

Tutorial

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 and Guohui Xiao^{1,3}¹Faculty of Computer Science, Free University of Bozen-Bolzano, 39100 Bolzano, Italy²Department of Computing Science, Umeå University, 901 87 Umeå, Sweden³Ontopic S.R.L., 39100 Bolzano, Italy⁴SIB Swiss Institute of Bioinformatics, 1015 Lausanne, Switzerland⁵Department of Ecology and Evolution, University of Lausanne, 1015 Lausanne, Switzerland*Correspondence: calvanese@inf.unibz.it<https://doi.org/10.1016/j.patter.2021.100346>

THE BIGGER PICTURE Knowledge graphs (KGs) have recently gained attention due to their flexible data model, which reduces the effort needed for integration across different, possibly heterogeneous, data sources. In this tutorial, we learn how to access scientific data stored in a relational database through the virtual knowledge graph (VKG) approach. In such an approach, the data are exposed as a KG and enriched with semantic information coming from a domain *ontology*. The KG is “virtual” in the sense that the data are not replicated but stay within the data sources and are accessed at query time.

We demonstrate the approach over scientific data coming from the biomedical domain and using the open-source VKG system *Ontop*. Since legacy data are exposed as a KG, users can access the data by means of a more convenient vocabulary provided by the domain ontology, benefit from automated reasoning capabilities, and do not need to focus on how the data are 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 them.

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.



Production: Data science output is validated, understood, and regularly used for multiple domains/platforms

SUMMARY

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.

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 generally, the advent of a systems approach for the analysis and modeling of complex biological systems,^{2,3} 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 recently⁴ as a general mechanism to represent data that are 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 platforms.



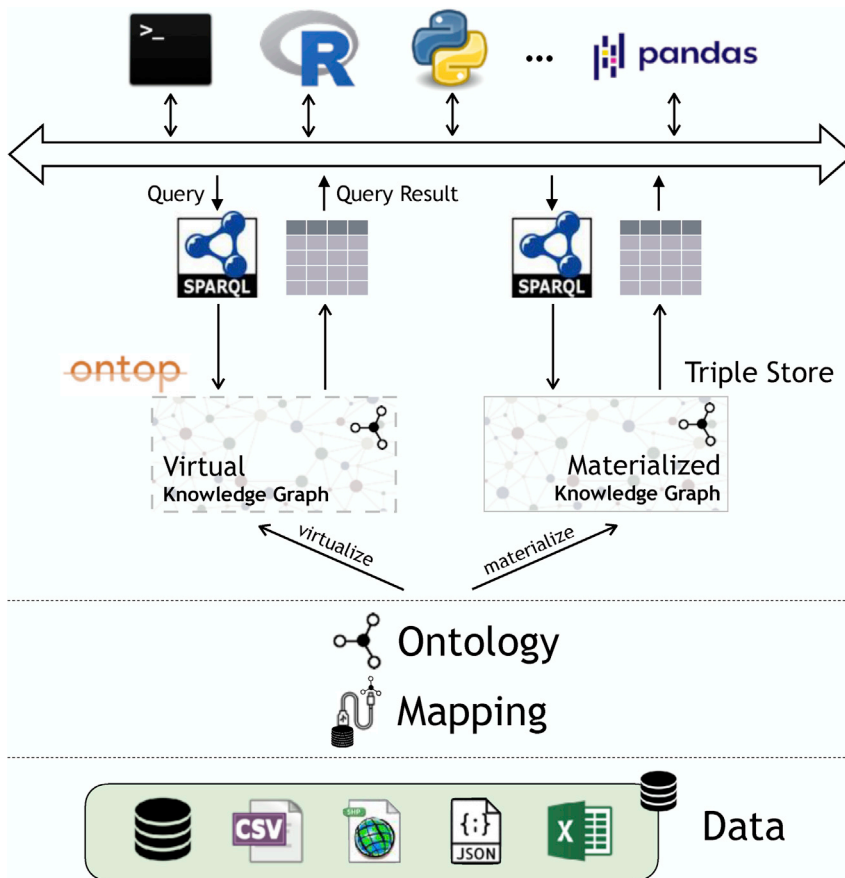


Figure 1. KG conceptual framework

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

- interoperability among different scientific *data platforms* (e.g., GDC⁸ and ELIXIR,⁹ *resources* (e.g., UBERON¹⁰ and CHEBI,¹¹ and *data models* (e.g., SBML¹² and BioPAX¹³);
- 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;
- enrichment of the information originally present in the data sources through the application of reasoning techniques that combine domain knowledge and data assertions.

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 2-fold. 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 of 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,⁵ Dremio,⁶ or Teiid,⁷ which expose such sources as if they were part of a single relational DB.

In a virtual knowledge graph (VKG),^{14,15} 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, while 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 KG in RDF,¹⁶ the ontology in OWL 2 QL,¹⁷ the mapping in R2RML,¹⁸ and the query in SPARQL.¹⁹

In this tutorial we make use of the VKG system *Ontop*^{20,21} to set up a KG in the biomedical domain, specifically in the area of cancer research.

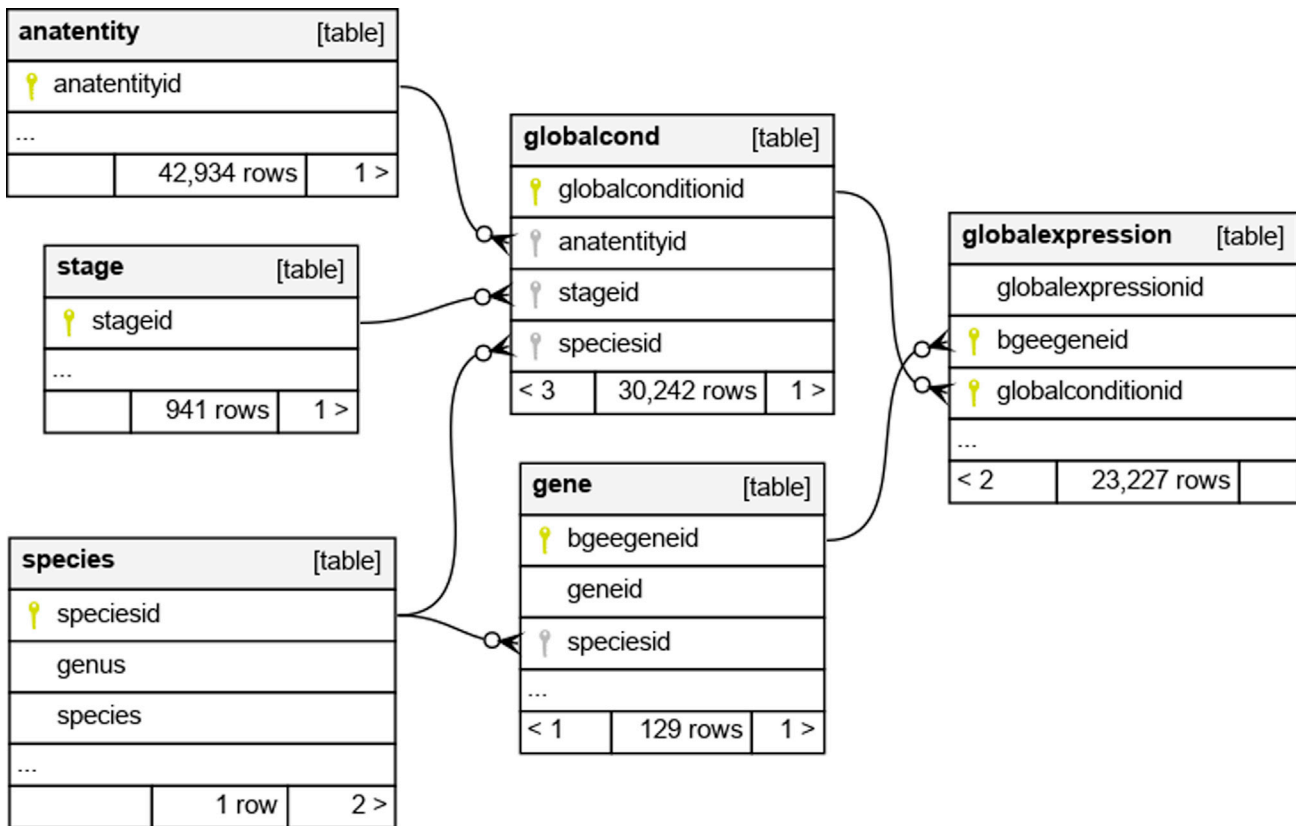


Figure 2. EasyBgee data schema (portion)

We observe that *Ontop* has been conceived as a VKG system, but it offers also functionalities for materializing a KG from a VKG specification consisting of an ontology, a relational data source, and a mapping between the two.

All the material used in this tutorial, as well as full details for its usage, are available in an online GitHub repository.²²

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 5 years, according to Sourceforge). *Ontop* is a state-of-the-art VKG system initially developed by the Research Center 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 (most notably, the two European projects FP7 Optique²³ and H2020 INODE²⁴), and it also has a number of commercial deployments, such as the UNICS open data platform by SIRIS Academic²⁵ (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 as-

pects, we refer the interested reader to the vast scientific literature.^{15,26–29}

THE EasyBgee DATASET

Gene expression is a key process to understand the relations 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 DB³⁰ is a public relational DB that consolidates and curates heterogeneous gene expression data sources.³¹

Figure 2 illustrates the data schema of the EasyBgee DB (which is available as a MySQL dump^{32,33}), 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 serialized 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:

Table 1. The namespace prefixes used in this tutorial

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#

IRI, internationalized resource identifier.

- the `anatomy` 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.

SETTING UP A 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 bioinformatics.

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),³⁴ 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).³⁵ As an example, GenEx imports from RO the “expressed in” property (actually identified by `obo:RO_0002206`) and its inverse property “expresses” (identified by `obo:RO_0002292`). 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 the bottom part of Figure 4). The domain and range of `genex:isExpressedIn` include `orth:Gene` and `genex:`

`AnatomicalEntity`, respectively. Therefore, we can assert that a gene is expressed in an anatomical entity (e.g., the body of the pancreas) using the `genex:isExpressedIn` property and automatically infer a more general statement about its RO super-property. 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 identifiers (called URIs, in RDF terminology). 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.³⁶ The classes of the species-specific developmental stage ontologies³⁷ are also considered as instances of the `efo:EFO_0000399` class, which has been imported from the experimental factor ontology (EFO).³⁸ 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` (labeled 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 allows 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.³⁴

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, to populate GenEx using a system, such as *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 ‘}’.

```
obo:{anatomyId$SPARQL} a
```

Mapping ID:	AnatomicalEntity
Target (Triples Template):	obo:{ <i>anatEntityId</i> SPARQL} a <i>genex:AnatomicalEntity</i> ; <i>dcterms:description</i> { <i>anatEntityDescription</i> }^^xsd:string ; <i>rdfs:label</i> { <i>anatEntityName</i> }^^xsd:string .
Source (SQL Query):	<pre>SELECT <i>anatEntityId</i>, replace(<i>anatEntityId</i>,':','_') AS <i>anatEntityId</i>SPARQL, <i>anatEntityName</i>, <i>anatEntityDescription</i> FROM <i>anatEntity</i></pre>
Mapping ID:	Gene_IsExpressedIn_anatEntity
Target (Triples Template):	oma:GENE_{ <i>geneId</i> } <i>genex:isExpressedIn</i> obo:{ <i>anatEntityId</i> SPARQL} .
Source (SQL Query):	<pre>SELECT <i>g.geneId</i>, REPLACE(<i>gc.anatEntityId</i>,':','_') as <i>anatEntityId</i>SPARQL FROM <i>globalExpression</i> AS <i>ge</i> JOIN <i>globalCond</i> AS <i>gc</i> ON <i>ge.globalConditionId</i> = <i>gc.globalConditionId</i> JOIN <i>gene</i> AS <i>g</i> ON <i>g.bgeeGeneId</i> = <i>ge.bgeeGeneId</i> WHERE <i>ge.callType</i>='EXPRESSED'</pre>

Figure 3. Examples of *Ontop* mapping assertions from the EasyBgee relational DB to a KG based on gene expression ontology

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

The meaning of such a mapping assertion is that it constructs a portion of the 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 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 an 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).

Notice that in Figure 3 we have used the standard abbreviated notation for RDF triple 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 labeled 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 a value for the property `up:scientificName` of a species from the UniProt core ontology,³⁹ 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 the section "The gene expression ontology" (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`³⁶ statements. For example, GenEx states that `orth:Gene` from



Figure 4. Class view and property view of Protégé

The top part of the figure shows the *class view* of Protégé. The selected class `obo:SO_0000704` has been declared to be as equivalent to the class `ortho:Gene`. The bottom part of the figure shows the *property view* of Protégé. The selected property `genex:isExpressedIn` has been declared as a sub-property of `obo:RO_0002206` and inferred as a 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`.

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 modeling 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^{41,42} and implemented systems.^{41,43}

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 for querying KGs.¹⁹ We rely on the *Ontop* plugin for the ontology editor Protégé,⁴⁴ 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.⁴⁵ 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

the orthology ontology (ORTH)⁴⁰ 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 `ortho:Gene` concept. As a result, we can interchangeably use `ortho:Gene` and `obo:SO_0000704`, and consider them as synonyms (see query 2 in “Querying the KG”).

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

SPARQL query editor: ⏏ ⏏ ⏏ ⏏

SPARQL Query
Prefixes...

```

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 .
}

```

Show or all results.
 Use short IRIs
 Stop
 Execute

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

SPARQL results

SQL translation

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.

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. 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 labeled 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.

```

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 .
}

```

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 sub-properties 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 sub-property 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.

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 an SPARQL endpoint so that users can query it. We now discuss these two options.

Materialization

For VKGs over a dataset 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 (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):

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

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

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`⁴⁶:

```
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
```

CONCLUSIONS

In this tutorial we have learned how to set up and exploit the 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 to expose its content as a KG, and how to query such a KG by means of the VKG system *Ontop*. All the software artifacts presented in this tutorial (ontology, mappings, data, etc.) are available through the online repository.²²

ACKNOWLEDGMENTS

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 European Regional Development Fund (ERDF) Investment for Growth and Jobs Program 2014-2020, and by the Free University of Bozen-Bolzano through the projects KGID, GeoVKG, and STyLoLa.

AUTHOR CONTRIBUTIONS

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

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

1. NIH. Biological pathways fact sheet. <https://www.genome.gov/about-genomics/fact-sheets/Biological-Pathways-Fact-Sheet>.
2. Tavassoly, I., Goldfarb, J., and Iyengar, R. (2018). Systems biology primer: the basic methods and approaches. *Essays Biochem.* 62, 487–500.
3. Kitano, H. (2002). Computational systems biology. *Nature* 420, 206–210.
4. Hogan, A., Blomqvist, E., Cochez, M., d’Amato, C., de Melo, G., Gutiérrez, C., Gayo, J.E.L., Kirrane, S., Neumaier, S., Polleres, A., et al. (2020). Knowledge graphs. arXiv, 2003.02320 <http://arxiv.org/abs/2003.02320>.
5. Denodo. The Denodo platform for data virtualization. <https://www.denodo.com>.
6. Dremio. The SQL Lakehouse Platform. <https://www.dremio.com/>.
7. Teiid. Teiid: Cloud-native data virtualization. <https://teiid.io/>.

8. NIH. Genomic data commons data portal. <https://portal.gdc.cancer.gov/>.
9. ELIXIR. ELIXIR: A distributed infrastructure for life-science information. <https://elixir-europe.org/>.
10. Uberon. The Uberon ontology. <https://uberon.github.io/>.
11. EMBL-EBI. Chemical Entities of Biological Interest. <https://www.ebi.ac.uk/chebi/>.
12. SBML. The Systems Biology Markup Language. <http://sbml.org/>.
13. BioPAX Workgroup. Biological Pathway Exchange. <http://www.biopax.org/>.
14. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., and Zakharyashev, M. (2018). Ontology-based data access: a survey. In Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI), J. Lang, ed. (AAAI Press.), pp. 5511–5519.
15. Xiao, G., Ding, L., Cogrel, B., and Calvanese, D. (2019). Virtual knowledge graphs: an overview of systems and use cases. *Data Intell.* 7, 201–223.
16. Cyganiak, R., Wood, D., and Lanthaler, M. (2014). RDF 1.1 Concepts and Abstract Syntax. <http://www.w3.org/TR/rdf11-concepts/>.
17. W3C OWL Working Group (2012). OWL 2 Web Ontology Language Document Overview (Second Edition). <http://www.w3.org/TR/owl2-overview/>.
18. Das, S., Sundara, S., and Cyganiak, R. (2012). R2RML: RDB to RDF Mapping Language. <http://www.w3.org/TR/r2rml/>.
19. Harris, S., and Seaborne, A. (2013). SPARQL 1.1 Query Language. <http://www.w3.org/TR/sparql11-query/>.
20. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., and Xiao, G. (2017). Ontop: answering SPARQL queries over relational databases. *Semantic Web J.* 8, 471–487.
21. Xiao, G., Lanti, D., Kontchakov, R., Komla-Ebri, S., Güzel-Kalayci, E., Ding, L., Corman, J., Cogrel, B., Calvanese, D., and Botoeva, E. (2020). The Virtual Knowledge Graph System Ontop. In Proc. of the 19th Int. Semantic Web Conf. (ISWC), 12507 (Springer), pp. 259–277.
22. Xiao, G.. Ontop tutorial repository. <https://github.com/ontop/ontop-patterns-tutorial/>.
23. Optique. Optique: scalable end-user access to big data. <http://optique-project.eu/>.
24. INODE. INODE: intelligent open data exploration. <http://www.inode-project.eu/>.
25. SIRIS. SIRIS academic. <https://www.sirisacademic.com/>.
26. Pinkel, C., Binnig, C., Jiménez-Ruiz, E., Kharlamov, E., May, W., Nikolov, A., Sasa, A., Skjæveland, M.G., Solimando, A., Taheriyani, M., et al. (2016). RODI: benchmarking relational-to-ontology mapping generation quality. *Semantic Web J.* 9, 25–52.
27. Namici, M., and De Giacomo, G. (2018). 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 (CEUR-WS) <http://ceur-ws.org/Vol-2211/paper-25.pdf>.
28. Chaves-Fraga, D., Priyatna, F., Cimmino, A., Toledo, J., Ruckhaus, E., and Corcho, O. (2020). GTFS-Madrid-Bench: a benchmark for virtual knowledge graph access in the transport domain. *J. Web Semantics* 65, 100596.
29. Arenas-Guerrero, J., Scrocca, M., Iglesias-Molina, A., Toledo, J., Pozo-Gilo, L., Dona, D., et al. (2021). 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 (CEUR-WS) <http://ceur-ws.org/Vol-2873/paper11.pdf>.
30. Swiss Institute of Bioinformatics. Bgee: gene expressions data in animals. <https://bgee.org>.
31. Bastian F.B., Roux J., Niknejad A., Comte A., Fonseca Costa S.S., de Farias T.M., Moretti S., Parmentier G., de Laval V.R., Rosikiewicz M., et al. The Bgee suite: integrated curated expression atlas and comparative transcriptomics in animals. *Nucleic Acids Res.*, 49:D831–D847.
32. EasyBgee data dump. ftp://ftp.bgee.org/current/easybgee_dump.tar.gz.
33. Sima, A.C., Mendes de Farias, T., Zbinden, E., Anisimova, M., Gil, M., Stockinger, H., Stockinger, K., Robinson-Rechavi, M., and Dessimoz, C. (2019). Enabling semantic queries across federated bioinformatics databases. *Database (Oxford)* 2019, baz106.
34. Mendes de Farias, T., Bastian, F., Wollbrett, J., Dessimoz, C., and Robinson-Rechavi, M.. Gene expression ontology. <https://biosoda.github.io/genex/>.
35. Smith, B., Ceusters, W., Klagges, B., Köhler, J., Kumar, A., Lomax, J., Mungall, C., Neuhaus, F., Rector, A., and Rosse, C. (2005). Relations in biomedical ontologies. *Genome Biol.* 6, R46.
36. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., and Rudolph, S. (2012). OWL 2 Web Ontology Language: Primer (Second Edition). <http://www.w3.org/TR/owl2-primer/>.
37. Mungall, C.. Developmental stage ontologies repository. <https://github.com/obophenotype/developmental-stage-ontologies/wiki>.
38. Malone, J., Holloway, E., Adamusiak, T., Kapushesky, M., Zheng, J., Kolesnikov, N., Zhukova, A., Brazma, A., and Parkinson, H. (2010). Modeling sample variables with an experimental factor ontology. *Bioinformatics* 26, 1112–1118.
39. UniProt Consortium. UniProt RDF schema ontology. <https://www.uniprot.org/core/>.
40. Mendes de Farias, T., Chiba, H., and Fernández-Breis, J.T. (2017). 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 (CEUR-WS) <http://ceur-ws.org/Vol-2042/paper36.pdf>.
41. Jiménez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I., Pinkel, C., Skjæveland, M.G., Thorstensen, E., and Mora, J. (2015). BootOX: practical mapping of RDBs to OWL 2. *Lecture Notes in Computer Science* 9367, 113–132.
42. Calvanese, D., Gal, A., Haba, N., Lanti, D., Montali, M., Mosca, A., and Shraga, R. (2020). ADaMaP: automatic alignment of data sources using mapping patterns. *Lecture Notes in Computer Science* 12757, 193–209.
43. Sequeda, J.F., and Miranker, D.P. (2015). Ultrawrap Mapper: a semi-automatic relational database to RDF (RDB2RDF) mapping tool. In Proc. of the 14th Int. Semantic Web Conf., Posters & Demonstrations Track (ISWC), Volume 1486 of CEUR Workshop Proceedings, S. Villata, J.Z. Pan, and M. Dragoni, eds. (CEUR-WS) http://ceur-ws.org/Vol-1486/paper_105.pdf.
44. Xiao, G.. Ontop 4.1 download page. <https://sourceforge.net/projects/ontop4obda/files/ontop-4.1.0/>.
45. Krämer, H., Cagan, R.L., and Zipursky, S.L. (1991). Interaction of bride of sevenless membrane-bound ligand and the sevenless tyrosine-kinase receptor. *Nature* 352, 207–212.
46. curl. Command line tool and library. <https://curl.se/>.

About the Authors

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.

Daide 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, which was his employer. He also worked as an R&D engineer at Dassault Systèmes. He did a postdoc at the University of Lausanne for biological data integration and interoperability. He has 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 Univer-

sity of Bozen-Bolzano (Italy). His research interests include logic-based formalisms for knowledge representation and conceptual modeling 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, and 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 (VKGs). He has authored more than 100 papers in these areas. He is the lead of the *Ontop* VKG platform and a co-founder and chief scientist of Ontopic, a startup whose mission is to bring VKG technology to industry.