PySPARQL Anything Showcase*

Marco Ratta$^{1[000-0003-3788-6442]}$, Enrico Daga$^{1[0000-0002-3184-5407]}$, and Luigi Asprino$^{2[0000-0003-1907-0677]}$

[1] The Open University, Walton Hall, Milton Keynes, UK
{marco.ratta,enrico.daga}@open.ac.uk
[2] University of Bologna, Italy
luigi.asprino@unibo.it

Abstract. In this demo paper we present *PySPARQL Anything*, the Python library of *SPARQL Anything*, an open source project for supporting semantic web technologists in building RDF graphs from heterogeneous sources. *PySPARQL Anything* enables developers to inject RDF graphs into their Python *RDFlib*, *NetworkX* or *pandas*-powered data science processes, opening new opportunities for developing complex, data-intensive pipelines for generating and manipulating RDF data. In addition, the library exposes a Python-based Command Line Interface (CLI) allowing easier installation and use.

Keywords: Knowledge Graph Construction · Façade-X · SPARQL Anything · Python

1 Introduction

Knowledge Graphs are nowadays first-class citizens in data science as they allow the seamless integration of diverse data [6]. Therefore, there has been increasing effort in supporting Python developers to work with RDF Knoweldge Graphs [4, 3]. In this demo, we aim to present and disseminate to the Semantic Web community *PySPARQL Anything*[3], the Python library of *SPARQL Anything*[4], an open source project that supports semantic web technologists in building RDF graphs from heterogeneous sources. *SPARQL Anything* is a data integration system that implements the *Façade-X* meta-model [2], resolving the heterogeneity of sources by structurally mapping them onto a set of RDF components upon which semantic mappings can be constructed. A technical report illustrating the overall architecture and functionalities of *SPARQL Anything* can be found in [1].

* The research leading to this publication has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement "Polifonia: a digital harmoniser of musical cultural heritage" (Grant Agreement N. 101004746), https://polifonia-project.eu. The publication reflects the author's views. The Research Executive Agency (REA) is not liable for any use that may be made of the information contained therein.

[3] https://github.com/SPARQL-Anything/PySPARQL-Anything

[4] https://github.com/SPARQL-Anything/sparql.anything

Using the JSON data hosted at *https://sparql-anything.cc/example1.json* for example, one can select the TV series starring "Courteney Cox" with the SPARQL query:

```
PREFIX xyz: <http://sparql.xyz/facade-x/data/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX fx: <http://sparql.xyz/facade-x/ns/>
SELECT ?seriesName
WHERE {
    SERVICE <x-sparql-anything:https://sparql-anything.cc
        /example1.json> {
        ?tvSeries xyz:name ?seriesName .
        ?tvSeries xyz:stars ?star .
        ?star fx:anySlot "Courteney Cox" .
    }
}
```

to directly obtain the results:

```
seriesName
"Cougar Town"
"Friends
```

The accumulated experience and feedback from the community of *SPARQL Anything* users, has lead to the decision of developing a Python integration. This is because of the emergent need to support the increasing community of Python users of Semantic Web technologies and the wide spread adoption of Python based tools for downstream tasks. *PySPARQL Anything* enables developers to inject RDF graphs into their Python *RDFlib*[5], *NetworkX*[6] or *pandas*[7]-powered data science processes, opening new opportunities for developing complex, data-intensive pipelines for generating and manipulating RDF data. Additionally the library exposes a Python-based Command Line Interface (CLI) allowing for easier installation and use. We demonstrate the usage of *PySPARQL Anything* via a "pythonic" re-interpretation of the *showcase-musicxml*[8] showcase, available at the *SPARQL Anything Github* repository for comparison.

2 PySPARQL Anything

PySPARQL Anything has been designed by borrowing some concepts of the Command behavioural pattern. The `pysparql_anything.SparqlAnything` class, with its `run, ask, select` and `construct` methods, defines the frontend interface of the system. The user specifies the parameters of their *SPARQL* request as

[5] https://github.com/RDFLib/rdflib
[6] https://github.com/networkx/networkx
[7] https://github.com/pandas-dev/pandas
[8] https://github.com/SPARQL-Anything/showcase-musicxml

Python keyword arguments. "Under the hood", these are automatically encapsulated by the language as a `dict` object that is passed, together with a receiver instance, to a specific execution method.

This receiver is a `pysparql_anything.SparqlAnythingReflection` object, which is a Python "reflection" of the `SPARQLAnything` class, the entry point of *SPARQL Anything*. This has been implemented using the *PyJNIus* [9] library.

The output of the user's request is either printed to the terminal, saved to a file (when using the `run` method), or returned as a Python object. Specifically, the tool supports returning the results of `SELECT` queries as `dict` or `pandas.DataFrame` objects and the results of `CONSTRUCT` queries as `rdflib.Graph` or `networkx.MultiDiGraph` objects. These can be achieved via the `select` and `construct` methods respectively. The results of `ASK` queries are returned as Python booleans when calling the `ask` method.

PySPARQL Anything also offers a CLI which processes the optional query arguments and passes them directly to the receiver object. This is accessed via the terminal using the `sparql-anything` command.

PySPARQL Anything is distributed on the *Python Package Index* (PyPI) [10] and is installed by typing the following in your machine's terminal.

```
$ pip install pysparql-anything
```

The code is also available at the corresponding *Github* repository[11].

3 Demo

In the demo, we will first illustrate the interface of *PySPARQL Anything* and then look at how *SPARQL* queries may be integrated into Python code.

As an example, the query showed in the introduction section above can be executed via either the command line interface as

```
!sparql-anything --query queries/select/testSelect.sparql
```

where one would obtain the same output as before, printed on the terminal,

```
seriesName
Friends
Cougar Town
```

or within a Python script or shell as

```
engine.select(
    query="queries/select/testSelect.sparql", output_type=pd.DataFrame
)
```

where one could (optionally) have the result returned as the `pandas.DataFrame` object:

[9] https://github.com/kivy/pyjnius
[10] https://pypi.org/project/pysparql-anything/
[11] https://github.com/SPARQL-Anything/PySPARQL-Anything

```
<class 'pandas.core.frame.DataFrame'>

    seriesName
0      Friends
1  Cougar Town
```

Furthermore, we will present an end-to-end scenario, based on a case study in computational musicology [5]. A music score in MusicXML is processed with *PySPARQL Anything* to generate a Knowledge Graph. Such graph is then analysed with Python libraries to derive interesting metrics such as statistics on note trigrams and derive a probability mass function of the data.

The demo can be accessed and executed via a live Google Colab notebook at the following address: https://bit.ly/pysa-demo

Step 1 In the first step, we setup the library and load the MusicXML files:

```
import pysparql_anything as sa
# Construct the SparqlAnything object
engine = sa.SparqlAnything()
# Assign the root directory of the files to a variable
root_dir = "showcase-musicxml/musicXMLFiles/AltDeu10/"
# Create a list of the names and paths to the xml files
xmls= [(name, os.path.join(root_dir, name)) for name in os.listdir(root_dir)]
```

Step 2 Next, we proceed with extracting melodic information, specifically, we show how one can use *PySPARQL Anything* to integrate SPARQL queries into a downstream task:

```
melody_dfs = [engine.select(
        query="showcase-musicxml/queries/getMelodyParam.sparql",
        values={"filePath": xml[1]},
        output_type=pd.DataFrame
    ) for xml in xmls]
```

Step 3 In the following code, we build trigrams from the data and count them:

```
# helper function to build and count the trigrams from a melody DataFrame
def count_trigrams(notes: list, trigrams_dict=dict()) -> dict[str, int]:
  for i in range(len(notes) - 2):
    trigram = notes[i] + "-" + notes[i + 1] + "-" + notes[i + 2]
    if trigram in trigrams_dict:
      trigrams_dict[trigram] += 1
    else:
      trigrams_dict[trigram] = 1
  return trigrams_dict
# Construct the trigrams and count their frequencies.
# Store the results in a dictionary
trigrams = dict()
for melody_df in melody_dfs:
  notes = list(melody_df["pitch"])
  count_trigrams(notes, trigrams)
```

Step 4 Finally, we produce the probability mass function of the data:

```
# Calculate the total number of trigrams in the dataset
total = sum(list(trigrams.values()))
# Construct the probability mass function of the trigrams in the dataset
pmf = {k: v / total for k, v in trigrams.items()}
# (Optional) Convert the pmf to a pd.DataFrame
pmf_df = pd.DataFrame(
    data=[[k, v] for k, v in pmf.items()],
    columns=["trigram", "P(trigram)"]
)
```

As a result we obtain

```
index,trigram,P(trigram)
0,A4—C5—D5,0.0011237357972281184
1,C5—D5—E5,0.0032463478586590086
2,D5—E5—E5,0.0011237357972281184
3,E5—E5—A4,6.242976651267325e—05
...
```

which can be compared to the result file *trigramAnalysis.csv* of the showcase.

4 Conclusion

In this demo paper we have proceeded to introduce *PySPARQL Anything* to the Semantic Web community. This is the Python library of the *SPARQL Anything* open source project that supports semantic web technologists in building RDF graphs from heterogeneous sources. We have also provided a description of the basic architecture underlying the backend of the system and have provided examples of how a *SPARQL* query can be executed with *PySPARQL Anything*. Further, a scenario from computational musicology illustrating how *SPARQL* queries can be integrated into a Python workflow has also been described.

The full live demo is available at https://bit.ly/pysa-demo.

References

1. Asprino, L., Daga, E., Dowdy, J., Mulholland, P., Gangemi, A., Ratta, M.: Streamlining knowledge graph construction with a fa\c {c} ade: The sparql anything project. arXiv preprint arXiv:2310.16700 (2023)
2. Asprino, L., Daga, E., Gangemi, A., Mulholland, P.: Knowledge Graph Construction with a façade: a unified method to access heterogeneous data sources on the Web. ACM Transactions on Internet Technology **23**(1), 1–31 (2023)
3. Dasoulas, I., Chaves-Fraga, D., Garijo, D., Dimou, A.: Declarative RDF construction from in-memory data structures with RML (2023)
4. Liang, L., Li, Y., Wen, M., Liu, Y.: Kg4py: A toolkit for generating python knowledge graph and code semantic search. Connection Science **34**(1), 1384–1400 (2022)
5. Ratta, M., Daga, E.: Knowledge graph construction from musicxml: an empirical investigation with sparql anything (2022)
6. Wilcke, X., Bloem, P., De Boer, V.: The knowledge graph as the default data model for learning on heterogeneous knowledge. Data Science **1**(1-2), 39–57 (2017)