A User-Friendly SPARQL Query Editor Powered by Lightweight Metadata

Vincent Emonet(✉), Ana-Claudia Sima, and Tarcisio Mendes de Farias

SIB Swiss Institute of Bioinformatics, Lausanne, Switzerland
{vincent.emonet,ana-claudia.sima,tarcisio.mendes}@sib.swiss

Abstract. SPARQL query editors often lack intuitive interfaces to aid SPARQL-savvy users to write queries. To address this issue, we propose an easy-to-deploy, triple store-agnostic and open-source query editor that offers three main features: (i) automatic query example rendering, (ii) precise autocomplete based on existing triple patterns including within `SERVICE` clauses, and (iii) a data-aware schema visualization. It can be easily set up with a custom HTML element. The tool has been successfully tested on various public endpoints, and is deployed online at https://sib-swiss.github.io/sparql-editor with open-source code available at https://github.com/sib-swiss/sparql-editor.

Keywords: SPARQL · Query editor · VoID metadata · Autocomplete

1 Introduction

The Semantic Web of data has been growing considerably in recent decades through the deployment of public query endpoints on the Web. For example, yummydata.org catalogues more than 55 SPARQL endpoints that are of interest to the biomedical community. However, SPARQL endpoints often lack an intuitive Web-based interface that effectively helps SPARQL-savvy users to write queries, notably missing autocomplete [3]. Several triple stores such as Stardog[1] and GraphDB[2] propose query editors but they are proprietary and triple store-dependent solutions, or they do not have an autocomplete solution based on a data-aware schema (i.e., a data schema built on the existing data). Alternatively, the *Qlever UI query editor* is open source[3] and provides autocomplete, but works only on the Qlever triple store. In [1], authors describe the Qlever autocomplete methodology that requires to send an SPARQL query for each autocomplete request. As a result, if this methodology is implemented over non-Qlever endpoints, each autocomplete request can take several seconds compromising usability and increasing the endpoint server load.

[1] https://www.stardog.com.
[2] https://graphdb.ontotext.com.
[3] https://github.com/ad-freiburg/qlever-ui.

E. Curry et al. (Eds.): ESWC 2025, LNCS 15832, pp. 3–7, 2026.
https://doi.org/10.1007/978-3-031-99554-5_1

Other open-source query editors exist; however, they are tailored to a specific dataset (e.g., *Wikidata Query Service* [5]) or lack basic relevant features, such as autocomplete (e.g., *LODEstar*[4] - additionally, the latter has not been maintained since 2018). Currently, a widely used open-source SPARQL editor is *YASGUI* [4] that is used by platforms such as GraphDB, Oxigraph and UniProt[5]. Although YASGUI offers user-friendly features like syntax highlighting, customizable autocomplete, endpoint selection, and modular results rendering, it lacks automatic adaptation to specific SPARQL endpoints. Its autocomplete suggestions are generic and unaware of the query context, requiring manual configuration for each endpoint. To the best of our knowledge, no existing query editor supports autocomplete within `SERVICE` clauses (i.e., federated queries).

To address these issues, we propose an SPARQL query editor based on YASGUI that is easy to deploy, open-source and triple store agnostic, dedicated to helping users write queries over Resource Description Framework (RDF) data. The editor automatically loads and renders query examples from any SPARQL endpoint, and provides a context-aware autocomplete feature.

2 Implementation

To support users to write queries for a given SPARQL endpoint, we extended the *Yasgui* environment [4] with three main features: (i) a query examples section, automatically constructed from endpoint metadata; (ii) precise autocomplete that adapts according to the SPARQL statements written by the user; and (iii) a data-aware schema visualization. To facilitate deployment, this extension is provided as an npm package[6] that can be easily set up by any SPARQL endpoint maintainer using a single custom HTML element `<sparql-editor/>`.

2.1 Displaying Query Examples

Our query editor automatically pulls SPARQL query examples stored as metadata in any given triple store and displays them to the user. To enable this, the SPARQL query examples should be described using the lightweight metadata schema proposed in [2]. In this methodology, SPARQL queries and their descriptions are defined mostly by reusing existing vocabularies such as the W3C recommendation Shapes Constraint Language (SHACL)[7]. By presenting the examples alongside the editor, users can easily inspect, run, or adapt the existing queries to their own information needs, or simply formulate completely new queries using the existing ones as source of inspiration. Figure 1 depicts the editor webpage. By clicking on the "Browse examples" button (bottom-right), users can inspect and search the full list of examples.

[4] https://github.com/EBISPOT/lodestar.
[5] https://sparql.uniprot.org.
[6] https://www.npmjs.com/package/@sib-swiss/sparql-editor.
[7] https://www.w3.org/ns/shacl.

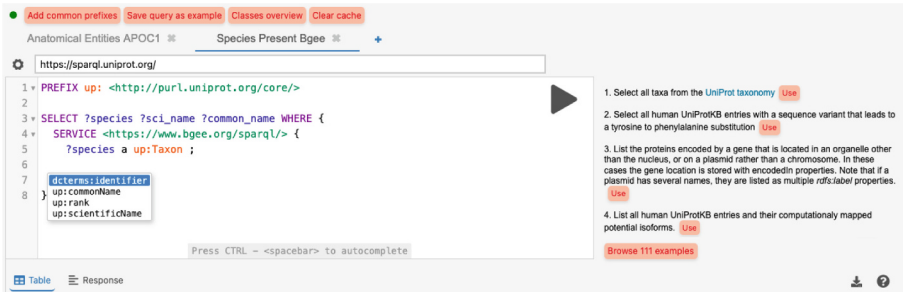

Fig. 1. Our query editor showing on the right side example queries stored as metadata and fetched from a SPARQL endpoint. By pressing CTRL+Space after `?species`, the autocomplete suggests only predicates which are asserted to instances of `up:Taxon`.

2.2 A Context Aware Autocomplete

Our SPARQL editor provides a precise autocomplete based on the Vocabulary of Interlinked Datasets (VoID)[8] descriptions of any RDF dataset. To generate VoID descriptions, SPARQL endpoint maintainers can rely on the VoID generator tool [2] that automatically produces VoID metadata that include statistics of the given dataset. The maintainers can make these metadata accessible through their respective SPARQL endpoints. The real-world applicability of this tool was demonstrated by running it in very large RDF datasets such as the UniProt SPARQL endpoint[9], which contains more than 210 billion triples.

Our editor retrieves the VoID metadata, if available, by querying directly the endpoint itself. The main information of interest processed by the editor from the VoID descriptions is represented by the classes instantiated in the endpoint and properties that relate them. This ensures that only properties that are stated and applicable according to the schema and the known type of a variable are suggested in the autocomplete drop-down menu (see Fig. 1). If no VoID metadata are found, the editor will try to retrieve all classes and properties. Finally, our autocomplete approach is lighter, faster, and complementary when compared to the state-of-the-art approaches such as the Qlever autocomplete. In addition, it works within SPARQL 1.1 federated `SERVICE` calls.

2.3 Building and Visualizing a Data-Aware Schema

By also considering the VoID descriptions, our query editor is able to build and visualize a simplified data schema that reflects exactly the RDF properties and classes used and accessible through a given SPARQL endpoint. Therefore, a data schema based on the actual contents of the endpoint, that is, a *data-aware schema*, is useful to aid users in writing SPARQL queries. To display the data-

[8] https://www.w3.org/TR/void.
[9] https://sparql.uniprot.org.

aware schema of a given endpoint, users can click on the "Classes overview" button in our graphical query editor interface (see Fig. 1, top orange buttons).

3 Demonstration

The proposed SPARQL query editor is publicly available and deployed at https://sib-swiss.github.io/sparql-editor. Users can choose either a predefined endpoint shown in the SPARQL endpoint field at the top, or input their own endpoint of interest in this field. Note that the new features described in Sect. 2 will only be available if the corresponding metadata are accessible through the assigned SPARQL endpoint. At the time of writing, 8 out of 10 public endpoints listed in our official query editor release included all the metadata needed for our tool to be fully operational. This also demonstrates that our approach is triple store agnostic, and readily deployable over diverse RDF data sources. Finally, non-savvy SPARQL users can use our ExpasyGPT tool at https://www.expasy.org/chat that generates SPARQL queries from plain-English text and run these queries with our SPARQL editor.

4 Limitations

- **Editor features limitations:** YASGUI relies on CodeMirror 5[10] which is outdated and does not have all the latest features of CodeMirror 6. We investigated the later one, but we realised that upgrading it would require to rewrite several core functionalities, such as autocomplete. Alternatively, we also consider replacing the editor with a VisualStudio based editor and a language server protocol (LSP) for SPARQL, for example, the one developed by the Qlever team[11].
- **Property autocomplete:** For property autocomplete to fully work, users need to explicitly define the subject type. This could be improved by reusing methods from the efficient and effective SPARQL autocompletion paper [1].
- **Target users:** This tool is intended for users that knows the SPARQL query language. Nevertheless, writing a query often requires exploratory steps to understand the data schema, that is how the data are structured. Our editor aims to reduce this friction and assist users in dealing with complex or unfamiliar endpoints.
- **Visual query builder:** Supporting less tech-savvy users through visual query building is currently out of scope; we recognize its value and believe our work on exposing and standardizing endpoint metadata could serve as a foundation for such tools. We welcome future contributions or collaborations that aim to expand the tool in this direction.

[10] https://codemirror.net/5/.
[11] https://github.com/IoannisNezis/Qlue-ls.

5 Conclusion

We have presented an easy-to-deploy, open source query editor over any SPARQL endpoint, which includes several novel features powered by lightweight metadata. The proposed features can greatly help SPARQL-savvy users to write queries over any SPARQL endpoint of interest. We also successfully applied our query editor to several public SPARQL endpoints that range from small to very large real-word RDF datasets. As a future work, we aim at improving the autocomplete feature to suggest next triple patterns by also considering predefined functions and rules. Currently, the proposed features will not work properly, if the needed metadata are not provided. To mitigate the later issue, we will work on an approach that probes endpoints and RDF data dumps, if available. Finally, we welcome contributions from the community to develop this further in our GitHub repository.

Acknowledgments. This work was funded by the Swiss National Science Foundation (SNSF) under the CHIST-ERA-22-ORD/SNSF TRIPLE project, grant: 20CH21_217482; the SNSF MetaboLinkAI project, grant: 10.002.786; and the swissuniversities CHORD in Open Science I, grant: "Swiss DBGI-KM".

References

1. Bast, H., Kalmbach, J., Klumpp, T., Kramer, F., Schnelle, N.: Efficient and effective SPARQL autocompletion on very large knowledge graphs. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, pp. 2893–2902 (2022)
2. Bolleman, J., et al.: A large collection of bioinformatics question-query pairs over federated knowledge graphs: methodology and applications. GigaScience **14** (2025). https://doi.org/10.1093/gigascience/giaf045
3. Jiang, S., Coblenz, M.: An analysis of the costs and benefits of autocomplete in IDEs. Proc. ACM Softw. Eng. **1**(FSE), 1284–1306 (2024)
4. Rietveld, L., Hoekstra, R.: The YASGUI family of SPARQL clients. Semantic Web **8**(3), 373–383 (2016)
5. Vrandečić, D., Pintscher, L., Krötzsch, M.: Wikidata: the making of. In: Companion Proceedings of the ACM Web Conference, WWW 2023 Companion, pp. 615–624. Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3543873.3585579