Accessing Scientific Data through Knowledge Graphs with *Ontop*

Diego Calvanese[1,2,3,*,†], Davide Lanti[1], Tarcisio Mendes De Farias[4,5], Alessandro Mosca[1], Guohui Xiao[1,3]

[1] Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, 39100, Italy

[2] Umeå University Sweden

[3] Ontopic S.R.L. Bolzano, 39100, Italy

[4] SIB Swiss Institute of Bioinformatics (Switzerland)

[5] University of Lausanne (Switzerland)

[*] Lead Contact

[†] Correspondence: `calvanese@inf.unibz.it`

The Bigger Picture

Knowledge Graphs (KGs) have recently gained attention [11] due to their flexible data model, which reduces the effort needed for integration across different, possibly heterogeneous, data sources. In this tutorial, we will learn how to access scientific data stored in a relational database through the *Virtual* Knowledge Graphs (VKGs) approach. In such an approach, the data is exposed as a KG and enriched with semantic information coming from a domain *ontology.* The KG is "virtual" in the sense that the data is not replicated, but stays within the data sources.

We demonstrate the approach over scientific data coming from the biomedical domain, and using the VKG system *Ontop*. By exposing legacy data as a KG, users can access the data by means of a more convenient vocabulary provided by the domain ontology, benefit from automatic reasoning capabilities, and do not need to focus on how the data is actually stored. Furthermore, the virtual approach allows for the use of KGs even in those contexts where the user does not own the data, nor is granted the rights to make a copy of it.

By relying on existing federation tools, the approach described here for accessing scientific data, can also be used to integrate multiple, heterogeneous, and possibly semi-structured and unstructured data sources.

Abstract

In this tutorial we learn how to set-up and exploit the Virtual Knowledge Graph (VKG) approach to access data stored in relational legacy systems, and to enrich such data with domain knowledge coming from different heterogeneous (biomedical) resources. The VKG approach is based on an ontology that describes a domain of interest in terms of a vocabulary familiar to the user, and exposes a high-level conceptual view of the data. Users can access the data by exploiting the conceptual view, and in this way they do not need to be aware of low-level storage details. They can easily integrate ontologies coming from different sources, and can obtain richer answers thanks to the interaction between data and domain knowledge.

1 Introduction and Motivation

Large scale molecular biology experiments, the adoption of computational tools and algorithms to study biological pathway networks[1], the effective analysis of genome sequences from various model organisms and, more in general, the advent of a systems approach for the analysis and modelling of complex biological systems [21, 13], all require advanced data management technologies that ease the access to and the integration of massive amounts of information coming from different, usually heterogeneous, data sources.

Knowledge Graphs (KGs) gained popularity recently [11] as a general mechanism to represent data that is not constrained to a rigid schema, and to enrich such data with domain semantics. The absence of a rigid schema, and the elementary yet flexible abstraction provided by "*subject-predicate-object*" triples at the basis of KGs, allow on the one hand for the evolution of data sources in all those situations where the "schema" of the data cannot be determined in advance, and on the other hand for the integration and interoperability of heterogeneous data sources and among different scientific data plat-

[1] `https://www.genome.gov/about-genomics/fact-sheets/Biological-Pathways-Fact-Sheet`

forms.

The semantic information in a KG is provided by an *ontology*, which is a structured formal representation of the concepts that are relevant in a domain of interest and of the relationships between them. The purpose of the ontology is twofold. On the one hand, it defines a vocabulary of terms to denote *classes* and *properties* that are familiar to the user. On the other hand, it extends the data with background knowledge, such as sub-class and sub-property axioms, axioms establishing which classes constitute the domain and range of properties, and axioms expressing the disjointness between classes or properties.

The data in a KG consists of a set of *data assertions* that use the vocabulary of classes and properties provided in the ontology. Data assertions are often obtained by mapping the data stored in various data sources to the terms of the ontology vocabulary. Intuitively, a *mapping* can be thought as a collection of queries that are used to construct the data assertions of the ontology by retrieving the necessary data from the sources.

The data sources are typically legacy systems and might come in different forms, such as relational databases (DBs), or as files in various formats (such as CSV, XML, JSON, or proprietary formats). For the purpose of this tutorial, we assume to deal with a single relational data source. To deal with multiple heterogeneous data sources one can resort to a data federation tool, such as Denodo[2], Dremio[3], or Teiid[4], which exposes such sources as if they were part of a single relational DB.

KGs, through an explicit and non-ambiguous representation of the semantics of the data, promote:

- interoperability among different scientific *data platforms* (e.g., GDC[5] and ELIXIR[6]), *resources* (e.g., UBERON[7] and CHEBI[8]) and *data models* (e.g., SBML[9] and BioPAX[10]);

- scientific reproducibility and replicability of experimental studies;

- knowledge discovery and data mining practices by exposing a conceptually sound view over a multiplicity of distinct and possibly non-interoperable data sources, therefore reducing the negative impact of inputting nonsensical or inconsistent data into statistical models and learning algorithms;

- the enrichment of the information originally present in the data sources, through the application of reasoning techniques that combine domain knowledge and data assertions.

In a *Virtual Knowledge Graph* (VKG) [22, 23], the data assertions are not materialized in a separate data store, but their presence in the KG is only virtual. Systems operating on VKGs are able to retrieve the data directly from the data sources only when it is required for a particular user query. In fact, query processing is delegated to the data sources. This is achieved by *unfolding* the mappings, thus translating user queries into queries over the data sources, whilst taking into account also the ontology background knowledge through a so-called *query rewriting* step. The advantage of VKGs is that information is always fresh and up-to-date with the data sources.

Despite the advantages of the virtual approach, it is sometimes convenient to actually *materialize* the data assertions. In such a case, we talk about *Materialized Knowledge Graphs* (MKGs). The main advantage of MKGs over VKGs is that usually a better performance in query answering can be achieved, especially in those situations where mappings are very complex and thus the unfolding of the virtual approach would give rise to complex queries over the data sources. This comes at the cost of maintaining a potentially very large MKG.

Figure 1 shows the conceptual framework of KGs, in both the materialized and virtual flavors. The elements of the KG framework are expressed in formal languages standardized by the World Wide Web Consortium (W3C), specifically: the Knowledge Graph in RDF [7], the ontology in OWL 2 QL [2], the mapping in R2RML [8], and the query in SPARQL [10].

In this tutorial we make use of the VKG system *Ontop* [4, 24] to set up a KG in the biomedical domain, specifically in the area of cancer research.

We observe that *Ontop* has been conceived as a VKG system, but it offers also functionalities for materializing a KG from a (V)KG specification consisting of an ontology, a relational data source, and a mapping between the two. All the material used in the tutorial, as well as full details for its usage are available in an online GitHub repository[11].

Related Work on VKG Systems. Among the open-source VKG systems, *Ontop* is one of the most *popular* (with over 30K+ downloads in the past

[2]`https://www.denodo.com/`
[3]`https://www.dremio.com/`
[4]`https://teiid.io/`
[5]`https://portal.gdc.cancer.gov/`
[6]`https://elixir-europe.org/`
[7]`https://uberon.github.io/`
[8]`https://www.ebi.ac.uk/chebi/`
[9]`http://sbml.org/`
[10]`http://www.biopax.org/`
[11]`https://github.com/ontop/ontop-patterns-tutorial/`

5 years, according to Sourceforge). *Ontop* is a state-of-the-art VKG system initially developed by the Research Centre for Knowledge and Data (KRDB) at the Free University of Bozen-Bolzano, and currently maintained as a community effort, involving both academic institutions and companies (most notably, Birkbeck University of London and Ontopic S.R.L.). The system has been adopted in many academic projects[12], and it also has a number of commercial deployments, such as the *UNiCS* open data platform by SIRIS Academic[13] (Spain) and the *Open Data Hub Virtual Knowledge Graph* project for publishing tourism data of South Tyrol (Italy). All the authors of this tutorial have a profound expertise in the system, some being the maintainers since its inception.

An overview of popular commercial and non-commercial VKG systems, as well as a comparison between *Ontop* and other systems, goes beyond the scope this work. For such aspects, we refer the interested reader to the vast scientific literature, e.g., [17, 23, 16, 6, 1].

2 The EasyBgee Dataset

Gene expression is a key process to understand the relation between genes and their function. It indicates or mediates the gene implication in functions, disease development, and organism, species or gene diversity. In this context, the Bgee DB[14] is a public relational DB that consolidates and curates heterogeneous gene expression data sources [3].

Figure 2 illustrates the data schema of the EasyBgee DB (which is available as a MySQL dump[15]) [19], a simplified version of the Bgee DB. Currently, EasyBgee 14.2 contains gene expression data of 29 species. In this tutorial, we consider a subset of EasyBgee to explain and demonstrate the main principles of the KG approach to data integration and data access. This subset has exactly the same data schema as the entire EasyBgee DB, however with considerably fewer data. It solely includes data related to 129 genes out of 17,559 in the fruit fly species (i.e., *Drosophila Melanogaster*). As a result, this subset corresponds to less than 8 MB of data when serialised in a MySQL or PostgreSQL dump text format. The DB subset dump is available for download in the GitHub repository of the tutorial.

As shown in Figure 2, EasyBgee consists of 6 tables, 29 columns, and 6 foreign-key constraints (represented with arrows). We provide a brief description of the tables:

- the `anatentity` table contains data about anatomic entities such as organs (e.g., “brain”);
- the `stage` table describes developmental stages related to several species (e.g., the “egg stage”);
- the `gene` table contains the gene names and descriptions from different species;
- the `species` table contains information about animal species, such as their scientific and common names;
- the `globalcond` table contains the experimental conditions of a gene expression analysis, such as the species, its developmental stage, and the anatomical entity considered in the analysis;
- the `globalexpression` table contains the gene expression patterns by relating with a score a gene to an experimental condition where the gene is expressed or absent.

3 Setting up the VKG

In this section we discuss how to set up an instance of a VKG system by means of a concrete use-case coming from the domain of Bio-informatics.

3.1 The Gene Expression Ontology

A crucial step in the deployment of a VKG system consists in the design or re-use of an ontology that suitably represents the implicit semantics of the underlying data. For the sake of the present tutorial, in the gene expression domain we highlight the *Gene Expression Ontology* (GenEx)[16], which is specifically designed to structure gene expression data from DBs such as EasyBgee. In addition to new terms defined in the ontology itself, GenEx imports terms from different vocabularies such as *Relation Ontology* (RO) [20]. As an example, GenEx imports from RO the *expressed in* property (actually identified by `obo:RO_0002206`[17]) and its inverse property *expresses* (identified by `obo:RO_0002292`[18]). Examples of data assertions using these properties are: “the insulin gene is *expressed in* the body of pancreas”, and its inverse statement, “the body of pancreas *expresses* the insulin gene”. Moreover, GenEx specializes the RO *expressed in* property by defining `genex:isExpressedIn` as its sub-property with a specific domain and range (see also bottom part of Figure 4). The domain and range of `genex:isExpressedIn` include `orth:Gene`[19] and `genex:AnatomicalEntity`, respectively. Therefore,

[12]Most notably, the two European projects FP7 Optique (`http://optique-project.eu/`) and H2020 INODE (`http://www.inode-project.eu/`).

[13]`https://www.sirisacademic.com/`

[14]`https://bgee.org`

[15]`ftp://ftp.bgee.org/current/easybgee_dump.tar.gz`

[16]`https://biosoda.github.io/genex/`

[17]`http://purl.obolibrary.org/obo/RO_0002206`

[18]`http://purl.obolibrary.org/obo/RO_0002292`

[19]`https://qfo.github.io/OrthologyOntology/#Gene`

we can assert that a gene is expressed in an anatomical entity (e.g., body of pancreas) using the `genex:isExpressedIn` property and automatically infer a more general statement about its RO superproperty. Similar definitions in the ontology contribute to enhance interoperability because different ontology terms (RO and GenEx specific) can be interchangeably used to retrieve gene expression calls. Notice that all terms are prefixed with labels that indicate the ontology they were originally defined in and that allow for compact URIs. The prefixes used in the present article, such as `genex:` and `orth:`, are defined in Table 1.

Furthermore, GenEx reuses other ontology terms not only as part of its terminology but also as part of the data assertions by acting as a controlled vocabulary. For example, in GenEx, the Uber-anatomy (UBERON) ontology classes are considered as instances of the classes `genex:AnatomicalEntity` or `efo:EFO_0000399` (i.e., "developmental stage") by punning[20]. The classes of the species-specific developmental stage ontologies[21] are also considered as instances of the `efo:EFO_0000399` class which has been imported from the Experimental Factor Ontology (EFO) [15]. Therefore, these controlled vocabulary terms are assigned as specific values of the `genex:isExpressedIn` property. As a result, we enhance semantic and data interoperability with other data sources that also adhere to the same controlled vocabularies. For example, while re-using the term `obo:UBERON_0000955`[22] (labelled as "brain") in our data assertions, we know that we are referring without any ambiguity to the very same organ as the one defined in the UBERON ontology. Notice also that, by applying the above strategy, we enable the enrichment of the information coming from the original data sources with further assertions coming from the ontology specification. For example, this enrichment allow us to retrieve the UBERON data assertions, such as cross-references (i.e., similar terms), related synonyms, and *part of* assertions (e.g., "the brain is *part of* the central nervous system") that are not available in the original data source (e.g., EasyBgee). For further details about GenEx, please consult its documentation[16].

3.2 Mapping EasyBgee to GenEx

The second main step in deploying a VKG system consists in the specification of its mapping. If we consider the EasyBgee data source, we can observe that for each of its tables there is at least one corresponding class in GenEx. Now, in order to populate GenEx using a system like *Ontop*, we need to define suitable mapping assertions from the tables we are interested in to their corresponding classes and properties in the ontology. Each of these mapping assertions consists of three components:

1. The *mapping identifier*, which uniquely identifies the mapping assertion.
2. The *source* part, which is an ordinary SQL query expressed over the (relational) data source.
3. The *target* part, consisting of a set of RDF triple patterns that make use of the answer variables of the SQL query in the source part. Such answer variables are written by enclosing them in curly brackets '`{`' and '`}`'.

The meaning of such a mapping assertion is that it constructs a portion of the (virtual) KG as specified in the target part, by retrieving the required data from the data source through the SQL query in the source part. More specifically, for each answer tuple $\vec{t}$ returned by such SQL query, the triples in the target part are asserted to hold in the (virtual) KG constructed by the mapping. These triples are obtained by substituting in the triple pattern each SQL answer variable (enclosed in '`{`' and '`}`') with the corresponding value in the answer tuple $\vec{t}$. We remark that when the KG is kept *virtual*, the triples are actually not constructed, but SPARQL queries are answered over the VKG (as if they were evaluated over such triples) by unfolding them to (SQL) queries over the data source.

Let us also notice that the mappings that are introduced in this tutorial are specified in the *Ontop* native mapping language, which is easier to learn and use. However, *Ontop* allows users to convert native mappings into R2RML mappings and vice-versa, and the R2RML version of the mappings introduced here is available in the tutorial GitHub repository.

To illustrate the use of mappings, let us consider the mapping assertion depicted in the upper part of Figure 3, between the EasyBgee DB and the GenEx ontology:

1. The mapping identifier is `AnatomicalEntity`.
2. The source part is a SQL query over the relation `anatEntity`, with answer variables `anatEntityId`, `anatEntityIdSPARQL`, `anatEntityName`, and `anatEntityDescription` (see also the data schema in Figure 2).
3. The target part consists of the following three RDF triple patterns, which refer to the answer variables of the SQL query (shown in violet, in Figure 3):

```
obo:{anatEntityIdSPARQL} a
  genex:AnatomicalEntity .
```

[20] https://www.w3.org/TR/owl2-new-features/#F12:_Punning

[21] https://github.com/obophenotype/developmental-stage-ontologies/wiki

[22] http://purl.obolibrary.org/obo/UBERON_0000955

```
obo:{anatEntityIdSPARQL} dcterms:description
  {anatEntityDescription} .
obo:{anatEntityIdSPARQL} rdfs:label
  {anatEntityName} .
```

Notice that in Figure 3 we have used the standard abbreviated notation for RDF triples patterns (and triples), where one avoids to repeat the subject in multiple triples with the same subject, by separating these triples with '`;`' (instead of '`.`').

When designing the mapping, we can take advantage of SQL functions and existing ontologies to deal with potential semantic and data heterogeneity in the source data. As illustrated in Figure 3, we use the SQL function `replace()` to modify the anatomical entity identifiers stored in the `anatEntity` table by replacing each '`:`' with '`_`'. In addition, we prepend the `obo:` prefix to the modified ids in order to obtain the exact corresponding UBERON term, such as the `obo:UBERON_0000955` term labelled as *brain*. Another example is a mapping assertion for the `species` table, where we use the SQL function `concat()` to concatenate the values from the two columns `species.genus` and `species.species` into a single value, which is then assigned as value for the property `up:scientificName` of a species from the UniProt core ontology[23], which GenEx imports.

A relevant feature that we can exploit when designing ontology and mapping in the VKG approach, is the possibility to semantically enrich the original data sources by making implicit information explicit. To illustrate such semantic enrichment, let us consider the second mapping assertion in Figure 3. There is no foreign key constraint in the original DB directly relating a gene (in the `gene` table) to an anatomical entity (in the `anatentity` table), as shown in Figure 2. Moreover, there is no column or table stating the explicit relation between genes and anatomic entities, as the one defined in the GenEx ontology by means of the `genex:isExpressedIn` property. Nonetheless, by means of the mapping assertion with id `Gene_IsExpressedIn_anatEntity`, the VKG system is able to assert `genex:isExpressedIn` properties (between instances of the classes `orth:Gene` and `genex:AnatomicalEntity`). In addition, thanks to the reasoning capabilities of *Ontop* and the sub-property and inverse property axioms in the GenEx ontology discussed in Section 3.1 (see also bottom part of Figure 4), the system automatically derives also data assertions for the property `obo:RO_0002292` (i.e., *expresses*). This is because `genex:isExpressedIn` is a sub-property of the inverse of `obo:RO_0002292`. Therefore, although we explicitly specified only a mapping assertion for the `genex:isExpressedIn` property, when the VKG system retrieves the corresponding data assertions, due to its reasoning capabilities, it infers also data assertions for the property `obo:RO_0002292`.

Similarly to inferences concerning the RO property *expresses*, via reasoning the VKG system is also capable of inferring instances of equivalent classes, i.e., classes that have been aligned in the ontology by means of `owl:equivalentClass`[24] statements. For example, GenEx states that `orth:Gene` from the Orthology Ontology (ORTH) [9] and `obo:SO_0000704` (which stands for *gene*) from the Sequence Ontology (SO) are equivalent, as shown in the top part of Figure 4. Therefore, the instances of one concept are also instances of the other one and vice-versa, although in the VKG specifications we include only a mapping assertion for the `orth:Gene` concept. As a result, we can interchangeably use `orth:Gene` and `obo:SO_0000704`, and consider them as synonyms (see Query 2 below).

3.3 Designing the Ontology and Mapping

As we have seen in the previous section, to enable the VKG approach, one needs an ontology describing the domain of interest and a mapping populating such ontology starting from the content of the database.

Designing an ontology is not an easy task, and in many domains (e.g., the biomedical one) ontologies are developed independently by trained experts and are already available to be re-used. However, if a domain ontology that suits the user requirements is not available, the user can rely on the open-source tool Protégé shown in this tutorial, which provides a graphical interface that helps in designing an ontology from scratch or in modifying an already existing one. Protégé also comes with a set of plugins for debugging and visualizing the ontology, and integrated reasoners that help the conceptual modelling activity.

Writing a mapping manually is a time-consuming, error-prone task, and automatic approaches to mapping generation are still an open research topic. Notable recent developments in this area involve both foundational research [12, 5] and implemented systems [12, 18].

4 Querying the KG

We are now ready to query the KG, created as described in the previous sections, through the SPARQL query language, a W3C recommendation

[23]`https://www.uniprot.org/core/`

[24]`https://www.w3.org/TR/2012/REC-owl2-primer-20121211/#a_EquivalentClasses`

for querying KGs [10]. We rely on the *Ontop* Plugin for the ontology editor Protégé [25], which provides a graphical user interface both for managing the mapping between a DB and an ontology, and also for specifying and executing queries over the resulting KG specification in the virtual setting. The *Ontop* Plugin visualizes the result of the query in a dedicated window.

We illustrate the main ideas behind query answering over a KG on three example queries.

Query 1. The query Q_1 shown in Figure 5 asks for the gene information that is associated to a given gene name, in this case the "boss" (the bride of sevenless) gene product that acts as a ligand for the sevenless tyrosine-kinase receptor during eye development [14]. Figure 5 shows the graphical interface provided by the *Ontop* Plugin for Protégé for formulating SPARQL queries and visualizing the result of their execution. The bottom part of the window shows the result of the execution of Q_1, which in case is a single set of bindings for the three answer variables of the SPARQL query.

Query 2. The following query Q_2 retrieves the anatomical entities, such as organs, where the gene labeled `"boss"` is expressed.

```
SELECT DISTINCT ?organ {
  VALUES ?gene_name {"boss"}
  ?gene a obo:SO_0000704 ;
                   # equivalent to orth:Gene
        rdfs:label ?gene_name ;
        genex:isExpressedIn ?organ .
  ?organ a genex:AnatomicalEntity .
}
```

We observe that results are still returned, although Q_2 asks for individuals of the non-mapped class `obo:SO_0000704`. This happens since the ontology states that `obo:SO_0000704` (which is labelled as gene) from SO is equivalent to the class `orth:Gene` from ORTH, and reasoning over the ontology axioms is applied. Intuitively, this happens by *rewriting* the query into one that uses the term `orth:Gene`, instead of `obo:SO_0000704`, and by retrieving results also for this rewritten query. Therefore, we can interchangeably use both terms, although we only wrote a mapping assertion for `orth:Gene`. As already mentioned in the previous section, this simplifies the design of the mapping, because it allows one to avoid to extensively and explicitly write a mapping assertion for each term in the ontology.

[25] `https://sourceforge.net/projects/ontop4obda/files/ontop-4.1.0/`

Query 3. Consider now the following query Q_3, which retrieves all genes expressed in the brain.

```
SELECT  ?gene_name  ?gene_page {
  ?organ obo:RO_0002292 ?gene ;
         rdfs:label "brain".
  ?gene  rdfs:seeAlso ?gene_page ;
         rdfs:label ?gene_name .
}
```

By reasoning over the axioms that state subproperties and inverse properties in the ontology, we can safely expect to get results for Q_3, although we did not introduce in the VKG system any specific mapping assertion for the property `obo:RO_0002292` used in the query. In this context, we exploit the axioms stating that `obo:RO_0002292` is the inverse property of `obo:RO_0002206`, and that `genex:isExpressedIn` from Genex is a subproperty of the latter. Similarly to what happened for Q_2, the query rewriting algorithm enriches Q_3 with a query making use of `genex:isExpressedIn` instead of `obo:RO_0002292`, and since we introduced in the VKG system a mapping assertion for `genex:isExpressedIn`, we can obtain indeed answers using the data stored in the underlying DB.

5 Deployment

Once the VKG has been developed, we can also make it available to external users. To do so, we can follow two approaches: *(1)* materialize the RDF triples into a file, which can then be uploaded to a file server, and downloaded by other users; *(2)* set up a SPARQL endpoint so that users can query it. We now discuss these two options.

5.1 Materialization

For VKGs over a data set of small size, one can use the *Ontop* Plugin for Protégé to materialize the triples that make up the KG. Otherwise, it is recommended to use the Command Line Interface (CLI) of *Ontop*. To do so, one can download *Ontop* CLI, unzip it, and invoke *Ontop* passing it the '`materialize`' directive. For example, to use the files provided in the online GitHub repository accompanying this tutorial, one can issue the following command[26]:

```
ontop materialize \
  --ontology=bgee_v14_genex.owl \
  --mapping=bgee_v14_genex.obda \
  --properties=bgee_v14_genex.properties \
  --output=bgee_v14_genex.ttl
```

[26] The `--ontology` and `--mapping` options are used to specify the files containing respectively an OWL 2 QL ontology and a set of mapping assertions (in the specific *Ontop* syntax), while the file supplied with the `--properties` option contains the connection parameter for the DB.

Then the triples are materialized into the file `bgee_v14_genex.ttl`. Such file can be shared and further analyzed, or it can be loaded into a triple store.

5.2 SPARQL Endpoint

Setting up a SPARQL Endpoint makes the VKG queryable as a standard HTTP service. This can be done either through a manual setup using the CLI of *Ontop*, or through a container-based deployment using Docker. We discuss now both options.

Using the CLI on *Ontop*. The following command starts the SPARQL endpoint and makes it available at URL `http://localhost:8080/sparql`.

```
ontop endpoint \
  --ontology=bgee_v14_genex.owl \
  --mapping=bgee_v14_genex.obda \
  --properties=bgee_v14_genex.properties \
  --portal=bgee_v14_genex.toml
```

The SPARQL endpoint may be accessed using any HTTP client, including SPARQL clients and tools using the standard SPARQL HTTP protocol. For instance, using `curl`[27]:

```
curl --request POST \
  --url http://localhost:8880/sparql \
  --header 'accept: application/json' \
  --header \
 'content-type: application/sparql-query' \
  --data 'SELECT * { ?s ?p ?o } LIMIT 5'
```

The endpoint also comes with a handy Web interface at `http://localhost:8080`, where users can formulate SPARQL queries.

Using the *Ontop* Docker Image. We have also developed a Docker-based deployment (defined in the `docker-compose.yml` file), which consists of two services: one for the MySQL DB, and another for the *Ontop* SPARQL endpoint. With this setup, one avoids manually configuring the MySQL DB and installing Java and *Ontop*. Instead, the following single command starts the whole tutorial, with the same services as those offered by the CLI:

```
docker-compose up
```

6 Conclusion

In this tutorial we have learned how to set-up and exploit the Virtual Knowledge Graph (VKG) approach to access data stored in relational legacy systems, and to enrich such data with domain knowledge coming from different heterogeneous (biomedical) resources. Specifically, we have shown how the Gene Expression Ontology can be mapped to the EasyBgee database so as to expose its content as a Knowledge Graph, and how to query such Knowledge Graph by means of the VKG system *Ontop*. All the software artifacts presented in this tutorial (ontology, mappings, data, etc.) are made available through the online repository[28].

[27] `https://curl.se/`

[28] `https://github.com/ontop/ontop-patterns-tutorial/`

Acknowledgements

This research has been partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by the Italian Basic Research (PRIN) project HOPE, by the EU H2020 project INODE, grant agreement 863410, by the CHIST-ERA project PACMEL, by the project IDEE (FESR1133) funded by the Eur. Reg. Development Fund (ERDF) Investment for Growth and Jobs Programme 2014-2020, and by the Free University of Bozen-Bolzano through the projects KGID, GeoVKG, and STyLoLa.

Author Contributions

The authors of the present work equally contributed to the writing and preparation of the tutorial.

Author Biographies

Diego Calvanese is a Full Professor at the Research Centre for Knowledge and Data (KRDB) at the Faculty of Computer Science of the Free University of Bozen-Bolzano (Italy), and Wallenberg Guest Professor in AI for Data Management at Umeå University (Sweden). His research interests include knowledge representation and reasoning, virtual knowledge graphs, ontology languages, description logics, conceptual data modeling and data integration. He is one of the editors of the Description Logic Handbook. He is a Fellow of EurAI and a Fellow of ACM. He is the originator and a co-founder of Ontopic, a startup whose mission is to bring the VKG technology to industry.

Davide Lanti is an Assistant Professor at the Research Centre for Knowledge and Data (KRDB) at the Faculty of Computer Science of the Free University of Bozen-Bolzano (Italy), where he carries out research on Virtual Knowledge Graphs, Semantic Web, Databases, and Description Logics. He received his MSc degree in Computational Logic jointly from the Technische Universität Dresden (Germany), and the Free University of Bozen-Bolzano (Italy). He received his PhD at the Faculty of Computer Science at the Free University of Bozen-Bolzano, Italy.

Tarcisio Mendes De Farias is a computer scientist specialized in data and knowledge management, currently working as a computational biologist in the SIB Swiss Institute of Bioinformatics. He obtained an MSc in Information and Communication Technology from the University of Technology of Compiègne in France, and a PhD in computer science from the University of Burgundy in collaboration with the ACTIVE3D – Sopra Steria company that was his employer. He also worked as a R&D engineer at Dassault Systèmes. He did a post-doc at the University of Lausanne for biological data integration and interoperability. He published more than 20 peer-reviewed articles.

Alessandro Mosca is Assistant Professor at the Research Centre for Knowledge and Data (KRDB) and member of the Smart Data Factory, the technology and knowledge transfer lab of the Faculty of Computer Science, Free University of Bozen-Bolzano (Italy). His research interests include logic-based formalisms for knowledge representation and conceptual modelling for data management. He works on the theoretical and methodological aspects behind the creation of ontology-based data management solutions which, in particular, subsume the design and development of formal ontologies, multi-format data integration and access services, efficient ontology-based query answering.

Guohui Xiao is an Assistant Professor at the Research Centre for Knowledge and Data (KRDB) at the Faculty of Computer Science of the Free University of Bozen-Bolzano (Italy). He received his BSc and MSc degrees from Peking University (China), and his PhD degree from Vienna University of Technology (Austria). His main research interests include knowledge representation, Semantic Web, database theory, and virtual knowledge graphs (VKG). He has authored more than 100 papers in these areas. He is the lead of the *Ontop* VKG platform. He is a co-founder and Chief Scientist of Ontopic, a startup whose mission is to bring the VKG technology to industry.

References

[1] Julián Arenas-Guerrero, Mario Scrocca, Ana Iglesias-Molina, Jhon Toledo, Luis Pozo-Gilo, Daniel Dona, Oscar Corcho, and David Chaves-Fraga. Knowledge Graph construction with R2RML and RML: An ETL system-based overview. In *Proc. of the 2nd Int. Workshop on Knowledge Graph Construction (KGCW)*, volume 2873 of *CEUR Workshop Proceedings*, `http://ceur-ws.org/`. CEUR-WS.org, 2021.

[2] Jie Bao et al. OWL 2 Web Ontology Language document overview (second edition). W3C Recommendation, World Wide Web Consortium, December 2012. Available at `http://www.w3.org/TR/owl2-overview/`.

[3] Frederic B. Bastian, Julien Roux, Anne Niknejad, Aurélie Comte, Sara S. Fonseca Costa, Tarcisio Mendes de Farias, Sébastien Moretti, Gilles Parmentier, Valentine Rech de Laval, Marta Rosikiewicz, Julien Wollbrett, Amina Echchiki, Angélique Escoriza, Walid H. Gharib, Mar Gonzales-Porta, Yohan Jarosz, Balazs Laurenczy, Philippe Moret, Emilie Person, Patrick Roelli, Komal Sanjeev, Mathieu Seppey, and Marc Robinson-Rechavi. The Bgee suite: Integrated curated expression atlas and comparative transcriptomics in animals. *Nucleic Acids Research*, 49(D1):D831–D847, 10 2020.

[4] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web J.*, 8(3):471–487, 2017.

[5] Diego Calvanese, Avigdor Gal, Naor Haba, Davide Lanti, Marco Montali, Alessandro Mosca, and Roee Shraga. ADaMaP: Automatic alignment of data sources using mapping patterns. In *Proc. of the 33rd Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, volume 12751 of *Lecture Notes in Computer Science*, pages 193–209. Springer, 2021.

[6] David Chaves-Fraga, Freddy Priyatna, Andrea Cimmino, Jhon Toledo, Edna Ruckhaus, and Oscar Corcho. GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain. *J. of Web Semantics*, 65:100596, 2020.

[7] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 concepts and abstract syntax. W3C Recommendation, World Wide Web

Consortium, February 2014. Available at http://www.w3.org/TR/rdf11-concepts/.

[8] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF mapping language. W3C Recommendation, World Wide Web Consortium, September 2012. Available at http://www.w3.org/TR/r2rml/.

[9] Tarcisio Mendes de Farias, Hirokazu Chiba, and Jesualdo Tomás Fernández-Breis. Leveraging logical rules for efficacious representation of large orthology datasets. In *Proc. of the 10th Int. Conf. on Semantic Web Applications and Tools for Health Care and Life Sciences (SWAT4LS)*, volume 2042 of *CEUR Workshop Proceedings,* http://ceur-ws.org/. CEUR-WS.org, 2017.

[10] Steve Harris and Andy Seaborne. SPARQL 1.1 query language. W3C Recommendation, World Wide Web Consortium, March 2013. Available at http://www.w3.org/TR/sparql11-query.

[11] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. CoRR Technical Report arXiv:2003.02320, arXiv.org e-Print archive, 2020. Available at http://arxiv.org/abs/2003.02320.

[12] Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G. Skjæveland, Evgenij Thorstensen, and José Mora. BootOX: Practical mapping of RDBs to OWL 2. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC)*, volume 9367 of *Lecture Notes in Computer Science*, pages 113–132. Springer, 2015.

[13] Hiroaki Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002.

[14] H. Krämer, R.L. Cagan, and S.L. Zipursky. Interaction of bride of sevenless membrane-bound ligand and the sevenless tyrosine-kinase receptor. *Nature*, 352(6332):207–212, July 1991.

[15] James Malone, Ele Holloway, Tomasz Adamusiak, Misha Kapushesky, Jie Zheng, Nikolay Kolesnikov, Anna Zhukova, Alvis Brazma, and Helen Parkinson. Modeling sample variables with an experimental factor ontology. *Bioinformatics*, 26(8):1112–1118, 03 2010.

[16] Manuel Namici and Giuseppe De Giacomo. Comparing query answering in OBDA tools over W3C-compliant specifications. In *Proc. of the 31st Int. Workshop on Description Logics (DL)*, volume 2211 of *CEUR Workshop Proceedings,* http://ceur-ws.org/. CEUR-WS.org, 2018.

[17] Christoph Pinkel, Carsten Binnig, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Wolfgang May, Andriy Nikolov, Ana Sasa, Martin G. Skjæveland, Alessandro Solimando, Mohsen Taheriyan, Christian Heupel, and Ian Horrocks. RODI: Benchmarking relational-to-ontology mapping generation quality. *Semantic Web J.*, 9:25–52, 2016.

[18] Juan F. Sequeda and Daniel P. Miranker. Ultrawrap Mapper: A semi-automatic relational database to RDF (RDB2RDF) mapping tool. In Serena Villata, Jeff Z. Pan, and Mauro Dragoni, editors, *Proc. of the 14th Int. Semantic Web Conf., Posters & Demonstrations Track (ISWC)*, volume 1486 of *CEUR Workshop Proceedings,* http://ceur-ws.org/. CEUR-WS.org, 2015.

[19] Ana Claudia Sima, Tarcisio Mendes de Farias, Erich Zbinden, Maria Anisimova, Manuel Gil, Heinz Stockinger, Kurt Stockinger, Marc Robinson-Rechavi, and Christophe Dessimoz. Enabling semantic queries across federated bioinformatics databases. *Database*, 2019, 11 2019. baz106.

[20] Barry Smith, Werner Ceusters, Bert Klagges, Jacob Köhler, Anand Kumar, Jane Lomax, Chris Mungall, Fabian Neuhaus, Alan Rector, and Cornelius Rosse. Relations in biomedical ontologies. *Genome biology*, 6:R46, 02 2005.

[21] Iman Tavassoly, Joseph Goldfarb, and Ravi Iyengar. Systems biology primer: the basic methods and approaches. *Essays in Biochemistry*, 62(4):487–500, 2018.

[22] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-based data access: A survey. In *Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 5511–5519. IJCAI Org., 2018.

[23] Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego Calvanese. Virtual Knowledge Graphs: An overview of systems and use cases. *Data Intelligence*, 1(3):201–223, 2019.

[24] Guohui Xiao, Davide Lanti, Roman Kontchakov, Sarah Komla-Ebri, Elem Güzel-Kalayci, Linfang Ding, Julien Corman, Benjamin Cogrel, Diego Calvanese, and Elena Botoeva. The virtual knowledge graph system Ontop. In *Proc. of the 19th Int. Semantic Web Conf. (ISWC)*, volume 12507 of *Lecture Notes in Computer Science*, pages 259–277. Springer, 2020.

7 Figure Titles and Legends

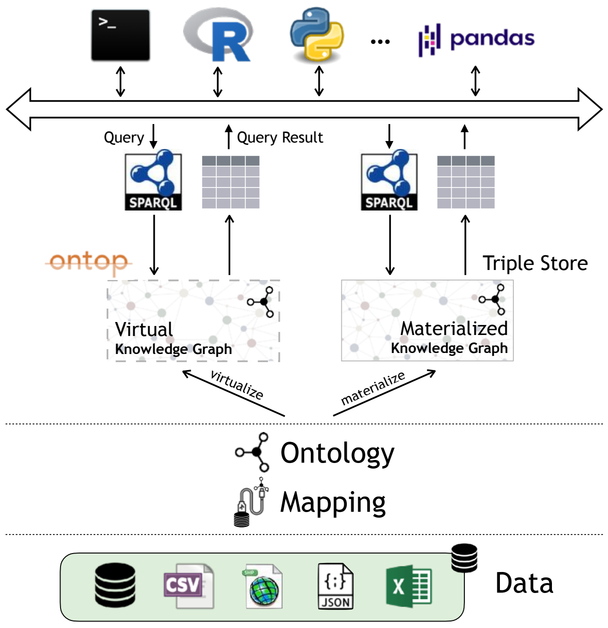


Figure 1: KG Conceptual Framework.

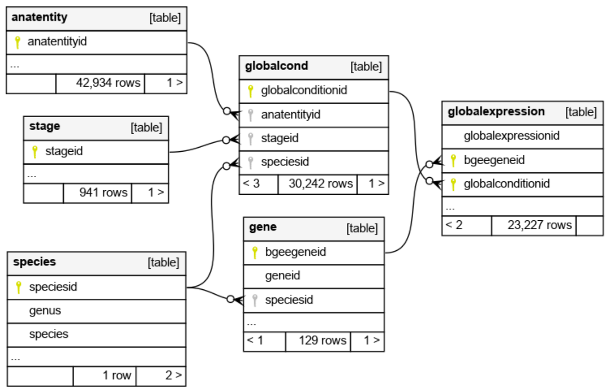
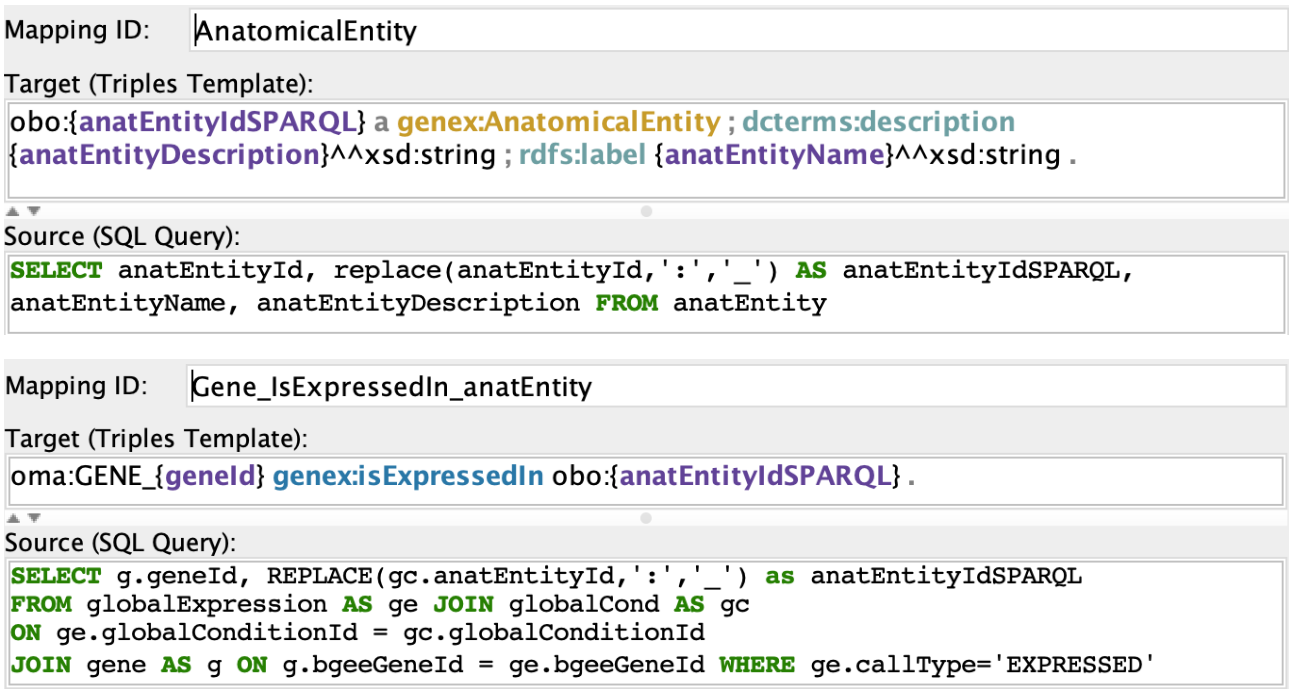


Figure 2: EasyBgee data schema (portion).


Mapping ID: AnatomicalEntity

Target (Triples Template):

```
obo:{anatEntityIdSPARQL} a genex:AnatomicalEntity ; dcterms:description
{anatEntityDescription}^^xsd:string ; rdfs:label {anatEntityName}^^xsd:string .
```

Source (SQL Query):

```
SELECT anatEntityId, replace(anatEntityId,':','_') AS anatEntityIdSPARQL,
anatEntityName, anatEntityDescription FROM anatEntity
```

Mapping ID: Gene_IsExpressedIn_anatEntity

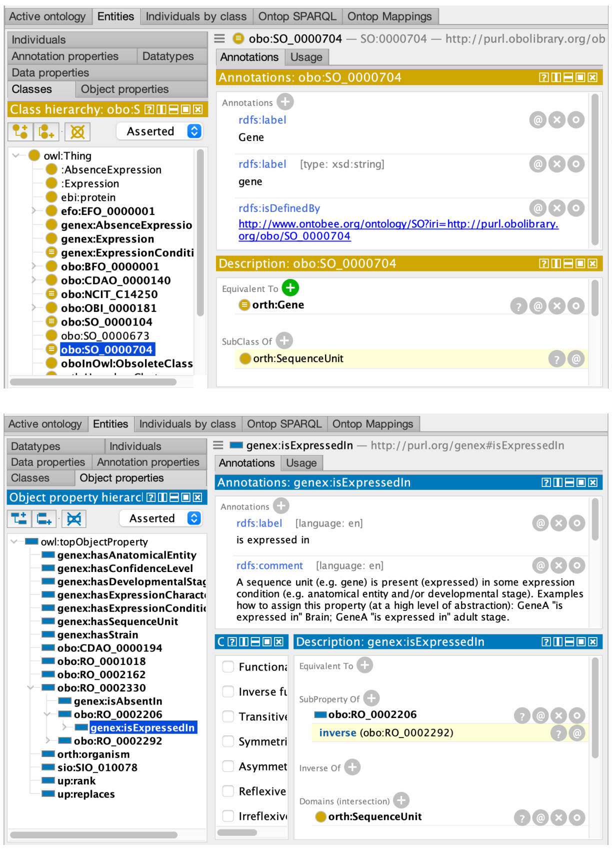
Target (Triples Template):

```
oma:GENE_{geneId} genex:isExpressedIn obo:{anatEntityIdSPARQL} .
```

Source (SQL Query):

```
SELECT g.geneId, REPLACE(gc.anatEntityId,':','_') as anatEntityIdSPARQL
FROM globalExpression AS ge JOIN globalCond AS gc
ON ge.globalConditionId = gc.globalConditionId
JOIN gene AS g ON g.bgeeGeneId = ge.bgeeGeneId WHERE ge.callType='EXPRESSED'
```

Figure 3: Examples of *Ontop* mapping assertions from the EasyBgee relational DB to a KG based on the Gene Expression Ontology.



rdfs:label
Gene

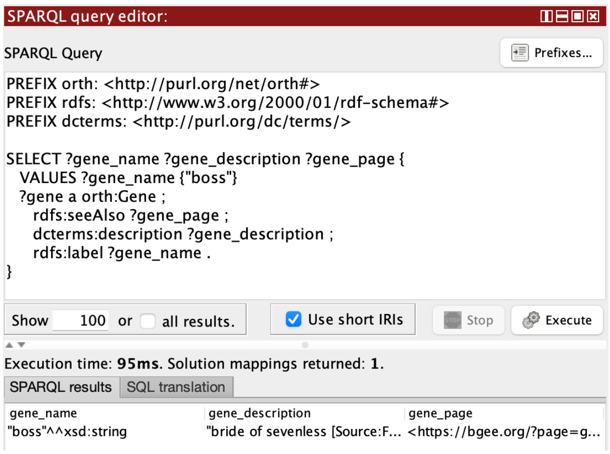
rdfs:label [type: xsd:string]
gene

rdfs:isDefinedBy
http://www.ontobee.org/ontology/SO?iri=http://purl.obolibrary.org/obo/SO_0000704

rdfs:label [language: en]
is expressed in

rdfs:comment [language: en]
A sequence unit (e.g. gene) is present (expressed) in some expression condition (e.g. anatomical entity and/or developmental stage). Examples how to assign this property (at a high level of abstraction): GeneA "is expressed in" Brain; GeneA "is expressed in" adult stage.

Figure 4: The top part of the figure shows the *class view* of Protégé. The selected class `obo:SO_0000704` has been declared as equivalent to the class `orth:Gene`. The bottom part of the figure shows the *property view* of Protégé. The selected property `genex:isExpressedIn` has been declared as sub-property of `obo:RO_0002206` and inferred as sub-property of the inverse of the `obo:RO_0002292` property. This inference occurs because `obo:RO_0002206` is explicitly defined as the inverse of `obo:RO_0002292`.



```
PREFIX orth: <http://purl.org/net/orth#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dcterms: <http://purl.org/dc/terms/>

SELECT ?gene_name ?gene_description ?gene_page {
  VALUES ?gene_name {"boss"}
  ?gene a orth:Gene ;
    rdfs:seeAlso ?gene_page ;
    dcterms:description ?gene_description ;
    rdfs:label ?gene_name .
}
```

Execution time: **95ms**. Solution mappings returned: **1**.

gene_name	gene_description	gene_page
"boss"^^xsd:string	"bride of sevenless [Source:F...	<https://bgee.org/?page=g...

Figure 5: SPARQL query interface of the *Ontop* Plugin for Protégé. Query 1 and the result of its execution are shown.

8 Table Titles and Legends

Table 1: The namespace prefixes used in this tutorial ("IRI" stands for "Internationalized Resource Identifier").

Prefix	**Namespace IRI**
`rdf:`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
`rdfs:`	`http://www.w3.org/2000/01/rdf-schema#`
`owl:`	`http://www.w3.org/2002/07/owl#`
`xsd:`	`http://www.w3.org/2001/XMLSchema#`
`orth:`	`http://purl.org/net/orth#`
`up:`	`http://purl.uniprot.org/core/`
`obo:`	`http://purl.obolibrary.org/obo/`
`dcterms:`	`http://purl.org/dc/terms/`
`genex:`	`http://purl.org/genex#`
`oma:`	`http://omabrowser.org/ontology/oma#`