Visual query creation in ViziQuer: a brief usage instruction

ViziQuer provides visual/diagrammatic environment for ontology-based data query definition and execution.

To work with ViziQuer one needs:

- An ontology (a data schema[1]), and
- A connection to SPARQL endpoint (if the generated SPARQL queries are to be directly executed from the ViziQuer environment).

The data schema specifies:

- the local name to full URI mapping for classes and properties,
- subclass and subproperty relations,
- class-to-property connections, and
- cardinalities (important to determine, if maximum cardinality for a property in the context of a class is 1, or not).

As of ViziQuer 0.2.2, there is option to have pre-configured data schemas and even full initial projects (including data schema, project parameters (endpoint configuration and others), and visual query examples).

There is public ViziQuer environment available from ViziQuer main page with user self-registration option available.

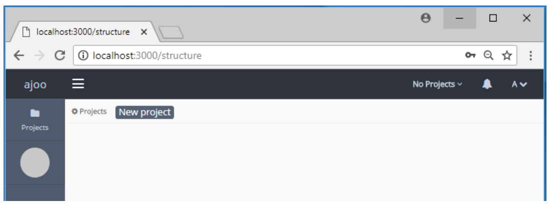
The ViziQuer software is open source (MIT licence), the instructions for local tool setup are available at the ViziQuer GitHub page: https://github.com/LUMII-Syslab/viziquer

Currently the administration (configuration) rights are assigned to the single user that has been registered (signed up) first on a given server.

We recommend Google Chrome for work within ViziQuer query environment.

Work with projects

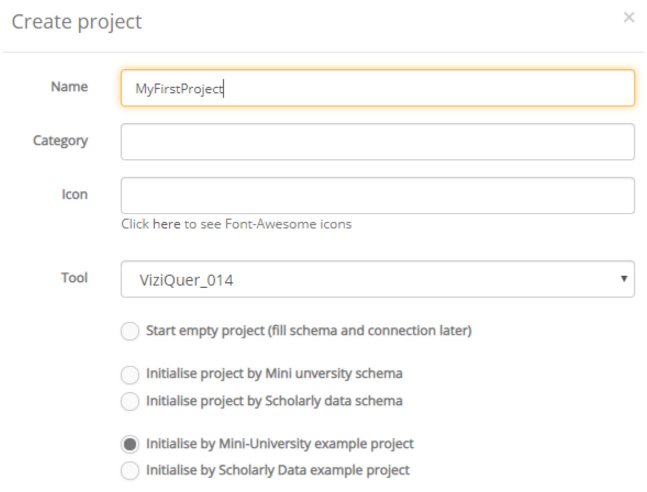
After sign-up to the query tool, an empty user environment (tab Projects) is displayed.



Choose 'New project', then enter the name you wish for the project and choose the tool from the drop-down menu (should there be no tools available in a local installation, ask the system administrator to complete the tool environment setup, as described in https://github.com/LUMII-Syslab/viziquer).

[1] ViziQuer uses a custom data schema format. Schema format description and examples are available from the ViziQuer main page.

Should the pre-configured data schemas and/or projects be installed at the endpoint, the user can choose using some of the available options during the new project dialogue[2].


Create project

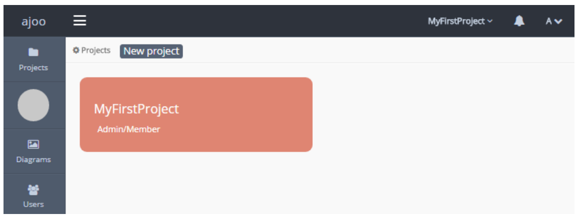
Name	MyFirstProject
Category	
Icon	
	Click here to see Font-Awesome icons
Tool	ViziQuer_014

○ Start empty project (fill schema and connection later)

○ Initialise project by Mini unversity schema
○ Initialise project by Scholarly data schema

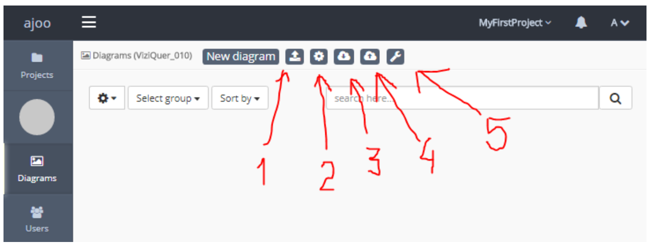
● Initialise by Mini-University example project
○ Initialise by Scholarly Data example project

Upon successful creation of a project, its icon becomes available:



Note. On the upper right hand of the project icon there is a small menu allowing to Duplicate, Edit (e.g. change the name), Delete and Leave the project (in case of shared use of the project).

Click the project icon, or Diagrams menu item on the left to get into the project environment.



To make the project environment working, upload the data schema (a JSON file) (#1) and tune the project settings (#2), in particular, a SPARQL endpoint and the named graph, as well as query engine type (OpenLink Virtuoso by default, General SPARQL option available).

You can also import a project from a project JSON serialization (#4; the diagrams from the imported project are added to the diagrams of the current project). The projects saved from this or from some other server can be imported (there is requirement that tool configurations on both projects coincide (are based on the same tool JSON configuration files).

[2] The administrator can upload the file jsons/services.json, or similar into the active tool (Configurator tab) to enable the initial schema and/or project upload suggestions (The same upload button, as for the configuration update is to be used).

Note. The data ontology is currently not included in a project JSON serialization, it has to be set up separately after project upload.

#3 allows to save the project into its JSON serialization (currently may work better on Firefox, than Chrome).

#5 is migration of the project onto new version of the query tool configuration.

Work with diagrams

A project may have several query diagrams, each holding a single or multiple queries.

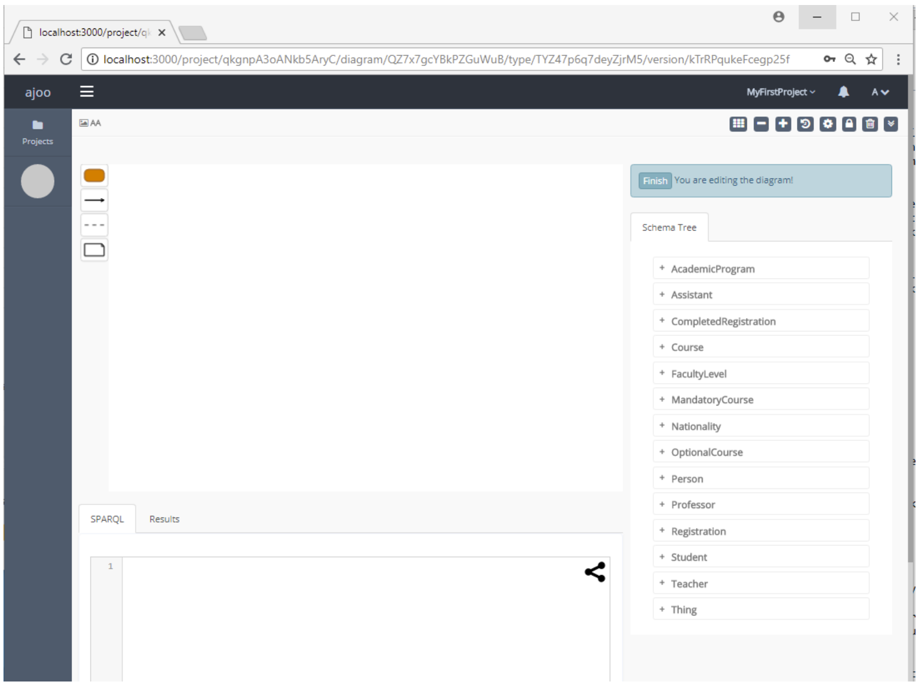
Each query in a diagram is a connected component that has a single main class node (orange round rectangle) and possibly a number of condition classes.

Each query shall have a spanning tree of all its nodes and edges that are not marked as condition edges (for instance, if a query does not have condition edges, it has to have a tree shape with the main query class being its root). The condition edges are added on top of the query tree structure.

A project diagram environment consists of diagram menu, symbol palette, visual query area, SPARQL / Results tabs and a property pane, initially showing the classes of the uploaded ontology (data schema).

The ontology classes are shown in the property pane whenever there is no element or element group selected in the visual query pane (a click on the empty space in the diagram shall bring up the class list into the property pane).

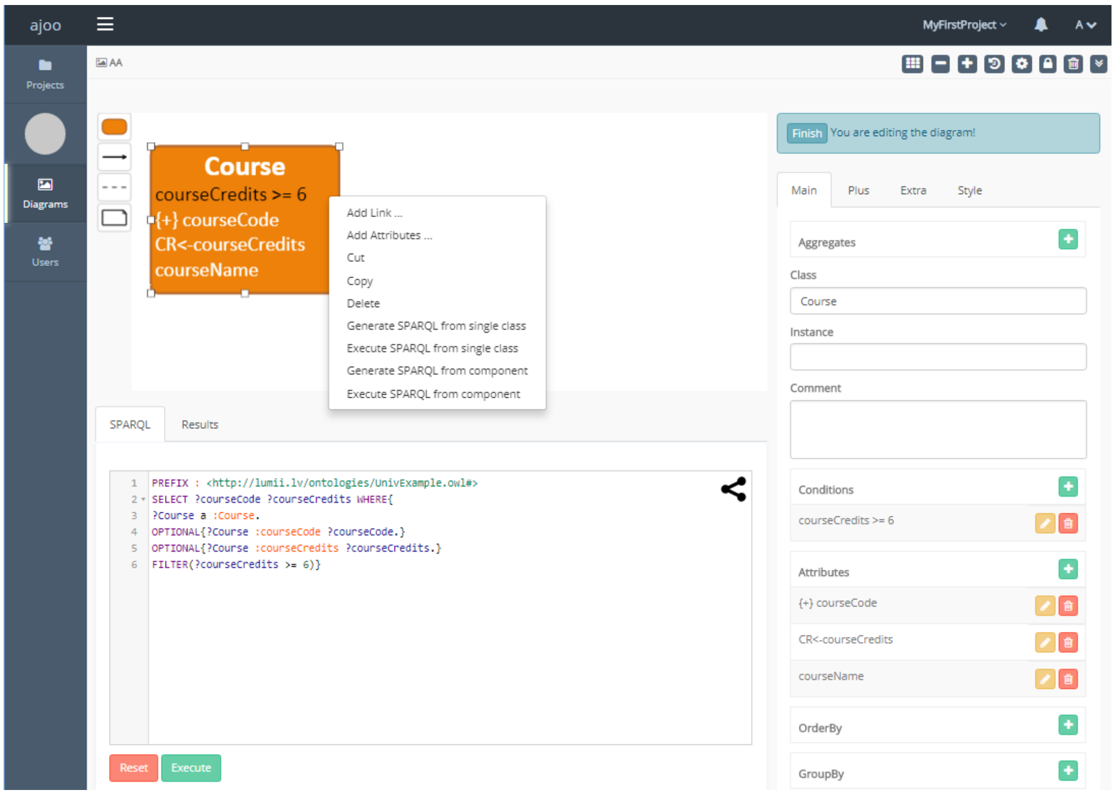
A double-click on the class name in the property pane inserts a new main query node, based on this class, into the visual query area.



Another way to start a query drawing is to click on the orange class seed in the symbol palette and then draw a class symbol within the visual diagram area, followed by choosing the class name in the class compartment in the node property pane.

Should a single node or single edge be selected/focused in the visual query area, the property pane shall show the properties of the selected/focused node or edge.

The following figure illustrates the query environment in action.



```
PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
SELECT ?courseCode ?courseCredits WHERE{
?Course a :Course.
OPTIONAL{?Course :courseCode ?courseCode.}
OPTIONAL{?Course :courseCredits ?courseCredits.}
FILTER(?courseCredits >= 6)}
```

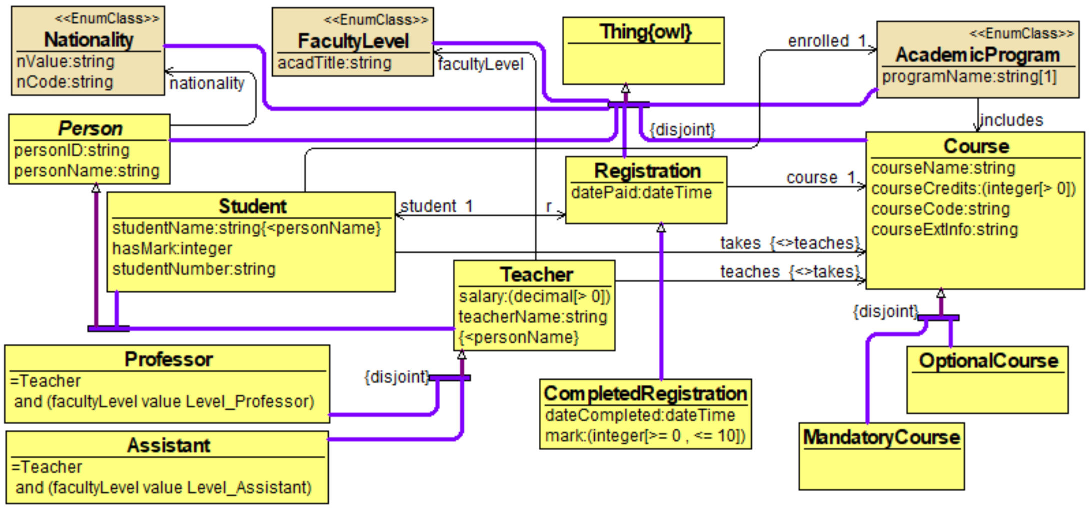
Class: Course

Conditions: courseCredits >= 6

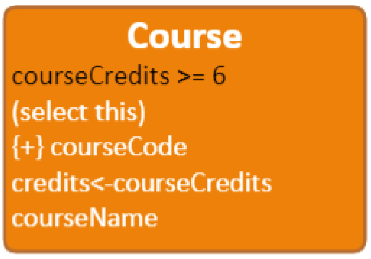
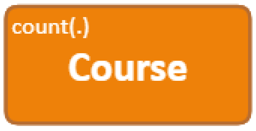
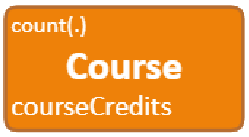
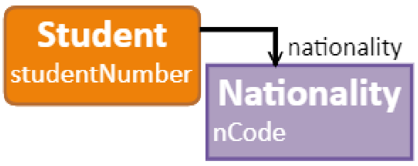
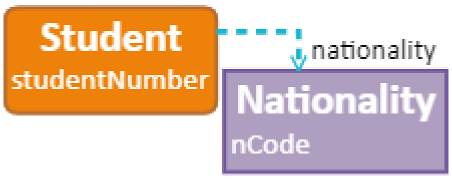
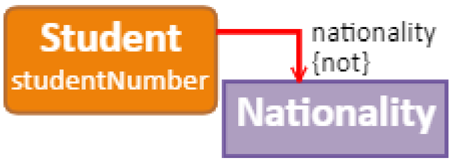
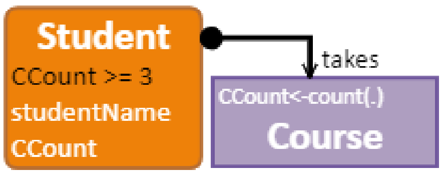
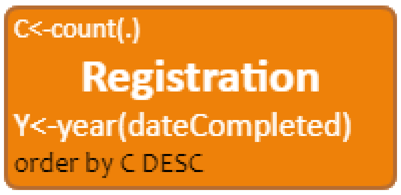
Attributes: {+} courseCode; CR<-courseCredits; courseName

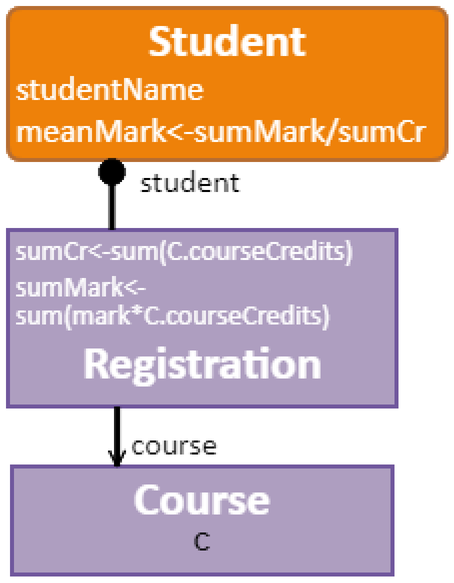
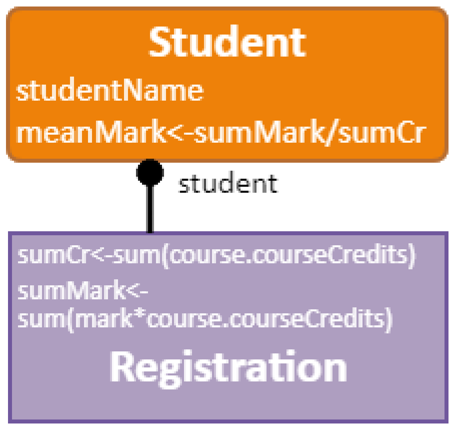
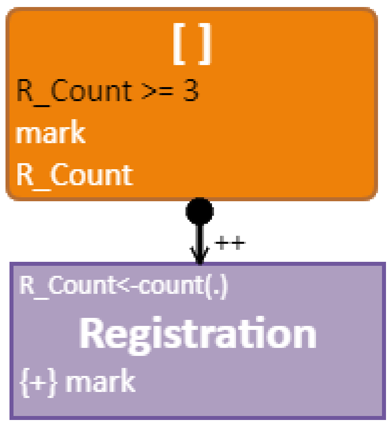
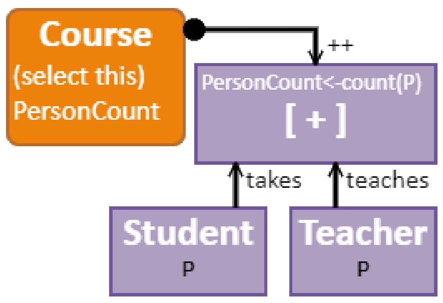
The ViziQuer tool is intended to be able to work with any user-supplied data schema.

The following figure visualizes a simple mini-University ontology, used as a data schema in the following examples.



The following table show some visual query examples over the mini-University ontology.

 Course courseCredits >= 6 (select this) {+} courseCode credits<-courseCredits courseName	Select instances of a class, show the values of the instance URI (specified by (select this)) and the designated instance attributes (courseCode, courseCredits, courseName). The attributes are optional by default; the mark {+} indicates that the corresponding attribute (courseCode) is required. The attribute courseCredits is renamed into credits in the query output. Only those instances whose attributes satisfy the given condition are selected.
	Count all courses.
	Count all courses, grouped by their attribute courseCredits values.
 Student studentNumber nationality Nationality nCode	Select student numbers together with students' nationality code (the nationality has to be present for the student).
 Student studentNumber nationality Nationality nCode	Select student numbers together with optional students' nationality code.
 Student studentNumber nationality {not} Nationality	Select student numbers of students that do not have a nationality specified.
 Student CCount >= 3 studentName CCount takes CCount<-count(.) Course	Select name and count of taken courses of students taking at least 3 courses. Note the subquery link (black bullet) considering the courses for each student separately. The subquery selection attributes have to be included into the main query select list, if they are to be included in the query output.
 C<-count(.) Registration Y<-year(dateCompleted) order by C DESC	Count registrations (a student can be registered for a course) by the year of the course completion date. In general, SPARQL functions over attribute values are allowed in ViziQuer expressions (either in attribute or condition fields). Note. Although the code completion support is currently limited just for entity names, more expressions are supported, if entered correctly.

 Student studentName meanMark<-sumMark/sumCr student sumCr<-sum(C.courseCredits) sumMark<- sum(mark*C.courseCredits) Registration course Course C	For every student registered at least for one course find the student's mean mark over all courses (= all registrations for a course).
 Student studentName meanMark<-sumMark/sumCr student sumCr<-sum(course.courseCredits) sumMark<- sum(mark*course.courseCredits) Registration	Using property paths notation the student's mean mark can be computed within a more compact diagram.
 [ ] R_Count >= 3 mark R_Count ++ R_Count<-count(.) Registration {+} mark	Find all marks for student participation in a course that have been assigned at least 3 times, together with the number of times the mark has been assigned. The subquery counts the registrations per assigned mark; the outer query consisting of a single non-data node corresponds to the outer query level that does the filtering. The mark ++ marks a query edge that is does not correspond to a data link among instances.
 Course (select this) PersonCount ++ PersonCount<-count(P) [+] takes teaches Student P Teacher P	For every course list its name together with the number of persons related to it either as students or teachers