What's New in SPARKLIS

Sébastien Ferré

Univ Rennes, CNRS, IRISA
Campus de Beaulieu, 35042 Rennes cedex, France
Email: ferre@irisa.fr

Abstract. Sparklis is a SPARQL query builder that can connect to any endpoint, and that interacts with users in natural language only. Users are guided in the building of their queries so that they do not have to know the schema, and so that empty results are almost completely avoided. This demo paper presents a number of recent extensions to Sparklis. Most notably, it now supports analytical queries, Wikidata statement qualifiers, and the display of results on a map or as a slideshow.

1 Introduction

Sparklis[1] [2] addresses the key problem of semantic search, striving to reconcile the expressivity of SPARQL, and the usability of point-and-click user interfaces. It succeeds in doing so by combining principles and techniques from query builders, faceted search, and natural language interfaces. It supports exploratory search [4] by guiding users in the incremental construction of queries, avoiding empty results, and showing both queries and suggestions in natural language only. Complex queries can be built safely because intermediate results and relevant suggestions are shown at all steps.

Sparklis is based on theoretical work that started in 2010 [1], and was first released online in 2014. Since then, it receives in the range of 200-2000 hits per month, and was officially adopted as the main search tool by two French institutions who had developed SPARQL endpoints for their data: Persée[2] and INIST[3]. A startup, Askelys, has even been recently created to develop customized UI and other services on top of Sparklis.

The purpose of this demonstration paper is to shortly present the main features that have been added to Sparklis since the last publication in 2016 [2]. They improve it by significantly increasing the coverage of SPARQL, by hiding some complex structures in RDF data, and by improving user experience.

2 Coverage of SPARQL

In 2016, Sparklis covered basic graph patterns (including cycles), UNION, OPTIONAL, NOT EXISTS, basic filters, basic aggregations, ORDER BY, SELECT

[1] Web application and screencasts at http://www.irisa.fr/LIS/ferre/sparklis/
[2] http://data.persee.fr/explorer/sparklis/
[3] https://www.loterre.fr/sparklis/

and ASK queries. The newly covered constructs are those related to expressions (FILTER, BIND, GROUP BY, aggregators) and named graphs (FROM, GRAPH). Apart from CONSTRUCT and DESCRIBE queries and updates, which are not yet covered, only a few features are still missing: transitive closures in property paths (partly covered by hierarchies of terms, see next section), SERVICE for federated queries, explicit LIMIT and OFFSET, and separator specification in GROUP_CONCAT. The only serious limit in expressivity is that graph patterns must be connected at all steps, for the sake of tractability of query evaluation.

Expressions. SPARQL expressions, which are found in the FILTER and BIND constructs, and in the SELECT and GROUP BY clauses, bring a considerable increase in expressivity. They enable to derive values that are not explicitly present in data (e.g., people age from their birth date), and to answer analytical queries based on aggregations. Examples of analytical queries over geographical data are (permalinks to are provided as short URLs):

1. *How many countries are there in Europe?* (simple aggregation) http://bit.ly/2OMxbJw
2. *Give me the average population and the average area of countries, for each continent.* (grouping and multiple aggregations) http://bit.ly/2OMwKPm
3. *What is the average GDP per capita, for each continent?* (aggregation over a BIND) http://bit.ly/2OJsErj
4. *Which continents have an average agricultural GDP smaller than 10% of their average service GDP?* (FILTER over two aggregations) http://bit.ly/2OLAQaM
5. *Give me the distribution of the number of islands in archipelagos.* (aggregation over an aggregation) http://bit.ly/2NV71QF

For all those questions, an equivalent SPARKLIS query can be built over the MONDIAL dataset[4]. See the *Learn/Examples* page to try them live.

Named Graphs. Named graphs are found in the GRAPH construct, and in FROM clauses. They allow for controling the source of triples, and are required on some endpoints that put triples in different named graphs. SPARKLIS offers two ways to specify named graphs. The first way is to add FROM clauses in SPARQL queries by setting two configuration options: the *default graph URI* for retrieving factual triples, and the *schema graph URI* (if different from the default graph) for retrieving schema triples. The second way is to produce GRAPH-constructs in SPARQL queries for a finer control. Those constructs are introduced by the new query construct `according to ...`, which can be applied to any subquery to declare that this subquery should be matched on a named graph. The possible named graphs are shown in the results, and the ellipsis in the construct can be refined to select the desired named graphs. For example, the query: `give me a city that` *`according to a census whose year`* $>$ *2000* `has a population` $>$ `1000000` returns "the cities with over 1M inhabitants, *according to a named graph that is a census posterior to 2000*".

[4] http://dbis.informatik.uni-goettingen.de/Mondial

3 Complex Structures in RDF Graphs

Sometimes RDF datasets contain complex structures that, although covered by Sparklis 2016, generate ugly verbalizations, and are difficult to explore. We have added new constructs that abstract over those complex structures, so that they become more intuitive to query and explore.

Reified Relationships. Because RDF is limited to binary relationships (properties), a common solution to represent n-ary relationships or to add metadata to binary relations is to reify those relationships into *statements* or *events*. Unfortunately, various representations are used without consensus (e.g., standard reification, custom reification, singleton properties, named graphs, Wikidata predicates [3]), and it is difficult for Sparklis to automatically discover reified relationships through endpoints. Because of the importance and peculiarity of the Wikidata modeling, we have added a *Wikidata mode* as an option. In other cases, schema-level triples have to be added to declare which combinations of properties and classes represent reified relationships. For example, in the MONDIAL dataset, the relationships between countries and languages are reified into "spoken language" statements, and percentage values are attached to those statements to qualify the relationship. We have added the triple (`mondial:ofCountry`, `nary:subjectObject`, `mondial:onLanguage`) to declare that in the "spoken language" relationship, property `mondial:ofCountry` points to the subject, and property `mondial:onLanguage` points to the object. Every other property defined on the reified relationship (e.g., `mondial:percentage`) can be discovered by Sparklis, and suggested as a relation modifier that is verbalized similarly to prepositions. As a consequence, the verbose query `give me every country that is the country of a spoken language whose language is English and whose percentage is > 25` can now be expressed more directly as `give me every country that has as a language English at a percentage > 25` (http://bit.ly/2PHT0pD).

Hierarchies of Terms. Some properties code for the child-to-parent relationship of a hierarchy. Their range is equal to their domain, and in SPARQL queries it is usually their transitive closure that is used. General-purpose and well-known examples are `rdfs:subClassOf` or `skos:broader`. Domain-specific hierarchies exist for geographical locations, historical periods, anatomical parts, etc. We have added the construct `(hierarchy) in ...` that uses the last selected property as a hierarchical relationship. The effects are to apply transitive closure to the property in the generated SPARQL query, and to display the values of that property as a hierarchical tree. When a value in the tree is selected, it is equivalent to select all values under the selected value in the tree. It is possible to select several values and combine them by and/or. For example, the verbose and approximate query `give me every place that is the part of the part of California or Arizona` (assuming two levels between places and states) can now be expressed as `give me every place (hierarchy) in California or Arizona`.

4 User Experience

User experience is important and we have also made improvements on this side.

- **Responsive design.** The UI has been given a responsive design with Bootstrap. It becomes therefore easier to use Sparklis on smaller screens.
- **Map view.** When entities in the query have standard properties for latitude/longitude, they are detected and a special property `has geolocation` is suggested. If selected, this property triggers the display of the entities in the query results on a Google map (ex., http://bit.ly/2NT2Vs9).
- **Slideshow view.** When the results contain images, those are collected and displayed as a slideshow. To provide context for the images inside the slideshow, the result row that contains the image is displayed under the image. For example, in DBpedia, a slideshow of scientist depictions can be obtained, along with biographic information http://bit.ly/2NRSwwT.
- **Full-text search.** In order to provide a better experience when using the dynamic filtering features, support has been added for the full-text search capabilities of a few RDF stores (Jena/fuseki and Virtuoso so far). It assumes that full-text indices have been set up on the endpoint. For example, cities with name similar to "pétersbourg" are retrieved by http://bit.ly/2OH8sqg.
- **YASGUI editor.** For advanced users who want to see and edit the generated SPARQL query, the well-known YASGUI editor [5] has been integrated to Sparklis. It also brings useful tools for the download of query results in different formats, and for the visualization of results (pivot table and charts).
- **Dataset-specific UI customization.** Properties of `schema.org` can be used to customize the appearance in Sparklis. Property `schema:position` can be defined for classes, properties and entities to control their ordering in lists of suggestions. Property `schema:logo` can be defined on entities with small images to be used as icons, in addition to the label (e.g., the logo of a company, the conservation status of a species).
- **Multilinguality.** The UI is now available in four languages: English, French, Spanish, and Dutch. Similar languages would be easy to add.

References

1. Ferré, S.: Conceptual navigation in RDF graphs with SPARQL-like queries. In: Kwuida, L., Sertkaya, B. (eds.) Int. Conf. Formal Concept Analysis. pp. 193–208. LNCS 5986, Springer (2010)
2. Ferré, S.: Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. Semantic Web: Interoperability, Usability, Applicability 8(3), 405–418 (2017)
3. Hernández, D., Hogan, A., Krötzsch, M.: Reifying RDF: What works well with wikidata. In: Int. Work. Scalable Semantic Web Knowledge Base Systems. vol. 1457, pp. 32–47 (2015)
4. Marchionini, G.: Exploratory search: from finding to understanding. Communications of the ACM 49(4), 41–46 (2006)
5. Rietveld, L., Hoekstra, R.: YASGUI: Not just another SPARQL client. In: The Semantic Web: ESWC 2013 Satellite Events, pp. 78–86. Springer (2013)