



1st Test Report

Document Due Date: 30/04/2021
Document Submission Date: 30/04/2021

Work Package 9

Type: Report

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 1st test report over all services of the project.

- Section 1 describes the automatic test procedures for INODE-SQL 2.0
- Section 2 describes the automatic test procedures for INODE-SPARQL 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

Authors(s)	<p>Koutrika Georgia, Eleftheraki Stavroula, Glenis Apostolis, Mandamadiotis Antonis (ATHENA)</p> <p>Amer-Yahia Sihem, Patil Yogendra, Personnaz Aurélien (CNRS)</p> <p>Lücke-Tieke Hendrik, May Thorsten (Fraunhofer)</p> <p>Litke Antonis, Papadakis Nikolaos, Papadopoulos Dimitris (Infili)</p> <p>Fabircius Maximilian, Subramanian Srividya (MPE)</p> <p>Bastian Frederic, Mendes de Farias Tarcisio, Stockinger Heinz (SIB)</p> <p>Massucci Francesco, Multari Francesco, Rull Guillem (SIRIS)</p> <p>Calvanese Diego, Lanti Davide, Mosca Alesandro, Guohui Xiao (UNIBZ)</p> <p>Braschler Martin, Brunner Ursin, Kosten Catherine, Smith Ellery, Stockinger Kurt (ZHAW)</p>
-------------------	--

Table of Contents

1 Testing INODE-SQL 2.0	5
1.1 Architecture of the Automatic Testing System	5
1.2 Structure of the Test Case Specification File	7
1.3 Example of Test Case Specification	8
1.4 Summary of Test Results	12
1.4.1 Cordis dataset	13
1.4.2 SDSS dataset	19
2 Testing INODE-SPARQL 1.0	22
2.1 Architecture of the Automatic Testing System	22
2.2 Structure of the Test Case Specification File	23
2.3 Example of Test Case Specification	24
2.4 Summary of Test Results	25
2.4.1 CORDIS SPARQL endpoint	25
2.4.2 SDSS SPARQL endpoint	28
2.4.3 OncoMX SPARQL endpoint	29
2.4.4 Bio-SODA for CORDIS	30
2.4.5 Bio-SODA for SDSS	33

1 TESTING INODE-SQL 2.0

In this section we describe the automatic integration testing performed on the INODE-SQL 2.0 component of the INODE system, which integrates the different services that deal with SQL data sources. This component is implemented by means of the OpenDataDialog 2.0 application. The integration tests target the backend of this application and simulate the different steps of a user interaction.

1.1 Architecture of the Automatic Testing System

The aim of the current testing system is to perform an end-to-end check of the INODE-SQL 2.0 system and verify that the different services integrated by the system work together to produce an answer to the user's requests.

We focus on testing the functionality of the system rather than its performance, which will be tested later in the project, since the development of the integrated services has also been, in general, more focused on providing functionality than performance during this first half of the project. We still report execution times, but these are merely informative and do not determine a test's failure or success. The only exception is a generous 20-minute timeout on a test's execution, which may cause it to fail. The purpose of this timeout is to guarantee that the testing application will finish in a reasonable amount of time, and to prevent tests from getting stuck on infinite loops or similar situations.

The architecture depicted in Figure 1.1 shows the testing application with its input and output, namely a test case specification in JSON format as input and an HTML report of the test results as output. The testing application executes the test cases specified in its input by communicating with the OpenDataDialog's backend. The OpenDataDialog 2.0 software is the current interface of the INODE-SQL 2.0 system. It has both a frontend and backend component. However, for the purpose of making the tests run automatically, it is more convenient to use the backend, since the frontend requires users to hover the mouse over certain areas to trigger contextual menus and tooltips, which is difficult to automatize.

The OpenDataDialog backend exposes an API that is called by the testing application in order to simulate the user interactions that are encoded in the test case specification. In turn, the API communicates with the different services that are integrated into the INODE-SQL system. These services are:

- Natural Language-to-SQL service (implemented by the combination of the SODA and ValueNet services)
- SQL-to-Natural Language service (implemented by the LOGOS service)
- Pipeline Operators service
- Query Recommendation service (implemented by the PyExplore service)

Each of these services provides one or more data exploration operators for the user to use. A typical user interaction with INODE-SQL via the OpenDataDialog application consists of a sequence of steps. At each step, the user chooses an operator to apply, and the system responds with a set of tables. Each table is the result of a SQL statement's execution. The SQL

statement itself is also reported back to the user together with a corresponding natural language explanation.

The first operator in a user interaction is always *by-nl*, in which the user provides a natural language query for the system to interpret. Subsequent operators will be applied on one of the tables produced by the previous operator. The *by-nl* operator can also be chained after another operator, which is done by the user editing the natural language explanation of one of the tables produced by the previous operator.

To simulate this kind of user interactions, the test case specification defines each test case as a sequence of operator tests, each of which indicates a SQL statement that is expected to be produced by the previous operator.

The specific test cases have been defined during the phase of manual testing of the OpenDataDialog frontend. As mentioned above, a user’s interaction with the frontend is difficult to automate, so we have tested it manually and taken the opportunity to identify relevant cases that are worth automating. These relevant cases are meant to be executed periodically to check whether changes to the services code break the system or not, and also whether known limitations of the services have been addressed.

In terms of datasets, the test cases cover the CORDIS and SDSS datasets, which correspond to the “R&I Policy Making” and “Astrophysics” use cases of INODE. The “Cancer Research” use case with its dataset, OncoMX, has not yet been integrated into INODE-SQL, but it is part of INODE-SPARQL, so its testing will be addressed in Section 2.

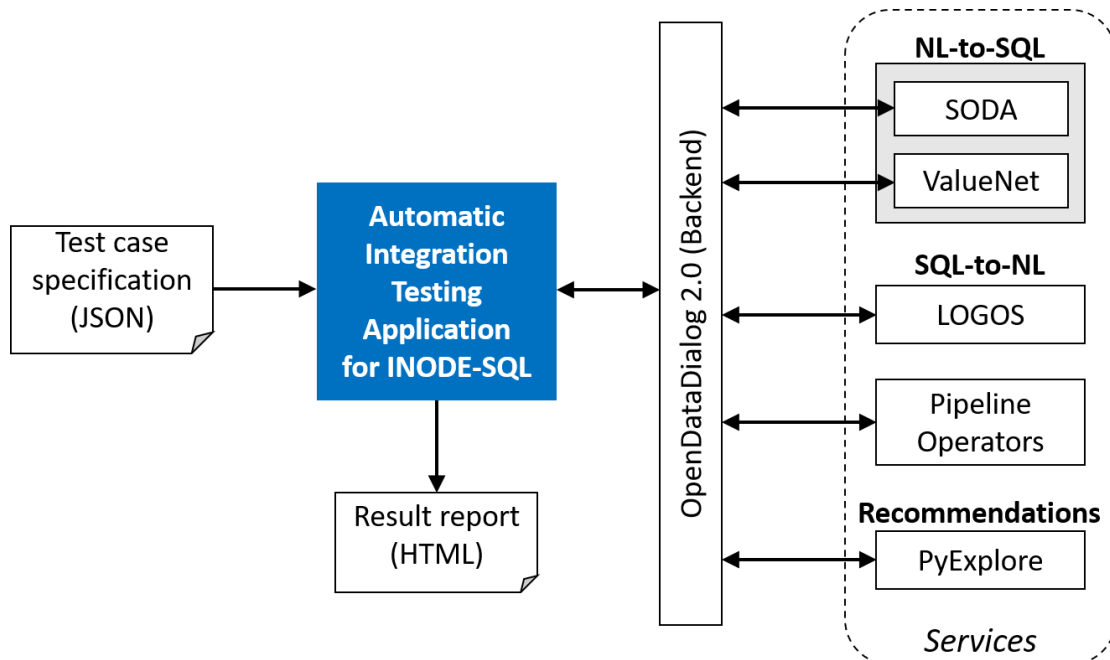


Figure 1.1. Architecture of the testing system for INODE SQL 2.0
 Input: Test case specification in JSON format. Output: HTML report of the test results.

1.2 Structure of the Test Case Specification File

The test cases are specified in JSON format, with the root of the specification file being a JSON object of the form:

```
{ "testCases": [ testCase1, testCase2, ... ] }
```

Each $testCase_i$ is a nested object that holds the name of the test case, the database on which the test case will be run, and the sequence of operators that form the test case:

```
{  
  "name": string,  
  "database": string,  
  "operators": [ operator1, operator2, ... ]  
}
```

Each $operator_i$ is also a nested object, which has the following general structure:

```
{  
  "type": string,  
  "inputQuery": string,  
  "params": { ... }  
}
```

The type of the operator can be one of the following:

- *by-nl*
- *by-recommendation*
- *by-superset*
- *by-facet*
- *by-filter*

The input query is a SQL statement for all operators with the exception of *by-nl*. In the case of *by-nl*, the input query is a string that contains both the natural language version of the query and the keyword-based version, separated by a "|" symbol :

natural language query | keyword-based query

The natural language version of the query is used as input for ValueNet, while the keyword-based version of the same query is used as input for SODA.

The input query is also interpreted as an expected output of the previous operator within the same test case, with the exception of the first operator.

Since the *by-nl* operator is the only one that generates SQL statements as output but does not require a SQL statement as input, it will always be used as the first operator in all test cases. This is because the natural language/keyword query that the *by-nl* operator takes as input is given by the user. It is also worth mentioning that *by-nl* can be applied again later in the sequence of operators that forms a test case. This simulates the user's editing of the natural language explanation that accompanies a SQL statement produced by the previous operator. As a result, the edited natural language expression will be processed by the system as a natural language query.

The "params" attribute is used to hold any additional parameters that are specific to certain operators. For the operators *by-nl*, *by-recommendation* and *by-superset*, the "params" attribute is left as an empty object, since they do not require additional parameters. In the case of the operator *by-facet*, an additional parameter is required to indicate which of the

columns returned by the input query is to be used to define groups. This is done by adding a “column” attribute to the “params” object with the name of the column as its value:

```
“params”: { “column”: string }
```

Similarly, the *by-facet* operator requires additional parameters to indicate the column, value and data type to be used as a filter. The filter in question will be a condition of the form: column = value. These three elements are specified by means of adding the attributes “column”, “value” and “type” to the “params” object:

```
“params”: {  
  “column”: string,  
  “value”: string,  
  “type”: string  
}
```

The accepted data types to be used as values for the “params.”type” attribute are “String” and “Number”. The data type refers to both the column and value used in the filter, that is, the column and value should be of the same type so that they can be safely compared.

1.3 Example of Test Case Specification

In this section, we show an example of a user interaction with INODE-SQL and how this would be encoded using the specification format defined in the previous section.

A typical user interaction with INODE-SQL begins with the selection of the database to work with and the introduction of a natural language query, all via the OpenDataDialog application’s frontend as shown in Figure 1.2. The user can also choose the natural language services to use in the query’s processing. However, for the purpose of testing, we focus on SODA and ValueNet, since these are the two systems developed within the INODE project. Therefore, we assume the user unchecks the Nalir+ system.

This first step in the user interaction is encoded in the test case specification by means of adding a new JSON object to the array of test cases. This object holds the “database” parameter that will be common to all the operators applied by the user during the test case. For this example, we assume the user chooses to use the CORDIS dataset:

```
{  
  “testCases”: [  
    {  
      “name”: “Example Test Case”,  
      “database”: “cordis”,  
      “operators”: []  
    }  
  ]  
}
```

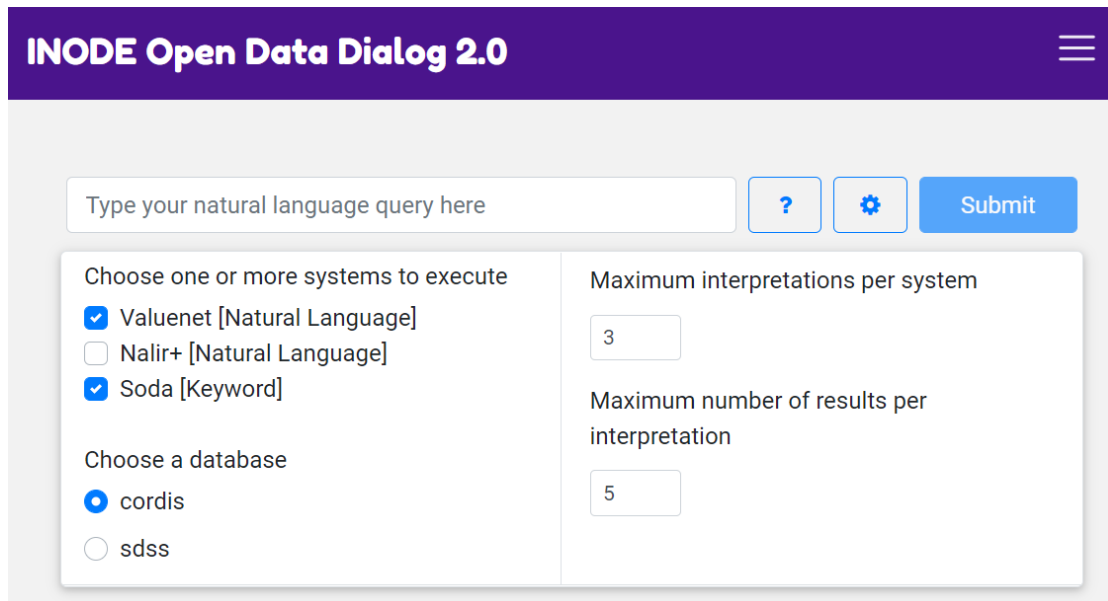


Figure 1.2. Manual application of the *by-nl* operator by a user.

Regarding the natural language query that kickstarts the data exploration, we assume the user introduces the query “Find projects where Alberto Broggi was involved”, so we encode this by adding a nested object into the “operators” array:

```

"operators": [
  {
    "type": "by-nl",
    "inputQuery": "Find projects where Alberto Broggi was involved|Find
projects with Alberto BROGGI",
    "params": {}
  }
]
    
```

The execution of the *by-nl* operator returns a set of tables to the user, which in the frontend are visualized by means of the multi-table explorer component as shown in Figure 1.3.

For the next step, we assume the user wants to apply the *by-superset* operator on the first table that was returned by the natural language query. To do this in the frontend, the user uses the contextual menu of the first table and clicks on the “Explore by superset” option (see Figure 1.4).

To simulate this application of *by-superset* in the automated test case, we add a new operator JSON object into the “operators” array, right after the object for *by-nl* that we added before:

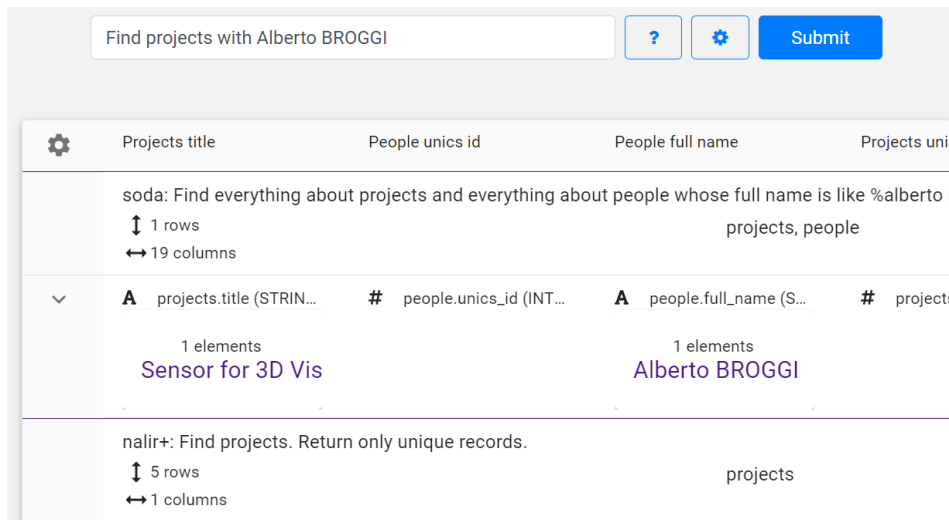


Figure 1.3. Result of *by-nl* operator in OpenDataDialog's frontend.

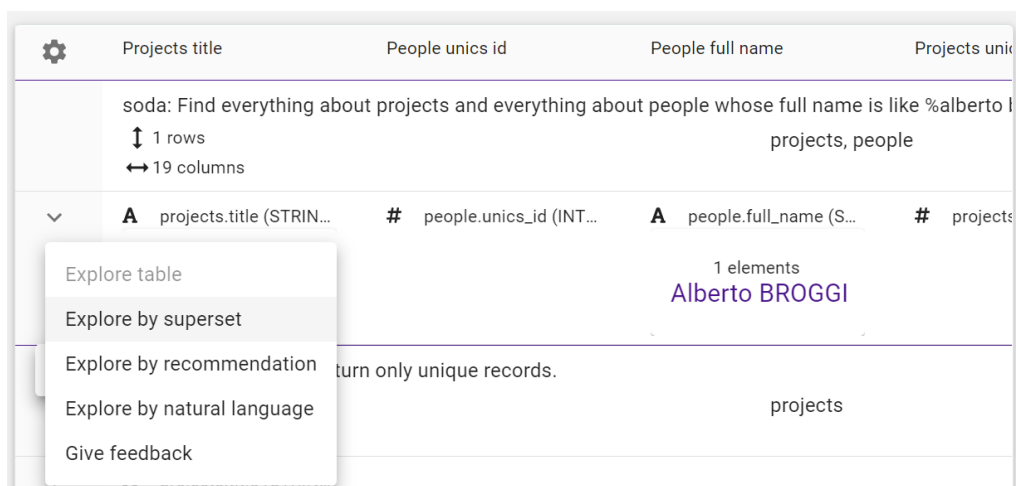


Figure 1.4. A user's manual application of *by-superset* on a table.

```

"operators": [
  ...,
  {
    "type": "by-superset",
    "inputQuery": "SELECT * FROM people, projects WHERE ((people.full_name
like '%Alberto BROGGI%')) AND
(people.unics_id=projects.principal_investigator)",
    "params": {}
  }
]
    
```

The JSON object for the *by-superset* operator contains as “inputQuery” the SQL statement that generated the selected table by the user. In the frontend, the user can see this SQL statement by hovering the mouse over the natural language explanation of the table (see Figure 1.5).

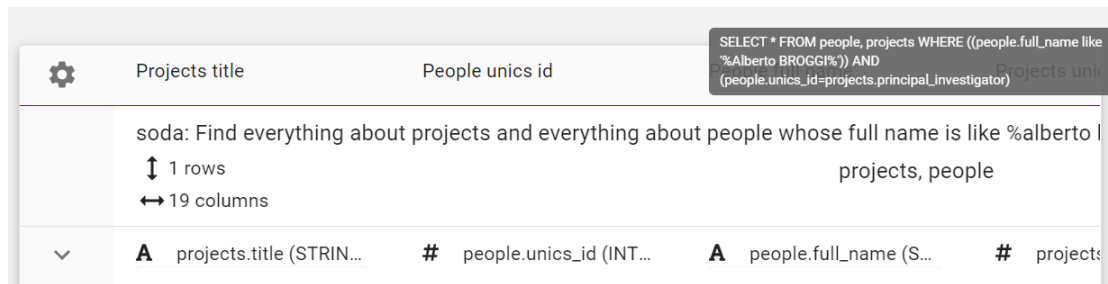


Figure 1.5. The SQL statement that generated a table is shown as a tooltip when hovering the mouse over the table’s natural language explanation.

The execution of *by-superset* produces a new set of tables that replaces the previous one on the multi-table explorer. In this particular case, there is only one table in the set.

At this point, the user can apply a new operator on the returned table. For example, let us assume that the user wants to apply the *by-facet* operator on the “projects.ec_call” column. To do this, the user opens the contextual menu for the corresponding column and clicks on the “Explore by facet” option (see Figure 1.6).

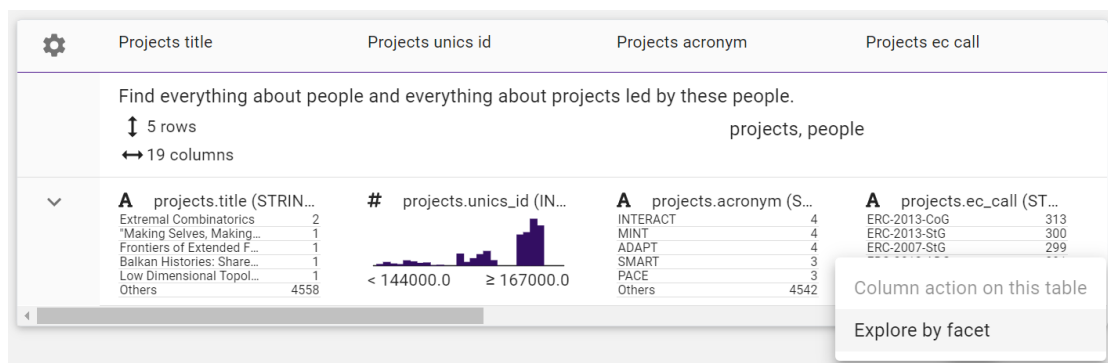


Figure 1.6. A manual application of the *by-facet* operator.

This interaction is encoded into the test case specification by adding a new JSON object into the “operators” array, following the objects that were previously added for *by-nl* and *by-superset*:

```

"operators": [
  ...
  {
    "type": "by-facet",
    "inputQuery": "SELECT * FROM projects, people WHERE
      projects.principal_investigator = people.unics_id",
    "params": { "column": "projects.ec_call" }
  }
]
    
```

As in the previous case, the input query for the *by-facet* operator is the SQL statement that corresponds to the table to which the column we are applying the operator belongs to. Additionally, *by-facet* has an extra parameter “column” that is added into the “params” object. This extra parameter indicates the name of the table’s column on which we are applying the operator.

To conclude the example, let us assume that the user applies a final operator to one of the tables obtained from the *by-facet* operator. The operator is *by-filter*, and it is applied on a specific cell of the table. The filter will apply a condition “column = value” where the column and value are determined by the cell on which the operator is applied. In the frontend interface, the user applies the *by-filter* operator by using the contextual menu of the corresponding table cell and choosing the option “Explore by filter” (see Figure 1.7).

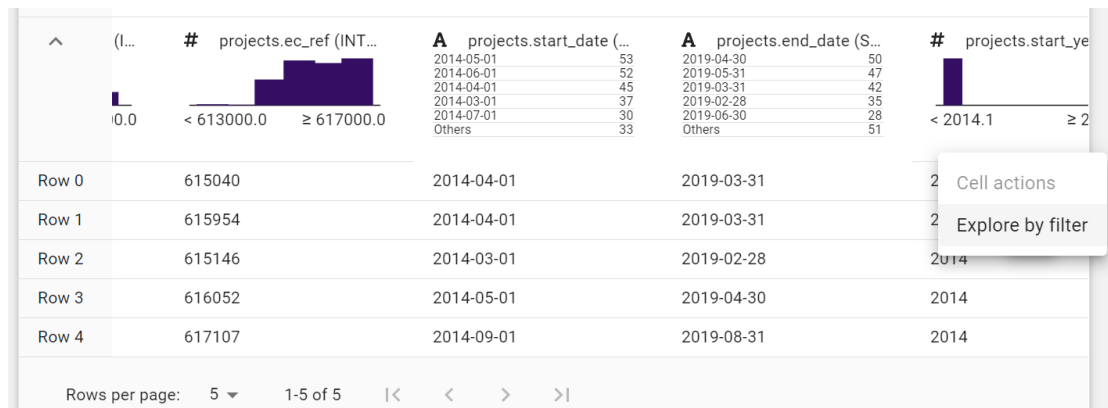


Figure 1.7. A user applying the *by-filter* operator on a table’s cell.

Similar to the previous operators, we can encode this final user interaction into the test case specification by adding another JSON object at the end of the “operators” array:

```

"operators": [
  ...,
  {
    "type": "by-filter",
    "inputQuery": "SELECT * FROM projects, people WHERE
projects.principal_investigator = people.unics_id AND projects.ec_call =
'ERC-2013-CoG'",
    "params": {
      "column": "projects.start_year",
      "value": "2014",
      "type": "Number"
    }
  }
]
    
```

In this case, the operator requires three extra parameters, namely the column, value and data type of the table cell on which we are applying the operator. These are added to the “params” object as three separate attributes. The input query, as in the previous two operators, corresponds to the SQL statement that generated the table in question.

1.4 Summary of Test Results

This section summarizes the results of the test cases that have been performed on the INODE-SQL 2.0 system for the CORDIS and SDSS datasets.

For each test case, we show a table with the results of the operators involved. We indicate the type of operator, the input query, the status of the test, the execution time (measured in seconds) and the result. The status of the test can be either OK, ERROR or WARNING.

Regarding the result, we indicate how many tables were produced by the operator or, in case the operator failed, the error message.

Since the system is still a work in progress, some test cases are expected to fail, as they point out issues that will be improved and worked upon during the remainder of the project.

We can see a warning in Test Case 1, which is due to the current non-deterministic nature of the *by-recommendation* operator. This means that the exact set of recommendations returned by the operator may change slightly between executions even though the input query remains the same. This causes the verification step at the beginning of each operator test to fail, since the input query of the operator that is applied after *by-recommendation* is not found in the *by-recommendation's* output. We mark this as a warning rather than an error, since it is an effect of the automation of the use case and it would not happen in a live situation where a human user is interacting with the system.

As mentioned, the test cases include some examples of operators receiving input queries that are still not supported, which causes them to fail. We can see this for the case of *by-recommendation* in Test Case 2, for the SQL-to-Natural Language service (which is executed behind the scenes for each operator) in Test Case 4 and Test Case 9, and for ValueNet (*by-nl* operator) in Test Case 8. These are known limitations of the current release that will be improved during the second half of the project, when these test cases are expected to go from ERROR to OK.

1.4.1 Cordis dataset

Test Case 1

Operator	Query	Status	Time	Result
by-nl	Find all projects Find all projects	OK	2.2116389 second(s)	4 table(s)
by-recommendation	SELECT * FROM project_subject_areas	OK	5.6037486 second(s)	7 table(s)
by-superset	SELECT * FROM project_subject_areas where project < 160324.5 and project >= 151707.5	WARNING	0.4760852 second(s)	1 table(s) Operator's input query not present in previous operator's output

by-filter	SELECT * FROM project_subject_areas	OK	0.2006989 second(s)	1 table(s)
-----------	-------------------------------------	----	---------------------	------------

Test Case 2

Operator	Query	Status	Time	Result
by-nl	Find projects that started after 2018 project start_year >2018	OK	6.392262 second(s)	4 table(s)
by-facet	SELECT * FROM project_erc_panels, projects WHERE ((projects.start_year>2018)) AND (project_erc_panels.project=projects.unics_id)	OK	1.102569 6 second(s)	5 table(s)
by-facet	SELECT * FROM projects, project_erc_panels WHERE projects.unics_id = project_erc_panels.project AND projects.start_year = 2018.0 and project_erc_panels.panel = 'PE5'	OK	0.442312 4 second(s)	4 table(s)
by-superset	SELECT * FROM projects, project_erc_panels WHERE projects.unics_id = project_erc_panels.project AND projects.start_year = 2018.0 and project_erc_panels.panel = 'PE5' and projects.ec_call = 'ERC-2017-STG'	OK	0.270621 4 second(s)	1 table(s)
by-recommendation	SELECT * FROM projects, project_erc_panels WHERE projects.unics_id = project_erc_panels.project AND	ERROR		Unsupported input query

project_erc_panels.panel =
'PE5' and projects.ec_call =
'ERC-2017-STG'

Test Case 3

Operator	Query	Status	Time	Result
by-nl	Find projects that started before 2018 project start_year < 2018	OK	413.6103965 second(s)	4 table(s)
by-recommendation	SELECT * FROM project_members, projects WHERE ((projects.start_year<2018)) AND (project_members.project=projects.unics_id)	OK	920.067432 second(s)	23 table(s)

Test Case 4

Operator	Query	Status	Time	Result
by-nl	Find institutions located in Italy Find institutions Italy	OK	11.8196104 second(s)	3 table(s)
by-superset	SELECT T1.institutions_name FROM institutions AS T1 JOIN countries AS T2 ON T1.country_id = T2.unics_id WHERE T2.country_name = 'Italy'	OK	1.7350534 second(s)	1 table(s)
by-recommendation	SELECT * FROM institutions, countries WHERE institutions.country_id = countries.unics_id	ERROR		Missing NL explanation

Test Case 5

Operator	Query	Status	Time	Result
by-nl	Find projects where Alberto Broggi was involved Find projects with Alberto BROGGI	OK	4.2821569 second(s)	2 table(s)
by-superset	SELECT * FROM people, projects WHERE ((people.full_name like '%Alberto BROGGI%')) AND (people.unics_id=projects.principal_investigator)	OK	1.3783319 second(s)	1 table(s)
by-facet	SELECT * FROM projects, people WHERE projects.principal_investigator = people.unics_id	OK	2.4823867 second(s)	5 table(s)
by-filter	SELECT * FROM projects, people WHERE projects.principal_investigator = people.unics_id AND projects.ec_call = 'ERC-2013-CoG'	OK	0.3047024 second(s)	1 table(s)

Test Case 6

Operator	Query	Status	Time	Result
by-nl	How many projects started in 2016? count(project) start_year = 2016	OK	3.5499375 second(s)	4 table(s)
by-nl	Find projects whose start year is 2016 Find projects whose start year is 2016	OK	120.1183993 second(s)	4 table(s)

Test Case 7

Operator	Query	Status	Time	Result
by-nl	What's the total cost of all projects? projects sum(total_cost)	OK	3.3498224 second(s)	4 table(s)
by-nl	Find the sum of total costs of projects started in 2016 Find the sum of total costs of projects started in 2016	OK	146.1818575 second(s)	4 table(s)
by-superset	SELECT sum(T1.total_cost) FROM projects AS T1 WHERE T1.start_year = 2016	OK	77.4844296 second(s)	1 table(s)

Test Case 8

Operator	Query	Status	Time	Result
by-nl	Find the topics of projects that ended in 2014 project_topics end_year = 2014	ERROR		ValueNet failed to answer

Test Case 9

Operator	Query	Status	Time	Result
by-nl	Find projects with a member from Greece projects name=Greece	OK	3.7594545 second(s)	2 table(s)
by-nl	Find the title and starting year of projects with a member from Greece Find the title and starting year of projects with a member from Greece	ERROR		Missing NL explanation

Test Case 10

Operator	Query	Status	Time	Result
by-nl	List all projects under the FP7-ICT programme projects programme FP7-ICT	OK	3.240206 second(s)	3 table(s)
by-superset	SELECT T1.project FROM project_programmes AS T1 WHERE T1.programme = 'FP7-ICT'	OK	0.4607939 second(s)	1 table(s)
by-recommendation	SELECT * FROM project_programmes	OK	3.3189414 second(s)	4 table(s)

Test Case 11

Operator	Query	Status	Time	Result
by-nl	Find all projects connected with institutes located in Schaffhausen area projects institutes eu_territorial_units Schaffhausen	OK	140.1998392 second(s)	4 table(s)
by-filter	SELECT * FROM eu_territorial_units, project_members, projects WHERE (eu_territorial_units.nuts_code=project_members.nuts3_code) AND (project_members.project=projects.unics_id)	OK	0.7564124 second(s)	1 table(s)
by-facet	SELECT * FROM project_members, projects, eu_territorial_units WHERE project_members.project = projects.unics_id AND project_members.nuts3_code = eu_territorial_units.nuts_code AND eu_territorial_units.nuts_code = 'NL329'	OK	2.5711141 second(s)	5 table(s)

1.4.2 SDSS dataset

Test Case 12

Operator	Query	Status	Time	Result
by-nl	count(type) type= 6	OK	66.2953099 second(s)	3 table(s)
by-nl	type=6	OK	11.4440518 second(s)	3 table(s)
by-filter	SELECT * FROM photoobj WHERE ((photoobj.type_i=6))	OK	7.8943731 second(s)	1 table(s)

Test Case 13

Operator	Query	Status	Time	Result
by-nl	count(all) survey segue2	OK	10.1853495 second(s)	3 table(s)
by-nl	survey segue2	OK	8.8422255 second(s)	3 table(s)
by-facet	SELECT * FROM specobj WHERE ((specobj.programname like '%segue2%'))	OK	39.0309774 second(s)	5 table(s)

Test Case 14

Operator	Query	Status	Time	Result
by-nl	type= 3	OK	12.1450108 second(s)	3 table(s)
by-facet	SELECT * FROM photoobj WHERE ((photoobj.type=3))	OK	32.5659334 second(s)	5 table(s)

by-recommendation	select photoobj.objid, photoobj.type, photoobj.clean, photoobj.ra, photoobj.dec, photoobj.u, photoobj.g, photoobj.r, photoobj.i, photoobj.z FROM photoobj WHERE photoobj.type = 3.0 and photoobj.ra > 137.13114622077 and photoobj.ra <= 156.015151078751	OK	36.7535051 second(s)	17 table(s)
-------------------	--	----	-------------------------	----------------

Test Case 15

Operator	Query	Status	Time	Result
by-nl	specobjid galspecline	OK	7.6288615 second(s)	2 table(s)
by-superset	SELECT * FROM galspecline, specobj WHERE (galspecline.specobjid=specobj. specobjid)	OK	4.6976364 second(s)	1 table(s)
by-facet	SELECT * FROM specobj, galspecline WHERE specobj.specobjid = galspecline.specobjid	OK	51.3292684 second(s)	5 table(s)

Test Case 16

Operator	Query	Status	Time	Result
by-nl	photoobj type =6	OK	17.0871967 second(s)	3 table(s)
by-recommendation	select photoobj.objid, photoobj.type, photoobj.clean, photoobj.ra, photoobj.dec, photoobj.u, photoobj.g, photoobj.r, photoobj.i, photoobj.z FROM photoobj WHERE ((photoobj.type_z=6))	OK	121.5473569 second(s)	17 table(s)

by-superset	<pre>select photoobj.objid, photoobj.type, photoobj.clean, photoobj.ra, photoobj.dec, photoobj.u, photoobj.g, photoobj.r, photoobj.i, photoobj.z FROM photoobj WHERE ((photoobj.type_z=6)) and clean < 0.5</pre>	OK	32.8233948 second(s)	1 table(s)
-------------	---	----	-------------------------	---------------

2 TESTING INODE-SPARQL 1.0

In this section we describe the automatic testing of the INODE-SPARQL services. These services have not yet been integrated into a single application, so we test them separately. As with INODE-SQL, we focus here on functionality and leave performance testing as future work. We do report execution times as part of the test results, but these are merely informative.

The SPARQL services that are presently available are the endpoints for the three application domains of INODE, and the Bio-SODA service for answering natural language queries over RDF data.

We focus on testing the online components of INODE-SPARQL. The offline work is tested indirectly since this work has a direct impact on the state of the online components. Offline work includes:

- Ontology and mapping construction for the three use cases.
- Enrichment of the datasets by extracting data from unstructured text.

The ontology and mapping construction is essential to the inner workings of the SPARQL endpoints. These are powered by the Ontop software, which exposes the underlying relational data as a virtual knowledge graph using a mapping that connects each database with the corresponding ontology.

The information extraction from unstructured text has resulted in new information being added into the datasets. This means new tables added to the relational databases, but also new classes and properties added to the ontologies to reflect this new data.

2.1 Architecture of the Automatic Testing System

The testing application runs queries against the SPARQL endpoints of the three domains of application, and also against the two instances of Bio-SODA that are currently available for the “R&I Policy Making” (CORDIS dataset) and “Astrophysics” (SDSS dataset) use cases. The Bio-SODA endpoint for “Cancer Research” will be developed during the second half of the project.

The test cases are specified in a single JSON file that the testing application takes as input. Each test case defines either a SPARQL query to be executed on one of the endpoints, or a natural language query to be sent to one of the Bio-SODA instances. The results of the tests are published as an HTML report.

The testing application executes the test cases sequentially, with a 20-minute timeout. If the corresponding SPARQL or natural language query succeeds and produces a non-empty result, the test passes. Otherwise, if the query’s execution ends up returning an error, or if the timeout is reached, then the test fails.

This architecture is graphically depicted in Figure 2.1.

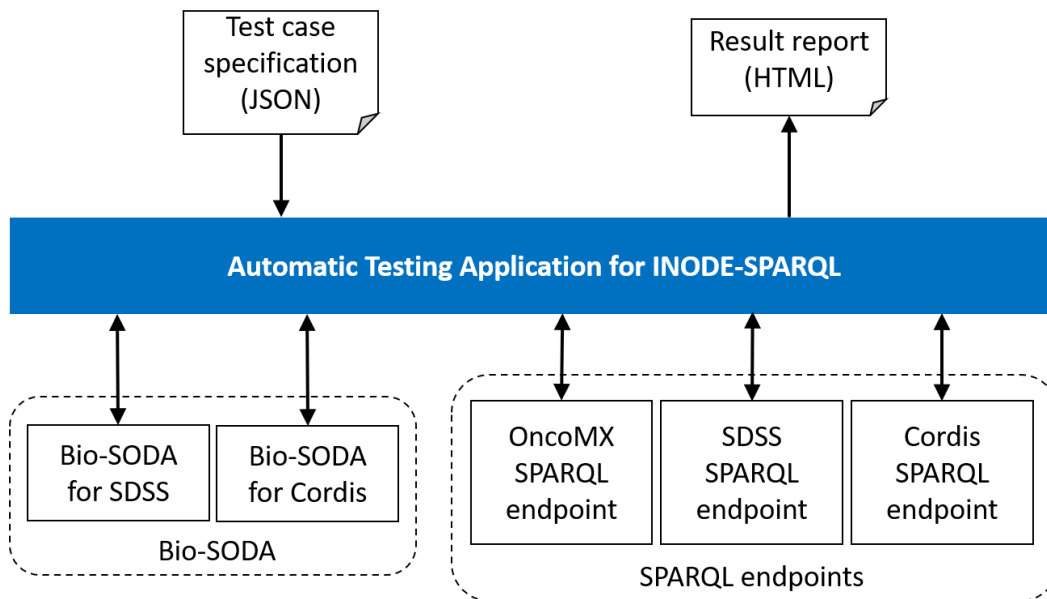


Figure 2.1. Architecture of the testing system for INODE-SPARQL 1.0.
Input: Test case specification in JSON format. Output: HTML report of the test results.

2.2 Structure of the Test Case Specification File

The test cases for INODE-SPARQL are specified all together in a single JSON file. The contents of this file is a JSON object with the following structure:

```
{ "testGroups": [ testGroup1, testGroup2, ... ] }
```

where each testGroup_i is a nested object in the form of:

```
{
  "name": string
  "type": string
  "url": string,
  "testCases": [ testCase1, testCase2, ... ]
}
```

A test group represents the set of test cases of a particular component of INODE-SPARQL. It is identified by a name, and has two properties that are common to all the included test cases, namely the type and url of the component being tested. The type can be either “sparql” or “biosoda”, while the url specifies which SPARQL endpoint or Bio-SODA instance the test group refers to.

The test cases inside a test group are also specified by means of nested objects. These hold the details that are specific to each test case. Each testCase_i has the following form:

```
{
  "description": string,
  "query": string
}
```


The description of a test case is a natural language explanation of the SPARQL query's meaning, while the "query" attribute holds the actual SPARQL query.

2.3 Example of Test Case Specification

The test cases targeted at the SPARQL endpoints simulate the user going into the corresponding endpoint's web page and manually writing and executing a given SPARQL query. For example, for the SDSS dataset, a user can go into the endpoint and write a query to do a rectangular search of the sky using straight coordinate constraints as shown in Figure 2.2.

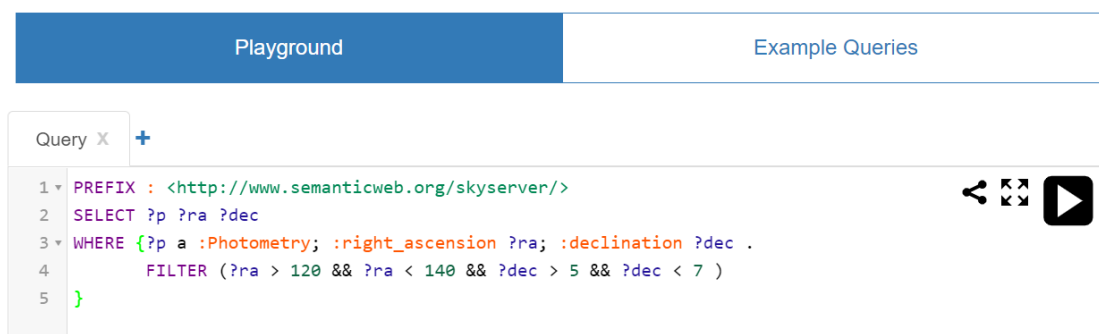


Figure 2.2. A manually written SPARQL query on the SDSS SPARQL endpoint.

This test case would be specified by means of defining a test group that targets the SDSS SPARQL endpoint and includes the appropriate query:

```

{
  "testGroups": [
    {
      "name": "SDSS SPARQL endpoint",
      "type": "sparql",
      "url": "url of the SDSS SPARQL endpoint",
      "testCases": [
        {
          "description": "Rectangular search using straight coordinate constraints",
          "query": "PREFIX : <http://www.semanticweb.org/skyserver/>
SELECT ?q ?ra ?dec ?mag_i ?mag_r ?redshift
WHERE {?q a :SpecQuasar;
:right_ascension ?ra; :declination ?dec;
:redshift ?redshift;
:hasPhotometry ?pq;
:hasSpectroWarning \"no\" .
?pq :psfmagnitude_i ?psfmag_i;
:extinction_i ?extinction_i;
:psfmagnitude_r ?psfmag_r;
:extinction_r ?extinction_r;
BIND((?psfmag_i-?rextinction_i) AS ?mag_i)
BIND((?psfmag_r-?extinction_r) AS ?mag_r) } LIMIT 100"
        }
      ]
    }
  ]
}
    
```

The SPARQL query is provided in the nested JSON object’s “query” attribute, while a textual description of the query’s meaning is given in the “description” attribute.

Similarly, the test cases for Bio-SODA simulate a user going into the corresponding Bio-SODA web page and typing a natural language query as shown in Figure 2.3.

Welcome!

BioSODA (question answering over domain knowledge graphs)

SDSS Demo Page

Figure 2.3. A manually written natural language query on Bio-SODA for the SDSS dataset.

This test case would be specified by means of a test group of type “biosoda” that includes the corresponding natural language query:

```
{
  "testGroups": [
    ...,
    {
      "name": "Bio-SODA for SDSS",
      "type": "biosoda",
      "url": "url of biosoda for SDSS",
      "testCases": [
        {
          "description": "show all hot massive blue stars",
          "query": "show all hot massive blue stars"
        }
      ]
    }
  ]
}
```

In this case, the “description” and “query” attributes of the nested JSON object can be the same since the query is already in natural language and descriptive enough.

2.4 Summary of Test Results

2.4.1 CORDIS SPARQL endpoint

The following table shows the results of the test cases executed on the Ontop-powered SPARQL endpoint of the CORDIS dataset. For brevity, the “Query” column shows the natural language description of the queries rather than the full SPARQL code. The “Status” column indicates the success or failure of the test. The “Time” and “Result” columns show the execution time and the size of the result (in number of rows) of each query, respectively.

We can see three expected test failures that are due to the involved SPARQL queries using the EXISTS and NOT EXISTS filters, which are not yet supported by Ontop.

Query	Status	Time	Result
What is the city of Opel Automobile?	OK	0.9447738 second(s)	2 row(s)
What is the country code of latvia?	OK	0.0494957 second(s)	1 row(s)
Projects funded by the fp7 program	OK	0.9383246 second(s)	25778 row(s)
Projects in the area of mathematics	OK	0.120523 second(s)	239 row(s)
Projects in mass spectrometry	OK	0.1585998 second(s)	16 row(s)
show ERC research domains in the diagnostics tools panel	OK	0.1444262 second(s)	426 row(s)
What are the participants of the project alfred?	OK	0.2885529 second(s)	14 row(s)
Starting year of the project theseus	OK	0.1033094 second(s)	4 row(s)
Organizations in the eawareness project	OK	0.2002004 second(s)	1 row(s)
Ending year of projects in the area of climate change	OK	0.095011 second(s)	62 row(s)
Panels of projects in genome editing	OK	0.1724891 second(s)	5 row(s)
Projects starting in 2019 with the university of zurich	OK	0.1849206 second(s)	17 row(s)
Count the ERC projects in the applied life sciences domain	OK	0.0504244 second(s)	1 row(s)

Topics of projects in life sciences	OK	0.5389166 second(s)	4463 row(s)
Linguistics projects related to the human mind	OK	0.0490751 second(s)	2 row(s)
All projects that started in 2015 in switzerland	OK	0.4493259 second(s)	1066 row(s)
ERC projects whose principal investigator is Michael Smith	OK	0.0808678 second(s)	1 row(s)
Grants received by projects in big data	OK	0.3642301 second(s)	1019 row(s)
Total grants received by projects in the area of materials technology	OK	0.1612239 second(s)	1 row(s)
Projects starting in 2016 whose host is the university of zurich	OK	0.3123766 second(s)	9 row(s)
Full name of principal investigators of projects hosted in france	OK	0.5385834 second(s)	586 row(s)
Titles of erc projects with coordinators and their geographic location	OK	1.6288298 second(s)	11761 row(s)
Universities which are coordinators in climate change projects	OK	0.5964093 second(s)	1877 row(s)
Countries with no projects	ERROR		Unsupported query
Projects with a cost higher than 1 million	OK	1.3043394 second(s)	25744 row(s)
Projects started after November 2019	OK	0.1470067 second(s)	451 row(s)
projects including participants from greece and romania	ERROR		Unsupported query

Projects not including participants from greece nor romania	ERROR		Unsupported query
Find the project with the highest funding	OK	0.8438056 second(s)	1 row(s)
Find the country with the highest number of projects	OK	1.4694185 second(s)	1 row(s)
Find projects similar to the one with acronym EPNET	OK	0.094106 second(s)	3 row(s)

2.4.2 SDSS SPARQL endpoint

The following table shows the results of the tests performed on the Ontop-powered SPARQL endpoint of the SDSS dataset. As in the CORDIS case, the “Query” column shows the natural language description of the query instead of the full SPARQL code.

Query	Status	Time	Result
Find unique objects in an RA/Dec box	OK	0.1315734 second(s)	318 row(s)
Find galaxies with g magnitudes between 18 and 19	OK	0.0507794 second(s)	10 row(s)
Rectangular search using straight coordinate constraints	OK	1.1033571 second(s)	100 row(s)
Retrieve both magnitudes (from photometry) and redshifts (from spectroscopy) of quasars	OK	0.0793449 second(s)	100 row(s)
count the number of spectra of each spectral classification (galaxy, quasar, star)	OK	6.0722554 second(s)	3 row(s)
show all spec galaxies with ascension < 130 declination > 5	OK	5.7851578 second(s)	100 row(s)
show all photo galaxies with magnitude_g <= 23 ascension < 130 declination > 5	OK	11.9255391 second(s)	100 row(s)

show all photo asteroids with mode of photo observation 1	OK	16.6689811 second(s)	2 row(s)
Show white dwarfs with redshift > 0	OK	0.1683359 second(s)	100 row(s)
show all hot massive blue stars	OK	0.2224401 second(s)	100 row(s)
show all spec stars with plate number 1760	OK	0.465288 second(s)	15 row(s)
show all spec stars with the subclass WDhotter	OK	0.0846733 second(s)	100 row(s)
Show the redshifts of all spectroscopies of quasars	OK	0.0564564 second(s)	100 row(s)
show all quasars with ascension > 120 and declination > 5.2	OK	0.0569022 second(s)	100 row(s)
show all star burst galaxies with velocity dispersion > 800	OK	1.3112031 second(s)	100 row(s)

2.4.3 OncoMX SPARQL endpoint

The following table shows the results of the tests performed on the Ontop-powered SPARQL endpoint of the OncoMX dataset. As with the CORDIS and SDSS cases, the “Query” column shows the natural language description of the query instead of the SPARQL code.

Query	Status	Time	Result
Cancer single biomarkers and their descriptions	OK	1.615295 second(s)	931 row(s)
Cancer single biomarkers for breast cancer	OK	0.3787531 second(s)	172 row(s)

Cancer biomarker panels and their descriptions including indicated cancer type	OK	0.5142104 second(s)	162 row(s)
All cancer types in the database	OK	121.1509951 second(s)	43 row(s)
All information about species in the database	OK	0.1016335 second(s)	10 row(s)
What are the cancer types where the A1BG gene expression is increased (up regulated)	OK	30.0814054 second(s)	8 row(s)
What are the cancer types where the A1BG gene expression is statistically significantly increased (up regulated)	OK	43.6702942 second(s)	4 row(s)
What are the healthy organs where the A1BG is expressed	OK	0.1649414 second(s)	74 row(s)
What are the healthy organs in human where the A1BG is not expressed	OK	0.2117581 second(s)	57 row(s)
Biomarkers related to breast at the EDRN phase one	OK	29.2211621 second(s)	18 row(s)
What are the genomic biomarkers for breast cancer?	OK	0.2847917 second(s)	4 row(s)

2.4.4 Bio-SODA for CORDIS

The following table lists the results of the tests performed on Bio-SODA for the CORDIS dataset. The “Query” column reports the natural language query that is the input of Bio-SODA. The “Result” column indicates the number of SQL interpretations found by Bio-SODA.

Query	Status	Time	Result
what is the city of opel automobile?	OK	2.7005263 second(s)	10 interpretation(s)
what is the country code of latvia?	OK	4.0636001 second(s)	10 interpretation(s)
projects funded by the fp7 program	OK	9.294793 second(s)	10 interpretation(s)
projects in the area of mathematics	OK	6.2149723 second(s)	10 interpretation(s)
projects in mass spectrometry	OK	6.1702605 second(s)	10 interpretation(s)
show ERC research domains in the diagnostics tools panel	OK	1.2999919 second(s)	10 interpretation(s)
what are the participants of the project alfred	OK	11.7598201 second(s)	10 interpretation(s)
Starting year of the project theseus	OK	7.383448 second(s)	10 interpretation(s)
organizations in the eawareness project	OK	9.0083834 second(s)	10 interpretation(s)
ending year of projects in the area of climate change	OK	9.5115913 second(s)	10 interpretation(s)
panels of projects in genome editing	OK	6.7781013 second(s)	9 interpretation(s)
projects starting in 2019 with the university of zurich	OK	8.8023423 second(s)	10 interpretation(s)
count the ERC projects in the applied life sciences domain	OK	1.5645649 second(s)	10 interpretation(s)

topics of projects in life sciences	OK	5.9510376 second(s)	10 interpretation(s)
linguistics projects related to the human mind	OK	5.1890979 second(s)	10 interpretation(s)
All projects that started in 2015 in switzerland	OK	9.0910271 second(s)	10 interpretation(s)
ERC projects whose principal investigator is Michael Smith	OK	0.2562195 second(s)	2 interpretation(s)
grants received by projects in big data	OK	5.6144143 second(s)	10 interpretation(s)
total grants received by projects in the area of materials technology	OK	6.7478131 second(s)	10 interpretation(s)
projects starting in 2016 whose host is the university of zurich	OK	5.7676314 second(s)	10 interpretation(s)
full name of principal investigators of projects hosted in france	OK	10.7205081 second(s)	10 interpretation(s)
titles of erc projects with coordinators from piemonte	OK	1.8668707 second(s)	10 interpretation(s)
universities which are coordinators in climate change projects	OK	5.4404761 second(s)	10 interpretation(s)
countries with no projects	OK	0.7676488 second(s)	10 interpretation(s)
projects with a cost higher than 1 million	OK	5.5270729 second(s)	10 interpretation(s)
projects started after November 2019	OK	7.7705004 second(s)	10 interpretation(s)
projects including organizations from greece and romania	OK	3.6111342 second(s)	10 interpretation(s)

projects not including organizations from greece nor romania	OK	2.6492122 second(s)	10 interpretation(s)
find the project with the highest funding	OK	6.4535117 second(s)	10 interpretation(s)
find the country with the highest number of projects	OK	6.4062802 second(s)	10 interpretation(s)

2.4.5 Bio-SODA for SDSS

The following table shows the results of the tests performed on Bio-SODA for the SDSS dataset. We can see a timeout on one of the tests, since that particular query takes longer than the 20-minute limit set up for each test.

Query	Status	Time	Result
show all spec galaxies with ascension < 130 declination > 5	OK	24.8047582 second(s)	5 interpretation(s)
show all photo galaxies with magnitude_g <= 23 ascension < 130 declination > 5	OK	42.0476488 second(s)	5 interpretation(s)
show all photo asteroids with mode of photo observation 1	OK	1.5283111 second(s)	1 interpretation(s)
show white dwarfs with redshift > 0	OK	2.8264661 second(s)	2 interpretation(s)
show all hot massive blue stars	OK	0.1978928 second(s)	1 interpretation(s)
show all spec stars with plate number 1760	OK	1.7333888 second(s)	5 interpretation(s)
show all spec stars with the subclass WDhotter	OK	132.6419338 second(s)	5 interpretation(s)
redshift of spectroscopy with class QSO	ERROR		Read timed out

show all quasars with ascension > 120 and declination > 5.2	OK	27.3872073 second(s)	5 interpretation(s)
show all star burst galaxies with velocity dispersion > 800	OK	2.6380886 second(s)	2 interpretation(s)