LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

VIZUĀLU SEMANTISKU VAICĀJUMU IZPILDE PĀR RELĀCIJU DATUBĀZĒM

MAĢISTRA DARBS

Autors: Ralfs Sedlenieks

Studenta apliecības Nr.: rs19064

Darba vadītājs: Dr.sc.comp. Kārlis Čerāns

RĪGA 2021

ANOTĀCIJA

Maģistra darba mērķis ir padziļināti izpētīt esošās metodes, rīkus un optimizācijas semantisko vaicājumu tulkošanai uz SQL un izpildei pār relāciju datubāzēm, un, veicot praktiskus eksperimentus, atrast galvenos cēloņus tam, kāpēc rīku ģenerētie vaicājumi nav optimāli.

Vizuālu semantiska līmeņa (ontoloģijas izmantojošu) vaicājumu pār datiem veidošana var ļaut gala lietotājam tieši piekļūt ontoloģiju formā organizētiem datiem. Uz ontoloģijām balstīta piekļuve datiem piedāvā iespējas pārveidot semantiskus SPARQL vaicājumus par SQL vaicājumiem, kas var tikt izpildīti pār relāciju datubāzēm. Tomēr pieejamās tehnoloģijas, piemēram, rīks Ontop, atbalsta tikai ierobežotu vaicājumu klašu translācijas iespējas, kā arī gadījumos, kad translācija ir iespējama, iegūtais rezultāts ir tālu no optimāla.

Maģistra darbā tiek apskatītas tehnoloģijas, kuras ir semantiskā tīmekļa pamatā, izpētītas tehnoloģijas un rīki, ar kuriem iespējams realizēt ontoloģijās balstītu pieeju datiem, apkopotas šobrīd pieejamās vaicājumu tulkošanas metodes un to ierobežojumi, aprakstīta eksperimentos izmantotā infrastruktūra un tās uzstādīšana, kā arī apskatīti iegūtie vaicājumi un to problēmas.

Atslēgvārdi: Ontoloģijā bāzēta datu piekļuve, vizuāli semantiski vaicājumi, relāciju datubāzes, R2RML, SPARQL, RDF

ABSTRACT

EXECUTION OF VISUAL SEMANTIC QUERIES OVER RELATIONAL DATABASES

The purpose of this master's thesis is to research existing methods, tools, and optimizations for translating SPARQL queries to SQL and to find the main causes of translation issues through practical experimentation.

Creating visual semantic queries over data can provide end-users direct access to data that is organized as an ontology. Ontology-based data access offers the possibility to transform semantic SPARQL queries to SQL queries, which can then be executed on a relational database. However, existing technologies, e.g., a tool called Ontop, only support the translation of a limited amount of query types and even when such a translation is possible, the resulting queries are often poorly optimized.

This master's thesis describes technologies that form the basis of the Semantic Web, analyzes technologies and tools that can be used in ontology-based data access, compiles currently available methods for query translation, as well as their limitations, describes the infrastructure setup used in the experiments and reviews the generated queries and problems within these queries.

Keywords: Ontology based data access, visual semantic queries, relational databases, R2RML, SPARQL, RDF

AUTOREFERĀTS

Maģistra darbā tika apkopotas dažādas pieejas, kas tiek izmantotas šobrīd pieejamos rīkos SPARQL vaicājumu tulkošanai uz SQL, kā arī izveidota infrastruktūra, kas ļāva novērtēt konkrētu rīku tulkoto vaicājumu kvalitāti. Šī novērtējuma rezultātā tika atrastas metodes, kā panākt, lai tulkotie vaicājumi būtu optimālāki, kā arī atrasti iespējamie uzlabojumi, kas ļautu iegūt vēl optimālākus vaicājumus.

Darba izstrādē izmantota informācija no 62 avotiem, no kuriem vairums ir zinātniski raksti, bet mazāka daļa ir standartu specifikācijas vai arī dažādi interneta avoti.

Darba pirmās divas nodaļas satur vispārīgu izmantoto tehnoloģiju aprakstu, kas veido pamatu tālākajās nodaļās apskatītajām tēmām un eksperimentiem. Lai gan pirmajās divās nodaļās iekļautie tehnoloģiju apraksti nav pārāk detalizēti, šo pamata tehnoloģiju apkopošana nav darba mērķis un tālākās nodaļās apskatītās vaicājumu tulkošanas pieejas, optimizācijas, kā arī infrastruktūras uzstādīšana un iegūtie rezultāti ir aprakstīti detalizētāk. Darba praktiskā daļa iekļāva vaicājumu izpildei nepieciešamās infrastruktūras uzstādīšanu un uzlabošanu, darbā izmantotās ontoloģijas un R2RML attēlojuma labošanu un pilnveidošanu, kā arī vaicājumu tulkojumu optimizāciju un dažādo vaicājumu variantu sarežģītības salīdzināšanu.

Darba teksts ir ticis pārlasīts, lai atrastu iespējamas gramatikas vai noformējuma kļūdas. Darba izstrāde un noformēšana tika veikta saskaņā ar Latvijas Universitātes Datorikas fakultātes veidoto dokumentu "Maģistra darba izstrādes un aizstāvēšanas metodiskie norādījumi", kā arī ir izpildītas minētā dokumenta 6. pielikumā atrodamās prasības. Nozares terminu pārbaudei un tulkošanai tika izmantota Latvijas Nacionālā terminoloģijas portāla vietne[1]. Pareizrakstības pārbaudei tika izmantoti teksta redaktorā "Word" pieejamie rīki.

Visas idejas, formulējumi un attēli, kas iegūti no literatūras avotiem, ir korekti atzīmēti ar literatūras atsaucēm. Attēli un tabulas bez šādām atsaucēm ir paša autora veidoti.

[1] https://termini.gov.lv/

SATURS

APZĪMĒJUMU SARAKSTS

IRI (International Resource Identifier) – starptautisks resursa identifikators

OBDA (Ontology-Based Data Access) – ontoloģijā balstīta datu pieeja

OWL (Web Ontology Language) – tīmekļa ontoloģijas valoda

RDBMS (Relational database management system) – relāciju datubāžu pārvaldības sistēma

RDF (Resource Description Framework) – resursu aprakstīšanas standarts

RDFS (RDF Schema) – RDF shēma

R2RML (RDB to RDF Mapping Language) – valoda attēlojumu definēšanai starp relāciju datubāzēm un RDF datu kopām

RML (RDF Mapping Language) – RDF attēlojumu valoda

SPARQL (SPARQL Protocol and RDF Query Language) - SPARQL protokols un RDF vaicājumu valoda

SQL (Structured Query Language) – strukturēta vaicājumvaloda

VKG (Virtual knowledge graph) – virtuāls zināšanu grafs

IEVADS

Mūsdienās izmantotās lietotnes, informācijas sistēmas un dažādu citu veidu programmatūra nav iedomājama bez datubāzēm, kurās glabāt nepārtraukti pieaugošos datu apjomus. Jau vairākas desmitgades šim mērķim tiek izmantotas relāciju datubāzes, tomēr šādas datubāzes nenodrošina semantisku vaicājumu izpildi pār datiem, lai iegūtu kontekstuālu informāciju vai izsecinātu informāciju, kas nav tiešā veidā glabāta datubāzē. Šo problēmu var risināt ar ontoloģijā balstītu datu pieeju. Šo pieeju papildinot ar rīku ViziQuer, ir iespējams veidot vizuālus semantiskus vaicājumus, kas ļauj šādu informāciju iegūt arī gala lietotājiem bez konkrētām tehniskām prasmēm.

Lai realizētu ontoloģijā balstītu datu pieeju relāciju datubāžu datiem, ir nepieciešams pārtulkot SPARQL vaicājumus par SQL vaicājumiem, tomēr eksistējošie rīki, piemēram Ontop, atbalsta tikai atsevišķu vaicājumu klašu tulkošanu, kā arī bieži vien ģenerē neoptimālus SQL vaicājumus. Optimizējot šos vaicājumus un paplašinot vaicājumu tulkošanas iespējas, ir iespējams uzlabot OBDA pieejas lietojamību un samazināt šīs pieejas radīto avota datubāzes noslodzi.

Maģistra darba mērķis ir padziļināti izpētīt esošās metodes, rīkus un optimizācijas semantisko vaicājumu tulkošanai uz SQL un izpildei pār relāciju datubāzēm, un, analizējot iegūtos vaicājumu tulkojumus, atrast iespējamus izmantoto rīku uzlabojumus.

Lai sasniegtu darba mērķi, jāizpilda šādi uzdevumi:

- Jāapskata semantiskā tīmekļa galvenās tehnoloģijas un standarti
- Padziļināti jāiepazīstas ar ontoloģijās balstītas datu pieejas tehnoloģijām un rīkiem
- Jāatrod un jāapraksta esošās metodes un optimizācijas semantisko vaicājumu tulkošanā uz SQL
- Jāizveido infrastruktūra vizuālo vaicājumu izpildei pār relāciju datubāzi
- Jāapskata izmantotās infrastruktūras tulkotie vaicājumi un to optimizācijas, lai atrastu iespējamus uzlabojumus

Maģistra darbā tiek apskatītas tehnoloģijas, kuras ir semantiskā tīmekļa pamatā, izpētītas tehnoloģijas un rīki, ar kuriem iespējams realizēt ontoloģijās balstītu pieeju datiem, apkopotas šobrīd pieejamās vaicājumu tulkošanas metodes un to ierobežojumi, aprakstīta eksperimentos izmantotā infrastruktūra un tās uzstādīšana, kā arī apskatīti iegūtie vaicājumi un to problēmas.

1. Semantiskā tīmekļa tehnoloģijas

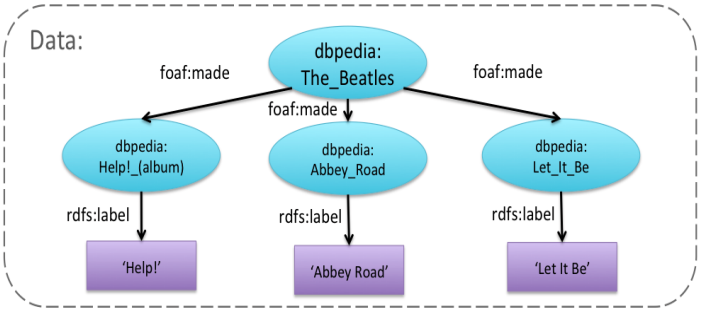
Šajā nodaļā tiks apskatītas semantiskā tīmekļa galvenās tehnoloģijas, kuras ir arī ontoloģijās balstītas datu pieejas (OBDA) pamatā.

1.1. RDF

Resursu aprakstīšanas satvars (RDF) ir datu modelis, kas ir paredzēts dažādu resursu (dokumentu, cilvēku, jēdzienu utt.) aprakstīšanai mašīnlasāmā formātā. [1] RDF modeli izmanto vairākiem mērķiem, piemēram,

- lai pievienotu mašīnlasāmu informāciju tīmekļa lapām, tādējādi uzlabojot lapas rezultātus tīmekļa meklētājprogrammās,
- lai papildinātu kādu datu kopu ar citām trešo pušu datu kopām,
- lai nodrošinātu standartizētu veidu datu apmaiņai starp datubāzēm.

RDF datu modelī visi apgalvojumi par resursiem tiek noformēti kā trijnieki (triples). Katrā trijniekā ir subjekts, predikāts un objekts. Subjekts un objekts norāda resursus, kuri tiek savstarpēji saistīti, bet predikāts precizē attiecību, kura pastāv starp šiem resursiem. Tā kā jebkurš resurss var būt gan subjekta, gan objekta pozīcijā, RDF datu kopās ir iespējams atrast saites starp dažādiem trijniekiem, kas ļauj atlasīt visu informāciju, kas saistīta ar konkrētu objektu. [1] Saites starp RDF datu kopas trijniekiem veido virzītu grafu, kura virsotnes ir resursi, bet šķautnes – attiecības starp resursiem. [2] Attēlā 1.1 ir redzams šāda grafa piemērs, kurā attēloti gan resursi (zilās virsotnes), gan literāļi (violetās virsotnes), gan arī dažādas attiecības starp šiem resursiem. RDF trijnieku pozīcijās var būt trīs dažādu tipu vērtības – IRI, literāļi un tukšie mezgli (blank nodes). IRI ļauj viennozīmīgi identificēt dažādus resursus un šāda tipa vērtības var izmantot visās trijnieka pozīcijās, jo ar šiem identifikatoriem var identificēt gan konkrētus objektus vai dokumentus, gan arī dažādas attiecības starp tiem. Literāļi ir vienkāršas vērtības, kurām nav savu identifikatoru un kuras izmanto, lai aprakstītu resursu īpašības. RDF trijniekos tās var izmantot tikai objekta pozīcijā. Literāļiem parasti ir nepieciešams norādīt arī datu tipu, lai atvieglotu RDF datu kopas parsēšanu. Tukšie mezgli ļauj aprakstīt "anonīmus" resursus, kuriem nav savu identifikatoru, un šāda tipa vērtības var atrasties trijnieka subjekta un objekta pozīcijās. RDF modelim piemītošās īpašības ļauj apvienot datus no dažādiem avotiem, pat ja



Att. 1.1 **RDF grafa vizuāls piemērs** *[12]*

šo avotu shēmas ir atšķirīgas. [2] Ja relāciju datubāžu avotu apvienošanai parasti nepieciešami papildu ETL procesi, tad RDF datu avotu apvienošanu atvieglo fakts, ka ikvienam resursam ir unikāls identifikators un ka šie identifikatori ir unikāli arī starp avotiem. [3]

RDF standartā aprakstītais modelis ir abstrakts un tam eksistē vairākas implementācijas, katrai no tām ir sava konkrētā sintakse. [1] Viena no RDF implementācijām ir Turtle, kas patiesībā ir vairāku līdzīgu sintakšu grupa. Šajā grupā ietilpst N-Triples, Turtle, TriG un N-Quads sintakses. N-Triples ir vienkāršākā no šīm sintaksēm un nodrošina visas pamata funkcijas, lai aprakstītu RDF datu kopas. Turtle sintakse papildina N-Triples, ieviešot dažādus sintaktiskus papildinājumus, piemēram, vārdtelpu prefiksus, sarakstus utt. TriG sintakse papildina Turtle, pievienojot iespēju definēt un nosaukt vairākus RDF grafus. N-Quads sintakse ir vienkāršs N-Triples paplašinājums, kurā RDF trijniekiem tiek pievienots vēl ceturtais elements, kas norāda, kuram grafam pieder konkrētais ieraksts. Gan N-Triples, gan N-Quads sintakses tipiski lieto, lai pārsūtītu lielas RDF datu kopas, kā arī tās apstrādātu ar līnijorientētiem teksta apstrādes rīkiem. Cita RDF konkrētā sintakse ir JSON-LD, kas ļauj RDF datu kopas aprakstīt, izmantojot JSON sintaksi. Šī sintakse ir paredzēta, lai atvieglotu RDF un saistīto datu integrāciju sistēmas, kuras jau lieto JSON formātu datu apmaiņai. [4] RDFa konkrētā sintakse ir paredzēta, lai iekļautu RDF datus HTML dokumentos. Viens no šīs sintakses pielietojumiem ir papildu informācijas pievienošana tīmekļa lapām, lai bagātinātu meklētājprogrammu dotos rezultātus šai lapai. [1] Lai iekļautu RDF datus XML dokumentos, var izmantot RDF/XML sintaksi.

Resursu identifikatorus, kuri identificē dažādas attiecības starp RDF grafa objektiem, mēdz grupēt "vārdnīcās". Šādas vārdnīcas piedāvātās attiecības iespējams izmantot RDF grafa šķautnēs, lai piešķirtu papildu nozīmi divu resursu attiecībai. [5] Viens no šādiem RDF paplašinājumiem ir RDF Schema (RDFS), kas ļauj pievienot papildu semantiku RDF datiem. RDFS ļauj grupēt resursus pa klasēm, kā arī norādīt semantiski bagātākas attiecības starp resursiem nekā pamata RDF modelī. Šie mehānismi ir līdzīgi klasēm objektorientētās programmēšanas valodās. RDFS vārdnīcā visas klases ir apakšklases rdfs:Resource klasei, bet visas īpašības (properties) ir "apakšīpašības" rdfs:property īpašībai. [6]

Kopumā ir iespējams secināt, ka RDF ir universāls modelis, kam eksistē vairākas konkrētas sintakses un ar kuru ir iespējams aprakstīt un uzglabāt semantiskus datus viegli integrējamā formātā, tomēr pamata RDF ir ierobežotas izteiksmes iespējas. Lai paplašinātu šīs iespējas, eksistē RDFS vārdnīca, kas definē vairākus semantiski bagātākus resursu attiecību veidus.

1.2. OWL

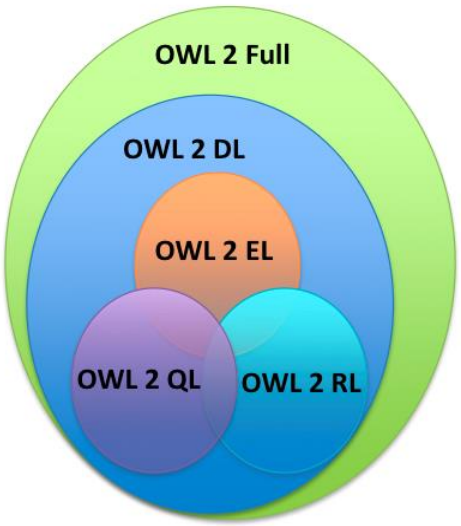
Tīmekļa ontoloģiju valoda (OWL) ir paredzēta, lai aprakstītu zināšanas par lietām/resursiem, kā arī par attiecībām starp šiem resursiem. OWL valodā veidotus dokumentus sauc par ontoloģijām. [7] Ontoloģijas satur precīzus apgalvojumus, kas apraksta ar konkrētās ontoloģijas tematu saistītus objektus vai jēdzienus, kā arī attiecības starp tiem. Šiem apgalvojumiem ir jābūt precīziem, lai nodrošinātu viennozīmību un panāktu, ka ontoloģiju apstrādājošās programmas strādātu paredzami. [8] Ar OWL veidotas ontoloģijas var apstrādāt ar dažādām loģiskās spriešanas programmām (t.s. secinātājiem jeb *reasoners*), lai pārbaudītu, ka izstrādātajā ontoloģijā nav zināšanu pretrunu, kā arī lai izsecinātu papildu informāciju, kas nav tiešā veidā norādīta. [7] OWL ontoloģijas balstās uz "atvērtās pasaules" pieņēmumu (open-world assumption), kas nozīmē, ka informācija, kas nav iekļauta ontoloģijā, vai arī kuru nav iespējams izsecināt no konkrētās ontoloģijas, tiek uzskatīta par patiesu. OWL valodā zināšanu attēlošanai izmanto trīs pamatjēdzienus: aksiomas, entītijas un izteiksmes. Aksiomas ir vienkārši apgalvojumi, kurus ontoloģijā pieņem par patiesiem. Ar šādiem apgalvojumiem gan ir par maz, lai spētu izveidot ontoloģijas, no kurām būtu iespējams arī izsecināt informāciju, kas tajās neparādās atklātā formā. OWL valodas semantika ļauj modelēt arī situācijas, kur viens apgalvojums seko no cita. Izmantojot secinātājus, ir iespējams atklāt šādas saites starp ontoloģijas aksiomām. OWL ontoloģiju apgalvojumi visbiežāk satur informāciju par konkrēta zināšanu apgabala objektiem un apraksta tos, izmantojot kategorijas vai norādot uz objekta attiecībām ar citiem objektiem. Par entītijām OWL kontekstā sauc objektus, kategorijas un attiecības, kas iekļauti ontoloģiju apgalvojumos. Šajā ziņā OWL valoda ir līdzīga RDF modelim, kurā arī visi objekti, to klases un dažādu veidu attiecības tika saukti par resursiem. OWL 2 standartā objektus sauc par indivīdiem, kategorijas par klasēm un attiecības par īpašībām. OWL entītijas ir iespējams apvienot izteiksmēs. [8]

Gan RDFS, gan OWL var izmantot, lai aprakstītu datus vai zināšanas atbilstoši RDF modelim, tomēr pastāv vairākas atšķirības, kas padara OWL par spēcīgāku apraksta rīku. Pirmkārt, OWL valodā ir plašāka vārdnīca, ko izmantot, lai definētu attiecības starp objektiem. Papildus RDFS predikātiem, OWL iekļauj arī kopu operāciju predikātus, piemēram, owl:unionOf, kas apraksta kopu apvienojumu, un predikātus, kas ļauj definēt no dažādiem datu avotiem iegūtu objektu ekvivalenci, kā arī ierobežot objekta īpašību iespējamās vērtības. Otrkārt, OWL valoda nosaka stingrāku ontoloģijas struktūru, ierobežojot predikātu un klašu izmantojumu, tā novēršot, piemēram, situācijas, kur viens objekts ir gan klase, gan instance. RDFS šādus ierobežojumus nenosaka. Šie ierobežojumi palīdz optimizēt secināšanas darbības pār OWL ontoloģijām. Visbeidzot, atšķirībā no RDFS, OWL valodā ir definētas anotācijas, kas

ļauj aprakstīt attiecības starp dažādām ontoloģijām, piemēram, norādot atpakaļsaderību (backwards compatibility) starp dažādām ontoloģijas versijām, vai arī importējot vienu ontoloģiju citā. [9]

Sākotnējā OWL valodas versija tika standartizēta 2004. gadā [10], bet mūsdienās plašāk izmantota OWL 2 versija. Jaunākajā OWL valodas versijā iekļautās izmaiņas tika radītas, balstoties uz reālu OWL lietotāju un rīku izstrādātāju pieredzēm. OWL 2 versijā ir pieejami atsevišķi sintaktiskie uzlabojumi, lai atvieglotu biežāk lietoto šablonu definēšanu, jaunas konstrukcijas, kas paplašina objektu īpašību iespējas, plašāks atbalsts datu tipu definēšanai, kā arī paplašinātas anotāciju iespējas. [11]

OWL 2 valodai eksistē vairāki apakšvarianti jeb profili, kas apraksta dažādas iespējamās OWL valodas realizācijas. Katrs no tiem ir pielāgots atšķirīgiem lietojuma scenārijiem. OWL 2 valodas profilu izteiksmes spēju salīdzinājums attēlots 1.2 attēlā. Pirmais profils, kas tiek saukts par OWL 2 / Full, atbilst pilnajai OWL 2 valodas specifikācijai un izteiksmes ziņā ir visspēcīgākais no visiem profiliem. Lai gan ar šo profilu iespējams izteikt sarežģītas konstrukcijas, tādā veidā iegūstot precīzāku zināšanu reprezentāciju, šis profils nav ideāls situācijām, kad izveidoto ontoloģiju plānots izmantot ar secinātājiem, jo daži tajā iekļautie secināšanas likumi ir pārāk sarežģīti, lai secināšanas rezultātu būtu iespējams izskaitļot saprātīgā laikā. [8] Lielākā daļa no tālāk minētajiem OWL 2 profiliem ir apakškopas no OWL 2 / DL profila, kas balstīts uz aprakstošo loģiku. [12] OWL 2 / EL profils ir paredzēts situācijām, kad ontoloģija satur lielu skaitu klašu un īpašību, starp kurām pastāv samērā sarežģītas attiecības. Šis profils ir optimizēts, lai uzlabotu secinātāju veiktspēju, kad tiek meklētas jaunas attiecības starp klasēm. Cits profils - OWL 2 / QL - ir paredzēts situācijām, kad ontoloģijā jāiekļauj liels skaits instanču datu, piemēram, kad ontoloģiju plānots izmantot kopā ar relāciju datubāzi. [13] Šī profila būtiskie ierobežojumi, salīdzinot ar pilno OWL 2 valodu, ļauj ontoloģijas vaicājumus izpildīt logaritmiskā laikā attiecībā pret apgalvojumu skaitu, kā arī būtiski atvieglo vaicājumu pārrakstīšanu uz SQL. Šīs īpašības tiek panāktas, atmetot tādas konstrukcijas kā, piemēram, kardinalitāšu ierobežojumus (ObjectExactCardinality, DataExactCardinality, utt.), indivīdu vai literāļu eksistences noteikšanu (ObjectHasValue, DataHasValue) un īpašību transitivitāti



Att. 1.2 **OWL 2 profilu salīdzinājums** *[62]*

(TransitiveObjectProperty). [14] Visbeidzot, OWL 2 / RL profils ir paredzēts izmantošanai kopā ar darījumloģikas programmām (business rules engines). Šo profilu var izmantot, lai bagātinātu RDF datus ar papildu nosacījumiem. [8]

No visa iepriekš apskatītā iespējams secināt, ka OWL valoda piedāvā daudz plašāku vārdnīcu, kas ļauj izveidot ievērojami detalizētākas ontoloģijas nekā RDF un RDFS. Būtisks ir arī fakts, ka OWL nodrošina, ka ar to veidotās ontoloģijas ir apstrādājamas ar dažādām loģiskās spriešanas programmām. Vairāki OWL profili ļauj pielāgot šo valodu dažādiem lietošanas gadījumiem.

1.3. SPARQL

Zem jēdziena SPARQL ietilpst gan pati vaicājumu valoda, kura ļauj izpildīt vaicājumus pār RDF datiem, gan arī protokols, kas definē SPARQL vaicājumu un rezultātu pārsūtīšanu, izmantojot HTTP protokolu. [15] SPARQL vaicājumu valoda tika veidota pēc SQL līdzības, tomēr starp abām valodām eksistē būtiskas atšķirības, jo SQL vaicājumi strādā ar datiem no relāciju datubāžu tabulām, savukārt SPARQL vaicājumi darbojas ar RDF grafiem, kuri ir trijnieku formā. [12] Viens no SPARQL valodas uzdevumiem ir atlasīt datus no dažādiem datu avotiem. [3] Pašā vienkāršākajā formā, SPARQL vaicājumus var iztēloties kā RDF grafus, kuru trijniekos vienā vai vairākās pozīcijās ir arī dažādi mainīgie. Šādus trijniekus var saukt arī par grafu šabloniem un tie ir SPARQL vaicājumu pamatā. SPARQL vaicājumu valodā atlases (SELECT) vaicājumi kā rezultātu atgriež datus, kuru RDF trijnieki atbilst vaicājuma grafa šablonam. [15] Tipisks SPARQL atlases vaicājums satur vairākus atslēgvārdus, bet pirmais no tiem gandrīz vienmēr ir PREFIX. Šis atslēgvārds norāda prefiksu deklarācijas, kurās katras izmantotās vārdnīcas URI tiek piekārtots kāds prefikss. Šos prefiksus izmanto, lai pašā vaicājumā nevajadzētu norādīt katra resursa vai predikāta vārdnīcas pilno URI, bet tā vietā varētu izmantot atbilstošo prefiksu. Tālāk tiek norādīts vaicājuma veids, kas tipiski ir atlase, kam atbilst atslēgvārds SELECT. Citi vaicājumu tipi tiks apskatīti vēlāk. Līdzīgi kā SQL, arī SPARQL atlases vaicājumos ir FROM atslēgvārds, kas norāda uz datu avotu – SPARQL gadījumā tā parasti ir RDF datu kopa. Lai atlasītu specifiskus datus, SPARQL vaicājumā tālāk seko WHERE atslēgvārds, ko izmanto, lai definētu grafa šablonu, pēc kura atlasīt datus. Papildus tam, WHERE nosacījumos iespējams definēt ne tikai grafu šablonus ar mainīgajiem, bet, izmantojot FILTER atslēgvārdu, ir iespējams norādīt konkrētus nosacījumus, pēc kuriem atlasīt ierakstus. [12] Pēc grafa šablona seko kāds no rezultāta modifikatoriem – ORDER BY, LIMIT vai OFFSET, kas attiecīgi norāda rezultātu kārtošanas kritēriju, ierobežo atgriezto rezultātu skaitu vai arī norāda rezultātu nobīdi. [15] Līdzīgi kā SQL, arī SPARQL eksistē iespējas atgriezt tikai unikālos ierakstus ar DISTINCT atslēgvārdu un veidot rezultātu

agregācijas ar GROUP BY atslēgvārdu. Kopš SPARQL versijas 1.1, šajā vaicājumu valodā tiek atbalstītas visas vienkāršās agregācijas funkcijas – skaitīšana, summēšana, minimālā/maksimālā vērtība, vidējā vērtība u.c. [16] Konkrētajā SPARQL versijā tika pievienoti arī atslēgvārdi RDF datu kopu modificēšanai, piemēram, INSERT, DELETE un CLEAR, kā arī iespēja definēt apakšvaicājumus, izmantot nosacījumu negācijas, ieviesta izteiksmju izmantošana pie SELECT atslēgvārda un citas izmaiņas. [17] Būtiska SPARQL valodas īpašība ir iespēja norādīt informāciju, kuru atgriezt tikai tad, ja tā ir pieejama. Šī īpašība tiek īstenota, izmantojot OPTIONAL atslēgvārdu, kas ļauj norādīt alternatīvus grafu šablonus, pret kuriem salīdzināt datus. [12]

Kā iepriekš tika minēts, eksistē vairāki SPARQL vaicājumu veidi un katram no tiem atbilst savs atslēgvārds. Iepriekš aprakstīts tika atlases vaicājums, kas izmanto SELECT atslēgvārdu. Šādi vaicājumi no RDF datu kopas atlasa datus, kas atbilst vaicājumā norādītajiem nosacījumiem un rezultātu parasti atgriež tabulas formā. SPARQL valodā eksistē arī ASK vaicājumi, kuri ļauj uzzināt, vai konkrētajā datu kopā eksistē vismaz viens ieraksts, kas atbilst vaicājumā dotajam grafa šablonam. Šie vaicājumi atgriež vērtības "true" vai "false". Nākamais vaicājumu veids ir CONSTRUCT. Vaicājumi ar šo atslēgvārdu atgriež nevis rezultātu tabulu, bet gan RDF trijnieku kopu, kas tiek veidota, atlasot grafa šablonam atbilstošos ierakstus un aizpildot šablona mainīgo vērtības ar šo ierakstu datiem. [12] Šāda tipa vaicājumus var izmantot, lai pārtulkotu RDF datus starp dažādām ontoloģijām. Visbeidzot, SPARQL valodā ir definēti arī DESCRIBE vaicājumi, kas atgriež RDF grafu, kurā tiek aprakstīts kāds konkrēts resurss. Tā kā eksistē dažādas implementācijas un formāti šī tipa vaicājumiem, tad tie tiek izmantoti salīdzinoši retāk. [15]

Nodaļas sākumā tika pieminēts arī SPARQL protokols, kas definē vaicājumu un rezultātu pārsūtīšanu, izmantojot HTTP protokolu. SPARQL protokolā vaicājumiem tiek izmantotas HTTP GET un POST metodes. POST metodes pārsvarā izmanto, kad vaicājums ir pārāk garš, lai to iekļautu GET pieprasījuma parametru virknē (query string). Tā kā SPARQL 1.1 ieviesa arī datu modificēšanas iespējas, arī protokols tika izmainīts, lai to atbalstītu. Datu modifikācijas pieprasījumiem tiek izmantotas PATCH vai POST metodes. [16]

Kā jau tika minēts iepriekš, SPARQL valoda tika veidota, lai tā būtu līdzīga SQL valodai, tomēr starp abām valodām eksistē vairākas būtiskas atšķirības, kas izriet no šīm valodām atbilstošo datu avotu īpašībām. Pirmā no atšķirībām ir redzama nezināmo vērtību apstrādē. Tā seko no fakta, ka relāciju datubāzēs nezināmi dati tiek aprakstīti ar "null" vērtību, savukārt, RDF datos nezināma informācija vispār netiek iekļauta, jo attiecīgie trijnieki vienkārši neeksistē. [3] Abas valodas ļauj atlasīt datus ar trūkstošām informācijas daļām – SQL valodā to

var panākt ar LEFT OUTER JOIN, bet SPARQL – ar OPTIONAL atslēgvārdu, tomēr abi atslēgvārdi strādā dažādos veidos. Nākamā atšķirība ir saistīta ar risinājumu kopu operācijām UNION un MINUS. Tā kā SQL tabulās datiem ir noteikts formāts, izmantojot šīs operācijas, ir jānodrošina, ka visās rezultātu kopās iegūto datu formāti būs vienādi. Tā kā SPARQL izmantotajos datos šādu shēmas ierobežojumu nav, tad arī šī problēma nav tik izteikta. Iespējams viena no būtiskākajām atšķirībām starp SPARQL un SQL ir SPARQL dotā iespēja izpildīt vaicājumus pār vairākiem datu avotiem vienlaicīgi. To lielā mērā nodrošina RDF modeļa pamatīpašības un šāda darbība SQL valodā nav realizējama bez papildu konfigurācijas. [3]

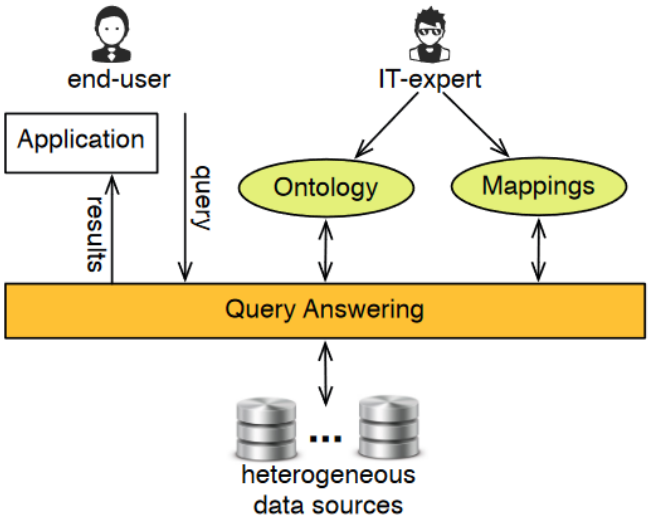
No apskatītajām SPARQL valodas īpašībām var secināt, ka šī vaicājumu valoda ir cieši saistīta ar RDF datu modeli un ļauj veikt dažādu tipu vaicājumus un operācijas pār RDF datu kopām. Lai gan valoda veidota pēc SQL līdzības, starp abām šīm valodām eksistē vairākas atšķirības, no kurām dažas var minēt kā SPARQL priekšrocības, piemēram, iespēju izpildīt vaicājumus pār vairākiem datu avotiem bez papildu sagatavošanās.

2. OBDA tehnoloģijas un rīki

Šajā nodaļā tiks apskatītas OBDA paradigmā izmantotās attēlojumu tehnoloģijas un atsevišķi rīki, kurus iespējams izmantot, lai realizētu OBDA risinājumus, kā arī aprakstīts ViziQuer rīks un vizuālu semantisku vaicājumu veidošana ar to. Nodaļā aprakstītie rīki tika izvēlēti, balstoties uz vienu no iepriekšējo gadu maģistra darbiem [18], kurā tika aprakstīti dažādi OBDA paradigmā izmantojami rīki un to darbināšana vienotā infrastruktūrā.

2.1. Ontoloģijās balstīta datu pieeja

Ontoloģijās balstīta datu pieeja jeb OBDA ir datu pieejas paradigma, kurā datiem piekļūst caur papildu konceptuālo slāni. Šo slāni tipiski veido RDFS vai OWL ontoloģijas, savukārt dati visbiežāk tiek glabāti relāciju datubāzēs. [19] OBDA pieejā gala lietotājiem ir pieejama ontoloģija, kas satur semantiski bagātu zināšanu apgabala konceptuālo modeli. Lietotāju vaicājumi tiek automātiski pārtulkoti un izpildīti pār datu avotiem. Tulkošanā tipiski tiek izmantoti attēlojumi, kas definē ontoloģijā atrodamo jēdzienu un datu avotā pieejamo objektu attiecības. [20] Viens no galvenajiem OBDA paradigmas ieguvumiem ir tas, ka gala lietotājiem ir pieejama interesējošā zināšanu apgabala ontoloģija un nav nepieciešamas zināšanas par dažādajiem datu avotiem, no kuriem tiek iegūta informācija. Tas ļauj panākt, ka gala lietotājiem ir mazāk jāpaļaujas uz IT ekspertiem, kad nepieciešams iegūt kādu konkrētu informāciju, kā arī šāda pieeja ļauj integrēt datu avotus, kuriem ir dažādas shēmas. [20] Tomēr, kā redzams attēlā 2.1, IT ekspertu iesaiste vēl joprojām ir nepieciešama, lai uzturētu ontoloģiju un attēlojumus.



Att. 2.1 **Tipiska OBDA risinājumu arhitektūra** *[20]*

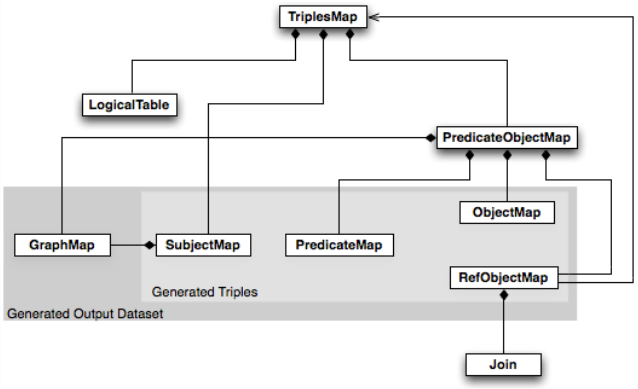
Viena no OBDA pieejas problēmām ir nepieciešamība pēc datiem piemērotas ontoloģijas un atbilstošajiem attēlojumiem, ko praksē bieži vien ir grūti vai dārgi iegūt. Papildus tam, gan ontoloģija, gan attēlojumi var mainīties atbilstoši gala lietotāju vajadzībām. [20] Kā vēl viena problēma tiek minēta tipisko OBDA risinājumu nespēja apstrādāt liela apjoma datus [21], tomēr eksistē projekts Optique, kurā šo un citas OBDA problēmas tiek mēģināts risināt. Vienā no pētījumiem [20], kur šis rīks tiek aprakstīts, tiek minēti arī divi praktiski lietošanas piemēri, kur abos gadījumos autori plāno pielietot Optique rīku, lai ar OBDA paradigmu ļautu

lietotājiem izpildīt semantiskus vaicājumus pār lielo datu avotiem, tomēr šo praktisko piemēru rezultātus neizdevās atrast.

Kopumā, apskatot pieejamo informāciju par OBDA paradigmu, var secināt, ka šai datu pieejas paradigmai ir daudz potenciālu pielietojumu un tās galvenās priekšrocības ir dažādu datu avotu integrācija, kā arī iespēja gala lietotājiem bez zināšanām par datu avotiem pašiem veidot vaicājumus pār ontoloģiju ar mazāku IT ekspertu iesaisti nekā tradicionālās sistēmās.

2.2. R2RML attēlojumu valoda

R2RML ir valoda, kas ļauj definēt pielāgotus attēlojumus starp relāciju datubāzēm un RDF datu kopām. [22] Šī valoda ir viena no divām W3C rekomendācijām, kas ietilpst RDB2RDF standartu grupā. Otra rekomendācija ir t.s. tiešais attēlojums (direct mapping), kurā netiek paredzētas iespējas pielāgot attēlojuma rezultātā iegūto RDF grafu, bet tā vietā visas darbības tiek veiktas pēc iepriekš definēta algoritma. [23] Ar R2RML definētie attēlojumi ir konceptuāli un R2RML apstrādes programmas var tos pielietot dažādos veidos. Biežāk lietotās pieejas ir datu materializācija, kur no relāciju datubāzes datiem, izmantojot R2RML attēlojumus, tiek izveidots RDF grafs, un virtuālā pieeja, kur fiziska RDF grafa vietā tiek izmantota saskarne, kas atbilstoši attēlojumiem izpilda vaicājumus pār relāciju datubāzi, lai iegūtu nepieciešamos datus. R2RML valodā attēlojumi tiek izteikti kā RDF grafi, izmantojot Turtle sintaksi. [22] Attēlā 2.2. redzams R2RML valodas pārskats, kurā norādīti galvenie attēlojumu elementi. R2RML attēlojums datus no avota iegūst ar t.s. "loģiskajām tabulām", kas var būt vienkāršas tabulas no relāciju datubāzes, datubāzes skati vai arī SQL vaicājumi. Lai katru no loģiskajām tabulām pārveidotu uz RDF, tiek izmantoti trijnieku attēlojumi (triples maps) jeb nosacījumi, pēc kuriem katra loģiskās tabulas rinda tiek pārveidota uz vienu vai vairākiem RDF trijniekiem. Katram šim nosacījumam ir trīs daļas – loģiskā tabula, no kuras ņemt datus, subjekta attēlojums (subject map), no kura tiek ģenerēti trijnieku subjekti, un predikātu-objektu attēlojumi (predicate-object map), kas norāda, kā tiek ģenerēti trijnieku objekti un to attiecības ar subjektiem jeb predikāti. [24] R2RML valodā tiek paredzēta arī iespēja norādīt grafu attēlojumus, kuri nosaka, kādos nosauktos grafos katrs trijnieks tiek ievietots. Ir divi galvenie veidi, kā iespējams pielietot ar R2RML veidotos attēlojumus – pirmais ir pārtulkot atsevišķas



Att. 2.2 **R2RML valodas pārskats** *[22]*

relāciju datubāžu datu porcijas uz RDF un tās ielādēt nepieciešamajā RDF datu kopā, un otrais – izmantot attēlojumus kopā ar tulkošanas programmu, lai ienākošos SPARQL vaicājumus pārtulkotu uz SQL. [25]

Kā jau iepriekš tika minēts, viena no R2RML attēlojumu alternatīvām ir tiešais attēlojums. Šajā attēlojumā no relāciju datubāzes shēmas un datiem tiek ģenerēts RDF grafs, ko sauc arī par tiešo grafu. R2RML kontekstā tiešo attēlojumu mēdz izmantot, lai iegūtu pamata attēlojumu, kuru pēc tam pielāgot savām vajadzībām. [26] Ar tiešo attēlojumu veidoto RDF grafu struktūra un vārdnīca atbilst avota datubāzes tabulu un kolonnu nosaukumiem. Šāda pieeja var būt noderīga, kad relāciju datubāžu datus nepieciešams ar minimālu darbu publicēt RDF formātā. [27] Tā kā tiešais attēlojums neatbalsta izveidotā RDF grafa vārdnīcas un saišu pielāgošanu, ar šo metodi nav iespējams sasaistīt datubāzē esošus objektus ar jau eksistējošiem RDF resursiem. [25] Tā kā tiešais attēlojums pieturas pie avota datubāzes struktūras un nepiedāvā iespējas veidot pielāgotus attēlojumus, ar šo pieeju iegūtajās RDF datu kopās novērojamas arī, piemēram, RDF trijniekos pārveidotas kodifikatoru tabulas no daudzu-daudzu attiecībām avota datubāzē. [28]

Veicot izpēti, izdevās atrast arī informāciju par attēlojumu valodu RML (RDF Mapping Language), kas vispārina un paplašina R2RML valodu, par datu avotu ļaujot izmantot ne tikai relāciju datubāzes, bet arī, piemēram, JSON un CSV dokumentus. [29] Lai gan RML valoda ir vispārīgāka nekā R2RML, tā cieši seko R2RML specifikācijai un izmanto to pašu sintaksi, tādēļ RML ir savietojama ar R2RML. Viens no šīs valodas pielietojumiem ir datu integrācijas rīks Karma, kurā tiek izmantota pusautomātiska pieeja attēlojumu iegūšanai no dažādiem datu avotiem – rīks vispirms uzģenerē attēlojumus, kas piekārto datus ontoloģijas klasēm un pēc tam lietotājiem ir iespējams šo modeli manuāli pielāgot. [30]

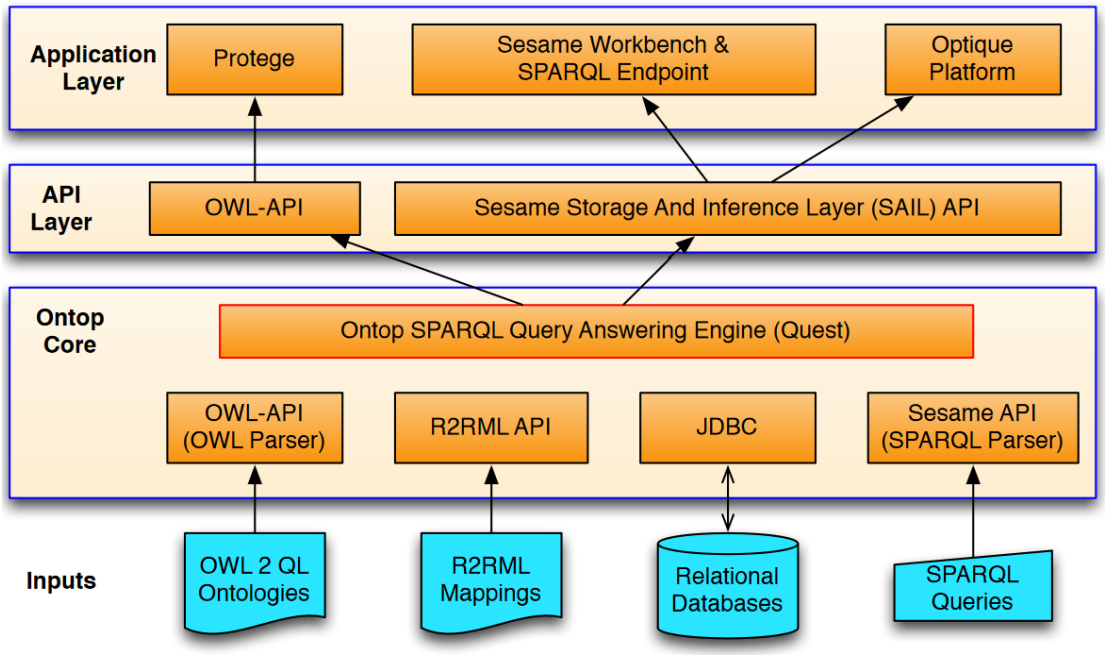
Pētot par attēlojumu valodām pieejamos materiālus, izdevās atrast arī vairākus darbus, kuros apskatītas R2RML valodas alternatīvas. [28, 27] Šajos darbos pieminētas tādas valodas kā Relational.OWL, R2O, R3M, D2RQ, Virtuoso RDF Views, Triplify u.c. Vienā no pētījumiem šīs valodas tika padziļināti salīdzinātas, izmantojot 15 dažādas īpašības un scenārijus, pēc kuriem šīs valodas tika iedalītas četrās grupās – tiešā attēlojuma valodas (tiešais attēlojums, SquirrelRDF), tikai lasāmu datu kopu vispārīga pielietojuma valodas (Virtuoso RDF Views, D2RQ un R2RML), lasāmu un rakstāmu datu kopu vispārīga pielietojuma valodas (R3M) un specializētas valodas (eD2R, R2O, Triplify). [28]

Apkopojot pieejamo informāciju par R2RML attēlojumu valodu, var secināt, ka ar šo valodu ir iespējams izmantot RDF datu modeli un SQL vaicājumus, lai definētu detalizētus un

konkrētam datu avotam pielāgotus attēlojumus, ar kuriem pārveidot relāciju datubāžu datus par RDF trijniekiem. Eksistē arī vairākas alternatīvas, kas paplašina R2RML piedāvātās iespējas, piemēram, ļaujot izmantot dažādus datu avotus, kas nav relāciju datubāzes, kā arī ir pieejamas vairākas citas valodas, kas pielāgotas konkrētiem lietojumiem.

2.3. Ontop

Ontop ir atvērtā pirmkoda OBDA platforma, kas relāciju datubāzes padara pieejamas semantiskiem vaicājumiem kā virtuālus RDF grafus. [19] Attēlā 2.3. ir redzama šī rīka arhitektūra, kuru veido četri slāņi – ievaddati, kas satur informāciju un datus no konkrētā zināšanu apgabala, Ontop kodols, kurā notiek vaicājumu tulkošana, optimizācija un izpilde, lietojumprogrammu saskarņu slānis, kas lietotājiem padara pieejamas Java saskarnes un lietotņu slānis, kurā ietilpst visas programmas, kas ļauj gala lietotājiem izpildīt SPARQL vaicājumus.



Att. 2.3 **Ontop rīka arhitektūras pārskats** *[19]*

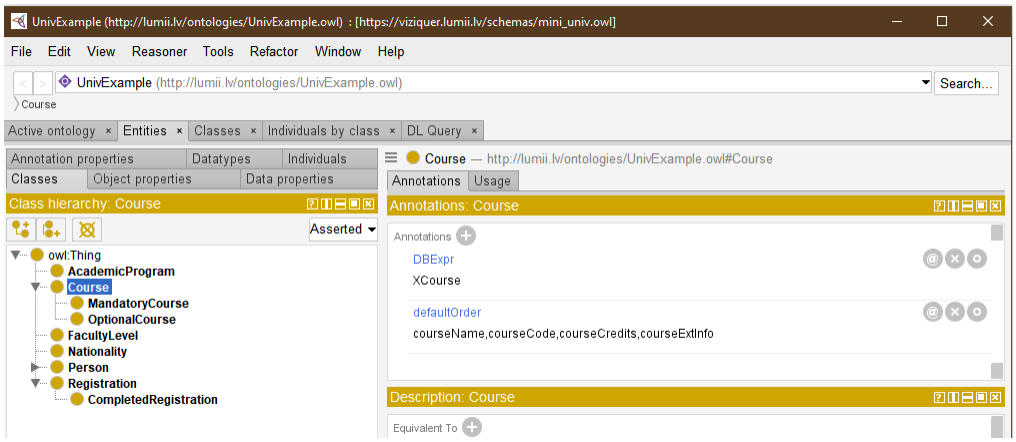
Viena no Ontop rīka īpašībām ir tā, ka tajā tiek pilnībā atbalstītas visas W3C rekomendācijas, kas saistītas ar OBDA paradigmu – OWL 2 /QL profils, R2RML attēlojumu valoda, SPARQL vaicājumu valoda u.c. [19] Ontop rīks atbalsta visas relāciju datubāžu sistēmas, kas atbilst SQL 99 standartam, tajā skaitā arī vairākas komerciālas un atvērtā pirmkoda datubāžu sistēmas, piemēram, Oracle, Microsoft SQL Server, PostgreSQL, MySQL, H2 un citas. [31] Ontop rīks atbalsta gan RDFS datu kopas, gan OWL 2 / QL ontoloģijas. Konkrētais OWL 2 profils tiek izmantots, jo vaicājumus pār tajā veidotām ontoloģijām vienmēr ir iespējams pārveidot par relāciju datubāžu vaicājumiem. [19, 8] Ontop rīks atbalsta divas attēlojumu valodas – R2RML un Ontop rīkam specifisku valodu, kura esot sintaktiski vienkāršāka nekā R2RML. Rīkā tiek

iekļautas arī funkcijas, lai pārtulkotu attēlojumus starp šīm valodām. Vienā no avotiem tiek minēts, ka rīks atbalsta gandrīz visas SPARQL 1.0 iespējas [19], tomēr, apskatot rīka versiju vēsturi, tika noskaidrots, ka jaunākās versijas atbalsta arī vairākas SPARQL 1.1 iespējas, piemēram, agregācijas funkcijas. [32] Rīkam ir pieejams arī Protégé spraudnis, kas ļauj izmantot grafisku saskarni, lai rediģētu attēlojumus, izpildītu SPARQL vaicājumus, pārbaudītu ontoloģijas konsekvenci, izveidotu ontoloģijas un attēlojumus no avota datubāzes un pildītu citas ar OBDA saistītas funkcijas. Pats Ontop rīks tiek izmantots arī Optique platformā, kas tika pieminēta, vispārīgi apskatot OBDA paradigmu 2.1. nodaļā. [19] Gan vecāko, gan jaunāko Ontop rīka versiju izmantotās vaicājumu tulkošanas pieejas ir aprakstītas 3.3. nodaļā.

Apskatot Ontop platformu, var secināt, ka viena no tās galvenajām iezīmēm ir atbilstība vairākām semantiskā tīmekļa tehnoloģiju specifikācijām, kas palīdz nodrošināt rīka korektu darbību. Ontop vaicājumu pārtulkošanā tiek izmantota specifiska pieeja, kas ļauj veikt optimizācijas, pirms vaicājums ir pārtulkots uz SQL un izpildīts datubāzē.

2.4. Protégé

Protégé ir atvērtā pirmkoda rīks, kas ļauj izveidot un uzturēt OWL ontoloģijas. Rīkam ir pieejama gan lejupielādējama versija (Protégé 5), gan arī tīmekļa vidē darbināma versija (WebProtégé). [33] Rīkā ir iekļauta plaša funkcionalitāte, tāpēc, lai palīdzētu to organizēt, visas



Att. 2.4 **Protégé rīka saskarne**

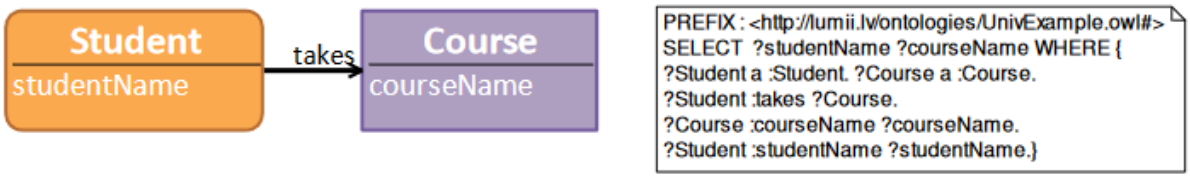
funkcijas ir izkārtotas cilnēs. Attēlā 2.4 redzams ekrānuzņēmums, kur Protégé rīka lejupielādējamā versijā tiek apskatīta ontoloģijas klašu hierarhija. Rīks piedāvā iespējas pielāgot, kuras cilnes tiek rādītas, kā arī mainīt to, kādas apakšcilnes ir redzamas, kad ir izvēlēta kāda konkrēta cilne, tādējādi lietotājiem ir iespējams grupēt dažādas funkcijas. Rīkam ir pieejams arī OWL spraudnis, kas pievieno secinātāja saskarni, kas ļauj ar loģisko secināšanu iegūt dažādu informāciju par ontoloģijas klasēm un to instancēm. Kā Protégé rīka galvenās priekšrocības, salīdzinot ar citiem ontoloģiju redaktoriem, vienā avotā tiek minētas

mērogojamība un paplašināmība. [34] Šajā pašā avotā tiek norādīts, ka citos pētījumos Protégé rīks esot bez problēmām ticis izmantots liela izmēra ontoloģiju izveidē un apstrādē, savukārt iepriekš minētā ciļņu sistēma nodrošinot vieglu integrāciju ar dažādiem papildu rīkiem un spraudņiem.

No apskatītās informācijas par Protégé rīku var secināt, ka tas piedāvā plašu funkcionalitāti OWL ontoloģiju rediģēšanai, ļauj pielāgot lietotāja saskarni, lai atvieglotu darbu, kā arī ir veidots, lai atbalstītu dažādus paplašinājumus, ar kuriem iespējams pievienot funkcionalitāti, kas nav iekļauta pamata rīkā.

2.5. ViziQuer

ViziQuer ir LUMII izstrādāts atvērtā pirmkoda rīks vizuālu semantisku vaicājumu veidošanai. [35] ViziQuer vaicājumu pieraksts ir balstīts uz UML klašu diagrammām, kuras tiek plaši pielietotas dažādās nozarēs. [36] Attēlā 2.5. redzams vienkārša vizuālā vaicājuma piemērs, kā arī tam blakus ir parādīts SPARQL vaicājums, ko rīks ģenerē no vizuālā vaicājuma. Šajā piemērā ar oranžu noapaļotu taisnstūri norādīta vaicājuma pamatklase "Student", bet ar gaiši violetu taisnstūri norādīta nosacījuma klase "Course". Ar melno bultu apzīmēta meklētā attiecība starp abām klasēm. Rīkā tiek atbalstītas trīs dažādu veidu attiecību saites – apstiprinošas (apzīmē ar nepārtrauktu melnu bultu), neobligātas (apzīmē ar raustītu gaiši zilu bultu) un noliedzošas (apzīmē ar sarkanu bultu un piezīmi "{not}"). [36]



Att. 2.5 **Vienkārša ViziQuer vaicājuma piemērs un tam atbilstoši ģenerētais SPARQL vaicājums** *[36]*

Vecākas ViziQuer rīka versijas tika veidotas kā darbvirsmas lietotnes Windows videi, tomēr jaunākās versijas izmanto tīmekļa saskarni. Katrs lietotājs var savā vidē veidot projektus un katrā no tiem var veidot diagrammas – katra no tām var saturēt vienu vai vairākus vaicājumus. [37] ViziQuer rīka lietošana un piedāvātās funkcijas ir sīkāk aprakstītas gan tā lietošanas pamācībā [37], gan vairākos iepriekšējo gadu maģistra darbos. [38, 18, 39]

Citi rīki, kas piedāvā līdzīgas funkcijas, ir OptiqueVQS un QueryVOWL, tomēr šiem rīkiem ir mazākas vaicājumu izteiksmes iespējas nekā ViziQuer, jo tie, piemēram, neatbalsta SPARQL 1.1. standarta agregācijas funkcijas. [36] Veicot papildu izpēti saistībā ar abiem iepriekš minētajiem rīkiem, neizdevās atrast informāciju, kas apstiprinātu, ka šāda

funkcionalitāte tiem ir pievienota.

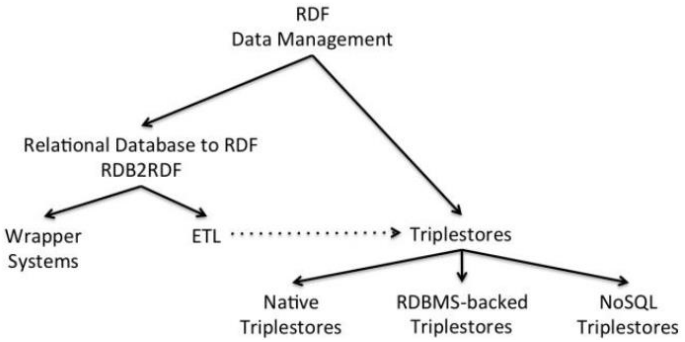
Apskatot par ViziQuer rīku pieejamo informāciju, var secināt, ka tas ir plaši pētīts vairākos pētījumos un maģistra darbos. Rīks atvieglo semantisku vaicājumu veidošanu dažādu kvalifikāciju cilvēkiem un izmanto uz UML balstītu vaicājumu vizuālā pieraksta formu, kā arī piedāvā plašākas vaicājumu izteiksmes iespējas nekā daži citi vizuālu semantisko vaicājumu rīki.

3. SPARQL vaicājumu tulkošana uz SQL

Šajā nodaļā tiks klasificētas un sīkāk izpētītas esošās metodes un optimizācijas SPARQL vaicājumu tulkošanai uz SQL vaicājumiem, lai ar OBDA pieeju izgūtu datus no relāciju datubāzēm.

3.1. RDF datu pārvaldības pieeju klasifikācija

Pirms apskatīt dažādās pieejas, ar kurām iespējams OBDA pieejas ietvaros veikt SPARQL vaicājumu tulkošanu uz SQL, ir vērts noskaidrot, kādas ir galvenās pieejas RDF datu glabāšanai. Attēlā 3.1. ir redzams, ka šīs pieejas iedalās divās pamata grupās – RDB2RDF pieejas un trijnieku glabātuvju (triplestores) pieejas. Otrajā grupā ietilpst gan īstas trijnieku glabātuves, gan relāciju un NoSQL datubāzes, kurās tiek glabāti trijnieki. ETL pieejās ar attēlojumu palīdzību relāciju datubāžu dati tiek pārveidoti RDF formātā un ielādēti trijnieku glabātuvēs vai datubāzēs. [40] Tā kā šādas pieejas vaicājumu tulkošanas kontekstā ir gandrīz vienādas ar trijnieku glabātuvju pieejām, tad no RDB2RDF pieejām sīkāk tiks apskatītas tikai t.s. "wrapper" pieejas.



Att. 3.1 **RDF datu pārvaldības pieejas** *[40]*

Trijnieku glabātuves ir datubāžu pārvaldības sistēmas, kurās RDF ir galvenais datu modelis un kurās iespējams izpildīt SPARQL vaicājumus pār glabātajiem datiem. Eksistē gan specifiski RDF datiem paredzētas sistēmas, gan arī sistēmas, kurās datu glabāšanai tiek lietotas relāciju vai NoSQL datubāzes. SPARQL vaicājumu tulkošana ir aktuāla otrā veida sistēmās, kur vaicājumi tiek pārtulkoti uz SQL un izpildīti pār datubāzēm, kuru shēmas modelē RDF datu modeli. [40] Šī darba ietvaros sistēmas, kurās RDF trijnieki tiek glabāti relāciju datubāzēs, tiks sauktas par trijniekus saturošām relāciju datubāzēm. Šīs pieejas balstās uz RDF grafu materializāciju, t.i., fizisku glabāšanu, un, lai gan šādi iespējams uzlabot vaicājumu atbildes laiku un mazāk paļauties uz sākotnējo datu avotu sasniedzamību, tomēr šādas pieejas prasa datu regulāru atjaunināšanu un uzturēšanu. [41]

Cita pieeju grupa ir "apvalka" jeb "wrapper" pieejas, kurās netiek veidota relāciju datubāžu datu kopija, bet gan izmantota loģiska šo datu reprezentācija RDF modelī jeb virtuāls RDF grafs. Šajās pieejās SPARQL vaicājumi tiek pārtulkoti uz SQL un nodoti RDBMS, lai tie tiktu izpildīti pār avota datiem. [40] Šīs metodes apiet problēmas, kas rodas, uzturot datu kopijas un ļauj vienmēr iegūt aktuālos datus. Lai gan viens no šo pieeju ieguvumiem ir iespēja izmantot

RDBMS piedāvātās vaicājumu optimizācijas, tomēr ir svarīgi panākt, ka tulkošanas procesā tiek ģenerēti vaicājumi, kurus RDBMS tik tiešām spēs optimizēt. [42]

Lai gan visās vaicājumu tulkošanas pieejās SPARQL vaicājumi tiek pārtulkoti uz SQL vaicājumiem, kuri tiek izpildīti pār relāciju datubāzēm, tomēr trijniekus saturošu relāciju datubāžu pieejās tulkotie vaicājumi tiek izpildīti pār datubāzēm, kuru shēmas modelē RDF trijnieku struktūru, savukārt virtuālo grafu vai "wrapper" pieejās pārtulkotie SQL vaicājumi tiek izpildīti pār patvaļīgām datubāžu shēmām, kas paredzētas attiecīgo lietotņu vai sistēmu relāciju datu glabāšanai. [40]

Apskatot dažādās pieejas RDF datu glabāšanai, var secināt, ka tās var iedalīt divās grupās - pieejas, kas izmanto fiziskas trijnieku glabātuves vai trijniekus saturošas relāciju datubāzes, un pieejas, kas veido virtuālas datu reprezentācijas. Papildus tam, ir iespējams secināt arī to, ka vaicājumu tulkošana ir būtiska sastāvdaļa abu grupu pieejās.

3.2. Trijniekus saturošu relāciju datubāžu pieejas

Šajā apakšnodaļā apskatītās vaicājumu tulkošanas pieejas paredzētas SPARQL vaicājumu tulkošanai uz SQL vaicājumiem pār relāciju datubāžu shēmām, kuras modelē RDF datu modeli.

Pētījumā [40] ir aprakstīts rīks Ultrawrap, kas, tāpat kā Ontop un citi līdzīgi rīki, nodrošina semantisku vaicājumu izpildi pār relāciju datubāzēm. Šajā rīkā izmantotā pieeja vaicājumu tulkošanai balstās uz SPARQL vaicājuma pārveidošanu par SPARQL algebras izteiksmju koku, kurā katrs koka iekšējais mezgls ir SPARQL algebras operators, bet katra lapa – RDF trijnieka šablons. Konkrētajā pieejā SPARQL algebras koka JOIN operācijas tiek pārveidotas par SQL INNER JOIN, UNION operācijas – par SQL UNION un SPARQL OPTIONAL konstrukcijas - par LEFT OUTER JOIN operāciju SQL vaicājumā. Ultrawrap rīka pieeja balstās uz t.s. Tripleview (trijnieku skatu), kas konceptuāli atbilst vienkāršai RDF trijnieku tabulai, tomēr šajā pieejā šis skats netiek materializēts. Trijnieku skata izveide daļēji balstās uz W3C definēto RDB2RDF tiešā attēlojuma (direct mapping) standartu. Lai uzlabotu vaicājumu ātrdarbību un veicinātu relāciju datubāzes indeksu izmantošanu, šis trijnieku skats ir papildināts ar vēl divām kolonnām – subjekta primārā atslēga un objekta primārā atslēga. Avotā minētās Ultrawrap rīka vaicājumu optimizācijas ir līdzīgas tām, kas tiek lietotas arī citos tālāk apskatītajos rīkos – neapmierināmu nosacījumu noteikšana un t.s. "self-join" operāciju likvidēšana (JOIN operācijas, kurās tabula tiek savienota ar sevi). Vienā no apskatītajiem avotiem [43] gan tiek minēts, ka Ultrawrap izmantotās optimizācijas atsevišķos testēšanas scenārijos tomēr nestrādājot. VKG sistēmu apkopojumā [44] minēts, ka Ultrawrap pārsvarā

paļaujas uz izmantotās relāciju datubāzes veiktajām SQL vaicājumu optimizācijām, jo tas esot paredzēts darbam ar komerciālām datubāzēm, kurās pieejama labāka vaicājumu plānošana nekā brīvi pieejamās datubāzēs.

Vienā no apskatītajiem pētījumiem [45] ir aprakstīta teorētiska pieeja, kā pielāgot SPARQL algebru tulkošanai uz SQL, katru SPARQL valodas konstrukciju izsakot ar relāciju algebras operācijām, kā arī minētas atsevišķas optimizācijas un semantiskas neatbilstības starp SPARQL un SQL. Pētījumā par pamatu tiek ņemta situācija, kurā RDF dati tiek trijniekus saturošā relāciju datubāzē. Jāpiemin, ka pētījumā netiek apskatītas agregācijas funkcijas, kas SPARQL standartā tika ieviestas tikai ar 1.1. versiju. Pētījumā tiek apskatītas arī vairākas vaicājumu tulkošanas optimizācijas, kuras ir iekļautas arī vairākos rīkos. Viena no piedāvātajām optimizācijām ir izvairīties no JOIN operācijām pār UNION rezultātiem, tā vietā nepieciešamo JOIN operāciju izpildot pār abām UNION daļām atsevišķi. Šī optimizācija tiek pieminēta arī pētījumā [43] un tika izmantota vecākās Ontop rīka versijās. Cita optimizācija, kas minēta arī pētījumā [45], ir JOIN operāciju "saplacināšana" jeb šo operāciju pārrakstīšana bez apakšvaicājumiem. Pētījumā tiek apskatītas arī dažas neatbilstības starp SPARQL un SQL semantiku, kā arī piedāvāti risinājumi, kā šīs problēmas apiet. Viena no problēmām ir saistīta ar JOIN operācijām un trūkstošām vērtībām - SQL JOIN operācijās rezultāts tiek atmests, ja kādam no JOIN nosacījumu atribūtiem ir NULL vērtība, tomēr SPARQL vaicājumos šāda pieeja nederētu. Tā vietā tiek piedāvāts JOIN rezultātu atmest tikai tad, ja tajā ir informācijas konflikts un kādam no mainīgajiem ir dažādas vērtības abās JOIN pusēs. Cita problēma ir saistīta ar OPTIONAL blokiem, kas satur citus OPTIONAL blokus – šādas konstrukcijas rada neskaidrību vaicājuma izpildes kokā, jo nav iespējams noteikt, kura no JOIN operācijām neizdosies, kas var ietekmēt vaicājuma rezultātu. Kā problēmas sakne tiek minēts fakts, ka SPARQL izpildes kokā rezultāti tiek apkopoti no kreisās uz labo pusi, savukārt relāciju modelī – no lejas uz augšu, tomēr risinājums šai problēmai pētījumā netiek piedāvāts. Pēdējā no problēmām saistīta ar mainīgo "redzamību" (scope) FILTER nosacījumos – SPARQL valodā šajos nosacījumos iespējams izmantot jebkuru no mainīgajiem, kas izmantoti vaicājumā, savukārt relāciju modelī atlases operācijai ir pieejami tikai mainīgie, kas ir tajā pašā apakškokā. Kā iespējams problēmas risinājums tiek minēta atlases operācijas pārvietošana augstāk izpildes kokā, tomēr šāda pieeja nebūs efektīva, ja nepieciešams piekļūt mainīgajam vairākus līmeņus augstāk.

Kādā citā no atrastajiem pētījumiem tika aprakstīts risinājums, kurā vaicājumu tulkošanai tiek izmantots papildu slānis ar tam specifisku valodu, kuru autori dēvēja par AQL (abstract query language). [46] Arī šajā pētījumā apskatītā pieeja balstās uz RDF datu glabāšanu

trijniekus saturošā relāciju datubāzē. Kā AQL valodas galvenos mērķus autori min trīs nosacījumus:

- Vienu SPARQL vaicājumu jāpārtulko par vienu SQL vaicājumu
- Tulkotajā vaicājumā būtu jāizmanto datu tipi no relāciju datubāzes
- Tulkošanas sistēma nedrīkst būt piesaistīta konkrētai datubāzes shēmai vai SQL dialektam

Vaicājumu tulkošana šajā pieejā notiek trīs soļos – vispirms no SPARQL vaicājuma tiek izveidots AQL objekts, tad tiek veiktas optimizācijas un vaicājuma pielāgošana izmantotajai RDBMS un pēc tam tiek veikta SQL vaicājuma izveide. AQL valodā vaicājums ir datu objekts, kas satur sākotnējā SPARQL vaicājuma kārtošanas un atlases nosacījumus, kā arī datu kopas deklarāciju, kura satur JOIN operāciju koku un katras JOIN operācijas kritērijus. Šajā pieejā AQL objekts tiek apstrādāts, pielietojot dažādas optimizācijas, samazinot ligzdoto JOIN operāciju skaitu, kā arī veicot datu tipu secināšanu. Pētījuma rezultātos tiek minēts, ka, izmantojot AQL, tika atrisinātas vairākas būtiskas vaicājumu tulkošanas problēmas, t.sk. dažas no pētījumā [45] minētajām, un, lai gan autori demonstrē arī tādus vaicājumus, kurus ar šo metodi nevar pārtulkot, tiek piebilsts, ka šādi vaicājumi praksē esot reti sastopami.

Pētījumā [47] autori apraksta grafu šablonu tulkošanas algoritmu BGPtoSQL, kā arī vaicājumu tulkošanas algoritmu SPARQLtoSQL. Pētījumā aprakstītā tulkošanas metode paredzēta trijniekus saturošām relāciju datubāzēm. Šajā pieejā SPARQL vaicājums tiek sadalīts vienkāršos grafu šablonos (BGP - basic graph patterns) un tiek izmantots vaicājuma modelis, kas satur sakārtotu SELECT klauzulas atribūtu sarakstu un vaicājuma klauzulu koku, kura sakne ir WHERE klauzula. Katram koka mezglam tiek piekārtots atbilstošais grafa šablons no sākotnējā vaicājuma. Tulkošanas algoritms balstās uz vaicājuma klauzulu koka apstaigāšanu "preorder" secībā. Katrā mezglā esošajam grafa šablonam, izmantojot BGPtoSQL algoritmu, tiek piekārtots SQL vaicājums un katrā solī ar LEFT OUTER JOIN tiek pievienots iepriekšējā koka mezgla SQL vaicājums. Pētījumā bija iekļauti arī praktisku eksperimentu rezultāti, kuri norāda, ka izveidotie algoritmi strādājot gana labi uz dažādu formu vaicājumiem ar līdz pat 50 OPTIONAL klauzulām, tomēr tulkoto SQL vaicājumu ātrdarbību ietekmē vaicājumu selektivitāte, un vaicājumi, kuru gala rezultāti un starprezultāti ir lieli, var strādāt lēni, galvenokārt LEFT OUTER JOIN operāciju dēļ. Citā šo autoru pētījumā [48] SPARQL konstrukciju semantika tiek izteikta ar relāciju algebru, kā arī tiek formāli pierādīts, ka izmantotā tulkošanas metode saglabā sākotnējā vaicājuma semantiku. Konkrētā vaicājumu tulkošanas metode balstās uz divām attēlojumu grupām – α-attēlojumi, kas satur informāciju par to, kurā relācijā tiek glabāti trijnieki, kas potenciāli atbilst konkrētam šablonam, un β-

attēlojumi, kas satur informāciju par to, kurš relācijas atribūts satur katru trijnieka komponentu. Pētījumā tiek piedāvātas arī vairākas optimizācijas, lai iegūtu vienkāršākus SQL vaicājumus:

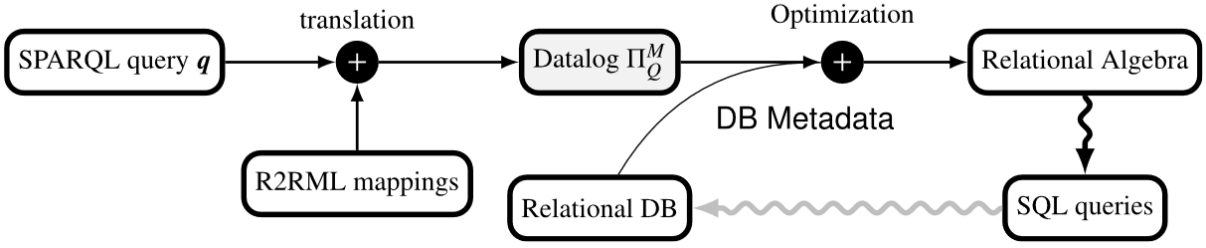
- Projekcijās atmest atribūtus, kas nav saistīti ar vaicājuma mainīgajiem
- Vaicājumos bez LEFT OUTER JOIN operācijām aizvietot COALESCE funkciju ar konkrēta mainīgā vērtību
- JOIN nosacījumu vienkāršošana, atmetot NULL pārbaudes, ja vaicājums nesatur LEFT OUTER JOIN operācijas
- Nosacījumos, kuri satur izteiksmes konjunkciju ar vērtību TRUE, atstāt tikai izteiksmi
- SPARQL UNION tulkojumā atmest LEFT OUTER JOIN, ja abām izmantotajām relācijām ir vienāda shēma un atribūtu projekcijas secība

Šie paši autori citā pētījumā [49] prezentē "nested optional join" operatoru relāciju datubāzēm, kas esot efektīvāks nekā standarta LEFT OUTER JOIN operators. Kā galvenās šī operatora priekšrocības tiek minētas efektīvāka NULL vērtības saturošu kortežu apstrāde, kā arī atbrīvošanās no nepieciešamības pēc NOT NULL pārbaudēm.

Apskatot pētījumus par dažādām vaicājumu tulkošanas pieejām, kas paredzētas trijniekus saturošām relāciju datubāzēm, var secināt, ka tās pārsvarā balstās uz materializētiem datiem un neizmanto R2RML vai līdzīgus attēlojumus. No apskatītajām metodēm vienīgais izņēmums šīm īpašībām bija rīks Ultrawrap, kas izmanto virtualizētu skatu, kura veidošanā lietots tiešais attēlojums. Šajās pieejās liela nozīme ir trijniekus saturošās relāciju datubāzes shēmas izvēlei un ir iespējamas dažādas optimizācijas, lai uzlabotu tulkoto vaicājumu ātrdarbību.

3.3. Virtuālu RDF grafu pieejas

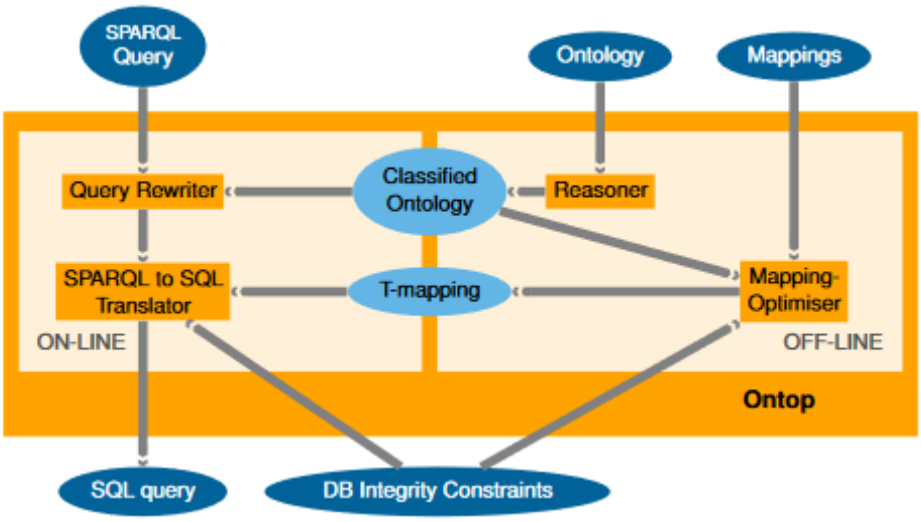
Šajā apakšnodaļā apskatītās vaicājumu tulkošanas pieejas paredzētas SPARQL vaicājumu tulkošanai uz SQL vaicājumiem pār relāciju datubāzēm ar patvaļīgām shēmām.



Att. 3.2 **SPARQL vaicājumu tulkošana uz SQL, izmantojot Datalog** *[43]*

Pētījumā [43] tiek piedāvāta vaicājumu tulkošanas metode, kas ļauj izmantot optimizācijas gan no loģiskās programmēšanas valodām, gan no SQL. Šī metode ir pamatā Ontop rīka pirmajām versijām un tā balstās uz virtuālu zināšanu grafu. Kā viena no galvenajām šādas pieejas priekšrocībām pētījumā tiek minēta iespēja efektīvāk izmantot RDBMS iestrādātās optimizācijas, kas esot viens no trūkumiem metodēm, kurās RDF dati tiek glabāti relāciju datubāzēs. Attēlā 3.2. ir redzams šīs metodes pārskats. Šajā pieejā no SPARQL vaicājuma un R2RML attēlojumiem tiek ģenerēta Datalog valodas programma, kas nav paredzēta izpildei, bet gan kalpo kā vaicājuma un attēlojumu formāla reprezentācija, kurai iespējams veikt vairākas strukturālas un semantiskas optimizācijas. Ģenerētajā Datalog programmā tiek saglabāta koka struktūra, kas atrodama sākotnējā SPARQL vaicājuma abstraktās sintakses kokā. Papildus SPARQL vaicājumam, procesā tiek iesaistīti arī R2RML attēlojumi, kas ļauj ģenerēt papildu Datalog programmas nosacījumus/likumus, kuri nodrošina metodes darbību uz patvaļīgām datubāžu shēmām. Ģenerētās programmas optimizācijas tiek veiktas, izmantojot daļējo izpildi (partial evaluation) un dažādas semantisku vaicājumu optimizācijas metodes. Pirmā no optimizācijām ir JOIN operāciju pārveidošana, lai URI šablonu vietā izmantotu vienkāršas atribūtu vērtības no datubāzes, piemēram, identifikatorus. Otrā optimizācija attiecas uz apakšvaicājumiem, kas satur UNION operācijas – šādus apakšvaicājumus izmantojot JOIN operācijās, kopējā ātrdarbība ir sliktāka nekā veidojot UNION apvienojumu no vaicājumiem, kuri satur JOIN operācijas. Papildus tiek aprakstītas arī tādas optimizācijas kā neapmierinātu nosacījumu atmešana un lieku JOIN operāciju noteikšana un atmešana. Kā redzams attēlā 3.2, veiktajās optimizācijās tiek ņemti vērā arī datubāzes metadati. Pēc pārveidojumiem un optimizācijām, Datalog programma tiek pārtulkota uz relāciju algebras izteiksmi, no kuras tiek iegūts SQL vaicājums. Divos no apskatītajiem pētījumiem [50, 43] tiek novērtēta šīs pašas pieejas tulkoto vaicājumu ātrdarbība salīdzinājumā ar citiem zināmiem rīkiem un tiek secināts, ka konkrētā pieeja standartizētos novērtējumos (BSBM un FishMark) dod labākus rezultātus nekā citi rīki. Izpētot citus avotus [19, 42], kuros aprakstīta Ontop rīkā izmantotā vaicājumu tulkošanas pieeja, tika secināts, ka iepriekš apskatītā pieeja vairs netiek izmantota rīka jaunākajās versijās.

Pētījumā [19] tiek aprakstīta Ontop arhitektūra, funkcionalitāte un savietojamība ar citiem rīkiem, kā arī detalizēti aprakstīts vaicājumu atbildēšanas process Ontop rīkā. Attēlā 3.3. attēlots šīs pieejas pārskats. Kā redzams, šis process notiek divās fāzēs – bezsaistes un tiešsaistes fāzē. Pirmajā fāzē OWL ontoloģija tiek klasificēta, izmantojot rīkā iebūvētu OWL 2/QL secinātāju un iegūstot ontoloģijas īpašību un klašu hierarhiju. Šis rezultāts tiek izmantots, lai ģenerētu t.s. T-attēlojumus (T-mappings), kuri pēc tam tiek optimizēti. Otrajā fāzē šie T-attēlojumi tiek lietoti, lai pārtulkotu SPARQL vaicājumu uz SQL. Iegūtais vaicājums pēc tam



Att. 3.3 **SPARQL vaicājumu tulkošana uz SQL jaunākajās Ontop versijās** *[19]*

tiek optimizēts un izpildīts. Šajā avotā aprakstītajā pieejā netiek pieminēta Datalog valoda, un, izpētot citus pieejamos avotus, konkrētāk – rakstu [42], tika noskaidrots, ka jaunākajās Ontop versijās vaicājumu tulkošanā vairs netiek lietota pāreja uz Datalog valodu. Kā iemesls šīm pārmaiņām tiek minētas grūtības realizēt SPARQL 1.1 agregātfunkcijas, kā arī citus operatorus ar Datalog valodas programmām. Pētījumā [19] aprakstītās vaicājumu optimizācijas iedalās divās grupās – strukturālās un semantiskās optimizācijas. Trīs strukturālās optimizācijas, kas tiek lietotas Ontop rīkā, ir JOIN operāciju pārvietošana iekš UNION operācijām, funkciju pārcelšana pēc iespējas augstāk vaicājuma kokā un apakšvaicājumu likvidēšana. Izmantotās semantiskās optimizācijas balstās uz datubāzes integritātes ierobežojumiem un ļauj aizvietot, piemēram, JOIN operācijas, kur tabulu savieno ar sevi, kā arī ļauj atrast neapmierinātus vai triviāli apmierinātus nosacījumus. Kā iespējamus nākotnes uzlabojumus Ontop rīkam autori min no datiem atkarīgas optimizācijas, kā arī pagaidām neatbalstītu SPARQL funkciju atbalstīšanu. Avotā [20] minēts, ka Ontop rīka vaicājumu tulkošanas sistēmu "Quest" izmanto arī Optique OBDA platformā, līdz ar to var secināt, ka arī šajā platformā tiek izmantota tāda pati vaicājumu tulkošanas pieeja kā Ontop.

Iepriekš jau pieminētais raksts [42] apskata Ontop rīka attīstību, kā arī rīka jaunākās versijas funkcijas un pieeju vaicājumu tulkošanai. Kā minēts iepriekš, Ontop rīkā kopš versijas v4 vaicājumu tulkošanā netiek izmantota Datalog valoda, bet gan uz relāciju algebru balstīta pieeja. Lai īstenotu šo pieeju, Ontop rīkā tiek lietota speciāli izstrādāta valoda Intermediate Query (IQ), ar kuru iespējams attēlot gan SPARQL, gan SQL vaicājumus. Papildus optimizācijām, kas izmantotas citos rīkos un iepriekšējās Ontop versijās, jaunākās Ontop versijas iekļauj arī optimizācijas OPTIONAL un MINUS operatoriem. Rakstā tiek atklāti arī vairāki ierobežojumi, kas vēl joprojām eksistē arī jaunākajās Ontop versijās – nav atbalstīta

vaicājumu datu avotu federācija (SERVICE), rīks neatbalsta arī SPARQL vaicājumus ar EXISTS operatoru, kā arī dažādas citas funkcijas un konstrukcijas. Daži no rīka trūkumiem minēti arī vienā no iepriekšējo gadu maģistru darbiem [18], kas tiks apskatīts tālāk. Lai gan Ontop rīkam ir atsevišķi ierobežojumi, vairākos pētījumos, kuri apkopoti referātā kursā "Zināšanu inženierija" [51], tiek secināts, ka eksistē rīki ar labāku veiktspēju, tomēr Ontop esot visstabilākais no VKG rīkiem.

Iepriekš minētajā maģistra darbā tika izveidots un testēts infrastruktūras prototips, kurā tiek izmantoti rīki ViziQuer, Ontop un Protégé, lai ar ViziQuer veidotos vizuālos vaicājumus pārtulkotu no SPARQL uz SQL un izpildītu pār relāciju datubāzēm. [18] Darbā tika norādīts uz vairākiem ierobežojumiem un problēmām ar vaicājumu tulkošanu Ontop rīkā:

- Ontop rīka veidotais SPARQL galapunkts neatbalstīja EXISTS operatorus, kuri tika iekļauti dažos no ViziQuer ģenerētajiem vaicājumiem. Apskatot Ontop rīka izmaiņu vēsturi [32], redzams, ka EXISTS un citi neatbalstītie operatori izmaiņās nav pieminēti, kas liek domāt, ka šī problēma vēl joprojām ir aktuāla
- Atsevišķos gadījumos, Ontop veidoto SQL vaicājumu apakšvaicājumos tika izmantots DISTINCT operators, pat ja sākotnējā SPARQL vaicājumā tas netika lietots, kā rezultātā tika iegūti nekorekti rezultāti
- 10 no 12 izpildītajiem vaicājumiem bija iespējams optimizēt, tos pārrakstot manuāli

Iepriekš minētās problēmas var mēģināt risināt, ieviešot papildu optimizācijas slāni starp ViziQuer un Ontop vai arī starp Ontop un RDBMS. Tā kā Ontop ir atvērtā pirmkoda rīks, cits variants būtu ieviest papildu optimizācijas pašā Ontop rīkā. Iepriekš minētajā maģistra darbā kā vēl viens iespējams risinājums tiek piedāvāta vaicājumu tulkošanas algoritma izstrāde pašā ViziQuer rīkā, tādējādi mazinot nepieciešamību pēc citu rīku iesaistīšanas, tomēr tas būtu ievērojami darbietilpīgāks risinājums nekā pārējie varianti.

Pētījumā [52] tiek īsumā apskatītas divu rīku – Mastro un Ontop – pieejas vaicājumu tulkošanai, kā arī salīdzināta abu rīku tulkoto vaicājumu ātrdarbība. Kā minēts pētījumā, rīks Mastro atbalsta tikai ierobežotu SPARQL valodas daļu un autoriem bija jāveic papildu darbs, lai panāktu, ka rīks atbalsta R2RML attēlojumus, jo līdz tam tas izmantoja specifiska formāta attēlojumus, kas tika definēti XML sintaksē. Mastro rīkā izmantotā vaicājumu tulkošanas metode sīkāk aprakstīta pētījumā [53] un, līdzīgi kā vecākās Ontop versijas, tā balstās uz konjunktīvu vaicājumu apvienojumiem (UCQ – union of conjunctive queries) un procesā sākotnējais vaicājums tiek reprezentēts kā Datalog programma. Sīkāka informācija par Mastro rīkā veiktajām optimizācijām pētījumos [52, 53] nebija pieejama. Citas komerciālas VKG

sistēmas ir Oracle Spatial and Graph un Stardog [44], tomēr arī šīm sistēmām neizdevās atrast detalizētu informāciju par tajās izmantotajām pieejām.

Materiālā [54] tiek apskatīta IBM veidotajā platformā SeDA (Semantic Data Access) izmantotā pieeja SPARQL vaicājumu tulkošanai. Šī pieeja balstās uz sākotnējā vaicājuma pārveidošanu par SPARQL algebras koku un katra fragmenta pārtulkošanu uz SQL. Procesā tiek izmantoti D2RQ attēlojumi un tā pamatā ir ideja pēc iespējas pilnīgāk saglabāt sākotnējā vaicājuma koka struktūru. Tāpat kā D2RQ rīkā, arī šajā platformā tiek izmantoti virtuāli RDF grafi. Atsevišķo fragmentu izpildes rezultāti tiek glabāti pagaidu tabulās relāciju datubāzē. Šajā materiālā tiek arī idejas līmenī apskatīta pieeja, kurā attēlojumus ģenerē no objektu modeļiem, tomēr detalizētāk tā netiek apskatīta.

Pētījumā [24] tiek paplašināta pētījumu [47, 48] vaicājumu tulkošanas metode, lai procesā iesaistītu R2RML attēlojumus un metodi pielāgotu RDB2RDF pieejai. Attēlojumu iesaistīšana vaicājumu tulkošanas procesā ļauj pieeju izmantot neatkarīgi no relāciju datubāzes shēmas. Šī pieeja tiek izmantota rīkā Morph un šādi ģenerētie vaicājumi ātrdarbības ziņā esot līdzvērtīgi manuāli rakstītiem SQL vaicājumiem. Gan pati pieeja, gan tajā izmantotās optimizācijas detalizēti aprakstītas avotā [55]. Šajā pieejā tiek izmantotas arī vairākas optimizācijas, piemēram, apakšvaicājumu skaita samazināšana, izpildot projekcijas un atlases pēc iespējas "zemāk" vaicājuma izpildes kokā, kā arī tādu JOIN operāciju likvidēšana, kur tabula tiek savienota pati ar sevi (t.s. "self-join"). Papildus iepriekš minētajām optimizācijām, pētījumā [55] tiek minētas arī tādas optimizācijas metodes kā OPTIONAL tulkošanā izmantoto LEFT OUTER JOIN operāciju aizvietošana ar efektīvākām konstrukcijām un IS NOT NULL pārbaužu atmešana, balstoties uz datubāzes metadatiem. Kā mazāk nozīmīgas optimizācijas tiek pieminētas arī, piemēram, tabulu secības maiņa JOIN operācijās, izmantojot relāciju datubāžu metadatos pieejamos tabulu izmērus, lai samazinātu JOIN starprezultātu izmērus, un lieku UNION operāciju atmešana. Lai gan šai metodei ir labi definēti teorētiskie pamati un tā ļauj izveidot efektīvus SQL vaicājumus, tomēr pētījumā [43] minēts, ka gan šī paplašinātā pieeja, gan arī pieeja, uz ko tā balstīta, darbojas ar OPTIONAL operatora semantiku, kas ir neprecīza attiecībā pret SPARQL 1.1. standartu. Atšķirībā no Ontop, Morph nav iespēju darboties ar ontoloģijām un veikt secināšanas operācijas. [44]

Pētījumos [24, 50] tiek salīdzināta tajos apskatīto tulkošanas pieeju ātrdarbība salīdzinājumā ar citiem rīkiem, t.sk. arī D2RQ platformu. Apskatot pētījumu [56], var secināt, ka D2RQ rīka lietotā pieeja izmanto virtuālu RDF grafu, lai realizētu SPARQL vaicājumu tulkošanu un izpildi pār relāciju datubāzēm, tomēr pētījumā netiek detalizēti apskatīta pati tulkošanas pieeja. Abos pētījumos [24, 50], eksperimentu ietvaros apskatot šī rīka tulkotos

vaicājumus, tiek secināts, ka ikviens SPARQL vaicājums tiek pārtulkots par vairākiem SQL vaicājumiem, kuru rezultāti tiek apstrādāti atmiņā, lai iegūtu gala rezultātu. Šāda pieeja ierobežo vaicājumu tulkošanas mērogojamību un palielina šī procesa atmiņas patēriņu. [50] Vēl kādā citā pētījumā tiek minēts, ka D2RQ rīka tulkoto vaicājumu veiktspēja esot atkarīga no izvēlētā pieejas veida izveidotajam RDF skatam (virtuālais grafs vai materializēti dati), kā arī rīka attēlojumu valodas iespējas esot pārāk ierobežotas gadījumiem, jo praksē bieži vien nepieciešami attēlojumi datubāzēm, kuras neatbilst tipiskiem datubāžu projektējumiem. [57] Kā viena no pirmajām VKG sistēmām, D2RQ pierādīja, ka ir iespējams atbildēt SPARQL vaicājumus pār relāciju datubāzēm, tos pārtulkojot uz SQL. [44] Detalizēta D2RQ rīkā izmantotās tulkošanas pieejas apraksta trūkums apgrūtina iespējamu uzlabojumu meklēšanu, kā arī vairākos pētījumos minētās rīka problēmas liek domāt, ka labāki rezultāti būtu sasniedzami, papildinot citu rīku lietotās pieejas.

Izpētot pieejamo informāciju par vaicājumu tulkošanas pieejām, kas lietotas rīkos, kuri balstās uz virtuāliem zināšanu grafiem, var secināt, ka visas apskatītās pieejas balstās uz R2RML vai līdzīgiem attēlojumiem, kā arī šajā pieejā ir pielietojamas dažas no tām pašām optimizācijām, kas tika minētas pie trijniekus saturošo relāciju datubāžu pieejām.

3.4. Citas pieejas un optimizācijas

Šajā nodaļā apskatītas dažādas citas pieejas un optimizācijas, kurās SPARQL vaicājumi tiek tulkoti uz valodām, kas nav SQL, vai arī kur tiek optimizēti sākotnējie SPARQL vaicājumi.

Vienā no apskatītajiem maģistra darbiem tika pētīta iespēja pielāgot ViziQuer rīku SQL vaicājumu izpildei pār MySQL relāciju datubāzi, tomēr šajā darbā autors nebija pētījis SPARQL vaicājumu tulkošanas iespējas [38]. Lai gan darba rezultātā tika papildināta ViziQuer rīka funkcionalitāte, nodrošinot iespēju izpildīt gan SPARQL, gan SQL vaicājumus, tomēr autora izvēlētā pieeja neizmanto SPARQL dotās iespējas attiecībā uz datu avotu integrāciju un vaicājumu izpildi pār vairākiem avotiem vienlaicīgi.

Avotā [58] tiek apskatīta nevis vaicājumu tulkošana, bet gan SPARQL vaicājumu optimizācijas, no kurām dažas var noderēt arī vaicājumu tulkošanas procesa uzlabošanā, lai vienkāršotu sākotnējos vaicājumus. Avotā tiek aprakstīta SPARQL vaicājumu izpilde un izceltas līdzības un atšķirības salīdzinājumā ar SQL vaicājumiem. Kā viena no SPARQL konstrukcijām, kas izpildē rada papildu JOIN operācijas, tiek minēta WHERE klauzula ar vairākiem trijnieku šabloniem, kuriem ir kopīgi mainīgie. Avotā tiek izceltas arī SPARQL un SQL atšķirības no vaicājumu izpildes viedokļa, piemēram, fakts, ka SQL vaicājumu rakstīšanai nepieciešamas zināšanas par izmantoto tabulu saturu, kā arī tas, ka SPARQL vaicājumi notiek

pār nezināmām datu struktūrām, bet SQL vaicājumi balstās uz tabulu struktūru. Lai optimizētu SPARQL vaicājumus, tiek piedāvāts izmantot pielāgotas SQL optimizācijas metodes, piemēram, indeksu izmantošanu. Tiek apskatītas arī citas metodes, piemēram, šablonu salīdzināšanas paralelizācija, SIP (Sideways Information Passing) pieeja, kurā paralēlās operācijas izmanto vienotu atmiņas apgabalu, kā arī vaicājumu optimizācija, izmantojot selektivitātes novērtējumus – trijnieku šablonus ar mazāko sagaidāmo rezultātu skaitu izpildot vispirms, iespējams samazināt starprezultātu izmērus.

Darba izstrādes gaitā tika apskatīti arī divi avoti, kas piedāvā pieejas SPARQL vaicājumu tulkošanai uz XQuery vaicājumiem pār XML datnēm. Pētījumā [59] apskatītais SPARQL2XQuery satvars tulko SPARQL vaicājumus uz XQuery, ko var izmantot, lai iegūtu informāciju no XML datiem. Tulkošanas procesā rīks izmanto attēlojumus starp OWL ontoloģijām un XML shēmām. Šajā pieejā tulkošana sākas ar SPARQL vaicājuma grafu šablonu normalizāciju, kur katru šablonu pārraksta vienkāršāku šablonu apvienojuma formā. Šie šabloni tiek dēvēti par UF-GP (Union-Free Graph Patterns) šabloniem. Pēc tam katrs šablons tiek pārtulkots uz XQuery, tad tiek pārtulkoti vaicājuma UNION operatori, rezultātu secības modifikatori un iegūtais XQuery vaicājums tiek pielāgots, lai iegūtu attiecīgajam SPARQL vaicājuma tipam (Select, Ask, Describe vai Construct) atbilstošu rezultātu. Pētījumā netiek apskatītas optimizācijas un veiktspēja.

Pētījumā [60] piedāvāts vienots risinājums SPARQL un SQL vaicājumu tulkošanai uz XQuery. Lai atvieglotu tulkošanas procesu, šajā pieejā tiek lietota RDF/XML sintakse, kā arī SQL/XML datu tipu attēlojumi. Pētījumā tiek apskatīts gan kā katru SPARQL konstrukciju pārtulkot uz XQuery. Tā kā galvenais apskatītā pētījuma mērķis bija iegūt pilnīgus un korektus vaicājumu tulkojumus, konkrētajā vaicājumu tulkošanas pieejā tika izmantotas tikai dažas relatīvi vienkāršas optimizācijas, piemēram, WHERE klauzulas nosacījumu pārvietošana pēc iespējas "zemāk" izpildes kokā, t.i., pēc iespējas tiek veikta nosacījumu pārbaude, lai samazinātu starprezultātu apjomu.

Kopumā var secināt, ka līdzīgi ar vaicājumu tulkošanu saistīti izaicinājumi ir sastopami arī citos SPARQL pielietojumos un ka eksistē vairākas metodes, kā iespējams pirms tulkošanas optimizēt SPARQL vaicājumus.

4. Vaicājumu izpildes infrastruktūra

Šajā nodaļā tiks aprakstīta darba praktiskajā daļā izmantotā infrastruktūra, precīzāk - konkrētu rīku izvēles pamatojums, Ontop nepieciešamās ontoloģijas un attēlojuma izveide, kā arī darbā izmantotās infrastruktūras uzstādīšana.

4.1. Infrastruktūrā iekļauto rīku izvēle

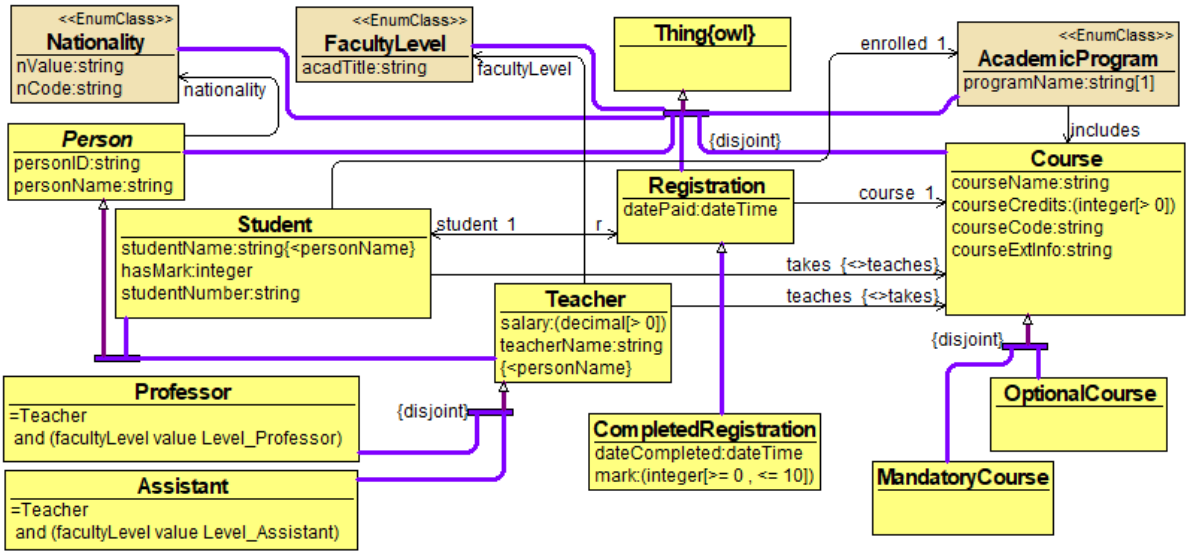
Darba ietvaros izmantotā vizuālu semantisku vaicājumu izpildes infrastruktūra tika veidota, balstoties uz 2.1. nodaļā attēloto OBDA pieejas shēmu, kā arī uz iepriekšējā nodaļā minēto maģistra darbu [18], kurā autors bija izmēģinājis ViziQuer rīka izmantošanu kopā ar Ontop un relāciju datubāzi. Atbilstoši minētajai OBDA pieejas shēmai, semantisko vaicājumu atbildēšanas un tulkošanas funkciju pilda Ontop, ontoloģiju un attēlojumu rediģēšanu iespējams veikt Protégé rīkā ar pievienotu Ontop spraudni, savukārt kā datu avots tika izmantota MySQL datubāze. Konkrētā datubāze tika izvēlēta, jo darba gaitā ir plānots darbināt gan Ontop, gan Morph rīkus, un šī datubāze ir viens no variantiem, kuru atbalsta abi rīki. [19, 61] Izveidotajā infrastruktūrā ViziQuer rīks darbojas kā papildu slānis virs tipiskās OBDA arhitektūras, un nodrošina iespēju definēt vizuālus semantiskus vaicājumus, kuri tiek pārveidoti par SPARQL vaicājumiem un nodoti tālākai apstrādei Ontop rīkā.

Praktisko eksperimentu veikšanai rīki Ontop un Morph tika izvēlēti divu iemeslu dēļ: pirmkārt, abu rīku pirmkods ir brīvi pieejams[2][3], kas dod iespēju īstenot iespējamos uzlabojumus, kā arī atvieglo piekļuvi rīku izmantošanai. Otrkārt, abi rīki balstās uz RDF grafu virtualizāciju – šī pieeja tika izvēlēta, jo, atšķirībā no trijniekus saturošu relāciju datubāžu pieejām, VKG pieejas var praktiski izmantot, lai veidotu semantiskā tīmekļa lietotnes, kuras darbojas paralēli eksistējošām programmām, kas izmanto relāciju datubāzi, tā izvairoties no pretrunām abu tipu programmu datos, jo netiek veidotas liekas datu kopijas. [40]

4.2. Ontoloģijas un attēlojuma izveide

Lai būtu iespējams salīdzināt rezultātus ar iepriekšējo gadu darbiem, praktisko eksperimentu veikšanai tika izvēlēts ViziQuer mājaslapā[4] pieejamais “mini-universitātes” piemērs, kura shēmas vizualizācija redzama attēlā 4.1. Šajā pašā vietnē ir pieejama arī šī piemēra ontoloģija, shēmas datne, kā arī datne, kas satur atbilstošo ViziQuer projektu ar rīka

[2] https://github.com/ontop/ontop
[3] https://github.com/oeg-upm/morph-rdb
[4] https://viziquer.lumii.lv/



Att. 4.1 **"Mini-universitātes" piemēra datu shēmas vizualizācija** *[37]*

instrukcijā atrodamajiem vizuālajiem vaicājumiem. Uzsākot praktisko darbu, nepieciešamais R2RML attēlojums tika ņemts no iepriekš minētā maģistra darba [18], tomēr, atverot ontoloģiju rīkā Protégé un pievienojot šo attēlojumu, tika atgrieztas vairākas kļūdas, piemēram, no attēlojumā norādītā SQL vaicājuma klases Course lauku vērtību iegūšanai nebija iespējams iegūt lauku "courseExtInfo", kas tiek veidots, savienojot divu kolonnu vērtības. Uzsākot darbu ar vaicājumu izpildi, tika atklāta arī kļūda attēlojuma fragmentā, kurā norādīts predikāts starp Student un Registration klasēm – šajā predikātā abas klases bija sajauktas vietām, kā rezultātā Registration objektu instances tika atgrieztas arī tad, ja vaicājumi tika veikti par Student vai Teacher klasēm.

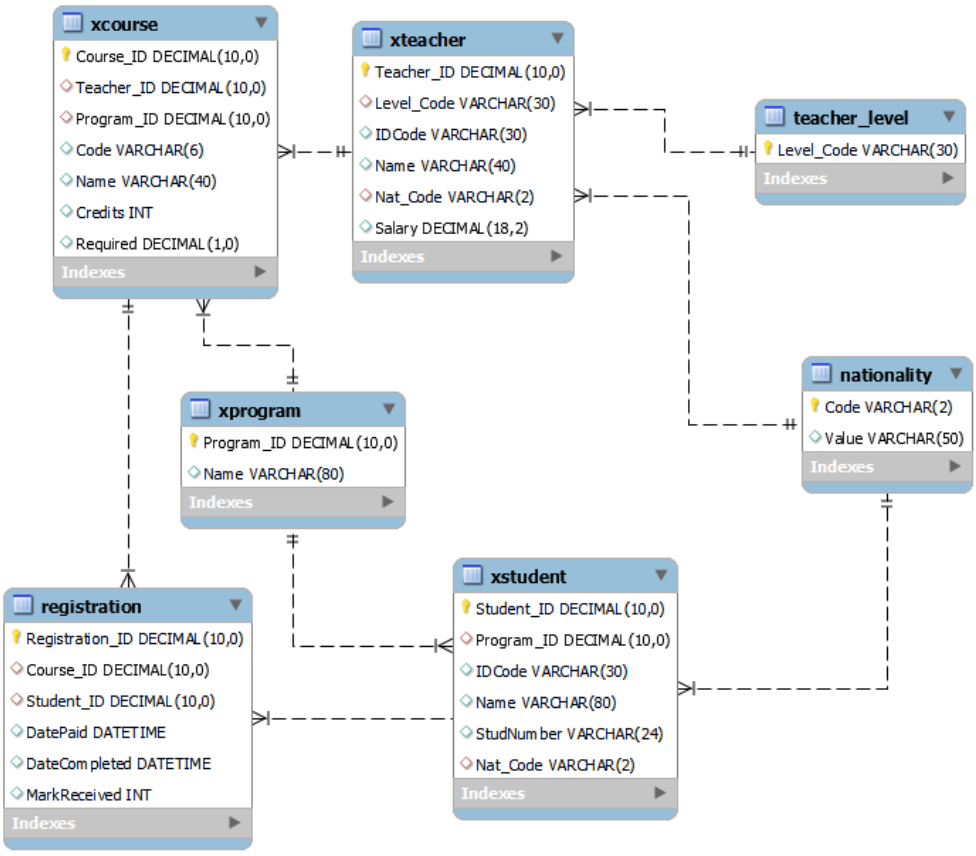
Pēc vaicājumu izpildes ar izlaboto attēlojumu, tika apsvērta arī ideja vienkāršot ontoloģiju, atmetot apakšklases, kuras netika tieši izmantotas ViziQuer rīka instrukcijas vaicājumos - Professor, Assistant, CompletedRegistration, MandatoryCourse un OptionalCourse. Ontoloģijas vienkāršošanu motivēja Ontop tulkoto SQL vaicājumu sarežģītība, kā arī Ontop rīka izdotie paziņojumi par OWL 2 / QL profila neatbalstītām konstrukcijām (ObjectHasValue, ObjectExactCardinality) sākotnējā ontoloģijā. Pēc ontoloģijas vienkāršošanas, iepriekš izveidotais attēlojums tika pielāgots jaunajai ontoloģijai, atmetot vai rediģējot konkrētās attēlojuma daļas, kas bija saistītas ar atmestajām klasēm. Vaicājumu izpildē izmantotās ontoloģijas galīgā versija apskatāma 1. pielikumā, savukārt izlabotais un vienkāršotais attēlojums atrodams 2. pielikumā.

Ontoloģijas un attēlojuma izveides procesā tika izmēģināta arī ontoloģijas un attēlojuma automātiska izveidošana ar Protégé rīka Ontop spraudni, tomēr šādi ģenerētajā ontoloģijā un attēlojumā klasēm, atribūtiem un saitēm tika izmantoti tabulu un lauku nosaukumi no datubāzes, kuri neatbilda konkrētā piemēra ontoloģijā izmantotajiem nosaukumiem, kā

rezultātā būtu nepieciešams papildu darbs, lai atrisinātu šīs neatbilstības. Šajā darbā galvenais izaicinājums būtu precīzi norādīt atribūtu, saišu un klašu nosaukumu atbilstību starp ontoloģiju, attēlojumu un datubāzi. Lai gan vairumā ar rīkiem ģenerētu ontoloģiju un attēlojumu ir sastopama iepriekš minētā nepilnība, tomēr automātiska ontoloģiju un attēlojumu ģenerēšana var kalpot par labu pamatu darba uzsākšanai ar datubāzēm, kurām nav pieejamas gatavas ontoloģijas vai attēlojumi. Vairums rīku, kas ļauj veikt šādu automātisku ģenerēšanu, balstās uz 2.2. nodaļā pieminēto tiešo attēlojumu, tomēr ir arī rīki, kas šo procesu veic pusautomātiski, ļaujot lietotājam ievadīt papildu datus ontoloģijas vai attēlojuma ģenerēšanas procesā. Kā piemērus citiem rīkiem, kas paredzēti ontoloģiju un attēlojumu automātiskai ģenerēšanai, var minēt BootOX, MIRROR, COMA un Karma. [44]

4.3. Infrastruktūras uzstādīšana

Kā minēts nodaļā 4.1., infrastruktūrā par datu avotu tika izvēlēta MySQL datubāze. Darba ietvaros lokāli uz Windows datora tika uzstādīts MySQL 8.0.24 serveris. Pēc uzstādīšanas tika izveidota datubāze ar nosaukumu “school2”. Datubāzes izveides SQL skriptam par pamatu tika ņemts skripts no iepriekšējā gada maģistra darba [18], kurā tika



Att. 4.2 **”Mini-universitātes” piemēram atbilstošās relāciju datubāzes shēma**

izmantota MS SQL Server datubāze. Pēc skripta sagatavošanas un pielāgošanas MySQL lietotajai sintaksei, datubāzes izveides skripts tika izpildīts un tika iegūta datubāze ar attēlā 4.2.

redzamo shēmu. Datubāzes izveidē izmantotais pielāgotais SQL skripts atrodams 3. pielikumā.

Pēc datubāzes uzstādīšanas tika sākts darbs pie Ontop, ViziQuer un Morph rīku konfigurācijas un uzstādīšanas. Lai atvieglotu rīku konfigurāciju un atkārtotu to apstādināšanu un piedarbināšanu, visi trīs rīki tika apvienoti vienā Docker Compose konfigurācijā. Šīs konfigurācijas saturs ir apskatāms 4. pielikumā. Docker rīka darbināšanā tika izmantota Docker Toolbox versija 19.03.1. Docker virtuālā mašīna tika darbināta ar VirtualBox 6.0.12. Visiem vaicājumu izveides un tulkošanas rīkiem tika izmantotas darba izstrādes laikā jaunākās pieejamās versijas - Ontop versija 4.1.0, Morph versija 3.12.5 un ViziQuer versija 0.2.12. Pirms Docker Compose izmantošanas, ir nepieciešams izveidot Docker iekšējo "bridge" tipa tīklu ar nosaukumu "mag_net" (norādīts Docker Compose konfigurācijas datnē), kas ļauj rīku konteineriem savstarpēji sazināties. To var izdarīt ar komandu "docker network create -d bridge mag_net". Pēc tam ir nepieciešams izveidot virtuālo disku jeb "volume", kuru izmantos ViziQuer rīka konteiners, lai saglabātu izveidotos lietotājus, projektus un vizuālos vaicājumus. Šo disku var izveidot ar komandu "docker volume create --name=vqdata". Kad šie soļi ir izdarīti, tad direktorijā, kurā atrodas Compose konfigurācijas datne, jāizveido mapes "configs" un "ontop". Mapei "ontop" nepieciešamas divas apakšdirektorijas – "input" un "jdbc". Mapē "configs" jāievieto rīkam Morph nepieciešamās datnes – attēlojums, Morph konfigurācijas datne, kā arī datne, kas satur SPARQL vaicājumu. Mapes "ontop" apakšdirektorijā "jdbc" ir jāievieto izmantotās datubāzes dziņa datne, šajā gadījumā – MySQL, bet direktorijā "input" – attēlojums, ontoloģija un Ontop konfigurācijas datne. Kad vide ir sagatavota, visus rīkus var iedarbināt ar komandu "docker-compose -f docker-compose-all.yml up -d". Ja nepieciešams apstādināt un nodzēst rīku Docker konteinerus, to var izdarīt ar komandu "docker-compose -f docker-compose-all.yml down". Abās komandās "docker-compose-all.yml" ir Compose konfigurācijas datnes nosaukums. Papildus šīm darbībām, ja ir nepieciešams piekļūt rīkiem caur "localhost" adresi, nevis Docker virtuālās mašīnas IP adresi, tad VirtualBox konfigurācijā nepieciešams norādīt portu pārvirzīšanu. Darbā izmantotajā konfigurācijā bija jāpārvirza ports 8080, ko izmanto Ontop galapunkts, un ports 8171, ko izmanto ViziQuer.

Kad rīka Ontop Docker konteiners ir gatavs darbam, tad Ontop galapunktam iespējams piekļūt adresē "localhost:8080". Veiksmīgas uzstādīšanas gadījumā, šajā adresē jābūt redzamam ievades laukam, kurā iespējams rakstīt SPARQL vaicājumus, kā arī tos izpildīt. Docker Compose konfigurācijas datnē Ontop konteineram starp norādītajām opcijām ir arī " ONTOP_DEV_MODE: "true"" – šādi iespējams Ontop darbināt izstrādes režīmā, kas padara pieejamu galapunktu "/ontop/reformulate", kas nevis pārtulko un izpilda SPARQL vaicājumus, bet gan atgriež no SPARQL vaicājuma iegūto SQL vaicājumu. Citas Ontop konfigurācijas

iespējas atrodamas Ontop Docker attēla lietošanas instrukcijā[5].

Lai atvieglotu iepriekš minētā Ontop galapunkta izmantošanu, tika izveidots Python skripts, kas no datnes nolasa SPARQL vaicājumu, to nodod Ontop vaicājumu tulkošanas galapunktam un atgriež tulkošanas rezultātu. Skripts kopā ar darbināšanas instrukciju ir publicēts vietnē GitHub[6] un ir brīvi pieejams.

Pirmo reizi atverot ViziQuer rīka saskarni, ir nepieciešams izveidot lietotāju. Kad tas ir izdarīts, ir jāizveido jauns projekts un jāizmanto iespēja "Upload project", lai augšupielādētu piemēra projekta konfigurācijas datni, kas iegūstama no jau iepriekš pieminētās ViziQuer rīka mājaslapas. Pēc šīs augšupielādes ir nepieciešams pielāgot projekta iestatījumus, kā SPARQL galapunktu norādot "http://localhost:8080/sparql", nodzēšot "Named graph" lauka vērtību, kā arī cilnē "Extra" atzīmējot iespēju "Simple condition implementation". Papildus šiem soļiem, lai ViziQuer varētu veiksmīgi veidot un izpildīt vizuālos vaicājumus, ir nepieciešams augšupielādēt shēmas JSON datni, izmantojot opciju "Import VQ Schema". Darba izstrādes gaitā tika mēģināts šo shēmas datni iegūt, izmantojot LU MII izstrādāto rīku OBIS-SchemaExtractor[7], tomēr rīku darbinot lokāli un tajā norādot uzstādīto Ontop galapunktu, neizdevās iegūt vēlamo shēmu ar pareizām apakšklasēm un klašu saitēm. Tā vietā tika izmantota shēmas datne, kas atrodama ViziQuer mājaslapā.

Rīka Morph konfigurācija ir nedaudz sarežģītāka – pirms Compose vides darbināšanas ir nepieciešams uzbūvēt Docker attēlu no rīka pirmkoda. To iespējams izdarīt no GitHub repozitorija lejupielādējot rīka pirmkodu un no pirmkoda mapes darbinot komandu "docker build -t morph-rdb .". Kad tas ir izdarīts, kā arī Docker Compose vide ir strādājoša, tad Morph rīku iespējams darbināt ar komandu "docker exec -it morph-rdb bash run-docker.sh morph-properties.properties", kur "morph-properties.properties" ir rīka konfigurācijas datnes nosaukums. Pēc komandas izpildes, Morph komandrindā izvadīs darbības paziņojumus, t.sk. arī iegūto SQL vaicājumu un konfigurācijas datnē definētā izvades datnē ierakstīs vaicājuma rezultātu.

[5] https://hub.docker.com/r/ontop/ontop-endpoint
[6] https://github.com/ralfs-sedlenieks/sparql-reformulation-helper
[7] https://github.com/LUMII-Syslab/obis-schemaextractor

5. Vaicājumu izpilde un novērtēšana

Šajā nodaļā aprakstīti ar Ontop un ViziQuer veiktie eksperimenti un novērtēti to rezultātā iegūtie vaicājumi, kā arī apskatīti ar Morph rīku iegūtie rezultāti.

5.1. Vaicājumu izpilde ar ViziQuer un Ontop

Lai novērtētu vaicājumus, kas iegūstami no uzstādītās infrastruktūras ar izlaboto attēlojumu, visi 12 vaicājumi tika izpildīti trīs piegājienos – pirmajā reizē tika izmantots izlabotais attēlojums ar sākotnējo ontoloģiju un bez izmaiņām vizuālajos vaicājumos, izņemot gadījumus, kad bija nepieciešamas minimālas izmaiņas, lai vaicājums nonāktu līdz Ontop rīkam. Otrajā piegājienā vaicājumos tika izņemtas OPTIONAL konstrukcijas un to vietā visi atribūti norādīti kā obligāti. Izņēmums ir 5. vaicājums, kurā tika atstāta viena no sākotnējām OPTIONAL konstrukcijām, bez kuras vaicājums būtu identisks 4. vaicājumam. Katrā izpildes piegājienā novērotās vispārīgās izmaiņas tiks pieminētas rezultātu sadaļā, bet šajā nodaļā tiks attēloti tikai vaicājumu pēdējā piegājiena vaicājumi, jo šajā piegājienā ir sagaidāmi visoptimālākie SQL vaicājumi.

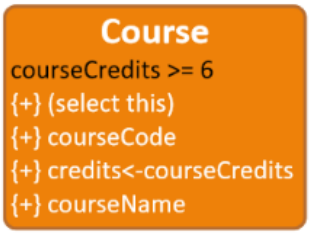
Šīs nodaļas turpmākajā daļā katra vaicājuma vizuālā, SPARQL un SQL forma tiks attēlota tabulās, papildus iekļaujot arī rindas, kas satur optimizētu tulkotā vaicājuma variantu ar samazinātu IS NOT NULL pārbaužu skaitu, kā arī manuāli rakstītu ekvivalentu SQL vaicājumu. Manuāli rakstītajos SQL vaicājumos tiks saglabāti tulkojumos izmantotie lauku un apakšvaicājumu nosaukumi (alias), lai tos būtu vieglāk salīdzināt ar tulkotajiem vaicājumiem. Lai uzlabotu rezultātu pārskatāmību, tabulās tiks apskatīts tikai katra vizuālā vaicājuma SPARQL galīgais variants, kurā atmestas OPTIONAL konstrukcijas, kā arī izmantota vienkāršotā ontoloģija un attēlojums. Katra vizuālā vaicājuma izskaidrojums atrodams ViziQuer lietošanas instrukcijā, kas pieejama caur ViziQuer rīka mājaslapu[8]. Apakšnodaļas beigās atrodama arī tabula, kurā salīdzinātas Ontop tulkoto, optimizēto, manuāli rakstīto un maģistra darbā [18] iegūto SQL vaicājumu sarežģītības. Kā *ad hoc* sarežģītības mērs šajā darbā tika izmantots SQL atslēgvārdu skaits katrā vaicājumā. Katrs vaicājums tika analizēts ar autora veidotu Python skriptu, kas atslēgvārdu skaitīšanu veica, izmantojot atslēgvārdu sarakstu. Skripts ir publicēts vietnē GitHub[9]. Lai gan šādam sarežģītības mēram ir trūkumi, piemēram, tas neskaita apakšvaicājumu nosaukumus, un skaita katru atslēgvārdu atsevišķi, tomēr tas ļauj vismaz aptuveni salīdzināt katra vaicājuma dažādo variantu sarežģītības. Jāpiebilst, ka programmas darbam vaicājumi bija jāsagatavo, ar atstarpēm atdalot atslēgvārdus no iekavām

[8] https://viziquer.lumii.lv/
[9] https://github.com/ralfs-sedlenieks/sql-keyword-counter-simple

un citiem simboliem.

5.1. tabula

Pirmā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

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

SQL	SELECT v1.`Code` AS `Code1m25`, v1.`Course_ID` AS `Course_ID1m25`, v1.`Credits` AS `Credits1m20`, v1.`Name` AS `Name1m17` FROM `xcourse` v1 WHERE ((v1.`Credits` >= 6) AND v1.`Code` IS NOT NULL AND v1.`Credits` IS NOT NULL AND v1.`Name` IS NOT NULL)
Optimizēts SQL	SELECT v1.`Code` AS `Code1m25`, v1.`Course_ID` AS `Course_ID1m25`, v1.`Credits` AS `Credits1m20`, v1.`Name` AS `Name1m17` FROM `xcourse` v1 WHERE v1.`Credits` >= 6;
Manuāli veidots SQL	Sakrīt ar optimizēto SQL vaicājumu

5.1. tabulā redzami pirmā vaicājuma tulkošanas un optimizācijas rezultāti. Kā redzams, vaicājuma variantā, kurā tika atmestas OPTIONAL konstrukcijas, kā arī izmantota vienkāršota ontoloģija, Ontop ģenerētais vaicājums ir tuvu optimālam. Vienīgais trūkums ir liekās IS NOT NULL pārbaudes un, manuāli izpildot papildu optimizāciju, kurā šīs pārbaudes tiek noņemtas, iegūtais optimizētais variants sakrīt ar manuāli rakstītu vaicājumu. Izmantojot sākotnējo ontoloģiju un atmetot OPTIONAL konstrukcijas, šajā vaicājumā jūtami uzlabojumi netika novēroti, bet pēc ontoloģijas vienkāršošanas, tulkotā SQL vaicājuma WHERE nosacījumi kļuva optimālāki, jo tika atmestas papildu pārbaudes "XCourse" tabulas laukam "Required", kas

sākotnējā ontoloģijā un attēlojumā tika izmantots, lai atšķirtu obligātos un neobligātos kursus. Būtisku uzlabojumu, salīdzinot ar iepriekšējā gada darbā [18] minētajiem rezultātiem, sniedza arī jaunākajā ViziQuer rīka versijā pieejamais projekta iestatījums "Simple condition implementation", kas tika pieminēts nodaļā 4.3. Izmantojot šo iestatījumu, ViziQuer šo vaicājumu spēja ģenerēt bez EXISTS nosacījuma, kuru Ontop rīks vēl joprojām neatbalsta.

5.2. tabula

Otrā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT (COUNT(?Course) AS ?Course_COUNT) WHERE{
	?Course a :Course.
	}

SQL	SELECT COUNT(*) AS `v0` FROM (SELECT DISTINCT v5.`Course_ID0m34` AS `Course_ID0m34` FROM (SELECT v1.`Course_ID` AS `Course_ID0m34` FROM `xcourse` v1 UNION ALL SELECT v3.`Course_ID` AS `Course_ID0m34` FROM `registration` v3 WHERE v3.`Course_ID` IS NOT NULL) v5) v7
Optimizēts SQL	SELECT COUNT(*) AS `v0` FROM (SELECT DISTINCT v5.`Course_ID0m34` AS `Course_ID0m34` FROM (SELECT v1.`Course_ID` AS `Course_ID0m34` FROM `xcourse` v1 UNION ALL SELECT v3.`Course_ID` AS `Course_ID0m34` FROM `registration` v3) v5) v7;
Manuāli veidots SQL	SELECT COUNT(*) AS `v0` FROM `xcourse` v1;

5.2. tabulā redzami otrā vaicājuma tulkošanas un optimizācijas rezultāti. Šajā vaicājumā OPTIONAL konstrukciju atmešana neko nemainīja, jo jau sākotnējais vaicājums nesaturēja šādas konstrukcijas. Vienkāršojot ontoloģiju un attēlojumu, bija novērojams būtisks tulkotā vaicājuma kvalitātes uzlabojums – pirms ontoloģijas vienkāršošanas, tulkotais vaicājums

saturēja vairākas UNION ALL operācijas, kā arī atkārtotas "XCourse" tabulas lauka "Required" vērtību pārbaudes. Tomēr arī ar vienkāršoto ontoloģiju iegūtais SQL vaicājums nav optimāls, jo satur nevajadzīgus apakšvaicājumus un UNION ALL operāciju, kā arī liekas IS NOT NULL pārbaudes. Veicot vaicājuma optimizāciju, noņemot liekās pārbaudes, iegūtais vaicājums vēl joprojām ir sarežģītāks par manuāli rakstīto.

5.3. tabula

Trešā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

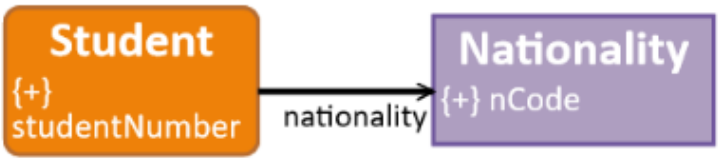
SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?courseCredits (COUNT(?Course) AS ?Course_COUNT)
	WHERE{
	?Course a :Course.
	?Course :courseCredits ?courseCredits.
	}
	GROUP BY ?courseCredits

SQL	SELECT v1.`Credits` AS `Credits1m20`, COUNT(*) AS `v0` FROM `xcourse` v1 WHERE v1.`Credits` IS NOT NULL GROUP BY v1.`Credits`
Optimizēts SQL	SELECT v1.`Credits` AS `Credits1m20`, COUNT(*) AS `v0` FROM `xcourse` v1 GROUP BY v1.`Credits`;
Manuāli veidots SQL	Sakrīt ar optimizēto vaicājumu

5.3. tabulā redzami trešā vaicājuma vizualizācijas un tulkošanas rezultāti. Katrā no piegājieniem, kad šis vaicājums tika izpildīts, tika iegūts aizvien optimālāks tulkotais SQL vaicājums. Sākotnējais vaicājums saturēja vairākas UNION ALL operācijas, kā arī LEFT OUTER JOIN savienojumu. Atmetot OPTIONAL konstrukcijas, tulkotajā SQL vairs netika iekļauta neviena no šīm operācijām. Izmantojot vienkāršoto ontoloģiju, tulkotais SQL vaicājums tika optimizēts vēl tālāk, jo WHERE nosacījumā vairs netika iekļautas "Required" lauka vērtību pārbaudes. Līdzīgi kā pirmajā vaicājumā, vienīgais trūkums šī vaicājuma galīgajā variantā ir lieka IS NOT NULL pārbaude. Papildu optimizācijas procesā atmetot šo pārbaudi, iegūtais vaicājums sakrīt ar manuāli rakstītu SQL vaicājumu.

5.4. tabula

Ceturtā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?studentNumber ?nCode WHERE{
	?Student a :Student.
	?Nationality a :Nationality.
	?Nationality :nCode ?nCode.
	?Student :studentNumber ?studentNumber.
	?Student :nationality ?Nationality.
	}

SQL	SELECT v1.`Nat_Code` AS `Code1m24`, v1.`StudNumber` AS `StudNumber1m10` FROM `xstudent` v1 WHERE (v1.`StudNumber` IS NOT NULL AND v1.`Nat_Code` IS NOT NULL)
Optimizēts SQL	SELECT v1.`Nat_Code` AS `Code1m24`, v1.`StudNumber` AS `StudNumber1m10` FROM `xstudent` v1 WHERE v1.`Nat_Code` IS NOT NULL;
Manuāli veidots SQL	Sakrīt ar optimizēto vaicājumu

5.4. tabulā redzami ceturtā vaicājuma tulkošanas rezultāti. Līdzīgi kā trešajā vaicājumā, ceturtā vaicājuma sākotnējā forma ar OPTIONAL konstrukcijām tika pārtulkota par SQL vaicājumu ar vairākām UNION ALL un LEFT OUTER JOIN operācijām, savukārt atmetot OPTIONAL konstrukcijas, tulkotajā vaicājumā šīs operācijas vairs neparādījās. Atšķirībā no trešā vaicājuma, ceturtā vaicājuma tulkojums neizmainījās pēc ontoloģijas vienkāršošanas. Vienīgais galīgā tulkojuma trūkums ir liekā "StudNumber" kolonnas IS NOT NULL pārbaude. Tautības koda laukam "Nat_Code" šī pārbaude ir nepieciešama, jo bez tās vaicājuma rezultātā tiktu iekļauta papildus rinda ar studentu, kuram nav norādīta tautība. Tāpat kā trešajā vaicājumā, arī ceturtā vaicājuma optimizētais variants sakrīt ar manuāli rakstīto.

5.5. tabula

Piektā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?studentNumber ?nCode WHERE{
	?Student a :Student.
	OPTIONAL{
	?Nationality a :Nationality.
	?Nationality :nCode ?nCode.
	?Student :nationality ?Nationality.
	}
	?Student :studentNumber ?studentNumber.
	}

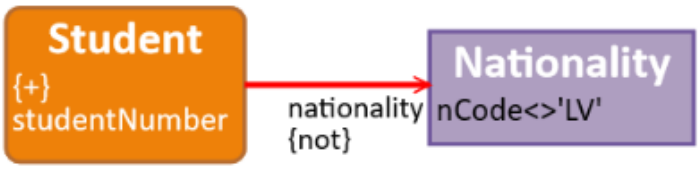
SQL	SELECT v1.`Nat_Code` AS `Code1m0`, v1.`StudNumber` AS `StudNumber1m10`, v4.`v1` AS `v1` FROM (`xstudent` v1 LEFT OUTER JOIN (SELECT v2.`Code` AS `Code1m1`, CASE WHEN v2.`Code` IS NOT NULL THEN 'placeholder1' ELSE 'placeholder2' END AS `v1` FROM `nationality` v2) v4 ON v1.`Nat_Code` = v4.`Code1m1`) WHERE v1.`StudNumber` IS NOT NULL
Optimizēts SQL	SELECT v1.`Nat_Code` AS `Code1m0`, v1.`StudNumber` AS `StudNumber1m10` FROM (`xstudent` v1 LEFT OUTER JOIN (SELECT v2.`Code` AS `Code1m1` FROM `nationality` v2) v4 ON v1.`Nat_Code` = v4.`Code1m1`);
Manuāli veidots SQL	SELECT v1.`Nat_Code` AS `Code1m0`, v1.`StudNumber` AS `StudNumber1m10` FROM `xstudent` v1;

5.5. tabulā redzami piektā vaicājuma tulkošanas rezultāti. Tā kā šajā vizuālajā vaicājumā izmantota neobligāta attiecība, kas ViziQuer rīkā attēlota ar zilu pārtrauktu bultu, tad arī galīgajā vaicājuma versijā tika atstāta šai attiecībai atbilstošā OPTIONAL konstrukcija, jo bez šīs konstrukcijas vaicājums būtu identisks iepriekšējam vaicājumam. Šī vaicājuma sākotnējā varianta tulkojums uz SQL saturēja vairākus apakšvaicājumus, UNION ALL operāciju, kā arī vairākus LEFT OUTER JOIN savienojumus. Atmetot visas iespējamās OPTIONAL konstrukcijas, no iepriekš minētajiem elementiem vaicājuma tulkojumā saglabājās apakšvaicājums un viena no LEFT OUTER JOIN operācijām. Papildus tam, tulkotā vaicājuma rezultātā bija atrodama lieka kolonna “v1”, kas saturēja vērtības, kuras ir atkarīgas no tautības

atribūta “Code” vērtības esamības. Izmainītā vaicājuma tulkošanā izmantojot vienkāršoto ontoloģiju, netika novērotas izmaiņas iegūtajā SQL vaicājumā. Vaicājuma papildu optimizācijas procesā tika atmesta iepriekš minētā kolonna “v1”, jo tās saturs bija atkarīgs no IS NOT NULL pārbaudes, kas liek domāt, ka šīs kolonnas un citos vaicājumos redzamo IS NOT NULL pārbaužu cēlonis ir viens un tas pats. Lai gan pēc šīs papildu optimizācijas vaicājums tik tiešām kļuva vienkāršāks, tas vēl joprojām saturēja lieku LEFT OUTER JOIN operāciju un apakšvaicājumu, kas neparādās manuāli rakstītajā vaicājumā.

5.6. tabula

Sestā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?studentNumber WHERE{
	?Student a :Student.
	FILTER NOT EXISTS{
	?Nationality a :Nationality.
	?Nationality :nCode ?nCode.
	FILTER(STR(?nCode) != 'LV')
	?Student :nationality ?Nationality.
	}
	?Student :studentNumber ?studentNumber.
	}

SQL	Tulkojums netika iegūts, jo Ontop neatbalsta NOT EXISTS konstrukciju

Alternatīvs SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?studentNumber WHERE{
	?Student a :Student.
	OPTIONAL{
	?Nationality a :Nationality.
	?Nationality :nCode ?nCode.
	FILTER(STR(?nCode) != 'LV')
	?Student :nationality ?Nationality.
	}
	FILTER(!BOUND(?Nationality))
	?Student :studentNumber ?studentNumber.
	}

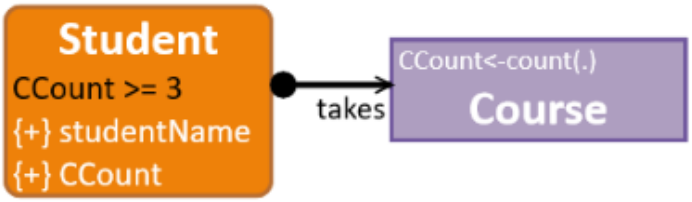
Alternatīvā vaicājuma SQL tulkojums	SELECT v1.`StudNumber` AS `StudNumber1m10` FROM (`xstudent` v1 LEFT OUTER JOIN (SELECT v2.`Code` AS `Code1m1`, CASE WHEN v2.`Code` IS NOT NULL THEN 'placeholder1' ELSE 'placeholder2' END AS `v1` FROM `nationality` v2

	WHERE v2.`Code` <> 'LV') v4 ON v1.`Nat_Code` = v4.`Code1m1`) WHERE ((v4.`v1` IS NULL OR v1.`Nat_Code` IS NULL) AND v1.`StudNumber` IS NOT NULL)
Optimizēts SQL	SELECT v1.`StudNumber` AS `StudNumber1m10` FROM (`xstudent` v1 LEFT OUTER JOIN (SELECT v2.`Code` AS `Code1m1` FROM `nationality` v2 WHERE v2.`Code` <> 'LV') v4 ON v1.`Nat_Code` = v4.`Code1m1`) WHERE (v4.`Code1m1` IS NULL);
Manuāli veidots SQL	SELECT v1.`StudNumber` AS `StudNumber1m10`, v1.`Nat_Code` FROM `xstudent` v1 WHERE v1.`Nat_Code` = 'LV' OR v1.`Nat_Code` IS NULL;

5.6. tabulā redzami sestā vaicājuma tulkošanas rezultāti. Kā redzams, šis vaicājums satur negācijas saiti, kas apzīmēta ar sarkanu bultu. Pat ar iepriekš minēto ViziQuer iestatījumu "Simple condition implementation", šajā vaicājumā ViziQuer izveido SPARQL vaicājumu, kas satur NOT EXISTS konstrukciju, kas netiek atbalstīta Ontop rīkā, tādēļ sākotnējam vaicājuma variantam netika iegūts SQL tulkojums. Lai iegūtu kādu rezultātu, ko būtu iespējams novērtēt, tika izveidots alternatīvs SPARQL vaicājums (rinda "Alternatīvs SPARQL"), kas ir ekvivalents sākotnējam vaicājumam, bet NOT EXISTS konstrukciju aizvieto ar OPTIONAL un !BOUND konstrukcijām. Alternatīvā SPARQL vaicājuma tulkojumā, līdzīgi piektā vaicājuma tulkojumā, parādās lieka Ontop pievienota kolonna, kas faktiski satur IS NOT NULL pārbaudes vērtības. Atmetot liekās IS NOT NULL pārbaudes, iegūtais optimizētais vaicājums tik un tā nav līdzvērtīgs manuāli rakstītajam SQL. Kā iespējamu cēloni šādam rezultātam var minēt to, ka Ontop rīks nespēj atpazīt dubulto noliegumu, kas atrodams šajā vaicājumā, proti, ir jāatlasa studenti, kuriem nav tautības, kurām kods nav vienāds ar "LV".

5.7. tabula

Septītā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?studentName ?CCount WHERE{
	?Student a :Student.
	{SELECT ?Student (COUNT(?Course) AS ?CCount) WHERE{

	?Course a :Course. ?Student :takes ?Course.} GROUP BY ?Student } ?Student :studentName ?studentName. FILTER(?CCount >= 3) FILTER(BOUND(?CCount)) }
SQL	SELECT v13.`Name1m14` AS `Name1m14`, v13.`v0` AS `v0` FROM (SELECT DISTINCT v6.`Name1m14` AS `Name1m14`, v6.`Student_ID0m30` AS `Student_ID0m30`, v11.`v0` AS `v0` FROM (SELECT v1.`Name` AS `Name1m14`, v1.`Student_ID` AS `Student_ID0m30` FROM `xstudent` v1 WHERE v1.`Name` IS NOT NULL UNION ALL SELECT v4.`Name` AS `Name1m14`, v3.`Student_ID` AS `Student_ID0m30` FROM `registration` v3, `xstudent` v4 WHERE (v4.`Name` IS NOT NULL AND v3.`Student_ID` = v4.`Student_ID`)) v6, (SELECT v9.`Student_ID0m0` AS `Student_ID0m0`, COUNT(*) AS `v0` FROM (SELECT DISTINCT v7.`Course_ID` AS `Course_ID1m2`, v7.`Student_ID` AS `Student_ID0m0` FROM `registration` v7 WHERE (v7.`Student_ID` IS NOT NULL AND v7.`Course_ID` IS NOT NULL AND v7.`DatePaid` IS NOT NULL)) v9 GROUP BY v9.`Student_ID0m0`) v11 WHERE (v11.`v0` IS NOT NULL AND (v11.`v0` >= 3) AND v6.`Student_ID0m30` = v11.`Student_ID0m0`)) v13
Optimizēts SQL	SELECT v13.`Name1m14` AS `Name1m14`, v13.`v0` AS `v0` FROM (SELECT DISTINCT v6.`Name1m14` AS `Name1m14`, v6.`Student_ID0m30` AS `Student_ID0m30`, v11.`v0` AS `v0` FROM (SELECT v1.`Name` AS `Name1m14`, v1.`Student_ID` AS `Student_ID0m30` FROM `xstudent` v1 UNION ALL SELECT v4.`Name` AS `Name1m14`, v3.`Student_ID` AS `Student_ID0m30` FROM `registration` v3, `xstudent` v4 WHERE (v3.`Student_ID` = v4.`Student_ID`)

	) v6, (SELECT v9.`Student_ID0m0` AS `Student_ID0m0`, COUNT(*) AS `v0` FROM (SELECT DISTINCT v7.`Course_ID` AS `Course_ID1m2`, v7.`Student_ID` AS `Student_ID0m0` FROM `registration` v7) v9 GROUP BY v9.`Student_ID0m0`) v11 WHERE ((v11.`v0` >= 3) AND v6.`Student_ID0m30` = v11.`Student_ID0m0`)) v13;
Manuāli veidots SQL	SELECT v1.`Name` AS `Name1m14`, v13.`v0` AS `v0` FROM `xstudent` v1 JOIN (SELECT v7.`Student_ID` AS `Student_ID0m0`, COUNT(*) AS `v0` FROM `registration` v7 WHERE v7.`Student_ID` IS NOT NULL AND v7.`Course_ID` IS NOT NULL GROUP BY v7.`Student_ID`) v13 ON v13.`v0` >= 3 AND v1.`Student_ID`= v13.`Student_ID0m0`;

5.7. tabulā redzams septītā vaicājuma tulkošanas rezultāts. Šī vaicājuma sākotnējā varianta tulkojums saturēja CASE, UNION ALL, JOIN un LEFT OUTER JOIN konstrukcijas. Pārveidojot vizuālo vaicājumu, lai ģenerētajā SPARQL vaicājumā nebūtu OPTIONAL konstrukciju, tulkojumā no iepriekš minētajām konstrukcijām saglabājās tikai UNION ALL. Veicot ontoloģijas vienkāršošanu, tālākus uzlabojumus tulkojumā neizdevās iegūt. Izpildot papildu optimizāciju, bija iespējams atmest vairākas IS NOT NULL pārbaudes, kā arī pat atsevišķas WHERE konstrukcijas, kuras saturēja tikai atmestās pārbaudes. Salīdzinot optimizēto vaicājumu ar manuāli rakstītu SQL vaicājumu, ir redzams, ka tulkojumā atrodamos vairāku līmeņu apakšvaicājumus ir iespējams aizvietot ar vienu apakšvaicājumu un vienu JOIN operāciju. Svarīgi piebilst, ka sākotnējā vaicājumā atrodamais DISTINCT operators ir lieks un šī operatora dēļ tulkotais vaicājums atgriež nekorektu rezultātu, jo tajā netiek ieskaitītas kursu reģistrācijas, kurām sakrīt studenta ID un kursa ID, bet ir atšķirīga apmaksas datuma ("DatePaid") lauka vērtība.

5.8. tabula

Astotā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

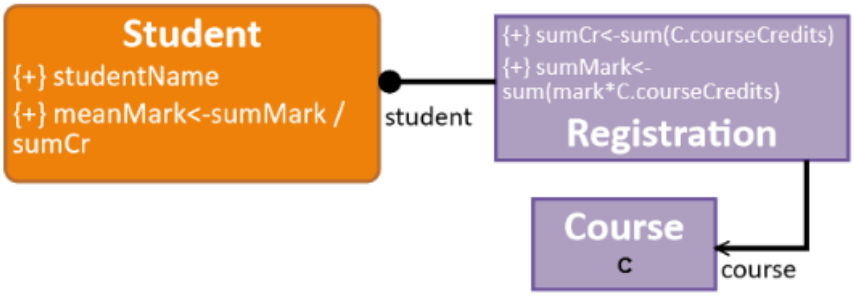
SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?Y (COUNT(?Registration) AS ?C) WHERE{
	?Registration a :Registration.
	OPTIONAL{
	?Registration a :Registration.
	?Registration :dateCompleted ?dateCompleted_1.
	BIND(YEAR(?dateCompleted_1) AS ?Y)
	FILTER(BOUND(?Y))
	}
	}
	GROUP BY ?Y
	ORDER BY DESC(?C)

SQL	SELECT v3.`v1` AS `v1`, v3.`v2` AS `v2`, COUNT(*) AS `v3` FROM (SELECT EXTRACT(YEAR FROM v1.`DateCompleted`) AS `v1`, v1.`DateCompleted` IS NOT NULL AS `v2` FROM `registration` v1) v3 GROUP BY v3.`v1`, v3.`v2` ORDER BY COUNT(*) DESC
Optimizēts SQL	SELECT v3.`v1` AS `v1`, COUNT(*) AS `v3` FROM (SELECT EXTRACT(YEAR FROM v1.`DateCompleted`) AS `v1` FROM `registration` v1) v3 GROUP BY v3.`v1` ORDER BY COUNT(*) DESC;
Manuāli veidots SQL	Sakrīt ar optimizēto vaicājumu

5.8. tabulā redzams astotā vaicājuma tulkošanas rezultāts. Šī vaicājuma SPARQL formā, ko ģenerēja ViziQuer, sākotnēji bija atrodama mainīgā "?dateCompleted_1" pārveidošana uz "xsd:dateTime" tipu, kas izraisīja kļūdu, kad vaicājums tika nodots Ontop rīkam. Lai iegūtu vaicājuma tulkojumu un rezultātu, bija nepieciešams šo pārveidošanu noņemt manuāli. Pēc šīs kļūdas izlabošanas sākotnēji iegūtie SQL tulkojumi saturēja apakšvaicājumus vairākos līmeņos, UNION ALL operācijas, kā arī CASE konstrukciju un LEFT OUTER JOIN savienojumu. No vaicājuma izņemot OPTIONAL konstrukcijas, iegūtais tulkojums bija identisks un uzlabojumi tulkojuma kvalitāte tika novēroti tikai pēc ontoloģijas un attēlojuma vienkāršošanas, kā rezultātā tika iegūts tabulā redzamais tulkojums. Optimizējot iegūto tulkojumu, tika atmesta apakšvaicājuma kolonna "v2", kas saturēja patiesumvērtības tam, vai "DateCompleted" laukā ir NULL vērtība. Līdzīga situācija bija redzama jau apskatītajā piektajā vaicājumā. Optimizācijā iegūtais vaicājums sakrita ar manuāli rakstītu vaicājumu.

5.9. tabula

Devītā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?studentName ?meanMark WHERE{
	?Student a :Student.
	{SELECT ?Student (SUM(?courseCredits_C) AS ?sumCr)
	(SUM(?mark_1*?courseCredits_C) AS ?sumMark) WHERE{
	?Registration a :Registration.
	?C a :Course.
	?C :courseCredits ?courseCredits_C.
	?Registration :mark ?mark_1.
	?Registration :student ?Student.
	?Registration :course ?C.}
	GROUP BY ?Student
	}
	?Student :studentName ?studentName.
	BIND(?sumMark/?sumCr AS ?meanMark)
	FILTER(BOUND(?meanMark))
	}

SQL	SELECT v14.`Name1m14` AS `Name1m14`, v14.`sum0` AS `sum0`, CASE WHEN (NOT (v14.`sum0` = 0.0)) THEN CAST((CAST(v14.`sum1` AS DECIMAL(60,30)) / CAST(v14.`sum0` AS DECIMAL(60,30))) AS CHAR CHARACTER SET utf8) ELSE NULL END AS `v0` FROM (SELECT DISTINCT v6.`Name1m14` AS `Name1m14`, v6.`Student_ID0m30` AS `Student_ID0m30`, v12.`sum0` AS `sum0`, v12.`sum1` AS `sum1` FROM (SELECT v1.`Name` AS `Name1m14`, v1.`Student_ID` AS `Student_ID0m30` FROM `xstudent` v1 WHERE v1.`Name` IS NOT NULL UNION ALL SELECT v4.`Name` AS `Name1m14`, v3.`Student_ID` AS `Student_ID0m30` FROM `registration` v3, `xstudent` v4 WHERE (v4.`Name` IS NOT NULL AND v3.`Student_ID` = v4.`Student_ID`)) v6, (SELECT v9.`Student_ID` AS `Student_ID0m1`,

	SUM(v7.`Credits`) AS `sum0`, SUM((v8.`MarkReceived` * v7.`Credits`)) AS `sum1` FROM `xcourse` v7, `registration` v8, `registration` v9, `registration` v10 WHERE (v7.`Credits` IS NOT NULL AND v8.`MarkReceived` IS NOT NULL AND v8.`DateCompleted` IS NOT NULL AND v9.`Student_ID` IS NOT NULL AND v8.`Registration_ID` = v9.`Registration_ID` AND v8.`Registration_ID` = v10.`Registration_ID` AND v7.`Course_ID` = v10.`Course_ID`) GROUP BY v9.`Student_ID`) v12 WHERE ((NOT (v12.`sum0` = 0.0)) AND v12.`sum0` IS NOT NULL AND v12.`sum1` IS NOT NULL AND v6.`Student_ID0m30` = v12.`Student_ID0m1`)) v14
Optimizēts SQL	SELECT v14.`Name1m14` AS `Name1m14`, v14.`sum0` AS `sum0`, CASE WHEN (NOT (v14.`sum0` = 0.0)) THEN CAST((CAST(v14.`sum1` AS DECIMAL(60,30)) / CAST(v14.`sum0` AS DECIMAL(60,30))) AS CHAR CHARACTER SET utf8) ELSE NULL END AS `v0` FROM (SELECT DISTINCT v6.`Name1m14` AS `Name1m14`, v6.`Student_ID0m30` AS `Student_ID0m30`, v12.`sum0` AS `sum0`, v12.`sum1` AS `sum1` FROM (SELECT v1.`Name` AS `Name1m14`, v1.`Student_ID` AS `Student_ID0m30` FROM `xstudent` v1 UNION ALL SELECT v4.`Name` AS `Name1m14`, v3.`Student_ID` AS `Student_ID0m30` FROM `registration` v3, `xstudent` v4 WHERE (v3.`Student_ID` = v4.`Student_ID`)) v6, (SELECT v9.`Student_ID` AS `Student_ID0m1`, SUM(v7.`Credits`) AS `sum0`, SUM((v8.`MarkReceived` * v7.`Credits`)) AS `sum1` FROM `xcourse` v7, `registration` v8, `registration` v9, `registration` v10 WHERE (v8.`MarkReceived` IS NOT NULL AND v8.`Registration_ID` = v9.`Registration_ID` AND v8.`Registration_ID` = v10.`Registration_ID` AND v7.`Course_ID` = v10.`Course_ID`)

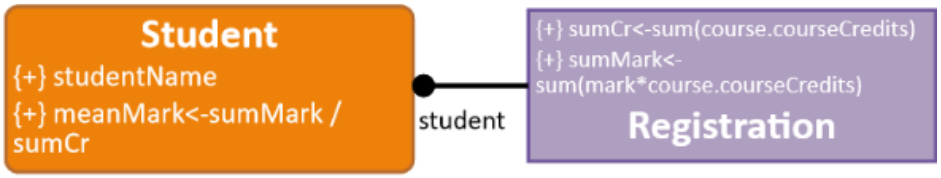
	GROUP BY v9.`Student_ID`) v12 WHERE ((NOT (v12.`sum0` = 0.0)) AND v6.`Student_ID0m30` = v12.`Student_ID0m1`)) v14;
Manuāli veidots SQL	SELECT v14.`Name1m14` AS `Name1m14`, v14.`sum0` AS `sum0`, CAST(v14.`sum1` AS DECIMAL(60,30)) / CAST(v14.`sum0` AS DECIMAL(60,30)) AS `v0` FROM (SELECT v1.`Student_ID` AS `Student_ID0m1`, v1.`Name` AS `Name1m14`, SUM(v3.`Credits`) AS `sum0`, SUM(v2.`MarkReceived` * v3.`Credits`) AS `sum1` FROM `xstudent` v1 JOIN `registration` v2 ON v2.`Student_ID` = v1.`Student_ID` JOIN `xcourse` v3 ON v3.`Course_ID` = v2.`Course_ID` WHERE v2.`MarkReceived` IS NOT NULL GROUP BY v1.`Student_ID`, v1.`Name`) v14;

5.9. tabulā redzams devītā vaicājuma tulkošanas rezultāts. Izmantojot attēlojumu, kas bija atrodams jau iepriekš minētajā maģistra darbā [18], šī vaicājuma rezultātā tika iegūtas gan prasītās vērtības, gan vairākas tukšas rindas. Padziļināta izpēte atklāja, ka šādu rezultātu izraisīja 4.2. nodaļā pieminētā kļūda sākotnējā attēlojumā. Papildus attēlojuma kļūdai, vaicājuma izpildi tā sākotnējā formā traucēja arī mainīgo "?sumMark" un "?sumCr" pārveidošana uz "xsd:decimal" tipu, kā arī ViziQuer rīka kļūda, kuras dēļ šo mainīgo nosaukumiem BIND konstrukcijā tika pievienots piedēklis "_1". Pēc šo problēmu atrisināšanas tika atklāta jauna kļūda, kuru izraisīja fakts, ka SPARQL vaicājumā mainīgo "?mark_1" un "?courseCredits_C" vērtības nebija obligātas, t.i., šīs vērtības bija definētas OPTIONAL konstrukcijā, kā rezultātā Ontop nevarēja izpildīt reizināšanu, kas ar šīm vērtībām tiek veikta vaicājumā. Pēc šo OPTIONAL konstrukciju izņemšanas, vienīgais neobligātais mainīgais bija "?studentName". Kad arī šis mainīgais tika padarīts par obligātu, vaicājuma tulkojumā vairs nebija atrodamas JOIN un LEFT OUTER JOIN operācijas, kā arī vienā no apakšvaicājumiem vairs neparādījās CASE konstrukcija. Atkārtojot vaicājuma tulkošanu ar vienkāršoto ontoloģiju, vienīgais uzlabojums bija redzams vienā no WHERE nosacījumiem, kur vairs netika pārbaudītas lauka "Required" vērtības. Izpildot tulkotā vaicājuma optimizāciju, bija iespējams atmest vairākas IS NOT NULL pārbaudes, tomēr arī šādi iegūtais vaicājums bija ievērojami sarežģītāks nekā manuāli rakstītais. Gan optimizētajā, gan manuāli rakstītajā vaicājumā tika saglabāta IS NOT NULL pārbaude laukam "MarkReceived", jo bez tās tika iegūts nekorekti rezultāti. Tulkotais vaicājums arī pēc optimizācijas satur nevajadzīgi daudz apakšvaicājumu, lieku UNION ALL operāciju, kā arī nevajadzīgu vidējās atzīmes pārveidošanu par teksta lauku.

Papildus šīm nepilnībām, vaicājumā arī tiek vairākas reizes pēc kārtas iekļauta “Registration” tabula.

5.10. tabula

Desmitā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
	SELECT ?studentName ?meanMark WHERE{
	?Student a :Student.
	{SELECT ?Student (SUM(?courseCredits) AS ?sumCr)
	(SUM(?mark_1*?courseCredits) AS ?sumMark) WHERE{
	?Registration a :Registration.
	?Registration :course/:courseCredits ?courseCredits.
	?Registration :mark ?mark_1.
	?Registration :student ?Student.}
	GROUP BY ?Student
	}
	?Student :studentName ?studentName.
	BIND(?sumMark/?sumCr AS ?meanMark)
	FILTER(BOUND(?meanMark))
	}

SQL	SELECT v14.`Name1m14` AS `Name1m14`, v14.`sum0` AS `sum0`, CASE WHEN (NOT (v14.`sum0` = 0.0)) THEN CAST((CAST(v14.`sum1` AS DECIMAL(60,30)) / CAST(v14.`sum0` AS DECIMAL(60,30))) AS CHAR CHARACTER SET utf8) ELSE NULL END AS `v0` FROM (SELECT DISTINCT v6.`Name1m14` AS `Name1m14`, v6.`Student_ID0m30` AS `Student_ID0m30`, v12.`sum0` AS `sum0`, v12.`sum1` AS `sum1` FROM (SELECT v1.`Name` AS `Name1m14`, v1.`Student_ID` AS `Student_ID0m30` FROM `xstudent` v1 WHERE v1.`Name` IS NOT NULL UNION ALL SELECT v4.`Name` AS `Name1m14`, v3.`Student_ID` AS `Student_ID0m30` FROM `registration` v3, `xstudent` v4 WHERE (v4.`Name` IS NOT NULL AND v3.`Student_ID` = v4.`Student_ID`)) v6, (SELECT v10.`Student_ID` AS `Student_ID0m1`,

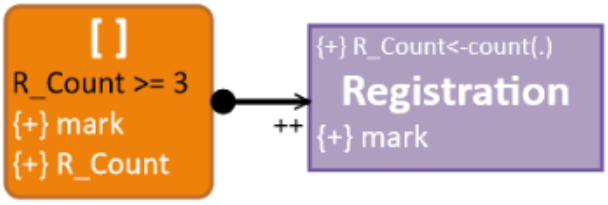
	SUM(v8.`Credits`) AS `sum0`, SUM((v9.`MarkReceived` * v8.`Credits`)) AS `sum1` FROM `registration` v7, `xcourse` v8, `registration` v9, `registration` v10 WHERE (v8.`Credits` IS NOT NULL AND v9.`MarkReceived` IS NOT NULL AND v9.`DateCompleted` IS NOT NULL AND v10.`Student_ID` IS NOT NULL AND v7.`Course_ID` = v8.`Course_ID` AND v7.`Registration_ID` = v9.`Registration_ID` AND v7.`Registration_ID` = v10.`Registration_ID`) GROUP BY v10.`Student_ID`) v12 WHERE ((NOT (v12.`sum0` = 0.0)) AND v12.`sum0` IS NOT NULL AND v12.`sum1` IS NOT NULL AND v6.`Student_ID0m30` = v12.`Student_ID0m1`)) v14
Optimizēts SQL	SELECT v14.`Name1m14` AS `Name1m14`, v14.`sum0` AS `sum0`, CASE WHEN (NOT (v14.`sum0` = 0.0)) THEN CAST((CAST(v14.`sum1` AS DECIMAL(60,30)) / CAST(v14.`sum0` AS DECIMAL(60,30))) AS CHAR CHARACTER SET utf8) ELSE NULL END AS `v0` FROM (SELECT DISTINCT v6.`Name1m14` AS `Name1m14`, v6.`Student_ID0m30` AS `Student_ID0m30`, v12.`sum0` AS `sum0`, v12.`sum1` AS `sum1` FROM (SELECT v1.`Name` AS `Name1m14`, v1.`Student_ID` AS `Student_ID0m30` FROM `xstudent` v1 UNION ALL SELECT v4.`Name` AS `Name1m14`, v3.`Student_ID` AS `Student_ID0m30` FROM `registration` v3, `xstudent` v4 WHERE v3.`Student_ID` = v4.`Student_ID`) v6, (SELECT v10.`Student_ID` AS `Student_ID0m1`, SUM(v8.`Credits`) AS `sum0`, SUM((v9.`MarkReceived` * v8.`Credits`)) AS `sum1` FROM `registration` v7, `xcourse` v8, `registration` v9, `registration` v10 WHERE (v9.`MarkReceived` IS NOT NULL AND v7.`Course_ID` = v8.`Course_ID` AND v7.`Registration_ID` = v9.`Registration_ID` AND v7.`Registration_ID` = v10.`Registration_ID`) GROUP BY v10.`Student_ID`) v12 WHERE ((NOT (v12.`sum0` = 0.0)) AND v6.`Student_ID0m30` = v12.`Student_ID0m1`)) v14;

Manuāli veidots SQL	Skatīt 5.9. tabulu

5.10. tabulā redzams desmitā vaicājuma tulkošanas rezultāts. Tā kā šis vaicājums atbilst devītajam vaicājumam, bet ir realizēts alternatīvā veidā, tad arī šī vaicājuma izpildē tika sastaptas tās pašas problēmas, kas jau aprakstītas pie devītā vaicājuma. Veicot OPTIONAL konstrukciju atmešanu, kā arī vienkāršojot ontoloģiju, arī šī vaicājuma uzlabojumi bija tādi paši kā devītā vaicājuma gadījumā. Desmitā un devītā vaicājuma tulkojumi uz SQL ir praktiski identiski un vienīgā atšķirība ir dažāda tabulu secība vienā no apakšvaicājumiem, kā rezultātā izmainās tabulām piešķirtie pseidonīmi, kas rada nebūtiskas izmaiņas kolonnu atlasē un WHERE nosacījumos. Arī papildu optimizācijas dotie ieguvumi ir tādi paši kā devītajā vaicājumā, kā arī abiem vaicājumiem sakrīt manuāli rakstītais SQL vaicājums.

5.11. tabula

Vienpadsmitā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?mark ?R_Count WHERE{
	{SELECT ?mark (COUNT(?Registration) AS ?R_Count) WHERE{
	?Registration a :Registration.
	?Registration :mark ?mark.}
	GROUP BY ?mark
	}
	FILTER(?R_Count >= 3)
	FILTER(BOUND(?R_Count))
	}

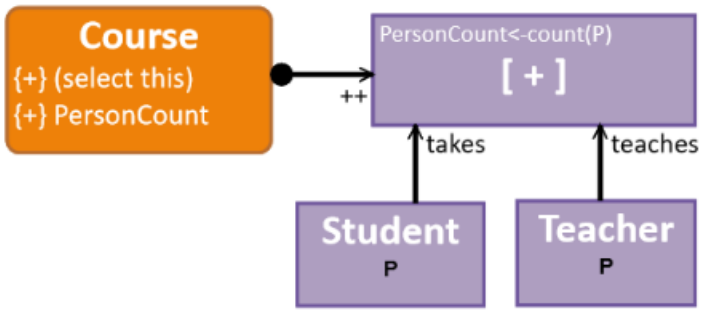
SQL	SELECT v3.`MarkReceived1m21` AS `MarkReceived1m21`, v3.`v0` AS `v0` FROM (SELECT v1.`MarkReceived` AS `MarkReceived1m21`, COUNT(*) AS `v0` FROM `registration` v1 WHERE (v1.`MarkReceived` IS NOT NULL AND v1.`DateCompleted` IS NOT NULL) GROUP BY v1.`MarkReceived`) v3 WHERE (v3.`v0` IS NOT NULL AND (v3.`v0` >= 3))
Optimizēts SQL	SELECT v3.`MarkReceived1m21` AS `MarkReceived1m21`, v3.`v0` AS `v0` FROM (SELECT v1.`MarkReceived` AS `MarkReceived1m21`, COUNT(*) AS `v0`

	FROM `registration` v1 WHERE v1.`MarkReceived` IS NOT NULL GROUP BY v1.`MarkReceived`) v3 WHERE v3.`v0` >= 3;
Manuāli veidots SQL	Sakrīt ar optimizēto vaicājumu

5.11. tabulā redzams vienpadsmitā vaicājuma tulkošanas rezultāts. ViziQuer rīkā no šī vaicājuma vizuālās formas ģenerējot SPARQL, mainīgajam "?R_Count" BOUND konstrukcijā tiek pievienots piedēklis "_1", kas bija manuāli jānoņem, lai iegūtu korektus rezultātus. Šī vaicājuma sākotnējā versija nesaturēja OPTIONAL konstrukcijas un tā tulkojums bija tuvs manuāli rakstītam SQL vaicājumam jau pirms izmaiņu veikšanas. ViziQuer rīkā mainīgos "?mark" un "?R_Count" norādot kā obligātus, vaicājumā parādījās BOUND filtrs, kas nodrošina, ka mainīgajam "?R_Count" būs piešķirta vērtība. Šīs izmaiņas rezultātā, vaicājuma tulkojumā uz SQL parādījās papildus IS NOT NULL pārbaude. Iegūtā vaicājuma optimizācijas rezultātā bija iespējams vienkāršot tajā atrodamos WHERE nosacījumus, tomēr IS NOT NULL pārbaude laukam "MarkReceived" tika atstāta, jo bez tās vaicājuma rezultātā tika iegūta lieka rinda ar tukšu vērtību šajā kolonnā. Tā kā šī vaicājuma vienīgie trūkumi, salīdzinājumā ar manuāli rakstītu vaicājumu, ir liekas IS NOT NULL pārbaudes, tad optimizētais vaicājums sakrita ar manuāli rakstīto.

5.12. tabula

Divpadsmitā ViziQuer vaicājuma vizuālā, SPARQL un SQL forma

ViziQuer	

SPARQL	PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
	SELECT ?Course ?PersonCount WHERE{
	?Course a :Course.
	{SELECT (COUNT(?P) AS ?PersonCount) ?Course WHERE{
	{
	?Course a :Course.
	?P a :Student.
	?P :takes ?Course.
	}
	UNION
	{

	?Course a :Course. ?P a :Teacher. ?P :teaches ?Course. }} GROUP BY ?Course } FILTER(BOUND(?PersonCount)) }
SQL	SELECT DISTINCT v5.`Course_ID0m34` AS `Course_ID0m34`, v14.`v2` AS `v2` FROM (SELECT v1.`Course_ID` AS `Course_ID0m34` FROM `xcourse` v1 UNION ALL SELECT v3.`Course_ID` AS `Course_ID0m34` FROM `registration` v3 WHERE v3.`Course_ID` IS NOT NULL) v5, (SELECT v12.`Course_ID0m0` AS `Course_ID0m0`, COUNT(*) AS `v2` FROM (SELECT v8.`Course_ID0m0` AS `Course_ID0m0`, v8.`Student_ID1m1` AS `Student_ID1m1`, NULL AS `Teacher_ID1m2`, 0 AS `v0` FROM (SELECT DISTINCT v6.`Course_ID` AS `Course_ID0m0`, v6.`Student_ID` AS `Student_ID1m1` FROM `registration` v6 WHERE (v6.`Student_ID` IS NOT NULL AND v6.`Course_ID` IS NOT NULL AND v6.`DatePaid` IS NOT NULL)) v8 UNION ALL SELECT v10.`Course_ID` AS `Course_ID0m0`, NULL AS `Student_ID1m1`, v10.`Teacher_ID` AS `Teacher_ID1m2`, 1 AS `v0` FROM `xcourse` v10 WHERE v10.`Teacher_ID` IS NOT NULL) v12 GROUP BY v12.`Course_ID0m0`) v14 WHERE (v14.`v2` IS NOT NULL AND v5.`Course_ID0m34` = v14.`Course_ID0m0`)
Optimizēts SQL	SELECT DISTINCT v5.`Course_ID0m34` AS `Course_ID0m34`, v14.`v2` AS `v2` FROM (SELECT v1.`Course_ID` AS `Course_ID0m34` FROM `xcourse` v1 UNION ALL SELECT v3.`Course_ID` AS `Course_ID0m34` FROM `registration` v3) v5, (

	SELECT v12.`Course_ID0m0` AS `Course_ID0m0`, COUNT(*) AS `v2` FROM (SELECT v8.`Course_ID0m0` AS `Course_ID0m0`, v8.`Student_ID1m1` AS `Student_ID1m1`, NULL AS `Teacher_ID1m2`, 0 AS `v0` FROM (SELECT DISTINCT v6.`Course_ID` AS `Course_ID0m0`, v6.`Student_ID` AS `Student_ID1m1` FROM `registration` v6) v8 UNION ALL SELECT v10.`Course_ID` AS `Course_ID0m0`, NULL AS `Student_ID1m1`, v10.`Teacher_ID` AS `Teacher_ID1m2`, 1 AS `v0` FROM `xcourse` v10) v12 GROUP BY v12.`Course_ID0m0`) v14 WHERE (v5.`Course_ID0m34` = v14.`Course_ID0m0`);
Manuāli veidots SQL	SELECT DISTINCT v5.`Course_ID0m34` AS `Course_ID0m34`, count(*) AS `v2` FROM (SELECT v3.`Course_ID` AS `Course_ID0m34` FROM `registration` v3 WHERE v3.`Course_ID` IS NOT NULL AND v3.`Student_ID` IS NOT NULL UNION ALL SELECT v4.`Course_ID` AS `Course_ID0m34` FROM `xcourse` v4 WHERE v4.`Teacher_ID` IS NOT NULL) v5 GROUP BY v5.`Course_ID0m34`;

5.12. tabulā redzami divpadsmitā vaicājuma tulkošanas rezultāti. Arī šī vaicājuma sākotnējā variantā nebija atrodamas OPTIONAL konstrukcijas un, norādot "Course" elementā atlasītos atribūtus kā obligātus, ģenerētajā vaicājumā tika pievienots filtrs ar BOUND nosacījumu mainīgajam "?PersonCount". Šis izmaiņas ietekme uz tulkoto vaicājumu bija nebūtiska un visbūtiskākie tulkotā vaicājuma kvalitātes uzlabojumi tika novēroti, izmantojot vienkāršoto ontoloģiju. Ontoloģijas un attēlojuma vienkāršošanas rezultātā tulkotajā vaicājumā būtiski samazinājās UNION ALL operāciju skaits, kā arī tika atmesti vairāki WHERE nosacījumi, kuros tika pārbaudīta lauka "Required" vērtība. Izpildot tulkotā vaicājuma papildu optimizāciju, tajā tika atmestas vairākas liekas IS NOT NULL pārbaudes un atsevišķas WHERE konstrukcijas. Tomēr, salīdzinot šo optimizēto vaicājumu ar manuāli rakstītu vaicājumu, ir redzams, ka to ir iespējams uzlabot vēl tālāk, samazinot apakšvaicājumu un UNION ALL operāciju skaitu. Jāpiebilst arī tas, ka Ontop tulkotais vaicājums deva nekorektu

rezultātu, kurā trīs no pieciem kursiem netika pieskaitīta kāda no iesaistītajām personām, un šī kļūda tika izlabota optimizētajā un manuāli rakstītajā variantā.

Šīs nodaļas sākumā minētā *ad hoc* sarežģītības mēra rezultāti katra vaicājuma dažādajiem variantiem ir redzami tabulā 5.13. Katra vaicājuma rindā mazākā vērtība ir atzīmēta treknrakstā un ar zaļu krāsu, bet lielākā vērtība – treknrakstā un ar sarkanu krāsu. Gadījumos, kad vairākās vienas rindas kolonnās ir vienādas vērtības, visas kolonnas ar vienādajām vērtībām ir atzīmētas atbilstošajā veidā.

5.13. tabula

Tulkoto, optimizēto un manuāli rakstīto SQL vaicājumu sarežģītības

Vaicājums	Ontop tulkojuma sarežģītība	Optimizētā tulkojuma sarežģītība	Manuāli rakstītā vaicājuma sarežģītība	Darbā [18] iegūtā tulkojuma sarežģītība
1	20	**8**	**8**	**40**
2	20	16	**4**	**32**
3	11	**7**	**7**	**72**
4	**12**	**8**	**8**	**8**
5	26	14	**4**	**66**
6	**32**	16	**8**	26
7	**62**	38	**24**	38
8	21	**16**	**16**	**48**
9	97	69	**33**	**120**
10	**97**	69	**33**	45
11	25	**17**	**17**	**26**
12	**67**	43	**28**	34

Kā redzams tabulas rezultātos, visos gadījumos manuāli rakstītie vaicājumi bija ar vismazāko sarežģītību, tomēr 5 no 12 vaicājumiem manuāli rakstītam vaicājumam līdzvērtīgu rezultātu sniedza optimizēts Ontop tulkojums. Ceturtajā vaicājumā manuāli rakstītam vaicājumam līdzvērtīgu sarežģītību sasniedza gan optimizētais Ontop tulkojums, gan maģistra darbā [18] ar vecāku Ontop versiju iegūtais tulkojums. Salīdzinot iepriekš minētajā darbā atrodamos vaicājumu tulkojumus ar šajā darbā iegūtajiem tulkojumiem, ir redzams, ka 7 no 12 vaicājumiem mazāk sarežģīts ir šajā darbā iegūtais tulkojums, kas iegūts ar jaunākajām ViziQuer un Ontop versijām, kā arī ar salabotu attēlojumu un vienkāršotu ontoloģiju. 5 no 12 vaicājumiem tika iegūts pretējs rezultāts.

Apkopojot vaicājumu tulkojumus, kas iegūti, izmantojot ViziQuer un Ontop rīkus, var secināt, ka vairumā gadījumu OPTIONAL konstrukcijas sarežģī Ontop rīka tulkotos

vaicājumus, pievienojot LEFT OUTER JOIN operācijas. Iegūto tulkojumu kvalitāti iespējams uzlabot, Ontop rīkam dodot ontoloģiju, kas ir pēc iespējas vienkāršāka, bet tajā paša laikā ļauj veikt visus nepieciešamos vaicājumus. Lai gan atmetot OPTIONAL konstrukcijas, tulkojumos bija atrodamas vairākas liekas IS NOT NULL pārbaudes, ir iespējams, ka izmantotajā ontoloģijā un attēlojumā nebija pietiekami daudz informācijas, lai Ontop rīks spētu atpazīt laukus, kuriem vienmēr būs netukša vērtība. Izstrādājot optimizāciju, kas automātiski atmestu šīs liekās pārbaudes, gandrīz pusē no vaicājumiem būtu iespējams iegūt rezultātu, kas ir līdzvērtīgs manuāli rakstītam vaicājumam.

5.2. Vaicājumu izpilde ar Morph

Lai papildinātu darbu ar rīku Ontop un Morph tulkoto vaicājumu kvalitātes salīdzinājumu, iepriekš apskatītie 12 vaicājumi no ViziQuer rīka instrukcijas tika izpildīti arī ar rīku Morph, tomēr ar šo rīku izdevās izpildīt tikai pirmos 3 no 12 vaicājumiem. Visos pārējos vaicājumos rīks atgrieza dažādus kļūdu paziņojumus. Jāpiebilst, ka šos rezultātus izdevās iegūt tikai pēc tam, kad no vaicājumiem tika izņemtas OPTIONAL konstrukcijas un kad bija vienkāršota ontoloģija un R2RML attēlojums. Pirms iepriekš apskatīto vaicājumu izpildes ar Morph, tika izmēģināta rīka darbība uz tā kodam pievienoto piemēra datubāzi un attēlojumu. Šajā gadījumā rīka atgrieztie tulkojumi bija optimāli un nebija daudz variantu to vienkāršošanai, tomēr to var izskaidrot ar konkrēto piemēru vienkāršību. Svarīgi piebilst, ka Morph rīkam nav iespējams kā parametru padot ontoloģijas datni, kas nozīmē, ka rīks vaicājumus tulko, balstoties tikai uz attēlojumā pieejamo informāciju. Šis fakts ir redzams arī apskatot ar Morph iegūtos ViziQuer vaicājumu tulkojumus, kas atrodami 5.14. tabulā.

5.14. tabula

Ar rīku Morph iegūtie vaicājumu SQL tulkojumi

Pirmā vaicājuma tulkojums uz SQL	SELECT 439720255 AS "mappingid_courseName", `V_1787`.`Course_ID` AS "Course", `V_1787`.`Name` AS "courseName", 1445222369 AS "mappingid_courseCode", `V_1787`.`Code` AS "courseCode", 1414934182 AS "mappingid_credits", 227100877 AS "mappingid_Course", `V_1787`.`Credits` AS "credits" FROM (SELECT Course_ID, Name, Code, Credits, Required, concat(Code,' (',Credits,')') AS CourseExtInfo FROM XCourse) V_1787 WHERE (`V_1787`.`Code` IS NOT NULL AND `V_1787`.`Credits` IS NOT NULL AND `V_1787`.`Name` IS NOT NULL AND `V_1787`.`Credits` IS NOT NULL) AND `V_1787`.`Credits` >= 6.0

Otrā vaicājuma tulkojums uz SQL	SELECT COUNT(`V_54765`.`Course_ID`) AS "Course_COUNT" FROM (SELECT Course_ID, Name, Code, Credits, Required, concat(Code,' (',Credits,')') AS CourseExtInfo FROM XCourse) V_54765
Trešā vaicājuma tulkojums uz SQL	SELECT `V_60585`.`Credits` AS "courseCredits", COUNT(`V_60585`.`Course_ID`) AS "Course_COUNT", 1705904476 AS "mappingid_courseCredits" FROM (SELECT Course_ID, Name, Code, Credits, Required, concat(Code,' (',Credits,')') AS CourseExtInfo FROM XCourse) V_60585 group by `V_60585`.`Credits`

Visu tabulā tulkoto vaicājumu struktūra ir līdzīga un FROM konstrukcijā esošajos apakšvaicājumos tie tiešā veidā izmanto attēlojumā atrodamo SQL skatu vaicājumus. Apskatot šos vaicājumus, ir redzams, ka tie nav tik kvalitatīvi kā Ontop rīka tulkojumi, jo tie vienmēr satur apakšvaicājumus, kā arī tulkojumos ir iekļauti R2RML attēlojuma fragmentu identifikatori. Šie rezultāti sakrīt arī ar informāciju, kas atrodama avotā [42], kur raksta autori ir salīdzinājuši Ontop sniegumu ar citiem līdzīgiem rīkiem un tika secināts, ka, lai gan Ontop rīkam ne vienmēr ir ātrākā veiktspēja, tā sniegums ir visstabilākais.

No rezultātiem, kas iegūti, darbinot Morph rīku, var secināt, ka šī rīka darbībai ir nepieciešami ļoti specifiski R2RML attēlojumi un šī iemesla dēļ Morph sniegums nav tik stabils kā citiem rīkiem, piemēram, Ontop.

REZULTĀTI

Darba izstrādes gaitā tika īsumā apskatītas galvenās semantiskā tīmekļa tehnoloģijas un standarti, aprakstītas ontoloģijās balstītas datu pieejas (OBDA) tehnoloģijas un rīki, apkopotas dažādas šobrīd pieejamās un zināmās metodes un optimizācijas semantisko vaicājumu tulkošanai uz relāciju datubāžu vaicājumiem, izveidota un papildināta infrastruktūra vizuālo vaicājumu izpildei pār relāciju datubāzi, kā arī analizēti ar izveidoto infrastruktūru iegūtie vaicājumu tulkojumi un to optimizācijas.

Darba pirmajās divās nodaļās atrodamais semantiskā tīmekļa tehnoloģiju un standartu apraksts, kā arī OBDA pieejā lietoto tehnoloģiju un rīku apskats veido darba teorētisko pamatu, kā arī sagatavo lasītāju darba tālākajām nodaļām.

Darba trešajā nodaļā tika apkopotas un klasificētas zinātniskos rakstos atrodamās semantisko vaicājumu tulkošanas pieejas un optimizācijas. Kā divas galvenās vaicājumu tulkošanas pieeju grupas tika izceltas trijniekus saturošu relāciju datubāžu pieejas, kas darbojas ar materializētiem RDF datiem, un virtuālo zināšanu grafu (VKG) pieejas, kurās dati, kas atrodami datu avotos, netiek dublēti, un tā vietā tiek izmantotas ontoloģijas un attēlojumi, lai nodrošinātu virtualizētu piekļuvi šiem datiem. Papildus šīm divām grupām tika apskatītas arī atsevišķas optimizācijas un tulkošanas pieejas, kuras neiederējās iepriekš minētajās grupās. Šis apkopojums un klasifikācija ļauj noskaidrot esošo situāciju semantisko vaicājumu tulkošanas pētījumu virzienā.

Darba ceturtajā nodaļā tika detalizēti aprakstīta vizuālu semantisku vaicājumu izpildei pār relāciju datubāzēm nepieciešamā infrastruktūra un tās uzstādīšana. Lai gan līdzīga infrastruktūra ir tikusi aprakstīta jau citos darbos, šajā darbā tika ieviesti šīs infrastruktūras papildinājumi, apvienojot rīkus vienā Docker Compose vidē, kā arī alternatīvu Ontop rīkam – Morph. Šīs infrastruktūras un tās uzstādīšanas apraksts kopā ar pielikumos atrodamajiem materiāliem var kalpot par atbalstu tālāku pētījumu veikšanai šajā jomā. Šajā nodaļā aprakstīts arī darbs ar konkrētu piemēra datubāzi, ontoloģiju un attēlojumu, kā arī īsumā pieminēti nepieciešamie darbi, kas būtu jāveic, un atsevišķi rīki, kas būtu jāizmanto, strādājot ar citu datubāzi, kuras zināšanu apgabalam nav pieejama gatava ontoloģija un attēlojums.

Darba piektajā nodaļā tika izmantota iepriekš aprakstītā infrastruktūra, lai izpildītu vizuālus semantiskus vaicājumus pār piemēra datubāzi. Veicot šo vaicājumu izpildi, tika atklāts, ka, izmantojot sākotnēji pieejamo piemēra ontoloģiju un attēlojumu, Ontop tulkotie vaicājumi ir tālu no optimāliem. Ar optimālu vaicājumu tiek saprasts vaicājums, kādu varētu uzrakstīt lietotājs ar tehniskām prasmēm un pietiekamām zināšanām par ontoloģijas zināšanu

apgabalu un avota datubāzi. Veicot izmaiņas pašos vaicājumos, tika noskaidrots, ka vairākos gadījumos vaicājumu tulkojumu sarežģītību palielina OPTIONAL konstrukciju esamība sākotnējā SPARQL vaicājumā. Lai gan dažos vaicājumos šo konstrukciju atmešana tikpat kā neizmainīja vaicājuma tulkojumu, bija arī vaicājumi, kuros šī darbība ievērojami samazināja LEFT OUTER JOIN savienojumu vai UNION operāciju skaitu. Atsevišķos gadījumos tika novērots, ka šāda darbība noveda pie papildu IS NOT NULL pārbaužu parādīšanās vaicājuma tulkojumā, ko, iespējams, var izskaidrot ar to, ka Ontop rīkam pieejamajos datos, t.i., ontoloģijā un attēlojumā, nav gana daudz informācijas, lai droši apgalvotu, ka konkrētas tabulas konkrētā kolonnā vienmēr būs netukša vērtība. Tā kā dažos no vaicājumiem tika atrastas arī citas liekas lauku vērtību pārbaudes WHERE nosacījumos, tika izlemts Ontop rīkam padot izmainītu piemēra ontoloģijas un attēlojuma variantu, kurā tika vienkāršotas apakšklašu hierarhijas, atmetot tās apakšklases, kuras netiek izmantotas izpildāmajos vaicājumos. Šī soļa rezultātā vairākos vaicājumos bija vērojami vienkāršāki WHERE nosacījumi, un vienā vaicājumā pat tika iegūts tulkojums, kas bija tuvs optimālam SQL vaicājumam.

Apkopojot iegūtos vaicājumu tulkojumus, to optimizētos un manuāli rakstītos variantus un iepriekšējā gada maģistra darbā [18] iegūtos tulkojumus, kā arī salīdzinot šos vaicājumus ar 5.1. nodaļā aprakstīto sarežģītības mēru, bija redzams, ka visos gadījumos manuāli rakstīts vaicājums bija ar vismazāko sarežģītību. Manuāli izpildot vaicājumu tulkojumu papildu optimizāciju, kuras ietvaros tika atmestas iepriekš minētās liekās IS NOT NULL pārbaudes, 5 no 12 vaicājumiem tika iegūts vaicājums, kas sakrita ar manuāli rakstītu vaicājumu. Šī optimizācija būtu vērtīgs papildinājums Ontop rīkā un ļautu uzlabot tulkoto vaicājumu kvalitāti. To varētu potenciāli īstenot, vai nu papildinot Ontop rīku, lai tas spētu izmantot papildu informāciju par datubāzes shēmu, vai arī realizējot šo optimizāciju ārpus Ontop rīka. Viens no vaicājumiem saturēja Ontop neatbalstīto EXISTS konstrukciju un līdz ar to nebija izpildāms, tomēr tam tika izveidots alternatīvs SPARQL vaicājums, kurā netiek izmantota šī konstrukcija. Padodot šo alternatīvo vaicājumu Ontop rīkam, izdevās iegūt SQL tulkojumu. Lai gan iegūtais tulkojums un tā optimizācija bija sarežģītāki nekā manuāli rakstīts vaicājums, šis rezultāts liek secināt, ka vērtīgs ViziQuer rīka papildinājums varētu būt negācijas saišu īstenošana ar OPTIONAL un !BOUND konstrukcijām. Fakts, ka 7 no 12 vaicājumiem šajā darbā iegūtie tulkojumi bija mazāk sarežģīti nekā iepriekš minētajā maģistra darbā iegūtie tulkojumi, liek domāt, ka ontoloģiju vienkāršošana ir noderīga metode, kā uzlabot Ontop rīka sniegtos vaicājumu tulkojumus.

Darbinot vaicājumu tulkošanas rīku Morph un salīdzinot tā dotos rezultātus ar tiem, kas tika iegūti ar Ontop, bija skaidri redzams, ka Ontop ir daudz stabilāks rīks ar mazāk specifiskām

prasībām pret tam doto attēlojumu, ko gan var izskaidrot ar to, ka Morph rīkam nav iespējas izmantot ontoloģiju un dotais attēlojums ir tā vienīgais informācijas avots. Šī īpašība bija redzama arī tajos tulkojumos, ko Morph spēja ģenerēt – tajos bija tiešā veidā izmantoti SQL vaicājumi, kas attēlojumā norādīti kā loģisko tabulu avoti. Izdarītie secinājumi par Ontop rīka stabilitāti sakrīt arī ar literatūrā atrodamo informāciju.

SECINĀJUMI

Maģistra darba izstrādes gaitā tika apskatītas tehnoloģijas, kas ir semantiskā tīmekļa pamatā, izpētīta ontoloģijās balstītas datu pieejas (OBDA) paradigma, kā arī atsevišķi rīki, kas tiek izmantoti OBDA sistēmu infrastruktūrā, apkopotas zinātniskos materiālos atrodamās vaicājumu tulkošanas pieejas, rīki un dažādas optimizācijas, uzstādīta, uzlabota un papildināta infrastruktūra vizuālu semantisku vaicājumu izpildei pār relāciju datubāzēm, kā arī apskatīti vaicājumu tulkošanas rezultāti, kas iegūti ar šo infrastruktūru.

Izpildot darbā izvirzītos mērķus, tika secināts, ka Ontop ir viens no stabilākajiem atvērtā pirmkoda semantisko vaicājumu tulkošanas rīkiem un, lai šis rīks spētu ģenerēt optimālākus vaicājumus, tam, iespējams, nepieciešama papildus informācija par avota datubāzi, kas nav atrodama ontoloģijā vai attēlojumā. No iegūtajiem rezultātiem var arī secināt, ka, strādājot ar Ontop rīku, tam ir vēlams dot ontoloģiju, kas nav pārlieku detalizēta, bet tajā pašā laikā ir pietiekami detalizēta, lai ļautu izpildīt nepieciešamos vaicājumus. Darbā iegūtie rezultāti ļauj secināt arī to, ka ar darbā minētajiem ViziQuer un Ontop rīku uzlabojumiem, izveidotā infrastruktūra vairākos gadījumos spētu dot kvalitatīvus vaicājumu tulkojumus, kas būtu līdzvērtīgi lietotāja rakstītiem vaicājumiem.

Ņemot vērā šī darba rezultātus, ir iespējams iezīmēt vairākus turpmāku pētījumu virzienus:

- Padziļināti izpētīt iespējas, ko piedāvā esošie ontoloģiju un attēlojumu ģenerēšanas rīki un atrast iespējamos šī procesa uzlabojumus, kas atvieglotu darbu ar datubāzēm, kurām nav pieejama gatava atbilstošā ontoloģija un attēlojums
- Pārbaudīt darbā izveidotās infrastruktūras darbību uz sarežģītākām datubāžu shēmām
- Salīdzināt tulkoto, optimizēto un manuāli rakstīto vaicājumu radīto slodzi dažādās relāciju datubāžu sistēmās

IZMANTOTĀ LITERATŪRA

[1] W3C, «RDF 1.1 Primer,» 24 06 2014. [Tiešsaiste]. Available: https://www.w3.org/TR/rdf11-primer/#section-Introduction. [Piekļūts 11 01 2021].

[2] RDF Working Group, «Resource Description Framework (RDF),» 25 2 2014. [Tiešsaiste]. Available: https://www.w3.org/RDF/. [Piekļūts 15 1 2021].

[3] Cambridge Semantics, «SPARQL vs SQL,» [Tiešsaiste]. Available: https://www.cambridgesemantics.com/blog/semantic-university/learn-sparql/sparql-vs-sql/. [Piekļūts 18 1 2021].

[4] W3C, «JSON-LD 1.1,» 16 6 2020. [Tiešsaiste]. Available: https://www.w3.org/TR/json-ld/. [Piekļūts 15 1 2021].

[5] Cambridge Semantics, «Learn RDF,» [Tiešsaiste]. Available: https://www.cambridgesemantics.com/blog/semantic-university/learn-rdf/. [Piekļūts 22 1 2021].

[6] W3C, «RDF Schema 1.1,» 25 2 2014. [Tiešsaiste]. Available: https://www.w3.org/TR/rdf-schema/. [Piekļūts 15 1 2021].

[7] OWL Working Group, «Web Ontology Language (OWL),» 11 12 2012. [Tiešsaiste]. Available: https://www.w3.org/2001/sw/wiki/OWL. [Piekļūts 16 1 2021].

[8] W3C, «OWL 2 Web Ontology Language Primer (Second Edition),» 11 12 2012. [Tiešsaiste]. Available: https://www.w3.org/TR/2012/REC-owl2-primer-20121211/. [Piekļūts 16 1 2021].

[9] Cambridge Semantics, «RDFS vs. Owl,» [Tiešsaiste]. Available: https://www.cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/rdfs-vs-owl/. [Piekļūts 16 1 2021].

[10] N. Bikakis, S. Christodoulakis , C. Tsinaraki , I. Stavrakantonakis un N. Gioldasis , «The XML and Semantic Web Worlds: Technologies, Interoperability and Integration. A survey of the State of the Art,» 2013. [Tiešsaiste]. Available: http://www.dblab.ntua.gr/~bikakis/XMLSemanticWebW3CTimeline.pdf. [Piekļūts 17 1 2021].

[11] W3C, «OWL 2 Web Ontology Language New Features and Rationale (Second Edition),» 11 12 2012. [Tiešsaiste]. Available: https://www.w3.org/TR/owl2-new-features/. [Piekļūts 17 1 2021].

[12] EUCLID Project, «Chapter 2: Querying Linked Data,» [Tiešsaiste]. Available: https://euclid-project.eu/modules/chapter2.html. [Piekļūts 18 1 2021].

[13] Cambridge Semantics, «Flavors of OWL,» [Tiešsaiste]. Available: https://www.cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/flavors-of-owl/. [Piekļūts 16 1 2021].

[14] W3C, «OWL 2 Web Ontology Language Profiles (Second Edition),» 11 12 2012. [Tiešsaiste]. Available: https://www.w3.org/TR/owl2-profiles. [Piekļūts 18 05 2021].

[15] Cambridge Semantics, «Learn SPARQL,» [Tiešsaiste]. Available: https://www.cambridgesemantics.com/blog/semantic-university/learn-sparql/. [Piekļūts 18 1 2021].

[16] EUCLID Project, «Querying Linked Data,» 28 2 2013. [Tiešsaiste]. Available: https://www.slideshare.net/EUCLIDproject/querying-linked-data. [Piekļūts 18 1 2021].

[17] W3C, «SPARQL 1.1 Query Language,» 21 3 2013. [Tiešsaiste]. Available: https://www.w3.org/TR/sparql11-query/. [Piekļūts 18 1 2021].

[18] K. Upenieks, «Vizuāli semantiski vaicājumi pār relāciju datubāzēm,» Rīga, 2020.

[19] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro un G. Xiao, «Ontop: Answering SPARQL queries over relational databases,» 2017.

[20] E. Kharlamov, E. Jiménez-Ruiz, D. Zheleznyakov un D. Bilidas, «Optique: Towards OBDA Systems for Industry,» 2013.

[21] S. A. Gomez un P. R. Fillottrani, «A Framework for OBDA: Current State and Perspectives,» 2019.

[22] W3C, «R2RML: RDB to RDF Mapping Language,» 27 9 2012. [Tiešsaiste]. Available: https://www.w3.org/TR/r2rml/. [Piekļūts 21 1 2021].

[23] RDB2RDF Working Group, «Relational Databases to RDF (RDB2RDF),» 21 9 2012. [Tiešsaiste]. Available: https://www.w3.org/2001/sw/wiki/RDB2RDF. [Piekļūts 21 1 2021].

[24] F. Priyatna, O. Corcho un J. Sequeda, «Formalisation and Experiences of R2RML-based SPARQLto SQL query translation using Morph,» 2014.

[25] EUCLID Project, «Chapter 3: Providing Linked Data,» [Tiešsaiste]. Available: https://euclid-project.eu/modules/chapter3.html. [Piekļūts 23 1 2021].

[26] W3C, «A Direct Mapping of Relational Data to RDF,» 27 9 2012. [Tiešsaiste]. Available: https://www.w3.org/TR/rdb-direct-mapping/. [Piekļūts 23 1 2021].

[27] G. Būmans, «Relational Database information availability to Semantic Web technologies,» Rīga, 2012.

[28] M. Hert, G. Reif un H. Gall, «A Comparison of RDB-to-RDF Mapping Languages,» 2011.

[29] B. De Meester, P. Heyvaert un T. Delva, «RDF Mapping Language (RML),» 6 10 2020. [Tiešsaiste]. Available: https://rml.io/specs/rml/. [Piekļūts 23 1 2021].

[30] The University of Southern California, «Karma: A Data Integration Tool,» 2016. [Tiešsaiste]. Available: https://usc-isi-i2.github.io/karma/. [Piekļūts 23 1 2021].

[31] T. Bagosi, D. Calvanese, J. Hardi, S. Komla-Ebri, D. Lanti, M. Rezk, M. Rodriguez-Muro, M. Slusnys un G. Xiao, «The Ontop Framework for Ontology Based Data Access,» 2014.

[32] Free University of Bozen-Bolzano, «Release notes | Ontop,» 18 10 2020. [Tiešsaiste].

Available: https://ontop-vkg.org/guide/releases.html. [Piekļūts 24 1 2021].

[33] M. A. Musen, «The Protégé project: A look back and a look forward,» 2015.

[34] E. Alatrish , «Comparison Some of Ontology Editors,» 2013.

[35] LUMII, « LUMII-Syslab / viziquer: Tool for Search in Structured Semantic Data,» [Tiešsaiste]. Available: https://github.com/LUMII-Syslab/viziquer. [Piekļūts 23 1 2021].

[36] K. Čerāns un J. Ovčiņņikova, «ViziQuer: Notation and Tool for Data Analysis SPARQL Queries,» 2016.

[37] LUMII, «ViziQuer/web Tool for RDF Data Analysis Queries,» [Tiešsaiste]. Available: https://viziquer.lumii.lv/. [Piekļūts 21 1 2021].

[38] I. Stinkevičs, «GRAFISKIE VAICĀJUMI RELĀCIJU DATUBĀZĒS,» 2018.

[39] I. Pīre, «Vizuālo vaicājumu ģenerēšana no tekstuālās formas,» Rīga, 2020.

[40] J. F. Sequeda un D. P. Miranker, «Ultrawrap: SPARQL Execution on Relational Data,» 2013.

[41] M. Gawinecki, «How schema mapping can help in data integration ? – Integrating the relational databases with ontologies,» 2008.

[42] G. Xiao, D. Lanti, R. Kontchakov, S. Komla-Ebri, E. Güzel-Kalaycı, L. Ding, J. Corman, B. Cogrel, D. Calvanese un E. Botoeva, «The Virtual Knowledge Graph System Ontop,» 2020.

[43] M. Rodríguez-Muro un M. Rezk, «Efficient SPARQL-to-SQL with R2RML mappings,» 2015.

[44] G. Xiao, L. Ding, B. Cogrel un D. Calvanese, «Virtual Knowledge Graphs: An Overview of Systems and Use Cases,» 2019.

[45] R. Cyganiak, «A relational algebra for SPARQL,» 2005.

[46] S. Kiminki, J. Knuuttila un V. Hirvisalo, «SPARQL to SQL Translation Based on an Intermediate Query Language,» 2010.

[47] A. Chebotko, S. Lu, H. M. Jamil un F. Fotouhi, «Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns,» 2006.

[48] A. Chebotko, S. Lu un F. Fotouhi, «Semantics preserving SPARQL-to-SQL translation,» 2009.

[49] A. Chebotko, S. Lu, M. Atay un F. Fotouhi, «Efficient Processing of RDF Queries with Nested Optional Graph Patterns in an RDBMS,» 2008.

[50] M. Rodriguez-Muro, M. Rezk, J. Hardi, M. Slusnys, T. Bagosi un D. Calvanese, «Evaluating SPARQL-to-SQL translation in Ontop,» 2013.

[51] P. Bricis, «ONTOP - VIRTUĀLĀ ZINĀŠANU GRAFU SISTĒMA,» 2021.

[52] M. Namici un G. De Giacomo, «Comparing Query Answering in OBDA Tools over W3C-Compliant Specifications,» 2018.

[53] R. Rosati un A. Almatelli, «Improving Query Answering over DL-Lite Ontologies,» 2010.

[54] IBM Corporation, «Semantic Data Access to Relational Databases,» 2005.

[55] F. Priyatna, «Methods and Techniques for the Generation and Efficient Exploitation of RDB2RDF Mappings,» 2015.

[56] C. Bizer un A. Seaborne, «D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs,» 2004.

[57] M. Strandhaug, «An R2RML Mapping Management API in Java,» 2014.

[58] L. London , R. Cardenas un R. Rimpi, «Query Optimization with RDF using SPARQL».

[59] N. Bikakis, N. Gioldasis, C. Tsinaraki un S. Christodoulakis, «Querying XML Data with SPARQL,» 2009.

[60] P. M. Fischer, D. Florescu, M. Kaufmann un D. Kossmann, «Translating SPARQL and SQL to XQuery,» 2011.

[61] Ontology Engineering Group, «Morph-RDB,» [Tiešsaiste]. Available: https://morph.oeg.fi.upm.es/tool/morph-rdb. [Piekļūts 20 05 2021].

[62] EUCLID Project, «Chapter 1: Introduction and Application Scenarios,» [Tiešsaiste]. Available: https://euclid-project.eu/modules/chapter1.html. [Piekļūts 23 1 2021].

PIELIKUMI

Vaicājumu izpildē izmantotā vienkāršotā ontoloģija

```
Prefix(:=<http://lumii.lv/ontologies/UnivExample.owl#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)

Ontology(<http://lumii.lv/ontologies/UnivExample.owl>
Annotation(:DBExpr "buildExtInfo(@T,@N)=concat(@T,' (',@N,')')")

Declaration(Class(:AcademicProgram))
Declaration(Class(:Course))
Declaration(Class(:FacultyLevel))
Declaration(Class(:Nationality))
Declaration(Class(:Person))
Declaration(Class(:Registration))
Declaration(Class(:Student))
Declaration(Class(:Teacher))
Declaration(Class(owl:Thing))
Declaration(ObjectProperty(:course))
Declaration(ObjectProperty(:enrolled))
Declaration(ObjectProperty(:facultyLevel))
Declaration(ObjectProperty(:includes))
Declaration(ObjectProperty(:nationality))
Declaration(ObjectProperty(:r))
Declaration(ObjectProperty(:student))
Declaration(ObjectProperty(:takes))
Declaration(ObjectProperty(:teaches))
Declaration(DataProperty(:acadTitle))
Declaration(DataProperty(:courseCode))
Declaration(DataProperty(:courseCredits))
Declaration(DataProperty(:courseExtInfo))
Declaration(DataProperty(:courseName))
Declaration(DataProperty(:dateCompleted))
Declaration(DataProperty(:datePaid))
Declaration(DataProperty(:hasMark))
Declaration(DataProperty(:mark))
Declaration(DataProperty(:nCode))
Declaration(DataProperty(:nValue))
Declaration(DataProperty(:personID))
Declaration(DataProperty(:personName))
Declaration(DataProperty(:programName))
Declaration(DataProperty(:salary))
Declaration(DataProperty(:studentName))
Declaration(DataProperty(:studentNumber))
Declaration(DataProperty(:teacherName))
SubClassOf(:AcademicProgram owl:Thing)
AnnotationAssertion(<http://obis.lumii.lv/2013/01/obis#defaultOrder>                    :Course
```

"courseName,courseCode,courseCredits,courseExtInfo")
AnnotationAssertion(<http://rdb2owl.lumii.lv/2012/1.0/rdb2owl#DBExpr> :Course "XCourse")
SubClassOf(:Course owl:Thing)
AnnotationAssertion(<http://obis.lumii.lv/2013/01/obis#defaultOrder> :FacultyLevel "acadTitle")
AnnotationAssertion(<http://rdb2owl.lumii.lv/2012/1.0/rdb2owl#DBExpr> :FacultyLevel "Teacher_Level {uri=('Level_', Level_Code)}")
AnnotationAssertion(<http://obis.lumii.lv/2013/01/obis#textPattern> :FacultyLevel "{#1}[-{#2}]")
AnnotationAssertion(<http://obis.lumii.lv/2013/01/obis#isEnumerated> :FacultyLevel "true")
SubClassOf(:FacultyLevel owl:Thing)
SubClassOf(:Nationality owl:Thing)
SubClassOf(:Person owl:Thing)
SubClassOf(:Registration owl:Thing)
SubClassOf(:Student :Person)
SubClassOf(:Teacher :Person)
ObjectPropertyDomain(:course :Registration)
ObjectPropertyRange(:course :Course)
ObjectPropertyDomain(:enrolled :Student)
ObjectPropertyRange(:enrolled :AcademicProgram)
ObjectPropertyDomain(:facultyLevel :Teacher)
ObjectPropertyRange(:facultyLevel :FacultyLevel)
ObjectPropertyDomain(:includes :AcademicProgram)
ObjectPropertyRange(:includes :Course)
ObjectPropertyDomain(:nationality :Person)
ObjectPropertyRange(:nationality :Nationality)
InverseObjectProperties(:student :r)
ObjectPropertyDomain(:r :Student)
ObjectPropertyRange(:r :Registration)
ObjectPropertyDomain(:student :Registration)
ObjectPropertyRange(:student :Student)
ObjectPropertyDomain(:takes :Student)
ObjectPropertyRange(:takes :Course)
DisjointObjectProperties(:takes :teaches)
ObjectPropertyDomain(:teaches :Teacher)
ObjectPropertyRange(:teaches :Course)
DataPropertyDomain(:acadTitle :FacultyLevel)
DataPropertyRange(:acadTitle xsd:string)
DataPropertyDomain(:courseCode :Course)
DataPropertyRange(:courseCode xsd:string)
DataPropertyDomain(:courseCredits :Course)
DataPropertyRange(:courseCredits DatatypeRestriction(xsd:integer xsd:minExclusive "0"^^xsd:integer))
DataPropertyDomain(:courseExtInfo :Course)
DataPropertyRange(:courseExtInfo xsd:string)
DataPropertyDomain(:courseName :Course)
DataPropertyRange(:courseName xsd:string)
DataPropertyDomain(:dateCompleted :Registration)
DataPropertyRange(:dateCompleted xsd:dateTime)
DataPropertyDomain(:datePaid :Registration)
DataPropertyRange(:datePaid xsd:dateTime)

```
DataPropertyDomain(:hasMark :Student)
DataPropertyRange(:hasMark xsd:integer)
DataPropertyDomain(:mark :Registration)
DataPropertyRange(:mark DatatypeRestriction(xsd:integer xsd:minInclusive "0"^^xsd:integer xsd:maxInclusive "10"^^xsd:integer))
DataPropertyDomain(:nCode :Nationality)
DataPropertyRange(:nCode xsd:string)
DataPropertyDomain(:nValue :Nationality)
DataPropertyRange(:nValue xsd:string)
DataPropertyDomain(:personID :Person)
DataPropertyRange(:personID xsd:string)
DataPropertyDomain(:personName :Person)
DataPropertyRange(:personName xsd:string)
DataPropertyDomain(:programName :AcademicProgram)
DataPropertyRange(:programName xsd:string)
DataPropertyDomain(:salary :Teacher)
DataPropertyRange(:salary DatatypeRestriction(xsd:decimal xsd:minExclusive "0"^^xsd:integer))
SubDataPropertyOf(:studentName :personName)
DataPropertyDomain(:studentName :Student)
DataPropertyRange(:studentName xsd:string)
DataPropertyDomain(:studentNumber :Student)
DataPropertyRange(:studentNumber xsd:string)
SubDataPropertyOf(:teacherName :personName)
DataPropertyDomain(:teacherName :Teacher)
DataPropertyRange(:teacherName xsd:string)
DisjointClasses(:AcademicProgram :Course :FacultyLevel :Nationality :Person :Registration)
)
```

Vaicājumu izpildē izmantotais R2RML attēlojums

```
@prefix : <http://lumii.lv/ontologies/UnivExample.owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<urn:mapping-835943492> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
     rr:sqlQuery "SELECT stud.Student_ID, stud.Program_ID    FROM XSTUDENT stud"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:template "http://lumii.lv/ontologies/UnivExample.owl#program/{Program_ID}";
        rr:termType rr:IRI
       ];
     rr:predicate :enrolled
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
     rr:template "http://lumii.lv/ontologies/UnivExample.owl#student/{Student_ID}";
     rr:termType rr:IRI
    ] .

<urn:mapping-1374849872> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
     rr:sqlQuery "SELECT tch.Teacher_ID, tch.Level_Code    FROM XTeacher tch"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:template "http://lumii.lv/ontologies/UnivExample.owl#facultyLevel/{Level_Code}";
        rr:termType rr:IRI
       ];
     rr:predicate :facultyLevel
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
     rr:template "http://lumii.lv/ontologies/UnivExample.owl#teacher/{Teacher_ID}";
     rr:termType rr:IRI
    ] .

<urn:mapping--796749985> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
     rr:sqlQuery "SELECT reg.Student_ID, reg.Registration_ID    FROM Registration reg"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:template "http://lumii.lv/ontologies/UnivExample.owl#student/{Student_ID}";
        rr:termType rr:IRI
       ];
     rr:predicate :student
    ];
```

```
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
     rr:template "http://lumii.lv/ontologies/UnivExample.owl#registration/{Registration_ID}";
     rr:termType rr:IRI
    ] .


<urn:mapping--1962920835> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
     rr:sqlQuery "SELECT cou.Program_ID, cou.Course_ID    FROM XCourse cou"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:template "http://lumii.lv/ontologies/UnivExample.owl#course/{Course_ID}";
        rr:termType rr:IRI
       ];
     rr:predicate :includes
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
     rr:template "http://lumii.lv/ontologies/UnivExample.owl#program/{Program_ID}";
     rr:termType rr:IRI
    ] .

<urn:mapping--6494452> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
     rr:sqlQuery "SELECT tch.Teacher_ID, tch.Nat_Code    FROM XTeacher tch"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:template "http://lumii.lv/ontologies/UnivExample.owl#nationality/{Nat_Code}";
        rr:termType rr:IRI
       ];
     rr:predicate :nationality
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
     rr:template "http://lumii.lv/ontologies/UnivExample.owl#teacher/{Teacher_ID}";
     rr:termType rr:IRI
    ] .

<urn:mapping--1304774845> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
     rr:sqlQuery "SELECT reg.Student_ID, reg.Course_ID    FROM Registration reg WHERE
reg.DatePaid IS NOT NULL"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:template "http://lumii.lv/ontologies/UnivExample.owl#course/{Course_ID}";
        rr:termType rr:IRI
       ];
     rr:predicate :takes
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
     rr:template "http://lumii.lv/ontologies/UnivExample.owl#student/{Student_ID}";
```

```
      rr:termType rr:IRI
    ] .

<urn:mapping--809197163> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
      rr:sqlQuery "SELECT cou.Teacher_ID, cou.Course_ID    FROM XCourse cou"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:template "http://lumii.lv/ontologies/UnivExample.owl#course/{Course_ID}";
          rr:termType rr:IRI
        ];
      rr:predicate :teaches
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
      rr:template "http://lumii.lv/ontologies/UnivExample.owl#teacher/{Teacher_ID}";
      rr:termType rr:IRI
    ] .

<urn:mapping--2086936556> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
      rr:sqlQuery "SELECT * FROM Teacher_Level"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:column "Level_Code";
          rr:termType rr:Literal
        ];
      rr:predicate :acadTitle
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
      rr:class :FacultyLevel;
      rr:template "http://lumii.lv/ontologies/UnivExample.owl#facultyLevel/{Level_Code}";
      rr:termType rr:IRI
    ] .

<urn:mapping--103498074> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
      rr:sqlQuery "SELECT nat.Code, nat.Value    FROM Nationality nat"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:column "Code";
          rr:termType rr:Literal
        ];
      rr:predicate :nCode
    ], [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:column "Value";
          rr:termType rr:Literal
        ];
      rr:predicate :nValue
```

```
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
      rr:class :Nationality;
      rr:template "http://lumii.lv/ontologies/UnivExample.owl#nationality/{Code}";
      rr:termType rr:IRI
    ] .

<urn:mapping--119188634> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
      rr:sqlQuery "SELECT tch.Teacher_ID, tch.Name, tch.Salary, tch.IDCode FROM
XTeacher tch"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:column "Name";
          rr:termType rr:Literal
        ];
      rr:predicate :teacherName
    ], [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:column "Salary";
          rr:termType rr:Literal
        ];
      rr:predicate :salary
    ], [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:column "Name";
          rr:termType rr:Literal
        ];
      rr:predicate :personName
    ], [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:column "IDCode";
          rr:termType rr:Literal
        ];
      rr:predicate :personID
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
      rr:class :Teacher;
      rr:template "http://lumii.lv/ontologies/UnivExample.owl#teacher/{Teacher_ID}";
      rr:termType rr:IRI
    ] .

<urn:mapping--1226459738> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
      rr:sqlQuery "SELECT reg.Registration_ID, reg.Course_ID    FROM Registration reg"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:template "http://lumii.lv/ontologies/UnivExample.owl#course/{Course_ID}";
          rr:termType rr:IRI
        ];
```

```
    rr:predicate :course
   ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
    rr:template "http://lumii.lv/ontologies/UnivExample.owl#registration/{Registration_ID}";
    rr:termType rr:IRI
   ] .


<urn:mapping--716491684> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
    rr:sqlQuery "SELECT Course_ID, Name, Code, Credits, Required, concat(Code,' (',Credits,')') AS CourseExtInfo    FROM XCourse"
   ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
    rr:objectMap [ a rr:ObjectMap, rr:TermMap;
      rr:column "Name";
      rr:termType rr:Literal
     ];
    rr:predicate :courseName
   ], [ a rr:PredicateObjectMap;
    rr:objectMap [ a rr:ObjectMap, rr:TermMap;
      rr:column "Credits";
      rr:termType rr:Literal
     ];
    rr:predicate :courseCredits
   ], [ a rr:PredicateObjectMap;
    rr:objectMap [ a rr:ObjectMap, rr:TermMap;
      rr:column "Code";
      rr:termType rr:Literal
     ];
    rr:predicate :courseCode
   ], [ a rr:PredicateObjectMap;
    rr:objectMap [ a rr:ObjectMap, rr:TermMap;
      rr:column "CourseExtInfo";
      rr:termType rr:Literal
     ];
    rr:predicate :courseExtInfo
   ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
    rr:class :Course;
    rr:template "http://lumii.lv/ontologies/UnivExample.owl#course/{Course_ID}";
    rr:termType rr:IRI
   ] .

<urn:mapping--1904323164> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
    rr:sqlQuery "SELECT stud.Student_ID, stud.Name, stud.IDCode, stud.StudNumber FROM XSTUDENT stud"
   ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
    rr:objectMap [ a rr:ObjectMap, rr:TermMap;
      rr:column "Name";
```

```
        rr:termType rr:Literal
       ];
     rr:predicate :personName
   ], [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:column "IDCode";
        rr:termType rr:Literal
       ];
     rr:predicate :personID
   ], [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:column "Name";
        rr:termType rr:Literal
       ];
     rr:predicate :studentName
   ], [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:column "StudNumber";
        rr:termType rr:Literal
       ];
     rr:predicate :studentNumber
   ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
     rr:class :Student;
     rr:template "http://lumii.lv/ontologies/UnivExample.owl#student/{Student_ID}";
     rr:termType rr:IRI
   ] .

<urn:mapping--601916929> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
     rr:sqlQuery "SELECT reg.Registration_ID, reg.DatePaid, reg.DateCompleted,
reg.MarkReceived    FROM Registration reg"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:column "MarkReceived";
        rr:datatype xsd:integer;
        rr:termType rr:Literal
       ];
     rr:predicate :mark
   ], [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:column "DateCompleted";
        rr:datatype xsd:dateTime;
        rr:termType rr:Literal
       ];
     rr:predicate :dateCompleted
   ], [ a rr:PredicateObjectMap;
     rr:objectMap [ a rr:ObjectMap, rr:TermMap;
        rr:column "DatePaid";
        rr:datatype xsd:dateTime;
        rr:termType rr:Literal
```

```
        ];
      rr:predicate :datePaid
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
      rr:class :Registration;
      rr:template "http://lumii.lv/ontologies/UnivExample.owl#registration/{Registration_ID}";
      rr:termType rr:IRI
    ] .

<urn:mapping-68302541> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
      rr:sqlQuery "SELECT prog.Program_ID, prog.Name    FROM XProgram prog"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:column "Name";
          rr:termType rr:Literal
        ];
      rr:predicate :programName
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
      rr:class :AcademicProgram;
      rr:template "http://lumii.lv/ontologies/UnivExample.owl#program/{Program_ID}";
      rr:termType rr:IRI
    ] .

<urn:mapping--142157752> a rr:TriplesMap;
  rr:logicalTable [ a rr:R2RMLView;
      rr:sqlQuery "SELECT stud.Student_ID, stud.Nat_Code    FROM XStudent stud"
    ];
  rr:predicateObjectMap [ a rr:PredicateObjectMap;
      rr:objectMap [ a rr:ObjectMap, rr:TermMap;
          rr:template "http://lumii.lv/ontologies/UnivExample.owl#nationality/{Nat_Code}";
          rr:termType rr:IRI
        ];
      rr:predicate :nationality
    ];
  rr:subjectMap [ a rr:SubjectMap, rr:TermMap;
      rr:template "http://lumii.lv/ontologies/UnivExample.owl#student/{Student_ID}";
      rr:termType rr:IRI
    ] .
```

Piemēra MySQL datubāzes izveides skripts

```
CREATE TABLE Nationality (
Code varchar(2) NOT NULL,
Value varchar(50) NULL);
ALTER TABLE Nationality ADD CONSTRAINT PK_Nationality PRIMARY KEY(Code);

CREATE TABLE Registration (
Registration_ID numeric(10, 0) NOT NULL,
Course_ID numeric(10, 0) NULL,
Student_ID numeric(10, 0) NULL,
DatePaid datetime NULL,
DateCompleted datetime NULL,
MarkReceived int NULL);
ALTER TABLE Registration ADD CONSTRAINT REGISTRATION_PK PRIMARY KEY(Registration_ID);

CREATE TABLE Teacher_Level (
Level_Code varchar(30) NOT NULL);
ALTER TABLE Teacher_Level ADD CONSTRAINT TEACHER_LEVEL_PK PRIMARY KEY(Level_Code);

CREATE TABLE XCourse (
Course_ID numeric(10, 0) NOT NULL,
Teacher_ID numeric(10, 0) NULL,
Program_ID numeric(10, 0) NULL,
Code varchar(6) NULL,
Name varchar(40) NULL,
Credits int NULL,
Required numeric(1, 0) NULL);
ALTER TABLE XCourse ADD CONSTRAINT COURSE_PK PRIMARY KEY(Course_ID);

CREATE TABLE XProgram (
Program_ID numeric(10, 0) NOT NULL,
Name varchar(80) NULL);
ALTER TABLE XProgram ADD CONSTRAINT PROGRAM_PK PRIMARY KEY(Program_ID);

CREATE TABLE XStudent (
Student_ID numeric(10, 0) NOT NULL,
Program_ID numeric(10, 0) NULL,
IDCode varchar(30) NULL,
Name varchar(80) NULL,
StudNumber varchar(24) NULL,
Nat_Code varchar(2) NULL);
ALTER TABLE XStudent ADD CONSTRAINT STUDENT_PK PRIMARY KEY(Student_ID);

CREATE TABLE XTeacher (
Teacher_ID numeric(10, 0) NOT NULL,
Level_Code varchar(30) NULL,
```

```
IDCode varchar(30) NULL,
Name varchar(40) NULL,
Nat_Code varchar(2) NULL,
Salary decimal(18, 2) NULL);
ALTER TABLE XTeacher ADD CONSTRAINT TEACHER_PK PRIMARY KEY(Teacher_ID);

INSERT INTO Nationality (Code, Value) VALUES ('DE', 'Germany');
INSERT INTO Nationality (Code, Value) VALUES ('EE', 'Estonia');
INSERT INTO Nationality (Code, Value) VALUES ('FR', 'France');
INSERT INTO Nationality (Code, Value) VALUES ('LT', 'Lithuania');
INSERT INTO Nationality (Code, Value) VALUES ('LV', 'Latvia');
INSERT INTO Nationality (Code, Value) VALUES ('UK', 'United Kingdom');
INSERT INTO Nationality (Code, Value) VALUES ('US', 'United States');

INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (1, 2, 1, CAST('2013-01-10 00:00:00.000' AS DateTime), CAST('2013-06-22 00:00:00.000' AS DateTime), 5);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (2, 4, 1, CAST('2013-01-09 00:00:00.000' AS DateTime), CAST('2013-06-20 00:00:00.000' AS DateTime), 8);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (3, 1, 2, CAST('2012-12-17 00:00:00.000' AS DateTime), CAST('2013-07-23 00:00:00.000' AS DateTime), 8);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (4, 3, 2, NULL, NULL, NULL);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (5, 2, 3, CAST('2014-01-08 00:00:00.000' AS DateTime), NULL, NULL);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (6, 2, 5, CAST('2014-01-08 00:00:00.000' AS DateTime), CAST('2014-06-12 00:00:00.000' AS DateTime), 4);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (7, 2, 6, CAST('2014-01-08 00:00:00.000' AS DateTime), CAST('2014-06-12 00:00:00.000' AS DateTime), 7);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (8, 2, 7, NULL, NULL, NULL);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (9, 1, 5, CAST('2014-01-09 00:00:00.000' AS DateTime), CAST('2014-06-12 00:00:00.000' AS DateTime), 8);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (10, 5, 5, CAST('2014-01-10 00:00:00.000' AS DateTime), CAST('2014-06-12 00:00:00.000' AS DateTime), 8);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (11, 4, 5, CAST('2014-01-11 00:00:00.000' AS DateTime), CAST('2014-06-12 00:00:00.000' AS DateTime), 7);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (12, 5, 6, CAST('2014-01-12 00:00:00.000' AS DateTime), NULL, NULL);
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (13, 5, 7, CAST('2014-01-13 00:00:00.000' AS DateTime), CAST('2014-06-12 00:00:00.000' AS DateTime), 6);
```

```
INSERT INTO Registration (Registration_ID, Course_ID, Student_ID, DatePaid, DateCompleted, MarkReceived) VALUES (14, 1, 5, CAST('2014-01-13 00:00:00.000' AS DateTime), CAST('2014-06-12 00:00:00.000' AS DateTime), 10);

INSERT INTO Teacher_Level (Level_Code) VALUES ('Assistant');
INSERT INTO Teacher_Level (Level_Code) VALUES ('AssocProfessor');
INSERT INTO Teacher_Level (Level_Code) VALUES ('Professor');

INSERT INTO XCourse (Course_ID, Teacher_ID, Program_ID, Code, Name, Credits, Required) VALUES (1, 3, 2, 'CC0140', 'Programming Basics', 6, 0);
INSERT INTO XCourse (Course_ID, Teacher_ID, Program_ID, Code, Name, Credits, Required) VALUES (2, 1, 1, 'CC0207', 'Semantic Web', 6, 1);
INSERT INTO XCourse (Course_ID, Teacher_ID, Program_ID, Code, Name, Credits, Required) VALUES (3, 2, 2, 'CT1113', 'Computer Networks', 3, 1);
INSERT INTO XCourse (Course_ID, Teacher_ID, Program_ID, Code, Name, Credits, Required) VALUES (4, 2, 1, 'CX5504', 'Quantum Computations', 3, 0);
INSERT INTO XCourse (Course_ID, Teacher_ID, Program_ID, Code, Name, Credits, Required) VALUES (5, 3, 1, 'CC2158', 'Programming Languages', 3, 0);

INSERT INTO XProgram (Program_ID, Name) VALUES (1, 'Computer Science');
INSERT INTO XProgram (Program_ID, Name) VALUES (2, 'Computer Engeneering');

INSERT INTO XStudent (Student_ID, Program_ID, IDCode, Name, StudNumber, Nat_Code) VALUES (1, 1, '123456789', 'Dave', 'SX43376', 'LV');
INSERT INTO XStudent (Student_ID, Program_ID, IDCode, Name, StudNumber, Nat_Code) VALUES (2, 2, '987654321', 'Eve', 'SX45675', 'EE');
INSERT INTO XStudent (Student_ID, Program_ID, IDCode, Name, StudNumber, Nat_Code) VALUES (3, 1, '555555555', 'Charlie', 'SX43212', 'UK');
INSERT INTO XStudent (Student_ID, Program_ID, IDCode, Name, StudNumber, Nat_Code) VALUES (4, 2, '345453432', 'Iva', 'SY44333', NULL);
INSERT INTO XStudent (Student_ID, Program_ID, IDCode, Name, StudNumber, Nat_Code) VALUES (5, 1, '222333444', 'Anna', 'SX00012', 'LV');
INSERT INTO XStudent (Student_ID, Program_ID, IDCode, Name, StudNumber, Nat_Code) VALUES (6, 1, '232454656', 'Lucia', 'SX00014', 'EE');
INSERT INTO XStudent (Student_ID, Program_ID, IDCode, Name, StudNumber, Nat_Code) VALUES (7, 1, '565678785', 'Bob Jr', 'SX89987', 'LV');

INSERT INTO XTeacher (Teacher_ID, Level_Code, IDCode, Name, Nat_Code, Salary) VALUES (1, 'Professor', '999999999', 'Alice', 'US', 1810.00);
INSERT INTO XTeacher (Teacher_ID, Level_Code, IDCode, Name, Nat_Code, Salary) VALUES (2, 'Professor', '777777777', 'Bob', 'LV', 1655.00);
INSERT INTO XTeacher (Teacher_ID, Level_Code, IDCode, Name, Nat_Code, Salary) VALUES (3, 'Assistant', '555555555', 'Charlie', NULL, 1033.50);

ALTER TABLE Registration ADD CONSTRAINT
REGISTRATION_COURSE_FK FOREIGN KEY(Course_ID)
REFERENCES XCourse (Course_ID)
ON DELETE SET NULL;

ALTER TABLE Registration ADD CONSTRAINT
REGISTRATION_STUDENT_FK FOREIGN KEY(Student_ID)
REFERENCES XStudent (Student_ID)
```

```
ON DELETE SET NULL;

ALTER TABLE XCourse ADD CONSTRAINT
COURSE_PROGRAM_FK FOREIGN KEY(Program_ID)
REFERENCES XProgram (Program_ID)
ON DELETE SET NULL;

ALTER TABLE XCourse ADD CONSTRAINT
COURSE_TEACHER_FK FOREIGN KEY(Teacher_ID)
REFERENCES XTeacher (Teacher_ID)
ON DELETE SET NULL;

ALTER TABLE XStudent ADD CONSTRAINT
FK_XSTUDENT_Nationality FOREIGN KEY(Nat_Code)
REFERENCES Nationality (Code);

ALTER TABLE XStudent ADD CONSTRAINT
STUDENT_PROGRAM_FK FOREIGN KEY(Program_ID)
REFERENCES XProgram (Program_ID)
ON DELETE SET NULL;

ALTER TABLE XTeacher ADD CONSTRAINT
FK_XTEACHER_Nationality FOREIGN KEY(Nat_Code)
REFERENCES Nationality (Code);

ALTER TABLE XTeacher ADD CONSTRAINT
TEACHER_TEACHER_LEVEL_FK FOREIGN KEY(Level_Code)
REFERENCES Teacher_Level (Level_Code)
ON DELETE SET NULL;
```

Docker Compose infrastruktūras konfigurācija

```
version: '3.4'
services:
  ontop:
    image: ontop/ontop-endpoint
    environment:
      ONTOP_ONTOLOGY_FILE: /opt/ontop/input/mini_univ.owl
      ONTOP_MAPPING_FILE: /opt/ontop/input/mini_univ_mapping.ttl
      ONTOP_PROPERTIES_FILE: /opt/ontop/input/mini_univ.properties
      ONTOP_CORS_ALLOWED_ORIGINS: "*"
      ONTOP_DEV_MODE: "true"
    volumes:
      - ./ontop/input:/opt/ontop/input
      - ./ontop/jdbc:/opt/ontop/jdbc
    container_name: ontop
    ports:
      - "8170:8170"
      - "8080:8080"
  main:
    image: viziquer/viziquer
    container_name: viziquer
    ports:
      - "8171:3000"
    links:
      - mongodb
    environment:
      MONGO_URL: mongodb://mongodb/vq
      STARTUP_DELAY: 10
  mongodb:
    image: mongo
    container_name: mongo
    volumes:
      - vqdata:/data/db
  morphrdb:
    container_name: morph-rdb
    image: morph-rdb:latest
    volumes:
      - ./configs:/configs
volumes:
  vqdata:
    external: true
networks:
  default:
    external:
      name: mag_net
```

Maģistra darbs "***Vizuālu semantisku vaicājumu izpilde pār relāciju datubāzēm***" izstrādāts LU Datorikas fakultātē.

Darba teksta galīgā versija izgatavota **24.05.2021**.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: **Ralfs Sedlenieks 24.05.2021**

(Autora paraksts un datums)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par p i e m ē r o t u / ~~n e p i e m ē r o t u~~ (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Darba vadītājs: **Dr.dat. Kārlis Čerāns 24.05.2021**

(Vadītāja paraksts un datums)

Darbs iesniegts maģistratūras sekretariātā **24.05.2021**.

(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: _____.

(Metodiķes paraksts)

Recenzents: **profesors Dr.dat. Ģirts Karnītis**

(Akad.amats, zin.grāds, vārds, uzvārds)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____

(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____

(Sekretāra paraksts)