LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

Dati kā ontoloģija - glabāšana, vaicāšana, vizualizācija

Promocijas darbs

Nozare: datorzinātnes
Apakšnozare: programmēšanas valodas un sistēmas

Autors:
Mārtiņš Zviedris
Stud. apl. Mz09200
Darba vadītājs:
profesors Dr. dat. Guntis Bārzdiņš

RĪGA 2014

Šis darbs izstrādāts ar Eiropas Sociālā fonda atbalstu projektā **«Atbalsts doktora studijām Latvijas Universitātē - 2».**

ANOTĀCIJA

Promocijas darba centrālā tēma ir semantiski dati un to lietojums datu atlasē un informācijas sistēmu izstrādē.

Darba centrālā tēze ir "Semantiskā Latvija" – ontoloģijas kā datu vienojošs aspekts. Tās tiek publicētas, ir pieejama platforma, kas interpretē ontoloģiju un ļauj tajā ievadīt un labot datus, ir pieejams grafisks rīks, kas ļauj sastādīt grafiskus vaicājumus pār izveidotajiem datiem.

Darbā ir izstrādāta grafiska vaicājumu valoda, ar kuras palīdzību var sastādīt grafiskus vaicājumus pār semantiskiem datiem. Vaicājumi tālāk tiek tulkoti uz SPARQL vaicājumu valodu. Pierādīts, ka vaicājumu valoda ir lietotājiem saprotama un uzlabo lasāmību.

Vaicājumu valodai ir izstrādāts arī eksperimentāls realizācijas prototips - ViziQuer. Praktiskos piemēros tas ir pārbaudīts un saņēmis pozitīvas lietotāju atsauksmes.

Līdztekus ir aprakstīta metodoloģija ontoloģijās balstītu informatīvo sistēmu (OBIS) izstrādei un ieskicēti teorētiskie pamati ontoloģiju repozitorija izveidei, lai padarītu semantiskas informācijas sistēmas pieejamākas ārpus zinātniskās vides.

SATURA RĀDĪTĀJS

IEVADS

Promocijas darbs izstrādāts Latvijas Universitātes aģentūrā "Matemātikas un informātikas institūts" un Latvijas Universitātes Datorikas fakultātē. Promocijas darba vadītājs ir profesors Guntis Bārzdiņš. Promocijas darba pamatā ir metodoloģija – kā lietotāji var ērtāk un ātrāk darboties ar datiem izmantojot datu konceptuālo modeli. Tās ir balstītas uz "Semantiskās Latvijas" vīziju [5]. Darbs tiek balstīts uz Semantiskā tīmekļa tehnoloģijām, kas ļauj darboties ar datiem, izmantojot tikai konceptuālo modeli.

Promocijas darba tēmas pārskats

Līdz ar relāciju datu bāzes pirmsākumiem un to atlases valodu SQL, noritēja arī darbs pie lietotājiem ērtākas un saprotamākas valodas izstrādes. Kā pirmais piedāvātais risinājums ir jāmin *Query by Example* [1].

Kā minēts rakstā [2], tad grafisku vaicājumu valodas shematiska izstrāde notika jau laikā, kad tekstu ievade datoros notika kā līniju aizpildīšana ar tekstu un bija ļoti ierobežotas grafiskās iespējas. Tā izpēte vairāk bija teorētiskā līmenī, ne tik daudz praktiskā.

Taču rakstā [3] ir parādīts, kā lietotāju sastādīto vaicājumu precizitāti ietekmē, ja tie ir sastādīti tekstuāli vai ja tie ir sastādīti izmantojot grafisku vaicājumu valodu. Raksta galvenie rezultāti parāda, ka vieglus un vidējas grūtības vaicājumus grafiski sastāda precīzāk un ātrāk kā iesācēju lietotāji, tā arī pieredzējuši un eksperta līmeņa lietotāji. Sarežģītas pakāpes vaicājumus sastādīja tikai eksperta līmeņa lietotāji, un rezultāti kā tekstuāli tā arī grafiski ir līdzīgi.

Taču tajā pašā laikā rakstā [3] ir izmantots datubāzu konceptuālais līmenis, uz kuru lietotāji sastāda vaicājumu, nevis tehniskās realizācijas līmenis. Līdz ar to, ja spējam pāriet no tehniskā līmeņa uz konceptuālo līmeni datu attēlošanā, tad diagrammas veidā formulēti vaicājumi var atvieglot lietotāju darbu vaicājumu sastādīšanā.

Laika gaitā ir izstrādātas dažāda tipa grafiskās valodas un pārskatā [4] tās ir iedalītas – uz formām bāzētas, diagrammatiskas, ikonveidīgas un vairāku tipu apvienojums. Tas norāda, ka problēmas risināšanai var izvēlēties dažādus variantus.

Līdz ar to darbs pie dažādu veidu uzlabojumiem grafisku vaicājumu atlases veikšanai datiem ir noritējis ļoti ilgu laika posmu un kā viens no galvenajiem ierobežojumiem sekmīgai to realizācijai ir konceptuālā līmeņa pieejamības trūkums.

Promocijas darba tēmas aktualitāte

Matemātikas un informātikas institūta zinātnieku grupa 2006. gadā publicēja rakstu [5] par Semantiskā tīmekļa vīzijas vienu no iespējamajiem ceļiem – "Semantiskā Latvija".

Bija pierasts, ka dati tiek glabāti relāciju datu bāzēs un tiem ir konceptuālais ER modelis, kā dati tiek saprasti, un realizācijas modelis, kas satur papildus tehniskus datus, piemēram, primārās atslēgas. Taču lietotājiem pieejams bija tikai realizācijas modelis, jo konceptuālais modelis pēc izstrādes parasti tika pazaudēts un labākajā gadījumā tas tika saglabāts papīra formāta dokumentācijā.

Līdz ar Semantiskā tīmekļa idejas popularizēšanu [6] datu attēlošanas pamatā tika izvirzītas ontoloģijas [7]. Viens no to būtiskajiem aspektiem ir tas, ka datu konceptuālais un fiziskais modelis sakrīt. Līdz ar to, lietotājiem ir iespējams pārskatīt datus konceptuālā līmenī, nevis fiziskajā, kā tas notiek relāciju datubāzu gadījumā.

"Semantiskās Latvijas" centrālai tēzei tika izvirzīts ontoloģiju portāls, kurā lietotāji varētu atrast savam domēnam atbilstošu ontoloģiju un to lejupielādēt. Tālāk lietotāji varētu ielasīt šo ontoloģiju speciālā rīkā, ar kura palīdzību tiktu izveidotas ontoloģijas datu līmeņa instances. Noslēgumā lietotājs varētu šīs instances saglabāt kā HTML lapu un padarīt pieejamu globālajā tīmeklī citiem lietotājiem. Turklāt, šie dati saturētu arī semantisku informāciju, ko varētu apstrādāt automātiski.

Kā noslēdzošā "Semantiskās Latvijas" tēze tika minēts grafisks rīks, ar kura palīdzību lietotāji varētu ērti sastādīt vaicājumus pār izveidotajiem datiem. Jo dati tiktu glabāti konceptuālā līmenī un lietotājiem tos būtu vieglāk saprast un par tiem sastādīt vaicājumus.

Balstoties uz šo ideju, tika uzsākts darbs pie "Semantiskās Latvijas" īstenošanas dzīvē. Taču idejas realizācija tika veidota ejot no apakšas uz augšu. Tas ir, sākotnēji noritēja darbs pie semantisku datu izveides no relāciju datubāzēs esošiem datiem un vaicājuma rīka izstrādes, lai būtu iespējams pārliecināties, ka lietotājiem būs ieguvums darbojoties ar konceptuālā līmeņa datiem.

Tika īstenots projekts "Valsts programma „Latvijas iedzīvotāju dzīvildzi un dzīves kvalitāti apdraudošo galveno patoloģiju zinātniska izpēte ar daudznozaru pētnieciskā konsorcija palīdzību". Projekts Nr. 14 „Vienotas un visiem pētniekiem pieejamas datubāzes izveide par galveno dzīvildzi un dzīves kvalitāti apdraudošo patoloģiju un to riska faktoru izplatību Latvijas iedzīvotājiem"". Projekta laikā tika transformētas medicīnas domēnā pieejamās datubāzes no

relāciju datubāzu modeļiem uz RDF[1] datiem, kas tiek glabāti SPARQL[2] piekļuves punktos. Paralēli tika izstrādāts grafisks (diagrammatisks) vaicājuma rīks darbam ar SPARQL piekļuves punktos saglabātajiem datiem.

Projekta rezultātā tika izveidotas medicīnas domēna ontoloģijas, kas reprezentēja relāciju datu bāzēs pieejamos datus. Izveidotās ontoloģijas apskatāmas pielikumā 3. Relāciju datubāzu dati tika transformēti par RDF datiem un saglabāti Virtuoso[3] SPARQL piekļuves punktā. Ar ViziQuer[4] rīka pirmās versijas prototipu bija iespējams pieslēgties SPARQL piekļuves punktam un veikt vaicājumus pār tur esošajiem datiem.

No mediķiem tika saņemtas pozitīvas atsauksmes par sasniegtajiem rezultātiem. Medicīnas zinātniekus apmierināja iespējas veikt grafiskus vaicājumus, kas viņiem patika labāk par iespēju veikt tekstuālus SQL vaicājumus.

Taču saglabājās dažādas problēmas, lai realizēto pieeju varētu pārnest uz plašāku apgabalu, kā piemēram minēts rakstā [8], trūka efektīvu ietvaru, lai pārnestu relāciju datubāzu datus uz ontoloģijām. "Semantiskās Latvijas medicīnas" projektā tas tika darīts ar SQL procedūrām, kas izejā deva RDF datnes.

Ņemot vērā, ka tika saņemti pozitīvas atsauksmes no medicīnas domēna zinātniekiem, tad tika realizēts nākamais solis un tika izveidota pieeja un uzsākts jauns projekts „Semantisko datubāzu platforma nozaru speciālistiem" (nr. 2011/0009/2DP/2.1.1.1.0/10/APIA/VIAA/112), kura ietvarā tika izstrādāta pieeja – "Ontoloģijās balstītas informatīvās sistēmas".

Pieejas pamatā tika izvirzīta nostādne, ka ir nepieciešams rīks, kurš saņem ontoloģiju un izveido grafisku saskarni ontoloģijas datu labošanai. Ideālā gadījumā rīks pieslēdzas SPARQL piekļuves punktam, kurā tiek saglabāti izveidotie RDF dati. Tas pārklāj "Semantiskās Latvijas" tēzi par darbu ar ontoloģiju datiem.

Tika izveidots "Ontoloģijās balstītu informatīvo sistēmu" (OBIS)[5] prototips. Tā ir tīmeklī bāzēta sistēma, kas saņem ontoloģijas shēmas daļu un kas atbilstoši dotai ontoloģijai ģenerē tīmeklī bāzētu sistēmu, ar kuras palīdzību pievienot un labot ontoloģijas datus. Ontoloģijas dati tiek glabāti SPARQL piekļuves punktā. Lietotāji par tiem var sastādīt vaicājumus, izmantojot ViziQuer rīku.

[1] http://www.w3.org/TR/2004/REC-rdf-mt-20040210/
[2] http://www.w3.org/wiki/SPARQL
[3] http://virtuoso.openlinksw.com/
[4] http://viziquer.lumii.lv/
[5] http://obis.lumii.lv/

Noslēdzošais posms ir ontoloģijas repozitorija izveide, kas apvienotu ontoloģijas, ar kurām tālāk darboties. Promocijas darbā ir ieskicēti teorētiskie pamati ontoloģijas repozitorija izveidei.

Ņemot vērā, ka izstrādātās publikācijas tiek citētas citu pasaules zinātnieku vidū, tas norāda, ka darba tēma ir pasaulē aktuāla. ViziQuer rīks ir citēts [59], [63], [64].

Promocijas darba svarīgākie rezultāti

- Izveidota grafiska vaicājumu valoda, ar kuras palīdzību lietotāji var formulēt grafiskus vaicājumus, kas tiek vēlāk pārveidoti uz SPARQL vaicājumiem.
- Pierādīta grafiskās vaicājumu valodas lasāmība kā iesācēju lietotājiem, tā arī pieredzējušiem lietotājiem. Vaicājumu valodas lasāmība ir uzskatāma kā svarīgs aspekts, lai būtu iespējams izstrādāt praktisku prototipu, kas nodrošina tās ērtu rakstīšanu.
- Izveidoti divi vaicājumu valodas prototipi. Pirmais no tiem pārklāj vienkāršās valodas konstrukcijas un parāda tās lietojamību. Otrais vaicājumu valodas prototips pārklāj pierādītās, ērti lasāmās valodas konstrukcijas.
- Izveidota SPARQL vaicājumu metožu kopa, kas no patvaļīga SPARQL piekļuves punkta izgūst ontoloģijas shēmu, ļauj to vizualizēt un atvieglo tālāku grafisku vaicājumu sastādīšanu.

Bez uzskaitītajiem svarīgākajiem rezultātiem darbā tika aprakstīti arī mazāk nozīmīgi rezultāti:

- Aprakstīta "Ontoloģijās balstītu informatīvo sistēmu" pieeja, kas ļauj lietotājiem darboties ar ontoloģijas instanču datiem un tos saglabāt SPARQL piekļuves punktā.
- Aprakstīta „Semantiskās Latvijas medicīna" projekta īstenošanas gaita, tajā sasniegtie rezultāti un iegūtās atziņas.
- Aprakstīti teorētiskie pamati ontoloģiju repozitorija izveidei, kas padarītu pieejamu "Ontoloģijās balstītu informatīvo sistēmu" izstrādi plašākām masām un būtu solis uz priekšu, lai īstenotu sākotnējo Semantiskā tīmekļa vīziju par aģentiem, kas apkopu datus un piedāvā ērtāko risinājumu.

Promocijas darba rezultāti atspoguļoti publikācijās

- G.Barzdins, E.Liepins, M.Veilande, M.Zviedris, "Ontology Enabled Graphical Database Query Tool for End-Users", Selected papers from DB&IS'2008, Hele-Mai

Haav (Eds.), Frontiesrs in Artificial Intelligence and Applicatons series, IOS Press, 2009. 187:105 – 116 (SCOPUS)

- G.Barzdins, S.Rikacovs, M.Veilande, M.Zviedris, "Ontological Re-Engeneering of Medical Databases", Proceedings of the Latvian Academy of Sciences, Section B, Vol. 63 (2009), No. 4/5 (663/664), pp. 153–155 (SCOPUS)
- M.Zviedris, G.Barzdins, "ViziQuer: A Tool to Explore and Query SPARQL Endpoints", The Semantic Web: Research and Applications, LNCS, 2011, Volume 6644/2011, pp. 441-445 (SCOPUS)
- M.Zviedris. Ontology repository for User Interaction. In Mathieu d'Aquin, Alexander Garcia Castro, Christoph Lange, Kim Viljanen edititors, ORES-2010 Workshop on Ontology Repositories and Editiors for the Semantic Web, volume http://CEUR-WS.org/Vol-596/ .CEUR, 2010. (SCOPUS)
- G.Barzdins, E.Liepins, M.Veilande, M.Zviedris, "Semantic Latvia Approach in the Medical Domain", Proceedings of the 8th International Baltic Conference (Baltic DB&IS 2008), June 2-5, Tallin, Estonia, Tallinn University of Technology Press, 89.-102. lpp.
- G.Barzdins, S.Rikacovs, M.Zviedris, "Graphical Query Language as SPARQL Frontend", In Grundspenkis, J., Kirikova, M., Manolopoulos, Y., Morzy, T., Novickis, L., Vossen, G. (Eds.), Local Proceedings of 13th East-European Conference (ADBIS 2009), pp. 93–107. Riga Technical University, Riga. (2009)
- M.Zviedris, A.Romane, G.Barzdins, K.Cerans, "Ontology-Based Information System", Wooju K., Ying D. and Hong-gee K. (Eds.) The 3rd Joint International Semantic Technology Conference, LNCS, tiks publicēta (2014) (SCOPUS)

Referāti konferencēs

- The 3rd Joint International Semantic Technology Conference, Seula, Dienvidkoreja, "Ontology-Based Information System";
- 8th Extended Semantic Web Conference, Krēta, Grieķija, 2011, "ViziQuer: A Tool to Explore and Query SPARQL Endpoints";
- 1st International Workshop on Ontology Repositries and Editors for the Semantic Web (ORES-2010, Krēta, Grieķija, 2010, "Ontology repository for User Interaction", M.Zviedris;

- 13th East-European Conference on Advances in Databases and Information Systems, Rīga, 2009, „Graphical Query Language as SPARQL Frontend", G.Bārzdiņš, S.Rikačovs, M.Zviedris;
- 1. Datorzinātņu dienas, Ratnieki, 2011, "Semantiskā tīmekļa dati un meklēšana tajos", M.Zviedris;
- LU 69. Konference, Rīga, 2011, "ViziQuer – SPARQL piekļuves punktu apskatei", M.Zviedris, G. Bārzdiņš;
- LU 67. konference, Rīga, 2009., "Grafisko ontoloģiju vaicājumu rīks RDF datu bāzēm", G.Bārzdiņš, S.Rikačovs, M.Zviedris;
- LU 66. konference, Rīga, 2008., "Grafiska ontoloģiju vaicājumu valoda un tās realizācija", M.Zviedris, J.Bārzdiņš;

Promocijas darba struktūra

Pirmajā nodaļā sniegts pārskats par datu shēmu uzbūvi. To konceptuālo līmeni, fizisko līmeni, kā arī atšķirībām, kas parādās starp relāciju datu bāzēm un ontoloģijām.

Otrajā nodaļā izklāstīta vaicājumu valodas versija "ViziQuer full". Tā pārklāj lielāko daļu SPARQL valodas konstrukcijas un ļauj lietotājam sastādīt sarežģītus vaicājumus.

Trešajā nodaļā aplūkoti vaicājumu valodas "ViziQuer full" atklātie trūkumi, veiktie uzlabojumu un aprakstīta vaicājumu valodas "ViziQuer lite" versija. Kā arī aprakstīti eksperimenta rezultāti, kas pierāda vaicājumu valodas versijas "ViziQuer lite" lasāmību.

Ceturtajā nodaļā aprakstīts "Semantiskās Latvijas medicīnas" projekts, tā laikā izdarītie darbi, darba gaitā sastaptās problēmas, kā arī projekta laikā sasniegtie rezultāti.

Piektajā nodaļā apskatīta "Ontoloģijās balstītas informatīvās sistēmas" izstrādes metodoloģija un ieskicēta tās arhitektūra, kā arī attīstības perspektīvas.

Sestajā nodaļā izklāstīti ontoloģiju repozitoriju teorētiskie pamati, lai tos varētu īstenot praksē un padarītu "Semantisko Latviju" pieejamu lietotājiem.

Septītajā nodaļā aprakstīta vaicājumu valodas "ViziQuer" prototipu realizācija, kā arī risinājums shēmas izgūšanai no SPARQL piekļuves punkta, lai atvieglotu priekšā teikšanas iespējas vaicājumu sastādīšanas laikā.

1. DATU SHĒMAS UN DATI

Datu glabāšana, vaicāšana un pārskats ir attīstījies līdz ar datoru un to operētājsistēmu attīstību. Rakstā [2] minēts, ka vaicāšanas rīku attīstība notika jau laikā, kad bija pieejami tikai vienkārši teksta redaktori. Pirmā teorētiskā grafiskā vaicājumu valoda tika piedāvāta 1975. gadā [1].

Mūsdienās organizācijām ir grūti pastāvēt bez dažādām iekšējām datu uzkrāšanas sistēmām. Taču tajā pašā laikā praksē piedāvātie ietvari ir nedaudz par sarežģītiem domēna ekspertiem un parasti tiek pieaicināti palīgā programmētāji, kas veic specifisku sistēmu izstrādi vai palīdz ar datu atlases vaicājumu sastādīšanu.

Kā populārākie risinājumi nelielām organizācijām tiek lietoti, piemēram, MS Excel[6] vai MS Access[7], kas nodrošina vienkāršu datu glabāšanu un pieeju tiem.

MS Excel ir vienkāršākais risinājums, ko var lietot bez programmētāja palīdzības. Taču tad visdrīzāk būs iespējams izmantot to, lai darbotos ar divdimensionāliem datiem. Rakstā [9] ir aprakstīta pieeja, kā MS Excel izmantot arī saistītu datu glabāšanai, bet tā diez vai būs piemērota domēna lietotājiem bez programmēšanas iemaņām.

MS Access piedāvā plašākas iespējas, taču to realizācijai parasti tiks piesaistīti atbilstoši speciālisti. Tāpēc iespējas izveidot un lietot datu sistēmu bez programmētāju palīdzības ir ierobežotas.

1.1. Relāciju modeļa un ontoloģiju piemēra izklāsts

Lai labāk izprastu situāciju, izklāstīsim kā notiek datu saglabāšana un kādi ir svarīgākie aspekti, lai tos vēlāk papildinātu, labotu vai veiktu pār tiem vaicājumus.

Datu shēmas ļauj konceptuāli saprast pieejamos datus, kā arī efektīvi realizēt piekļūšanu tiem. Viena no populārākajiem datu aprakstīšanas modeļiem ir relāciju algebra un relāciju datu bāzes, kas ir realizētas balstoties uz šo algebru. Vairāk par relāciju algebru ir iespējams uzzināt resursā [10].

Populārākie praksē pieejamie datu glabāšanas un apstrādes risinājumi ir balstījušies uz relāciju datu bāzes modeli, piemēram pieminētais MS Access vai Oracle APEX[8]. Taču jāuzsver, ka relāciju datu bāzes modelis sastāv no vairākiem modelēšanas līmeņiem, kur tehniskās realizācijas līmenī parādās papildus tabulas un datu lauki, kas apgrūtina lietotāju iespējas izprast

[6] http://office.microsoft.com/lv-lv/excel/
[7] http://office.microsoft.com/lv-lv/access/
[8] http://apex.oracle.com/i/

sistēmas būtību, jo rīki piedāvā darboties tieši ar tehnisko realizāciju, nevis konceptuālo līmeni, kur tehniskie dati netiek iekļauti.

Relāciju datubāzu izstrāde notiek izmantojot ER modeļus [11]. Tas tiek iedalīts vairākos līmeņos, kur informācijas struktūras līmenis tiek izmantots, lai aprakstītu pieejamos datus. Šis līmenis parasti tiek dēvēts arī par konceptuālo līmeni. Tālāk tas tiek pārveidots par tehniskās realizācijas modeli, kas parāda – kā dati tiks glabāti relāciju datubāzē un kā savā starpā tie saistīsies.

Sākot datubāzes izstrādi, tiek izveidots konceptuālais modelis, un tas tiek minēts sistēmas izstrādes dokumentācijā (ja tāda tiek izveidota). Taču tālāk tiek realizēts tehniskais modelis, ar kuru notiek lietotāju interakcija un konceptuālais modelis ir saglabāts tikai uz papīra, vai arī tiek izveidotas sistēmai specifiskas lietotnes, kas nodrošina konceptuālā modeļa saskarni.

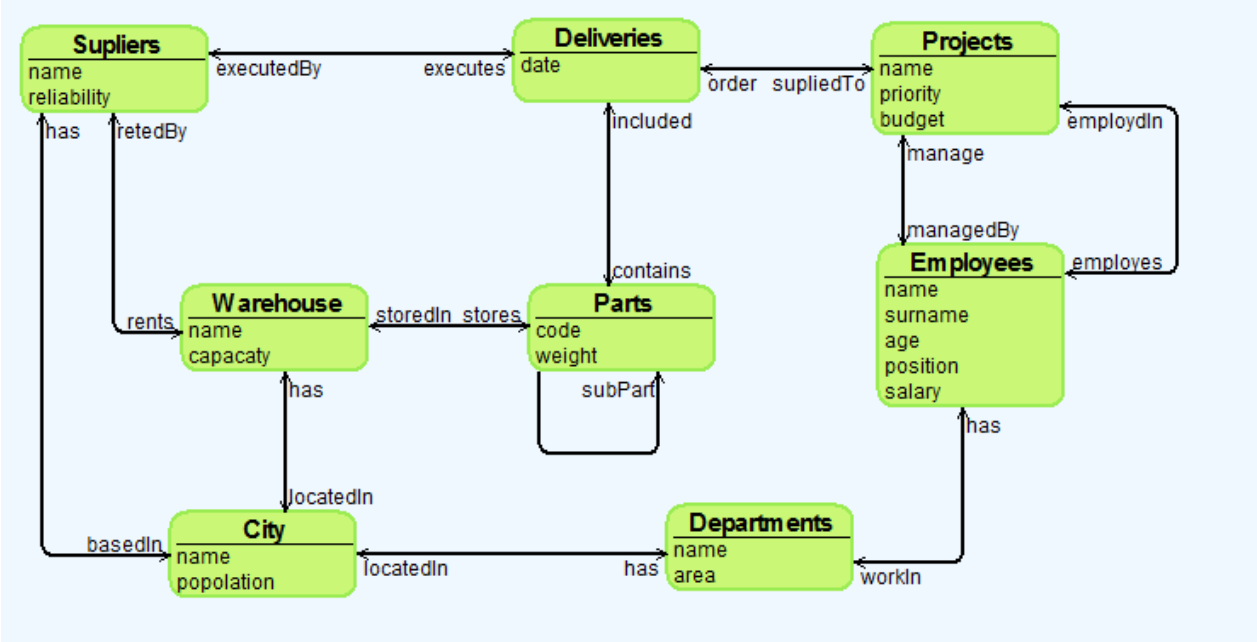
Lietotājs darbojas ar datiem, izmantojot programmētāju izstrādāto saskarni. Taču šādā gadījumā ir papildus izmaksas, jo ir jāpiesaista programmētājs, kas izveido katrai sistēmai specifisko starpslāni, kā arī vēlāk veic izmaiņas sistēmas uzbūvē un vēlāk arī palīdz uzturēt sistēmu.

Ja turpretim aplūkojam Semantiskā tīmekļa [6] lietotās tehnoloģijas, tad priekšplānā ir izvirzītas ontoloģijas, kuras var iedalīt vairākās struktūrās, taču pagaidām paliksim pie divām centrālajām – ontoloģijas shēmas daļu un ontoloģijas datiem. Populārākā valoda ontoloģiju pierakstam ir OWL[9], kam ir izstrādāts arī organizācijas W3C standarts. Medicīnas domēnā tiek lietota arī iespēja ontoloģijas pierakstīt OBO[10] formātā.

Lai labāk izprastu atšķirības starp relāciju datubāzēm un ontoloģijām, tad apskatīsim piemēru par uzņēmuma projektiem un piegādātājiem. Izklāstīsim to abos veidos, lai varētu labāk uzsvērt to atšķirības.

Dotajā piemērā aprakstīti uzņēmumā esošie projekti, darbinieki, kas tajos ir iesaistīti, kā arī piegādes, kas ir saistītas ar projektiem. Piegādēs ir atzīmētas detaļas, kas tajās ir iekļautas, kā arī piegādātājs. Darbiniekiem ir dota informācija par departamentu, kurā tas strādā. Papildus ir informācija arī par piegādātāju noliktavām un pilsētām, kurās tās atrodas. Konceptuālo datu modeli var apskatīt 1.1. attēlā.

[9] http://www.w3.org/TR/owl-features/
[10] http://www.geneontology.org/GO.format.obo-1_2.shtml



1.1. att. ***Datu konceptuālais modelis***

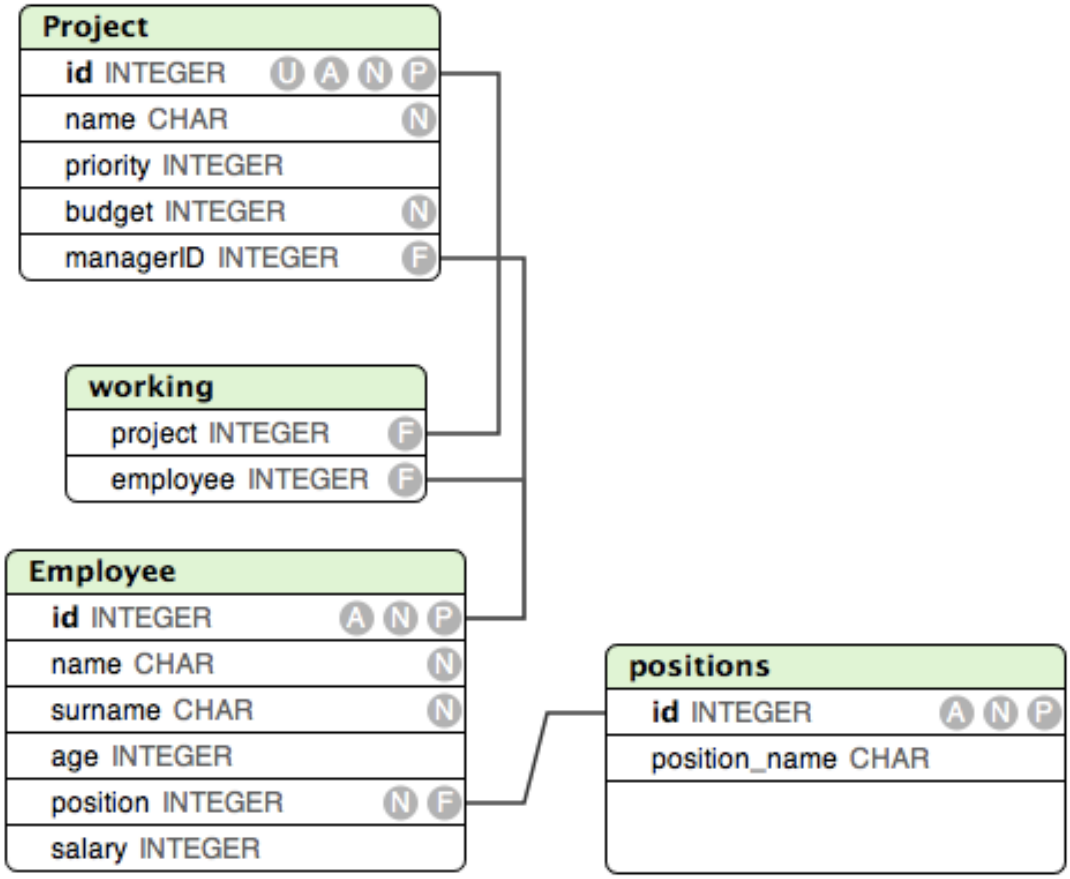
Lai izprastu atšķirības starp relāciju datu modeļa un ontoloģijām tehniskās realizācijas līmenī, tad apskatīsim tikai divas objektu klases un to savstarpējās saistības, tas ir, apskatīsim projektus un tiem piesaistītos darbiniekus – kā tie ir realizēti relāciju datubāzes tehniskajā līmenī un kāds ir realizācijas kods, kā arī kā tie ir realizēti ontoloģijās un kāds ir realizācijas kods.

Attēlā 1.2. varam apskatīt SQL dizaina izskatu (tehniskās realizācijas līmenis) dotajām klasēm. Ņemot vērā, ka uzņēmumā ir darbinieka pozīcijas ir predefinētas, tad to vērtībai ir jābūt vienai no dotajām.

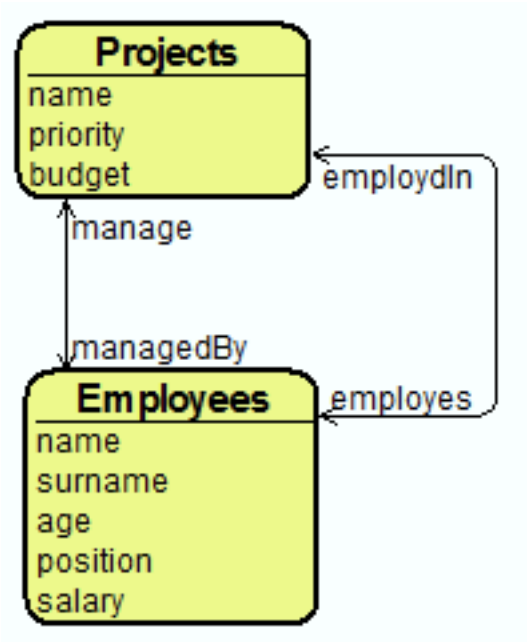
Salīdzinājumam attēlā 1.3. ir dota konceptuālā un arī tehniskā shēma vienlaicīgi ontoloģijas gadījumā. Šeit gan mēs izmantojam līdzīgu ontoloģijas vizualizāciju, kā tas ir darīts, piemēram, OWLGrEd[11] ontoloģiju redaktorā.

Pielikumā 1. ir dots abu realizāciju pieraksts tekstuālā sintaksē. Abos gadījumos uzdotās shēmas pārskatīšana tekstuālā sintaksē nav pārlieku ērta, lai arī SQL pieraksts ir nedaudz vienkāršāks. Tomēr abos gadījumos tika izmantots grafisks rīks, kas uzģenerēja tekstuālo sintaksi.

[11] http://owlgred.lumii.lv/



1.2. att. ***Projekta un darbinieku klašu SQL dizaina izskats***



1.3. att. ***Konceptuālais modelis ontoloģiju vidē***

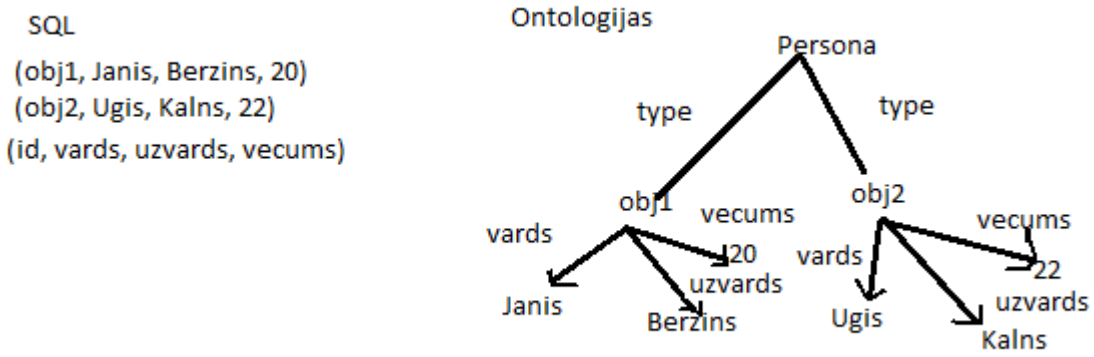
Aplūkosim atšķirības, kas parādās starp relāciju datu bāzēm un ontoloģijām, balstoties uz iepriekš izklāstīto piemēru.

Pirmkārt, kā notiek objektu identifikācija. Relāciju datubāzēs izmanto unikālos identifikatorus, ar kuru palīdzību tiek atpazīti objekti. Konceptuālā modelī tie neparādās, taču tehniskajā realizācijā tiem ir nozīmīga loma. Tie tiek ģenerēti automātiski, taču parādās kā atsevišķs lauks datu modeļa realizācijā.

Ontoloģiju gadījumā katram objektam tiek piesaistīts unikāls identifikators, taču tas ir URI[12], ar kura palīdzību objektu var globāli identificēt (ideālā gadījumā arī tam piekļūt). Pārējās objekta vērtības tiek piesaistītas šim identifikatoram.

Tas ir izskaidrojams arī ar abu pieeju atšķirīgo konceptuālo pieeju datu glabāšanai. SQL vidē tie ir korteži, kur visa objekta informācija tiek ievietota dotajā kortežā, kamēr ontoloģiju pasaulē viss tiek glabāts grafa formātā, kas tiek kodēts ar trijnieku palīdzību – subjekts, predikāts un objekts. 1.4. attēlā varam aplūkot, kā izpaužas minētā atšķirība.

Ontoloģiju gadījumā objekta identifikators tiek paslēpts, kā savienojošā virsotne, un tas tiek darīts konsekventi, kamēr SQL gadījumā tas var būt unikāls identifikators (piemēram, unikāls ID lauks), taču ir iespējami arī gadījumi, kad kortežu identificē ar vairāku objektu apvienojuma unikalitāti (piemēram, vārda un vecuma apvienojums ir unikāls), līdz ar to iespējamā realizācija nav viennozīmīga, jo tehnisko ID lauku ieviešana nav obligāta prasība, bet labā prakse, kas uzlabo sistēmas ātrdarbību.



1.4. att. ***SQL korteži un Ontoloģiju trijnieki***

Otrkārt, atšķirīgi izpaužas relācijas (asociācijas, saites) starp objektiem. Ontoloģiju gadījumā tās ir šķautnes, kas savieno divus objektus, līdzīgi kā objektam tika pievienoti atribūtu vērtības.

[12] http://en.wikipedia.org/wiki/Uniform_resource_identifier

Relācijas vārds tiek rakstīts uz šķautnes. Viena un tā paša nosaukuma relācija var tikt izmantota patvaļīgi daudz reižu un tā var savienot jebkādus divus objektus.

Turpretī relāciju modelī ir iespējami divas dažādas realizācijas. Pirmā iespēja ir tā sauktā <1 pret n> relācija, kad objekts no vienas grupas var atbilst vairākiem citiem objektiem otrā grupā, taču otrā grupā katram objektam atbilst tieši viens no minētās objektu klases, piemēram, katram projektam ir menedžeris, taču menedžeris var pārvaldīt vairāk nekā vienu projektu. Varam aplūkot piemēra realizāciju 1.2. attēlā, kur katram projektam ir *managerID* lauks, kurā ieraksta menedžera identifikatoru, kas to pārvalda.

Otrā objektu kopu savstarpējā attiecība ir tā sauktā <n pret n> relācija, kad katram objektam no pirmās kopas var atbilst vairāki objekti otrajā kopā, un objektiem otrā kopā arī ir iespējama saistība ar vairāk kā vienu objektu no pirmās kopas. Šādā gadījumā tiek ieviesta starp tabula, kurā tiek realizēta šīs atbilstības. Piemēram, darbinieks var strādāt vairākos projektos, kā arī katrā projektā nodarbina vairākus darbiniekus. 1.2. attēlā var aplūkot starp tabulu, kas attēlo minētā piemēra realizāciju.

Lai labāk izprastu, kā dažādās relācijas izpaužas relāciju datubāzu datu līmenī, tad 1.5. attēlā varam aplūkot vienkāršus datus, kas ir balstīti uz 1.2. attēlā redzamo datu shēmu.

Projekti

id	name	priority	budget	managerID
1	Bibliotēkas sistēm	3	5000	1
2	Banku sistēma	9	25000	1

nodarbinātība

project	employee
1	1
2	1
2	2
1	3
2	3

Darbinieki

id	name	surname	age	position	salary
1	Janis	Berzins	37	1	1000
2	Ugis	Kalns	27	3	300
3	Juris	Jansons	31	2	700

amati

id	position_name
1	manager
2	senior
3	assistant

1.5. att. ***Datu aizpildījuma piemērs.***

Treškārt, ir atšķirīgi klasifikatoru uzdošanas paņēmieni. Relāciju modelī klasifikatorus uzdod kā parastu tabulu un tā ne ar ko pārlieku neatšķiras no citām tabulām. Klasifikatoru tabulu raksturiezīmes parasti ir tas, ka tajās ir tikai dažas kolonas, kur pirmā apzīmē klasifikatora identifikatoru, bet otrā tā atšifrējumu, kā arī ir iespējams dažas papildinformācijas kolonas. Papildus arī tās ir saistītas ar <1 pret n> relāciju tikai ar vienu citu tabulu (retāk vairākām). Pēc šīm īpašībām tabulas puslīdz automātiski būtu iespējams atpazīt, taču ne pilnībā automātiski.

Ontoloģiju modelī klasifikatorus uzdod ar iespējamo vērtību pārskaitījumu. Ņemot vērā, ka grafā ir grūti realizēt sarakstu, tad tā tehniskais pieraksts ir visai sarežģīts, tomēr tas ir automātiski viegli atšifrējams un padara iespējamu datu pārskaitījumu attēlot vienkāršākā veidā nekā relāciju modelī. Ja klasifikators sastāv no vairākām saistītām vērtībām, tad var tikt izmantota arī atsevišķa klase, kurai tiek uzlikts nosacījums, kādas instances var būt dotajā klasē. Ontoloģijas pārskaitāmo vērtību saraksta piemēru varam aplūkot 1.6. attēlā.

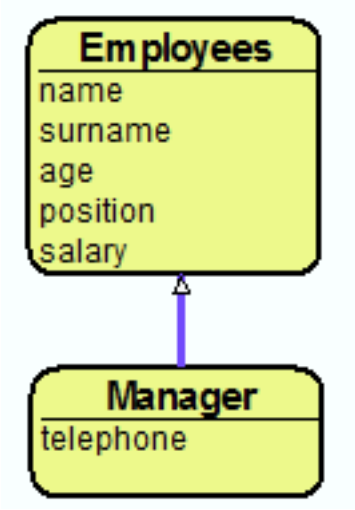
```
<!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#position -->
    <owl:DatatypeProperty rdf:about="#position">
        <rdfs:domain rdf:resource="#Employee"/>
        <rdfs:range rdf:resource="&xsd;string"/>
        <rdfs:range>
            <rdf:Description>
                <rdf:type rdf:resource="&owl;DataRange"/>
                <owl:oneOf>
                    <rdf:Description>
                        <rdf:type rdf:resource="&rdf;List"/>
                        <rdf:first rdf:datatype="&xsd;string">Assistant</rdf:first>
                        <rdf:rest>
                            <rdf:Description>
                                <rdf:type rdf:resource="&rdf;List"/>
                                <rdf:first rdf:datatype="&xsd;string">Manager</rdf:first>
                                <rdf:rest>
                                    <rdf:Description>
                                        <rdf:type rdf:resource="&rdf;List"/>
                                        <rdf:first rdf:datatype="&xsd;string"
                                            >SeniorProfessional</rdf:first>
                                        <rdf:rest rdf:resource="&rdf;nil"/>
                                    </rdf:Description>
                                </rdf:rest>
                            </rdf:Description>
                        </rdf:rest>
                    </rdf:Description>
                </owl:oneOf>
            </rdf:Description>
        </rdfs:range>
    </owl:DatatypeProperty>
```

1.6. att. ***Ontoloģiju pārskaitījuma pieraksts***

Ceturtkārt, ontoloģijas neierobežo, ka objekts pieder tikai vienai objektu kopai, kamēr relāciju pasaulē objekts pieder vienmēr tikai un vienīgi vienai relāciju kopai (tabulai). Līdz ar šīs īpašības esamību, ontoloģiju pasaule piedāvā arī vienkāršu secināšanas iespēju jeb objektu kopu savstarpēju iekļaušanu.

Tas ir, viena objektu kopa var būt apakšklase otrai objektu kopai. Tas nozīmē, ka visi objekti, kas pieder apakšklasei, uzreiz pieder arī tās virsklasei, turklāt tiem izpildās visas

virsklases īpašības, piemēram, tiem ir visi virsklases atribūti. Piemēru varam aplūkot 1.7. attēlā, kur ieviešam papildus klasi menedžeriem. Ņemot vērā, ka visi menedžeri ir arī darbinieki, tad tiem jau ir visas darbinieku īpašības, taču papildus ir arī telefona numurs, lai kritiskās situācijās ar viņiem varētu operatīvi sazināties.



1.7. att. ***Apakšklašu attiecība ontoloģijā***

Piektkārt, atšķiras objektu identificēšanas principi. Relāciju modelī identificēšana tiek veidota iekšēji modelī un vienādu objektu identificēšana vairākos modeļos, kur katrs glabājas citā vietā var būt visai problemātiska (izstrādājam noteiktus likumus, kas noteiks, kad objekti būs vienādi).

Kamēr ontoloģiju modelī objektu identificēšanai lieto URI, kas ļauj dažādām datu kopām atsaukties vienai uz otru jeb unikālo objektu visās datu kopās atzīmēt ar vienu un to pašu URI, kas vēlāk ļauj identificēt, ka tiek norādīts viens un tas pats objekts. Tas var atvieglot datu vaicājumus pār vairākām datu kopām, jo objekts tiks atpazīts, kā viens un tas pats visās datu kopās. Viens no pieņēmumiem ir, ka šī īpašība varētu būt ļoti noderīga, lai radītu viedus objektus [12].

Esam izklāstījuši galvenās modeļu atšķirības. Kā redzams, tad ontoloģiju gadījumā ir mazāk tehniskās informācijas vai arī tā ir daudz labāk noslēpta no lietotājiem.

1.2. Datu ievade, atlase

Neatkarīgi no izvēlētā modeļa tālāk vēlamies jau darboties ar datu instancēm. Pievienot jaunas, vai arī atlasīt jau esošās.

Jaunu instanču pievienošanai ir specifiskas tekstuālās komandas SQL un SPARQL, kas pievieno papildus datus, piemēram, SQL to var izdarīt ar INSERT INTO EMPLOYEE (Martins,

Lapa, 36, 3, 500), kas pievienos jaunu ierakstu pie darbiniekiem. Darbinieka ID tiks automātiski pievienots, un tas nav jānorāda.

Ontoloģiju pasaulē datu pievienošana notiek līdzīgi ar SPARQL valodas palīdzību un piemēru varam apskatīt 1.8. attēlā. Varam salīdzināt, ka tehniskās realizācijas līmenī ontoloģiju gadījumā ieraksta pievienošana ir daudz garāka, tomēr tā ir pārskatāma un lasāma.

```
PREFIX onto: <http://ontolibrary.org/company/>
INSERT DATA
{
                    onto:ML rdf:type onto:Employee.
                    onto:ML onto:name “Martins”.
                    onto:ML onto:surname “Lapa”.
                    onto:ML onto:age “36”.
                    onto:ML onto:position “assistant”.
                    onto:ML onto:salary “500”.
}
```

1.8. att. ***SPARQL UPDATE piemērs***

Taču datu pievienošana notiek ar ievades formu palīdzību, ko izveido programmētājs vai arī tās automātiski tiek ģenerētas balstoties uz datu shēmu, tāpēc lietotājam nav jāzina SQL komanda datu pievienošanai. Vairāk par to var izlasīt darba 5. nodaļā.

Datu ievade un to saglabāšana ir tikai pirmais posms darbā ar datiem. Tas, iespējams, ir posms, kurā lietotāji patērē ļoti daudz laika un tā ir laikietilpīgākā daļa darbā ar datubāzēm, tomēr datubāzu būtība slēpjas nākamajā solī, kad vēlamies ievadītos datus izmantot, lai veiktu tālākās analīzes. Datu atlase un to tālāka analīze jau ir kļuvusi par neatņemamu mūsdienu informācijas sabiedrības sastāvdaļu. Tā ir labā prakse, ja organizācijas savus lēmumus cenšas pamatot ar datu analīzi.

Taču lietotājiem, kas nav ieguvuši izglītību IT jomā un labi apguvuši datu atlases valodu, piemēram, SQL, parasti pietrūkst prasmju un zināšanu, lai pašu spēkiem atlasītu nepieciešamos datus. Apgūt un lietot tekstuālas datu atlases valodas šiem lietotājiem praktiski ir ļoti sarežģīti – tiek piesaistīti speciāli programmētāji, kas nodarbojas ar datu atlases pārformulēšanu no lietotāja vēlamā jautājuma uz tekstuālu SQL vaicājumu.

Jāatzīmē, ka šādos gadījumos var rasties kļūdas, ja programmētājs nepareizi izprot lietotāja vēlamo vaicājumu. Lai domēna speciālisti paši varētu ātri un vienkārši veikt datu atlases, ir nepieciešams labāks risinājumus par tekstuālām valodām, kas tiem ļautu, ar iespējami mazu apmācības laiku, veikt vienkāršu datu atlases vaicājumus, kuru rezultātus viņi tālāk varētu analizēt, lai pieņemtu lēmumus.

Piemēram, tiek izstrādāti speciāli risinājumi, piemēram, biznesa inteliģences sistēmas [13], kas apvieno organizācijā pieejamos datus un ļauj tās darbiniekiem pārskatīt datus. Taču tas bieži ir laikietilpīgs un dārgs process. Tāpēc būtu jāmeklē ērtāki un vienkāršāki risinājumi.

Mūsuprāt, risinājums šai problēmai ir grafiskas vaicājumu valodas. Pirmkārt, grafiskām valodām ir ļoti grūti ievadīt nepareizu sintaksi, jo grafiskais redaktors parasti ir pieskaņots valodas sintaksei un citus elementus tas neļauj ievadīt – darbs norit tikai ar valodas elementiem. Tas atvieglo lietotājiem sastādīt korektus vaicājumus, kur bieži tekstuālajā sintaksē rodas problēmas, jo ir iespējams pieļaut dažādas vienkāršas sintakses kļūdas.

Otrkārt, vienkāršās valodas konstrukcijas ir viegli saprotamas un lasāmas, kas atvieglo vienkāršu vaicājumu pareizu un ātru sastādīšanu, jo lietotājs samērā viegli saprot vaicājuma loģiku, ko pats ir uzrakstījis. Arī esošie zinātniskās iestrādes atspoguļo, ka lietotāji spēj precīzāk un ātrāk sastādīt vaicājumus grafiskā formātā [3].

Arī grafiska sintakse ir pārskatāma, kas lietotājam ļauj patstāvīgi atrast un novērst loģiskās kļūdas, ja tādas ir pieļautas, sastādot vaicājumu. Papildus priekšrocība var būt arī grafisku valodu vaicājumu atkal izmantošana, jo grafiskā vaicājumā var samērā ātri saprast tā būtību un izmantot jau kāda rakstītu vaicājumu, nevis sastādīt to no jauna.

Apzināmies, ka šajā jomā ir izstrādāti dažādi risinājumi, piemēram [14, 15]. Taču tajā pašā laikā tie ir balstīti uz relāciju datubāzēm, kas satur tehnisko līmeni, kas lietotājiem apgrūtina darbu.

Mēs uzskatām, ka Semantiskajā tīmeklī un ontoloģijās ir iespējams izstrādāt ērtākas vaicājumu valodas, jo tur konceptuālais līmenis sakrīt ar tehnisko līmeni, kā dati ir reprezentēti.

Izstrādājam vaicājumu valodu ViziQuer, kurai ir vairākas versijas. ViziQuer Full valodas versijā, galvenais uzsvars ir likts uz aspektu – iespējami daudz pārklāt SPARQL tekstuālās konstrukcijas ar grafiskiem, vizuāliem elementiem. ViziQuer Full valodas versija ir aprakstīta 2. nodaļā.

Pilnīgākas versijas attēlošana ar dažādiem grafiski atšķirīgiem elementiem ļauj izteiksmīgāk aprakstīt vaicājumus un lietot dažādas vaicājuma daļas, kas procentuāli vaicājumos netiek tik bieži lietotas. Piemēram, SPARQL specifikācijā esošo konstrukciju UNION, lai apvienotu atbildes no vairākiem vaicājuma grafiem, ko vaicājumu sastādīšanai, piemēram, Dbpedia piekļuves punktā lieto tikai 11% gadījumu [16]. Turklāt, tā vairāk ir piemērota, lai apvienotu datus, ja tie ir līdzīgi.

Taču dažādas grafiskas konstrukcijas arī apgrūtina vaicājumu lasāmību un saprotamību, kā arī ir grūtāk apmācīt lietotājus, kam šī vaicājumu valoda ir paredzēta – lietotāji bez plašām priekšzināšanām datu bāzu atlases veidošanā.

Balstoties uz pilno ViziQuer vaicājumu valodu un analīzes datiem, ko esam ieguvuši no lietotājiem bez priekšzināšanām par SPARQL, izveidosim un secīgi aprakstīsim vaicājumu valodas ViziQuer lite versiju, kas ir aprakstīta 3. nodaļā.

Lielā mērā ViziQuer lite versijas izveidi iespaidoja aptaujas, kurās aicinājām aptaujas dalībniekus izklāstīt kā viņi saprot uzzīmētos vaicājumus un vai tie ir pietiekami ātri un precīzi saprotami.

2. VIZIQUER FULL

Šajā nodalāt aprakstīsim grafiskās vaicājumu valodas ViziQuer Full grafiskos elementus un kā tie var tikt translēti uz SPARQL.

2.1. Valodas termini

2.1.1. Valodas mainīgie

Valodas pieejā tiek ieturēta konsekvence, ka visi mainīgie sākas ar jautājuma zīmi, neatkarīgi no vietas, kur tie tiek lietoti. Objektu kopu gadījumā var piekļūt mainīgo atribūtiem, ja atribūtu no mainīgā atdala ar punktu, tādā veidā rodas navigācija pa saistītiem elementiem.

2.1.2. Piemēros lietotās vārdu telpas

Vārdu telpas tiek izmantotas, lai saīsinātu pierakstāmos terminus. Visi objekti semantiskajā tīmeklī tiek definēti izmantojot URI, kur pirmā daļa ir vārda telpa, bet otrā daļa ir konkrētais identifikators vārda telpas ietvaros.

- Onto – reprezentē ontoloģijas vārdu telpu, pār kuru veidosim vaicājumu, tiek uzskatīta par noklusēto vārdu telpu.
- Konf – reprezentē ontoloģiju vārdu telpu, kurā ir konferenču klase. Visas pārējās klases atrodas Onto vārdu telpā.
- RDF - http://www.w3.org/1999/02/22-rdf-syntax-ns# - reprezentē RDF valodas definīcijas vārdu telpu.
- XSD - http://www.w3.org/2001/XMLSchema# - reprezentē XML shēmas vārdu telpu, izmanto datu tipu definīcijām.

2.2. Objektu un objektu kopas atlase

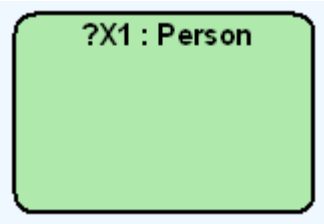
Semantiskā tīmekļa raksturīgā iezīme ir tāda, ka ir individuāli objekti, kas nepieder nevienai objektu klasei vai arī objekti, kas pieder vienai vai vairākām klasēm. Objektiem ir dažādas atribūtu vērtības, kā arī saites ar citiem objektiem.

Par primāro valodas ViziQuer elementu tiek ņemtas objektu kopas, kur visi objekti pieder kādai noteiktai klasei, jo vēlamies darboties ar datiem, kas glabājas atbilstoši kādai ontoloģijai.

Objektu kopas atlases grafiskais elements tiek attēlots kā taisnstūris ar noapaļotiem stūriem. Tam ir trīs sadaļas – augšā ir objektu kopas mainīgā vārds un klases vārds, kas ierobežo

objektu kopu. Zem tā ir sadaļa, kurā tiek ierakstīti objektu kopas atribūtu ierobežotāji. Pēdējā sadaļā, kas no augšējām divām tiek atdalīta ar svītru, tiek ierakstīti visi tie atribūtu vārdi, ko vēlamies izmantot atbildes tabulas veidošanā.

Piemērā, kas attēlots 2.1. attēlā, atlasām visus personas klases objektu, un šo objektu kopu apzīmējam ar mainīgo X1. Tālāk varēsim izmantot šo objektu kopas atlasi, lai veiktu sarežģītākas darbības, kas tiks aprakstītās nākamajās sadaļās. Kā arī jebkurā brīdī varēsim atsaukties un izmantot mainīgo X1, lai veiktu papildus ierobežojumus šiem, atlasītajiem, objektiem.



2.1. att. ***Klases atlases elements***

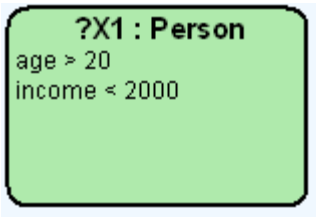
SPARQL sintakse: `?X1 RDF:type Onto:Person`

2.2.1. Objektu kopas ierobežošana

Reti ir nepieciešamība darboties ar visiem objektu kopas elementiem; biežāk vēlamies ierobežot objektu kopu un tikai tad veikt tālākas darbības. Ja šādi ierobežojumi ir uzstādīti, tad tie tiek parādīti tieši zem kopas atlases.

Ierobežojumus var uzstādīt uz objektu kopas dažādajiem atribūtiem un tiek attēloti tādā veidā, ka vispirms ir atribūts un tālāk seko nosacījums, kas izpildās visiem atlasītajiem atribūtiem. Atribūta nosacījumā ir iespējams lietot ierobežojumus, kas ir pieejami SPARQL definīcijā. Visiem atlasītajiem objektiem izpildās visi nosacījumi, kas ir definēti, tas nozīmē, ka starp nosacījumiem tiek lietota AND operācija.

Ja vēlamies rakstīt sarežģītāku atlases nosacījumu, tad to ir iespējams darīt atsevišķā nosacījumu kastē, kas aprakstīta nodaļā par vaicājumu atbildes formēšanu un atkārtotu izmantošanu. Piemērā, kas attēlots 2.2. attēlā, tiek vispirms atlasītas visas personas, tad tiek piemērots ierobežojuma filtrs, ka tās ir vecākas par 20 gadiem un ka tām ir ienākumi, kas ir mazāki par 2000.



2.2. att. ***Klases atlase ar atribūtu nosacījumiem***

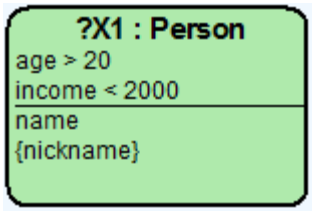
SPARQL sintakse:

```
?X1 RDF:type Onto:Person.
?X1 Onto:age ?age.
?X1 Onto:income ?income.
FILTER (?age > XSD:integer("20") &&
?income < XSD:integer("2000"))
```

2.2.2. Atribūti atbildes tabulas veidošanai

Vaicājuma gala mērķis ir iegūt datu tabulu, ko izmantot tālākā analīzē, tāpēc ir nepieciešamas atzīmēt vērtības, ko sagaidām gala rezultātā. Atribūti, ko vēlamies redzēt atbildē tiek attēloti objektu klases kastes apakšējā daļā. Tie tiek atdalīti ar svītru no augšējās daļas, kur atrodas klases definīcija un vērtību ierobežojumi, ja tādi ir uzdoti.

Tā kā pastāv iespēja, ka atribūtu vērtības nav definētas, tad ir divas atribūtu atlases iespējas – atlasīt tikai definētos atribūtus, vai arī atlasīt atribūtu vērtības un tur, kur tie nav definēti atstāt atbildes tabulas laukus neaizpildītus. 2.3. attēlā ir piemērs, kur iepriekšējam vaicājumam papildus vēlamies noskaidrot to personu vārdus un atbilstošās iesaukas, ja tādas ir zināmas. Kā redzam, tad atribūti, kuru vērtībām ir jābūt obligāti definētām tiek attēloti vienkārši kā atribūta vārds, bet neobligātie atribūti, kuru vērtības atbildes tabulā var būt tukši lauki, tiek likti figūriekavās.



2.3. att. ***Klases atribūtu izvēle atbildei***

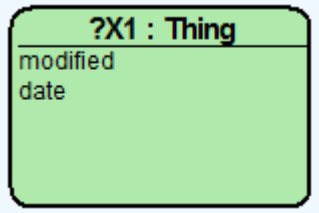
SPARQL sintakse:

```
SELECT ?name ?nickname WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
OPTIONAL {?X1 Onto:nickname ?nickname.}
?X1 Onto:age ?age.
?X1 Onto:income ?income.
FILTER (?age > XSD:integer("20") &&
?income < XSD:integer("2000")) }
```

2.2.3. Objektu atlase bez klases definīcijas

Ne visas semantisko datu piekļuves punkti satur labi definētus datus, kur visiem objektiem būtu definēti klašu tipi. Veicot datu atlasi, piekļuves punktos ar slikti definētiem datiem vajag izmantot objektu atlases kopu, kas nedefinē klases tipu.

Ir iespējams objektu kopas mainīgajam nenorādīt klases tipu, tādā gadījumā tiks atlasīti visi iespējamie objekti. Lai to izdarītu, ir jānorāda, ka vēlas atlasīt no *Thing* klases, jo tā pēc definīcijas satur visus objektus, kas ir definēti datos. Piemērā, kurš norādīts 2.4. attēlā, tiek atlasīti visi objekti, kam ir atribūti, kur norādīts datums un labošana. Šo objektu klasi tālāk varam lietot arī sasaistītu ar relāciju pie citām, noteikta tipa objektu klasēm.



2.4. att. ***Vispārīga objektu atlase***

SPARQL sintakse:

```
SELECT ?modified ?date WHERE {
?X1 Onto:modified ?modified.
?X1 Onto:date ?date.}
```

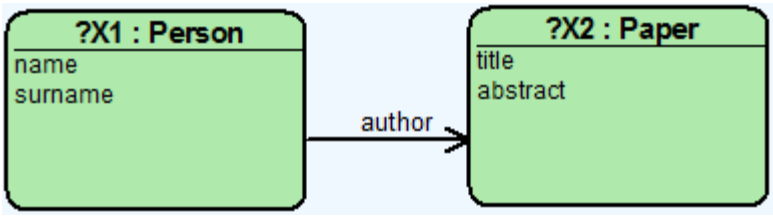
2.3. Objektu kopu sasaite

Objektu kopas savā starpā ir saistītas ar relācijām, kas apraksta objektu savstarpējās attiecības. Lai veidotu sarežģītākus vaicājumus, tad vēlamies relācijas izmantot grafiskajā vaicājumu valodā, lai varētu attēlot objektu savstarpējo sasaisti.

2.3.1. Objektu noteiktā sasaite

Atlasot objektus no vairākām kopām un tos savā starpā sasaistot ar bultu, atlasām objektus, kam izpildās arī savstarpējā saistība. Virs sasaistes bultas tiek rakstīts relācijas vārds, kas izmantots, lai objektus savā starā saistītu.

Piemērā, kas parādīts 2.5. attēlā, esam atlasījuši visu personu kopu kā ?X1 un atzīmējuši, ka gribam redzēt personu vārdus un uzvārdus. Kopā ?X2 esam atlasījuši visas zinātniskās publikācijas un atzīmējuši, ka vēlamies redzēt to nosaukumus un abstraktus. Ar relāciju „autors" sasaistām abas kopas savā starpā, kas nozīmē, ka visām personām ?X1 izpildīsies, ka tās ir sarakstījušas kādu publikāciju no kopas ?X2.



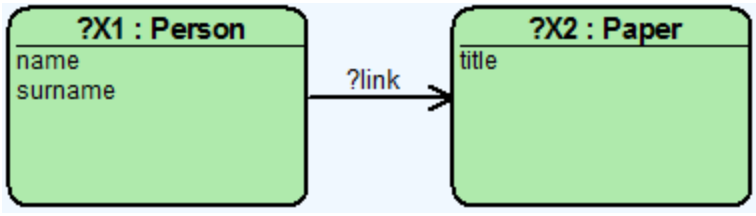
2.5. att. ***Savstarpēji savienotas klases***

SPARQL sintakse:

```
SELECT ?name ?surname ?title ?abstract WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
?X2 RDF:type Onto:Paper.
?X1 Onto:author ?X2.
?X2 Onto:title ?title.
?X2 Onto:abstract ?abstract.}
```

2.3.2. Sasaistes vārda noteikšana

Atsevišķos gadījumos objektus kopas savā starpā ir saistītas ar dažādu vārdu saitēm, piemēram, persona var būt publikācijas autors, taču iespējama arī otra saite, ka persona ir publikācijas recenzents. Tāpēc varam vēlēties noskaidrot arī saites vārdu, kā objekti savā starpā ir sasaistīti. Varam skatīt piemēru, kas attēlots 2.6. attēlā, kur vēlamies noskaidrot, kā tieši persona ir saistīta ar publikāciju.



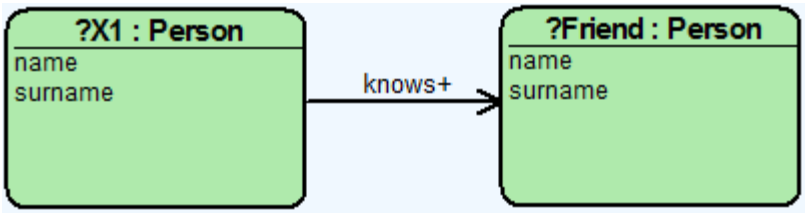
2.6. att. ***Relācijas vērtības noteikšana starp objektu klasēm***

SPARQL sintakse:

```
SELECT ?name ?surname ?link ?title WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
?X2 RDF:type Onto:Paper.
?X1 ?link ?X2.
?X2 Onto:title ?title.}
```

2.3.3. Patvaļīga ceļa pierakstīšana

SPARQL 1.1. piedāvā iespēju rakstīt sasaistes lomas vietā arī dažādas sarežģītākas izteiksmes – *„Property path”*. Grafiskā valodā lietotājs šīs izteiksmes var rakstīt sasaistes lomas vārda vietā. To notācija būs tieši tāda pati, kā parastai saistībai, tikai viena lomas vārda vietā tiks attēlota visa, lietotāja ierakstītā, atlases ceļa izteiksme. Piemēram, vēlamies atlasīt visas tās personas, kuras ir personas paziņu grafā bezgalīgā attālumā. Grafiski piemērs attēlots 2.7. attēlā.



2.7. att. ***„Property path" lietojums lomas vārda vietā***

SPARQL sintakse:

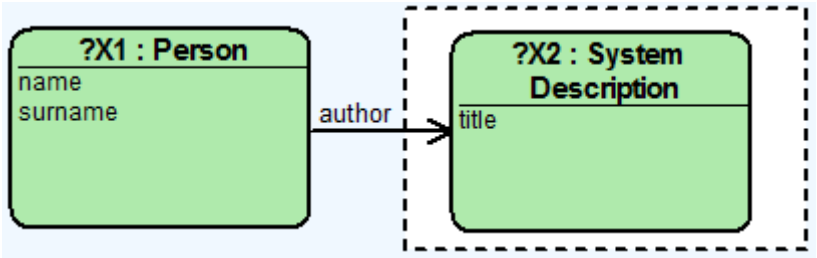
```
SELECT ?x1name ?x1surname ?frienname ?friendsurname
WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?x1name.
?X1 Onto:surname ?x1surname.
? Friend RDF:type Onto:Person.
?X1 Onto:knows+ ?Friend.
? Friend Onto:name ?friendname.
? Friend Onto:surname ?friendsurname.}
```

2.4. Vaicājuma grupēšanas iespējas

2.4.1. Neobligātā grupa

Ir gadījumi, kad nezinām vai sasaiste vienmēr pastāv, taču vēlamies atlasīt tos objektus, kuriem sasaite nav definēta. Tāpēc tiek ieviesta neobligātā grupa, kas ļauj piesaistīt objektu kopas, kuras var būt, bet var arī nebūt, kas neietekmē visu pārējo atlasi.

Piemērā, kas attēlots 2.8. attēlā, tiek atlasītas visas personas ar to vārdiem un uzvārdiem, taču gadījumos, kad tie ir autori arī kādas sistēmas aprakstam, tad tiek atlasīts arī sistēmas apraksts. Tas nozīmē, ka atbildes tabulā būs 3 kolonas, no kurām vārds un uzvārds būs aizpildīts pilnīgi visās rindās, bet virsraksta kolona būs aizpildīta tikai tajās rindās, kur persona būs autors kādam sistēmas aprakstam. Citās rindās šīs kolonas vērtība būs tukša.



2.8. att. ***Neobligātā grupa vaicājumā***

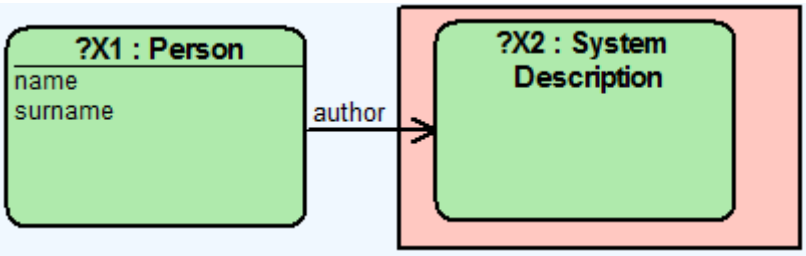
SPARQL sintakse:

```
SELECT ?name ?surname ?title WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
OPTIONAL {?X2 RDF:type Onto:SystemDescription.
?X1 ?link ?X2.
?X2 Onto:title ?title.}}
```

2.4.2. Nolieguma grupa

Ja vēlamies veikt apgalvojumu, kur kādam objektam nav sasaiste ar citiem objektiem, tad ir jālieto nolieguma grupa. Tas nozīmēs, ka sasaistītajiem objektiem nepastāv saite ar objektiem, kas ir atlasīti negācijas grupā.

Piemērā, kas attēlots 2.9. attēlā, tiek atlasītas tikai tās personas, kas nav autori sistēmas aprakstam. Ja persona būs autors kādam sistēmas aprakstam, tad viņš atbildes tabulā neparādīsies. Varam to uztvert arī tā, ka atlasam tās personas no iepriekšējā piemēra vaicājuma, kurām pēdējā kolonā vērtība nav definēta.



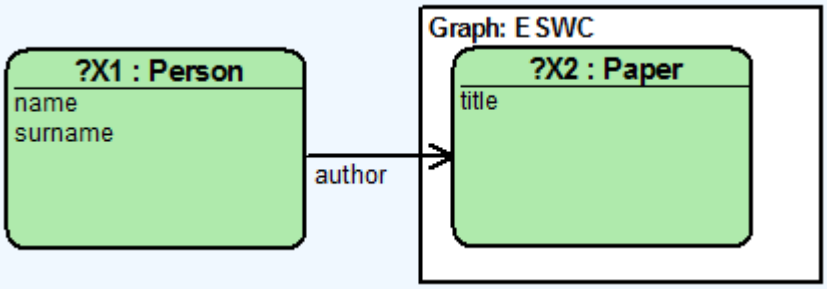
2.9. att. ***Nolieguma grupa vaicājumā***

SPARQL sintakse:

```
SELECT ?name ?surname ?title WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
OPTIONAL {?X2 RDF:type Onto:SystemDescription.
?X1 ?link ?X2.
?X2 Onto:title ?title.}
FILTER (!bound(?X2))}
```

2.4.3. Grafa norādīšana grupai

Semantiskie dati vienā datu piekļuves punktā tiek glabāti dažādos grafos, tāpēc vēlamies vaicājumus veikt pār dažādiem grafiem. Piemēram, piekļuves punktā par konferenču datiem, katrai konferenču sērijai ir savs grafs. Ja vēlamies atlasīt personas, kurām ir raksts kādā no ESWC konferencēm, tad uzliekam nosacījumu, ka raksts atrodas grafā ESWC. Grafiski piemērs attēlots 2.10. attēlā.



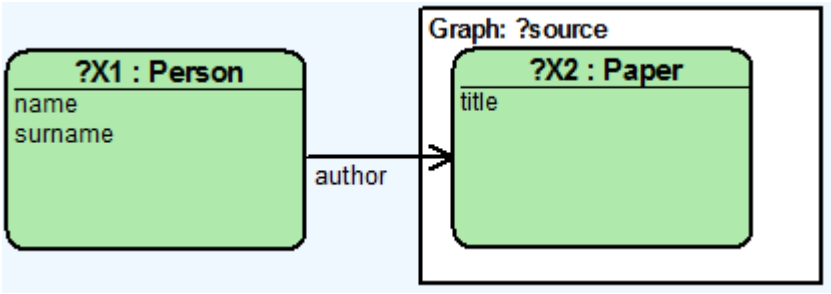
2.10. att. ***Grafa norādīšana vaicājumā***

SPARQL sintakse:

```
SELECT ?name ?surname ?title WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
Graph Onto:ESWC {
      ?X2 RDF:type Onto:Paper.
      ?X1 Onto:author ?X2.
      ?X2 Onto:title ?title} }
```

2.4.4. Grafa atrašana grupai

Gadījumos, kad dati ir vairākos grafos, tad ir vēlme arī atlasīt grafu, kurā ir derīgie dati, tāpēc varam uzstādīt arī jautājumu, par grafu, kurā ir atbildes vērtības. Piemēru skatīt 2.11. attēlā, kurā vēlamies noskaidrot grafu, kurā atrodas dati par rakstiem, kuriem persona ir autors.



2.11. att. ***Noskaidrot, kurā grafā atrodas dati***

SPARQL sintakse:

```
SELECT ?name ?surname ?title ?source WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
Graph ?source {
      ?X2 RDF:type Onto:Paper.
      ?X1 Onto:author ?X2.
      ?X2 Onto:title ?title}
}
```

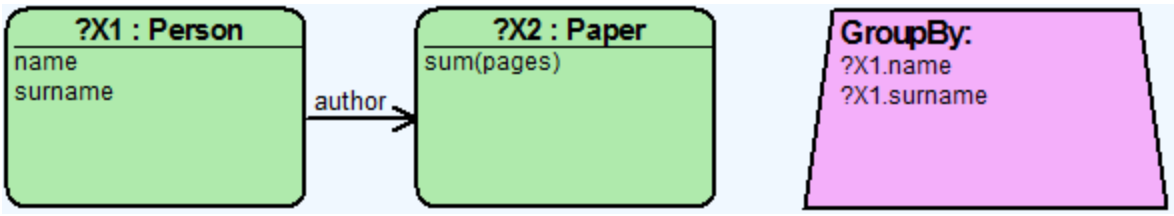
2.5. Vaicājuma papildiespējas

Katram vaicājumam ir iespējamas vairākas statiskas kastes, kas var parādīties vienā eksemplārā. Papildu iespējas ir domātas sarežģītāku vaicājumu sastādīšanai un vienkāršākos gadījumos var iztikt bez to lietošanas.

2.5.1. *Agregāta funkcijas un atbildes grupēšana*

Reizēm ir nepieciešams atlasīt atribūtu vērtības, ar kurām ir veiktas elementāras aritmētikas darbības, piemēram, summēšana vai unikālo vērtību saskaitīšana. Šādu iespēju nodrošina SPARQL iebūvētās agregāta funkcijas.

Jāņem vērā fakts, ka pielietojot agregāta funkcijas ir nepieciešams norādīt, kādi atribūti tiek grupēti, lai zinātu, kādās grupās ir nepieciešams veikt agregāta funkciju darbības. Tāpēc papildus ir jāaizpilda arī grupēšanas kaste, kur norāda atribūtus, pēc kuriem veikt grupēšanu. Piemērs, kas attēlots 2.12. attēlā, demonstrē vaicājumu, kur vēlamies atrast personas vārdu, uzvārdu, kā arī saskaitīt lappušu skaitu personas publicētajos rakstos.



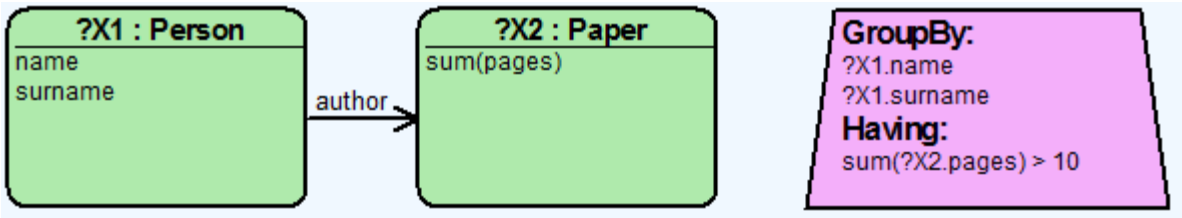
2.12. att. ***Agregāta funkcijas un grupēšanas iespējas***

SPARQL sintakse:

```
SELECT ?name ?surname sum(?pages) WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
?X2 RDF:type Onto:Paper.
?X1 Onto:author ?X2.
?X2 Onto:pages ?pages}
} GROUP BY ?name ?surname
```

2.5.2. *Filtrs pār grupēta vaicājuma agregāta funkcijas vērtīgu*

Ja vēlamies pielietot filtru kādai agregāta funkcijas vērtībai, tad tas nozīmē, ka lietojam filtra vērtību pār visu vaicājumu. Šī filtrēšanas iespēja ir atzīmēta grupēšanas kastē, kā *Having* sadaļa. Šeit lietotājs pieraksta visus filtrus pār visu vaicājumu. Attēlā 2.13. var aplūkot iepriekšējo vaicājumu, kas ir papildināts ar nosacījumu – ka lappušu skaitam katrai personai jābūt lielākam par 10.



2.13. att. ***Agregāta funkcijas un grupēšanas iespējas***

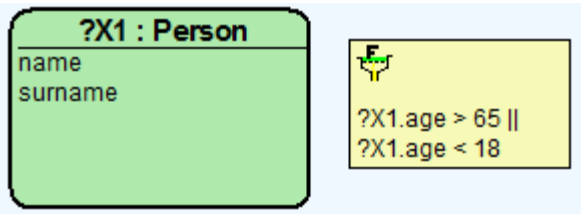
SPARQL sintakse:

```
SELECT ?name ?surname sum(?pages) WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
?X2 RDF:type Onto:Paper.
?X1 Onto:author ?X2.
?X2 Onto:pages ?pages}
} GROUP BY ?name ?surname
HAVING sum(?pages) > 10
```

2.5.3. Vispārīgs filtrs

Iespējas pie katras atlasītās kastes definēt atlases nosacījumus, kā tas ir parādīts sadaļā par objektu kopas nosacījumu, definēšanas ir visai ierobežotas, jo varam pielietot tikai AND operāciju. Reizēm lietotājiem ir nepieciešams sastādīt sarežģītākas atlases iespējas, lai to nodrošinātu ieviešam papildus speciālu kasti, kur lietotājs var brīvi rakstīt ierobežojošu nosacījumus, kas atbilst SPARQL filtra izteiksmes. Vienīgā atšķirība ir tā, ka filtra izteiksmē ir jāatsaucas uz mainīgajiem, kas ir ieviesti grafiskajā izteiksmē.

Attēlā 2.14. varam apskatīt vaicājuma piemēru, kur vēlamies atlasīt personas, kas nav formāli darbspējīgas, jo vēl nav sasniegušas pilngadību vai jau atrodas pensijas vecumā.



2.14. att. ***Grafisks vaicājums ar filtra izteiksmi***

SPARQL sintakse:

```
SELECT ?name ?surname WHERE {
?X1 RDF:type Onto:Persona
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
?X1 Onto:age ?age.
FILTER (?age > xsd:integer(65)
||?age < xsd:integer(18) )}
```

2.5.4. Vārdu telpas definīcija

Bieži vaicājuma sastādīšanai tiek izmantotas dažādas vārdu telpas. Ir speciāla kaste, kurā definējam katrai vārdu telpai savu saīsinājumu, lai vaicājumu lasot varētu atpazīt, no kuras vārdu telpas ir dotais elements.

Attēlā 2.15. ir nodemonstrēts piemērs, kur kastē ar apzīmējumu N ir definētas vārda telpas. Lai atvieglotu vaicājuma lasīšanu – lietotājs norāda ar figūriekavām, kuras vārdu telpas vēlas redzēt vaicājuma objektu klasēs. Piemēram, vārdu telpu Onto lietotājs nevēlas likt klāt pie klasēm, bet Konf vārdu telpu vēlas attēlot. Pie objektu klasēm vārdu telpa tiek likta pirms klases vārda un atdalīta no tā ar simbolu #.

2.15. att. ***Vārdu telpu lietojums vaicājumos***

SPARQL sintakse:

```
Prefix Onto:  <http:\\web.person.com\>
Prefix Konf:  <http:\\conferences.com\>
SELECT ?name ?surname ?title WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.
?X2 RDF:type Konf:Paper.
?X1 Onto:author ?X2.
?X2 Konf:title ?title}
```

2.6. Vaicājumu atbildes formēšana un vaicājumu atkārtota izmantošana

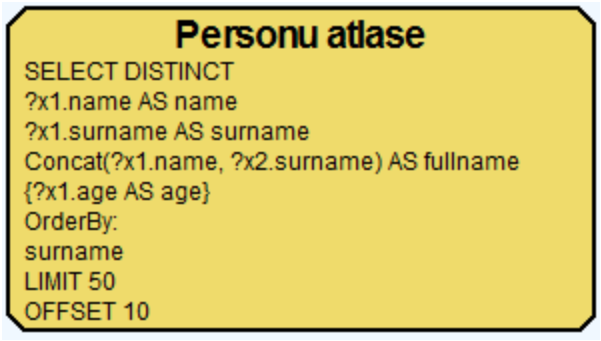
2.6.1. Vaicājuma atbildes formēšana

Pēc vaicājumu sastādīšanas un pirms izpildes ir nepieciešams noformēta tā atbildes tabulu. Atbildes formēšanā tiek piedāvātas visas tās pašas iespējas, kas SPARQL valodas atbildes definēšanai ar *SELECT*. Piemēru varam apskatīt 2.16. attēlā.

Katram vaicājumam ir lietotāja piešķirts nosaukums, lai vēlāk pēc šī nosaukuma varētu uz vaicājumu atsaukties, šajā gadījumā tas ir "Personu atlase". Sākotnēji ir vaicājumu atbildes samazināšanas iespējas – rādīt visas atbildes, rādīt visas unikālās atbildes rindas plus vēl kaut kādas rindas uzstādot uz *REDUCED*, vai arī rādīt tikai unikālās atbildes rindas ar *DISTINCT* uzstādījumu.

Tālāk seko visu vaicājumā atlasīto mainīgo parādīšana kā atbildes kolonas un katrai kolonai tiek piešķirts arī unikāls vārds. Ir iespējams uzstādīt arī vaicājuma atbildes kārtošanas secību ar *ORDER_BY,* kur var uzstādīt, lai atbildes rindas tiktu kārtotas pēc vienas vai vairākām

kolonām augošā vai dilstošā secībā. Noslēgumā ir iespējams ar *LIMIT* uzstādīt cik daudz atbildes rindas ir nepieciešamas un ar *OFFSET* ir iespējams uzstādīt cik rindas no sākuma nav nepieciešamas.

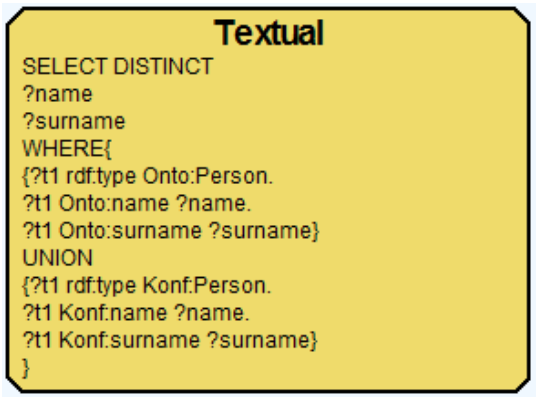


2.16. att. ***Vaicājuma atbildes formāts***

2.6.2. Vaicājuma atbilde kā tekstuāls vaicājums

Grafiskā notācija neļauj pierakstīt vaicājumus, kuros būtu lietots UNION vai MINUS, jo tās ir ļoti reti lietotas operācijas, kas tikai sarežģītu ViziQuer grafisko notāciju bez būtiskiem ieguvumiem lietotājiem.

Lai tomēr lietotājiem būtu iespējams pierakstīt arī šādus vaicājumus, tad tiek piedāvāta papildus iespēja, ka lietotājs pats sastāda tekstuālu vaicājumu. Grafiskā notācija ir tāda pati, kā vaicājuma atbildes formēšanai, tikai šajā gadījumā tas satur arī pilno SPARQL, ko ievada pats lietotājs. Piemēru varam aplūkot 2.17. attēlā, kur lietotājs ir ievadījis SPARQL vaicājumu, kas satur UNION operāciju. SPARQL sintakse vaicājumam ir identiska tam, kas ir attēlots kastē.



2.17. att. ***Tekstuāls vaicājums***

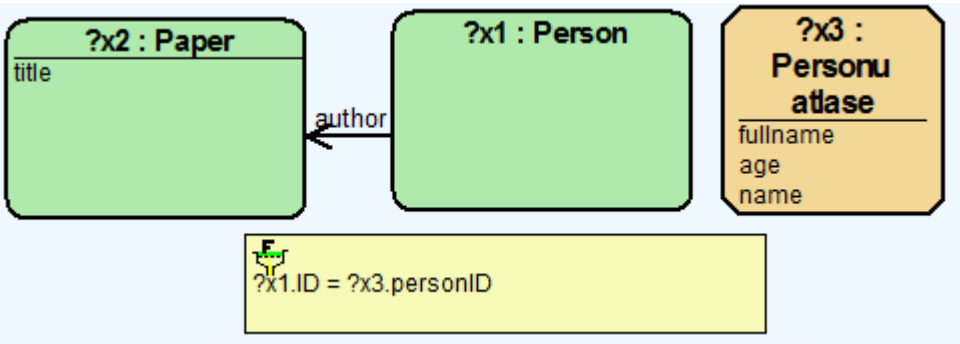
SPARQL sintakse:

```
SELECT DISTINCT ?name ?surname WHERE {
{?t1 RDF:type Onto:Person.
?t1 Onto:name ?name.
?t1 Onto:surname ?surname.}
UNION
{?t1 RDF:type Konf:Person.
?t1 Konf:name ?name.
?t1 Konf:surname ?surname.}}
```

2.6.3. Saglabāta vaicājuma izmantošana

Ja vēlamies iekļaut vaicājumā kādu citu, jau iepriekš sastādītu un saglabātu vaicājumu, tad to var izdarīt ar speciālu vaicājuma kasti, kam ir tādas pašas īpašības kā parastai objektu klasei, izņemot to, ka tai nav iespējams pievienot relāciju, jo vaicājumiem nav definētas relācijas.

Attēlā 2.18. varam aplūkot piemēru, kur izmantojam saglabāto vaicājumu. Pirmkārt, ir divas savstarpēji saistītas objektu klases Persona un Publikācija. Kā arī ir saglabātais vaicājums – Personu atlase, kurā ir atlasītas personas ar noteiktiem nosacījumiem. Jāatzīmē, ka vaicājumam netiek rādītas visas atbildes kolonas, bet tikai tās, ko izvēlas iekļaut jaunajā atbildē. (Vai nu var operēt tikai ar vaicājuma atbildes kolonām, vai arī var operēt arī ar mainīgajiem, kas ir vaicājumā, piemēram, ?x3.?persona.ID)



2.18. att. ***Saglabātā vaicājuma lietošana vaicājumā***

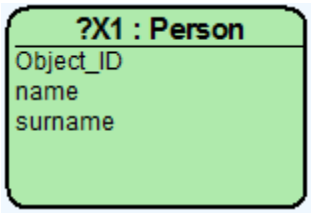
SPARQL sintakse:

```
SELECT ?title ?fullname ?age ?name WHERE {
?x1 RDF:type Onto:Person.
?x2 RDF:type Onto:Paper.
?x1 Onto:author ?x2.
?x2 Onto:title ?title.
      [Papildus tiek iekļauts vaicājums, kas atbilst
      apakšvaicājumam „Personu atlase”]
FILTER (?x1 = ?x3)
}
```

2.7. Vaicājumu piemēri

2.7.1. Objekta identifikatora iekļaušana atbildē

Reizēm objektu identifikatoros ir iekodēta derīga informācija. Lai vaicājuma atbildē iegūtu šos identifikatorus vajag atlasīt speciālo objektu klases atribūtu „*Object_ID”*, kas ir pieejams visām klasēm un atbilst objekta identifikatoram. Piemēru skatīt 2.19. attēlā.



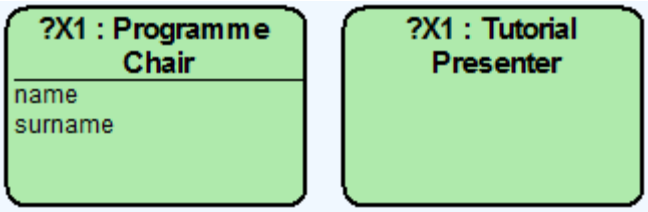
2.19. att. ***Objekta identifikatora atlase***

SPARQL sintakse:

```
SELECT ?X1 ?name ?surname WHERE {
?X1 RDF:type Onto:Person.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.}
```

2.7.2. Objektu klašu mainīgo vienādība

Ontoloģijas pieļauj iespēju, ka viens un tas pats objekts pieder vairāk nekā vienai klasei. Piemēram, varētu interesēt cilvēki, kas vienlaicīgi ir kādas programmas priekšsēdētājs, kas ir bijis arī kādas apmācības pasniedzējs. Šādā gadījumā mums ir jālieto viens un tas pats mainīgā vārds, kā tas ir parādīts 2.20. attēlā.



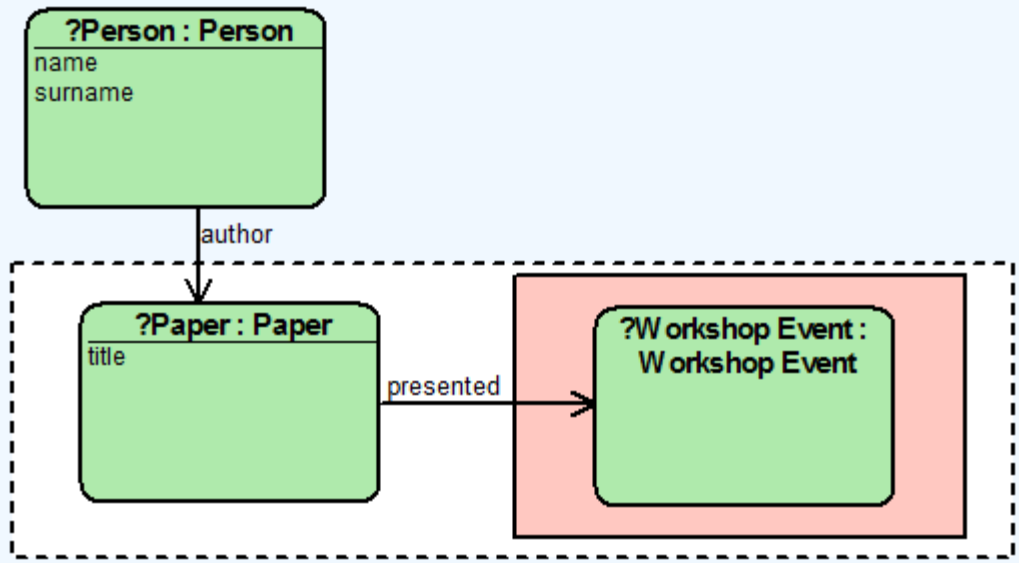
2.20. att. ***Objekta piederība vairākām klasēm***

SPARQL sintakse:

```
SELECT ?name ?surname WHERE {
?X1 RDF:type Onto:ProgrammeChair.
?X1 RDF:type Onto:TutorialPresenter.
?X1 Onto:name ?name.
?X1 Onto:surname ?surname.}
```

2.7.3. Grupu savstarpējā iekļaušana

Grupas ir īpaši elementi, kas sevī var iekļaut, kā citas grupas, tā arī parastas objektu klases. Šādā veidā ir iespējams veidot sarežģītus vaicājumus ar vairākiem noliegumiem. Piemēru skatīt 2.21. attēlā, kur meklējam personas un to sarakstīto publikāciju nosaukumus, ja tādas ir pieejamas, kas nav iesniegtas *workshopos*.



2.21. att. ***Grupu savstarpēja iekļaušana***

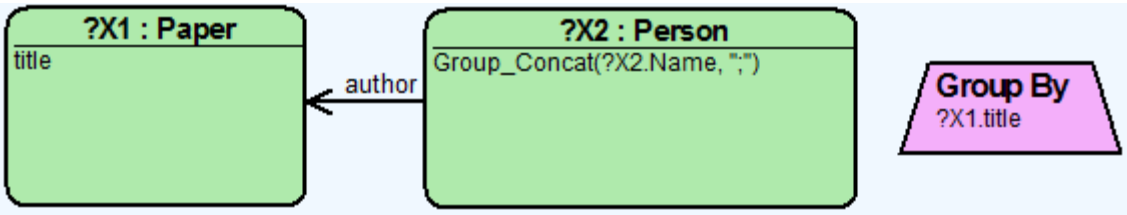
SPARQL sintakse:

```
SELECT ?name ?surname ?title WHERE {
?Person RDF:type Onto:Person.
?Person Onto:name ?name.
?Person Onto:surname ?surname.
OPTIONAL{
      ?Person Onto:author ?Paper.
      ?Paper RDF:type Onto:Paper.
      ?Paper Onto:title ?title.
      OPTIONAL{
            ?Paper Onto:presented ?WorkshopEvent.
            ?WorkshopEvent RDF:type Onto:WorkshopEvent.
            FILTER (!bound(?WorkshopEvent))
      }
}}
```

2.7.4. Vārdu apvienošana ar konkatenāciju un Group By

Gadījumā, ja vienam elementam atbilst vairāki citi elementi, kā tas ir gadījumā, kad publikāciju ir sarakstījuši vairāki autori, tad ir normāli, ka vēlamies vienā rindā atlasīt publikācijas nosaukumu un visus tās autorus.

Šajā gadījumā varam izmantot *Group Concat* iespēju, kad visi atbilstošie elementi tiek apvienoti vienā laukā. Tas ir, 2.22. attēlā dotajā piemērā mēs atlasīsim katrai publikācijai pa vienai rindai, kur būs divi lauki – pirmajā publikācijas nosaukums, bet otrā būs apvienoti visu autoru vārdi, kas savā starpā būs atdalīti ar „;”.



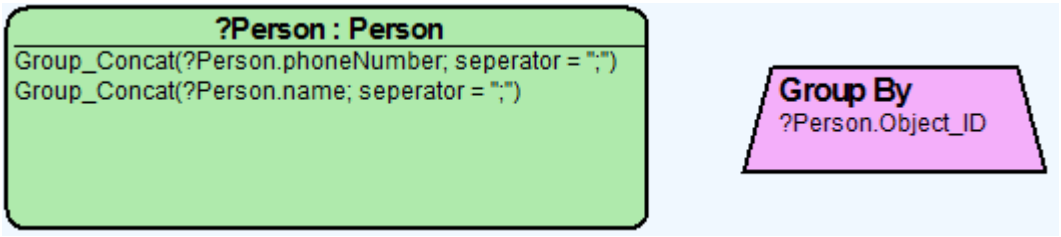
2.22. att. ***Vairāku vērtību apvienošana vienā***

SPARQL sintakse:

```
SELECT ?title (GROUP_CONCAT(?name; seperator = ";") AS
?autorList) WHERE {
?X1 RDF:type Onto:Paper.
?X2 RDF:type Onto:Person.
?X2 Onto:author ?X1.
?X1 Onto:title ?title.
?X2 Onto:name ?name.
} GROUP BY ?title
```

2.7.5. Grupēšana pēc objekta identifikatora

Ja visi atlasāmās vērtības ir agregāta funkcijas, tad ir nepieciešams tās grupēt pēc kādas vērtības, kas nav iekļauta atbildes tabulā. Šādos gadījumos loģiski tos būtu grupēt pēc objekta identifikatora, kā tas ir parādīts piemērā 2.23. Attēlā, kur atlasām personu vārdus un telefonu numurus, kur abos gadījumos ir iespējams, ka būs vairāk nekā viena vērtība katrai personai.



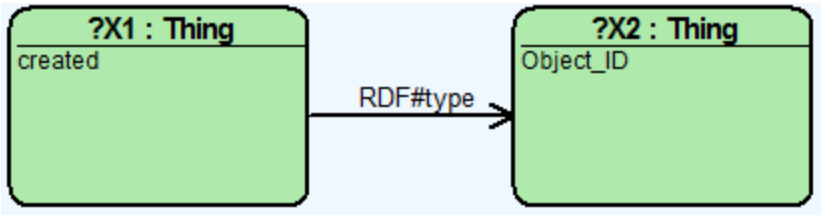
2.23. att. ***Grupēšana pēc objekta identifikatora***

SPARQL sintakse:

```
SELECT (GROUP_CONCAT(?phoneNumber; seperator = ";") AS
?phones) (GROUP_CONCAT(?name; seperator = ";") AS ?name)
WHERE {
?Person RDF:type Onto:Person.
?Person Onto:phoneNumber ?phoneNumber.
?Person Onto:name ?name.
} GROUP BY ?Person
```

2.7.6. Klases nosaukuma noskaidrošana

Ja gadījumā ir jāstrādā ar datiem, kur ir svarīgi uzzināt klases vārdu, kam pieder objekts, tad varam izmantot manuālu iespēju to noskaidrot tādā veidā, ka uzdodam īpašu *RDF type* asociāciju, kas savieno objektu ar tās klases tipu. Piemēru var apskatīt 2.24. attēlā.



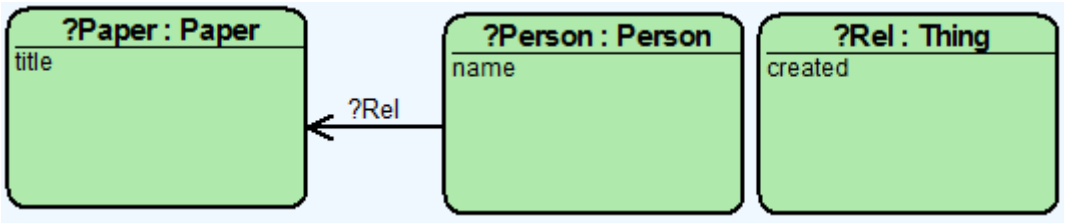
2.24. att. ***Klases vārda noteikšana***

SPARQL sintakse:

```
SELECT ?created ?X2  WHERE {
?X1 RDF:type ?X2.
?X1 Onto:created ?created.
}
```

2.7.7. Atsaukšanās uz relāciju, kā klasi

Ja ir ļoti īpatnēja ontoloģijas uzbūve, ka relācija arī ir saistīta tālāk, tad varam izmantot iespēju relāciju sākotnēji apzīmēt kā mainīgo, un tālāk šī mainīgā vārdu izmantot kā klases vārdu, lai varētu veidot tālāku relāciju tīklu. Šis ir ļoti eksotisks gadījums, kas praksē ir ļoti reti sastopams, jo parasti tiek lietota tikai viena relācijas instance, kas savieno visus objektus savā starpā, tāpēc šāds vaicājums ļautu iegūt papildus informāciju tikai meta līmenī. Piemēru skatīt 4.25. attēlā.



2.25. att. ***Relācija kā klase***

SPARQL sintakse:

```
SELECT ?title ?name ?created ?Rel  WHERE {
?Person RDF:type Onto:Person.
?Person ?Rel ?Paper.
?Paper RDF:type Onto:Paper.
?Person Onto:name ?name.
?Paper Onto:title ?title.
?Rel Onto:created ?created
}
```

2.8. SPARQL iespējas, kam nav piemeklēta grafiska notācija un iespējamie uzlabojumi

- UNION – ļauj rakstīt vaicājumā vairākas alternatīvās izteiksmes un to rezultātus apvienot vienā tabulā. Normāli tiek lietots, lai vienkopus atlasītu datus, kas apzīmēti divās dažādās vārdu telpās, bet loģiski ir tādi paši dati. Iespējams ierakstīt izmantojot SPARQL rakstīšanas iespēju pie atbildes formēšanas.

- MINUS – ļauj no atrastajām atbildēm atņemt atbildes, kas izpilda kādus citus nosacījumus un nav vēlams iekļaut atbildes tabulā. Iespējams ierakstīt izmantojot SPARQL rakstīšanas iespēju pie atbildes formēšanas.
- Neatbalsta ASK, DESCRIBE un CONSTRUCT vaicājumu formas
- FEDERATE QUERIES, kas teorētiski ļauj apvienot atbildes no vairākiem SPARQL piekļuves punktiem
- Vaicājumi ir veidoti, lai varētu iegūt datus no SPARQL piekļuves punktu, bet tos nav iespējams atjaunot vai labot ar UPDATE vai INSERT, ko piedāvā SPARQL piekļuves punkti.
- Ieviest speciālu notāciju, kad tiek veidota saistība ar iekļautu apakš vaicājumu

3. VIZIQUER LITE

Jebkuras valodas pamatā ir spēja to saprast un lietot, kā tekstuālas, tā arī grafiskas. Izveidojot iespējamu pilnu valodas pārklājumu, risinām matemātikas uzdevumu, lai atrastu veidu kā noklāt tekstuālu valodu ar grafiskiem elementiem. (Teorētiski to izdarīt būtu ļoti vienkārši, ja pārveidotu SPARQL valodas izteikumu regulārajā izteiksmē par kastēm.) Taču vēlamies izveidot optimālo attiecību starp valodas saprotamību un valodas izteiksmīgumu.

Viena no svarīgākajām sākotnējām tēzēm tika uzstādīta iespēja, ka šo valodu spēs lietot cilvēki bez formālas izglītības datorzinātnēs. Tātad varam pārformulēt šo tēzi sekojoši – ja grafiskā valoda ir grūti saprotama lietotājiem bez datorzinātņu izglītības vai ir nepieciešama apjomīga apmācība, tad tas apgrūtinās arī valodas praktisko lietojumu un valodas pilns pārklājums būs palicis kā matemātisks uzdevums, ne risinājums.

Ņemot vērā, ka slikti lasītāji vēlāk kļūst par sliktiem rakstītājiem [17], tad papildus izvirzīta hipotēze, ka izstrādājot uzlabotu valodas versiju, kuru lielākais vairums varēs ātri un precīzi izlasīt, tā vēlāk būs pietiekami parocīga, lai arī sastādītu korektus vaicājumus.

Lai pārbaudītu valodas lietojamību ir nepieciešami testi, kas apstiprina, cik ērti tā ir lasāma, kādas konstrukcijas lietotāji ātri saprot, bet kurās vietās parādās problēmas, kas ir jāizmaina. Izmantojot šādus iteratīvus testus ir iespējams uzlabot sākotnēji izstrādātās grafiskās vaicājumu valodas versiju uz versiju, kas nodrošina mazāku pārklājumu, taču tajā pašā laikā ir vieglāk saprotama lietotājiem.

Ņemot vērā, ka grafiskai valodai ir arī savs atbalsta rīks, kas ļauj sastādīt šīs valodas izteikumus, tad ir arī ļoti grūti veikt nodalīšanu starp grafiskās valodas sintaksi, kas tiek izdrukāta uz papīra, un grafiskās valodas sintaksi, kas ir pieejama elektroniskajā iekārtā, kur papildus ir iespēja interaktīvi darboties ar šo sintaksi.

3.1. Sākotnējo eksperimentu rezultāti medicīna domēnā

Pirmie valodas aprobācijas eksperimenti tika veikti medicīnas domēnā [18]. Vairāk par aprobāciju var izlasīt nodaļā 4.6. Aprobācijas rezultāti bija pozitīvi un lietotāji bija apmierināti ar iespējām ātri un vienkārši atlasīt datus tālākai apstrādei. Eksperimentālie rezultāti tika iegūti neformālu sarunu rezultātā ar medicīnas nozares pārstāvjiem, kas izmantoja ViziQuer vaicājumu rīku datu atlasei.

Tomēr jāatzīmē, ka eksperimentu rezultāti bija subjektīvi, jo kopējais lietotāju skaits bija ļoti neliels, apmēram desmit aktīvi lietotāji, kā arī šie lietotājiem bija ieguvuši priekšzināšanas SQL vaicājumu valodā (izmantojot SQL veica datu atlases citās datu kopās), tāpēc darbs ar līdzīgu datu atlases metodoloģiju nesagādāja lielas problēmas un apmācība nebija nepieciešama.

Turklāt, formālā valodas notācija lietotājus nesatrauca, un galvenie ierosinājumi tika saņemti attiecībā uz ViziQuer valodas atbalstošā rīka funkcionalitātes pilnveidošanu. Piemēram, pie ievadītās klases redzēt kādas klases tai ir pievienotas, un pa kādām asociācijām ir iespējams pievienot tās vaicājumam.

Lai arī viena no funkcionalitātes pilnveidošanām ietvēra arī aicinājumu pilnveidot grafiskās valodas notāciju, jo medicīnas darbinieki nodarbojās ar datu apkopošanu un tālāku statistikas analīzi, tad tika minēta vēlme atlasīt agregātfunkciju aprēķinus vaicājumu valodā. Piemēram, noskaidrot – cik daudz traumas katrs pacients ir guvis.[13]

Jāatzīmē, ka medicīnas domēna eksperimentālajā izstrādē tika izmantota ViziQuer vaicājuma valodas versija, kurā bija realizēta tikai klases kopas atlase, atribūtu atlases, nosacījumu uzstādījumi atribūtiem, kā arī klašu atlašu savstarpēja savienošana ar asociācijām. Kas norādīja uz hipotēzi, ka proporcionāli lielā skaitā gadījumu (vairāk kā 80%) lietotājiem pietiks tikai ar vienkāršu konstrukciju lietojumu.

3.2. Iteratīvo eksperimentu rezultāti

Kā eksperimentu galveno tēzi izvirzām ideju – labi lasītāji ir arī labi rakstītāji [17]. Ja tas empīriski ir pierādīts ikdienā lietotai tekstuālai valodai, kas ir abstrakta, tad izdarām pieņēmumu, ka tam ir jāstrādā arī grafiskām valodām.

Tātad mūsu eksperimentu galvenais uzstādījums ir panākt, ka izstrādāto grafisko vaicājumu valodu cilvēki var ātri un vienkārši izlasīt. Ja esam pierakstījuši vaicājumu grafiski, tad cilvēks var to izlasīt un pierakstīt tekstuālā valodā. Tad interpretējam cilvēka pierakstīto izteikumu un mēģinām salīdzināt cik ļoti tas sakrīt ar sākotnējo izteikumu.

Eksperimentāli ir vieglāk pārbaudīt – vai lietotāji ir apguvuši grafisko valodu un var ar to pierakstīts uzdotos jautājumus, jo ir iespējams formāli pārbaudīt, vai uzrakstītais vaicājums atlasa

[13] Eksperimentālā izstrāde tika veikta 2008. gadā un tai laikā bija apstiprināts SPARQL standarts, kas sevī neietvēra agregātfunkcijas. SPARQL 1.1 standarta izstrādes pirmā versija, kas sevī ietver agregātfunkcijas, tika apstiprināts 2009. gada 22. oktobrī, lai arī paša rekomendācija ir apstiprināta 2013. gada 21. martā. Taču tajā pašā laikā eksperimentālajā izstrādē izmantojām Virtuoso RDF datu bāzi, kas bija izstrādājusi SPARQL papildinājumu datu atlasei un ļāva veikt agregātfunkciju atlasi. Ievedot šādu vaicājuma valodas paplašinājumu tajā laikā nozīmētu pašas valodas izmantošanas iespēju uz specifiskiem SPARQL piekļuves punktiem.

precīzi tos pašus rezultātus, kas tikai atlasīta bāzes piemērā, no kura tika veidots jautājums. Pārbaudīt, vai cilvēki ir pareizi sapratuši uzrakstīto apgalvojumu, nav tik vienkārši.

Mums ir jāsalīdzina divi dabiskās valodas izteikumi un jāsaprot, vai tie apgalvo vienu un to pašu vai nē. Ņemot vērā, ka dabiskajā valodā par vienu un to pašu lietu var izteikties ļoti dažādi, tad šajā procesā ir iespējams pieļaut neprecizitātes un to ir grūtāk formāli novērtēt. Katra izteikuma pareizību testa sastādītājs var formāli novērtēt tikai pēc viņa individuālā viedokļa. Pielikumā 2. tiks uzdotas visas saņemtās atbildes, lai būtu iespējams novērtēt, cik korekti ir novērtēti iegūtie rezultāti.

3.2.1. Valodas sākotnēs bāzlīnijas novērtējums

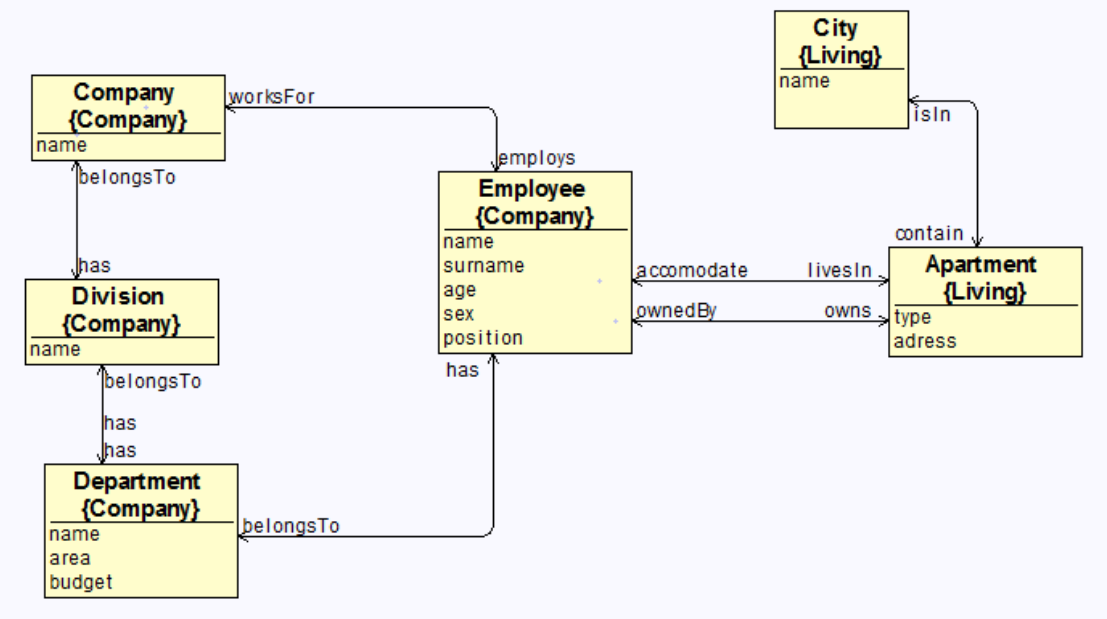
Lai novērtētu izstrādāto ViziQuer vaicājumu valodas pilnās versijas lasāmību tika izstrādāta aptauja, kurā tika iekļauti 17 sagatavoti vaicājumi grafiskā formā, kuri lietotājiem bija jāpārformulē dabiskajā valodā.

Sagatavotie vaicājumi pārklāja visus vaicājuma valodas pamatelementus – klašu kopas atlasi, atribūtu atlasi klašu kopā, nosacījumu uzstādīšana, vairāku klašu kopu savienošana, negāciju nosacījumu uzstādīšana, neobligātās grupas. Atbilstoši ir iespējams novērtēt, kurus no grafiskajiem elementiem lietotāji saprot labāk, kurus ir problemātiskāk saprast un kādas ir kļūdas, kuras lietotāji pieļauj.

Atbilstoši identificējot biežāk pieļautās kļūdas un grafiskos elementus, kas izraisa šo nepietiekamo izpratni, ir iespējams veikt labojumus grafiskās valodas specifikācijā un veikt atkārtotu pārbaudi, lai pārliecinātos, ka nepilnības ir novērstas.

Pielikumā 2 ir apskatāma aptaujas anketa. Aptauja tika veikta tiešsaistē izmantojot pūļresursēšanas (*crowdsourcing)* pieeju un to aizpildīja 14 respondenti, izvērtējot atbilžu atbilstību būtībai, 5 respondentu atbildes neatbilda uzdotajiem jautājumiem, tāpēc tālāk tiek uzskatīts, ka atbildēja tikai 9 respondenti. Katrs respondents par anketas aizpildīšanu saņēma minimālu atlīdzību (50 centus), kas kalpoja kā motivācija censties un atbildēt iespējami labi. Jāatzīmē, ka pirmajā aptaujā netika ievākta informācija par lietotāju prasmēm darbā ar datoru un datubāzu zināšanām, kas būtiski ietekmē cik labi lietotāji orientējas šādos vaicājumos. Taču tajā pašā laikā bija iespējams iegūt priekšstatu par galvenajiem trūkumiem izstrādātajā valodā.

Pielikumā 2 ir apskatāmas arī respondentu atbildes uz uzstādītajiem jautājumiem, kā arī iekavās ir doti komentāri par šīm atbildēm – vai tās atbilst jautājumam, kā arī vai šajā atbildē ir kādi trūkumi attiecībā pret sagaidāmo atbildi.



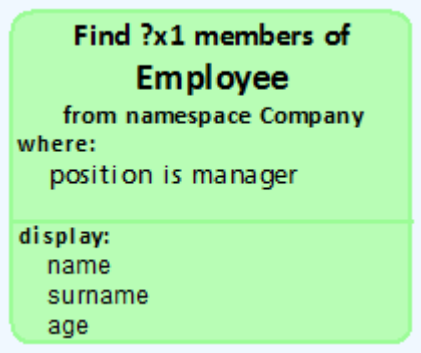
3.1. att. ***Respondentu aptaujā izmantotā datu shēma.***

Lai darbs būtu pārskatāms, nodemonstrēsim arī šeit dažus no aptaujas anketas piemēriem, kā arī atbilstoši atbildes, kas norāda uz trūkumiem izstrādātajā vaicājuma valodā. Attēlā 3.1. ir apskatāma datu shēma, uz kuras pamata tika veidoti vaicājumi. Tā attēlo ļoti vienkāršotu kompāniju struktūru, kur ir kompānijas, kurās ir nodaļas (*division*) un apakšnodaļas, departamenti (*department*). Darbinieki strādā kompānijā, kā arī ir piesaistīti noteiktai apakšnodaļai. Darbiniekiem var piederēt kāds īpašums, kā arī viņi dzīvo kādā īpašumā. Īpašums atrodas kādā pilsētā. Respondenti, kuri izskaidroja atbildes uz vaicājumiem netika iepazīstināti ar šo shēmu.

Ņemot vērā, ka šī bija pirmā iterācija vaicājumu valodas saprotamības analīzei, tad parādījās pietiekami daudz un dažādas problēmas, no kurām dažas šķita kā ļoti saprotamas lietas IT cilvēkiem, taču atklājās, ka tās nav tik saprotamas cilvēkiem, kam nenākas nodarboties ar šādiem jautājumiem ikdienā.

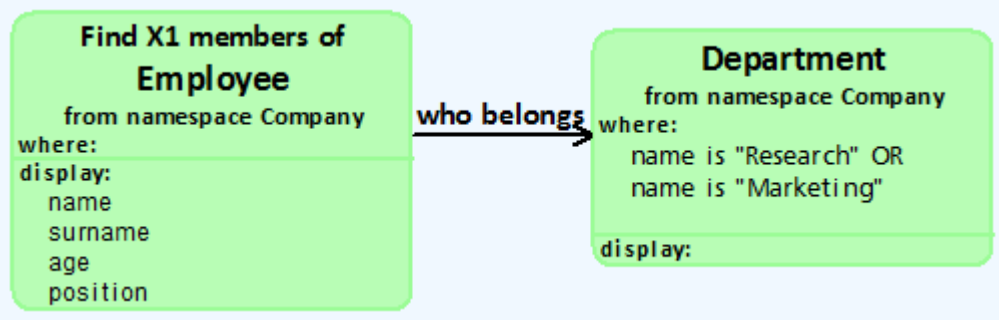
Svarīgi, ka testa rezultāti norāda uz pozitīvu iezīmi, jo vienkāršākās vaicājuma valodas konstrukcijas parādīja, ka lietotāji saprata un izskaidroja uzzīmēto vaicājumu. Piemēram, visi aptaujas dalībnieki izprata, vienkāršāko vaicājumu, kas ir attēlots 3.2. attēlā. Lai arī 2 no 9 aptaujātajiem neuzrādīja datus, kas tiks atgriezti izpildot šādu vaicājumu, taču pēc būtības saprata vaicājuma apgabalu.

Tātad varam izdarīt secinājumus, ka daļēji vaicājuma valoda sasniedz savu mērķi, jo lietotāji saprot tās pamatkonstrukciju un tālākās iterācijās ir iespējams uzlabot vaicājumu valodu, lai tā kļūtu saprotamāka lietotājiem.



3.2. att. ***Vienas klases datu atlase vaicājumā***

Arī otra fundamentālā valodas konstrukcijas (asociācija), kas saista vairākas atlasītās objektu kopas, lietotājiem ir saprotama, ko parāda lietotāju atbildes uz 3.3. attēlā esošo grafisko vaicājumu. 8 no 9 respondentiem pareizi norāda, ka darbiniekiem ir jāstrādā zinātnes vai mārketinga nodaļās. Tas parāda, ka arī otrs būtiskais grafiskās vaicājumu valodas relācijas elements lietotājiem ir saprotams.



3.3. att. ***Vaicājums, kas atrod darbinieku datus, kas strādā zinātnes vai mārketinga nodaļā***

Taču tajā pašā laikā parādījās dažādas problēmas vaicājumu valodas konstrukcijās, kas lietotājiem nebija tik viegli saprotamas. Piemēram, vārdu telpas (*namespace*) jēdziens, kas tiek plaši lietots Semantiskā tīmekļa datos, lietotājiem radīja grūtības. Pirmkārt, ikdienā šāds jēdziens nav sastopams un parādās tikai specifiskā apgabalā, tāpēc cilvēku loks, kam tas kaut ko izsaka ir ļoti ierobežots. Ja vēlamies, lai grafisko valodu izprastu cilvēki ne no IT nozares, tad šāda jēdziena izmantošana grafikā nav atbalstāma, jo rada apmulsumu.

Vārda telpas (*namespace*) jēdziena apskati nodrošinās grafiskās vaicājuma valodas atbalsta rīks. Tieši atbalsta rīks nodrošina, lai uzrakstītās vaicājumu valodas sintakse būtu korekta.

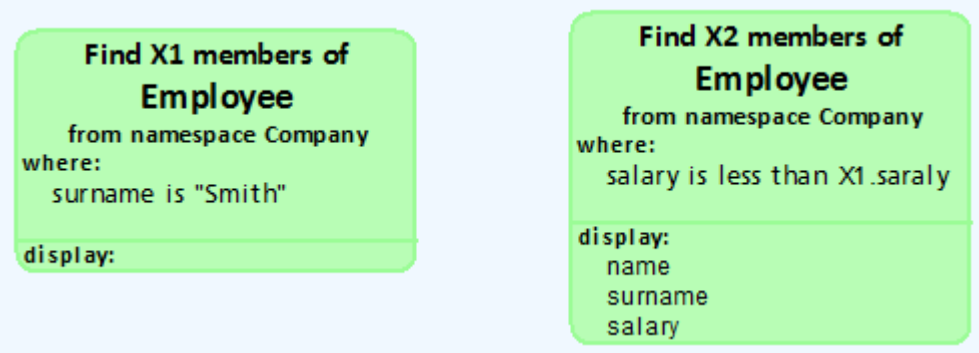
Mēs esam nonākuši pie pirmās izmaiņas valodas konstrukcijā, jo aptaujas dalībnieki netieši norāda uz faktu, ka neizprotamas lietas tiek ignorētas un tās nav jārāda. Tāpēc no valodas grafiskās notācijas izņemsim vārda telpas (*namespace*) attēlojumu.

Kā otru problēmu, kas apgrūtina aptaujas dalībnieku izpratni, var minēt mainīgos, kas tiek lietoti, lai apzīmētu atlasītās objektu kopas. Motivācija ieviest mainīgos bija, lai varētu vaidot vaicājumus, kur nosacījumos atsaucas uz šo mainīgo vērtībām. Taču aptaujas dalībniekiem radās problēmas izskaidrot vaicājumu, kas saturēja nosacījumu, kurā notiek atsaukšanās uz citas objektu kopas atlases mainīgo.

Aptaujas anketas piemērs ir attēlots 3.4. attēlā, kur tiek vaicāts atrast visus darbiniekus, kas saņem mazāku algu nekā darbinieks Smits.[14] Aptaujas rezultāti uzrāda, ka 4 no 9 dalībniekiem īsti neizprot šo atsaukšanās mehānismu. Būtiskais secinājums ir tāds, ka mainīgo vērtības sarežģīt grafisko vaicājumu.

Mainīgajiem parasti tiek piešķirtas intuitīvas vērtības, kas sevī ietver arī to izcelsmi. Piemēram, ja vaicājumā tiek atlasīta tikai viena darbinieku klase, tad to tā arī saucam par darbinieku mainīgo. Ja tiek atlasītas divas klases, tad, piemēram, viena var būt padotie, kamēr otra klase var būt menedžeri.

Ņemot vērā aspektu, ka klases vērtība, no kuras tiek atlasīti objekti, tiek normāli iekļauta mainīgā nosaukumā (varam atzīmēt, ka šī ir viens no metodoloģiskajiem norādījumiem labu un lasāmu grafisku vaicājumu sastādīšanā), tad nav nozīme grafikā dublēt informāciju un tajā tiek norādīts tikai mainīgā vārds. Klases identifikators parādās tikai atbalstošajā rīkā, un bez šī klases identifikatora ir iespējams izlasīt sastādīto grafisko vaicājumu būtību, ja ir sastādīti izmantojot saturīgus mainīgo nosaukumus.



3.4. att. ***Anketas jautājums, kur tiek lietota mainīgo atsaukšanās***

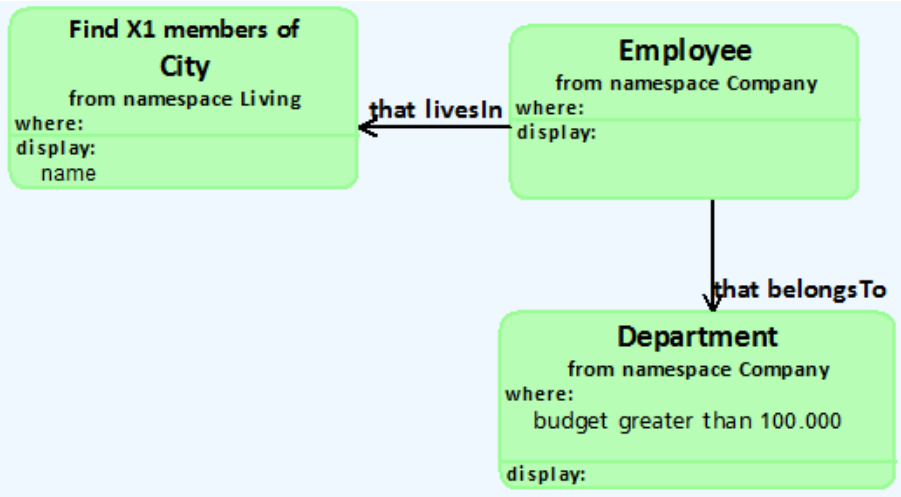
[14] Formāli varam diskutēt, ka varētu būt vairāki darbinieki ar uzvārdu Smits, kas nozīmētu, ka darbinieks parādās atbildē tik daudz reizes, cik ir Smiti, kuri pelna vairāk nekā konkrētais darbinieks. Taču šajā gadījumā ir būtiski pārbaudīt, vai aptaujas dalībnieki izprot atsaukšanās principu. Ja šis princips netiek izprast, tad nav būtiski cik daudzi Smiti ir ierakstīti atbildē un kādos gadījumos atbildes lauki dublētos. Fakts, ka mums šī situācija ir papildus jāizskaidro un neviens no aptaujas dalībniekiem to nav pamanījies jau norāda, ka šāda mainīgo lietošana atsaucoties uz citiem elementiem var būt ne tik intuitīvi saprotama.

Piedāvātā mainīgo vārdu izņemšana sasaucas ar pirmo problēmu, jo skatoties korekti uz Semantiskā tīmekļa datu uzdošanas principiem, būtisku lomu spēlē arī vārda telpa (*namespace*). Var būt situācija, ka vienā SPARQL piekļuves punktā ir dati, kur ir vairākas klases ar vienādu nosaukumu, taču ar dažādām vārdu telpām.

Piemēram, vienā SPARQL piekļuves punktā ir apvienoti divu uzņēmumu dati, kur katrs uzņēmums lieto atšķirīgas vārdu telpas, taču abos gadījumos parādās klase darbinieki. Šādā gadījumā norādot klasi, bet, nenorādot vārdu telpu, var rasties tikai papildus pārpratumi. Kamēr sastādot vaicājumu labās prakses stilā un izmantojot saturīgus mainīgo nosaukumus, mums būs divas objektu atlases klases, kur viena tiks apzīmēta kā A uzņēmuma darbinieki, bet otra kā B uzņēmuma darbinieki.

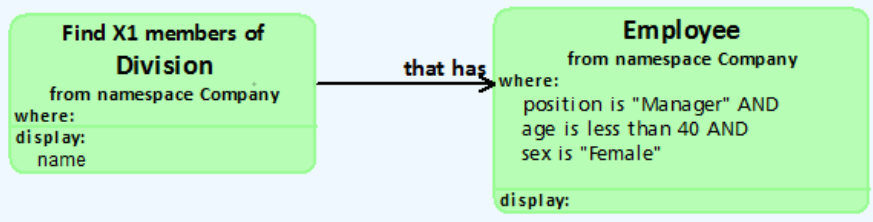
Šādi esam vienkāršojuši un padarījuši saprotamāku mūsu grafisko vaicājumu valodu, jo sākotnējā notācijā izmantojām trīs grafiskos elementus (mainīgā vārdu, klases vārdu, vārdu telpu), lai apzīmētu objektu kopu, kas tiek atlasīta. Tagad mēs to aizstājam ar vienu elementu – mainīgā vārdu, kā arī paļaujamies, ka lietotāji šos vārdus rakstīs saprotami. (Nesaprotamus tos varēja rakstīt arī iepriekš, kas tāpat būtu apgrūtinājis vaicājumu lasāmību.)

Nākamā lieta, kas apgrūtina vaicājumu lasāmību, ir tā, ka visas objektu atlases klases tiek attēlotas vienādas, tāpēc lietotājiem ir pašiem jāizvēlas kopa (klase), no kuras sākt vaicājuma lasīšanu. Ir grūti noteikt precīzus principus, kā tas tika darīts aptaujas anketās, taču trīs galvenie principi bija – izvēlēties klasi, kas ir pazīstamākā un kas liekas svarīgākā, piemēram, varam aplūkot attēlu 3.5., kur ir attēlots vaicājums, ka vēlamies atrast tās pilsētas, kurās dzīvo darbinieki, kas strādā nodaļās, kuru budžets ir lielāks par 100.000.



3.5. att. ***Aptaujas jautājums, kas norāda problēmas lasīšanas secībai***

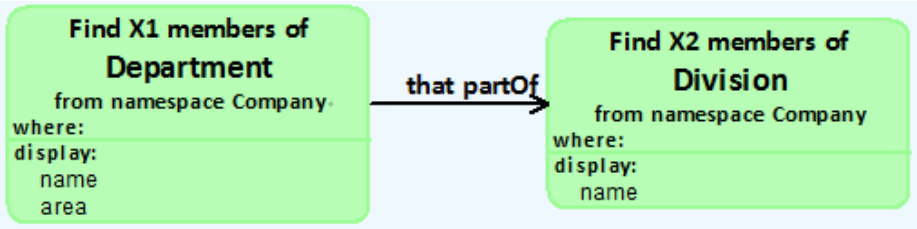
Piemērā, kas attēlots 3.5. attēlā, 8 no 9 respondentiem vēlējies sākt vaicājuma atlasi no darbinieka, lai arī darbinieka klasē vispār nav jāatlasa neviens atribūts. Vairāki citi vaicājumi apstiprina šādu hipotēzi. Piemēram, uz vaicājums, kas ir attēlots 3.6. attēlā, arī 7 no 9 respondentiem vēlējās noskaidrot informāciju par personu, lai arī vaicājuma būtība bija atrast nodaļas, kurās menedžere ir sieviete jaunāka par 40 gadiem.



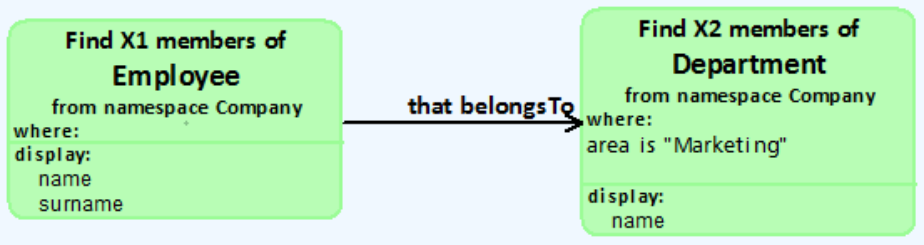
3.6. att. ***Vaicājums, kurā jāatrod nodaļas, kuru menedžeri ir sievietes jaunākas par 40 gadiem.***

Otra būtiskā lieta, kas ir saistīta ar to, ka nav skaidrs, no kuras klases sākt lasīt un kā atlasīt datus ir tas, ka aptaujas dalībnieki ļoti vāji demonstrēja izpratni par datu apvienošanu, ja tie ir atlasīti vairākās klašu kopās un tad savienoti ar relāciju. (IT cilvēki to sauktu par *join* operāciju.)

Piemēram, 3.7. attēlā ir vaicājums, kur jāatlasa visas apakšnodaļas un virziens, kurā tā darbojas, kā arī nodaļa, kuras sastāvā tā ietilpst. Tikai 1 no 9 cilvēkiem saprata un pareizi uzrakstīja, kas tiek prasīts šajā vaicājumā. Arī citos vaicājumos, kur parādījās datu apvienošana, lietotāji (8 no 9 aptaujas dalībniekiem neatzīmēja, ka jāatlasa arī apakšnodaļas nosaukums) atlasīja tikai datus no vienas klases, kā tas ir 3.8. attēla piemērā, kur jāatlasa darbinieki no apakšnodaļām, kas nodarbojas ar mārketingu un ir jāuzrāda apakšnodaļas nosaukums.



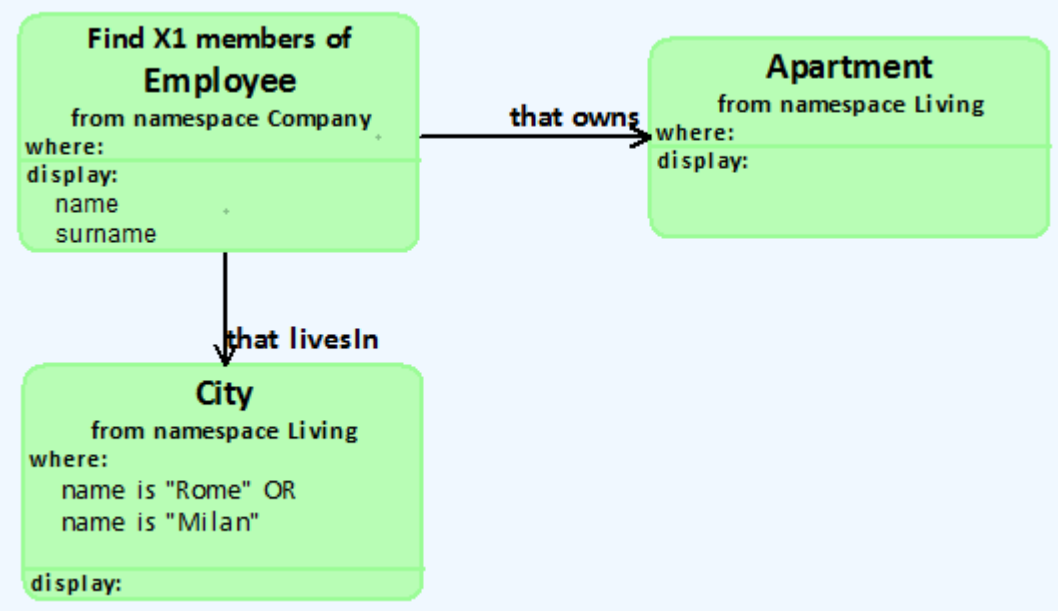
3.7. att. ***Atlasīt apakšnodaļas un nodaļas, kurām tās pieder.***



3.8. att. ***Atlasīt darbiniekus no mārketinga apakšnodaļām.***

Ja aplūkojam 3.11. attēlu, kur arī parādās *join* operācija apvienojumā ar neobligātu jēdzienu lietošanu, tad redzam, ka šajā gadījumā darbiniekiem adresi pievienoja 5 no 9 aptaujas dalībniekiem. To var skaidrot arī ar situāciju, ka iepriekšējos gadījumos varēja rasties problēmas ar to, ka divām klasēm bija vienādi atribūtu nosaukumi.[15]

Apkopojot šos rezultātus varam izvirzīt hipotēzi, ka galvenā klase, no kuras sākas vaicājuma veidošana/lasīšana, atvieglos vaicājuma izpratni. Kā arī ir jāizvēlas pareizs nosaukums, ko pielikt klāt atribūtiem no citām klasēm, ko pievienot atbildei, piemēram, "Pievienot rezultātiem:". Tas palīdzētu novērst problēmas ar vaicājuma secības lasāmību, kā arī saprast kādi rezultāti vēl ir jāpievieno. Kā arī galveno klasi veidot citādākā krāsā, formā, lai lietotājs izprastu, ka tā ir atšķirīga.



3.9. att. ***Darbiniekam pieder dzīvoklis un dzīvo Milānā vai Romā.***

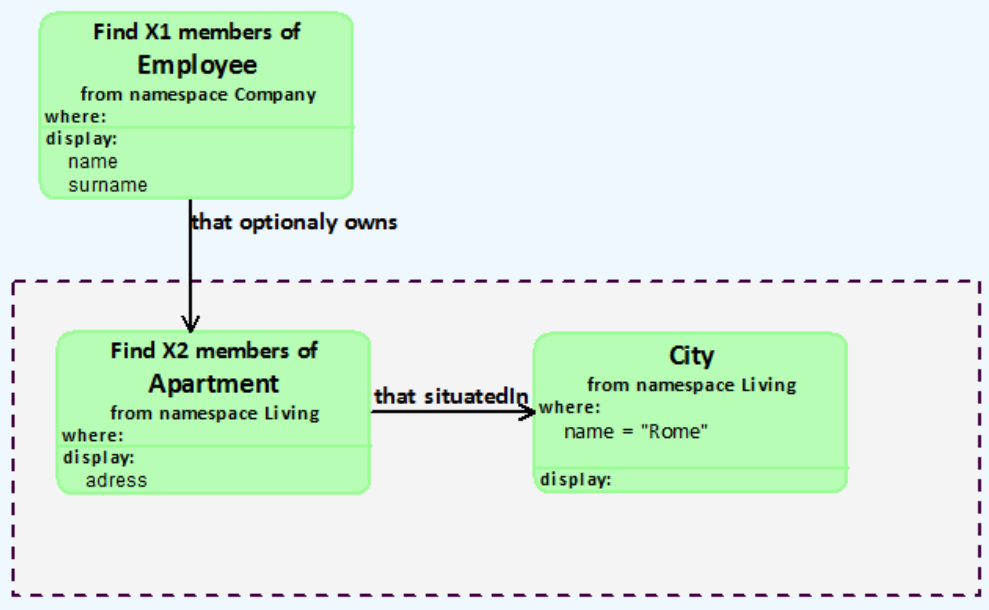
Kā mazāku trūkumu var minēt to, ka lietotāji apvieno arī līdzīgus jēdzienus kopā. Piemēram, 3.9. attēlā ir vaicājums, kur tiek jautāti darbinieki, kam pieder dzīvoklis, kā arī dzīvo Romā vai Milānā. 6 no 9 aptaujas dalībniekiem intuitīvi apvieno, ka darbiniekam pieder dzīvoklis Romā vai Milānā. Ņemot vērā, ka šāds vaicājums ir ļoti netipisks ikdienas situācijām, jo parasti dzīvoklī, kurš pieder, arī tiek dzīvots. Līdz ar to ir grūti ieteikt kādus konkrētus risinājums, kā var risināt

[15] Formāli OWL nepieļauj šādu situāciju, ka viens atribūts var būt definēts vairākām klasēm. Tad tiek ieviesta virsklase, kas ir šo klašu apvienojums. Taču relāciju datubāžu gadījumos tā ir daudz ierastāka prakse, jo atribūta vērtībai ir jābūt unikālai konkrētā klasē, nevis globāli unikālai.

šādu kognitīvo problēmu. Šādiem gadījumiem jau ir nepieciešama lietotāju papildu apmācība[16], lai varētu atšķirt gadījumu, kad īpašnieks dzīvo citā pilsētā nekā tā, kurā viņam pieder dzīvoklis.

Līdzīga problēma parādās arī ar neobligāto grupu jēdzienu. Piemēram, 3.10. attēlā ir attēlots vaicājums, kurā tiek atlasīti darbinieki, kā arī dzīvokļi, kas tiem pieder, ja tie atrodas Romas pilsētā. Gadījumos, kad darbiniekam nepieder šādi dzīvokļi, tad rādīt tikai darbinieka datus.

Ja mēģinām šo vaicājumu izskaidrot dabiskajā valodā, tad redzam, ka ir grūti to izdarīt vienā teikumā, kā arī tiek lietots termins "ja, tad". Ikdienas izteikumos parasti kaut ko apzīmējot šie dati arī ir, taču datubāzēs gadās situācijas, ka atribūtu ir iespējams atlasīt, taču tam nav piesaistīta vērtība, tāpēc tas uzreiz samazinās atbilžu kopu. Izmantojot neobligātos jēdzienus varam saglabāt kopas lielumu un bieži tas palīdz iegūt labāku priekšstatu par datu bāzē esošajiem datiem.



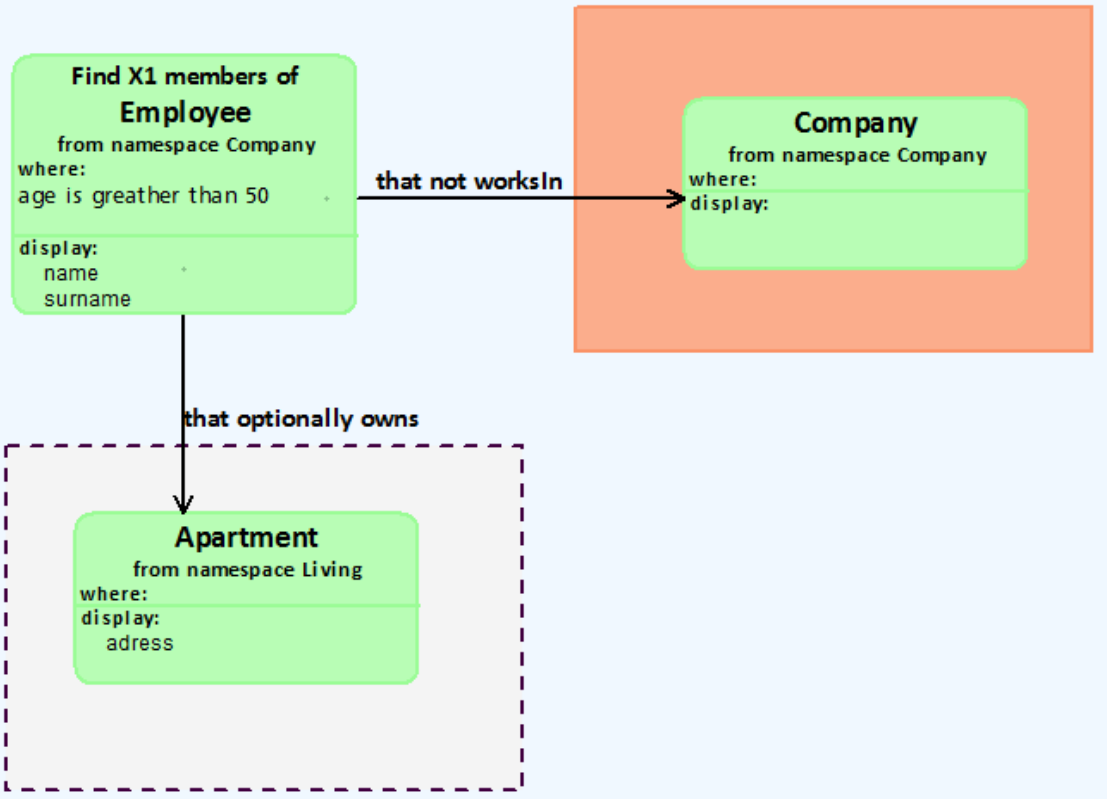
3.10. att. ***Darbinieki un viņiem piederošie dzīvokļi Romā.***

Taču līdzīgi kā ar kognitīvi tuviem jēdzieniem, kas tiek apvienoti, tāpat arī ar neobligātiem jēdzieniem ir grūti rast risinājumu, kas tos labi atspoguļotu. Arī šādos gadījumos ir vēlama papildus apmācība un lietotāju informēšana, kā tos atšķiras no standarta gadījuma.

Taču tajā pašā laikā lietotāji labi izprot negācijas jēdzienu. Varam aplūkot 3.11. attēlu, kur tiek attēlots vaicājums, kur ir jāatrod cilvēki vecāki par 50 gadiem, kas ir bezdarbnieki. Ja viņiem

[16] Atzīmēsim, ka lietotāji, kas aizpildīja anketu netika speciāli apmācīti un sagatavoti vaicājumu pārformulēšanai dabiskā valodā. Mēs apzināmies, ka lietotājiem atbildēt bez apmācības uz šādiem jautājumiem ir grūti, taču tajā pašā laikā tas daudz labāk parāda grafiskās valodas nepilnības, kas nav pietiekami intuitīvas un rada problēmas, lai tos izprastu.

pieder dzīvoklis, tad parādīt arī šī dzīvokļa adresi. 7 no 9 aptaujas dalībniekiem atzīmēja, negāciju. Kā arī viens aptaujas dalībnieks vaicājumu bija pārformulējis nosacījuma formā, kur figurēja negācija.



3.11. att. ***Darbinieki vecāki par 50 gadiem, kas ir bezdarbnieki. Atrast vārdu un adresi, ja pieder dzīvoklis.***

Apkopojām vaicājumu valodā saprotamās un nesaprotamās lietas, izvirzījuši hipotēzes, ko ir nepieciešams uzlabot, lai padarītu vaicājumu valodu lietotājiem labāk lasāmu.

Lielākā daļa aptaujas dalībnieki intuitīvi saprata, par ko tiek sastādīti vaicājumi, kas ir pozitīvi, taču tajā pašā laikā ir vairākas nepilnības. Pirmkārt, nav skaidrs no kurienes sākt lasīt vaicājumu, ir lieka informācija, kas apgrūtina vaicājuma lasīšanu, kā arī ir detaļas, kas ir neskaidras un ko var uzlabot ar papildus, nelielu apmācību.

3.3. ViziQuer vaicājuma valodas uzlabotā versija

Ņemot vērā galvenās problēmas, kas radās aptaujas dalībniekiem izskaidrojot pilnās versijas ViziQuer vaicājumus, tika izveidota uzlabota vaicājuma valodas versija ViziQuer lite.

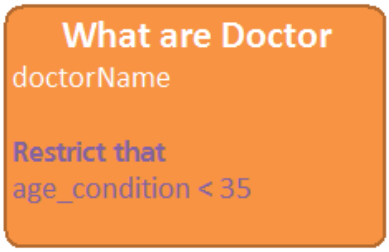
ViziQuer lite primārais mērķis bija padarīt vienkāršās konstrukcijas saprotamas lietotājiem, lai tās būtu viegli lasāmas un sasniegtu vidēji vairāk par 80% pareizu atbilžu izlasot

un izskaidrojot uzrakstītos grafiskos vaicājumus. Kā arī sniegtu iespējas pieredzējušiem lietotājiem uzrakstīt pilnīgākus un vispārīgākus SPARQL vaicājumus.

3.3.1. ViziQuer lite vaicājumu valodas elementi

Kā galvenā atšķirība starp pilno un lite versiju, ir jāmin objektu klases atlases elementu iedalīšana divu atšķirīgu notāciju elementos. Kā arī liekās un tehniskās informācijas maskēšana no grafiskās notācijas. Tehniskās informācijas apskati nodrošina vaicājumu valodas atbalstošais rīks, kas nodrošina vaicājumu sastādīšanu.

Lite versijas katrā vaicājumā tiek ieviesta primārā klase, kam ir jābūt obligāti katrā vaicājumā un tā vaicājumā ir tieši viena. Šādas klases uzdevums ir atvieglot lietotājam vaicājuma lasīšanu un arī sastādīšanu, jo tiek uzstādīts vaicājuma fokuss uz vienu konkrētu objektu kopu, kurai tiek uzstādīti papildus nosacījumi, kā arī vajadzības gadījumā tiek atlasītas un piekārtotas papildus vērtības no citām objektu kopā. Attēlā 3.12. var aplūkot kā izskatās centrālā objektu klases atlases simbola notācija.



3.12. att. ***Centrālā objektu atlases kopa, kas atlasa doktoru vārdus, kas jaunāki par 35 gadiem***

SPARQL sintakse:

```
?Doctor a Onto:Doctor.
?Doctor Onto:name ?doctorName.
?Doctor Onto:age ?age_condition.
FILTER (?age_condition < XSD:integer("35"))
```

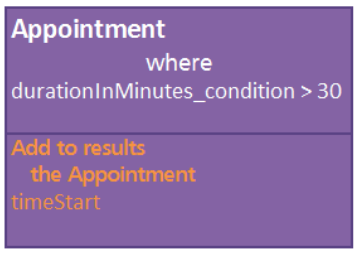
Viena no fundamentālajām atšķirībām starp lite un full versiju ir tā, ka grafiskajā notācijā tiek ieviesti papildus vārdi, kas palīdz vaicājumu lasīt. Šajā gadījumā papildinām vaicājumu ar "*What are Doctor"*, kā mērķis ir vedināt lietotāju domāt par to, ka vēlamies atlasīt doktora vērtības, kas ir minētas zemāk. Līdzīgi ir papildināta arī nosacījumu daļa ar "*Restrict that*", kas liek domāt, ka šeit tiek minēti papildus nosacījumi datu atlasei.

Ja pievēršam uzmanību SPARQL sintaksei un grafiskai sintaksei, tad var redzēt, ka grafiskā sintakse vairs neizmanto vārda telpas (*namespace*) definīciju, kā arī grafiskajā sintaksē parādās tikai mainīgā vārds, ar kuru apzīmē atlasīto objektu kopu, bet neparādās objektu kopas nosaukums datu bāzē. Tas uzliek lielāku atbildību grafiskā vaicājuma rakstītājam, jo ir jāizvēlas

semantiski atbilstošs vaicājuma vārds, taču ļoti bieži programmēšanas metodika iesaka mainīgos dēvēt atbilstošos nosaukumos. Līdzīgi tas ir arī ar atlasītajiem atribūtiem, ka tiek rādīts tikai semantiskais atribūta nosaukums, bet tiek noklusēts, kā šis atribūts tiek izveidots.

Atribūtu gadījumā gan ir jāmin arī tas, ka tie var būt ne tikai parasti atribūti, bet arī funkciju rezultāti, kur kā parametru padod kādu no atribūtiem, piemēram, vidējā vērtība doktoru vecumam. Līdz ar to, tiek izmainīta sintakse, kā pierakstīt agregāta funkcijas. Papildus arī gadījumos, kad tiek izmantota kāda no agregātfunkcijām atribūta rēķināšanai, tiek automātiski pielikts noklusētā *Group By* vērtība pēc visiem pārējiem atribūtiem, kas parādās atbildes tabulā.

Tāpat ir otra objektu kopu atlases notācija, kas tiek izmantota visām pārējām vaicājumā lietotajām kopu atlasēm. Šī notācija ir samērā līdzīga centrālai objektu kopas atlases notācijai. Tiek nedaudz pamainīta forma un krāsa, kā arī tiek lietoti citi papildus vārdi, lai būtu vieglāk lasīt, ka šī objektu kopa ir piesaistīta centrālai objektu kopai. Tāpat ir mainīta arī sadaļu secība, jo vispirms nāk nosacījumu daļa, kas tiek uzlikta konkrētai klasei (šis tiek darīts ar domu, ka primāri papildus atribūtu klases tiek liktas kā nosacījumi), tad seko sadaļa ar atribūtiem, kas tiek pievienota atbildei. Grafiskās notācijas piemērs ir apskatām 3.13. attēlā.

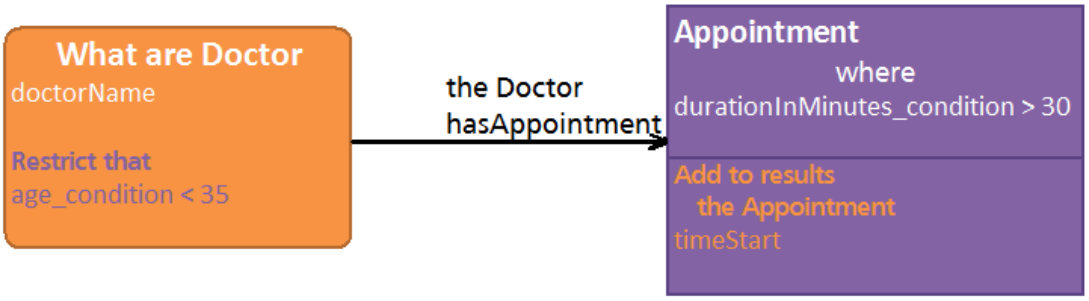


3.13. att. ***Atlasīt vizītes, kas ir ilgākas par 30 minūtēm, un atbildei pievienot vizīšu sākuma laiku.***

Otra būtiskā atšķirība, kas tika izmainīta valodas notācijā, ir tā, ka negāciju un neobligātās grupas tiek aizstātas ar speciālām saitēm. Tika veiktas vairākas lietotāju aptaujas, kur tika vaicāts lietotājiem izskaidrot vaicājumus ar negācijām un neobligātām daļām, kur bija vaicājumi, kas saturēja ViziQuer full notāciju ar apvilktām kastē, notāciju ar bultām un notāciju, kas apvienoja kastes un bultas.

Ņemot vērā, ka ir grūti prezentēt precīzus rezultātus (bieži ir jāpielieto veselais saprāts un intuīcija, lai saprastu, kas tieši tiek izteikts uzrakstītajā vaicājuma dabiskās valodas skaidrojumā, kas arī var saturēt neprecizitātes), šajā daļā tiks izlaists formāls pamatojums izdarītajai izvēlei. Uzskatīsim, ka izdarītās izvēles pamatotību lasītājs varēs pārbaudīt iepazīstoties ar visas ViziQuer lite versijas aptaujas anketas pārbaudes rezultātiem.

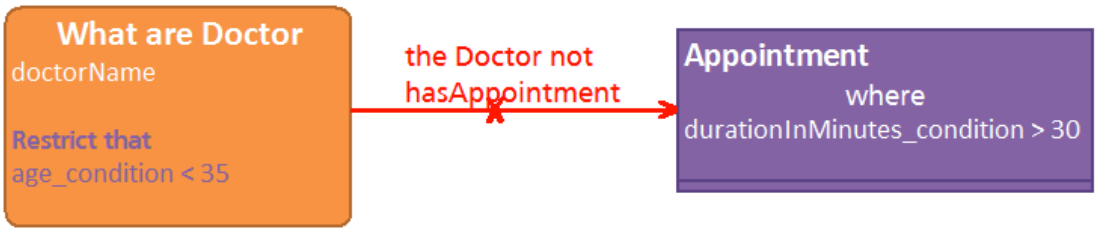
Tad varam aplūkot parasto objektu kopu sasaisti attēlā 3.14., negāciju 3.15. attēlā un neobligāto saiti 3.16. attēlā. Ņemot vērā, ka sasaiste nenorāda precīzu grupas apgabalu, tad neobligātās un negācijas sasaišu gadījumā nedrīkst veidot vaicājumā ciklu, kurš sevī iekļauj kādu no neobligātās vai negācijas tipa saiti.



3.14. att. ***Doktoriem piesaistīti apmeklējumi.***

SPARQL sintakse:

```
?Doctor a Onto:Doctor.
?Doctor Onto:name ?doctorName.
?Doctor Onto:age ?age_condition.
?Doctor onto:hasAppointment ?Appointment.
?Appointment a Onto:Appointment.
?Appointment Onto:durationInMinutes
?durationInMinutes_conditions.
?Appointment Onto:timeStart ?timeStart.
FILTER (?age_condition < XSD:integer("35"))
FILTER (?durationInMinutes _condition > XSD:integer("30"))
```



3.15. att. ***Doktori, kuriem nav vizītes ilgākas par 30 minūtēm***

SPARQL sintakse:

```
?Doctor a Onto:Doctor.
?Doctor Onto:name ?doctorName.
?Doctor Onto:age ?age_condition.
NOT EXIST {
?Doctor onto:hasAppointment ?Appointment.
?Appointment a Onto:Appointment.
?Appointment Onto:durationInMinutes
?durationInMinutes_conditions.
FILTER (?durationInMinutes _condition > XSD:integer("30"))
}
FILTER (?age_condition < XSD:integer("35"))
```

3.16. att. ***Rādīt doktoru, kā arī tās vizītes sākuma laiku, kas ir garākas par 30 minūtēm.***

SPARQL sintakse:

```
?Doctor a Onto:Doctor.
?Doctor Onto:name ?doctorName.
?Doctor Onto:age ?age_condition.
Optional {
?Doctor onto:hasAppointment ?Appointment.
?Appointment a Onto:Appointment.
?Appointment                              Onto:durationInMinutes
?durationInMinutes_conditions.
?Appointment Onto:timeStart ?timeStart.
FILTER (?durationInMinutes _condition > XSD:integer("30"))
}
FILTER (?age_condition < XSD:integer("35"))
```

Ar aprakstītajiem pieciem grafiskajiem elementiem esam pārklājuši vienkāršo un biežāk lietoto vaicājumu daļu, kas nodrošina datu atlasi vienas klases ietvaros, datu apvienošanu no vairākām klasēm, kā arī nosacījumu uzstādīšanu – kā vienas klases ietvaros, tā arī papildus nosacījumus saistītās klasēs.

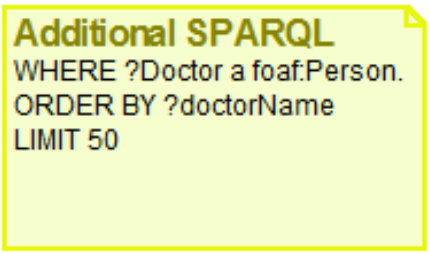
Taču, lai valoda būtu pilnīgāka un ļautu lietotājiem ievadīt daudz patvaļīgāku vaicājumu, kas pārklātu plašāku izteikumu lauku, tad ievedīsim papildus grafisku elementu – "*SPARQL full*", kur lietotāji var pierakstīt klāt patvaļīgus izteikumus jau esošajiem, piemēram, papildus nosacījumu, ka vajag atlasīt tos objektus, kuriem sakrīt kādas atribūtu vērtības.

Tāpat elements "*SPARQL full*" nodrošina iespēju noformēt SPARQL atbildi atlasot vēl papildus atribūtu vērtības, uzstādot vai ir nepieciešams atlasīt visas vērtības, vai arī tikai unikālās (lietot *Distinct* īpašību pār atbildes kopu), kā arī atzīmēt pēc kādiem papildus atribūtiem grupēt, vai limitēt atbildes lielumu.

Ja pieejam ļoti formāli, tad neievadot grafikā nevienu elementu ir iespējams elementā "*SPARQL full*" pierakstīt patvaļīgu SPARQL vaicājumu, kas ietilpst *SELECT* apakškopā.[17] Attēlā 3.17. varam aplūkot "*SPARQL full*" piemēru. Dotajā elementā tiek pateikts, ka papildus ar citiem

[17] Formāli SPARQL ir iespējams pierakstīt arī *ASK*, *DESCRIBE* un *CONSTRUCT* vaicājumus. Taču katram vaicājumu tipa ir atšķirīgs mērķis. Ja *ASK* pēc būtības ir apakšgadījums *SELECT* un atgriež rezultātu, vai izvēlētajam risinājumam ir rezultāti, tad *DESCRIBE* atgriež papildus datus par izvēlētajiem resursiem, bet *CONSTRUCT* ļauj izveidot jaunus RDF datus.

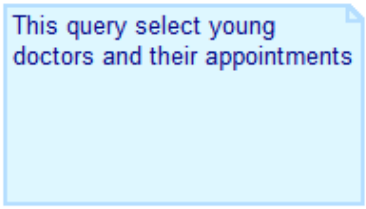
elementiem uzzīmētajam vaicājumam vajadzēs pievienot arī nosacījumu, ka doktors arī pieder klasei foaf[18]:Persona. Tāpat ir atzīmēts, ka atbildi vajadzēs kārtot pēc ?doctorName un vajadzēs rādīt tikai pirmos 50 ierakstus.



3.17. att. ***SPARQL full grafiskās valodas vaicājuma elements.***

Izmantojot "SPARQL full" sintaksi, nepieciešamības gadījumā, eksperta lietotāji varēs pierakstīt arī ļoti sarežģītus un komplicētus vaicājumus. Tas palīdzēs uzturēt visus vaicājumus vienā vietā un pārklās arī plašās SPARQL vaicājumu iespējas, kas citādi tiek reti lietotas.

Tāpat vaicājumus var papildināt ar tekstuāliem komentāriem, kur var ietvert papildus informāciju, lai vaicājumi vēlāk būtu vieglāk saprotami. Piemēram, jau vārdiski ierakstīt, kāds vaicājums grafiski ir izveidots. Komentāra elementa piemērs ir attēlots 3.18. attēlā.

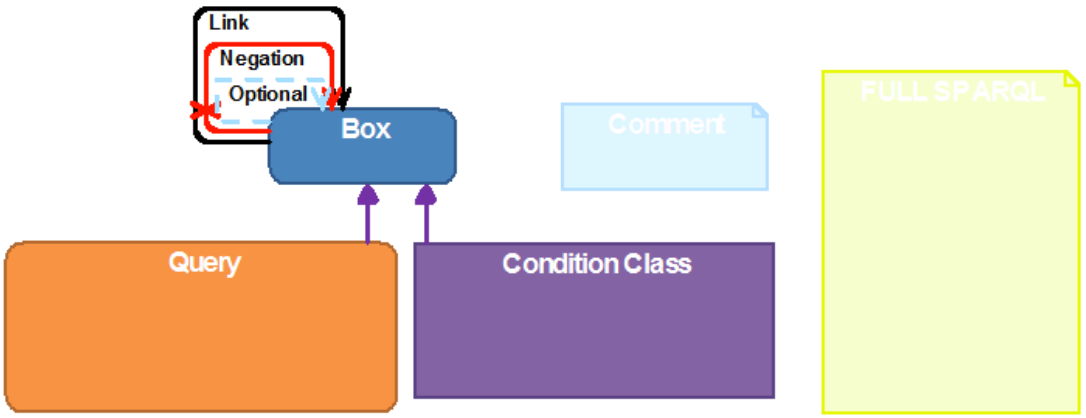


3.18. att. ***Komentāra elementa piemērs vaicājuma valodā.***

Lai arī tīri teorētiski esam izveidojuši stiprāku izteiksmes iespēju ViziQuer lite versijā salīdzinot ar ViziQuer full, kas tika aprakstīts iepriekš, tomēr lielākā daļa sarežģīto lietu ir jāpieraksta tādā pašā tekstuālā formātā kā parasti SPARQL vaicājumi. Līdz ar to esam samazinājuši pašu valodu tikai uz vienkāršu datu kopu atlasi, nosacījumu uzstādīšanu šīm kopām, kā arī savstarpēju kopu sasaisti.

Varam aplūkot izstrādātās vaicājumu valodas ViziQuer lite versijas grafiskās notācijas sintaksi 3.19. attēlā. Tur ir ieviests papildus, abstrakts grafisks elements "*Box*", kas apzīmē objektu kopas atlasi un ļauj nešķirot vai tas ir vaicājuma sākuma elements, vai kāds no papildinošajiem elementiem.

[18] http://www.foaf-project.org/ projektā ir definēta "Friend of a Friend (FOAF)" ontoloģija.



3.19. att. ***ViziQuer lite grafiskā notācija.***

3.4. ViziQuer lite vaicājuma valodas lasīšanas analīze

Līdzīgi kā ViziQuer full gadījumā, arī ViziQuer lite versijā vēlamies pārbaudīt vai tā ir lietotājiem saprotama, kā arī – vai ir kādas izteiktas konstrukcijas, kuras apgrūtina lietotājiem vaicājumu lasāmību un tās būtu jāmaina.

Lai lasāmības pārbaude būtu korekti salīdzināma ar ViziQuer full pārbaudi, tad tika izvēlēts tieši tas pats grafisko vaicājumu komplekts, kas tika uzdots dalībniekiem, lai viņi tos pārrakstītu saviem vārdiem no grafiskā formas dabiskajā valodā.

Taču šajā gadījumā tika ieviesti nelieli ierobežojumi tam, ko ir jāsatur atbilžu rezultātiem. Ņemot vērā, ka pirmā pieredze ar pūļresursēšanas izmantošanu parādīja, ka ir atsevišķi negodprātīgi dalībnieki, tad, lai mazinātu šādu dalībnieku atalgošanu, atsevišķiem vaicājumiem bija jāsatur minimālas teksta frāzes, kas parādās grafikā, piemēram, regulāra izteiksme ".*(ame).*". Tas nozīmē, ka lietotājiem ir jāuzraksta atbilde, kur parādās vārds "*name*".

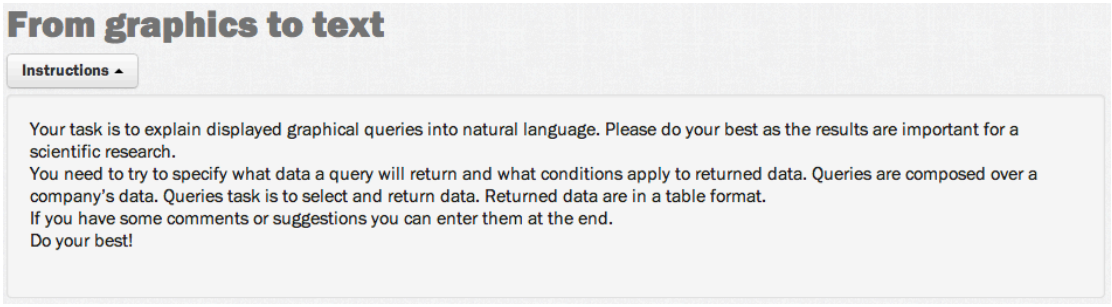
Šādi ierobežojumi uzlabo atbilžu rezultātu, jo vairākos vaicājumos būtu jāparādās kādai noteiktai frāzei, kas parādās arī vaicājumā. Taču vēlamies vērst uzmanību, ka tam nevajadzētu būtiski iespaidot aptaujas dalībnieku sniegtos atbilžu rezultātus.

Pirmkārt, visās vietās, izņemot vienu, dalībniekiem vajadzēja skaidrojošajā tekstā minēt tikai vienu vārda daļu, turklāt tā parasti tika uzdota bez pirmā burta, lai izslēgtu variantu, ka tas var nenostrādāt tāpēc, ka kāds to ir pierakstījis ar lielo burtu, nevis mazo. Vienīgajā izņēmumā tika prasītas divu vārdu frāzes, kas varēja būt uzdotas patvaļīgā secībā.

Otrkārt, daļa no vaicājumiem nesaturēja šādas pārbaudes, un neatbilstības regulārās izteiksmes pārbaudei tiek uzrādītas tikai pilnībā aizpildot un mēģinot iesūtīt atbildes. Līdz ar to šādos gadījumos šīm atbildēm vajadzētu atspoguļot vājās vietas vaicājumu skaidrošanā, ja tādas būtu.

Pat, pielietojot šādu negodprātīgu lietotāju atsijāšanas metodi, atbilžu sarakstā bija daži aptaujas dalībnieki, kas bija atraduši veidu kā sakopēt dažādos grafikā esošos vārdu pa visiem laukiem. Šīs atbildes neiekļāvām kopējā statistikā, kas tika iegūta analizējot lietotāju atbildes.

Papildus arī iedalījām aptaujas dalībniekus divās daļās. Pirmā aptauja saturēja vienkāršu skaidrojumu par darba uzdevumu, kas ir jāveic un tālāk lietotājiem bija nepieciešams izskaidrot dabiskā valodā esošos vaicājumus. Darba uzdevuma aprakstu ir iespējams aplūkot 3.20. attēlā.



3.20. att. ***Aptaujas dalībnieku instrukcija uzsākot darbu.***

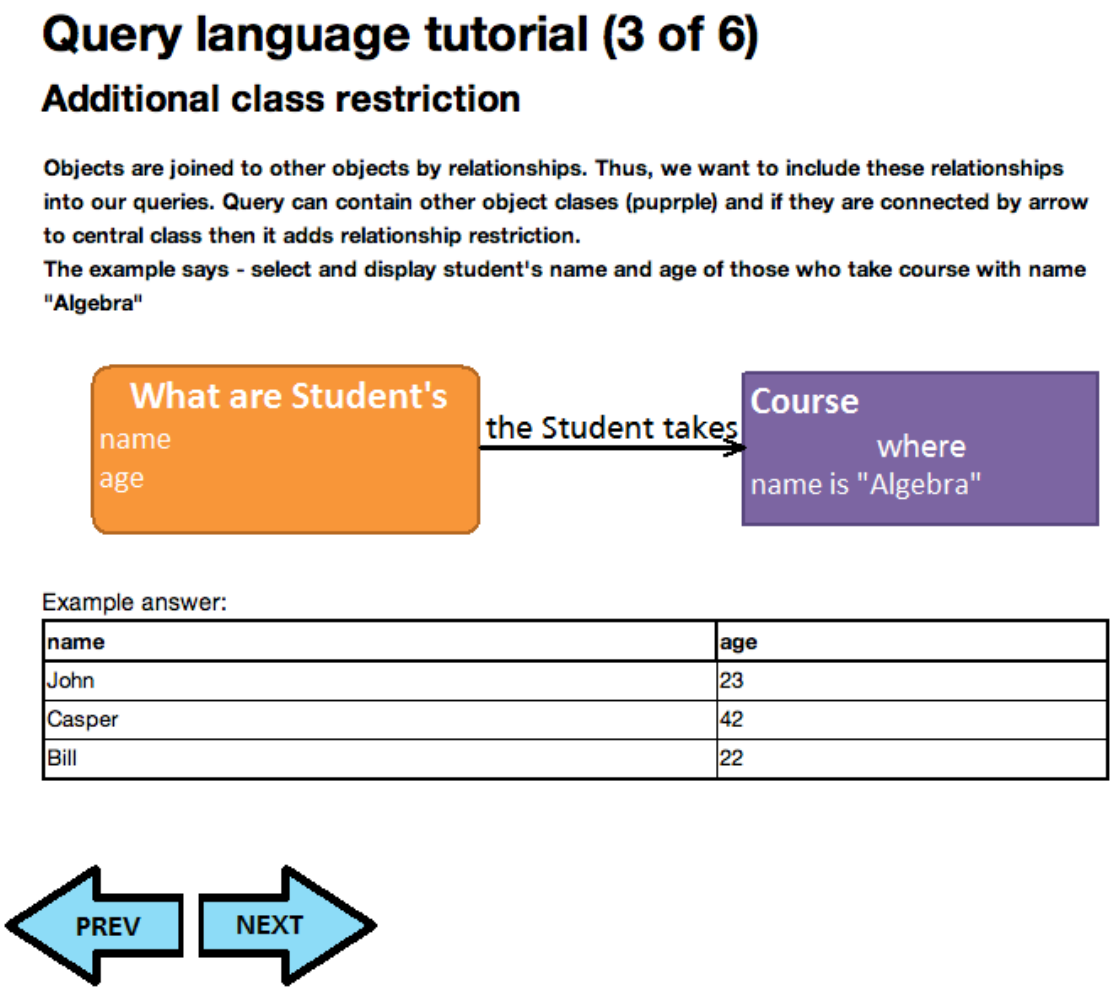
Kamēr otrajā aptaujā dalībniekiem tika piedāvāta arī papildus iespēja aiziet uz saiti, kur bija iespēja iepazīties ar īsu pamācību par grafiskās valodas vaicājuma konstrukcijām.

Apmācībā tika parādītas visas konstrukcijas. Secīgi tik parādīti piemēri:

- Vienas klases objektu un to atribūtu atlase;
- Vienas klases objektu atlase, papildus uzstādot atribūtu nosacījumu klases ietvaros;
- Vienas klases objektu atribūtu atlase un uzstādīta saistītā klase kā papildus nosacījums (šo piemēru varam aplūkot 3.21. attēlā);
- Divu klašu saistība, kad atribūtu vērtības tiek atlasītas no abām klasēm;
- Papildus klases nosacījums, ja tas ir saistīts ar negācijas saiti;
- Atribūtu atlase no divām klasēm, ja otrā klase ir piesaistīta ar neobligāto saiti;

Lai rezultāti būtu iespējami objektīvi, tad abos variantos tika aptaujāti 30 dalībnieki, taču katrā no aptaujām tika izvērtēti 26 respondenti, jo pārējo 4 respondentu atbildes bija zemas kvalitātes un to motivācija bija iegūt atlīdzību, nevis sniegt korektas atbildes. Kā motivācija aptaujas dalībniekiem tika izsniegta atlīdzība 37 ASV centi, kas, mūsuprāt, bija pietiekama, lai atbildes sniegtu pietiekami labā līmenī un tās būtu objektīvi novērtējamas.[19]

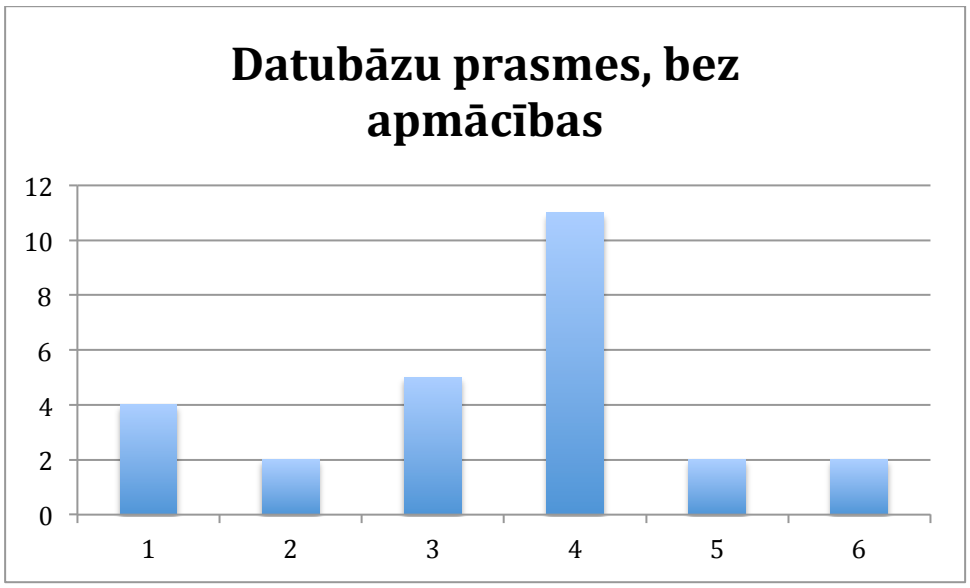
[19] Ir jāņem vērā, ka vidēji dalībnieki patērēja 20 minūtes, lai aizpildītu aptaujas anketu, kas nozīmē, ka atlīdzība bija 1 dolārs stundā. Ātrākie dalībnieki atbildēja 10 minūtēs, kamēr lēnākie anketas aizpildīšanai veltīja līdz pat 50 minūtēm. Papildus mēs uzskatām, ka dalībnieki centās izdarīt to iespējami labi, jo uzdevums bija izaicinošs un interesants arī pašiem dalībniekiem, par ko liecina, piemēram, atstātais komentārs "I haven't look at database queries in a long time, this was fun!". Lai arī bija vairāki komentāri, kas atzīmēja, ka samaksa par šādas grūtības uzdevumu ir nepietiekama.



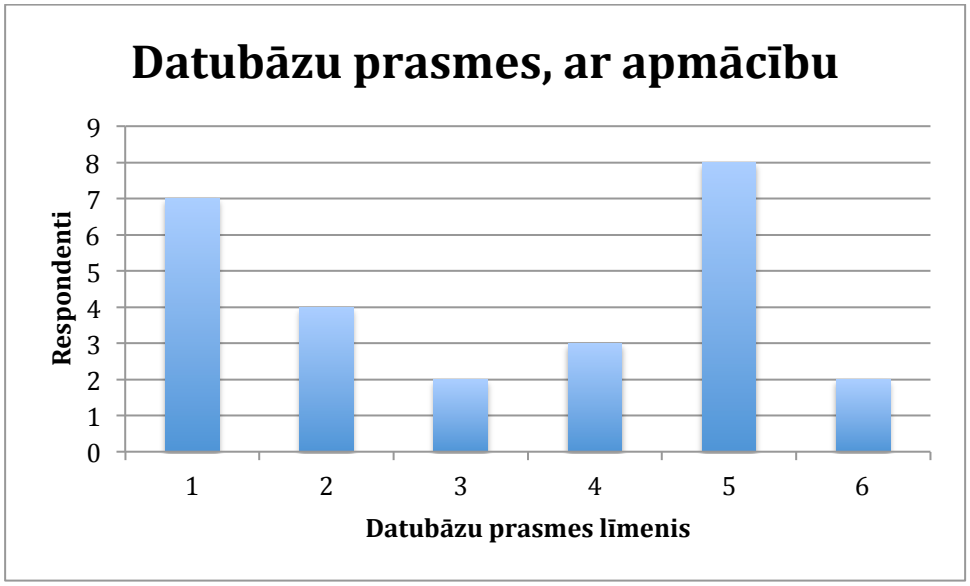
3.21. att. ***Vaicājuma apmācības piemērs.***

Aptaujas dalībnieki varēja atzīmēt, viņuprāt, savu zināšanu līmeni darbojoties ar datubāzēm skalā no 1-6 un viņu prasmes darbā ar datoru skalā no 1-7. Attēlos 3.22, 3.23, 3.24 un 3.25 ir aplūkojams apkopojums – kāds ir lietotāju sadalījums pa prasmju līmeņiem. Kā arī ir izdalīti atsevišķi grafiki lietotājiem, kas aizpildīja anketu ar apmācību, kā arī tie, kas aizpildīja anketu bez apmācības. Ja apskatam lietotāju atzīmēto vidējo prasmju līmeni darbā ar datu bāzēm, tad gadījumā ar apmācību tas ir 3.27, kamēr bez apmācības tas ir 3.42 no 6. Atbilstoši IT zināšanas ir 5.08 un 5.31 no 7.

Kā redzam, tad vidējie rādītāji ir tuvu mūsu mērķauditorijai, jo ar vaicājumu valodu visdrīzāk darbosies lietotāji ar novērtējumu datubāzu zināšanās ar 3, kā arī IT zināšanās ar 5, kas ir tuvu iegūtajām vidējām vērtībām, tāpēc iegūtie rezultāti atbilstoši atspoguļos izstrādātās vaicājuma valodas lasāmību tās mērķauditorijā.



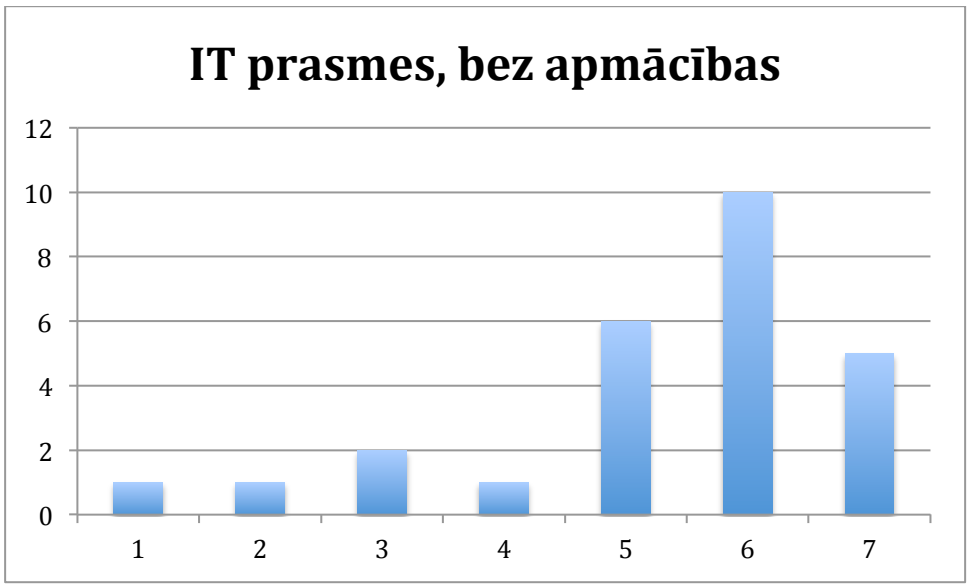
3.22. att. ***Aptaujas bez apmācības dalībnieku prasmes darbā ar datu bāzēm***



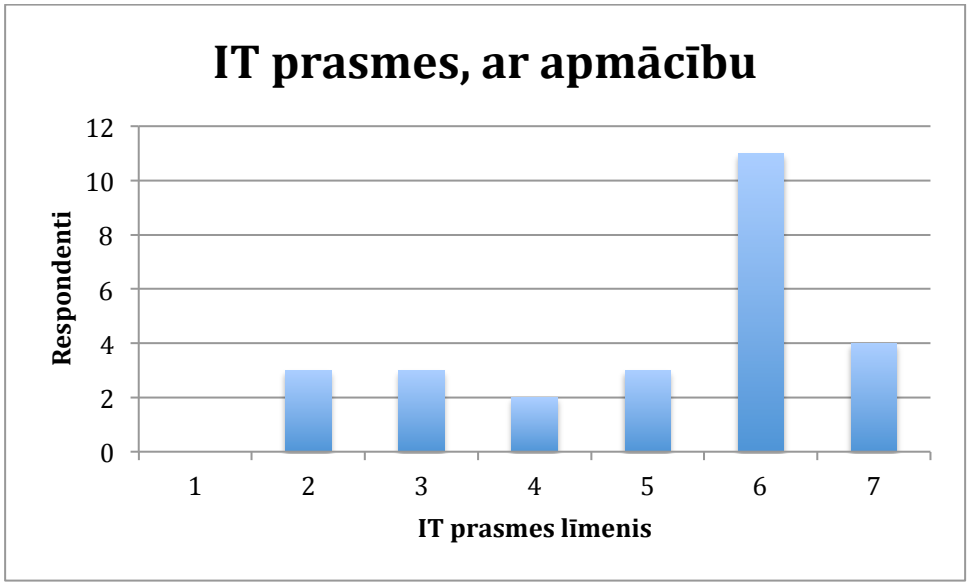
3.23. att. ***Aptaujas bez apmācības dalībnieku prasmes darbā ar datu bāzēm***

Ja aplūko grafikus, tad ir arī atbilstoši, ka veidojas nedaudz atšķirīgi sadalījumi, kas var norādīt arī uz apmācību būtību grafiskās vaicājumu valodas apgūšanā. Ja bez apmācībām ir izteikti pīķi, kas var norādīt uz to, ka ir nepieciešams kaut kāds zināšanu līmeni, pēc kura izprot uzdevuma būtību.

Kamēr dalībniekiem ar apmācību līkne ir daudz vienmērīgāk sadalīta. To var izskaidrot ar situāciju, ka lietotāji, iepazīstoties ar apmācību, saprot labāk uzdevuma būtību, kā arī gūst priekšstatu kā to labāk risināt, tāpēc nav nepieciešamas tik plašas iepriekš iegūtas priekšzināšanas.



3.24. att. ***Aptaujas bez apmācības dalībnieku prasmes darbā ar datu bāzēm***



3.25 att. ***Aptaujas bez apmācības dalībnieku prasmes darbā ar datu bāzēm***

3.5. ViziQuer lite vaicājuma valodas respondentu datu analīze

Izstrādājot ViziQuer lite versiju izvirzījām mērķi, ka tā būs lasāma 80% respondentu. Tāpēc varam apskatīt vaicājumus, kurus nav spējuši pietiekami labi izlasīt vairāk nekā 20% respondentu.

Izvērtējot atbildes ir viegli atzīmēt tās, kas ir pilnīgi pareizas, kamēr ir tādas atbildes, ko ir grūti atzīmēt gan kā pareizas, gan nepareizas. Šādos gadījumos izvēlējāmies biežāk pozitīvo risinājumu un atzīmējām tās kā pareizas. Piemēram, neobligāto grupu gadījumā, ja ir jāatlasa darbinieka vārds, uzvārds un adrese, ja tāda ir, tad pozitīvi atzīmējām arī tās atbildes, kur

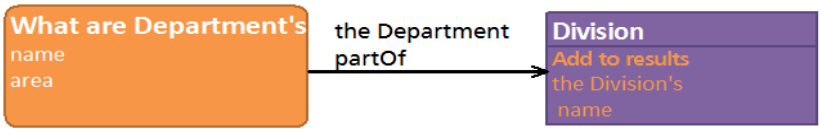
vienkārši teica, ka ir jāatlasa personas vārds, uzvārds un adrese. Tas pēc būtības ir pareizi, taču trūkst piebildes, ka dažām personām adrese var būt tukša.

Pirmkārt, tas ir vaicājums, kas ir attēlots 3.9. attēlā, ka ir jāatlasa darbinieki, ja viņiem pieder dzīvoklis un viņi dzīvo Milānā vai Romā. Jau iepriekš identificējām, ka šī problēma vairāk ir kognitīva un ir grūti piedāvāt saprātīgu risinājumu. Atbilstoši 38.5% respondentu apmācības gadījumā un 34.6% respondentu bez apmācības nespēja pilnīgi korekti izskaidrot šo vaicājumu, jo visbiežākā kļūda bija tā, ka tika norādīts, ka darbiniekiem pieder dzīvoklis Milānā vai Romā.

Nopietnāka problēma parādījās vaicājumā, kas ir attēlots 3.26. attēlā. Šajā vaicājumā ir jāatlasa apakšnodaļas dati un jāpieliek klāt nodaļas nosaukums. Abos gadījumos kļūdas pieļāva 23.1% respondentu. Kā galvenais pārpratums ir minēts tas, ka nodaļa ir kā papildus nosacījums, nevis arī atgriež papildus datus. Taču daļēji šo var izraisīt grafiskajā notācijā lietotais artikuls pie nodaļas atribūtu pievienošanas. Lietojot *the* tiek apzīmēts, ka tas ir tikai viens, kā arī, aprakstot vaicājumu, lietotāji izmantoja pie nodaļas šo pašu *the* artikulu.

Grafiskās valodas realizācijā aizstāsim *the* artikulu ar *this* apzīmētāju, kas norādīs, ka ir jāpievieno atbildei šīs klases atribūti rezultātam, taču nenozīmēs, ka tāda var būt tikai viena vērtība, kā tas bija *the* artikula gadījumā.

Taču esam ieguvuši būtisku uzlabojumu lietotāju izpratnē par datu kopu apvienošanu, jo sākotnēji pareizi šo lietu spēja izskaidrot 1 no 9 cilvēkiem (11.1%), kamēr tagad šajā jautājumā nepilnības ir 23.1%. Turklāt, arī atlikušā daļa ir tuvu vēlamajam rezultātam, jo darbojoties ar vaicājumiem būtu redzams, ka specifiska nodaļa netiek specificēta, kurai vajag atgriezt apakšnodaļu rezultātus. Daļēji tas atspoguļojas arī citās lietotāju atbildēs, kas pieļāva kļūdu šajā vaicājumā, vēlāk nepieļāva kļūdu skaidrojot kādu citu apvienojumu, kur pievienotai klasei ir papildus uzdoti nosacījumi.



3.26. att. ***Atrast apakšnodaļas un to nodaļas.***

Gadījumā bez apmācības aptaujas dalībniekiem radās problēmas arī izskaidrot vaicājumu, kur notiek atsaukšanās nosacījumu daļā uz citu elementu grafā. Vaicājums ir apskatāms 3.4. attēlā. Metodoloģiski iesakām šādu vaicājumu neveidot un izmantot "SPARQL *full*" grafisko elementu, lai pierakstītu minēto nosacījumu. Taču saglabājām atbilstību ar sākotnējo aptauju, lai tās būtu salīdzināmas.

Šajā vaicājumā kļūdījās 23.1% respondentu. Galvenā pieļautā kļūda bija tā, ka aptaujas dalībnieki vēlējās uzstādīt uzvārdu Smits, kā nosacījumu darbiniekiem, kuriem ir jāatlasa alga, tāpēc tika minēts vienkārši, ka ir jāatlasa darbinieks ar uzvārds Smits algu saraksts.

Taču vaicājumu daudz labāk izskaidroja respondenti, kuri iepazinās ar apmācību, jo tur kļūdu pieļāva tikai 7.7% respondentu.

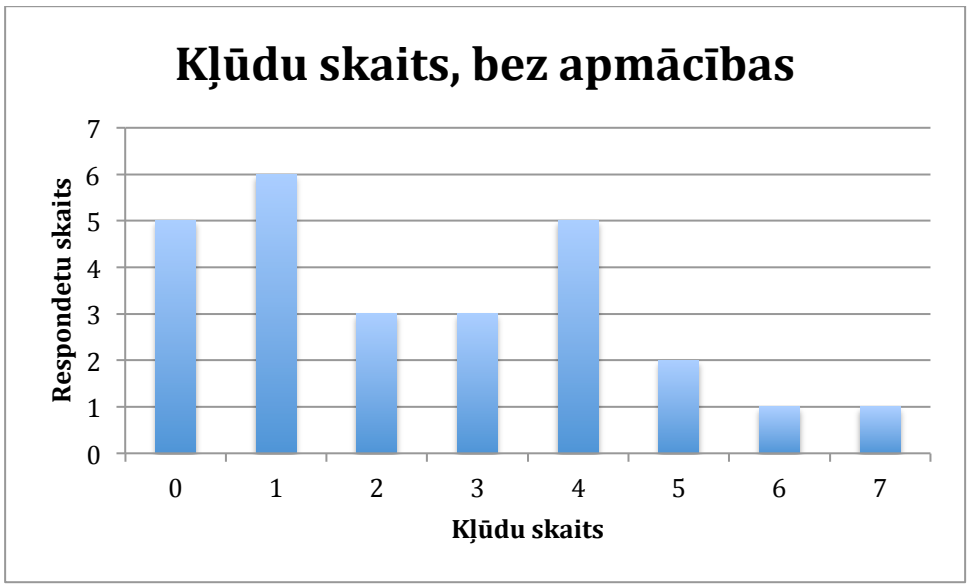
Tāpat abu aptauju dalībnieki saglabāja problēmas precīzi izskaidrot vaicājumus, kur figurēja neobligātās grupas jēdziens. Lai arī šajos vaicājumos identificējamo kļūdu procents bija 10% dalībniekiem ar apmācību un 19% dalībniekiem bez apmācības.

Taču bija arī pietiekami liels procents lietotāju, kuri neatzīmēja šo neobligāto daļu. Dalībniekiem ar apmācību tas bija krietni mazāks un bija apmēram 15%, kamēr dalībniekiem bez apmācības tas sastādīja apmēram 30%. Tas viennozīmīgi ir uzlabojums attiecībā pret ViziQuer full vaicājumu valodas lasāmību, kur lasītāji gandrīz vispār neatzīmēja šādu īpašību.

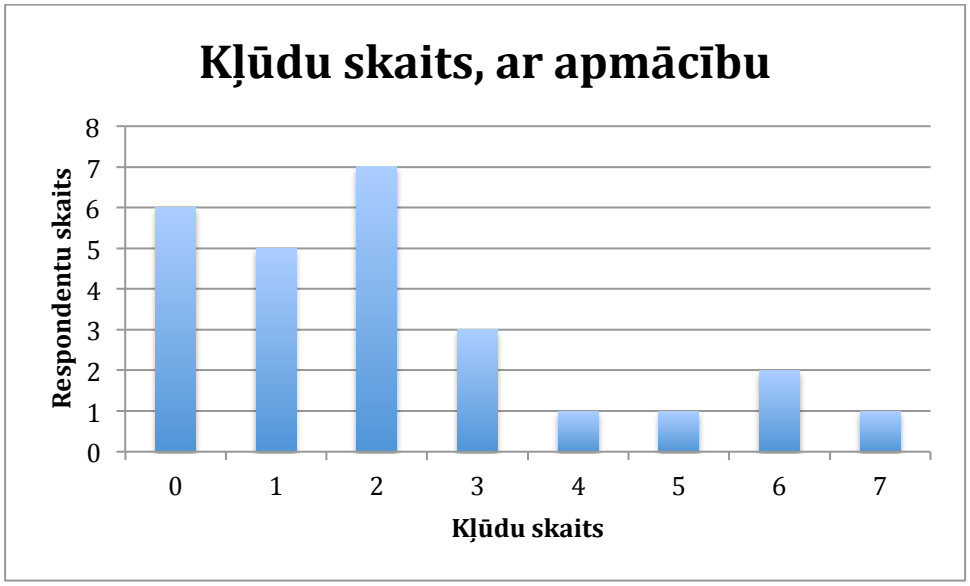
Esam novērsuši lielāko daļu nepilnību, ko identificējām analizējot ViziQuer full valodas grafisko konstrukciju lasāmību. Ņemot vērā, ka tikai divi vaicājumi neizpildīja izvirzīto kritēriju, ka tie būs lasāmi 80% gadījumu, tad varam uzskatīt, ka uzlabotā versija apmierina mūsu izvirzītās prasība.

Attēlos 3.27 un 3.28. var aplūkot kļūdu sadalījumu. Kā redzams, tad gadījumā ar apmācību tas stipri vairāk ir koncentrējies pie mazāka kļūdu skaita, kamēr gadījumā ar apmācību, tas ir lēzenāks. To arī atspoguļo vidējais kļūdu skaits 2.16 gadījumā ar apmācību un 2.46 gadījumā bez apmācības. Tas arī vedina domāt, ka pat īsa apmācība spēj uzlabot rezultātus.

Daudz izteiktāki, apmācības nozīme parādās lietotājiem ar vājākām priekšzināšanām. Ja izdalām divas grupas, kur cilvēki ar zināšanām par datubāzēm no 1-3 ir vienā grupā, bet no 4-6 otrā grupā, tad var daudz izteiktāki atzīmēt apmācību nozīmi, jo bez apmācības tiek pieļautas vidēji 3.91 kļūdas, kamēr ar apmācību tās ir 2.77 kļūdas. Varam redzēt, ka pat īsai apmācībai ir būtiska loma, lai uzlabotu rezultātus. Kamēr eksperta lietotājiem šī atšķirība īsti nepastāv, jo bez apmācības ir vidēji 1.4 kļūdas, kamēr ar apmācību tās ir vidēji 1.5. Tātad redzam, ka iepriekšējas zināšanas un pieredze ļoti atvieglo un palīdz vaicājuma valodas uztverē.



3.27. att. ***Kļūdu skaits aptaujās bez apmācības***



3.28. att. ***Kļūdu skaits aptaujās ar apmācību.***

Ja aplūkojam vidējo laiku, kas ir patērēts aptaujas aizpildīšanai, tad patiesībā eksperta lietotāji tam ir vidēji patērējuši par apmēram 3 minūtēm vairāk nekā lietotāji ar mazākām priekšzināšanām. Ar apmācību eksperta lietotāji vidēji patērēja 26 minūtēs, kamēr lietotāji ar mazākām priekšzināšanām tam veltīja 23 minūtes (šai laikā ir iekļauta arī iepazīšanās ar apmācību). Bez apmācības eksperta lietotāji vidēji patērēja 23 minūtes, kamēr lietotāji ar mazākām priekšzināšanām tam veltīja vidēji 20 minūtes. Ar aptaujai veltīto laiku arī daļēji varam skaidrot, kāpēc eksperta lietotāju rezultāts ir labāks, jo aptaujas atbildes ir pildītas rūpīgāk. Tāpat varam novērtēt arī laiku, kas tika veltīts apmācībai, kas ir vidēji 3 minūtes, kā arī redzam, ka

lietotājiem ar mazākām priekšzināšanām šīs trīs papildus minūtes būtiski uzlabo rezultātus, kamēr ekspertu lietotājiem tas pārlieku nepalīdz tos uzlabot.

Subjektīvi var izvērtēt arī atbilžu tekstuālo kvalitāti. Izteikti tas parādījās lietotājiem ar minimālām zināšanām par datubāzēm, jo atbildes bija formulētas reizēm vairāk kā minējums, vai tas ir pareizi. Turklāt, īpaši izteikti tas bija anketā, kur nebija iespējams iepazīties ar apmācību. Kamēr eksperta lietotāji to bieži formulēja ļoti noteikti, reizēm pat lakoniski.

3.6. ViziQuer lite vaicājuma valodas noslēdzošie secinājumi

Aptaujas anketās bija sastopami arī vairāki interesanti atbilžu varianti, kas ļauj paskatīties uz uzdotajiem vaicājumiem nedaudz citādāk. Piemēram, vaicājums atlasa visus dalībniekus, kam reizē arī iekļaujam izpildīt komandu. "*Attention all employees in production, state your name, surname and address.*". Lai arī tas maz pasaka par vaicājuma valodas konstrukcijām, bet tas ir lielisks metodoloģisks piemērs kā vieglāk ir izskaidrot vaicājumu būtību un padarīt to saprotamāku lietotājiem.

Otrs interesantais piemērs sevī saturēja ne tikai vaicājuma skaidrojumu, bet arī piemēru, kur uzņēmumā tas varētu tikt lietots. "*This gives employees name, the department they work in and their address. More specific that the other Queries. This could be helpful with sending out payroll checks and other notices.*". Arī šajā gadījumā šis ieteikums var tikt izmantots metodoloģiskā apmācībā, lai labāk izprastu, kur vaicājumi var tikt lietoti.

Ņemot vērā, ka mūsu uzdevums ir tikai pārveidot grafiskus vaicājumus uz tekstuālu SPARQL formātu, ko tālāk izpilda SPARQL piekļuves punkts, tad nerisinām problēmu, kā datus parādīt lietotājam. Taču vairākas atbildes uzsvēra šo īpašību, ka nav tikai būtiski kādus laukus vaicājums atgriezīs, bet arī kādā formātā tie tiks attēloti lietotājam. "*For a given department name all employees*" – mūsuprāt, aptaujas dalībnieks vēlas redzēt nodaļas un atbilstoši katrai nodaļai arī ieraudzīt visus darbiniekus. Vai līdzīgs piemērs – "*I cannot figure out what they are looking for here. Could it be a list of the employees by department?*". Kā arī – "*This query will return a list of employee name and surname in each named department*" – kas viss uzskatāmāk parāda, ka anketas aizpildītājs redz, ka katram nosauktajai nodaļai ir piekārtots saraksts ar visiem darbiniekiem, kas ir šajā nodaļā.

Taču vispozitīvākie bija daži komentāri, kas norādīja, ka vaicājuma valoda ir izstrādāta pareizā virzienā un tā var kļūt lietotājiem draudzīga un ērti lietojama – "*I haven't looked at*

database queries in a long time, this was fun!"[20] vai "*I like the graphical query. It is short and precious enough.*", vai ļoti īss un konkrēts "*queries are good*".

Noslēdzošais secinājums ir ļoti vienkāršs – esam izstrādājuši lasāmu vaicājumu valodu, ir nepieciešama ļoti īsa apmācība, lai uzlabotu lietotāju orientāciju vaicājumu lasīšanā, kā arī cilvēki atzīst, ka attēlotie vaicājumi ir skaidri un pietiekami īsi.

[20] Cilvēkiem ir tendence darīt lietas, kas viņiem patīk un sagādā prieku. Ja vaicājuma valodas grafisks formāts to spēj darīt, tad ir lielāka iespējamība, ka cilvēki to praktiski lietos.

4. MEDICĪNAS PROJEKTS

4.1. Semantiskās Latvijas vīzija

Tim Berners Lī un ko prezentēja semantisko tīmekli [6] – tā pamatā liekot divas fundamentālas lietas. Pirmkārt, semantiski dati, kas ir lietotājiem saprotami un nesatur lieku, tehnisku informāciju. Otrkārt, tie ir pieejami aģentiem, rīkiem, lai ar tiem varētu darboties un izgūt jaunu informāciju.

Šīs idejas kontekstā tika arī izvirzīta tēze par "Semantisko Latviju" [5], kā dažādu datu bāžu apvienojumu, kas būtu pieejams lietotājiem. Ņemot vērā, ka tas nesaturētu tehnisko informāciju, tad tas kļūtu lasāmāks un arī ērtāk lietojams plašākam lietotāju lokam.

Taču centrālā raksta "Semantiskā Latvija" tēze slēpjas faktā – kā semantiskos datus padarīt patiešām pieejamus lietotājiem. Kādi ir iespējamie scenāriji to lietošanā un kādi ir nepieciešamie rīki, lai to praksē varētu īstenot.

Tieši šādā kontekstā, kā viena no tēzēm arī tiek uzsvērta nepieciešamība pēc grafiskas vaicājumu valodas, ar kuras palīdzību būtu iespējams lietotājiem uzstādīt vaicājumus un atlasīt datus. Raksta kontekstā šī valoda tiek saukta par "DEMO". Ar vaicājumu valodas grafiku varēja iepazīties darba 2. un 3. nodaļā.

Taču otra būtiskā raksta tēze – kā padarīt pieejamas ontoloģijas lietotājiem. Ka centrā būtu jānoliek ontoloģiju repozitorijs, ar apstiprinātām ontoloģijām, kuras lietotāji varētu izvēlēties un izmantot, lai radītu semantisku datu tīmekļa informāciju.

Rakstā tiek izvirzīta tēze, ka šādu ontoloģiju izmantošanai būtu jānotiek trīs vienkāršos soļos:

1. Apmeklējot "Semantiskās Latvijas" ontoloģiju portālu, tiek sameklēta atbilstošā ontoloģija.
2. Ontoloģija tiek ielādēta rīkā, kas ļauj rediģēt tās instanču datus
3. Rīks piedāvā saglabāt datus HTML formātā

Ņemot vērā, ka raksts aprakstīja vīziju 2006. gadā, tad daudzas no lietām tehnoloģiski nebija pietiekami attīstītas, lai to varētu pienācīgi īstenot. Ja atskatāmies no šī brīža situācijas, tad darba turpinājumā aprakstīsim jau precīzāku vīzijas izklāstījumu.

Lai arī "Semantiskās Latvijas" netiek minēta semantisko datu ieguve no relāciju datu bāzēm, tomēr praksē tas bija viens no sākotnējiem punktiem, kur tika iegūti praktiski lietojami semantiskie dati.

Ņemot vērā "Semantiskajā Latvijā" izteiktās idejas par plašākai publikai pieejamiem datiem un ontoloģijām, lai šie dati būtu pārskatāmi un lietojami tika īstenots projekts "Valsts programma „Latvijas iedzīvotāju dzīvildzi un dzīves kvalitāti apdraudošo galveno patoloģiju zinātniska izpēte ar daudznozaru pētnieciskā konsorcija palīdzību". Projekts Nr. 14 „Vienotas un visiem pētniekiem pieejamas datubāzes izveide par galveno dzīvildzi un dzīves kvalitāti apdraudošo patoloģiju un to riska faktoru izplatību Latvijas iedzīvotājiem"", kurā piedalījās darba autors.

Darba autora galvenie uzdevumi bija saistīti ar vaicājumu valodas prototipa izstrādi, kuru varētu izmantot, lai lietotāji spētu atlasīt datus, kā arī daļēju dalību ontoloģiju izveides procesā. Papildus informāciju par medicīnas projekta īstenošanas detaļām lasītājs var iegūt rakstā [18].

Ja skatāmies no šodienas skatupunkta, tad medicīnas joma ir viena no labākajiem piemēriem semantiskā tīmekļa jomā. Pateicoties šīs nozares ieinteresētībai Semantiskā tīmekļa tehnoloģijās ir izstrādāts viens no populārākajiem un zināmākajiem ontoloģiju redaktoriem – Protege [19]. Tāpat ir izstrādāti arī ontoloģiju repozitoriji, kur ir pieejamas dažādas medicīnas ontoloģijas, piemēram, BioPortal [20].

Ja atskatāmies uz "Semantiskās Latvijas" ideju, tad BioPortāl platforma nodrošina centralizētu piekļuvi ontoloģijām, iespēju tajā meklēt dažādus jēdzienus, definēt savstarpējas attiecības starp ontoloģijām.

Lai arī atšķirībā no "Semantiskās Latvijas" idejas, BioPortal ir vairāk balstīts uz *wiki* pieeju[21], kamēr "Semantiskajā Latvijā" tika prezentēta ideja par centralizēti izstrādātām ontoloģijām, kas būs pietiekami visaptverošas, lai spētu apmierināt lietotāju vēlmes.

Svarīgi atzīmēt, ka tai laikā tika likti pirmsākumi arī atvērtajiem datiem (*Open Data*) pirmsākumiem, piemēram, rosinot publicēt semantiskā formātā Lielbritānijā pieejamos publiskā sektora datus [21]. Ja atskatāmies no šodienas skatupunkta, tad pirmkārt, atvērti semantiskie dati jau ir pieejami Lielbritānijas portālā data.gov.uk.

Taču šobrīd šis temats ir ļoti izplatīts visā pasaulē, piemēram, Eiropas Savienības atvērto datu portāls[22], kas satur norādes arī uz citiem Eiropas Savienību valstu iekšienē esošajiem projektiem, vai Japānas atvērto datu portāls[23].

[21] Wiki pieeja mūsu izpratnē ir kopienas veidoti dati, ko uztur pati kopiena. Ja kāds, kas izveidojis sākotnējos datus (ontoloģiju) ir pieļāvis kādas nepilnības, tad citiem lietotājiem ir iespēja to izlabot.
[22] https://open-data.europa.eu
[23] http://linkedopendata.jp/

Latvijas medicīnas atvērto datu projekts bija tos padarīt pieejamus Latvijas medicīnas zinātniskajam personālam, lai būtu iespējams veikt plašākus pētījumus šajā jomā. Taču tajā pašā laikā nepadarīt šos datus pilnīgi publiski pieejamus, jo tie satur sensitīvu informāciju.

4.2. Projekta izejas dati

Latvijas medicīnas datubāzes pārklāj svarīgākās pataloģijas, kas apdraud cilvēku veselību un dzīves kvalitāti, piemēram, diabēts vai vēzis. Tās tiek glabātas relāciju datubāzēs.

Dati glabājas anonimizēti formā attiecībām uz personām. Taču tie joprojām saglabā daļēji sensitīvu informāciju, piemēram, vecumu un dzīvesvietas novadu. Ņemot vērā, ka primārais uzdevums bija datus padarīt pieejamus medicīnas jomas zinātniekiem, tad anonimizācijas detaļas sīkāk šeit neaplūkosim.

Mūsu uzdevums bija izveidot ontoloģijas, kas pārklāj šīs slimības, kā arī ontoloģijas savā starpā saistīt, lai būtu iespējams veikt vaicājumus, kā viena slimība spēj ietekmēt kādu citu slimību.

Daļēji mūsu uzdevumu, apvienot ontoloģijas, atviegloja fakts, ka visi transformējamie dati glabājās sistēmā PREDA, ko pārrauga "Slimību profilakses un kontroles centrs". Tādejādi dažādo datu kopu apvienošanas darbs jau bija izpildīts un bija izdalīts centrālais kodols, kas saturēja datus par personām, kamēr pārējos reģistros tika piesaistīti personas dati no šī centrālā kodola.

Ņemot vērā, ka bija pieejamas visas relāciju datubāzu shēmas, kā arī anketas kādā formātā dati tiek ievākti no pacientiem, tad primārais uzdevums bija uzzīmēt ontoloģijas, kas reprezentētu šos datus.

Ņemot vērā, ka teorētiski varējām zīmēt daudz plašākas ontoloģijas, kas sevī iekļautu vairāk informāciju par dažādām slimībām, tomēr šajā gadījumā atzīmējām, ka praktiskos nolūkos to nav nozīmes darīt. Ontoloģijas būs pieejamas medicīnas zinātniekiem, kas veiks datu atlasi, balstoties uz pieejamajām ontoloģijām. Ja datu atlases atsevišķās reizēs atgriezīs tukšu kopu, jo zem uzzīmētajām ontoloģijām nebūs dati, tad tas tikai apgrūtināts atbilstošu vaicājumu sastādīšanu. Tāpēc šajā gadījumā izvēlējāmies zīmēt ontoloģijas, kas nesaturēs klases vai atribūtus, kurām nebūtu nevienas vērtības, taču tajā pašā laikā maksimāli pārklāt datubāzēs pieejamo informāciju.

Lai ontoloģijas attēlotu saprotamas lietotājiem, tad adaptējām UML klašu sintaksi OWL ontoloģiju attēlošanai. Tai laikā populārākais redaktors, Protege, piedāvāja ontoloģiju izveidi dažādos logos, kur vienā tika veidot klašu struktūra, vienā tika veidota relāciju struktūra, kamēr

vēl vienā tika veidoti atribūti. Rezultātā ontoloģija tika iegūta OWL/XML formātā, kas medicīnas zinātniekiem nebija saprotama.[24] Tāpēc papildus ontoloģija tika atsevišķi uzzīmēta Grade rīkā izmantojot UML klašu sintaksi.[25]

Kā jau iepriekš tika ieskicēts rakstā [22], tad izvēlējāmies pārveidot relāciju datus RDF formātā. Kā galvenā priekšrocība bija tehnisko datu atkrišana, kas ir relāciju datu bāzēs un kas medicīnas zinātniekiem nav tik viegli saprotami. Turklāt, atkrītot tehniskiem datiem, lietotājiem ir vieglāk formulēt vaicājumus datu izgūšanai, jo netiek pierakstīta lieka tehniskā informācija.

Kā projekta galvenie uzdevumi tika izvirzīti datu pieejamības izveidošana medicīnas zinātniekiem, kā arī praktiskā ceļā pārbaudīt semantisko datu procesu no relāciju datu bāzēm līdz lietotājiem, kas ar tiem darbosies.

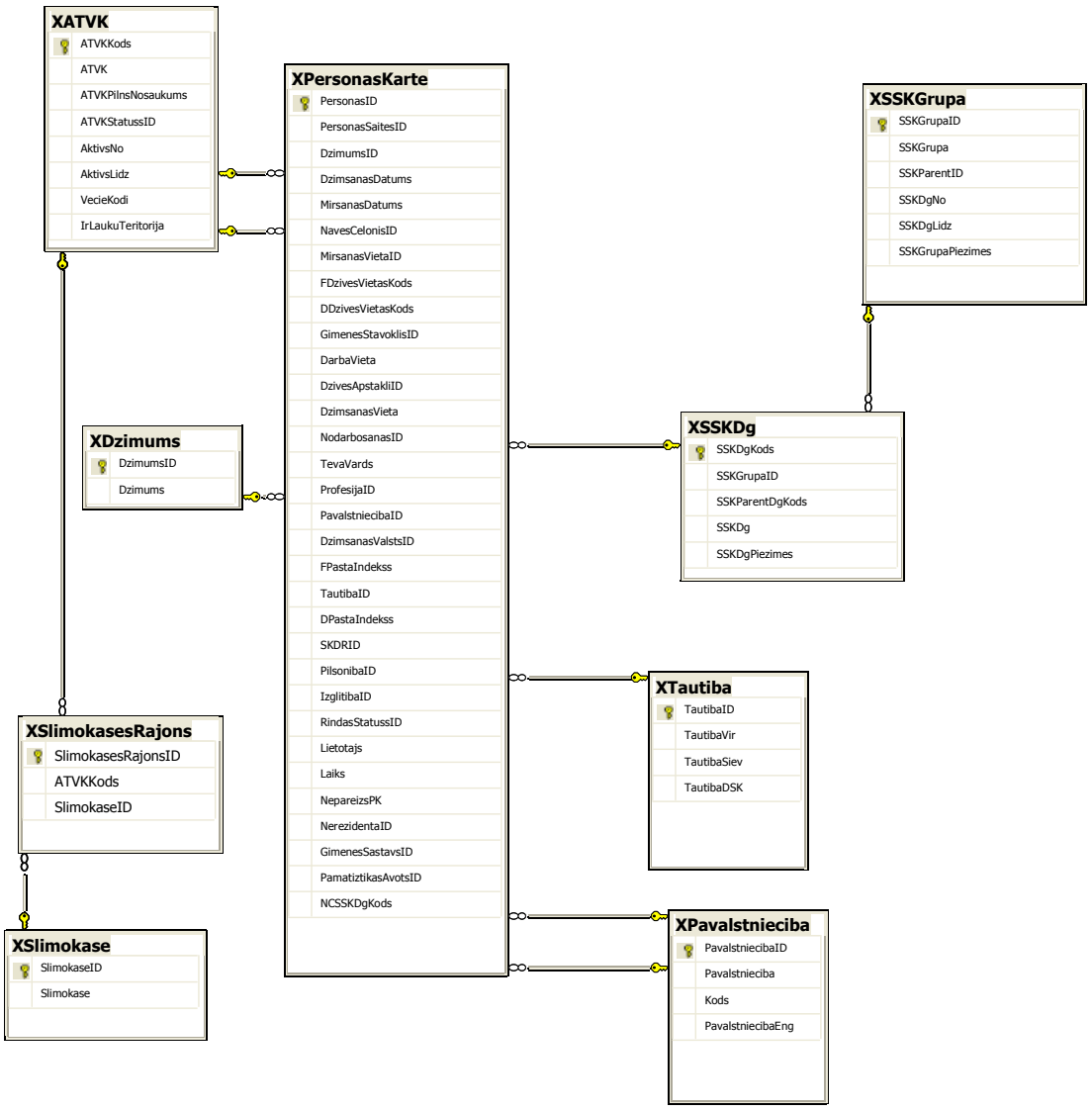
4.3. Medicīnas datu bāzes pārskats

Laikā ap 2008. gadu pavisam bija pieejami vismaz 20 dažādi medicīnas reģistri no 6 dažādiem avotiem (pārsvarā slimnīcām). Taču ņemot vērā, ka mums bija pieejami tikai anonīmi cilvēku dati, tad mums nebija iespējams savilkt dažādo avotu datus atbilstoši cilvēkiem. Kā arī izvēlējāmies koncentrēties uz PREDA datubāzes sistēmu, kas ir "Slimību profilakses un kontroles centra" pārraudzībā[26].

Esošās sistēmas dati tiek glabāti Microsoft SQL datubāzē un tās apjoms ir apmēram 300 000 dažādu personu. Kā arī papildus katram cilvēkam ir piesaistīti viens vai vairāki slimību vai traumas atgadījumi.

Sistēmas centrālā daļa ir attēlota 4.1. attēlā. Tā sevī aptver pacientu personisko informāciju, piemēram, pacienta tautību vai administratīvo teritoriju. Atbilstoši pacienta kartei tālāk tiek piesaistīta informācija katrā no konkrētajiem slimību reģistriem.

[24] Pārliecinājāmies par šo sarunu rezultātā ar medicīnas pētniekiem.
[25] Šobrīd ir izveidots rīks, kas ļauj ne tikai uzzīmēt ontoloģiju struktūru adaptētā UML klašu diagrammu sintaksē, bet arī izgūt uzzīmētās ontoloģijas pierakstu OWL/XML sintaksē – OwlGrEd (owlgred.lumii.lv).
[26] Tai laikā tā bija VSMTVA – "Veselības statistikas un medicīnas traumu valsts aģentūra"



4.1. att. ***PREDA datu sistēmas centrālā daļa.***

Ņemot vērā, ka medicīnas joma ir valsts pārziņā un tā pēc būtības var saturēt sensitīvu informāciju, tad visi datubāzes dati tiek ievākti ar ministru kabinetā apstiprinātu formu palīdzību. Piemērs šādai formai ir aplūkojams 4.2. attēlā, kur ir attēlota daļa no tuberkulozes datu ievākšanas pacienta kartēm.

Kā redzams, tad liela daļa informācijas tiek strikti klasificēta atbilstoši pacienta kartes aizpildīšanai. Turklāt, ir izdotas arī striktas vadlīnijas klasifikatoru aizpildīšanai, kura piemērs ir aplūkojams 4.3. attēlā, kur ir aprakstīts kādos gadījumos vajag klasifikatoru aizpildīt, kā arī kādas ir pieejamās klasifikatoru vērtības.

Tuberkulozes pacienta karte

Ārstniecības iestādes nosaukums _____

Kods ☐☐☐☐-☐☐☐☐☐

Administratīvās teritorijas kods, kurā pacients ņemts reģistra uzskaitē ☐☐☐☐☐☐

1. **Pacienta vārds** _____
2. **Pacienta uzvārds** _____
3. **Pacienta personas kods** ☐☐☐☐☐☐-☐☐☐☐☐
4. **Pacienta dzimums** *(1-vīrietis; 2-sieviete)* ☐
5. **Tautība** _____ ☐☐☐
6. **Deklarētās dzīvesvieta**_____ _____
7. **Deklarētās dzīvesvietas administratīvā teritorija** ☐☐☐☐☐☐
8. **Faktiskās dzīvesvieta** _____
9. **Faktiskās dzīvesvietas administratīvā teritorija** ☐☐☐☐☐☐
10. **Nodarbošanās** _____ ☐☐☐

(ja pacients strādā medicīnas iestādē, norādīt ieņemamo amatu: *221 - farmakologs, 222- ārsts, 223 – galvenā medmāsa,232 - medmāsa, vecmāte, 244 – patronāžas māsa, 322 – feldšeris, 311 – laborants, 513 – sanitārs, 999 - cits)* ☐☐☐

11. **Valsts, kurā pacients dzimis** _____ ☐☐☐
12. **Pacientam nav noteiktas dzīvesvietas** *(apstiprinošas atbildes gadījumā atzīmēt kodu „1")* ☐
13. **Datums, kad pacients iekļauts reģistrā** *(dd.mm.gggg)* ☐☐.☐☐.☐☐☐☐
14. **Novērošanas grupa** _____ ☐☐☐
15. **Diagnozes kods** *(SSK-10)* ☐☐☐.☐ ☐☐☐.☐
16. **Datums, kad diagnoze noteikta pirmo reizi** *(dd.mm.gggg)* ☐☐.☐☐.☐☐☐☐
17. **Situācija, kādā diagnoze atklāta** *(1–vēršoties pie ārstniecības personas pēc palīdzības, 16 - profilaktiskajā apskatē, 17 - autopsijā)* ☐☐
18. **Diagnoze apstiprināta:** ☐
    - 18.1. TM-skopiski *(1- pozitīvs,2- negatīvs, 3- nav izdarīts* ☐
    - 18.2. TM uzsējumā *(1- pozitīvs,2- negatīvs, 3- nav izdarīts)* ☐
    - 18.3. citā veidā *(1- histoloģiski; 2- klīniski rentgenoloģiski;3- citā veidā)* ☐
19. **Riska faktors:** *(atzīmēt kodu 1, ja ir nosauktais riska faktors):*
    - 19.1. pārmērīga alkohola lietošana ☐
    - 19.2. narkotisko vielu lietošana ☐
    - 19.3. ciešs kontakts ar TM + slimnieku ☐
    - 19.4. persona atradusies apcietinājumā vai izcietusi brīvības atņemšanas sodu ieslodzījuma vietā ☐
    - 19.5. cukura diabēts ☐
    - 19.6. glikokortikoīdu lietošana ☐
    - 19.7. HIV tests *(1- pozitīvs, 2 – negatīvs, 3 – nav noteikts, 9 – nav zināms)* ☐
      - 19.7.1. saņem kotrimoksazola ārstēšanu ☐
      - 19.7.2. saņem antiretrovirālo ārstēšanu ☐

4.2. att. ***Tuberkulozes pacienta kartes piemērs.***

VAINĪGĀ DZIMUMS

Nepieciešamais koda garums:
n

Definīcija:
Tās personas dzimums, kura ir radījusi ievainojumu.

Konteksts:
Šis datu elements sniedz papildus informāciju par personu, kuras vardarbīga rīcība ir izraisījusi ievainojumu.

Lietošanas vadlīnijas:
Šis datu elements ir jākodē visos uzbrukuma rezultātā gūtu ievainojumu gadījumos.

VAINĪGĀ DZIMUMS: PILNS KODU SARAKSTS

1 Vīrietis

2 Sieviete

9 Nav zināms

4.3. att. ***Vainīgā dzimuma klasifikatora lietošanas paskaidrojums.***

Tātad mums ir trīs galvenie objekti, ko izmantot, lai varētu izveidot saprotamas ontoloģijas, uz kuru pamata vēlāk medicīnas zinātnieki varēs veikt datu atlasi. Tās ir, relācijas datubāzes shēma, anketu formas, kas parāda kādi dati tiek glabāti, kā arī izmantotie klasifikatori.

Ņemot vērā, ka relāciju datubāzes datu shēma (ER modelis) ir mūsu datu izejas kopa, tad tas kalpo kā primārais pamats. Atbilstoši ER modelī esošajām shēmām veidojam ontoloģiju klases, kā arī izmantojam ārējās atslēgas, lai veidotu sasaisti starp klasēm.

Galvenais uzdevums bija saprast un izanalizēt relācijas datu shēmā esošo jēdzienus, jo daži no tiem bija viegli uztverami un saprotami, kamēr citi radīja problēmas, piemēram, attēlā 4.1. esošā tabula "XSSKDg", kas satur, kā "XSSKDgKods" tā arī vienkārši lauku "XSSKDg". Vai, piemēram, tabulā "PacientaKarte" esošs lauks "Lietotājs".

Kā redzams, tad relāciju datu bāze satur ne tikai anketā esošos datus, bet tajā tiek pievienota arī "biznesa loģikas" informācija. Šāda informācija nav saistoša medicīnas zinātniekiem un tikai rada apjukumu, ja būtu jāstrādā ar relācijas datu bāzi neizmantojot semantisku starpslāni, kas

likvidē nevajadzīgo tehnisko informāciju, kā arī nerāda informācijas laukus, kas attiecas uz "biznesa loģiku".

4.4. Semantiskā līmeņa izveide

Pamatā izvēlējāmies lietotājiem saprotamu datu attēlošanas formātu, kas, mūsuprāt, ir grafiska diagramma līdzīgai datu bāzu ER shēmām vai UML klašu diagrammām. Izmantojām UML klašu diagrammas OWL profilu, kas formāli ir sintaktisks pieraksta formāts W3C izstrādātajam semantiskā tīmekļa ontoloģiju valodai OWL.

Kā galveno šī pieraksta formāta īpašība ir jāizceļ, ka tas apvieno divas būtiskas lietas. Pirmkārt, tas ļauj formāli korekti pierakstīt OWL ontoloģijas, kas ir semantiskā tīmekļa pamatā, kā arī viens no mūsu projekta stūrakmeņiem. Otrkārt, tā ir zināma un praksē pārbaudīta tehnoloģija, kas tiek plaši lietota, izstrādājot informatīvas sistēmas.

Ja būtu pieejam tikai relāciju datu bāzes shēma, tad tas ļoti apgrūtinātu labas un saprotamas ontoloģijas izveidi. Tas prasītu konkrētu datu lauku vērtību analīzi un iespēju robežās medicīnas zinātnieku piesaisti, kas varētu palīdzēt ar šo datu interpretāciju. Taču mums papildus ir pieejamas datu uzkrāšanas formas, kuras piemērs ir parādīts 4.2. attēlā.

Datu uzkrāšanas formas parāda mums vismaz daļu no pieejamajiem datiem. Kā arī tās papildus atzīmē datu formātu, kā arī izdala dažādus datu tipus. Datu uzkrāšanas formas šajā gadījumā atviegloja semantiskā līmeņa izveidi, jo tās jau semantiski aprakstīja datubāzē pieejamos datus.

Ņemot vērā šo datu uzkrāšanas formu nepieciešamību relāciju datu bāzu izprašanā, tas papildus norāda uz pārāk tehniskajām datu bāzu shēmām, kas gala lietotājiem nav draudzīgas. Ja katram no lietotājiem, kas gribētu sastādīt kādu vienkāršu vaicājumu pār šiem datiem, būtu papildus jāstudē daudzās un dažādās datu uzkrāšanas formas, tad visdrīzāk mēs sagaidītu rezultātu, ka šis darbs visdrīzāk tiktu uzticēts programmētājam un krietni apgrūtinātu medicīnas zinātnieku darbu, jo būtu grūti ātri un vienkārši piekļūt datiem.

Ja pretstatā šīm, ne tik labi saprotamajām relāciju datu bāzu shēmām, domājam par UML stila grafveida pieeju datu attēlošanā, kas nesatur ne tehnisko informāciju, ne "biznesa loģikas" detaļas, tad arī izprotam, ka tas lietotājiem varētu atvieglot piekļuvi datiem "pa tiešo". Mēs piedāvāsim izmantot vaicāšanas rīku balstītu uz šīm UML stila grafveida attēlojumu. Tādā veidā gala lietotāji varēs sastādīt vaicājumus par to, ko viņi patiešām redz, nevis domāt kura no attēlotās informācijas ir nepieciešama, bet kurai nav būtiska pielietojuma tālākā izpētē.

Ideālā gadījumā mēs veidotu ontoloģiju, kas atbilst medicīnas zinātnieku priekšstatam par katru no apgabaliem, pār kuriem viņi vēlas veikt datu vaicājumus. Taču šajā gadījumā mūs ierobežo pieejamo datu daudzums. Kā jau tika minēts, tad nav nozīmes izveidot ontoloģijas daļu, kura neatspoguļo reālus datus.

No šādu skatupunkta mums ir uzlikts ierobežojums, lai varētu brīvi veidot semantisku līmeni, jo nevaram papildināt ontoloģijas līmeni ar trūkstošām lietām, bet tikai attēlot jau eksistējošos datus.

Taču tajā pašā laikā ir pietiekami daudz iespējas uzlabot jau esošo relāciju datu bāzē esošo datu saprotamību. Pirmkārt, izvēlēties un saskaņot datu lauku nosaukumus ar medicīnas zinātnieku pārstāvjiem, lai nebūtu lieki jādomā par datu nozīmi. Otrkārt, nodalīt klasifikācijas datus no brīvi ievadāmajiem.

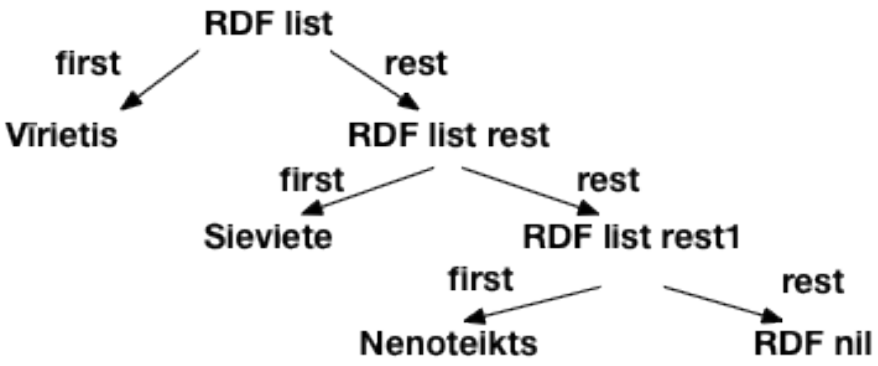
Otrais punkts ir īpaši svarīgs, jo tas ļauj ne tikai semantiski saprast kādi dati ir pieejami datu bāzē, bet arī atzīmēt kādus datus un kādā veidā ir iespējams izgūt no datu bāzes. Piemēram, relāciju datu bāzē ir tabula "XDzimums", kas atspoguļo personas dzimumu. Ja medicīnas speciālists apskata šādu datu bāzi un vēlas izgūt statistiku par kādiem dzimumiem, tad viņam vispirms būtu jāveic vaicājums, lai uzzinātu – kādi vispār dzimumi ir pieejami. Piemēram, var būt vīrietis, sieviete, nezināms, transvestīts vai kāda papildus klasifikācija.

Šo klasifikāciju ir noteikusi kāda "biznesa loģika", kas relāciju datubāzes shēma tiek paslēpta, kā arī netiek papildus izklāstītas pieejamās klasifikācijas vērtības. Veidojot sistēmu, kas darbojas ar šo relāciju datu bāzi, tiek iekļautas pārbaudes, kas ziņo, ja klasifikatoru sistēma tiek pārkāpta vai atbilstoši tiek izveidotas sistēmas daļas, kas asistē lietotājiem, lai ievadītu sistēmā dotā klasifikatoru vērtības.

Taču gadījumā, ja ir nepieciešama pašu datu analīze, tad parasti vairs nav pieeja specificēta sistēma, kas palīdzēs norādīt uz datu klasifikatoriem. Ja aplūkojam 4.1. attēlu, tad varam redzēt, ka nav izdalītas nekādas pazīmes, pēc kurām varētu pateikt, kuras relāciju tabulas satur brīva formāta datus, bet kurās ir ievadīti klasifikatori, kas nav maināmi.

Lai arī jāatzīst, ka īsti korekti šī problēma nav atrisināta arī OWL valodā, jo nav izdalītas speciālas klases, kas būtu paredzētas tieši klasifikatoru pielietošanai. Taču tajā pašā laikā ir pieejamas dažādas loģikas iespējas, ko var izmantot, lai aprakstītu šādas klasifikatoru klases, piemēram, OWL valodā ir pieejama iespēja uzrakstīt, ka kāda klase ir ekvivalenta, tai skaitā tur ir iespējams minēt visu instanču pārskaitījumu.

Otra iespēja OWL valodā izteikt klasifikatorus ir – izmantojot RDF sarakstus. Atbilstoši visas instances ir iespējams sagrupēt RDF sarakstā, kas veidojas pirmā elementa un pārējiem elementiem, jo saraksts ir jāpieraksta ar RDF trijniekiem. Varam aplūkot piemēru 4.4. attēlā.



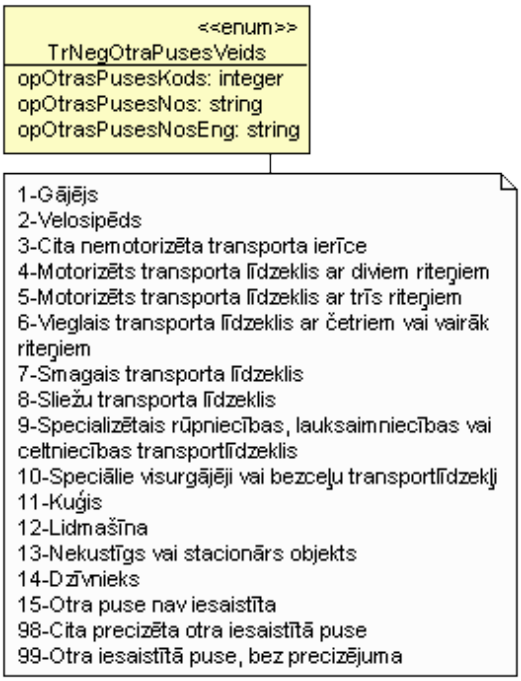
4.4 att. ***RDF saraksta piemērs dzimuma pārskaitījumam.***

Izmantojot RDF sarakstus var definēt atribūtus, kuru vērtību apgabals ir tieši konkrētais saraksts, nevis vispārīgs atribūta tipa apzīmējums, piemēram, *string* vai *integer*.

Izsverot abu risinājumu plusus un mīnusus tika izvēlēts risinājums lietot ekvivalentas klases. Kā galvenais arguments šim lietojuma bija fakts, ka daudziem klasifikatoriem vienlaicīgi ir vairākas atribūtu vērtības, piemēram, nosaukums latviski, nosaukums angliski un klasifikatora skaitliskā vērtība. Ja to vēlētos realizēt ar sarakstu palīdzību, tad tas būtu saraksts no saliktiem atribūtiem, kas vairs nebūtu saprotami medicīnas zinātniekiem, jo notiktu līdzīga problēma relāciju datubāzēm, ka pa vidu parādās liekas vai nesaprotamas detaļas. Piemēru ar pārskaitāmā datu tipa attēlojumu ontoloģijā var aplūkot 4.5. attēlā, kur ir apskatāms pārskaitāmā tips transporta negadījumā iesaistītajām personām.

Dzimuma klasifikatora gadījumā, viens atribūts pēkšņi pārtop par trīs dažādām vērtībām. Ņemot vērā, ka dažādās situācijās ir vēlme veikt atlasi pēc vienas no trīs vērtībām, tad ir grūti šādu situāciju vispārīgi modelēt, jo parastu atribūtu gadījumā ir viena vērtība, kamēr klasifikatoru gadījumos var būt vairākas vērtības vienlaicīgi.

Taču izmantojot ekvivalentas klases ir iespējams veidot objektus, kuriem vienlaicīgi ir patvaļīgi daudz atribūtu vērtības un ir iespējams uzstādīt nosacījumu jebkurai atribūta vērtībai.



4.5. att. ***Pārskaitāmā datu tipa attēls ontoloģijā.***

Taču ņemot vērā citus OWL valodas lietojumus, ir jāuzstāda papildus īpašības ekvivalentai klasei, lai šīs ekvivalentās klases klasifikatorus varētu atšķirt no OWL secināšanā lietotajām OWL ekvivalentajām klasēm, kuras mērķis ir veikt datu secināšanu un aizpildīt ekvivalentās klases ar secināšanas rezultātiem no citām klasēm.

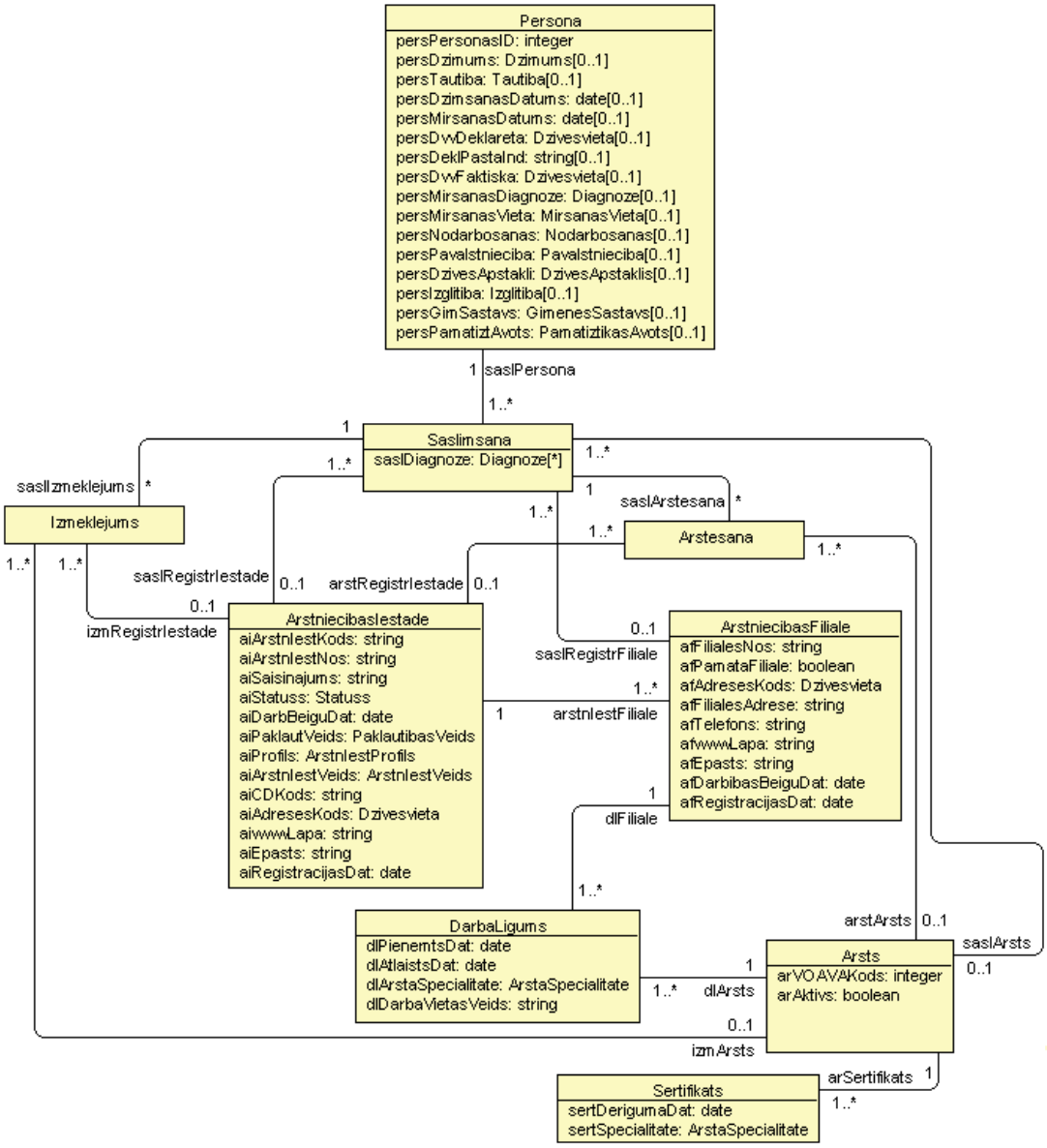
Šajā gadījumā gan ir jāmin atkāpe, ka OWL valoda pēc būtības darbojas atvērtās pasaules semantikas, kas nozīmē, ja klasei ir minēts, ka tā satur vismaz vienu instanci, bet šādas instances datos nav pieejamas, tad tiks izveidota anonīma instance, kas piederēs dotai klasei. Kamēr relāciju datubāzes un arī mūsu pieeja darbojas slēgtās pasaules principiem, ka ir tikai tie dati, kas ir izveidoti un ko mēs redzam. Datu papildināšana ar anonīmām instancēm netiek veikta.

Taču noslēpti relāciju datubāzē ir ne tikai datu klasifikatori, bet arī īpašību klasifikācija. Piemēram, traumas var būt ar vainīgo, tas var būt transporta negadījums, kā arī traumai var būt hospitalizācija. Katrā no šiem gadījumiem ir atšķirīgas atribūtu kopas, kuru vērtības tiek aizpildītas.

Relāciju datu bāzē tiek lietoti vienkārši atribūtu apzīmētāji, kas klasificē vai papildus atribūti šai īpašībai ir aizpildīti vai nē. Taču, mūsuprāt, tas nav pietiekami uzskatāmi, kā arī apgrūtina izpratni par dažādajiem gadījumiem.

Tāpēc mēs papildus izmantojam ontoloģijās (arī UML klašu diagrammās) pieejamo iespēju klases iedalīt specifiskās apakšklasēs. Ņemot vērā, ka viens objekts var piederēt vairākām apakšklasēm, tad tas neizslēdz iespēju veikt vaicājumus pār apakšklašu apvienojumu.

Lai arī ir pieejami rīki pus automatizētai pārejai no relāciju datubāzēm uz ontoloģijām [23, 24], tomēr ņemot vērā iepriekš uzskaitītās problēmas, piemēram, klasifikatorus vai relācijas datubāzu klašu nosaukumus, labāk ir izvēlēties veidot ontoloģiju no nulles. Tas ļaus labāk izprast esošos datus un izveidot tīrāku un medicīnas zinātniekiem saprotamāku ontoloģiju, kas ir galvenais darba mērķis.



4.6. att. ***Medicīnas sistēmas kodola ontoloģija***

Arī zīmējot ontoloģiju no nulles ir ne tikai iespējams izvēlēties atbilstošus nosaukumus, bet arī veidot ontoloģijas izkārtojumu, lai tas būtu ērti uztverams. Ņemot vērā, ka ontoloģijas attēls būs viens no aspektiem, kas palīdzēs medicīnas zinātniekiem sastādīt vaicājumus, tad šī attēla pārskatāmība būs svarīgs aspekts arī izpratnē par ontoloģijā esošajiem un pieejamajiem datiem.

Attēlos 4.6. un 4.7. ir iespējams aplūkot divas no izveidotajām ontoloģijām. Pilns, izveidoto ontoloģiju pārskats, pieejams pielikumā 3. Šīs ontoloģijas arī parāda, kādi relāciju datubāzu lauki un to vērtības iespaido piederību pie vienas vai otras apakšklases. 4.6. attēlā ir parādīta sistēmas centrālā ontoloģija par personām, kas to apvieno. Bet 4.7. attēlā ir parādīta viena no specifiskajām medicīnas datu ontoloģijām, šajā gadījumā tā ir traumu ontoloģija.

Varam pievērst uzmanību izvēlētajiem atribūtu nosaukumiem, kas daļēji ir saistīti ar ontoloģijās esošajiem ierobežojumiem, kas nosaka, ka atribūts var tikt piesaistīts tikai vienai klasei. Ja tas ir piesaistīts vairākām klasēm, tad tas ir piesaistīts šo klašu apvienojumam.

Galvenokārt šī atšķirība no relāciju datubāzēm ir tāpēc, ka ontoloģijā vārdu lietošana ir globāla, kamēr relāciju datu bāzēs tā ir neatkarīga vienas tabulas ietvaros. Taču tas ir saistīts arī ar pašu datu glabāšanu, jo datu bāzēs tie tiek glabāti relācijas tabulas ietvaros un visa datu bāze tiek strukturēta pa tabulām, kamēr semantiskajās datu bāzēs viss tiek attēlots ar trijniekiem, kas sastāv no subjekta, relācijas un objekta. Tā kā visi trijnieki glabājas vienuviet un netiek papildus sadalīti pa kādām citām struktūrām, tas arī uzspiež lietot globāli unikālus vārdus.

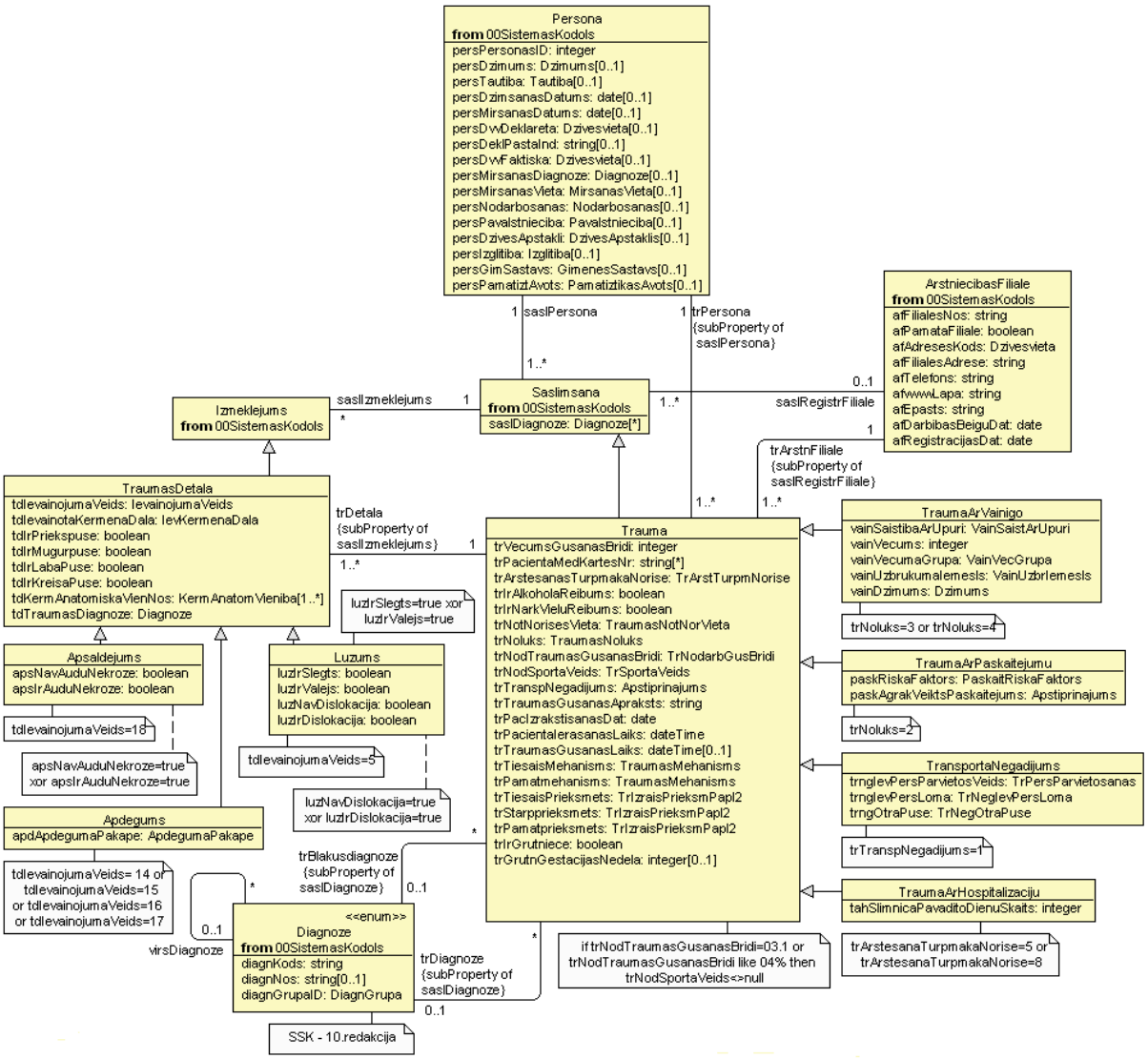
Piemēram, traumu ontoloģijas gadījumā daudzos klasifikatoros tiek lietots kods. Tas ir unikāls katras klases ietvarā taču jāņem vērā augstāk esošā īpašība un jāpievieno papildinājums koda nosaukumam, lai to padarītu unikālu arī ontoloģijā.

Klasifikatoru gadījumā šo situāciju būtu iespējams risināt ar abstraktas virsklases ieviešanu, kas saturētu atribūtu kods un tādā veidā nevajadzētu ieviest papildus piedēkli atribūta nosaukumiem.

Papildus šo problēmu varētu risināt ar vārda telpas palīdzību, ka tiktu lietotas dažādas vārda telpas dažādām klasēm, bet tas būtu pretrunā ar labās prakses principiem, ka vienas ontoloģijas ietvaros tiek lietota viena vārda telpa.

Kā papildus problēma abiem gadījumiem ir jāmin arī atbilstošu rīku trūkums, kuri spētu nodrošināt šādu risinājumu ērtu īstenošanu. Ieviešot abstraktu klasi, tā parādās arī ontoloģijas attēlā un pārveido attēlu, jo projekta realizācijas laikā nebija iespējams attēlot virsklases un apakšklases attiecību tekstuāli[27], bet tā bija jāattēlo kā grafiska līnija.

[27] Piemēram šobrīd grafveida ontoloģijas redaktors OwlGrEd piedāvā alternatīvu, ka apakšklašu attiecības var tikt attēlotas kā līnijas, tā arī tekstuāli apgalvojumi.



4.7. att. ***Traumu ontoloģija.***

Ja esošajā gadījumā klasifikatori grafiskajā attēlojumā ir klasificējami kā lapas, tad ieviešot papildus saiti uz abstraktu virsklasi tie vairs nebūtu šādi klasificējami. No prakses varam minēt, ka grafa izvietotājos ir vienkāršāk īstenot saprotamu grafa lapu izkārtojumu nekā saistīta grafa izkārtojumu.

Vārda telpu gadījumā pieejamie rīki vadās no labās prakses un ontoloģiju ļauj zīmēt vienā vārda telpā. Ja tiek lietota cita vārda telpa, tad tās lietojums tiek uzrādīts grafiskā attēlā un tāpat satur lietotājiem lieku un nesaprotamu informāciju.

Kā arī pievienojot visiem atribūtiem kā piedēkli saīsinātu klases vārdu, ieturam konsekvenci, kas bieži ir lietotājiem būtiska. Lai arī ar šobrīd pieejamo rīku nodrošinājumu būtu

iespējams ontoloģiju realizēt lietotājiem saprotamā veidā, neizmantojot piedēkļus, kas vēlāk apgrūtina vaicājumu sastādīšanu, jo piedēkļi apgrūtina orientēšanos atribūtu sarakstos.

Sagatavojot relāciju datu bāzei atbilstošas ontoloģijas, esam gatavi veikt nākamo soli, lai translētu datus no relāciju datubāzu formāta uz semantisko datu formātu, RDF.

4.5. Datu translācijas process

Veicot datu translācijas procesu un vairāku ontoloģiju apvienošanu vispirms ir jāatrisina – kā apvienot loģiski vienādos objektus, kas datu bāzē kaut kādu iemeslu dēļ ir attēloti vairākās vietās, vai arī vienkārši ir attēloti vairākās datu bāzēs. Vienādos objektus, apvienojot vienā datu glabātavā, arī ir jāapvieno par vienotu objektu.

Piemēram, apvienojot vairākas datu bāzes, kurās ir klase ar personām, tad ir ļoti liela varbūtība, ka viena un tā pati persona būs ierakstīta vairākās datu bāzēs. Pēc datu bāžu apvienošanas šī persona arī parādīsies vairākas reizes datos, taču mēs vēlamies, lai būtu iespējams identificēt, ka tā ir viena un tā pati persona, kas atvieglotu vaicājumu sastādīšanu.

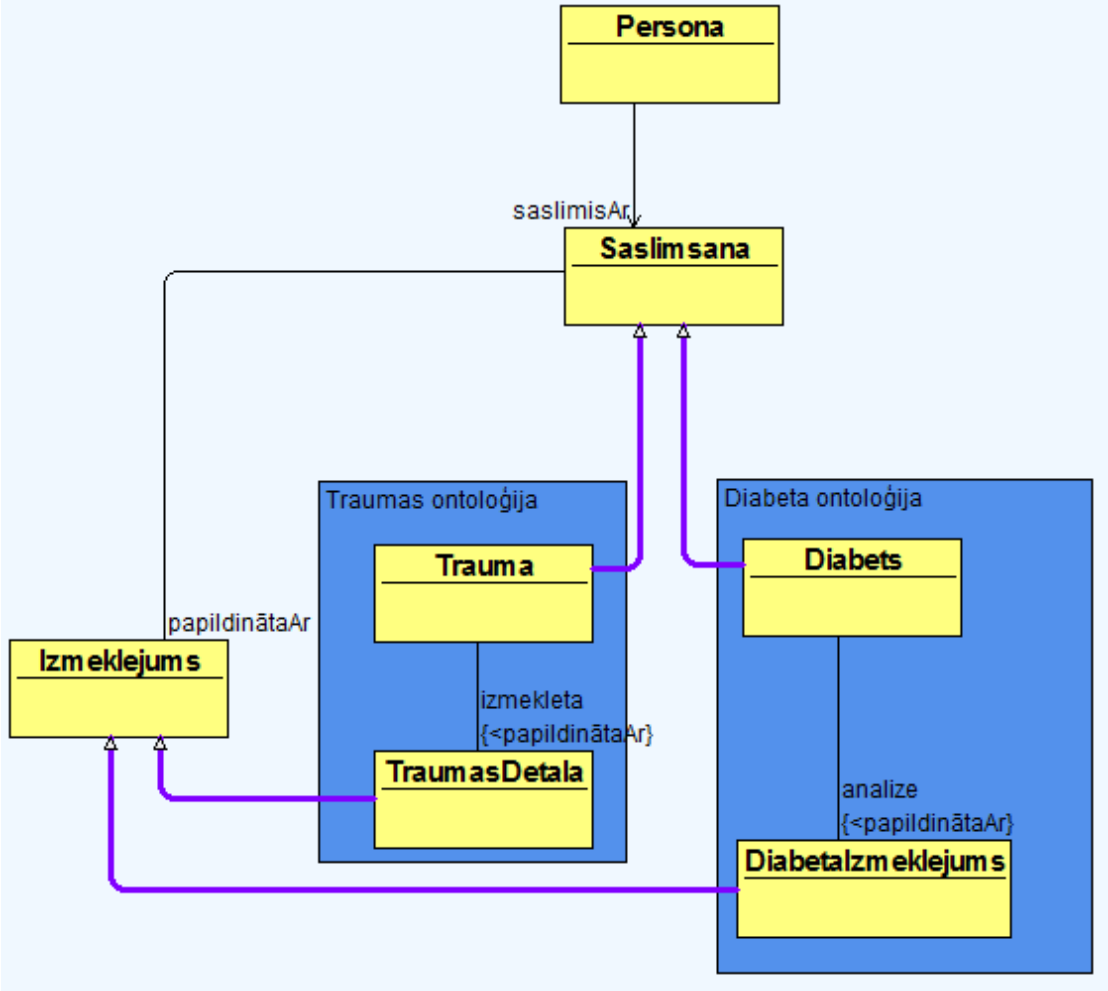
OWL ontoloģijās[28] ir iestrādāts risinājums šādiem gadījumiem, jo ir iespējams lietot "owl:sameAs" asociāciju starp vienādajiem indivīdiem. Taču šajā gadījumā tāpat ir jāveido manuāli vai automātiski šīs asociācijas.

Pirmkārt, var rakstīt SPARQL UPDATE vaicājumus, kas sameklē noteiktu paternus un izveido šādas "owl:sameAs" saites atrastajos paternos. Dotā pieeja prasa labu datubāzu satura pārzināšana, kā arī atbilstošu likumu izveidošanu, lai varētu noteikt, kurās vietās ir jāveido "owl:sameAs" atsauces.

Otrkārt, ir iespējams lietot kādu no rīkiem, kas veic automātisku vienādo objektu meklēšanu, kā arī izveido nepieciešamās "owl:sameAs" atsauces. Piemēram, Sesame datubāzes spraudnis Elmo-smusher[25].

Taču mūsu medicīnas datu piemērā šī problēma jau bija atrisināta PREDA informācijas sistēmas izveides rezultātā. Sistēmas centrā bija izveidots sistēmas kodols, kas satur datus kopīgos datus par personām, slimnīcām, diagnozēm. Sistēmas kodols ir apskatāms 4.6. attēlā. Atbilstoši katras slimības dati ir piesaistīti sistēmas kodolam, ko varam apskatīt arī traumas ontoloģijas piemērā 4.7. attēlā, kur personas klase, ārstniecības filiāle un saslimšana ar diagnozi ir izvilkta no sistēmas kodola. Shematiski šī saistība ir apskatāma 4.8. attēlā.

[28] OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref/



4.8. att. ***PREDA ontoloģiju sasaiste caur sistēmas kodolu.***

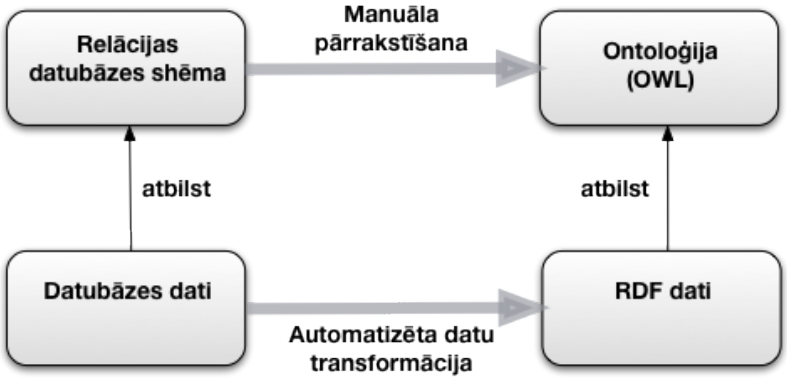
Līdz ar to mums nebija jāsaskaras ar dotās problēmas risināšanu. To jau bija atrisinājuši pirms mums. Taču pielikt klāt vēl vienu ontoloģiju no citas datu bāzes radītu papildus problēmas ar šo sasaisti, jo personas tiek identificētas pēc personas koda, bet PREDA gadījumā personas ir sasaistītas kopā, taču personas dati ir anonīmi un ir neiespējami piesaistīt tiem klāt vēl papildus vienādo objektu identifikāciju, jo personas kods nav zināms.

Risinot sasaisti ar sensitīviem datiem būtu nepieciešams šo sasaisti veikt jau datubāzes līmenī, pirms dati tiek eksportēti, vai arī pielietot vienādu anonimizācijas metodi, kas rēķinātu *hash* vērtību no personas koda, lai arī tas ietvertu papildus riskus personas datu drošībai.

Līdz ar to esam nonākuši pie svarīgākā uzdevuma, kura būtība bija pārveidot relāciju datubāzē esošos datus pārveidot par semantiskiem datiem, kas būtu pieejami RDF formātā.

Vispārīga shematika, kā ir iespējams pārveidot relāciju datu bāzes datus par semantiskiem datiem, ir attēlota 4.9. attēlā. Jau iepriekš aprakstījām, kā ir iespējams manuāli pārrakstīt relāciju

datubāzes shēmu par OWL ontoloģiju. Turpinājumā aprakstīsim kādām metodēm ir iespējams transformēt datu instances.



4.9. att. ***Datu transformācijas shematisks attēlojums.***

Transformāciju no relāciju datubāzes datiem uz RDF trijniekiem ir iespējams realizēt ļoti dažādos veidos. Svarīgi rezultātā ir iegūt RDF trijniekus, piemēram, XML formātā, kā tie ir attēloti 4.10. attēlā. Tālāk šos trijniekus būs iespējams ielasīt SPARQL piekļuves punktā un darboties ar tiem izmantojot SPARQL vaicājumu valodu.

4.10 attēlā dotajā piemērā varam aplūkot arī uzsvērto īpašību, ka viens objekts var piederēt vairākām klasēm, kamēr relāciju datubāzēs objekts pieder tieši vienai tabulai.[29] Piemērā ir attēlota Trauma_2, kas vienlaicīgi ir transporta negadījums un trauma ar lūzumu, kas attēlo situāciju, ka transporta negadījumā ir gūts lūzums.

```
<http://lumii.lv/preda#Trauma_2> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://lumii.lv/preda#BrokenBone> .
<http://lumii.lv/preda#Trauma_2> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://lumii.lv/preda#TrafficAccident>.
<http://lumii.lv/preda#VezaSaslimsana_1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://lumii.lv/preda#Cancer> .
<http://lumii.lv/preda#Persona_140> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://lumii.lv/preda#Person> .
<http://lumii.lv/preda#Persona_141> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://lumii.lv/preda#Person> .
<http://lumii.lv/preda#Persona_148> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://lumii.lv/preda#Person> .
<http://lumii.lv/preda#VezaSaslimsana_1> <http://lumii.lv/preda#cancPerson> <http://lumii.lv/preda#Persona_148> .
<http://lumii.lv/preda#VezaSaslimsana_1> <http://lumii.lv/preda#cancStage> "bad"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://lumii.lv/preda#Trauma_2> <http://lumii.lv/preda#injPerson> <http://lumii.lv/preda#Persona_148> .
<http://lumii.lv/preda#Persona_148> <http://lumii.lv/preda#pcPhone> "+37125252525"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://lumii.lv/preda#Persona_148> <http://lumii.lv/preda#persID> "1111111111"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://lumii.lv/preda#Persona_140> <http://lumii.lv/preda#persID> "2222222222"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://lumii.lv/preda#Persona_141> <http://lumii.lv/preda#persID> "333333333"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://lumii.lv/preda#Persona_140> <http://lumii.lv/preda#persName> "Anna"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://lumii.lv/preda#Persona_141> <http://lumii.lv/preda#persName> "Jurs"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://lumii.lv/preda#Persona_148> <http://lumii.lv/preda#persName> "Zigmunds"^^<http://www.w3.org/2001/XMLSchema#string> .
```

4.10. att. ***RDF trijnieku XML serializācija***

Projekta realizācija tika īstenota izmantojot SQL procedūras, kas atlasīja datus relāciju datubāzē, bet atbilstošos rezultātus ierakstīja kā RDF trijniekus XML formātā datnēs. Tas galvenokārt tika

[29] Formāli ņemot ontoloģiju pasaulē ir objekti, kas nepieder nevienai klasei, kas gan tiek definēts, ka tie pieder owl:Thing klasei, kas satur pilnīgi visus ontoloģijas objektus.

izvēlēts tāpēc, ka tobrīd nebija pieejami alternatīvi rīki, kas būtu ērti lietojami, lai to izdarītu labāk.

Manuāla SQL procedūru rakstīšanas galvenās problēmas bija, ka tās nebija pietiekami pārskatāmas, kā arī bija iespējams pieļaut kļūdas. Piemēram, netransformējot visus datus vai kaut ko transformējot vairākas reizes, bet neievedot savstarpēju attiecību, ka tie atsaucas uz vienu objektu. 4.11. attēlā var aplūkot Martas Veilandes rakstītu SQL procedūru, kas izveido atbilstošus RDF trijniekus apdeguma pakāpes atribūtam. Atbilstoši katrai traumas detaļas instancei, kas ir apdegums, tiek izveidots atribūts apdeguma pakāpe ar atbilstošu apdeguma pakāpes vērtību.

```
SELECT
'<http://www.owl-ontologies.com/Ontology1228576854.owl#'
+d.[TraumasDetala_inst]+'>
<http://www.owl-ontologies.com/Ontology1228576854.owl#apdApdegumaPakape> <http://www.owl-
ontologies.com/Ontology1228576854.owl#'
+a.[ApdegumaPakape_inst]+'> .'
FROM [Traumas].[dbo].[IDBIevainojumsDetalas] d,[Traumas].[dbo].[IDBKlasifikators] a
where d.[IevainojumaVeidsID] in (14,15,16,17) and d.[SmagumaPakapeKods] is not null
and d.[SmagumaPakapeKods]<>'0' and d.[SmagumaPakapeKods]=a.[KlasifikatorsID]
```

4.11. att. ***SQL transformācijas piemērs uz RDF.***

Mūsdienās jau ir izstrādāti dažādas metodes, kas piedāvā veikt transformāciju no relāciju datubāzu datiem uz RDF formāta datiem ērtākā veidā. Kā arī ir pieejama variācija, vai tiek veikta transformācija tiešsaistē, tas ir, dati tiek glabāti relāciju datubāzē, bet tiek nodrošināta piekļuve, kas atbalsta SPARQL vaicājumus, kas tālāk tiek translēti par SQL vaicājumiem.

Tiešsaistes transformācijām tiek lietotas *mapping* valodas, kas attēlo sasaisti starp relāciju datubāzes datiem un izvēlēto ontoloģiju. Populārākie pieejamie risinājumi ir D2RQ [26], Virtuoso RDF views [27] vai RDB2OWL [28].

Risinājumu galvenās priekšrocības nodrošina to, ka var līdz pastāvēt relāciju datubāze, uz kuras pamatu darbojas esošās informatīvās sistēmas, kas to papildina un maina datus, kā arī SPARQL pieeja šiem datiem. Turklāt, būtiskā priekšrocība šādā gadījumā ir tas, ka SPARQL vienmēr piekļūst aktuālajiem datiem, kamēr veidojot datu transformāciju uz RDF XML sintaksi un ielādi SPARQL piekļuves punktā nodrošina pieeju datu momentuzņēmumam noteiktā laikā. Turklāt, lai aktualizētu datus, ik pa laikam ir jāatjauno datu momentuzņēmums.

RDB2OWL gadījumā jāmin arī tas, ka ir iespējams pārskatāmi izveidot no kurienes mērķa ontoloģija iegūs nepieciešamos datus. Tādā veidā mums ir pārskatāms, ka visas klases un atribūti jaunizveidotajā ontoloģijā iegūs reālus datus no relāciju datubāzes, kas ir liela priekšrocība, ja ontoloģija ir liela un grūti pārskatāma.

Ja aplūkojam ātrdarbības analīzi [29], tad varam redzēt, ka translējot datus kā momentuzņēmumu, iegūstam ātrdarbības priekšrocības vaicājumu izpildei salīdzinot ar tiešsaistes transformācijām. Ņemot vērā, ka mūsu gadījumā svarīgāka bija katra vaicājuma ātrdarbība, nevis pieeja tiešsaistes datiem, tad izvēlējāmies risinājumu, kas izveido RDF datu momentuzņēmumu. Turklāt, papildus ierobežojums mūsu gadījumā bija tas, ka mums nebija tiešsaistes piekļuve relāciju datubāzei, bet saņēmām tikai relāciju datubāzes momentuzņēmumus.

Kā alternatīvs risinājums RDF datu momentuzņēmuma veidošanai no relāciju datubāzes datiem, var minēt modeļu transformāciju pielietojumu [30]. Kā galveno priekšrocību šāda risinājuma izvēlē varam minēt to, ka transformācijas var veidot grafiski, kas ļauj gūt labāku priekšstatu pār izmantotajām transformācijām.

Noslēdzošais posms ir datu ielāde SPARQL piekļuves punktā. Projekta sākumā izmantojām Sesame [31] repozitoriju, taču tā ātrdarbība kļuva neapmierinoša, kad to papildinājām ar vairāku ontoloģiju datiem. Projekta turpinājumā tika izmantots Virtuoso piekļuves punkts[30], kas nodrošināja labāku ātrdarbību SPARQL vaicājumiem.

Ņemot vērā, ka ontoloģijas konstruēšanā izmantojām apakšklases (owl:subClass), tad, lai SPARQL vaicājumi darbotos korekti, ir nepieciešams arī definēt secināšanas likumus SPARQL piekļuves punktam. Plašāk par secināšanas likumiem un to pielietošanu vaicājumu atbildes iegūšanā var uzzināt [32].

Mūsu izvēlētajā gadījumā, ja instance tiek ievadīta klasei – "TraumaArNegadījumu", tad meklējot klases "Trauma" instances tā netiks atrasta, ja nav ieslēgta *entailment rules*, kas nodrošina, ka apakšklases instances tiek rādītas pie virsklasēm, kā arī tas darbojas gadījumos, ja ir definēta "rdfs:subPropertyOf", kas nosaka, ka viena relācija ietver sevī kādu citu relāciju, piemēram, 4.7. attēlā parādītajā Traumas ontoloģijā ir attēlota šāda attiecība, kad "saslIzmeklējums" relācija sevī ietver arī apakšrelāciju "trDetala".

Ontoloģijā varētu tikt lietoti arī OWL DL[31] nosacījumu pieraksts. Taču tādā gadījumā būtu nepieciešams lietot pilnīgāku secinātāju, kā nodrošina SPARQL piekļuves punkti. Piemēram, Pellet [33] secinātāju vai arī SPARQL piekļuves punktu StarDog[32], kas daļēji atbalsta šo nosacījumu pārbaudi.

Taču jāatzīmē, ka mūsu gadījumā mēs paļaujamies, ka nosacījumu pārbaude jau tiek veikta relāciju datubāzē un to nodrošina "biznesa loģika". Turklāt, atrodot kļūdas datos, mums

[30] http://virtuoso.openlinksw.com/
[31] http://www.w3.org/TR/2004/REC-owl-guide-20040210/#OwlVarieties
[32] http://stardog.com/

nav iespējas tās labot, jo iegūstam tikai pieeju relāciju datubāzu momentuzņēmumiem, bet ne izejas datiem, kas nozīmētu, ka viena un tā pati kļūda būtu jālabo katru reizi, kad saņemam jaunu momentuzņēmumu.

Papildus apraksts par datu transformāciju no relāciju datubāzēm uz semantiskajiem datiem ir atrodams [22], kā arī ir plašāks izklāsts par OWL DL secinātājiem.

4.6. Piekļuve izveidotajiem datiem un noslēdzošie secinājumi

Veicot visas iepriekš aprakstītās darbības, esam nonākuši līdz vēlamajam rezultātam, kad medicīnas zinātniekiem ir pieejams SPARQL piekļuves punkts, kurā viņi var formulēt vaicājumus un izgūt vajadzīgos datus. Esam realizējuši vienu no "Semantiskās Latvijas" idejām, ka lietotājiem ir iespējams sastādīt "semantiski tīrus vaicājumus". Tas ir, nav jāzina papildus tehniskās detaļas.

Kā jau tika minēts, tad dati sākotnēji tika ielādēti Sesame SPARQL piekļuves punktā, taču vēlāk tas tika nomainīts pret Virtuoso serveri, kas nodrošināja ātrdarbības uzlabojumus.

Ņemot vērā, ka dati bija ierobežotas piekļuves, bet SPARQL piekļuves punkti īsti neatbalsta piekļuves tiesības[33], tad tika izmantots IP adrešu reģistrs, kurā medicīnas zinātniekiem bija jāpiereģistrējas, lai no šīs IP adreses viņš varētu piekļūt semantiskajiem datiem. Šāda pieejas tiesību organizēšana neizslēdz nesankcionētu piekļuvi datiem, taču atrisināja minimālo uzdevumu.

Medicīnas darbinieki varēja izgūt datus veidojot tekstuālus SPARQL vaicājumus, kā piemērs ir attēlots 4.12. attēlā, par pamatu ņemot ontoloģiju attēlus, kas būtu izdrukāti un "pielikti pie sienas". Atbilstoši 4.12. attēlā ir nodemonstrēts tekstuāls SPARQL vaicājums, kas ļauj atrast personas vārdu un telefonu tiem pacientiem, kam ir bijusi kā trauma, tā arī vēzis.

Taču tekstuālu SPARQL gadījumā medicīnas zinātniekam ir jāzina ne tikai SPARQL sintakse un semantika, bet arī "pie sienas esošās" ontoloģijas vārda telpas, kas apgrūtina vaicājuma pierakstīšanu.

[33] Virtuoso serverī ir iespējams definēt lietotājus, kam ir dažādas piekļuves tiesības datiem, taču tas darbojas tikai tad, ja tiek izmantots tīmeklī bāzētā Virtuoso aplikācija, kas nodrošina arī tekstuālu SPARQL izpildi. Ņemot vērā faktu, ka mūsu vēlme ir izmantot grafisku vaicājumu formulēšanas rīku, tad šāda iespēja mūs neapmierināja.

```
Query:
select ?name ?phone
where {
?p <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://lumii.lv/preda#Person> .
?t <http://lumii.lv/preda#injPerson> ?p .
?v <http://lumii.lv/preda#cancPerson> ?p .
?p <http://lumii.lv/preda#persName> ?name .
?p <http://lumii.lv/preda#pcPhone> ?phone .
}
--------------------------------------------------
Query Results (1 answers):
name                    | phone
==================================================
"Zigmunds"^^xsd:string | "+37125252525"^^xsd:string
```

4.12. att. ***SPARQL tekstuāla vaicājuma piemērs***

Ņemot vērā, ka tekstuāla SPARQL izstrāde medicīnas zinātniekiem sagādā problēmas, tad arī paralēli tika izstrādāts prototips, kas atbalstītu grafisku SPARQL vaicājumu formulēšanu.

Kā ir minēts rakstā [5], tad ir būtiski, lai lietotājiem būtu pieejami rīki, ar kuru palīdzību veidot SPARQL vaicājumus. Kā arī svarīgi, lai šie rīki sniegtu pietiekamu atbalstu vaicājumu izveidē, kas ļoti atvieglo vaicājumu izveidi.

Tā brīža pieejamie rīki Tabulator [35] un Longwell [36] īsti neatbalstīja izvirzītās prasības. Būtiskākā, rezultātus nebija iespējams eksportēt saprotamā datu formātā, piemēram, Excel. Kā arī abi risinājumi bija balstīti uz paradigmu, ka ir izvēlēta sākuma dati un tad notiek tālāka navigēšana pa tiem. Medicīnas zinātnieku gadījumā tas īsti nederēja, jo bija nepieciešama ne tikai navigēšana, bet arī datu kopas atlase.

Viena no eksperimentālajām izstrādēm SPARK [37] piedāvāja vaicājumus formulēt izmantojot vairākus atslēgas vārdus, tad izmantojot ontoloģijas shēmu lietotājam tiek piedāvāti dažādi SPARQL vaicājumi, no kuriem lietotājs izvēlas sev vispiemērotāko. Šajā gadījumā ir problēmas, ka ir grūti uzstādīt nosacījumus konkrētiem datiem, kā arī medicīnas zinātniekam būtu labi jāorientējas SPARQL vaicājumu tekstuālajā sintaksē, lai izprastu, kurš no vaicājumiem atbilst tam, ko viņš vēlas pavaicāt.

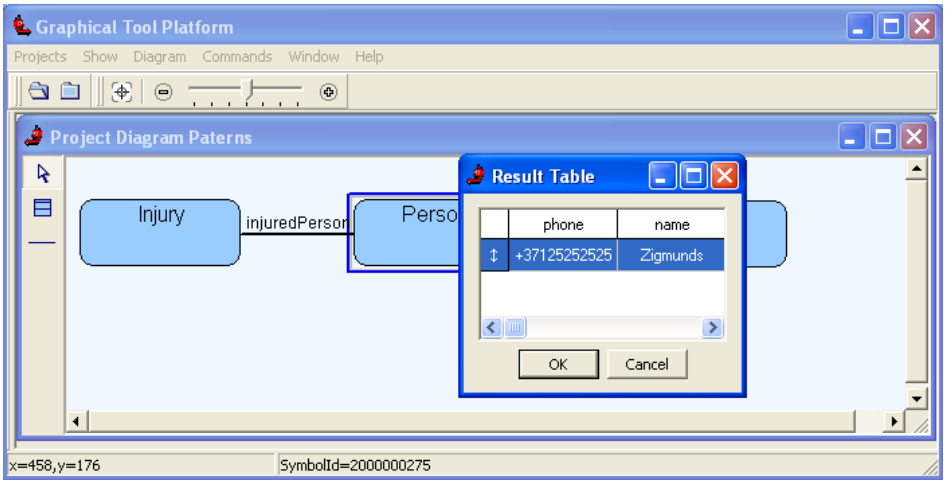
Ņemot vērā tajā brīdī pieejamo risinājumu iespējas un trūkumus, izvēlējāmies izstrādāt rīku, ar kura palīdzību medicīnas zinātnieki varētu veikt datu atlases ērtā veidā, kā arī rezultātu eksportēt uz Excel formātu un to izmantot tālākām analītiskām darbībām.

Jau otrajā nodaļā ir pieejams grafiskās valodas sintakses un semantikas apraksts. Kā lasītājs var pārliecināties, tad tas pārklāj lielāko daļu vaicājumu pamatelementus un nodrošina saprotamu vaicājumu izveidi.

Vēl vienkāršotākas grafiskās valodas sintakse tika izmantota, lai izveidotu eksperimentālu prototipu, kuru medicīnas zinātnieki var izmantot datu atlasei. Tā izveide notika izmantojot

modeļu bāzētas izstrādes tehnoloģijas. Par pamatu tika izvēlēta rīku būves GrTP [34] platforma un transformācijas tika veidotas ar valodas LX [38] palīdzību.

Kā tika prognozēts [5], tad strādājoša prototipa izveide bija iespējama samērā īsā laikā, lai arī dažādu uzlaboju pievienošana un rīka uzlabošana prasīja jau daudz lielāku laika ieguldījumu. 4.13. attēlā var aplūkot izstrādātā prototipa ekrānšāviņu, kas parāda izveidotu vaicājumu, kā arī atbildes piemēru vaicājumam.[34]



4.13. att. ***DEMO rīka prototipa ekrānšāviņš.***

Detalizētāk izstrādātā rīka versija tiks aplūkota nodaļā 7 un tās realizācija ir izklāstīta nodaļā 7.4. Kā redzam no shematiska pieejas apraksta, tad noslēdzošais vaicājumu formulēšanas rīks ir būtiska sastāvdaļa, lai noslēgtu kopējo pāreju uz semantiskajiem datiem un padarītu tos patiešām pieejamus medicīnas zinātniekiem.

Diskusijās ar medicīnas zinātniekiem tika saņemti pozitīvs vērtējums, ka ar vaicājumu rīku ir iespējams ātrāk un vienkāršāk veikt datu atlases, kā tas ir, piemēram, izmantojot SQL vaicājumu valodu. Lietotājus, kas praktiski lietoja vaicājumu rīku datu atlasei, apmierināja izstrādātā valodas sintakse un semantika. Taču tajā pašā laikā tika izteikti dažādi ierosinājumi vaicājuma rīka funkcionalitātes uzlabošanai.

Piemēram, sākotnējā versijā ontoloģijas klases bija nepieciešams ievadīt atsevišķi un rīks nepiedāvāja iespēju pievienot vai vismaz redzēt jau ievadīto klašu saistītās klases, kā rādija prakse, tad medicīnas zinātni labprāt lieto rīku, ja blakus nav jātur izdrukāta ontoloģijas bilde.

Līdzīgi tika izteikts vaicājums – vai būtu iespējams jau ontoloģijā atlasīt agregātu vērtības dažādiem datiem, piemēram, cik daudz traumas vienam cilvēkam ir bijušas. Ņemot vērā, ka tas ir būtiski tālākai datu analīzei. Taču šajā gadījumā bija problemātiskāk piedāvāt risinājumus, jo tā

[34] Rīka darbības laikā tika noslēpts ģenerētais SPARQL. Medicīnas zinātnieki pa tiešo varēja iegūt atbildes rezultātu.

brīža SPARQL valodas rekomendācija neietvēra agregātfunkcijas. Kā arī sākotnēji izmantotais Sesame repozitorijs neatbalstīja šādu iespēju.

Iespējams, tieši agregātfunkciju iespējas trūkums sākotnēji ierobežoja datu atlases izmantošanu, jo bija grūti iegūt apkopojošus datus, kas tālāk varētu tikt izmantoti statistikas apkopošanai un hipotēžu izvirzīšanai medicīnas jomā.

Taču projekta kopējā tendence parādīja, ka ir iespējama pāreja no relāciju datiem uz semantiskiem datiem. Lai arī tehnoloģiskais atbalsts apgrūtina pārejas veikšanu un tālāku datu analīzi, tomēr lietotājiem ir saistošs gala rezultāts un viņi to novērtē atzīstami.

5. ONTOLOĢIJĀS BALSTĪTAS INFORMATĪVĀS SISTĒMAS

5.1. Ontoloģijās balstītu informatīvo sistēmu idejiskais pamats

Pozitīvā pieredze ar semantisko datu saprotamību medicīnas apgabalā [18] norādīja, ka tie ir ērti un labi lietotājiem saprotami. Ņemot vērā šo pieredzi un rakstā "Semantiskā Latvija" pausto ideju, ka būtu iespējams ņemt centralizētu ontoloģiju un veidot tajā esošos datus, ir nākamais solis – izveidot sistēmu, ietvaru, ar kura palīdzību īstenot ontoloģijas datu veidošanu.

Nodaļā 4. tika aprakstīta, izmantojot medicīnas datu piemēru, vispārīga pieeja kā iegūt semantiskus datus no relāciju datubāzē pieejamajiem datiem, kas risina problēmu – kā no izveidotām relāciju datubāzēm transformēt datus uz semantiskiem datiem.

Taču katru dienu tiek veidotas jaunas sistēmas. Ja redzam, ka semantiskām sistēmām ir pieejamas priekšrocības datu analīzei, tad iespējams lietderīgāk būtu atrast metodoloģiju un tehniskas metodes, lai būtu iespējams izveidot strādājošas, semantiskas informatīvās sistēmas.

J.F.Sova (*J.F.Sowa*) ir teicis – "Katras informatīvās sistēmas pamatā ir ontoloģija – tiešā vai netiešā veidā" [39]. Visas informatīvās sistēmas tiek būvētas tā, ka to pamatā ir kāda zināšanu bāze. Taču zināšanu bāzi var aprakstīt un attēlot arī ar ontoloģiju, ne tikai, piemēram, relāciju datu bāzi.

Ņemot vērā, ka relāciju datu bāze tiek aizpildīta ar datiem un ka virs tās tiek veidota informatīvā sistēma, un ka relāciju datu bāzi var pārveidot par ontoloģiju, tad varam uzreiz izveidot informācijas sistēmu, kas ir balstīta uz ontoloģiju.

Primārais jautājums, kas būtu jāuzstāda – kādi ir iespējamie ieguvi, ko varētu dot ontoloģijās balstītu informatīvo sistēmu izstrāde?

Pirmkārt, nodaļā 4. ir minētais ieguvums, ka spējam izveidot semantiski saprotamus datus, ar kuriem lietotājiem ir ērtāk darboties. Otrkārt, tā ir iespēja, vismaz daļēji, veidot informatīvās sistēmas automātiski.

Ņemot vērā, ka ontoloģija tiek iekļauta semantika, tad mēs izvirzām tēzi, ka, izmantojot šo semantiku, ir iespējams ģenerēt informatīvās sistēmas saskarnes automātiski, lai ar to palīdzību labotu un papildinātu ontoloģijas datus, kā arī šīs saskarnes formas būtu lietotājiem saprotamas un ērti lietojamas.

Tālāk aprakstītā tehnoloģija kā ievaddatus saņem OWL [40] ontoloģiju, automātiski izveido tīmeklī bāzētu informatīvo sistēmu darbam ar ontoloģijas datiem – datu ievadu, datu labošanu un dzēšanu. Esošā informatīvās sistēmas pieeja ir izstrādāta, ka tā ņem vērā tikai RDFS[35] ontoloģijas daļu. Galvenokārt tas ir tāpēc, ka primārais prototipa izstrādes process ir veltīts tam, lai tā būtu lietotājiem draudzīga un ērti lietojama. Papildus funkcionalitāti ir paredzēts attīstīt laika gaitā.

Mūsu aprakstītajā pieejā mēs uztveram, ka ontoloģija iedalās divās daļās. Pirmā ir tā, kas attēlo ontoloģijas faktualo daļu un sakrīt ar relāciju datubāzu shēmas uzdošanu. Piemēram, ir iespējams atzīmēt, ka Pēteris ir persona vai Pēterim pieder Audi markas mašīna ar numuru "PD 1".

Otrā pieejas daļa ir balstīta uz ontoloģijās pieejamām secināšanas iespējām, kā arī nosacījumu likumiem, kas pārbauda ontoloģijas faktuālās daļas datu pareizību. Šie nosacījumi atbilst relāciju datubāzēs uzdotajiem nosacījumiem un trigeriem, kas var izveidot papildus informāciju, vai datu skatiem. Piemēram, var būt nosacījums – "Ja persona ir vīrietis un ja viņam ir vismaz viens bērns, tad viņš ir tēvs". Relāciju datubāzu gadījumā šim piemēram būtu iespējams izveidot speciālu skatu, kur tiktu parādīti visi tēvi.

Mūsu pieejas primārā daļa ir balstīta uz faktuālo daļu, kur gala lietotāji darbojas ar ontoloģijas objektu informāciju – to atribūtu vērtībām, saitēm starp objektiem, objektu piederību klasēm.

Daļa no pamatojuma, kāpēc koncentrējamies uz RDFS ontoloģiju apakškopu (klasēm, to atribūtiem, relācijām un apakšklašu attiecībām), ir tas, ka tas šobrīd ir vienīgais, ko nodrošina šobrīd pieejamie SPARQL piekļuves punktu serveri[36].

Mēs redzam, ka piedāvātā ontoloģijās balstītā metodoloģija būtu ļoti piemērota mazām organizācijām, jo bez lieliem ieguldījumiem būtu iespējams ērti un ātri izstrādāt tīmeklī bāzētu informatīvo sistēmu datu uzkrāšanai. Mazas organizācijas bieži uzskata, ka informatīvo sistēmu izstrāde ir dārga un tālāk prasa sarežģītu uzturēšanu. Vai arī izvēlas vienkāršas Excel datnes, kurās ir grūti nodrošināt datu kvalitāti, kā arī datu savstarpēju saistību ar relācijām.

Mūsu piedāvātajā pieejā, mazajām organizācijām būtu jāapraksta tikai sava struktūra ontoloģijā, kas nodrošinātu vienkāršu, strādājošu sistēmu. Ņemot vērā, ka ontoloģijas nesatur papildus tehnisku informāciju, tad lietotājiem būtu jāsaprot izstrādātā sistēma.

[35] http://www.w3.org/TR/rdf-schema/

[36] Mēs izmantojam Virtuoso (http://virtuoso.openlinksw.com/) SPARQL piekļuves punktu, kas nodrošina RDFS secināšanas daļu un atbalsta SPARQL 1.1. Šobrīd tiek izstrādāts arī StarDog SPARQL piekļuves punktu serveris, kas atbalstīs pilnu OWL secināšanu, taču pagaidām tas neatbalsta SPARQL 1.1. datu atjaunošanas daļu.

Piemēram, frizētavā būt jāizveido tika sava ontoloģija, lai iegūtu vienkāršu, strādājošu sistēmu. Frizētavas darbiniekiem būtu iespējams labot informāciju par to, kad un kādi frizieri ir pieejami, kad ir aizņemti laiki. Turpretim frizētavas klienti varētu piekļūt sistēmai lasīšanas režīmā (vai arī SPARQL piekļuves punktam), lai pārbaudītu, kad ir iespējams pierakstīties pie friziera. Tas ļautu tuvoties sākotnēji aprakstītai idejai par semantisko tīmekli, ka ir aģenti, kas apkopo informāciju un piedāvā optimālos variantus.

Aprakstītās idejas tika izstrādātas darba grupā projekta ERAF projekta nr. "2011/0009/2DP/2.1.1.1.0/10/APIA/VIAA/112" ietvaros. Sistēmas prototipa izveidi ir īstenojusi Aiga Romāne.

5.2. Veterinārās klīnikas piemērs ontoloģijas balstītai informatīvās sistēmas izstrādei

Lai būtu labāk saprotama ontoloģijās balstīta informatīvās sistēmas izstrāde, tad nodemonstrēsim ļoti vienkāršotu piemēru. Esam izvēlējušies veterinārās klīnikas apgabalu. Parasti veterinārās klīnikas nav pārāk lielas un atbilst mazu organizāciju statusam.

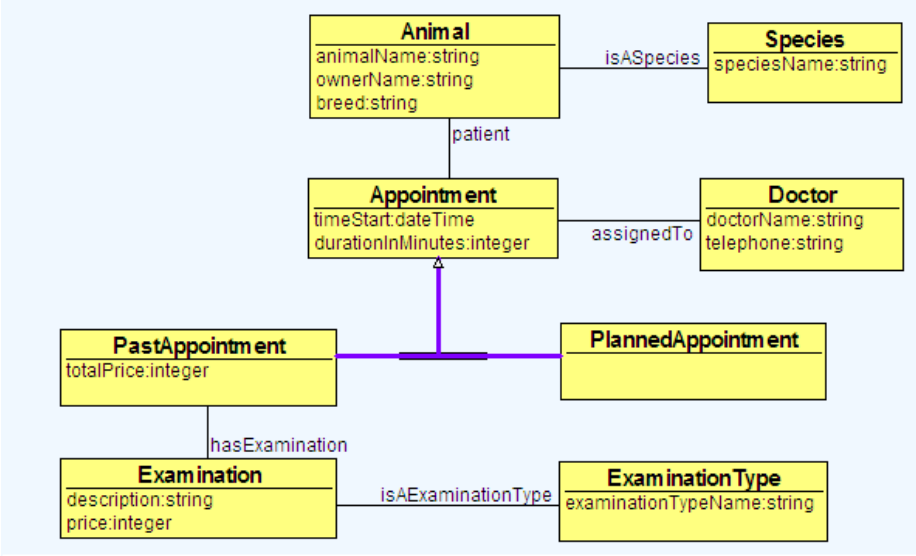
Lai veidotu ontoloģijas balstītu informatīvo sistēmu (turpmāk saīsināsim kā OBIS), mums ir nepieciešama OWL ontoloģija, kas reprezentē atbilstošo domēnu. Šajā gadījumā vēlamies, lai tā aprakstītu veterināro klīniku.

Mēs varam pārbaudīt esošos ontoloģiju repozitorijus, lai mēģinātu atrast ontoloģiju, kas pārklātu mūsu veterinārās klīnikas apgabalu. Piemēram, DERI vārdnīcu katalogu[37]. Vai meklēt semantiskajā tīmeklī atbilstošu ontoloģiju izmantojot Swoogle[38].

Tā kā mums neizdevās atrast atbilstošu ontoloģiju, kas to pārklātu, tad izveidojām savu veterinārās klīnikas ontoloģijas piemēru. Mēs izmantojām OWLGrEd ontoloģiju redaktoru [41], taču varētu izmantot arī, piemēram, Protege ontoloģiju redaktoru[39].

Ņemot vērā, ka sistēma ir paredzēta mazai veterinārai klīnikai, tad iekļausim ļoti minimālus datus – ārsti, kas strādā, dzīvnieki, kas tiek izmeklēti un dzīvnieku suga, apmeklējumi, kas iedalās plānotajos un vēl paredzētajos, un veiktās darbības apmeklējuma laikā, kā arī kādai kategorijai pieder veiktā darbība (izdarām pieņēmumu, ka ārsts, pie kura notiek apmeklējums, ir arī ārsts, kas veic izmeklējumu). Uzzīmētā ontoloģija grafveida formātā ir apskatāma 5.1. attēlā.

[37] http://vocab.deri.ie/
[38] http://swoogle.umbc.edu/
[39] http://protege.stanford.edu/



5.1. att. ***Veterinārās ontoloģijas piemērs.***

5.2.1. Jaunas OBIS sistēmas izveide

Kad esam izvēlējušies vai izveidojuši nepieciešamo domēna ontoloģiju, tad esam nonākuši līdz solim, kur mums ir jāizveido strādājoša informatīvā sistēma.

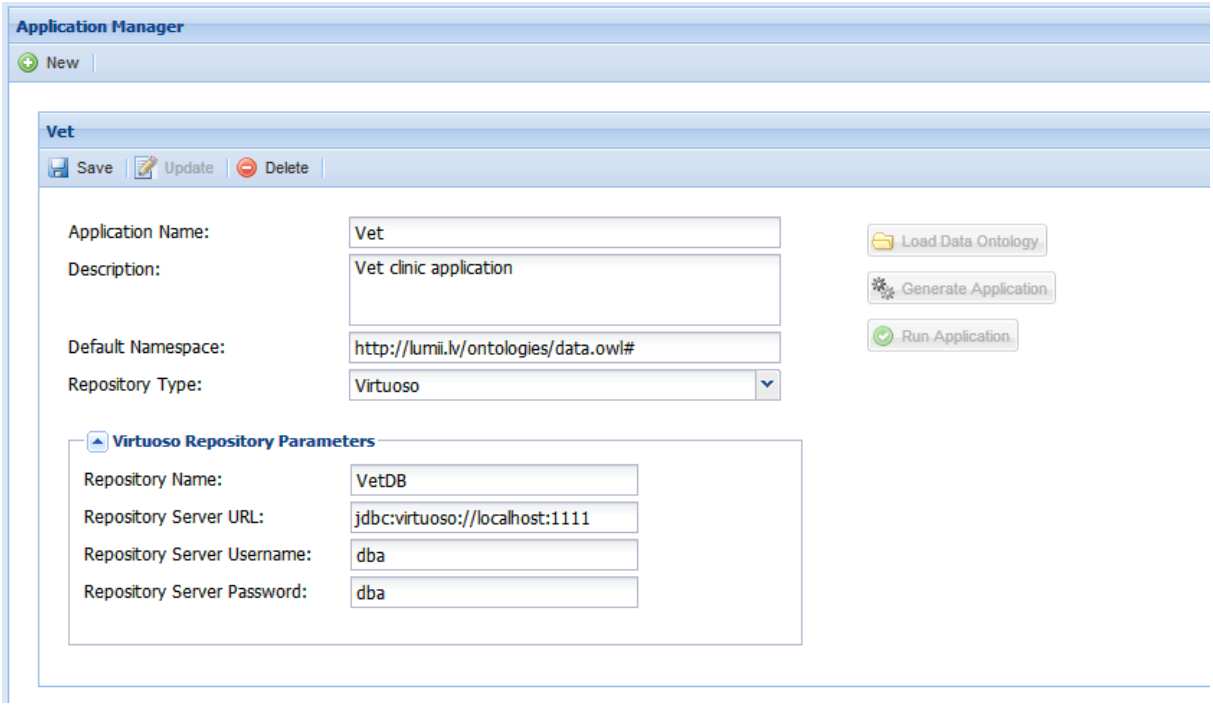
Šim solim ir jābūt iespējami vienkāršam, lai lietotāji ātri nonāktu pie strādājošas sistēmas, ko vēlāk būtu iespējams uzlabot.

Ņemot vērā, ka tīmeklī bāzētām sistēmām ir nepieciešams servera atbalsts, tad ir divas iespējas. Pirmkārt, izveidot serveri, ko lietotājs var uzstādīt pie sevis, to palaist un tad sākt darboties ar OBIS sistēmu. Lai arī tas process nav sarežģīts, tomēr tas prasa noteiktas zināšanas un darba uzsākšana nav tik vienkārša. Turklāt, papildus ir jāuzstāda arī SPARQL piekļuves punkta serveris, kas nodrošina datu glabāšanu.

Otrkārt, ir iespēja izveidot serveri, kurš jau nodrošina šādu servisu un ļauj lietotājam tikai ielādēt ontoloģiju un norādīt adresi, kā piekļūt šīs ontoloģijas informatīvai sistēmai, lai sāktu darbu. Kā arī piedāvāt izmantot SPARQL piekļuves punktu, vai arī norādīt savu SPARQL piekļuves punkta servera adresi.

No ielādētās ontoloģijas datiem sistēma ģenerē informatīvo sistēmu, ka katrai klasei atbilst tabula, atbilstoši katras klases atribūtam tabulā ir kolona. Relācijas gan tiek attēlotas pie katra indivīda, kur uzrādās, kādi ir pieejamie dati, ar ko tas ir saistīts.

Varam aplūkot 5.2. attēlā OBIS sistēmas izveides logu, kur ir norādīts, kā izveidot jaunu OBIS sistēmu un uzsākt darbu.

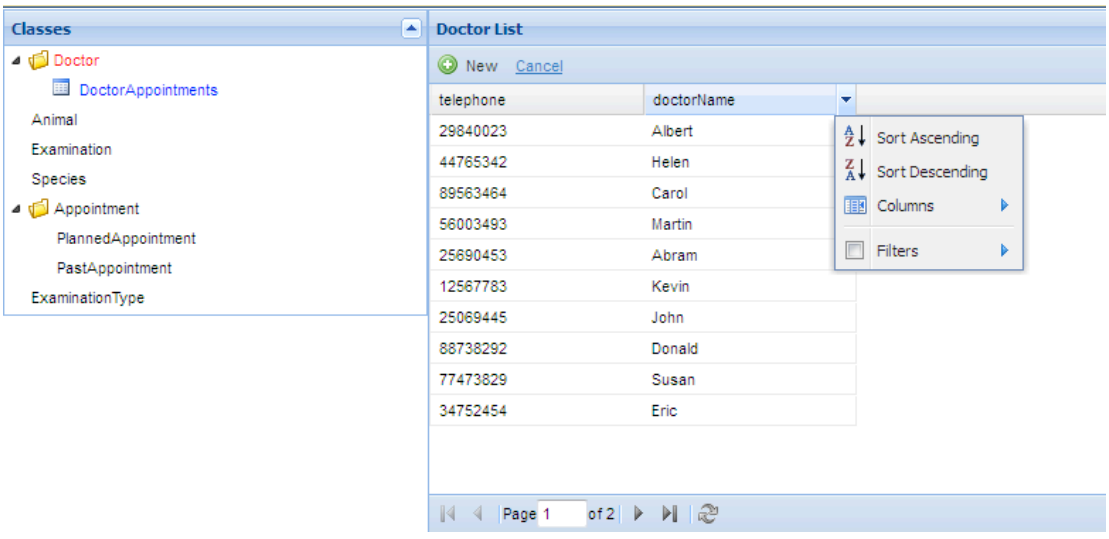


5.2. att. ***OBIS sistēmas pārvaldības rīks***

5.2.2. OBIS sistēmas datu pievienošanas un labošanas daļa

Kad konkrētā OBIS sistēma (veterinārās klīnikas) ir uzstādīta uz OBIS sistēmas servera, tad tā ir pieejama gala lietotājiem, lai tie varētu darboties ar datu pievienošanu un labošanu.

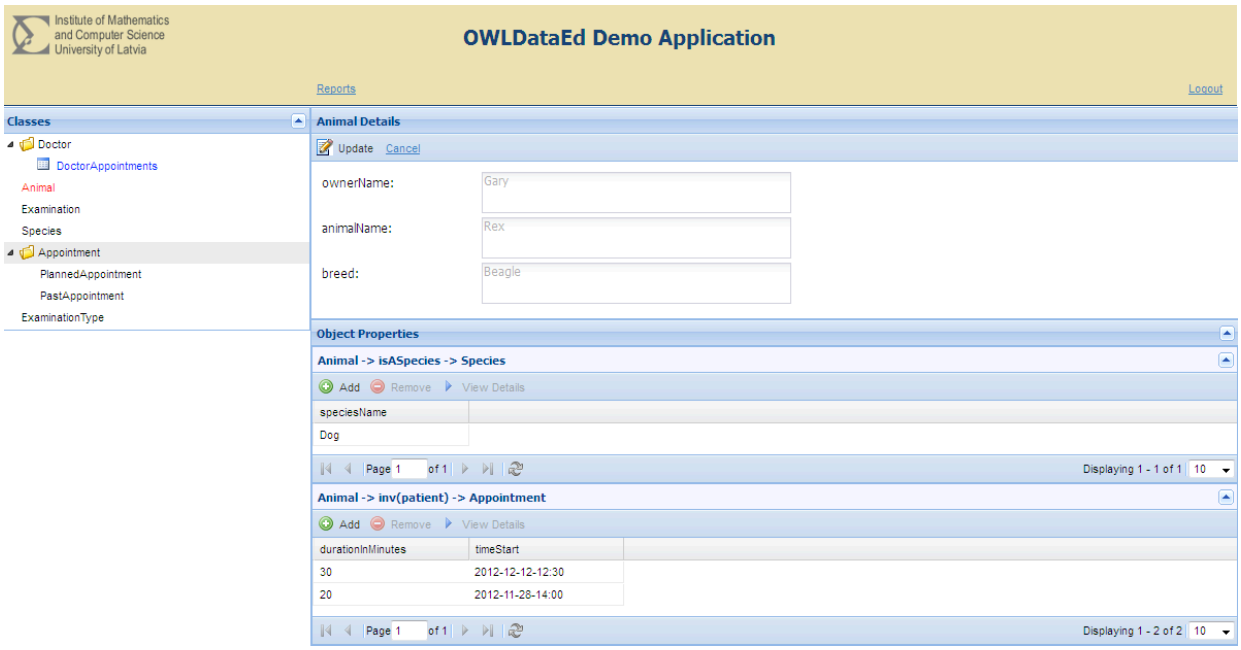
Automātiski ģenerētā datu ievades un labošanas sistēma ir samērā vienkārša. Tā sastāv no divām daļām – datu struktūras paneļa un datu paneļa, kur tiek rādīti vai nu visi klases dati, vai arī konkrēti izvēlētā indivīda dati.



5.3. att. ***Klases instanču labošanas logs.***

Datu struktūras panelis satur kokveida formā attēlotu ontoloģiju līdzīgi kā tas tiek darīt rīkā Protege[40]. Kokveida struktūra tiek veidota balstoties uz klašu mantošanās īpašību. Ja klase ir apakšklase tikai *owl:thing*, tad šī klase tiks rādīta pirmajā līmenī, ja tā ir apakšklase kādai citai klasei, tad tā tiks rādīta izvērsumā zem klases, kurai tā ir apakšklase.

Klases instanču labošana notiek blakus esošajā panelī, kā tas ir parādīts 5.3. attēlā. Ja tiek izvēlēta viena klase kokveida izvēlnē, tad izvēlētās klases instances tiek attēlotas blakus esošajā datu panelī. Attēlošanai tiek izvēlēts tabulas formāts, jo tās ir populāras datu attēlošanai un cilvēki tās ir labi apguvuši. Sistēmas lietotājs var apskatīt kādi objekti ir konkrētajā klasē, meklēt vai filtrēt tos pēc papildus vērtībām. Papildus sistēmas lietotājs var arī veidot jaunus objektus, ja viņam ir dotas šādas tiesības. Ja tiek izvēlēts viens konkrēts objekts, tad datu tabulas vietā tiek atvērts logs, kurā notiek darbība ar izvēlēto objektu.



5.4. att. ***Instances pārlūkošana un labošana.***

Tiek parādītas visas objekta atribūtu vērtības secīgi, kā arī izejošās un ienākošās[41] saites uz citām klasēm un tur piesaistītajiem objektiem. Detalizētu piemēru ir iespējams aplūkot 5.4. attēlā. Sistēmas lietotājs šādā veidā viegli var pievienot, labot un dzēst instanču datus izmantojot SPARQL 1.1 protokolu, kamēr dati tiek glabāti SPARQL piekļuves punktā.

[40] http://protege.stanford.edu/

[41] Lai arī formāli ontoloģijās saites savā starpā var saistīt ar "*inverseOf*" īpašību, tādā veidā tās pārklājas, taču praksē tiek lietotas tikai īpašības vienā virzienā un apgrieztā īpašība netiek definēta. Tas ir saistīts ar to, ka esošie SPARQL piekļuves punktu serveri nenodrošina servisu, kas apkalpotu "*inverseOf*" īpašību, tāpēc sistēmas uzturētājam pašam būtu jāveido papildus trijnieki vai arī jāpārformulē vaicājumi. Tāpēc izvēlamies šobrīd ērtāko risinājumu, lai informācija reizēm dublētos, bet tomēr būtu pieejamas visas definētās saites.

Turklāt, ar šādu attēlošanu varam atrisināt būtisku semantiskā tīmekļa problēmu, ka indivīdu datiem tiek lietoti URI, kas nav atverami.[42] Ņemot vērā, ka mūsu sistēma veidojot jaunus datus ģenerē to URI, tad atbilstoši šis URI tā, lai to ievadot tas parādītu izvēlēto instanci, kas ir izdarāms, ja URI tiek ģenerēts formā "http://sistēmas adrese/unikālais objekta identifikātors". Atbilstoši, ja pārlūkā tiek ievadīta šī URI, tad mūsu sistēma saņems šo pieprasījumu un atvērs konkrētā objekta detalizācijas lapu.

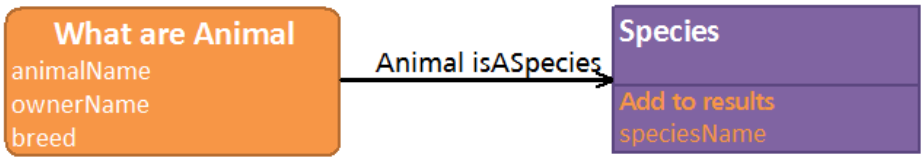
5.2.3. Atskaites OBIS sistēmā

Relāciju datubāzēs tiek veidoti skati, vēlāk definēto skatu dati tiek attēloti kā atskaites lietotājiem. Šī ir svarīga funkcionāla īpašība, kas ļauj apkopot informāciju un paskatīties uz datubāzē esošajiem datiem no cita skatupunkta.

Arī mēs uzskatām, ka skati ir svarīga OBIS sastāvdaļa. Taču vēlamies papildināt skatus ar papildu īpašību, ka ir skatā attēlotajos datos ir iespējams navigēt arī uz konkrētiem instanču datiem.

Ja aplūkojam nodaļā 3. aprakstīto grafisko vaicājumu valodu, tad redzam, ka katram vaicājumam tiek definēta arī centrālā klase. Tāpēc šī centrālā klase var tikt saglabāt arī vaicājumā izveidotajā skatā. (Tehniski varam to realizēt, ka centrālās klases objekti tiek atlasīti ar ?ID mainīgo, lai to atpazītu sistēmā.) Atbilstoši varam izvēlēties jebkuru rindu skata tabulā un navigēt uz centrālās klases objektu, kas veido izvēlēto rindu, lai detalizēti apskatītu konkrētā objekta īpašības, kas ir veidojis izvēlēto rindu.

Piemēram, vēlamies aplūkot dzīvnieku datus, lai uzreiz tiem būtu piesaistīts arī sugas vārds. 5.5. attēlā varam aplūkot grafisko vaicājumu, kā tas izskatās un 5.6. attēlā ir dots atbilstošs SPARQL vaicājums. Kā arī 5.7. attēlā ir parādīts kā tas izskatās OBIS sistēmā.

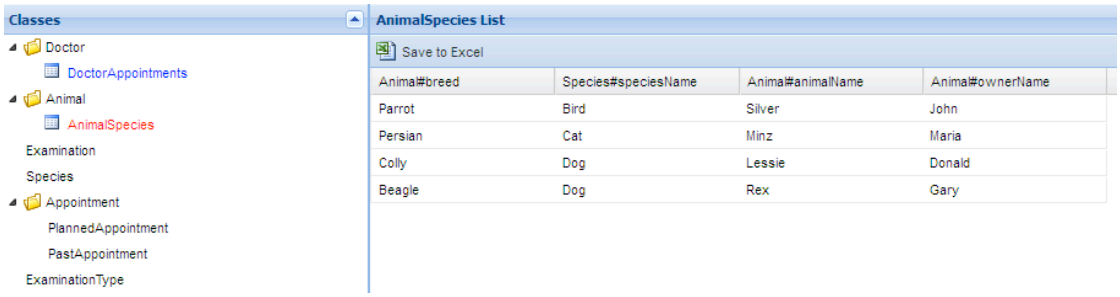


5.5. att. ***ViziQuer vaicājums atskaites definēšanai***

[42] Personīga komunikācija ar pētnieku Fumihiro Kato, kas darbojas Linked Data Japānas kustībā.

```
SELECT DISTINCT
    ?ID   ?Animal__animalName ?Animal__breed   ?Animal__ownerName
    ?Species__speciesName
WHERE {
          ?ID rdf:type :Animal.
          OPTIONAL { ?ID :animalName ?Animal__animalName.}
          OPTIONAL { ?ID :breed ?Animal__breed.}
          OPTIONAL { ?ID :ownerName ?Animal__ownerName.}
          ?ID :isASpiecies ?Species.
          ?Species rdf:type :Species
          OPTIONAL { ?Species :speciesName ?Species__speciesName.}}
```

5.6. att. ***ViziQuer vaicājuma SPARQL sintakse***

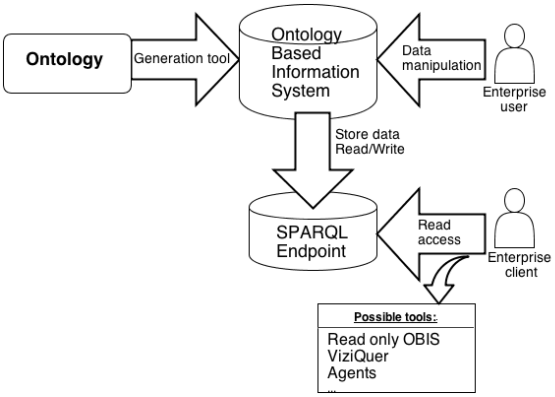


5.7. att. ***Atskaites rezultāts OBIS sistēmā.***

Ņemot vērā, ka visi OBIS izveidotie dati tiek saglabāti SPARQL piekļuves punktā, tad ir iespējams veidot arī SPARQL vaicājumus un izpildīt tos pa tiešo bez tīmeklī bāzētās informatīvās sistēmas palīdzības.

5.3. OBIS arhitektūra

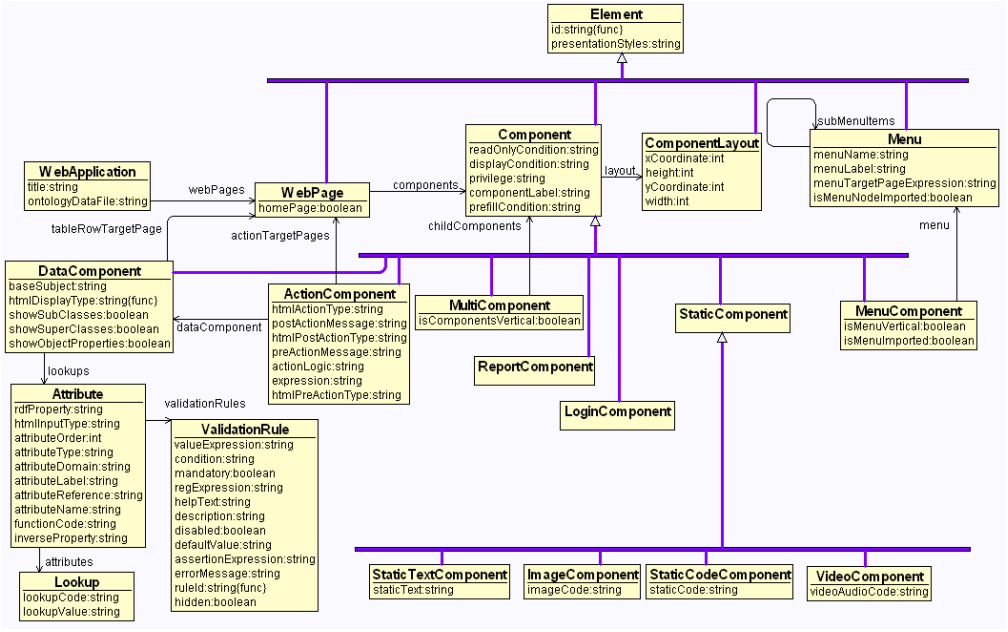
OBIS arhitektūras pārskats ir parādīts 5.8. attēlā. Arhitektūra iedalās 3 daļās – pirmkārt, ontoloģija, kas reprezentē datu struktūru. Otrkārt, OBIS informatīvā sistēma, kas ir tikusi ģenerēta no datu ontoloģijas un pielāgota organizācijai un tās darbinieku lietošanai, kā arī, treškārt, SPARQL piekļuves punkts lasīšanas režīmā, kas ir pieejams organizācijas klientiem.



5.8. att. ***OBIS arhitektūra***

Mēs esam definējuši modeļu transformācijas no OWL ontoloģijas uz tīmeklī bāzētu OBIS interpretācijas modeli[43], ko var aplūkot 5.9. attēlā. Katra izveidotā OBIS informācijas sistēma ir instanču kopa, kas pieder OBIS interpretācijas modelim un ir pilnībā modeļu bāzētas pieejas realizācija [42].

Secīgi tas būtu tā, ka OBIS ģenerācijas rīks ielasa doto ontoloģiju un izveido OBIS modeļa instanču kopu. Tālāk OBIS interpretācijas rīks ielasa doto instanču kopu un attēlo to, kā tīmekļa lapas. Šādā veidā ir vienkārši, ērti un automatizēti iespējams iegūt strādājošu tīmeklī bāzētu informatīvo sistēmu.



5.9. att. OBIS meta modelis.

Ņemot vērā, ka centrālā daļa ir OBIS modelis, tad ir iespējama sistēmas papildus konfigurācija, lai izmainītu un uzlabotu ģenerētās sistēmas instanču kopu. Varam minēt trīs iespējamos scenārijus – OBIS ģenerācijas rīks parāda sistēmas konstrukciju un ļauj to pielāgot savām vajadzībām, veidojot dažādas papildus lapas, pievienojot papildus izvēlnes. Otrkārt, ontoloģiju ir iespējams papildināt ar sistēmas konfigurācijas izskatu, ko OBIS ģenerācijas rīks saprot un ģenerē tai atbilstošu sistēmu. Piemēram, atribūtu secība klasē.

Trešā iespēja ir veidot papildus konfigurācijas datnes, balstoties uz ontoloģijas klasēm. Tā kā ontoloģiju var izmantot vairāku sistēmu ģenerēšanai, tad ir iespējams sarakstīt dažādas lapu konfigurācijas, kur jaunas sistēmas veidotājs izvēlēsies tās, kuras vislabāk atbilsta veidojamai sistēmai. Piemēram, katram ir sava vēlamā atribūtu secība.

[43] Interpretācijas modeļa galvenā izstrādātāja ir Aiga Romāne.

Mēs neiedziļināsimies sīkāk sistēmas papildus konfigurācijas detaļās, jo katrs no variantiem satur savus plusus un mīnusus, kā arī tā ir detalizēta analīze, lai norādītu, kurš no risinājumiem labāk atbalsta lietotāju vajadzības un vēlmes.

Papildus sistēmas aspekts tiek noslēpts arī tabulārās datu formās, kas ir ierastas relāciju datu bāzēm, taču nav tik ierastas semantiskā tīmekļa apgabalā. Ja relāciju datubāzē katram objektam tabulā var būt nulle vai viena konkrētā atribūta vērtība, tad semantiskā tīmekļa gadījumā tas nav ierobežots.

Tāpēc ir jāpielieto specifiski vaicājumi, kur, izmantojot SPARQL 1.1 agregācijas funkciju *GROUP_CONCAT,* ir iespējams apvienot vairākas vērtības vienā. Piemēram, personai var būt uzdoti vairāki telefona numuri un visas vērtības tiek apvienotas vienā laukā ar *GROUP_CONCAT* palīdzību.

Esošajā sistēmas realizācijā gan tiek pieņemts, ka atribūtu skaits ir ierobežots ar maksimālo kardinalitāti 1, lai nodrošinātu apmierinošu sistēmas ātrdarbību. Mūsuprāt, ontoloģijā būtu speciāli jāatzīmē atribūtu kardinalitāte, kur ir iespējama lielāka par 1, lai tas neietekmētu sistēmas ātrdarbību.

Otra problēma, ko tikai ieskicēsim ir OWL nosacījumu pārbaude ontoloģijā. Acīmredzams risinājums OWL nosacījumu implementācijai būtu to pārrakstīšana par SPARQL vaicājumiem līdzīgi kā tas tiek darīts darbā [43].

Mūsu piedāvātais risinājums būtu izveidot papildus sistēmas nosacījumu daļu, kas darbotos servisa veidā. Līdz ar ontoloģijas ielādi OBIS sistēmas ģenerācijai, tiktu uzģenerēti dotai ontoloģijai specifiski nosacījumi, ko varētu noteiktos brīžos pieprasīt izpildīt.

Ņemot vērā, ka sistēmā notiek darbība ar indivīdiem, tad mēs vēlamies arī pārbaudīt vai pievienotajam, labotajam indivīdam izpildās arī visi dotie nosacījumi. Tad padodot sistēmas nosacījumu daļai konkrēto indivīdu tiktu izsecinātas klases, kurai tas pieder, kā arī nosacījumi, kas tam ir jāpārbauda. Piemēram, vai tam ir aizpildīti visi atribūti. Ja tiek veidota saite starp diviem indivīdiem, tad pārbaudei tiktu padoti abi indivīdi, starp kuriem tiek veidota jaunā saite. Ja pārbaudes rezultātā tiek atklāta nepilnība, tad par to tiek ziņots lietotājam.

Ņemot vērā, ka nav sistēmas kā lietotājam piespiest izlabot kļūdas, tad papildus ir iespējamas nosacījumu pārbaudes arī visas klases indivīdiem, lai atgrieztu sarakstu ar kļūdām, kas būtu jālabo.

Detalizētāk par OBIS sistēmas tehnisko realizāciju ir iespējams izlasīt rakstā [44]. Tehniskās realizācijas autore ir Aiga Romāne.

Tehniskās realizācijas pamatā ir 5.9. attēlā aplūkojamais tīmekļa bāzētās sistēmas modelis, kuram ir izveidots interpretators balstīts uz GWT[44]/GXT[45] pieejamajām tīmekļa komponentēm. Izvēlamies dotajos ietvaros piedāvātās grafiskās komponentes un izveidojam modeli, ko mēs varam attēlot.

Ontoloģijas pārveidošana kā OBIS modeļa instance tiek veidota, ka dotā ontoloģija ir instance vispārīgā ontoloģiju metamodelī. Vispārīgais ontoloģijas metamodelis tiek pielīdzināts OBIS modelim. Tad saņemot instanču kopu vispārīgajā ontoloģiju metamodelī, mēs to pārveidojam par instanču kopu OBIS modelī.

5.4. OBIS izvērtējums un perspektīva

Izstrādātās pieejas pārbaudi veicām Latvijas Universitātē izstrādātā lingvistikas projektā, kur bija nepieciešams apskatīt FrameNet[46] ontoloģijas datus, kā arī pievienot jaunus, lai tos adaptētu no angļu valodas uz latviešu valodu.

OBIS informatīvās sistēmas prototips tika izvēlēts tāpēc, ka lingvistikas zinātnieki bieži nāca ar ierosinājumiem par datu shēmas labojumiem un izmaiņām. Šādā veidā mēs atvieglojām darbu, jo izmainot tikai datu shēmu, mēs automātiski tikām pie strādājošas sistēmas tīmekļa versijas.

Lingvistikas zinātnieki atzinīgi novērtēja pieejamās tabulas formāta pārskatus, kurās var pārskatīt izvēlētās klases instances. Taču papildus izrādījās arī būtiska prasība, ka ir nepieciešami skati, kur izvēlētās klases instances tiek papildinātas ar īpašībām no piesaistītajām instancēm.

Ņemot vērā, ka lingvistikas zinātnieki ne tikai pārskatīja tabulas formāta datus, kas tika izveidoti ar pārskatu palīdzību, bet arī vēlējās tos labot, tad tas noteica centrālās klases ieviešanu šajos pārskatos, lai tie kļūtu labojami instanču līmenī.

Sasniedzot nepieciešamos OBIS sistēmas uzlabojumus, lingvistikas zinātnieki bija apmierināti ar tā piedāvātajām iespējām darboties ontoloģijas datu līmenī.

Sistēmas ātrdarbības pārbaude tika veikta izmantojot medicīnas projektā izmantotās ontoloģijas un tur pieejamos datus, kas ir aprakstīta nodaļā 4. Tika ielādētas apmēram 60 klases un apmēram 300 000 indivīdu. OBIS sistēma strādāja, lai arī lielāku klašu gadījumā bija problēmas ar ielādes laiku, jo "Personas" klases ielāde, kas satur apmēram 70 000 indivīdus, aizņēma apmēram 30 sekundes.

[44] GWT (Google Web Toolkit) - https://developers.google.com/web-toolkit/
[45] GXT (Google Extended Toolkit) - http://www.sencha.com/products/gxt
[46] http://framenet.icsi.berkeley.edu

Ātrdarbības problēmas ir daļēji skaidrojamas ar sistēmas tehnisko realizāciju un funkcionalitāti, jo tā ielādē visus datus tabulā un tālāk ļauj lietotājam meklēt vajadzīgos datus. Līdz ar to esošajā risinājumā ilgāks laika sprīdis ir sistēmas tabulas ielādei, bet vēlāk notiek ātrāka meklēšana tabulas ietvaros.

5.4.1. Līdzīgās pieejas

Ir līdzīgi rīki, kas pretendē uz ontoloģijām balstītām sistēmām, piemēram, WebProtege [45]. Taču WebProtege primārais fokuss ir uz ontoloģiju shēmu rediģēšanu, ontoloģiju datu aizpildīšanas daļa nav tik labi attīstīta. Arī rīkā nav ērti veidi kā attēlot datus, jo neieviešot ierobežojumu uz datu kardinalitāti, tos nav iespējams attēlot tabulārā formā.

Papildus problēma ir tā, ka WebProtege nenodala ontoloģijas datu shēmas labošanu no ontoloģijas datu labošanas. Lai arī ar ontoloģijas datu shēmas labošanu būtu jānodarbojas tikai informatīvās sistēmas administratoram. Netiek nodalītas arī sistēmas datu lasīšanas režīms no rakstīšanas režīma.

Līdzīgs rīks ir arī OntoWiki [46], kas ir paredzēts tieši ontoloģiju datu labošanai. Tāpat rīks piedāvā arī pieeju SPARQL piekļuves punktam, kurā glabājas visi izveidotie dati. Rīka jaunākā versija satur arī vizuālu SPARQL vaicājumu sastādīšanas rīku [47]. Kā wiki bāzēta pieeja tā nenodrošina stingru datu atbilstību ontoloģijas datu shēmai. Ontoloģijas datu shēma vienkārši ir kā rekomendācija, pēc kuras būtu jāievada dati, kas īsti neatbilst priekšstatam par informatīvajām sistēmām.

Arī relāciju datubāzēs ir līdzīgi risinājumi, kas piedāvā dažādas sagataves, kuras var izmantot, lai veidotu informatīvo sistēmu, kas ir balstīta uz relāciju datubāzi. Piemēram, šādu pieeju atbalsta Oracle APEX[47].

Galvenā atšķirība no OBIS informatīvās sistēmas ir tā, ka Oracle APEX nenotiek sākotnējā sistēmas ģenerācija. Lietotājam pašam ir jāliek klāt pa vienam solim nepieciešamās lietas. Taču tajā pašā laikā izstrādātāji ir izvēlējušies iet nedaudz citu ceļu – ir definētas dažādas *transformācijas* starp relāciju datubāzes tabulu un to izskatu informatīvajā sistēmā. Tiek piedāvātas dažādas vispārīgas lapu sagataves, kas ir neatkarīgas no datu struktūras. Veidojot sistēmu, lietotājs komponē šīs sagataves un pieliek to sasaisti ar izveidoto relāciju datu bāzi.

[47] http://apex.oracle.com/

5.4.2. Noslēdzošie secinājumi

Pirmie eksperimentālie rezultāti norāda, ka izstrādātā OBIS pieeja sistēmu izstrādē var sniegt ātru rezultātu, lai tiktu pie vienkāršām un strādājošām tīmeklī bāzētām sistēmām, kas apmierina gala lietotājus. Arī praktiskie eksperimenti parāda, ka apjomam līdz 10 000 ierakstiem vienā tabulā esošā sistēmas ātrdarbība ir piemērota.

Taču tajā pašā laikā ir ļoti plašas iespējas kā sistēmu pilnveidot un padarīt vēl ērtāku un labāku. Pirmkārt, jau pieminējām, ka ir iespējams pilnveidot trīs dažādus risinājumus, kā uzlabot automātiski ģenerētās sistēmas ērtumu un estētisko skatījumu. Otrkārt, ir nepieciešama metode kā aprakstīt, kad un kādi indivīdi nomaina klasi, kuriem tie pieder. Piemēram, minētajā veterinārās klīnikas gadījumā ir "Plānotie apmeklējumi" un "Notikušie apmeklējumi". Vienā brīdī mums ir jāpārvieto "Plānotā apmeklējuma" instance uz "Notikušajiem apmeklējumiem".

Taču tajā pašā laikā varam apskatīties, kā OBIS pieeja iederas kopējā skatā par Semantiskā tīmekļa vīziju, ko aprakstīja Tims Berners Lī un kolēģi [6]. Kā tika minēts, tad Semantiskajā tīmeklī primāra loma tika atvēlēta automātiskiem aģentiem, kas varētu apkopot datus no dažādiem servisiem. Piemēram, aplūkot tuvākajā apkārtnē esošās veterinārās klīnikas un atbildēt, kurā no tām ir pieejams brīvs laiks apmeklējumam pēc stundas.

OBIS pieejas ieviešana varētu būt solis šādas idejas praktiskā īstenošanā, jo visas sistēmas būtu balstītas uz vienotu ontoloģiju, kas nozīmētu, ka datus savā starpā būtu iespējams salīdzināt un veikt pār tiem vaicājumus.

Otra nepieciešamā lieta būtu ontoloģiju repozitorijs. Ja rakstā [5] tika minēts centrālais valsts repozitorijs, kurš saturētu vispārīgās ontoloģijas, kā arī sistēmu, kā būtu iespējams veidot tajā datus un publicēt tīmeklī, tad rakstā [48] ir precīzāk aprakstīts, kā šajā repozitorijā būtu iespējams ne tikai glabāt ontoloģijas, bet arī sasaisti ar to datiem. Izklāstīsim šo ideju nodaļā 6.

6. ONTOLOĢIJU REPOZITORIJI

6.1. Ontoloģiju repozitoriju nepieciešamība

Ja lasām rakstu par Semantiskās Latvijas [5] vīziju, tad kā trīs galvenie aspekti tika minēti – grafveida vaicājuma rīks, kas ļautu viegli atlasīt datus, sistēma, kurā ielādēt ontoloģiju, lai labotu tās datus un kas nodrošina iespēju datus saglabāt semantiskā formātā, pēdējais aspekts ir centralizēts repozitorijs, kurā glabājas ontoloģijas.

Ja pirmos divus aspektus aplūkojām, tad atliek pēdējais vīzijas posms, kur mums ir nepieciešams ieskicēt ontoloģiju repozitoriju. Ko mēs no tā sagaidām un kādus iespējamos ieguvumus tas var sniegt.

Ja zinām, ka mums ir pieejama OBIS sistēma, tad redzam ieguvumus, ko mums dod ontoloģiju repozitorijs un tajā esošās ontoloģijas, jo tas paver iespēju mūsu domēnā atrast jau izveidotu ontoloģiju, ko atliek ielādēt OBIS sistēmā, lai varētu sākt strādāt ar mums izveidotu informatīvo sistēmu.

Varam aplūkot organizācijas W3 saiti[48], kur ir minēti populārāki ontoloģiju repozitoriji. Populārākais no šiem ir BioPortal [20], kas apkopo medicīnas domēnā esošās ontoloģijas. Turklāt, ontoloģiju repozitoriju ietvaru ir iespējams adaptēt savām vajadzībām.

BioPortal repozitorijs satur dažādas ontoloģijas par medicīnu un tas ir labākais šāds apkopojums, taču sfēra ir ļoti ierobežota, kā arī šajā sfērā īsti nav nepieciešamas vienkāršas un ātras informatīvās sistēmas.

Taču, ja aplūkojam piedāvāto repozitorija ietvara funkcionalitāti, tad tā ir krietni labāka par citiem sarakstā minētajiem ontoloģiju repozitorijiem, piemēram "DERI Vocabularies"[49]. Kas tikai attēlo dažādas ontoloģijas, bet nepiedāvā gandrīz nekādus servisus, kas tās grupētu vai ļautu tajās meklēt nepieciešamo informāciju.

BioPortal repozitorija ietvars piedāvā ontoloģijas meklēšanas iespējas, kā arī dažādu statistiku par ontoloģijām. Ir iespējama ļoti vienkārša arī atsauce uz projektiem, kas to izmanto, bet tā nesatur nekādu papildus specifisku informāciju. Ir arī iebūvēta ontoloģijas vizualizācija.

Taču tajā pašā laikā tam ir ierobežotas iespējas strukturēt ontoloģijas pa dažādiem domēniem. Dotajā ietvarā tiek ievietotas tikai viena domēna ontoloģijas. Piemēram, vēlamies

[48] http://www.w3.org/wiki/Ontology_repositories
[49] http://vocab.deri.ie/

vienā sadaļā ievietot ar servisiem saistītās ontoloģijas, kamēr citā domēnā ievietot dažādas ontoloģijas, kas reprezentē ražošanu.

Viens no priekšlikumiem ir izstrādāt specifiskas domēnu meta ontoloģijas, kas ļautu vienkāršāk atrast nepieciešamās ontoloģiju konstrukcijas domēna ietvaros.

Ja izvirzām noteikumu, ka ontoloģija tiek izmantota, lai aizpildītu kādas konkrētas sistēmas datus un padarītu tos pieejamus caur SPARQL piekļuves punktu, tad vēlamies atzīmēt konkrētās SPARQL piekļuves punktu adreses, lai varētu veikt plašāku datu apkopojumu.

Lasītājs var gūt priekšstatu par ontoloģiju repozitoriju svarīgumu sasaistē ar to datiem rakstā [48].

6.2. Ontoloģiju izgūšana no SPARQL piekļuves punkta

Tika izstrādāta iniciatīva[50], lai veidotu datu shēmas aprakstus un norādītu, kādi dati būs pieejai SPARQL piekļuves punktā. Taču šobrīd šī iniciatīva ir apstājusies. Vietā gan ir izveidota jauna W3C interešu grupa[51], kuras mērķis ir izveidot pilnīgākus aprakstus, lai tuvinātu RDF datu publicētājus un to lietotājus.

Taču jāatzīmē, ka šī ir tikai interešu grupa, kas nav nonākusi līdz oficiālam standartam. Arī pēdējais publiski pieejamais ierosinājums ir vairāk nekā 2 gadus vecs, kas norāda, ka šajā virzienā netiek veltīta pietiekama aktivitāte.

Taču ņemot vērā, ka neviens no šiem risinājumiem praktiski vēl nav īstenots, bet vaicājumus par esošajiem SPARQL piekļuves punktu datiem mums vajag sastādīt, tad tika izveidots alternatīvs risinājums, kā izgūt ontoloģiju no SPARQL piekļuves punkta un tur esošajiem datiem.

Shēmas izgūšanas piemēram izmantojām SPARQL piekļuves punktu[52], kas satur datus par dažādām Semantiskā tīmekļa konferencēm, cilvēkiem, kas tajā ir piedalījušies. Turpmāk tekstā uz to atsauksimies kā uz testa piekļuves punktu (angliski tas tiek saukts par *Dog Food* piekļuves punktu). Vaicājumu kopa tika pārbaudīta 2010. gada decembrī.

Varam aplūkot 7.2. Attēlā esošo ontoloģijas metamodeli. Mēs esam sarakstījuši SPARQL vaicājumu kopu, kas izgūst datus no piekļuves punkta un secīgi aizpilda ontoloģijas metamodeli. Ideālā gadījumā ontoloģija, kurai tiek veidoti dati, arī tiek ierakstīta SPARQL piekļuves punktā.

[50] http://www.w3.org/wiki/SparqlEndpointDescription
[51] http://www.w3.org/TR/void/
[52] http://data.semanticweb.org/sparql

Un ar vaicājumu, kas ir attēlots 6.1. Attēlā būtu iespējams izgūt visas OWL klases, kas ir SPARQL piekļuves punktā.

```
select distinct ?classuri ?classname where {
?classuri a <http://www.w3.org/2002/07/owl#Class>.
OPTIONAL {
?classuri <http://www.w3.org/2000/01/rdf-schema#label> ?classname}}
```

6.1. att. ***SPARQL vaicājums OWL klašu atlasei***

Ja 6.1. attēlā dotais vaicājums tiek palaists testa piekļuves punktā, tad tiek atgrieztas 44 dažādas klases. Taču ir arī papildus norāde, ka testa piekļuves punktā esošā ontoloģija nav pilnīgi definēta, jo apmēram vienai trešdaļai no atrastajām klasēm nav piekārtots tās vārds ar "*rdfs label*". Labā prakse nosaka, ka klases vārds tiek uzdots šādā veidā.

Papildus 6.1. attēlā redzamajā vaicājumā atlasījām tās klases, kas ir definētas kā *owl:class*, lai arī atsevišķos piekļuves punktos var tikt izmantota RDFS datu shēmas definēšana un tās var tikt definētas ar *rdfs:class*.

Taču varam modificēt klašu atlasīšanas vaicājumu. Ja 6.1. attēlā dotais vaicājums atlasīja visas piekļuves punktā definētās klases, arī tās, kurām nav neviena instance, tad vēlamies atlasīt klases, kuras apvieno instanču kopas.

Līdz ar to, varam atlasīt instances un atrast kādām klasēm tās pieder. Tad šīs klases arī būs tās, kas ir jāizmanto datu attēlošanas procesā. Vaicājums, kas atlasa objektu un to reprezentējošās klases ir attēlots 6.2. attēlā.

Ja tas tiek veikts testa piekļuves punktā, tad atrod vairāk nekā 100 dažādas klases. Tas ir divtik vairāk nekā ar OWL konstrukcijām definētās.

Taču šeit gan jāatzīmē, ka no šīm klasēm izšķirojam tehniskās klases, piemēram, klases, kurām vārda telpa ir OWL. Atbilstoši OWL klase, kas apvieno visus atribūtus arī ir klase, bet šāda klase mums neinteresē priekšā teikšanas kontekstā.

```
select distinct ?classuri ?classname
where {?object a ?classuri.
OPTIONAL {?classuri <http://www.w3.org/2000/01/rdf-schema#label>
?classname}}
```

6.2. att. ***Vaicājums visu netukšo klašu atrašanai***

Piemērs skaidri norāda, ka eksistējošos SPARQL piekļuves punktos nevar paļauties uz OWL definēto shēmu, jo tā tikai daļēji pārklāj pieejamos datus. Tāpēc ir jāraksta sarežģītāki vaicājumi. Tāpat mēs gūstam arī priekšstatu, ka daudz būtiskāk ir pārklāt ontoloģijas shēmas daļu, kurai ir pieejami reāli dati, citādāk lietotājs sastādīs vaicājumu, kurš vienkārši nekad neko nevarēs atgriezt, jo kāda no vaicājumā izmantotajām klasēm nesaturēs nevienu instanci. Un mēs esam

ieinteresēti atrast tikai to ontoloģijas shēmas daļu, kas palīdzēs lietotājam sastādīt saturīgus vaicājumus.

Izmantojot 6.2. attēlā nodemonstrēto vaicājumu iegūstam nepieciešamo rezultātu, ka tas pārklāj visas SPARQL piekļuves punktā esošās klases, kurām eksistē instances un pār kurām lietotāji varēs sastādīt saturīgus vaicājumus.

Nākamajā solī atlasīsim visas asociācijas (*object properties)*, kas saista klašu instances savā starpā. SPARQL vaicājums, ar kura palīdzību to var izdarīt, ir attēlots 6.3. attēlā.

Ņemot vērā, ka esam atraduši visas klases, tad šajā vaicājumā vajag atrast visas asociācijas, kā arī atzīmēt, no kuras klases asociācija iziet un kurā klasē asociācija ieiet.

```
select distinct ?class1 ?assoc ?class2 ?assocName
where {?object1 a ?class1. ?object2 a ?class2. ?object1 ?assoc ?object2.
OPTIONAL {?assoc <http://www.w3.org/2000/01/rdf-schema#label> ?assocName}} }
```

6.3. **att.** ***Vaicājums asociāciju atrašanai***

Atbilstoši varam atrast divus objektus, kas pieder kādām klasēm. Visas relācijas, kas tos savienos atbilstoši, būs asociācijas.

Ņemot vērā, ka šis vaicājums atrod visas asociācijas uzreiz, tad tas var izpildīties ļoti ilgi. Taču tas uzskatāmi parāda ideju, ko vēlamies izpildīt.

Praktiskos piemēros efektīvāk ir realizēt iteratīvus vaicājumus, ka tiek izvēlēta viena noteikta klase kā *class1* un šīs klases URI tiek uzstādīts kā konstante. Tālāk tiek atlasītas visas asociācijas, kas no šīs klases iziet. Šāds vaicājuma ātrdarbības izmaiņas var būt būtiskas, jo dažiem SPARQL piekļuves punktiem ir uzstādīti laika ierobežojumi atbildes rēķināšanai un 6.3. attēlā dotais vaicājums var neizpildīties dotajā laika ierobežojumā.

Tāpēc praktiskās situācijās tiek izvēlēts risinājums – izpildīt vairākus atsevišķus vaicājumus, kuru kopējā laika summa ir lielāka. Tas ar lielāku varbūtību garantē korektu shēmas izgūšanu, jo katrs mazais vaicājums iekļausies laika ierobežojumā, kamēr viens lielais vaicājums to var neizdarīt.

Turklāt, darbības laikā veicam pārbaudi, vai šī asociācija jau pieder kādai no dotās klases virsklasēm un tā vēlreiz nav jāiekļauj modelī.

Nākamajā solī atlasām visus dotās ontoloģijas atribūtus (*data property*). To var izdarīt ar vaicājumu, kas ir dots 6.4. attēlā.

Ņemot vērā, ka visas klases ir atlasītas, tad mums vajag atlasīt atribūtu, tā tipu un piesaistīt atribūtu pie atbilstošajām klasēm, kuru instances satur šo atribūtu.

```
select  distinct ?attr ?attrType ?attrName ?class
 where {
?object a ?class.
?object ?attr ?value.
OPTIONAL {
?attr <http://www.w3.org/2000/01/rdf-schema#range>  ?attrType }
OPTIONAL {?attr <http://www.w3.org/2000/01/rdf-schema#label> ?attrName}
OPTIONAL {?value a ?class2 }
FILTER (!bound(?class2)&&!isBlank(?value)) }
```

6.4. att. ***Vaicājums atribūtu atlasei.***

Atribūtu atlase norit līdzīgi asociāciju atlasei, taču ir jāuzstāda papildus nosacījumi. Ja asociācija saista divas instances, tad atribūts ir piesaistīts instancei, bet otrs piesaistītais objekts nedrīkst būt klases instance.

Turklāt, piesaistīto vērtību nedrīkst ierobežot tikai ar pieejamajām *string, integer* un citām definētajām vērtībām, jo atribūta vērtība var būt arī URI, kas norāda piemēram kādu mājaslapas adresi. (Lai arī korekti to var īstenot adresi norādot kā *string* datu vērtību, tomēr praksē bieži tas tiek ignorēts, piemēram, testa piekļuves punktā FOAF[53] klases "Persona" mājaslapu adreses tika norādītas kā URI.)

Arī papildu atrodam atribūtu tipu, ja tas ir definēts ar RDFS līdzekļiem. Ja atribūta tipu nav iespējams noteikt, tad piešķiram tam *string* vērtību.

Tomēr, atsevišķās situācijās nav iespējams viennozīmīgi noteikt atribūtus, jo retos gadījumos var būt definēta sasaista starp vienu objektu, kam ir norādīta klase, bet otram objektam šī klases norāde nav pielikta. (Jau minētajā mājaslapu piemērā, tās teorētiski var definēt arī ar klasi, kurai ir papildus īpašības par mājaslapu.)

Noslēdzošā izspiegošanas daļa ir saistīta ar apakšklašu attiecību atrašanu starp klasēm. Vaicājumu, kas tiek izmantots šīs īpašības noteikšanai var aplūkot 6.5. attēlā. Algoritmiski gan tā tiek izpildīta pēc klašu izspiegošanas, lai rezultātus varētu izmantot asociāciju un atribūtu izspiegošanā.

Taču metodoloģiski to ietvērām kā pēdējo, jo to noteikšana nedaudz atšķiras no visiem iepriekš aprakstītajiem vaicājumiem.

```
select distinct ?subClass ?superClass
where { ?subClass <http://www.w3.org/2000/01/rdf-schema#subClassOf>
?superClass}
```

6.5.att. ***Vaicājums apakšklašu attiecības noteikšanai.***

Ja gadījumos ar klašu, asociāciju un atribūtu atlasi bija iespējams izmantot SPARQL piekļuves punktā esošos datus, tad apakšklašu gadījumā to ir ļoti grūti izdarīt. Apakšklases attiecību

[53] http://www.foaf-project.org/

definīcija tiek lietota tikai ontoloģijas shēmas definēšanas līmenī un datos šī definīcija tiešā veidā nav atrodama.

Ja aplūkojam izvēlētā testa piekļuves punkta shēmu, tad ir viegli pateikt, ka klase "Doktorantu sekcijas vadītājs*"* (*Doctoral Consortium Chair*) būtu jābūt apakšklasei "Amats" (*Role)*.

Lai arī izmantojot 6.5. attēlā esošo vaicājumu mums ir iespējams atrast dokumentētās apakšklašu attiecības, tomēr ir ļoti grūti izveidot algoritmus, kas varētu to izsecināt no pieejamajiem datiem.

Šajā jomā ir izstrādāts darbs [55], kas izmantojot pieejamo informāciju par līdzīgajiem atribūtiem un relācijām mēģina noteikt apakšklašu attiecības. Lai arī ar to palīdzību daļēji var uzlabot risinājumu, tomēr tas dod tikai varbūtisku rezultātu.

Tomēr izsecinot dažādās īpašības no datiem, nav tik būtiski, ka mums nav pieejamas virsklašu un apakšklašu attiecības. Tās mums ļauj pārskatāmāk attēlot ontoloģiju. Tās ir būtiskas, lai izmantotu OWL secināšanas iespējas. Taču secināšanas iespējas var izmantot tikai SPARQL piekļuves punkta serveris. Ja tur nav tieši pateiktas apakšklašu attiecības, tad tās arī nekādi netiks izmantotas.

Galvenā priekšrocība vaicājuma sastādīšanā no virsklašu attiecībām ir tā, ka tās piedāvā konkrētai klasei atribūtu un asociāciju vērtības, kas ir virsklasēm. Ja izgūstam shēmu tikai no RDFS un OWL izteikumiem, tad tas ir būtiski.

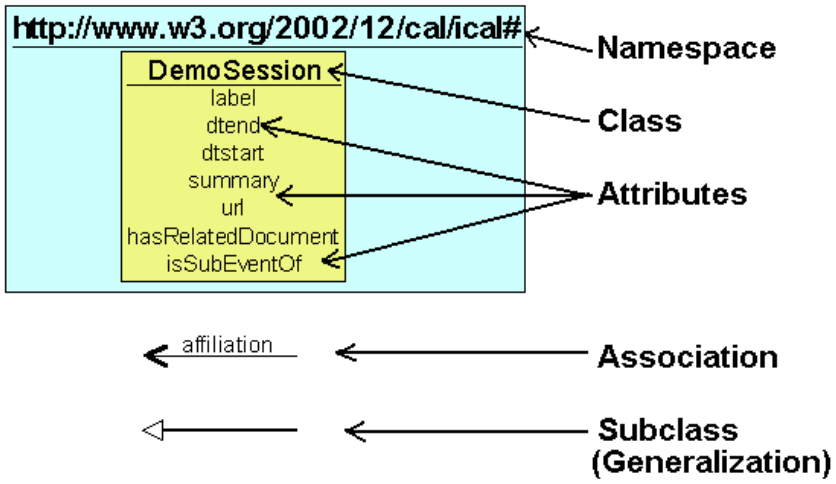
Taču mēs shēmu konstruējam no pieejamajiem datiem. Ja virsklasei tiek pieliekts atribūts "vārds", tad visdrīzāk arī loģiskai apakšklasei arī parādīsies šis atribūts. Tātad veicot secināšanu no datiem, ontoloģijas līmenī nav būtiski vai ir definētas apakšklašu attiecība, datos tāpat parādīsies tās vērtības, kas reāli tiek mantotas.

6.2.1. Izgūtās ontoloģijas attēlošana

Kad esam ieguvuši ontoloģiju no piekļuves punkta, tad varam to attēlot lietotājam, lai tas iegūtu labāku priekšstatu par piekļuves punktā pieejamajiem datiem.

Ontoloģijas vizualizācija tika realizēta tikai pirmajā ViziQuer rīka eksperimentālajā versijā. Izmantojām medicīnas projektā [18] iegūtās idejas par ontoloģijas attēlošanu.

Attēlā 6.6. var aplūkot grafiskos elementus, ko ieviesām, lai attēlotu ontoloģiju. Tie pārklāj izspiegošanas rezultātā iegūtos datus, kā arī papildus pievienojam iespēju tos grupēt pa vārda telpām, ņemot vērā, ka vārda telpa apvieno līdzīgus elementus.

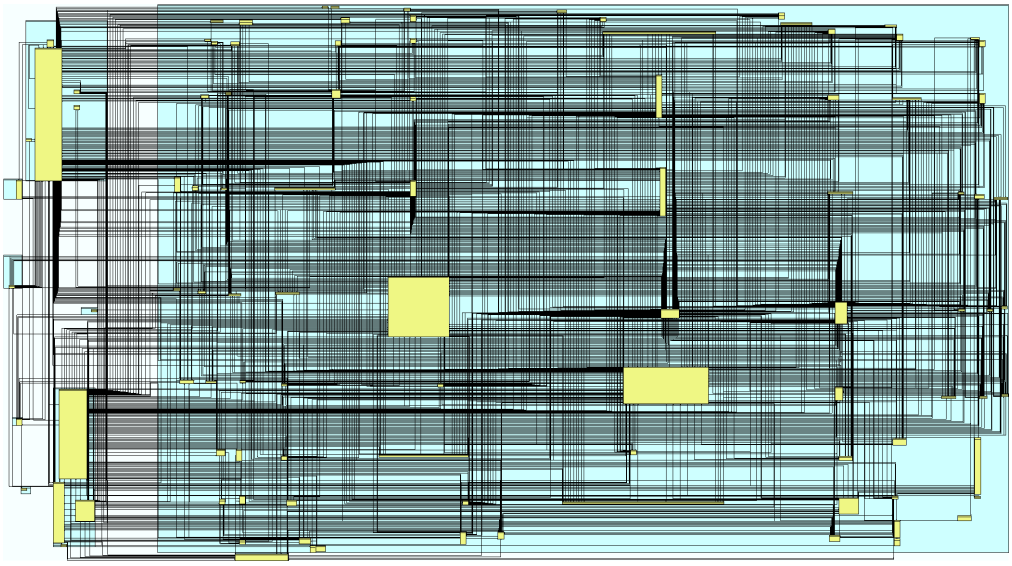


6.6. att. ***Vizualizācijas grafiskie elementi.***

Izstrādājām 3 dažādas ontoloģijas attēlošanas iespējas, ko pārbaudījām uz testa piekļuves punkta datiem.

Pirmkārt, attēlojām ontoloģiju izmantojot visu pieejamo informāciju. Sadalījām to pa vārda telpām un attēlojām visas asociācijas, kas tika izspiegotas. Ņemot vērā, ka ontoloģijā nebija pilnībā definētas apakšklašu attiecības, tad ontoloģijas attēlojums sanāk ļoti nepārskatāms un tajā ir praktiski neiespējami orientēties.

Varam aplūkot no testa SPARQL piekļuves kopas izgūto ontoloģiju, kas ir attēlota pilnībā 6.7. attēlā. Attēls vairāk ir shematisks, lai nodemonstrētu, ka tajā ir ļoti grūti orientēties.



6.7.att. ***Testa SPARQL piekļuves punkta ontoloģijas pilnais attēlojums.***

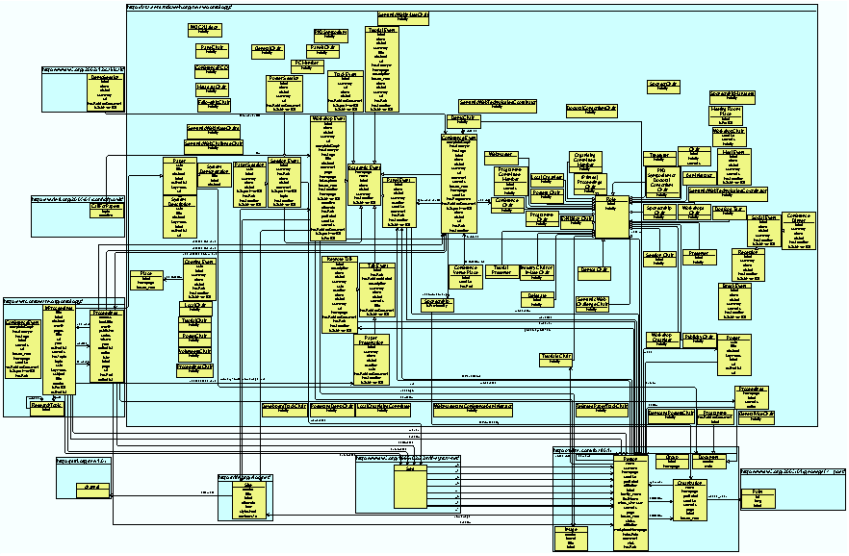
Attēls 6.7. arī parāda virsklašu attiecību definēšanas nozīmību, jo savādāk asociāciju skaits var pieaugt pārāk strauji un padarīt attēlu nepārskatāmu. Šādā gadījumā labāk ir izvēlēties Protege piedāvāto kokveida struktūras attēlojumu, kur asociācijas ir iespējams redzēt katrai klasei atsevišķi.

Ja aplūkojam formālo OWL definīciju, tad no tās izriet, ja viena un tā pati asociācija iziet no vairākām klasēm, tad tā iziet no šo klašu apvienojuma (citiem vārdiem sakot, virsklases). Ņemot vērā, ka fiktīvas virsklases ieviest vaicājumā būtu ļoti grūti, jo virsklašu klasifikācija sevī ietver ne tikai sintaksi, bet arī semantiku un ir gadījumi, kad ir būtiski tās attēlot tā, ka veidojas virsklašu struktūra ar dziļumu lielāku par 2, tad izvēlamies risinājumu attēlot katru asociāciju tieši vienu reizi.

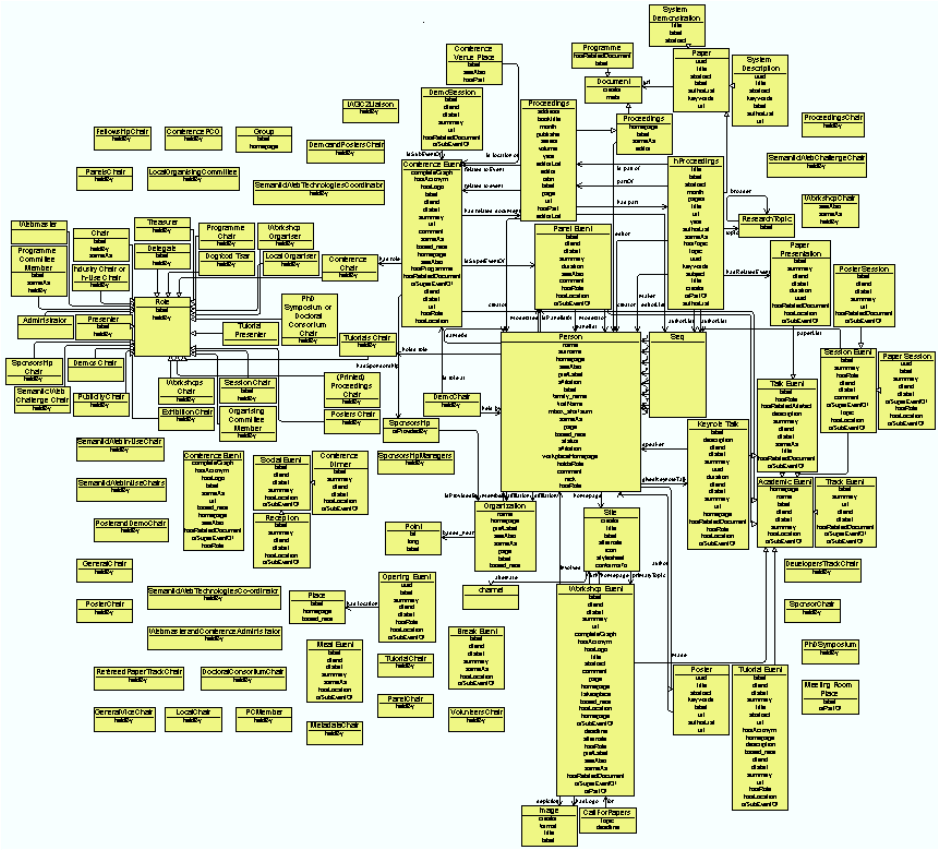
Izvēlamies vienu no daudzajiem asociācijas attēlojumiem un to arī parādām. Citus asociācijas attēlojumus ignorējam ontoloģijas attēlojumā, taču saglabājam tos vaicājuma sastādīšanas laikā, lai nodrošinātu priekšā teikšanas iespējas.

Piedāvātā risinājuma izskatu varam aplūkot 6.8. Attēlā, kur ir parādīta tieši tā pati testa SPARQL piekļuves punkta ontoloģija ar alternatīvo shēmas vizualizāciju.

Piedāvājam uz ontoloģijas shēmu paskatīties arī bez iedalījuma pa vārda telpām. No tā nedaudz mainās ontoloģijas izkārtojums, jo vairs nav jāgrupē vispirms klases pa vārda telpām. Taču tāpat saglabājam, ka katra asociācija tiks attēlota tikai vienu reizi, lai izvairītos, ka bilde atkal paliek pārāk sarežģīta. Izveidoto testa SPARQL piekļuves punkta ontoloģijas attēlojumu bez vārda telpām varam aplūkot 6.9. attēlā.



6.8. att. ***Testa SPARQL piekļuves punkta uzlabotā ontoloģijas vizualizācija.***



6.9. att. ***Testa SPARQL piekļuves punkta ontoloģijas attēlojums bez vārda telpām.***

Svarīgākais secinājums ir ļoti vienkāršs – ja SPARQL piekļuves punkta pārvaldnieki nerūpējas par datu kvalitāti un necenšas tos izveidot atbilstoši ontoloģijai, kuru arī publicē, tad izvilktā ontoloģija no SPARQL piekļuves punkta var palīdzēt vaicājuma sastādīšanā, bet tās pārskatāmība būs ierobežota un lietotājam būs jāpavada kāds laika sprīdis, lai labāk saprastu piekļuves punktā esošos datus.

6.3. Ontoloģijas repozitorija piemēra izklāsts balstoties uz medicīnas datiem

Veidojot datu translāciju medicīnas projekta ietvaros [18] izvēlējāmies risinājumu – dažādo ontoloģiju datus apvienot un ievietot vienā SPARQL repozitorijā. Primāri tas bija saistīts ar to, ka tehnoloģiskās iespējas atlasīt datus no dažādiem repozitorijiem un tos apvienot nebija pieejamas.

Ņemot vērā, ka galvenais projekta uzstādījums bija datu integrācija no dažādām ontoloģijām, tad izvēlējāmies naivo pieeju apvienot datus un likt uzsvaru uz vaicājumu tālāku vaicāšanu no viena resursa.

Taču, apvienojot visas ontoloģijas vienā vietā, parādījās nākamā problēma, ka medicīnas zinātnieki spēj saprast katru atsevišķo ontoloģiju, taču pārskatīt un saprast visu ontoloģiju apvienojumu bija ļoti sarežģīti. Vienīgais risinājums, sastādot vaicājumus, bija navigēšanas iespēja, ka lietotājs redzēja jau ievadītai klasei piesaistītās klases.

Lai uzskatāmāk varētu nodalīt šo datu apvienošanu arī vaicājuma sastādīšanas laikā, tad naivajam risinājumam ir nepieciešami uzlabojumi. Kā ontoloģiju glabāšanai dažādos repozitorijos, kur katrā repozitorijā būtu tieši viena ontoloģija ar tai piekārtotiem datiem, tā arī vaicājumu sastādīšanai, kas šobrīd ir centrēts uz darbu ar vienu datu avotu.

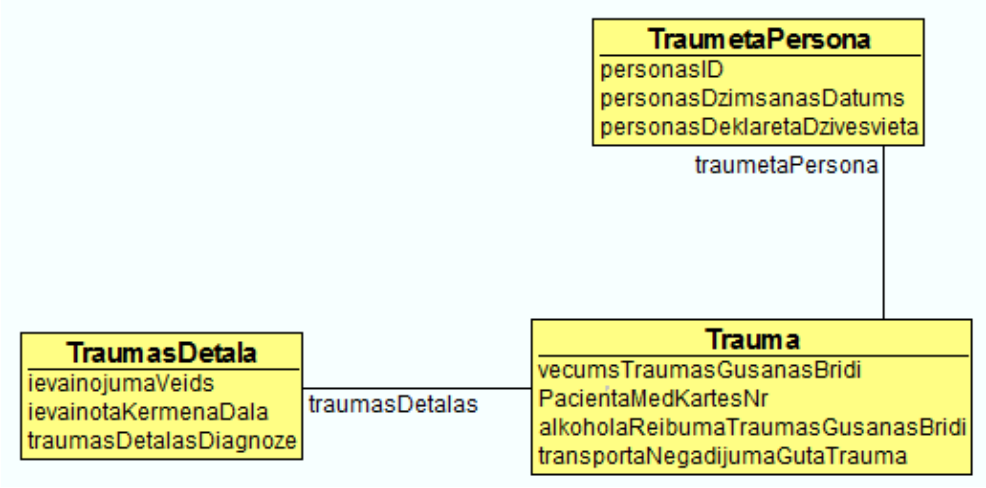
Ja aplūkojam šobrīd pieejamos tehniskos risinājumus, tad SPARQL 1.1 specifikācija pieļauj iespēju apvienot vaicājumos datus no vairākiem datu avotiem, jo ir ieviesti apvienojošie vaicājumi (*SPARQL federate queries*). Tas ļauj rakstīt vienu vaicājumu, kas apvieno datus no dažādiem repozitorijiem un tos prezentē lietotājam.

6.3.1. Medicīnas domēna ontoloģiju raksturojums

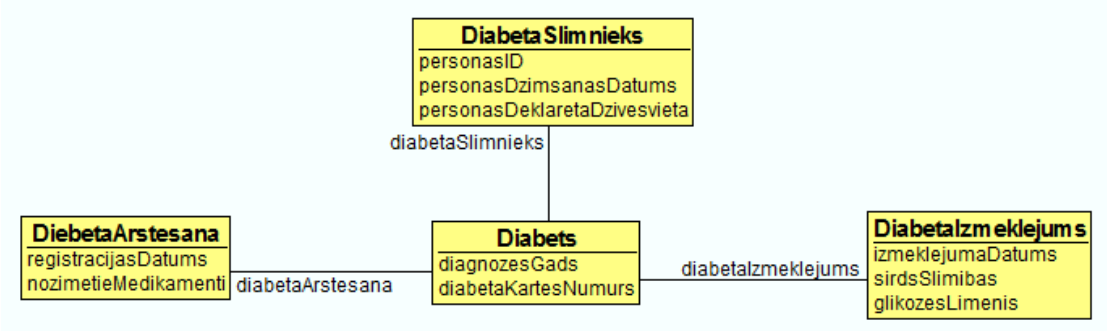
Medicīnas ontoloģiju piemērā mums bija pieejamas 6 ontoloģijas, kurām bija līdzīga struktūra. Ja apskatāmies no medicīnas zinātnieka skatupunkta, tad viņš redz tikai vienu ontoloģiju, kas ir ielādēta SPARQL piekļuves punktā. Kā arī 12 ontoloģijas, kas ir izdrukātas un noliktas pie sienas (12. ontoloģija reprezentē sistēmas kodolu).

Attēlos 6.10., 6.11. un 6.12. ir vēža saslimšanas, traumu un diabēta ontoloģiju centrālās daļas. Uz centrālajām daļām esam izveidojuši sistēmas kodolu un tā centrālo daļu, kas ir attēlota 6.13. attēlā.

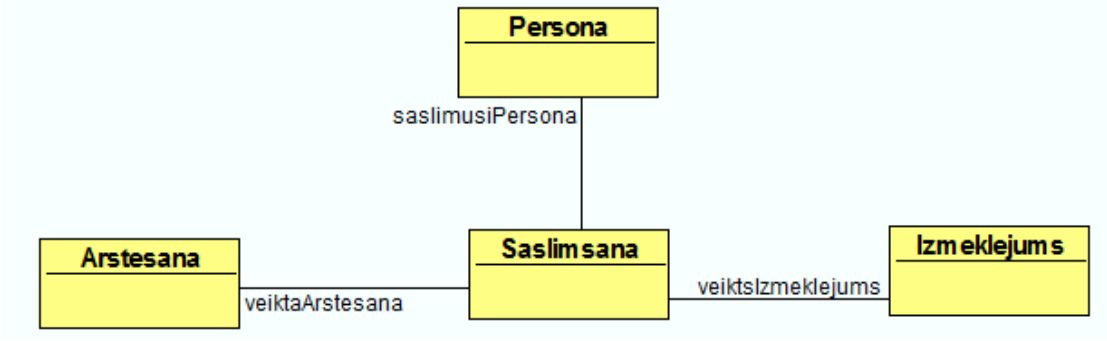
6.10. att. ***Vēža saslimšanas ontoloģijas centrālā daļa***



6.11. att. ***Traumu ontoloģijas centrālā daļa***



6.12. att. ***Diabēta ontoloģijas centrālā daļa***



6.13. att. ***Izveidotā centrālā daļa, kas saista slimību ontoloģijas***

Sistēmas centrālā ontoloģija nebija pieejama un tieši redzama PREDA datu sistēmā, kad sākām darbu. Bija pieejama sistēmas centrālā daļa, kas attēloja datus, kas visām ontoloģijām ir kopīgi, taču tā neparādīja kopējo slimības ontoloģiju struktūru. Taču, veidojot atsevišķās ontoloģijas, mēs izvilkām kopējo daļu un izveidojām to par sistēmas centrālo ontoloģiju.

Sistēmas centrālā daļa kalpo trīs galvenajiem lietojumiem – pirmkārt, ir iespējams veidot sasaisti starp ontoloģijām, jo parāda kopīgās daļas, mūsu piemērā tās ir personas, ārsti,

ārstniecības iestādes. Otrkārt, tie parāda ontoloģijas dizaina shēmas, kā veidot jaunu ontoloģiju[54]. Treškārt, sistēmas centrālā daļa iekļauj arī to atslēgas klasi, kas nosaka ontoloģijas būtību un pēc kuras medicīnas zinātnieki vēlēsies tās sameklēt un apvienot.

Ja visas ontoloģijas ir apvienotas vienā SPARQL piekļuves punktā, tad mums nav jāpievērš uzmanība, kur glabājas dati. Taču realitātē katras ontoloģijas ieviesējs glabās datus savā SPARQL piekļuves punktā. Turklāt, vienai ontoloģijai varētu atbilst vairāki SPARQL piekļuves punkti, kurā tiek glabāti tās dati.

Mēs vēlamies šos SPARQL piekļuves punktus atrast pēc kādām noteiktām īpašībām, lai šo SPARQL piekļuves punktu datus varētu apvienot veicot dažādus datu vaicājumus. Ir nepieciešama īpašība, pēc kuras meklēsim datus.

Mūsu gadījumā ontoloģijām ir pieejama centrālā klase - saslimšanas klases. Līdz ar to mēs varam veidot ontoloģijas repozitoriju, kur tiek izveidota arī meta ontoloģija, kurā var meklēt tās ontoloģijas, kas mums ir saistošas. Meta ontoloģijas instancēs tiks glabāti dati par dažādām saslimšanas ontoloģijām, kā arī SPARQL piekļuves punktiem, kuros būs iespējams atrast to datus.

Attēlā 6.13. var aplūkot modeli repozitorija modeli, kas būtu piemērots medicīnas domēnam. Medicīnas zinātnieki varētu atrast, piemēram, vairākas traumu ontoloģijas un veikt datu izpēti pār lielāku datu kopu, vai arī atrast traumu ontoloģiju (un tās datus), atrast diabēta ontoloģiju (un tās datus) un veikt izpēti kā diabēta slimība ietekmē iegūtās traumas un vai starp iegūtajām traumām un diabētu ir kāda saistība.

6.4. Ontoloģiju repozitoriju perspektīva

Ja aplūkojam Semantiskās Latvijas [5] vīziju, tad piemēroti ontoloģiju repozitoriji ir trūkstošais posms, lai padarītu semantisko tīmekli plašāk pieejamu.

Turklāt, ontoloģiju repozitorijs pildītu vienlaicīgi divas svarīgas īpašības, tas ne tikai piedāvātu ontoloģijas, uz kuru pamata varētu izveidot OBIS informatīvās sistēmas, bet tajā pašā laikā piedāvātu saites uz izveidoto informatīvo sistēmu datiem, lai lietotājs ātrāk un ērtāk spētu iegūt atbildi uz savu pieprasījumu, kā tas ir aprakstīts sākotnējā Semantiskā tīmekļa vīzijā [6].

Pirmkārt, izmantojot ontoloģiju repozitorijā piedāvātās ontoloģijas, būtu iespējams veidot datus, kas ir savā starpā salīdzināmi un sasaistāmi. Otrkārt, pēc pieejamajām datu saitēm būtu iespējams veikt arī vaicājumus pār datiem un to apkopojumu.

[54] Varam atsaukties uz ontoloģiju repozitoriju http://ontologydesignpatterns.org/, kura mērķis ir apkopot dažādās shēmas, kas tiek izmantotas ontoloģiju veidošanai.

Lai arī ontoloģiju repozitoriji diktētu uzlabojumus esošajās vaicājumu valodās, piemēram, ViziQuer vaicājumu valoda, kas ir aprakstīta nodaļā 7., ir balstīta uz vienu datu avotu un tajā pieejamo ontoloģiju. Ontoloģiju repozitoriju gadījumā tā būtu jāpapildina un jāpielāgo ar divām īpašībām – vienas ontoloģijas datu atlase vairākos SPARQL piekļuves punktos, vai arī vairāku dažādu ontoloģiju apvienošana vaicājumā, kur katra datus glabā savā SPARQL piekļuves punktā.

Taču ir pieejami vairāki citi eksperimentāli izstrādājumi, kas risina problēmu, lai apvienotu datus no vairākiem avotiem, piemēram, DERI pipes [49].

7. VIZIQUER REALIZĀCIJA

Nodaļās 2. un 3. lasītājs var iepazīties ar vaicājumu valodas konstrukcijām. Jebkurš šīs konstrukcijas varētu zīmēt ar roku uz papīra un tās veidotu saprātīgu vaicājumu. Taču mūsu gadījumā galvenā vaicājuma būtība ir tā, ka tas tālāk tiks translēts uz SPARQL vaicājumu un nosūtīts izpildei.

Līdz ar to veidojot rīku, kas atbalsta vaicājumu sastādīšanu, ir divas būtiskās lietas – kā izveidot to tā, lai lietotājiem būtu ērti un ātri sastādīt vaicājumus. Kā arī kā panākt to, ka vaicājumi ir saturīgi un tie atgriezīs arī rezultātus.

Ērtumu nodrošina rīks, kas atbalsta vaicājumu sastādīšanu, lai lietotājiem būtu iespējas veikt darbības, lai sastādītu viņu biezāk lietotos vaicājumus pietiekami ātri. Kamēr vaicājumu saturīgumu nodrošina tas, ka vaicājumi tiek sastādīti izmantojot ontoloģiju shēmu.

7.1. Vaicājuma rīka pārskats

Esam izveidojuši vaicājumu valodas atbalsošu rīka prototipu ViziQuer, atbalsta vaicājumu sastādīšanu grafiski, tos translē uz SPARQL, kā arī ir iespējams tos nosūtīt izpildei uz SPARQL piekļuves punktu.

Raksts par pirmā rīka prototipu ir pieejams [50]. Rīka pirmais prototips atbalsta mazāku SPARQL vaicājumu valodas pārklājumu, kamēr rīka otrais prototips pārklāj nodaļā 3. aprakstīto ViziQuer lite un ir pieejam ViziQuer mājaslapā[55].

Vaicājuma rīka konstrukcija iedalās divās galvenajās fāzēs. Pirmā ir ontoloģijas izgūšana no SPARQL piekļuves punkta vai tās ielāde no OWL datnes. Ontoloģijas ielasīšana tiek aprakstīta šīs nodaļas 3. apakšnodaļā.

Ontoloģijas izgūšana no SPARQL piekļuves punkta var būt visai laikietilpīga, jo ir nepieciešams nosūtīt tiešsaistē pietiekami daudzus vaicājumus, lai izspiegotu SPARQL piekļuves punktā esošo datu shēmu, kas reprezentē tur esošos datus.

Arī ontoloģijas lielums, ar kuru ir paredzēts strādāt ViziQuer rīkā ir ieteicams līdz 100 OWL klasēm, jo lielāku skaitu būs grūti pārskatīt un ērti izvēlēties klases, no kurām sastādīt grafiskus vaicājumus.

Piemēram, DBpedia SPARQL piekļuves punktā[56] ir vairāk nekā 100 000 dažādas OWL klases, kas apgrūtina pilnas ontoloģijas izmantošanu grafiskā vaicājuma sastādīšanai. Priekšā

[55] http://viziquer.lumii.lv/
[56] http://DBpedia.org/sparql

teikšanas procesā ir pārāk grūti pārskatīt tik daudz un dažādas klases, kas apgrūtina pareizās klases izvēli un kas procesu padara pārāk laikietilpīgu.

Rīka pirmajā prototipa versijā tika izstrādāta arī iespēja aplūkot izgūto OWL ontoloģiju grafiskā formātā. Tā bija būtiska iespēja, jo ontoloģija tika izgūta no SPARQL piekļuves punkta, tāpēc lietotājam ne vienmēr pa rokai bija ontoloģijas grafisks attēls. Lai gūtu labāku priekšstatu par apgabalu un datu kopu, kas glabājas šajā SPARQL piekļuves punktā un pār kura datiem lietotājs sastādīts grafisko vaicājumu, lietotājs varēja iepazīties ar grafisko shēmu ViziQuer rīkā.

ViziQuer lite rīka versijā tika piedāvāta papildus iespēja ielādēt ontoloģiju no OWL datnes, kuru var apskatīt piemēram rīkā OWLGrEd [51]. Tāpēc iespēja grafiski aplūkot ontoloģiju netika realizēta. Taču ontoloģijas struktūru var aplūkot kokveida formātā (kā tas ir rīkā Protege[57]) sastādot vaicājumu.

Rīka otra īpašība ir tālāk nodrošināt SPARQL vaicājumu sastādīšanu atbilstoši ielasītai ontoloģijai.

Lietotājs veido grafisku vaicājumu, vaicājuma sastādīšanas laikā priekšā teikšanai tiek nodrošināta ielasītā ontoloģija. Atliek vien izveidot kastes un tās savā starpā savienot, lai iegūtu SPARQL vaicājumu, kas reprezentē to, ko lietotājs vēlējās atlasīt.

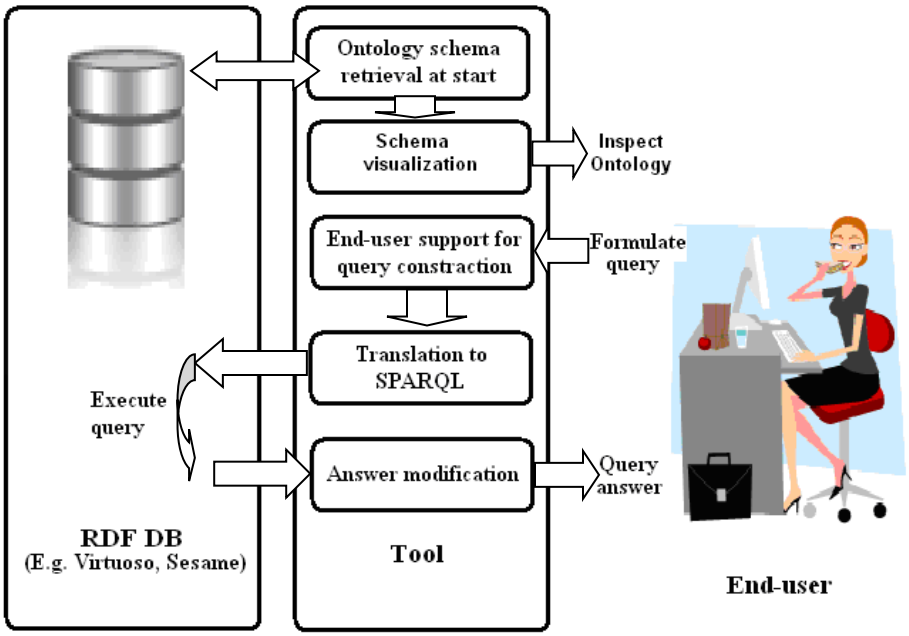
ViziQuer pirmajā prototipā tika īstenotas divas paradigmas vaicājumu konstruēšanā – pirmkārt, bija iespējams ievietot vaicājumā vajadzīgās klases un savā starpā savienot ar asociāciju – rīks piedāvā iespējamos ceļus kā izvēlētās klases savā starpā var tikt savienotas, turklāt, ceļā ir iespējamas papildus klases, kas arī tiek pievienotas vaicājumam. Līdzīgu risinājumu piedāvā rīks RelFind [52].

Otrkārt, tiek īstenota daļēji *fasetu* (*facated*) paradigma vaicājuma sastādīšanā. Mums ir pieejamās klases un saites tālāk. Lietotājs atbilstoši tās var pārskatīt un, balstoties uz tām, virzās tālāk un papildina vaicājumu. Lai arī ir atšķirīgā nianse, ka *fasetu* meklēšana parasti centrā noliek vienas klases datus, un pievienotās klases parasti kalpo kā nosacījumu. Kamēr mūsu risinājumā mēs veidojam grafisku vaicājuma pārskatu. Līdzīgu pieeju nodrošina piemēram Faceted Graph [54] vai Longwell [36].

ViziQuer lite versijā realizēta gan tika tikai *fasetu* meklēšanas princips, jo pārrunās ar lietotājiem medicīnas projektā [18] tika novērots, ka tas ir daudz biežāk lietots risinājums, kā arī gadījumi, kad ir jāsavieno divas ļoti attālas klases savā starpā ir ļoti reti. Galvenokārt tāpēc, ka vaicājums tiek veidots par saistītiem datiem, nevis tiek meklēti ceļi, kā šie dati var būt saistīti.

[57] http://protege.stanford.edu/

Attēlā 7.1. varam aplūkot vaicājuma rīka uzbūvi un secību kādā ir paredzēta lietotāja darbības, lai sastādītu grafiskus vaicājumus.



7.1. att. ***ViziQuer rīka darbības principu shematisks attēlojums***

7.2. Ontoloģijas shēma

Kā tika aprakstīts iepriekšējā apakšnodaļā, tad ļoti būtisku lomu vaicājumu sastādīšanā spēlē tieši ontoloģija. Tā nosaka – kā dati ir grupēti un kādus sakarīgus jautājumus būtu iespējams noformulēt.

Teorētiski SPARQL piekļuves punktā esošie dati var būt negrupēti, bet tikai liela kaudze ar RDF trijniekiem, kur nav tādas apvienojošās daļas, datu shēmas, kas tos saistītu kopā.

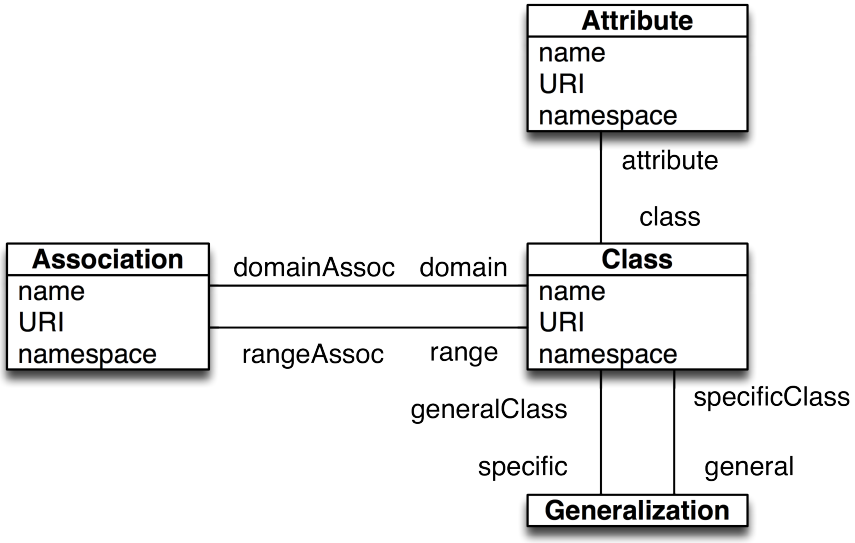
Taču sastādīt vispārīgus vaicājumus par šādām datu kopā arī ir neiespējami. Vaicājumi varētu atrast paternus, kas saturētu noteiktas instances, taču paterni nebūtu pietiekami vispārīgi, lai šādu SPARQL vaicājumu rezultāti būtu saistoši tālākām darbībām.

Taču praksē ļoti bieži izmanto OWL ontoloģijas, lai dati tiktu grupēti pa klasēm. Tālāk tiek definētas atribūtu kopas, kas piemīt klases objektiem, kā arī relācijas, kas saista klases objektus ar citu klašu objektiem.

Pirmkārt, tas ļauj iegūt strukturētus datus, pie kā tik ļoti esam pieraduši ikdienā un kas tiek izmantoti ļoti daudzās informatīvās sistēmās. Otrkārt, šī struktūra ļauj veikt OWL iebūvēto līdzekļu izmantošanu, piemēram, secināšanu vai nosacījumu pārbaudi, kas parasti tiek definēti klašu līmenī, jo objektu līmenī būtu ļoti grūti pierakstīt tik daudz un dažādus nosacījumus.

Mūsu rakstā [32] ir detalizēti aprakstīta ontoloģijas shēmas daļa, ko ņemam vērā konstruējot grafiskus vaicājumus, lai nodrošinātu priekšā teikšanas iespējas lietotājiem. Priekšā teikšanas daļai tiek izmantoti ontoloģiju valodas OWL apakškopas RDFS divi galvenie punkti – klases un to atribūti (*properties*, kas sevī ietver kā klasiskos atribūtus, tā asociācijas un apakšklases attiecības).

Attēlā 7.2. varam aplūkot RDFS metamodeli. Ontoloģijas, kas tiek ielādēta, lai palīdzētu sastādīt vaicājumus, tiek ielādēta kā šī metamodeļa instanču kopa.



7.2. att. ***Ontoloģiju metamodelis***

7.3. Ontoloģijas ielāde rīkā

Nodaļā 6.2. ir aprakstīta iespēja ontoloģijas shēmu izspiegot no SPARQL piekļuves punkta. Tas noder gadījumos, kad ontoloģija nav pieejama, taču vaicājumus vēlamies sastādīt pār datiem, kas atrodas SPARQL piekļuves punktā.

Ja ontoloģija ir pieejam, tad to var ielasīt no datnes, kas aprakstīts apakšnodaļā 7.3.1.

7.3.1. Ontoloģijas ielasīšana no datnes

ViziQuer lite versijā esam īstenojuši iespēju ontoloģiju ielasīt arī no OWL datnes. Tas ir īstenots tāpēc, ka bieži lietotājiem ir pieejama pati ontoloģija un ir nepieciešams sastādīt SPARQL vaicājumus, ko vēlāk nosūtīt izpildei vai iekļaut sistēmas realizācijā.

Tas ļauj strādāt ar vaicājumu sastādīšanu, par pamatu ņemot tikai ontoloģiju, un nav nepieciešams strādājošs SPARQL piekļuves punkts, kur glabāt ontoloģiju un datus, lai darbotos ar SPARQL vaicājumu sastādīšanu.

Tehniskās realizācijas pamatā ir izmantots OWL API[58], kas nodrošina ontoloģijas ielādi no OWL datnes un saskarni darbam ar ielādēto ontoloģiju. Tas atvieglo procesu, jo nav manuāli jāparsē ontoloģija, bet tiek nodrošināta augstāka līmeņa saskarne.

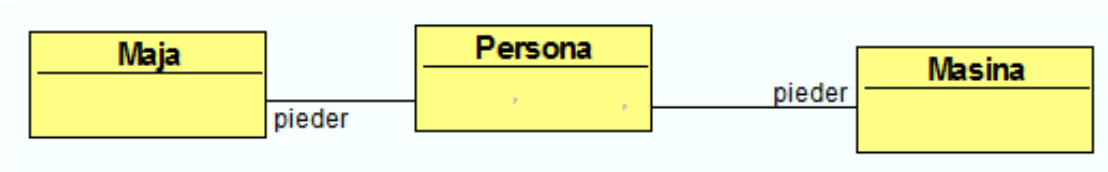
Algoritms ontoloģijas apstrādei faktiski ir tāds pats, kāds tas ir ontoloģijas izgūšanai no SPARQL piekļuves punkta. Kas secīgi ir aprakstīts zemāk:

- Ielādēt ontoloģiju no datnes OWL API;
- Izveidot sarakstu ar OWL klasēm;
- Izveidot sarakstu ar atribūtiem, norādīt atribūtiem tipu un klasi, kurai atribūts pieder;
- Izveidot sarakstu ar asociācijām, norādīt sākuma klasi un beigu klasi;
- Izveidot apakšklašu sarakstu un norādīt, kura klase ir virsklase, bet kura apakšklase.

Izveidotā Java programma, kas nodrošina ontoloģijas ielādi un funkcijas, kas nodrošina katra soļa izpildi.

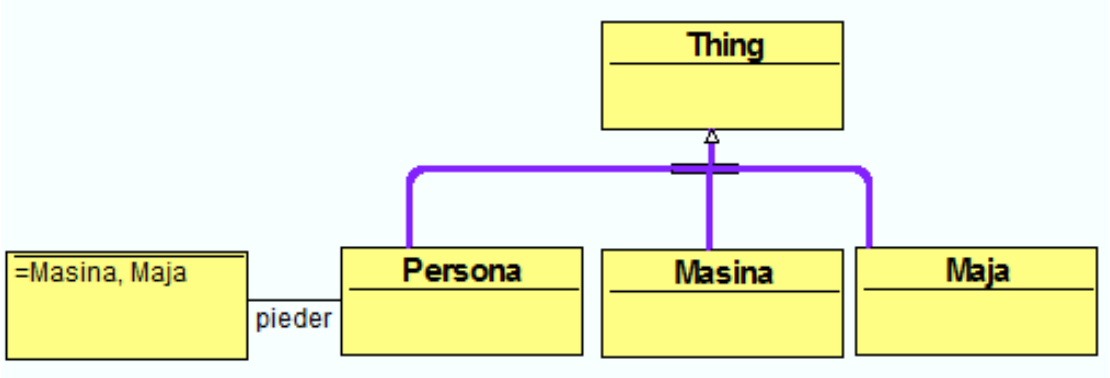
Arī šajā gadījumā ir problēmas ar metodoloģiju, kā precīzi apstrādāt ontoloģijas asociācijas ar vienādiem vārdiem. Ja izvēlamies OWLGrEd[59] rīku ontoloģijas veidošanai un ja gadījumā divās vietās lietojam asociāciju ar vienādu vārdu, tad tiek izveidotas virtuālas virsklases, kuras ir kā asociācijas sākuma klašu apvienojums un asociācijas beigu klašu apvienojums.

Tas korekti atspoguļo OWL semantiku, kas norāda, ka asociācijai ir tikai viens sākums un vienas beigas. Tālāk šis sākums un beigas tiek izteikti ar loģisku formulu. Šajā gadījumā tas ir visu sākuma klašu apvienojums. Lai arī tas nav intuitīvi, ka tiek redaktorā uzzīmēta bilde, kur viena un tā pati asociācija tiek izveidota divās vietās, bet pēc ontoloģijas saglabāšanas un ielādes pa jaunu ir parādījusies jauna klase, kas ir minēto divu klašu šķēlums un ar kuru tagad ir saistīta ontoloģija. Lai labāk izprastu situāciju, tad aplūkosim attēlu 7.3. un 7.4., kur ir attēlota minētā situācija.



7.3. att. ***Ontoloģijas piemērs.***

[58] http://owlapi.sourceforge.net/
[59] http://owlgred.lumii.lv/



7.4. att. ***Ontoloģijas piemērs pēc atkārtotas ielādes.***

Tāpēc apstrādes laikā veicam papildus darbību, ka likvidējam anonīmo virsklasi (lai arī formāli tā nav dota kā virsklase, bet tāda ir konstrukcijas būtība) un izveidojam atkal pa jaunu divas atsevišķās asociācijas.

Asociācijas vai atribūti, kas ir piesaistīti anonīmām klasēm, kuru konstrukcija ir izveidota ar kādām citādākām loģiskām darbībām kā klašu apvienojums, tiek ignorēti.

7.4. ViziQuer pirmā prototipa realizācija

7.4.1. Izmantotās tehnoloģijas – GrTP platforma un L0 transformācijas

Rīks tika izstrādāts pilnībā izmantojot MDE izstrādes principus. Pamatā ir grafisks modelis, kuram ir interpretators, kas to attēlo par kā rīku, kurā tiek attēloti grafi, ar kuriem ir iespējams veikt dažādas interaktīvas darbības. Tālāk ir transformācijas, kas apstrādā lietotāja darbības, izmaina modeli un attiecīgi paziņo interpretatoram, ka ir veiktas izmaiņas.

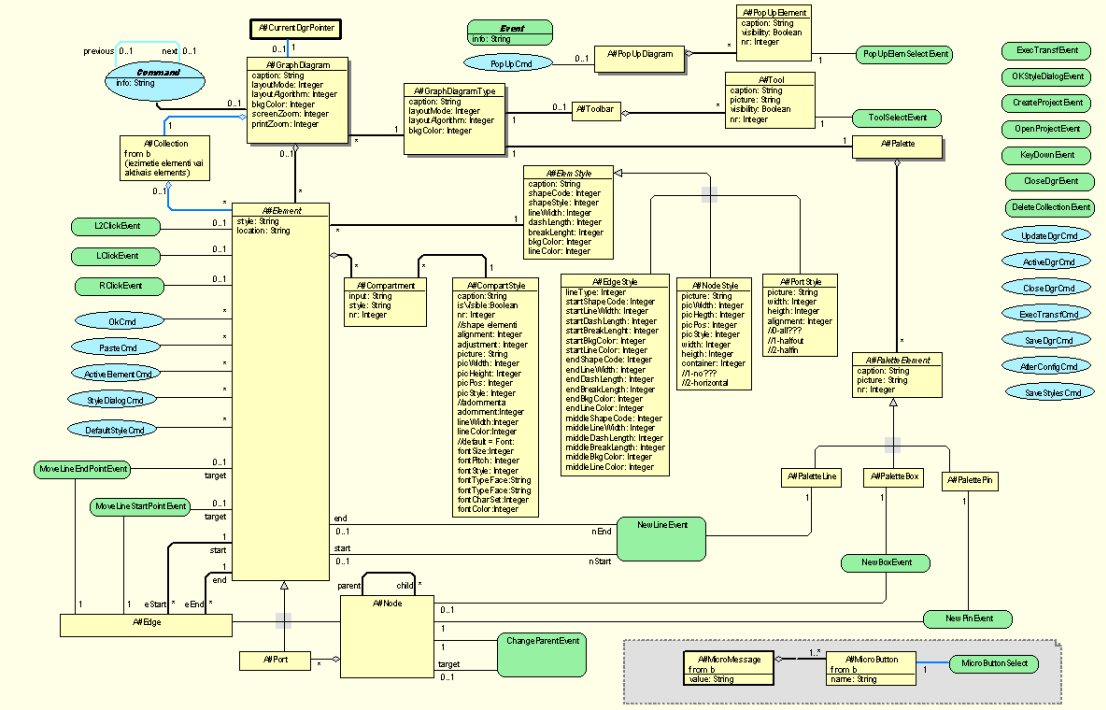
Rīks tika veidots izmantojot LU MII izstrādāto GrTP platformu [34], kas nodrošina grafveida rīku izstrādi. Rīka pamatā ir metamodelis, kas ir attēlots 7.5. attēlā.

Kā arī formu izvēlnes, ar kurām darbojas lietotājs, tiek veidotas izmantojot dialogu dzinēju, kas ir aprakstīt rakstā [58].

Detalizētu priekšstatu par šīs metodoloģijas tehniskajām niansēm, iespējām un metodoloģiksu pielietojumu var izlasīt S.Kozloviča disertācijā [56].

Tā darbības pamatā ir piecas galvenās sastāvdaļas. Pirmkārt, grafa attēlošanas mehānisms – virsotnes (*node*) un škautņu (*edge*) klases, kas pieder kādai diagrammai (*graphDiagram*). Šo klašu instances veido grafa attēlu rīkā. Papildus ir arī kompartment (*compartment*) klases instances, kas nodrošina teksta attēlošanu pie virsotnēm un škautnēm. Grafa attēlošanas

mehānismu papildina arī stilu daļa, kas nosaka, kā izskatīsies katra virsotne, škautne un pievienotie teksti.



7.5. att. ***GrTP metamodelis.***

Otrkārt, ir lietotāja izsaukti notikumi, piemēram, labās peles klikšķis vai jaunas līnijas novilkšana. Katram no notikumiem tiek piešķirta semantika, kur tiek ņemts vērā arī notikuma konteksts (piemēram, starp kādiem elementiem tiek vilkta līnija). Katra notikuma apstrādei tiek izveidota transformācija valodā L0 [57], kas veic manipulācijas ar modeļa datiem.

Treškārt, ir komandas, ar kurām izveidotās transformācijas var paziņot par izmaiņām izveidotajā rīkā, piemēram, paziņot, ka kāda virsotne vairāk nav jārāda. Tā ir būtiska sastāvdaļa, lai līdz galam īstenotu lietotāju veiktās darbības un par to paziņotu rīku būves platformai par tām, lai varētu tās atspoguļot arī grafiski.

Ceturtkārt, ir mūsu[60] sarakstītās transformācijas, kas nodrošina rīka loģisko darbību. Atbilstoši, tās saņem lietotāja veikto notikumu, nosaka darbības semantiku, veic izmaiņas metamodelī un vajadzības gadījumā par to paziņo rīka būves platformai.

Lai arī shematiski tas izklausās ne pārāk sarežģīti, tomēr praksē tas nozīmē ļoti dažādu darbību veikšanu. Tiek aprakstīta pilnībā lietotāja veikto darbību loģika, kur ir jāizsecina, kādas darbības tālāk ir jāveic. Tāpat ir jāveido grafiskie elementi un jāpiesaista to izskats. Jāveido grafiskās logu saskarnes, kur lietotāji veic dažādas darbības.

[60] Transformācijas pilnībā ir sarakstījis tikai promocijas darba autors Mārtiņš Zviedris.

Piektkārt, rīka papildinātie, loģiskie modeļi. Mūsu gadījumā gan ir tikai viens papildus modelis – ontoloģiju shēmas kodēšanai. Atbilstoši veidojot jaunu grafisku elementu tiek veidota arī sasaiste ar ontoloģijas elementu. Vēlāk šī sasaiste tiek izmantota, lai ģenerētu SPARQL vaicājumu no grafikas elementiem.

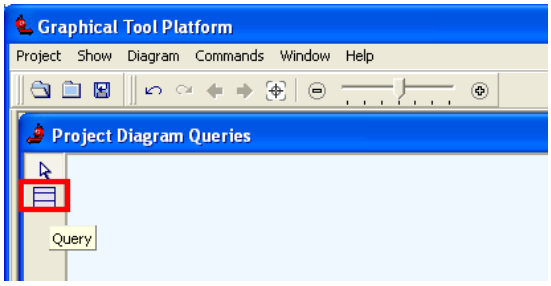
Lai arī varētu ieviest papildus modeli vaicājumiem, tomēr tas netiek darīts, jo būtu jāuztur paralēli starp grafisko attēlojumu un loģisko attēlojumu. Tāpēc mēs izvēlamies vaicājumu ģenerēšanai izmantot grafikā uzzīmētās sintakses stilu daļu.

7.4.2. ViziQuer prototipa grafiskā saskarne

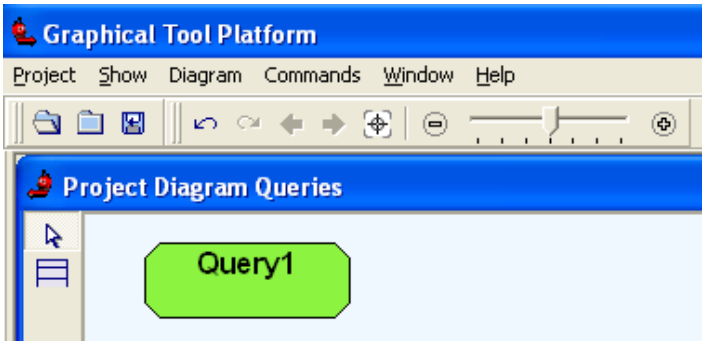
Darbs ar ViziQuer sākas izveidojot jaunu projektu vai atverot jau iepriekš izveidotu.

Veidojot jaunu projektu, ir jānorāda SPARQL piekļuves punkta adrese. Atbilstoši ViziQuer rīks pieslēgsies dotajam SPARQL piekļuves punktam un ar metodi, kas aprakstīta nodaļā 6.2., tiks izgūta ontoloģija no SPARQL piekļuves punkta.

Tālākā rīka granularitāte ir vaicājumi. Projekta logs, kurā veidot vaicājumus ir attēlots 7.6. attēlā un 7.7. attēlā var aplūkot kā izskatās izveidots vaicājums.



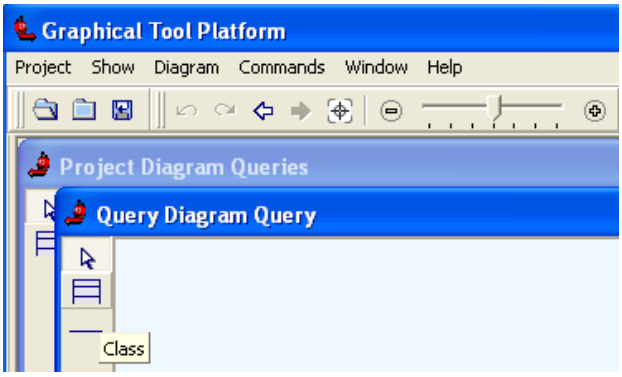
7.6. att. ***Projekta vaicājumu logs.***



7.7. att. ***Vaicājumu pārskats projektā.***

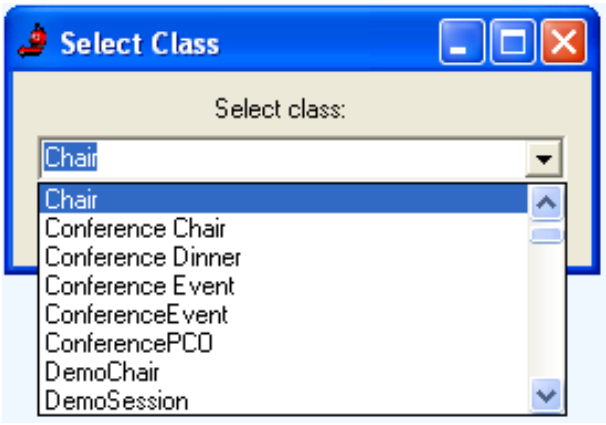
Katram projekta aizmetnī izveidotajam vaicājumam atbilst grafiski izveidots, detalizēts vaicājums, kam ir iespējams uzģenerēt SPARQL vaicājumu un to nosūtīt izpildei SPARQL piekļuves punktam.

ViziQuer pirmā prototipa versijā ir pieejami tikai divi grafiskie elementi – vaicājuma atlases klase un relācija, kas savieno divas klases. 7.8. Attēlā var aplūkot kā izskatās vaicājuma logs ar diviem paletes elementiem.

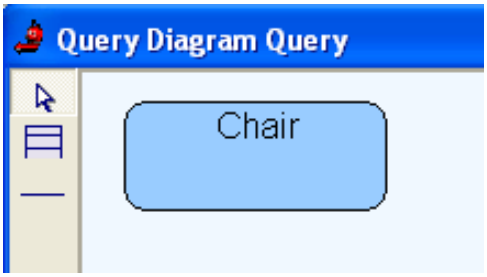


7.8. att. ***ViziQuer prototipa paletes elementi.***

Ar klases elementu var veidot jaunus klases elementus. To darot, tiks piedāvāts lietotājam saraksts ar visām SPARQL piekļuves punktā esošajām klasēm, no kurām lietotājs var izvēlēties to, kas viņu interesē. Detalizēti izvēles procesu un rezultātu var aplūkot 7.9. un 7.10. attēlā.



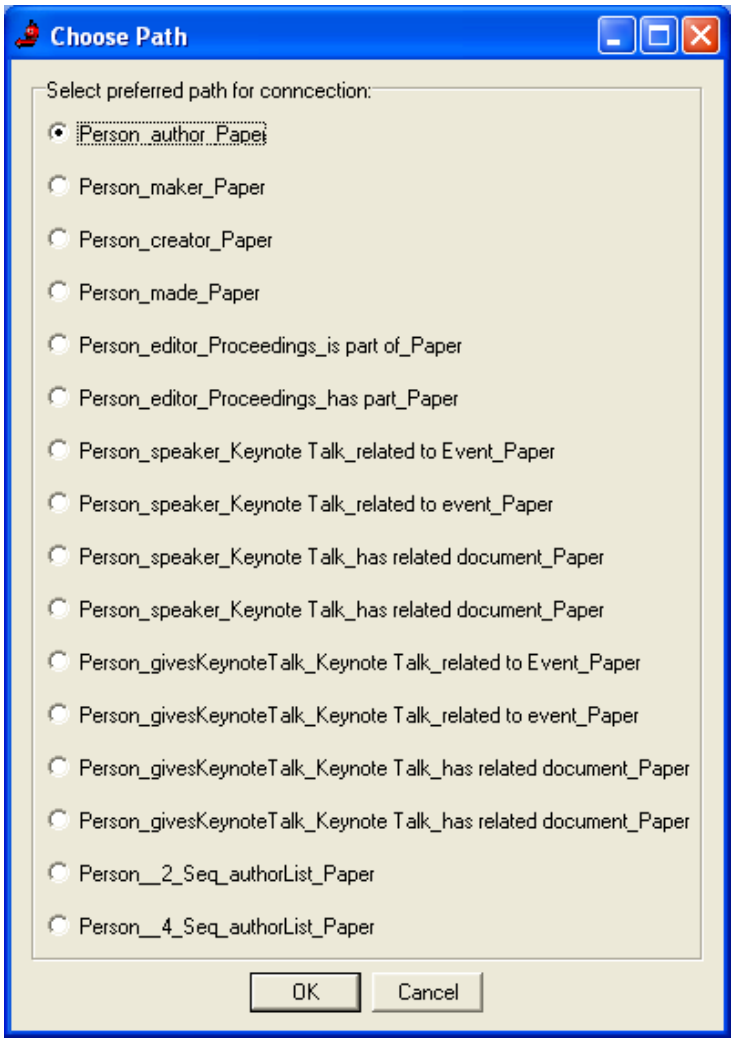
7.9. att. ***Vaicājuma klases pievienošanas izvēlne.***



7.10.att. ***Pievienota vaicājuma klase.***

Pievienot saites uz jauniem elementiem ir iespējams divos dažādos veidos. Pirmkārt, izvēloties uzlecošo izvēlni uz kādas no klasēm mums ir iespējams izvēlēties klasi un saiti, ko pievienojam vaicājumam.

Otrkārt, ja vaicājumā ir pievienotas divas klases, tad mums ir iespējams starp tām novilkt saiti un uzzināt veidus, kā šīs divas klases savā starpā var savienot. Atbilstoši piemēru, kā izskatās izvēlne, kurā izvēlēties vajadzīgo klašu savienojumu var apskatīt 7.11. attēlā.



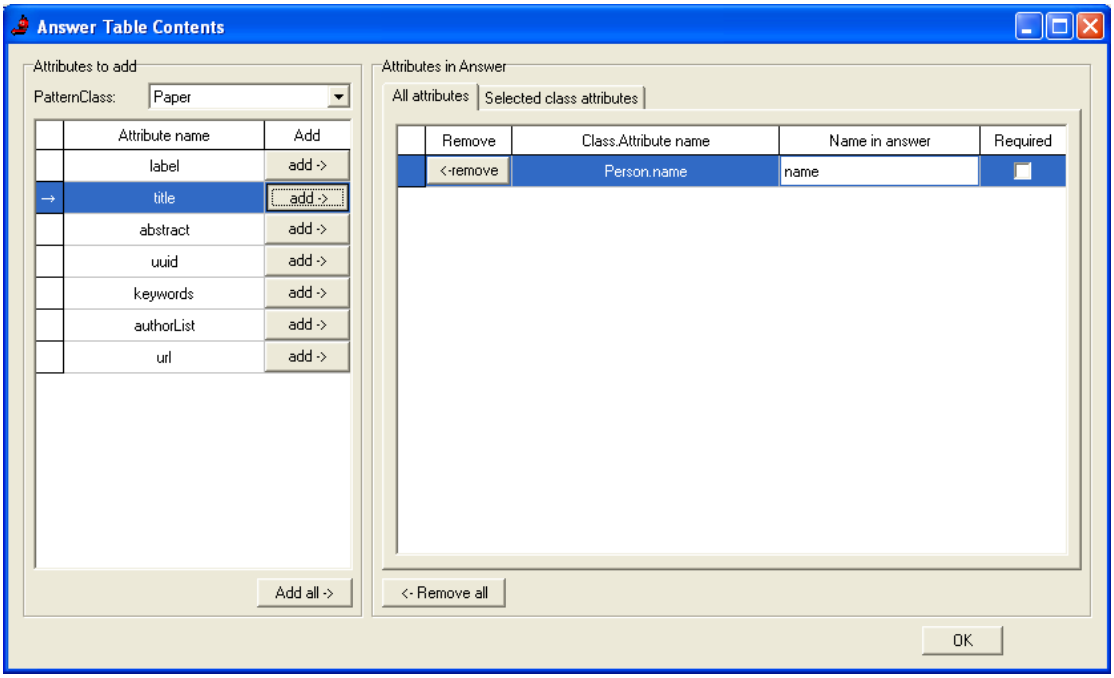
7.11. att. ***Vaicājuma ceļa pievienošanas izvēlne.***

Lai atrastu iespējamos ceļus, kas savieno divas izvēlētās klases tiek lietots algoritms, kas ir līdzīgs īsākā ceļa noteikšanai, taču tajā ir veikta papildus modifikācija.

Atbilstoši tiek sanumurētas virsotnes ontoloģijā, ka attālumā 0 ir sākuma virsotne un ka pārējām ir ierakstīts attālums līdz sākuma virsotnei. Veidojot ceļu, no beigu virsotni uz sākuma virsotni, iespējams iet vai nu uz mazāku attālumu, vai arī tādu pašu līmeni, bet ne vairāk kā vienu reizi.

Primāri novērojums, kāpēc šādu papildus īpašību bija nepieciešams īstenot un nederēja tikai īsākais ceļš, ir tāds, ka atsevišķās reizēs, kad klases ir attālumā 1, tomēr gribam tās savienot caur kādu citu klasi. Piemēram, medicīnas ontoloģiju gadījumā savienot ārstu ar saslimšanu varam vēlēties vai nu pa tiešo, vai arī pa vidu parādīt ārstēšanas metodes, kas tika pielietotas.

Tāpat vaicājumā ir iespēja definēt atribūtus no izvēlētajām klasēm, ko vēlamies redzēt atbildē. Atbilstoši ir parādīts logs 7.12. attēlā, kurā definēt atbildes atribūtus un to nosaukumus.



7.12. att. ***ViziQuer atbildes atribūtu definēšana.***

Noslēgumā lietotājs uzģenerē no izveidotā vaicājuma SPARQL vaicājumu, nosūta to izpildei uz SPARQL piekļuves punktu, no kura tika izvilkta ontoloģijas shēma. Atbildē lietotājs saņem pirmo 200 rindu rezultātu, kā arī iespēju eksportēt pilno atbildi Excel formātā.

7.5. ViziQuer lite prototipa realizācija

7.5.1. Izmantotās tehnoloģijas – TDA konfigurators un lQuery transformācijas

Rīka ViziQuer lite prototipa realizācija noritēja arī GrTP platformā. Taču atšķirībā no rīka pirmās versijas, tika izmantotas tehnoloģijas, kas piedāvā augstāka līmeņa arhitektūru un rīka stabilitāti.

Viena no problēmām rīka pirmajā versijā bija saistīta ar L0 transformācijām, kas atsevišķu kļūdu gadījumā izvadīja lietotājam paziņojumu par kļūdu, bet tajā pašā rīks beidza darbu.

Turklāt, atsevišķos gadījumos kļūdas radīja L0 transformāciju īpatnības, piemēram, mainīgais, kurš tiek nodots caur divām apakš funkcijām, pazaudē savu sākotnējo vērtību, ja funkcija to saņem un nodot tālāk, taču neveic nekādas darbības.

Taču tajā pašā laikā transformāciju ideoloģija nodrošināja iespēju ātri veidot grafveida rīka prototipu. Tāpēc tika izvēlēta cita transformāciju valoda – lQuery [53].

Viena no transformācijas valodas priekšrocībām ir tā, ka tā tiek interpretēta darbības gaitā, tāpēc tā ļauj turpināt darbu ar rīku, pat ja ir notikusi kļūdu. Tāpat tika arī ekonomēts izstrādes laiks uz transformāciju kompilēšanu, kas L0 gadījumā varēja pieaugt līdz vairākām minūtēm.

Otrs tehnoloģiskais uzlabojums ir saistīts ar LU MII izstrādāto konfiguratoru[61]. Ja GrTP platformas gadījumā mums bija jāizstrādā arī domēna specifiskā valoda, kā arī pašiem jāveic visas nepieciešamās domēna specifiskās valodas pārbaudes, tad konfigurators piedāvā papildus interfeisu darbam ar domēna specifiskām valodām.

Konfigurators piedāvā iespēju grafiski izveidot domēna specifiskās valodas specifikāciju, kā arī nodrošina servisus valodas elementu pareizības pārbaudēm. Piemēram, saite var savienot tikai divas vaicājuma objektu atlases klases savā starpā, bet tā nevar savienot komentāru ar kādu citu elementu.

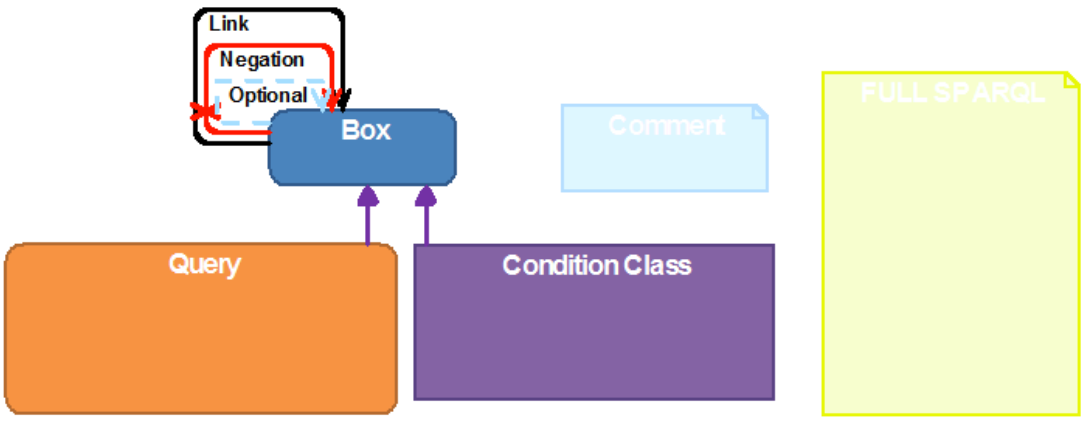
Konfigurators piedāvā arī papildus interfeisa funkcionalitāti. Piemēram, kopēšanas funkcijas grafiskajiem elementiem, kā arī iespēju konfigurācijas logā pārskatāmi atzīmēt kādas transformācijas ir jāizsauc pie kādām lietotāja darbībām.

Varam aplūkot ViziQuer lite grafisko konfigurāciju 7.13. attēlā. Ņemot vērā, ka semantiski centrālā vaicājuma klase un parastā vaicājuma klase ir ekvivalentas, taču atšķiras to sintakse, tad tiek ieviesta abstrakta virsklase, kas atvieglo saišu savilkšanas iespējas, kad viens no saites galiem ir parastā vaicājumu klase vai centrālā vaicājuma klase.
Tālāk varam aplūkot ViziQuer lite centrālās vaicājuma klases lietotāju izsaukto darbību apstrādes konfigurācijas logu, kas ir aplūkojams 7.14. attēlā.

Visa rīka funkcionalitātes izpilde un SPARQL vaicājumu ģenerēšana no uzzīmētā grafiskā vaicājumu, tāpat ir jāizpilda manuāli sarakstītām transformācijām. Taču ir vieglāk pārskatīt kādas transformācijas un kādos gadījumos tiks izsauktas.

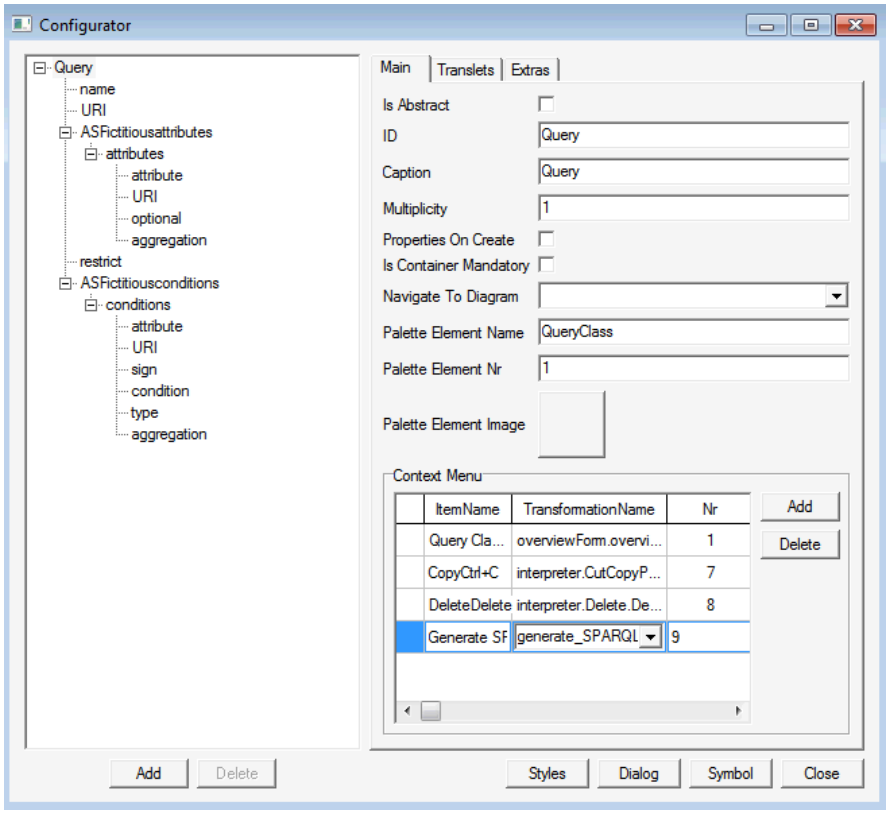
[61] http://grade2.lumii.lv/index.html

Taču tajā pašā laikā ir jāraksta arī pietiekami sarežģītas transformācijas. Piemēram, pievienojot jaunu saiti un jaunu vaicājuma elementu no izvēlnes formas, ir jāizveido šie grafiskie elementi un jāatjauno grafiskā diagramma.



7.13. att. ***ViziQuer lite grafiskā konfigurācija***

Tāpēc konfigurators atvieglo pārskatīt kur un kādas transformācijas ir jāraksta, taču tajā pašā laikā tāpat ir jāsaraksta visas rīka specifiskās transformācijas, kas nav triviāls uzdevums.



7.14. att. ***ViziQuer centrālās klases konfigurācija***

7.5.2. ViziQuer lite ģenerēto SPARQL vaicājumu regulārā izteiksme

Lai atvieglotu ViziQuer lite vaicājumu ģenerēšanu, kā arī vēlāk pārbaudi, tad tika izstrādāta regulārā izteiksme, pēc kuras tiek veikta SPARQL vaicājuma ģenerēšana, kas attēlota 7.15 attēlā.

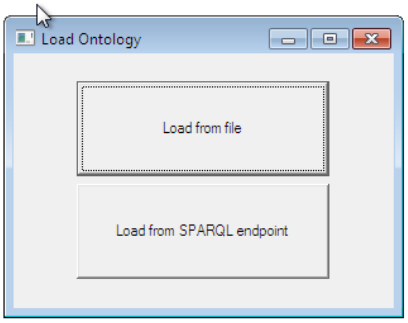
Jāatzīmē, ka šajā regulārajā izteiksmē nav iekļauti visi FULL SPARQL definētās iespējas. Tās faktiski tiek liktas vaicājuma sākumā, lai mainītu atbildes tipu, kā arī beigās, lai mainītu atbildes lielumu.

```
Query := Head AttrList Where {
     TableGraph
     NegationGraph
     FilterExpression
     AttributeSelection
     OptionalGraph
     OptionalAttributeList
     FULLSPARQL
}
TableGraph := classDef | classDef join TableGraph | join
     classDef := variable a classURI | Null
     join := variable relURI variable
NegationGraph := NOT EXIST {
                 join
                 Graph } | Null
OptionalGraph := OPTIONAL {
                 join
                 Graph }  | Null
Graph := TableGraph | NegationGraph | OptionalGraph
FilterExpr := SPARQL Filter expression
AttrSelection := attr | attr AttrSelection
OptionalAttributeList := Optional { attr } OptionalAttributeList
| Null
FULLSPARQL := SPARQL graph selection
```

7.15. att. ***ViziQuer lite regulārā izteiksme***

7.5.3. ViziQuer lite grafiskā saskarne

Ņemot vērā, ka ViziQuer lite ir realizēts arī GrTP platformā, kā ViziQuer prototips, tad projekta uzsākšana ir līdzīgas. Tikai ViziQuer lite gadījumā ir sākotnējā izvēlne – ielasīt ontoloģiju no datnes, vai arī izgūt ontoloģiju no SPARQL piekļuves punkta. Izvēlni var aplūkot 7.16. attēlā.

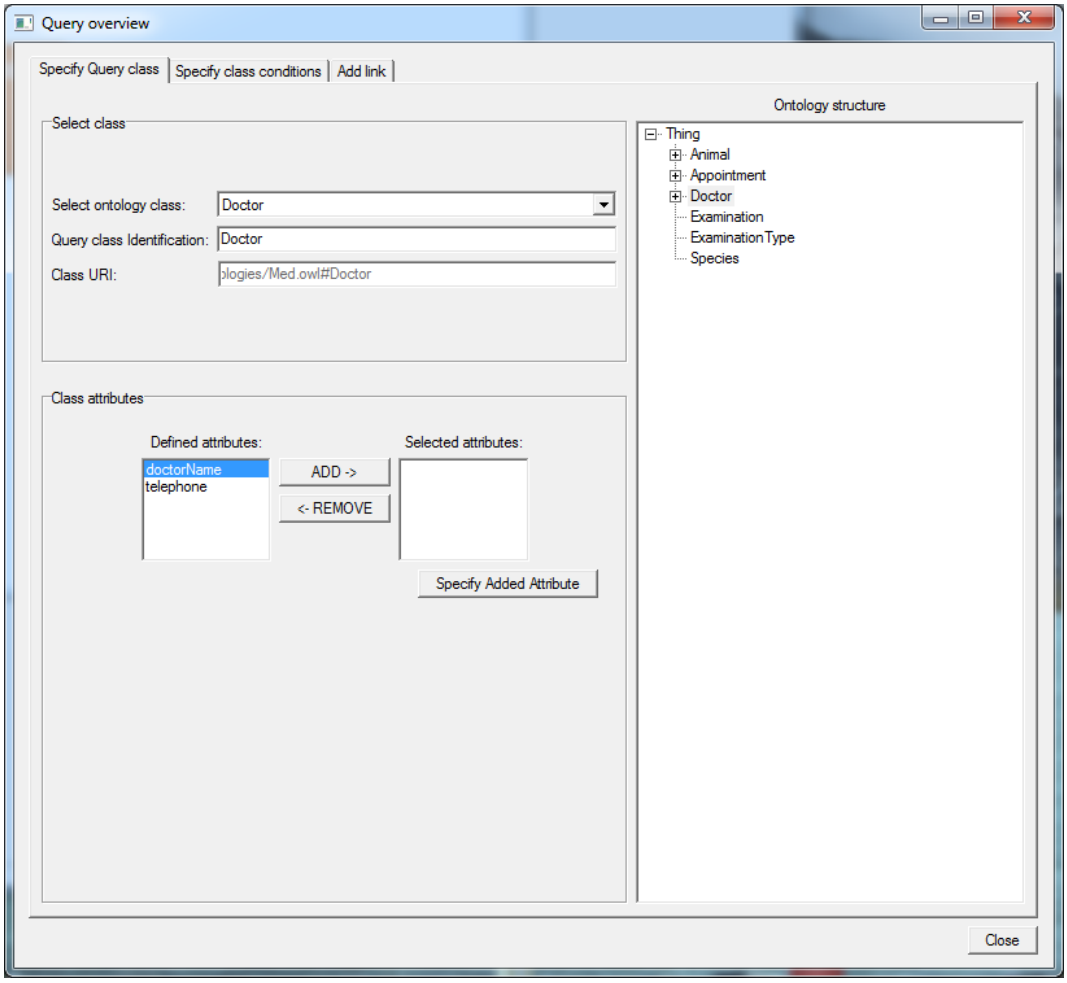


7.16. att. ***Ontoloģijas ielasīšanas izvēlne.***

Izvēloties iespēju ielasīt ontoloģiju no datnes, lietotājam ir jānorāda datne uz personīgā datora, kurā glabājas izvēlētā ontoloģija. Ja izvēlas ielasīšanu no SPARQL piekļuves punkta, tad ir jānorāda tā adrese, vai arī var izvēlēties kādu no jau pieejamajiem SPARQL piekļuves punktiem, piemēram, http://data.semanticweb.org/sparql.

Projekta logs, salīdzinājumā ar ViziQuer sākotnējo prototipu, nav mainījies, un to papildus neaprakstīsim.

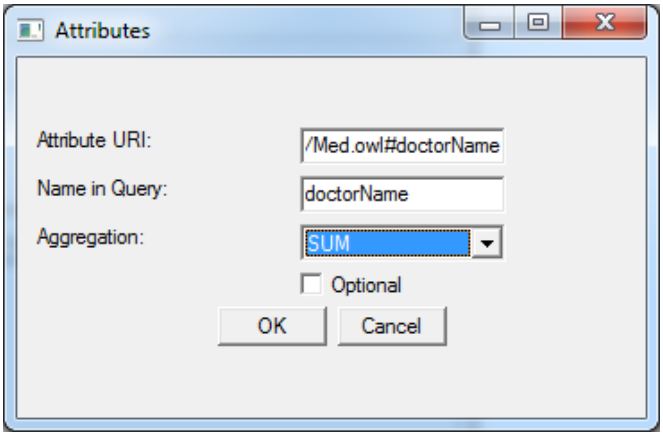
Vaicājuma sastādīšanā centrālā loma ir atvēlēta vaicājuma klases pārskata formai. Tā sastāv no 3 cilnēm, kur viena ir paredzēta objektu atlases klases specificēšanai un atribūtu pievienošanai atbildei, tā ir apskatāma 7.17. attēlā. Tālāk ir nosacījumu cilne, kur papildus tiek uzstādīti nosacījumi objektu atlases klases indivīdiem, kas ir apskatāma 7.19. attēlā. Un ir navigācijas cilne, kas ļauj pievienot jaunus atlases objektus un saites uz tiem, kas būtu piesaistīti esošajiem elementiem, tā ir apskatāma 7.20. attēlā.



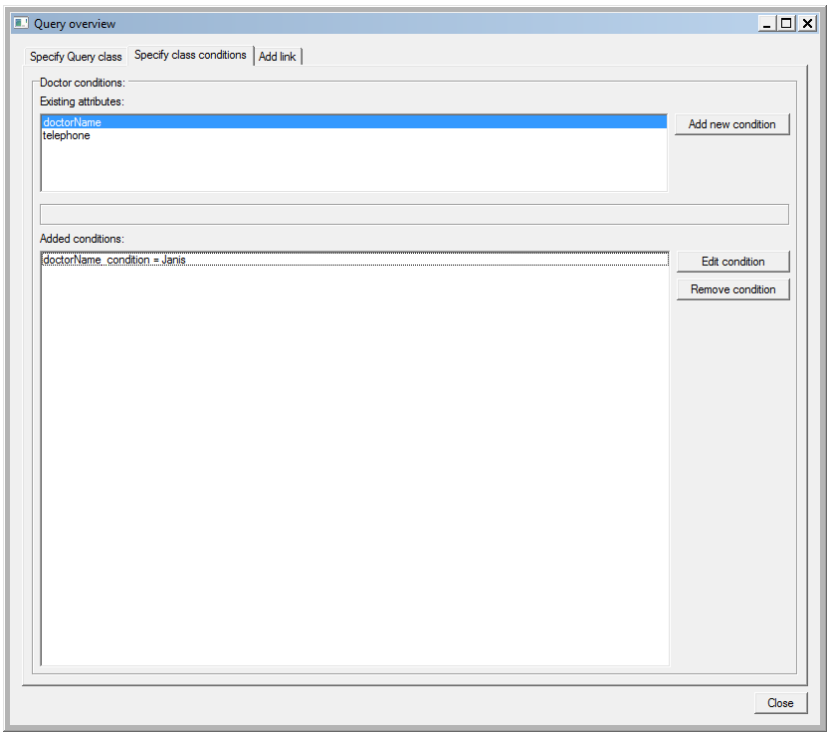
7.17. att. ***ViziQuer lite objektu atlases klases formas pārskats.***

Objektu atlases klase sastāv no trīs galvenajām daļām – ontoloģijas pārskata kokveida formātā – formas labajā pusē. Tālāk ir vaicājuma atlases klases specifikācija – ontoloģijas klase, kura tiek specificēta, kā arī ir iespējams uzdot mainīgo, kas tiks lietots vaicājuma sastādīšanai, kā arī uz ko varēs atsaukties sastādot papildus vaicājuma daļas "*Full SPARQL*" sadaļā.

Pēdējā daļa ir vaicājuma atbildes papildināšana ar atlasītās klases atribūtiem. Primāri varam pievienot dažādus atribūtus atbildei un pēc tam specificēt to īpašības formā, kas ir attēlota 7.18. attēlā.



7.18. att. ***Atribūtu detaļu specificēšanas forma.***



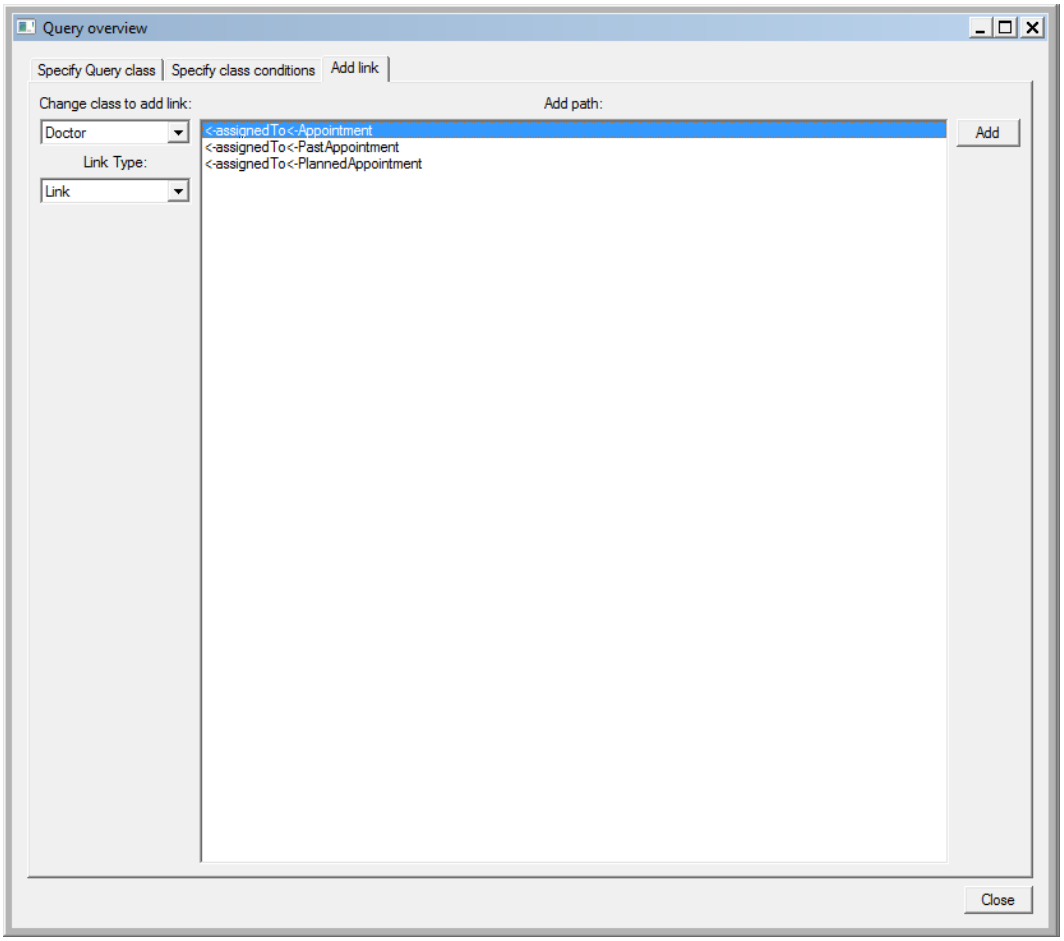
7.19. att. ***Atlases klases nosacījumu specificēšanas forma.***

Atribūta specificēšanas laukā tiek uzrādīts URI, kas tika izveidots, kad lietotājs pievienoja kādu no sagatavotajiem atribūtiem. Taču lietotājs var patvaļīgi mainīt šo URI, lai atlasītu atribūta vērtības, kas var nebūt definētas ontoloģijas shēmā.

Lietotājs var arī izvēlēties vai vēlas atribūta vērtību iegūt kā agregācijas funkciju no atlasītajiem atribūtiem, piemēram, summēt visas atlasītās samaksas vērtības, grupējot tās pēc pārējām atlasītajām vērtībām, kam nav piemērotas agregācijas funkcijas.

Lietotājs var atzīmēt arī, ka atribūta vērtība var arī nebūt obligāti aizpildīta, kā arī izvēlēties nosaukumu, kādu vēlēsies redzēt atribūtam sastādītajā SPARQL atbildē.

Vaicājuma nosacījumu formā, kas redzam 7.19. attēlā, lietotājam ir pieejami visi izvēlētās klases atribūti, kas ir doti ontoloģijas shēma. Lietotājs atbilstoši katram no tiem var uzstādīt ierobežojumu, piemēram, vārds ir "Jānis". Visi nosacījumi savā starpā tiek savienoti ar loģisko "Un".



7.20. att. ***Vaicājuma saišu pievienošanas forma.***

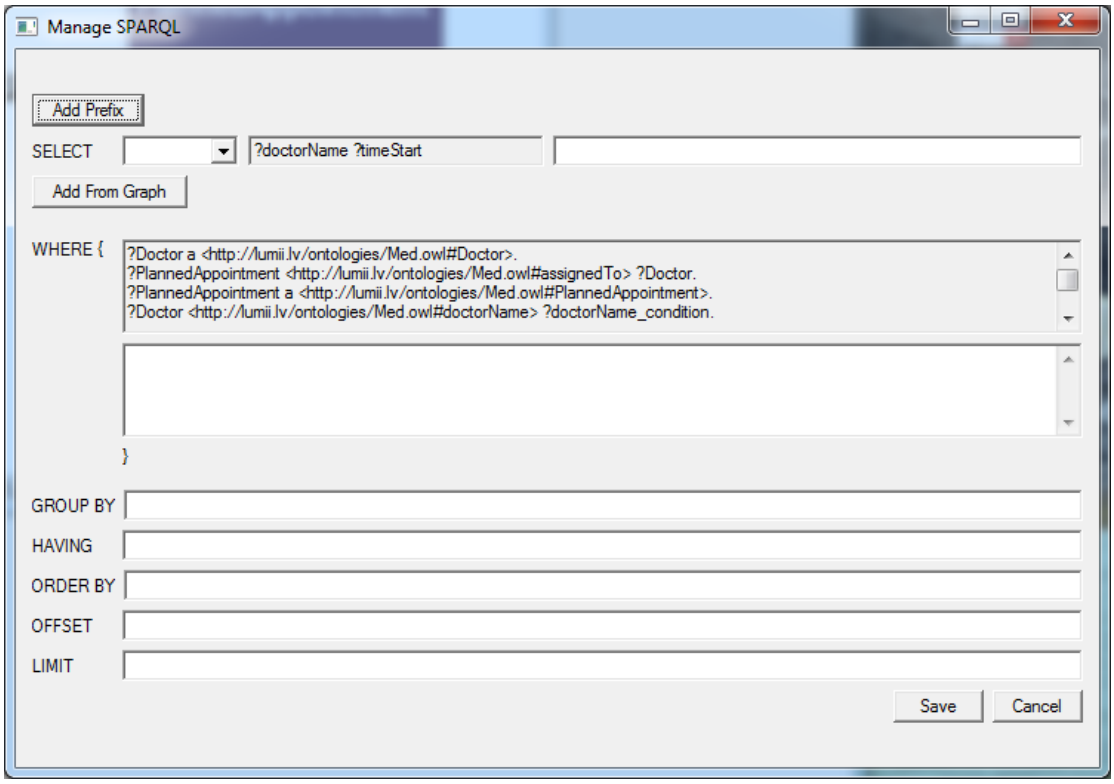
Noslēdzošā cilne, 7.20. attēlā, ir atvēlēta vaicājuma paplašināšanai un ļauj pievienot jaunas vaicājuma atlases klases. Šādā veidā ir realizēta iespēja, ka lietotāji izvēlas sākotnējo klasi un tad tai vieno klāt nosacījumus tās apkārtnē.

Ja lietotājs izvēlas pievienot jaunu saiti un atlases klasi, tad tā tiek pievienota grafiski un fokuss tiek uzstādīts uz pievienoto klasi. Lietotājs var pārslēgt klases cilni vai nosacījumu cilni un uzstādīt papildus atlases vērtības vai kritērijus pievienotai klasei.

Apvienojot šīs trīs cilnes vienā formā, piedāvājam lietotājam iespēju sastādīt vaicājumu tikai ar šīs formas palīdzību, kas ietaupa lietotāja laiku, jo tiek minimizētas liekās darbības.

Lietotājam ir iespējams ievadīt klases arī atsevišķi, un tās savienot ar izvēlēto saites tipu izmantojot paletes elementus. Taču tur papildus tiek piedāvāts serviss ar asociācijām, kas šos elementus savieno pa tiešo. Vai arī lietotājs pats var ierakstīt asociācijas URI, kam vajadzētu savienot elementus.

Līdzīgi ir arī komentāra elements, kur lietotājs var ievadīt patvaļīgu tekstu, lai vēlāk varētu labāk izprast uzrakstīto vaicājumu, vai pievienot citas svarīgas detaļas, ko vēlas pateikt par vaicājuma uzbūvi.



7.21. att. ***FULL SPARQL papildināšanas forma.***

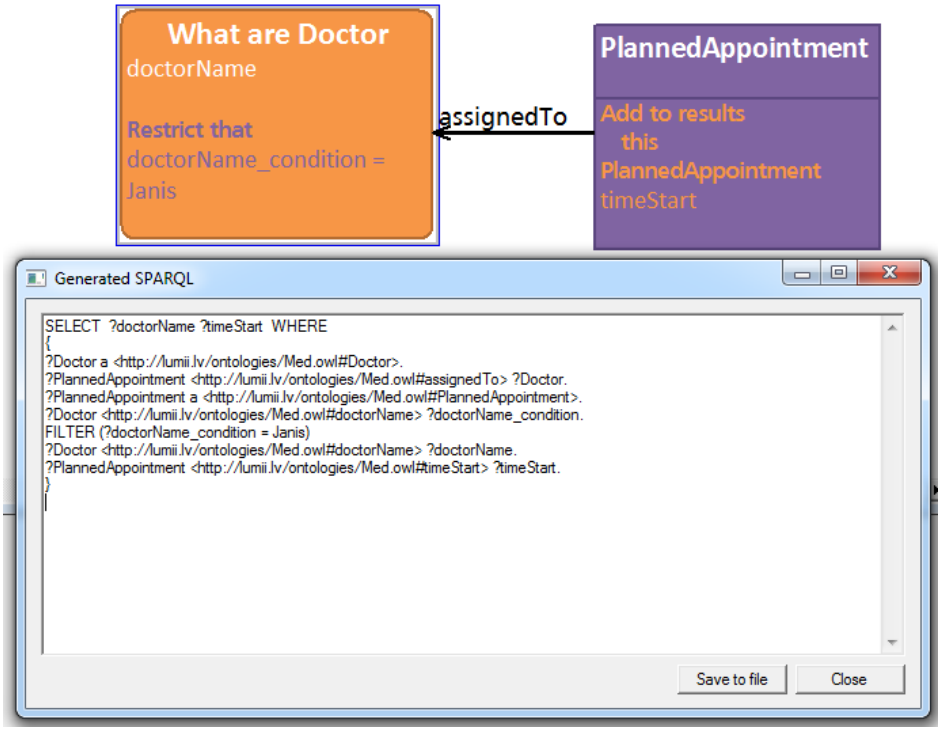
Viens no būtiskākajiem papildinājumiem ir iespēja papildināt grafisko SPARQL vaicājumu ar lietotāja definētajām konstrukcijām. Forma, kur to ir iespējams izdarīt ir parādīta 7.21. attēlā.

Atbilstoši tajā ir aizpildītas sadaļas, kas tiek saņemas no grafiskā vaicājuma un tās nav iespējams labot. Tās ir, atbildes atribūti un SPARQL grafa specifikācijas daļa.

Tālāk ir iespējams papildināt SPARQL vaicājumu. Primāri tas ir saistīts ar atbildes specificēšanu un tās formātu, piemēram, pēc kādiem atribūtiem sakārtot atbildi vai kādu atbildes rezultātu kopu vēlamies redzēt (tikai atšķirīgās atbildes vai arī uzrādīt dublētas atbildes).

Taču tajā pašā laikā nezīmējot grafisku vaicājumu, mēs varam tekstuāli uzrakstīt patvaļīgu SPARQL, kas formāli pārklāj pilnu SPARQL vaicāšanas iespēju. Raksturīgāk papildināt vaicājumu, izmantojot *full SPARQL* vaicājuma formu, būt ar specifiskiem nosacījumiem, kā arī konkrētām identifikācijas vērtībām, ko būtu grūti uzstādīt grafiskā formātā. Piemēram, pievienot saiti uz konkrēta objekta URI.

Kad grafiskais vaicājums ir sastādīts, tad varam iegūt tā translāciju uz tekstuālu SPARQL izmantojot uzlecošo izvēlni. Translācijas rezultāts ir aplūkojams 7.22. attēlā.



7.22. att. ***Grafiskā vaicājuma rezultāts kā tekstuāls SPARQL.***

Esam aprakstījuši kā lietotājs var izveidot grafisku SPARQL vaicājumu un iegūt tā rezultātu kā tekstuālu SPARQL, ko tālāk izmantot datu atlasē, vai arī programmu risinājumus.

Salīdzinot ar ViziQuer sākotnējo prototipu tas ir izveidots stabilāks, ļauj ielasīt ontoloģiju no datnes un darboties bez piekļuves SPARQL piekļuves punktam.

Vaicājuma rīks pieejams lejupielādei un lietošanai http:\\viziquer.lumii.lv\.

7.6. Līdzīgie risinājumi

Esam izveidojuši unikālu vaicājumu valodu, kas spēj izsekot un attēlot SPARQL piekļuves punktā esošo ontoloģijas shēmu. Turklāt, attēlojam tikai shēmas daļu, kas satur instanču datus.

Kā minēts rakstā [59], tad mūsu pieeja tiek raksturota kā unikāla, jo lietotājam ir iespēja arī pārskatīt shēmu, kad tiek sastādīti vaicājumi.

Rakstā [62] mūsu pieeja ir raksturota, kā query-by-navigation un salīdzināta ar Tabulator[35], SEWASIE[60] un Visor[61]. Taču tajā pašā laikā mūsu pieeja ir unikāla ar to, ka visas minētās pieejas darbojas ar navigāciju pa datiem, kamēr mēs navigējam pa sastādīto ontoloģijas shēmu. Turklāt, rezultātā iegustam SPARQL vaicājumu, kuru izpildot lietotājs saņem tabulu ar datiem tālākai analīzei.

Pēdējā īpašība ir būtiska, jo medicīnas projektā medicīnas zinātniekiem bija būtiski eksportēt datus Excel formātā un tos tālāk apstrādāt jau datu analīzes rīkā.

Esam realizējuši unikālu rīku, kurš spēj nedaudz atšķirties no citiem praksē realizētajiem rīkiem.

8. NOSLĒGUMS

Esam praksē veiksmīgi realizējuši 2 no 3 "Semantiskās Latvijas" izvirzītajām tēzēm – grafiska vaicājumu valoda, kas atvieglo vaicājumu sastādīšanu, un ontoloģijām balstītas informatīvās sistēmas, kas ļauj darboties ar semantiskiem datiem.

Dotie trīs punkti tomēr ir pietiekami apjomīgi, lai tos būtu grūti uzreiz realizēt, tāpēc izvēlējāmies sākt procesa veidošanu no apakšas uz augšu. Papildinot to ar vēl vienu būtisku lietu – datu transformāciju no relāciju datu bāzēm uz semantiskiem datiem.

Izstrādājot grafisku vaicājumu valodu to bija iespējams praktiski pārbaudīt – vai tā eksperimentāli apmierina medicīnas zinātnieku vajadzību un vai tā tiek uzskatīta par pietiekami labu, lai būtu tālāk attīstāma.

Medicīnas zinātnieku neformālais atzinums norādīja, ka grafiska vaicājumu valoda pār semantiskiem datiem var atvieglot lietotāju darbu, lai veidotu analītiskus pārskatus. Tas norādīja, ka tā var pārklāt apgabalu, kur lietotāji nav pārāk labi rakstot SQL, bet viņiem nesagādā problēmas sastādīt grafisku vaicājumu.

Promocijas darbā spējām izstrādāt un pierādīt grafiskās vaicājumu valodas lasāmību. Taču tajā pašā laikā ir otrs valodas aspekts – vaicājumu rakstīšana. Ja vaicājumu valodas lasāmības pārbaude ir vairāk teorētisks darbs, tad vaicājumu pieraksts vairāk ir tehnisks darbs, kas prasa izveidot ideālu atbalsta rīku, lai tas minimizētu un atvieglotu vaicājuma sastādīšanu.

Viens no centrālajiem rīka uzdevumiem ir pateikt rakstītājam priekšā to, ko lietotājs vēlas ierakstīt. Līdzīgi kā Google meklētājā ierakstot sintaktiski nepareizu vārdu mums tiek norādīts, ka tam ir maz rezultāti, varbūt vēlamies meklēt citu vārdu? Kā arī rakstot vārdu tiek piedāvātas populārākās iespējas kā to var pabeigt.

Ja izvēlamies konkrētu realizācijas platformu, tad tehniski esam ierobežoti ar šīs platformas tehniskajām iespējām. Mēs varam teorētiski uzzīmēt, mūsuprāt, ideālo vaicājumu sastādīšanas rīku, taču bez tehniskās realizācijas mēs nevarēsim pārbaudīt, cik ļoti ātri un pareizi vaicājumi tiek sastādīti.

Piemēram, ierasti nosacījumi tiek uzstādīti laukā ierakstot vērtības, kas ierobežo konkrēto lauku. Taču teorētiski, mūsuprāt, pārskatāmāk būtu iespējams iekrāsot vērtību apgabalus uz līnijas, kas atbilst mūsu atlasei. Līdzīgi kā matemātikas stundās māca iekrāsot viena argumenta funkcijas vērtību apgabalu.

Izstrādātās ontoloģijām balstītās informatīvās sistēmas parāda, ka semantisku sistēmu izstrāde var būt pietiekami ātra un ka veiksmīgi realizēt semantiskā tīmekļa uzstādījumu par semantisko datu pieeju lietotājiem.

Lai arī nenoliedzami savā darbā tikai ļoti aptuveni esam ieskicējuši OBIS nišu, tomēr uzskatām, ka tai ir perspektīva. Jau šobrīd katrai sistēmas lietotāju grupai ir savs skatījums. Mūsu naivajā pieejā ir izstrādāts tikai viens skatījums, taču, veidojot dažādus skatījumus, arī parādītos OBIS perspektīva praktiskos piemēros.

Varam apskatīt, ka mūsdienās populāri kļūst dažādi ietvari un iespējas lietot dažādus šablonus, piemēram, varam paņemt emuāru sistēmu WordPress[62] un pielāgot izskata šablonu, kas mums patīk vislabāk.

Līdzīgi ontoloģiju repozitorijs varētu atvieglot OBIS sistēmu izstrādi. Turklāt, sniedzot papildus priekšrocību, jo tas varētu apvienot dažādo sistēmu datus, lai gala lietotājiem būtu iespējas labāk orientēties informācijā.

Semantiskās tehnoloģijas attīstās, un tām ir perspektīva padarīt informāciju pieejamāku un lietojamāku cilvēkiem. Ar praktiskiem piemēram esam paradījuši, ka šī perspektīva var nest dažādus labumus salīdzinot ar esošajām pieejām sistēmu izstrādē un datu apstrādē.

[62] https://wordpress.org/

IZMANTOTĀ LITERATŪRA

[1] *Zloof, M. M. (1975, May). Query by example. In Proceedings of the May 19-22, 1975, national computer conference and exposition (pp. 431-438). ACM.*

[2] *Sawyer, P., & Mariani, J. A. (1995). Database systems: challenges and opportunities for graphical HCI. Interacting with Computers, 7(3), 273-303.*

[3] *Catarci, T., & Santucci, G. (1995). Diagrammatic vs textual query languages: a comparative experiment. In Visual Database Systems 3 (pp. 69-83). Springer US.*

[4] *Catarci, T., Costabile, M. F., Levialdi, S., & Batini, C. (1997). Visual query systems for databases: A survey. Journal of Visual Languages & Computing,8(2), 215-260.*

[5] *Barzdins, J., Barzdins, G., Balodis, R., Cerans, K., Kalnins, A., Opmanis, M., & Podnieks, K. (2006). Towards Semantic Latvia. In Proceedings of Seventh International Baltic Conference on Databases and Information Systems, Communications, Vilnius, Lithuania, O. Vasileckas, J. Eder, A. Caplinskas (Eds.), Vilnius, Technika (pp. 203-218).*

[6] *Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web.Scientific american, 284(5), 28-37.*

[7] *McGuinness, D. L., & Van Harmelen, F. (2004). OWL web ontology language overview. W3C recommendation, 10(2004-03), 10.*

[8] *Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., & Horrocks, I. (2013, October). OptiqueVQS: towards an ontology-based visual query system for big data. In Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems (pp. 119-126). ACM.*

[9] *G.Barzdins (2005), "UML + Spreadsheets as a Telecommunications Network Documentation and Analysis Tool", Databases and Information Systems, J.Barzdins and A.Caplinskas (Eds.), Vol.118, IOS Press, 2005, pp.238-246. ISSN 0922-6389 .*

[10] *Date, C. J. (2006). An Introduction To Database Systems, 8/E. Pearson Education India.*

[11] *Chen, P. P. S. (1976). The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems (TODS), 1(1), 9-36.*

[12] *Wahlster, W. (2013). The semantic product memory: an interactive black box for smart objects. In SemProM (pp. 3-21). Springer Berlin Heidelberg.*

[13] *Negash, S. (2004). Business intelligence. Communications of the Association for Information Systems, 13(1), 177-195.*

[14] *Keramopoulos, E., Pouyioutas, P., & Ptohos, T. (2008). The GOQL Language and its Formal Specifications. IJCSA, 5(2), 23-51.*

[15] Polyviou, S., Evripidou, P., & Samaras, G. (2004, June). Query by browsing: A visual query language based on the relational model and the desktop user interface paradigm. In The 3rd Hellenic Symposium on Data Management.

[16] Gallego, M. A., Fernández, J. D., Martínez-Prieto, M. A., & de la Fuente, P. (2011). An empirical study of real-world SPARQL queries. In 1st International Workshop on Usage Analysis and the Web of Data (USEWOD2011) at the 20th International World Wide Web Conference (WWW 2011), Hydebarabad, India.

[17] Juel, C. (1988). Learning to read and write: A longitudinal study of 54 children from first through fourth grades. Journal of educational Psychology, 80(4), 437.

[18] Barzdins, G., Liepins, E., Veilande, M., & Zviedris, M. (2008). Semantic Latvia approach in the medical domain. In Proc. 8th International Baltic Conference on Databases and Information Systems (pp. 89-102). Tallinn University of Technology Press.

[19] Knublauch, H., Fergerson, R. W., Noy, N. F., & Musen, M. A. (2004). The Protégé OWL plugin: An open development environment for semantic web applications. In The Semantic Web–ISWC 2004 (pp. 229-243). Springer Berlin Heidelberg.

[20] Noy, N. F., Shah, N. H., Whetzel, P. L., Dai, B., Dorf, M., Griffith, N., ... & Musen, M. A. (2009). BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic acids research, 37(suppl 2), W170-W173.

[21] Alani, H., Dupplaw, D., Sheridan, J., O'Hara, K., Darlington, J., Shadbolt, N., & Tullo, C. (2007). Unlocking the potential of public sector information with semantic web technology. In The Semantic Web (pp. 708-721). Springer Berlin Heidelberg.

[22] Barzdins, G., Barzdins, J., & Cerans, K. (2009). From Databases to Ontologies.

[23] Kupfer, A., Eckstein, S., Neumann, K., & Mathiak, B. (2006). Keeping track of changes in database schemas and related ontologies. In Databases and Information Systems, 2006 7th International Baltic Conference on (pp. 63-68). IEEE.

[24] Borgida, A. (2007). Knowledge Representation Meets Databases-a View of the Symbiosys. In Description Logics.

[25] Leigh, J., Mika, P. (2007). Elmo User Guide. URL: http://www.openrdf.org/doc/elmo/1.0-beta2/user-guide/

[26] Bizer, C., & Seaborne, A. (2004, November). D2RQ-treating non-RDF databases as virtual RDF graphs. In Proceedings of the 3rd international semantic web conference (ISWC2004) (Vol. 2004).

[27] Blakeley, C. (2007). RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping). OpenLink Software.

[28] Būmans, G., & Čerāns, K. (2011). Advanced RDB-to-RDF/OWL mapping facilities in RDB2OWL. In Perspectives in Business Informatics Research (pp. 142-157). Springer Berlin Heidelberg.

[29] Rodriguez-Muro, M., Rezk, M., Hardi, J., Slusnys, M., Bagosi, T., & Calvanese, D. Evaluating SPARQL-to-SQL translation. In 2nd OWL Reasoner Evaluation Workshop (ORE 2013) (p. 94).

[30] Rikacovs, S. (2011). Export of Relational Databases to RDF Databases by Model Transformations. In Perspectives in Business Informatics Research (pp. 158-166). Springer Berlin Heidelberg.

[31] Broekstra, J., Kampman, A., & Van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying rdf and rdf schema. In The Semantic Web—ISWC 2002 (pp. 54-68). Springer Berlin Heidelberg.

[32] Barzdins, G., Rikacovs, S., & Zviedris, M. (2009). Graphical query language as SPARQL frontend. In Local Proceedings of 13th East-European Conference (ADBIS 2009) (pp. 93-107).

[33] Parsia, B., & Sirin, E. (2004, November). Pellet: An owl dl reasoner. In Third International Semantic Web Conference-Poster (p. 18).

[34] Bārzdiņš, J., Zariņš, A., Čerāns, K., Kalniņš, A., Rencis, E., Lāce, L., ... & Sproģis, A. (2007). GrTP: Transformation Based Graphical Tool Building Platform. In MoDELS07, Workshop: Model Driven Development of Advanced User Interfaces (MDDAUI-2007), available at http://ceur-ws. org (Vol. 297).

[35] Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., ... & Sheets, D. (2006, November). Tabulator: Exploring and analyzing linked data on the semantic web. In Proceedings of the 3rd International Semantic Web User Interaction Workshop (Vol. 2006).

[36] Longwell project page. SIMILE Project (2006). URL: http://simile.mit.edu/wiki/Longwell

[37] Zhou, Q., Wang, C., Xiong, M., Wang, H., & Yu, Y. (2007). SPARK: adapting keyword query to semantic search. In The Semantic Web (pp. 694-707). Springer Berlin Heidelberg.

[38] Barzdins, J., Kalnins, A., Rencis, E., & Rikacovs, S. (2008). Model transformation languages and their implementation by bootstrapping method. InPillars of computer science (pp. 130-145). Springer Berlin Heidelberg.

[39] Sowa J.F., (2011), Serious Semantics, Serious Ontologies, panel presented at Semantic Technology Conference (SEMTEC 2011), San Francisco

[40] Motik, B., Patel-Schneider, P. F., Parsia, B., Bock, C., Fokoue, A., Haase, P., ... & Smith, M. (2009). OWL 2 web ontology language: Structural specification and functional-style syntax. W3C recommendation, 27, 17.

[41] Barzdins J., Barzdins G., Cerans K., Liepiņs R., Sprogis A., (2010), OWLGrEd: a UML Style Graphical Notation and Editor for OWL 2. In: Clark, K., Sirin, E. (eds.) Proc. 7 th International Workshop "OWL: Experience and Directions" (OWLED-2010)

[42] Soley R, (2000), Model Driven Architecture. OMG white paper

[43] Tao, J., Sirin, E., Bao, J., McGuinness, D.L (2010) Integrity constraints in OWL. In: Fox, M., Poole, D. (eds.) Proc. of the 24th AAAI Conference on Artificial Intelligence, AAAI 2010. AAAI Press

[44] M.Zviedris, A.Romane, G.Barzdins, K.Cerans, (2014) "Ontology-Based Information System", Wooju K., Ying D. and Hong-gee K. (Eds.) The 3rd Joint International Semantic Technology Conference, LNCS, tiks publicēta

[45] Tudorache T., Nyulas C., Noy N., Musen M., (2012), WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web. Semantic Web Journal, pages 1–11.

[46] Auer S., Dietzold S., Riechert T., (2006). OntoWiki – A Tool for Social, Se- mantic Collaboration. In In Proceedings of the 5th International Semantic Web Conference, ISWC2006, volume 4273 of LNCS. Springer

[47] Riechert T., Morgenstern U., Auer S., Tramp S., Martin M., (2010) Knowledge engineering for historians on the example of the catalogus professorum lipsiensis in ISWC2010, ser. LNCS. Shanghai / China: Springer.

[48] Zviedris, M., (2010). Ontology repository for User Interaction. In Mathieu d'Aquin, Alexander Garcia Castro, Christoph Lange, Kim Viljanen edititors, ORES-2010 Workshop on Ontology Repositories and Editiors for the Semantic Web, volume http://CEUR-WS.org/Vol-596/ .CEUR

[49] Le-Phuoc, D., Polleres, A., Tummarello, G., & Morbidoni, C. (2008). DERI pipes: visual tool for wiring web data sources.

[50] Zviedris, M., & Barzdins, G. (2011). ViziQuer: a tool to explore and query SPARQL endpoints. In The Semanic Web: Research and Applications (pp. 441-445). Springer Berlin Heidelberg.

[51] Liepins, R., Cerans, K., & Sprogis, A. (2012). Visualizing and Editing Ontology Fragments with OWLGrEd. In I-SEMANTICS (Posters & Demos) (pp. 22-25).

[52] Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., & Stegemann, T. (2009). Relfinder: Revealing relationships in rdf knowledge bases. In Semantic Multimedia (pp. 182-187). Springer Berlin Heidelberg.

[53] Liepiņš, R. lQuery: A Model Query and Transformation Library. Datorzinātne un informācijas tehnoloģijas, 27.

[54] Heim, P., Ertl, T., & Ziegler, J. (2010). Facet graphs: Complex semantic querying made easy. In The Semantic Web: Research and Applications (pp. 288-302). Springer Berlin Heidelberg.

[55] Völker, J., & Niepert, M. (2011). Statistical schema induction. In The Semantic Web: Research and Applications (pp. 124-138). Springer Berlin Heidelberg.

[56] Kozlovičš, S., (2013), "Transformāciju vadīta arhitektūra un tās grafiskie prezentācijas dziņi", promocijas darbs

[57] Rikacovs, S. (2008). The base transformation language L0+ and its implementation. Datorzinātne un informācijas tehnoloģijas, 75.

[58] Kozlovics, S. (2010). A Dialog Engine Metamodel for the Transformation-Driven Architecture. Scientific Papers, University of Latvia, 756, 151-170.

[59] Fellmann, M. (2013), SEMANTIC PROCESS ENGINEERING Konzeption und Realisierung eines Werkzeugs zur semantischen Prozessmodellierung, promocijas darbs

[60] Catarci, T., Dongilli, P., Di Mascio, T., Franconi, E., Santucci, G., & Tessaris, S. (2004). An ontology based visual tool for query formulation support. In ECAI(Vol. 16, p. 308).

[61] Popov, I. O., Schraefel, M. C., Hall, W., & Shadbolt, N. (2011). Connecting the dots: a multi-pivot approach to data exploration. In The Semantic Web–ISWC 2011 (pp. 553-568). Springer Berlin Heidelberg.

[62] Soylu, A., Skjæveland, M. G., Giese, M., Horrocks, I., Jimenez-Ruiz, E., Kharlamov, E., & Zheleznyakov, D. (2013). A Preliminary Approach on Ontology-Based Visual Query Formulation for Big Data. In Metadata and Semantics Research (pp. 201-212). Springer International Publishing.

[63] Klímek, J., Helmich, J., & Nečaský, M. (2013). Payola: Collaborative Linked Data Analysis and Visualization Framework. In The Semantic Web: ESWC 2013 Satellite Events (pp. 147-151). Springer Berlin Heidelberg.

[64] Kramer, K., Dividino, R. Q., Groner, G., (2013), SPACE: SParql index for efficient Auto ComplEtion, International Semantic Web Conference Posters & Demos, volume 1035 of CEUR Workshop Proceedings, page 157-160. CEUR-WS.org

1. PIELIKUMS

Datu shēmas tehniskā relācija SQL

```
/* SQLEditor (Generic SQL)*/

CREATE TABLE positions
(
id INTEGER NOT NULL AUTO_INCREMENT,
position_name CHAR,
PRIMARY KEY (id)
);

CREATE TABLE Employee
(
id INTEGER NOT NULL AUTO_INCREMENT,
name CHAR NOT NULL,
surname CHAR NOT NULL,
age INTEGER,
position INTEGER NOT NULL,
salary INTEGER,
PRIMARY KEY (id)
);

CREATE TABLE Project
(
id INTEGER NOT NULL AUTO_INCREMENT  UNIQUE,
name CHAR NOT NULL,
priority INTEGER,
budget INTEGER NOT NULL,
managerID INTEGER,
PRIMARY KEY (id)
);

CREATE TABLE working
(
project INTEGER,
employee INTEGER
);

ALTER TABLE Employee ADD FOREIGN KEY (position) REFERENCES positions (id);
ALTER TABLE Project ADD FOREIGN KEY (managerID) REFERENCES Employee (id);
ALTER TABLE working ADD FOREIGN KEY (project) REFERENCES Project (id);
ALTER TABLE working ADD FOREIGN KEY (employee) REFERENCES Employee (id);
```

Datu shēmas tehniskā realizācija OWL, kas pierakstīta izmantojot RDF/XML sintaksi

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY projects "http://www.semanticweb.org/ontologies/2012/8/projects.owl#" >
]>
<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2012/8/projects.owl#"
     xml:base="http://www.semanticweb.org/ontologies/2012/8/projects.owl"
     xmlns:projects="http://www.semanticweb.org/ontologies/2012/8/projects.owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
```

```
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<owl:Ontology rdf:about=""/>

<!--
////////////////////////////////////////////////////////////////////////////////////
//
// Object Properties
//
////////////////////////////////////////////////////////////////////////////////////
 -->
<!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#employedIn -->
<owl:ObjectProperty rdf:about="#employedIn">
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:range rdf:resource="#Project"/>
</owl:ObjectProperty>

<!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#manages -->
<owl:ObjectProperty rdf:about="#manages">
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:range rdf:resource="#Project"/>
</owl:ObjectProperty>

<!--
////////////////////////////////////////////////////////////////////////////////////
//
// Data properties
//
////////////////////////////////////////////////////////////////////////////////////
 -->
<!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#age -->

<owl:DatatypeProperty rdf:about="#age">
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#budget -->
<owl:DatatypeProperty rdf:about="#budget">
    <rdfs:domain rdf:resource="#Project"/>
    <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#name -->
<owl:DatatypeProperty rdf:about="#name">
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:domain rdf:resource="#Project"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#position -->
<owl:DatatypeProperty rdf:about="#position">
    <rdfs:domain rdf:resource="#Employee"/>
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:range>
        <rdf:Description>
            <rdf:type rdf:resource="&owl;DataRange"/>
            <owl:oneOf>
                <rdf:Description>
                    <rdf:type rdf:resource="&rdf;List"/>
                    <rdf:first rdf:datatype="&xsd;string">Assistant</rdf:first>
                    <rdf:rest>
                        <rdf:Description>
                            <rdf:type rdf:resource="&rdf;List"/>
                            <rdf:first rdf:datatype="&xsd;string">Manager</rdf:first>
                            <rdf:rest>
                                <rdf:Description>
                                    <rdf:type rdf:resource="&rdf;List"/>
                                    <rdf:first rdf:datatype="&xsd;string"
                                        >SeniorProfessional</rdf:first>
                                    <rdf:rest rdf:resource="&rdf;nil"/>
                                </rdf:Description>
                            </rdf:rest>
                        </rdf:Description>
                    </rdf:rest>
```

```
                    </rdf:Description>
                </owl:oneOf>
            </rdf:Description>
        </rdfs:range>
    </owl:DatatypeProperty>

    <!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#priority -->
    <owl:DatatypeProperty rdf:about="#priority">
        <rdfs:domain rdf:resource="#Project"/>
        <rdfs:range rdf:resource="&xsd;integer"/>
    </owl:DatatypeProperty>

    <!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#salary -->
    <owl:DatatypeProperty rdf:about="#salary">
        <rdfs:domain rdf:resource="#Employee"/>
        <rdfs:range rdf:resource="&xsd;integer"/>
    </owl:DatatypeProperty>

    <!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#surname -->
    <owl:DatatypeProperty rdf:about="#surname">
        <rdfs:domain rdf:resource="#Employee"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#Employee -->
    <owl:Class rdf:about="#Employee"/>
    <!-- http://www.semanticweb.org/ontologies/2012/8/projects.owl#Project -->
    <owl:Class rdf:about="#Project"/>
</rdf:RDF>

<!-- Generated by the OWL API (version 2.2.1.1138) http://owlapi.sourceforge.net -->
```

2. PIELIKUMS

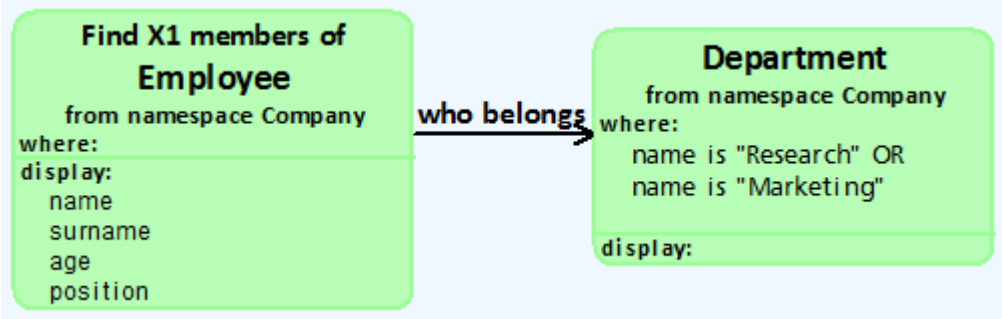
From graphics to text

Instructions

There are 18 graphically displayed queries that represents data questions. You need to rephrase all queries from graphic to natural language. Try to phare queries as natural language questions.

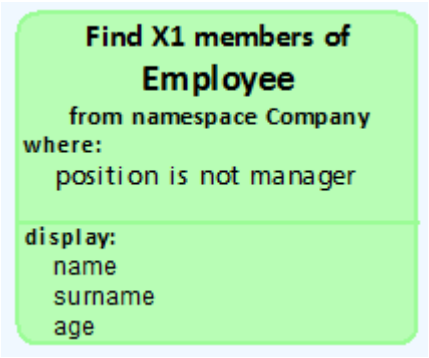
Find name, surname and age of all managers.

What are the full names and ages of the namespace company employees, with the position of manager ? (OK)
Find employees listed as managers. The data the query will get is the name, surname, and age of managers within the company. (Ok)
Display the name, surname, and age of the position manager. (Ok)
Find the managers from this list of employees. (no return data)
Find the name, surname, and age of the company's employees that are managers. (Ok)
Find the name, surname and age of the manager from namespace company. (Ok)
You are to fins 1 employee from a company. the dadta you will get is position, name surname and age (Problemas ar mainīgo, nav nosacījums)
Find the name, surname, and age, of all employees of namespace company who's position is manager. (Ok)
The name, surname and age of a manager from a specific company. (Ok, problēmas ar namespace)
employee from name space company job position is manager display the name,surname,age



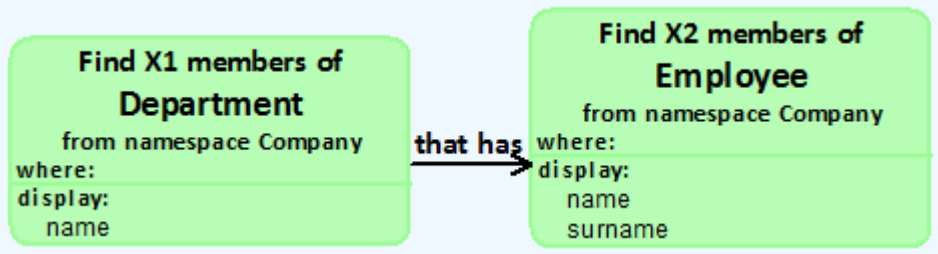
Find information about employees that works in Research or Marketing departments.

1. **What are the full names, ages and positions of namespace company employees that work in the marketing or research departments? (Ok)**
2. **Find employees in the research and marketing departments. The data the query will get is the names, surname, age, and position of employees in marketing and research departments. (Ok)**
3. **Display the name, surname, age, and position of people in the Research or Marketing departments. (Ok)**
4. **Find the employees in this company who wok for research or marketing divisiions. (No return data)**
5. **Find the name, surname, age, and positions on employees in the research or marketing department of the company. (Ok)**
6. **Find the name, surname,age abd position of employees who belong to Research or Marketing department. (Ok)**
7. **you are looking for an employye who belongs to a particular branch od the company. You will get his position, such as research marketing (Not ok, problems with connection)**
8. **Find the name, surname, age, and position, of all employees of namespace Company who belong to the research or marketing departments. (Ok)**
9. **The name, surname, age and position of an employee at a specific company in the research or marketing department. (Problems with namespace)**
10. **find x1members of employee from name space company where name surname,age,position who belongs the department from name space company where name is research or marketing,display none**



Find information about all employees that is not managers.

What are the full names and ages of the namespace non-management employees? (Ok)
Fine employees in the company that are not managers. The date the query will provide is the name, surname, and age of employees who are not managers. (Ok)
Display the names, surnames, and ages of people who aren't managers. (Ok)
Find the employees at this company who are not managers. (Nav return data)
Find the name, surname, and age of all the employees in the company that aren't managers. (Ok)
Find the name, surname and age of employees who are not managers. (Ok)
you are looking for an employye who is not a manager. You will get his name, surname and age (Ok)
Find name, surname, and age of all employees of namespace company whose position is not manager. (Ok)
The name, surname and age of an employee at a specific company who is not a manager. (Ok, bet ir specific company)
find x1members of employee from name space company where position is not manager display the name,surname,age



Find all department names and list all employees that work there.

What are the full names of the employees of the Namespace Company? (No link to department, no name of department)

Find employees in each department. The data the query will provide is the name and surname for ever employee in each department. (No name of department)

Display the name of people who have names and surnames? (Problems with class name)

Find a member of this department that works at this company. (No employee names)

Find the departments that the company's employees are in and include their name and surname. (+/- ok)

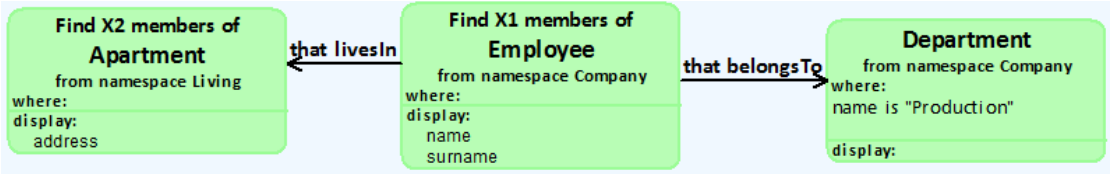
Find the name of department which has more than one employee (No employee names)

you are looking for 1 member of a department that has two employyes in the company. You will find their first and last names (Problems with variables, dont display department)

Find all members of a provided department in namespace company by searching for a particular name and surname. (Connection as additional search)

The name and surname of 2 employees in a specific department of a company. (Problems with variable, name memberOf, no department name)

find x1members of Department from name space company where display the name that has find x2members of employee from name space company where display the name,surname



Find apartment address for employees with name and surname that works in production department.

What are the full names of the employees who live in namespace living apartments and work in the production department? (No apartment adress)

Find employees that work in the production department and live in an apartment. The data the query will provide is the address, name, and surname of employees who work in the production department and live in an apartment. (+/- Ok)

Display the full names of employees in the Production department who live at apartments. (No apartment adress)

Two employees who work in this department live in that apartment. (Problems with variables and return data)

For each employee in the company, write their name, surname, and address if they are in the Production department. (Ok)

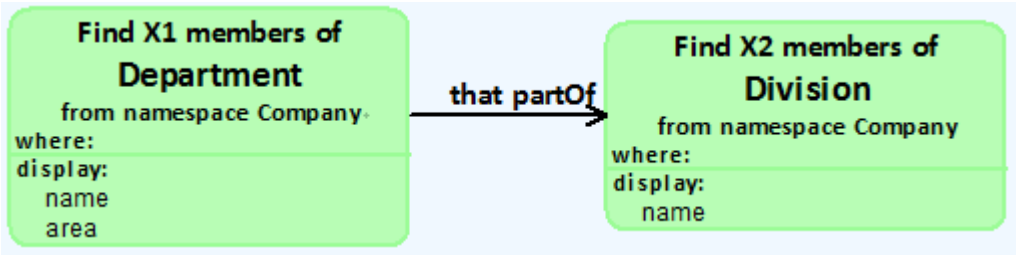
Find the name, surname and address of the employees who work in Production department and lives in the same apartment (Ok)

you are looking for 2 people who livi in an apartment. one of them is an employee in the company. you will finds their address, name, surname and posistion (Problems with variables, department)

Find name and surname of an employee of namespace company who work in the Production department that lives at a certain address. (Address as a condition not output)

The name and surname of an employee living at a specified apartment from the Production department of a company. (No address output)

find x1members of employee from name space company where display the name,surname that lives in find x2members of apartment from name space living where display the address, x1members of employee from name space company that belongs to department from name space company where name is production display none



Find all department with area and divisions.

What are the names and areas of the members of the namespace department that is part of the namespace company division? (No name of division)

Find employees in all departments that are members of a division. The data the query will provide is the name and area of employees in divisions in all departments. (Just wants to find employee(members) and answers justs from one class)

Find the people who are part of a particular division within a department. (Also wants people(members))

This member of the department is part of that division. (No answer data)

Write the name, and area of members of the department that are members of the division. (No name of division)

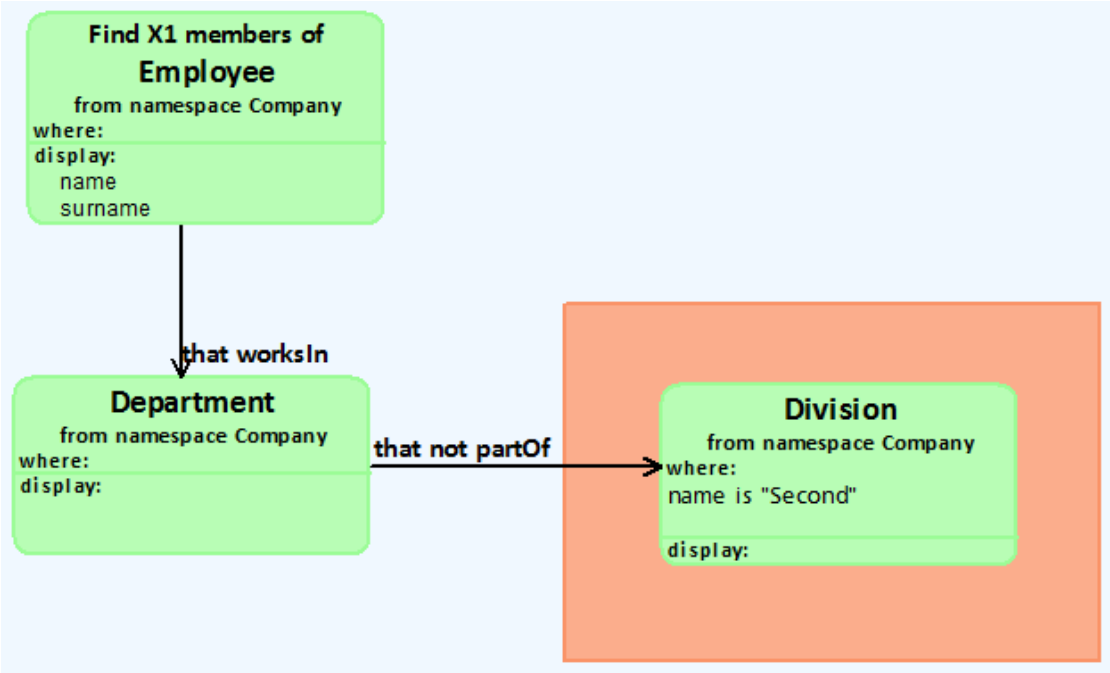
Find the name, area and division of the department that is part of the provided division (Ok)

you are looking for 1 member of the company that are part of a division that has two member. You will get name and area (Problems with variables)

Find the name and area of members of a certain department that is a part of a certain division of namespace company. (Problems with members and no division name)

The name and area of a department in a specific department at a specified company. (No division mentioned)

find x1members of Department from name space company where display the name,area that part of find x2members of Division from name space company where display the name



Find employees that works in departments that is not in Second division.

What are the full names of the employees who work in the namespace company department that is not part of the namespace company second division? (Ok)

Find employees in all departments in a second division. The data the query will provide is the name and surname of all employees in a second division in the company. (No negation)

Find the full names of employees that aren't in the second division of their departments. (Ok)

This department member is not in that division. (No second mentioned)

Write the name, and surname of employees that work in the department but isn't in the second division (Ok)

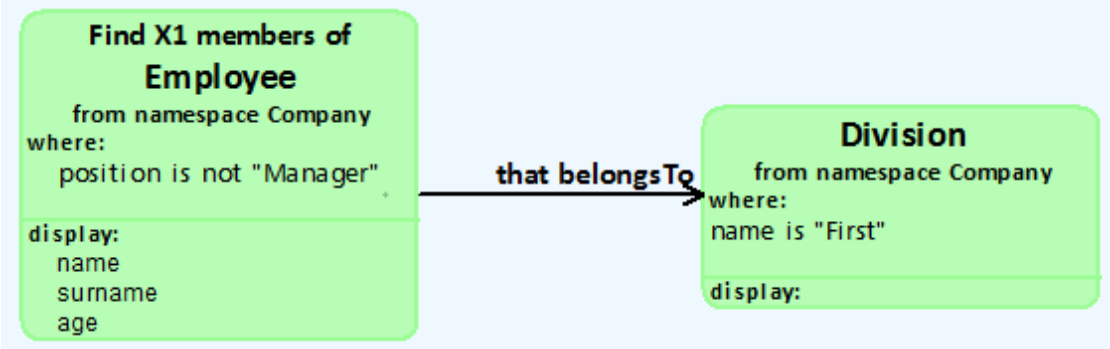
Find the name and surname of the employees who work in the department that is not part of the Second division. (Ok)

you are looking for an employye who works in a specific department and division. you will get his name (Not negation on division and not condition on division)

Find name and surname of employees of namespace company that work in a particular department that is not part of the Second division. (Ok)

The name and surname of an employee from a department of a company that is not part of the Second division within the company. (Ok)

find x1members of employee from name space company where display the name,surname that works in Department from name space company where and display is none that not part of division from name space company where name is second display is empty



Find employees that are not managers and belongs to First division.

What are the full names and ages of the employees of the first division who are not managers? (Ok)

Find employees in the first division who are managers. the data the query will provide is the name, surname, and age of managers in the first division. (No negation of managers)

Find the full names and ages of employees who are in the first division but who aren't managers. (Ok)

Find an employee who is not a manager and who belongs to this division. (No specific conditions of Division mentioned)

Find the name, surname, and age of employees non-manager employees that are in the first division. (Ok)

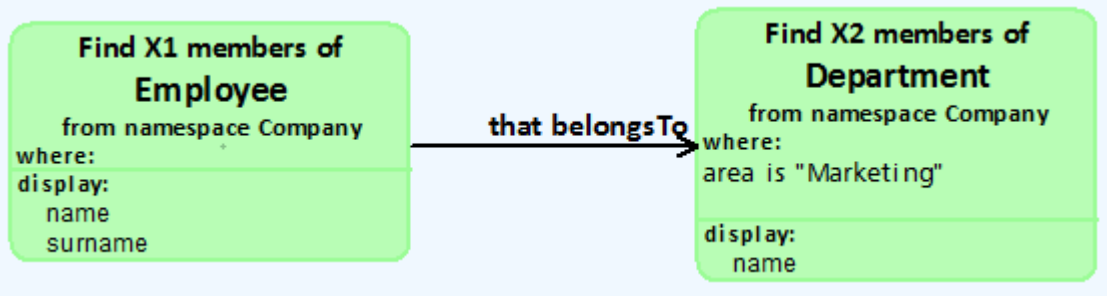
Find the name, surname and age of the employee who is not a manager but belongs to first division. (Ok)

looking for an employye who is not a manager that works in a specific divisions. you will find name, surname, age and division (No specific conditions of Division mentioned)

Find name, surname, and age, of all employees of the namespace company that are not managers, that belong to the first division. (Ok, problems with namespace)

The name, surname and age of an employee at a specified company who is not a manager from the First division. (Ok, but namespace as specific company)

find x1members of employee from name space company where position is not manager display the name,surname,age that belongs to division from name space company where name is first display is none



Find employees that works in marketing area departments.

What are the full names of the employees of namespace company that work in the marketing department? (No department name in answer)

Find employees in the marketing department. The data the query will provide is the name and surname of employees in the marketing department. (No department name in answer)

Find the full names of employees who work in marketing. (No department name in answer)

This employee works in the marketing department. (Nothing about answer, but seems that miss department name)

Find the name and surname of employees that are members of the marketing department. (No department name in answer)

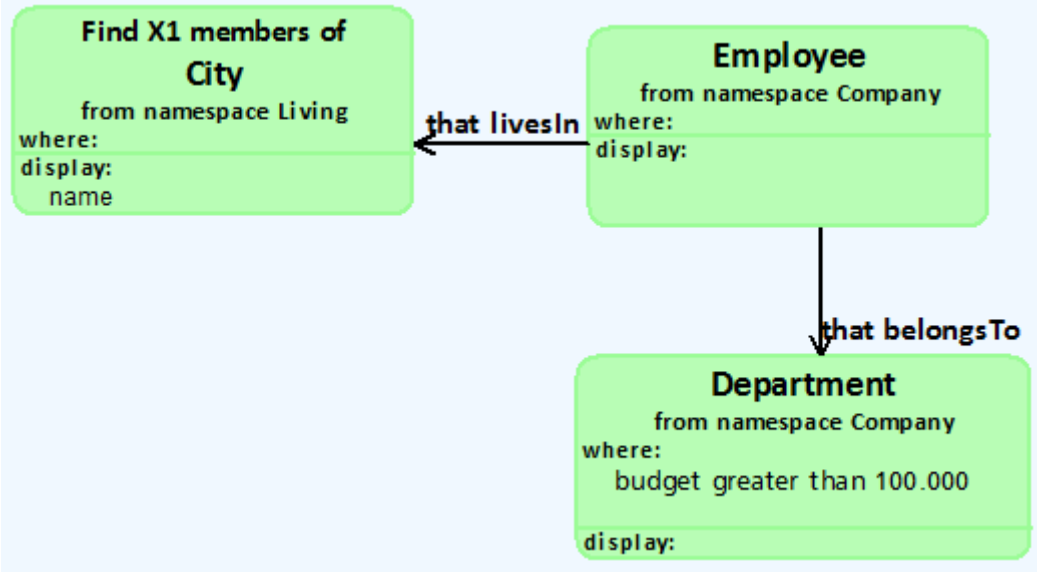
Find name, surname, department name of all employees belonging to marketing department. (Ok)

you are looking for an employye that has 2 employyes in marketing. You will get name, and surname (Problems with variables)

find the name and the surname of employees of namespace company that belong to the marketing department. (No department name in answer)

The name and surname of an employee from the Marketing department at a specified company. (No department name in answer, problems with namespace)

find x1members of employee from name space company where is none display the name,surname,age that belongs to find x2members of department from name space company where area is marketing display the name



Find Cities where lives employees from departments with budget greater than 100.000

What are the names of the namespace company employees who live in city and work in a namespace department with a budget higher than 100,000? (Find employee name, not a city name)

Find employees who live in the city in departments with a budget greater than $100,000. The data the query will provide is the name of employees who live in the city and work in departments with budgets greater than $100,000/ (Find employee name, not a city name)

Find the names of employees who meet two criteria: they live in a particular city, and their department's budget is over 100,000. (Find employee name, not a city name)

Find an employee from a department with a budget exceeding 100.000 who lives in this city. (Find employee name, not a city name)

Name employees that are lives in city and belongs to a department with a budget greater than 100,000 (Find employee name, not a city name)

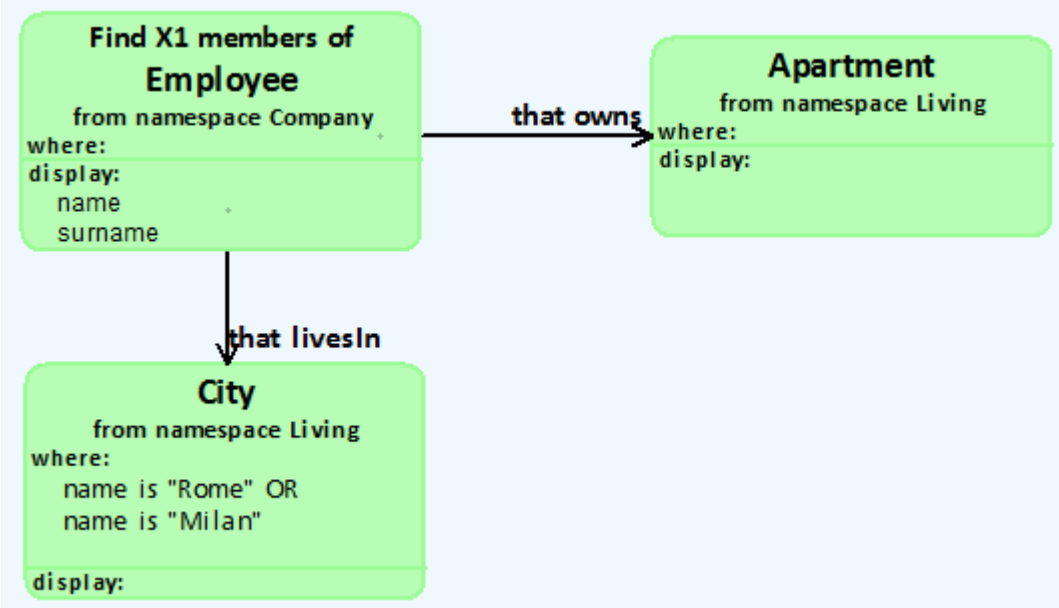
Find the name of the city that employee lives if the employee belongs to a department with a budget greater than 100,000 (OK)

looking for someone in a specific city who is an employye of the company and has a certain budget. You will get name, position and budget (Thinks about employee and find some attributes that is not needed)

find employees of namespace company that live in a particular city and belong to a department with a budget greater than 100.000. (Find employee name, not a city name)

An employee from a company's department having a budget greater than 100,000, who lives in a specific city. (Find employee name, not a city name)

find x1members of city from name space living where display the name that lives in employee from name space company where and display is none that belongs to department from namespace company where budget greater than 100.000 display is none



Find employees that owns apartments and lives in Rome or Milan.

What are the full names of the namespace company employees who own an apartment in Rome or Milan? (Misleading states that apartment is owned in the same places as lives)

Find employees who live in an apartment in either Rome or Milan. The data the query will provide is the name and surname of employees who live in an apartment in either Rome or Milan. (No addition about info that apartment is owned and also combines)

Find the full names of employees who live in apartments in either Rome or Milan. (combaines)

Find an employee who owns an apartment in this city. (combaines)

Find the name and surname of employees that own a apartment and lives in Rome or Milan (Ok)

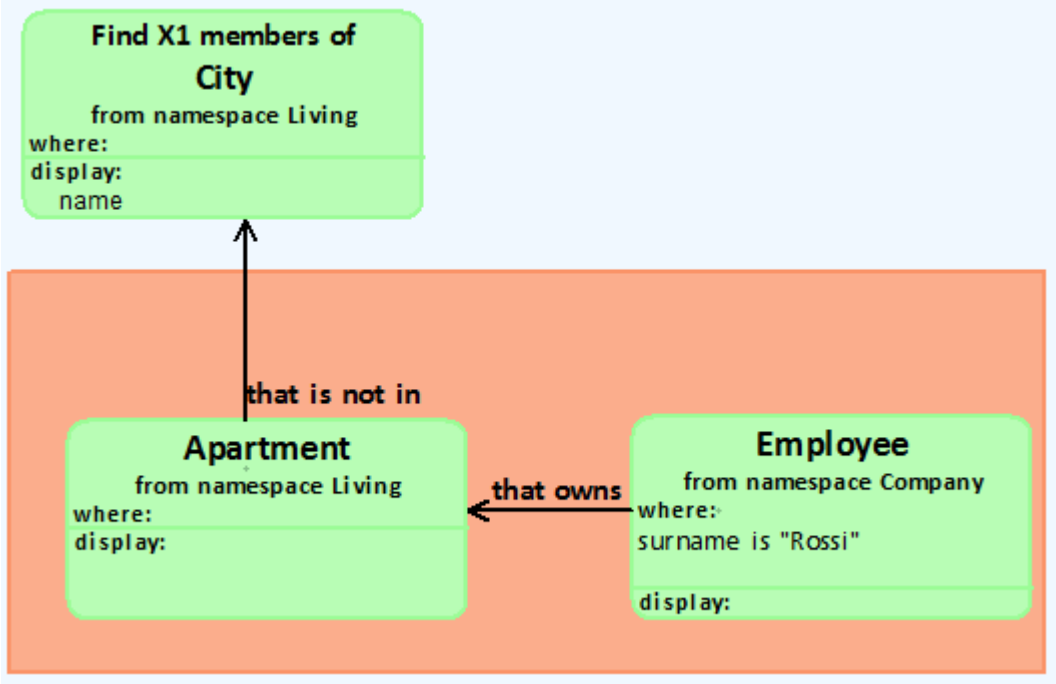
Find the name and surname of employees who live in Rome or Milan and own an apartment. (Ok)

you are looking for an employyee who lives in an apartment in a certain city. You will get name, surname and location (too many attributes, and no specificē conditions)

Find the name and surname of an employee that owns a particular apartment in Rome or Milan. (combaines)

The name and surname of an employee at a specific company, who lives in Rome or Milan and owns an apartment. (Ok)

find x1 members of employee from name space company where display the name surname that owns apartment from name space living where and display is none,find x1 members of employee from name space company that lives in city from name space living where name is rome or milan and display is none



Find cities where is not any apartment owned by Rossi.

What are the names of the employees that are not named Rossi and do not live in an apartment in the city? (red box as negation to attribute? and starts Employee)

Fine an employee who's surname is Rossi who owns an apartment but is not living in a city. The data the query will provide is the name of the employee with the surname Rossi who owns an apartment but does not live in a city. (Starts from Employee)

Find the people who live in a city that aren't in an apartment owned by someone with the last name Rossi. (Query is ok, but problems that finds people(members) not a City)

Find an employee named Rossi who owns an apartment but doesn't live in this city. (Starts from Employee)

Find employees with the surname Rossi who own an apartment that isn't in the city. (Starts from Employee)

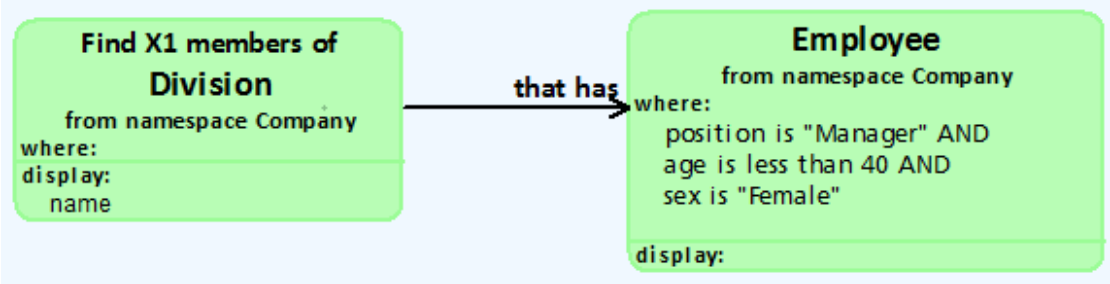
Find employees with a surname 'Rossi' and own an apartment that is not in a city (Starts from Employee)

you are looking for a city that has an apartment where an employye lives. You will get name (Ok, for city but problems that ignores negation)

Find Employees with the surname Rossi that own a particular apartment that is not in a a particular city. (Starts from Employee)

Company Employees with the Surname Rossi that own an apartment that is not in a certain city. (Starts from Employee)

find x1 members of city from name space company where display the name that is not an apartment from name space living where and display is none,that owns employee from name space company where surname is rossi and display is none



Find division that has manager female younger than 40.

What are the names of the female employees of namespace who are managers and are younger than 40? (find employees not division)

Fine employees who are managers who are females and under 40 years old and in a division. The data the query will provide is the name of employees in a division who are managers and females under 40 years old. (find employees not division)

Find the female managers of a division who are less than 40 years old. (find employees not division)

Find a female employee from this division who is a manager and under 40. (find employees not division)

Find the name of division members that are managers, whose age is under 40, and is female. (find employees not division)

Find the names of division which has an female manager with less than 40year of age. (ok)

you are looking foe a division that has an employee that is an 40 year old femaile manager. You will get name (ok)

Find employees of a particular division that are mangers, who are less than 40 and female. (find employees not division)

The names of employees within a specified division of a company who are female managers under the age of 40. (find employees not division)

find x1 members of division from name space company

Find Smith and all employees that has salary less than Smiths salary.

What employees of namespace company named Smith who make less than a X1 salary? (Problems with referencing to a variable)

Find employees who's salary is less than the employee with the surname Smith. The data the query will provide is the name, surname, and salary of employees who's salary is less than the employee with the surname Smith. (Ok)

Find employees named Smith, then find the full names and salaries of employees who make less money than people named Smith make. (Ok)

Find an employee from this company named Smith and two employees who earn less than X1 salary. (Problems with variables)

Find employees who have the surname Smith and give the name ,surname, and salary of employees whose salary is less than the surname Smith employees. (Ok)

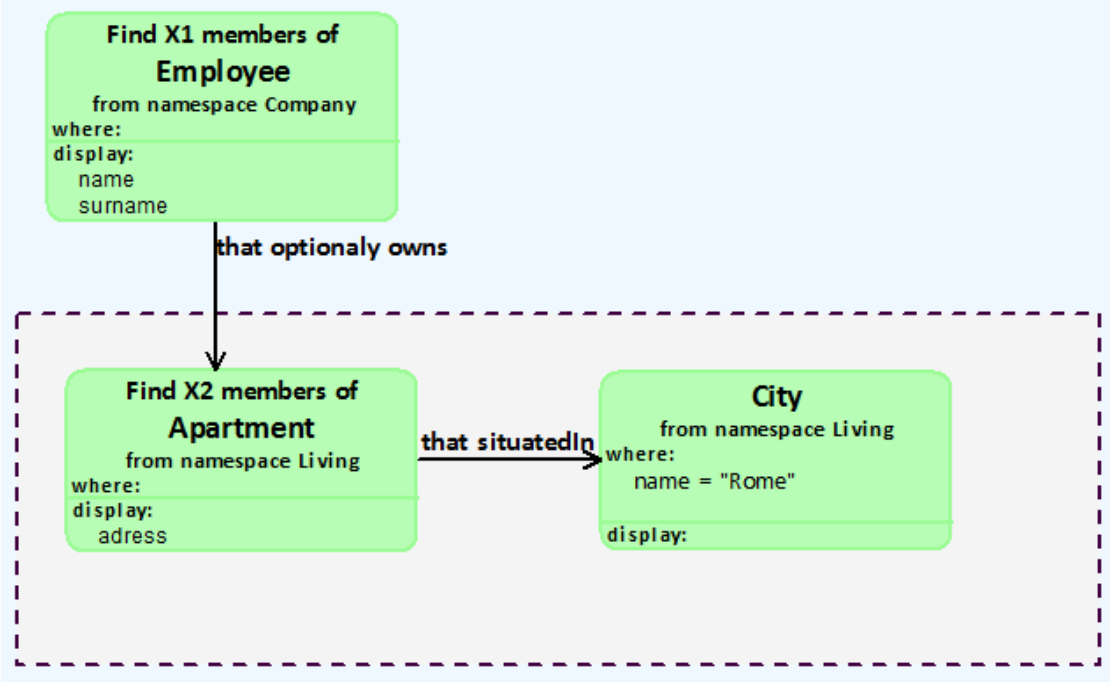
Find the name, surname and salary of employees who earn salary less than an employee with surname as Smith. (Ok)

you are looking for an employye that has two employees that make a specific salary. You will get name, sur name and salary (Problems with variables)

Find employees whose surname is Smith. Then find name, surname, and salary of employees who make less than employees named Smith. (Ok)

The name, and salary of a company's employees with the surname Smith whose salary is less than X1 of salary. (Problems with referencing to a variable)

find x1 members of employee from name space company



Find Employees and if owns apartment in Rome then show also it.

What are the full names of the namespace employees who live in an apartment in Rome? (Ignores optionally and no second answer class)

Find employees that optionally own an apartment that is in Rome. The data the query will provide is the name, surname, and address of employees who optionally own an apartment in Rome. (Not clearly optional thing expressed)

Find the full names of people who might own apartments in Rome. (Is more like list people that we suspect that owns appartment)

Find an employee who might own an apartment in Rome. (Is more like list people that we suspect that owns appartment)

Gibe the name,surname, and address of employees that optionally owns an apartment that is situated in Rome (Ok, but not clearly optional expressed)

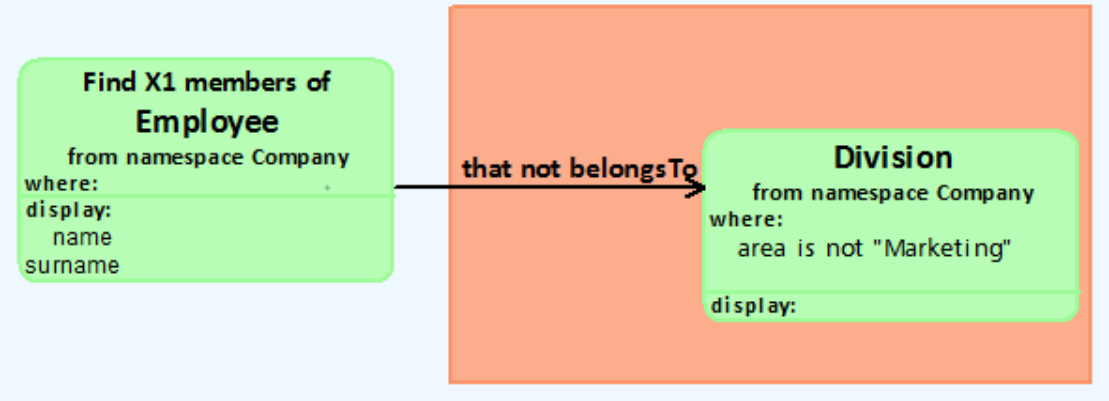
Find the name, surname and address of employees who have atleast an apartment in the Rome city (No optional expressed)

you are looking for an employee that lives in an apartment in a city. You will get name, address and city (No Rome mentioned and find city)

Find the name and surname of employees that optionally owns an apartment in Rome. (Mise address and question about optional possibility)

The name and surname of a company's employee who optionally owns an apartment in Rome. (Mise address and question about optional possibility)

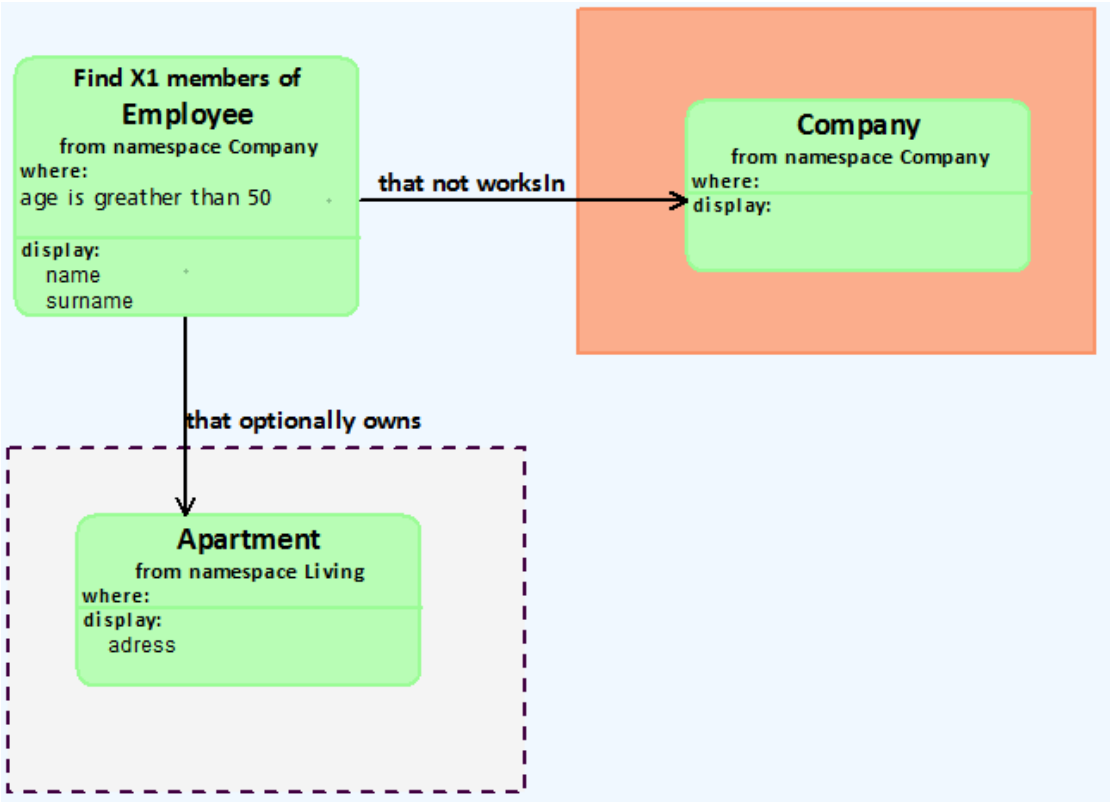
find x1 members of employee from name space company



Find employees that is not in Marketing division.

What are the full names of the namespace employees that do not work in marketing? (+/- Ok)
Find employees who do not work in a marketing division. The data the query will provide are the name and surname of employees who do not work in a marketing division. (Ok)
Find the full names of people who work in marketing. (Double negative) (Double negative?)
Find an employee who does not work in the marketing division. (Ok)
Give the name and surname of employees that aren't in the marketing division.
Find name and Surname of employees who do not belong to Marketing area. (Ok)
you are looking for an employee in a division that is not markerting. You will get name, sur name and division (Ok)
find the name and surname of employees that do not work in the marketing division. (Ok)
The name and surname of an amployee at a company who is not in the marketing division. (Ok)
find x1 members of employee from name space company

(Nepareiza bilde, bet viens ir pamanījis dubulto negāciju)



Find employees older than 50 that does not work anywhere and if they own apartment then show also info about it.

What are the full names of the namespace employees who own an apartment and are older than 50 years old? (no mention about not having work)

Find employees who are older than 50 that optionally own an apartment but do not work in the company. The data the query will provide are the name, surname, and address of employees over 50 who optionally own an apartment but do not work in the company. (Ok)

Find the full names of employees over 50 who don't work for the company and who may live in apartments. (No apartment address)

There are no employees at this company under the age of 50 who own their own apartments. (Refolmulated as a constraint)

Find the name,surname, and address of employees over the age of 50 that optionally own an apartment but isn't in the company. (Ok)

Find the name, surname and address of employee who do does not work for the company but may have an apartment. (Ok)

you are looking for an employee that works for a company in a city. You will get name, surname comapny and address (Where is city?)

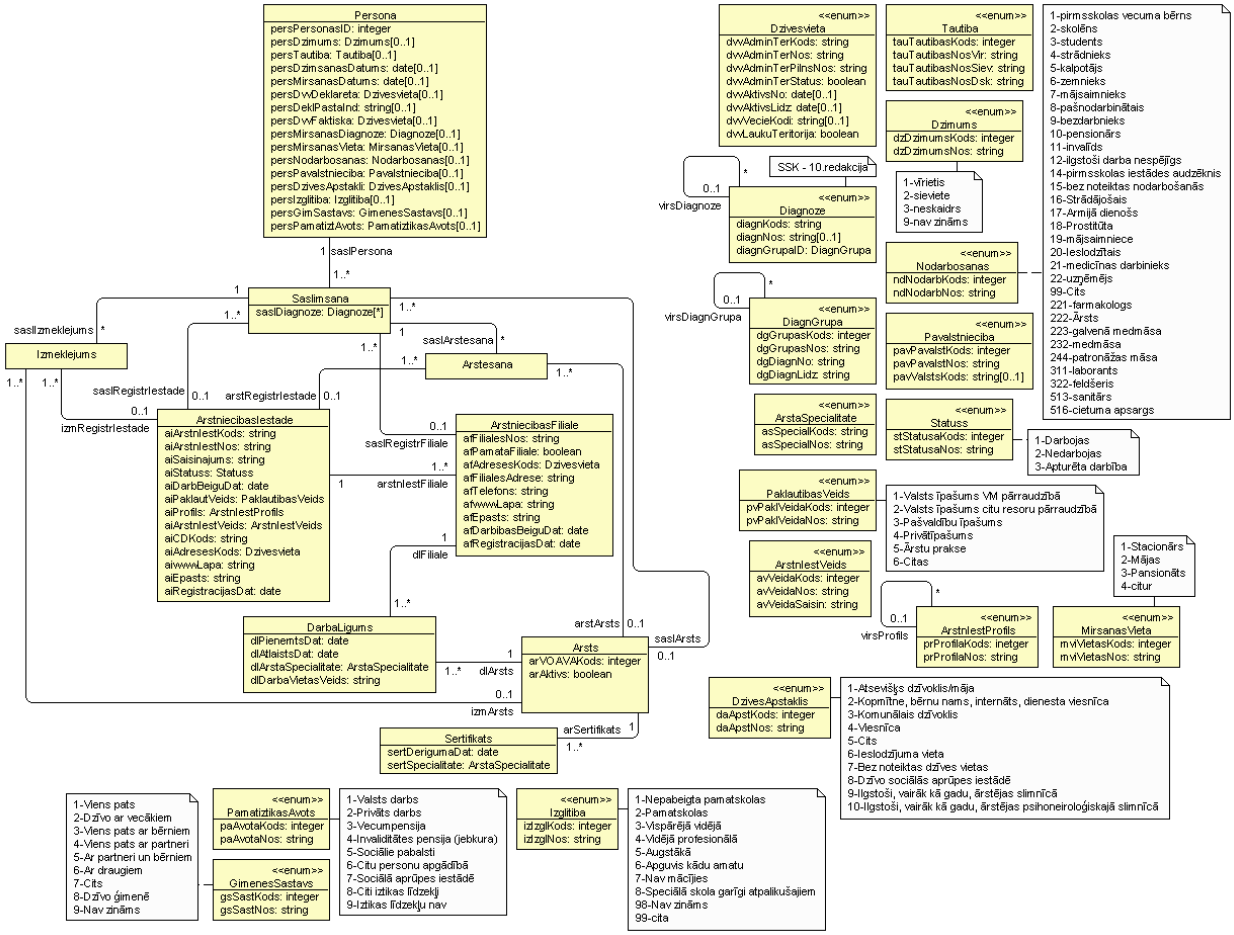
find the name and surname of employees older than 50 that optionally owns an apartment that does not work in namespace company. (No address)

The name and surname of a company's employee over the age of 50 that does not work for a specified company but optionally owns an apartment. (No address)

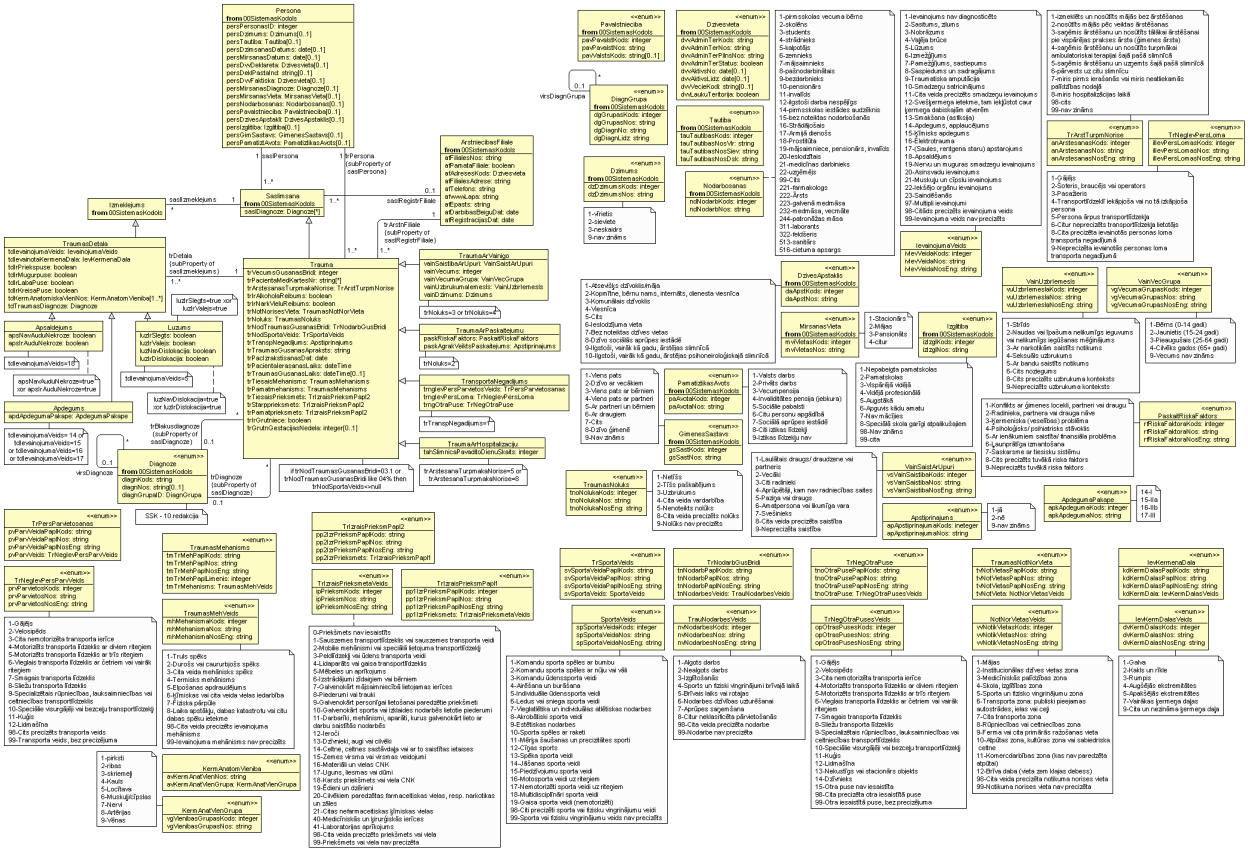
find x1 members of employee from name space company

3. PIELIKUMS

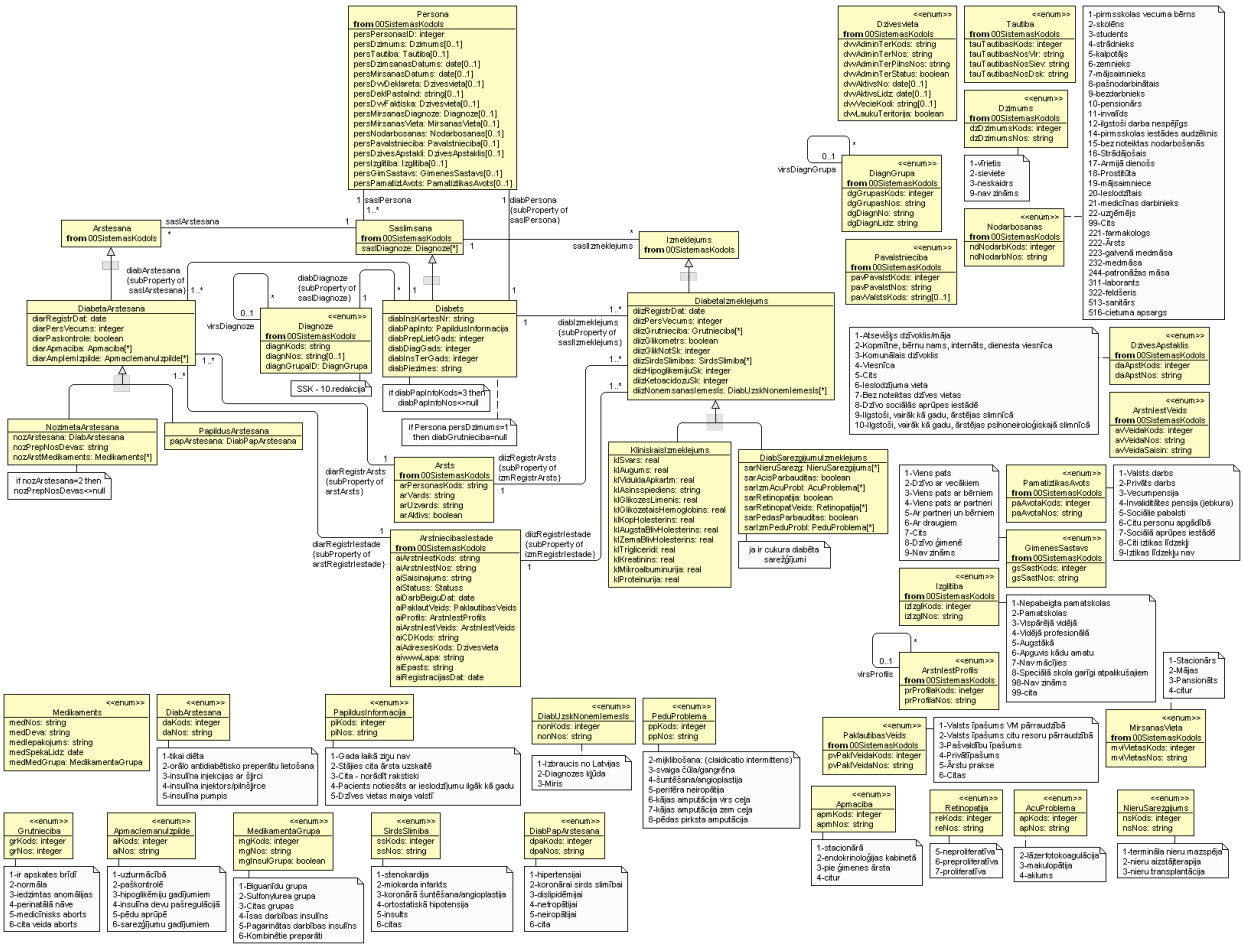
Sistēmas kodols



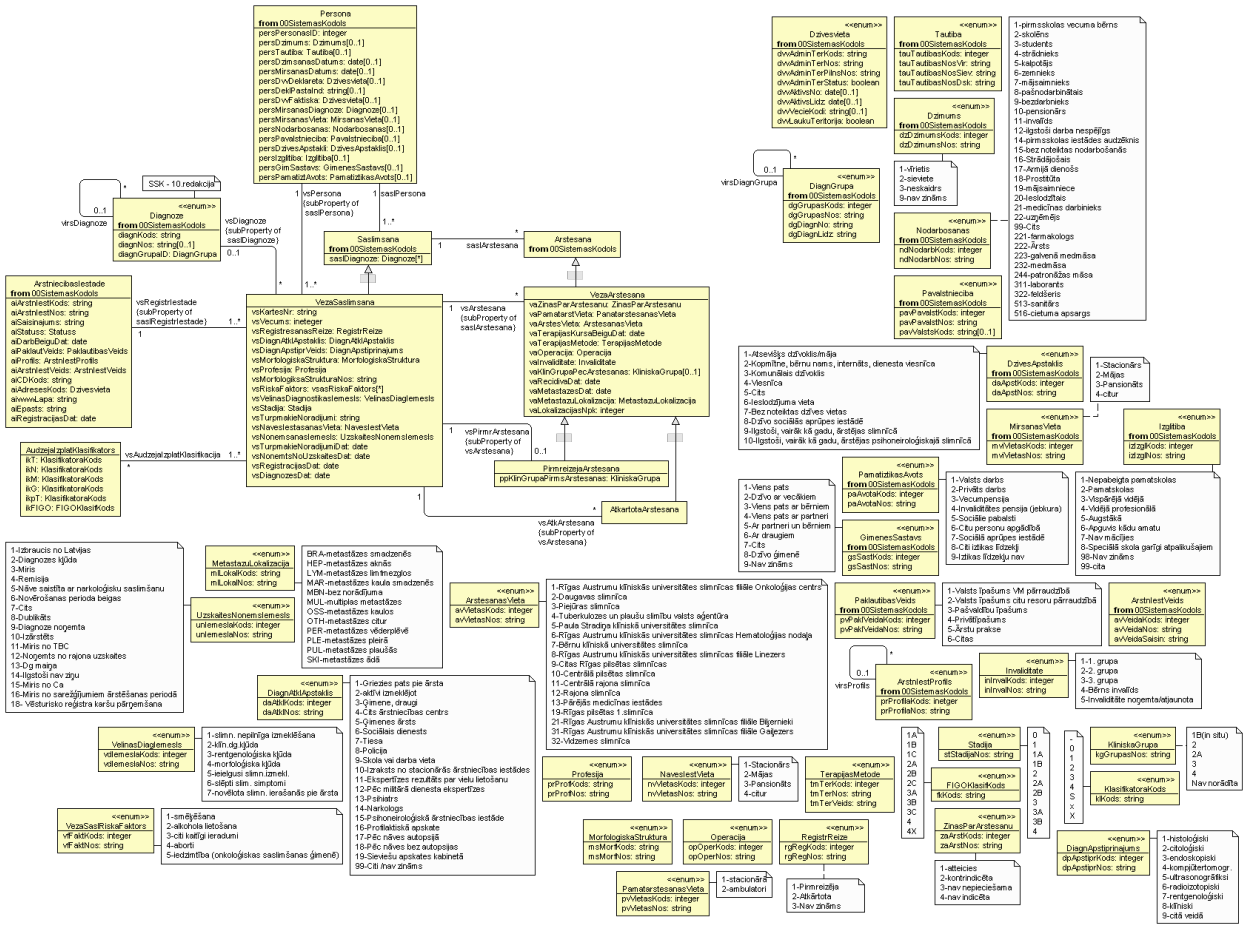
Traumu reģistrs



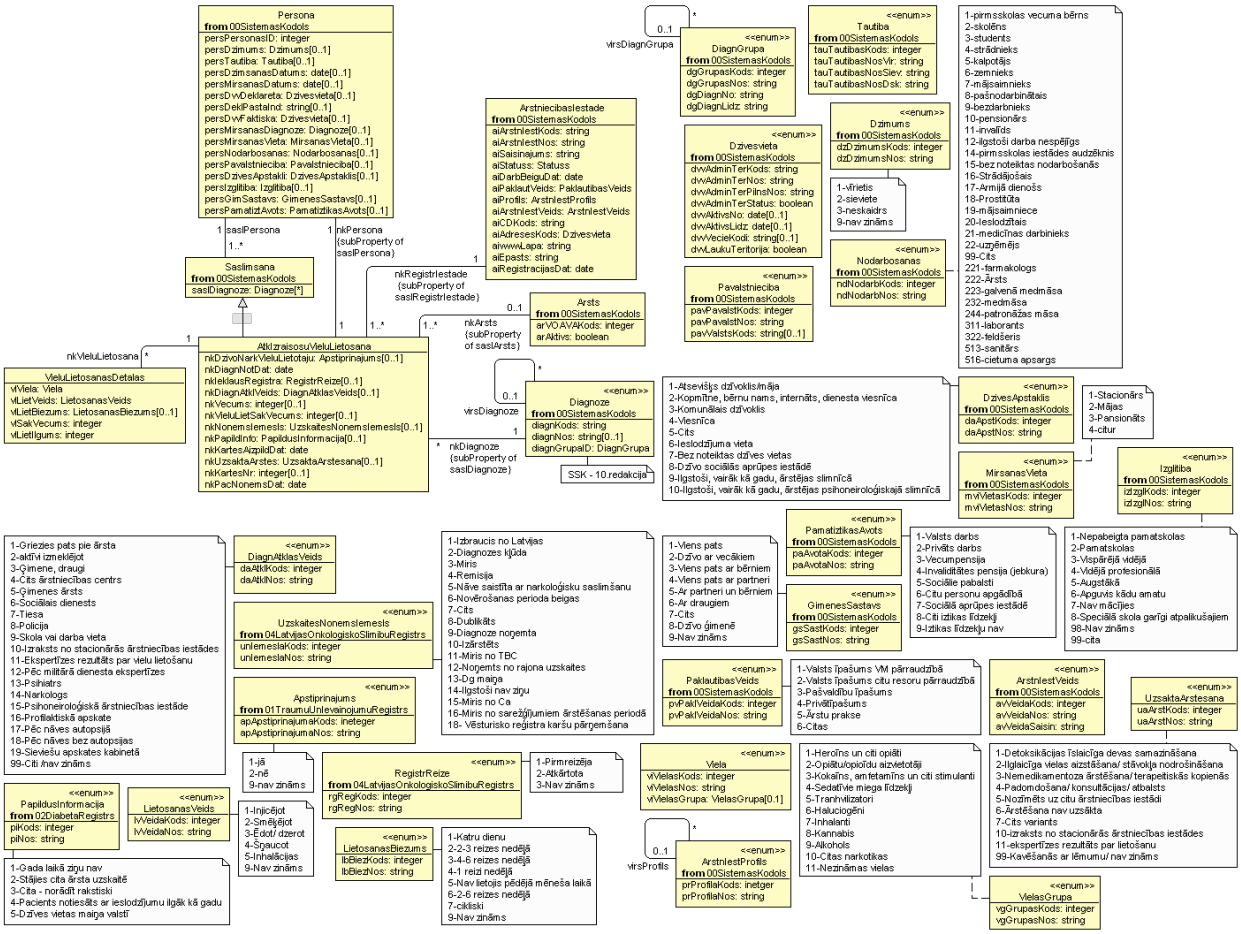
Diabēta reģistrs



Veza reģistrs



Narkaloģijas reģistrs



Psihisko traucējumu reģistrs