LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

VIZUĀLAM RĪKAM DATU SHĒMAS ĢENERĀCIJA NO SHACL

MAĢISTRA DARBS

Autors: **Ēriks Žeibe**

Studenta apliecības Nr.: ez18039

Darba vadītājs: profesors Dr. dat. Kārlis Čerāns

RĪGA 2024

ANOTĀCIJA

Semantiskā tīmekļa tehnoloģijām turpinot attīstīties, parādās jauni publiski pieejami zināšanu grafi un ar tiem saistītās tehnoloģijas. Viena no plaši lietotām zināšanu grafu vaicājumu valodām ir SPARQL. Lai gan pēc sintakses tā līdzinās SQL valodai, tai ir savas īpašās iezīmes, kas var būt grūti uztveramas iesācējiem un neprofesionāļiem. Šim mērķim ir radīti dažādi SPARQL vizualizācijas rīki, kas ļauj veidot vaicājumus ar diagrammām un citām vizuālām kontrolēm.

Maģistra darba fokuss ir uz LUMII izstrādātu rīku ViziQuer. Šī rīka lietošanu atvieglo priekšā-teikšanas funkcionalitāte, kas ir iestrādāta dažādos laukos, zinot konkrētā zināšanu grafa metadatus. Šie metadati jeb datu shēma tiek iegūta ar atsevišķi izstrādātu rīku OBIS-SchemaExtractor palīdzību. Maģistra darba ietvaros tika izstrādāts atsevišķs rīks, kurš no SHACL specifikācijā rakstīta faila spēj aizpildīt ViziQuer datubāzi ar datu shēmas informāciju, nodrošinot ViziQuer priekšā-teikšanas funkcionalitāti. Tā ir alternatīva OBIS-SchemaExtractor rīka pieejai, izmantojot standartizētu specifikāciju SHACL.

Atslēgvārdi: Semantiskais tīmeklis, zināšanu grafi, RDF, ViziQuer, SPARQL, SHACL

ABSTRACT

DATA SCHEMA GENERATION FROM SHACL FOR A VISUALIZATION TOOL

As Semantic Web technologies continue to evolve, new publicly available knowledge graphs and related technologies are emerging. One widely used query language for knowledge graphs is SPARQL. Although syntactically similar to SQL, it has its unique features that can be difficult for beginners and non-professionals to grasp. For this purpose, various SPARQL visualization tools have been created, allowing users to construct queries with diagrams and other visual controls.

The focus of the master's thesis is on the tool ViziQuer, developed by LUMII. The usability of this tool is enhanced by an autocomplete functionality integrated into various fields, utilizing the metadata of a specific knowledge graph. This metadata, also known as data schema, is obtained with the help of a separately developed tool, OBIS-SchemaExtractor. Within the scope of the master's thesis, a separate tool was developed that can populate the ViziQuer database with data schema information from a file written in the SHACL specification, thus ensuring ViziQuer’s autocomplete functionality. This approach provides an alternative to the OBIS-SchemaExtractor tool by using the standardized SHACL specification.

Keywords: Semantic web, knowledge graphs, RDF, ViziQuer, SPARQL, SHACL

AUTOREFERĀTS

Maģistra darba ietvaros tika apkopoti un praktiski izmēģināti 4 dažādi SPARQL vizuālo vaicājumu rīki un tika sīkāk apskatīta ViziQuer rīka piedāvātā funkcionalitāte. Darbā tika papildinātas ViziQuer iespējas, izstrādājot atsevišķu rīku, kurš spēj nolasīt SHACL specifikācijā rakstītu zināšanu grafa metadatus un šo informāciju ierakstīt ViziQuer datubāzē, nodrošinot priekšā-teikšanas funkcionalitāti. Izstrādes gaitā tika apskatītas 9 SHACL parsēšanas bibliotēkas, no kurām tika praktiski izmēģinātas 3 un pēc analīzes veikšanas 1 no tām tika izmantota izstrādātajā rīkā. Darba izstrādē tika izmantoti 24 informācijas avoti.

Darba pirmajā nodaļā tiek apskatīts semantiskā tīmekļa jēdziens un ar to saistītās tehnoloģijas. Īpaši svarīga no tām ir SHACL specifikācija, kuru ierobežotā apjomā spēj atpazīt maģistra darbā izstrādātais rīks. Otrajā nodaļā tiek plašāk apskatīta vizuālā pieeja SPARQL vaicājumu veidošanā, izskatot dažādus SPARQL vizualizācijas rīkus, tai skaitā ViziQuer, uz kuru ir fokuss šajā darbā. Trešajā nodaļā ir aprakstīta maģistra darbā izstrādātā rīka prasības, izstrādes gaita, kā arī tiek demonstrēti piemēri, kas gūti no šī rīka lietošanas.

Rīka izstrādes gaitā tika veiktas regulāras konsultācijas ar darba vadītāju par ViziQuer rīka darbības principiem un tā saderību ar SHACL specifikācijā definētiem zināšanu grafa datu shēmām. Tika arī testēti dažāda tipa ViziQuer iespējamie vizuālo vaicājumu veidi.

Maģistra darbs ir izstrādāts uz kursa darba pamata. Kopējā darba izstrādes laika apjoms ir 8 mēneši, no kura 5 mēneši ir darba praktiskās daļas izstrāde, ieskaitot SHACL rīka izstrādi, ViziQuer rīka uzstādīšanu lokālajā ierīcē, kā arī maģistra darba praktiskās daļas teksta izstrādi.

Lai nodrošinātu darba atbilstošu noformējumu un gramatisko kvalitāti, darbs ir vairākkārtīgi pārlasīts, kā arī pārbaudīts ar Word programmatūras pareizrakstības pārbaudēm. Darba izstrādē tika ievērotas vadlīnijas noslēgumu darbu izstrādāšanai Latvijas Universitātē, kā arī nozarei specifiskie termini tika pārbaudīti un tulkoti izmantojot Latvijas Nacionālā terminoloģijas portāla vietni.

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS 7

IEVADS 8

1. SEMANTISKĀ TĪMEKĻA TEHNOLOĢIJAS 9

1.1. RDF 9

1.2. RDFS 10

1.3. OWL 11

1.4. SPARQL 12

1.5. SHACL 13

2. VIZUĀLĀ PIEEJA SPARQL VAICĀJUMU VEIDOŠANĀ 15

2.1. ViziQuer 15

2.2. OptiqueVQS 16

2.3. RDFExplorer 19

2.4. Sparnatural 21

3. SHACL-2-VIZIQUER RĪKA IZSTRĀDE 23

3.1. Rīka darbības konteksts 23

3.2. ViziQuer datu shēmas ekstrakcija ar OBIS-SchemaExtractor 25

3.3. Plānotā funkcionalitāte 26

3.4. ViziQuer uzstādīšana lokālajā ierīcē 27

3.4.1. ViziQuer uzstādīšana 27

3.4.2. Data Shape Server uzstādīšana 28

3.4.3. Datubāzes uzstādīšana 28

3.5. SHACL failu pieejamība 28

3.6. SHACL interpretatora izvēle 29

3.7. ViziQuer tabulu struktūra 32

3.8. ViziQuer tabulu struktūras aizpildes process 33
3.9. Realizētā risinājuma tehniskais apraksts 34
3.10. No SHACL ģenerēta RDF avota pielietošanas demonstrējums 36
3.10.1. Valstīs runājošo valodu skaitu iegūšana 36
3.10.2. Uz salām esošo pilsētu atlase 40
REZULTĀTI 47
SECINĀJUMI 48
PIELIKUMI 51

APZĪMĒJUMU SARAKSTS

API *(Application Programming Interface)* – programmatūras saskarne, kas ļauj divām atsevišķām lietotnēm savā starpā komunicēt.

HTML *(Hypertext Markup Language)* – valoda, ar kuru tiek aprakstīta tīmekļa lapu struktūra.

IRI *(International Resource Identifier)* – interneta protokola standarts, kas paplašina URI standartu, palielinot atļauto simbolu kopu.

JSON *(JavaScript Object Notation)* – bieži lietots datu apmaiņas formāts, kura pamatā tiek lietota JavaScript valodas objektu notācija.

OWL *(Web Ontology Language)* – semantiskā tīmekļa valoda, kas spēj attēlot bagātas un sarežģītas zināšanas par lietām.

RDF *(Resource Description Framework)* – ietvars, kurš attēlo datu savienojamību tīklā.

RDFS *(Resource Description Framework Schema)* – klašu, to lauku kopums, kuru definē ar RDF datu modeļa palīdzību, lai nodrošinātu pamata elementus ontoloģiju aprakstīšanai.

REST *(Representational State Transfer)* – programmatūras arhitektūras stils, kas nosaka vadlīnijas Globālā tīmekļa arhitektūru projektēšanai un izstrādei.

SHACL *(Shapes Constraint Language)* – W3C standarts, kas ļauj aprakstīt RDF grafus.

SPARQL *(SPARQL Protocol and RDF Query Language)* – standarta vaicājumu valoda un protokols priekš saistītajiem atvērtajiem datiem un RDF datiem.

SQL *(Structured Query Language)* – vaicājumu valoda, lai glabātu un apstrādātu informāciju relāciju datubāzēs.

URI *(Uniform Resource Identifier)* – unikāla simbolu virkne, kas identificē kādu loģisku vai fizisku resursu, kuru lieto tīmekļa tehnoloģijas.

URL *(Uniform Resource Locator)* – atsauce uz resursu, kas atrodas datortīklā, kā arī mehānisms tā iegūsanai.

URN *(Uniform Resource Name)* – URI, kas lieto urn shēmu.

XML *(Extensible Markup Language)* – iezīmēšanas valoda, kas nodrošina likumus lai aprakstītu jebkādus datus.

IEVADS

Semantiskais tīmeklis reprezentē nākamo lielo evolūciju informācijas apvienošanas un attēlošanas ziņā. Tas ļauj datiem būt savienotiem no dažādiem avotiem tādā veidā, ka to spēj izprast datorierīces, lai veiktu arvien izsmalcinātākus uzdevumus.

Lai tehniski būtu iespējama informācijas ieguve no uzkrātajiem semantiskā tīmekļa avotiem jeb zināšanu grafiem, W3C ir izstrādājis īpašu vaicājumu valodu SPARQL, kuras sintakses principi līdzinās plaši pazīstamajai SQL vaicājumu valodai. Tā kā SPARQL vēl nav guvis tik lielu popularitāti ārpus semantiskā tīmekļa lietojuma, tad var bieži gadīties jauni lietotāji vai neprofesionāļi, kuriem sākotnēji ir apgrūtinoša šīs valodas lietošana. Šī iemesla dēļ ir radīti dažādi vizualizācijas rīki, kas ļauj būvēt vaicājumus, izmantojot diagrammu izveidi, no kurām rīks spēj ģenerēt SPARQL vaicājumus. Šis darbs ir fokusēts uz LUMII izstrādātu rīku ViziQuer un tā funkcionalitātes paplašināšanu, izmantojot SHACL valodas iespējas. Daļa no iespējām ir – datu shēmas ģenerēšana, kas ir alternatīvs veids, kā definēt metadatus priekšā-teikšanas funkcionalitātei.

Darba mērķis ir ViziQuer rīka datu shēmas ģenerācija, izmantojot SHACL valodu, paplašinot iespējas, kā rīkā tiek iegūti metadati priekšā-rādīšanas funkcionalitātei. Lai to paveiktu tiek izvirzīti sekojoši uzdevumi:

- Literatūras apskats par semantiskā tīmekļa tehnoloģijām.
- ViziQuer rīka alternatīvu apskats saistībā ar vizuālu SPARQL vaicājumu veidošanu.
- ViziQuer rīka funkcionalitātes sīkāks apskats.
- Izpēte, ka SHACL valodu var integrēt ViziQuer rīka datu shēmas ģenerēšanā.
- Risinājuma izstrāde, kurš spēj ielasīt SHACL failu ViziQuer datubāzē priekšā-teikšanas funkcionalitātes nodrošināšanai.

Darbs ir strukturēts 3 nodaļās. Pirmajā nodaļā tiek veikts semantisko tīmekļa tehnoloģiju apskats, iekļaujot RDF, OWL, SPARQL, SHACL. Otrajā nodaļā tiek veikta 4 vizuālu SPARQL vaicājumu veidošanas rīku apskats. Tai skaitā tiek arī apskatīts ViziQuer rīks un tā piedāvātā funkcionalitāte. Trešajā nodaļā tiek apskatīta SHACL ielasīšanas rīka shacl-2-viziquer izstrādes prasības, gaita un rezultāti.

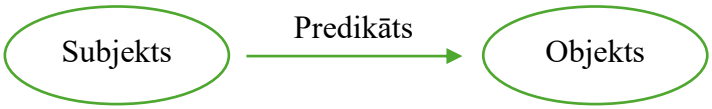
1. SEMANTISKĀ TĪMEKĻA TEHNOLOĢIJAS

Semantiskais tīmeklis ir Tima Bernera Li vīzija, kas paredz Globālā tīmekļa paplašināšanos, nodrošinot programmatūrai mašīnlasāmus metadatus par publicēto informāciju un datiem. Tā rezultātā datoriem būtu iespēja iegūt jēgpilnu izpratni, līdzīgi kā cilvēkiem apstrādājot informāciju, lai sasniegtu mērķus [1]. Vīzijas īstenošanai W3C ir izstrādājis vairākus standartus, kas apraksta tehnoloģijas, kuru savstarpējā sadarbība spēj nodrošināt semantisko tīmekli. Šajā nodaļā tiks apskatītas galvenās no šīm tehnoloģijām.

1.1. RDF

RDF ir formāla valoda, lai aprakstītu strukturētu informāciju. Tās mērķis ir dot lietotnēm iespēju savā starpā apmainīties ar datiem caur tīmekli, nezaudējot to oriģinālo nozīmi. Atšķirībā no HTML un XML, galvenais nolūks nav attēlot dokumentus, bet gan ļaut tālāku informācijas apstrādi un pārveidošanu. RDF tiek bieži uztverts kā pamata attēlošanas formāts izstrādājot semantisko tīmekli [2].

RDF dokuments apraksta virzītu grafu, kurš sastāv no mezgliem, kas ir savienoti ar virzītām šķautnēm (skat. 1.1. att.). Gan mezgliem, gan šķautnēm tiek piešķirti identifikatori, lai tos varētu savā starpā atšķirt. Atšķirībā no XML, kas ir pamatā paredzēts, lai aprakstītu informāciju kokveida struktūrā, RDF dokumenti apraksta informāciju grafu struktūrā. Tas tiek darīts tādēļ, ka RDF mērķis nav aprakstīt dokumentu struktūru, bet gan dažādu objektu savstarpējās attiecības. Grafi to nodrošina labāk nekā koki, kuri pamatā ir tikai grafa paveids.



1.1. att. **RDF grafa trijnieks ar 2 mezgliem un 1 savienojošu šķautni**

Resursiem, kā piemēram XML, ir raksturīga problēma, ka tiem nav vienotu identifikatoru. Pat, ja divi dokumenti satur informāciju par līdzīgām tēmām, lietotie identifikatori var būt pavisam nesaistīti. Var gadīties arī, ka vienu un to pašu resursu identificē ar vairākiem identifikatoriem un, līdzīgi, vienu identifikatoru lieto dažādiem resursiem. Lai šo nenoteiktību risinātu, RDF lieto URI, lai skaidri identificētu katru resursu. URI ir URL vispārinājums, kuru mēdz arī saukt par URN. Identifikatorus mēdz arī dēvēt par IRI, kas ir URL vispārinājums. Tie ļauj unikāli identificēt resursu un ne vienmēr tie norāda uz tīmekļa resursu, lai gan ir redzama vizuāla līdzība [2].

RDF vizuālais diagrammu attēlojums ļauj cilvēkam vieglāk uztvert šos grafus, bet tas nav piemērots priekš datorsistēmu lietošanas. Ir dažādi formāti un valodas, kā RDF grafus var serializēt mašīnlasāmā formātā. Daži populārākie no tiem ir Turtle, N-Triples, JSON-LD, RDF/XML.

1.2. RDFS

RDFS nodrošina datu modelēšanas vārdnīcu priekš RDF datiem. Tas ir RDF shematisks paplašinājums, kas nodrošina mehānismus, kā aprakstīt grupas ar saistītiem resursiem un attiecībām. Šajos mehānismos ietilpst klašu un lauku sistēma, kas ir līdzīga objektorientētajām programmēšanas valodām, kā piemēram, C#. Atšķirībā no tipiskajām klašu un lauku sistēmām RDFS nedefinē kādi lauki ir konkrētai klasei, bet gan definē to, uz kurām klasēm attiecās konkrētais lauks. Lietojot RDFS shēmu, ir neformāli pieņemts lietot "rdfs" nosaukumu telpu (namespace) un to identificē ar IRI "http://www.w3.org/2000/01/rdf-schema#".

Apskatot klašu jēdzienu, ir vairākas pamatklases, kuras piedāvā RDFS vārdnīca. Tālāk tiks aprakstītas dažas svarīgākās no tām:

- rdfs:Resource – Visu, ko apraksta RDF, dēvē par resursiem. Visi resursi ir rdfs:Resource instances.
- rdfs:Class – Zem šīs klases ietilpst visi RDF resursi, kas ir klases. rdfs:Class ir apakšklase klasei rdfs:Resource.
- rdfs:Literal – Zem šīs klases ietilpst visas literāļu vērtības, kā piemēram, teksts, skaitlis. rdfs:Literal līdzīgi kā rdfs:Class ir apakšklase klasei rdfs:Resource.
- rdfs:Datatype – Zem šīs klases ietilpst RDF modeļa atbalstītie datu tipi. Visas rdfs:Datatype instaces ir rdfs:Literal apakšklases.
- rdf:Property – Zem šīs klases ir RDF lauki.

Aplūkojot lauku jēdzienu, līdzīgi kā klasēm, RDFS vārdnīca piedāvā vairākus pamatlaukus, kurus var lietot. Tālāk tiks aprakstīti daži svarīgākie no tiem:

- rdfs:range – Norāda, kādu klašu instances var saturēt konkrētais lauks. Piemēram, "Majdzivnieks rdfs:range Ziditajs" norāda to, ka lauks Majdzivnieks var saturēt klases Ziditajs instances.
- rdfs:domain – Norāda to, kurām klasēm ir izvēlētais lauks. Piemēram, trijnieks "PedejaIzvirdumaDatums rdfs:domain Kalns" norāda, ka, ja kāda trijnieka predikāts ir PedejaIzvirdumaDatums, tad šis resurss ir Kalns instance.
- rdf:type – Norāda, ka resurss ir klases instance. Šis predikāts nāk no RDF vārdnīcas.

- rdfs:subClassOf – Norāda apakšklašu sakarību.
- rdfs:subPropertyOf – Norāda apakšlauku sakarību [3].

Analizējot sīkāk lauku rdfs:domain ir novērojama nepilnība. rdfs:domain ļauj pēc lauka izspriest, kāda ir resursa klase. Bet tas vairākos gadījumos varētu būt neprecīzi. Aplūkojam sekojošu gadījumu:

- ex:Name rdfs:domain ex:Person;
- ex:Name rdfs:domain ex:Island;

Šeit ir novērojams, ka, ja RDF datos objekts satur tikai lauku ex:Name, nav skaidrs, vai šis objekts ir ex:Person, vai ex:Island. Lai nerastos šādas situācijas, nekas cits neatliek, kā RDFS klasi raksturot ar vairākiem pēc iespējas unikāliem laukiem tā, lai šo lauku kopa būtu pēc iespējas unikāla starp citām definētajām RDFS klasēm. Tad pēc RDF datu objekta laukiem ir iespējams viennozīmīgi noskaidrot datu objekta RDFS klasi.

1.3. OWL

OWL ir semantiskā tīmekļa valoda, kas ir paredzēta, lai attēlotu bagātas un sarežģītas zināšanas par lietām, lietu grupām un to savstarpējām attiecībām. Valoda ir balstīta uz loģiku tā, lai to varētu vienkāršoti lietot datorprogrammas. OWL dokumenti tiek dēvēti par ontoloģijām, kuras ir iespējams publicēt iekš World Wide Web un kurās ir iespēja atsaukties uz citām OWL ontoloģijām. OWL ir daļa no W3C semantiskā tīmekļa tehnoloģiju kopas, kas iekļauj arī RDF, RDFS, SPARQL un citas tehnoloģijas. Šobrīd jaunākā versija ir "OWL 2", kas tika publicēta 2012. gadā un kuru izstrādāja W3C tīmekļa ontoloģijas darba grupa [4]. OWL 2 ontoloģijas nodrošina klases, laukus, indivīdus un datu vērtības, kas tiek glabāti kā semantiskā tīmekļa dokumenti. OWL 2 ontoloģijas var tikt izmantotas kopā ar informāciju, kas ir pierakstīta kā RDF, kā arī dalīšanās ar OWL 2 ontoloģijām notiek caur RDF dokumentiem [5].

Šī valoda ir radīta ar mērķi formulēt, dalīties un spriest par zināšanām par kādu interesējošu domēnu. Lai izprastu, kā OWL 2 reprezentē zināšanas, būtu jāizprot 3 pamata jēdzieni:

- Aksiomas – Pamata apgalvojumi, kurus OWL ontoloģija izsaka.
- Entītijas – Elementi, kurus lieto, lai atsauktos uz īstās dzīves objektiem.
- Izteiksmes – Entītiju kombinācijas, lai veidotu sarežģītus aprakstus.

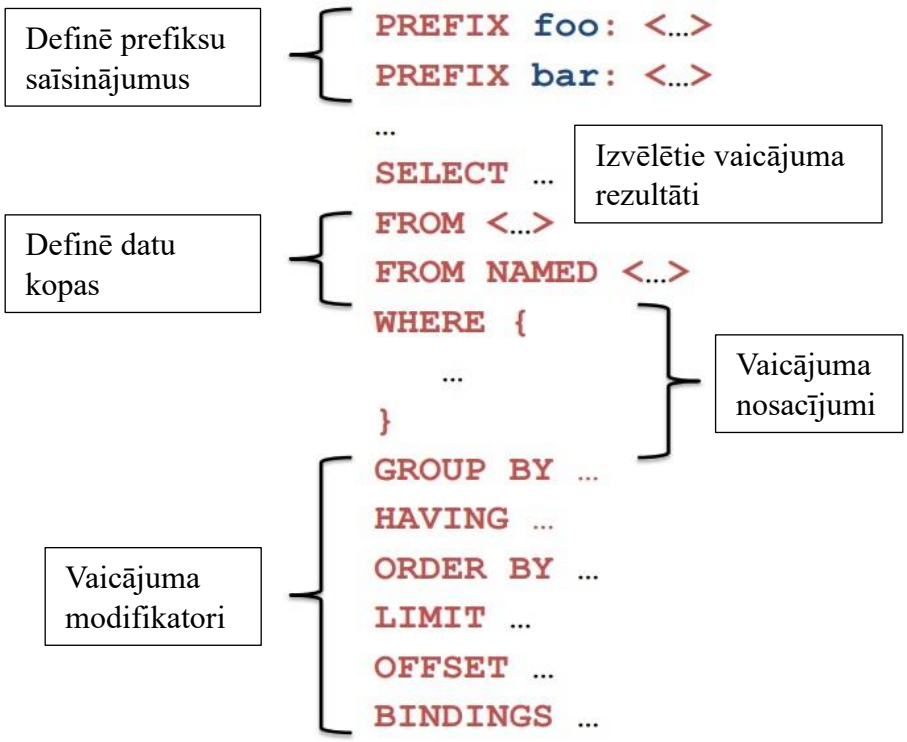
Lai gan OWL 2 mērķis ir uzkrāt zināšanas, tas nav spējīgs attēlot visus cilvēka zināšanu aspektus. Valoda ir vairāk tēmēta, kā spējīga modelēšanas valoda priekš cilvēka zināšanu noteiktām daļām. Modelēšanas rezultātā tiek iegūtas ontoloģijas [6].

1.4. SPARQL

SPARQL ir standarta vaicājumu valoda un protokols priekš saistītajiem atvērtajiem datiem un priekš RDF datubāzēm. Valoda ir izstrādāta daudzveidīgu datu vaicāšanai, tādēļ tā spēj efektīvi izvilkt informāciju no nevienveidīgiem datiem, kas glabājas dažādos formātos un avotos. SPARQL pēdējo versiju izstrādāja 2013. gadā W3C. Līdzīgi kā SQL ļauj lietotājiem iegūt datus no relāciju datubāzes, SPARQL nodrošina to pašu funkcionalitāti NoSQL grafu datubāzēm, piemēram, GraphDB. Papildus tam SPARQL vaicājumus ir iespējams izpildīt uz jebkuru datubāzi, kas ir apskatāma kā RDF grafs caur starpnieku. Piemēram, relāciju datubāzēm var izpildīt SPARQL vaicājumus, izmantojot RDB2RDF kartēšanas programmatūru [7].

SPARQL piedāvā 4 vaicājumu pamatformas:

- SELECT – Atgriež visu vai apakškopu no mainīgajiem, kam atbilst uzdotais vaicājums (skat. 1.2. att.).
- CONSTRUCT – Atgriež RDF grafu, kurš ir veidots, aizvietojot mainīgos trijnieku veidnēs.
- ASK – Atgriež Būla vērtību, norādot, vai uzdotais vaicājums izpildās.
- DESCRIBE – Atgriež RDF grafu, kas apraksta atrastos resursus.



1.2. att. **SPARQL SELECT vaicājuma struktūra** [8]

SPARQL vaicājumu sākumā ir raksturīga saīsinājumu definēšana priekš RDF resursiem (skat. 1.2. att.). Tas tiek darīts ar operatoru PREFIX ar mērķi radīt īsāku vaicājuma kopējo pierakstu. Visvairāk izmantotā no vaicājumu pamatformām ir SELECT vaicājums. Tā kopējā struktūra stipri līdzinās SQL valodai. SPARQL vaicājumos ir iespējams arī izmantot apakšvaicājumus, dažādas iebūvētās funkcijas, loģiskos un matemātisko operatorus un citas iespējas. Kopš SPARQL jaunākās versijas 1.1, valoda ļauj veikt arī datu izmaiņas caur tādām vaicājumu pamatformām kā INSERT, DELETE, LOAD, CLEAR, CREATE, DROP [8].

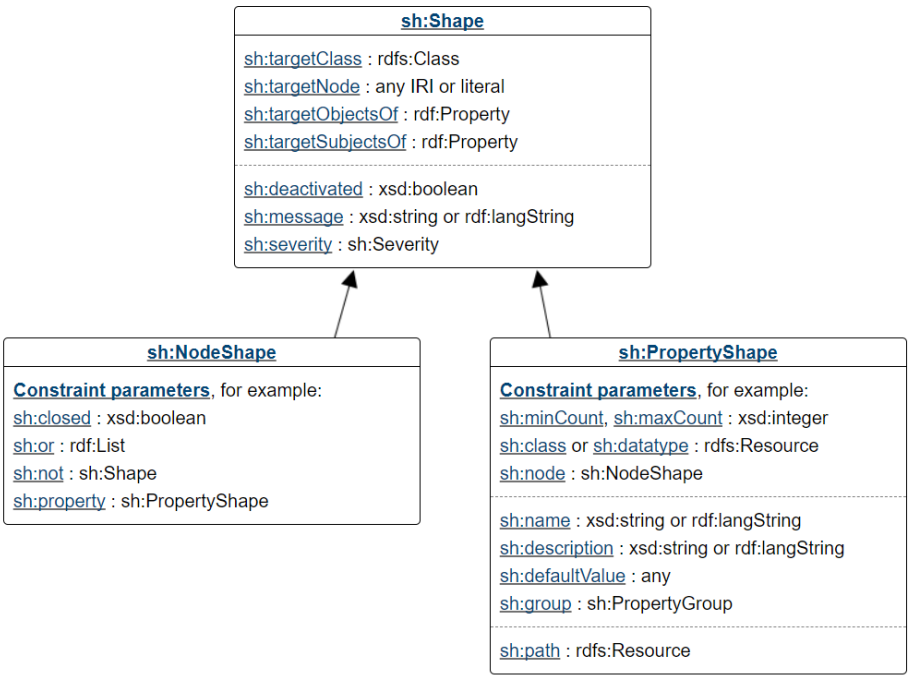
1.5. SHACL

SHACL ir W3C izstrādāts standarts ar mērķi validēt RDF tipa grafu datubāžu saturu atšķirībā no RDFS shēmām un OWL ontoloģijām, kas apraksta datu struktūru. RDFS un OWL definīcijas ļauj veikt tikai secinājumus par datiem, piem., kāda ir konkrēta datu mezgla klase, savukārt SHACL ļauj nodefinēt stingrākus likumus, kuriem ir jāizpildās datos. Validēšanas nosacījumi un ierobežojumi tiek izteikti, aprakstot tādus konceptus kā Shape. Katrā no Shape tiek norādīts, uz kuriem no RDF grafa mezgliem attiecās konkrētā Shape validācijas likumi, piem., tas varētu

attiekties uz kādas klases visām instancēm vai arī uz visiem objektiem, kam ir konkrēts lauks. Ir pieejami 2 veidu Shape:

- Mezglu Shape (NodeShape) – ierobežojumi tiek definēti uz norādītajiem grafa mezgliem.
- Lauku Shape (PropertyShape) - ierobežojumi tiek definēti uz norādīt grafa mezglu laukiem [9].

Valoda piedāvā daudz iespēju ar ierobežojumiem, ko var definēt, piemēram, laukam var norādīt tā tipu, izmēru, skaitu u.c. Daļa no biežāk lietotajiem laukiem ir aplūkojami 1.3. attēlā. SHACL piedāvā arī tādas iespējas, kā SPARQL funkciju definēšana, likumi, kas attiecas uz mezglu izteiksmēm, likumu izpildes secība.



1.3. att. **SHACL biežāk lietotie validācijas lauki** [24]

2. VIZUĀLĀ PIEEJA SPARQL VAICĀJUMU VEIDOŠANĀ

SPARQL valoda ir specifiska vaicājumu valoda, kas ir paredzēta semantiskā tīmekļa komūnai. Šajā komūnā SPARQL ir guvis lielu popularitāti, bet to nevar salīdzināt ar tādām plaši zināmām un lietotām vaicājumu valodām, kā MySQL, T-SQL u.c. Lai valodas iesācējiem un neprofesionāļiem būtu vienkāršāk lietot tās iespējas, ir radīti vairāki rīki, kas nodrošina SPARQL vaicājumu veidošanu, izmantojot vizuālo pieeju, piemēram, diagrammas. Šādas pieejas mērķis ir panākt, ka lietotājam nav jāzina visas SPARQL valodas sintakses iezīmes, bet gan ir vairāk jāorientējas SPARQL valodas iespējās, kuras pēc tam var modelēt, izmantojot lietotājam draudzīgus un saprotamus rīkus. Šādu vizuālu rīku izstrāde rada daudz izaicinājumu. Tiem ir jāspēj balansēt starp 2 svarīgām prasībām – "nodrošināt visas iespējas, kuras piedāvā teksta pieejas SPARQL" un "nodrošināt lietotājam draudzīgāku lietošanu, kā tas ir teksta pieejai". Ja ir labi izpildītas šīs 2 prasības, tad rīkam ir potenciāls piesaistīt arī profesionālus lietotājus, kam tekstuālu vaicājumu rakstīšana nesagādā problēmas, bet šāda vizuāla rīka lietošana laika un piepūles ziņā ir labāka. Viens no veidiem, kā vizuāli rīki spēj samazināt vaicājumu veidošanas piepūli, ir piedāvājot vēlamās vērtības izmantojot automātiskās pabeigšanas laukus. Salīdzinot vizuālo pieeju ar tekstuālo, var apgalvot arī, ka vizuālā pieeja mēdz upurēt daļu no valodas iespējām un funkcionalitātes, liekot uzsvaru uz svarīgāko konkrētā lietojuma funkciju ērtu un vienotu lietošanu.

Šajā nodaļā tiks apskatīti un izvērtēti populārākie rīki, kuros ir iespējams izstrādāt SPARQL vaicājumus, izmantojot vizuālo pieeju.

2.1. ViziQuer

ViziQuer ir LUMII izstrādāta vizuālo vaicājumu notācija un rīks, kas spēj aprakstīt bagātīgus datu vaicājumus, izmantojot diagrammas. Tajā tiek veikta vizuālu diagrammu kartēšana uz tekstuālu SPARQL valodu, nodrošinot bagātīgu vizuālo vaicājumu izpildi pār zināšanu grafiem. Rīks ir pieejams publiskā GitHub repozitorijā ar nosaukumu LUMII-Syslab/viziquer.

Diagrammas piedāvā plašu un bagātu funkcionalitāti, uz kuras pamata tiek ģenerēti SPARQL vaicājumi. Tālāk tiks apskatīta daļa no diagrammu iespējām:

- Pamata vizuālie vaicājumi – Diagrammas tiek veidotas no taisnstūriem, kas apraksta instances un no šķautnēm, kas apraksta instanču attiecības. Instancēm var izvēlēties atlasāmos laukus, to izteiksmes, norādīt vai lauks ir obligāts un veikt citas darbības (skat. 1. pielik.).

- Agregācija un grupēšana – Instancēm ir iespējams veikt dažādas agregācijas operācijas, piemēram, count(), avg(). Veicot agregāciju, vienlaicīgi var norādīt citus atlasāmos laukus. Šajā gadījumā tiek veikta grupēšana pēc šiem laukiem. ViziQuer nodrošina gan agregāciju, gan grupēšanu pēc vairākiem laukiem (skat. 2. pielik.).
- Apakšvaicājumi – Tā ir būtiska SPARQL 1.1. funkcionalitāte, kas ievērojami paplašina vaicāšanas iespējas. ViziQuer diagrammas to nodrošina ar apakšvaicājuma šķautni un taisnstūri, kas ir pievienots pie šīs šķautnes. Tipiski apakšvaicājums ir agregācija. Šķautne var būt obligāta, neobligāta vai ar negāciju (skat. 3. pielik.).
- Atsauču saites – Papildus kokveida struktūrai, ViziQuer ļauj veidot atsauces starp atsevišķiem mezgliem, norādot papildus attiecību nosacījumus, kam ir jāizpildās (skat. 4. pielik.).
- Vaicājumu struktūras paplašinājumi – Ir pieejami dažādi šķautņu tipi: brīvās šķautnes tips "++" un vienādo datu šķautnes tips "==". Ir iespējams veidot vienības mezglu "[ ]" un apvienojuma mezglu "[+]" priekš papildus vaicājumu strukturēšanas. Mezgliem un laukiem var piešķirt nosaukumus (aliases), lai uz tiem atsauktos no citām vaicājuma vietām (skat. 5. pielik.).
- Izteiksmju notācija – Rīks ļauj izmantot tekstuālas izteiksmes gan nosacījumos, gan laukos. Tipiski ar izteiksmēm tiek veidots salīdzināšanas nosacījums. Pamatā tiek pārklātas visas SPARQL piedāvāto izteiksmju iespējas. Papildus tam ir pieejami dažādi saīsinājumi darbam ar tekstu.
- Izpētīšanas vaicājumi – diagrammās var ievietot SPARQL mainīgo nosaukumus, sākot ar "?", tādā veidā sniedzot iespēju pētīt, kā no diagrammas tiek ģenerēts SPARQL vaicājums (skat. 6. pielik.).

ViziQuer ir iespējams uzstādīt lokālajā ierīcē. Uzstādīšanas soļi ir detalizēti aprakstīti rīka GitHub vietnē. 3.4. sadaļā ir apskatāms sīkāks apraksts, kā ViziQuer tika uzstādīts uz lokālās ierīces priekš maģistra darba vajadzībām.

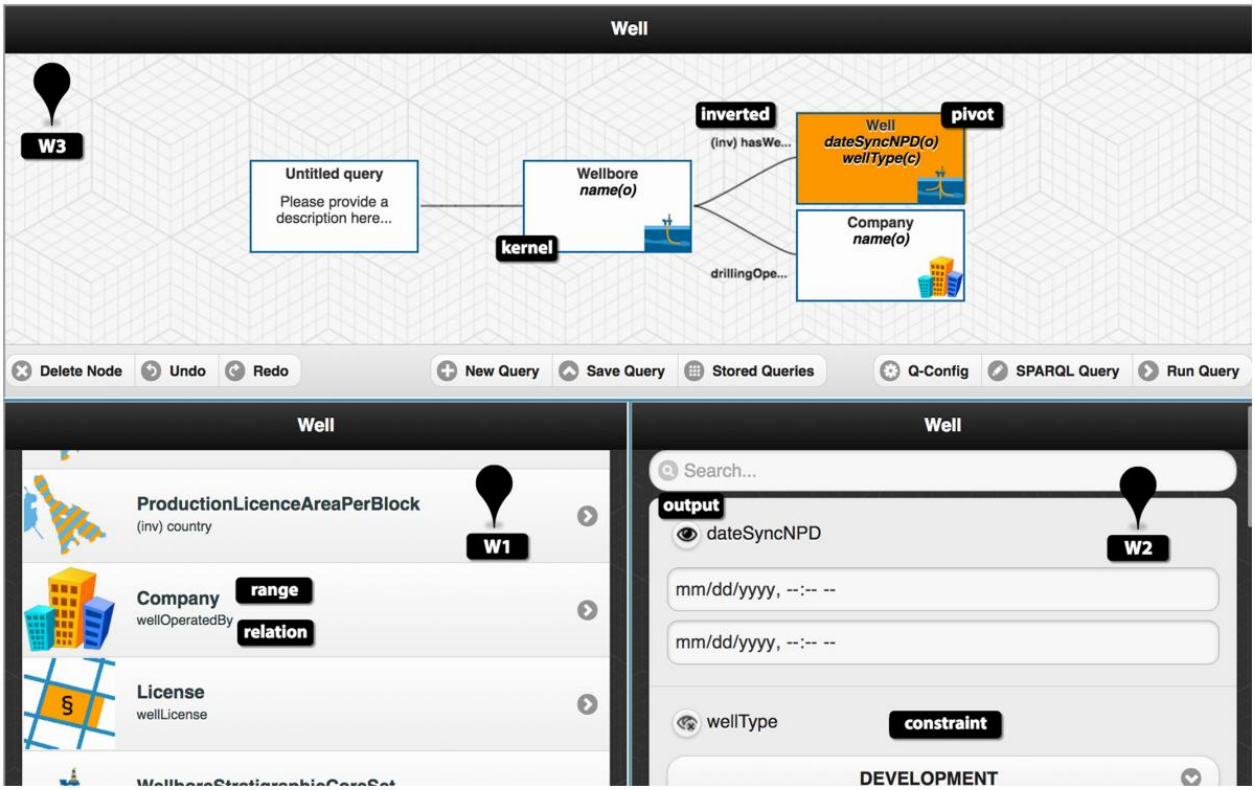
2.2. OptiqueVQS

OptiqueVQS ir vizuālu vaicājumu formulēšanas rīks ar mērķi izpaust informācijas vajadzības vaicājumu veidā pār ontoloģijām. Tas sastāv no saskarnes un navigācijas grafa, kas ir izvilkts no izvēlētās ontoloģijas. Saskarnes komponentes tiek attēlotas atbilstoši informācijai, kas glabājas navigācijas grafā. Navigācijas grafs RDF modeļa veidā tiek pārvaldīts, izmantojot RDFox

argumentētāju (reasoner). Rīks sniedz iespēju ielādēt jebkuru OWL ontoloģiju un izsaukt vaicājumus jebkuram SPARQL galapunktam. OptiqueVQS ir ar atvērtu pirmkodu, kurš ir pieejams GitLab vietnē [10].

OptiqueVQS lietotnes dizains ir iedalāms vairākos biežāk lietotajos logrīkos, kuri tālāk tiks apskatīti:

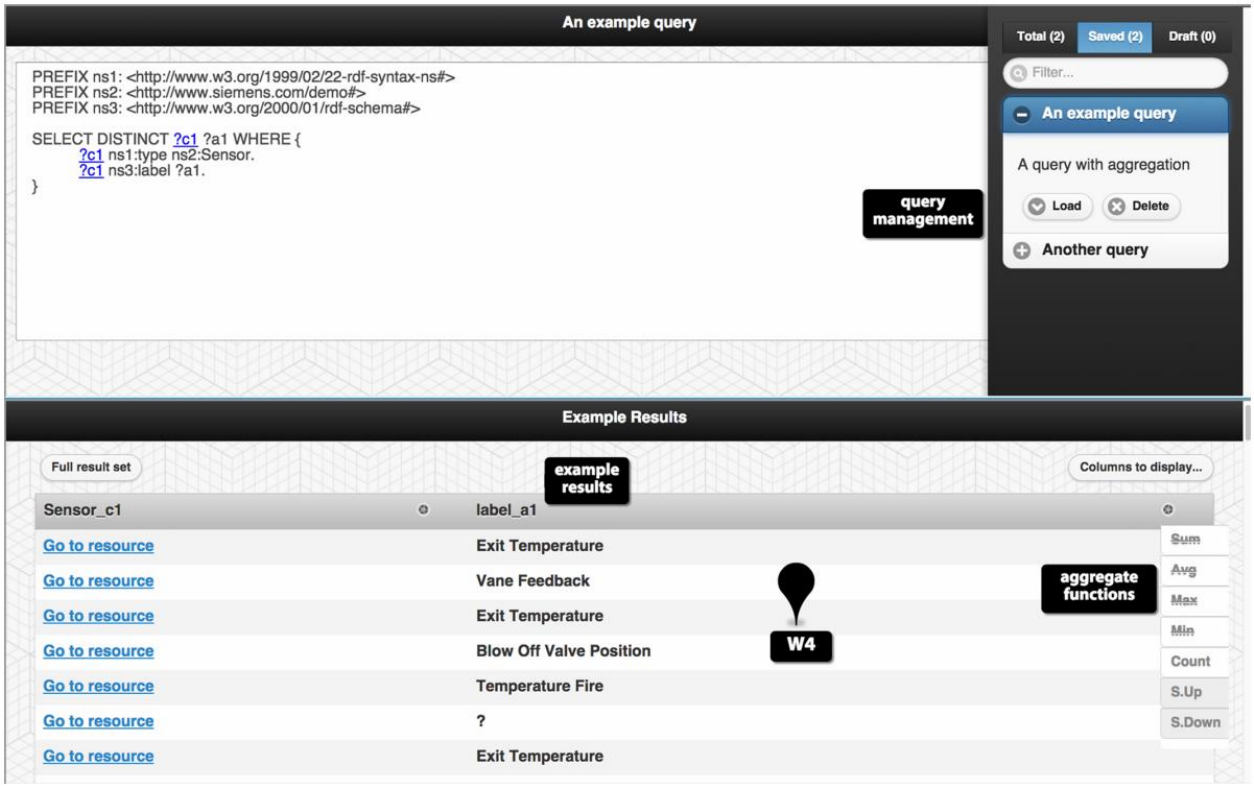
1) 2.1. attēlā apakšējā kreisajā malā ar W1 ir atzīmēts izvēlnes balstīts Query by Navigation (QbN) logrīks, kurā ar nosaukumiem un bildēm tiek attēloti koncepti, starp kuriem lietotājs var izvēlēties, tādā veidā pakāpeniski būvējot vaicājumu.
2) 2.1. attēlā apakšējā labajā malā ar W2 ir atzīmēts logrīks, kas piedāvā formas atbilstoši izvēlētajam konceptam. Tajās lietotājs var atlasīt interesējošos mainīgo atribūtus vai arī ierobežot tos, uzliekot filtrus.
3) Augšā, W3 logrīkā var apskatīt izveidotā vaicājuma vizualizāciju, kurā ir iespējams pārvietot un izvēlēties interesējošos blokus (skat. 2.1. attēlā).



2.1. att. **OptiqueVQS rīka sākotnējais skats** [11]

4) Spiežot pogu "Run Query" tiek izpildīts būvētais vaicājums un 2.2. attēlā ir apskatāms skats pēc vaicājuma izpildes. Augšā ir redzams uzģenerētais SPARQL vaicājums skatīšanās režīmā. Vaicājumus ir iespējams pārvaldīt, tos saglabājot un ielādējot. Apakšā ar W4 ir

atzīmēts rezultātu skats, kas tiek iegūts pēc vaicājuma izpildes. Tajā dati tiek attēloti tabulārā veidā ar iespēju veikt agregāciju, kārtošanu, eksportēšanu un citas operācijas [11].



2.2. att. **OptiqueVQS rīka sākotnējais skats** [11]

Šī darba izstrādes brīdī, rīkam bija ierobežotas iespējas tā izmēģināšanā un uzstādīšanā. Tālāk tiks aprakstīti OptiqueVQS uzstādīšanas veidi atbilstoši GitLab vietnē pieejamajai dokumentācijai un to rezultāti:

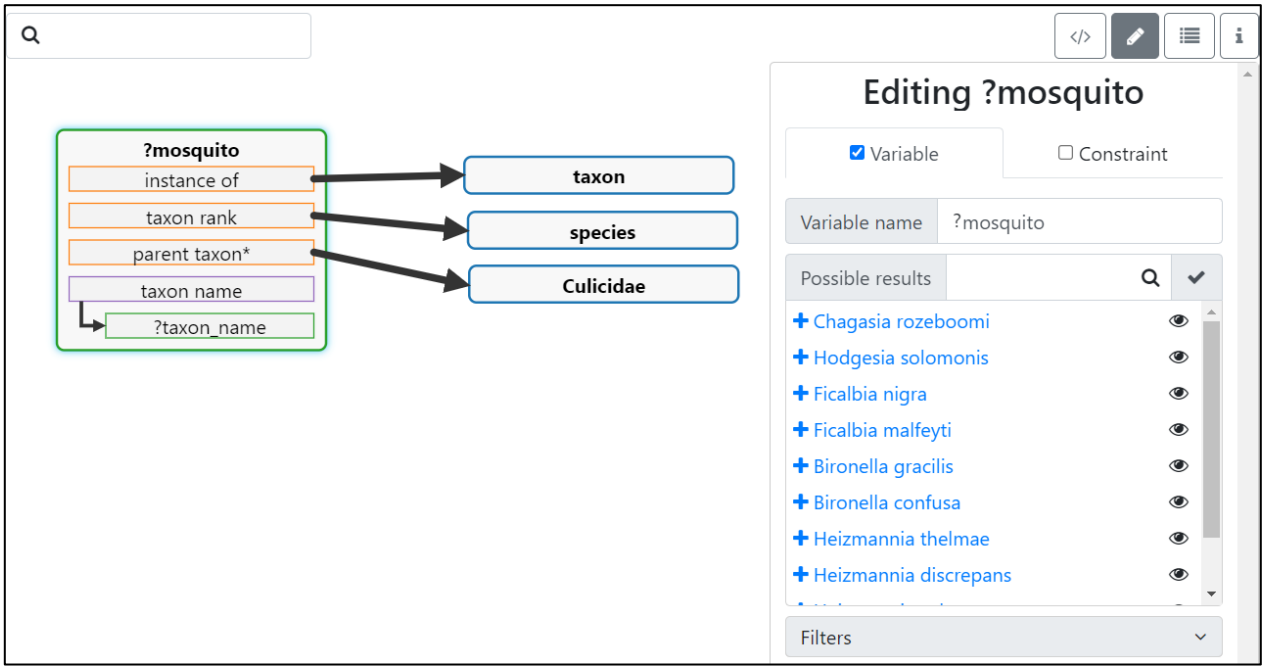
- Tīmekļa saite uz publisku oriģinālo versiju – vietne nebija pieejama, attēlojot 401 HTTP statusa kodu un tālāk pieprasot ielogošanās informāciju, kas nebija pieejama.
- Kompilētas versijas lejupielāde un darbināšana – OptiqueVQS rīks veiksmīgi atvērās, bet brīdī, kad tiek ielādēta ontoloģija, rīks izdod kļūdu un tālāk nav lietojams.
- Manuāla pirmkoda lejupielāde un kompilēšana – rūpīgi pildot GitLab vietnē aprakstītos soļus, varēja veiksmīgi uzstādīt un lietot OptiqueVQS rīku lokālajā ierīcē. Rīka iedarbināšanai bija nepieciešamība pēc instrukcijas manuāli uzstādīt tādas atkarības (dependencies) kā RDFox un ontology-services-toolkit, izmantojot Maven būvēšanas rīku.

2.3. RDFExplorer

RDFExplorer ir Čīles pētnieku izstrādāts rīks, kurš nodrošina vizuālu un interaktīvu uz grafiem balstītu izpēti, kas ļauj neprofesionāļiem vienlaicīgi navigēt un izpildīt vaicājumus zināšanu grafos. Ir veikti pētījumi, ka, lietojot šo rīku ar Wikidata zināšanu grafu, lietotāji spēj paveikt vairāk uzdevumus nekā izmantojot Wikidata vaicājumu palīgu un rakstot vaicājumus tekstiski. Lietojamības aptaujā arī tika novērots, ka lietotāji labprātāk vēlas lietot RDFExplorer nekā manuāli formulēt vaicājumus [12].

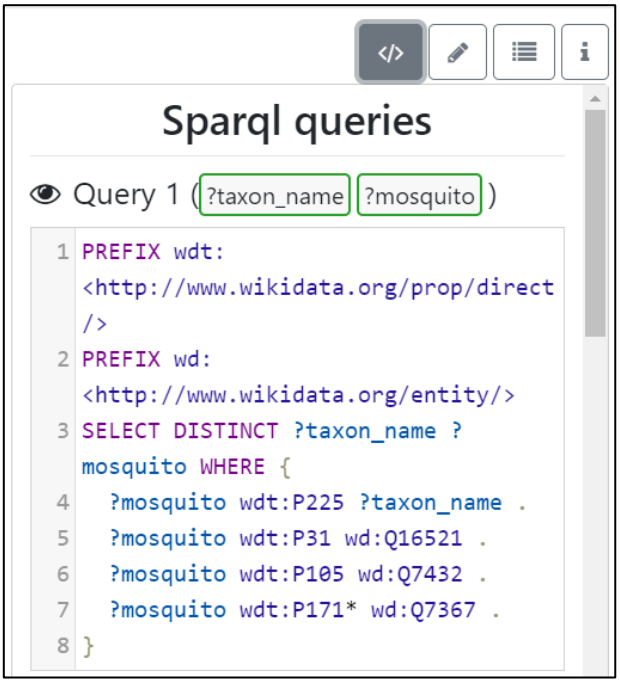
Rīks ir ar atvērtu pirmkodu, kurš atrodas GitHub vietnē. To var lietot gan tiešsaistē caur vietni https://www.rdfexplorer.org, gan arī uzstādīt uz lokālās ierīces. Uzstādot šo rīku lokāli, izstrādātājam ir iespēja mainīt dažādus iestatījumus – piemēram, nomainīt mērķa zināšanu grafu no Wikidata uz DBpedia [13].

Atverot RDFExplorer ir iespēja iziet cauri lietotnes tūrei, kurā pa soļiem tiek nodemonstrētas rīka iespējas. Šī iespēja ir noderīga jauniem rīka lietotājiem. Lai sāktu modelēšanu, lietotājam ir iespēja meklēt kādu konkrētu objektu, izmantojot meklēšanas lauku augšējā kreisajā stūrī, vai arī pievienot jaunu mainīgo, izmantojot konteksta izvēlni, kas pieejama no labā peles klikšķa. 2.3. attēlā ir redzams RDFExplorer. Tajā ir jau izveidots vaicājums, kas atlasa moskītus un to taksonomiskās grupas, kuriem atbilst norādītie nosacījums. Mainīgo sākumā tiek liktas "?" zīmes. Pa labi ir redzams moskīta bloka rediģēšanas panelis. Tajā var mainīt mainīgā nosaukumu, pārskatīt rezultātus, pielikt filtrus, kā arī aizvietot mainīgo ar konstanti. Diagrammā zem "?mosquito" ir redzami vairāki nosacījumi, kam ir jāizpildās, kā arī tiek atlasīts mainīgais "?taxon_name".



2.3. att. **RDFExplorer rīks**

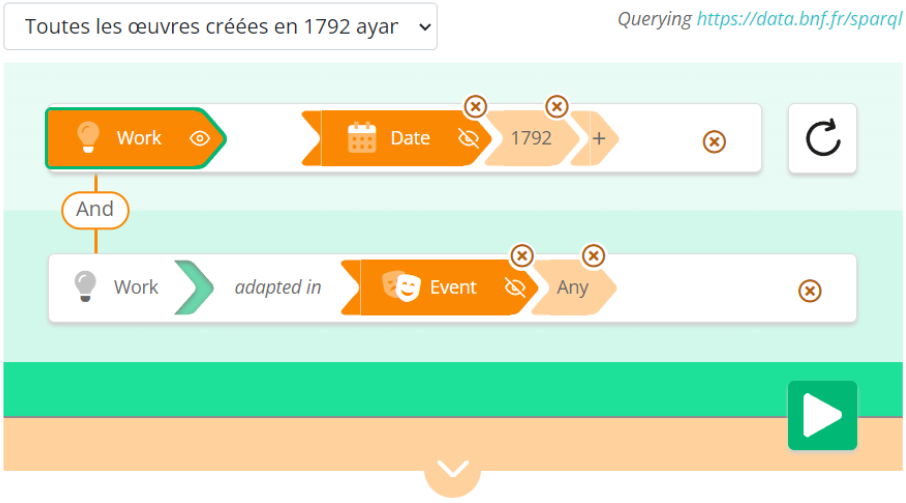
Labajā panelī pārslēdzot sadaļas, ir pieejama "Sparql queries" sadaļa., kas ir apskatāma 2.4. attēlā. Tajā tiek attēlots uzģenerētie SPARQL vaicājumi balstoties uz izveidotajām diagrammām. Attēlā var redzēt abus iepriekšminētos mainīgos zaļās kastēs. Vaicājuma rezultāti ir apskatāmi, vai nu atverot diagrammas bloka paneli un apskatot pieejamās vērtības, vai arī iekopējot uzģenerēto vaicājumu attiecīgajā SPARQL gala punktā. Tā kā tiešsaistes RDFExplorer vietne izmanto Wikidata zināšanu grafu, tad šajā gadījumā vaicājums būtu jāizpilda vietnē https://query.wikidata.org.



2.4. att. **RDFExplorer rīka "Sparql queries" sadaļa**

2.4. Sparnatural

Sparnatural ir vizuāls klienta puses SPARQL vaicājumu būvētājs, kura mērķis ir izpētīt un navigēt RDF zināšanu grafus. Tas atbalsta parastu grafu šablonu izveidi, izvēloties vērtības no "autocomplete" tipa meklētājiem, izkrītošajām izvēlnēm un citiem rīkiem. To konfigurē caur OWL konfigurācijas failu (to var rediģēt Protégé un citos rīkos), kurš definē komponentē attēlotās klases un laukus. Nākotnē izstrādātāji ir plānojuši pievienot SHACL balstītu konfigurāciju [14]. Rīka lietotāja saskarne ir iedalāma 3 dažādās sadaļās, par kurām tālāk tiks sīkāk aprakstīts.



2.5. att. **Sparnatural rīka vaicājuma izstrādes sadaļa**

Atšķirībā no citiem apskatītajiem rīkiem, Sparnatural vaicājumi tiek būvēti secīgi izvēloties objektus, filtrus un nosacījumus nevis brīvi veidojot diagrammas un sasaistot tās. Piedāvātās vizuālās būvēšanas iespējas ir stipri pielāgotas konkrētajam lietojumam un zināšanu apgabalam. 2.5. attēlā redzamajā piemērā ir redzama implementācija, kas ir balstīta uz https://data.bnf.fr zināšanu apgabalu, kura mērķis ir padarīt Francijas Nacionālās Bibliotēkas datus vairāk pieejamus tīmeklī. Pirmais solis, būvējot vaicājumu, ir izvēlēties objektu, kurš tiek meklēts, šajā gadījumā "Work". Tālāk katrā no rindām pēc nepieciešamības tiek norādīts, kādai īpašībai ir jāizpildās, piemēram, 1. rindā tiek norādīts, ka mākslas darbs ir radīts 1792. gadā.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdarelationships: <http://rdvocab.info/RDARelationshipsWEMI/>
SELECT DISTINCT ?aaa WHERE {
  ?aaa rdf:type <http://rdvocab.info/uri/schema/FRBRentitiesRDA/Work>.
  ?aaa <http://rdvocab.info/Elements/dateOfWork> <http://data.bnf.fr/date/1792/>.
  ?aaa ^<http://rdvocab.info/RDARelationshipsWEMI/workManifested> ?Event_2.
  ?Event_2 rdf:type <http://purl.org/dc/dcmitype/Event>.
}
ORDER BY (?aaa)
```

2.6. att. **Sparnatural rīka SPARQL ģenerēšanas sadaļa**

Vaicājuma veidošanas laikā, reālā laikā no tā pamata tiek ģenerēts SPARQL vaicājums, kurš atrodas zem vizuālā skata (skat. 2.6. att.). Tam ir pieejama sintakses iekrāsošana, ar to var dalīties, kā arī izpildīt.

Table 1000 results in 0.24 seconds

Simple view Ellipse Filter query results Page size: 50

Could not render results with the table plugin, the results currently are rendered with the Table plugin.

	this
1	<http://data.bnf.fr/temp-work/0013f85ca4f785a464f48b57d668a725/#about>
2	<http://data.bnf.fr/temp-work/0026018500006f5e0a0aa2767a83abb6/#about>
3	<http://data.bnf.fr/temp-work/003e963d3870167bc6dd4d066255b5a0/#about>
4	<http://data.bnf.fr/temp-work/0075461a5460b8ab46a272193655e310/#about>
5	<http://data.bnf.fr/temp-work/016cc6ad9a5d622954df07690454ff28/#about>
6	<http://data.bnf.fr/temp-work/0180dead2342333cae788eeb16706fc8/#about>

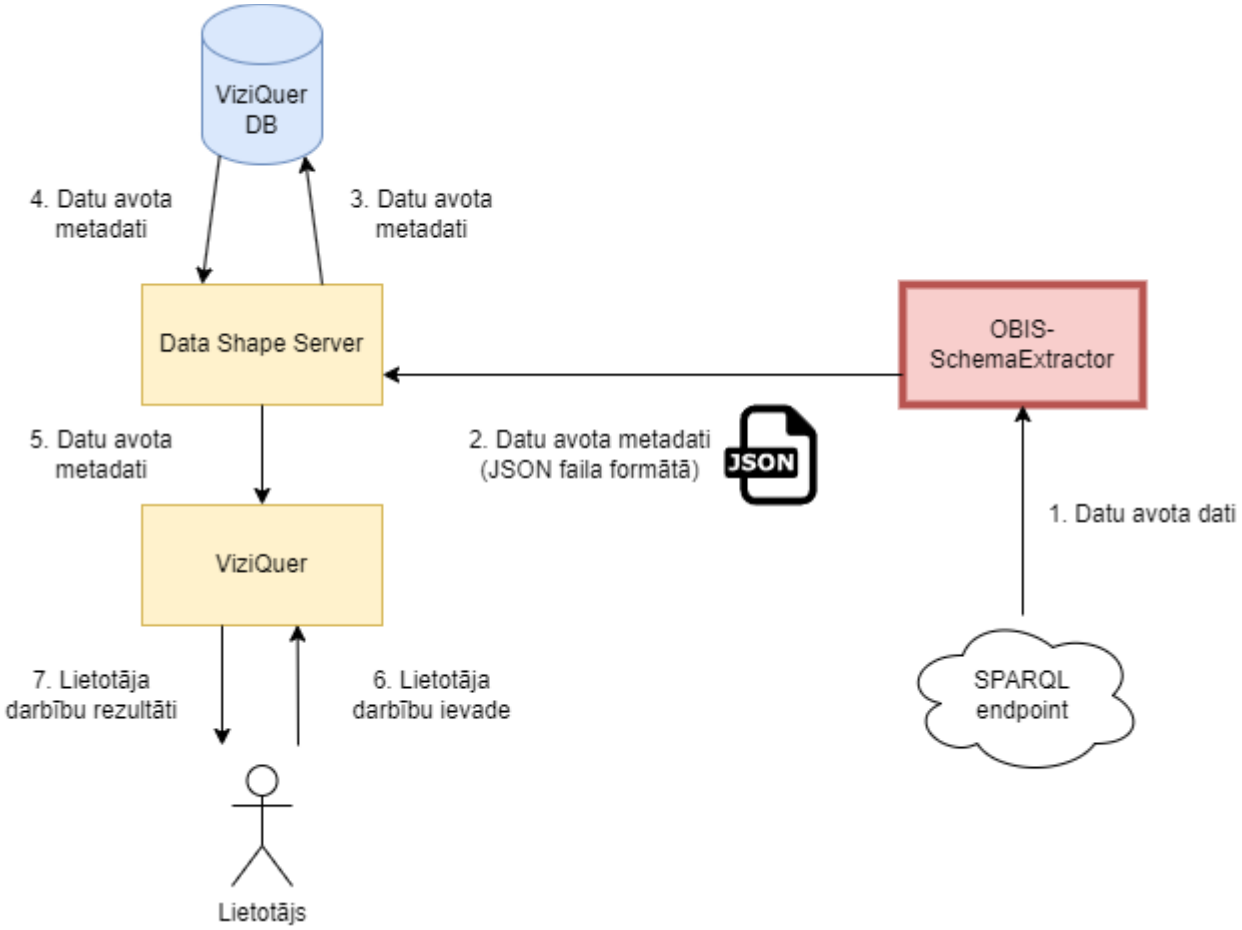
2.7. att. **Sparnatural rīka rezultātu sadaļa**

Pēc vaicājuma izpildes, pašā apakšā, ir redzami zināšanu apgabala dati, kas iegūti, izpildot SPARQL vaicājumu (skat. 2.7. att.). Ir iespējams izvēlēties ierakstu skaitu lapā, lejupielādēt datus, kā arī meklēt rezultātos.

3. SHACL-2-VIZIQUER RĪKA IZSTRĀDE

3.1. Rīka darbības konteksts

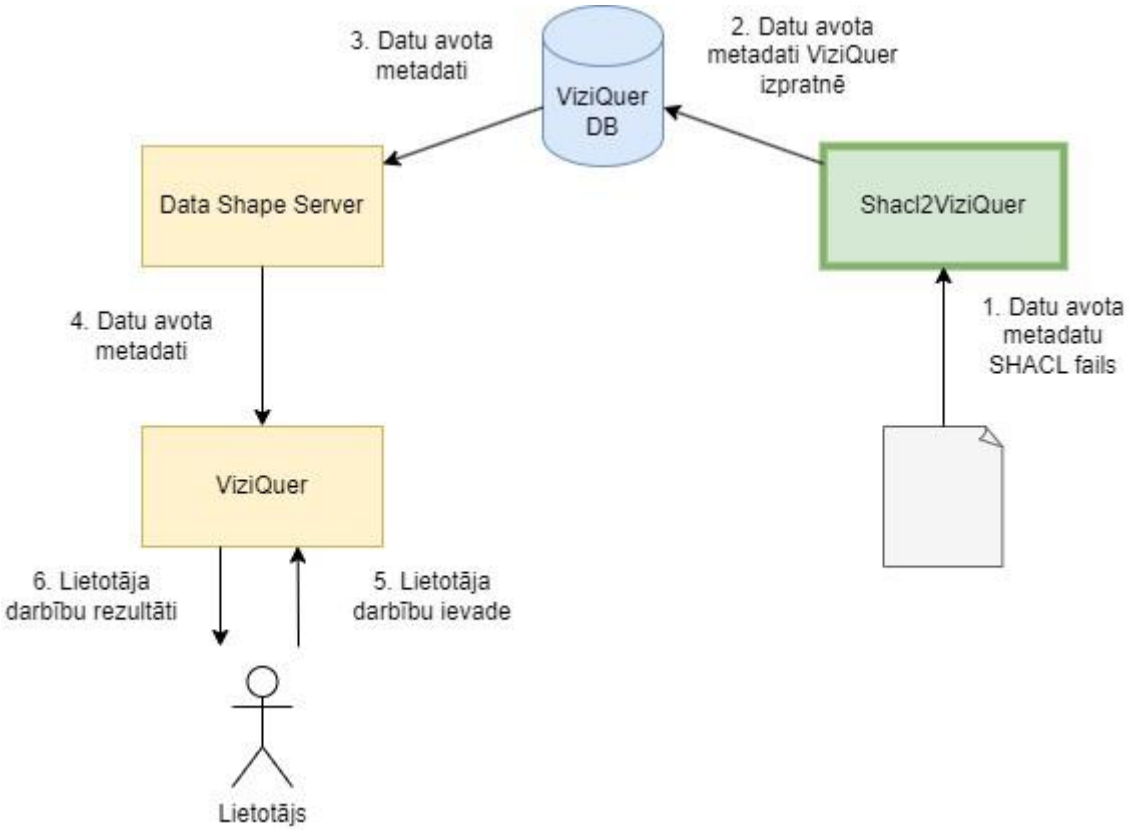
Lai labāk izprastu to, kā shacl-2-viziquer rīks ietilpst pašreizējā ViziQuer rīku kopskatā, tika izveidotas 2 datu plūsmu diagrammas, attēlojot ViziQuer rīku kopskatu pirms shacl-2-viziquer ieviešanas un pēc.



3.1. att. **Esošā ViziQuer rīku ekosistēma**

3.1. attēlā ir apskatāma sākotnējā datu plūsma starp ViziQuer rīkiem, lai ViziQuer datubāzē saglabātu konkrēta datu avota metadatus, kurus pēc tam ViziQuer rīks izmanto, lai uzlabotu lietotāja pieredzi lietotnē. Ar sarkano krāsu ir atzīmēts OBIS-SchemaExtractor rīks, kuram kā alternatīva maģistra darba ietvaros tika izstrādāts shacl-2-viziquer rīks. Tālāk tiks aprakstīts attēlā redzamais metadatu izgūšanas un attēlošanas process pa soļiem:

1. Metadatu ekstrakcija no SPARQL galapunkta - tiek veikti vairāki īpaši sagatavoti SPARQL vaicājumi izvēlētajam datu avotam, lai iegūtu visaptverošu metadatu informāciju par datu avotu.
2. Metadatu saglabāšana JSON failā - pēc SPARQL vaicājumu izpildes iegūtā metadatu informācija tiek sagatavota un saglabāta JSON failā.
3. JSON faila imports ViziQuer datubāzē – Data Shape Server projektā ar atsevišķu palīgrīku tiek veikts JSON faila imports ViziQuer DB. Rezultātā ViziQuer rīkam ir pieejami metadati jeb datu shēma par konkrētu datu avotu, piemēram, DBpedia RDF datiem.
4. Datu avota metadatu nodošana Data Shape Server servisam.
5. Datu avota metadatu nodošana ViziQuer rīkam.
6. Lietotājs lieto ViziQuer lietotni, veidojot vizuālus SPARQL vaicājumu un tos izpildot.
7. Lietotājam ViziQuer rīks piedāvā uzlabotu lietošanas pieredzi, piedāvājot dažāda veida priekšā rādīšanas funkcionalitāti, zinot izvēlētā datu avota metadatus.



3.2. att. **ViziQuer rīku ekosistēma ar shacl-2-viziquer rīku**

3.2. attēlā ir redzama ViziQuer rīku kopskats iekļaujot maģistra darbā izstrādāto shacl-2-viziquer rīku kā OBIS-SchemaExtractor rīka alternatīvu. Ir novērojams, ka kopējais soļu skaits ir samazinājies, jo, tā vietā, lai ģenerētu JSON failu, kuru Data Shape Server spēj importēt datubāzē, shacl-2-viziquer pa tiešo raksta iegūto metadatu informāciju ViziQuer datubāzē. Tālāk ir sīkāk aprakstīti attēlā redzamie soļi:

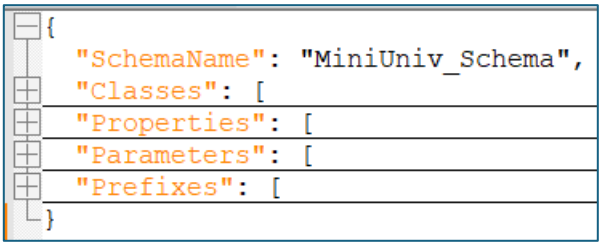
1. SHACL faila nolasīšana shacl-2-viziquer rīkā – shacl-2-viziquer nolasa un parsē izvēlētā datu avota SHACL failu uz programmā izveidotajām datu struktūrām.
2. Datu avota metadatu ierakstīšana ViziQuer datubāzē.
3. Datu avota metadatu nodošana Data Shape Server servisam.
4. Datu avota metadatu nodošana ViziQuer lietotnei.
5. Lietotājs lieto ViziQuer lietotni, veidojot vizuālus SPARQL vaicājumu un veicot to izpildi.
6. Līdzīgi, kā 3.1. attēlā, lietotājam ViziQuer rīks piedāvā uzlabotu lietošanas pieredzi.

3.2. ViziQuer datu shēmas ekstrakcija ar OBIS-SchemaExtractor

Viena no būtiskām iespējām vizuālos SPARQL vaicājumu izstrādes rīkos ir priekā-teikšanas funkcija. Tā ir nepieciešama brīžos, kad diagrammās tiek izvēlēts klases nosaukums, atlasāmie lauki vai filtri. Datus par zināšanu grafu ir iespējams iegūt brīdī, kad tiek izpildīts SPARQL vaicājums. Diagrammu veidošanas brīdī SPARQL vaicājums vēl nav pieejams, tādēļ ir nepieciešams veids, kā iegūt vairāk informāciju par zināšanu grafu, pirms tiek izpildīts izstrādātais SPARQL vaicājums.

ViziQuer šo problēmu risina, izmantojot datu shēmas, kas satur metadatus par zināšanu apgabaliem. Šīs shēmas tiek glabātas ViziQuer datubāzē un tās tiek iegūtas, veicot datu shēmu ekstrakcijas no zināšanu grafiem, izmantojot tam īpaši izstrādātu rīku – OBIS-SchemaExtractor. Tā ir uz Java balstīta tīmekļa lietotne, kura nodrošina REST API kontrolierus priekš datu shēmas ekstrakcijas no SPARQL vai OWL/RDF gala punktiem. Rezultātā tiek iegūts JSON fails, kuru ar īpaša Data Shape Server importa rīka palīdzību iespējams ieimportēt datubāzē.

Zināšanu apgabali var saturēt lielu datu daudzumu. Šī iemesla dēļ, shēmas ekstrakcijas process ir sadalīts vairākos atsevišķos vaicājumos, kas secīgi iegūst pašu būtiskāko informāciju, neiegūstot visus grafa datus. Daļa no šiem soļiem ir sekojoši: iegūst visas klases un to instanču skaitu, iegūst visus laukus un to trijnieku skaitus, iegūst apakšklašu attiecības u.c [15]. 3.3. attēlā ir apskatāms daļējs uzģenerētais JSON fails.



3.3. att. **OBIS-SchemaExtractor iegūtā datu shēma**

3.3. Plānotā funkcionalitāte

shacl-2-viziquer rīkam idejiski ir vienkāršs pamatuzdevums – nolasīt izvēlēto SHACL failu un tā informāciju ierakstīt norādītajā ViziQuer datubāzē. Sarežģītība slēpjas tajā, kā SHACL failā specificētu struktūru var ierakstīt ViziQuer esošajā tabulu struktūrā, kā arī tajā, kuras no visām SHACL valodas iespējām var ierakstīt tabulu struktūrā. Maģistra darba ietvaros ir plānots attīstīt shacl-2-viziquer rīku tik tālu, lai tas spētu pilnvērtīgi parsēt 2023. gadā veiktā pētījuma DBpedia SHACL failu [16]. Balstoties uz to, kādas SHACL valodas iespējas tiek lietotas šajā failā, tālāk seko shacl-2-viziquer SHACL faila parsēšanas prasības:

- <http://rdfs.org/ns/void#entities> NodeShape tipa atribūta vērtības rakstīšana ViziQuer datubāzē gan priekš klasēm, gan priekš laukiem tā, lai ViziQuer rīkā būtu redzams klašu un lauku skaits.
- <http://www.w3.org/ns/shacl#targetClass> NodeShape tipa atribūta interpretēšana, lai būtu skaidrs, kura no DBpedia klasēm tiek aprakstīta. Rezultātā ViziQuer rīkā jāspēj ieteikt klašu nosaukumus, vizuāli veidojot SPARQL vaicājumus.
- <http://www.w3.org/ns/shacl#property> NodeShape tipa atribūta interpretēšana, lai ViziQuer datubāzē varētu veidot saites starp laukiem un klasēm, kā rezultātā, ViziQuer rīkā veidojot vizuālu vaicājumu pār kādu klasi, tiek piedāvāti tajā esošie lauki.
- <http://www.w3.org/ns/shacl#path> PropertyShape tipa atribūta interpretēšana, lai būtu skaidrs, kuru no DBpedia laukiem apraksta šis Shape. Rezultātā ViziQuer rīkā vizuāli veidojot vaicājumus, izvēloties klases atribūtus, jābūt piedāvātiem DBpedia lauku nosaukumiem atbilstoši šī lauka vērtībai.
- <http://www.w3.org/ns/shacl#or> PropertyShape tipa atribūta interpretēšana. Šajā laukā atrodas dažādas citas lauku grupas, kurām ir dažādi apstrādes nosacījumi, kas tālāk tiks aprakstīti.

- <http://rdfs.org/ns/void#entities> PropertyShape tipa atribūta interpretēšana, lai ViziQuer rīkā būtu apskatāmi lauku instanču un saišu instanču skaiti. Ja ir vairāki entities atribūti iekš iepriekšminētās OR izteiksmes, tad ViziQuer datubāzē jāsaglabā gan lauku instanču kopskaits sasummējot visas OR sastāvdaļas, gan jāsaglabā katrai saitei attiecīgais entities skaits pie nosacījuma, ka OR atribūtu grupā ir norādīts arī class atribūts.
- <http://www.w3.org/ns/shacl#class> PropertyShape tipa atribūta interpretēšana, lai ViziQuer rīkā veidojot vizuālos vaicājumus būtu skaidrs, kādas klases vērtības satur konkrētais lauks. Class var arī nebūt aizpildīts, ja laukam ir vērtības tips, piemēram, string. Rezultātā ViziQuer rīkā, attiecīgi norādītajam class tipam, jāspēj vizuāli veidot saites ar attiecīgo klasi.

3.4. ViziQuer uzstādīšana lokālajā ierīcē

Lai varētu pārbaudīt shacl-2-viziquer rīka rezultātus, ir nepieciešams lokāli uzstādīt ViziQuer un ar to saistītās programmas. Uzstādīšanā ietilpst sekojoši soļi:

- ViziQuer uzstādīšana.
- Data Shape Server uzstādīšana.
- Datubāzes uzstādīšana.

ViziQuer un Data Shape Server programmas ir publiski pieejamas GitHub vietnēs, kurās ir detalizēti aprakstīti uzstādīšanas soļi [17, 18]. Papildus tam ir nepieciešama datubāzes uzstādīšana – šis process arī ir aprakstīts Data Shape Server projektā. Pēc šo soļu izpildes, lokāli tika veikta papildus konfigurēšana, lai ViziQuer risinājums darbotos maģistra darba vajadzībām. Tālāk tiks sīkāk aprakstīts, kā tika veikta ViziQuer vides uzstādīšana uz lokālās vides ar Windows 11 operētājsistēmu.

3.4.1. ViziQuer uzstādīšana

Rīka pamatā tiek lietots MeteorJS ietvars, tādēļ uzstādīšanas pirmajos soļos ir nepieciešams ieinstalēt šo ietvaru. MeteorJS ietvaram ir nepieciešama Node.js <= 14 versija, bet lokālajā ierīcē bija jaunāka versija, tādēļ tika lietots Node Version Manager (NVM), kurš ļauj vienlaicīgi datorā pārvaldīt dažādas Node.js versijas. Tā kā tika lietota Windows 11 operētājsistēma, tad attiecīgi tika instalēta nvm-windows pakotne caur Node Package Manager (NPM). Pēc NVM instalēšanas, mainot Node versiju uz vecāku, bija nepieciešams arī pielāgot NPM, jo tas arī ir balstīts uz Node.js. Rezultātā tika instalēta arī npm-windows-upgrade pakotne, ar kuras palīdzību ir iespējams pārslēgt

NPM versijas uz vēlamo. Pēc Node.js uzstādīšanas uz 14. versiju, tika apskatīta nodejs.org vietnē piedāvātā saderīguma tabula, lai atrastu nepieciešamo NPM versiju [19].

Pēc NPM veiksmīgas uzstādīšanas, tika pielāgots projekta konfigurācijas fails app/.env, ievadot lokālā Data Shape Server URL adresi http://localhost:3344/api un tad tika palaista ViziQuer programma ar komandu "npm run dev". Tika lietota ViziQuer versija no development zara ar commit hash saīsinājumu 67d4fb2a.

3.4.2. Data Shape Server uzstādīšana

Pēc ViziQuer uzstādīšanas sekoja Data Shape Server uzstādīšana pēc GitHub vietnē aprakstītajiem soļiem. Vispirms tika veikta projekta klonēšana uz lokālo ierīci ar "git clone" komandu. Tālāk tika veikta konfigurācijas pielāgošana, veicot labojumus server/.env failā, norādot Postgres servera savienojuma informāciju un norādot Data Shape Server servisa portu 3344.

Data Shape Server esot uzstādītam, tālāk sekoja RDF avotu konfigurēšana caur pirmkodu – tika pārslēgta atpakaļ avotu nolasīšana no datubāzes uz avotu nolasīšanu uz pirmkodu (no util.get_KNOWN_DATA2() uz util.get_KNOWN_DATA()). Pēc tam pirmkodā pieejamais RDF avotu saraksts tika papildināts ar vēl vienu, kurš norāda uz DBpedia SPARQL piekļuves punktu un kura metadatus datubāzē aizpildīja maģistra darbā izstrādātais shacl-2-viziquer rīks. Lokāli tika lietota Data Shape Server versija no main zara ar commit hash saīsinājumu b452993.

3.4.3. Datubāzes uzstādīšana

Data Shape Server darbībai ir nepieciešama Postgres datubāze, tādēļ šajā sadaļā tiks aprakstīta, kā tā tika uzstādīta. Tika izpildīta instrukcija, kas ir aprakstīta projekta GitHub vietnes db-templates mapē. Tālāk tika veikta "empty" datubāzes dublēšana un pārsaukšana uz dbpedia_shacl datubāzi. Šajā datubāzē shacl-2-viziquer rīks veiks metadatu aizpildīšanu.

3.5. SHACL failu pieejamība

Ir svarīgi noskaidrot, kāda ir SHACL failu pieejamība RDF avotiem vairāku iemeslu dēļ. Pirmkārt – lai zinātu, uz doto brīdi, uz kādiem RDF avotiem ir iespējams pielietot izstrādāto shacl-2-viziquer rīku priekš metadatu iegūšanas. Otrkārt – lai būtu reālistiski piemēri, uz kuriem var testēt shacl-2-viziquer darbību. Tālāk šajā sadaļā tiks apskatīta, kāda ir SHACL failu pieejamība dažādiem RDF avotiem.

Tīmekļa resursos pētot tādus RDF avotus kā DBpedia, Wikidata, Schema.org, Virtuoso atklājās, ka šie avoti nepiedāvā SHACL failus, kas aprakstītu RDF datu struktūru. Lai gan šīs vietnes SHACL failus nepiedāvā, publiski ir pieejami dažādi pētījumi, kuros ir izdevies izveidot SHACL failus par šiem avotiem. Kā viens no uzskatāmākajiem tika atrasts 2023. gada pētījums, kuru ir veikuši 3 Dānijas kolēģi. Pētījumā tika izstrādāta Java programma, kura spēj parsēt RDF failus ar .nt formātu un kura ir publiski pieejama GitHub vietnē. Pētījuma ietvaros tika arī pielietots šis rīks, lai iegūtu SHACL failus par 4 dažādiem RDF avotiem – WikiData, DBpedia, YAGO-4, LUBM. Apskatot failu izmērus, visapjomīgākais ir Wikidata SHACL fails ar 1.9 GB apjomu un vismazākais ir LUMB_SHACL fails ar 167.3 kB [16]. shacl-2-viziquer rīka testēšanā pamatā tiks lietots DBpedia_SHACL.ttl fails ar mērenu apjomu – 14.0 MB.

Vietnē Zenodo papildus iepriekšminētajam pētījumam ir arī pieejams 3 Spānijas kolēģu 2020. gadā veikts pētījums. Tiešā veidā apmeklējot vietni nav pieejama pētījuma rakstiskā daļa, bet ir pieejami 2 piemēri ar uzģenerētiem DBpedia SHACL failiem 2.4 MB izmērā [20]. Salīdzinot ar 2023. gadā ģenerēto SHACL failu, šī faila izmērs ir mazāks, tātad tas ir atšķirīgs. Tas dod iespēju daudzveidīgāk apskatīt un interpretēt DBpedia RDF datu struktūru. Viens no aspektiem, kas šiem failiem ir unikāls salīdzinājumā ar 2023. gadā ģenerēto failu, ir tas, ka pie NodeShape ir pievienoti rdfs:label, sh:name un sh:description lauki, kas sniedz vairāk informāciju par aprakstītajiem RDF datiem.

3.6. SHACL interpretatora izvēle

Veicot SHACL interpretatoru analīzi un izvēli tika izvērtēti vairāki faktori. Vispirms tika izvērtēts, kādā programmēšanas valodā vajadzētu izstrādāt shacl-2-viziquer rīku, kas ierobežotu SHACL interpretatoru izvēli. Pēc tam tika izvērtēti dažādi atrastie interpretatori, salīdzināti savā starpā un tika izvēlēts vispiemērotākais no interpretatoriem. Šajā sadaļā tiks sīkāk aprakstīts, kā norisinājās šie posmi.

Izvērtējot to, kādā programmēšanas valodā izstrādāt shacl-2-viziquer rīku, viens no kandidātiem bija C# valoda, ņemot vērā autora iemaņas. Otrs no galvenajiem variantiem bija JavaScript programmēšanas valoda, kurā ir arī izstrādāts ViziQuer un ar to saistītie rīki. SHACL ir cieši saistīts ar semantiskā tīmekļa tehnoloģijām, tādēļ tīmekļa skriptēšanas valoda JavaScript piedāvā vairākas ar RDF un SHACL interpretēšanu saistītas bibliotēkas. Papildus tam, tīmekļa standartu un vadlīniju izstrādes apvienība World Wide Web Consortium (W3C) ir veltījusi vairākus rakstus par JavaScript bibliotēkām, kas ir paredzētas RDF apstrādei jeb par RDFJS bibliotēkām.

Ņemot to vērā, shacl-2-viziquer programmas izstrādei tika izvēlēta JavaScript programmēšanas valoda.

Pēc programmēšanas valodas izvēles ir nepieciešams atrast konkrēto JavaScript bibliotēku, ar kuru tiks veikta SHACL failu nolasīšana un interpretēšana. Bibliotēkas meklēšana tika veikta populārajā Node pakotņu repozitorijā npmjs.com. Pakotnes tika meklētas, izmantojot frāzi "shacl" un sakārtojot rezultātus pēc popularitātes. Darba ietvaros tika apskatītas pirmās 9 no atrastajām bibliotēkām. Kopā meklējuma rezultātos tika atrastas 54 pakotnes. Tālāk seko sīkāks bibliotēku apskats.

- **rdf-validate-shacl; @types/rdf-validate-shacl** – pēc bibliotēkas uzstādīšanas un lietošanas, veiksmīgi izdodas ielasīt SHACL failu. Pēc faila ielasīšanas bibliotēka piedāvā veikt RDF datu validāciju balstoties uz ielasīto SHACL failu, tomēr ielasītā SHACL faila datu struktūras lietot citiem mērķiem ir apgrūtinoši, jo tas ir iebūvēts bibliotēkā un tam nav atsevišķi izstrādāta atbalsta.
- **@comunica/actor-rdf-parse-shaclc; @comunica/actor-rdf-serialize-shaclc** - cieši saistīti ar @comunica projektu, tādēļ ir apgrūtinoši pielāgot un lietot priekš ViziQuer vajadzībām. Comunica ir atvērta pirmkoda projekts, kurš nodrošina zināšanu grafu vaicāšanas ietvaru priekš JavaScript. Tas ļauj elastīgi rakstīt SPARQL un GraphQL vaicājumus pār RDF datu kopām, kas atrodas tīmeklī. Projektu galvenokārt uztur Comunica Association.
- **shacl-engine** – bibliotēkā tiek sagaidīts, ka ttl paplašinājuma failā būs gan SHACL izteiksmes, gan arī dati. shacl-2-viziquer rīkam tas neder, jo rīks nolasīt ttl failu, kurā ir tikai SHACL izteiksmes.
- **shaclc-parse** – vienkārša un viegli lietojama bibliotēka. Tomēr tā neatbalsta pietiekami daudz iespēju. Tai ir atbalsts tikai SHACLC (Shacl compact syntax) sintaksē rakstītiem SHACL failiem.
- **clownface-shacl-path** - ļauj atrast elementus iekš RDF zināšanu grafiem, izmantojot SHACL kā meklēšanas valodu. Tas ir pielīdzināms tam, kā ar XPath ir iespējams atrast elementus iekš XML datiem. Funkcionalitātes nodrošināšanas tiek lietota clownface bibliotēka, kas pēc idejas līdzinās Comunica bibliotēkai – ļauj iegūt un analizēt RDF datus. shacl-2-viziquer noteiktajam uzdevumam ir nepieciešama SHACL interpretēšana un analīze, nevis RDF grafu meklēšanas funkcionalitāte, tādēļ šī bibliotēka nav labākais variants.

- **@rdfine/shacl** - bibliotēka ir maz lietota un bez apraksta, tādēļ sīkāk tā netiek izskatīta.
- **shaclc-write** - ļauj rakstīt RDF/JS quads iekš SHACL kompaktās sintakses dokumentiem. Šis arī nepiedāvā SHACL izteiksmju atsevišķu analīzi, kas ir vajadzīgs šim uzdevumam.

3.1. tabula

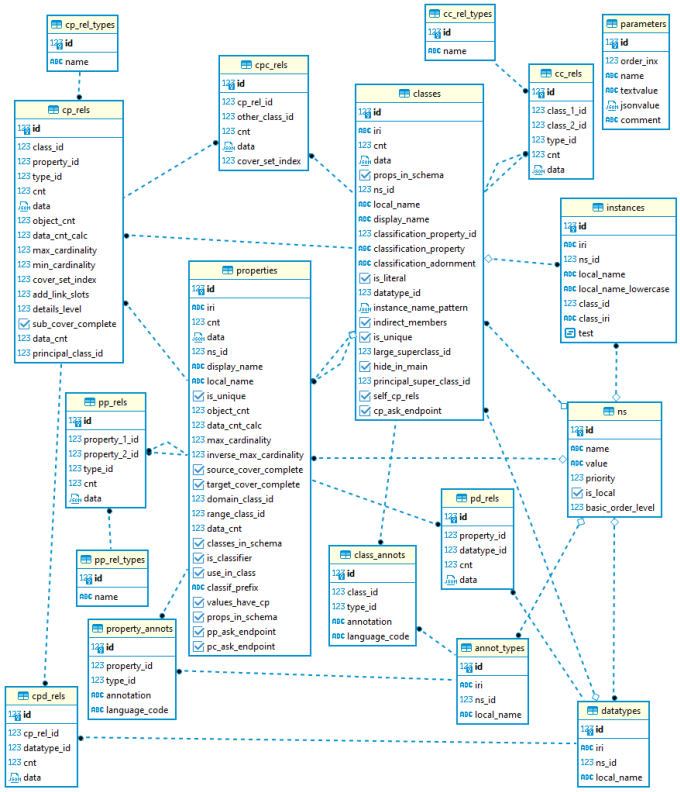
SHACL interpretatoru bibliotēku popularitātes apkopojums

	NPM iknedēļas lejuplādes	GitHub zvaigznes	GitHub komiti
rdf-validate-shacl	1965	88	315
@comunica/actor-rdf-parse-shaclc @comunica/actor-rdf-serialize-shaclc	2717 1375	402	2193
shacl-engine	1400	17	26
shaclc-parse	2645	0	99
clownface-shacl-path clownface	400 8643	4 36	134 196
@rdfine/shacl	50	14	692
shaclc-write	1356	0	47

Pēc apkopotajiem datiem (skat. 3.1. tabulu) var novērot, ka GitHub lietotāju vidū visvairāk ir iecienītas @comunica saistītās bibliotēkas, aiz tām sekojot rdf-validate-shacl bibliotēkai. Tā kā @comunica saistītās SHACL bibliotēkas ir cieši saistītas ar pašu Comunica ietvaru, tad darba ietvaros sākotnēji tika izvēlēts tieši rdf-validate-shacl ietvars. Lietošanas gaitā atklājās, ka ielasītā SHACL faila datu struktūras lietot shacl-2-viziquer mērķiem ir apgrūtinoši, jo šīs struktūras ir iebūvēts bibliotēkā un tām nav atsevišķi izstrādāta atbalsta. Kā rezultātā tika izlemts izmantot citu bibliotēku, kuru arī lietoja rdf-validate-shacl - @zazuko/env-node. Tā nav specifiski SHACL parsēšanas un validācijas bibliotēka, bet gan tā ir vispārīgāka RDF failu parsēšanas bibliotēka. Rezultātā, no bibliotēkas iegūtā datu struktūra ir salīdzinoši vienkārša un attiecas uz jebkāda tipa RDF ttl failiem, tai skaitā SHACL failiem. Datu struktūra pamatā sastāv no objektu masīva, kur objekts satur RDF trijnieku – objektu, predikātu un subjektu. Tālāk interpretēt SHACL specifiskos apzīmējumus un jēdzienus jau būtu shacl-2-viziquer rīka uzdevums. Lai gan @zazuko/env-node nav starp populārākajām RDF parsēšanas bibliotēkām, tā ir vienkārši lietojama un tās funkcionalitāte ir pietiekama priekš shacl-2-viziquer rīka izstrādes.

3.7. ViziQuer tabulu struktūra

Pēc SHACL faila parsēšanas, shacl-2-viziquer rīkam ir jāveic RDF avota datu shēmas ierakstīšana ViziQuer datubāzē. Lai nodrošinātu pilnvērtīgāku ViziQuer funkciju atbalstu, ir svarīgi zināt, tabulu struktūru, kurā tika rakstīti no SHACL iegūtā RDF avota datu shēma. Šajā sadaļā tiks aprakstītas būtiskākās ViziQuer datubāzes tabulas, kurās shacl-2-viziquer rīks veic aizpildīšanu.



3.4. att. **ViziQuer datubāzes ER modelis**

3.4. attēlā ir redzamas visas datubāze tabulas, kas pieder vienam RDF datu avotam. Katra atsevišķā RDF datu avota tabulas ViziQuer datubāzē nodala, piešķirot un iekonfigurējot dažādas shēmas. Piemēram, DBpedia datu avotam tabulai classes varētu piekļūt caur šādu shēmas un nosaukuma salikumu – "dbpedia.classes". Tālāk sekos būtiskāko tabulu apraksts, kurus shacl-2-viziquer rīks aizpilda:

- **classes** – Satur informāciju par RDF avotā pieejamām klasēm, to nosaukumiem, instanču skaitiem.
- **properties** – Apraksta RDF avota iespējamos laukus, to nosaukumus un to īpašības. Gadījumā, ja laukam ir tikai 1 augšklase, kurai viņš pieder, neskaitot mantošanu, tad domain_class_id satur šīs klases id.

- **cp_rels** – Šajā tabulā tiek reģistrētas saites starp classes un properties tabulām, piefiksējot saites tipu, saišu skaitu un kardinalitāti.
- **cpc_rels** – Tabulā tiek glabātas saites starp classes un cp_rels tabulu. Tas tiek darīts, lai varētu glabāt sarežģītākas RDF avota attiecības, kurās laukam ir gan ienākošā klase, gan izejošā klase.
- **pp_rels** – Satur dažāda tipa saites, kas ir starp 2 laukiem. Piemēram, šajā tabulā tiek pildītas saites starp lauku un no tā izejošajiem iespējamiem apakšlaukiem.

3.8. ViziQuer tabulu struktūras aizpildes process

ViziQuer datubāzē esošajām tabulām var būt vairāki pielietojumi atkarībā no mērķa, kuru nepieciešams sasniegt, tāpēc šajā nodaļā tiek pa soļiem izklāstīts ViziQuer tabulu aizpildes process.

1. **Klašu aizpilde -** Katram SHACL faila NodeShape unikālajam targetClass laukam tiek izveidots jauns ieraksts classes tabulā, piefiksējot, kādas ir RDF avota klases. Klašu aizpildes rezultātā, ViziQuer vizuālo vaicājumu veidošanā lietotājam tiek piedāvāts iespējamo klašu saraksts.
2. **Lauku aizpilde -** Katram SHACL faila PropertyShape unikālajam path laukam tiek izveidots jauns ieraksts properties tabulā, uzskaitot, kādi ir RDF avota lauki.
3. **cp_rels tabulas aizpilde -** Lai ViziQuer vizuālo vaicājumu veidošanā pēc klases pievienošanas diagrammā varētu lietotājs ērti pievienot laukus, shacl-2-viziquer ir nepieciešams aizpildīt cp_rels tabulu ar "outgoing" tipa saitēm. Tādā veidā ViziQuer rīkam ir zināms kuri lauki pieder kurām klasēm, kā rezultātā vizualizācijas skatā pie klasēm ir pieejama lauku priekšā teikšanas funkcionalitāte.
4. **Tabulu cpc_rels un cp_rels aizpilde -** Klasēm var būt dažāda tipa lauki – literāļi vai objekti. Literāļu gadījumā, lauks satur vienkāršu vērtību, piemēram, teksts, skaitlis. Savukārt objektu gadījumā vērtība ir objekts. Šī tipa vērtība pievieno papildus saiti – no lauka uz mērķa objektu. Lai ViziQuer rīks atpazītu lauka iespējamos izejošos objektus, datubāzē vajag aizpildīt 2 tabulas cpc_rels un cp_rels. cp_rels tabulā ir jāpievieno "incoming" tipa saites starp classes un properties tabulām, lai norādītu, kādi katram laukam var būt iespējamie objekti. Piemēram, laukam "students" dažas no iespējamām objektu klasēm varētu būt "Vīrietis" vai "Sieviete". Savukārt cpc_rels tabulā tiek savienoti kopā classes tabulas un cp_rels tabulas tikko izveidotie ieraksti. Tas tiek darīts ar mērķi norādīt

sākotnējo klasi, kurai ir pieejama šī lauku vērtību klašu informācija. Iepriekšminētajā piemērā par lauku "students" tās varētu būt, piemēram, klases "Kurss" un "Universitate".

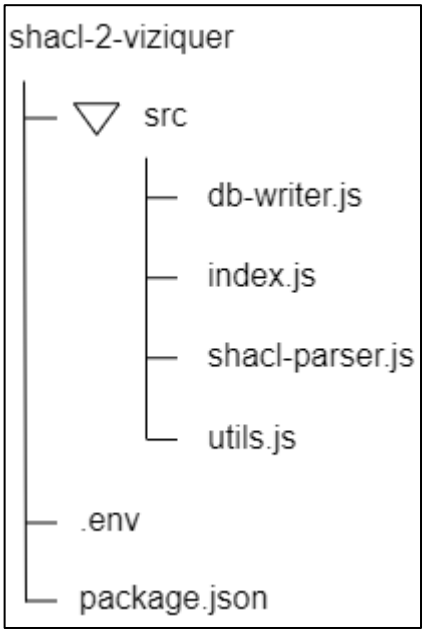
5. **Tabulas pp_rels aizpilde ar "followed by" saitēm -** Iepriekšējais solis nodrošināja, ka ViziQuer vizualizācijas rīkā ir iespējams veidot saiti no klases uz kādu citu klasi, caur klases lauku. Tomēr tas nenodrošina, ka šajā mērķa klasē, kas ir laukam, var izvēlēties, kādi tajā būs apakšlauki. Lai to varētu izdarīt, nepieciešams aizpildīt pp_rels tabulu ar "followed_by" tipa saitēm. Tas ir jāveic starp visiem lauku un mērķa klases lauku pāriem. Rezultātā ViziQuer vizualizācijas rīkā ir iespējams veidot saites starp klasēm un mērķa klasē arī iespējams izvēlēties apakšlaukus.
6. **Tabulas pp_rels aizpilde ar "common_subject" saitēm -** Pēc saišu veidošanas ViziQuer vizualizācijas rīkā, lai būtu iespējams pamatklasei pievienot vai mainīt laukus līdzīgi, kā tas ir vienkāršajā gadījumā bez saitēm, nepieciešams papildināt pp_rels tabulu ar "common_subject" tipa saitēm. Tās ir jāveido starp visiem lauku un augšklašu lauku pāriem.

3.9. Realizētā risinājuma tehniskais apraksts

Šajā sadaļā tiks aprakstīti, kā tehniski tika izstrādāts shacl-2-viziquer risinājums, iekļaujot risinājuma izstrādes vidi, projekta failu struktūru, kā arī pirmkodā iekļautās bibliotēkas.

Risinājuma izstrādē tika lietotas sekojošas izstrādes vides:

- Visual Studio Code v1.88.1 – plaši lietots, daudzfunkcionāls pirmkoda redaktors, kuru 2015. gadā izlaida Microsoft kompānija.
- pgAdmin 4 v7.8 – PostgreSQL datubāžu izstrādes un pārvaldības platforma, kura tika radīta 2002. gadā.



3.5. att. **shacl-2-viziquer projekta būtiskāko failu struktūra**

3.5. attēlā ir apskatāma shacl-2-viziquer risinājuma būtiskāko failu struktūra. Tālāk seko sīkāks failu apraksts:

- src – mape, kas satur rīka JavaScript pirmkodu.
- db-writer.js – nodrošina sagatavoto datu struktūru ierakstīšanu ViziQuer datubāzes tabulās.
- index.js – sākotnēji izpildāmais fails.
- shacl-parser.js – nodrošina SHACL faila parsēšanu un informācijas ierakstīšanu atsevišķās datu struktūrās.
- utils.js – palīgfunkcijas, piemēram, getIriName().
- .env – satur shacl-2-viziquer rīka konfigurāciju, piem., datubāzes savienojuma datus.
- package.json – projekta fails, kurš tiek izveidots visiem projektiem, kas lieto NPM. Satur projekta metadatus, datus par bibliotēkām, kā arī skriptus.

Pirmkodā tika lietotas sekojošas bibliotēkas:

- lodash – utilītbibliotēka, kas nodrošina utilītfunkcijas priekš bieži sastopamiem programmēšanas uzdevumiem, izmantojot funkcionālās programmēšanas paradigmu.
- pg-promise, pg-connection-string – bibliotēkas, kas nodrošina nebloķējošu PostgreSQL klientu priekš Node.js lietotnēm.
- @zazuko/env-node – bibliotēka, kas nodrošina RDF failu parsēšanu. Sīkāk aprakstīts 3.6. sadaļā.

3.10. No SHACL ģenerēta RDF avota pielietošanas demonstrējums

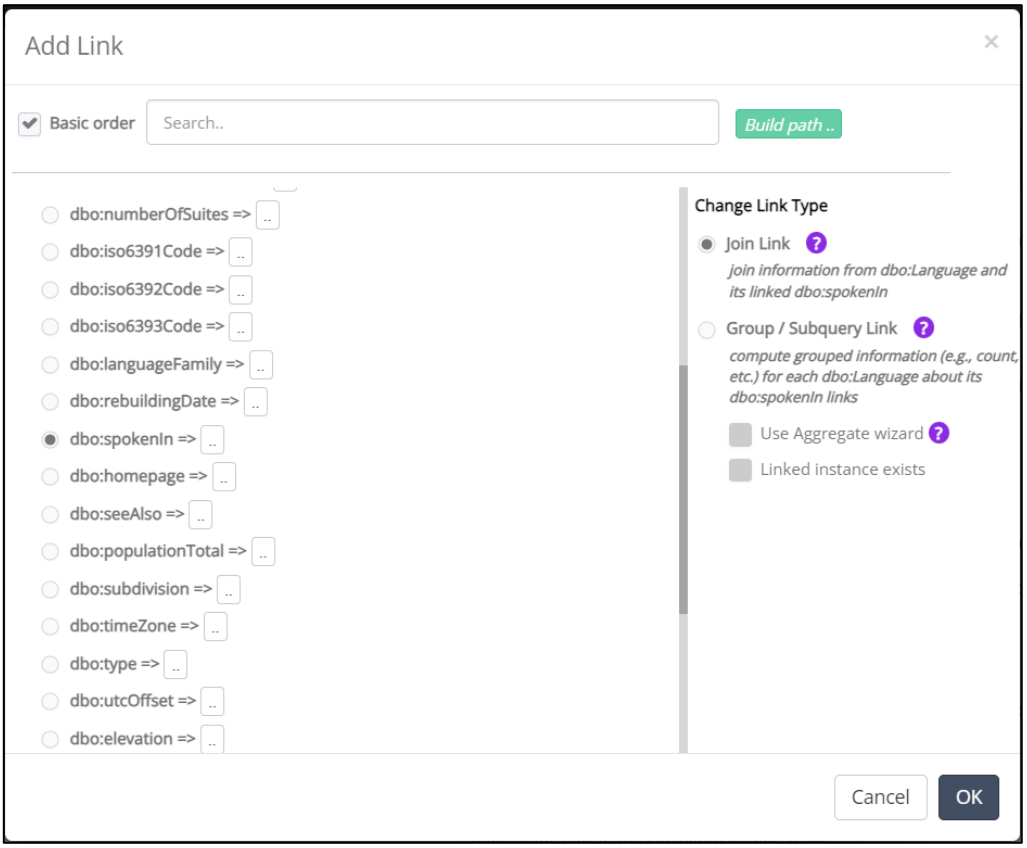
Šajā sadaļā tiks apskatīti dažādi piemēri, kuros tiek lietota no SHACL faila iegūtā meta informācija.

3.10.1. Valstīs runājošo valodu skaitu iegūšana

Šajā piemērā tiks veidots un izpildīts vizuālais vaicājums, kurš saskaita, cik valstīs tiek runāta katra no valodām. Rezultējošā tabula tiek sakārtota dilstošā secībā pēc valstu skaita.

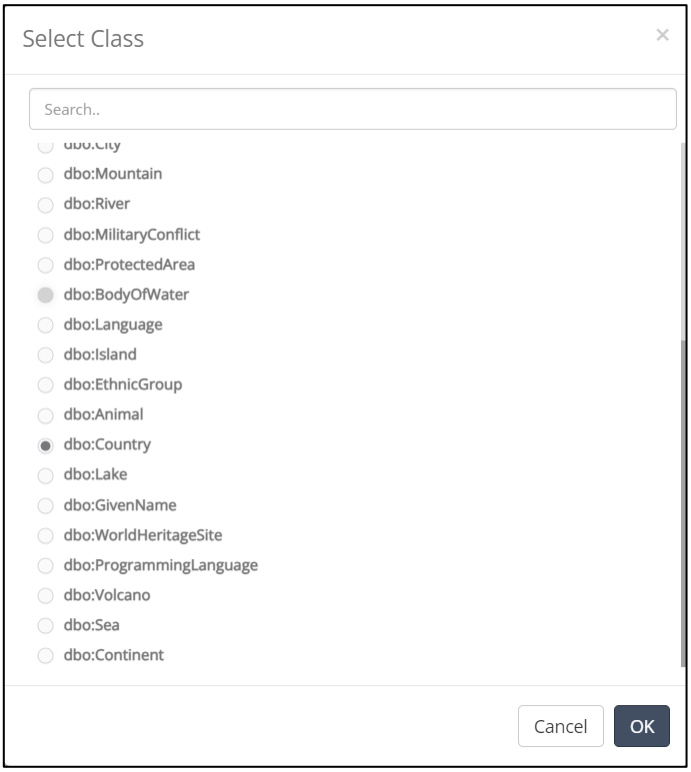
3.6. att. **Valodas piemēra klases izveide**

Vispirms vizuālajā vidē tiek izveidots galvenā interesējošā klase (skat. 3.6. att.). Pēc klases ievilkšanas, labajā panelī ir iespējams iestatīt dažādus atlases nosacījumus. Šajā solī laukā "Name" tiek izvēlēts, ka interesē dati par klasi dbo:Language. Attēlā ir novērojams, ka no SHACL faila iegūtie metadati sniedz lietotājam klašu priekšā-teikšanu, kad tiek nospiests ctrl+space.



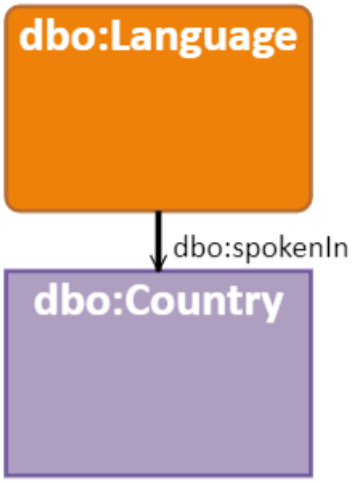
3.7. att. **Valodas piemēra saites izvēle**

Pēc klases izvēlēšanās uz klasi tiek veikts labais klikšķis un tiek izvēlēta opcija “Add Link …”. Rezultātā tiek atvērts 3.7. attēlā redzamais logs. Tajā ir iespējams pievienot klasei saiti uz kādu citu klasi. Sarakstā ir redzami visi izvēlētās klases saišu tipi, kuri ir iegūti no nolasītā SHACL faila. Šajā gadījumā tiek izvēlēta saite dbo:spokenIn un tiek spiesta “..” poga, kas ir redzama pa labi no saites.



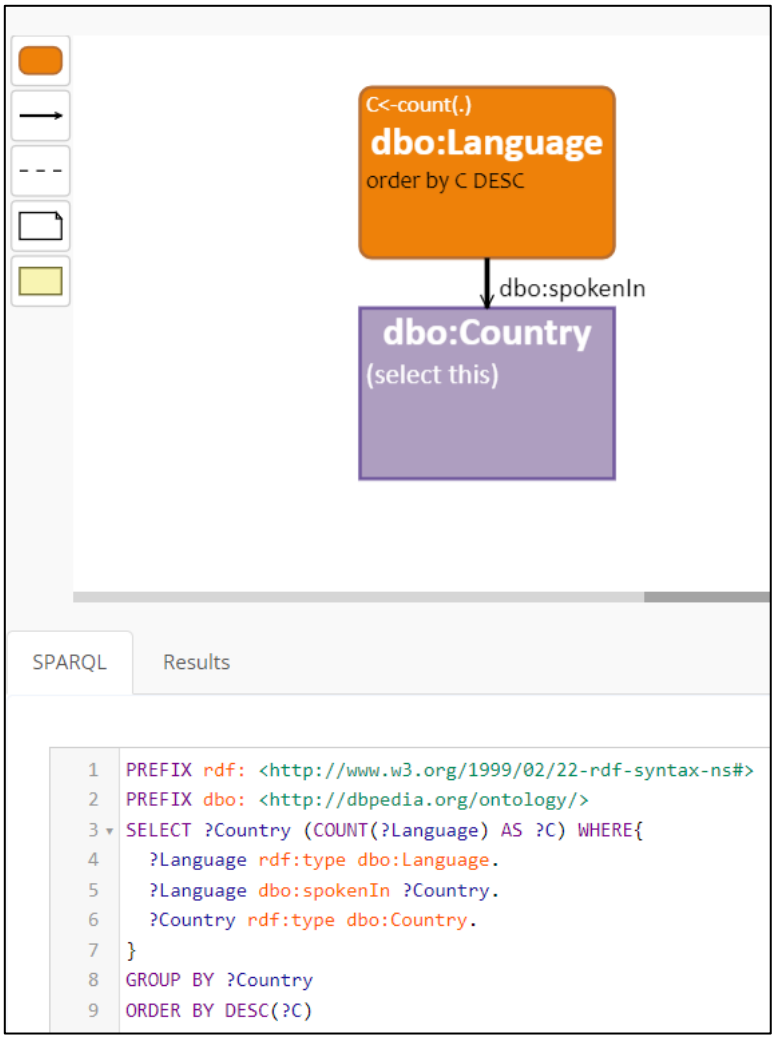
3.8. att. **Valodas piemēra saites mērķa klases izvēle**

3.8. attēlā ir novērojams, ka izvēlētajai saitei jeb predikātam pretī var atrasties dažādas klases. Šajā piemērā tas ir interpretējams tā, ka atlasītās valodas ir iespējams sasaistīt vai nu ar salām uz kurām tās tiek runātas, vai nu ar kontinentiem, kuros tās tiek lietotas, vai ar valstīm u.c. Šajā gadījumā tiek izvēlēta valodu sasaiste ar valstīm, kurās tās tiek lietotas.



3.9. att. **Valodas piemēra vizuālais vaicājums bez laukiem**

Pēc iepriekšminēto operāciju izpildes ViziQuer vidē ir pieejams 3.9. attēlā redzamais vaicājuma vizualizācija. Lai varētu no tā ģenerēt SPARQL vaicājumu, kā minimums, ir jāizvēlas atlasāmie lauki, kas tiks veikts tālākos soļos.



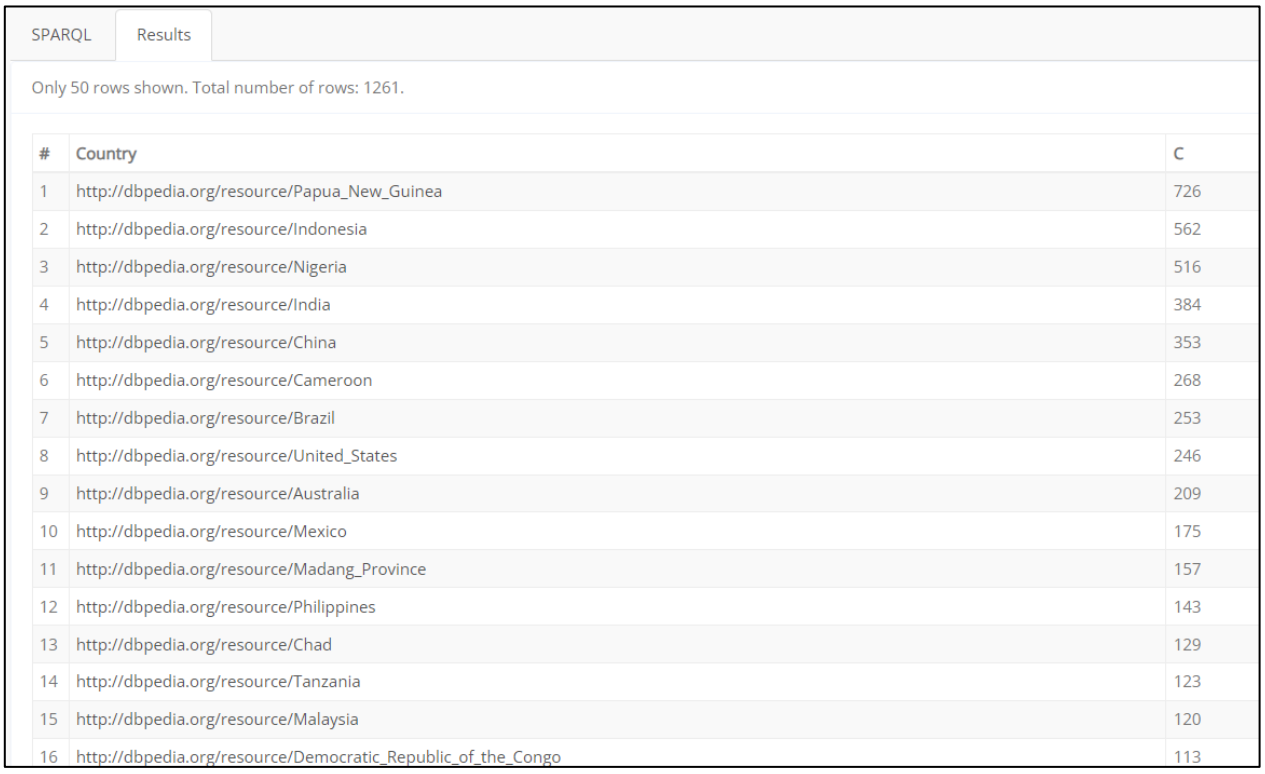
3.10. att. **Valodas piemēra vizuālais vaicājums**

Izmantojot labajā sānā pieejamo paneli ir nodefinēti vairāki vizuālā vaicājuma nosacījumi:

- No dbo:Country klases tiek atlasīts tās IRI kods, norādot sānu panelī, "Attributes" sadaļā, tādu atribūtu, kā "(select this)".
- dbo:Language ieraksti tiek sagrupēti pēc valsts, norādot Aggregate sadaļā agregācijas izteiksmi, kas saskaita katras valodas valstis. Ar "count(.)" tiek norādīts "saskaitīt visus grupas elementus", kur grupa ir dbo:Language IRI kods. Ar "C" tiek norādīts aizstājvārds iegūtajam skaitam.

- dbo:Language ieraksti tiek sakārtoti pēc valstu skaita dilstošā secībā, iekonfigurējot dbo:Language klases "OrderBy" sadaļu labajā sānu panelī.

Pēc vizuālā vaicājuma sagatavošanas tika veikts labais peles klikšķis un tika izvēlēta opcija "Generate query SPARQL", kā rezultātā attēla apakšā tika izveidots SPARQL vaicājums (skat. 3.10. att.).


SPARQL Results

Only 50 rows shown. Total number of rows: 1261.

#	Country	C
1	http://dbpedia.org/resource/Papua_New_Guinea	726
2	http://dbpedia.org/resource/Indonesia	562
3	http://dbpedia.org/resource/Nigeria	516
4	http://dbpedia.org/resource/India	384
5	http://dbpedia.org/resource/China	353
6	http://dbpedia.org/resource/Cameroon	268
7	http://dbpedia.org/resource/Brazil	253
8	http://dbpedia.org/resource/United_States	246
9	http://dbpedia.org/resource/Australia	209
10	http://dbpedia.org/resource/Mexico	175
11	http://dbpedia.org/resource/Madang_Province	157
12	http://dbpedia.org/resource/Philippines	143
13	http://dbpedia.org/resource/Chad	129
14	http://dbpedia.org/resource/Tanzania	123
15	http://dbpedia.org/resource/Malaysia	120
16	http://dbpedia.org/resource/Democratic_Republic_of_the_Congo	113

3.11. att. **Valodas piemēra vaicājuma rezultāti**

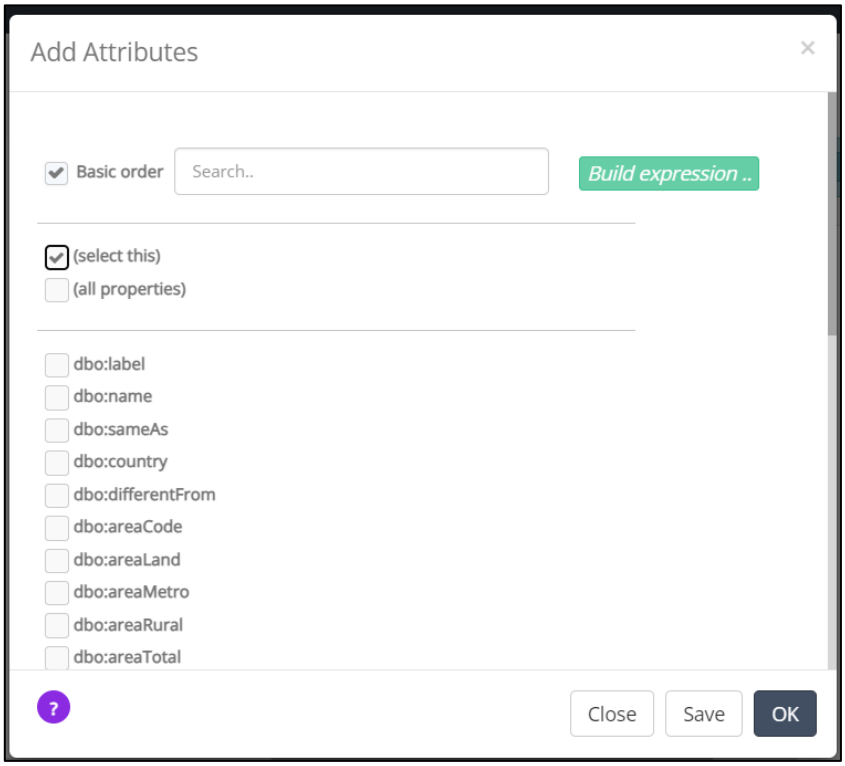
Pēc vaicājuma izpildes, cilnē "Results" ir pieejami rezultāti (skat. 3.11. att.). Kā var novērot, visvairāk valodās tiek runāts Papua-Jaungvinejā, aiz viņas sekojot Indonēzijai, Nigērija, Indijai un citām valstīm.

3.10.2. Uz salām esošo pilsētu atlase

Šajā piemērā tiks veidots un izpildīts vizuālais vaicājums, kurš atlasa visas pilsētas, kurām kā valsts ir norādīta sala.

3.12. att. **Pilsētu piemēra klases izveide**

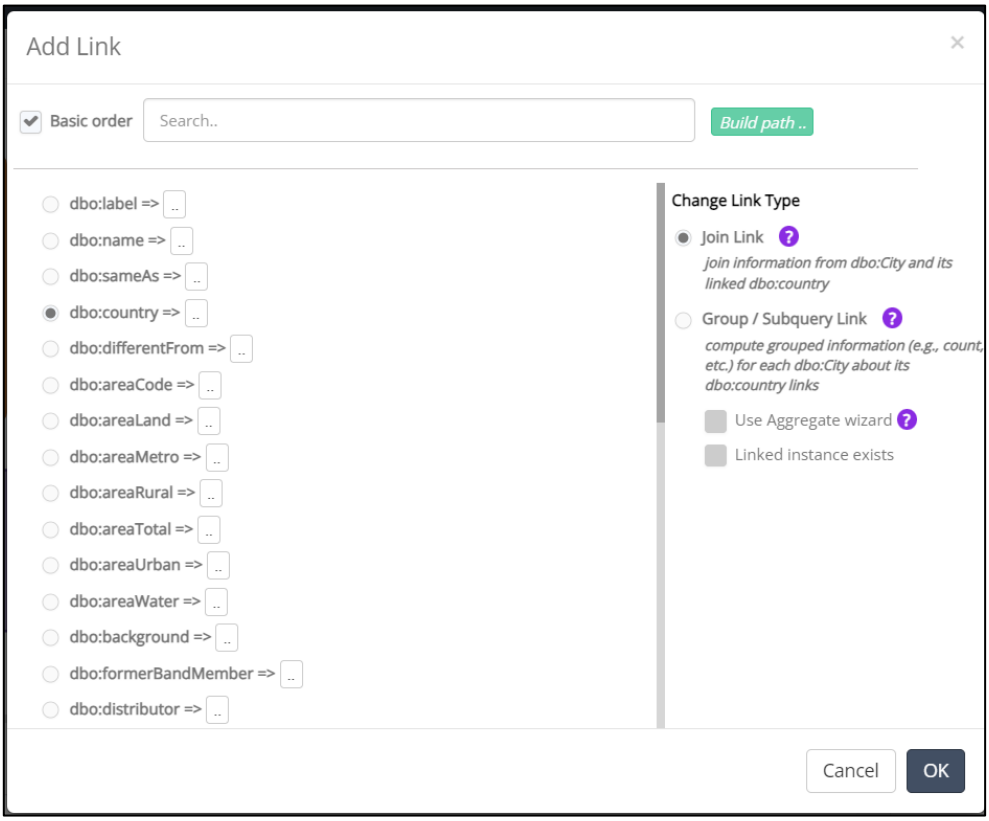
Vispirms tiek izveidota City klase (skat. 3.12. att.). Izvēloties klasi, labajā sānu panelī izkrītošajā izvēlnē tiek piedāvāti dažādi klašu nosaukumi laukā "Name".



3.13. att. **Pilsētu piemēra lauka izvēle**

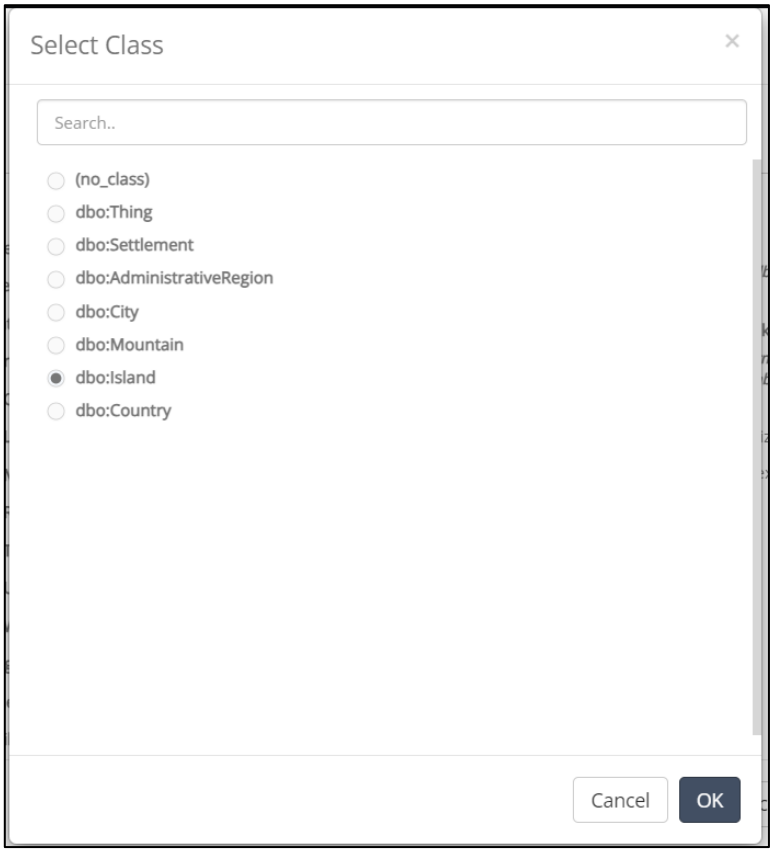
Spiežot uz City klases ar labo klikšķi un izvēloties "Add Field …" parādās 3.13. attēlā redzamais saraksts. Tajā tiek piedāvāts saraksts ar City klases iespējamiem laukiem, kas ir iegūts no ielasītā SHACL faila. Šajā gadījumā tiek izvēlēts "(select this)" lauks, lai tiktu atlasīts pilsētas IRI kods. Netiek izvēlēts dbo:name lauks, jo tas saturēs tukšas vērtības. Tas ir skaidrojams ar to, ka daļa no RDF datu aprakstošajiem trijniekiem atrodas citās nosaukumu telpās. Ielasītajā SHACL failā bija pieejami tikai dbo (https://dbpedia.org/ontology) nosaukumu telpa, bet daļa no aprakstošajiem laukiem var atrasties arī citās vārdu telpās. Lai ViziQuer priekšā rādīšanas funkcionalitāte šajā gadījumā būtu pilnīgāka, būtu jāiegūst un jāielasa arī SHACL faili par citām

vārdu telpām. Kā alternatīva tam ir manuāli ierakstīt lauku nosaukumus bez priekšā-teikšanas funkcionalitātes, piemēram, rdfs:label.



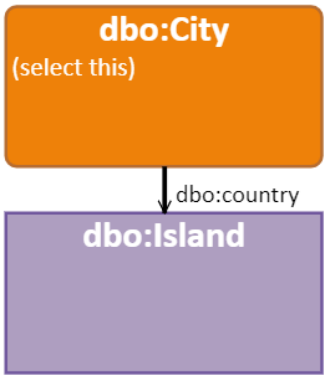
3.14. att. **Pilsētu piemēra saites izvēle**

Nākamajā solī pie City klasei tiek veidota saite ar klasi Island. Lai to veiktu, uz City klases tiek veikts labais klikšķis un tiek izvēlēta opcija "Add Link …". 3.14. attēlā ir redzams, kādi saišu veidi ir pieejami City klasei. Šajā gadījumā tiek izvēlēta saite "dbo:country" un nospiesta poga "..", lai izvēlētos, kādas klases atlasīt otrā galā saitei.



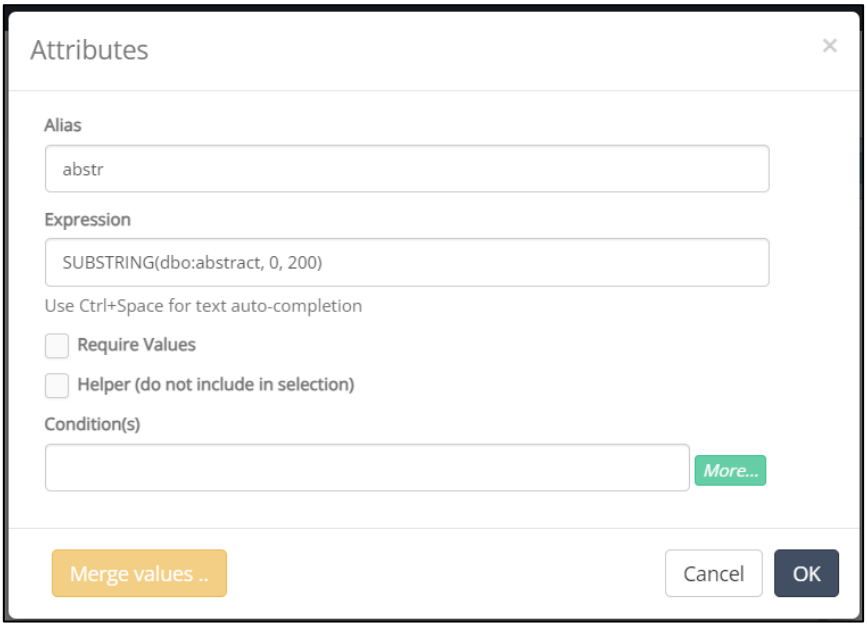
3.15. att. **Pilsētu piemēra saites mērķa klases izvēle**

3.15. attēlā ir redzams, ka dbo:country saitei tiek izvēlēta mērķa klase dbo:Island un pēc tam tiek spiests "OK".



3.16. att. **Pilsētu piemēra vizuālais vaicājums ar pilsētas lauku**

Pēc iepriekš veiktajām operācijām ViziQuer vizuālajā rīkā ir pieejama 3.16. attēlā redzamā diagramma.

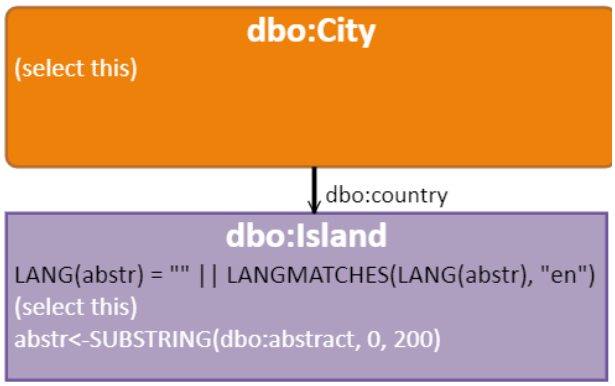


3.17. att. **Pilsētu piemēra kopsavilkuma lauka izvēle**

dbo:Island objektam caur labo sānu paneli sadaļā Attributes tiek pievienots jauns atribūts, kurš saturēs salas aprakstu (skat. 3.17. att.). Izteiksme satur tekstu, kurš no lauka dbo:abstract paņem pirmos 200 simbolus. Šim laukam tiek piešķirts aizstājvārds abstr un tālāk tiek spiests "OK".

3.18. att. **Pilsētu piemēra saites izvēle**

dbo:Island klasei tiek arī pievienots atlases nosacījums, kurā tiek atlasīti tie apraksti, kuri ir tukši vai arī kuri ir uzrakstīti angļu valodā (skat. 3.18. att.). Tas tiek darīts tādēļ, ka vienai salai var būt vairāki apraksti, bet interesē tieši angliskais apraksts, ja tāds ir pieejams.

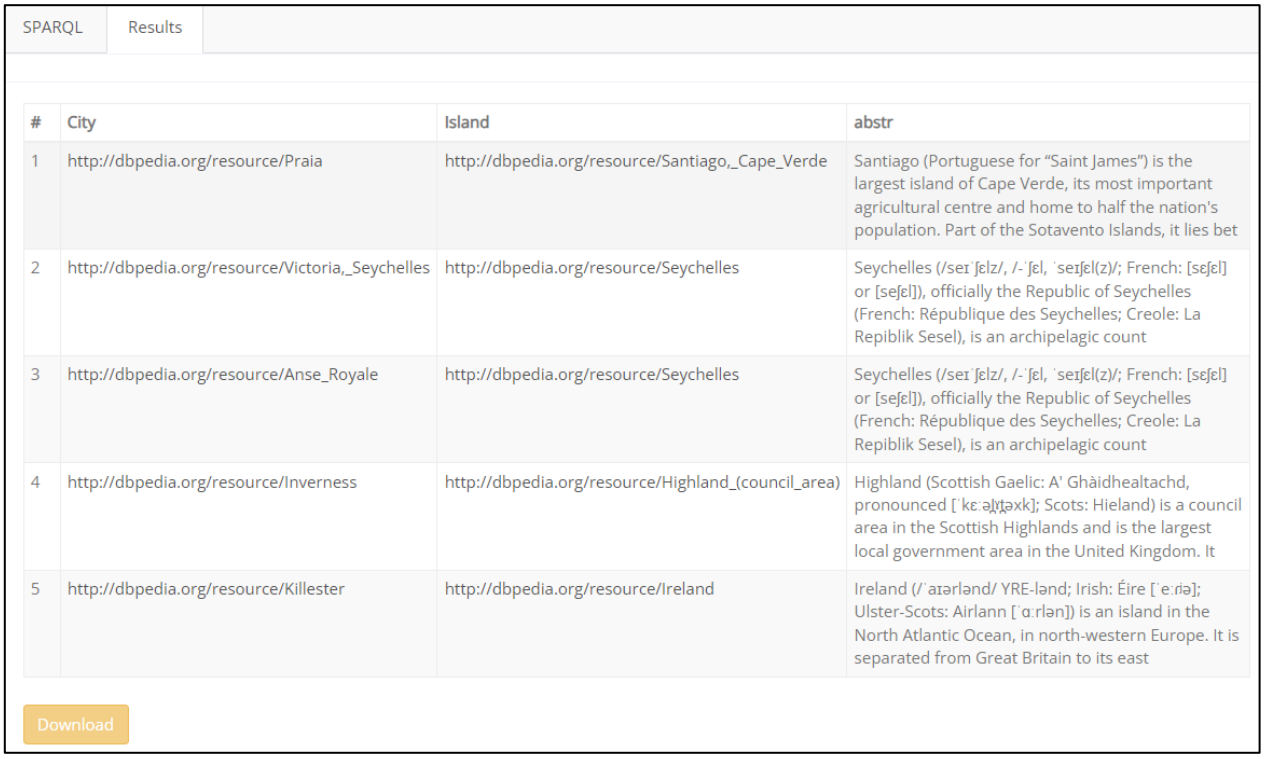


3.19. att. **Pilsētu piemēra rezultējošais vizuālais vaicājums**

Pēc iepriekš veiktajām operācijām un pēc "(select this)" lauka pievienošanas ViziQuer rīkā ir redzama 3.19. attēlā redzamā diagramma.

3.20. att. **Pilsētu piemēra SPARQL vaicājums**

Uz dbo:City veicot labo klikšķi un izvēloties "Generate query SPARQL" tiek uzģenerēts 3.20. attēlā redzamais vaicājums, kurš pēc tam tiek izpildīts.



3.21. att. **Pilsētu piemēra vaicājuma rezultāti**

Pēc vaicājuma izpildes "Results" cilnē tiek attēloti 5 rezultātu ieraksti (skat. 3.21. att.). Gluži kā definēts vizuālajā vaicājumā, tas satur pilsētas IRI, salas IRI un salas apraksta pirmos 200 simbolus.

REZULTĀTI

Darba ietvaros teorētiskajā daļā tika apskatīts semantiskā tīmekļa jēdziens un ar to saistītās tehnoloģijas, iekļaujot RDF datu modeli, RDF datu vaicājumu valodu SPARQL, RDF datu aprakstošo valodu SHACL, kā arī citas tehnoloģijas.

Lai apzinātu SPARQL vizualizācijas rīku iespējas, tika izpētīti un praktiski izmēģināti 4 SPARQL vaicājumu veidošanas rīki ar vizuālo pieeju, tai skaitā ViziQuer rīks.

Darba praktiskajā daļā tika izstrādāts shacl-2-viziquer rīks, kurš spēj no SHACL specifikācijā rakstīta faila nolasīt RDF datu avota metadatus un ierakstīt to norādītajā ViziQuer datubāzē tādā veidā, ka ViziQuer rīkā šim datu avotam ir pieejama priekšā-teikšanas funkcionalitāte, kas atvieglo veidot vizuālos vaicājumus. Rīka izstrādē tika ņemta vērā SHACL specifikācijas apakškopa ar mērķi panākt, ka rīks spēj ielasīt reālus RDF datu avotu aprakstošus SHACL failus. Šī darba ietvaros tika sekmīgi realizēta tieši DBpedia RDF datu avota SHACL faila nolasīšana un ierakstīšana ViziQuer datubāzē. Fails tika iegūts no publiski veikta pētījuma, kura ietvaros ir pieejami SHACL faili arī citiem RDF datu avotiem. Lai gūtu izpratni par shacl-2-viziquer vietu ViziQuer rīku kopumā, tika apskatīti rīku darbības principi, kā arī izveidotas datu plūsmu diagrammas, kas attēlo esošo ViziQuer rīku kopumu, kā arī ViziQuer rīku kopumu variantā, ja tiek lietots shacl-2-viziquer kā OBIS-SchemaExtractor rīka alternatīva.

shacl-2-viziquer rīka izstrādes gaitā tika izpētīta un apkopota SHACL failu pieejamība RDF avotiem, kā arī tika izpētīti, praktiski izmēģinātas un salīdzinātas vairākas JavaScript valodā rakstītas SHACL bibliotēkas un viena no tām tika arī pielietota shacl-2-viziquer izstrādē.

Izstrādātais rīks tika publicēts GitHub vietnē un ir pieejams apskatei caur saiti https://github.com/BrownEagle22/shacl-2-viziquer.

SECINĀJUMI

Pētot dažādus rīkus vizuālu SPARQL vaicājumu veidošanā, var secināt, ka ViziQuer piedāvā salīdzinoši plašu funkcionalitāti attiecībā uz diagrammu, to lauku un attiecību definēšanu. ViziQuer piedāvātie saišu tipi, lauku tipi, diagrammu tipi funkcionalitātes ziņā pārsvarā pārsniedz visus pārējos apskatītos rīkus. Var arī secināt, ka priekšā-teikšanas funkcionalitāte ir ļoti būtiska, jo tā bija pieejama visos apskatītajos rīkos. Ar tās palīdzību, lietotājam ir daudz vieglāk meklēt un atrast vēlamos objektus un to laukus. Pievēršoties uzģenerētajai SPARQL struktūrai, ViziQuer izcēlās ar iespējām, cik sarežģītu SPARQL vaicājumu ir iespējams izstrādāt izmantojot diagrammas, kā arī ar to optimizēšanas un atkļūdošanas iespējām, piemēram, diagrammās iespējams izmantot SPARQL vaicājuma mainīgos, lai varētu izsekot līdzi tam, kā savā starpā tiek kartēti diagrammas un faktiskie SPARQL vaicājuma mainīgie. Saistībā ar SHACL valodas izmantošanu, salīdzināmajos rīkos bija ļoti ierobežota valodas pielietošana. No apskatītajiem rīkiem tikai Sparnatural nodrošināju daļēju atbalsta SHACL valodas apstrādei priekš vizualizācijas. Lai to realizētu, tika definēta SHACL valodas apakškopa, kuru spētu interpretēt Sparnatural rīks. Šajā ziņā ViziQuer rīkam ir iespēja izvirzīties starp konkurentiem.

Analizējot, kā SHACL valodu var iestrādāt ViziQuer rīkā, tika apskatīta rīka datubāzes shēma, kura ir jāaizpilda ar metadatiem, kā arī tika pārskatīts, kā tas šobrīd tiek darīts priekšā-teikšanas funkcionalitātes nodrošināšanai. No tā var secināt, ka pašreizējais shēmas izgūšanas process var būt laikietilpīgs, it īpaši uz lieliem zināšanu grafiem. Izstrādāto shacl-2-viziquer rīku lietojot uz DBpedia SHACL failu, tas tiek dažu sekunžu laikā apstrādāts un ielasīts datubāzē, kas ir salīdzinoši labs rādītājs, kuru nākotnē ir iespēja sīkāk pētīt. shacl-2-viziquer salīdzinājumā ar OBIS-SchemaExtractor ir ātrdarbības potenciāls, jo, tā vietā, lai veiktu vairākus SPARQL vaicājumus caur tīmekli, tiek nolasīts tikai viens SHACL fails.

SHACL failu pieejamības pētīšanas gaitā tika konstatēts, ka apskatītie RDF datu avoti vēl nepiedāvā SHACL failus, bet tos ir iespējams iegūt ar citu pētījumu starpniecību. Tas sniedz iespēju lietot šos failus ViziQuer vajadzībām, bet, ņemot vērā šos apstākļus, failu kvalitāte un aktualitāte ir vēl uzlabojams aspekts.

Pēc shacl-2-viziquer lietošanas var secināt, ka rīks spēj nolasīt publiski pieejamu SHACL failu un ievadīt tā informāciju ViziQuer datubāzē, nodrošinot priekšā-teikšanas un citas saistītās funkcijas, tādā veidā papildinot iespējas, kā lietotāji var piekļūt RDF avotu metadatiem ērtākai lietošanai.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] "What Is the Semantic Web?," *Ontotext*, [Tiešsaiste]. Pieejams: https://www.ontotext.com/knowledgehub/fundamentals/what-is-the-semantic-web/#:~:text=The%20Semantic%20Web%20is%20a,and%20data%20on%20the%20Web.

[2] P. Hitzler, M. Krötzsch un S. Rudolph, *Foundations of Semantic Web technologies*, Florida, USA: CRC Press, 2010.

[3] D. Brickley un R. V. Guha, «RDF Schema 1.1,» *W3C*, 25.02.2014. [Tiešsaiste]. Pieejams: https://www.w3.org/TR/rdf-schema.

[4] O. W. Group, "OWL," *W3C*, 11.12.2012. [Tiešsaiste]. Pieejams: https://www.w3.org/OWL/.

[5] W. O. W. Group, "OWL 2 Web Ontology Language," *W3C*, 2012. [Tiešsaiste]. Pieejams: https://www.w3.org/TR/owl2-overview/.

[6] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider un S. Rudolph, "OWL 2 Web Ontology Language," *W3C*, 2012. [Tiešsaiste]. Pieejams: https://www.w3.org/TR/2012/REC-owl2-primer-20121211/#What_is_OWL_2.3F.

[7] "What is SPARQL?," *Ontotext*, [Tiešsaiste]. Pieejams: https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/#:~:text=SPARQL%2C%20pronounced%20'sparkle'%2C,can%20be%20mapped%20to%20RDF.

[8] L. Feigenbaum, "SPARQL By Example:," *Cambridge Semantics*, [Tiešsaiste]. Pieejams: https://www.iro.umontreal.ca/~lapalme/ift6281/sparql-1_1-cheat-sheet.pdf.

[9] "What is SHACL?," *Oxford Semantic Technologies*, [Tiešsaiste]. Pieejams: https://www.oxfordsemantic.tech/faqs/what-is-shacl.

[10] S. Retnadhas, E. Jimenez-Ruiz un M. Giese, "OptiqueVQS," 15.01.2021. [Tiešsaiste]. Pieejams: https://gitlab.com/ernesto.jimenez.ruiz/OptiqueVQS.

[11] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jimenez-Ruiz, M. Giese, M. G. Skjæveland, D. Hovland, R. Schlatte, S. Brandt, H. Lie un I. Horrocks, "OptiqueVQS: a Visual Query System over Ontologies for Industry," [Tiešsaiste]. Pieejams: https://semantic-web-journal.net/system/files/swj1568.pdf.

[12] H. Vargas, C. Buil-Aranda, A. Hogan un C. López, "RDF Explorer: A Visual SPARQL Query Builder," [Tiešsaiste]. Pieejams: https://aidanhogan.com/docs/rdf_explorer_sparql_interface.pdf.

[13] H. Varga, “RDFExplorer,” 07.10.2019. [Tiešsaiste]. Pieejams: https://github.com/hvarg/RDFExplorer/tree/master.

[14] “Sparnatural SPARQL query builder,” 26.01.2023. [Tiešsaiste]. Pieejams: https://github.com/sparna-git/Sparnatural.

[15] K. Čerāns, J. Ovčiņņikova, U. Bojārs, M. Grasmanis, L. Lāce un A. Romāne, “Schema-Backed Visual Queries over Europeana and other Linked Data Resources,” *Institute of Mathematics and Computer Science, University of Latvia*, 2021. [Tiešsaiste]. Pieejams: https://viziquer.lumii.lv/papers/2021_ESWC_Demo_Schema.pdf.

[16] K. Rabbani, M. Lissandrini un K. Hose, «Automatically Extracted SHACL Shapes for WikiData, DBpedia, YAGO-4, and LUBM & Associated Coverage Statistics,» *Zenodo*, 02.02.2023. [Tiešsaiste]. Pieejams: https://zenodo.org/records/7598613.

[17] «Data Shape Server (DSS),» *SysLab*, 22.01.2024. [Tiešsaiste]. Pieejams: https://github.com/LUMII-Syslab/data-shape-server.

[18] «ViziQuer,» *SysLab*, 18.03.2024. [Tiešsaiste]. Pieejams: https://github.com/LUMII-Syslab/viziquer.

[19] “Previous Releases,” *Node*, [Tiešsaiste]. Pieejams: https://nodejs.org/en/about/previous-releases.

[20] A. Cimmino, A. Fernández-Izquierdo un R. García-Castro, «DBpedia SHACL shapes,» *Zenodo*, 09.09.2020. [Tiešsaiste]. Pieejams: https://zenodo.org/records/4021024.

[21] J. OVČIŅŅIKOVA, A. ŠOSTAKS un K. ČERĀNS, “Visual Diagrammatic Queries in ViziQuer: Overview and Implementation,” *Institute of Mathematics and Computer Science, University of Latvia*, 2023. [Tiešsaiste]. Pieejams: https://www.bjmc.lu.lv/fileadmin/user_upload/lu_portal/projekti/bjmc/Contents/11_2_07_Ovcinnikova.pdf.

[22] R. W. Group, “RDF,” *W3C*, 25.02.2014. [Tiešsaiste]. Pieejams: https://www.w3.org/RDF/.

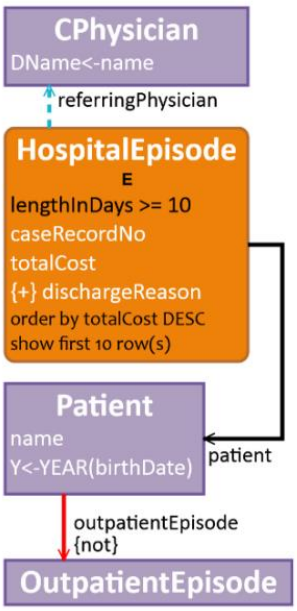
[23] B. McBride, «RDF Vocabulary Description Language 1.0: RDF Schema,» *W3C*, 2004. [Tiešsaiste]. Pieejams: https://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[24] H. Knublauch un D. Kontokostas, “Shapes Constraint Language (SHACL),” *W3C*, 20.07.2017. [Tiešsaiste]. Pieejams: https://www.w3.org/TR/shacl.

PIELIKUMI

1. pielikums

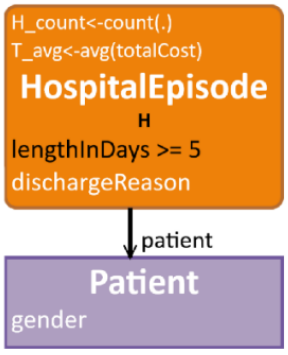
ViziQuer diagrammas piemērs [21]




```
PREFIX : <http://lumii.lv/ontologies/2016/mini-bkus-en#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?caseRecordNo ?totalCost ?dischargeReason
                              ?DName ?name ?Y WHERE{
  ?E a :HospitalEpisode.
  ?E :patient ?Patient.
  ?Patient a :Patient.
  OPTIONAL{?E :caseRecordNo ?caseRecordNo.}
  OPTIONAL{?E :totalCost ?totalCost.}
  ?E :dischargeReason ?dischargeReason.
  ?E :lengthInDays ?lengthInDays.
  OPTIONAL{?Patient :name ?name.}
  OPTIONAL{?Patient :birthDate ?birthDate.}
  OPTIONAL{ ?E :referringPhysician ?CPhysician.
    ?CPhysician a :CPhysician.
    OPTIONAL{?CPhysician :name ?DName.} }
  FILTER NOT EXISTS{
    ?Patient :outpatientEpisode ?OutpatientEpisode.
    ?OutpatientEpisode a :OutpatientEpisode. }
  BIND(YEAR(xsd:dateTime(?birthDate)) AS ?Y)
  FILTER(?lengthInDays >= 10)
} ORDER BY DESC(?totalCost) LIMIT 10
```

2. pielikums

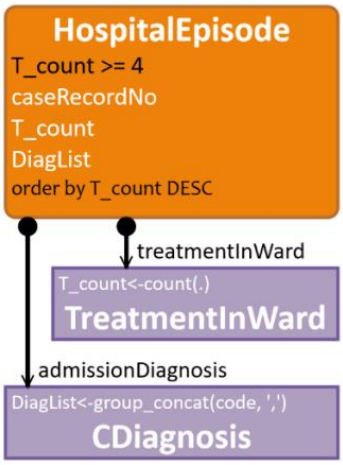
ViziQuer diagrammas piemērs ar grupēšanu un agregāciju [21]




```
PREFIX : <http://lumii.lv/ontologies/2016/mini-bkus-en#>
SELECT ?dischargeReason ?gender (COUNT(?H) AS ?H_count)
                    (AVG(?totalCost) AS ?T_avg) WHERE{
  ?H a :HospitalEpisode.
  ?H :patient ?Patient.
  ?Patient a :Patient.
  OPTIONAL{?H :dischargeReason ?dischargeReason.}
  OPTIONAL{?H :totalCost ?totalCost.}
  ?H :lengthInDays ?lengthInDays.
  OPTIONAL{?Patient :gender ?gender.}
  FILTER(?lengthInDays >= 5)
} GROUP BY ?dischargeReason ?gender
```

3. pielikums

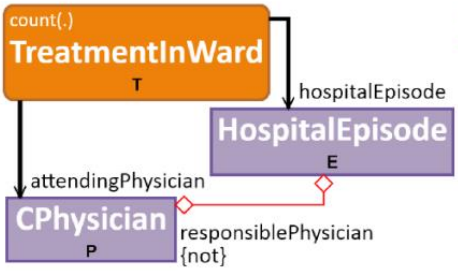
ViziQuer diagrammas piemērs ar apakšvaicājumiem [21]




```
PREFIX : <http://lumii.lv/ontologies/2016/mini-bkus-en#>
SELECT ?caseRecordNo ?T_count ?DiagList WHERE{
  ?HospitalEpisode a :HospitalEpisode.
  {SELECT ?HospitalEpisode
        (COUNT(?TreatmentInWard) AS ?T_count) WHERE{
    ?HospitalEpisode :treatmentInWard ?TreatmentInWard.
    ?TreatmentInWard a :TreatmentInWard.}
    GROUP BY ?HospitalEpisode}
  {SELECT ?HospitalEpisode
        (GROUP_CONCAT(?code; SEPARATOR=',') AS ?DiagList) WHERE{
    ?HospitalEpisode :admissionDiagnosis ?CDiagnosis.
    ?CDiagnosis a :CDiagnosis.
    OPTIONAL{?CDiagnosis :code ?code.}}
    GROUP BY ?HospitalEpisode}
  OPTIONAL{?HospitalEpisode :caseRecordNo ?caseRecordNo.}
  FILTER(?T_count >= 4)
} ORDER BY DESC(?T_count)
```

4. pielikums

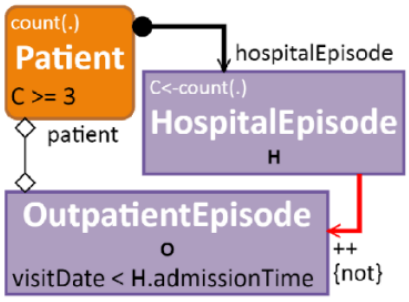
ViziQuer diagrammas piemērs ar atsauču saiti [21]




```
PREFIX : <http://lumii.lv/ontologies/2016/mini-bkus-en#>
SELECT (COUNT(?T) AS ?T_COUNT) WHERE{
  ?T a :TreatmentInWard.
  ?T :hospitalEpisode ?E.
  ?E a :HospitalEpisode.
  ?T :attendingPhysician ?P.
  ?P a :CPhysician.
  FILTER NOT EXISTS{?E :responsiblePhysician ?P.}
}
```

5. pielikums

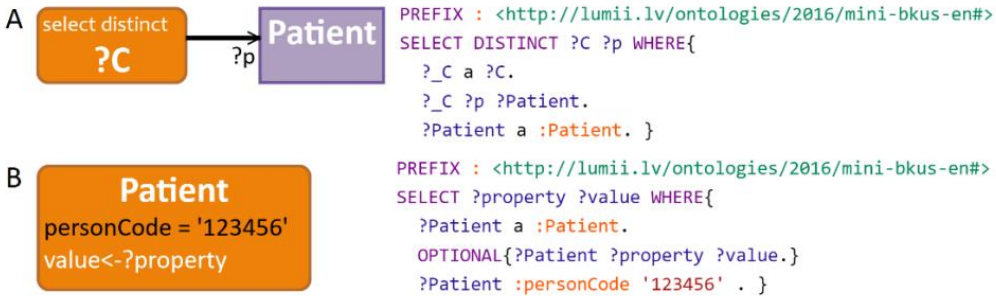
ViziQuer diagrammas piemērs ar vaicājumu struktūras paplašinājumiem [21]




```
PREFIX : <http://lumii.lv/ontologies/2016/mini-bkus-en#>
SELECT (COUNT(?Patient) AS ?Patient_COUNT) WHERE{
  ?Patient a :Patient.
  {SELECT ?Patient (COUNT(?H) AS ?C) WHERE{
    ?Patient :hospitalEpisode ?H.
    ?H a :HospitalEpisode.
    FILTER NOT EXISTS{
      ?O a :OutpatientEpisode.
      ?O :patient ?Patient.
      ?H :admissionTime ?admissionTime_H.
      ?O :visitDate ?visitDate.
      FILTER(?visitDate < ?admissionTime_H) }}
    GROUP BY ?Patient }
  FILTER(?C >= 3) }
```

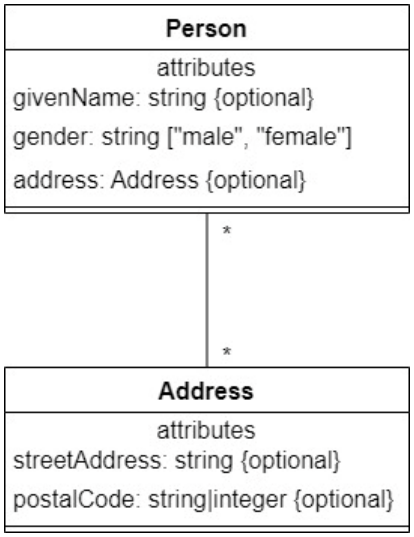
6. pielikums

ViziQuer diagrammas piemērs ar izpētīšanas vaicājumiem [21]



7. pielikums

UML modelis



8. pielikums

SHACL vienkāršots piemērs priekš konvertācijas uz ViziQuer datubāzes tabulām

```
@prefix dash: <http://datashapes.org/dash#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

schema:PersonShape
    a sh:NodeShape ;
    sh:targetClass schema:Person ;
    sh:property [
        sh:path schema:givenName ;
        sh:datatype xsd:string ;
    ] ;
```

```
    sh:property [
        sh:path schema:gender ;
        sh:in ( "female" "male" ) ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path schema:address ;
        sh:node schema:AddressShape ;
    ] .

schema:AddressShape
    a sh:NodeShape ;
    sh:closed true ;
    sh:property [
        sh:path schema:streetAddress ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path schema:postalCode ;
        sh:or ( [ sh:datatype xsd:string ] [ sh:datatype xsd:integer ] ) ;
    ] .
```