Second Component Release

Document Due Date: 28/02/2021
Document Submission Date: 28/02/2021

Work Packages 3, 4, 5, 6, 7, 8

Type: Other (Software)
Document Dissemination Level: Public

INODE
Intelligent Open Data Exploration
is funded by the Horizon 2020 Framework Programme of the EU for Research and Innovation.
Grant Agreement number: 863410— INODE — H2020-EU.1.4.1.3.

(This page has been intentionally left blank)

Executive Summary

This deliverable provides the second software component release of the INODE project and includes all 6 services covered by WPs 3 to 8. We assume that the reader is familiar with Deliverable D3.1-D8.1, which describes INODE 1.0, i.e. the status of INODE at project month 10. This deliverable describes the status and the new activities of INODE at project month 16.

- **Section 1** presents all **new INODE features**: new components that are added as well as how existing components are upgraded. It provides a summary of the main features and pointers to the respective sections that provide more details.
- **Section 2** and **Section 3** describe **INODE-SQL 2.0** and **INODE-SPARQL 1.0**, the two user-facing services for SQL and SPARQL data sources, respectively
- **Section 4** describes the **API extensions** of the INODE services and **Section 5** the **data model extensions** of the three use cases "Research & Innovation Policy Making" (CORDIS), "Astrophysics" (SDSS), and "Cancer Research" (OncoMX) with respect to INODE 1.0.

Project Information

Project Name	Intelligent Open Data Exploration
Project Acronym	INODE
Project Coordinator	Zurich University of Applied Sciences (ZHAW), CH
Project Funded by	European Commission
Under the Programme	H2020-EU.1.4.1.3. - Development, deployment and operation of ICT-based e-infrastructures
Call	H2020-INFRAEOSC-2019-1
Topic	INFRAEOSC-02-2019 - Prototyping new innovative services
Funding Instrument	Research and Innovation action
Grant Agreement No.	863410

Document Information

Document reference	D3.2
Document Title	Second Component Release
Work Package reference	WP3, WP4, WP5, WP6, WP7, WP8
Delivery due date	28/02/2021
Actual submission date	28/02/2021
Dissemination Level	Public
Authors(s)	Koutrika Georgia, Eleftheraki Stavroula, Glenis Apostolis, Mandamadiotis Antonis (ATHENA) Amer-Yahia Sihem, Patil Yogendra, Personnaz Aurélien (CNRS) Lücke-Tieke Hendrik, May Thorsten (Fraunhofer) Litke Antonis, Papadakis Nikolaos, Papadopoulos Dimitris (Infili) Fabricius Maximilian, Subramanian Srividya (MPE) Bastian Frederic, Mendes de Farias Tarcisio, Stockinger Heinz (SIB) Massucci Francesco, Multari Francesco, Rull Guillem (SIRIS) Calvanese Diego, Lanti Davide, Mosca Alesandro, Guohui Xiao (UNIBZ) Braschler Martin, Brunner Ursin, Kosten Catherine, Sima Ana, Smith Ellery, Stockinger Kurt (ZHAW)

Table of Contents

1 THE INODE SYSTEM

Our new version of INODE system has expanded in three directions:

1. **System components:** Existing components have been enhanced and new components have been added. For example, for enabling natural language queries, we offer different text-to-SQL systems.
2. **Sources:** INODE enables natural language querying over sources that allow for SQL or SPARQL queries: **INODE-SQL 2.0** and **INODE-SPARQL 1.0**. More precisely, INODE-SQL 2.0 is the evolved version of what was offered in INODE 1.0, i.e. at project month 10, while INODE-SPARQL 1.0 is the new service for SPARQL-based data sources.
3. **Use cases:** Two of our three use cases are fully supported by all services. The third and most complex use case is supported by parts of the services without full integration of natural language capabilities.

Table 1.1 summarizes the progress in INODE. It shows which new components are added and which are upgraded. It provides a summary of the main features and pointers to the respective sections that provide more details.

System (Work Package)	**Summary of Features**	**Section**
Integrated Query Processing Services (WP3)	UPDATE	
Query Execution (Task 3.1)	• Better support for both SQL and SPARQL data types. • Support of geospatial data sources and of the GeoSPARQL query language.	1.1.1
Source Federation (Task 3.2)	• Support of different SQL federation engines for integrating multiple data sources.	1.1.2
Data Analytics (Task 3.3)	• Support of SPARQL aggregate functions for data analytics.	1.1.3
Answer Justification (Task 3.4)	• A prototype (not yet integrated in the main development branch of Ontop) relying on ProvSQL, a tool for provenance developed in the context of RDBMSs.	1.1.4

Data Linking and Modelling (WP4)		1.2
Mapping Construction (Tasks 4.1 and 4.2)	**MPBoot:** UPDATE • Improvements in the automated generation of ontologies from data sources. • Ability to encode in the ontology schema additional information available in the data source, such as taxonomic relationships. • Ability to exploit a given SQL query workload to derive semantic connections between classes and generate object properties.	1.2.1
Knowledge Base Construction (Task 4.3)	**Information Extraction:** UPDATE • Improvements in the process of extracting triples from NL text (open information extraction) for both engines and linking them to specific ontologies (entity linking). • Implementation of a unified extraction approach to efficiently consolidate the extracted triples from both engines. • Support for SQL queries targeting the distilled knowledge from information extraction processes.	1.2.2
Data Access & Exploration (WP5)		1.3
By Natural Language (Task 5.3)	**ValueNet**: NEW Natural language to SQL with a neural network-based transformer architecture approach. **Bio-SODA**: NEW Natural language to SPARQL with a graph-based approach.	1.3.1

	SODA: UPDATE Natural language to SQL (Adaptations for the SDSS dataset).	
By Example, By analytics (Tasks 5.1 and 5.2)	**By-neighbors.** NEW This operator searches the neighborhood of a set of items and returns close sets. **By-distribution.** NEW This operator searches the whole data space for sets whose value distributions are the same as the input set.	1.3.2
User Assistance (WP6)		1.4
Explanations (Task 6.1)	**Logos:** UPDATE ● Extended to translate the SQL queries produced by ValueNet, as well as the queries generated by the data exploration operators. ● Improvements in terms of query semantics. ● Improvements in terms of translation accuracy.	1.4.1
Recommendations (Task 6.2)	**PyExplore** NEW ● Recommends interesting queries for the user by leveraging data correlations and diversity. ● Handles datasets with mixed numeric and categorical attributes. ● The recommended queries have an augmented WHERE-clause if there was a WHERE-clause in the initial query, or a new WHERE-clause if there was no WHERE-clause.	1.4.2
Multi-Modal Discovery (WP7)	UPDATE	1.5
Visual Result Exploration (Task 7.1)	Visual exploration improved by increasing information density and providing a better overview over multiple search results.	1.5.1

Visual Query Manipulation (Task 7.2)	Interactions have been extended to reflect the arguments needed by the available exploration operators.	1.5.2
Integrated seamless query-response loop (Task 7.3)	Improved user experience and UI performance. Consolidated the code base (consolidation and streamlining of multi-table-explorer user experience) in preparation for OpenDataDialog 3.0.	1.5.3
Evaluation (WP8)	Logging NEW	1.6
	A new logging mechanism to record specific system information is designed and put into place, in order to perform evaluation of the INODE system.	

Table 1.1: Summary of developments.

SERVICES	**INODE-SQL 2.0**	**INODE-SPARQL 1.0**
Integrated Query Processing (WP3)		Ontop
Data Linking and Modelling (WP4)		
Mapping Construction		MPBoot
Knowledge Base Construction	Triple Extraction from NL-text	
Data Access & Exploration (WP5)		
- by NL	SODA, ValueNet	BioSODA
- by-example, by-analytics	Pipeline Operators: added one new instance of by-example (by-neighbors) and one new instance of by-analytics (by-distribution)	

User Assistance (WP6)		
- Explanations	Logos	
- Recommendations	PyExplore	
Multi-Modal Discovery (WP7)	Multi-Table Explorer	
Evaluation (WP8)	Logging	
Use Cases	CORDIS, SDSS, OncoMX	CORDIS, SDSS, OncoMX

Table 1.2: Summary of the INODE services provided by each work package that are part of INODE-SQL 2.0 (SQL-based) and INODE-SPARQL 1.0 (SPARQL-based).

An overview of the INODE system architecture with the major services is given in Figure 1.1. The main interface for users to interact with the INODE-SQL 2.0 part of the system in the present release is the OpenDataDialog 2.0 web application. This application acts as the integration layer for the services that are colored green in Figure 1.1.

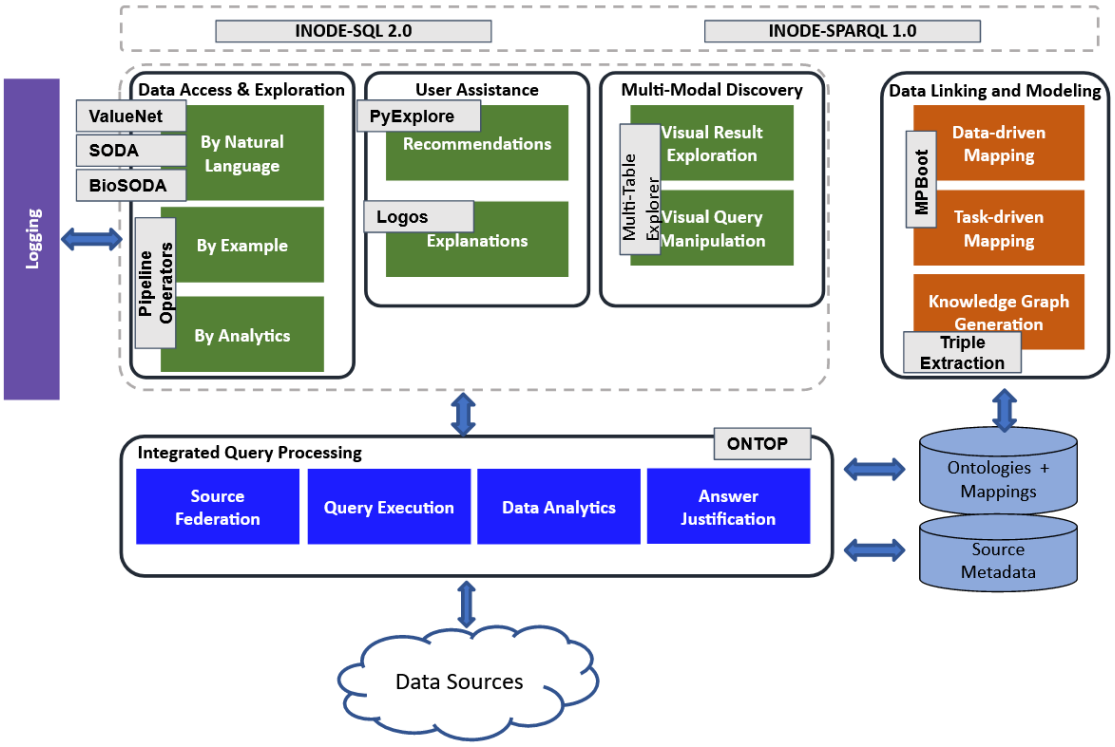


Figure 1.1: Services of INODE system architecture. The services shown in green refer to "OpenDataDialog", the services in orange to "OpenDataLinking" and the services in blue as

"Backend Services". INODE-SQL 2.0 is the user-facing service that allows access over SQL data sources and INODE-SPARQL 1.0 also access over RDF knowledge graphs.

1.1 Integrated Query Processing

Integrated query processing is the core service of OpenDataDialog in INODE-SPARQL 1.0, as it provides the SPARQL query answering capability used by almost all other components. Such a service relies on *Ontop*[1], a popular Virtual Knowledge Graph (VKG) system. A VKG is a virtual representation in the form of a graph of the information coming from multiple, possibly heterogeneous, data sources. Such representation relies on the RDF recommendation as the format to represent the data as a graph, and on OWL 2 QL[2] as the language to represent ontologies. Both are W3C standards for VKGs. The RDF graph is virtual in the sense that it does not contain data extracted from the data sources (which are physically stored somewhere, e.g. in a relational database), but rather each query over the VKG is translated by Ontop into a query over the original data sources. This is done through reformulation techniques well studied in the literature, as well as a number of optimizations performed by Ontop itself, so as to reduce to a minimum the overhead introduced by virtualization and translation. The link between the VKG and the data sources is realized through a *domain-specific ontology*, providing a vocabulary for SPARQL queries abstracting from storage details, and through a set of *mappings* relating elements in the ontology to queries over the data sources.

Ontop is not a prototype tool, but a complex, well-engineered, and well-established software artifact that relies on and interacts with several technologies. Most notably, Ontop supports all major open and commercial RDBMSs (e.g., PostgreSQL, Oracle, DB2, Microsoft SQL Server, etc.), and it supports all relevant W3C standards (RDF, RDFS, OWL 2 QL, R2RML, SPARQL, and GeoSPARQL). Ontop can be used in several different ways: as a SPARQL endpoint to query through HTTP, as an API to enrich other Java applications, or simply as a shell tool.

Progress within INODE. To cope with the challenges brought by INODE, Ontop has been improved in several aspects. The design of the new Ontop v4 release has been described in the paper *the Virtual Knowledge Graph System Ontop*[3] published at ISWC 2020. Below we highlight some improvements with respect to the tasks.

1.1.1 Query execution over rich types of data sources

We have redesigned the typing system of Ontop and implemented better support for both *SQL and SPARQL datatypes*. Proper handling of datatypes in SPARQL (e.g., xsd:integer, xsd:string, xsd:time) and in SQL (e.g., INT, VARCHAR, DECIMAL) is crucial in VKG, but it is also rather challenging. The reason is that SQL is *statically typed* in the sense that all values in a given relation column (both in the database and in the result of a query) have the same datatype. In contrast, SPARQL is *dynamically typed*: a variable can have values of different

[1] https://ontop-vkg.org/
[2] https://www.w3.org/TR/owl2-profiles/#OWL_2_QL
[3] 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 *International Semantic Web Conference (ISWC 2020)*, volume 2, pages 259–277, 2020.

datatypes in different solution mappings. Also, the output datatype of a SPARQL function depends on the types or language tags of its arguments (e.g., if both arguments of '+' are xsd:integer, then so is the output, and if both arguments are xsd:decimal, then so is the output). In particular, to determine the output datatype of an aggregate function in SPARQL, one has to look at the datatypes of values in the group, which can vary from one group to another. In Ontop v4, we have *redesigned the typing system*, which supports all standard SQL datatypes of a database, and SPARQL datatypes in queries. Moreover, Ontop is able to perform *type inference* in SPARQL normal functions and SPARQL aggregate functions.

We have implemented support of geospatial data sources and of the GeoSPARQL query language. Supporting geospatial data sources is important in many use cases e.g., oil exploration[4], maritime security[5], and LinkedGeoData[6]. In the past, we have contributed to the development of a prototype system called Ontop-spatial[7] as a fork of Ontop, which supports a limited fragment of GeoSPARQL. However, this fork is not maintained anymore, and its functionality did not catch up with the latest developments of Ontop. To address this issue, we have reimplemented GeoSPARQL support in Ontop v4. With respect to the old implementation, it has significantly improved the compliance with the Open GeoSPARQL Consortium (OGC) GeoSPARQL standard. All of the geospatial functions defined in GeoSPARQL are implemented. In particular, it features improved handling of units (such as degrees and meters) and different spatial reference systems (SRIDs). We have tested it over PostgreSQL/PostGIS and H2/H2GIS, and it should work with all relational database systems that are compliant with the OGC Implementation Standard Simple Feature Access.

All the activities described in the paragraphs above comply with the goals of Task 3.1.

1.1.2 Source Federation

We have implemented support of SQL federation engines for integrating multiple data sources. In Ontop v4, we support popular federation engines, namely Denodo, Dremio, and Teiid. These engines have different features and licenses, and can be applied in different scenarios. For each federation engine, we needed to implement specific adaptors, which include metadata extraction (e.g., table definitions and constraints) and SQL dialect translators. Moreover, extensive tests were needed to make sure Ontop interacts with these federation engines correctly and efficiently. Federation is an essential capability that will bring several benefits to our use cases. For instance, with respect to the CORDIS use case, it would be interesting to integrate the data in the CORDIS portal with the information from the European Patent Office (EPO[8]), allowing questions such as retrieving the number of EU patent owners who have also played the role of Principal Investigator in an EU project in the 5 years preceding the patent registration. All these activities comply with the goals of Task 3.2.

---

[4] Evgeny Kharlamov, Dag Hovland, Martin G. Skjæveland, Dimitris Bilidas, Ernesto Jim´enez- Ruiz, Guohui Xiao, Ahmet Soylu, Davide Lanti, Martin Rezk, Dmitriy Zheleznyakov, Martin Giese, Hallstein Lie, Yannis Ioannidis, Yannis Kotidis, Manolis Koubarakis, and Arild Waaler. Ontology based data access in statoil. Journal of Web Semantics, 44:3–36, 2017.

[5] Stefan Brüggemann, Konstantina Bereta, Guohui Xiao, and Manolis Koubarakis. Ontology-based data access for maritime security. In Extended Semantic Web Conference (ESWC), 2016.

[6] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. Linkedgeodata: A core for a web of spatial open data. Semantic Web Journal, 3(4):333–354, 2012. http://linkedgeodata.org/.

[7] Konstantina Bereta, Guohui Xiao, and Manolis Koubarakis. Ontop-spatial: Ontop of geospatial databases. Journal of Web Semantics, 58, 2019.

[8] https://data.epo.org/linked-data/

1.1.3 Data Analytics

We have implemented support of SPARQL aggregate functions for data analytics. Aggregate functions are a basic capability provided by essentially any database system, and typically application scenarios require them. Aggregate functions are needed by essentially all our scenarios, as they allow basic queries such as to count the number of EU projects a certain institution has participated in. One important challenge to provide aggregate functions over VKGs is to *treat cardinalities* carefully in order to obtain results that are compliant with the SPARQL semantics (e.g., for SUM and AVG): the SQL queries in a mapping produce bags (i.e., multisets) of tuples, but their induced RDF graphs contain no duplicates and thus are sets of triples; however, when a SPARQL query is evaluated, it results in a bag of solution mappings. To make the evaluation of aggregate functions feasible in practical scenarios, we had to develop novel and dedicated *optimization techniques*, involving a sophisticated treatment of the SQL DISTINCT operator. In Ontop v4, we support all 6 SPARQL aggregate functions: COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT, and SAMPLE. All these activities comply with the goals of Task 3.3.

1.1.4 Answer Justification

We want the users to be able to reconstruct why a specific answer was returned by Ontop. Such justification is not only in terms of where the data comes from (data provenance), but also in terms of which ontology axioms and mapping assertions were involved during the query reformulation phase (ontology provenance and mapping provenance, respectively). To support this task, we have implemented a prototype (not yet integrated in the main development branch of Ontop) relying on ProvSQL[9], a tool for provenance developed in the context of RDBMSs. ProvSQL only supports PostgreSQL, and to the best of our knowledge no other tool exists for other types of data sources that would be robust enough.

1.2 Data Linking and Modelling

1.2.1 Mapping Construction

We have improved the functionalities of the MPBoot system, which automatically generates ontologies and mappings from available data sources. The latest **MPBoot 2.0** showcased in INODE-SPARQL 1.0 supports data-driven and task-driven bootstrapping.

In the case of **data-driven bootstrapping**, MPBoot is now able to produce richer ontologies and mappings compared to the Direct Mapping approach, supporting domain and range axioms for data and object properties as well as subclass relations. MPBoot 2.0 also introduces a semi-automatic approach for ontology generation, allowing the user to specify the portions of the ontology to be generated automatically depending on their needs.

In the case of **task-driven bootstrapping**, MPBoot allows the user to drive the bootstrapping process by specifying a SQL query workload in order to derive semantic connections between classes and generate object properties accordingly. This can be extremely useful for cases

[9] https://github.com/PierreSenellart/provsql

where there is no explicit connection between classes and the expressivity of the SQL language is the only way to infer such connections.

1.2.2 Knowledge Base Construction

We have introduced a set of improvements in the **OpenDataLinking** component of INODE-SQL 2.0 focusing on information extraction from unstructured text.

With regard to the **triple extraction** process, we have implemented a **triple refinement** approach to combine the outputs of the precision-oriented approach of one engine and the recall-oriented approach of the second engine. This allows for an efficient unification of both engines' results, without sacrificing performance. An additional set of fine-tuning parameters are also added to ensure the seamless integration of the triple refinement system to the OncoMX data.

We have also increased the **synergy among the OpenDataLinking and OpenDataDialog** components. The distilled knowledge from the aforementioned information extraction processes is used to enrich the existing data models and can now be queried under the INODE-SQL.2.0 system.

1.3 Data Access & Exploration

1.3.1 By Natural Language

Progress within INODE. For translating natural language questions to SQL or SPARQL, INODE now supports three services with different capabilities: SODA, ValueNet and BioSODA. SODA (already integrated from the previous version) and the newly added ValueNet are both *text-to-SQL* services - also referred to as *NL-to-SQL* service. The former enables advanced keyword queries, while the latter allows for natural language queries. BioSODA is a *text-to-SPARQL* service - also referred to as *NL-to-SPARQL* service.

***ValueNet*[10] is a text-to-SQL system based on neural networks.** As an input, ValueNet receives a question in natural language and a specific database. As an output, ValueNet delivers a fully fledged SQL query which is then executed against said database and delivers the information the user asked for. ValueNet uses the power of large, pre-trained language models to understand a natural language question and synthesizes a SQL query which represents that question most accurately.

In contrary to classical, rule-based text-to-SQL systems such as SODA[11], ValueNet contains no engineered knowledge but learns from large, open source text-to-SQL data corpora like Spider[12]. After training ValueNet on a general text-to-SQL corpus we either apply the trained

[10] Ursin Brunner and Kurt Stockinger: "ValueNet: A Natural Language-to-SQL System that Learns from Database Information", *ICDE 2021*

[11] Blunschi, L., Jossen, C., Kossmann, D., Mori, M., & Stockinger, K. (2012). SODA: Generating SQL for business users. *Proceedings of the VLDB Endowment*, *5*(10), 932-943.

[12] Tao Yu et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task", *EMNLP 2018*

model of ValueNet to our project database (e.g., CORDIS) in a zero shot setting[13], or we fine-tune ValueNet on a low number of training samples prepared for that specific database (few shot learning).

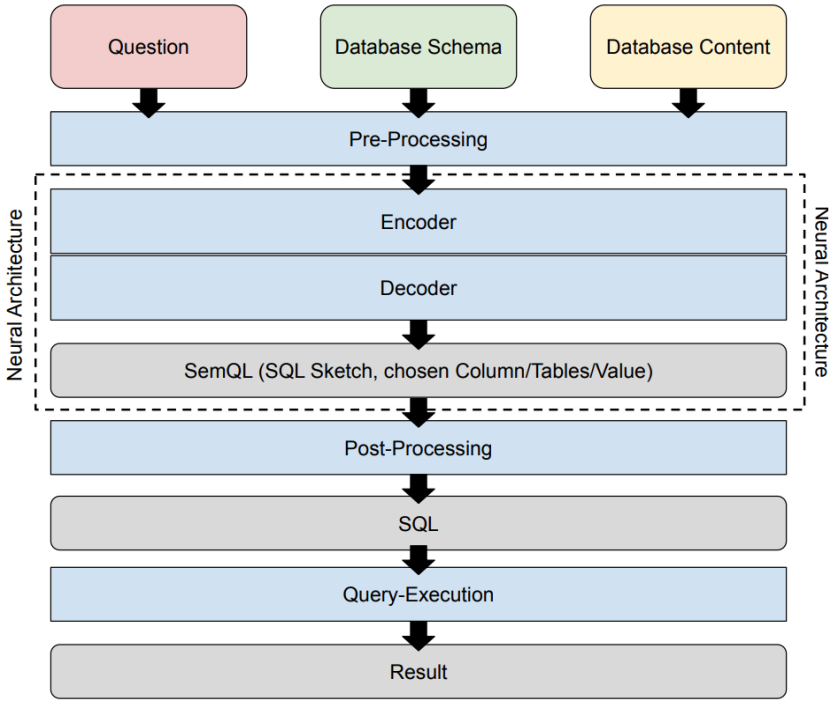


Figure 1.2: ValueNet high level architecture.

The flow of data in a text-to-SQL scenario is described in Figure 1.2. The user poses a *question* in natural language, which is going to be the input for ValueNet, together with the database (*schema* and *content*).

In the first stage, ValueNet will do *pre-processing* on the question/schema. One important sub-task here is to look for potential values in the question which are contained in the database (e.g., "Show me all *John Smith* in *Texas*"). These "hints" together with the input data are then submitted to the second stage.

The second stage is the core of the architecture, a large neural network built as an *encoder-decoder* architecture. The *encoder* is a large, pre-trained language model (based on transformers like BERT or BART) which aims to understand the natural language question. The *decoder* takes the output of the encoder and synthesizes a query step by step using a recurrent neural network. The output of the decoder is not SQL, but an intermediate, semantic query language (*SemQL*), which aims to abstract from some technical details of SQL (e.g., filters modeled by "WHERE" or "HAVING", a technical detail never specified by a user).

[13] In a zero shot setting, ValueNet has never been trained on the project database before, but just applies it's knowledge learned on general open-source datasets.

The *post-processing* step then translates the intermediate language (*SemQL*) into *SQL*. This translation is, in contrast to the encoder-decoder stage, a deterministic process. SemQL can get translated not only to SQL, but to any structured query language similar to SQL (e.g., SPARQL).

Bio-SODA[14] is a question answering system over domain knowledge graphs, in particular over RDF graph databases. The strength of Bio-SODA is that it uses a generic, graph-based approach, in order to answer natural language questions. Therefore it is domain-independent and does not require prior training data in order to be adapted to a new dataset.

Within INODE, we applied Bio-SODA to the CORDIS and SDSS databases, made available as virtual knowledge graphs through Ontop (for details see Section 1.1, Integrated Query Processing). We illustrate the question answering pipeline through the following example - consider the question: "*What are all spec galaxies with right ascension < 130 and declination > 5?*"[15]. The answering pipeline of Bio-SODA is explained below.

First, the main concepts in the questions are identified by a lookup against an inverted index over the RDF data. In this simple example, Bio-SODA will identify one (unambiguous) candidate match per concept, namely the entity: "*Spec Galaxy*", as well as the properties *"right ascension"* and "*declination*", shown in Figure 1.3 below. Bio-SODA also detects two numerical filters to be applied to these properties, based on the comparison operators used in the question. For now, we support basic numerical operators, such as ">", "<", "=", ">=", or "<=".

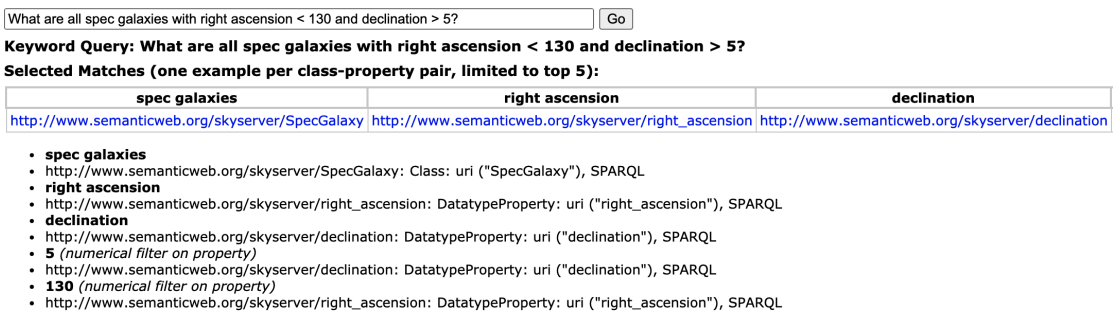


Figure 1.3: Example question and candidate matches in Bio-SODA.

In the second step, the system computes the minimal connected subgraph that covers all the candidate matches. In this case, the minimal subgraph contains only the class "SpecGalaxy" and its directly connected data properties, "right_ascension" and "declination". In the final step, Bio-SODA constructs the corresponding SPARQL query based on this subgraph, applying also the two filters mentioned in the question. The resulting SPARQL query is executed against the SPARQL endpoint of SDSS and selected results are then presented to the user in tabular form, as illustrated in Figure 1.4.

[14] Ana Claudia Sima, Tarcisio Mendes de Farias, Maria Anisimova, Christophe Dessimoz, Marc Robinson-Rechavi, Erich Zbinden and Kurt Stockinger, "Bio-SODA - A Question Answering System for Domain Knowledge Graphs", under review at the *Semantic Web Journal*

[15] "spec galaxy" denotes a galaxy that has spectroscopic data available in SDSS.

Keyword: spec galaxies **Match:** http://www.semanticweb.org/skyserver/SpecGalaxy
Keyword: declination **Match:** http://www.semanticweb.org/skyserver/declination
Keyword: ascension **Match:** http://www.semanticweb.org/skyserver/right_ascension

SPARQL query:

```
SELECT DISTINCT ?specgalaxy ?specgalaxy_declination ?specgalaxy_right_ascension WHERE {
    ?specgalaxy <http://www.semanticweb.org/skyserver/declination> ?specgalaxy_declination.
    ?specgalaxy a <http://www.semanticweb.org/skyserver/SpecGalaxy>.
    ?specgalaxy <http://www.semanticweb.org/skyserver/right_ascension> ?specgalaxy_right_ascension.
    FILTER (?specgalaxy_right_ascension < 130 )
    FILTER (?specgalaxy_declination > 5 )
    }
    LIMIT 100
```

?specgalaxy	?specgalaxy_declination	?specgalaxy_right_ascension
http://www.semanticweb.org/skyserver/specobj/specobjid=469515959745406976	14.2740340000000003	8.34072899999999962
http://www.semanticweb.org/skyserver/specobj/specobjid=864034402506351001 6	28.4444729999999986	12.2750170000000001
http://www.semanticweb.org/skyserver/specobj/specobjid=8690898482326949888	32.0786220000000029	12.7790130000000008
http://www.semanticweb.org/skyserver/specobj/specobjid=7425387786285633536	33.6727890000000016	15.1399609999999996
http://www.semanticweb.org/skyserver/specobj/specobjid=8695560417467191296	34.1520119999999991	18.0093870000000003
http://www.semanticweb.org/skyserver/specobj/specobjid=8650501876222283776	28.2784429999999993	21.5287369999999996
http://www.semanticweb.org/skyserver/specobj/specobjid=8652574712550150144	24.7360210000000009	23.4612680000000005
http://www.semanticweb.org/skyserver/specobj/specobjid=12463751143212994560	8.98383310000000002	24.2210299999999989

Figure 1.4: Example SPARQL query and selected results.

It is important to note that the properties "right_ascension" and "declination" used in the RDF graph, close to the terms used in the NL query, occur under different (and less explicit) names in the original SDSS database ("ra" and "dec", respectively). The *mapping* between the names used in the RDF graph and the attribute names in the source database is provided by the OpenDataLinking service of INODE.

1.3.2 Pipeline Operators

We are given a relational database that we represent as a set of records. A new set of records could be obtained by joining multiple tables using different exploration operators. The exploration operators we formulated in our pipeline are instances of `by-example`. In its general form, `by-example` takes an example set D of items and returns one or several sets D' of items that are related to items in D by some interpretation. We here recall the interpretations and equivalent SQL queries for operators: `by-superset`, and `by-facet`, and `by-overlap`.

Exploration operator `by-superset`. This operator takes a set D of items and a set A of attributes and returns the smallest superset of D that preserves the values of attributes in A. The corresponding SQL is: `SELECT FROM` D `WHERE` P_A

The algorithm for implementation of `by-superset` can be summarized as: receive input D (a set of items), and A (a set of attributes). Then, put in D' the most overlapping set with D. And then, take the next set and put it in S. As long as the time limit is not exceeded and the overlapping threshold is not crossed, browse sets S, and each time replace D' by S, if the following conditions are satisfied: S is smaller than D', then input set D is included in S and values of attributes in A are preserved.

Exploration operator `by-facet`. The operator `by-facet`(D, A) takes a set D of items and a set A of attributes and returns as many subsets of D as there are combinations of

values of the attributes in A. This is akin to faceted search where k is dictated by the number of combinations of values of the attributes in A.

The SQL expression for `by-facet` is: `SELECT` * `FROM` D `GROUPBY` A

The algorithm for implementation of the `by-facet` operator can be summarized as: split the input set to several subsets by taking input D (a set of items), A (a set of attributes), and k (the number of result subsets, returned in D'). `by-facet` is equivalent to the SQL operator *group by*. So we use a *group by*, then we take the k largest sets.

Exploration operator `by-overlap`. This operator takes a set D of items and returns k sets of items in D such that each set D_i, $i = 1 \ldots k$ overlaps the least with D and overlap between those sets is minimized.

The operator `by-overlap` is a composition of `by-superset` and `by-subset`. In SQL, it takes a set D of items to which is associated a conjunction P of predicates, and finds an overlapping set D' of items that contains some items satisfying P' obtained by dropping one conjunct in P and some items satisfying P'' obtained by modifying P (either by dropping conjuncts and keeping at least one conjunct per attribute, or by adding a new conjunct).

The SQL expression is as follows:

`SELECT` * `FROM` D, d `WHERE` $P'(d)$

`SELECT` * `FROM` D'', d `WHERE` $P''(d)$

The `by-overlap` operator is implemented with a greedy algorithm. The algorithm for implementation of the `by-overlap` operator can be summarized as: receive input D (an input set of items), k (the number of overlapping sets we must return), $tlimit$ (a time limit and μ (a threshold that represents the maximum overlap between a result set and D). First, we use the index to put in D' the k sets that overlap the least with D. Then, we take the next set s, and as long as the time limit is not exceeded, and the overlap threshold is not crossed, we update D', if s satisfies $| overlap(D' \backslash s') + s | < | overlap(D') |$

The operators: `by-superset`, `by-facet`, and `by-overlap` are integrated in INODE-SQL 2.0.

Considerable effort was invested into switching the pipeline operators from an "in-memory" data back-end with pre-computed list of sets and set indexes, to a more generic and common SQL back-end. This change allows the use of the pipeline operators with any SQL database, without any adaptation, making them more generic and versatile.

As part of future work we will improve the implementation of `by-overlap` to work without the pre-computed set indexes. Moreover, two more operators allowing similar jumps through the data-space were developed: `by-neighbors` and `by-distribution`. The semantics of these operators is provided in Section 4.1.3. We provide a screenshot of their API documentation and of their application on SDSS.

Operator `by-neighbors`. The `by-neighbors` operator exploits the ordinal attributes, i.e., the attributes with an intrinsic order (binned numerical values, discrete numerical

values, dates, times). The user provides a list of such attributes and an input set, and the operator returns the sets whose values for the input attributes are in the neighborhood of the values in the input set.

This operator allows the user to explore the data adjacent to a given place in space (the input set), by changing one dimension above and below and keeping the others constant.

Operator `by-distribution`. The `by-distribution` operator uses the same "ordered" attributes as `by-neighbors`, but instead of looking for neighbors on a given list of attributes, it takes all the ordered attributes in the input set description, and finds all the sets having the same difference between each ordered attribute.

This operator allows the user to find other sets showing the same relations between the attributes as the example. It allows the exploration of remote places of the dataspace while keeping a similarity to the example.

1.4 User Assistance

1.4.1 Explanations

Progress within INODE. In INODE Release 1.0, *Logos* was used to translate SQL queries from SODA and Nalir+ into natural language. Logos is a so-called *SQL-to-NL* service. In the current release INODE-SQL 2.0, Logos is extended to translate the SQL queries produced by ValueNet, as well as to translate the output queries of the data exploration pipelines. At present, Logos supports natural language explanations for both CORDIS and SDSS databases.

In what follows, we show the improvements of Logos per category, giving examples.

Improvements in terms of query semantics. The current version of Logos includes mainly grammatical changes. Those changes were made not only for the system to be compatible with other systems as well (e.g., Nalir+, SODA, etc.), but also to further develop Logos capabilities of translating SQL queries that are more demanding (in terms of translation).

The newly developed extensions enabled Logos to support the following query types:

<u>select top – limit clauses</u>

CORDIS query example (select top):

```
SELECT TOP 10 * FROM projects WHERE projects.title LIKE
'%theseus%';
```

Translation: Find everything about projects whose title is like %theseus%. Limit the results to top 10.

SDSS query example (limit):

```
SELECT specobjid FROM specobj WHERE class = 'STAR' and
zwarning = 0 LIMIT 100;
```

Translation: Find spectroscopic objects whose class is STAR and redshift warning is 0. Limit the results to top 100.

not like operator

CORDIS query example:
```
SELECT * FROM projects WHERE ec_fund_scheme NOT LIKE
'%MSCA-IF-EF%';
```

Translation: Find everything about projects whose fund scheme is not like %MSCA-IF-EF%.

in – not in operators

CORDIS query example:
```
SELECT total_cost, ec_max_contribution,
framework_program, ec_fund_scheme
FROM projects WHERE framework_program IN ('FP7') AND
ec_fund_scheme NOT IN ('ERC-SG', 'CP-SoU');
```

Translation: Find the total costs, max contributions, framework programs and fund schemes of projects whose framework program is in {FP7} and fund scheme is not in {ERC-SG, CP-SoU, MC-IAPP, ERC-CG}.

projections including "count(*)"

CORDIS query example:
```
SELECT COUNT(*) FROM projects WHERE start_year = 2012;
```

Translation: Find the cardinality of projects whose start year is 2012.

Improvements in terms of translation. Another important addition to our latest release is the annotated database graph (see Section 5). This feature enables the generation of more natural, human-like translations.

CORDIS query example:
```
SELECT p.full_name FROM people p, projects pr
WHERE pr.principal_investigator = p.unics_id;
```

- Logos v.1: Find the full names of people associated with projects.
- Logos v.2: Find people that are principal investigators of projects.

SDSS query example:
```
SELECT n.* FROM neighbors n, photoobj p
WHERE p.objid = n.objid AND p.b = 1.072 AND p.l =
174.535;
```

- Logos v.1: Find everything about neighbors associated with photoobj whose b is 1.072 and l is 174.535.

- Logos v.2: Find everything about nearest neighbors of photometric objects whose galactic latitude is 1.072 and galactic longitude is 174.535.

Moreover, specific attributes from relations have been chosen to serve as representatives of their corresponding relations. We call these attributes *heading attributes* and when appearing in the SELECT-part of a query, the way of producing the natural language explanation changes, giving a more natural result.

CORDIS query example:

```
SELECT title FROM projects WHERE start_year > 2018;
```

- Logos v.1: Find the titles of projects whose start year is greater than 2018.
- Logos v.2: Find projects whose start year is greater than 2018.

SDSS query example:

```
SELECT specobjid FROM specobj;
```

- Logos v.1: Find the specobjids of specobj.
- Logos v.2: Find spectroscopic objects.

Working with the CORDIS database, we noticed that some relations were used only to connect other relations (bridge tables), storing indices. Manually reported, those relations are excluded from the translation procedure.

CORDIS query example:

```
SELECT pr.title FROM projects pr, project_subject_areas
psa, subject_areas sa
WHERE pr.unics_id = psa.project AND psa.subject_area =
sa.code AND sa.title = 'Robotics';
```

- Logos v.1: Find the title of projects, for projects associated with project subject areas, and for project subject areas associated with subject areas whose title is robotics.
- Logos v.2: Find projects on subject areas whose title is robotics.

1.4.2 Recommendations

Generating query recommendations in INODE-SQL 2.0 leverages the query capabilities of PyExplore. *PyExplore*[16] is a data exploration tool aimed at helping end users formulate queries over new datasets. PyExplore takes as input an initial query from the user along with some parameters and provides interesting queries by leveraging data correlations and diversity. It is able to handle datasets with mixed numeric and categorical attributes.

[16] A. Glenis, G. Koutrika. PyExplore: Query Recommendations for Data Exploration without Query Logs. ACM SIGMOD, 2021

As input to PyExplore, we consider a query of the form:

```
SELECT A FROM T WHERE P
```

where T is a set of *tables* joined for the query, A is a subset of the table *attributes* projected in the query result, and P is a *conjunction of selection predicates*.

PyExplore then produces a set of *ranked queries with an augmented WHERE-clause* if there was a WHERE-clause in the initial query or a new WHERE-clause if there was no WHERE-clause.

The first step of the recommendation process is to find 'interesting' subsets of query attributes. PyExplore leverages two notions: attribute correlation and diversity.

- *Correlation-based*. Correlation is the measure of how two features are correlated. For example, the month-of-the-year is correlated with the average daily temperature, and the hour-of-the-day is correlated with the amount of light outdoors. Data scientists are interested in correlated attributes to highlight relationships between attributes of the data set.

  First, PyExplore computes the correlation of each pair of attributes in the query results. One challenge is how to deal with different types of attributes. For comparison between numerical attributes, it uses *Pearson correlation*[17], for categorical-categorical, it uses *Cramér's V* [18], and for categorical-numerical, it uses *Correlation Ratio*[19]. To make all correlation metrics in the same range, i.e., [0, 1], we take the absolute value of the Pearson Correlation. Then, the inverse of the absolute value of the correlation matrix is used as a distance matrix, which is given as input to a clustering algorithm that creates clusters of correlated attributes.

  PyExplore uses two options for clustering correlated attributes: (a) *hierarchical clustering*[20] with complete linkage takes as input the maximum number **size_max** of attributes per cluster and decides the number of clusters accordingly, and (b) *OPTICS*[21], which is a density-based algorithm that decides how many clusters to create and also clusters all outliers together. This cluster with outliers is ignored by the recommendation algorithm.

- *Diversity-based*: Intuitively, an attribute that has a diverse set of values is interesting because it allows the user to explore a larger part of the initial query results compared to a less diverse attribute. To compute diversity for numerical columns, PyExplore uses the *normalized Shannon entropy*. For categorical columns, it computes the ratio between the unique values in the column and the total rows in the column. Then, subsets of diverse attributes up to a *size_max* size are generated in a greedy manner.

Note that both correlation and diversity are computed on-the-fly on the results of the initial user query.

[17] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
[18] https://en.wikipedia.org/wiki/Cram%C3%A9r%27s_V
[19] https://en.wikipedia.org/wiki/Correlation_ratio
[20] https://en.wikipedia.org/wiki/Hierarchical_clustering
[21] Ankerst, Mihael et al,OPTICS: Ordering points to identify the clustering structure

The query recommendation process generates the queries using clustering to find interesting subsets of the data, and then feeds the results of the clustering into a decision tree classifier to obtain the rules for producing the queries.

Result Clustering. For each subset of attributes identified by the first step, PyExplore clusters the initial query results using the values of the attributes in the subset. It uses two options.

The first is K-means with scaling and encoding categorical values as dummy variables.

However, encoding categorical values as dummy variables can lead to increased time and space complexity for data sets with high-cardinality categorical values. To overcome this problem, PyExplore uses K-modes. Specifically, to enable the clustering of categorical data in a fashion similar to k-means, the algorithm proposed by Huang (1997)[22] uses a simple matching dissimilarity measure, replaces the means of clusters with modes, and uses a frequency-based method to update modes in the clustering process to minimise the clustering cost function. The algorithm proposed by Huang (1998)[23], through the definition of a combined dissimilarity measure, further integrates the k-means and the algorithm presented by Huang (1997) to allow for clustering objects described by mixed numeric and categorical attributes.

Query Generation: For each subset, the resulting cluster labels are fed into a decision tree classifier to produce the split points of the data. The resulting split points are used to create the recommended SQL queries.

More concretely, PyExplore traverses the decision tree from the leaves up to the root, and for each path from the starting leaf to the root, it generates an output query. The conditions of the WHERE clause of each query describe the cluster boundaries as they are described by each path in the decision tree. Since PyExplore uses clustering to obtain partitions of the data space, it leverages clustering quality metrics to obtain a ranking of the produced recommended queries. Specifically, it uses density as a quality metric for clustering. Higher density score is better, meaning that the respective query describes a very dense area of the data.

The following example shows the recommendations of the running query on the CORDIS dataset (see Figure 1.5):

CORDIS query example:

```
SELECT total_cost, ec_max_contribution,
framework_program, ec_fund_scheme FROM projects;
```

Recommendations:

Here we see that framework_programe and ec_fund_scheme are correlated and form a view. We see that the recommended queries propose values for funding schema that make sense such as “FP7” and “H2020”.

[22] Zhexue Huang. Clustering large data sets with mixed numeric and categorical values. In *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 1997. DOI: 10.1.1.94.9984.

[23] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery* 2(3): 283-304. 1998.

View	Query	Scores
['ec_max_contribution']	SELECT total_cost,ec_max_contribution,framework_program,ec_fund_scheme FROM projects where `ec_max_contribution` < 34022656.0 SELECT total_cost,ec_max_contribution,framework_program,ec_fund_scheme FROM projects where `ec_max_contribution` < 137245424.0 and `ec_max_contribution` >= 34022656.0 SELECT total_cost,ec_max_contribution,framework_program,ec_fund_scheme FROM projects where `ec_max_contribution` >= 137245424.0	0.5990866422653198
['framework_program' 'ec_fund_scheme']	SELECT total_cost,ec_max_contribution,framework_program,ec_fund_scheme FROM projects where `framework_program` in ('H2020', 'IC', 'CIP') and `ec_fund_scheme` in ('PB', 'ERC-SyG', 'MSCA-IF-EF-RI', 'ERC-POC-LS', 'COFUND-PPI', 'CSA', 'CIP-EIP-EI-PMRP', 'ERC-POC', 'MSCA-IF-EF-CAR', 'MSCA-ITN-EJD', 'Shift2Rail-RIA-LS', 'PA', '0.0', 'CS2-IA', 'Shift2Rail-IA-LS', 'BBI-CSA', 'IMI2-RIA', 'BBI-IA-FLAG', 'SESAR-RIA', 'CS2-CSA', 'ERA-NET-Cofund', 'IMI2-CSA', 'FCH2-CSA', 'BBI-IA-DEMO', 'IA-LS', 'SME-1', 'CSA-LS', 'ERC-COG', 'FCH2-IA', 'RIA-LS', 'H2020-EEN-SGA', 'MSCA-IF-EF-SE', 'ECSEL-IA', 'MSCA-RISE', 'ECSEL-CSA', 'BBI-RIA', 'FCH2-RIA', 'MSCA-COFUND-DP', 'Shift2Rail-CSA', 'BPN', 'MSCA-IF-EF-ST', 'SME-2', 'CS2-RIA', 'SGA-RIA', 'COFUND-EJP', 'TN', 'ERC-ADG', 'COFUND-PCP', 'ERC-LVG', 'SGA-CSA', 'ECSEL-RIA', 'RIA', 'Shift2Rail-IA', 'Shift2Rail-RIA', 'ERC-STG', 'MSCA-ITN-EID', 'SESAR-IA', 'MSCA-ITN-ETN', 'PPI', 'PCP', 'SESAR-CSA', 'MSCA-COFUND-FP', 'IA', 'MSCA-IF-GF') SELECT total_cost,ec_max_contribution,framework_program,ec_fund_scheme FROM projects where `framework_program` in ('FP7') and `ec_fund_scheme` in ('PB', 'ERC-SyG', 'MSCA-IF-EF-RI', 'ERC-POC-LS', 'COFUND-PPI', 'CSA', 'CIP-EIP-EI-PMRP', 'ERC-POC', 'MSCA-IF-EF-CAR', 'MSCA-ITN-EJD', 'Shift2Rail-RIA-LS', 'PA', '0.0', 'CS2-IA', 'Shift2Rail-IA-LS', 'BBI-CSA', 'IMI2-RIA', 'BBI-IA-FLAG', 'SESAR-RIA', 'CS2-CSA', 'ERA-NET-Cofund', 'IMI2-CSA', 'FCH2-CSA', 'BBI-IA-DEMO', 'IA-LS', 'SME-1', 'CSA-LS', 'ERC-COG', 'FCH2-IA', 'RIA-LS', 'H2020-EEN-SGA', 'MSCA-IF-EF-SE', 'ECSEL-IA', 'MSCA-RISE', 'ECSEL-CSA', 'BBI-RIA', 'FCH2-RIA', 'MSCA-COFUND-DP', 'Shift2Rail-CSA', 'BPN', 'MSCA-IF-EF-ST', 'SME-2', 'CS2-RIA', 'SGA-RIA', 'COFUND-EJP', 'TN', 'ERC-ADG', 'COFUND-PCP', 'ERC-LVG', 'SGA-CSA', 'ECSEL-RIA', 'RIA', 'Shift2Rail-IA', 'Shift2Rail-RIA', 'ERC-STG', 'MSCA-ITN-EID', 'SESAR-IA', 'MSCA-ITN-ETN', 'PPI', 'PCP', 'SESAR-CSA', 'MSCA-COFUND-FP', 'IA', 'MSCA-IF-GF') SELECT total_cost,ec_max_contribution,framework_program,ec_fund_scheme FROM projects where `ec_fund_scheme` in ('CP-FP', 'MC-IAPP', 'CP-FP-SICA', 'JTI-CP-IMI', 'JTI-CP-ENIAC', 'CP-SICA', 'MC-IIF', 'CP-CSA-Infra', 'MC-CIG', 'CP-IP-SICA', 'CP-CSA', 'CP-SoU', 'BSG-SME', 'CPCSA', 'CP-CSA-Infra-PP', 'MC-ERG', 'MC-COFUND', 'CP-IP', 'JTI-CS', 'BSG-SME-AG', 'JTI-CP-ARTEMIS', 'JTI-CP-FCH', 'MC-IEF', 'NoE', 'ERC-CG', 'MC-ITN', 'CSA-ERA-Plus', 'CP-TP', 'CSA-ERANET', 'MC-IIFR', 'CSA-SA(POC)', 'MC-IOF', 'CSA-SA', 'BSG-CSO', 'MC-IRSES', 'CSA-CA', 'ERC-SG', 'JTI-CSA-FCH', 'MC-IRG') SELECT total_cost,ec_max_contribution,framework_program,ec_fund_scheme FROM projects where `ec_fund_scheme` in ('CP', 'ERC-AG')	0.4657362103462219

Figure 1.5: Recommendations for a CORDIS query.

The following screen shows the recommendations of the query on the SDSS dataset (see Figure 1.6):

SDSS query example:

```
SELECT objid,ra,dec FROM photoobj LIMIT 10000;
```

View	Query	Scores
['dec']	SELECT objid,ra,dec FROM photoobj where dec >= 33.46176 SELECT objid,ra,dec FROM photoobj where dec < 30.594868 SELECT objid,ra,dec FROM photoobj where dec >= 30.594868 and dec < 33.46176	0.9990967512130737
['objid' 'ra']	SELECT objid,ra,dec FROM photoobj where ra < 150.3222 SELECT objid,ra,dec FROM photoobj where ra >= 150.3222 and ra < 183.21983 SELECT objid,ra,dec FROM photoobj where ra < 234.83829 and ra >= 183.21983 SELECT objid,ra,dec FROM photoobj where ra >= 234.83829	1

Figure 1.6: Recommendations for an SDSS query.

As we can see objid and ra are correlated and form a view. We can also see that the recommendation system choses some interesting values as split points for ra and dec.

1.5 Multi-Modal Discovery

The goal of the Multi-Modal Discovery layer is to implement means for result exploration and follow-up operator execution. In doing so, it seeks to help users understand the options they have for finding the data they need through visual exploration of results at each exploration step and interactive manipulation and optimization of exploration operators.

Visual exploration of intermediate results aims at enabling users to visually manage the actual content. When necessary, users can revise their exploration steps through interactive manipulation and optimization of exploration operators.

Progress within INODE. Work on visual guidance and exploration of search results as well as interactive manipulation and optimization of queries has been continued and work to accommodate for the integration requirements has already been started. In summary, many minor changes have been implemented since the INODE 1.0 release, consolidating and streamlining the Multi-Table Explorer user experience as well as the code base in preparation for OpenDataDialog 3.0. Noteworthy changes to the previous release (described in D3.1) are described below.

1.5.1 Visual Result Exploration

Relevance-based column ordering. The SDSS data set often results in tables with 500 different columns, which, for example, represent various measurements and parameters recorded during sky observations. To ease overview and navigation, columns are now ordered by relevance, putting the important columns to the left of the Multi-Table Explorer

view while less relevant columns are sorted to the right. This should, on average, reduce the need to scroll horizontally. In this release, the relevance of columns is increased based on interviews with the use case providers, only. However, the system is already prepared to apply more complex relevance measures in a future release, for example to integrate the correlation-based or diversity-based measures described in Section 1.4.2 *Recommendations*. Ongoing (non-integrated) work for arranging and structuring search results will be presented in Deliverable D3.3.

Improved information density. To maximize the information conveyed to the user, the resolution of histograms has been increased, data type meta information is available as a tool tip, axis labels have been added, and popup menus have been redesigned to save space. A comparison of the previous and the current release is given in Figure 1.7.

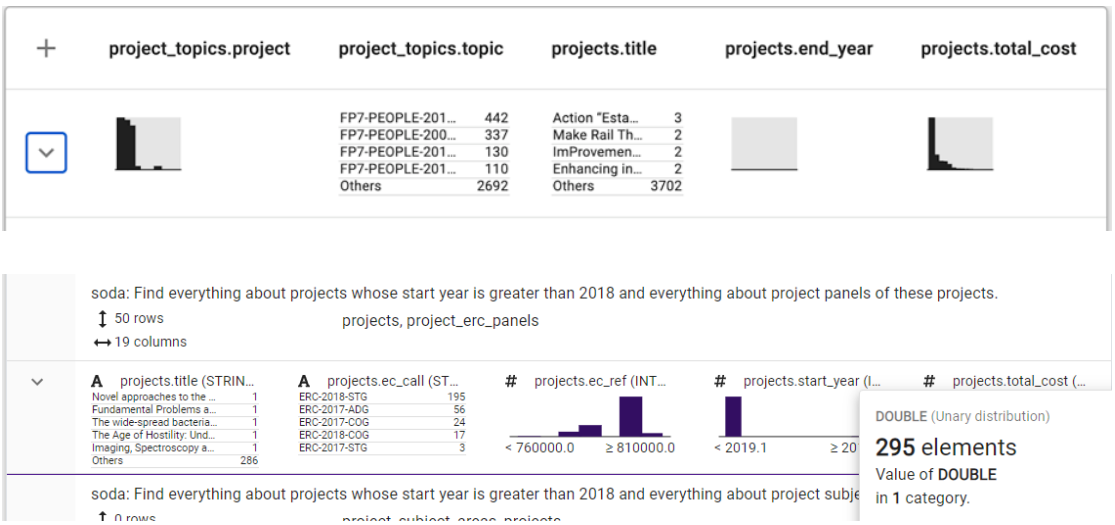


Figure 1.7: Comparing the upper to the lower image, a higher data-ink-ratio as well as less wasted screen real estate can be observed, while columns are still distinguishable from each other.

Table overview information. We now include meta information provided by our partner services into the Multi-Table Explorer view, for example, the width and height of the table, but also the natural language explanation as well as its accompanying SQL statement. Also, we added pagination to enable the user to view excerpts of interesting tables.

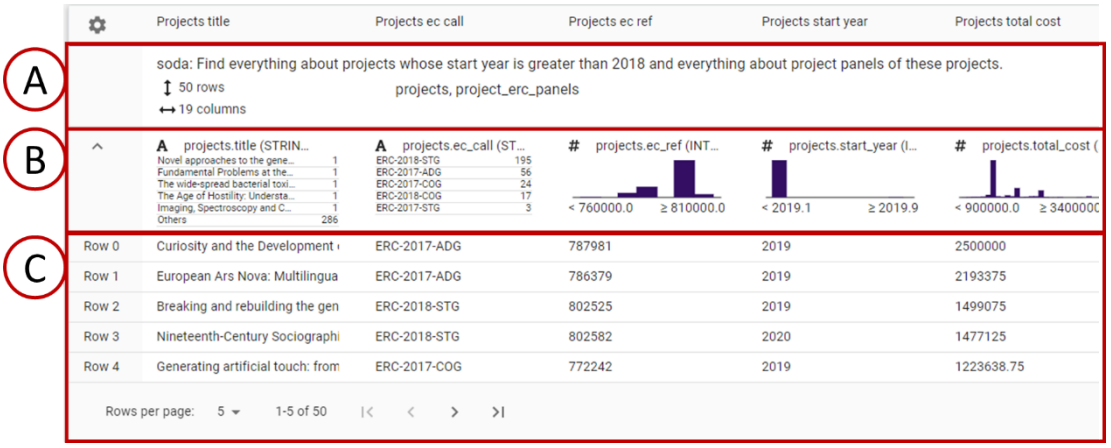


Figure 1.8: (A) provides relevant meta information about a result, such as its NL representation, width and height of the fetched result, and other information. (B) shows overview information about various columns. The user can toggle the display of (C) to browse through the actual table rows on demand.

Support for more data types. Besides numerical (integer, double) and categorical (string) data types, the Multi-Table Explorer now supports image URLs as a separate data type (as needed for the SDSS dataset, see Figure 1.9). On the one hand, it serves as a validation that new data types can be added and, on the other hand, is an intermediate step to integrate the galaxy explorer app prototype which demoes `by-distribution` and `by-neighbors` operators.

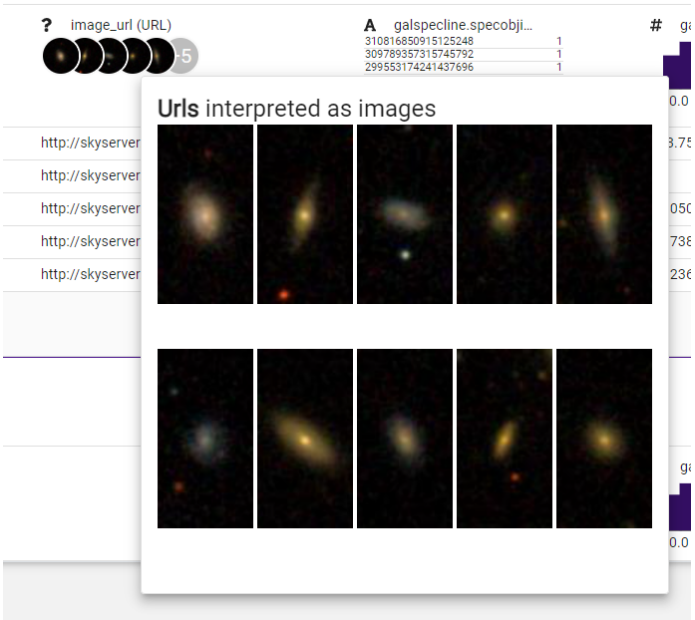


Figure 1.9: URL columns show a small preview of a random set which fits the size constraints for columns (top left of the image), but also shows a large tooltip with more and larger previews if the user hovers over it.

1.5.2 Visual Query Manipulation

As new services for exploring and interacting with the data have been added, we extended the table explorer to trigger these actions. This includes the ability to explore by recommendations (see Figure 1.10 (left)) and to refine a natural language query based on a selected table result (see Figure 1.10 (middle)). Some advanced techniques, for example, explore-by-neighbors, support only specific data types, but implementation of those constraints has been postponed after M18.

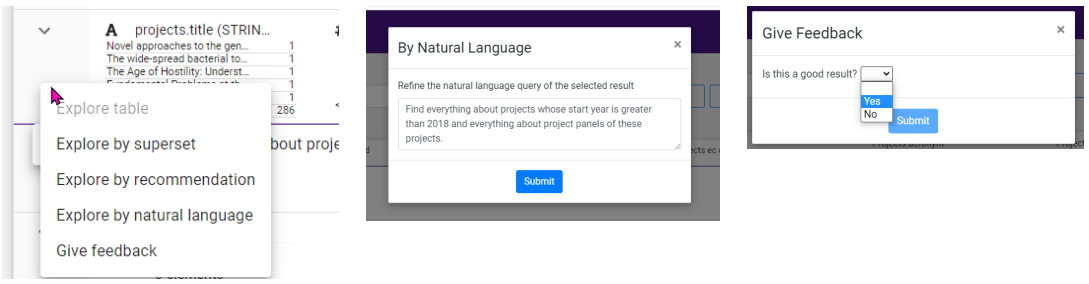


Figure 1.10: (left) new operators ' Explore by recommendation' , 'Explore by natural language' and an 'Give feedback' action have been integrated. (center) If a user clicks 'Explore by natural language', they can adapt the natural language query to suit their needs in a modal. (right) Also, users can provide relevant feedback if a search result suits their needs or not.

1.5.3 Integrated seamless query-response loop

Originally, Task 7.3 was scheduled to begin by M13, but we started earlier to enable integration of our partners' operators where they semantically fit into the application. Most of the work has already been shipped in OpenDataDialog 1.0. However, **performance improvements** have been integrated as part of this task**.** As users expect to have a fluent experience when surfing the web, we took measures to reduce the time to first contentful paint on the front end. One issue was that rendering hundreds of histograms, tables and other information introduced performance issues on mid-range and low-end computers. We reduced this by limiting the initial view to the most important columns first. The users are still able to select whatever they are interested in, or even display all information available. This improvement does not mitigate server-client related loading times but increases the snappiness of the application.

1.6 Evaluation

In order to implement the evaluation of the INODE system (WP8), it is necessary to design a logging mechanism first, so that we can record various system-generated events. In this version of the INODE system, we developed a logging mechanism that records various system parameters. Therefore for WP8, we completed Deliverable 8.1 (defining quantitative measures for our system), Deliverable 8.2 (evaluation report on the quantitative measures for our systems), and outline plans for Deliverable 8.3 (design of study for qualitative evaluation part).

As the main goal is system evaluation, we present our work related to:

1. methodology to extract & report the evaluation parameters (covers Deliverable 8.1)
2. set of quantitative evaluation parameters (covers Deliverable 8.2)
3. methodology to mathematically analyse the extracted evaluation parameter (covers Deliverable 8.3 - partially).

1.6.1 Logging of system parameters

System evaluation requires analysis of various system factors. So as a first step, it is necessary to come up with a technique to extract and then record these parameters from the INODE system. We have implemented a dedicated (separate) logging mechanism for recording the evaluation parameters during each user session. We design our dedicated logging mechanism with features such as:

- The logging mechanism *structure* is kept *flexible* so as to easily deal with the addition of new evaluation parameters.
- The logging mechanism assigns a *user ID to each user session* and stores it as a web cookie, thus ensuring identification of the same users across multiple sessions.
- Logs are *JSON objects* with predefined log structure, which can be parsed easily using various python libraries.
- The logging mechanism ensures *no delays in recording user events* by requesting time stamps from the frontend.

Log entries are recorded by means of calls to a logging API. The calls to this API are issued at the moment from the integration layer, which corresponds mainly to our INODE OpenDataDialog 2.0 application (the main interface for users to interact with INODE-SQL 2.0). The integration layer also includes a couple of wrapper services, namely the NL-to-SQL, i.e., SODA and ValueNet, and SQL-to-NL (i.e., Logos) services. These provide a REST API to the underlying tools. In the case of NL-to-SQL, the Nalir+ and SODA systems are executable Java applications that communicate via their standard input and output, so a wrapper is required to give them a REST API. ValueNet already has such an API, but in order to homogenize it with the API for Nalir+ and SODA developed in the previous release (INODE-SQL 1.0), we have also assigned its management to the NL-to-SQL service. In this case, the NL-to-SQL wrapper acts as a proxy, translating a request directed towards the common NL-to-SQL API into a request directed towards the specific ValueNet API. Similarly, the SQL-to-NL service provides a REST API to the Logos tool, which is a Java library.

The OpenDataDialog frontend logs user interactions, such as mouse clicks. The backend side of the application logs the inputs, outputs, and latency of the Recommendations and Pipeline Operator APIs (both of which are called from the OpenDataDialog's backend). The two wrapper services log, similarly, the inputs, outputs, and latency of the NL systems and Logos tool, respectively.

The architecture of the logging mechanism for INODE is shown in Figure 1.11. The term "container" in the architecture figure denotes a docker container, since the different services/components of INODE-SQL 2.0 are deployed as docker containers. The logging API is also deployed in its own docker container. This architecture includes the possibility to call the logging API from within the Recommender and Pipeline Operators services, so as to record

parameters that can only be measured internally, even though these services are not directly calling the logging API in the present release.

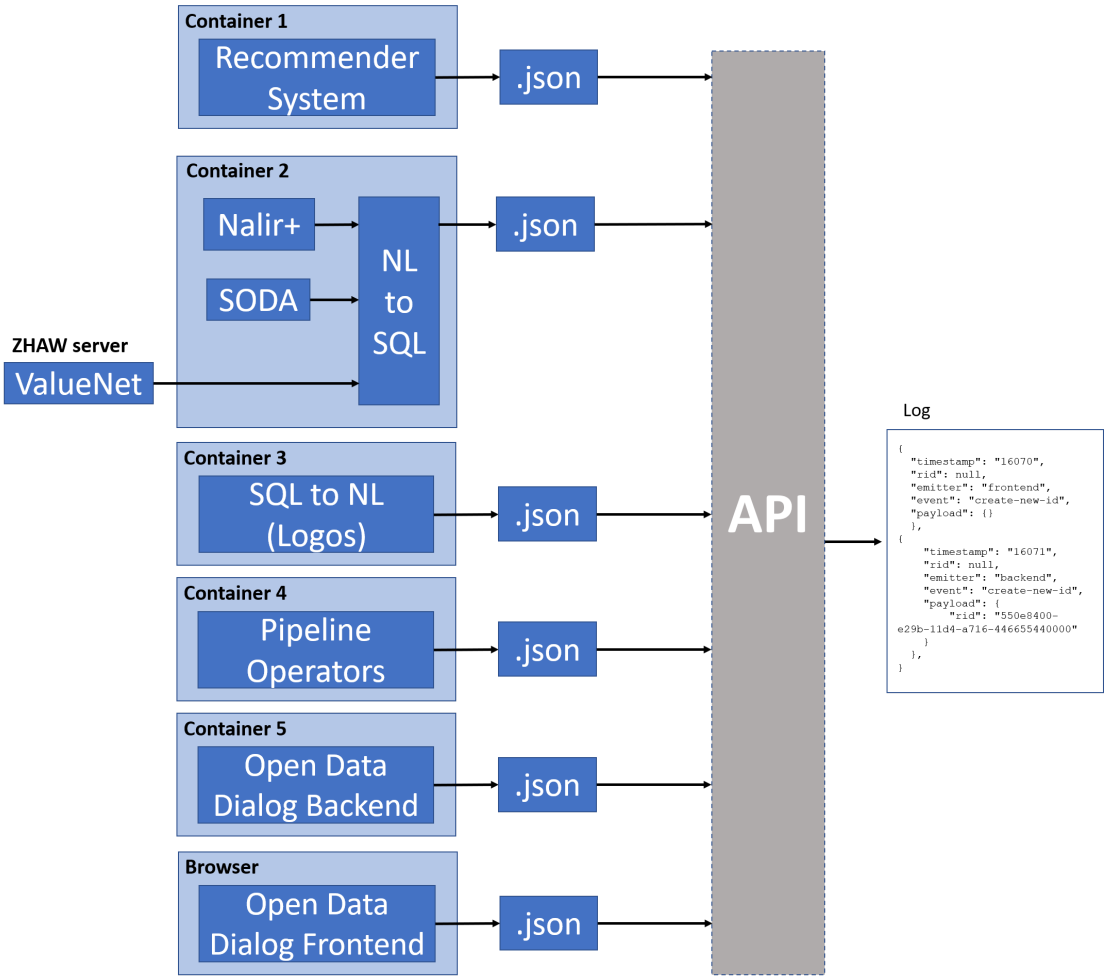


Figure 1.11: System architecture for logging mechanism for INODE system.

1.6.2 Quantitative evaluation parameters

Once the logging mechanism is in place, the next thing is to record certain system parameters to produce quantitative measures for system evaluations. Currently we extract the evaluation parameters shown in Table 1.3 from our logs.

	Parameter	Description
1	Session start time	Timestamp at which the users start using the session.

2	Session stop time	Timestamp at which the users stop using the session.
3	Pool of operators	Recording the pool of operators that each participant utilized during the experiment.
4	UI objects	Recording clicks on a fixed set of objects (only UI objects), that the user initiates.
5	Query Execution start time	Time at which the query execution is started (e.g., after NL to SQL query translation).
6	Query Execution end time	Time at which the query execution is ended (e.g., NL to SQL query translation).
7	Query Execution Latency	Time taken for query execution measured at the backend (e.g., after NL to SQL query translation).
8	Questionnaire Output	Recording each participant's responses to the questionnaire, at the end of the session.

Table 1.3: List of evaluation parameters that are recorded to perform quantitative and qualitative system evaluation.

1.6.3 Data analysis

Now that the evaluation parameters are available, the next part is to analyze these parameters extracted for each user by applying statistical methods. This section outlines the methodology that we will use to evaluate our set of operators by using data from user feedback and statistical analysis.

For the purpose of user-driven pipeline evaluation, our goal is to understand the effects of user perceived feelings of Accomplishment, Effort, Mental Demand, Controllability, and Temporal Demand. Our main motivation is to understand the following:

- What number of interactions (more or less) is better for a user's perceived overall satisfaction?
- Which set of operators are required to achieve user perceived satisfactory results?

Two factors are considered, namely *exploration operators* (a set of all operators S vs S' with one less operator) and *interventions* allowed for the user (upper bound: N/2 times vs N times). Refer to Section 1.3.2 for a definition of exploration operators. All other factors are kept constant for all experiments, namely dataset, AI algorithm, exploration platform, dimensions to group on etc. The use case under study is to find various types of galaxies from the SDSS dataset. At the end of the session, the user feedback is collected from a questionnaire designed to record the effects of user perceived feelings of *Accomplishment*, *Effort*, *Mental Demand*, *Controllability*, and *Temporal Demand*. The data from this questionnaire will be used to create split plots that will allow us to give insights into which

operators work better for subsetting a given data set, in order for the user to properly explore and reach the final dataset.

2 INODE-SQL 2.0 In Action

2.1 OpenDataDialog

In this section we will demonstrate the services of INODE-SQL 2.0 based on the two datasets of CORDIS (policy research) and SDSS (astrophysics). We adapt to the typical flow of the web application: Starting from scratch, the user enters a natural language query which is interpreted by up to three NL-to-SQL operators. As multiple SQL translations result from this, the user then assesses the multiple results by investigating the data with the Multi-Table Explorer, which includes SQL-to-NL translations of the results. Then, the user continues to explore the data with the help of the by-recommendation operators as well as the new pipeline operators.

2.1.1 NL-to-SQL: Translating Natural Language Questions to SQL

NL-to-SQL has been enhanced in this release. We first demonstrate ValueNet on the CORDIS use case and afterwards demonstrate SODA's capabilities on the astrophysics use case.

2.1.1.1 Querying CORDIS in NL with ValueNet

For this release we apply ValueNet to the CORDIS data. As ValueNet requires a GPU-enabled infrastructure to run its large neural network, it is hosted outside of INODE-SQL 2.0 in the ZHAW GPU Cloud. Due to the robust API design of INODE-SQL 2.0, integration of ValueNet as an outside REST service did not require any additional effort.

ValueNet can now be selected as a new system in the INODE OpenDataDialog 2.0 settings (see Figure 2.1).

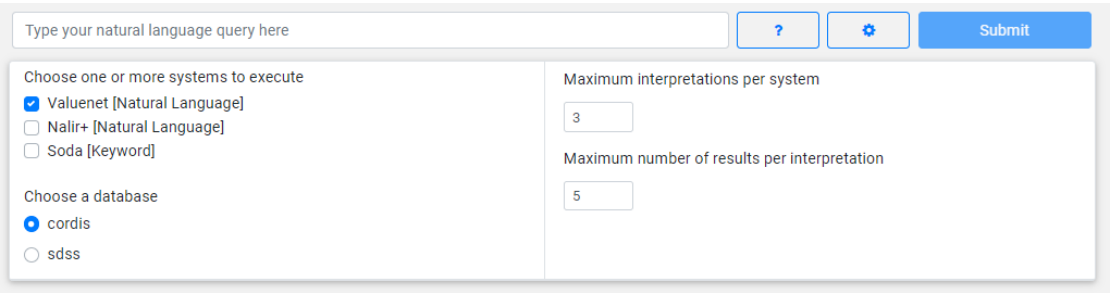


Figure 2.1: The landing page of INODE-SQL 2.0 shows ValueNet as NL-to-SQL translation system for the CORDIS database.

We start exploring ValueNet by asking the question *"Find projects that started before 2016"*. ValueNet synthesizes the correct SQL "`SELECT * FROM projects AS T1 WHERE T1.start_year < 2016`" and the INODE OpenDataDialog displays the data as expected (see Figure 2.2).

While this question is not especially complex, it still requires a certain knowledge to synthesize the correct SQL: The word *"started"* is referring to the column *"`start_year`"*, *"before"* should get translated to the "<" operator and the simple mentioning of "projects" is referring to *"`SELECT *`"*. All this knowledge is learned by ValueNet from scratch during training on a publicly available data corpus.

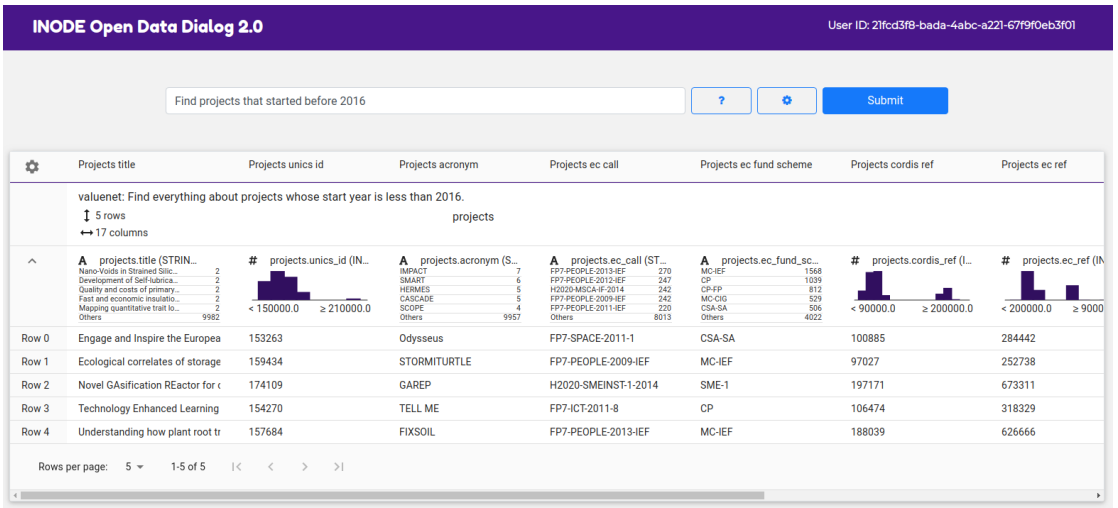


Figure 2.2: ValueNet interpreting the question "Find projects that started before 2016" on the CORDIS database.

The next question we ask ValueNet is *"Show the name of members in projects costing less than the average project cost*." (see Figure 2.3) which ValueNet synthesizes correctly into:

```
SELECT T1.member_name
FROM project_members AS T1
     JOIN projects AS T2 ON T1.project = T2.unics_id
WHERE T2.total_cost < (SELECT AVG(T23.total_cost) FROM
projects AS T23);
```

This query has a much higher complexity than the first one: it contains a JOIN between two tables, an aggregation function (`AVG`) on a numeric column and a nested query which is indeed necessary for the question at hand.

It is interesting to see that ValueNet correctly understands the sub-sentence *"...costing less than the average"*. It has learned this knowledge from somewhat similar questions on different databases and is now able to correctly generalize on the CORDIS schema.

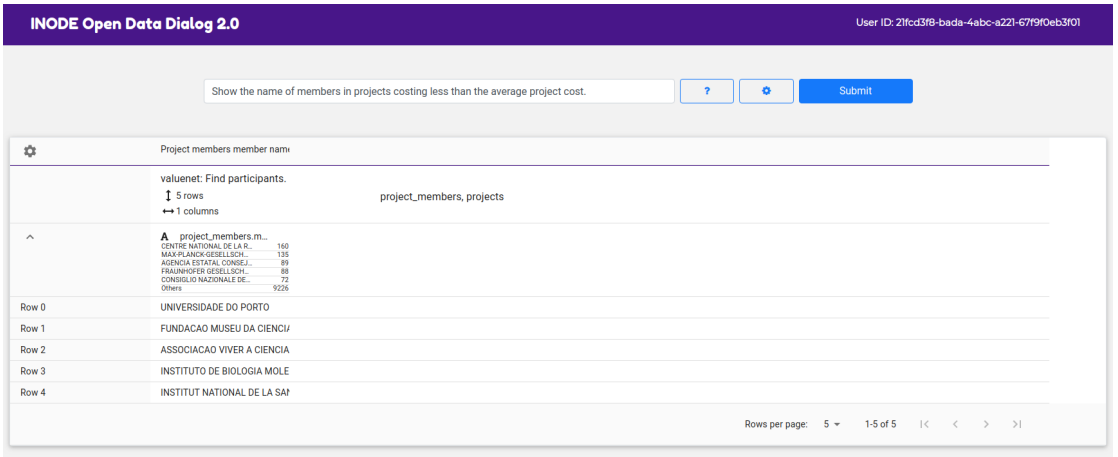


Figure 2.3: ValueNet interpreting the question "Show the name of members in projects costing less than the average project cost." on the CORDIS database.

The last example we test ValueNet on is *"Show me the acronym of projects with a duration of more than 5 years."*. ValueNet incorrectly synthesizes the query:

```
SELECT T1.acronym FROM projects AS T1 WHERE T1.end_year > 5
```

(see Figure 2.4).

When analyzing the query we see it is correct except for the fact that *"duration"* got translated to "`end_year`". Here, the model lacks the knowledge that "duration" represents a time span, which needs to be calculated by subtracting the start from the end year. We expect the model to be able to synthesize such queries correctly by adding more explicit training data or by explicitly enriching the ontology with concepts such as duration.

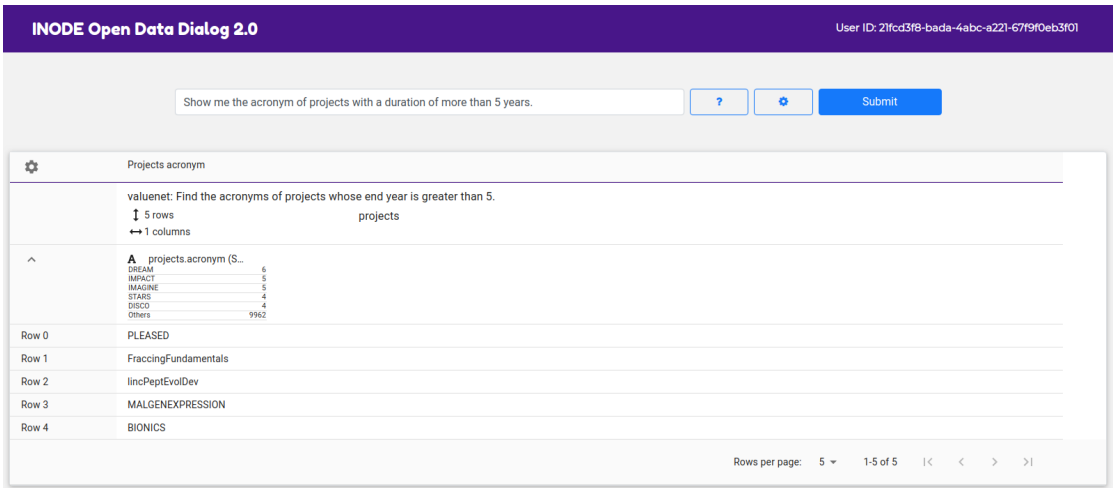


Figure 2.4: ValueNet interpreting the question *"Show me the acronym of projects with a duration of more than 5 years."* on the CORDIS database.

2.1.1.2 Querying SDSS in NL with SODA

For INODE-release 1.0, SODA was applied to the CORDIS data. For the latest release, i.e., INODE 2.0, SODA has been configured to work with the *SDSS* data. Due to our stable API

design, no new functionality was needed to add to SODA for this software release. The landing page of SODA configured with SDSS is shown in Figure 2.5.

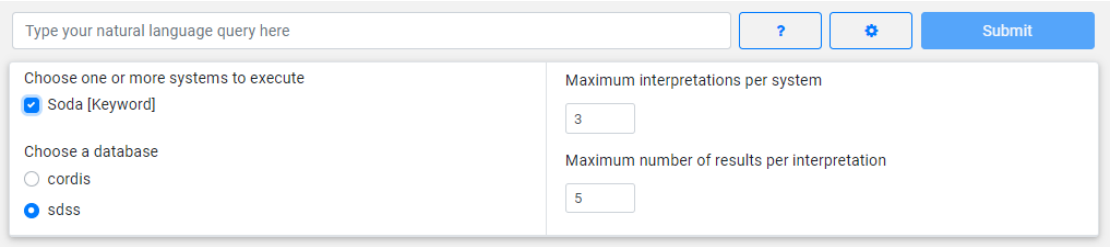


Figure 2.5: The landing page of INODE-SQL 2.0 shows SODA as an NL-to-SQL translation system and the newly added SDSS database.

For the following natural language question "Show all objects in photoobj", SODA returns the correct SQL query, "`SELECT * FROM photoobj`" (see Figure 2.6).

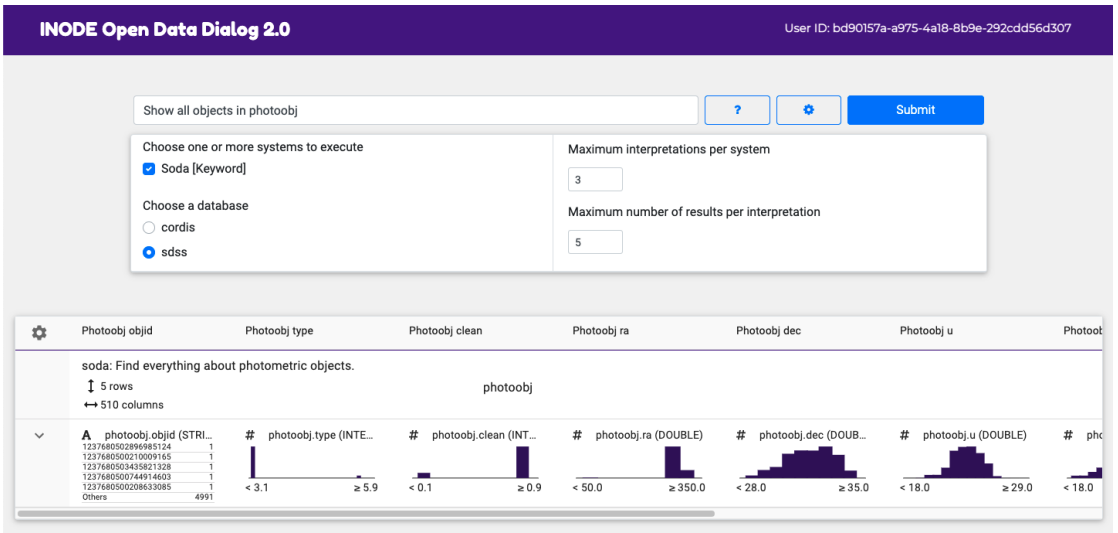


Figure 2.6: The landing page of INODE-SQL 2.0 shows the NL query "Show all objects in photoobj" for the SDSS database as well as the SQL interpretation returned by SODA.

The following natural language question "Find all objects with right ascension greater than 100" can be answered by SODA in a keyword question format, "`ra > 100 photoobj`". SODA returns the correct query in the list of possible interpretations (see Figure 2.7).

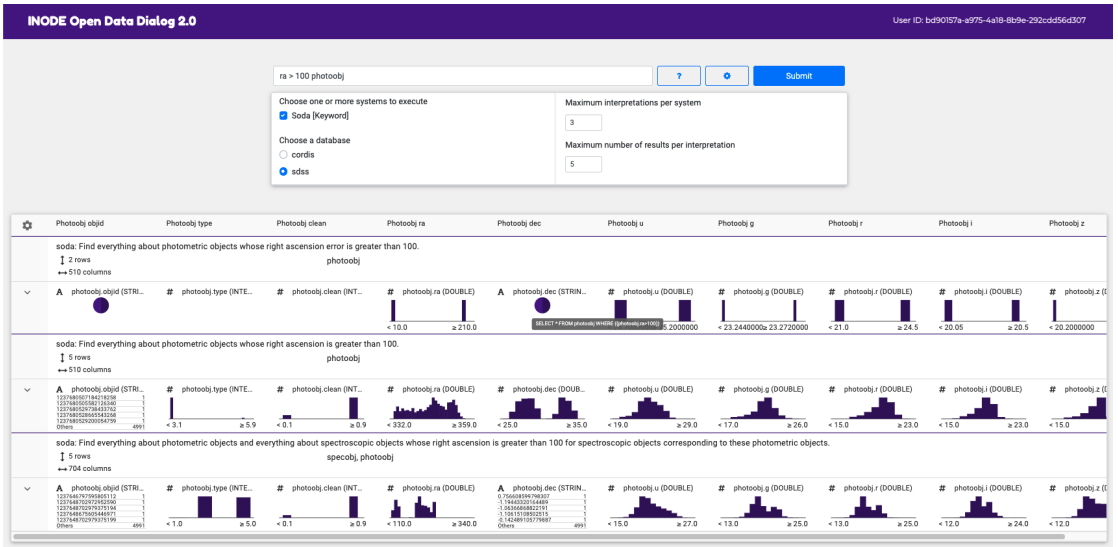


Figure 2.7: SODA keyword query "`ra > 100 photoobj`" for SDSS database.

From all of the information, which SODA took as input in the previous two examples, it was able to produce correct SQL queries. However, SODA currently cannot handle queries that contain information from two or more tables that have columns of the same name, which is common across the 5 tables of the SDSS database. For example "Find all photometrically observed galaxies with right ascension greater than 100 and declination less than 100", which corresponds to the following SODA keyword query "`Photoobj ra > 100 dec < 100`". Here, SODA does not return the correct query, which should only contain elements from a single table, `photoobj`, rather than queries with joins to another table in the database (see Figure 2.8).

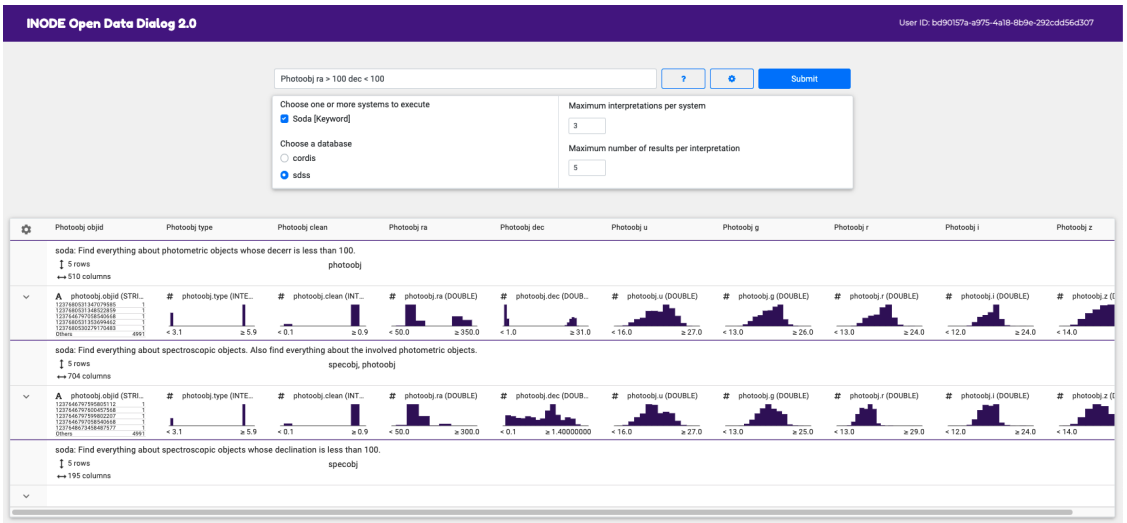


Figure 2.8: Incorrectly returned interpretation for the keyword query "`Photoobj ra > 100 dec < 100`". In the upper result, the NL explanation provides a valuable hint why this might be the case: "... whose decerr is less than 100". Furthermore, the histograms of column 4 (photoobj ra) indicate that the "`ra > 100`" condition has been ignored.

In order to mitigate this problem, the domain ontology of the SDSS dataset will be enriched with additional semantic information about objects in the sky.

2.1.2 SQL-to-NL: Explaining SQL Queries Using Natural Language

Logos has been improved to generate better explanations for the CORDIS data as we explained in Section 1.4.1, compared to the earlier version of INODE. In the current version, we have enabled explanations for SDSS queries. Furthermore, NL explanations now appear in several places on the UI (a) to help the user understand which queries were generated for a user NL query as well as (b) to explain the queries executed by the pipeline operators, hence offering a better user experience.

We will show examples of how it works with SDSS queries.

Let us assume that we want to address the query "Find all photometrically observed stars" against the SDSS database. By clicking the gear button (see Figure 2.5) , a menu appears that enables the user to choose between the available NL to SQL translation systems and databases (CORDIS, SDSS), as well as to pick the desired number of interpretations and results per interpretation (tuples). For the SDSS database, the available system is SODA.

In addition, let us say that we want only two interpretations produced by the system SODA and only 10 tuples to be retrieved for these produced queries. We continue by typing the NL query we want to submit, in this case "photoobj type = 6" where type 6 corresponds to photometric objects which are stars.

Once submitted, the interpretations returned by the INODE system are those depicted in Figure 2.9. Each interpretation refers to a different query produced by the SODA system. One of those queries is possibly the one we are looking for. Certainly, that query is the first one "Find everything about photometric objects whose object type classification is 6".

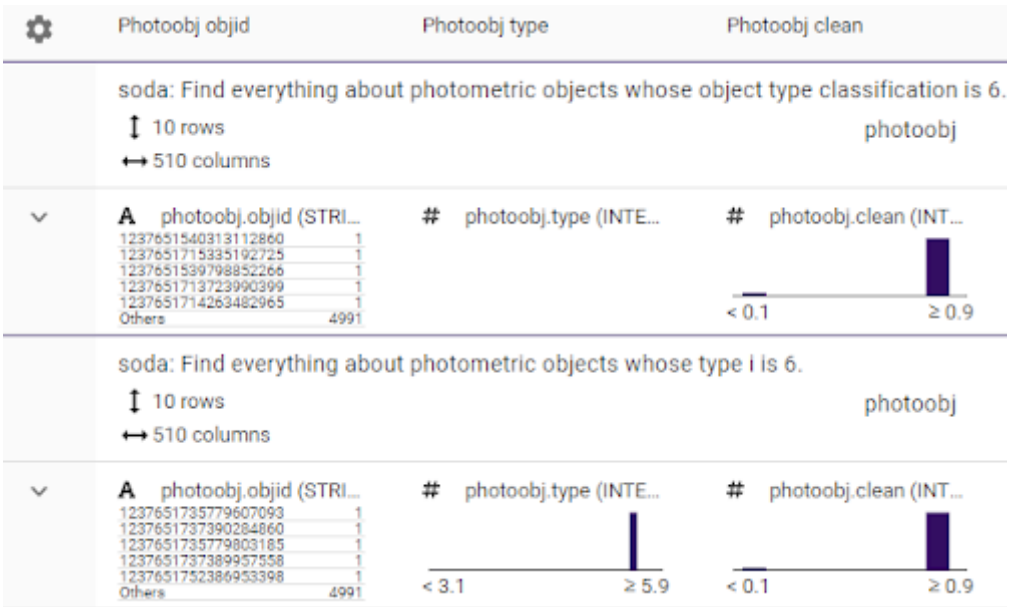


Figure 2.9: Returned interpretations.

One may ask more about each of the produced interpretations. For instance, let us say that we want to explore more about the query under study (first query retrieved by the system). The only thing we have to do is to press the pointing-down arrow and afterwards continue by pressing the three dots button. A menu showing the available exploration operators is shown to the user (see Figure 2.10).

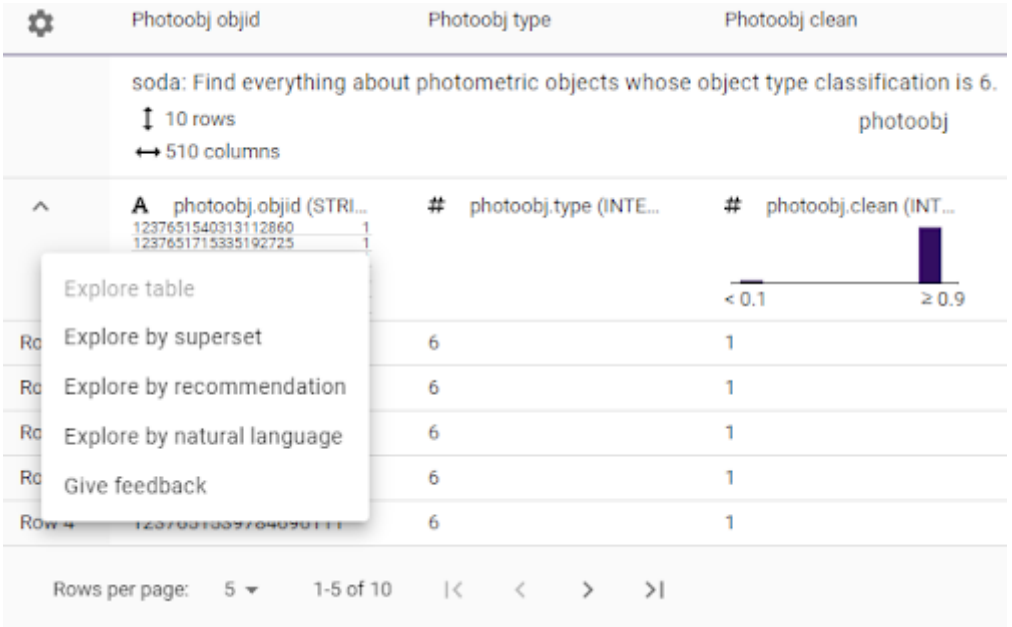


Figure 2.10: Operators menu.

For any pipeline operator (e.g., by-recommendation, etc.) the process is the same. Assuming that we want to explore by superset, we press the corresponding button and we get the following explained query (see Figure 2.11).

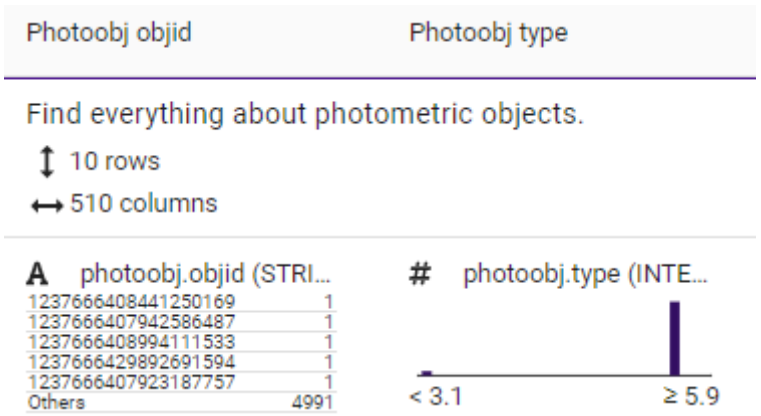


Figure 2.11: Interpretation returned using the by-superset operator.

Undoubtedly, the set containing all the photometric objects is a superset of the set containing only the photometric objects that are stars.

2.1.3 Recommending Queries (PyExplore)

The figures below, show where the query recommendations appear on the INODE user interface. Pressing the pointing-down arrow and the three dots button, we can select from the menu the option to see recommendations based on data correlations that are generated by PyExplore.

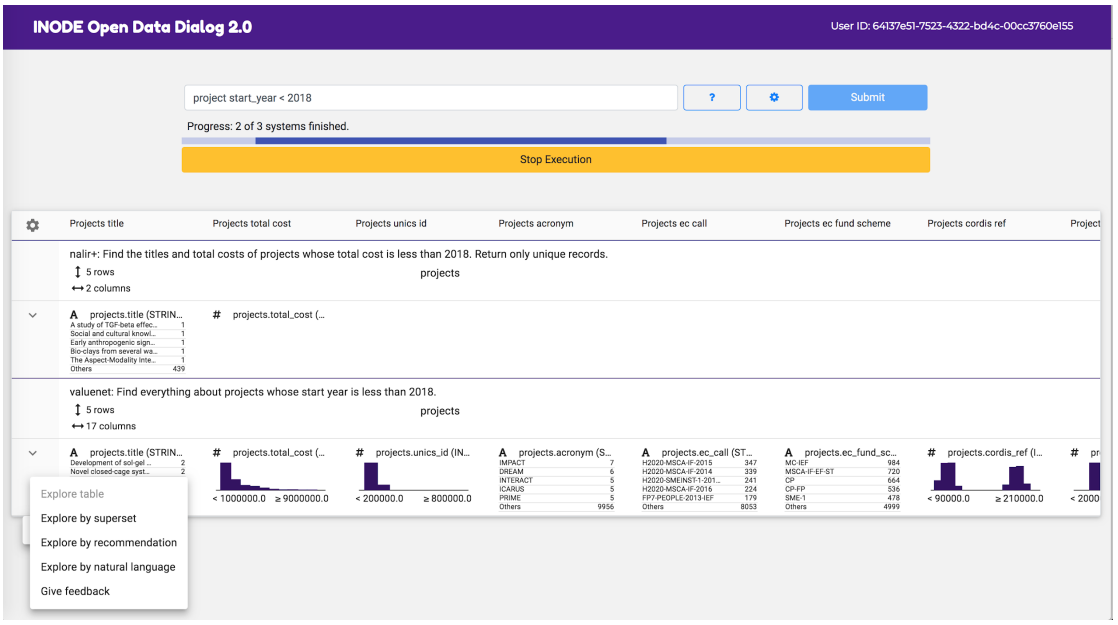


Figure 2.12: Selecting the by-recommendation operator.

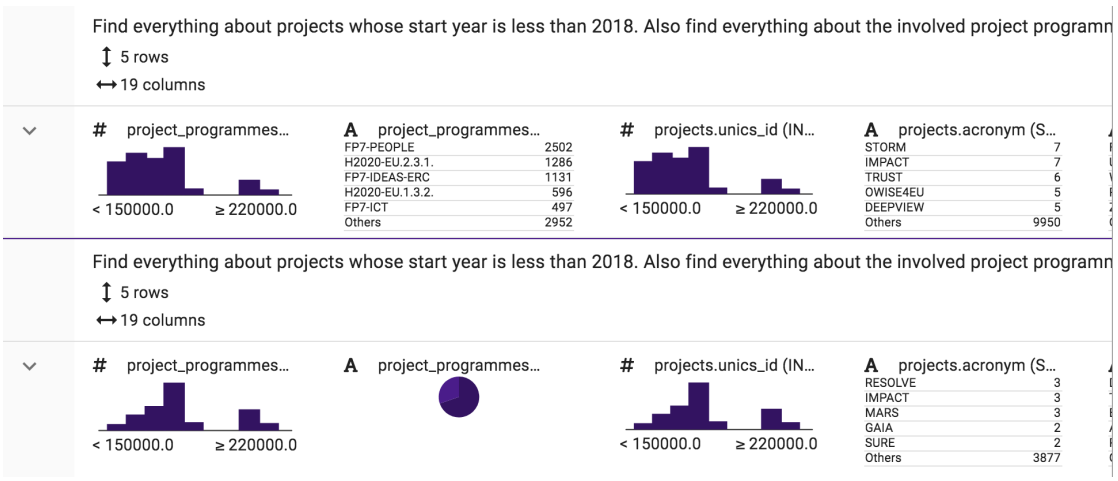


Figure 2.13: Recommended queries for projects with start year less than 2018.

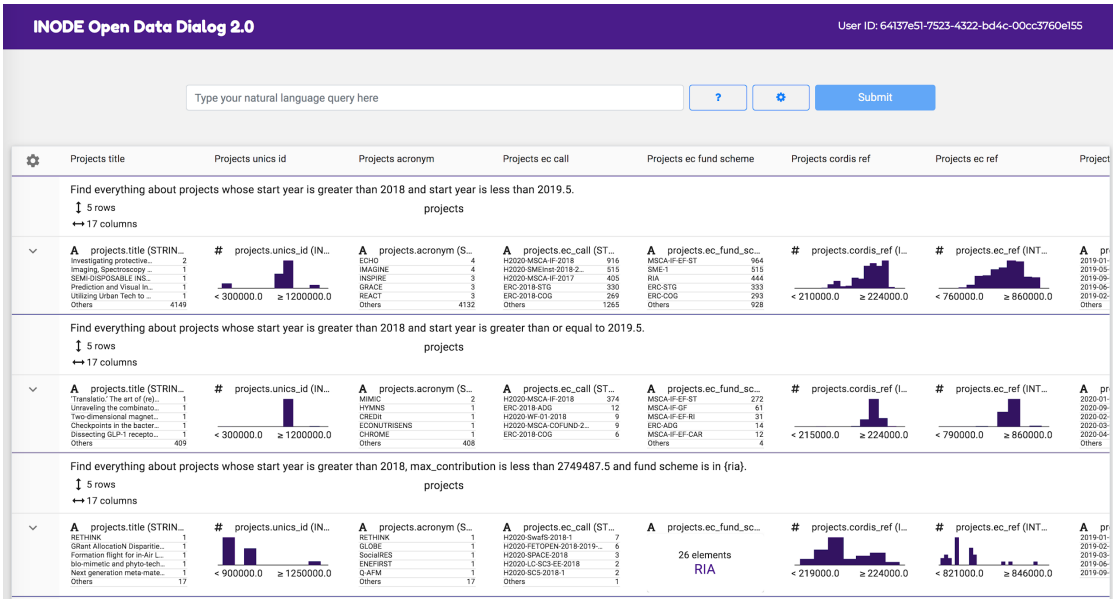


Figure 2.14: Recommended queries for projects with start year greater than 2018.

2.1.4 Pipeline Operators

The following screenshots are from the "Galaxy exploration app", since the operators could not be integrated in the current OpenDataDialog 2.0 release. First steps towards integration have been undertaken however, for example the support to image URL data types as seen in Section 1.4.

In this app, the data is composed of 2.6M galaxies with 10 continuous numerical attributes binned in 10 equal depth bins.

Those attributes have an intrinsic order allowing us to use by-neighbors and by-distribution:

- **By-neighbors**

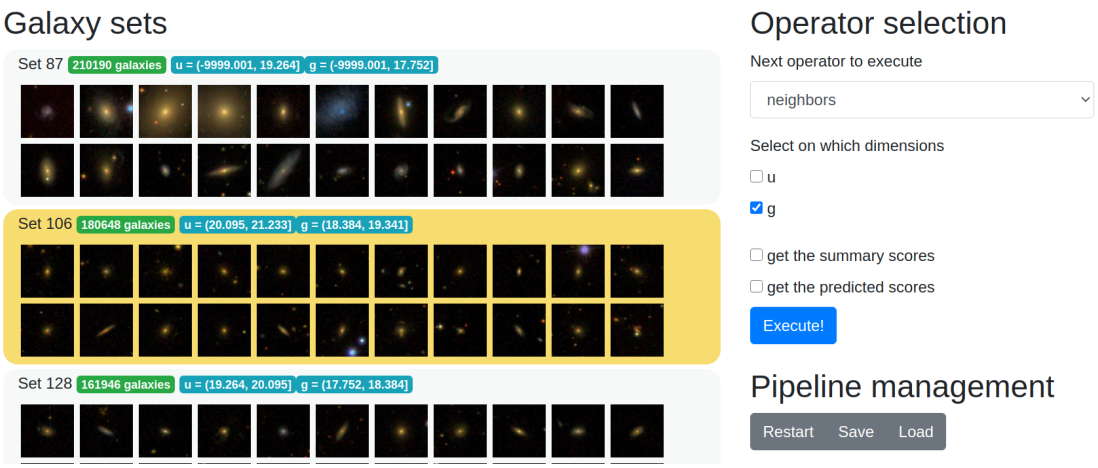


Figure 2.15: Selection of the input subset (on the left hand side), the by-neighbors operator (in the drop-down menu on the right side), and the dimension to use (magnitude g).

The astrophysicist using the application finds a set of interests during their exploration. The set is described by two ranges of values: on the magnitude (on the spectrum range) 'u' the values are filtered between 20.095 and 21.233, and on the magnitude 'g' between 18.384 and 19.341. The scientist would like to see sets with the same range of values on the magnitude 'u', but slightly different values on the magnitude 'g'.

They select the set, the by-neighbors operator and the magnitude 'g', and obtains the following results:

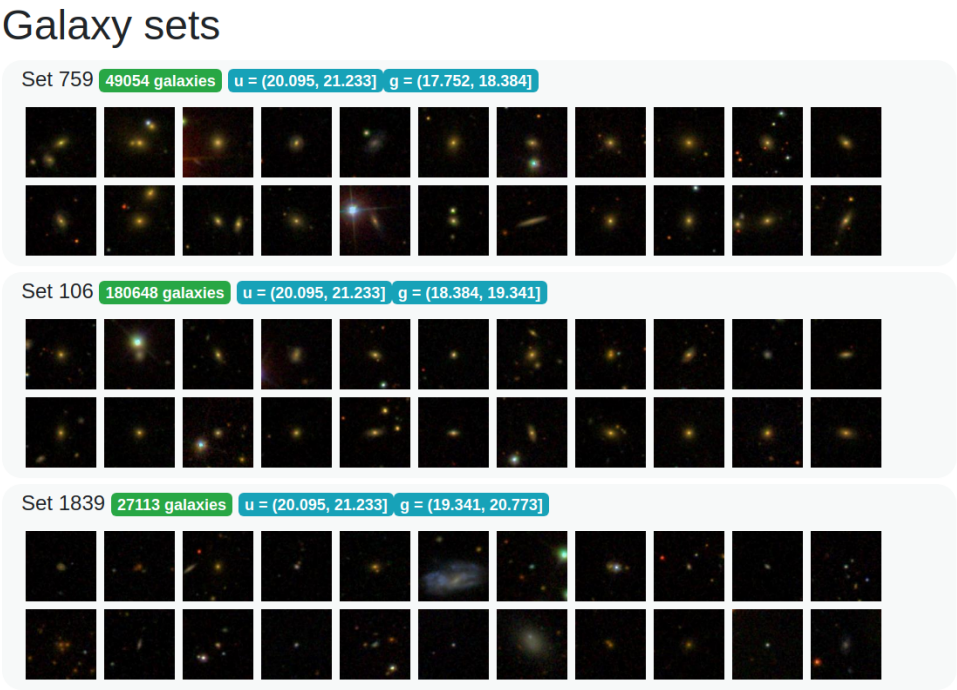


Figure 2.16: The resulting sets, the input set in the middle, the set on the lower range on the magnitude 'g' above, and the set on the higher range on the magnitude 'g' below.

The input set, plus two sets sharing the same range on the magnitude 'u', with the ranges below and above on the magnitude 'g':

- 17.752 -> 18.384
- 19.341 -> 20.773

- **By-distribution**

The astrophysicist using the application finds a set of interests during their exploration. The set is described by three ranges of values, on the magnitude (on the spectrum range) 'u' the values are filtered between 21.233 and 22.228, on the magnitude 'g' between 18.384 and 19.341, and on the magnitude 'i' between 17.063 and 17.545. They find the relations between the three magnitudes in this set interesting, and would like to see more sharing the same relations to see if an interesting pattern appears.

Figure 2.17: The input set selected with the three magnitude ranges.

They select the set and the by-distribution operator, and obtain the following results:

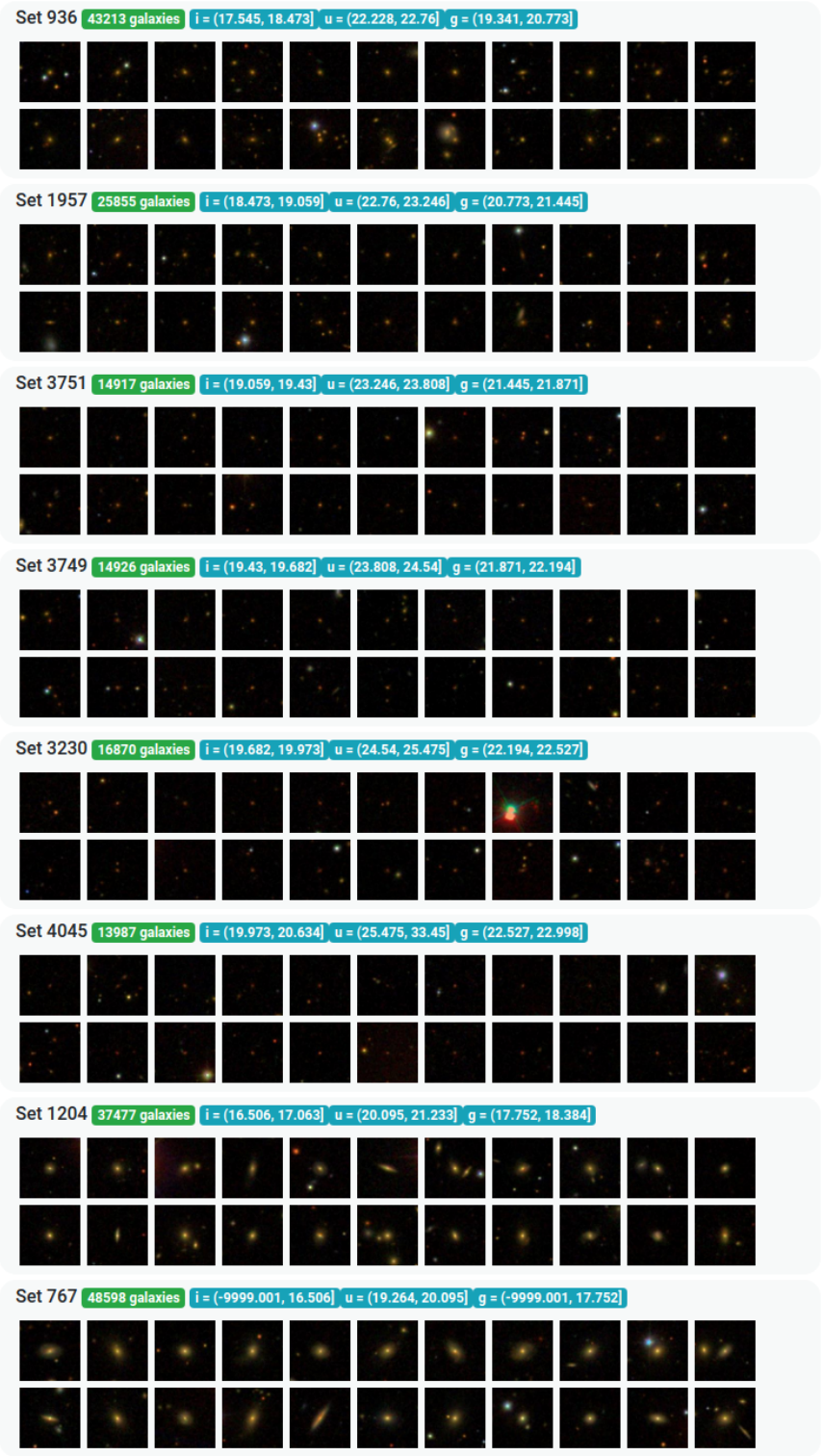


Figure 2.18: The sets resulting from the use of the by-distribution operator.

The ranges describing the input set in the magnitudes u, g, and i were the third on u, the second on g and the second on i. So we can simplify it as the following description: [3,2,2].

By-distribution has returned all the sets keeping the same difference between the range indexes.

Hence the resulting sets [1,0,0], [2,1,1], [4,3,3], [5,4,4], [6,5,5], [7,6,6] , [8,7,7], and [9,8,8] provide a wide range of very different galaxies to study.

2.1.5 Summary

We have demonstrated many of the new features within OpenDataDialog 2.0. It became apparent that all operators, despite their complexity and variety, can be used from within one single web application.

2.2 OpenDataLinking

In this section we will demonstrate the OpenDataLinking services of INODE-SQL 2.0 based on several open datasets as well as on cancer research (OncoMX).

The OpenDataLinking component of INODE-SQL 2.0 introduces a set of *improvements on the triple extraction process* as well as a *unified extraction approach* that relies on a triple refinement algorithm implemented by ZHAW to efficiently consolidate the extracted triples from both the ZHAW and the Infili engines. Our system encompasses a set of information extraction methods to distill structured knowledge from unstructured text by identifying references to named entities as well as stated relationships between such entities.

2.2.1 Triple Refinement

The need for a triple refinement method originated from the fact that a simple combination of the outputs stemming from the *precision-oriented ZHAW approach* and the *recall-oriented Infili approach* did not automatically yield optimal results. The triple refinement method is based on a reverse implementation of the ZHAW dependency parsing approach for triple extraction and can be explained with the following example from the CaRB development set[24]:

```
Instead of having system calls specifically for process management,
Plan 9 provides the codice_13 file system.
```

A triple output by the Infili engine is as follows:

```
Plan 9 ; provides ; the codice_13 file system Instead of having system
calls specifically for process management
```

[24] Sangnie Bhardwaj, Samarth Aggarwal, and Mausam. CARB: A crowdsourced benchmark for open IE. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2020.

This triple, while scoring well on recall, has low precision due to the unnecessary clause `Instead of having system calls specifically for process management.`

On the other hand, the ZHAW engine often omits messy triples (i.e., triples stemming from complex sentences, containing one or more uninformative/unnecessary clauses) entirely, in order to preserve precision. By feeding the Infili triples back into the ZHAW engine, we can gain new triples, but maintain precision. Hence, the idea of the combined system is to *increase both the recall and the precision of the triple extraction process*.

The refinement algorithm marks the tokens from an extracted triple on the dependency tree for that sentence. It then searches for the three head-words of the subject, predicate, and object. These head-words are the roots of the largest connected components of the subject, predicate, and object sub-trees. The three head-words are then fed into the ZHAW pipeline to produce a modified triple.

In contrast, the ZHAW engine searches for these head-words itself, before expanding them into full triples. However, since the ZHAW engine uses high-precision rules, many head-words are not found with this method, and we leverage the high-recall Infili engine to find new triples, then refine them. An example of this is shown in Figure 2.19.

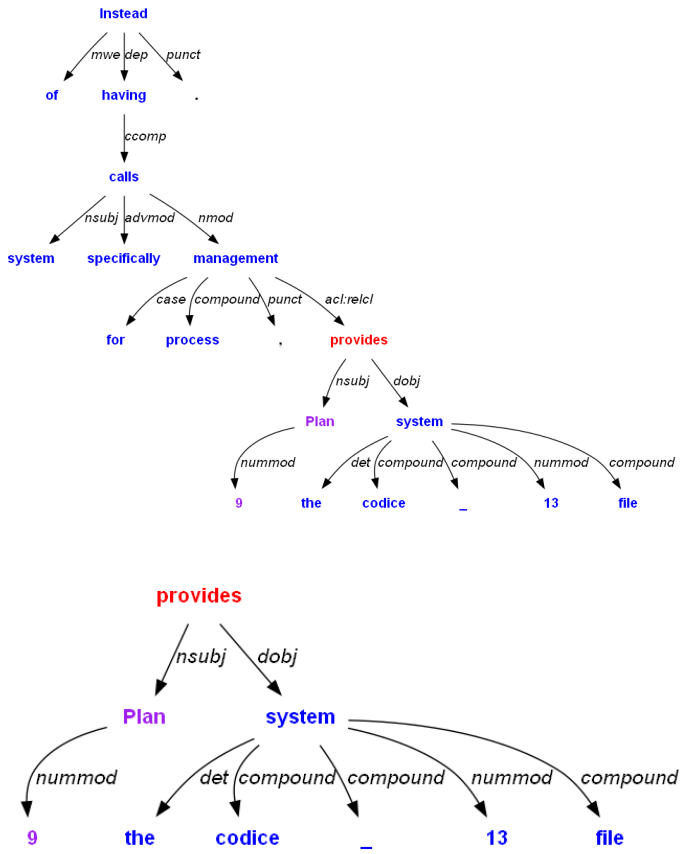


Figure 2.19: Original dependency tree with old triple marked, and pruned dependency tree with new refined triple marked.

The new triple is now:

```
(Plan 9 ; provides ; the codice_13 file system)
```

Compared to the output from the Infili engine, the subordinate clause `Instead of having system calls specifically for process management` is discarded by the triple refinement process.

Additional improvements were also implemented to ensure the seamless integration of the triple refinement system with the OncoMX data. More specifically, the following settings have been added to the ZHAW triple extractor and combiner, to adapt it to various use cases:

- `enhanced_predicates`: When this setting is enabled, the predicate contains additional descriptive information, and the subject and object are simplified. This is better suited to the OncoMX use case, since the predicate remains in plain text, while the subject and object are mapped to ontologies. For example in the sentence:

  ```
  Blood E2F3 mRNA levels were significantly higher in lung cancer patients when compared to either patients with benign lung diseases or healthy subjects.
  ```

  The enhanced_predicates version is as follows:

  ```
  (Blood E2F3 mRNA levels ; were significantly higher in ;
  lung cancer patients)
  ```

  While the standard version keeps the shorter extracted predicate:

  ```
  (Blood E2F3 mRNA levels ; were ; significantly higher in lung
  cancer patients)
  ```

- `entity_context`: This setting provides additional contextual (temporal, location, etc.) information for the subject and object of each triple. For example in the aforementioned sentence, we can produce triples with an additional field dedicated to such information, as shown below:

  ```
  (Blood E2F3 mRNA levels ; were significantly higher in ;  lung
  cancer patients ; when compared to either patients with benign
  lung diseases or healthy subjects)
  ```

- `split_triples`: This setting allows for splitting or merging conjunctive phrases into sub-triples depending on the current needs. For example, in the following sentence:

  ```
  Long non-coding RNA CCAT2 plays an important role in
  tumorigenesis, tumor growth and metastasis.
  ```

  We can extract three separated triples for each sub-entity:

  ```
  (Long non-coding RNA LncRNA CCAT2 ; plays ; an important role in
  tumor growth)
  (Long non-coding RNA LncRNA CCAT2 ; plays ; an important role in
  metastasis)
  ```

```
(Long non-coding RNA LncRNA CCAT2 ; plays ; an important role in
tumorigenesis)
```

Or merge them into a single entity:

```
(Long non-coding RNA LncRNA CCAT2 ; plays ; an important role in
tumorigenesis tumor growth and metastasis)
```

2.3 Integration of OpenDataLinking with OpenDataDialog

This section aims at showcasing the *information extraction as well as entity linking capabilities* (i.e., the aligning of textual mentions of named entities to their corresponding entries in a knowledge base) of the INODE-SQL 2.0 system as an integrated platform. To this end, we provide a set of preliminary SQL queries, their natural language explanation, and the expected query result, both for the OncoMX dataset (biomarkers use case) and for the CORDIS dataset. It should be noted that the NL-to-SQL functionality is not yet implemented for the Biomarkers Use Case. However, the following examples serve as an early demonstration of the final INODE system.

Biomarkers Use Case

SQL Query 1	`SELECT distinct gene, uberonname, uberon`
	`FROM triples_fully_linked_v2`
	`WHERE`
	`(predicate like '%overexpress%' or`
	`(predicate like '%express%' and (subject like '%over%' or`
	`triples_fully_linked_v2.object like '%over%')) or`
	`(subject like '%overexpress%' or triples_fully_linked_v2.object like`
	`'%overexpress%')) and`
	`(subject like '%cancer%' or triples_fully_linked_v2.object like`
	`'%cancer%') and polarity='TRUE'`

NL equivalent	Find all anatomic entities where genes are overexpressed due to some cancer reported in the literature.
Explanation	As described in D3.1, one of the goals of the OpenDataLinking component was the extraction of triples from NL-text of cancer-related Pubmed articles. The triples were linked to existing concepts (anatomical entities) of the Uberon ontology and to genes of the OncoMX database upon extraction, then added to the latest version of the OncoMX database. Based on this, we are now able to find all literature cases (derived from triple extraction on Pubmed articles) that include "over-expression" of a gene on a human body part with cancer. "Over-expression" synonyms such as "increased expression" are not considered in this case.

Result (latest version- v2)

#	gene	uberonname	uberon
1	AMACR	prostate gland	UBERON:0002367
2	NR4A1	breast	UBERON:0000310
3	MDK	lung	UBERON:0002048
4	HMGB1	lung	UBERON:0002048
5	LAPTM4B	lung	UBERON:0002048
6	ERBB2	cutaneous appendage	UBERON:0000021
7	RHOA	breast	UBERON:0000310
8	GPI	prostate gland	UBERON:0002367
9	IGF2	tissue	UBERON:0000479
10	AGR2	prostate gland	UBERON:0002367
11	ECM1	tissue	UBERON:0000479
12	EGFR	lung	UBERON:0002048
13	EPCAM	prostate gland	UBERON:0002367
14	SLPI	tissue	UBERON:0000479
15	KLK6	tissue	UBERON:0000479
16	EGFR	breast	UBERON:0000310
17	EGFR	colon	UBERON:0001155
18	CD24	breast	UBERON:0000310
19	HSP90AB1	lung	UBERON:0002048
20	CD59	lung	UBERON:0002048
21	TNF	adult cerebral ganglion	UBERON:6110636
22	GDNF	tissue	UBERON:0000479
23	AGR2	urine	UBERON:0001088
24	RET	breast	UBERON:0000310
25	IGFBP2	lung	UBERON:0002048
26	TMPRSS2	prostate gland	UBERON:0002367
27	TSPAN8	tissue	UBERON:0000479
28	CDH3	adult organism	UBERON:0007023
29	GPI	posterior communicating artery	UBERON:0001628
30	MET	cutaneous appendage	UBERON:0000021
31	EGFR	thyroid gland	UBERON:0002046
32	MUC1	breast	UBERON:0000310
33	EGFR	prostate gland	UBERON:0002367
34	MET	lung	UBERON:0002048
35	ERG	prostate gland	UBERON:0002367
36	HMGB1	tissue	UBERON:0000479
37	PSCA	prostate gland	UBERON:0002367
38	CXCL13	blood	UBERON:0000178
39	TFF3	prostate gland	UBERON:0002367
40	CDH3	tissue	UBERON:0000479
41	FYN	prostate gland	UBERON:0002367
42	ERBB2	breast	UBERON:0000310
43	CXCL13	breast	UBERON:0000310
44	CEACAM5	colon	UBERON:0001155
45	EGFR	tissue	UBERON:0000479
46	MALAT1	liver	UBERON:0002107
47	PSAT1	colon	UBERON:0001155
48	MYC	prostate gland	UBERON:0002367
49	CYP1B1	prostate gland	UBERON:0002367
50	ECM1	breast	UBERON:0000310
51	RAB5A	breast	UBERON:0000310
52	SLPI	colon	UBERON:0001155
53	RAC1	tissue	UBERON:0000479
54	L1CAM	tissue	UBERON:0000479
55	TSPAN8	lung	UBERON:0002048
56	TNF	adult organism	UBERON:0007023
57	IGFBP2	tissue	UBERON:0000479
58	PSCA	posterior communicating artery	UBERON:0001628
59	EGFR	head	UBERON:0000033
60	EGFR	neck	UBERON:0000974
61	MIF	prostate gland	UBERON:0002367
62	S100A4	lung	UBERON:0002048
63	TNF	adult brain	UBERON:6003624
64	HMGB1	blood serum	UBERON:0001977
65	MET	colon	UBERON:0001155
66	CCND1	breast	UBERON:0000310
67	MUC1	posterior communicating artery	UBERON:0001628
68	EPCAM	breast	UBERON:0000310
69	EGFR	integument	UBERON:0002199
70	UBQLN1	breast	UBERON:0000310

Result (older-v1)

#	gene	uberonname	uberon
1	MTA1	lung	UBERON:0002048
2	TSPAN13	tissue	UBERON:0000479
3	LPIN1	breast	UBERON:0000310
4	HMGB1	tissue	UBERON:0000479
5	AMFR	breast	UBERON:0000310
6	RHOA	breast	UBERON:0000310
7	RAC1	breast	UBERON:0000310
8	ERBB3	breast	UBERON:0000310
9	ERBB2	breast	UBERON:0000310
10	SOX2	lung	UBERON:0002048
11	AMACR	breast	UBERON:0000310
12	EGFR	lung	UBERON:0002048
13	ERBB2	colon	UBERON:0001155
14	EGFR	tissue	UBERON:0000479
15	CEACAM5	colon	UBERON:0001155
16	EGFR	breast	UBERON:0000310
17	EGFR	colon	UBERON:0001155
18	SLPI	colon	UBERON:0001155
19	SOX2	breast	UBERON:0000310
20	MUC1	primary somatosensory cortex	UBERON:0008933
21	HMGB1	primary somatosensory cortex	UBERON:0008933
22	EGFR	head	UBERON:0000033
23	EGFR	neck	UBERON:0000974
24	TFF3	tissue	UBERON:0000479
25	EGFR	primary somatosensory cortex	UBERON:0008933
26	ERBB2	pancreas	UBERON:0001264
27	PRMT1	lung	UBERON:0002048
28	CCND1	breast	UBERON:0000310
29	STAT1	neck	UBERON:0000974
30	STAT1	head	UBERON:0000033
31	ERBB2	gland	UBERON:0002530
32	MET	breast	UBERON:0000310
33	MUC1	breast	UBERON:0000310
34	MYC	breast	UBERON:0000310
35	MET	lung	UBERON:0002048

Improve-ments over v1

The efficient combination of the two OIE engines' outputs by leveraging the triple refinement approach introduced in v2, leads to the doubling of the extracted triples that are linked with gene overexpression due to cancer. Hence, the new version increases the *recall* for triple extraction.

SQL Query 2

```
SELECT distinct gene, uberonname, uberon
FROM triples_fully_linked_v2
WHERE
(predicate like '%overexpress%' or
(predicate like '%express%' and (subject like '%over%' or
triples_fully_linked_v2.object like '%over%')) or
(subject like '%overexpress%' or triples_fully_linked_v2.object like
'%overexpress%')) and
(subject like '%cancer%' or triples_fully_linked_v2.object like
'%cancer%') and
polarity='TRUE' and uberonname='breast'
```

NL equivalent

What are the genes overexpressed in breast cancer reported in the literature?

Explanation

Similar to the above query, we again exploit the extraction of triples from the literature (PubMed articles) and their mapping to genes and anatomical entities. This time, however, we specialize our query on finding all literature cases that include "over-expression" of a gene, specifically on breast cancer.

<table>
<tr><td></td><td>Once again, “over-expression” synonyms such as “increased expression” are not considered.</td></tr>
<tr><td>Result (latest version- v2)</td><td>
<table>
<tr><th>#</th><th>gene</th><th>uberonname</th><th>uberon</th></tr>
<tr><td>1</td><td>EGFR</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>2</td><td>EPCAM</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>3</td><td>CD24</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>4</td><td>NR4A1</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>5</td><td>ECM1</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>6</td><td>RAB5A</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>7</td><td>CXCL13</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>8</td><td>UBQLN1</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>9</td><td>MUC1</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>10</td><td>RET</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>11</td><td>RHOA</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>12</td><td>ERBB2</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>13</td><td>CCND1</td><td>breast</td><td>UBERON:0000310</td></tr>
</table>
</td></tr>
<tr><td>Result (older-v1)</td><td>
<table>
<tr><th>#</th><th>gene</th><th>uberonname</th><th>uberon</th></tr>
<tr><td>1</td><td>EGFR</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>2</td><td>SOX2</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>3</td><td>LPIN1</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>4</td><td>CCND1</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>5</td><td>AMACR</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>6</td><td>AMFR</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>7</td><td>MET</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>8</td><td>MUC1</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>9</td><td>MYC</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>10</td><td>RHOA</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>11</td><td>RAC1</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>12</td><td>ERBB3</td><td>breast</td><td>UBERON:0000310</td></tr>
<tr><td>13</td><td>ERBB2</td><td>breast</td><td>UBERON:0000310</td></tr>
</table>
</td></tr>
<tr><td>Improve-ments over v1</td><td>While the number of literature extractions remains the same, the latest version provides less false positives (i.e., triples that are incorrectly linked to gene overexpression due to breast cancer). This is also verified in the next example where we compare with the curated gene list. Hence, the new version increases the precision for triple extraction.</td></tr>
</table>

<table>
<tr><td>SQL Query 3</td><td>
<pre>select *
FROM (select distinct gene, uberonname, uberon
FROM triples_fully_linked_v2
WHERE (predicate like '%overexpress%' or
(predicate like '%express%' and (subject like '%over%' or
triples_fully_linked_v2.object like '%over%')) or
(subject like '%overexpress%' or triples_fully_linked_v2.object like
'%overexpress%')) and
(subject like '%cancer%' or triples_fully_linked_v2.object like
'%cancer%') and polarity='TRUE' and uberonname='breast') as t1
LEFT JOIN
(select distinct de.gene_symbol, d.name from differential_expression
as de join disease as d on d.id = de.doid
where d.name = 'breast cancer' and expression_change_direction='up'
and de.statistical_significance = 'Yes') as t2 on
t2.gene_symbol=t1.gene</pre>
</td></tr>
<tr><td>NL equivalent</td><td>Compare breast cancer differentially expressed upregulated genes that are statistically significant with the same genes overexpressed in breast cancer from the literature.</td></tr>
</table>

Explanation

One of the goals of INODE is to enrich existing ontologies by exploiting text mining (triple extraction, entity linking) approaches. In our case, the OncoMX database already contains a curated list of genes that are found to be expressed on different cancer types.Therefore, the purpose of this query is to compare the curated results from the existing *differential_expression* table of OncoMX with literature extractions based on our OIE approach. Only the rows with non-null value columns correspond to validated extractions that were found in the *differential_expression* table (aka the literature extraction confirms the curated mention).

In the results that follow, we can see that some false positives have been recorded by the information extraction engines (with NULL columns). For example, that EGFR is shown in literature to be overexpressed in breast cancer while the curated OncoMX cancer differential expression dataset states as being down regulated. This contradiction is also the case of other information extraction approaches used by domain experts. Our hypothesis is that the contradictory results depend on the experiment conditions and conclusions performed and reported in different research articles.

Result (latest version- v2)

#	gene	uberonname	uberon	gene_sumbol	name
1	RET	breast	UBERON:0000310	RET	breast cancer
2	CCND1	breast	UBERON:0000310	CCND1	breast cancer
3	UBQLN1	breast	UBERON:0000310	UBQLN1	breast cancer
4	ERBB2	breast	UBERON:0000310	ERBB2	breast cancer
5	CD24	breast	UBERON:0000310	CD24	breast cancer
6	EPCAM	breast	UBERON:0000310	EPCAM	breast cancer
7	ECM1	breast	UBERON:0000310	ECM1	breast cancer
8	MUC1	breast	UBERON:0000310	MUC1	breast cancer
9	CXCL13	breast	UBERON:0000310	CXCL13	breast cancer
10	NR4A1	breast	UBERON:0000310		
11	EGFR	breast	UBERON:0000310		
12	RAB5A	breast	UBERON:0000310		
13	RHOA	breast	UBERON:0000310		

Result (older-v1)

#	gene	uberonname	uberon	gene_sumbol	name
1	CCND1	breast	UBERON:0000310	CCND1	breast cancer
2	RAC1	breast	UBERON:0000310	RAC1	breast cancer
3	SOX2	breast	UBERON:0000310	SOX2	breast cancer
4	ERBB2	breast	UBERON:0000310	ERBB2	breast cancer
5	ERBB3	breast	UBERON:0000310	ERBB3	breast cancer
6	AMACR	breast	UBERON:0000310	AMACR	breast cancer
7	AMFR	breast	UBERON:0000310	AMFR	breast cancer
8	MUC1	breast	UBERON:0000310	MUC1	breast cancer
9	LPIN1	breast	UBERON:0000310		
10	EGFR	breast	UBERON:0000310		
11	RHOA	breast	UBERON:0000310		
12	MYC	breast	UBERON:0000310		
13	MET	breast	UBERON:0000310		

Improvements over v1

We observe that the latest version provides a higher number of verified results compared to v1, as a result of the efficient triple unification approach.

SQL Query 4

```
SELECT *
FROM (select distinct gene, uberonname, uberon
```

<table>
<tr><td></td><td>

```
FROM triples_fully_linked_v2
WHERE (predicate like '%overexpress%' or
(predicate like '%express%' and (subject like '%over%' or
triples_fully_linked_v2.object like '%over%')) or
(subject like '%overexpress%' or triples_fully_linked_v2.object like
'%overexpress%')) and
(subject like '%cancer%' or triples_fully_linked_v2.object like
'%cancer%') and polarity='TRUE' and uberonname='breast') as t1
LEFT JOIN
(select distinct de.gene_symbol, d.name from differential_expression as
de join disease as d on d.id = de.doid  where d.name = 'breast cancer'
and expression_change_direction='down' ) as t2 on
t2.gene_symbol=t1.gene
```

</td></tr>
<tr><td>NL equivalent</td><td>Compare breast cancer differentially expressed downregulated genes with the same genes overexpressed in breast cancer from the literature.</td></tr>
<tr><td>Explana-tion</td><td>The purpose of this query is similar to SQL Query 3. This time we compare the upregulated genes found by information extraction to the downregulated genes of the curated table to discover potential false assignments. Ideally we would like all records to contain NULLs as a result of the JOIN, since a non-null row means that a gene found by information extraction to be upregulated in breast, is actually downregulated according to the curated table.</td></tr>
<tr><td>Result (latest version-v2)</td><td>

#	gene	uberonname	uberon	gene_sumbol	name
1	EGFR	breast	UBERON:(	EGFR	breast cancer
2	NR4A1	breast	UBERON:(	NR4A1	breast cancer
3	RHOA	breast	UBERON:(	RHOA	breast cancer
4	CD24	breast	UBERON:0000310		
5	ECM1	breast	UBERON:0000310		
6	CXCL13	breast	UBERON:0000310		
7	CCND1	breast	UBERON:0000310		
8	RET	breast	UBERON:0000310		
9	MUC1	breast	UBERON:0000310		
10	RAB5A	breast	UBERON:0000310		
11	EPCAM	breast	UBERON:0000310		
12	UBQLN1	breast	UBERON:0000310		
13	ERBB2	breast	UBERON:0000310		

</td></tr>
<tr><td>Result (older-v1)</td><td>

#	gene	uberonname	uberon	gene_sumbol	name
1	EGFR	breast	UBERON:0000310	EGFR	breast cancer
2	MET	breast	UBERON:0000310	MET	breast cancer
3	MYC	breast	UBERON:0000310	MYC	breast cancer
4	LPIN1	breast	UBERON:0000310	LPIN1	breast cancer
5	RHOA	breast	UBERON:0000310	RHOA	breast cancer
6	CCND1	breast	UBERON:0000310		
7	MUC1	breast	UBERON:0000310		
8	ERBB3	breast	UBERON:0000310		
9	ERBB2	breast	UBERON:0000310		
10	AMACR	breast	UBERON:0000310		
11	SOX2	breast	UBERON:0000310		
12	AMFR	breast	UBERON:0000310		
13	RAC1	breast	UBERON:0000310		

</td></tr>
<tr><td>Improvem ents over v1</td><td>Similar to Query 4, we observe that the latest version provides a smaller number of false positives (i.e., downregulated genes in the curated list that are shown as upregulated), as a result of the efficient triple unification approach.</td></tr>
</table>

SQL Query 5

```
SELECT distinct *
FROM triples_fully_linked_v2
WHERE (predicate like '%biomarker%'
or subject like '%biomarker%' or object like '%biomarker%' ) and
(predicate like '%cancer%'
or subject like '%cancer%' or object like '%cancer%' ) and
polarity='TRUE' and uberonname != 'tissue'
```

NL equivalent

Find all cancer biomarkers from the literature.

Explanation

This is a simple query showcasing the information extraction capabilities of the OpenDataLinking component. We focus on finding all literature cases (derived from triple extraction on Pubmed articles) that include the keywords “cancer” and “biomarker” in the extracted triples.

Result (latest version-v2)

#	gene	uberonname	uberon	gene_sumbol	name	subject	predicate	object	polarity
1	1978	27902750	TNF	UBERON:0013	venous blood	stress variables profile of antic	were biochemically assessed from	venous blood of fifty ovarian cancer patients and twe	TRUE
2	794	25559835	TNF	UBERON:0000	breast	tnf polymorphisms	could serve as	useful predictive biomarkers for breast cancer risk an	TRUE
3	2815	21980040	CRYAB	UBERON:0002	prostate gland	lower cryab expression	is a prognostic biomarker for	several types of cancer such as that of the prostate an	TRUE
4	2562	27644245	FABP5	UBERON:0000	lymph node	fatty acid-binding protein 5 fat	was found in previous study to biomarker	be a potential for lymph node metastasis of cervical c	TRUE
5	681	27644245	FABP5	UBERON:0002	lymph	fatty acid-binding protein 5 fat	was found in previous study to biomarker	be a potential for lymph node metastasis of cervical c	TRUE
6	1074	23803082	KLK3	UBERON:0001	posterior communic	klk3 gene products like human	are important biomarkers in	the clinical diagnosis of prostate cancer pca	TRUE
7	1411	27902750	TNF	UBERON:0000	blood	stress variables profile of infla	were biochemically assessed from	venous blood of fifty ovarian cancer patients and twe	TRUE
8	2991	21980040	CRYAB	UBERON:0002	prostate gland	lower cryab expression	is a prognostic biomarker for	several types of cancer such as that of the prostate an	TRUE
9	519	27328733	S100A4	UBERON:0002	liver	nuclear expression of the calci	is a biomarker of	increased invasiveness in cholangiocarcinoma a prim	TRUE
10	1887	25563194	XRCC1	UBERON:0002	lung	xrcc1 genetic polymorphism	acts	a potential biomarker for lung cancer	TRUE
11	2621	26416047	CCND1	UBERON:0000	breast	ccnd1 mutations	may serve as	biomarkers for early diagnosis of breast cancer	TRUE
12	622	26416047	CCND1	UBERON:0000	breast	ccnd1 and cdk4 mutations	may serve as	biomarkers for early detection of breast cancer	TRUE
13	1673	17683075	AMACR	UBERON:0002	prostate gland	expression of the alpha-methy	has been established as	a specific biomarker for the diagnosis of prostate can	TRUE
14	2802	21980040	CRYAB	UBERON:0000	neck	lower cryab expression	is a prognostic biomarker for	several types of cancer such as that of the neck	TRUE
15	1636	23098186	QSOX1	UBERON:0000	breast	qsox1	could be posited as	a new biomarker of good prognosis in breast cancer	TRUE
16	3050	28196064	EGFR	UBERON:0000	breast	egfr expression levels	are	key biomarkers for breast cancer patient response to	TRUE
17	384	29277780	MMP1	UBERON:0002	lung	the mmp1 -1607 1g allele	is a non-significant protective biomarker for	lung cancer in taiwan	TRUE
18	1584	27902750	TNF	UBERON:0000	blood	stress variables profile of antic	were biochemically assessed from	venous blood of fifty ovarian cancer patients and twe	TRUE
19	1574	21980040	CRYAB	UBERON:0000	neck	lower cryab expression	is a prognostic biomarker for	several types of cancer such as that of the prostate an	TRUE
20	2564	26416047	CCND1	UBERON:0000	breast	ccnd1 and cdk4 mutations	may serve as	biomarkers for early diagnosis and detection of breas	TRUE
21	2604	17683075	AMACR	UBERON:0002	prostate gland	expression of the alpha-methy	has been established as	a sensitive biomarker for the diagnosis of prostate ca	TRUE
22	1022	27902750	TNF	UBERON:0000	blood	stress variables profile of antic	were biochemically assessed from	venous blood of fifty ovarian cancer patients and twe	TRUE
23	2366	21980040	CRYAB	UBERON:0000	neck	lower cryab expression	is a prognostic biomarker for	several types of cancer such as that of the prostate an	TRUE
24	498	27328733	S100A4	UBERON:0002	liver	nuclear expression of the calci	is a biomarker of	increased invasiveness in cholangiocarcinoma a prim	TRUE
25	2199	23803082	KLK3	UBERON:0002	prostate gland	klk3 gene products like human	are important biomarkers in	the clinical diagnosis of prostate cancer pca	TRUE
26	2943	27902750	TNF	UBERON:0013	venous blood	stress variables profile of antic	were biochemically assessed from	venous blood of fifty ovarian cancer patients and twe	TRUE
27	2542	21980040	CRYAB	UBERON:0000	head	lower cryab expression	is a prognostic biomarker for	several types of cancer such as that of the prostate an	TRUE
28	1233	21980040	CRYAB	UBERON:0007	craniocervical region	lower cryab expression	is a prognostic biomarker for	several types of cancer such as that of the prostate an	TRUE
29	448	26707566	EGFR	UBERON:0002	lung	epidermal growth factor recep	is a crudal biomarker for	prediction of response to tyrosine kinase inhibitors in	TRUE
30	538	28041942	PTN	UBERON:0000	breast	ptn	could be considered as	a potential biomarker for the presence of breast canc	TRUE
31	1109	28196064	EGFR	UBERON:0000	breast	egfr expression levels	are	key biomarkers for breast cancer patient response to	TRUE
32	2343	27902750	TNF	UBERON:0013	venous blood	stress variables profile of infla	were biochemically assessed from	venous blood of fifty ovarian cancer patients and twe	TRUE

Result (older-v1)

#	gene	uberonname	uberon	gene_sumbol	name	subject	predicate	object	polarity	source
1	1076	26823785	CDKN1A	UBERON:0000310	breast	Both CDKN1A / p21	are presented	as possible candidates for breast cancer bio	TRUE	infili
2	3347	18668212	LEPR	UBERON:0000310	breast	Leptin	evaluated	the relationship among the leptin receptor	TRUE	infili
3	3189	25409685	POLQ	UBERON:0000310	breast	genetic variants	be related	to POLQ as new population biomarkers of r	TRUE	infili
4	294	22446113	EGFR	UBERON:0002107	liver	EGFR	is in biomarker	patients of liver cancer in those with expos	TRUE	both
5	1532	27644245	FABP5	UBERON:0000029	lymph nod	protein 5 FABP5	was found	in our previous study to be a potential bior	TRUE	infili
6	1824	24747263	AFP	UBERON:0000178	blood	exercise frequency	improves	what cancer-related biomarkers such as ca	TRUE	infili
7	2016	22899247	ERBB2	UBERON:0000310	breast	product protein	is	a key biomarker for breast cancer The ERBE	TRUE	infili
8	2666	23567490	CCND1	UBERON:0001155	colon	prospective trials	are warranted	CCND1 rs9344 G A may be a predictive and\	TRUE	infili
9	1533	27644245	FABP5	UBERON:0002391	lymph	protein 5 FABP5	was found	in our previous study to be a potential bior	TRUE	infili
10	2684	21980040	CRYAB	UBERON:0000033	head	lack or lower CRYAB expression	is	a prognostic biomarker for several types of	TRUE	infili
11	2682	28870920	MMP7	UBERON:0000310	breast	GG genotype at MMP7 A-181G	may serve	as a biomarker for early detection and prec	TRUE	infili
12	1528	27709523	S100A6	UBERON:0002048	lung	serum levels of S100A6	were suggested	as a novel biomarker for various inflammat	TRUE	infili

Improvements over v1

A significant increase in the number of extractions showcases the upgraded capabilities of the latest triple extraction approach compared to v1. Moreover, the improvements introduced in the OIE process lead to “cleaner” predicates (i.e., that contain less non-informative context).

R&I Use Case

SQL Query 1

```
SELECT acronym, objective
FROM projects
WHERE unics_id IN
(SELECT neighbour FROM projects
INNER JOIN project_neighbours
ON projects.unics_id = project_neighbours.project
WHERE acronym = 'THINFRAME')
```

NL equivalent

Find 3 closest contextual neighbours of the THINFRAME EU-funded project and their descriptions.

Explanation

As described in D3.1, in the context of the R&I Use Case, we leverage the natural language text of each project stored in the SIRIS database to find semantic neighbours of the existing CORDIS projects, based on their vector representation similarities. We then enrich the SIRIS database with the

	discovered neighbour pairs. This simple query aims at finding the 3 most similar projects to a given one, based on their contextual similarity. Note that while this approach is similar to the `by-neighbour` operator introduced in Section 2.1.4, the current approach focuses on the semantic similarity of unstructured text (expressed in the form of word/sentence embeddings). On the contrary, the `by-neighbour` operator is relevant only for ordered attributes and thus cannot be leveraged for NL-text.
Result	Data Output Explain Messages Notifications acronym text / objective text 1 PORTAFACT — PU foam and EPS panel construction systems are considered to be the best materials used in the pre-cast industry, given their high insulation performance, the reduced we 2 SOLARGAIN — Buildings are a major end-user of European energy, representing 40% of energy consumption and 1,800 million tonnes of CO2 emissions. Improved energy efficiency within 3 NANOINSULATE — NANOINSULATE will develop durable, robust, cost-effective opaque and transparent vacuum insulation panels (VIPs) incorporating new nanotechnology-based core materi

SQL Query 2	`SELECT acronym, objective`
	`FROM projects`
	`WHERE unics_id IN`
	`(SELECT neighbour FROM projects`
	`INNER JOIN project_neighbours`
	`ON projects.unics_id = project_neighbours.project`
	`WHERE acronym = 'SOLUS' AND framework program='H2020')`

NL equivalent	Find closest contextual neighbours of the SOLUS H2020 project and their descriptions.
Explana-tion	This is similar to the above query; we are searching for contextual neighbours of a project. However, now we set the additional condition that the mentioned project is part of the H2020 framework. This aims at discarding any potential synonyms from different frameworks that could influence the results.
Result	Data Output Explain Messages Notifications acronym text / objective text 1 LUCA — This is a trans-disciplinary project that joins endocrinologists ("end-users"), radiologists ("end-users"), physicists who are 2 MAMMOCARE — Breast cancer is the most frequent cancer among women and one of the leading causes of cancer-related mortality and 3 SHINE — By only considering the United States of America and Europe, 1 in 12 women is affected by Breast cancer (BC). It is the la

SQL Query 3	`SELECT acronym, title, objective, framework_program`
	`FROM projects WHERE unics_id IN`
	`(SELECT neighbour FROM projects`
	`INNER JOIN project_neighbours`
	`ON projects.unics_id = project_neighbours.project`
	`WHERE objective LIKE '%urban transport%'`
	`ORDER BY distance ASC LIMIT 5)`

NL equivalent	Show the acronym, title, objective and framework of the 5 contextually closest projects to urban transport.
Explana-tion	In this query we do not directly search for contextual neighbours of a particular project; instead we focus our search on a specific topic/category (e.g., urban transport) and search for neighbours that share the same topic. Note that, while this query implies that there must be at least one project containing the phrase "urban transport" in its objective description, the neighbours identified for that project do not necessarily include the same text. This showcases the added value of vector representations for NL-text, in the context of semantic similarity.

Result

Data Output Explain Messages Notifications

	acronym text	title text	objective text	framework_program text
1	SWARM	Demonstration of Small 4-Wheel fuel c...	This project will establish a demonstration fleet of small passenger vehicles tha...	FP7
2	WeSmartPark	Giving Drivers Access to Conventional ...	WeSmartPark represents a novel technology suite that allows utilizing existing ...	H2020
3	CONCEPT	CONductive fast Charge system for El...	Heliox, a company specialized in switch mode power technology, is developing ...	H2020
4	USharePark	USharePark: smart secUres system to ...	USharePark is a peer-to-peer (P2P) smart parking solution intended to optimize ...	H2020
5	Matrix Charging	Matrix Charging: Novel, automated ch...	Over the next 15 years continuing growth of urbanisation & sprawl will increase ...	H2020

SQL Query 4

```
SELECT acronym, title, ec_call
FROM projects
WHERE framework_program='H2020' AND unics_id IN
(SELECT neighbour FROM projects
INNER JOIN project_neighbours
ON projects.unics_id = project_neighbours.project
WHERE acronym = 'ZEOSOL')
```

NL equivalent

Find the closest contextual neighbours of the ZEOSOL project that are under the H2020 framework and show their acronym, title and call topic.

Explana-tion

In this query we are searching for similar projects of ZEOSOL, but we are limiting the results on neighbours that belong only to the H2020 framework. We are also querying for additional information (acronym, title, and topic).

Result

Data Output Explain Messages Notifications

	acronym text	title text	ec_call text
1	SWS-HEATING	Development and Validation of an Innovative Solar Comp...	H2020-LCE-2017-RES-RIA-TwoStage
2	Solar QUEST	QUalitative Electricity STorage for Solar energy	H2020-SMEInst-2018-2020-2

SQL Query 5

```
SELECT unics_id, acronym, title, start_year, total_cost
FROM projects
WHERE framework_program='H2020' AND start_year > 2018 AND
total_cost<8000000 AND unics_id IN
(SELECT neighbour FROM projects
INNER JOIN project_neighbours
ON projects.unics_id = project_neighbours.project
WHERE acronym = 'GOLIATH'
ORDER BY distance ASC)
```

NL equivalent

Find the closest contextual neighbours of the GOLIATH project that are under the H2020 framework, started later than 2018 and have a total cost less than 8M, and show their ID, acronym, title, start year, and cost.

Explana-tion

This is a more complex query , as we are searching for similar projects to GOLIATH based on multiple conditions. Consequently, the results are limited (from the 3 contextual neighbours extracted for each project by OpenDataLinking, only one fulfills every condition).

Result

Data Output Explain Messages Notifications

	unics_id [PK] integer	acronym text	title text	start_year integer	total_cost double precision
1	892702	EDCMET	Metabolic effects of Endocrine Disrupting Chemicals: n...	2019	5980408.75

3 INODE-SPARQL 1.0 In Action

3.1 OpenDataDialog

3.1.2 NL-to-SPARQL: Translating Natural Language Questions to SPARQL

One of the important advantages of making data available in a knowledge graph, which can then be queried via SPARQL, is that the knowledge graph itself, as well as the corresponding domain ontology, can make use of a terminology closer to the user's natural language. This can greatly benefit the process of searching and exploring the data, in particular through natural language questions. In the case of a virtual knowledge graph, these benefits do not need to come at the cost of changing the original data sources (for example, the relational databases of CORDIS and SDSS), given that these can be kept unmodified, through the use of relational-to-RDF mappings. In the following, we provide a few concrete examples to illustrate the benefits of question answering over the virtual knowledge graphs of CORDIS and SDSS with Bio-SODA.

3.1.2.1 Querying CORDIS in NL with Bio-SODA

The CORDIS database can be easily queried in natural language, for example in order to look for all projects of a certain principal investigator of interest or all members of a given project. A simple example is shown in Figure 3.1. Further examples are available on the CORDIS demo page for Bio-SODA.

ERC projects whose principal investigator is Michael Smith [Go]

Keyword Query: ERC projects whose principal investigator is Michael Smith

Selected Matches (one example per class-property pair, limited to top 10):

principal investigator	ERC project	Michael Smith
http://unics.cloud/ontology#principalInvestigator	http://unics.cloud/ontology#ERC-Project	http://unics.cloud/ontology#Person-1211644

- **principal investigator**
- http://unics.cloud/ontology#principalInvestigator: ObjectProperty: label ("principal investigator"), SPARQL
- **ERC project**
- http://unics.cloud/ontology#ERC-Project: Class: label ("ERC project"), SPARQL
- **Michael Smith**
- http://unics.cloud/ontology#Person-1211644: Person: fullName ("Michael Smith"), SPARQL

Keyword: principal investigator **Match:** http://unics.cloud/ontology#principalInvestigator
Keyword: ERC project **Match:** http://unics.cloud/ontology#ERC-Project
Keyword: Michael Smith *[Will be matched through a SPARQL filter on the class <http://unics.cloud/ontology#Person> property <http://unics.cloud/ontology#fullName>]*

SPARQL query:

```
SELECT DISTINCT ?ercproject ?ercproject_title ?person ?person_fullname WHERE {
    ?ercproject a <http://unics.cloud/ontology#ERC-Project>.
    ?ercproject <http://unics.cloud/ontology#principalInvestigator> ?person.
    ?ercproject <http://unics.cloud/ontology#title> ?ercproject_title.
    ?person <http://unics.cloud/ontology#fullName> ?person_fullname.
    FILTER (contains(lcase(str(?person_fullname)), "michael smith"))
    }
    LIMIT 10
```

?ercproject	?ercproject_title	?person	?person_fullname
http://unics.cloud/ontology#Project-167219	A European Approach to Conflict Resolution? Institutional Learning and the ESDP	http://unics.cloud/ontology#Person-1211644	Michael Smith

Figure 3.1: An example natural language question over the CORDIS knowledge graph using Bio-SODA.

3.1.2.2 Reasoning over the CORDIS Ontology

Although the CORDIS ontology is quite simple, it still presents hierarchies that can be exploited by a reasoner. For instance, consider the following query in natural language.

"titles of erc projects with coordinators and their geographic location"

The geographic locations in CORDIS conform to the EU classification based on the *NUTS classification*. The NUTS classification[25] (Nomenclature of territorial units for statistics) is a hierarchical system for dividing up the economic territory of the EU.

Specifically, the CORDIS ontology presents the following structure:

Since there are three territorial units, without reasoning a user should explicitly ask for each of them. That is, the query above should be formulated as:

"titles of erc projects with coordinators and their NUTS 1 location or NUTS 2 location or NUTS 3 location"

The reasoner relieves the user from such a burden, and the query can simply be formulated according to the hierarchy above as:

"titles of erc projects with coordinators and their NUTS location".

These are some of the reasoning capabilities that are currently supported by Ontop. These capabilities, as well as additional ones illustrated next, will also be integrated with Bio-SODA in the next release.

3.1.2.3 Querying SDSS in NL with Bio-SODA

In the lack of a domain ontology, the original relational database of SDSS cannot be easily searched in natural language, given that it requires understanding the low-level structure of the database, for example, that the attributes "`ra`" and "`dec`", refer to "right ascension" and "declination", respectively. Furthermore, one possible common use case, querying for photometrically observed galaxies, requires the user to know that a galaxy is defined by the numerical value 3 for the attribute "`type`" of the table "`PhotoObj`". However, in the virtual knowledge graph of SDSS, made accessible through Ontop and realized through the OpenDataLinking services, this semantic information can be easily made explicit, for example, through `rdfs:labels`, allowing the user to search by the attribute name in a much more convenient manner. Therefore, Bio-SODA can easily answer a question formulated in natural language, such as "What are all the photo galaxies with right ascension > 100 and declination < 100?". The process is illustrated in Figure 3.2 and Figure 3.3. Note that the equivalent keyword query directly over the relational database would have to be formulated as "`photoobj type = 3 ra > 100 dec < 100`".

[25] https://ec.europa.eu/eurostat/web/nuts/background

what are all the photo galaxies with right ascension > 100 and declination < 100? Go

Keyword Query: what are all the photo galaxies with right ascension > 100 and declination < 100?

Selected Matches (one example per class-property pair, limited to top 5):

right ascension	declination	photo galaxies
http://www.semanticweb.org/skyserver/right_ascension	http://www.semanticweb.org/skyserver/declination	http://www.semanticweb.org/skyserver/PhotoGalaxy

- **right ascension**
- http://www.semanticweb.org/skyserver/right_ascension: DatatypeProperty: uri ("right_ascension"), SPARQL
- **declination**
- http://www.semanticweb.org/skyserver/declination: DatatypeProperty: uri ("declination"), SPARQL
- **photo galaxies**
- http://www.semanticweb.org/skyserver/PhotoGalaxy: Class: uri ("PhotoGalaxy"), SPARQL
- **100** *(numerical filter on property)*
- http://www.semanticweb.org/skyserver/declination: DatatypeProperty: uri ("declination"), SPARQL
- http://www.semanticweb.org/skyserver/right_ascension: DatatypeProperty: uri ("right_ascension"), SPARQL

Figure 3.2: An example natural language question and candidate matches over the SDSS knowledge graph using Bio-SODA.

Keyword: right ascension **Match:** http://www.semanticweb.org/skyserver/right_ascension
Keyword: declination **Match:** http://www.semanticweb.org/skyserver/declination
Keyword: photo galaxies **Match:** http://www.semanticweb.org/skyserver/PhotoGalaxy

SPARQL query:

```
SELECT DISTINCT ?photogalaxy ?photogalaxy_declination ?photogalaxy_right_ascension WHERE {
    ?photogalaxy a <http://www.semanticweb.org/skyserver/PhotoGalaxy>.
    ?photogalaxy <http://www.semanticweb.org/skyserver/right_ascension> ?photogalaxy_right_ascension.
    ?photogalaxy <http://www.semanticweb.org/skyserver/declination> ?photogalaxy_declination.
    FILTER (?photogalaxy_declination < 100 )
    FILTER (?photogalaxy_right_ascension > 100 )
    }
    LIMIT 100
```

?photogalaxy	?photogalaxy_declination	?photogalaxy_right_ascension
http://www.semanticweb.org/skyserver/photoobj/objid=1237651538724913417	53.9425086434111023	240.293028035850995
http://www.semanticweb.org/skyserver/photoobj/objid=1237651538725307086	53.2549480158233024	241.181820051768995
http://www.semanticweb.org/skyserver/photoobj/objid=1237651538726092947	51.8962555655258981	242.899020917994989
http://www.semanticweb.org/skyserver/photoobj/objid=1237651539238846553	58.1171620443342007	136.130439099716
http://www.semanticweb.org/skyserver/photoobj/objid=1237651539238912177	58.2576259547793995	136.040207912815987

Figure 3.3. Corresponding SPARQL query and results table for the natural language question over the SDSS knowledge graph using Bio-SODA.

Further higher-level concepts have also been made available in the SDSS ontology, in order to allow users to easily search for more complex astrophysical objects with certain properties of interest, such as star forming galaxies with a specific velocity dispersion or starburst galaxies with a particular redshift. An example is shown in Figure 3.4.

show all star burst galaxies with velocity dispersion > 800 Go

Keyword Query: show all star burst galaxies with velocity dispersion > 800

Selected Matches (one example per class-property pair, limited to top 5):

velocity dispersion	star burst galaxies
http://www.semanticweb.org/skyserver/Spectroscopy#velocity_dispersion http://www.semanticweb.org/skyserver/Spectroscopy#velocity_dispersion_error	http://www.semanticweb.org/skyserver/StarBurstGalaxy

- **velocity dispersion**
- http://www.semanticweb.org/skyserver/Spectroscopy#velocity_dispersion: DatatypeProperty: uri ("velocity_dispersion"), SPARQL
- http://www.semanticweb.org/skyserver/Spectroscopy#velocity_dispersion_error: DatatypeProperty: uri ("velocity_dispersion_error"),
- **star burst galaxies**
- http://www.semanticweb.org/skyserver/StarBurstGalaxy: Class: uri ("StarBurstGalaxy"), SPARQL
- **800** *(numerical filter on property)*
- http://www.semanticweb.org/skyserver/Spectroscopy#velocity_dispersion: DatatypeProperty: uri ("velocity_dispersion"), SPARQL
- http://www.semanticweb.org/skyserver/Spectroscopy#velocity_dispersion_error: DatatypeProperty: uri ("velocity_dispersion_error"),

Keyword: velocity dispersion **Match:** http://www.semanticweb.org/skyserver/Spectroscopy#velocity_dispersion
Keyword: star burst galaxies **Match:** http://www.semanticweb.org/skyserver/StarBurstGalaxy

SPARQL query:

```
SELECT DISTINCT ?starburstgalaxy ?starburstgalaxy_velocity_dispersion WHERE {

    ?starburstgalaxy a <http://www.semanticweb.org/skyserver/StarBurstGalaxy>.

    ?starburstgalaxy <http://www.semanticweb.org/skyserver/Spectroscopy#velocity_dispersion> ?starburstgalaxy_velocity_dispersion.

    FILTER (?starburstgalaxy_velocity_dispersion > 800 )

    }

    LIMIT 100
```

?starburstgalaxy	?starburstgalaxy_velocity_dispersion
http://www.semanticweb.org/skyserver/specobj/specobjid=5378668592073297920	850
http://www.semanticweb.org/skyserver/specobj/specobjid=5468713096897122304	850
http://www.semanticweb.org/skyserver/specobj/specobjid=6171291136004739072	850
http://www.semanticweb.org/skyserver/specobj/specobjid=6170048688167342080	850
http://www.semanticweb.org/skyserver/specobj/specobjid=3143600302393944064	805.9375
http://www.semanticweb.org/skyserver/specobj/specobjid=6644059696411201536	850
http://www.semanticweb.org/skyserver/specobj/specobjid=2423035070810974208	835.973816
http://www.semanticweb.org/skyserver/specobj/specobjid=6776924132343042048	850

Figure 3.4: Example natural language question targeting starburst galaxies in the SDSS knowledge graph using Bio-SODA.

3.1.2.4 Reasoning over the SDSS Ontology

The SDSS database provides two main catalogs of objects: those coming from *photometric* observations done with cameras, and those coming from *spectrometric* observations done with a multi object, fiber spectroscopic instrument. These two categories bring different kinds of information about the objects in the sky, with some overlapping. Consider again the query

"What are all the photo galaxies with right ascension > 100 and declination < 100?"

and let us slightly modify it, by not only restricting ourselves to photometric observations (photo galaxies), but wanting to consider *all* the galaxies. One option might then be the query:

"What are all the photo and spec galaxies with right ascension > 100 and declination < 100?"

Observe that this solution is not optimal, as it requires the users to be aware of the fact that there exist two different catalogs, and that they need to explicitly ask for both.

Thanks to reasoning, however, we have a better alternative. The SDSS ontology we devised provides a convenient hierarchy specifying that both photo and spec galaxies represent, in fact, galaxies:

Hence, why not to ask for galaxies directly, ignoring the catalog they come from?

"What are all the galaxies with right ascension > 100 and declination < 100?"

The reasoner implemented in Ontop will automatically translate such a query into a query retrieving both photo galaxies and spec galaxies, relieving the final user from that burden.

3.1.3 Enabling SPARQL Queries over OncoMX

3.1.3.1 Enriching OncoMX with Ontologies using Ontop

Originally, the OncoMX datasets are composed of semi-structured data related to cancer biomarkers. Based on them, we built a relational database with corresponding relational schema. Moreover, we took advantage of the Ontop tool over the created OncoMX relational database to integrate external RDF-based ontologies. For example, nucleotides and amino acids are simply stored in the OncoMX relational database as IUPAC (*International Union of Pure and Applied Chemistry*) codes such as "A" to represent the Adenine nucleobase. In addition to this, "A" is also a code for the amino acid Alanine. We solve this ambiguity by enriching the OncoMX data with the *Chemical Entities of Biological Interest (ChEBI)* vocabulary[26]. This enables us to retrieve more information that is not in the original OncoMX datasets, such as the nucleobase name, its synonyms, and chemical formula.

In addition, we use the following vocabularies and ontologies to enrich the OncoMX dataset:

- Experimental Factor Ontology (EFO)[27]
- National Cancer Institute Thesaurus (NCIt) OBO edition[28]
- Uber-anatomy ontology (UBERON)[29]
- Ontology for Biomedical Investigations (OBI)[30]
- OPMI: Ontology of Precision Medicine and Investigation[31]

---

[26] de Matos, P., Alcántara, R., Dekker, A., Ennis, M., Hastings, J., Haug, K., Spiteri, I., Turner, S., & Steinbeck, C. (2010). Chemical Entities of Biological Interest: an update. *Nucleic acids research*, *38*(Database issue), D249–D254. https://doi.org/10.1093/nar/gkp886

[27] Malone J, Holloway E, Adamusiak T, Kapushesky M, Zheng J, Kolesnikov N, Zhukova A, Brazma A, Parkinson H: Modeling Sample Variables with an Experimental Factor Ontology. Bioinformatics 2010, 26(8):1112-1118

[28] https://github.com/NCI-Thesaurus/thesaurus-obo-edition

[29] Mungall, C.J., Torniai, C., Gkoutos, G.V. *et al.* Uberon, an integrative multi-species anatomy ontology.*Genome Biol* 13, R5 (2012). https://doi.org/10.1186/gb-2012-13-1-r5

[30] Bandrowski, A., Brinkman, R., Brochhausen, M., Brush, M. H., Bug, B., Chibucos, M. C., Clancy, K., Courtot, M., Derom, D., Dumontier, M., Fan, L., Fostel, J., Fragoso, G., Gibson, F., Gonzalez-Beltran, A., Haendel, M. A., He, Y., Heiskanen, M., Hernandez-Boussard, T., Jensen, M., … Zheng, J. (2016). The Ontology for Biomedical Investigations. *PloS one*, *11*(4), e0154556. https://doi.org/10.1371/journal.pone.0154556

[31] He Y, Ong E, Schaub J, Dowd F, O'Toole JF, Siapos A, Reich C, Seager S, Wan L, Yu H, Zheng J, Stoeckert C, Yang X, Yang S, Steck B, Park C, Barisoni L, Kretzler M, Himmelfarb J, Iyengar R, Mooney SD, for the Kidney Precision Medicine Project Consortium. OPMI: the Ontology of Precision Medicine and Investigation and its support for clinical data and metadata representation and

- Sequence types and features ontology (SO)[32]
- Semantic Science Integrated Ontology (SIO)[33]

The enrichment is done by manually implementing Ontop mappings such as the one shown in Figure 3.5. In this mapping, we can interpret the "SELECT" projections as the body of a Horn-like rule that is written with the SQL syntax. This SELECT query, when executed over the OncoMX DB, retrieves the information needed to derive the rule head. In this case, the rule head is actually an RDF graph pattern that is structured based on the cancer biomarker ontology we are developing as an RDF data schema for OncoMX data. This ontology reuses the previously mentioned vocabularies among others, for example, FALDO[34] for describing the nucleotide and amino acid locations, as part of its terminological and assertion boxes.

```
sequence_alteration
oncomx:SEQ_ALT_{id}-{peptide_id} a obo:SO_0001059 ; faldo:reference <https://identifiers.org/ensembl:{peptide_id}> ; faldo:location
oncomx:LOCATION_PROT_{id}-{peptide_pos} ; :alteredFrom obo:{ref_aa} , sio:{ref_aa_SIO} , obo:{ref_aa_X} ; :alteredTo obo:{alt_aa} , sio:{alt_aa_SIO
, obo:{alt_aa_X} . oncomx:LOCATION_PROT_{id}-{peptide_pos} a faldo:ExactPosition ; faldo:position {peptide_pos}^^xsd:integer .
oncomx:SEQ_ALT_{id}-{uniprotkb_ac} a obo:SO_0001059 ; faldo:reference <http://purl.uniprot.org/uniprot/{uniprotkb_ac}> ; faldo:location
oncomx:LOCATION_PROT_{id}-{aa_pos_uniprotkb} ; :alteredFrom obo:{ref_aa} , sio:{ref_aa_SIO} , obo:{ref_aa_X} ; :alteredTo obo:{alt_aa} ,
sio:{alt_aa_SIO} , obo:{alt_aa_X} . oncomx:LOCATION_PROT_{id}-{aa_pos_uniprotkb} a faldo:ExactPosition ; faldo:position
{aa_pos_uniprotkb}^^xsd:integer . oncomx:SEQ_ALT_{id}-{gene_symbol} a obo:SO_0001059 ; faldo:reference
<https://identifiers.org/hgnc.symbol:{gene_symbol}> ; faldo:location oncomx:LOCATION_GENE_{id}-{cds_pos} ; :alteredFrom obo:{ref_nt} ;
:alteredTo obo:{alt_nt} . oncomx:LOCATION_GENE_{id}-{gene_symbol} a faldo:ExactPosition ; faldo:position {cds_pos}^^xsd:integer .
SELECT id, CASE ref_nt WHEN 'A' THEN 'CHEBI_16708'  WHEN 'C' THEN 'CHEBI_16040'  WHEN 'G' THEN 'CHEBI_16235'  WHEN 'T' THEN
'CHEBI_17821'  WHEN 'U' THEN 'CHEBI_17568' END AS ref_nt, CASE alt_nt WHEN 'A' THEN 'CHEBI_16708'  WHEN 'C' THEN
'CHEBI_16040'  WHEN 'G' THEN 'CHEBI_16235'  WHEN 'T' THEN 'CHEBI_17821'  WHEN 'U' THEN 'CHEBI_17568' END AS alt_nt, cds_pos,
aa_pos_uniprotkb, CASE ref_aa WHEN 'A' THEN 'CHEBI_16449'  WHEN 'C' THEN 'CHEBI_15356'  WHEN 'D' THEN 'CHEBI_22660'  WHEN 'E'
THEN 'CHEBI_18237'  WHEN 'F' THEN 'CHEBI_28044'  WHEN 'G' THEN 'CHEBI_15428'  WHEN 'H' THEN 'CHEBI_15971'  WHEN 'I' THEN
'CHEBI_17191'  WHEN 'K' THEN 'CHEBI_25094'  WHEN 'L' THEN 'CHEBI_25017'  WHEN 'M' THEN 'CHEBI_16811'  WHEN 'N' THEN
'CHEBI_22653'  WHEN 'P' THEN 'CHEBI_17203'  WHEN 'Q' THEN 'CHEBI_18050'  WHEN 'R' THEN 'CHEBI_16467'  WHEN 'S' THEN
'CHEBI_17115'  WHEN 'T' THEN 'CHEBI_16857'  WHEN 'V' THEN 'CHEBI_16414'  WHEN 'W' THEN 'CHEBI_27897'  WHEN 'Y' THEN
'CHEBI_18186'  END AS ref_aa, CASE ref_aa  WHEN '*' THEN 'SIO_010448' END AS ref_aa_SIO, CASE ref_aa WHEN 'X' THEN
'ANY_CODON'  END AS ref_aa_X,CASE alt_aa WHEN 'A' THEN 'CHEBI_16449'  WHEN 'C' THEN 'CHEBI_15356'  WHEN 'D' THEN
'CHEBI_22660'  WHEN 'E' THEN 'CHEBI_18237'  WHEN 'F' THEN 'CHEBI_28044'  WHEN 'G' THEN 'CHEBI_15428'  WHEN 'H' THEN
'CHEBI_15971'  WHEN 'I' THEN 'CHEBI_17191'  WHEN 'K' THEN 'CHEBI_25094'  WHEN 'L' THEN 'CHEBI_25017'  WHEN 'M' THEN
'CHEBI_16811'  WHEN 'N' THEN 'CHEBI_22653'  WHEN 'P' THEN 'CHEBI_17203'  WHEN 'Q' THEN 'CHEBI_18050'  WHEN 'R' THEN
'CHEBI_16467'  WHEN 'S' THEN 'CHEBI_17115'  WHEN 'T' THEN 'CHEBI_16857'  WHEN 'V' THEN 'CHEBI_16414'  WHEN 'W' THEN
'CHEBI_27897'  WHEN 'Y' THEN 'CHEBI_18186'  END AS alt_aa, CASE alt_aa  WHEN '*' THEN 'SIO_010448' END AS alt_aa_SIO, CASE
alt_aa WHEN 'X' THEN 'ANY_CODON'  END AS alt_aa_X, peptide_pos, mutation_freq, data_source, doid, peptide_id,
dm.ensembl_transcript_id, mp.uniprotkb_ac, gene_symbol FROM disease_mutation as dm join map_protein_disease_mutation as mp on
mp.ensembl_transcript_id = dm.ensembl_transcript_id left join xref_gene_uniprot as hugo on hugo.uniprotkb_ac=mp.uniprotkb_ac
```

Figure 3.5: An Ontop mapping illustrating how nucleotides and amino acid codes are mapped into the Chemical Entities of Biological Interest (ChEBI) vocabulary terms.

3.1.3.2 Reasoning over the OncoMX Ontology

Another added value of accessing the OncoMX data via ontologies with Ontop is the fact of applying *reasoning such as subsumption*. This contributes to simplify the query writing and knowledge discovery.

For example, let us consider the NL question (Q): *What are the genomic biomarkers for breast cancer?* In the OncoMX relational database, there is no information about the fact that all gene and genetic biomarkers are indeed genomic biomarkers. Therefore, without this information, the response of this question will not be complete, and solely biomarkers

analysis. The 10th International Conference on Biomedical Ontology (ICBO-2019), July 30 - August 2, 2019, Buffalo, NY, USA. 10-page full length article.

[32] Mungall CJ, Batchelor C, Eilbeck K (Feb 2011). "Evolution of the Sequence Ontology terms and relationships". Journal of Biomedical Informatics. 44 (1): 87–93. doi:10.1016/j.jbi.2010.03.002.

[33] Dumontier, M., Baker, C.J., Baran, J. *et al.* The Semantic Science Integrated Ontology (SIO) for biomedical research and knowledge discovery. *J Biomed Semant* 5, 14 (2014). https://doi.org/10.1186/2041-1480-5-14

[34] Bolleman, J.T., Mungall, C.J., Strozzi, F. *et al.* FALDO: a semantic standard for describing the location of nucleotide and protein feature annotation. *J Biomed Semant* 7, 39 (2016). https://doi.org/10.1186/s13326-016-0067-z

explicitly annotated with the "Genomic" tag are retrieved by excluding those with either "Gene" or "Genetic" tags.

Figure 3.6 shows a portion of the OncoMX relational data schema about biomarkers. As a result, to retrieve all biomarkers from the OncoMX relational DB the question would need to be rewritten to explicitly include the genomic biomarker subtypes: "Genetic" and "Gene". This results in the following expanded question (EQ): "*What are the genomic, genetic and gene biomarkers for breast cancer?*".

An example of a corresponding SQL query for the expanded question (EQ) is depicted as follows:

```
SELECT gene_symbol, biomarker_description
FROM biomarker AS b JOIN
  biomarker_edrn AS edrn ON b.id = edrn.id JOIN
  anatomical_entity AS anat ON anat.id = edrn.uberon_anatomical_id
WHERE anat.name = 'breast' AND
biomarker_type IN ( 'Genomic', 'Gene',  'Genetic')
```

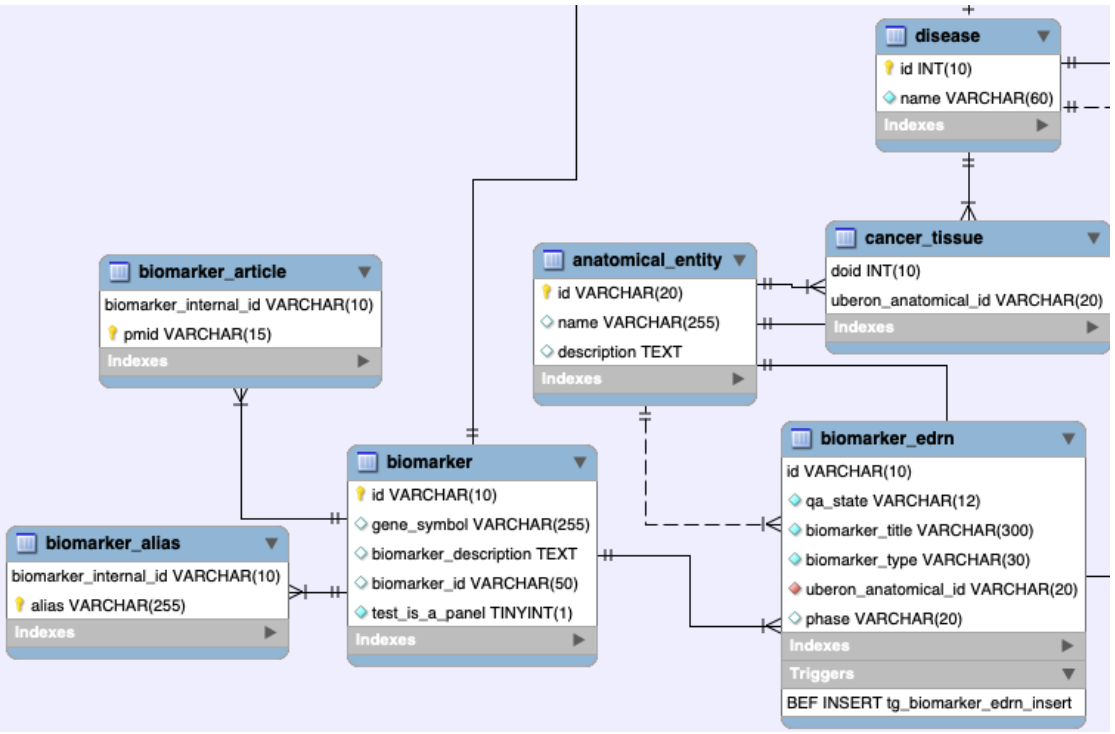


Figure 3.6: A portion of the OncoMX relational data schema for structuring biomarker related data.

Based on the latest OncoMX data, on the one hand, a direct translation of the (Q) question to SQL only retrieves one result: the CDH1 gene as a genomic biomarker, no subtypes are considered. On the other hand, by considering an ontological approach with Ontop over the OncoMX relational database, we are able to fetch four biomarkers in total (i.e., CDH1, MMP10, STOM, MYC genes) where three of them are explicitly defined as gene biomarkers, that are also genomic biomarkers. This is only possible because Ontop infers that the

instances of gene biomarkers are genomic biomarkers too. Figure 3.7 exemplifies a corresponding SPARQL query translated from the question (Q).

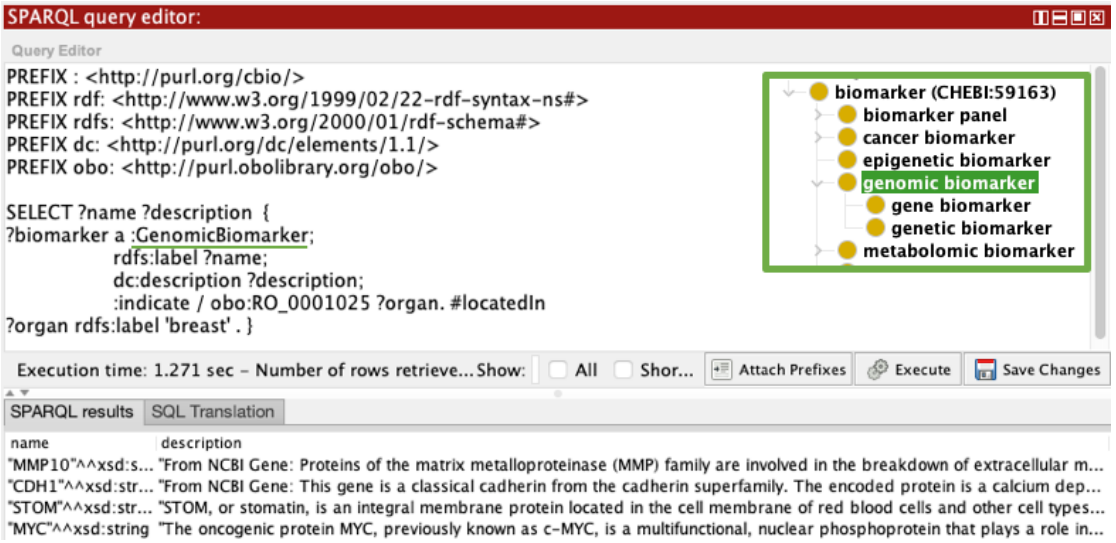


Figure 3.7: Ontop SPARQL query editor in Protege. It illustrates the corresponding SPARQL query and answers for the question: What are the genomic biomarkers for breast cancer? The green box on the top right of this figure shows a portion of the ontology used, stating that gene and genetic biomarkers (subclasses) are also genomic biomarkers (superclass). Based on this portion of the ontology, thanks to the Ontop reasoner, the MMP10, STOM, MYC gene biomarkers are also retrieved as a genomic biomarker, rather than solely CDH1.

3.1.3.3 Querying OncoMX with SPARQL

Table 3.1 depicts three questions, their corresponding SPARQL queries as part of our 11 query test catalog, and their number of results along with their execution time.

Question	SPARQL query	Results
What are all cancer types in the database?	SELECT * { ?doid a efo:EFO_0000408. #disease ?doid rdfs:label ?name; terms:identifier ?id.}	Retrieves: 43 results in ~22 seconds doid / name / id <http://purl.obolibrary.org/obo/DOID_119> "vaginal cance..." "119" <http://purl.obolibrary.org/obo/DOID_169> "neuroendocri..." "169" <http://purl.obolibrary.org/obo/DOID_184> "bone cancer"... "184" <http://purl.obolibrary.org/obo/DOID_219> "colon cancer"... "219"
What are the cancer types where the A1BG gene expression is increased (up regulated)?	SELECT ?cancer { ?x a :CancerDifferentialExpression ; genex:hasSequenceUnit <https://identifiers.org/hgnc.symbol:A1BG> ; :hasTargetDisease / rdfs:label ?cancer;	Retrieves: 8 results in ~11 seconds

(response time ~1 second)	:hasExpressionChange sio:SIO_000640. #increased (up) }	cancer "stomach canc... "urinary bladd... "head and nec... "lung cancer"^... "breast cancer... "kidney cancer... "esophageal c... "thyroid cance...
What are the healthy organs where the A1BG is expressed?	SELECT ?organ { ?gene rdfs:label 'A1BG'. ?gene genex:isExpressedIn/ rdfs:label ?organ.}	Retrieves: 74 results in ~0.15 second. organ "lower esophagus mucosa"... "left testis"^^xsd:string "Ammon's horn"^^xsd:string "left adrenal gland cortex"... "heart"^^xsd:string

Table 3.1. Querying OncoMX with Ontop and SPARQL query language.

3.2 OpenDataLinking

3.2.1 Mapping from Relational Schema to Ontology via Ontop

Manually writing ontologies and mappings, starting from the relational schema of one or more available data sources, is a tedious and error-prone process. For this reason, in INODE our objective is to automate as much as possible the generation of an ontology and mappings that are well suited for extracting data from the available data sources.

In INODE-SPARQL 1.0, MPBoot goes beyond the W3C Direct Mapping Recommendation (which was the technique used for INODE 1.0), overcoming several shortcomings of such recommendation. For convenience, we will refer to the system described in Release 1.0 as *Mapping Patterns Bootstrapper 1.0* (MPBoot 1.0), and to the current system as MPBoot 2.0. In compliance with the INODE Tasks 4.1 and 4.2, MPBoot 2.0 supports the bootstrapping process along two main directions: *data-driven bootstrapping* and *task-driven bootstrapping.*

Data-driven bootstrapping. In its default modality, MPBoot takes as input a configuration file, containing the connection parameters to a relational data source, and produces an ontology and mappings that reflect how the data is organized within the data source. This is also the modality envisioned by the Direct Mapping recommendation. The main shortcoming of Direct Mapping, highlighted in Release 1.0, is that the generated ontologies

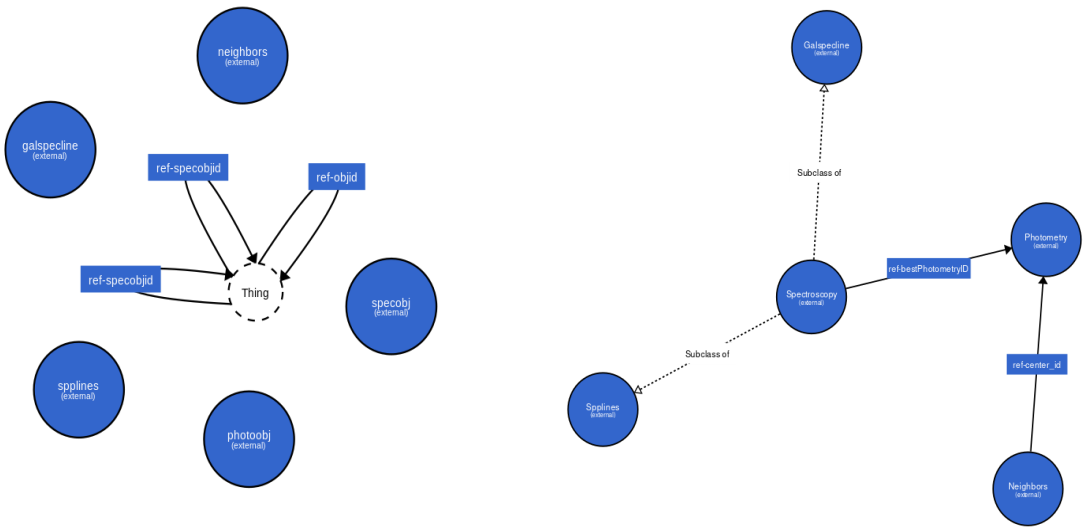


Figure 3.8: SDSS ontology generated by MPBoot 1.0 (left) vs MPBoot 2.0 (right).

are poorly structured. To overcome this limitation, MPBoot 2.0 exploits our theoretical results on *mapping patterns* and the capabilities of the OWL 2 QL language to encode in the ontology schema additional information available in the data source, such as taxonomic relationships. Specifically, MPBoot 2.0 generates:

- *Domain* and *range axioms* for data-properties, by relying on the conversion between DB types and RDFS recommended by the W3C
    - See https://www.w3.org/TR/csv2rdf/#datatypes
- *Domain and range axioms* for object-properties.
- *Subclass relations*.

Figure 3.8 compares the ontology produced by MPBoot 1.0 to the one produced by MPBoot 2.0 over the SDSS dataset. We observe that the ontology produced by MPBoot 2.0 has a richer structure, as it contains semantic connections (object properties and class subsumptions) between different classes.

Another shortcoming of Direct Mapping we notice here is that it is fully automatic: the syntactic specification of the ontology classes and relations is exactly the same of the one of the data sources at hand. This makes the generated ontologies and mappings very hard to be easily adapted towards specific user-needs. MPBoot 2.0 overcomes this limitation by supporting a *semi-automatic approach* to the generation of mappings and ontologies. Specifically, MPBoot 2.0 allows the user to specify, through a configuration file, information about:

- What *tables and attributes* of the DB to use *for the automatic bootstrapping process* (e.g., the user could be interested in mapping only a subset of the available tables and attributes, rather than all of them);

- What *names to use* for the generation of ontology individuals, classes, and properties obtained from specific tables and attributes of the DB.

The first item allows users to differentiate the portions of the ontology to generate automatically from those that require manual crafting. The second item allows them to use names that are more suitable to their needs and understanding of the domain terminology, rather than those chosen by the database designer (this is particularly useful in the astrophysics scenario, where several column names use abbreviations that are hard to understand from a natural language perspective). Thanks to these capabilities of MPBoot 2.0, the generated ontology and mappings can be easily merged with manually-written (imported) ontologies and mappings. Such strategy has proved itself successful in the generation of the ontology and mappings for the SDSS dataset (see Section 4).

Task-driven bootstrapping. As discussed so far, MPBoot drives the bootstrapping process according to the constraints declared in the DB schema. In real-world scenarios, however, such constraints might not be available, e.g., due to performance considerations or because of data source denormalization. As a minimal example of this, consider the DB schema in Figure 3.9, which is part of the widely-used Spider dataset[35]:

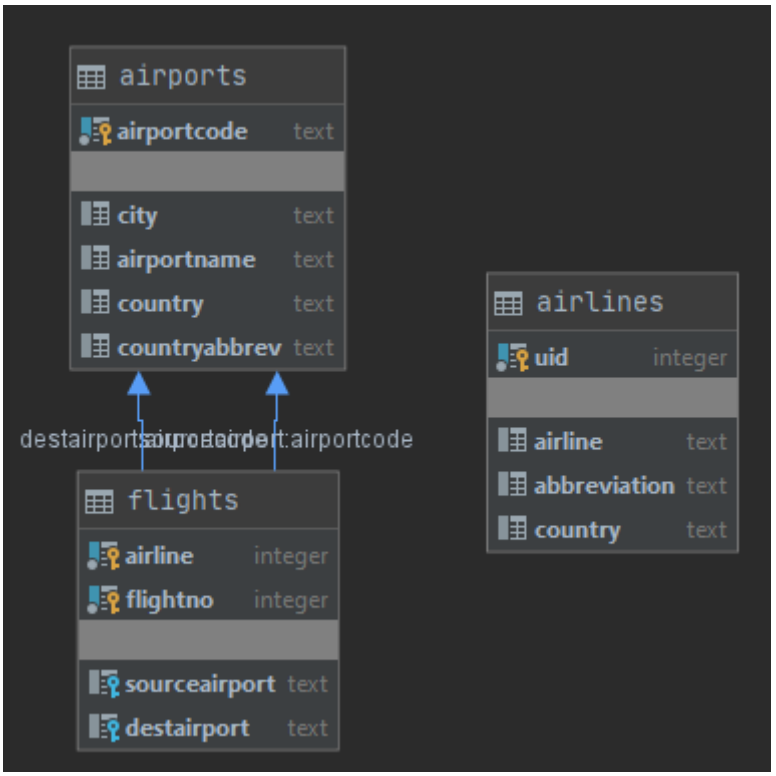


Figure 3.9: Data model of the flights database as part of the Spider dataset. We can observe the orphan table "airlines" which is not linked with the table "flights" via a primary/foreign key relationship.

[35] https://yale-lily.github.io/spider

In such DB schema, there is no explicit (primary/foreign key) relationship between `flights` and `airlines`. Hence, the bootstrapper will not generate an object property relating instances of the class Flights (capturing the elements in the `flights` table) to the instances of the class Airlines (capturing the elements in the airlines table).

Not having an object property in the ontology has, on the query-answering process, a much more severe impact than not having a foreign key in the database. Thanks to the expressivity of the SQL language, in fact, it is still possible to retrieve the country of the airline serving a specific flight:

```
SELECT F.airline, F.flightno, A.country
FROM flights AS F JOIN airlines AS A ON F.airline = A.uid
WHERE f.airline = "Lufthansa" AND f.flightno = "2234"
```

On the SPARQL side, however, without an explicit object property connecting flights and airlines the above SQL query **cannot be expressed at all**. Hence, for the database in Figure 3.9, the data-driven bootstrapping process results in schema-to-ontology transformation which is **not lossless**.

The idea of task-driven mappings is to *exploit a given SQL query workload* in order to identify and fix these situations. With respect to our example, MPBoot will parse the SQL query and derive that the attribute `airline` of table `flights` refers to the primary key of table `airlines`. Hence, it will conclude that there should be a semantic connection between the classes Flights and Airlines, and generate an object property accordingly. For our example, it will generate the following mapping (where we highlighted in green the newly created "airlines#join-uid" object property):

```
Id      BOOTSTRAPPED-MAPPING-ID1
target  <http://semanticweb.org/flights/airlines/uid={airlines_uid}>
        <http://semanticweb.org/flights/airlines#join-uid>
        <http://semanticweb.org/flights/flights/Airline={fA};FlightNo={fN}> .
source   SELECT "airlines"."uid" AS airlines_uid, "flights"."Airline" AS fA,
        "flights"."FlightNo" AS fN
        FROM   "airlines", "flights" WHERE airlines.uid = flights.airline
```

It is now possible to express the SPARQL equivalent of our SQL query:

```
SELECT ?country WHERE {
  ?flight :airline "Lufthansa" ;
          :flight_no "2234" ;
          :airlines#join-uid ?airline .
  ?airline :country ?country .
}
```

In Section 4 below, when talking about data models, we will provide and discuss visualizations for the SDSS ontology, which has been semi-automatically bootstrapped by means of MPBoot.

4 API Specification

In this section we provide the API specification of INODE-SQL 2.0 and INODE-SPARQL 1.0. We will only specify the new APIs.

4.1 OpenDataDialog

4.1.1 NL-to-SQL and SQL-to-NL

The API design reported as part of INODE-SQL 1.0 was made general enough to accommodate new datasets and natural language systems. In this sense, the addition of the astrophysics dataset SDSS and the natural language system ValueNet to the present release of INODE-SQL 2.0 has not required any modification to the API specification of these two services.

The introduction of a logging mechanism oriented towards evaluation has required a new piece of information to be passed to the two services, so that the inputs, outputs, and latency of the NL-to-SQL and SQL-to-NL translations can be properly recorded and associated to a user's session. This information is an identifier for the user, which is passed in an HTTP header named `X-Reference-ID`.

4.1.2 Multi-Table Explorer

The API has changed only slightly due to the addition of new data types and the inclusion of additional table metadata such as source SQL statement and SQL-to-NL explanation. However, this was an internal API change and did not have any impact on our partners' components which integrate into the Multi-Table Explorer (see Section 1.5 for details).

4.1.3 Pipeline Operators

Our operators are described in Section 1.3.2. Here, we provide the semantics of the two new operators: `by-neighbors` and `by-distribution`, which have been added to INODE-SQL 2.0.

***4.1.3.1* Exploration operator `by-neighbors`**. This operator takes a set S and returns k sets s', such that for each predicate ($a = v$), where a is an ordered attribute return two sets s', s''. The algorithm for implementation of the `by-neighbors` operator can be summarized as: receive input S (an example set), A (a set of attributes defining a distribution) and k (the number of resulting sets). Then for each predicate ($a = v$) in $s.description$ where a is an ordered attribute, return two sets s',s'' with the following description:

- replace ($a = v$) in $s.description$ with ($a = v+1$) to obtain $s'.description$
- replace ($a = v$) in $s.description$ with ($a = v-1$) to obtain $s''.description$

The screenshot of by-neighbors is as follows:

By Neighbors

Gets the lower and higher sets on a list of ordonned attributes.

REQUEST BODY SCHEMA: application/json

database required	string (Database name) Enum: "unics_cordis" "sdss" The name of the database to work on
inputSet › required	object (Inputset) The definition of the operator input set (parsed SQL query)
attributes required	Array of strings (Attributes) The list of ordonned attributes to look for neighbor sets

Responses

200 Successful Response

RESPONSE SCHEMA: application/json

error	integer (Error) Default: 0 The error status, 1 if an error has occurred, 0 otherwise
errorMsg	string (Errormsg) The error message
payload	Array of strings (Payload) The list of queries resulting of the operation

422 Validation Error

RESPONSE SCHEMA: application/json

detail ›	Array of objects (Detail)

***4.1.3.2* Exploration operator `by-distribution`.** This operator takes a set of items D and a set of attributes A and returns k sets of items that are distinct from the D and whose items share the same distribution of values for each attribute in A as items in D.

The SQL expression is as follows: `SELECT * FROM` $D \backslash D'$ `WHERE` $P_A^{\approx}$

The algorithm for implementation of the `by-distribution` operator can be summarized as: receive input D (an example set), A (a set of attributes defining a distribution) and k (the number of resulting sets). It returns D' that contains the k largest sets that overlap the least with the input set and which are the most disjoint. All sets have the same distribution as the input one.

The screenshot of by-distribution is as follows:

By Distribution

Return a list of sets with similar description values distribution (for ordered desctiption attributes)

REQUEST BODY SCHEMA: application/json

database required	string (Database name) Enum: "unics_cordis" "sdss" The name of the database to work on
inputSet › required	object (Inputset) The definition of the operator input set (parsed SQL query)

Responses

200 Successful Response

RESPONSE SCHEMA: application/json

error	integer (Error) Default: 0 The error status, 1 if an error has occurred, 0 otherwise
errorMsg	string (Errormsg) The error message
payload	Array of strings (Payload) The list of queries resulting of the operation

422 Validation Error

RESPONSE SCHEMA: application/json

detail ›	Array of objects (Detail)

4.1.3.3 by-recommendation operator (PyExplore)

This is an example of the REST-endpoint showcasing a sample request and response of a by-recommend operator.

Below we can see an example with the different values for the request body:

Schemas

```
pyExploreBody ⌄ {
    selectDB*                   string
                                title: Database name

                                The name of the database to work on

                                Enum:

                                ⌄ [ SDSS, cordis, dummy ]
    max_completions*            integer
                                title: Number of completions
                                default: 4

                                The total number of completions

    initial_query*              string
                                title: Initial query

                                The initial query

    clustering_method*          string
                                default: anglo
                                title: The clustering method for the correlation step

                                The clustering method for the correlation step

                                Enum:

                                ⌄ [ anglo, optics ]
    view_size_max*              integer
                                default: 3
                                title: The maximum number of attributes per view

                                The maximum number of attributes per view for hierarchical clustering

    clustering2*                string
                                default: H2O_Kmeans
                                title: The clustering method for the second Step

                                The clustering method for the second step

                                Enum:

                                ⌄ [ Kmeans, H2O_Kmeans, MB_Kmeans ]
    silhouetteSel*              string
                                default: False
                                title: Compute Silhuette score

                                Compute Silhuette Score

                                Enum:

                                ⌄ [ False ]
    doScore*                    string
                                default: True
                                title: Perform scoring for the queries

                                Perform scoring for the queries

                                Enum:

                                ⌄ [ True, False ]
    frac_rows*                  number
                                default: 1.0
                                title: Fraction of rows to use for sampling

                                Fraction of rows to use for sampling

    frac_columns*               number
                                default: 1.0
                                title: Fraction of columns to use for sampling

                                Fraction of columns to use for sampling
```

PUT `/pyexplore` Returns query recommendations

Returns query recommendations

Parameters Try it out

No parameters

Request body required — application/json

Example Value | Schema


```
{
  "max_completions": "4",
  "initial_query": "SELECT * FROM input_df",
  "clustering_method": "anglo",
  "view_size_max": "3",
  "selectDB": "dummy",
  "clustering2": "H2O_Kmeans",
  "use_silhouette": "False",
  "compute_score": "True",
  "frac_rows": "1.0",
  "frac_columns": "1.0",
  "view_method": "correlation",
  "target_bins": "128",
  "bit_instead_of_nat": "True",
  "method": "mixed",
  "vectorization_clustering": "TF_IDF",
  "vec_method1": "TF_IDF",
  "do_app_dt": "False"
}
```


Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls Accept header.	*No links*

Example Value | Schema


```
[
  {
    "results": "[{\"name\":\"SELECT * FROM TABLE1\",\"score\":0.9,\"json_array\":\"[{\\\"cluster\\\":0,\\\"cou
nts\\\":2,\\\"view\\\":\\\"view1\\\"},{\\\"cluster\\\":1,\\\"counts\\\":2,\\\"view\\\":\\\"view1\\\"}]\",\"vie
w\":\"view1\"},{\"name\":\"SELECT * FROM TABLE2 where attribute2 == value and attrubute 3 == value3\",\"score
\":0.8,\"json_array\":\"[{\\\"cluster\\\":0,\\\"counts\\\":2,\\\"view\\\":\\\"view1\\\"},{\\\"cluster\\\":
1,\\\"counts\\\":2,\\\"view\\\":\\\"view1\\\"}]\",\"view\":\"view1\"},{\"name\":\"select * from table\",\"scor
e\":0.3,\"json_array\":\"[{\\\"cluster\\\":0,\\\"counts\\\":2,\\\"view\\\":\\\"view1\\\"},{\\\"cluster\\\":
1,\\\"counts\\\":2,\\\"view\\\":\\\"view1\\\"}]\",\"view\":\"view1\"},{\"name\":\"SELECT * FROM TABLE1\",\"sco
re\":0.9,\"json_array\":\"[{\\\"cluster\\\":0,\\\"counts\\\":1,\\\"view\\\":\\\"view2\\\"},{\\\"cluster\\\":
1,\\\"counts\\\":2,\\\"view\\\":\\\"view2\\\"}]\",\"view\":\"view2\"}]",
    "correlation_array": "{\"columns\":[\"attr1\",\"attr2\"],\"index\":[\"attr1\",\"attr2\"],\"data\":[[1.0,0.
5],[0.5,1.0]]}"
  }
]
```

```
    view_method*              string
                              default: correlation
                              title: view selection method

                              view selection method

                              Enum:

                               [ correlation, diversity, random ]
    target_bins*              number
                              default: 128
                              title: Number of bins for calculating diversity

                              Number of bins for calculating diversity

    dit_instead_of_nat        string
                              default: True
                              title: Whether or not to use Bit of Natural numbers when computing diversity

                              Whether or not to use Bit of Natural numbers when computing diversity

                              Enum:

                               [ True, False ]
    method_arg*               string
                              default: mixed
                              title: whether or not we used mixed or string workflow

                              view selection method

                              Enum:

                               [ mixed, string ]
    vectorization_clustering* string
                              default: TF_IDF
                              title: Vectorization method for the dataset in the string workflow

                              Vectorization method for the dataset in the string workflow

                              Enum:

                               [ TF_IDF, doc2vec, CountVector ]
    vec_method1*              string
                              default: TF_IDF
                              title: Vectorization method for the first step in the string workflow

                              Vectorization method for the first step in the string workflow

                              Enum:

                               [ TF_IDF, Doc2Vec ]
    do_app_dt*                string
                              default: False
                              title: whether or not to use the approximate decision tree

                              whether or not to use the approximate decision tree

                              Enum:

                               [ True, False ]

}
example: OrderedMap { "max_completions": "4", "initial_query": "SELECT * FROM input_df", "clustering_method": "anglo", "view_size_max":
"3", "selectDB": "dummy", "clustering2": "H2O_Kmeans", "use_silhouette": "False", "compute_score": "True", "frac_rows": "1.0",
"frac_columns": "1.0", "view_method": "correlation", "target_bins": "128", "bit_instead_of_nat": "True", "method": "mixed",
"vectorization_clustering": "TF_IDF", "vec_method1": "TF_IDF", "do_app_dt": "False" }
```


4.1.4 Integrated Query Processing

Ontop is compliant with the standard SPARQL HTTP protocol. One can use POST or GET requests to communicate with a SPARQL endpoint powered by Ontop.

Example

For example, suppose that the SPARQL endpoint is http://localhost:8080/sparql. Then one can query this endpoint through a POST request:

```
POST http://localhost:8080/sparql
Content-Type: application/sparql-query
Accept: application/json

PREFIX : <http://example.org/voc#>
SELECT DISTINCT ?teacher {
  ?teacher a :Teacher .
}
```

The above request can be sent through, for instance, a cURL command:

```
$ curl --request POST \
 --url http://localhost:8080/sparql \
 --header 'accept: application/json' \
 --header 'content-type: application/sparql-query' \
  --data 'PREFIX : <http://example.org/voc#> SELECT DISTINCT ?teacher {?teacher a :Teacher
.}'
{
 "head" : {
  "vars" : [
   "teacher"
  ]
 },
 "results" : {
  "bindings" : [
   {
    "teacher" : {
     "type" : "uri",
     "value" : "http://example.org/voc#uni1/academic/1"
    }
   },
   {
    "teacher" : {
     "type" : "uri",
     "value" : "http://example.org/voc#uni1/academic/2"
    }
   },
        // ...
  ]
 }
}%
```

Language-specific API

Any language-specific API supporting the SPARQL HTTP protocol can be used in combination with Ontop. Notable examples include:

- Java with the Maven dependency: org.eclipse.rdf4j:rdf4j-client
- Python with the RDFLib lirary https://github.com/RDFLib/sparqlwrapper
- JavaScript in HTML using standard Fetch API following the SPARQL HTTP protocol

A demo on how to use the SPARQL API through a programmatic interface is demonstrated at https://github.com/ghxiao/ontop-endpoint-demo.

4.2 OpenDataLinking

4.2.1 OpenDataLinking Triple Extraction from NL-text Endpoint (REST-endpoint)

The API of OpenDataLinking Triple Extraction from NL-text remains unchanged from INODE v1.0.

4.2.2 MPBoot API

The bootstrapping of ontologies and mappings is an offline activity which happens before the deployment of the INODE system. For such a reason, MPBoot does not provide a programmatic interface, but is a command-line tool.

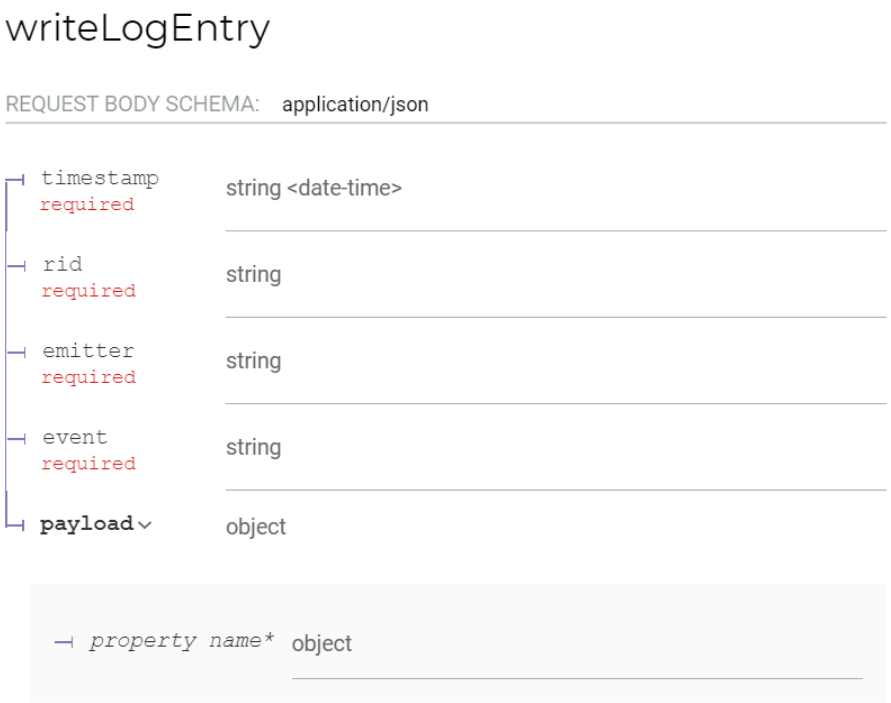
```
$ ./ontop help bootstrap
NAME
        ontop bootstrap - Bootstrap ontology and mapping from the database
SYNOPSIS
        ontop bootstrap [ {-a | --advanced} ] [ {-b | --base-iri} <base IRI> ]
        [ {-c | --constraint} <constraint file> ]
        {-m | --mapping} <mapping file>
        {-p | --properties} <properties file>
        [ {-r | --renamings} <Aliases CSV file> ]
        [ {-t | --ontology} <ontology file> ]
        [ {-w | --workload} <Workload JSON file> ]
OPTIONS
        -a, --advanced
        Enable patterns-based generation (MPBoot)

        -b <base IRI>, --base-iri <base IRI>
        base uri of the generated mapping

        -c <constraint file>, --constraint <constraint file>
        user supplied DB constraint file

        -m <mapping file>, --mapping <mapping file>
        Mapping file in R2RML (.ttl) or in Ontop native format (.obda)

        -p <properties file>, --properties <properties file>
        Properties file

        -r <Aliases CSV file>, --renamings <Aliases CSV file>
        Renamings file [with -a only]

        -t <ontology file>, --ontology <ontology file>
        OWL ontology file

        -w <Workload JSON file>, --workload <Workload JSON file>
        Query workload [with -a only]
```

For instance, the command exploiting task-driven generation used for the airlines example in Section 3.2.1 is the following:

```
$ ./ontop bootstrap -a -b "http://www.inode-project.eu" \
                    -t "onto/flight_2.owl" -m "onto/flight_2.obda" \
                    -p "onto/flight_2.properties" -w "onto/flights_queries.json"
```

4.3 Logging Services

The logging API consists of a single operation for writing a log entry:



The log entry to be recorded is passed as the operation's parameter. This log entry is a JSON object that consists of a timestamp, a reference identifier (RID), the name of the INODE service/component that triggers the event being recorded (emitter), the type of event being recorded, and a payload object that provides additional details on the event and whose attributes depend on the specific event type.

Consider an example for the logging of evaluation parameters, where a participant starts a session in the INODE-SQL 2.0 frontend. The entries recorded in the log are shown in Figure 4.1.

```
{
```

```
    "timestamp": "2021-01-26T12:48:50.992Z",
    "rid": null,
    "emitter": "frontend",
    "event": "create-new-id",
    "payload": {}
  },
  {
    "timestamp": "2021-01-26T12:48:50.992Z",
    "rid": null,
    "emitter": "backend",
    "event": "create-new-id",
    "payload": {
        "rid": "550e8400-e29b-11d4-a716-446655440000"
    }
  },
  {
    "timestamp": "2021-01-26T12:48:51.992Z",
    "rid": null,
    "emitter": "frontend",
    "event": "receive-id",
    "payload": {
        "rid": "550e8400-e29b-11d4-a716-446655440000"
    }
  },
  {
    "timestamp": "2021-01-26T12:48:52.992Z",
    "rid": "550e8400-e29b-11d4-a716-446655440000",
    "emitter": "frontend",
    "event": "session-started",
  },
```


Figure 4.1: An example of a log structure.

The logging example demonstrated in Figure 4.1 shows an extract from the logging mechanism in action when the user first makes a request to the web app or "`frontend`" at timestamp `"2021-01-26T12:48:50.992Z"`. The "`frontend`" makes a request or an "`event`" called "`create-new-id`" for an user id (or "`rid`") to the "`backend`" at the same time instant. The "`backend`" generates the user ID (or "`rid`") as "`payload`" "`550e8400-e29b-11d4-a716-446655440000`" and sends to the "`frontend`" to

complete the request. The "frontend" acknowledges the "event" as "receive-id" and the specific user ID (or "rid") is assigned to the INODE web app. The "frontend" changes the current "event" from "receive-id" to "session-started".

Using our logging mechanism we are now able to record the Query Execution start time and end time and calculate the Query Execution Latency for different systems (e.g. Nalir+, SODA, Logos, ValueNet). In order to find the query execution latency for each system, a fixed set of payload was given to each system. The average query execution for latency for SODA is 2939.57 ms, for Logos it is 36.62 ms, for Nalir+ it is 1403.3 and for ValueNet it is 6924.38 ms.

5 Data Models

In this section we provide the database graphs, the entity-relationship diagrams and their corresponding ontology views for the three use cases *research & innovation policy making (CORDIS), astrophysics* (SDSS), and *cancer research* (OncoMX).

In contrast with the previous release, the database graphs of both CORDIS and SDSS databases are annotated and stored only once in the system. Special labels have been created for the majority of the nodes and the edges of the graphs (in order for Logos to produce more natural explanations). Each node in the graph corresponds to a table or an attribute. Edges connect either table with table (foreign-primary key relationships) or an attribute with its corresponding table. Given an input query, Logos produces its natural language explanation by traversing an extended part of this graph (called query graph) corresponding to that particular query.

5.1 Research & Innovation Policy Making (CORDIS)

Except for the database graph, there were no changes on the data model since the INODE-SQL 1.0 release.

The new database graph of CORDIS produced by Logos is illustrated in Figure 5.1.

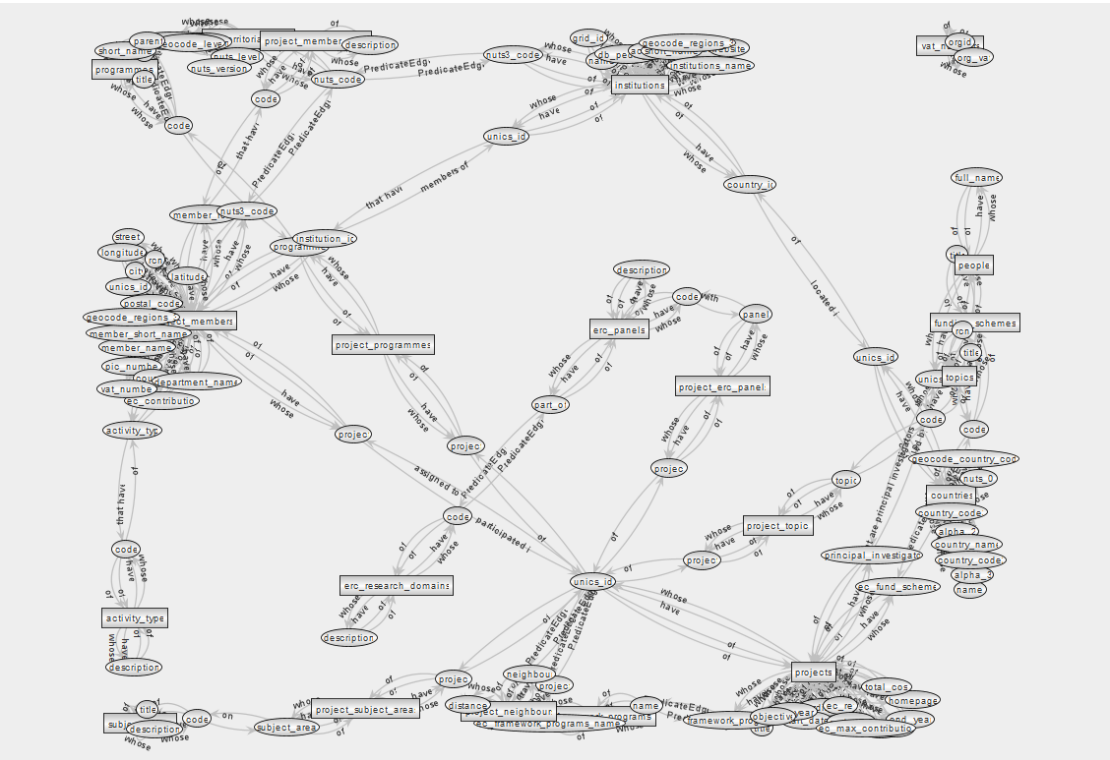


Figure 5.1: CORDIS database graph created by Logos.

5.2 Astrophysics (SDSS)

There are no changes to the core data model (entity-relationship diagram) with respect to INODE 1.0. However, the effort towards the *creation of the ontology and mappings* for this scenario has progressed, since INODE 1.0, in two directions: one *automatic (data-driven)* and the other *manual*.

The first direction has been followed through MPBoot, which allowed us to specify the portions of the database that needed to be mapped automatically, and to freely choose the names for instances, classes, and properties. Thanks to this flexibility, it has been possible to directly merge the bootstrapped ontology together with the one we manually produced for INODE 1.0. An additional effort has then been spent in the manual refinement of the merged ontology. A visualization of the result of these activities is provided in Figure 5.2.

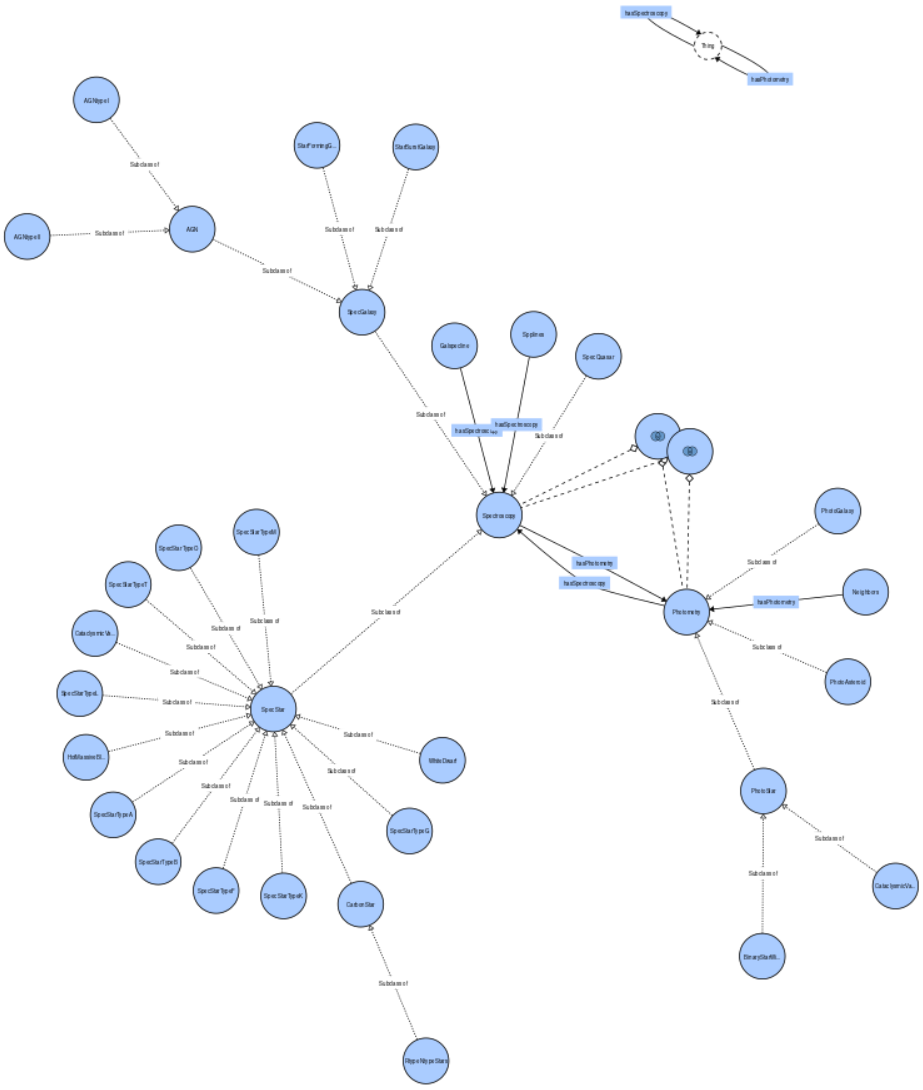


Figure 5.2: SDSS ontology 2.0 (WebVOWL visualization).

By retrieving information about the attributes, their corresponding tables and the relationships among them, the SDSS database graph of Logos is created (see Figure 5.3).

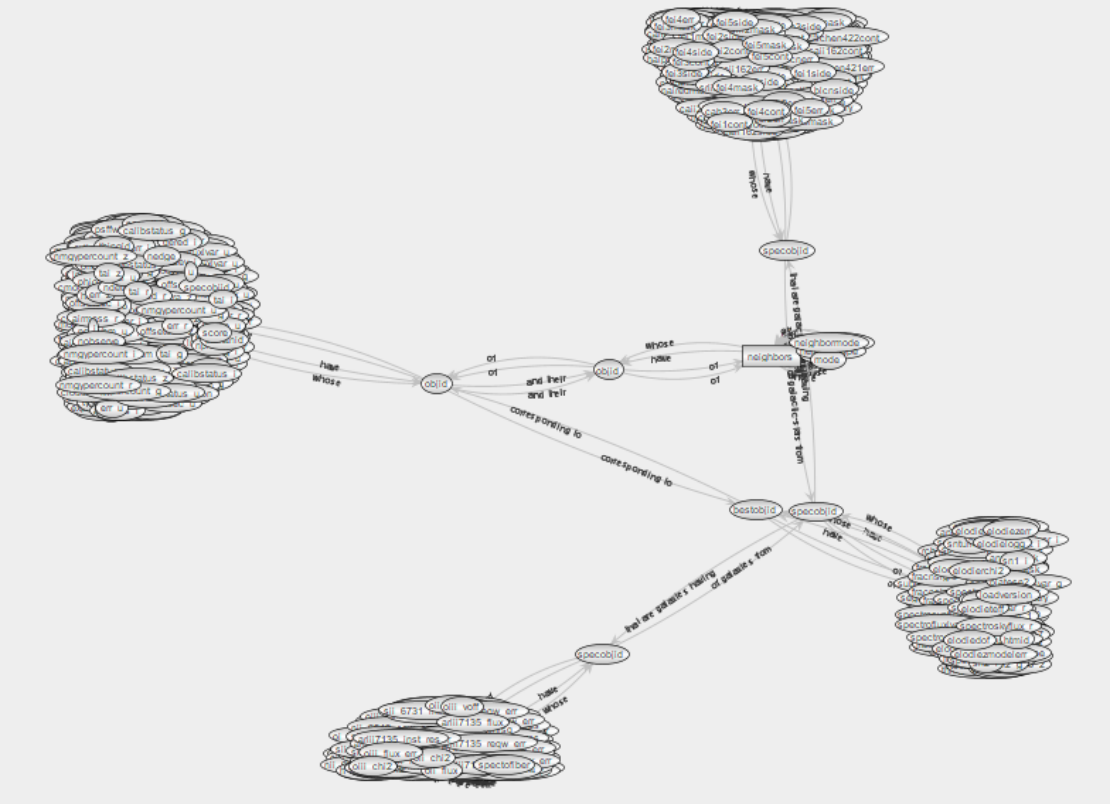


Figure 5.3: SDSS database graph created by Logos.

5.3 Cancer Research (OncoMX)

The OncoMX relational data schema slightly changed, to be better compliant with version 1_0_25 of the OncoMX original datasets[36]. Moreover, we did some patches to correct minor issues in the database schema and data used to populate it. The main modifications are listed in Table 5.1. Furthermore, the latest OncoMX Extended Entity-Relationship (EER) diagram is available for download[37].

[36] https://data.oncomx.org
[37] ftp://ftpbgee.unil.ch/inode/oncomx_v1_0_25_SQL_schema.pdf

Table	Modification
project_study	Removed
differential_expression	doid (disease ontology id) column added
differential_expression	study_id column removed
cancer_tissue	Added. It relates a cancer type to an organ.
disease_mutation	Added ensembl_transcript_id as a foreign key to relate disease_mutation with map_protein_disease_mutation table

Table 5.1: The main changes in the OncoMX relational data model.

We also improved the cancer biomarker ontology (CBIO) developed by us to better describe the OncoMX data. CBIO does not aim to solely serve as a controlled vocabulary but also as a data schema in the cancer biomarker domain. In addition to OWL 2 built-ins (e.g., *owl:sameAs*), we use SKOS[38] terms such as *skos:narrowMatch* to define abstract mappings with external ontologies that often acts as controlled vocabularies. For example, the CBIO term

MetabolomicBiomarker skos:narrowMatch <http://purl.obolibrary.org/obo/NCIT_C18520>

is an ontology term for the Metabolic Marker definition from National Cancer Institute Thesaurus.

Moreover, CBIO can be visualised with the WebVOWL tool[39]. Figure 5.4 illustrates a portion of CBIO by using the WebVOWL tool and Figure 5.5 shows part of the biomarker class hierarchy. Figure 5.6 demonstrates an intersection node (i.e., *Gene* node) in the OncoMX knowledge graph generated with Ontop to relate/link the different OncoMX datasets via "has sequence unit" property assertions.

[38] SKOS Simple Knowledge Organization System, https://www.w3.org/2004/02/skos/.

[39] http://www.visualdataweb.de/webvowl/#iri=http://purl.org/cbio

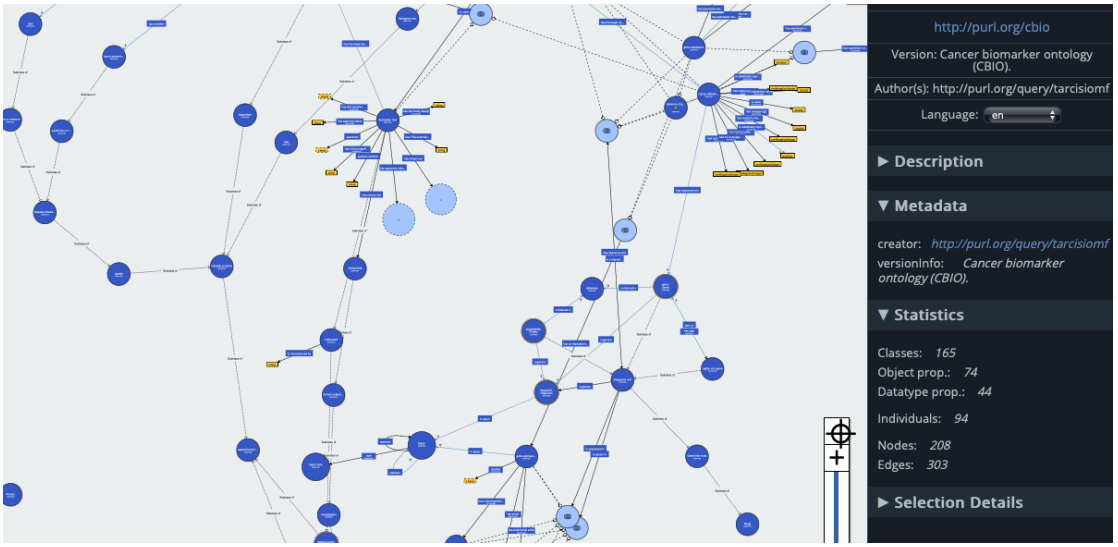


Figure 5.4: Visualizing the cancer biomarker ontology with the WebVOWLtool[32].Currently, it is composed of 165 classes and 118 properties.

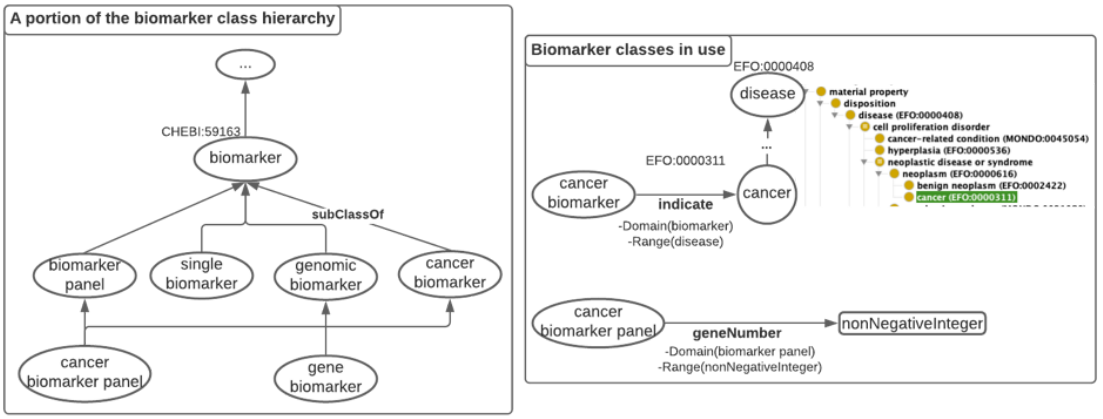


Figure 5.5: Biomarker ontological terms.

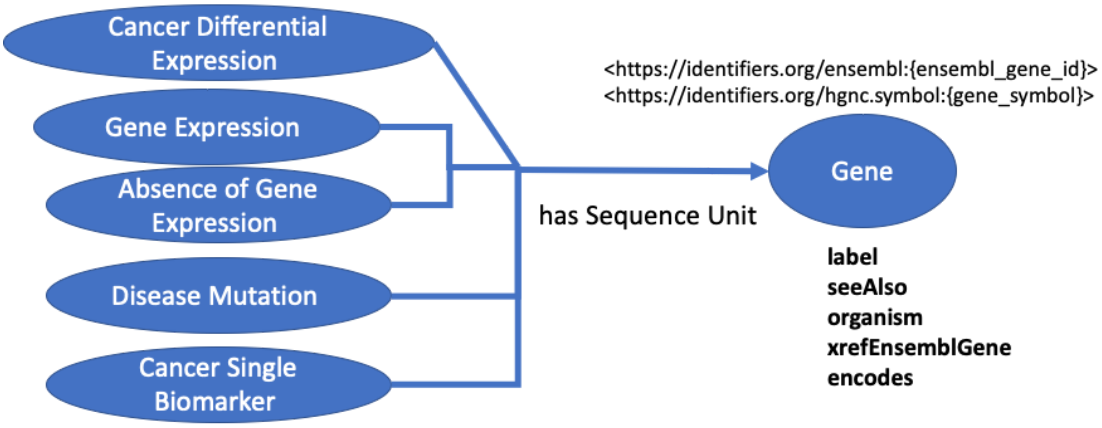


Figure 5.6: Gene instances as an intersection node among different OncoMX datasets in the OncoMX knowledge graph generated with Ontop. Edges represent "has sequence unit" property assertions and nodes are instances of the respectives named OWL classes. Bold terms below the gene node list examples of gene attributes.