ViziQuer: Notation and Tool for Data Analysis SPARQL Queries

Kārlis Čerāns[1], Jūlija Ovčiņņikova[1]

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

Abstract. We present a UML class diagram style notation for data analysis SPARQL query definition and its implementation in the ViziQuer tool that provides query definition environment and query translation into SPARQL 1.1. The notation and its implementation within the tool allows for rich value selection and condition expression language, as well as integrated data aggregation facilities, both essential for data analysis query definition.

Keywords: Visual query creation, SPARQL, RDF, Aggregate queries, Data analysis 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. The database-to-ontology mapping techniques (cf. [3,4]) and ontology-based data access technologies [5,6] create the potential for SPARQL usage also over the massive amount of data stored and maintained in relational databases.

Still, the entirely textual form of SPARQL queries hinders its direct usage for IT professionals and non-professionals alike. A number of diagrammatic query notations to help formulating SPARQL queries have been proposed, including ViziQuer [7,8], OptiqueVQS [9] and QueryVOWL [10]. Their expressivity, however, is limited mostly to basic forms of queries, notably excluding support for aggregate queries included in SPARQL 1.1 [1] (except for authors' earlier demonstration [8]) and means for integrating rich expression language for conditions and selection attributes.

In a real-case scenario [11] it has been identified that users could formulate basic SPARQL queries via graphical notation and be satisfied with it on this query level. Still they lacked expressive query power to calculate different aggregated data that are

[1] Supported, in part, by Latvian State Research program NexIT project No.1 "Technologies of ontologies, semantic web and security" and ERDF project "Rich Visual Queries for Ontology-Based Data Access" (Ref. No. 1.1.1.1/16/A/277)"

important for any data inquiries of statistical nature. The ongoing work of re-engineering the example of [11] makes explicit the need of rich expression language in the queries, as well. The support for aggregation and rich expressions, as presented originally in this paper, makes a visual query language suitable for data analysis query formulation currently served mostly by various business intelligence tools.

We describe here the ViziQuer notation and tool for data analysis query definition and translation into SPARQL 1.1, involving data aggregation facilities and rich expression language integration. The ViziQuer notation, like the one of OptiqueVQS tool [9], is based on UML class diagrams used widely and successfully in engineering; the UML class diagrams have inspired also OWL ontology editor OWLGrEd [12].

The presentation of the ViziQuer language is organized in the form of query patterns covering different query definition aspects and corresponding to the situations naturally arising in the data analysis query creation. The query tool usage can be started just after mastering the simplest query definition patterns, so the language and tool usage is kept low-entry. The advanced query patterns, including the ones for expression language, should not be regarded as prohibitive for motivated end users (similarly, as also non IT-experts can master using expression notation, e.g., in Microsoft Excel).

In the following, Section 2 introduces basic query notation, following by the aggregate query patterns in Section 3 and expression patterns in Section 4. Section 5 concludes the paper. The resources for the example of this paper are available at http://viziquer.lumii.lv/demo/miniUniv.

2 Basic Query Notation

A query in the ViziQuer notation is a graph of class boxes connected with association links. In a typical query both the class boxes and association links will have the class/association names specified and at least some class boxes would contain specified selection attributes. The interpretation of such a query graph is to define a class instance pattern with an instance corresponding to each specified class, its data properties corresponding to the specified attributes, and the object property links between the instances corresponding to the associations linking the classes.

The query diagram shows the local names of classes, associations and attributes, their mapping to the full names is available in the data schema model that has to be loaded in the tool before query creation[2].

We shall demonstrate the query constructs on a generic mini-University data set example involving students, courses and academic programs (cf. e.g. [8] for its brief description). The query in Fig. 1 specifies selection of all names of students together with the names of their taken study courses from this data set. In the ViziQuer notation one of the classes has to be marked as the main query class (specified as orange round rectangle), the others are condition classes (light violet rectangles); the choice of the main query class shall become important in further query patterns.

[2] There are options of loading the data schema from an OWL ontology and from a SPARQL endpoint (actual data schema).

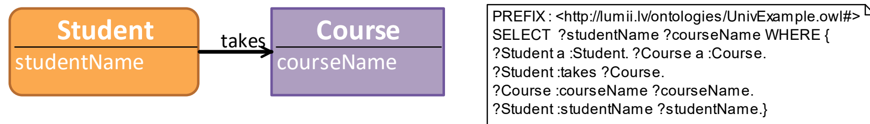

Fig.1. A simple query and its translation into SPARQL

Fig. 2 illustrates the explicit instance reference names introduced and added to the query output (the instance URI is returned). The attribute conditions (marked as purple texts) and the alias option for selection items are illustrated, as well. The instance references and aliases are used for variable name generation, they can also be referred to from other query parts. If a class instance reference name is not explicitly specified, the class name can be used instead of it, however, the implicit instances of the same class appearing in different parts of the query are considered to be different.

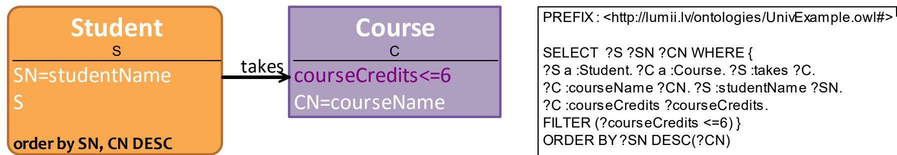

Fig.2. A query with instance references, conditions, aliases and ordering

2.1 Optional and Negation Link Patterns

There can be affirmative (black solid line), optional (blue/light dashed line) or negation (red line with stereotype {not}) links between classes within the query. The presence of optional or negation links in the query require it to have a tree-shaped structure (this shall be relaxed in Section 2.2). The interpretation of optional or negation link is to mark the entire subgraph placed behind the link (from the viewpoint of the main query class) as optional or negated respectively.

Fig. 3 illustrates the optional and negation links among the classes, as well as the optional stereotype for the attributes (we consider the *Nationality* and *Registration* (a student registers for a course) classes, along with the *Student* class for the illustration).

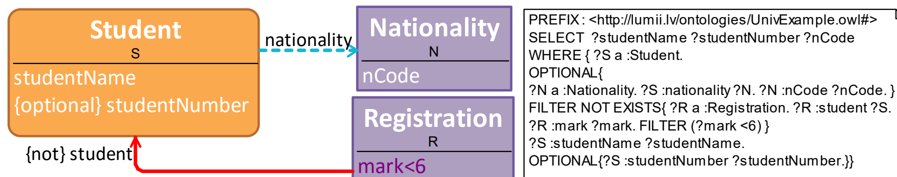

Fig. 3. Optional and negation links, optional attributes

2.2 Condition Link Pattern

The condition links (marked by the {condition}-stereotype) are meant to extend the tree-shaped query structure introduced in Section 2.1. The interpretation of a condition

link is to add a triple connecting the link end nodes to the query pattern in the case of affirmative link and add a respective triple non-existence filter in the case of negation link (notice the difference from marking the entire query part behind the link as negated in the case of non-condition negation links).

Fig. 4 illustrates two queries with condition pattern: one with affirmative links and the other involving negations. Notice that the second query with "double negation" expresses universal quantification: finding names for all students taking all courses included in the academic program they are enrolled in.

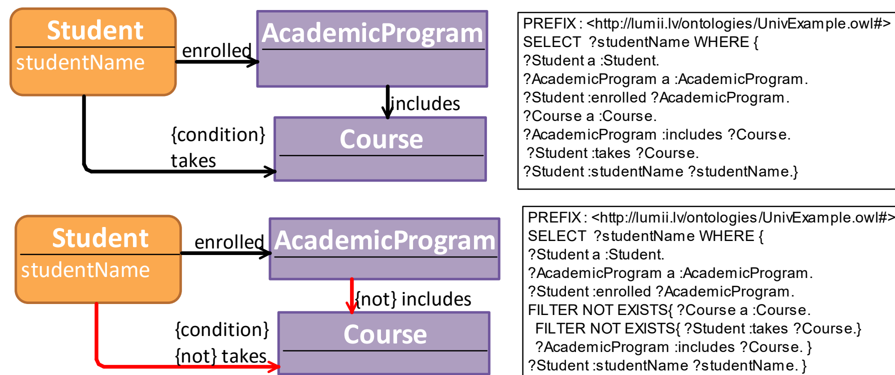

Fig. 4. Condition links: beyond the tree-shaped structure

2.3 Meta-information Query Patterns

The meta-information queries can be obtained by placing explicit variables in the class name and/or link name positions within the query, as illustrated in Fig. 5.

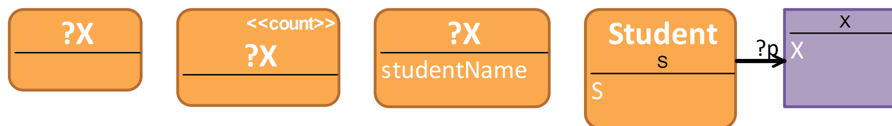

Fig. 5. (i) find all classes, (ii) find all classes together with their instance count, (iii) list class names and *studentName* property values of all instances with this property, (iv) select all student class instances with all object and data properties and their values.

3 Aggregate Query Patterns

The simplest aggregate query pattern is a count of class instances. It can be specified either using a class stereotype <<count>>, or by creating an attribute expression with the count function applied to the class instance reference.

Simple extensions of the basic count pattern allow counting main class instances satisfying conditions specified in either the main class itself, or in a condition class (cf. Fig. 7). In the case of a condition class present in a counting (or other pure aggregation)

query, its semantics by default is just asserting the existence of a respective linked instance with the specified properties. The alternative semantics of computing the aggregation over the query patterns involving possibly multiple condition class instances for a single main class instance (so, a single main class instance could be observed several times in the query) can be achieved by the <<all>> stereotype placed on the condition class.

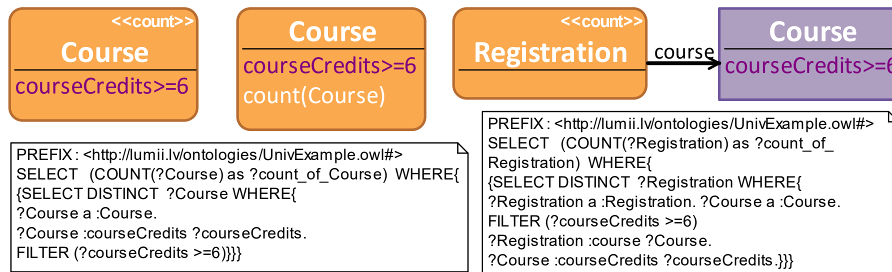

Fig. 6. Two count notation options in single-class query, and count in a query with condition.

3.1 Simple Statistics Patterns

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(mark)* in Fig. 7.

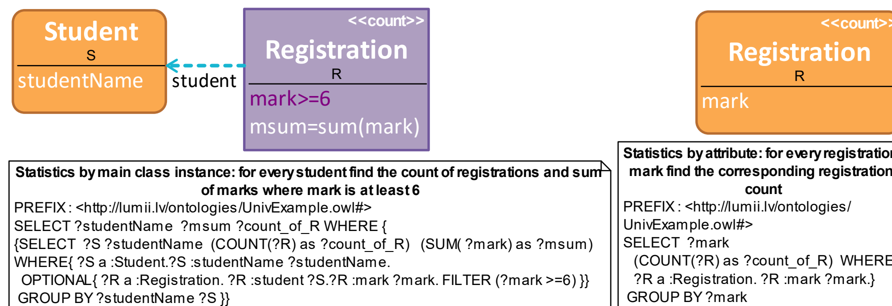

Fig. 7. The statistics by class instance and statistics by attributes patterns

The statistical queries are obtained by including both the scalar (i.e. non-aggregated) and aggregated expressions that are obtained by within the query result set. The aggregate expressions are evaluated by default against the grouping set that includes the main class instance and all non-aggregated attributes included in the query; the main class instance can be excluded from the grouping set by setting a main class stereotype (e.g. <<count>>) in the query. Two important subclasses of simple statistics patterns are statistics by attributes where the <<count>> stereotype is attached to the main query class and statistics by main class instance where a separate statistics row(s) is (are) computed for each main class instance. Fig. 7 illustrates both these patterns.

3.2 Filters over Aggregate Results

The attribute conditions specified in the class nodes are to be evaluated before the aggregation computation and they limit the scope of the aggregation. The filters that compute conditions on aggregate results can be placed in a *having*-compartment within the main query class.

3.3 Existential and Universal Stereotypes

Figure 8 shows two semantically different ways of counting the students receiving any specific mark. They involve either counting each student for each mark at most once, or counting the student as many times, as there are registrations by this student with the specified mark. In general, for any class in the query the <<exists>> and <<all>> stereotypes can be specified to mark, whether only existence of a class instance is to be observed during the aggregation computation, or all instances are to be counted separately; the default stereotype for a condition class is <<exists>>; notice that <<all>> stereotypes for multiple classes within a query can create multiplicative blow-up of aggregation scope, and, therefore, also of aggregate numbers in the query results.

The main class, if not stereotyped, causes creation of separate aggregation scopes for each instance; this corresponds to the effect of the stereotype <<find>>. In general, the <<exists>>, <<all>> and <<find>> stereotypes can be placed on any class, including the query and the condition classes, to change their grouping behavior within the aggregate queries.

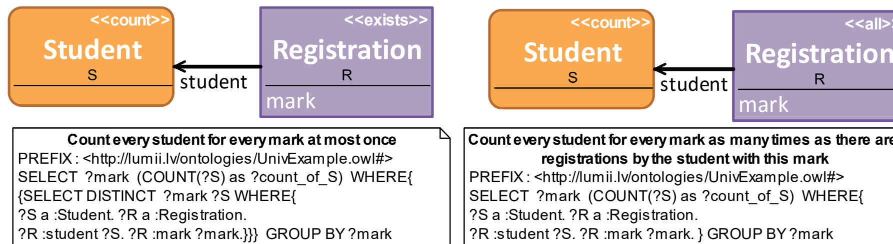

Fig. 8. Existential and universal stereotypes

3.4 Explicit Subquery Pattern

The default aggregate computation rule of using a single grouping set for all computing all aggregate functions within the query is not sufficient, for instance, in the cases of nested aggregation. Therefore, an explicit {subquery}-stereotype is introduced for attributes and links that turns the query fragment within the subquery scope a subquery related to the subquery parent class instances (the subquery parent class is the class containing the {subquery}-attribute, or the class at the end of the {subquery}-link on its main class side). A typical subquery attribute would be a group-concatenation of multiple same named data properties of a class instance. A subquery link example is in Fig. 9, where for every student class instance the minimal registration mark is found

and then all students having the minimal mark at least 7 are counted. Note that since the minimal mark is computed in a subquery, it can be used within the condition (and not the *having*) compartment of the main query class.

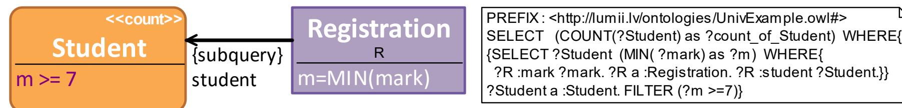

Fig. 9. Explicit subquery pattern example

4 Expression Notation and Patterns

The basic expression pattern in ViziQuer is that of a class attribute specification either for the query selection list, or within a selection or filtering expression. In either case the attribute specification corresponds to:

(i) creating a variable name that is derived from the attribute name (typically, by prefixing the local variable name by '?'; additional decorations can be added to make the variable names unique within the query) and

(ii) linking the class instance variable by the property corresponding to the attribute to the created attribute variable.

If an expression, say `a+b`, is specified in the selection list for the class whose instance variable in the SPARQL translation is `?p`, its translation shall involve `?p :a ?a. ?p :b ?b. BIND(?a+?b AS ?expr_1)` in the SPARQL query pattern part and `?expr_1` in the query selection list; if there were an expression alias specified, as in `c=a+b`, it would be used as the variable name both in the BIND-clause: `BIND(?a+?b AS ?c)` and in the select list.

The general rule for selection and filtering expression forming in ViziQuer is to allow expressions following the SPARQL expression syntax, with the modification expecting a (possibly qualified) attribute name in the place of a variable name within the original SPARQL notation.

The attribute names in the ViziQuer expressions may be qualified by class instance reference names present in the query, or by property path expressions; in either case the qualifications shall use the UML style separator '.' (cf. Fig. 10).

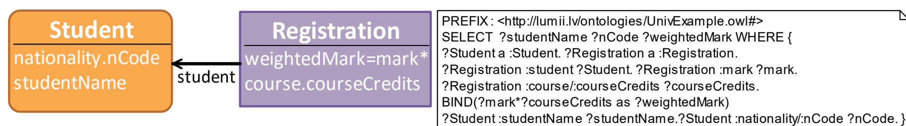

Fig. 10. For all students show the nationality codes (path expression) and weighted marks in all registrations (path expression within arithmetic expression)

4.1 Negated Condition Pattern

Asserting the class instance, say p, attribute value to satisfy a (non-negated) predicate, e.g. mark>=7, implies the existence of the attribute value AND that it satisfies the predicate: ?p :mark ?mark. FILTER (?mark>=7). So, a negation of the assertion means

that either the attribute value dos not exist OR that it exists but not satisfy the predicate, cf. Fig. 11; it should be distinguished from requiring the negation of the condition present in the query. This pattern explains specifics of handling null values in conditions, often important in practical query situations.

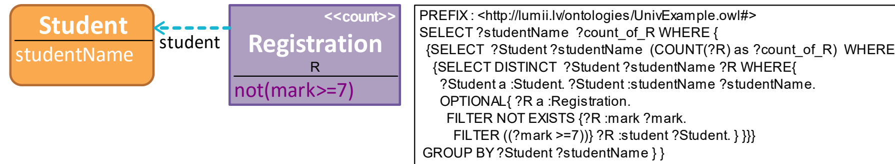

Fig. 11. For all students count the number of registrations not having marks 7 or above

4.2 Specific Expression Patterns

A common specific expression pattern often required in statistic data analysis, however, not supported in standard SPARQL 1.1, is a date value difference. We provide a date value difference functions in ViziQuer (cf. Fig. 12) for accessing vendor-specific SPARQL endpoints; our current implementation is targeted towards OpenLink Virtuoso [13], other target environments can be added based upon the construct availability in the SPARQL endpoint.

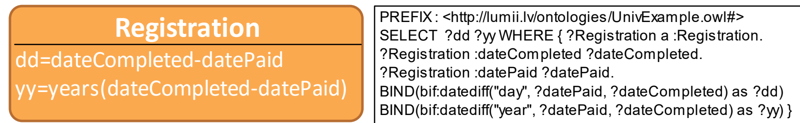

Fig. 12. Date value difference, expressed in days (the default) and in years

4.3 Multiple Statistics Pattern

The expression language incorporated in ViziQuer allows for joining multiple statistical inquiries into single query. Figure 14 shows the way, how to compute for each student simultaneously the count of all taken courses, as well as the counts of taken big (courseCredits>=0) and small (courseCredits<0) courses. We notice that the expressions involved in the query resemble ones that can be used for simple statistical data processing in Microsoft Excel. Should the further practical query tool usage experiments confirm the initially observed importance of this query pattern, a special notation might be designed to ease the query formulation options in accordance to it.

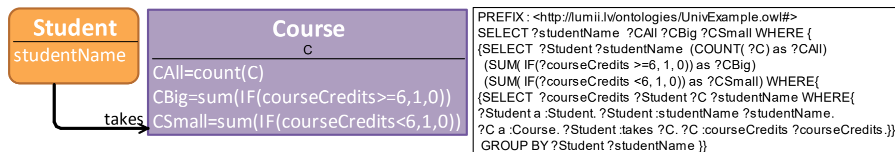

Fig. 13. Counting taken courses from different course sets

5 Discussion and Conclusions

The explained patterns for UML style visual data analysis query definition are implemented in the ViziQuer tool that is freely available at http://viziquer.lumii.lv.

The introduced notation and patterns can be seen also as an attempt to push forward the UML style diagrammatic SPARQL query definition in general with the aim of covering data analysis queries that are currently in practical situations handled by business intelligence suites with data residing in relational databases. The presented notation can be criticized, updated, extended and offered alternative implementations.

The initial practical experience with defining queries for Latvian Medicine Registries example [11] show that the notation can be near to sufficient for the end user statistical needs; the practical application of the notation as well as its further fine tuning shall be continued. At the same time the initial usage of the notation with simplest basic and aggregate query patterns can be kept low-entry.

The future work plans include re-implementing the tool within the web environment, as well as adding result visualization component to it.

References

1. SPARQL 1.1 Overview. W3C Recommendation 21 March 2013, http://www.w3.org/TR/sparql11-overview/
2. Resource Description Framework (RDF), http://www.w3.org/RDF/
3. R2RML: RDB to RDF Mapping Language, http://www.w3.org/TR/r2rml/
4. D2RQ. Accessing Relational Databases as Virtual RDF Graphs, http://d2rq.org/
5. Optique. Scalable End-User Access to Big Data, http://optique-project.eu
6. Calvanese, D., Cogrel, B., Komla-Ebri, S., Lanti, D., Rezk, M., Xiao, G.: How to Stay Ontop of Your Data. In: Databases, Ontologies and More. ESWC (Satellite Events) 20-25, (2015)
7. Zviedris, M., Barzdins, G.: ViziQuer: A Tool to Explore and Query SPARQL Endpoints. In: The Semantic Web: Research and Applications, LNCS, Volume 6644/2011, pp. 441-445, (2011)
8. Cerans, K., Ovcinnikova, J., Zviedris, M.: SPARQL Aggregate Queries Made Easy with Diagrammatic Query Language ViziQuer. In: Proceedings of the ISWC 2015 PD, CEUR Vol. 1486, (2015), http://ceur-ws.org/Vol-1486/paper_68.pdf
9. Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I.: OptiqueVQS: Towards an Ontology based Visual Query System for Big Data. In: MEDES. (2013)
10. Haag, F., Lohmann, S., Siek, S., Ertl, T.: QueryVOWL: Visual Composition of SPARQL Queries. In: The Semantic Web: ESWC 2015 Satellite Events. LNCS, Vol.9341, pp. 62-66. Springer, (2015), http://vowl.visualdataweb.org/queryvowl/
11. 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. Haav, H.M., Kalja, A. (eds.), TUT Press, pp. 89-102. (2008)
12. Barzdins, J., Cerans, K., Liepins, R., Sprogis, A.: UML Style Graphical Notation and Editor for OWL 2. In: Proc. of BIR'2010, LNBIP, Springer, vol. 64, pp. 102-113, (2010)
13. Blakeley, C.: RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping), OpenLink Software, (2007)