When using it we iterate over various events in XML document and extract information about these events. Once we are done with the current event, we move to the next

one and continue. The events can be for example the start of element, the end of element or characters data.

```
…
while (xsr.nextTag() == XMLStreamConstants.START_ELEMENT) {
            Entry entry = (Entry) unmarshaller.unmarshal(xsr);

  OrganismType organism = entry.getOrganism();
  String organismTaxonomyId = ((organism != null) &&
                  (!organism.getDbReference().isEmpty())) ?
                  organism.getDbReference().get(0).getId() : "";

  if (organismTaxonomyId.equals("9606")) {

    if (entry.getAccession().isEmpty())
      continue;

    ProteinType prot = entry.getProtein();

    String name = entry.getName().get(0);
    String fullName = ((prot.getRecommendedName() != null) &&
              (prot.getRecommendedName().getFullName() != null)) ?
              prot.getRecommendedName().getFullName().getValue() : "";
    String alternativeName = ((!prot.getAlternativeName().isEmpty()) &&
            (prot.getAlternativeName().get(0).getFullName() != null)) ?
          prot.getAlternativeName().get(0).getFullName().getValue() : "";

    String gene = "";
    if (!entry.getGene().isEmpty()) {
      GeneType geneType = entry.getGene().get(0);
      if (!geneType.getName().isEmpty()) {
        gene = geneType.getName().get(0).getValue();
      }
    }

    SequenceType seq = entry.getSequence();
    String sequence = seq.getValue();
    int sequenceLength = seq.getLength();
    int sequenceMass = seq.getMass();

    String function = "";
    String pathway = "";
    String subunit = "";
    String tissue = "";
    String ptm = "";
    String similarity = "";

    for (CommentType comment : entry.getComment()) {
      if (comment.getText().isEmpty())
        continue;

      String s = comment.getText().get(0).getValue();

      if (comment.getType().equals("function")) {
        function = s;
      } else if (comment.getType().equals("pathway")) {
        pathway = s;
      } else if (comment.getType().equals("subunit")) {
        subunit = s;
      } else if (comment.getType().equals("tissue specificity")) {
        tissue = s;
      } else if (comment.getType().equals("PTM")) {
        ptm = s;
```

```
      } else if (comment.getType().equals("similarity")) {
        similarity = s;
      }
    }

    Query<?> protein = insert(
                       var("p")
                       .isa("protein")
                       .has("name", name)
                       .has("fullName", fullName)
                       .has("alternativeName", alternativeName)
                       .has("proteinGene", gene)
                       .has("function", function)
                       .has("proteinPathway", pathway)
                       .has("subunit", subunit)
                       .has("tissue", tissue)
                       .has("ptm", ptm)
                       .has("similarity", similarity)
                       .has("sequence", sequence)
                       .has("sequenceLength", sequenceLength)
                       .has("sequenceMass", sequenceMass)
    );

    loader.add(protein, keyspace);

    entryCounter++;

    int cnt = 1;
    for (String accessionName : entry.getAccession()) {
      Query<?> accession =
          match(var("p").isa("protein").has("name", name)).
          insert(
             var("acc" + cnt).
               isa("proteinAccession").
               has("accession", accessionName),
             var("rel" + cnt).
               isa("entityReference").
               rel("identified", "p").rel("identifier", "acc"+cnt)
          );

          loader.add(accession, keyspace);

          entryCounter++;
           cnt++;
    }

    if (entryCounter % 1000 == 0) {
       System.out.print(".");
    }
  }
}

System.out.println(" done");
…
```

4.3 Data sources download

For your convenience, all the used data sources have been collected and they are available for download from the URL

http://194.119.214.173/biograkn/.

Create a destination directory on your computer, e.g., *~/datasources*, and then copy the downloaded files into it.

Alternatively, use a preferred method of yours to download all the files together. For example, with *wget*:

```
$ wget -A .bz2 -r -nd -nv -P ~/datasources http://194.119.214.173/biograkn/
```

Then, uncompress the files:

```
$ bunzip2 ~/datasources/*
```

4.4 Load the ontology and the inference rules

With GRAKN.AI up and running, load the ontology and the inference rules:

```
$ cd [YOUR-GRAKN-1.2.0-DIR]
$ ./graql console -k biograkn -f [WORKDIR]/ontology.gql
$ ./graql console -k biograkn -f [WORKDIR]/rules.gql
$ cd [WORKDIR]
```

Note that you can use an ad-hoc keyspace, such as *biograkn* used above.

4.5 Import data

The data import process is handled by the java program *BuildBioGrakn* contained in the previously downloaded .jar file.

Its usage is briefly explained by running it with *-h* command line option:

```
$ java -jar BuildBioGrakn.jar -h
usage: BuildBioGrakn
-d <arg>   data source path
-h         print this help
-k <arg>   keyspace
```

With no options, the program will use the following default values:

name	value
data directory	*~/datasources*
keyspace	*biograkn*

The output should be as follow (except for the execution time, obviously!):

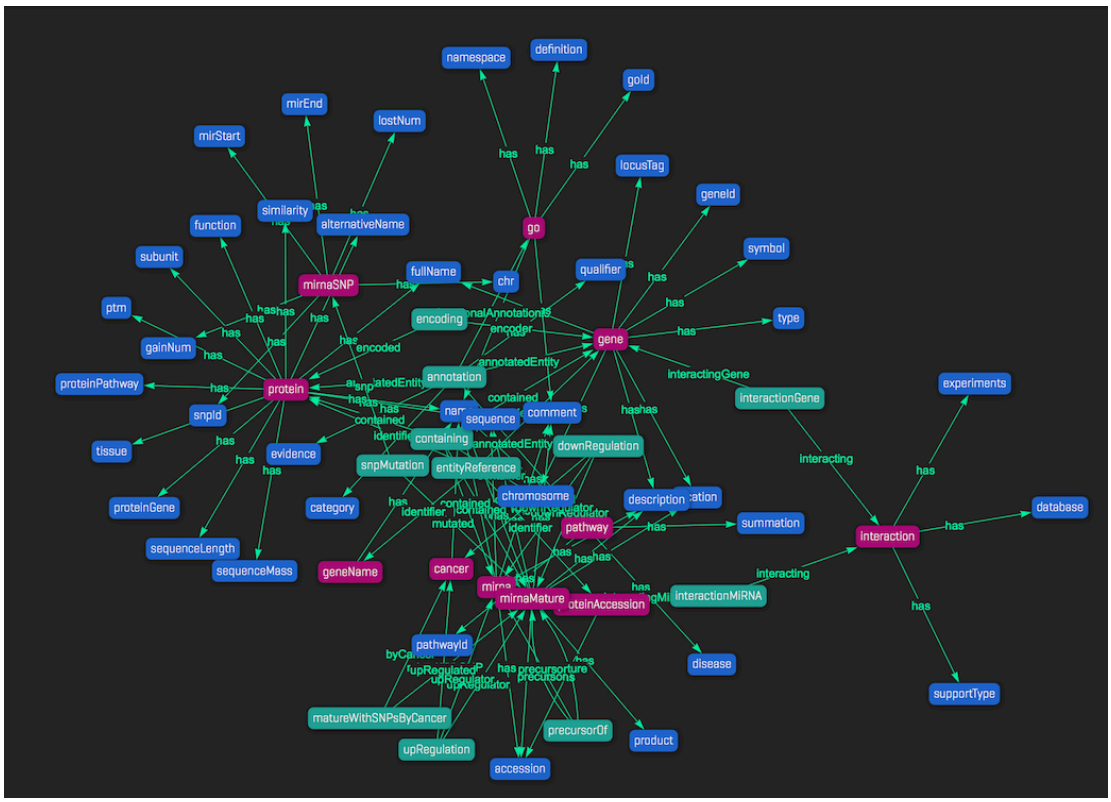
```
$ java -jar BuildBioGrakn.jar

Building BioGrakn ...

Importing NCBI Gene ..................... done
Importing Gene Ontology ..................... done
Importing Gene2GO ..................... done
Importing miRBase ..................... done
Importing Reactome ..................... done
Importing Reactome2GO ..... done
Importing Reactome2miRNA ..... done
Importing miRCancer ..... done
Importing Uniprot ..................... done
Importing Uniprot2Reactome ..... done
Importing HGNC ..................... done
Importing miRNASNP ..... done
Importing miRTarBase ..... done

BioGrakn built in 0 hours 31 minutes 21 seconds
```

Now, if you open the web visualizer, querying for "all types" will show something like this:



5 References

[1] K. H. Cheung, A. K. Smith, K. Y. Yip, C. J. Baker and M. B. Gerstein, "Semantic Web approach to database integration in the life sciences," in *Semantic Web*, Springer US, 2007, pp. 11-30.

[2] C. T. Have and L. J. Jensen, "Are graph databases ready for bioinformatics?," *Bioinformatics,* vol. 29, no. 24, pp. 3107-3108, 2013.

[3] A. Messina, H. Pribadi, J. Stichbury, M. Bucci, S. Klarman and A. Urso, "BioGrakn: A Knowledge Graph-Based Semantic Database for Biomedical Sciences," in *Conference on Complex, Intelligent, and Software Intensive Systems*, Turin, 2017.

[4] Grakn Labs Ltd, "GRAKN.AI," [Online]. Available: https://grakn.ai. [Accessed 2 5 2018].

[5] G. D. Schuler, J. A. Epstein, H. Ohkawa and J. A. Kans, "Entrez: molecular biology database and retrieval system," in *Methods in enzymology*, vol. 266, 1996, pp. 141-162.

[6] The Gene Ontology Consortium, "The Gene Ontology project in 2008," *Nucleic Acids Research,* no. 34, pp. D440-444, 2008.

[7] The UniProt Consortium, "UniProt: a hub for protein information," *Nucleic Acids Research,* vol. 43, no. D1, pp. D204-D212, 2015.

[8] D. Croft, A. F. Mundo, R. Haw, M. Milacic, J. Weiser, G. Wu, M. Caudy, P. Garapati, M. Gillespie, M. R. Kamdar, B. Jassal, S. Jupe, L. Matthews, B. May, S. Palatnik, K. Rothfels, V. Shamovsky, H. Song, M. Williams, E. Birney, H. Hermjakob, L. Stein and P. D'Eustachio, "The Reactome pathway knowledgebase," *Nucleic Acids Research,,* vol. 42, no. D1, pp. D474-7, 2014.

[9] A. Kozomara and S. Griffiths-Jones, "miRBase: integrating microRNA annotation and deep-sequencing data," in *Nucleic acids research*, vol. 39, 2011, pp. 152-157.

[10] B. Xie, Q. Ding, H. Han and D. Wu, "miRCancer: a microRNA-cancer association database constructed by text mining on literature," in *Bioinformatics*, vol. 29, 2013, pp. 638-644.

[11] J. Gong, Y. Tong, H. M. Zhang, K. Wang, T. Hu, G. Shan, J. Sun and A. Y. Guo, "Genome-wide identification of SNPs in microRNA genes and the SNP effects on microRNA target binding and biogenesis," *Human Mutation,* vol. 33(1), pp. 254-263, 2012.

[12] C. Chou, S. Shrestha, C. Yang, N. Chang, Y. Lin, K. Liao, W. Huang, T. Sun, S. Tu, W. Lee, M. Chiew, C. Tai, T. Wei, T. Tsai, H. Huang, C. Wang, H. Wu, S. Ho, P. Chen, C. Chuang, P. Hsieh, Y. Wu, W. Chen, M. Li, Y. Wu, X. Huang and N, "miRTarBase update 2018: a resource for experimentally validated microRNA-target interactions," *Nucleic Acids Research,* vol. 46, no. D1, pp. D296-D302, 2018.

[13] K. A. Gray, B. Yates, R. L. Seal, M. W. Wright and E. A. Bruford, "Genenames.org: the HGNC resources in 2015," *Nucleic Acids Research,* vol. 43, no. D1, pp. D1079-D1085, 2015.

[14] A. Messina, "ETLs for importing NCBI Entrez Gene, miRBase, mirCancer and microRNA into a bioinformatics graph database," Palermo, 2015.

[15] A. Messina, "ETLs for importing UniProtKB, HGNC, and Reactome into a bioinformatics graph database," Palermo, 2016.

[16] European Bioinformatics Institute, "EMBL Outstation," [Online]. Available: ftp://ftp.ebi.ac.uk/pub/databases/embl/doc/usrman.txt. [Accessed 15 12 2015].

[17] R. C. G. Holland, T. A. Down, M. Pocock, A. Prlić, D. Huen, K. James, S. Foisy, A. Dräger, A. Yates, M. Heuer and M. J. Schreiber, "BioJava: an open-source framework for bioinformatics," *Bioinformatics,* vol. 24, no. 18, pp. 2096-2097, 15 Sep 2008.

[18] Java Community Process, "JSR 222: Java Architecture for XML Binding (JAXB) 2.0," 2009. [Online]. Available: https://jcp.org/en/jsr/detail?id=222. [Accessed 04 02 2016].

[19] Java Community Process, "JSR 173: Streaming API for XML," 04 Mar 2014. [Online]. Available: https://jcp.org/en/jsr/detail?id=173. [Accessed 12 Feb 2016].