# SPARQL Aggregate Queries made easy with Diagrammatic Query Language ViziQuer

Kārlis Čerāns[1], Jūlija Ovčiņņikova, Mārtiņš Zviedris

karlis.cerans@lumii.lv, julija.ovcinnikova@lumii.lv, martins.zviedris@lumii.lv
Institute of Mathematics and Computer Science, University of Latvia
Raina blvd. 29, Riga, LV-1459, Latvia

**Abstract.** We present a novel way to draw SPARQL aggregate queries via diagrammatic query language – ViziQuer. Since the introduction of SPARQL different graphical languages have been proposed to make SPARQL more user-friendly. In SPARQL 1.1 aggregate queries were introduced that are key to meaningful query formulation. However, diagrammatic query languages lacked this important end-user feature to make the diagrammatic SPARQL extensions powerful enough.

**Keywords:** Visual query creation, SPARQL, RDF, Aggregate queries

## 1 Introduction

SPARQL [1] is de facto query language for RDF [2] databases. Semantic RDF/SPARQL technologies offer a higher-level view on data compared to the classical relational databases (RDB) with SQL query language. Thus, semantic technologies enable more direct involvement of various domain experts in data set definition, exploration and analysis. Still, the textual form of SPARQL queries hinders its direct usage for IT professionals and non-professionals alike.

The diagrammatic query languages introduced to help formulating SPARQL queries, for instance, an earlier version of ViziQuer [3], or Optique VQS [4], do not support aggregate query formulation that is available in SPARQL 1.1. In a real-case scenario [5] it was identified that users could formulate basic SPARQL queries via graphical notation and that they were satisfied with the diagrammatic solution for very basic queries. Still they lacked expressive power to calculate different aggregated data.

The demonstration will show creation of aggregate SPARQL queries in the ViziQuer notation that is the main novelty of this paper. An extended outline of the design of the visual aggregate queries appears as [6]. This demo and paper present a novel and more refined SPARQL query generation algorithm that relies on explicit distinctness list notion for aggregate queries thus allowing correctly capturing a wider range of intuitive queries within the diagrammatic notation.

[1] Supported, in part, by Latvian State Research program NexIT project No.1 "Technologies of ontologies, semantic web and security".

## 2 Basic Query Notation

The visual/diagrammatic query definition is based on the data schema definition as OWL ontology or RDF Schema. We use the following example mini-University ontology that is presented in Figure 1 in graphical OWLGrEd ontology editor notation [7].

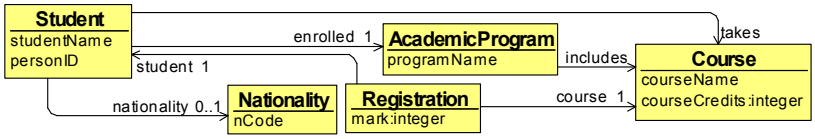


**Fig. 1.** A mini-University ontology fragment in the OWLGrEd notation (cf. owlgred.lumii.lv)

A query in ViziQuer is a graph of class instance nodes connected with links corresponding to triples connecting these instances. Each node shows the instance class name (e.g. *Registration*, *Student* in Fig.2), possibly an explicit instance reference (e.g. *R* and *S*), as well as conditions (e.g. *mark>=4*) and selection instances and attributes (e.g. *R* and *mark* for *Registration* class). One of the classes in the query is marked as the main query class (shown as orange round rectangle) while all other classes (shown as violet rectangles) are called condition classes [8]. The semantics of a basic query is to find all instance graphs matching the pattern defined by the query and list the selection instances and/or attributes for each instance graph. The order by, limit and offset clauses for the query can be marked within the main query class, as well.

There can be affirmative (black solid line), optional (blue/light dashed line) or negation (red line with stereotype {not}) links within the query. The default interpretation of optional or negation link is to mark the entire subgraph placed behind the link (from the viewpoint of main query class) as optional or negated respectively. A negation link with {condition} stereotype is interpreted as the non-existence of the respective link between its end instances (the query graph is required to have a spanning tree consisting of all its non-condition links).

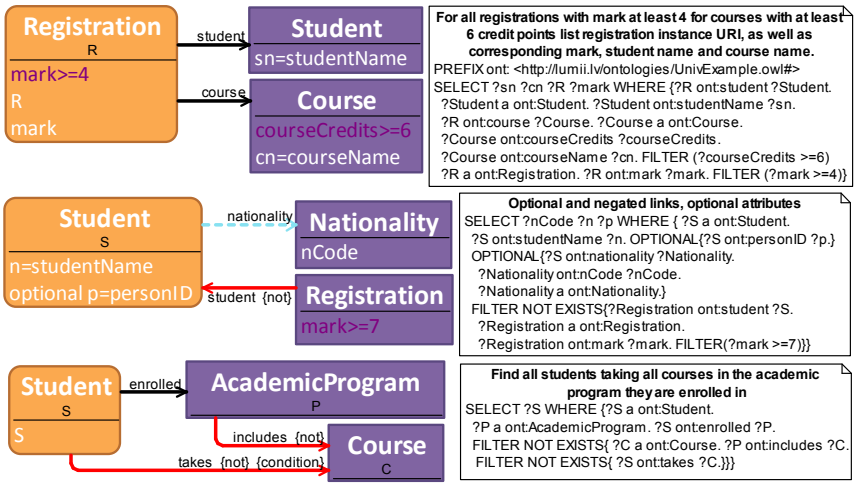


**Fig. 2.** Basic query examples

## 3 Introducing Aggregate Queries

The aggregation options can be included into the queries just by introducing into class instance attribute lists aggregate expressions where an SPARQL aggregate function (e.g. *count*, *sum*, *avg*) is applied to a non-aggregated (i.e. plain) attribute expression, for instance, as in *sum(courseCredits)* in Fig. 3.

The semantics idea is to compute aggregate values taking as the grouping set all non-aggregated attributes specified in the query. A direct implementation of this idea would, however, lead to counterintuitive results since the aggregated instance attribute value would be included into the aggregation as many times as the instance appears in some instance graph matching the query. We offer a more refined semantics that we explain for the case, if all aggregate attributes are placed within single class of the query, we call it *aggregation class*. In the case of aggregate attributes in different classes separate subqueries are to be made for each aggregation class with their results merged (cf. [6]).

The SPARQL query generation follows three steps: (i) the raw query with aggregation function arguments (plain attributes) instead of aggregate attributes is generated; (ii) the distinctness list for aggregation computation over the raw query is formed, consisting of all attributes (both non-aggregated and aggregated ones alike) and instances of so-called *multiplicative* classes. The multiplicative class set by default includes the main query class and the grouping class; the set can be extended by ascribing the <<all>> stereotype to a class in the query, a class can be excluded from the set by the <<exists>> stereotype; (iii) the aggregation over the distinctness-list selection from the raw query is formed by aggregating the aggregation attributes and grouping on all non-aggregated attributes.

Figure 3 depicts two variants of the natural language query "find all nationalities and the sum of credit points of courses taken by students of this nationality". The first query counts every course once per nationality, while the second one - once per nationality and student, since the *Student* class is in the multiplicative class set for the query and therefore an extra *?S* appears in the query distinctness list (leading possibly to counting credit points of a single course several times per nationality).

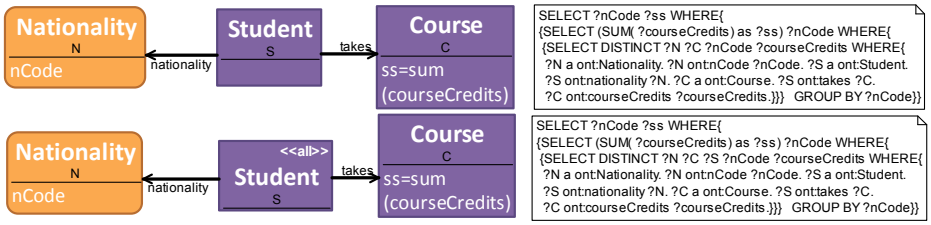


**Fig. 3.** Simple aggregation demonstration

The ViziQuer tool supports also explicit subquery introduction via {group} stereotype on affirmative and optional links [6], useful both for more involved query formulation (e.g. "find all courses passed by at least 10 students with mean mark (over all passed courses) at least 7") and for merging the results of aggregate queries with different multiplicative class sets.

## 4 Discussion and Conclusions

The demonstrated ViziQuer tool is freely available online at viziquer.lumii.lv and the users are welcome to download it, import their ontologies/RDF schemas and start creating visually their own SPARQL queries. The introduced notation raises a hope of introducing a wider range of specialists to direct use of RDF/SPARQL-organized data as the ViziQuer tool will assist in creating complex statistical queries (the need for initial user training is foreseen). The potential usage scenarios for the ViziQuer tool involve both exploring SPARQL endpoints [3] and re-engineering relational databases [9].

There is an important practical query pattern "find all x with their related most common y" (e.g. find all courses with the most often received marks in them) that can be expressed in a slightly extended diagrammatical query notation, still queries of this kind cannot be naturally expressed in SPARQL. A practical workaround to this problem would be either to re-formulate such queries to return larger result sets from which the needed results can be obtained e.g. in a spreadsheet, or to translate visual queries directly into SQL, if there is a relational database behind the SPARQL endpoint.

The use cases have also shown the possibility of attribute expression creation and translation. Still, the standard SPARQL functions [1] appear insufficient for practical queries e.g. concerning duration calculation. Notably, the Virtuoso RDF data store [10] supports the extensions allowing the necessary date and interval value handling.

## References


1. SPARQL 1.1 Overview. W3C Recommendation 21 March 2013 [WWW] http://www.w3.org/TR/sparql11-overview/
2. Resource Description Framework (RDF), http://www.w3.org/RDF/
3. Zviedris, M.; Barzdins, G.: ViziQuer: A Tool to Explore and Query SPARQL Endpoints. In: The Semantic Web: Research and Applications, LNCS, 2011, Volume 6644/2011, pp. 441-445
4. Soylu, A.; Giese, M.; Jiménez-Ruiz, E.; Kharlamov, E.; Zheleznyakov, D.; Horrocks, I.: OptiqueVQS: Towards an Ontology based Visual Query System for Big Data. In: MEDES. 2013.
5. Barzdins, G.; Liepins, E.; Veilande M.; Zviedris M.: Semantic Latvia Approach in the Medical Domain. In Proc. of 8th International Baltic Conference on Databases and Information Systems. H.M.Haav, A.Kalja (eds.), TUT Press, pp. 89-102. (2008).
6. Cerans, K.; Ovcinnikova, J.; Zviedris, M.: Towards Graphical Query Notation for Semantic Databases. In Proc. of BIR'2015, LNBIP, Springer 2015, vol. 229.
7. Barzdins, J.; Cerans, K.; Liepins, R.; Sprogis, A.: UML Style Graphical Notation and Editor for OWL 2. In Proc. of BIR'2010, LNBIP, Springer 2010, vol. 64, p. 102-113.
8. Zviedris, M.; Liepins, R.: Readability of a diagrammatic query language. In VL/HCC 2014 IEEE Symposium on, S. 227–228. IEEE, 2014.
9. Cerans, K.; Barzdins, G.; Bumans, G.; Ovcinnikova, J.; Rikacovs, S.; Romane, A.; Zviedris, M.: A Relational Database Semantic Re-Engineering Technology and Tools. In Baltic Journal of Modern Computing (BJMC), Vol. 3 (2014), No. 3, pp. 183-198.
10. Blakeley, C.: RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping), OpenLink Software, 2007.