Kauno technologijos universitetas

Informatikos fakultetas

Veiklos žodynų ir taisyklių transformavimo į duomenų modelius metodas

Baigiamasis magistro projektas

Lina Bružaitė
Projekto autorė

lekt. Algirdas Šukys
Vadovas

Kaunas, 2019

Kauno technologijos universitetas

Informatikos fakultetas

Veiklos žodynų ir taisyklių transformavimo į duomenų modelius metodas

Baigiamasis magistro projektas

Informacinių sistemų inžinerija (6211BX009)

Lina Bružaitė
Projekto autorė

lekt. Algirdas Šukys
Vadovas

prof. dr. Audrius Lopata
Recenzentas

Kaunas, 2019

Kauno technologijos universitetas

Informatikos fakultetas

Lina Bružaitė

Veiklos žodynų ir taisyklių transformavimo į duomenų modelius metodas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Linos Bružaitės, baigiamasis projektas tema „Veiklos žodynų ir taisyklių transformavimo į duomenų modelius metodas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka)	(parašas)

Bružaitė, Lina. Veiklos žodynų ir taisyklių transformavimo į duomenų modelius metodas. Magistro baigiamasis projektas / vadovas lekt. Algirdas Šukys; Kauno technologijos universitetas, Informatikos fakultetas.

Mokslo kryptis ir sritis: Informatikos inžinerija, technologijos mokslai

Reikšminiai žodžiai: *veiklos žodynas, veiklos taisyklės, SBVR, duomenų modeliai, modelių transformcija*

Kaunas, 2019. 91 p.

SANTRAUKA

Siekiant užtikrinti informacinių sistemų kūrėjų ir užsakovų tarpusavio supratimą, naudojami veiklos žodynai ir taisyklės, kurie gali būti aprašomi struktūruota natūraliąja kalba. Blogai aprašytos ar interpretuotos taisyklės tampa informacinių sistemų klaidų priežastimi. Siekiant išvengti žmogiškojo faktoriaus klaidų, ar greičiau prisitaikyti prie kintančių organizacijos procesų, ieškoma būdų automatizuoti šį procesą. Po atliktos esamų sprendimų analizės, realizuotas naujas sprendimas - veiklos žodynų ir taisyklių transformavimo į duomenų modelius metodas, kuris leidžia išplėsti duomenų modelių kūrimo galimybes transformuojant duomenų bazėje veikiančias veiklos taisykles. Metodas realizuotas modeliais grindžiamos inžinerijos pagrindu, naudojant SBVR ir šiame darbe aprašytą DDL metamodelius. Testavimo ir eksperimento rezultatai parodo, kad sukurtas metodas gana korektiškai atpažįsta veiklos žodyno konceptus ir struktūrines veiklos taisykles. Ateityje tobulinant metodą siūloma papildyti SBVR metamodelį bei sukurti sąsają su duomenų bazių valdymo sistema.

Bružaitė, Lina. *Solution for Transforming SBVR Specifications to Data Models*: Master‘s thesis in Information Systems Engineering / supervisor lect. Algirdas Šukys. The Faculty of Informatics, Kaunas University of Technology.

Research area and field: Informatics Engineering, Technology Science

Key words: *business rules, business vocabulary, SBVR, data models, model transformation*

Kaunas, 2019. 91 p.

SUMMARY

In order to maintain a mutual understanding between information system developers and their clients, business vocabularies and rules are used. These can be described using structured natural language. Poorly described or misinterpreted business rules lead to errors in information systems. To avoid human made mistakes or to quickly adapt to rapidly changing organizational processes, new ways of automating this process need to be found. After an analysis of existing solutions, a new solution has been implemented. It allows for the expansion of current data model creation possibilities by transforming business rules in the database. The method is based on model driven architecture using SBVR and DDL metamodels. The test and experiment results show that the created method recognizes the concepts of business vocabularies and structured rules correctly. Future improvement of this method should be carried out by expanding the SBVR metamodel and creating a connection to a database management system.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

OMG – (angl. *Object Management Group*) tarptautinė standartų organizacija, kurios tikslas sukurti modeliavimo standartus programoms, sistemoms ir veiklos procesams

SBVR – (angl. *Semantics of Business Vocabulary and Business Rules*) OMG sukurtas VT apibrėžimo standartas, orientuotas į semantinę natūraliosios kalbos sakinių, kuriais išreiškiamos VT, struktūrą

Reliacinis modelis – duomenų aprašymo modelis, kuris remiasi predikatų logika bei aibių teorija.

SQL (struktūrinė užklausų kalba) - apima duomenų bazės struktūros kūrimą , duomenų saugojimą, peržiūrą bei manipuliavimą duomenimis reliacinėse duomenų bazėse.

Modalumas – teiginių svarstymo būdas; teiginių klasifikavimas pagal charakteristikas, kurie gal būti būtinai teisingi, galbūt teisingi, privalomi, leistini ir pan.

ER modelis (angl. *entity–relationship model*)- esybių ryšių modelis.

DBVS (Duomenų bazių valdymo sistema) - kompiuterinė programa ar programų paketas, skirtas duomenų bazės valdymui.

ATL (angl. *ATLAS Transformation Language*) – modelių transformavimo kalba.

Acceleo – kodo generatorius, skirtas vykdyti modelių transformavimą į tekstą.

ĮVADAS

Darbas priklauso informacinių sistemų inžinerijos magistrantūros studijų programai.

Darbo problematika ir aktualumas

Informacinių sistemų kūrimo proceso metu siekiama suprojektuoti, realizuoti ir ištestuoti informacines sistemas. Kuriama sistema, kuri patenkintų ar viršytų klientų pageidavimus ir neviršytų laiko ir kainos apribojimų. Pagrindinė informacinės sistemos paskirtis yra informacijos perdavimas, saugojimas, manipuliavimas bei atvaizdavimas.

Sistemos kūrimo procesas susideda iš nuoseklių etapų: reikalavimų analizės, projektavimo, kūrimo, testavimo, diegimo ir palaikymo. Kiekvieno etapo rezultatai būtini sekančiam etapui vykdyti. Dalių išdėstymas priklauso nuo pasirinktos projekto valdymo metodikos ir gali skirtis eiliškumas, tačiau projektavimas ir kūrimas yra glaudžiai tarpusavyje susiję etapai, kurie eina vienas po kito. Šiame darbe daugiausia dėmesio bus skiriama duomenų modelio sudarymui, kuris vykdomas šių etapų metu.

Viena iš didžiausių problemų kuriant informacinę sistemą yra komunikacijos stygius tarp užsakovų ir sistemos kūrėjų. Klientai turi viziją, kaip turi veikti sistema, tačiau ne visada supranta sistemos modelių, duomenų bazės schemos. Jiems sunku patvirtinti, ar suprojektuota sistema jiems yra tinkama. Komunikacija su kūrėjų komanda turi būti pakankamai griežta tam, kad būtų palaikomas tvarkingas sistemos projektavimas ir kodo kūrimas. Siekiant užtikrinti vienareikšmišką suprantamumą tarp sistemų analitikų, informacinių sistemų kūrėjų, užsakovų bei kitų dalykinės srities specialistų, naudojami veiklos žodynai ir taisyklės, kurie gali būti aprašomi struktūruota natūraliąja kalba. Natūraliąja kalba aprašytos specifikacijos gali būti transformuojamos į programinį kodą.

Kita problema, su kuria susiduriama projektuojant ir eksploatuojant informacines sistemas – kintančios veiklos procesų ir apribojimų taisyklės (kitaip vadinamos veiklos taisyklėmis), bei veiklos žodynas. Veiklos taisyklės užtikrina organizacijoje esančių veiklos procesų ir apribojimų atitikimą sistemos veikimui, bei informacijos kokybę. Modeliuojant informacinę sistemą pasitaiko, kad kuriamos sistemos modelių ir funkcionalumo nespėjama suderinti su pasikeitusiomis taisyklėmis. Informacinės sistemos derinimo ar kūrimo metu tenka veiklos žodyną ir taisykles patikslinti, papildyti ar kitaip atnaujinti. Organizacijoms nuolat keičiantis, sistemos turi gebėti nesunkiai prisitaikyti prie atsiradusių pokyčių. Tačiau pakeitimai neturėtų sukelti prieštaravimų tarp esamų veiklos taisyklių.

Kai kurios veiklos taisyklės yra labai svarbios projektuojant sistemos duomenų bazę, kadangi jos glaudžiai susijusios su duomenų bazės logine schema. Siekiant užtikrinti duomenų teisingumą bei vientisumą, būtina į taisykles atsižvelgti nuo pat sistemos kūrimo pradžios. Duomenų bazės apribojimų nustatymas, kuriant informacines sistemas, nėra automatizuotas, todėl analizė bei kūrimas ilgiau užtrunka. Be to, atsiranda klaidų galimybė. Šiame darbe bus kuriamas metodas, leisiantis aprašyti organizacijos žodyną ir taisykles bei transformuoti juos į duomenų modelius.

Darbo tikslas ir uždaviniai

Tiriamojo darbo tikslas yra išplėsti duomenų modelių kūrimo galimybes, vykdant transformacijas iš natūralia kalba aprašytų veiklos žodynų ir taisyklių.

Tikslui įgyvendinti išsikelti šie uždaviniai:

1. išanalizuoti:
    - informacinių sistemų kūrimo procesą;
    - veiklos žodynų ir taisyklių sudarymo principus, *SBVR* standartą;
    - su duomenų bazėmis susijusias kalbas ir technologijas (t.y., *DDL*, *SQL* ir kt.);
    - esamus veiklos žodynų ir taisyklių transformavimo į duomenų modelius sprendimus;
    - taisyklėmis išreikštų apribojimų realizavimą duomenų bazės priemonėmis;
    - modelių transformavimo technologijas;
2. sukurti metodą, leidžiantį aprašyti veiklos žodyną ir taisykles bei transformuoti į duomenų modelius;
3. realizuoti sprendimo prototipą;
4. atlikti eksperimentą, įvertinti tyrimo rezultatus.

Darbo rezultatai ir jų svarba

Atlikta esamų sprendimų analizė ir sukurtas veiklos žodynų ir taisyklių transformavimo į duomenų modelius metodas, leidžiantis transformuoti duomenų bazėje veikiančias veiklos taisykles.

Darbo struktūra

Darbą sudaro šios pagrindinės dalys:

- duomenų bazės kūrimo proceso analizė;
- veiklos žodyno ir taisyklių užrašymo analizė;
- modelių transformavimo kalbų analizė;
- kuriamo metodo reikalavimų analizė ir projektavimas;
- sprendimo realizacija;
- eksperimentas bei rekomendacijos;
- išvados ir pastebėjimai.

1. VEIKLOS TAISYKLIŲ TRANSFORMAVIMO Į DUOMENŲ MODELIUS PROCESO ANALIZĖ

1.1. Analizės tikslas

Analizės tikslas – išanalizuoti veiklų žodyno ir taisyklų užrašymą struktūrizuota kalba pagal *SBVR* standartą, jų transformavimą į *SQL* užklausas ir duomenų modelius, esamus sukurtus įrankius, kurie padėtų patobulinti automatinį veiklos taisyklių transformavimą į duomenų modelius.

1.2. Tyrimo objektas, sritis ir problema

Tyrimo problema

Veiklos žodyno ir taisyklių kūrimas, priežiūra ir vientisumo užtikrinimas tampa sudėtingas uždavinys, kai veiklos taisyklių sukuriama pakankamai daug. Jeigu taisyklės aprašo painius ir sudėtingus procesus, jų priežiūros sudėtingumas kylant taisyklių skaičiui gali didėti eksponentiškai. Blogai aprašytos ar interpretuotos taisyklės tampa informacinių sistemų klaidų priežastimi. Dėl šios priežasties ieškoma būdų, kaip optimizuoti šį procesą išvengiant žmogiškojo faktoriaus klaidų. Veiklos taisyklių analizei gali būti naudojamas standartizuotas tekstas, kuris leidžia žodyną ir taisykles sieti automatiškai. Jeigu standartizuotos taisyklės perkeliamos į programinį kodą rankiniu būdu, tikimybė įvelti klaidų vis dar išlieka pakankamai didelė. Informacinės sistemos taip pat pasižymi nuolatiniais pokyčiais, todėl ypač svarbus greitas jų prisitaikymas prie kintančių veiklos taisyklių.

Problema aktuali tiek verslo atstovams, tiek informacinių sistemų kūrėjams, kurie kurdami remiasi taisyklių dokumentais. Veiklos taisyklės padeda suderinti informacinės sistemos ir organizacijos veiklą. Kadangi dalis taisyklių logikos atsiranda duomenų bazės įrašuose, šiame darbe dėmesys kreipiamas į duomenų modelių sudarymo procesą.

Tyrimo objektas

Duomenų modelių sudarymo procesas. Tyrimo metu bus analizuojami duomenų modelių sudarymo principai, nagrinėjami galimi sąryšiai tarp duomenų bazės schemos ir veiklos žodyno ir taisyklių specifikacijų.

Tyrimo sritis

Veiklos žodynų ir taisyklių specifikavimas, duomenų bazių projektavimas, modeliais grindžiamos transformacijų technologijos.

1.3. Informacinių sistemų kūrimo procesas

Informacinės sistemos vystymosi gyvavimo ciklas prasideda jos idėja ir baigiasi klientui pateikta bei patvirtinta IS galutine versija. Kaip jau minėta anksčiau, kartu su šiuo procesu vyksta įvairūs etapai: planavimas, analizė, projektavimas, programavimas, testavimas, diegimas bei priežiūra (1.1 pav.).

1.1 pav. Sistemos kūrimo gyvavimo ciklas [1]

Analizės etapas apima vartotojų poreikių analizę ir projekto tikslų tobulinimo galimybes pagal numatytas sistemos funkcijas ir operacijas. Trys pagrindinės analizės etape veiklos rūšys yra:

- specifikuoti reikalavimus,
- nubraižyti veiklos diagramas,
- atlikti pirkimo / kūrimo naudingumo analizę [2].

Viena iš svarbiausių analizės fazės sudedamųjų dalių yra reikalavimų užrašymas. Projekto sėkmė tiesiogiai priklauso nuo surinktų reikalavimų detalumo. Šiame etape yra visiškai nesvarbu įgyvendinimo bei techninės detalės. Pavyzdžiui, nesvarbu sistemos kūrimui naudojamos technologijos, ar tai būtų *Oracle* duomenų bazė, ar *Java* programavimo kalba. Pagrindinis dėmesys yra skiriamas užsakovo poreikiams rinkti. Svarbiausias uždavinys yra sužinoti, ką realizuoti, o ne realizacijos subtilybės.

Programinės įrangos inžinerijos institutas (angl. *Software Engineering Institute*, SEI) apibrėžė reikalavimų valdymo procesą, kaip kliento poreikių užtikrinimą, kuriam turi būti skiriamas dėmesys nuo pat projekto pradžios iki pristatymo. Visi pakeitimai turi būti stebimi su atitinkamais projekto dokumentų ir planų pakeitimais. Reikalavimų valdymas yra projektų reikalavimų identifikavimo, dokumentavimo, perdavimo, stebėjimo ir valdymo procesas, taip pat tų reikalavimų pakeitimas. Tai nėra vienetinis įvykis sistemos kūrime, o greičiau tęstinis procesas. Jeigu nėra skiriamas pakankamas dėmesys sistemos reikalavimams, tai gali būti pirmas žingsnis link didesnių nei planuota išlaidų, projekto teikimo vėlavimų ar klaidingų specifikacijų.

Svarbiausias žingsnis nustatant projekto reikalavimus yra aktyvus užsakovo įtraukimas į reikalavimų rašymo procesą. Labai svarbu, kad projekto vadovas kartu su sistemos kūrėjų komanda gautų tinkamus veiklos reikalavimus. Sistemos naudotojai peržiūri reikalavimų dokumentą ir nustato, ar sutinka su užfiksuotais reikalavimais.

Veiklos žodynų ir taisyklių naudojimas šiame procese techniškai reiškia kūrimo procesą perkelti vienu abstrakcijos lygiu aukščiau. Nelieka duomenų bazės ar kodo kūrimo nuo pagrindų. Vietoje to reikia smulkiai aprašyti, ką sistema turi daryti, ir tada iš aprašymo ją generuoti. Informacinės sistemos kūrimo proceso metu atliekami darbai:

- užrašyti taisykles užsakovo nurodytu formatu;

- perduoti dokumentą programuotojams.

Struktūrizuota natūralia kalba veiklos taisyklių užrašymas, bei automatinis transformavimas į programinį kodą, gali pakeisti šį procesą iš esmės. Naujasis variantas:

- nustatyti visas veiklos taisykles ir žodyną;
- užpildyti duomenis sistemoje, kuri geba atlikti modelių transformavimą;
- sugeneruoti iš veiklos žodyno ir taisyklių IS sistemai reikalinga kodą;
- papildyti generuotą kodą, jeigu ne visos taisyklės buvo transformuotos.

Tokiu būdu gali būti generuojamas tiek programinis kodas, tiek duomenų bazės struktūra, diagramos ir t. t. Šiame darbe akcentuojamas duomenų bazės scenarijaus generavimas.

1.4. **Duomenų bazės schemos kūrimas**

Duomenų modeliavimas – tai procesas, kai kuriamas duomenų modelis, skirtas informacinei sistemai, taikant tam tikrus formalius kūrimo metodus. Duomenų modelis atvaizduoja:

- duomenų objektus;
- asociacijas tarp skirtingų duomenų objektų;
- taisykles.

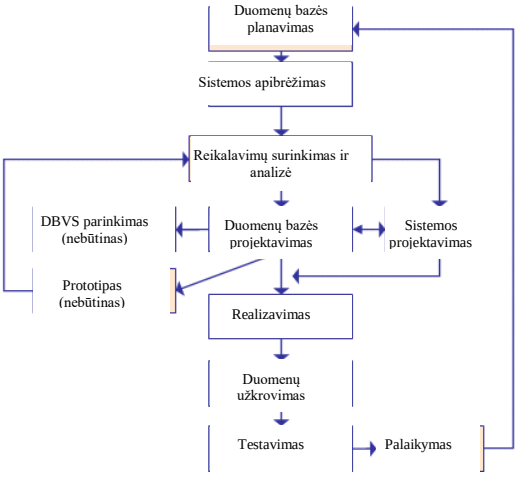
Duomenų modeliavimas padeda vizualiai pateikti duomenų struktūrą ir užtikrina duomenų atitikimą veiklos taisyklėms, apribojimų reikalavimams ar įstatymams. Tai pat užtikrinamas naudojamų vardų nuoseklumas, numatytosios reikšmės, semantika, duomenų saugumas užtikrinant jų kokybę.

Duomenų modelis apibrėžia, kokie duomenys yra būtini ir kaip jis turėtų būti kuriamas, o ne kokias operacijas reikia atlikti. Duomenų modelis yra kaip architekto pastato planas, kuris padeda sukurti konceptualų modelį ir nustatyti duomenų elementų ryšį.

Duomenų bazės projektavimo metu projektuotojas nustato, kokie duomenys turi būti saugomi ir kaip tarpusavyje susiję duomenų elementai. Saugomų duomenų nustatymo procesas paprastai laikomas reikalavimų analizės dalimi. Projektuotojo, kuris atsakingas už duomenų bazės kūrimą, viena iš užduočių yra surinkti informaciją iš asmenų, kurie turi pakankamai kuriamos sistemos srities žinių. Nustačius saugomų duomenų struktūras, modeliuojamos jų tarpusavio priklausomybės. Keičiant duomenis reikia žinoti, kurie susiję duomenys taip pat turės pasikeisti. Tai nustačius galima suskirstyti duomenis į loginę struktūrą, kuri vėliau gali būti susieta su duomenų bazių valdymo sistemos palaikomais saugojimo objektais. Reliacinių duomenų bazių atveju objektai yra lentelės, kuriose saugomi duomenys eilutėse ir stulpeliuose. Objektinėse duomenų bazėje saugojimo objektai tiesiogiai atitinka dalykinės srities objektus. Naudojama objektinė programavimo kalba, kuria rašomos programos, valdančios ir pasiekiančios duomenis. Ryšiai gali būti apibrėžiami kaip susijusių objektų klasių atributai arba kaip metodai, veikiantys objektų klasėse. Šis susiejimas paprastai atliekamas taip, kad kiekvienas susijusių duomenų rinkinys, priklausantis nuo vieno objekto, yra įrašomas į lentelę. Tuomet ryšiai tarp šių priklausomų objektų yra saugomi kaip ryšiai tarp įvairių objektų. Kiekviena lentelė gali būti loginio objekto arba vieno ar daugiau loginių objektų egzempliorių tarpusavio ryšio įgyvendinimas. Tuomet ryšiai tarp lentelių gali būti saugomi kaip nuorodos, jungiančios vaikines

lenteles su tėvinėmis lentelėmis. Kadangi sudėtingiems loginiams ryšiams yra kuriama atskira lentelė, ji galimai turės nuorodų į daugiau nei vieną tėvinę lentelę.

Duomenų bazė paprastai yra pagrindinė informacinės sistemos dalis. Toliau pateiktame paveikslėlyje parodyta, kaip sistemos kūrimo ciklas apima duomenų bazių kūrimą.

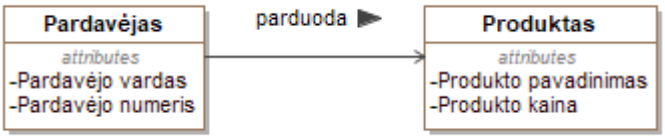


1.2 pav. Duomenų bazės kūrimo procesas [3]

Paveikslėlyje matyti, kad duomenų bazės projektavimo procesas skirstomas į tris dalis:

- konceptualus duomenų bazės projektavimas;
- loginis duomenų bazės projektavimas;
- fizinis duomenų bazės projektavimas.

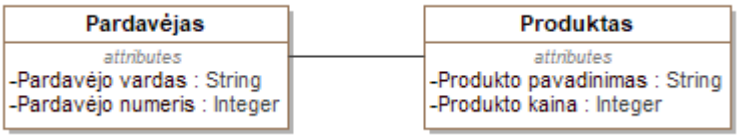
Pagrindinis konceptualaus modelio tikslas yra sukurti subjektus, jų atributus ir jų santykius. Šiame duomenų modeliavimo lygmenyje beveik nėra duomenų apie duomenų bazės realizaciją.



1.3 pav. Konceptualus duomenų modelio pavyzdys

Konceptualūs duomenų modeliai, sukuria bendrą žodyną visiems suinteresuotiesiems subjektams, nustatydami pagrindines sąvokas ir taikymo sritį. Šis modelis grindžiamas sistemos reikalavimais.

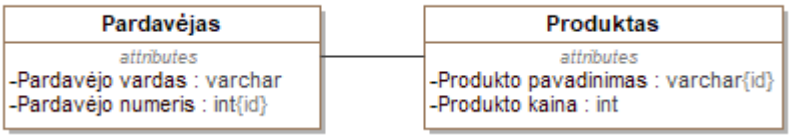
Loginis duomenų modelis prideda papildomos informacijos į koncepcinio modelio elementus. Jis apibrėžia duomenų elementų struktūrą ir nustato jų tarpusavio ryšius.



1.4 pav. Loginio duomenų modelio pavyzdys

Loginio duomenų modelio pranašumas yra sukurtas pagrindas fizinio modelio formavimui. Tačiau modeliavimo struktūra išlieka bendra. Šiuo duomenų modeliavimo lygiu nėra apibrėži pirminiai ar išoriniai raktai.

Fizinių duomenų modelis aprašo duomenų bazės specifinį duomenų modelio įgyvendinimą. Jis siūlo duomenų bazės abstrakciją ir padeda sukurti schemą. Taip yra dėl fizinių duomenų modelio siūlomų meta duomenų gausos.



1.5 pav. Fizinio duomenų modelio pavyzdys

Šio tipo duomenų modelis taip pat padeda vizualizuoti duomenų bazės struktūrą. Jis padeda modeliuoti duomenų bazių stulpelių raktus, apribojimus, indeksus, trigerius ir kitas reliacines DBVS funkcijas.

Šio darbo tikslas yra transformuoti veiklos žodyną ir taisykles į fizinį duomenų modelį. Tai padarius, būtų galima iš karto kurti duomenų bazę iš veiklos žodyno ir taisyklių.

1.5. **SBVR transformavimo į duomenų modelius proceso analizė**

Duomenų transformavimo procesas – tai vieno duomenų formato arba struktūros konvertavimas į kitą formatą arba struktūrą. Tai viena iš esminių užduočių šalinant, saugant ar integruojant duomenis ir programas. Transformavimo procesas paprastai atliekamas tiek rankiniu, tiek automatiniu būdu. Naudojami įrankiai priklauso nuo transformuojamų duomenų formato, sudėtingumo bei apimties. Norint transformuoti veiklos žodyną ar taisykles, reikia jas aprašyti struktūrizuota kalba.

Jeigu veiklos taisyklės neformaliai nustatomos ir dokumentuojamos pradiniuose projekto etapuose, veiklos taisyklių ir žodyno rinkimas yra atsitiktinis. Be to, organizacijos projektai, pvz., naujo produkto įdiegimas arba sudėtingo proceso perplanavimas, gali lemti naujų veiklos taisyklių atsiradimą. Ši atsitiktinio ar atsirandančio veiklos taisyklių rinkimo praktika yra pažeidžiama dėl nesuderinamų ar net prieštaraujančių veiklos taisyklių kūrimo skirtinguose organizaciniuose padaliniuose arba per tą patį organizacinį vienetą. Šis nenuoseklumas sukelia sunkumų, kuriuos gali būti sunku rasti ir išspręsti. Siekdami rasti geriausią veiklos taisyklių rinkimo ir dokumentavimo būdą, verslo analizės ekspertai sukūrė veiklos taisyklių metodiką (angl. *Business Rules Methodology*). Ši metodika apibrėžia veiklos taisyklių priėmimo procesą natūralia kalba, patikrinamu ir suprantamu būdu.

Programos, specifiškai skirtos veiklos taisyklių rinkimui, vykdymui, vadinamos veiklos taisyklių valdymo sistemomis (angl. *Business Rules Management Systems,* BRMSs). Jos naudoja taisyklių užrašymo standartus tam, kad standartizuoti veiklos taisyklių kūrimo elementus. Vienas iš tokių standartų yra SBVR standartas. Veiklos taisyklių manifestas [4] padarė didelę įtaką palengvinant bendravimą tarp žmonių ir informacinių technologijų. Šio dokumento rezultatas yra *OMG* (angl. *Object Management Group,* OMG) sukurtas standartas „Veiklos žodyno ir taisyklių semantika“ (angl. *Semantics of Business Vocabulary and Rules,* SBVR), kuris susideda iš lingvistikos, formalios logikos bei praktinių tyrimų. *SBVR* yra vienas iš pirmųjų *OMG* standartų, leidžiančių kurti išsamias organizacijoms skirtas natūralios kalbos specifikacijas. Pirmoji *SBVR* specifikacijos versija buvo pateikta 2008 m. sausio mėn.

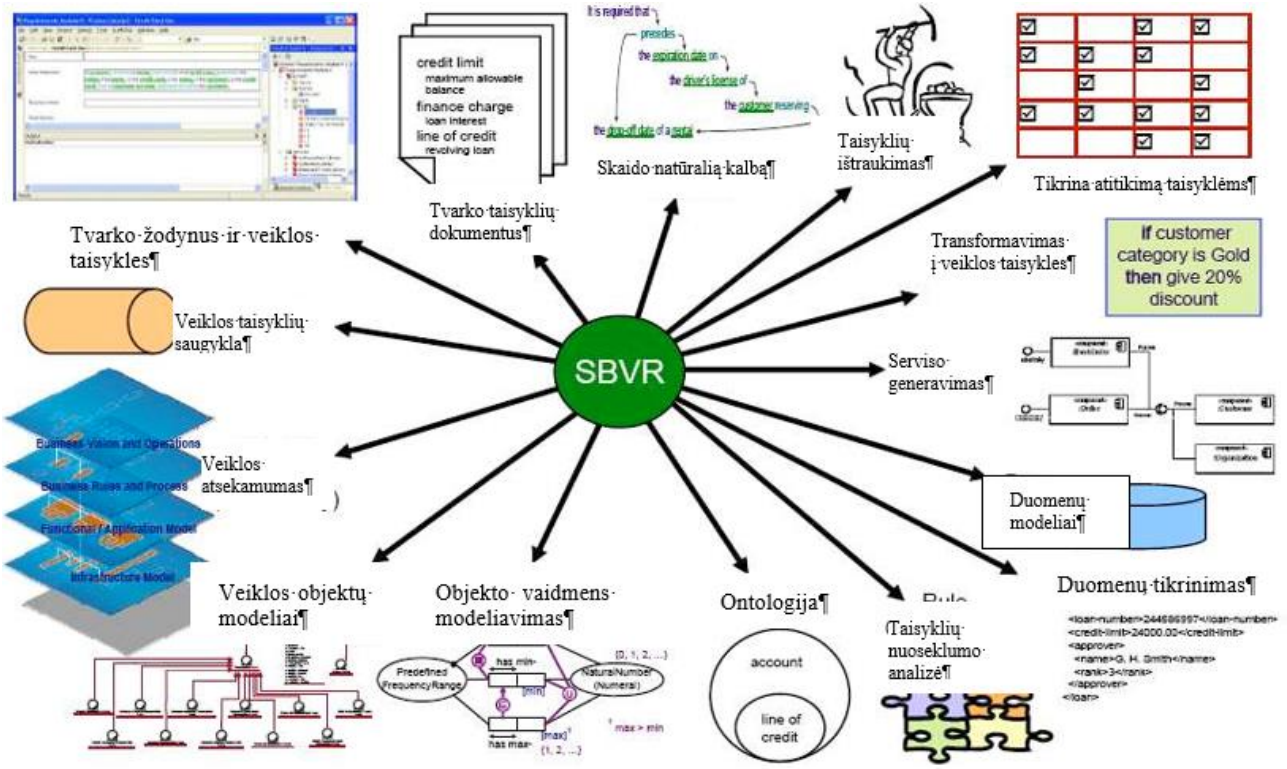
Veiklos taisyklės išraiškos yra analizuojamos ir interpretuojamos atsižvelgiant į veiklos žodynus, kad būtų sukurtos semantinės formuluotės, atitinkančios organizacijos sąvokas. Paprastai veiklos taisyklės išreiškiamos terminais, kuriais operuoja verslo atstovai. SBVR standartas leidžia formuoti žodynų, koncepcijų ir semantinių formuluočių modelius.

Naudojant *SBVR* standartą ir pagalbinius dokumentų kūrimo įrankius galima pasiekti aiškų organizacijos tikslų supratimą ir bendradarbiavimą, kuriant aukštos kokybės dokumentus. Suformuoti dokumentai gali būti pagrįsti *SBVR* koncepcijomis ir koncepciniais modeliais. *SBVR* terminai leidžia aiškiai išreikšti informaciją apie veiklos taisyklę. Tokie dokumentai yra daug greičiau ir lengviau suprantami. Maksimalus standartinio *SBVR* standarto naudojimas užtikrina geriausią bendravimą su suinteresuotosiomis išorės šalimis. Tikslinės žinios taip pat lengviau ir tiksliau užfiksuojamos naudojant *SBVR* koncepcijos formavimo ypatybes.

SBVR yra paremtas ISO terminologijos mokslo standartų, ISO 704: 2000 ir ISO 1087-1: 2000 pagrindu. Šie standartai nurodo, kaip sukonstruoti ir dokumentuoti:

- koncepcijas (žinomas pagal jų apibrėžimus, kurie yra tapatūs skirtingose kalbose), ir
- pavadinimus (terminai ir pavadinimai konkrečioje natūralioje kalboje) kurie naudojami tarp dalykinės srities atstovų [5].

SBVR apibrėžia *XML* keitimo formatą tam, kad *SBVR* ekosistemoje (1.6 pav.) galėtų sąveikauti įvairios priemonės ir paslaugos. *SBVR* modelis gali būti sukurtas vienu įrankiu ir naudojamas kito. *SBVR* ekosistema palaiko įrankių, kurie kurti atskirai, tačiau puikiai gal veikti kartu, sąveiką.



1.6 pav. SBVR ekosistema - įvairių priemonių ir įrankių, naudojamų verslo procesų prasmei perduoti, visuma [5]

Štai keletas įrankių rūšių, kurios jungiasi *SBVR* ekosistemoje:

- žodynų ir taisyklių valdymo priemonės ir saugyklos. Palaikyti integraciją su verslo dokumentais ir atsekamumą su šaltiniais, organizacijos tikslais ir net senais IT elementais;
- kalbų procesoriai, kurie gamina *SBVR* modelius iš žodžių ir verslo pareiškimų. Tikimasi, kad kalbų procesoriai bus naudojami ne tik anglų kalba, bet ir daugeliu kitų kalbų.;
- generatoriai, skirti įvairiems tikslams (pvz., *.Net*, *J2EE*, reliacinės duomenų bazės, taisyklės varikliai, darbo eigos varikliai, apribojimų varikliai);
- transformacijos į kitų rūšių modelius, tokius kaip ontologijos, koncepciniai modeliai, veiklos objektų modeliai (*UML*);
- kokybės analizės priemonės, patikrinančios *SBVR* modelių nuoseklumą, duomenų pasikartojimą, bei užbaigtumą.

Skirtingos *SBVR* ekosistemos dalys patiria skirtingus iššūkius ir reikalauja skirtingų žinių, todėl svarbu, kad šios dalys nebūtų realizuotos kaip viena funkcija ir galėtų būti nepriklausomos viena nuo kitos.

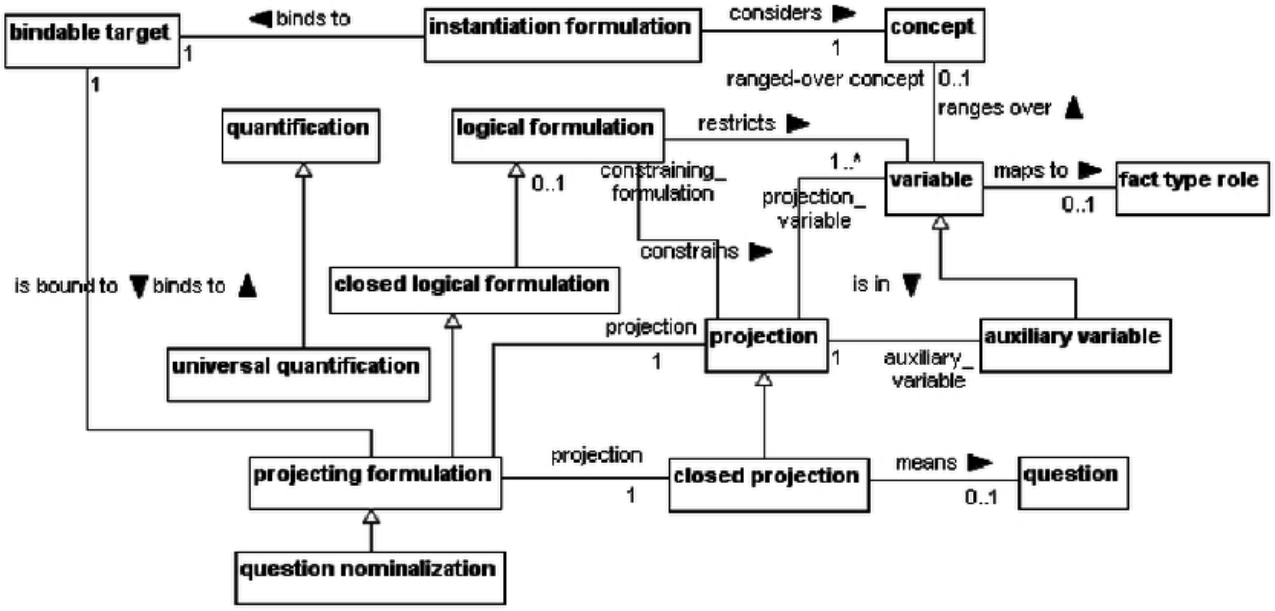
Šiame darbe bus naudojamas *SBVR* standartas, kuris gali būti interpretuojamas programinės įrangos priemonėmis bei leidžia aprašyti taisykles struktūrizuota natūralia kalba. Tai leistų panaikinti būtent šį atotrūkį tarp organizacijos reikalavimų ir formalių sistemos modelių generuojant duomenų modelių kūrimo scenarijų iš *SBVR* standartu užrašytų veiklos žodyno bei taisyklių. Taip dalis veiklos taisyklių būtų transformuotos be papildomo sistemų projektuotojo įsikišimo.

1.5.1. SBVR standarto struktūra

SBVR yra pritaikomas pagal bendrą *OMG* modeliais pagrįstos architektūros (*MDA*) sampratą ir yra orientuotas į veiklos taisykles, jų koncepcijas, žodynus. Svarbiausi *SBVR* aspektai:

1. formali (pirmosios eilės predikatų) logika;
2. semantinių bendruomenių atpažinimas per bendras reikšmes, galimai turinčias keletą vartotojų (kalbų) pogrupių, kurie turi savo kalbas ir specializuotus žodynus;
3. bendrų reikšmių grupė, pateikta sąvokomis, veiksmažodiniais konceptais ir veiklos taisyklėmis;
4. loginė formuluotė, skirta suvokti bendrų reikšmių grupių semantiką. Ji pagrįsta taip pat formalia logika;
5. veiklų atstovavimas loginiam semantiniam formulavimui naudojant pogrupiams priimtinus žodynus.

Pagrindiniai semantiniai *SBVR* aspektai yra žodynai ir terminologijos, daiktavardiniai ir veiksmažodiniai konceptai bei operacinės ir struktūrinės veiklos taisyklės. *SBVR* loginių formuluočių metamodelis pateiktas 1.7 pav.



1.7 pav. *SBVR* metamodelis [6]

1.5.1.1. Žodynai ir terminologijos

SBVR specifikacija apima skirtingų reikšmių tipų ir jų pateikimo aprašymą. Specifikacija apibrėžia struktūrizuotą anglų kalbos žodyną (angl. *SBVR Structured English*), kaip vieną iš galimų užrašymo variantų, kurie gali būti naudojami *SBVR* metamodelio vaizdavime. *SBVR* struktūrizuota anglų kalba apima dalį anglų kalbos žodžių ir gramatinių formuluočių.

Viena iš naujų funkcijų, kurias pristatė *SBVR* yra aiškus žodynų (išorinių) ir vardų srities apibrėžimas. *SBVR* tinkamo konteksto žodynai vadinami kalbų bendruomenėmis arba žodynais [7].

Kiekvienas įrašas skirtas atitinkamai koncepcijai. Kiekviena koncepcija turi aprašymą, pavadinimą ar išraiškos formą. Žemiau pateikiama žodyno struktūra, bei antraščių paaiškinimai.

<Pirminis pavadinimas>

Apibrėžimas (angl. Definition**):**
Išraiška, kuri gali būti logiškai pakeista pirminiu pavadinimu. Apibrėžimas yra visiškai formalus.

Bendra konceptas (angl. General concept):
Nurodo konceptą, kuris apibendrina pirminį konceptą.

Koncepto tipas (angl. Concept Type):
„Koncepto tipo" antraštė naudojama norint įvesti sąvokos įrašo tipą. Paprastai tai netaikoma, jei konceptas neturi jokio kito tipo, išskyrus tai, kas yra akivaizdu iš pirminio pavadinimo. Pavadinimas gali būti numanomas dėl individualios sąvokos. Bet kuri sąvoka netiesiogiai reiškia daiktavardinio koncepto sąvoką.

Būtinybė (angl. Necessity):
Antraštė „Būtinybė" naudojama nurodyti kažkam, kas yra būtinai tiesa. Būtinybė yra orientavimo elementas, išreikštas kaip struktūrinės veiklos taisyklės teiginys. Orientacinis teiginys gali būti išreikštas formaliai arba neformaliai. Toks teiginys gali būti pateikiamas kaip loginė formuluotė.

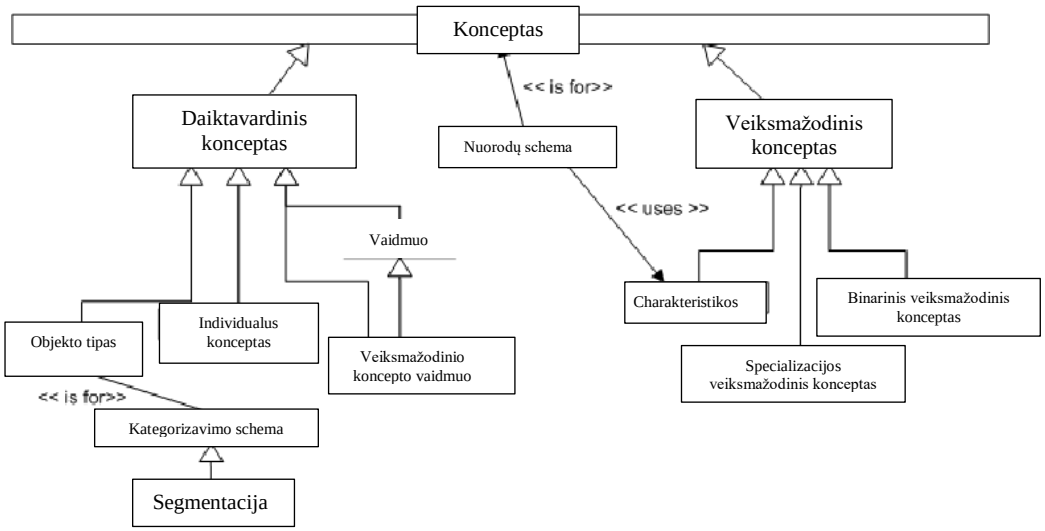
Pavyzdys, kuriame klientas apibūdinamas kaip žmogus (bendrinis konceptas), o patirtis turi tekstinį aprašymą. Patirtis kategorizuoja konceptą ir per būtinumo antraštę siejama su žmogaus konceptu.

```
person
customer
            General_concept: person
experience
            Definition: "practical contact with and observation of facts or events"
            Concept_type: categorization_type
            Necessity: is_for general_concept person
```

Egzistuoja ir kitos antraštės kaip sinoniminė forma, kur galima nurodyti kitą susijusį konceptą.

1.5.1.2. Daiktavardiniai ir veiksmažodiniai konceptai

SBVR faktais orientuotas būdas atėjo iš veiklos taisyklių manifesto, kuris teigia kad taisyklės yra grindžiamos faktais, o faktai grindžiami sąvokomis, išreikštomis terminais. Taigi terminai išreiškia veiklos konceptus, faktai patvirtina arba paneigia šiuos konceptus, o veiklos taisyklės apriboja arba paremia šiuos faktus. *SBVR* standartas realizuoja šį požiūrį, pateikdamas daiktavardinį konceptą kaip terminą, o veiksmažodinį konceptą kaip faktą. Žemiau esančioje diagramoje (1.8 pav.) pavaizduota šių komponentų struktūra.



1.8 pav. Daiktavardiniai ir veiksmažodiniai SBVR konceptai [8]

SBVR daiktavardinio koncepto samprata

SBVR 1.4 specifikacijoje daiktavardinis konceptas suprantamas kaip sąvoka, kurios prasmė yra daiktavardis, kuris diferencijuojamas į:

- objekto tipus, kurie yra daiktavardiniai konceptai, priskiriantys atributus pagal bendrąsias jų savybes;
- individualius konceptus, kurie atitinka tik vieną objektą;
- vaidmenis, kurie yra daiktavardiniai konceptai, darant prielaidą, kad konceptas tam tikroje situacijoje atliko šį vaidmenį.

```
rental_car
branch
```

SBVR veiksmažodinio koncepto samprata

SBVR 1.4 specifikacijoje [9] „veiksmažodis“ yra apibrėžiamas kaip „sąvoka, kuri yra veiksmažodinės frazės reikšmė, kuri apima vieną ar daugiau daiktavardžio sąvokų ir kurių atvejai yra faktai“. *SBVR* išraiška, pateikta struktūrizuota anglų kalba:

```
rental_car is_stored_at branch
```

1.5.1.3. SBVR veiklos taisyklių tipai

Struktūrinės veiklos taisyklės

Struktūrinės veiklos taisyklės yra duomenų objektų apribojimai, su kuriais susijusi organizacijos veikla. Galimi įvairūs duomenų objektų apibūdinimo lygiai. Kalbant žemiausiu lygmeniu, bet koks organizacijos terminų apribojimas yra struktūrinė veiklos taisyklė.

SBVR v 1.4 specifikacijoje struktūrinė taisyklė apibrėžiama kaip būtinumo reikalavimas. Teiginys gali būti formuluojamas keliais būdais: būtinumas, negalimumas, galimybių apribojimai.

Būtinumo teiginys apibūdinamas kaip „struktūrinės taisyklės teiginys, kuris yra išreikštas labiau teigiamai, nei neigiamai, atsižvelgus į būtinumą“.

```
It is necessary that rental has exactly 1 requested car group.
```

Veiklos taisyklės teiginys, kuris neįmanomas:

```
It is impossible that table has exactly 2 legs.
```

Galimybių apribojimas:

```
It is possible that author sells book if book is finished.
```

Operacinės veiklos taisyklės

Elgsenos veiklos taisyklės yra teiginiai apie dinaminius veiklos aspektus. Jos nurodo, kas gali arba bus padaryta duomenų objektams, paveiktiems tam tikrų įvykių.

Apribojimo veiklos taisyklės leidžia apibrėžti sąlygas, kuriomis vadovaujasi organizacijos veikla. Jos išreiškia leidžiamas duomenų objektų būsenas ir užtikrina, kad gali būti atliekami tik tie veiksmai, dėl kurių liks taip pat teisingos duomenų objektų būsenos.

SBVR operacinės veiklos taisyklė apibrėžiama kaip „veiklos taisyklė, kuri yra įsipareigojimo reikalavimas“. Teiginys formuluojamas viena iš šių formų: įsipareigojimas, draudimas, ribotas leidimas. Įsipareigojimo teiginio pavyzdys:

```
A rental must incur a location penalty charge if the drop-off location of the rental is
not the EU-Rent site of the return branch of the rental.
```

Veiklos taisyklės draudimo formulavimas :

```
A rental must not be open if a driver of the rental is a barred driver.
```

Riboto leidimo veiklos taisyklė:

```
It is permitted that a rental is open only if an estimated rental charge is
provisionally charged to the credit card of the renter of the rental.
```

Operacinės veiklos taisyklės yra taikomos, bet ne visada automatizuotos, todėl tokių taisyklių laikymąsi turi užtikrinti žmogus.

1.5.2. Duomenų bazė. Duomenų bazių kalbos

Formaliai „duomenų bazė“ reiškia susijusių duomenų rinkinį ir jo organizavimo būdą [10]. Prieiga prie šių duomenų paprastai teikiama per duomenų bazių valdymo sistemą (angl. *Database Management System*, DBMS), kurią sudaro integruotas kompiuterių programinės įrangos rinkinys, leidžiantis vartotojams bendrauti su viena ar daugiau duomenų bazių ir suteikia prieigą prie visų duomenų bazėje esančių duomenų (nors gali egzistuoti apribota prieiga prie tam tikrų duomenų). DBVS teikia įvairias funkcijas, leidžiančias saugoti ir gauti didelius informacijos kiekius, ir suteikia būdų, kaip tvarkyti šią informaciją.

Duomenų bazės ir DBVS gali būti suskirstytos pagal duomenų bazės modelius, kuriuos jie palaiko (pvz. reliacinį ar *XML*), kompiuterio tipus, kuriuose jie veikia (serverių grupė ar mobilus telefonas), užklausos kalbas, naudojamas prieigai prie duomenų bazės (pvz. *SQL* arba *XQuery*), ir pagal vidinę inžineriją, kuri daro įtaką našumui, atsparumui ir saugumui.

Duomenų bazės kalbos yra specialios paskirties kalbos, kurios leidžia atlikti vieną ar daugiau šių užduočių, kurios kartais išsiskiria kaip antrinės kalbos:

- duomenų valdymo kalba (DCL) - kontroliuoja prieigą prie duomenų;
- duomenų apibrėžimo kalba (*DDL*) - apibrėžia duomenų tipus, pvz., jų kūrimą, keitimą ar atšaukimą ir jų tarpusavio ryšius;
- duomenų manipuliavimo kalba (*DML*) - atlieka tokias užduotis kaip duomenų įvedimas, atnaujinimas ar ištrynimas;
- duomenų užklausos kalba (*DQL*) - leidžia ieškoti informacijos ir apskaičiuoti gautą informaciją.

Duomenų apibrėžimo kalba (DDL)

Daugelyje duomenų aprašymo kalbų stulpelių ir duomenų tipams apibrėžti naudojama deklaratyvi sintaksė [11]. Tačiau struktūrizuotos užklausos kalba (pvz., *SQL*) naudoja imperatyvių veiksmažodžių rinkinį, kurio paskirtis yra keisti duomenų bazės schemą pridedant, keičiant ar ištrinant lentelių ar kitų elementų apibrėžtis. *DDL* komandos:

- CREATE– naudojama kuriant naują duomenų bazę, lentelę, indeksą arba procedūrą.

Tipinis užklausos panaudojimas yra kuriant duomenų bazės lentelę:

```
CREATE TABLE [table name] ( [column definitions] ) [table parameters]
```

- DROP – naudojama šalinant esamą duomenų bazę, lentelę, indeksą ar vaizdą

DROP komanda *SQL* pašalina komponentą iš reliacinės duomenų bazės valdymo sistemos. Objektų tipai, kurie gali būti pašalinti, priklauso nuo to, kokia DBVS yra naudojama, bet labiausiai palaiko lentelių, naudotojų ir duomenų bazių šalinimą. Tipiškas naudojimas:

```
DROP objecttype objectname.
```

- ALTER – naudojama egzistuojančio duomenų bazės objekto modifikacijai.

ALTER komanda *SQL* keičia objekto savybes reliacinės duomenų bazės valdymo sistemoje. Objektų tipai, kuriuos galima pakeisti, priklauso nuo DBVS naudojimo. Tipiškas naudojimas yra:

```
ALTER objecttype objectname parameters.
```

- TRUNCATE – naudojamas duomenų ištrynimui iš lentelės. Ši užklausa yra greitesnė nei DELETE.

```
TRUNCATE TABLE table_name;
```

Duomenų manipuliavimo kalba (DML)

SQL duomenų apdorojimo kalba apima *SQL* duomenų keitimo komandas, kurios keičia saugomus duomenis, bet ne schemos ar duomenų bazės objektus [12]. Manoma, kad nuolatinių duomenų bazių objektų, pvz., lentelių ar procedūrų, tvarkymas per *SQL* užklausas, yra atskira duomenų apibrėžimo kalbos (*DDL*) dalis.

Duomenų manipuliavimo kalbos turi funkcijas, kurias nurodo pradinis žodis komandoje, kuris beveik visada yra veiksmažodis. *SQL* atveju šie veiksmažodžiai yra:

- SELECT ... FROM ... WHERE ...
- SELECT ... INTO ...
- INSERT INTO ... VALUES ...
- UPDATE ... SET ... WHERE ...
- DELETE FROM ... WHERE ...

Šiame darbe naudojamos duomenų manipuliavimo ir duomenų apibrėžimo kalbos.

Duomenų bazės kalbos

Duomenų bazės kalbos būdingos konkrečiam duomenų modeliui. Žinomesni pavyzdžiai:

- *SQL* - sujungia duomenų apibrėžimo, duomenų manipuliavimo ir užklausos vaidmenis vienoje kalboje. Tai buvo viena pirmųjų komercinių kalbų reliaciniam modeliui, nors kai kuriais

atžvilgiais ji nukrypsta nuo reliacinio modelio, kaip aprašyta „Codd“. 1986 m. *SQL* tapo Amerikos nacionalinio standartų instituto (angl. *American National Standards Institute,* ANSI) standartu, o Tarptautinės standartizacijos organizacijos (angl. *International Organization for Standardization,* ISO) standartu - 1987 metais.

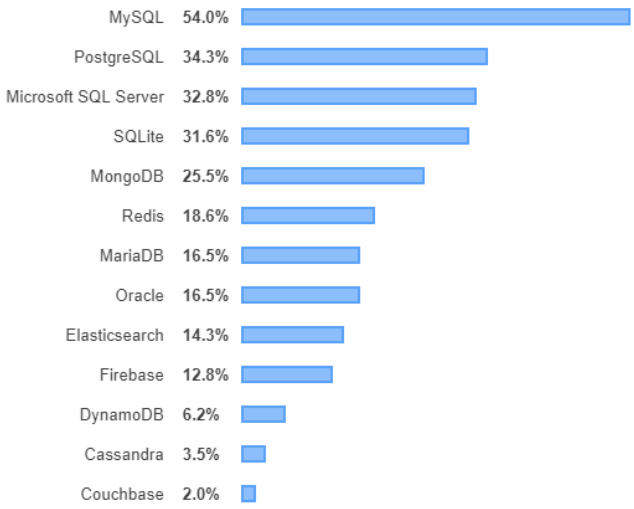
- *OQL* - objekto modelio kalbos standartas. Jis turėjo įtakos kai kurių naujesnių užklausų kalbų, tokių kaip *JDOQL* ir *EJB QL*, dizainui.
- *XQuery* - standartinė *XML* užklausų kalba, įdiegta *XML* duomenų bazių sistemose, pvz., *MarkLogic* ir *eXist*, reliacinėmis duomenų bazėmis su *XML* galimybėmis, pvz., *Oracle* ir *DB2*, taip pat *XML* atminties procesoriais, tokiais kaip *Saxon*.
- *SQL* / *XML* - sujungia *XQuery* su *SQL*.

Į duomenų bazės kalbą taip pat gali būti įtrauktos tokios funkcijos:

- konkrečios DBVS konfigūracijos ir saugojimo valdymas;
- skaičiavimai, kuriais siekiama pakeisti užklausos rezultatus, pvz.: skaičiavimas, sumavimas, vidurkis, rūšiavimas, grupavimas ir kryžminės nuorodos;
- apribojimų vykdymas.

SQL kalba

Pastaruoju metu manoma, kad duomenys yra vertingiausias turtas ir jų tinkamas surinkimas yra svarbus uždavinys sprendimų priėmimo procese. Nepaisant pastarųjų metų *NoSQL* staigaus išpopuliarėjimo, *SQL* išlieka kaip universali kalba duomenų analizės atlikimui. Ją palaiko dažniausiai naudojami duomenų bazių valdymo įrankiai, pvz., *MySQL*, *PostgreSQL*, *Microsoft SQL Server* ir *SQLite*:



1.9 pav. Populiariausios duomenų bazės , pagal StackOverflow apklausos rezultatus (2019m.) [13]

Tai nenuostabu, nes *SQL* kalba yra ypač veiksminga manipuliuojant duomenimis. Be to, duomenys, saugomi reliacinėje duomenų bazėje, yra dinamiški, o tai reiškia, kad gali būti gauti, modifikuoti ir manipuliuoti lengvai su kai kuriomis pagrindinėmis *SQL* užklausomis.

SQL [14] kalba yra suskirstyta į keletą kalbų elementų, įskaitant:

- sąlygos, kurios yra komandų ir užklausų sudedamosios dalys;
- išraiškos, kurios gali sukurti skaliarines vertes arba lenteles, sudarytas iš stulpelių ir duomenų eilučių;
- predikatų, kurie nurodo sąlygas, kurios gali būti vertinamos pagal *SQL* trijų verčių logiką (angl. *Three-value logic*, 3VL) (tikrosios / klaidingos / nežinomos) arba Būlio algebros vertes ir naudojami apriboti komandų ir užklausų reikšmes ar poveikį;
- užklausos, kurios gauna duomenis pagal konkrečius kriterijus. Tai yra svarbus *SQL* elementas.
- komandos, kurios gali turėti nuolatinį poveikį schemoms ir duomenims, arba gali kontroliuoti darbų eiliškumą, jungtis, sesijas ar diagnostiką.
    - *SQL* komandose taip pat yra kabliataškio (";") simbolis. Nors tai nereikalinga kiekvienoje platformoje, jis apibrėžiamas kaip standartinė *SQL* gramatikos dalis;
- *SQL* komandose ir užklausose paprastai ignoruojami tarpai, todėl lengviau formatuoti *SQL* kodą.

1.5.3. Duomenų bazių apribojimai

Duomenų bazių valdymo sistemos leidžia realizuoti įvairių tipų apribojimus, kurie padeda išlaikyti duomenų vientisumą ir neprieštaringumą. Apribojimai yra taikomi duomenims, kuriuos galima įterpti, atnaujinti ištrinti iš reliacinės duomenų bazės lentelės. Veiklos taisyklę galima išreikšti tam tikru apribojimu duomenų bazės objektui, pavyzdžiui lauko apribojimui, arba ryšio ypatumui. Egzistuoja keletas duomenų bazės apribojimų tipų [15]:

- NOT NULL – privalomos reikšmės apribojimas

Duomenų bazėje, kol nėra tam užkertamas kelias, galima įvesti stulpeliams NULL reikšmes. Apribojimas NOT NULL uždraudžia stulpeliui saugoti NULL reikšmes. Tai reiškia, kad turi būti išsaugota reikšmė, ir naujas įrašas nebus išsaugotas arba atnaujintas tol, kol visi stulpeliai turintys tokius apribojimus neturės tuščios reikšmės. Žemiau pateiktas *DDL* scenarijus, kuriame identifikatorius, vardas ir pavardė privalo turėti reikšmes:

```
It is necessary that person has name and person has surname.

CREATE TABLE PERSON (
    ID int NOT NULL,
    SURNAME varchar(255) NOT NULL,
    NAME varchar(255) NOT NULL,
    AGE int
);
```

- UNIQUE – unikalios reikšmės apribojimas

Apribojimas UNIQUE užtikrina, kad visos reikšmės, esančios tame stulpelyje bus unikalios. Tiek UNIQUE tie PRIMARY KEY užtikrina unikalumą stulpeliui arba stulpelių junginiams, nes

PRIMARY KEY automatiškai paveldi UNIQUE apribojimą. Žemiau pateikti variantai, kaip galima būtų realizuoti stulpelių unikalumus:

```
-- Unikalus ID laukas
CREATE TABLE PERSON (
    ID int NOT NULL UNIQUE,
    SURNAME varchar(255) NOT NULL,
    NAME varchar(255),
    AGE int
);

-- Unikalus vardas ir pavardė
CREATE TABLE PERSON (
    ID int PRIMARY KEY,
    SURNAME varchar(255) NOT NULL,
    NAME varchar(255),
    AGE int,
    ADD CONSTRAINT Unique_Person UNIQUE (NAME, SURNAME)
);
```

- DEFAULT – pradinės reikšmės apribojimas

Apribojimas DEFAULT skirtas nurodyti pradinę reikšmę stulpeliui. Pradinė reikšmė stulpeliui bus suteikta tik tada, kai nebus nieko kito nurodyta būtent tam stulpeliui. Žemiau pateikiamas *DDL* scenarijaus pavyzdys, kai nustatoma miesto pradinė reikšmė:

```
CREATE TABLE PERSON (
    ID int NOT NULL,
    SURNAME varchar(255) NOT NULL,
    NAME varchar(255),
    AGE int,
    CITY varchar(255) DEFAULT 'Kaunas'
);
```

- CHECK – reikšmės patikrinimo apribojimas

Apribojimas CHECK skirtas riboti reikšmių aibę, kurią galima įrašyti į stulpelį. Žemiau pateiktas *DDL* scenarijaus kūrimo variantas, kai įrašas bus išsaugotas arba atnaujintas tik tada, kai žmogui daugiau nei 20 metų:

```
It is necessary that person has at least 20 years.

CREATE TABLE PERSON (
    ID int NOT NULL,
    SURNAME varchar(255) NOT NULL,
    NAME varchar(255),
    AGE int,
```

```
    CHECK (AGE>20)
);
```

- Raktų apribojimai – pirminis ir išorinis raktai

Pirminio rakto apribojimas nustato unikalumą kiekvienam duomenų bazės įrašui. Tai reiškia, kad šiam stulpeliui turi būti unikalios reikšmės (UNIQUE), bei reikšmės negali būti null (NOT NULL). Lentelėje gali būti tik vienas PRIMARY KEY apribojimas susidedantis iš vieno arba kelių stulpelių.

```
CREATE TABLE PERSON (
    ID int NOT NULL PRIMARY KEY,
    SURNAME varchar(255) NOT NULL,
    NAME varchar(255),
    AGE int
);
```

Duomenų bazių reliaciniame modelyje pirminis raktas yra konkretus minimalaus stulpelių rinkinio pasirinkimas, kuris išskirtinai nurodo eilutę duomenų bazės lentelėje. Paprastai tariant, pirminis raktas tai atributas, pagal kurį yra atpažįstamas įrašas. Dažniausiai tam pasirenkamas unikalus stulpelis „ID“.

Išorinis raktas apibrėžiamas sekančioje duomenų bazės lentelėje, tačiau jis susijęs su pirmos lentelės unikaliu pirminiu raktu. Kadangi išorinio rakto tikslas yra nustatyti konkrečią nuorodą į lentelės eilutę, paprastai reikalaujama, kad išorinis raktas būtų lygus pirminiam raktui tam tikroje pradinės lentelės eilutėje, arba kitaip neturėtų stulpelis reikšmės (NULL). Ši taisyklė vadinama referencinio vientisumo apribojimu (angl. *Referential Integrity Constraint*) tarp dviejų lentelių. Kadangi šių apribojimų pažeidimai gali būti daugelio duomenų bazių problemų šaltinis, dauguma duomenų bazių valdymo sistemų suteikia mechanizmus, užtikrinančius, kad kiekvienas išorinis raktas atitiktų nuorodą į kitos lentelės eilutę .

```
CREATE TABLE ORDER (
    ORDER_ID int NOT NULL PRIMARY KEY,
    ORDER_NO int NOT NULL,
    PERSON_ID int FOREIGN KEY REFERENCES PERSON(ID)
);
```

1.5.4. MDE ir modelių transformacija

Modeliais grindžiama inžinerija (angl. *Model Driven Engineering,* MDE) tai programinės įrangos kūrimo metodika, orientuota į duomenų modelių kūrimą ir naudojimą, kurie yra visų su konkrečia problema susijusių temų koncepciniai modeliai. Modeliais grindžiamoje inžinerijoje modeliai laikomi vienijančia koncepcija. Bendrai modelis yra sistemos vaizdas. Jis apibūdina kai kurias sistemos ypatybes. Modeliavimo kalbos abstrakčioji sintaksė, išreikšta modeliu, vadinama metamodeliu. Metamodeliai, savo ruožtu, išreikšti modeliavimo kalba, vadinami metametamodeliu.

MDE modeliavimo paradigma laikoma veiksminga, jei jos modeliai yra prasmingi naudotojui, kuris yra susipažinęs su dalykine sritimi, ir jie jie gali būti sistemos kūrimo pagrindu. Modeliai kuriami bendraujant tarpusavyje produktų vadybininkams, dizaineriams, programuotojams ir programų naudotojams. Sukurti modeliai leidžia kurti programinę įrangą ir sistemas.

Modelių transformacija grindžiamoje inžinerijoje modelių kūrimas ir modifikavimas yra automatizuotas. Modelio transformacijos tikslas - sutaupyti pastangų ir sumažinti klaidas, kiek tai yra įmanoma, automatizuojant modelių kūrimą ir modifikavimą.

Modelio transformacija gali būti parašyta bet kuria bendrine programavimo kalba, bet taip pat yra specializuotų modelių transformavimo kalbų. Abipusės transformacijos geriausiai parašomos ta kalba, kuri užtikrina, kad kryptys būtų tinkamai susijusios. Standartizuotos OMG modelių transformacijos kalbos bendrai vadinamos QVT.

1.5.5. Veiklos žodynų transformavimo kalbų analizė

Šiuo metu modeliai apima vis daugiau inžinerinių procesų (pvz., programinės įrangos inžinerija). Tačiau daugeliu atvejų jie vis dar daugiau aprašomi dokumentacijose nei integruojami tiesiai į inžinerijos procesus.

Tuo tarpu modeliais grindžiamoje inžinerijoje modeliai laikomi pagrindiniais objektais ir yra naudojami daugelyje sistemos kūrimo procesų. Modeliais grindžiamas požiūris leidžia modelių projektuotojams ir programuotojams pateikti modelių manipuliavimo operacijų rinkinį.

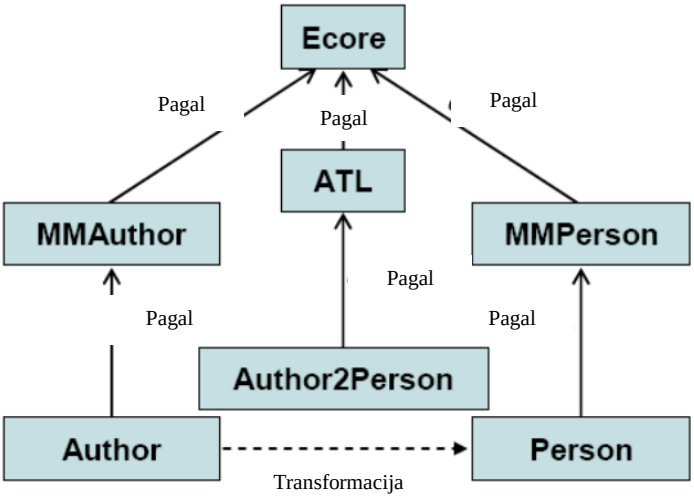
Modelio transformacija, kuri iš esmės yra programa, veikianti modeliais, gali būti parašyta bendrojo programavimo kalba, pvz., *Java*. Tačiau specialios paskirties modelių transformavimo kalbos gali suteikti pranašumų, pvz., patogesnę sintaksę, kuri palengvina kreipimąsi į modelio elementus.

Šiuo metu akademinėje bendruomenėje plėtojamos dauguma modelių transformavimo kalbų. OMG standartizavo modelių transformavimo kalbų šeimą, vadinamą QVT, tačiau šį sritis vis dar yra nepakankamai ištobulinta.

1.5.5.1. ATL

ATL (angl. *ATLAS Transformation Language*) yra modelio transformacijos kalba ir įrankių rinkinys, sukurtas ir prižiūrimas OBEO ir Atlanmod. Ją inicijavo „Atlanmod“ komanda (anksčiau vadinama „ATLAS Group“)[16].

Šiuo metu ATL transformacijos variklis palaiko dvi iš šių egzistuojančių technologijų: OMG Meta-objektų paslaugas ir Eclipse modeliavimo karkaso Ecore metametomodelį. Tai reiškia, kad ATL gali apdoroti metamodelius, kurie nurodyti pagal MOF arba "Ecore" semantiką [17].

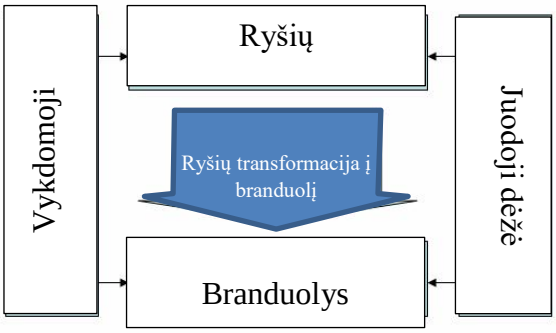


1.10 pav. ATL ir ECORE [18]

1.10 pav. parodyta, kaip transformuojamas modelis Author į modelį Person, kurie turi savo metamodelius MMAuthor ir MMPerson. ATL yra modelių transformavimo kalba, leidžianti nurodyti, kaip vieną (ar daugiau) modelių galima transformuoti iš kito ar kitų modelių rinkinio. Kitaip tariant, ATL pateikia sąvokų rinkinį, kuris leidžia apibūdinti modelio transformacijas.

1.5.5.2. QVT

QVT pavadinime užšifruota, kad šis OMG standartas padengia transformacijas, rodinius ir užklausas kartu. Modelio užklausos bei modelio rodiniai gali būti vadinami modelio transformacija jei šiai transformacijai naudosime programą, kuri naudoja kitus modelius [19].



1.11 pav. QVT struktūrinė schema

QVT standartas apibrėžia tris transformacijos elementus, veikiančius MOF 2.0 (angl Meta-Object Facility) meta modeliu:

- QVT vykdomoji (angl. QVT Operational) – vienakryptės transformacijos tarp metamodelių kalba;

- QVT ryšių (angl. QVT Relations) – vienakryptės ir dvikryptės transformacijos tarp modelių, kurios gali tikrinti, ar modeliai atitinka vienas kitą;
- QVT branduolio (angl. QVT Core) – transformuojamo meta modelio struktūrinės savybės ir sąryšiai tarp jų. Kiekvienam transformuojamam modeliui turi būti aprašytas branduolys.

Kiti elementai, esantys tik pagalbininkai transformacijos procese yra vaizduojami QVT juodojoje dėžėje (angl. *QVT Black Box*).

1.5.5.3. ATL ir QVT palyginimas

Tarp ATL ir QVT yra daug bendro: abi naudoja OCL, abi nusakomos MOF metamodeliu, abi palaiko *XMI* kaip pirminio modelio ir galutinio rezultato formatus. Iš architektūrinės pusės QVT ir ATL lyginamos [20] pagal šias kategorijas: abstraktumas, paradigma, kryptingumas, kardinalumas, ir t.t.. Rezultatai rodo, jog ATL ir QVT yra suderinamos ir galimos tarpusavio transformacijos. Tad pasirinkus viena iš šių transformavimo sprendimų būtų prieinama įrankių bazė, bibliotekos, sprendimų analizės, standartų palaikymas, ir t.t. Dėl didesnio populiarumo panašiose transformacijose buvo pasirinkta ATL kalba.

1.6. Tyrimo objekto naudotojų analizė

Veiklų žodynų ir taisyklių, užrašytų struktūrizuota natūralia kalba, automatizuotas transformavimo į duomenų modelius metodas gali būti naudingas IS analitikams, kurie geba veiklos taisykles užrašyti ir analizuoti. Sutaupomas programuotojų darbas. Taip pat esat poreikiui pakeisti taisyklę nebūtinas programuotojo įsikišimas. Užsakovas yra suinteresuotas, kad kuo efektyviau būtų išnaudoti kaštai, skirti informacinės sistemos kūrimui.

1.1 lentelė. Vartotojo tipo ir savybių lentelė

Vartotojo tipas	**Vartotojo savybės**
Informacinės sistemos analitikas	Bendrauja su dalykinės srities ekspertu ir aprašo veiklos taisykles. Analitikas nėra sistemos projektuotojas, tačiau suvokia duomenų modelius.
Užsakovas	Nebūtinai išmanantis informacinių sistemų technologijas, bet puikiai suprantantis veiklos taisykles.

1.2 lentelė. Problemos, su kuriomis susiduria vartotojai

Problema	**Kaip viskas vyksta dabar**
Programuotojas analizuoja analizės dokumentus	Išanalizavus analizės dokumentus yra kuriami sprendimai kaip įgyvendinti duomenų bazių apribojimus.
Atliktas DB darbas nėra dokumentuojamas	Programuotojas suprogramuoja duomenų bazės apribojimus. Tik gebantis suprasti kodą, gali sužinoti kaip realizuota IS vieta.

Pakeitimai užtrunka	Norint įvykdyti užsakovo norimus pakeitimus reikia pirma atlikti analizę kokia yra esama realizacija, o po to programuotojas atlieka pakeitimus.

1.7. **Esamų problemos sprendimo metodų analizė**

Nagrinėjami metodai, naudojantys veiklos taisyklių ir žodyno užrašymo *SBVR* standartą, bei veiklos taisykles tiesiogiai siejantys su duomenų baze.

Automatinis duomenų tikrinimas naudojant SBVR

Šiame dokumente autoriai iškėlė problemą: keičiantis aplikacijoms arba veiklos taisyklėms, informacija turėtų likti nuosekli [21]. Sukurtas sistemos karkasas, kuris tikrina duomenų nuoseklumą. Veiklos taisyklės yra panaudojamos *SQL* užklausų generavimui. Šis automatinis procesas įmanomas tik jei duomenų bazėje naudojami tikslūs pavadinimai ir organizacija naudoja standartizuotą žodyną. Sistemos karkasas bandytas universiteto akademinėje sistemoje, kur nėra jokios sistemos dokumentacijos, ir kūrimo procesas vyksta nuolat. Tikrinimas koncentruojasi į jau esamų įrašų tinkamumą.

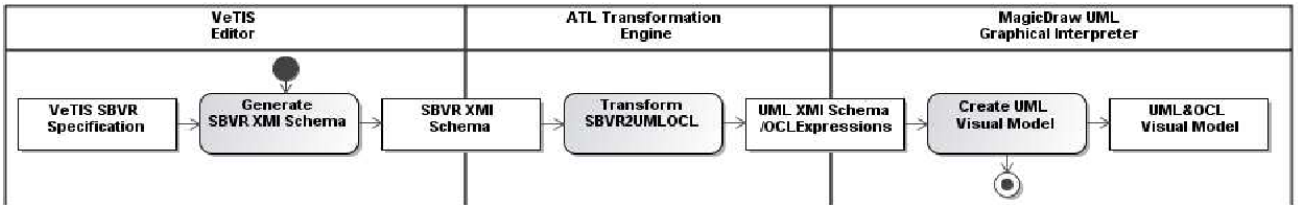
SBVR2SQL kompiliatorius

Šiame dokumente įgyvendinamas *SQL* duomenų modelių ir sakinių sudarymas iš *SBVR* struktūrine anglų kalba užrašytų sakinių [22]. Šis įrankis kurtas internetinėms naršyklėms, kurios palaiko *Web SQL* duomenų bazę. Koncentruojamas dėmesys kaip paversti sakinius užrašytus anglų kalba (*SBVR-SE*) į *SBVR* loginį formulavimą (*SBVR-LF*), po to į *SQL* tiek *DDL* kūrimo scenarijų, tiek į *DML* užklausas. Sukurtų *SQL* užklausų pagrindinis tikslas yra patikrinti ar taisyklė atitinka turimus duomenis duomenų bazėje. Grąžinamas atsakymas *true* arba *false*.

VeTIS + MagicDraw

VeTIS - tai *SBVR* standartu užrašytų veiklos žodyno ir taisyklių redagavimo programa integruota į *MagicDraw* įrankį, galinti generuoti *UML* klasių diagramas su *OCL* apribojimais iš *SBVR* žodynų ir taisyklių [23]. *VeTIS* redaktorius buvo kurtas su Eclipse 3.4.1 platforma, remiantis *SBVR* 1.0 metamodeliu.

Viena pagrindinių *VeTIS* įrankio savybių yra *SBVR* specifikacijos transformacija (t.y. veiklos žodynų ir veiklos taisyklių) *SBVR* struktūrizuota anglų kalba į *SBVR* 1.0 *XMI* formatą, o vėliau - į UML klasės modelį su OCL apribojimais. Tuo tikslu naudojama ATL transformacijos kalba. Paskutiniame etape UML modelis vizualizuojamas *MagicDraw* įrankyje, naudojant *OpenAPI*. Transformavimo procesas pateiktas 1.2 paveiksle.



1.12 pav. Iš SBVR į UML it OCL transformacijos žingsniai [23]

Duomenų bazės schema projektuojama dalykinės srities esybių-ryšių modelio pagrindu. *MagicDraw* įrankis turi priemones automatiniam duomenų bazės schemai generuoti iš pasirinkto esybių ryšių modelio. *MagicDraw* aplinkoje galima nustatyti transformavimo parametrus:

- pirminį raktą;
- išorinį raktą;
- lentelės pavadinimą;
- duomenų įvedimo apribojimų transformavimo strategiją.

Transformavimo rezultatas (DB schema) išsaugomas nurodytoje projekto vietoje, DB schemos pakete. *DDL* generavimas, naudojant modeliavimo įrankį (MagicDraw) susideda iš trijų komponentų [24]:

- variklis, kuris vykdo *Oracle DDL* generavimą. Jis renka konteksto kintamuosius ir juos sujungia į šabloną;
- šablonas, kurį apibrėžia vartotojas. Šiame dokumentas pateikiama greičio variklio šablono kalbos (VTL) sintaksė. VTL sintaksė yra manipuliuojama konteksto kintamaisiais tekstinio scenarijaus (angl. *script*) generavimui;
- konteksto kintamieji – modeliai, gauti per OpenAPI. Programų inžinerijos nustatytos informacijos ir vartotojo apibrėžti kintamieji.

Pagrindiniai šio metodo trūkumai:

- pasikeitus duomenų modeliui ar veiklos taisyklėms, duomenų bazę reikia generuoti iš naujo, kas reiškia lentelių perkūrimą ir esamų duomenų šalinimą;
- ryšiams tarp esybių palaikomi standartiniai kardinalumai.

1.3 lentelė. Esamų sprendimų palyginimas

Palyginimo kriterijus		Automatinis duomenų tikrinimas naudojant SBVR	SBVR2SQL kompiliatorius	VeTIS + MagicDraw
Kuriamos SQL užklausos		+	+	+
Skirtas jau turimiems duomenims patikrinti		+	+	-
	Nustatomi kardinalumai	-	-	+/-
	Naudojamas EXISTS	+/-	+	-
	Naudojami HAVING,WHERE,GROUP	+	+	-
	Generuojami INSERT	-	-	+
	Generuojami UPDATE	-	-	-

Skirtas užtikrinti, kad ateityje nebūtų įrašomi klaidingi duomenys		-	-	+
	Naudojami TRIGGER, UNIQUE, DEFAULT	-	-	-
	Generuojami PK, FK	-	-	+
	Naudojami CHECK	-	-	+/-
	Pakeitimai daromi nepakeičiant esamų duomenų			-

1.8. **Darbo tikslas, uždaviniai, planas ir siekiami privalumai**

Tikslas

Išplėsti duomenų modelių kūrimo galimybes, vykdant transformacijas iš natūralia kalba aprašytų veiklos žodynų ir taisyklių.

Uždaviniai

1. Išanalizuoti:
   - informacinių sistemų kūrimo procesą;
   - veiklos žodynų ir taisyklių sudarymo principus, *SBVR* standartą;
   - su duomenų bazėmis susijusias kalbas ir technologijas (t.y., *DDL*, *SQL* ir kt.);
   - esamus veiklos žodynų ir taisyklių transformavimo į duomenų modelius sprendimus;
   - taisyklėmis išreikštų apribojimų realizavimą duomenų bazės priemonėmis;
   - modelių transformavimo technologijas;
2. sukurti metodą, leidžiantį aprašyti veiklos žodyną ir taisykles bei transformuoti į duomenų modelius;
3. realizuoti sprendimo prototipą;
4. atlikti eksperimentą, įvertinti tyrimo rezultatus.

Planas

Užduotys	**Termino pabaiga**
SBVR standarto analizė	2017 11 15
Duomenų bazių sintaksės analizė	2017 11 15
Literatūros analizė	2017 11 17
Išsianalizuoti sukurtas programas	2017 12 10
Naudojamų įrankių analizė	2017 12 20
Išsamesnė literatūros analizė	2017 12 20
Reikalavimų formulavimas	2018 04 15
Realizacija	2019 01 28
Eksperimentai	2019 03 15
Išvados, rezultatai	2019 04 01

1.9. **Siekiamo sprendimo apibrėžimas**

Šiame darbe siekiama sukurti veiklos taisyklių ir žodyno transformavimo į duomenų modelius metodą. Jis skirtas veiklos žodyno ir taisyklių transformavimui į duomenų modelius, t.y. esybių atpažinimui,

atributų bei sąryšių suradimui. Sukurtu automatiniu transformavimo algoritmu siekiama automatiškai atpažinti taisykles ir pateikti jų kūrimą *SQL* kalba užrašytomis užklausomis. Sprendimo pateikimas *SQL* scenarijuose leis programuotojui pasitikrinti transformavimo rezultatus, modifikuoti pagal esančias išimtis.

1.10. **Analizės išvados**

1. Atlikus *SBVR* standarto analizę paaiškėjo jog šis standartas leidžia aprašyti žodyną ir taisykles ir transformuoti šias specifikacijas į programinės įrangos modelius.
2. Veiklos taisyklės, kurios gali būti formuojamos pagal *SBVR* standartą, yra skirstomos į struktūrines ir operacines. Analizės metu buvo nagrinėjama duomenų bazės kalbų sintaksė ir paaiškėjo, jog norint užkirsti kelią klaidingų duomenų įvedimui, galime struktūrines veiklos taisykles realizuoti *DDL* apribojimais ir kitomis DB priemonėmis.
3. Atlikus įrankių analizę matoma, jog nemažai nagrinėta kaip patikrinti duomenų nuoseklumą, taisyklių laikymąsi ir pan., tačiau šie įrankiai skirti patikrinti, ar duomenų bazėje saugomi duomenys tenkina taisykles, bet netikrina įvedamų duomenų teisingumo. Didžiausias indėlis į apsaugą nuo būsimų klaidingų duomenų yra atliktas *VeTIS* sistemoje, todėl iš šios sistemos galima perimti naudingų savybių.

2. SBVR TRANSFORMAVIMO Į DUOMENŲ MODELIUS SPRENDIMO REIKALAVIMŲ SPECIFIKACIJA IR PROJEKTAS, FORMALUS APRAŠAS

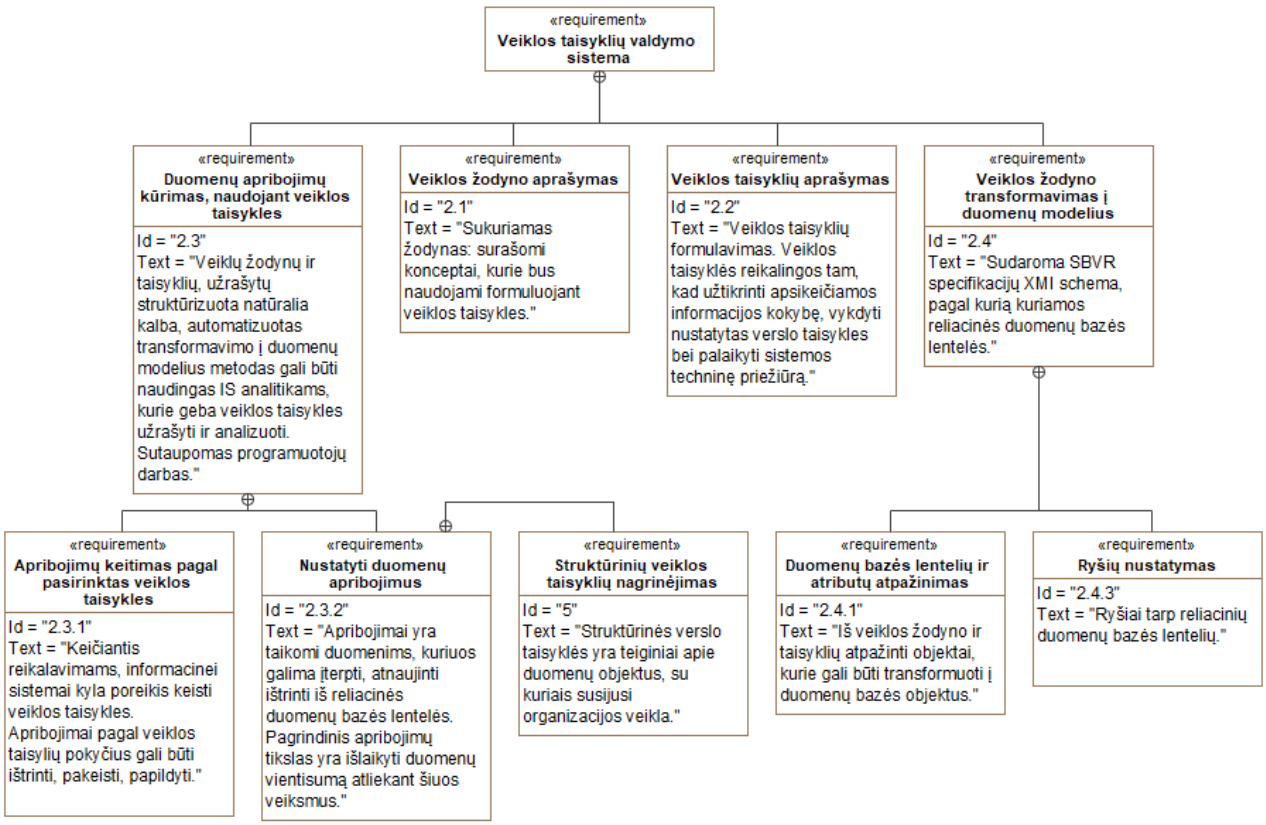
2.1. Reikalavimų specifikacija

Kuriant informacinės sistemos veiklos taisykles pirmiausia reikia apsirašyti veiklos žodyną, tik po to atskirai aprašomos veiklos taisyklės. Šiam tikslui planuojami du atskiri tekstiniai failai. Pagal tai, ar naudotojas yra atsidaręs veiklos žodyną ar veiklos taisyklių kūrimo langą priklauso, kokį funkcionalumą sistema vykdys.

2.1.1. Sistemos reikalavimų modelis

SBVR veiklos žodyno ir taisyklių transformavimo metodo realizavimui sukurtos sistemos reikalavimai skirstomi į šias dalis:

- veiklos žodyno aprašymas – naudotojas turi galimybę sukurti veiklos žodyną. Žodyne surašomi konceptai, kurie bus naudojami formuluojant veiklos taisykles;
- veiklos taisyklių aprašymas – sistemoje privalo būti galimybė naudotojui aprašyti veiklos taisykles. Veiklos taisyklės reikalingos tam, kad užtikrinti apsikeičiamos informacijos kokybę, vykdyti nustatytas veiklos taisykles bei palaikyti sistemos techninę priežiūrą;
- veiklos žodyno transformavimas į duomenų modelius - sudaroma *SBVR* specifikacijų *XMI* schema, pagal kurią kuriamos reliacinės duomenų bazės lentelės;
- duomenų apribojimų kūrimas, naudojant veiklos taisykles - veiklų žodynų ir taisyklių, užrašytų struktūrizuota natūralia kalba, automatizuotas transformavimo į duomenų modelius metodas gali būti naudingas IS analitikams, kurie geba veiklos taisykles užrašyti ir analizuoti. Taip yra taupomas programuotojų darbas.



2.1 pav. Veiklos žodyno ir taisyklių transformavimo į duomenų modelius sistemos reikalavimų modelis

Reikalavimas „Veiklos žodyno transformavimas į duomenų modelius" skaidomas į smulkesnius reikalavimus:

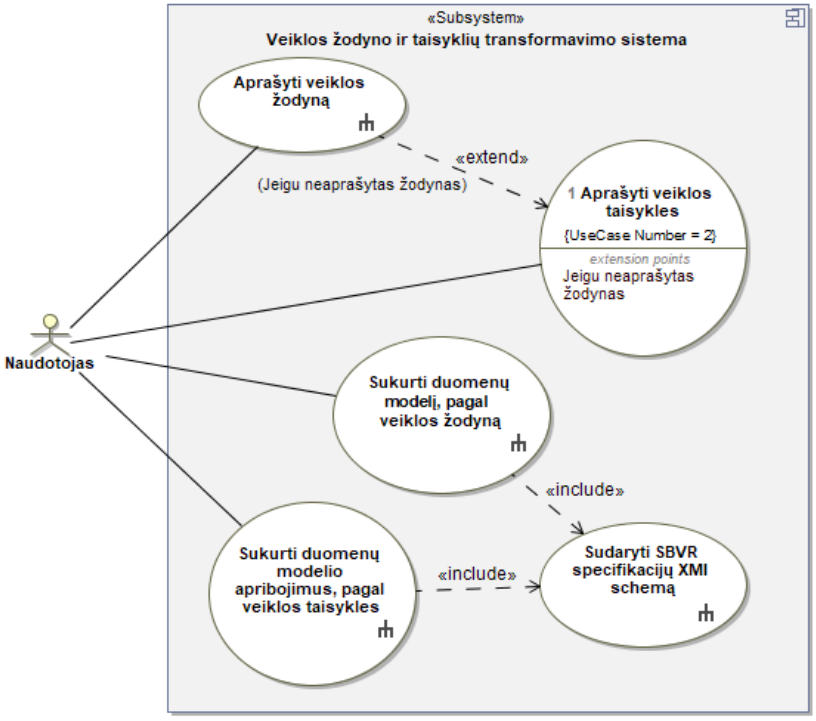
- duomenų bazės lentelių ir atributų atpažinimas – sistemai atliekant automatinį modelių transformavimą, iš veiklos žodyno ir taisyklių turi būti atpažinti objektai, transformuojami į duomenų bazės objektus;
- ryšių nustatymas – atpažinus reliacines duomenų bazės lenteles, jas sistema tarpusavyje susieja ryšiais.

Reikalavimas „Duomenų apribojimų kūrimas, naudojant veiklos taisykles" detalizuojamas į smulkesnius reikalavimus:

- nustatyti duomenų apribojimus – pagal užrašytas veiklos taisykles nustatyti apribojimai yra taikomi duomenims, kuriuos galima įterpti, atnaujinti ar ištrinti iš reliacinės duomenų bazės lentelės. Pagrindinis apribojimų tikslas yra išlaikyti duomenų vientisumą atliekant šiuos veiksmus. Sistema turi gebėti atpažinti struktūrines veiklos taisykles, ir esant objektų atitikmenims į duomenų modelį, transformuoti į duomenų bazės apribojimus;
- apribojimų keitimas pagal pasirinktas veiklos taisykles - keičiantis reikalavimams, informacinei sistemai kyla poreikis keisti veiklos taisykles. Apribojimai pagal veiklos taisyklių pokyčius gali būti ištrinti, pakeisti, papildyti.

2.1.2. Panaudojimo atvejai

Geresniam prototipo panaudojimo supratimui patiekta panaudojimų atvejų diagrama 2.2 pav. Sistemos paskirtis transformuoti *SBVR* aprašytas veiklos taisykles į *DDL* kūrimo scenarijų. Šiuo metu atliekant transformacijas išeikvojama per daug žmogiškųjų išteklių.

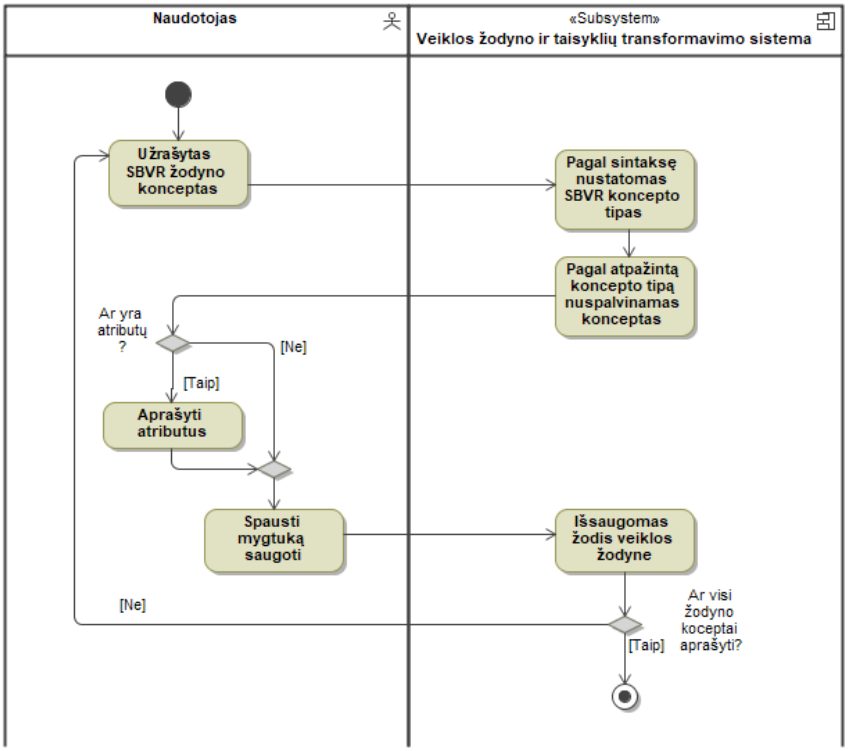


2.2 pav. Panaudojimo atvejų diagrama

2.1-2.5 lentelėse pateikiamos panaudojimo atvejų specifikacijos, o 2.3 pav.-2.7 pav. Pateikiamos panaudojimo atvejų diagramos.

2.1 lentelė. PA 1. Aprašyti veiklos žodyną

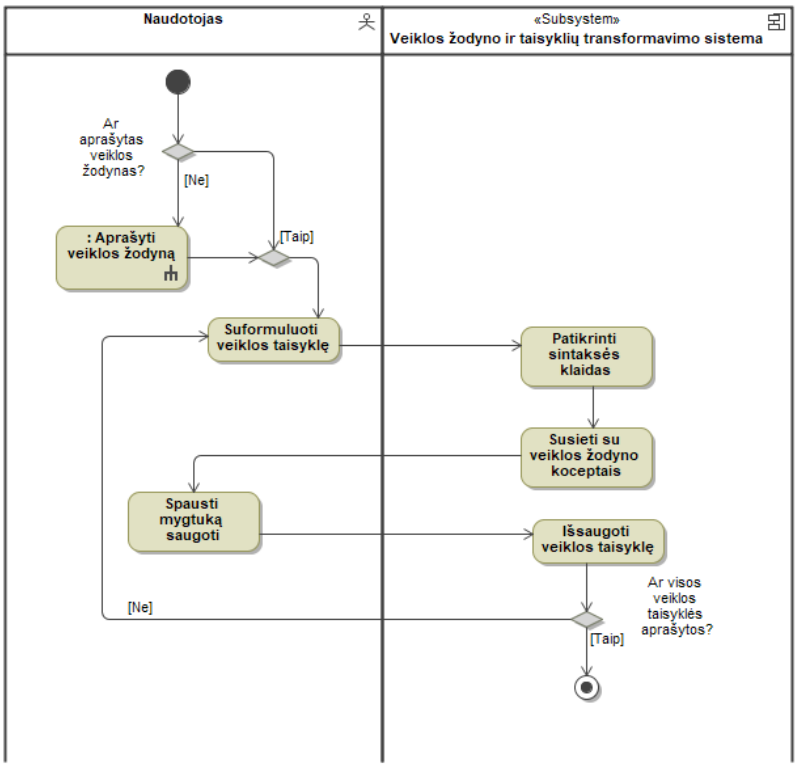
PA 1. Aprašyti veiklos žodyną		
Tikslas/uždavinys. Sukurti veiklos žodyną.		
Aprašymas. Surašomi žodyno konceptai, kuriems galima priskirti atributus. Jie išsaugomi.		
Prieš-sąlyga	Naudotojas atsidaręs veiklos žodyno ir taisyklių transformavimo į duomenų modelius programos redaktorių	
Aktorius	Naudotojas	
Sužadinimo sąlyga	Naudotojui reikia užrašyti veiklos žodyną.	
Pagrindinis scenarijus		
Pagrindinis įvykių srautas		**Sistemos reakcija**
• Sukuriamas žodyno failas .voc • Surašomi konceptai, kiekvienas iš naujos eilutės • Surašomi koncepto atributai • Paspaudžiamas mygtukas „Saugoti“		• Sukurtas failas. • Užfiksuoti žodyno konceptai nuspalvinami • Duomenys išsaugomi
Po-sąlyga		Naudotojas sėkmingai sukūrė veiklos žodyną.



2.3 pav. PA 1. „Aprašyti veiklos žodyną“ veiklos diagrama

2.2 lentelė. PA 2. Aprašyti veiklos taisykles

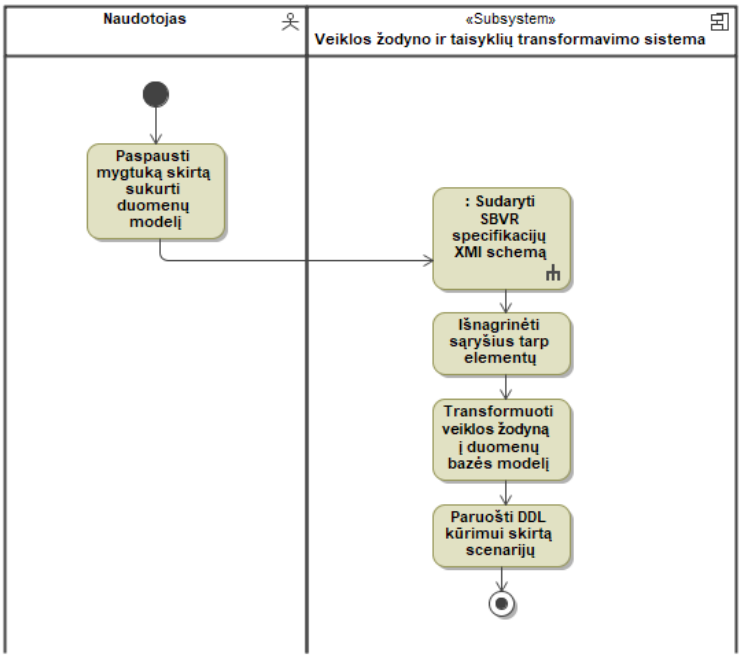
PA 2. Aprašyti veiklos taisykles	
Tikslas/uždavinys. Užrašyti veiklos taisykles	
Aprašymas. Sukuriamas veiklos taisyklių sąrašas.	
Prieš-sąlyga	Naudotojas atsidaręs veiklos žodyno ir taisyklių transformavimo į duomenų modelius programos redaktorių bei sukūręs veiklos žodyną.
Aktorius	Naudotojas
Sužadinimo sąlyga	Naudotojas nori užrašyti veiklos taisykles.
Pagrindinis scenarijus	
Pagrindinis įvykių srautas	**Sistemos reakcija**
• Sukuriamas veiklos taisyklių failas .rules • Užrašomos taisyklės naudojantis veiklos žodynu. • Paspaudžiamas mygtukas „Saugoti“	• Sukurtas failas. • Konceptai spalvinami pagal veiklos žodyno spalvas. • Duomenys išsaugomi
Po-sąlyga	Naudotojas sėkmingai sukūrė veiklos taisykles.



2.4 pav. PA 2. „Aprašyti veiklos taisykles“ veiklos diagrama

2.3 lentelė. PA 3. Sukurti duomenų modelį, pagal veiklos žodyną

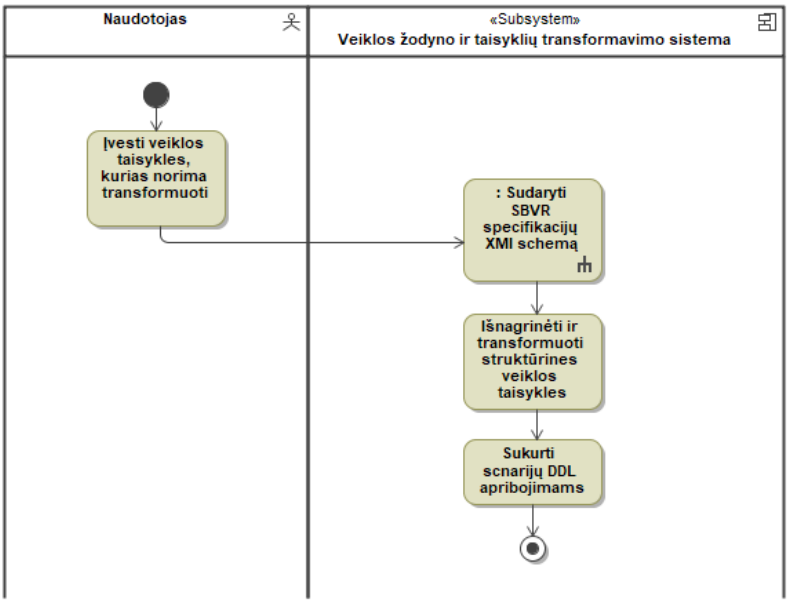
PA 3. Sukurti duomenų modelį, pagal veiklos žodyną	
Tikslas/uždavinys. Veiklos žodyno transformavimas į duomenų modelį	
Aprašymas. Veiklos žodyno analizė, sąryšių nustatymas ir transformavimas į duomenų bazės modelį	
Prieš-sąlyga	Naudotojas atsidaręs veiklos žodyno ir taisyklių transformavimo į duomenų modelius programos redaktorių bei sukūręs veiklos žodyną.
Aktorius	Naudotojas
Sužadinimo sąlyga	Naudotojas nori transformuoti veiklos žodyną į duomenų modelį
Pagrindinis scenarijus	
Pagrindinis įvykių srautas	**Sistemos reakcija**
• Paspaudžiamas mygtukas, skirtas transformuoti veiklos žodyną į duomenų modelį • Sukuriamas DDL kūrimui skirtas scenarijus	• Mygtukas paspaustas, rodomas krovimo langas. • Sukurtas scenarijus išsaugomas darbiniame kataloge
Po-sąlyga	Sėkmingai sukurtas duomenų modelis



2.5 pav. PA 3. „Sukurti duomenų modelį, pagal veiklos žodyną“ veiklos diagrama

2.4 lentelė. PA 4. Sukurti duomenų modelio apribojimus, pagal veiklos taisykles

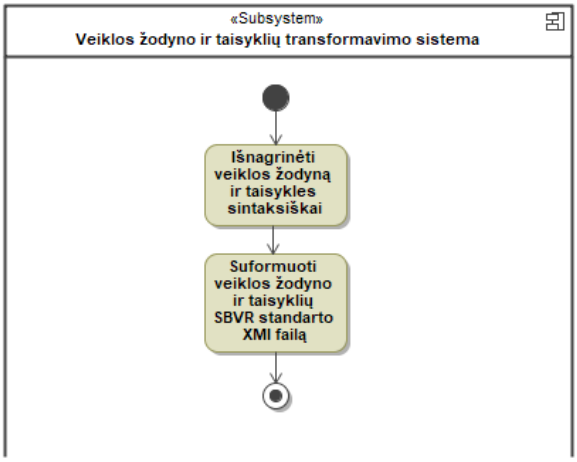
PA 4. Sukurti duomenų modelio apribojimus, pagal veiklos taisykles	
Tikslas/uždavinys. Veiklos taisyklių transformavimas į duomenų modelio apribojimus	
Aprašymas. Struktūrinių bei operacinių veiklos taisyklių analizė ir transformavimas į duomenų bazės modelio apribojimus	
Prieš-sąlyga	Naudotojas atsidaręs veiklos žodyno ir taisyklių transformavimo į duomenų modelius programos redaktorių bei sukūręs veiklos taisykles.
Aktorius	Naudotojas
Sužadinimo sąlyga	Naudotojas nori transformuoti veiklos taisykles į duomenų modelio apribojimus
Pagrindinis scenarijus	
Pagrindinis įvykių srautas	**Sistemos reakcija**
• Paspaudžiamas mygtukas, skirtas transformuoti veiklos taisykles į duomenų modelio apribojimus • Sukuriamas DDL apribojimų kūrimui skirtas scenarijus	• Mygtukas paspaustas, rodomas krovimo langas. • Sukurtas scenarijus išsaugomas darbiniame kataloge
Po-sąlyga	Sėkmingai sukurti duomenų modelio apribojimai



2.6 pav. PA 4. „Sukurti duomenų modelio apribojimus, pagal veiklos taisykles“ veiklos diagrama

2.5 lentelė. PA 5. Sudaryti SBVR specifikacijų XMI schemą

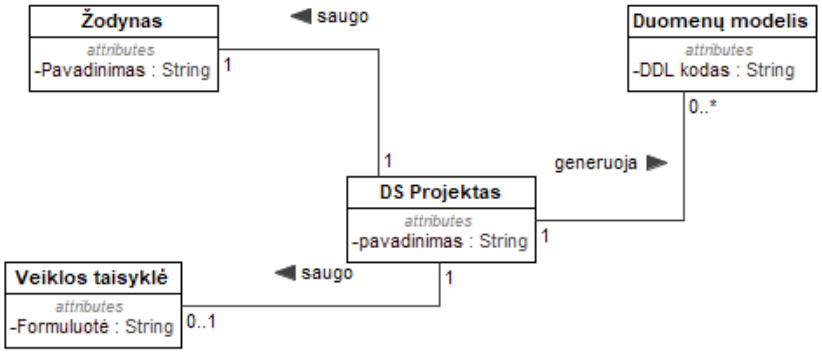
PA 5. Sudaryti SBVR specifikacijų *XMI* schemą		
Tikslas/uždavinys. Sukurti *XMI* schemą		
Aprašymas. Norint atlikti SBVR specifikacijų transformavimą, būtina sukurti *XMI* schemą.		
Prieš-sąlyga	Paspaustas mygtukas veiklos žodyno arba veiklos taisyklių transformavimui.	
Aktorius	Naudotojas	
Sužadinimo sąlyga	Naudotojas nori transformuoti veiklos žodyną į duomenų modelį arba veiklos taisykles į duomenų modelio apribojimus.	
Pagrindinis scenarijus		
Pagrindinis įvykių srautas		**Sistemos reakcija**
• Paspaustas mygtukas, skirtas transformuoti veiklos žodyną į duomenų modelį arba veiklos taisykles į duomenų modelio apribojimus		• Nagrinėjamos veiklos taisyklės arba veiklos žodynas semantiškai • Sugeneruojama *XMI* schema
Po-sąlyga		Sėkmingai sugeneruota *XMI* schema.



2.7 pav. PA 5. „Sudaryti SBVR specifikacijų XMI schemą“ veiklos diagrama

2.2. **Dalykinės srities modelis**

Sistema, gebanti transformuoti veiklos žodyną ir veiklos taisykles visų pirma turi turėti šiuos objektus kaip žodyną, aprašytas veiklos taisykles. Vienas dalykinės srities projektas turi vieną aprašytą žodyną, kurį naudojant aprašomos veiklos taisyklės. Iš projekte esančios informacijos galima sugeneruoti duomenų modelį.

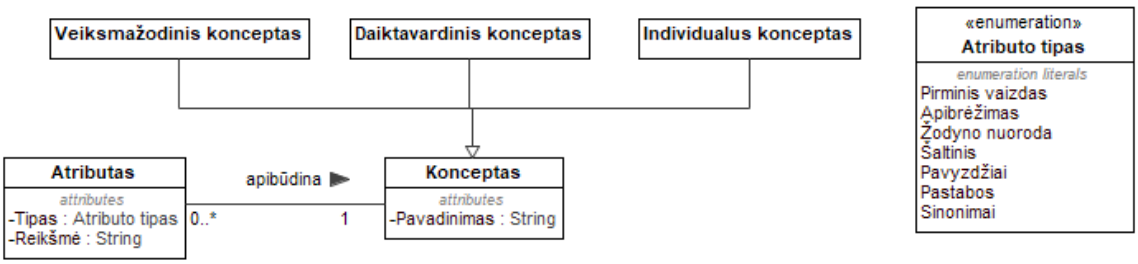


2.8 pav. Veiklos žodyno ir taisyklių projekto sudedamosios dalys

Žodyną sudaro dalykinės srities konceptai, kuriuos panaudojant konstruojamos veiklos taisyklių formuluotės.

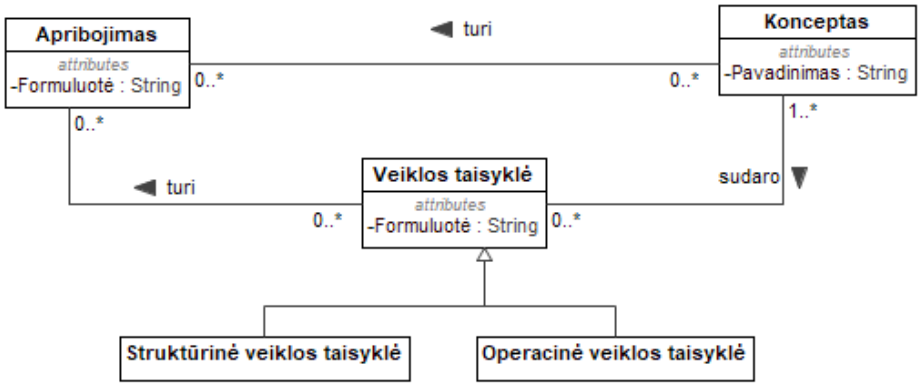
2.9 pav. Dalykinės srities koncepto priklausomybės veiklos žodyno ir taisyklių projekte

Konceptai gali būti trijų tipų: veiksmažodiniai, daiktavardiniai ir individualūs.



2.10 pav. Konceptų tipai

Taip pat kiekvienam konceptui galima priskirti atributų, pavyzdžiui apibrėžimą, nuorodą, šaltinį ar sinonimą. Kai kurie konceptai, kaip individualūs, gali būti transformuojami į apribojimus.

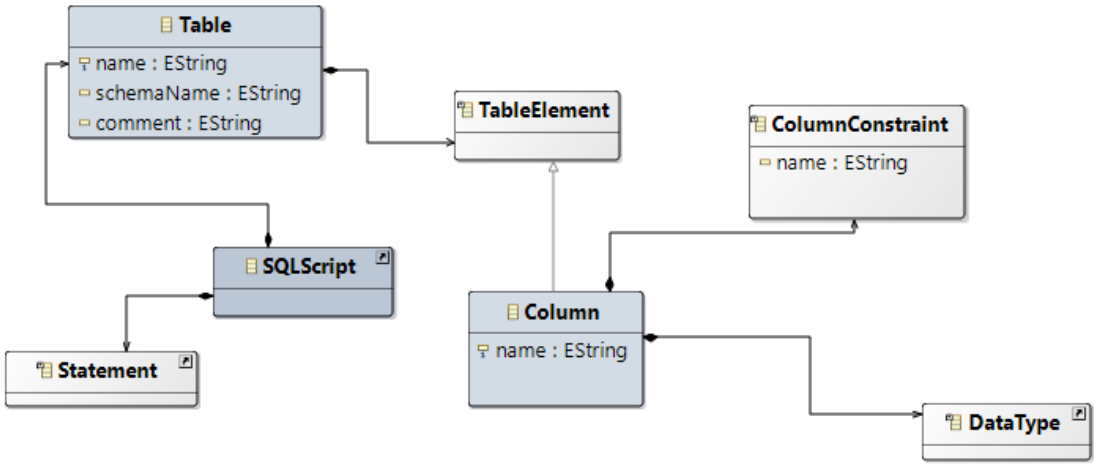


2.11 pav. Dalykinės srities veiklos taisyklių priklausomybės veiklos žodyno ir taisyklių projekte

Tačiau dauguma apribojimų atsiranda iš veiklos taisyklių formuluočių. Veiklos taisyklės skirstomos į struktūrines ir operacines, tačiau transformacijoms naudojamos tik struktūrinės veiklos taisyklės.

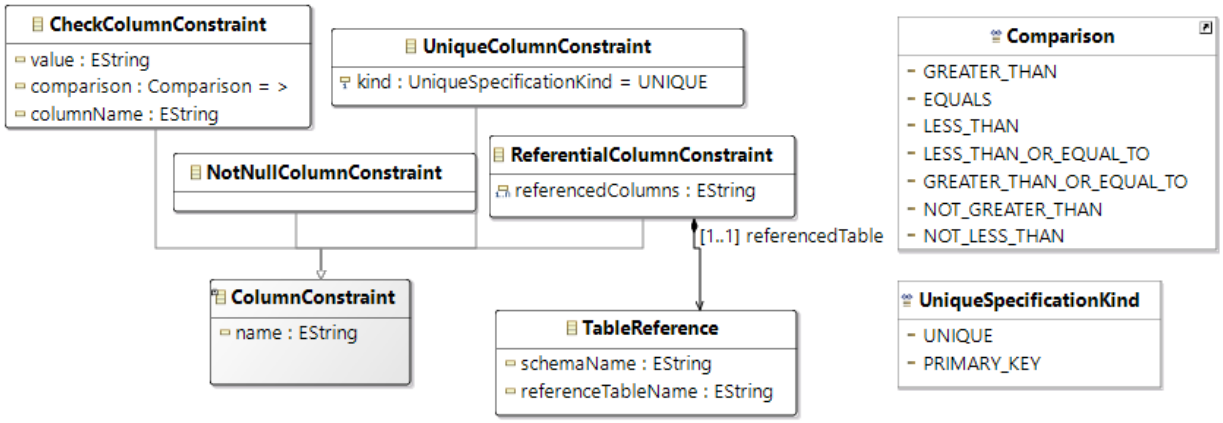
DDL ir DML metamodelis

Norint atlikti modelių transformaciją reikia turėti metamodelį iš kurio bus vykdoma transformacija, ir metamodelį, į kurį vyks transformacija. *SBVR* metamodelis nurodytas 1.5.1 skyriuje. Sudarant duomenų modelių metamodelį, pradinis objektas pasirinktas *SQLScript,* kadangi duomenų modelio kūrimo scenarijų gali sudaryti tiek lentelės kūrimas, tiek užklausos, tiek komandos ar trigeriai. Taigi ryšys iš *SQLScript* nurodytas į objektus *Table* ir *Statement* (2.12 pav.).



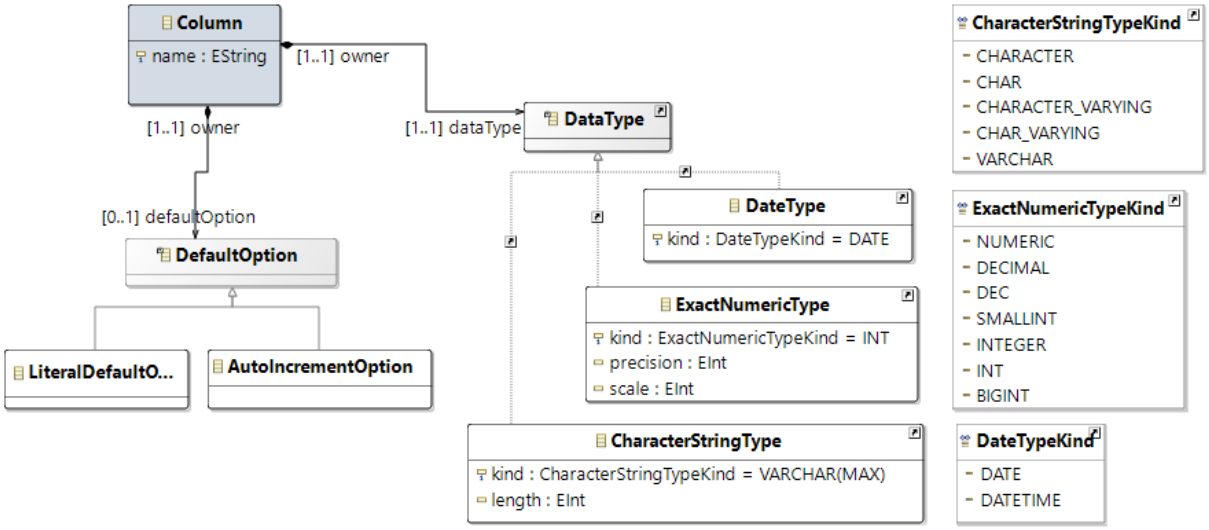
2.12 pav. Duomenų modelio kūrimui skirtas metamodelis

Sekantys pagrindiniai elementai metamodelyje yra lentelė ir stulpelis. Stulpelis privalo turėti duomenų tipą. Kadangi transformacijos svarbus aspektas yra apribojimai, išskirtas stulpelio apribojimas kaip atsiras elementas.



2.13 pav. Lentelės stulpelio apribojimų elementai

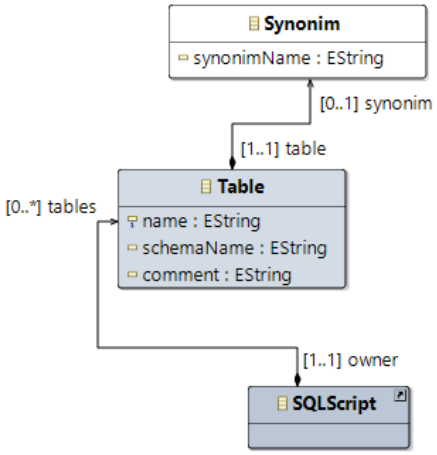
Išskirti keturi stulpelio apribojimo tipai: tuščios reikšmės, unikalumo, reikšmės patikrinimo ir ryšio su kita lentele apribojimai (2.13 pav.). Lentelės stulpelis kaip minėta aukščiau turi privalomą nurodytą duomenų tipą. Išskirtos trys tipų kategorijos: datos, skaitinė ir teksto (2.14 pav.).



2.14 pav. Lentelės stulpelio duomenų tipai ir pradinės reikšmės

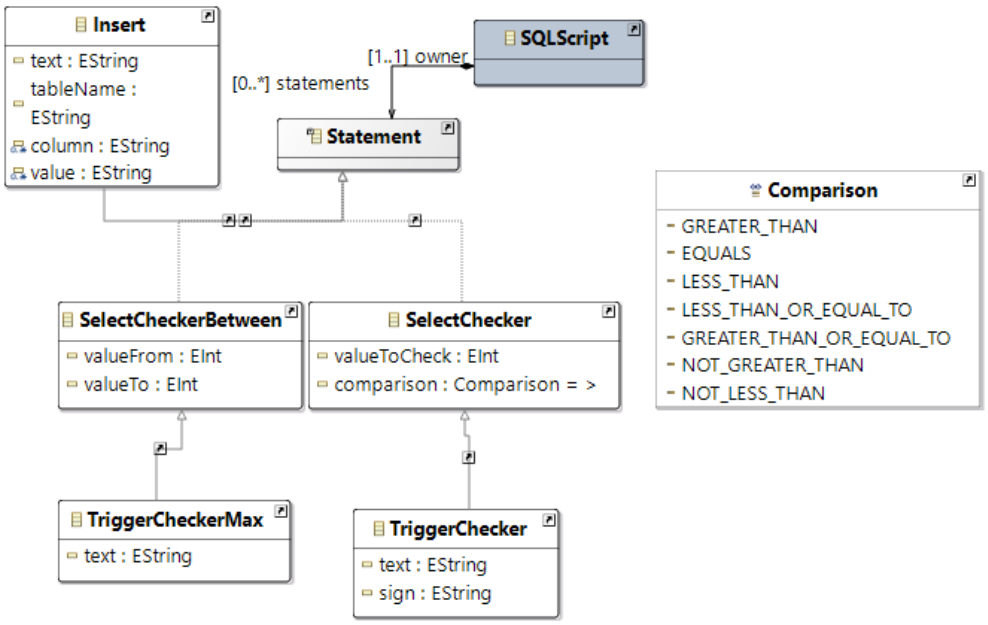
Šalia tipų 2.14 pav. matomi tipų klasifikatoriai, kurių reikšmės priskiriamos atributui *kind*. Stulpeliui galima priskirti iš anksto numatytą arba generuojamą pradinę reikšmę.

Jeigu duomenų modelyje tą pačią reliacinėje duomenų bazėje lentelę būtų norima pavadinti keliais pavadinimais, galima kitus pavadinimus realizuoti kaip sinonimus (2.15 pav.).



2.15 pav. Lentelės sinonimo priskyrimas

Kaip minėta aukščiau, reliacinę duomenų bazę gali sudaryti ne tik lentelės ar stulpelių kūrimas, tačiau ir kiti elementai. Galimai veiklos taisykles atitinkantys elementai: įrašymo į duomenų bazė komanda, SELECT užklausa ar trigeriai (2.16 pav.).

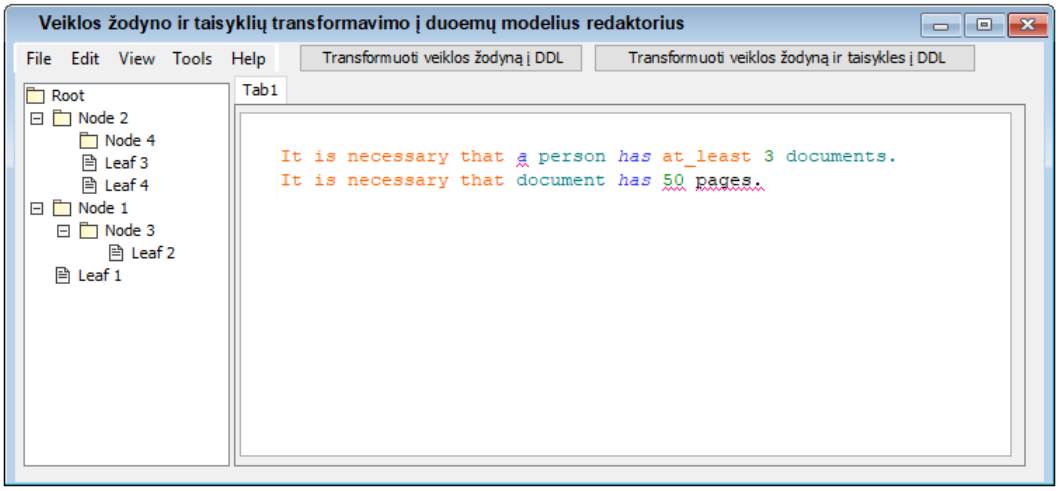


2.16 pav. Duomenų modelio elementai: komandos ir užklausos

2.3. **Naudotojų sąsajos modelis**

Kurtoje veiklos žodyno ir taisyklių transformavimo į duomenų modelius sistemoje egzistuoja vienas pagrindinis langas (2.17 pav.), kuriame naudotojas gali atlikti pagrindines funkcijas:

- sukurti projektą, kuriame saugos žodyną, taisykles, transformacijos rezultatus;
- sukurti žodyno failą, kuriame galima surašyti konceptus ir jų atributus;
- sukurti taisyklių failą, kuriame galima surašyti veiklos taisykles;
- atlikti veiklos žodyno transformaciją į duomenų modelius, neįtraukiant veiklos taisyklių, kurios gali būti transformuojamos į kardinalumus ar privalomas reikšmes;
- atlikti veiklos taisyklių transformaciją į duomenų modelius, kai įtraukiami tiek žodynas tiek taisyklės;
- peržiūrėti sugeneruotas *XMI* schemas, jeigu būtų norima jas eksportuoti į kitą sistemą arba analizuoti transformacijos tarpinius rezultatus;
- peržiūrėti transformacijos rezultatus: duomenų modelių kūrimo scenarijus, kurie išskirstyti sukurto projekto kataloge pagal tipus.

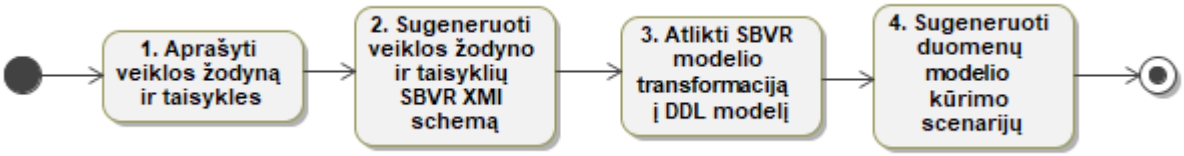


2.17 pav. Naudotojo sąsajos modelis

SBVR žodyno ir taisyklių aprašymui panaudotas jau egzistuojantis sprendimas, į kurį integruojamos minėtos transformacijos. Sukurtas metodas iškviečiamas mygtukais „Transformuoti veiklos žodyną į *DDL*“ ir „Transformuoti veiklos žodyną ir taisykles į *DDL*“.

2.4. **Veiklos žodyno ir taisyklių transformavimo metodo formalus aprašas**

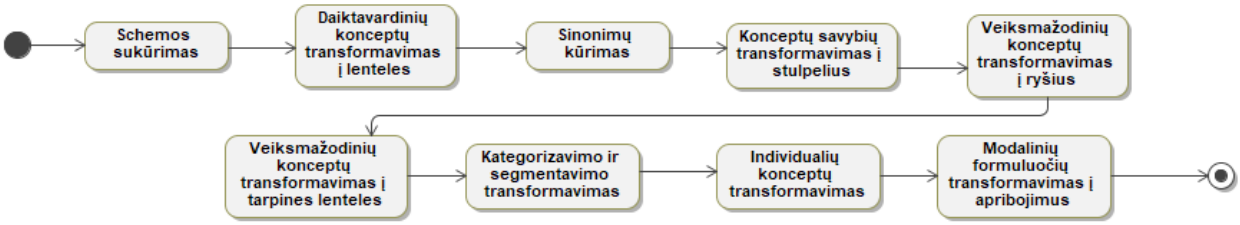
Transformacijos metodo pagrindinis tikslas yra atpažinti *SBVR* elementus, kuriuos įmanoma transformuoti į duomenų bazės išraiškas. Pagrindiniai metodo proceso žingsniai pateikti 2.18 pav.



2.18 pav. Sprendimo proceso diagrama

Pirmosios dvi dalys yra realizuotos [25], todėl šiame darbe didžiausias dėmesys skiriamas transformacijos į *DDL* metodo sukūrimui bei *SQL* kodo generavimui iš *XMI* schemos.

Transformacija iš *SBVR* metamodelio į *DDL* metamodelį pradedama nuo bendros schemos sukūrimo. Schemos pavadinimas nurodomas programiniame kode, todėl norint schemą pavadinti kitaip, teks keisti transformacijos kodą. Sekantis kviečiamas metodas transformuoja veiklos žodyne aprašytus konceptus į duomenų bazės lenteles. Patikrinama, ar vienam konceptui priskirtas vienas pavadinimas. Radus jų daugiau sukuriami lentelės sinonimai, kurie turi nuorodą į pagrindinį pavadinimą. Randamos konceptų savybės, kurios transformuojamos į lentelės stulpelius. Toliau nustatomi lentelių tarpusavio sąryšiai. Jeigu ryšys tarp lentelių reikalauja tarpinės lentelės, sukuriama lentelė, turinti pirminį raktą ir abu šalutinius raktus.

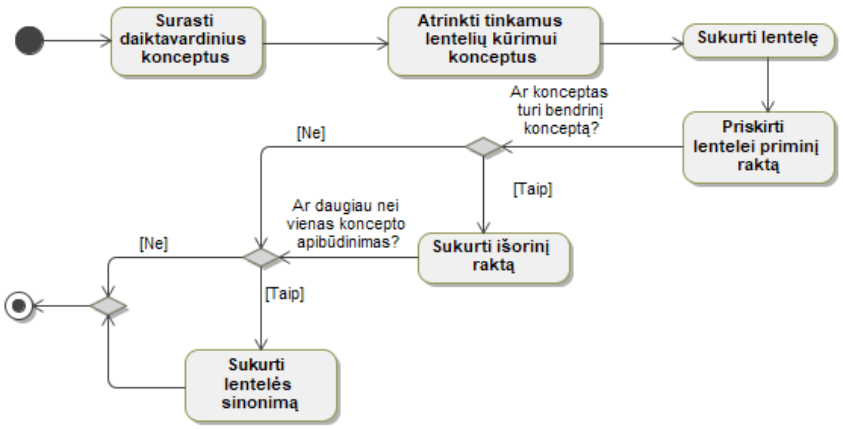


2.19 pav. Bendrinė transformacijos veiklos diagrama

Koncepto reikšmių išvardinimui galima naudoti kategorizavimą arba segmentavimą. Jeigu reikšmių aibė yra baigtinė, naudojamas segmentavimas, kuris transformuojamas į stulpelio reikšmės apribojimus. Kitu variantu aprašomas kategorizavimas, kuris transformuojamas į papildomą lentelę, naujus jos įrašus ir papildomą stulpelį, kuris nurodo ryšį su nauja lentele. Individualių konceptų reikšmės perkeliamos į naujų lentelės įrašų komandas. Transformuojant veiklos taisykles nustatomi duomenų apribojimai bei lentelių kardinalumai. Aprašytas procesas pateiktas 2.19 pav. diagramoje.

2.4.1. SBVR žodyno ir taisyklių transformavimas į duomenų bazės lenteles

Transformacijos procese svarbiausi atributai yra duomenų bazės lentelės. Lentelės pavadinimas nustatomas iš žodyno, tačiau gali būti ir pakoreguota reikšmė, atsižvelgiant į ryšių logiką.



2.20 pav. Daiktavardinių konceptų transformavimo veiklos diagrama

Surandami *SBVR* metamodelio *ObjectType* atributai, kurie gali būti transformuojami į lentelę. Priskiriamas lentelės pavadinimas, schemos vardas, komentaras apie lentelę, bet sukuriamas pirminis raktas. Jeigu rastas konceptas turi bendrinį konceptą, sukuriamas ryšys tarp dviejų lentelių.

2.6 lentelė. Žodyno atributų transformavimo į duomenų bazės lenteles žingsniai

Žodynas:

`author`
`Synonym: book_author`
`book`

Pseudo kodas:

```
in:SBVR!ObjectType
out:SQL!Table(
   name = in.expression,
   schemaName = 'dbo',
   comment = 'Table ' + in.expression,
   elements (
      SQL!Column (
         name = 'ID',
         dataType = SQL!ExactNumericType.INT,
         defaultOption = SQL!AutoIncrementOption,
         constraint = SQL!NotNullColumnConstraint,
         constraint = SQL!UniqueColumnConstraint.PRIMARY_KEY
      ),
      SQL!Column (
         name = in.general.expression + '_ID'
         dataType = SQL!ExactNumericType.INT,
         constraint = SQL!ReferentialColumnConstraint (
            name='FK' + in.expression + in.general,
            referenceTableName = in.general.expression,
            referencedColumns = 'ID'
            )
      )
   ),
   synonim = in.notPreferedTerm
)
```

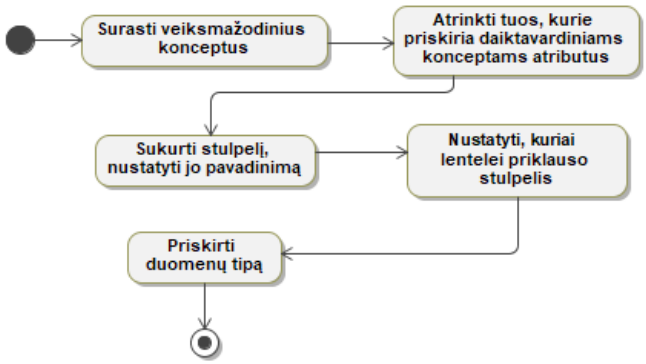
Rezultatas:

```
CREATE TABLE AUTHOR (
      ID INT IDENTITY(1,1) PRIMARY KEY NOT NULL
);
CREATE SYNONYM BOOK_AUTHOR FOR AUTHOR;

CREATE TABLE BOOK (
      ID INT IDENTITY(1,1) PRIMARY KEY NOT NULL
);
```

2.4.2. Lentelės stulpelių kūrimas

SBVR žodyne konceptai, kurie susieti su daiktavardiniais konceptais, transformuojamais į duomenų bazės lenteles, ryšiu *property_association* yra transformuojami į lentelės stulpelius. Šiuo atveju dėmesys kreipiamas tik į paprastus stulpelius duomenims saugoti (ne ryšių raktai).



2.21 pav. Atributų transformavimo veiklos diagrama

Reliacinės lentelės stulpeliai kuriami iš *SBVR* metamodelyje esančių *SBVR!FactType* elementų, kurių tipas yra *SBVR!IsPropertyOfFactType*.

2.7 lentelė. Žodyno transformacijos į stulpelius žingsniai ir pavyzdžiai

Žodynas:

```
book
pages
      General_concept: integer
book has pages
      Concept_type: property_association
title
      General_concept: text
book has title
      Concept_type: property_association
```

Pseudo kodas:

```
in:SBVR!IsPropertyOfFactType
out:SQL!Column(
   name = in.role.first.expression,
   dataType = in.role.first.type,
   owner = in.role.last
)
```

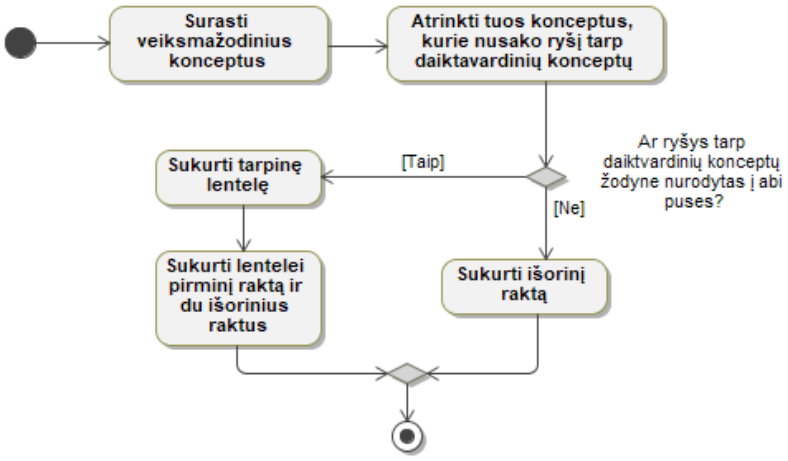
Rezultatas:

```
CREATE TABLE BOOK (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
      PAGES INT,
      TITLE VARCHAR(255)
);
```

Stulpelio tipas (tekstinis, skaičius ar pan.) nustatomas nurodant bendrinio koncepto reikšmę. Ši reikšmė konvertuojama į duomenų bazei tinkamą tipą.

2.4.3. Transformacija į ryšius tarp lentelių

Transformuojant žodyne ir taisyklėse nurodytus ryšius kuriami ryšių raktai, kurie priklausomai nuo tipo atsiranda vienoje iš lentelių arba bendroje. *SBVR* metamodelyje veiksmažodinė rolė charakterizuoja konceptą.



2.22 pav. Ryšių tarp daiktavardinių konceptų nustatymo veiklos diagrama

Sukurti ryšį tarp reliacinės bazės lentelių naudojami duomenų bazės apribojimai, pirminiai bei išoriniai raktai. *SBVR* metamodelio elementas *FactType* turi vaidmenis, kuriems yra priskiriami *ObjectType* elementai. Jeigu tarp lentelių yra ryšys vienas su daug, tada kuriamas apribojimas, bet išorinis raktas. Jeigu ryšys yra daug su daug, tuo atveju yra kuriama tarpinė lentelė, turinti abiejų lentelių pirminius raktus. Žodynas, pseudo kodas, bei transformavimo rezultatas, kai nereikia kurti tarpinės lentelės, pateikti 2.8 lentelėje.

2.8 lentelė. Žodyno transformacijos į ryšius žingsniai ir pavyzdžiai (kai nėra tarpinės lentelės)

Žodynas:

```
book
chapter
book contains chapter
```

Pseudo kodas:

```
in:SBVR!AssociativeFactType (SBVR!SententialForm.allInstances().size() = 1)
out:SQL!Column(
   name = in.role.first,
   dataType = SQL!ExactNumericType.INT,
   owner = in.role.last,
   constraint = SQL!ReferentialColumnConstraint (
      owner = in.role.first,
      name = 'FK' + in.role.last + in.role.first,
      referencedTable = TableReference (
         referenceTableName = in.role.first
      ),
      referencedColumns = 'ID'
   )
)
```

Rezultatas:

```
CREATE TABLE BOOK (
     ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY
);

CREATE TABLE CHAPTER (
     ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
     BOOK_ID INT
);

IF OBJECT_ID ('FK_CHAPTER_BOOK_1') IS NULL ALTER TABLE CHAPTER ADD CONSTRAINT FK_CHAPTER_BOOK_1
FOREIGN KEY (BOOK_ID) REFERENCES BOOK(ID);
```

Norint nustatyti ar ryšys gali būti daug su daug žiūrima, ar žodyne yra aprašytos abi susiejimo kryptys. Pavyzdys kai ryšys yra daug su daug pateiktas 2.9 lentelėje.

2.9 lentelė Žodyno transformacijos į ryšius žingsniai ir pavyzdžiai (kai reikalinga tarpinė lentelė)

Žodynas:

```
author
book
author has book
book belongs_to author
     Synonymous_form: author has book
```

Pseudo kodas:

```
in:SBVR!AssociativeFactType (SBVR!SententialForm.allInstances().size() > 1)
out:SQL!Table(
   name = in.role.first + in.role.last,
   schemaName = 'dbo',
   comment = 'Table ' + in.role.first + in.role.last,
   elements (
```

```
      SQL!Column (
         name = 'ID',
         dataType = SQL!ExactNumericType.INT,
         defaultOption = SQL!AutoIncrementOption,
         constraint = SQL!NotNullColumnConstraint,
         constraint = SQL!UniqueColumnConstraint.PRIMARY_KEY
      ),
      SQL!Column (
         name = in.role.first + '_ID',
         dataType = SQL!ExactNumericType.INT,
         constraint = SQL!ReferentialColumnConstraint (
            name = 'FK' + in.role.first,
            referencedTable = SQL!TableReference (
               referenceTableName = in.role.first
            ),
            referencedColumns = 'ID'
         ),
         constraint = SQL!NotNullColumnConstraint
      ),
      SQL!Column (
         name = in.role.last + '_ID',
         dataType = SQL!ExactNumericType.INT,
         constraint = SQL!ReferentialColumnConstraint (
            name = 'FK' + in.role.last,
            referencedTable = SQL!TableReference (
               referenceTableName = in.role.last
            ),
            referencedColumns = 'ID'
         ),
         constraint = SQL!NotNullColumnConstraint
      )
   )
)
```

Rezultatas:

```
CREATE TABLE AUTHOR (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY);

CREATE TABLE BOOK (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY);

CREATE TABLE BOOK_AUTHOR (
      BOOK_ID INT NOT NULL,
      AUTHOR_ID INT NOT NULL,
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY);

IF OBJECT_ID ('FK_BOOK') IS NULL ALTER TABLE BOOK_AUTHOR ADD CONSTRAINT FK_BOOK_1 FOREIGN KEY
(BOOK_ID) REFERENCES BOOK(ID);
IF OBJECT_ID ('FK_AUTHOR') IS NULL ALTER TABLE BOOK_AUTHOR ADD CONSTRAINT FK_AUTHOR_1 FOREIGN
KEY (AUTHOR_ID) REFERENCES AUTHOR(ID);
```

Kuriant tarpinę lentelę taip pat jau sukuriamas pirminis raktas, bei NOT NULL apribojimas abiem išoriniams raktams.

2.4.4. Individualių konceptų transformavimas

SBVR žodyne yra galimybė konceptui nurodyti konkrečią reikšmę – individualų konceptą. Nuspręsta jį transformuoti į lentelės įrašą, kadangi individualus konceptas negali turėti unikalių savybių, o jas paveldi iš savo bendrinio koncepto. Tėvinėje lentelėje sukuriamas laukas „NAME“, kuriame saugoma tekstinė reikšmė.

2.10 lentelė. Žodyno transformacijos į duomenų bazės įrašus žingsniai ir pavyzdžiai

Žodynas:

```
chapter
First_chapter
      General_concept: chapter
Second_chapter
      General_concept: chapter
```

Pseudo kodas:

```
in:SBVR!IndividualConcept
out:SQL!Insert(
   tableName = in.instantiationFormulation.concept.designation.expression,
   column = 'NAME',
   value = in.designation.expression
)
```

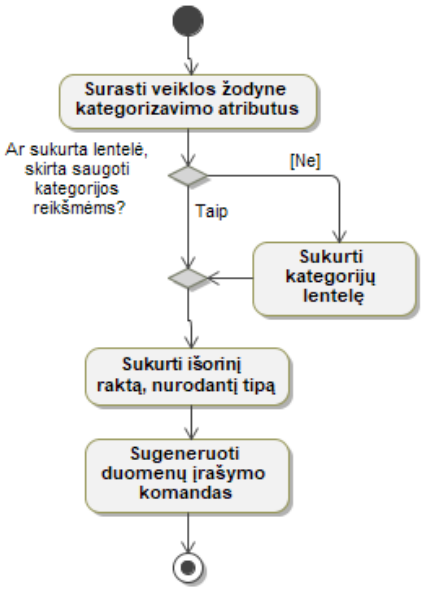
Rezultatas:

```
CREATE TABLE CHAPTER (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
      NAME VARCHAR(255));

INSERT INTO CHAPTER (NAME) VALUES ('First_chapter');
INSERT INTO CHAPTER (NAME) VALUES ('Second_chapter');
```

2.4.5. Kategorizavimas ir segmentavimas

Nurodant koncepto kategorijas, galima vienos lentelės įrašui priskirti vieną arba daugiau tam tikros kategorijos elementų. Galimų reikšmių sąrašas gali būti begalinis arba baigtinis. Jeigu veiklos žodyne aprašyta reikšmių aibė nėra baigtinė, naudojamas kategorizavimo tipas, kurio transformacijos eiga pateikta 2.23 pav.



2.23 pav. Kategorizavimo transformacija

Parašyta funkcija, kuri nustato ar reikalinga papildoma tarpinė lentelė. Jeigu papildoma lentelė nereikalinga, ir vienas įrašas gali įgyti vieną iš nurodytų tipų, sukuriamas stulpelis, kuris siejasi su lentele *CATEGORY*, kurioje ir saugomos kategorijos. Pavyzdys be tarpinės lentelės:

2.11 lentelė. Kategorizacijos transformavimo pavyzdys

Žodynas:

```
book
cover_color
      Concept_type: categorization_type
      Necessity: is_for general_concept book
Book_by_cover_color
      Necessity: categorization_scheme for general_concept book that subdivides book by
cover_color
blue
      General_concept: book
      Necessity: is_included_in Book_by_cover_color
black
      General_concept: book
      Necessity: is_included_in Book_by_cover_color
```

Pseudo kodas:

```
in:SBVR!CategorizationScheme
out:SQL!Table(
   name = 'CATEGORY',
   schemaName = 'dbo',
   comment = 'Table ' + 'CATEGORY',
   elements (
      SQL!Column (
         name = 'ID',
         dataType = SQL!ExactNumericType.INT,
         defaultOption = SQL!AutoIncrementOption,
         constraint = SQL!NotNullColumnConstraint,
         constraint = SQL!UniqueColumnConstraint.PRIMARY_KEY
      ),
      SQL!Column (
         name = 'NAME'
         dataType = SQL!ExactNumericType.VARCHAR,
      )
   )
),
out2:SQL!Column(
   name = in.isBasedOn.expression + 'TYPE_ID',
   owner = in.isFor.expression,
   dataType = SQL!ExactNumericType.INT,
   constraint = SQL!ReferentialColumnConstraint (
      owner = out,
      name = 'FK' + out.expression + 'CATEGORY',
      referencedTable = SQL!TableReference (
         referenceTableName = 'CATEGORY'
      ),
      referencedColumns = 'ID'
   )
),
out3:SQL!InsertStatement(
   tableName = 'CATEGORY',
   column = 'NAME',
   value = for(category in ic.containsCategory).expression
)
```

Rezultatas:

```
CREATE TABLE BOOK (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
      COVER_COLOR_TYPE_ID INT);

CREATE TABLE CATEGORY (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
      NAME VARCHAR(255));
```

```
IF OBJECT_ID ('FK_BOOK_CATEGORY') IS NULL ALTER TABLE BOOK ADD CONSTRAINT FK_BOOK_CATEGORY_1
FOREIGN KEY (COVER_COLOR_TYPE_ID) REFERENCES CATEGORY(ID);

INSERT INTO CATEGORY (NAME) VALUES ('Blue');
INSERT INTO CATEGORY (NAME) VALUES ('Black');
```

Jeigu konceptui sukurta reikšmių aibė, turinti kategoriją, tačiau ši aibė yra baigtinė, tuo atveju aprašoma segmentavimas, kuris transformuojamas į duomenų įrašymo apribojimą.

2.12 lentelė. Segmentacijos transformavimo pavyzdys

Žodynas:

```
book
payment
     Concept_type: categorization_type
     Necessity: is_for general_concept book
Book_by_payment
     Necessity: segmentation for general_concept book that subdivides book by payment
free
     General_concept: book
     Necessity: is_included_in Book_by_payment
paid
     General_concept: book
     Necessity: is_included_in Book_by_payment
```

Pseudo kodas:

```
in:SBVR!Segmentation
out:SQL!Column(
   name = in.isBasedOn.expression,
   owner = in.isFor.expression,
   dataType = SQL!CharacterStringType.VARCHAR,
   constraint = SQL!CheckColumnConstraint (
      owner = in.isFor.expression,
      columnName = in.isBasedOn.expression,
      comparison = SQL!Comparison.EQUALS
      value = for(category in ic.containsCategory).expression
   )
)
```

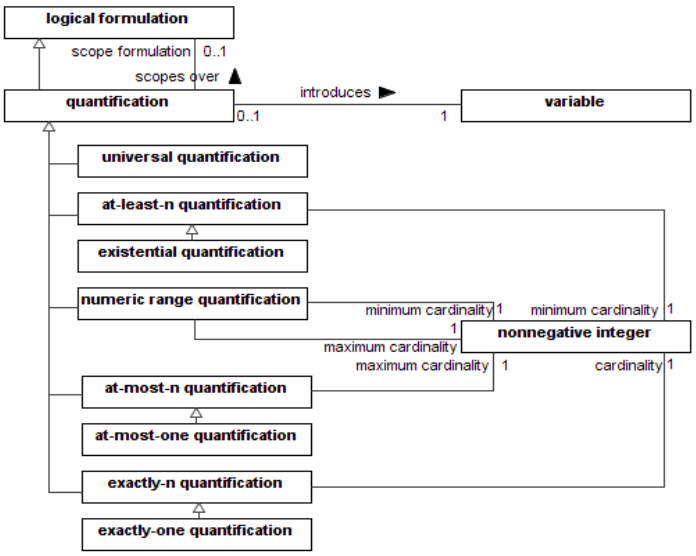
Rezultatas:

```
CREATE TABLE BOOK (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
      PAYMENT VARCHAR(255));

ALTER TABLE BOOK ADD CONSTRAINT CHK_PAYMENT CHECK (PAYMENT = 'Free' OR PAYMENT = 'Paid');
```

2.4.6. Taisyklių ir apribojimų transformavimas

Struktūrinės veiklos taisyklės gali būti užrašytos daiktavardinio koncepto atributo reikšmės apribojimui, ryšio tarp dviejų konceptų tiksliam kiekiui nustatyti ir pan. Tam naudojamos *SBVR* metamodelio elementas *SBVR!Quantifications*. *SBVR* kiekybinis nustatymas tai kintamajam nusakoma loginė formuluotė, kuri reiškia, kad ribotas susietų kintamųjų skaičius atitinka apibrėžtos apimties formuluotę.



2.24 pav. SBVR kiekybinis nustatymas [6]

Lyginant su duomenų bazės modeliu, duomenų bazėje galima apriboti tik maksimalų duomenų kiekį arba reikalauti, kad bent vienas įrašas būtų įterptas. Kardinalumo apribojimo, kuris nusako, kad turi egzistuoti mažiausiai trys susiję įrašai, deja, negalima atlikti. Šiame darbe naudojami trys pagrindiniai galimi apribojimų pritaikymo tipai:

- apribojimas NOT NULL. Jeigu tai stulpelis, galima padaryti kad įterpiant įrašą jis būtų privalomas. Jeigu tai susijusi lentelė, tai antrinis raktas visada privalo būti užpildytas;
- įterpiant arba atnaujinant įrašą kviečiamas trigeris, kuris patikrina, ar šis keitimas arba įterpimas nepažeis nustatytų veiklos taisyklių. Jeigu pažeidžia, įrašas tiesiog nesukuriamas;
- sugeneruojama užklausa, kuri patikrina ar įrašai duomenų bazėje atitinka veiklos taisykles. Pagrindinė užklausos funkcija yra aptikti ar yra minėta problema: per mažai įrašų. Naudotojo ekrane parodoma, kiek turi būti susijusių įrašų ir kiek jų šiuo metu yra.

2.4.6.1. Skaitinio kiekio diapazono nustatymas

Skaitinio diapazono kiekio nustatymas (angl. *numeric range quantification*) $\exists^{n-m}x$ reiškia, kad reikšmė, suformuluota iš *x*, turi ne mažiau kaip *n* ir ne daugiau kaip *m* atvejų, kur n=1, 2, 3, …; $n \geq 1$; m=2, 3, …; $m \geq 2$; $m >$ n).

Šiam apribojimui taikyti parinkti visi aukščiau aprašyti būdai.

2.13 lentelė. Skaitinio kiekio diapazono realizacija

Žodynas ir taisyklės:
book chapter book *contains* chapter It is necessary that book *contains* at_least 5 and at_most 9 chapter.
Pseudo kodas:
`in:SBVR!ModalFormulation (SBVR!NecessityFormulation or SBVR!ObligationFormulation)`

```
if (in.logicalFormulation.scopeFormulation.typeOf(SBVR!NumericRangeQuantification))(
   out:SQL!TriggerCheckerMax(
      parentTable = in.logicalFormulation.variable.rangedOverConcept,
      childTable = in.logicalFormulation.scopeFormulation.variable.rangedOverConcept,
      valueTo = in.logicalFormulation.scopeFormulation.maximumCardinality.value
   ),
   out2:SQL!SelectCheckerBetween(
      parentTable = in.logicalFormulation.variable.rangedOverConcept,
      childTable = in.logicalFormulation.scopeFormulation.variable.rangedOverConcept,
      valueFrom = in.logicalFormulation.scopeFormulation.minimumCardinality.value,
      valueTo = in.logicalFormulation.scopeFormulation.maximumCardinality.value
   ),
   out3:SQL!NotNullColumnConstraint(
      owner = columnAllInstances.name('FK' + out.childTable.value +
out.parentTable.value)
   ),
   out4:SQL!NotNullColumnConstraint(
      owner = columnAllInstances.name(
           in.logicalFormulation.scopeFormulation.variable.rangedOverConcept.role.value)
   )
)
```

Rezultatas:

```
CREATE TABLE BOOK (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY );

CREATE TABLE CHAPTER (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
      BOOK_ID INT NOT NULL );

IF OBJECT_ID ('FK_CHAPTER_BOOK') IS NULL ALTER TABLE CHAPTER ADD CONSTRAINT FK_CHAPTER_BOOK
FOREIGN KEY (BOOK_ID) REFERENCES BOOK(ID);

CREATE OR ALTER TRIGGER BOOK_CHAPTER_COUNT_TRIGGER ON CHAPTER
AFTER INSERT
AS
      DECLARE @COUNT int;
      DECLARE @NEW_ROW_ID int = (SELECT BOOK_ID FROM INSERTED);
      SELECT @COUNT = COUNT(*) FROM CHAPTER INNER JOIN BOOK ON BOOK_ID = BOOK.ID AND BOOK_ID =
@NEW_ROW_ID;

      IF @COUNT > 9
      BEGIN
          rollback
          RAISERROR ('Violated business rules', 16, 1);
      END
GO

SELECT *, (SELECT COUNT(*) FROM CHAPTER WHERE BOOK_ID = BOOK.ID) AS CHAPTER_COUNT_NOW, 5 AS
CHAPTER_COUNT_MUST_BE_FROM, 9 AS CHAPTER_COUNT_MUST_BE_TO
FROM BOOK WHERE (SELECT COUNT(*) FROM CHAPTER WHERE BOOK_ID = BOOK.ID) > 9 OR (SELECT COUNT(*)
FROM CHAPTER WHERE BOOK_ID = BOOK.ID) < 5;
```

2.4.6.2. Mažiausio n nustatymas

Mažiausias N kiekio nustatymas (angl. *at-least-n quantification*) $\exists^n x$ reiškia, kad kintamasis x turi mažiausiai n atvejus, kur n=1, 2, 3, … ; $n \geq 1$, ir taip nustato mažiausią atvejų kiekį. Šiam apribojimui tenkinti pasirinktas apribojimas NOT NULL, kuris apibrėžia kaip mažiausiai vieną atvejį, o patikrinimui, jeigu N yra daugiau nei vienas, naudojama SELECT užklausa.

2.14 lentelė. Mažiausio kiekio nustatymo realizacija

Žodynas ir taisyklės:

```
author
book
author has book

It is necessary that author has at_least 7 book.
```

Pseudo kodas:

```
in:SBVR!ModalFormulation (SBVR!NecessityFormulation or SBVR!ObligationFormulation)
if (in.logicalFormulation.scopeFormulation.typeOf(SBVR!AtLeastNQuantification))(
   out:SQL!SelectChecker(
      parentTable = in.logicalFormulation.variable.rangedOverConcept,
      childTable = in.logicalFormulation.scopeFormulation.variable.rangedOverConcept
      valueToCheck = in.logicalFormulation.scopeFormulation.minimumCardinality.value
      comparison = SQL!Comparison.LESS_THAN
   ),
   out2:SQL!NotNullColumnConstraint(
      owner = columnAllInstances.name('FK' + out.childTable.value +
out.parentTable.value)
   )
)
```

Rezultatas:

```
CREATE TABLE AUTHOR (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY );

CREATE TABLE BOOK (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
      AUTHOR_ID INT NOT NULL );

IF OBJECT_ID ('FK_BOOK_AUTHOR') IS NULL ALTER TABLE BOOK ADD CONSTRAINT FK_BOOK_AUTHOR FOREIGN
KEY (AUTHOR_ID) REFERENCES AUTHOR(ID);

SELECT *, (SELECT COUNT(*) FROM BOOK WHERE AUTHOR_ID = AUTHOR.ID) AS BOOK_COUNT_NOW, 7 AS
BOOK_COUNT_MUST_BE
FROM AUTHOR WHERE (SELECT COUNT(*) FROM BOOK WHERE AUTHOR_ID = AUTHOR.ID) < 7;
```

2.4.6.3. Esamo kiekio nustatymas

Esamo kiekio nustatymas (angl. *existential quantification*) ∃*x* reiškia, kad turi būti yra bent vienas *x* specifinis mažiausiai *n* kiekybinio nustatymo atvejis, kur *n*=1. Apribojimams naudojamas NOT NULL.

2.15 lentelė. Esamo kiekio nustatymo realizacija

Žodynas ir taisyklės:

```
author
name
      General_concept: text
author has name
     Concept_type: property_association

It is necessary that author has name.
```

Pseudo kodas:

```
in:SBVR!ModalFormulation (SBVR!NecessityFormulation or SBVR!ObligationFormulation)
if (in.logicalFormulation.scopeFormulation.typeOf(SBVR!ExistentialQuantification))(
   out:SQL!NotNullColumnConstraint(
      owner = columnAllInstances.name('FK'
           + in.logicalFormulation.scopeFormulation.
           variable.rangedOverConcept.object.value
```

```
            + in.logicalFormulation.variable.rangedOverConcept.value)
   ),
   out2:SQL!NotNullColumnConstraint(
      owner = columnAllInstances.name(
            in.logicalFormulation.scopeFormulation.variable.rangedOverConcept.role.value)
   )
)
```

Rezultatas:

```
CREATE TABLE AUTHOR (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
      NAME VARCHAR(255));

ALTER TABLE AUTHOR ALTER COLUMN NAME varchar(500) NOT NULL;
```

2.4.6.4. Maksimalaus kiekio nustatymas

Maksimalaus kiekio nustatymas (angl. *at-most-n quantification*) $\exists^{0..n}$ *x* reiškia, kad reikšmė, suformuluota iš *x* turi daugiausia *n* atvejų, kur *n*=1, 2, 3, …; $n \geq 1$) ir tokiu būdu apriboja maksimalų atvejų arba savybių skaičių. Apribojimams sukurti naudojamas NOT NULL ir trigeriai. SELECT užklausų generavimas nėra būtinas, kadangi duomenų bazė neleis įterpti daugiau įrašų nei galima.

2.16 lentelė. Maksimalaus kiekio nustatymo realizacija

Žodynas ir taisyklės:

```
book
publisher
book has publisher

It is necessary that book has at_most 2 publisher.
```

Pseudo kodas:

```
in:SBVR!ModalFormulation (SBVR!NecessityFormulation or SBVR!ObligationFormulation)
if (in.logicalFormulation.scopeFormulation.typeOf(SBVR!AtMostNQuantification))(
   out:SQL!TriggerCheckerMax(
      parentTable = in.logicalFormulation.variable.rangedOverConcept,
      childTable = in.logicalFormulation.scopeFormulation.variable.rangedOverConcept,
      valueTo = in.logicalFormulation.scopeFormulation.maximumCardinality.value
   ),
   out2:SQL!NotNullColumnConstraint(
      owner = columnAllInstances.name('FK' + out.childTable.value
            + out.parentTable.value)
   ),
   out3:SQL!NotNullColumnConstraint(
      owner = columnAllInstances.name(
            in.logicalFormulation.scopeFormulation.variable.rangedOverConcept.role.value)
   )
)
```

Rezultatas:

```
CREATE TABLE BOOK (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY );

CREATE TABLE PUBLISHER (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
      BOOK_ID INT NOT NULL );

IF OBJECT_ID ('FK_PUBLISHER_BOOK') IS NULL ALTER TABLE PUBLISHER ADD CONSTRAINT
FK_PUBLISHER_BOOK FOREIGN KEY (BOOK_ID) REFERENCES BOOK(ID);

CREATE OR ALTER TRIGGER BOOK_PUBLISHER_COUNT_TRIGGER ON PUBLISHER
AFTER INSERT
AS
```

```
      DECLARE @COUNT int;
      DECLARE @NEW_ROW_ID int = (SELECT BOOK_ID FROM INSERTED);
      SELECT @COUNT = COUNT(*) FROM PUBLISHER INNER JOIN BOOK ON BOOK_ID = BOOK.ID AND BOOK_ID =
@NEW_ROW_ID;

      IF @COUNT > 2
      BEGIN
          rollback
          RAISERROR ('Violated business rules', 16, 1);
      END
GO
```

2.4.6.5. Tikslios reikšmės nustatymas

Tikslaus kardinalumo reikšmės nustatymas (angl. *exactly n quantification*) $\exists^n x$ reiškia, kad reikšmė, suformuluota iš *x*, turės būtent *n* atvejus arba savybes, kur n=1, 2, 3, …; $n \geq 1$. Apribojimams sukurti taikyti visi galimi 2.4.6 skyriuje išvardinti būdai.

2.17 lentelė. Tikslaus kiekio nustatymo realizacija

Žodynas ir taisyklės:

```
author
book
author has book
book belongs_to author
      Synonymous_form: author has book

It is necessary that book belongs_to exactly 2 author.
```

Pseudo kodas:

```
in:SBVR!ModalFormulation (SBVR!NecessityFormulation or SBVR!ObligationFormulation)
if (in.logicalFormulation.scopeFormulation.typeOf(SBVR!ExactlyNQuantification))(
   out:SQL!SelectChecker(
      parentTable = in.logicalFormulation.variable.rangedOverConcept,
      childTable = in.logicalFormulation.scopeFormulation.variable.rangedOverConcept,
      valueToCheck = in.logicalFormulation.scopeFormulation.cardinality.value,
      comparison = SQL!Comparison.EQUALS
   ),
   out2:SQL!NotNullColumnConstraint(
      owner = columnAllInstances.name('FK' + out.childTable.value
            + out.parentTable.value)
   ),
   out3:SQL!TriggerChecker(
      parentTable = in.logicalFormulation.variable.rangedOverConcept,
      childTable = in.logicalFormulation.scopeFormulation.variable.rangedOverConcept,
      valueToCheck = in.logicalFormulation.scopeFormulation.cardinality.value,
      comparison = SQL!Comparison.GREATER_THAN
   ),
   out4:SQL!NotNullColumnConstraint(
      owner = columnAllInstances.name(
            in.logicalFormulation.scopeFormulation.variable.rangedOverConcept.role.value)
   )
)
```

Rezultatas:

```
CREATE TABLE AUTHOR (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY );

CREATE TABLE BOOK (
      ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY );
```

```
CREATE TABLE BOOK_AUTHOR (
	BOOK_ID INT NOT NULL ,
	AUTHOR_ID INT NOT NULL ,
	ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY );

IF OBJECT_ID ('FK_BOOK') IS NULL ALTER TABLE BOOK_AUTHOR ADD CONSTRAINT FK_BOOK FOREIGN KEY
(BOOK_ID) REFERENCES BOOK(ID);
IF OBJECT_ID ('FK_AUTHOR') IS NULL ALTER TABLE BOOK_AUTHOR ADD CONSTRAINT FK_AUTHOR FOREIGN KEY
(AUTHOR_ID) REFERENCES AUTHOR(ID);

CREATE OR ALTER TRIGGER BOOK_BOOK_AUTHOR_COUNT_TRIGGER ON BOOK_AUTHOR
AFTER INSERT
AS
	DECLARE @COUNT int;
	DECLARE @NEW_ROW_ID int = (SELECT BOOK_ID FROM INSERTED);
	SELECT @COUNT = COUNT(*) FROM BOOK_AUTHOR INNER JOIN BOOK ON BOOK_ID = BOOK.ID AND BOOK_ID
= @NEW_ROW_ID;

	IF @COUNT > 2
	BEGIN
	    rollback
	    RAISERROR ('Violated business rules', 16, 1);
	END
GO

SELECT *, (SELECT COUNT(*) FROM BOOK_AUTHOR WHERE BOOK_ID = BOOK.ID) AS BOOK_AUTHOR_COUNT_NOW, 2
AS BOOK_AUTHOR_COUNT_MUST_BE
FROM BOOK WHERE (SELECT COUNT(*) FROM BOOK_AUTHOR WHERE BOOK_ID = BOOK.ID) = 2;
```

2.4.7. Transformacijos kokybiniai reikalavimai

Sukurtam transformacijos metodo produktui keliami tam tikri kokybiniai reikalavimai;

- visiems stulpeliams nustatyti duomenų tipai ir dydžiai;
- visoms lentelėms sukurti pirminiai ir jei reikia išoriniai raktai;
- *DDL* scenarijus išskirstytas pagal tipus: lentelių kūrimo, apribojimų, trigerių, tikrinimo užklausų ir kt.;
- *DDL* scenarijus turi būti palaikomas Microsoft *SQL* Server.

2.5. **Sprendimo reikalavimų ir projekto apibendrinimas**

1. Reikalavimo specifikavimo dalyje sudaryti ir detalizuoti pagrindiniai veiklos žodyno ir taisyklių transformavimo į duomenų modelius reikalavimai:
    a. veiklos žodyno transformavimas - atpažįstamos duomenų bazės lentelės, suteikiami atitinkami pavadinimai, raktai, sukuriami ryšiai tarp lentelių;
    b. duomenų apribojimų kūrimas – apribojimai nustatomi pagal veiklos taisykles.
2. Sudarytas panaudojimo atvejų modelis, kuriame atvaizduojamos naudotojui galimos funkcijos:
    a. veiklos žodyno aprašymas;
    b. veiklos taisyklių aprašymas;
    c. veiklos žodyno transformavimas į duomenų modelius;
    d. duomenų apribojimų kūrimas, naudojant veiklos taisykles.
3. Kiekvienas panaudojimo atvejis detalizuotas veiklos diagrama bei lentele.

4. Sukurtas duomenų modelių kūrimo metamodelis. Išskirti pagrindiniai metamodelio elementai.
5. Pateikta planuojamo sprendimo vizualizacija, tai yra sprendimo pagrindinis langas, ir jame esančios funkcijos.

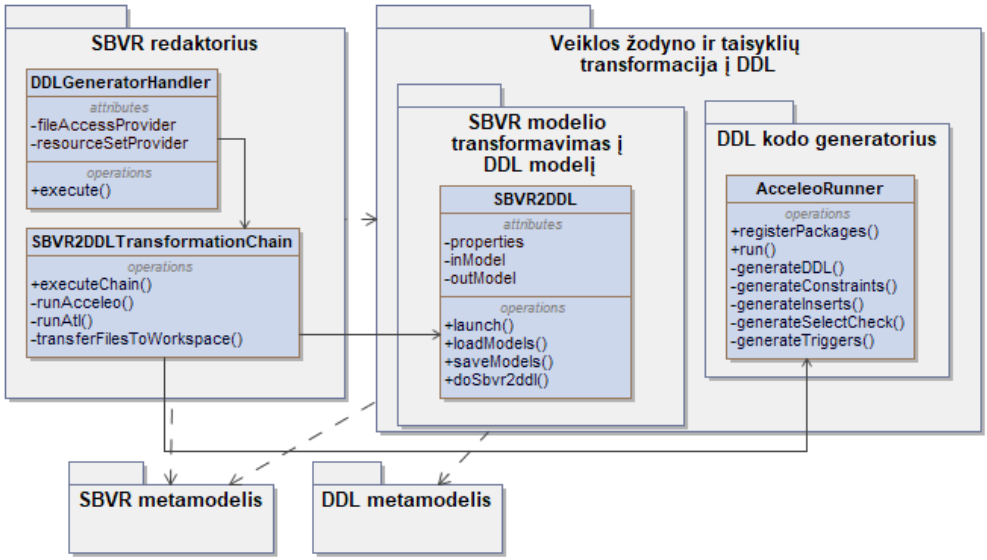
3. SBVR TRANSFORMAVIMO Į DUOMENŲ MODELIUS SPRENDIMO PROJEKTAS

Veiklos žodyno ir taisyklių transformavimo į duomenų modelius metodo realizaciją sudaro šios dalys:

- sistemos loginė architektūra;
- sistemos detalizuota loginė architektūra;
- diegimo modelis;

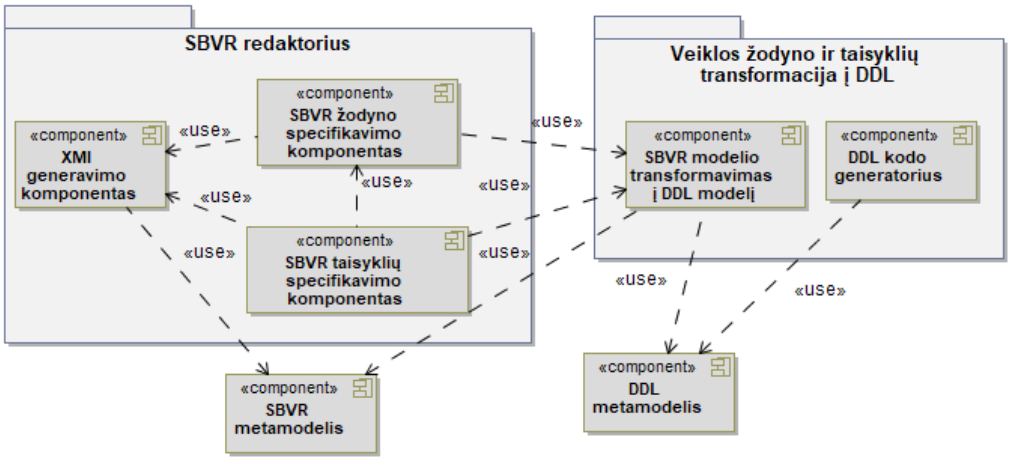
3.1. **Sistemos loginė architektūra**

Sistemos loginėje architektūroje atvaizduojamas pagrindinis kuriamo metodo paketas – veiklos žodyno ir taisyklių transformavimas į duomenų modelius, kuris yra panaudojamas *SBVR* anksčiau sukurtame redaktoriuje. Modelių transformacijos bei kodo generavimas atliktas remiantis *SBVR* ir *DDL* metamodeliais, kurie aprašyti 1.5.1 ir 2.2 skyriuose.

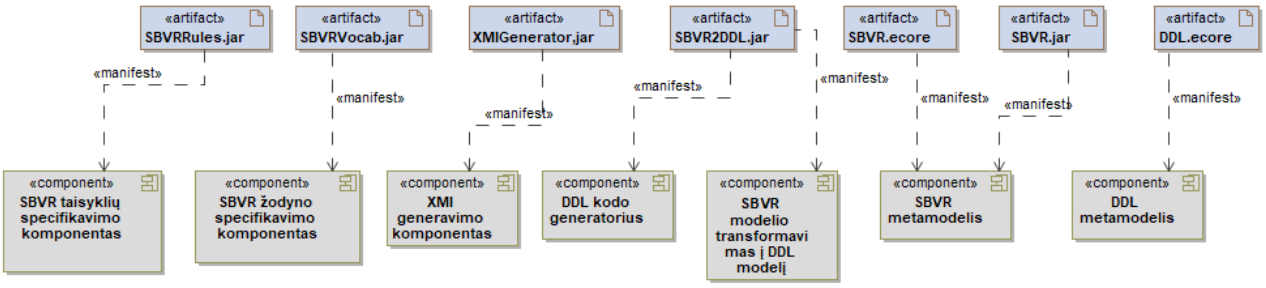


3.1 pav. Detalizuota sistemos loginė architektūra

Detalizuotoje sistemos loginėje architektūroje (3.1 pav.) pateikiami pagrindinių klasių atributai bei metodai. *SBVR* redaktorius papildytas *DDLGeneratorHandler*, kuri patikrina, ar projekte yra sukurtas žodyno ir veiklos taisyklių failai. Jeigu jie sukurti, inicijuojamas transformavimo procesas, t.y. kviečiama *SBVR2DDLTransformationChain* klasė. Jos viduje kviečiami metodai inicijuoja modelių transformavimą, bei kodo generavimą į *DDL* scenarijų.



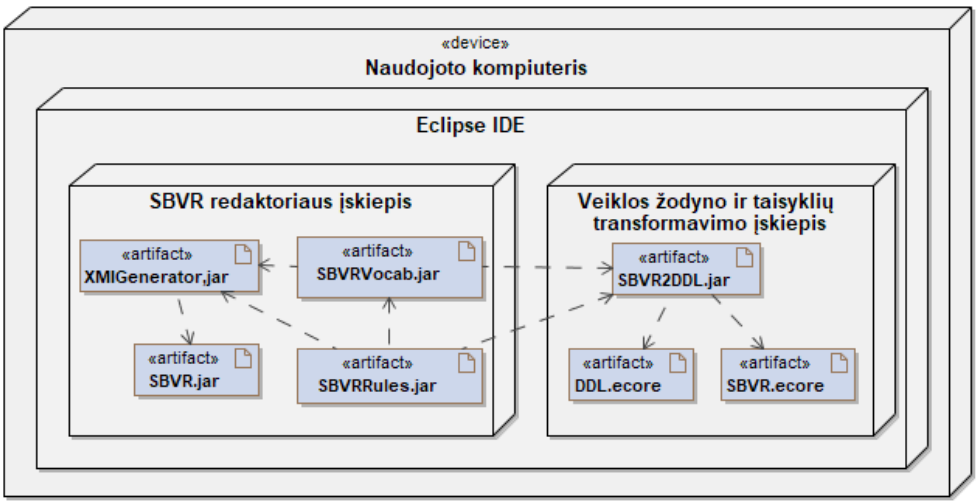
3.2 pav. Sistemos komponentų diagrama



3.3 pav. Sistemos artefaktų ir komponentų sąsaja

3.2. **Modelių transformavimo sistemos diegimo modelis**

SBVR žodyno ir taisyklių transformacijos į duomenų modelius metodą galima vykdyti įsirašius *SBVR* redaktorių, kuris papildytas reikiamu šiame darbe kurtu moduliu. Redaktorius kurtas naudojant *Eclipse* įrankį. *Xtext* įskiepis reikalingas veiklos žodyno ir taisyklių užrašymui. Modelių transformacijai naudojamas *ATL* įskiepis bei *SBVR2DDL.alt* failas, o *SQL* kalba parašyto kodo generavimui naudojamas *Acceleo* įskiepis.



3.4 pav. Modelių transformavimo sistemos diegimo modelis

3.3. **Realizacijos projekto apibendrinimas**

1. Pateikti pagrindiniai modeliai, pagal kuriuos kurta metodo realizacija: sistemos loginė architektūra, sistemos detalizuota loginė architektūra, diegimo modelis;
2. Loginėje architektūroje pateiktos pagrindinės sistemos dalys ir jų naudojami komponentai. Detalizuotoje loginėje architektūroje parodytos pagrindinės realizacijos klasės, jų metodai, bei sąsajos;
3. Norint vykdyti programą naudotojo kompiuteryje reikia turėti papildomą programinę įrangą su reikalingais įskiepiais, bei projekto jar failus, kuriuose saugomas sistemos funkcionalumas.

4. SPRENDIMO REALIZACIJA IR TESTAVIMAS

4.1. Sprendimo realizacijos ir veikimo aprašas

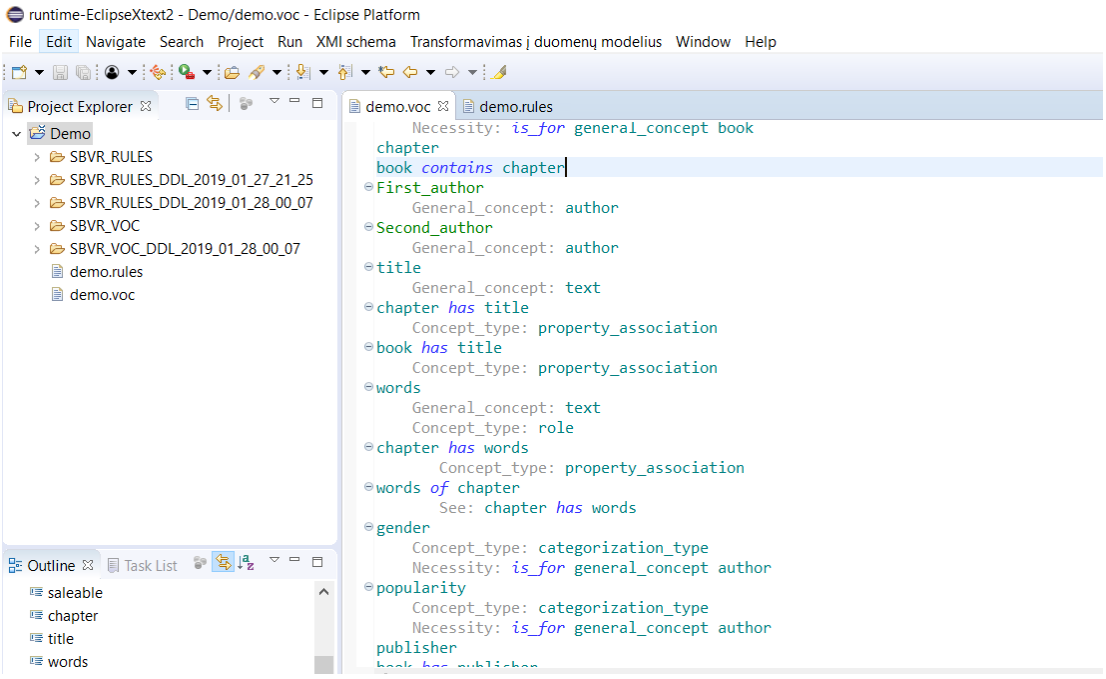
Sprendimo realizacija

Sprendimo realizacija – veiklos žodyno ir taisyklių transformavimo į duomenų modelius sistema. Programa sukurta naudojant *Java*, *ATL* modelių transformavimo kalbas, *Xtext* įrankį, bei *Acceleo*. *Acceleo* yra kodo generatorius, skirtas efektyviai įgyvendinti MDA požiūrį ir pagerinti programinės įrangos kūrimo produktyvumą.

Sistemos veikimo principas

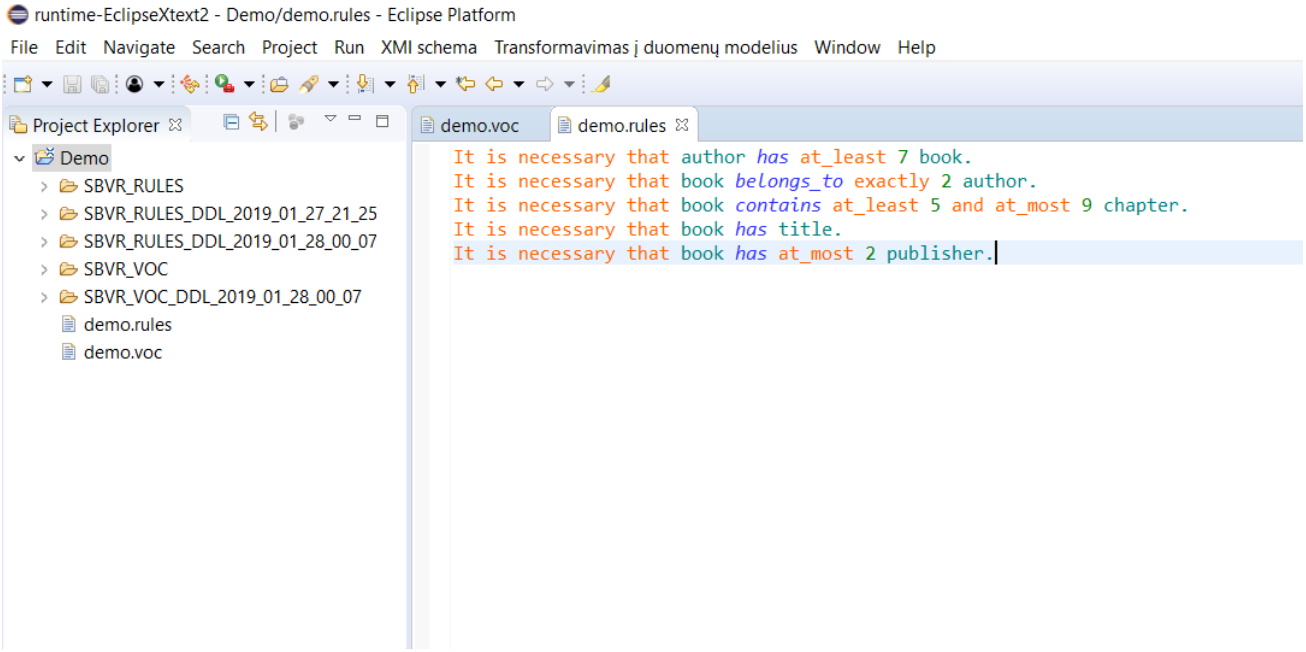
Sistema turi vieną pagrindinį langą, kuriame naudotojas gali kurti *SBVR* veiklos žodyną bei taisykles.

Susikuriamas projektas, kuris turi būti *Xtext* tipo. Jame galima sukurti failą, skirtą saugoti žodynui. Failo pavadinimas nėra svarbus, tik būtina, kad jo plėtinys būtų „voc“. Žodyne žodžiai rašomi iš naujos eilutės, sąryšiai nurodomi žemiau, nei aprašyti terminai. Naudojant *Xtext* įrankį terminai yra spalvinami pagal nustatytus tipus.



4.1 pav. SBVR žodyno failas

Veiklos taisyklėm saugoti sukuriamas tokio pat pavadinimo failas, kurio plėtinys turi būti „rules“.



4.2 pav. SBVR taisyklių failas

Realizuoto sprendimo funkcijos

4.1 lentelė. Realizuotame sprendime atliekamos funkcijos ir jų veikimo principas

Nr.	Funkcija	Veikimo principas
1.	Žodyno kūrimas	Susikuriamas failas su plėtiniu voc.
2.	Taisyklių kūrimas	Susikuriamas failas su plėtiniu rules.
3.	Žodyno transformavimas į duomenų modelių kūrimo scenarijų	Paspaudus meniu esantį mygtuką „Transformavimas į duomenų modelius“ -> „Sukurti veiklos žodyno DDL“ sistema automatiškai atlieka transformaciją, sukuria aplankalą, kurio pavadinime atsispindi sukūrimo data ir laikas. Jame scenarijai išsaugomi atskiruose failuose pagal tipą, tai pat sukuriamas failas „script.sql“, kuriame sudėta visa užklausų seka.
4.	Žodyno ir taisyklių transformavimas į duomenų modelių kūrimo scenarijų	Analogiškai paspaudus meniu punkte „Transformavimas į duomenų modelius“ -> „Sukurti veiklos žodyno ir taisyklių DDL“ sistema atlieka tuos pačius veiksmus kaip aukščiau aprašyti, tačiau šį kartą įtraukia ir aprašytas veiklos taisykles.

Žodynas

Žemiau pateikiamas žodyno pavyzdys, apimantis daugelį *SBVR* teikiamų žodyno funkcijų

```
author
book
author has book
book belongs_to author
            Synonymous_form: author has book
pages
```

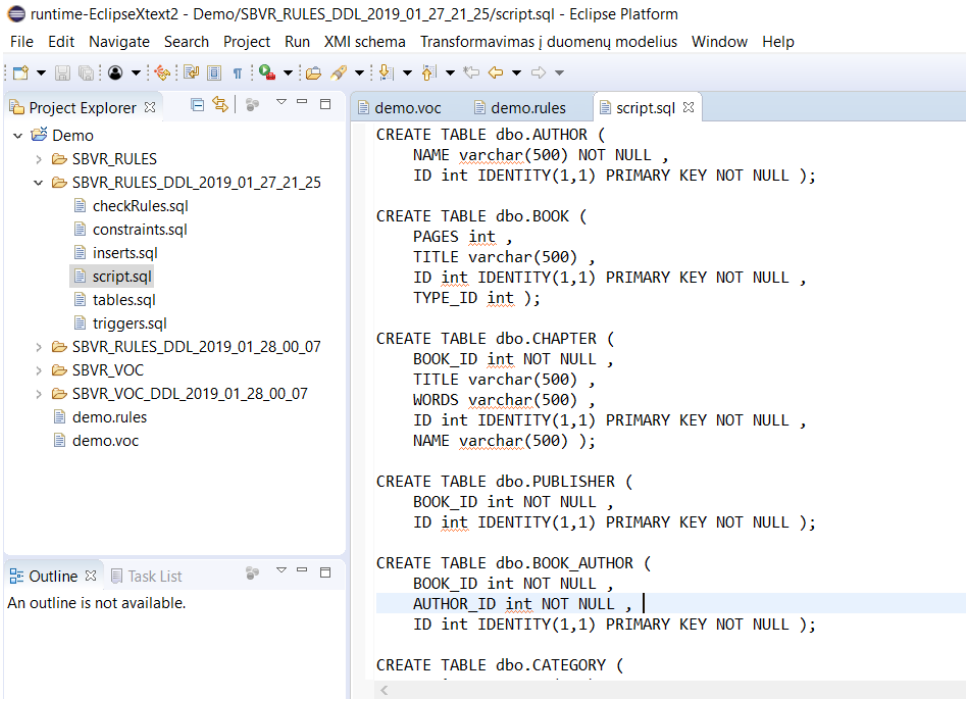
```
            General_concept: integer
book has pages
  Concept_type: property_association
saleable
            Concept_type: categorization_type
            Necessity: is_for general_concept book
chapter
book contains chapter
First_author
            General_concept: author
Second_author
            General_concept: author
title
            General_concept: text
chapter has title
            Concept_type: property_association
book has title
            Concept_type: property_association
words
            General_concept: text
            Concept_type: role
chapter has words
       Concept_type: property_association
words of chapter
       See: chapter has words
gender
            Concept_type: categorization_type
            Necessity: is_for general_concept author
popularity
            Concept_type: categorization_type
            Necessity: is_for general_concept author
publisher
book has publisher
```

Taisyklės

Žemiau pateikiamas veiklos taisyklių pavyzdys, kuris naudoja aukščiau pateiktą žodyną:

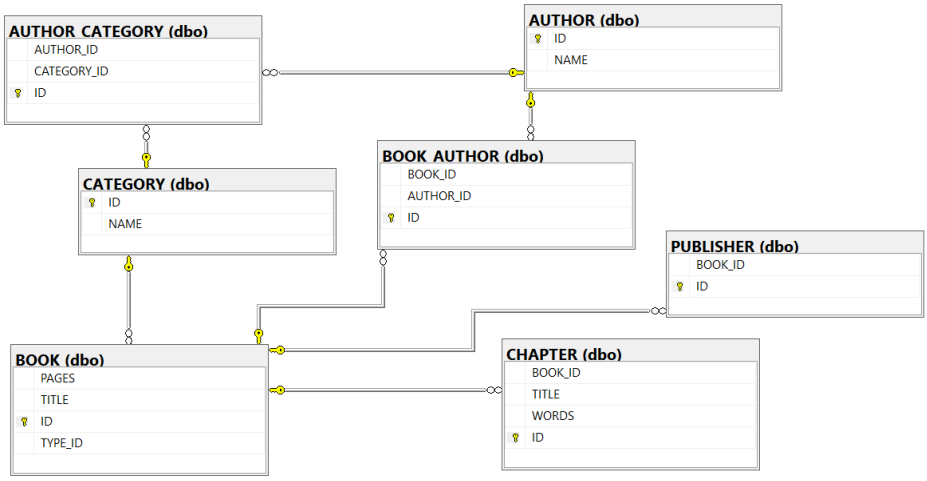
```
It is necessary that author has at_least 7 book.
It is necessary that book belongs_to exactly 2 author.
It is necessary that book contains at_least 5 and at_most 9 chapter.
It is necessary that book has title.
It is necessary that book has at_most 2 publisher.
```

Atlikus veiklos žodyno ir taisyklių automatinį transformavimą į duomenų bazės kūrimo scenarijų, ir jį įvykdžius duomenų bazėje, gauname štai tokį scenarijų:



4.3 pav. Duomenų bazės elementų kūrimo scenarijus

Įvykdžius užklausas duomenų bazėje gaunama duomenų bazės lentelių diagrama:



4.4 pav. Atlikus automatinį transformavimą ER diagrama

4.2. **Sprendimo testavimas**

Sistemos veikimo patikrinimui pasirinktas kiekvienos transformavimo taisyklės izoliuotas nagrinėjimas, pateikiant duomenis skirtus tos taisyklės veikimo patikrinimui. Nagrinėjamos metamodelių *XMI* schemos. Rezultatai pateikti 4.2 lentelėje.

4.2 lentelė. Testavimo duomenys ir rezultatai

Taisyklė	Duomenys	Rezultatas	Ar atitinka norimą?
ObjectType2 Table	author book	Sukurtos dvi lentelės su pirminiais raktais.	Taip
	author auhor	Sukurtos dvi identiškos lentelės.	Ne
	author Synonym: book_author	Sukurta lentelė ir jos sinonimas.	Taip
	person author General_concept: person	Sukurtos dvi lentelės, autorių lentelė turi nuorodą į žmonių lentelę.	Taip
Associative FactType2 Constraint	author book author *has* book	Sukurtas ryšys iš knygos lentelės į autoriaus lentelę.	Taip
Associative FactType2 Table	author book author *has* book book *belongs_to* author Synonymous_form: author *has* book	Sukurta tarpinė lentelė *BOOK_AUTHOR*, turinti nuorodas į abi lenteles.	Taip
Categorization SchemeAnd Segmentation Trasformation	book payment Concept_type: categorization_type Necessity: *is_for* general_concept book Book_by_payment Necessity: segmentation *for* general_concept book that *subdivides* book by payment free_event General_concept: book Necessity: *is_included_in* Book_by_payment paid_event General_concept: book Necessity: *is_included_in* Book_by_payment	Lentelėje BOOK sukurtas laukas PAYMENT, kuriam sukurti įvesties apribojimai	Taip
	book genre Concept_type: categorization_type Necessity: *is_for* general_concept book Genres Necessity: categorization_scheme *for* general_concept book that *subdivides* book by genre art General_concept: book Necessity: *is_included_in* Genres autobiography General_concept: book Necessity: *is_included_in* Genres	Sukurta lentelė *CATEGORY* su lauku *NAME*, kurioje įterpti įrašai su reikšmėmis *art* ir *autobiography*. Knygų lentelėje sukurtas laukas *GENRE_TYPE_ID*, kuris turi nuorodą į kategorijų lentelę.	Taip
IsPropertyOf FactType2 Column	book title book *has* title Concept_type: property_association pages General_concept: integer book *has* pages Concept_type: property_association	Sukurta knygų lentelė su stulpeliais saugoti pavadinimui ir puslapių skaičiui.*Title* stulpelio duomenų tipas yra tekstas, o *pages* duomenų tipas – skaitinė reikšmė.	Taip

Individual Concept2Insert	author Ayn_Rand General_concept: author Ernest_Hemingway General_concept: author	Sukurta autoriaus lentelė, su papildomu lauku, skirtu saugti koncepto pavadinimui. Sugeneruotos duomenų įterpimo komandos.	Taip
AtLeastN Quantification	It is necessary that author *writes* at_least 7 book.	Sukurta užklausa, kuri išrenka įrašus, kurie netenkina sąlygos. Jeigu knygų lentelėje bus mažiau nei 7 įrašai susieti su tuo pačiu autoriumi, užklausa parodys nepilnų knygų sąrašą, ir reikšmę, kiek jų turėtų būti.	Taip
Existential Quantification	It is necessary that author *writes* book.	Ryšiui tarp autoriaus ir knygos priskirtas NOT NULL apribojimas.	Taip
ExactlyN Quantification	It is necessary that author *writes* exactly 7 book.	Sukurta užklausa, kuri išrenka įrašus, kurie netenkina sąlygos, t.y. knygų sąrašą, kurių tenka mažiau nei 7 vienam autoriui. Taip pat sukurtas trigeris, kuris neleidžia vienam autoriui priskirti daugiau nei 7 knygų.	Taip
AtMostN Quantification	It is necessary that author *writes* at_most 3 books.	Sukurtas trigeris, kuris neleidžia autoriui priskirti daugiau nei 3 knygų.	Taip
NumericRange Quantification	It is necessary that author *writes* at_least 5 and at_most 9 book.	Ryšiui tarp autoriaus ir knygos priskirtas NOT NULL apribojimas. Sukurtas trigeris, kuris neleidžia autoriui priskirti daugiau nei 9 knygų. Sukurta užklausa, kuri išrenka knygas, kurių priskirta mažiau nei penkios vienam autoriui.	Taip
Atomic Formulation2 Constraint	It is necessary that pages *of* book less_than 9999. It is necessary that pages *of* book greater_than 10.	Knygos lentelei sukurtas stulpelis, skirtas knygos puslapių skaičiui saugoti. Sukurti CHECK apribojami šiam stulpeliui.	Taip
	It is necessary that pages *of* book less_than 9999 and pages *of* book greater_than 10.	Nesukurti apribojimai	Ne

Atliktas programos transformacijas galima matyti redaktoriaus konsolėje:

```
CREATED TABLE (ObjectType): 'PERSON'
CREATED TABLE (ObjectType): 'BOOK'
CREATED TABLE (ObjectType): 'AUTHOR'
CREATED TABLE (FactType): 'BOOK_AUTHOR'
Created Check Constraint for column: PAYMENT ->  free_event: ''
Created Check Constraint for column: PAYMENT ->  paid_event: ''
CREATED TABLE (CategorizationType): 'CATEGORY'
Created insert statement. COL: NAME VALUE: art TABLE: 'CATEGORY'
Created insert statement. COL: NAME VALUE: autobiography TABLE: 'CATEGORY'
```

```
Table: BOOK. Created column: : OUT!TITLE
Table: BOOK. Created column: : OUT!PAGES
Created insert statement. COL: NAME VALUE: Ayn_Rand TABLE: 'AUTHOR'
Created insert statement. COL: NAME VALUE: Ernest_Hemingway TABLE: 'AUTHOR'
------------  AtLeastNQuantification: '------------'
Created SelectCheck for tables: AUTHOR, BOOK_AUTHOR. Value to check: 7. Sign: lessThen
------------  ExistentialQuantification: '------------'
Created NOT NULL constraint for collumn: FK_BOOK_AUTHOR_2
------------  ExactlyNQuantification: '------------'
Created NOT NULL constraint for collumn: FK_BOOK_AUTHOR_2
Created SelectCheck for tables: AUTHOR, BOOK_AUTHOR. Value to check: 7. Sign: equals
Created Trigger for tables: AUTHOR, BOOK_AUTHOR. Value to check: 7. Sign: moreThen
------------  AtMostNQuantification: '------------'
Created NOT NULL constraint for collumn: FK_BOOK_AUTHOR_2
Created Trigger for tables: AUTHOR, BOOK_AUTHOR. Value to check (max): 3
------------  NumericRangeQuantification: '------------'
Created NOT NULL constraint for collumn: FK_BOOK_AUTHOR_2
Created SelectCheck for tables: AUTHOR, BOOK_AUTHOR. Value to check between: 5 and 9
Created Trigger for tables: AUTHOR, BOOK_AUTHOR. Value to check (max): 9
```

Jeigu žodynas ar taisyklės aprašytos nekorektiškai, konsolėje galimai atsiras klaida, tačiau tai priklauso nuo individualaus atvejo.

5. EKSPERIMENTINIS SBVR MODELIŲ TRANSFORMAVIMO Į DUOMENŲ MODELIIUS SISTEMOS TYRIMAS

5.1. Eksperimento planas

Siekiant įvertinti sukurto metodo, skirto duomenų modeliams kurti tinkamumą, atliekama eksperimentinė dalis, kurios planas susideda iš:

4. Aprašomos kelios dalykinės sritys, kurios gali būti pritaikomos kuriant duomenų modelius:
    a. darbuotojų informacinė sistema;
    b. renginių bilietų informacinė sistema;
    c. knygų informacinė sistema.
5. Nubraižomos ER diagramos kiekvienai aprašytai dalykinei sričiai.

Pagal dalykinę sritį nustatomos esybės bei ryšiai. Sužymimi kardinalumai, kurie nebūtinai įmanomi duomenų bazės modeliuose, tačiau atitinka veiklos taisykles.

6. Sukuriamas veiklos žodynas bei veiklos taisyklės.

Dalykinės srities aprašymui naudojama struktūrizuota anglų kalba *SBVR* standartu.

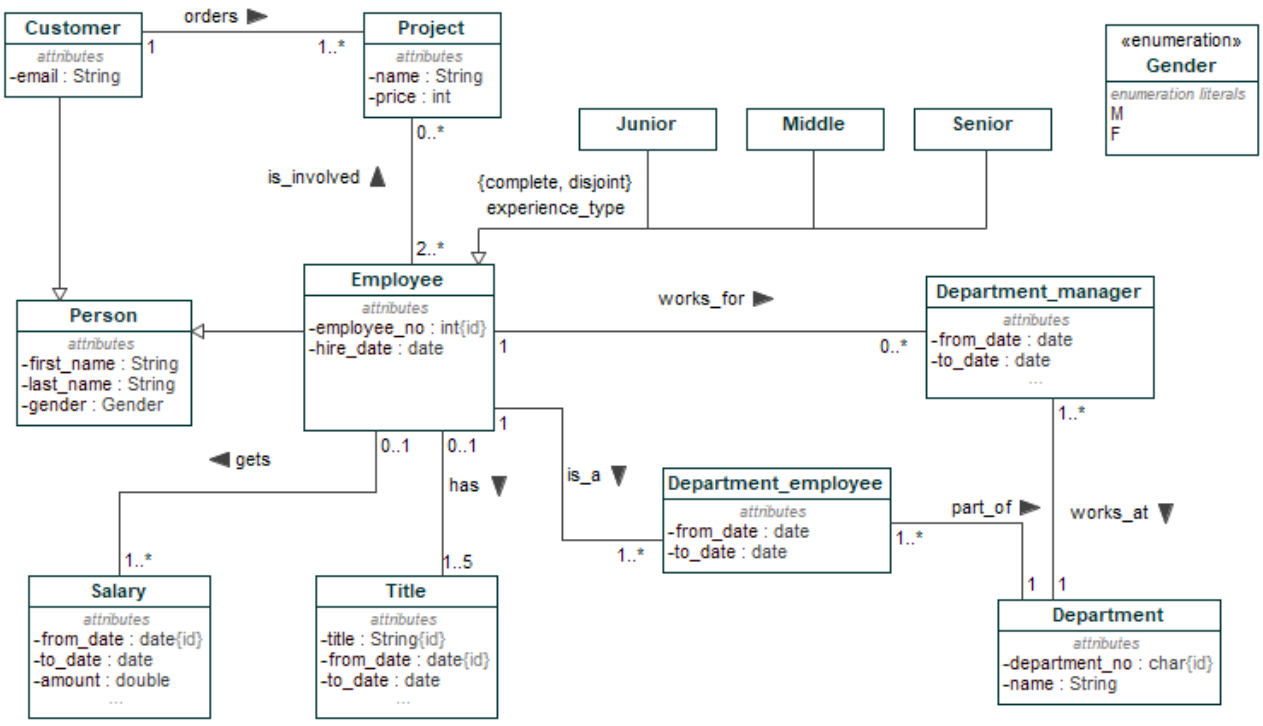
7. Transformuojamos veiklos taisyklės kartu su žodynu į duomenų modelių kūrimo scenarijų.
8. Sukuriami duomenų modeliai, kurie tiktų aprašytai dalykinei sričiai.
9. Atliekamas gautų sukurtų objektų įvertinimas ir lyginimas su sukurtais duomenų modeliais. Randami šie parametrai:
    a. parametras, kiek lentelių atitinka lyginant su pagal dalykinę sritį, rankiniu būdu sukurtais duomenų modeliais;
    b. parametras, rodantis atitikusių sukurtų ryšių kiekį;
    c. parametras, rodantis duomenų tipų atitikimą;
    d. parametras, rodantis sukurtų įvesties apribojimų atitikimą;
    e. parametras, sukurtų ryšių tarp lentelių apribojimų atitikimą.
10. Atliekamas gautų rezultatų vertinimas
11. Pateikiamos eksperimentinės dalies išvados, pritaikyto sprendimo rekomendacijos.

5.2. Eksperimento rezultatai

Eksperimentui atlikti stengiamasi dalykines sritis praplėsti įvairiomis charakteristikomis tam, kad aprėpti didesnę dalį sistemos funkcionalumo ir įvertinti jos veiklą.

Darbuotojų informacinės sistemos kūrimo rezultatai

Informacinė sistema skirta saugoti duomenis apie darbuotoją. Saugomi asmens duomenys, alga, pareigos, vadovai, skyrius, kuriame jis ar ji dirba. Norint aiškiau suvokti dalykinę sritį, nubraižyta esybių ir ryšių diagrama. Nurodyti ryšių pavadinimai, kardinalumas, klasių hierarchija, taip pat išskirta darbuotojo patirtis, kuri skirstoma į jaunesnysis, vidutinis ir vyresnysis darbuotojas. ER diagrama pateikta 5.1 pav.



5.1 pav. Darbuotojų informacinės sistemos ER diagrama

Darbuotojo duomenys užrašyti struktūrizuota natūralia kalba, naudojant *SBVR* standartą. Esybės bei jų atributai išreikšti daiktavardiniais konceptais, duomenų tipai jų atributais. Žodyno fragmentai pateikti 5.1 lentelėje.

5.1 lentelė. Darbuotojų IS veiklos žodyno fragmentai

Bendri konceptai
person employee General_concept: person
Veiksmažodiniai konceptai
employee *has* department_employee Synonymous_form: department_employee *part_of* employee
Savybių priskyrimas
hire_date General_concept: datetime employee *has* hire_date Concept_type: property_association
Individualus konceptas
Jonas General_concept: person First_department Concept_type: individual_concept General_concept: name

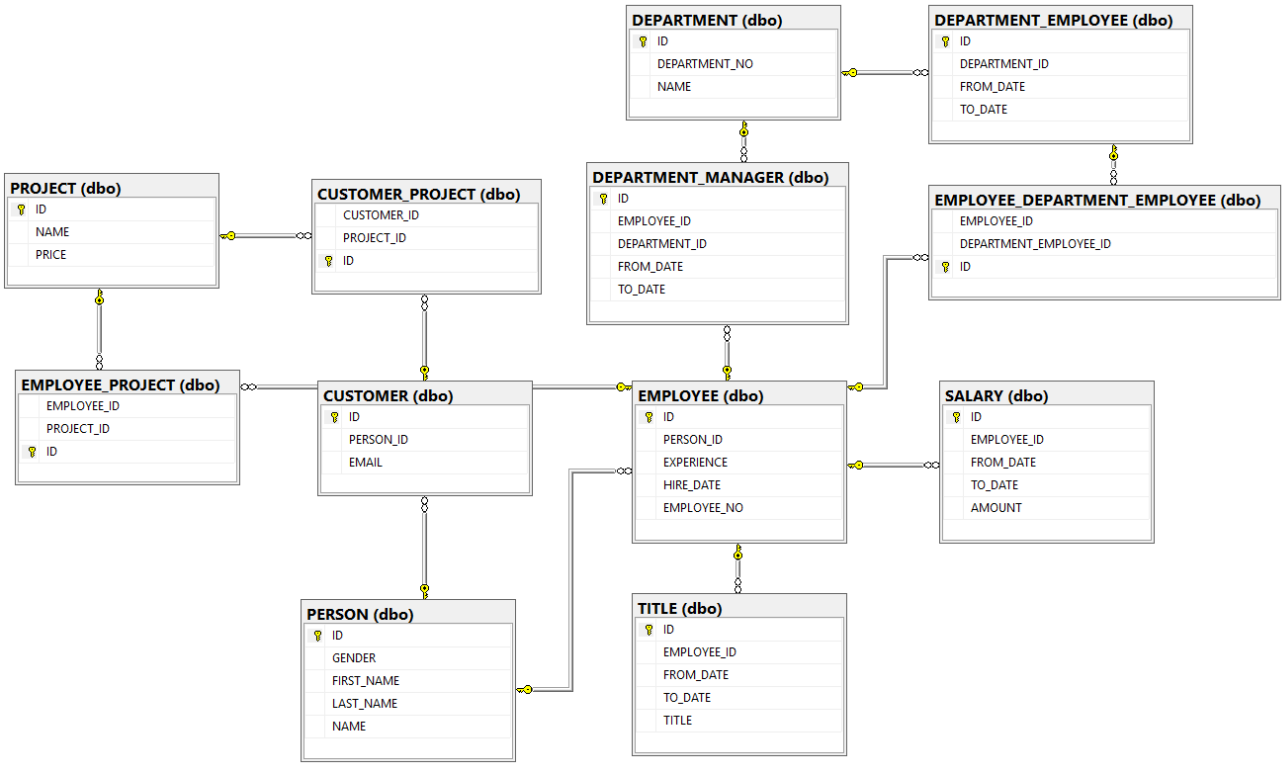
Segmentacija ir kategorizacija
experience Concept_type: categorization_type Necessity: *is_for* general_concept employee Experience_type Necessity: segmentation *for* general_concept employee that *subdivides* employee by experience junior General_concept: employee Necessity: *is_included_in* Experience_type middle General_concept: employee Necessity: *is_included_in* Experience_type senior General_concept: employee Necessity: *is_included_in* Experience_type
Apibūdinimai
employee General_concept: person Definition: "Information_to_store_about_employee"

Veiklos taisyklių fragmentai, skirti privalomų laukų nustatymui bei ryšių kardinalumui nustatyti pateikti 5.2 lentelėje.

5.2 lentelė. Darbuotojų IS veiklos taisyklių fragmentai

Privalomi atributai
It is necessary that salary *has* amount and salary *has* from_date and salary *has* to_date.
Kardinalumo apribojimai
It is necessary that project *has* exactly 1 customer. It is necessary that employee *has* at_least 1 and at_most 5 title. It is necessary that department_employee *part_of* at_most 1 employee.

Pilnas veiklos žodynas ir taisyklių rinkinys pateiktas darbo 8.1 priede. Gautas sprendimas, panaudojant modelių transformaciją aprašytai dalykinei sričiai, pavaizduotas 5.2 pav.



5.2 pav. Darbuotojų informacinės sistemos duomenų modelis

Darbuotojų IS transformacijos palyginimo su pirma pavyzdine duomenų baze rezultatai pateikti 5.3 lentelėje, išskiriant eksperimento plane nurodytus parametrus.

5.3 lentelė. Darbuotojų IS transformacijos palyginimo su pirma pavyzdine duomenų baze rezultatai

Parametras	Kriterijus	Sutampa	Skiriasi	Atitikimas, procentais
Lentelės	Duomenų bazės lentelių skaičius	10	2	83.33
	Pirminiai raktai	8	2	80.00
	Stulpeliai	21	3	87.50
	Sinonimai	0	0	100.00
	Lentelių ar stulpelių aprašymai	0	1	100.00
Ryšiai	Ryšiai tarp lentelių, išoriniai raktai	9	2	81.82
Duomenų tipai	Skaitiniai duomenų tipai	1	1	50.00
	Datos duomenų tipai	9	0	100.00
	Tekstiniai duomenų tipai	7	3	70.00
Įvesties apribojimai	Privalomi laukai	5	4	55.56
	Galimų įvedamų reikšmių apribojimas	2	0	100.00
	Segmentavimo, kategorizavimo, individualių konceptų transformavimo sprendimas	4	1	80.00
Ryšių apribojimai	Kardinalumo minimalios reikšmės apribojimas	7	0	100.00
	Kardinalumo minimalios reikšmės patikrinimo užklausa	3	0	100.00
	Kardinalumo maksimalios reikšmės apribojimas	2	1	66.67

Kitas palyginimas pateiktas 8.6 priede. Bendri rezultatai palyginus transformacijos galutinį rezultatą su paruoštoms siektinoms duomenų bazėmis pateikti 5.4 lentelėje.

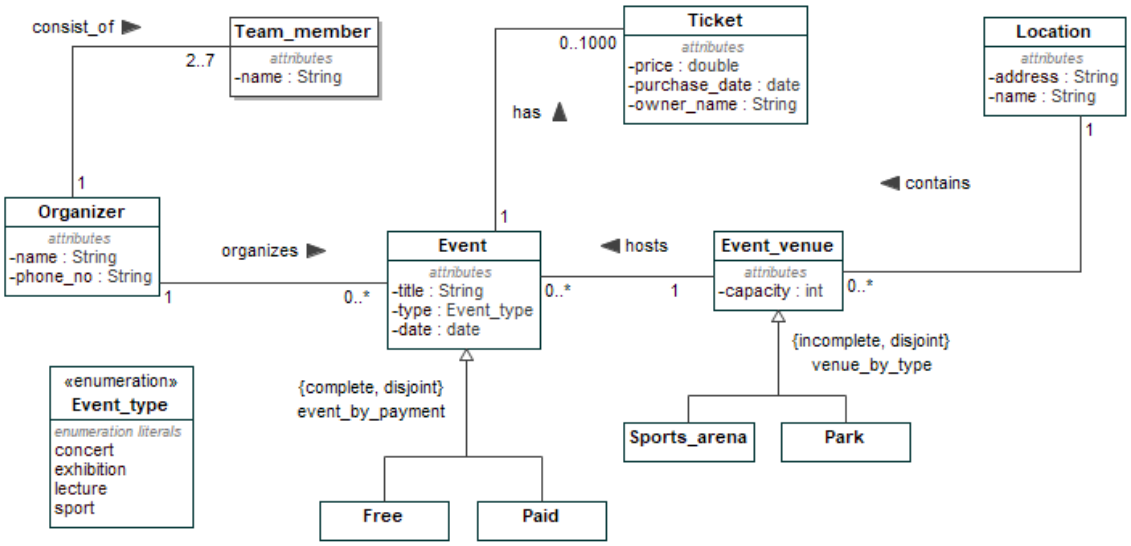
5.4 lentelė Darbuotojų IS transformacijos palyginimas

	Atitikimas su pirma DB			Atitikimas su antra DB		
	Atitikimas, procentais	Svoris	Atitikimas įvertinus svorį, procentais	Atitikimas, procentais	Svoris	Atitikimas įvertinus svorį, procentais
Lentelių kiekis	83.33	0.40	33.33	76.92	0.40	30.77
Lentelių atributai	91.88	0.10	9.19	95.83	0.10	9.58
Ryšiai tarp lentelių	81.82	0.15	12.27	75.00	0.15	11.25
Duomenų tipai	73.33	0.05	3.67	76.67	0.05	3.83
Įvesties apribojimai	78.52	0.10	7.85	68.15	0.10	6.81
Kardinalumų apribojimai, tikrinimai	88.89	0.20	17.78	88.89	0.20	17.78
Viso			**84.09**			**80.03**

Atlikus darbuotojų informacinės sistemos duomenų modelio kūrimo scenarijaus generavimą pastebėta, kad sukurta per daug lentelių dėl ryšio tarp daiktavardinių konceptų nurodymo abiem kryptimis, t.y. metodas suprato taip kaip ryšį daug-daug, kadangi taisyklėse aprašytas apribojimas toms pačiom esybėm du kartus. Pastebėta, kad ne visi tuščios reikšmės apribojimai transformuoti, kadangi modelis nepalaiko veiklos taisyklės parašytos naudojant konjunkciją arba disjunkciją.

Renginio bilietų informacinės sistemos kūrimo rezultatai

Transformacijos procesui analizuoti sukurtas bilietų informacinės sistemos esybių ryšių modelis, kuriame aprašytas renginys, jo planuotojas, bilietai bei renginio vieta. Renginys gali būti arba mokamas arba nemokamas. Šios sistemos ER diagrama pateikta 5.3 pav.



5.3 pav. Renginio bilietų informacinės sistemos ER diagrama

Sistemos žodyno fragmentai pateikti 5.5 lentelėje.

5.5 lentelė. Renginio bilietų IS veiklos žodyno fragmentai

Bendri konceptai
organizer event
Veiksmažodiniai konceptai
event *has* ticket organizer *organizes* event event_venue *hosts* event
Savybių priskyrimas
location address location *has* name Concept_type: property_association location *has* address Concept_type: property_association
Individualus konceptas
Concert Concept_type: individual_concept General_concept: event_type Exhibition Concept_type: individual_concept General_concept: event_type Tiketa General_concept: seller_name
Segmentacija ir kategorizacija
Event_venue_by_type Necessity: categorization_scheme *for* general_concept event_venue that *subdivides* event_venue by event_venue_type sports_arena General_concept: event Necessity: *is_included_in* Event_venue_by_type park General_concept: event Necessity: *is_included_in* Event_venue_by_type payment Concept_type: categorization_type Necessity: *is_for* general_concept event Event_by_payment Necessity: segmentation *for* general_concept event that *subdivides* event by payment free_event General_concept: event Necessity: *is_included_in* Event_by_payment Definition: event that *has* price equals 0 paid_event General_concept: event Necessity: *is_included_in* Event_by_payment Definition: event that *has* price greater_than 0
Apibūdinimai

```
seller
            Synonym: ticket_seller
            Description: "A place, where you can buy tickets"
```

Veiklos taisyklių fragmentai pateikti 5.6 lentelėje.

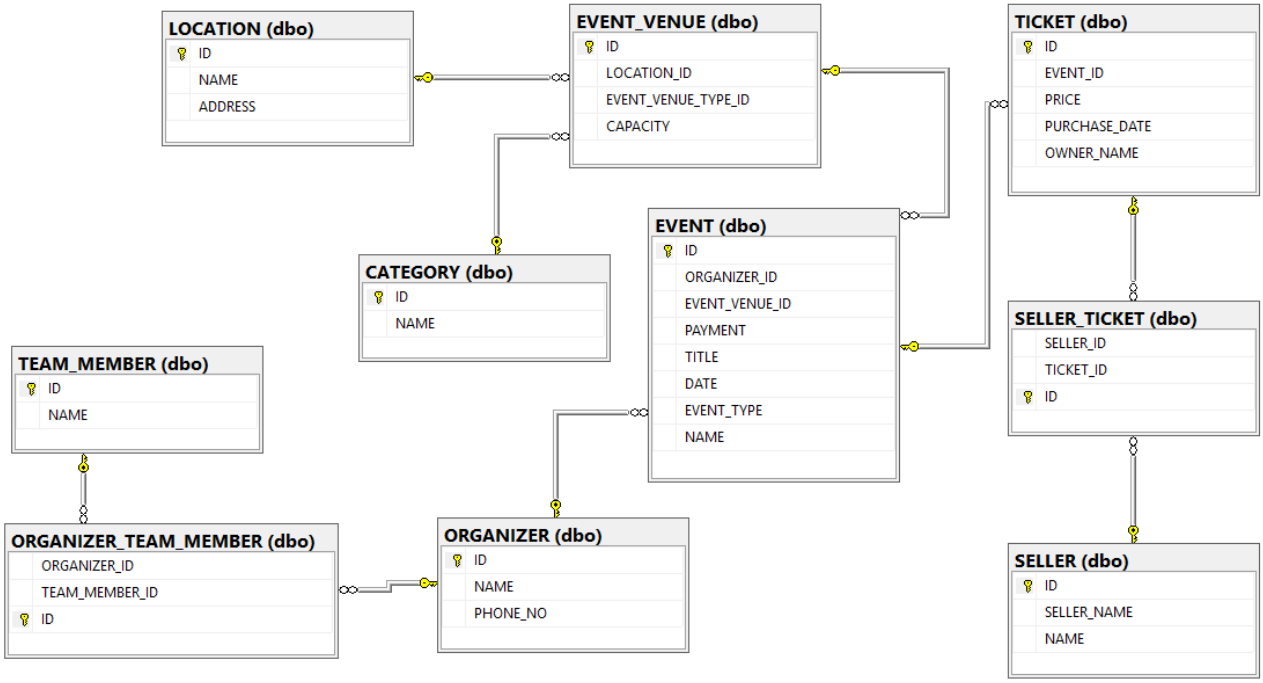
5.6 lentelė Renginio bilietų IS veiklos taisyklių fragmentai

Privalomi atributai
`It is necessary that event has event_type.`
Kardinalumo apribojimai

`It is necessary that team_member is_part_of exactly 1 organizer.`
`It is necessary that seller can_sell at_least 10 ticket.`
`It is necessary that event has at_most 1000 ticket.`
`It is necessary that organizer consist_of at_least 2 and at_most 7 team_member.`

Atributų apribojimai
`It is necessary that capacity of event_venue greater_than 123.`

Šios dalykinės srities veiklos žodynas ir taisyklių rinkinys pateiktas darbo 8.2 priede. Gautas sprendimas renginio bilietų dalykinei sričiai pavaizduotas 5.4 pav.



5.4 pav. Renginio bilietų informacinės sistemos siektinas duomenų modelis

Sukurtos duomenų bazės, nenaudojant transformatoriaus, taip pat jų tikslus palyginimas su iš modelių transformacijos gauta duomenų baze aprašyti 8.6 priede. Galutiniai rezultatai pateikti 5.7 lentelėje.

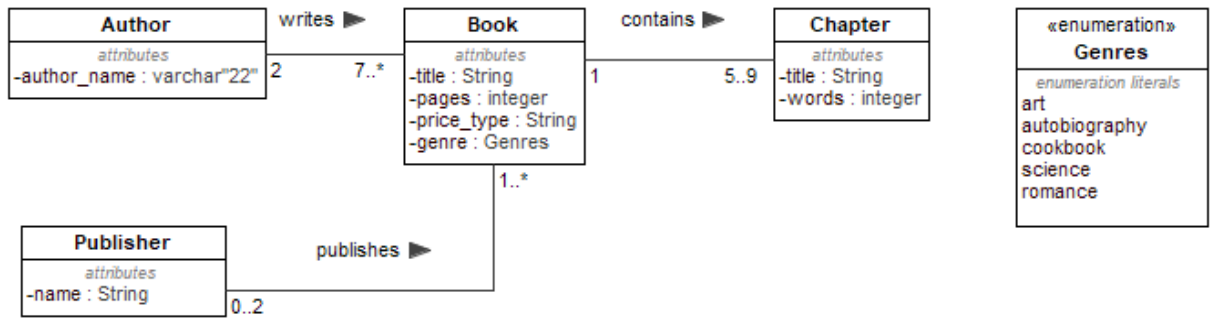
5.7 lentelė. Renginio bilietų IS transformacijos palyginimas

	Atitikimas su pirma DB			Atitikimas su antra DB		
	Atitikimas, procentais	Svoris	Atitikimas įvertinus svorį, procentais	Atitikimas, procentais	Svoris	Atitikimas įvertinus svorį, procentais
Lentelių kiekis	90.00	0.40	36.00	72.73	0.40	29.09
Lentelių atributai	97.06	0.10	9.71	85.03	0.10	8.50
Ryšiai tarp lentelių	77.78	0.15	11.67	60.00	0.15	9.00
Duomenų tipai	59.26	0.05	2.96	72.22	0.05	3.61
Įvesties apribojimai	83.33	0.10	8.33	92.59	0.10	9.26
Kardinalumų apribojimai, tikrinimai	93.33	0.20	18.67	93.33	0.20	18.67
Viso			**87.34**			**78.13**

Atlikus renginio bilietų informacinės sistemos duomenų modelio kūrimo scenarijaus generavimą pastebėta, kad dėl paprastesnės dalykinės srities logikos pirmu atveju pasiektas mažesnis klaidų kiekis. Lyginant su antru duomenų modeliu pasiektas 78% atitikimas, kadangi renginio ir renginio vietos tipams saugoti pasirinktas atskirtų lentelių variantas, o transformavimo metodas kuria vieną bendrą kategorijų lentelę. Lentelių kiekio svoris skaičiuojant atitikimą yra pakankamai didelis, todėl bendras atitikimo procentas žymiai krenta.

Knygų informacinės sistemos kūrimo rezultatai

Aprašyta knygų dalykinė sritis, kurioje fiksuojama informacija apie knygą, jos leidėjus ir autorius. Šios dalykinės srities ER diagrama pateikta 5.5 pav.



5.5 pav. Knygų informacinės sistemos ER diagrama

Duomenys apie knygas užrašyti struktūrizuota natūralia kalba, naudojant *SBVR* standartą. Žodyno fragmentai pateikti 5.8 lentelėje.

5.8 lentelė. Knygų IS veiklos žodyno fragmentai

Bendri konceptai
author author_of_a_book See: author

book
Veiksmažodiniai konceptai
book *belongs_to* author Synonymous_form: author *writes* book book *has* publisher Synonymous_form: publisher *publishes* book
Savybių priskyrimas
title General_concept: text chapter *has* title Concept_type: property_association book *has* title Concept_type: property_association pages General_concept: integer Description: "Number of pages in a book" book *has* pages Concept_type: property_association
Individualus konceptas
The_Hunger_Games Concept_type: individual_concept General_concept: book Ayn_Rand General_concept: author_name
Segmentacija ir kategorizacija
Genres Necessity: categorization_scheme *for* general_concept book that *subdivides* book by genre art General_concept: book Necessity: *is_included_in* Genres price_type Concept_type: categorization_type Necessity: *is_for* general_concept book Book_by_price Necessity: segmentation *for* general_concept book that *subdivides* book by price_type cheap General_concept: book Necessity: *is_included_in* Book_by_price
Apibūdinimai
Ernest_Hemingway General_concept: author_name Description: "Author of: Anna Karenina, Far Away and Long Ago and so on."

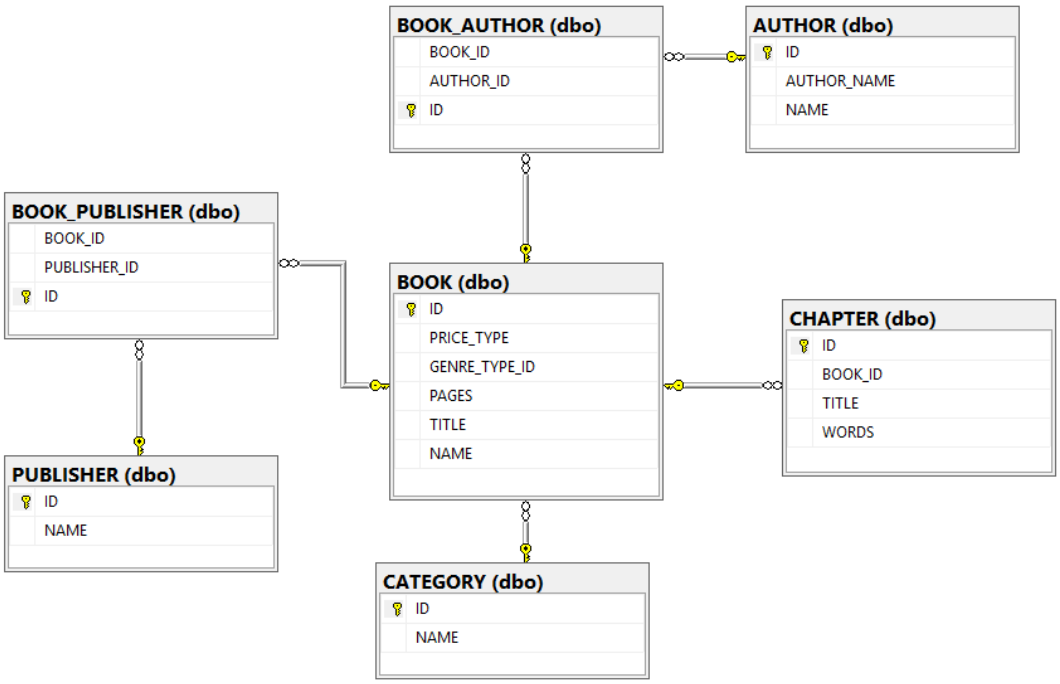
Veiklos taisyklių fragmentai, skirti privalomų laukų nustatymui bei ryšių kardinalumui nustatyti pateikti 5.9 lentelėje.

5.9 lentelė. Knygų IS veiklos taisyklių fragmentai

Privalomi atributai
It is necessary that book *has* title.
Kardinalumo apribojimai

```
It is necessary that author writes at_least 7 book.
It is necessary that book belongs_to exactly 2 author.
It is necessary that book contains at_least 5 and at_most 9 chapter.
```

Pilnas veiklos žodynas ir taisyklių rinkinys pateiktas darbo 8.3 priede. Gautas sprendimas aprašytai dalykinei sričiai pavaizduotas 5.6 pav.



5.6 pav. Knygų informacinės sistemos siektinas duomenų modelis

Sukurtų duomenų modelių knygų dalykinei sričiai scenarijai, bei palyginimo su transformacijos metu gautu duomenų modeliu parametrų rezultatai pateikti 8.6 priede. 5.10 lentelėje pateiktas galutinis palyginimų rezultatas.

5.10 lentelė. Knygų IS transformacijos palyginimas

	Atitikimas su pirma DB			Atitikimas su antra DB		
	Atitikimas, procentais	Svoris	Atitikimas įvertinus svorį, procentais	Atitikimas, procentais	Svoris	Atitikimas įvertinus svorį, procentais
Lentelių kiekis	100.00	0.40	40.00	85.71	0.40	34.29
Lentelių atributai	94.44	0.10	9.44	90.87	0.10	9.09
Ryšiai tarp lentelių	100.00	0.15	15.00	83.33	0.15	12.50
Duomenų tipai	73.81	0.05	3.69	73.81	0.05	3.69
Įvesties apribojimai	50.00	0.10	5.00	50.00	0.10	5.00
Kardinalumų apribojimai, tikrinimai	100.00	0.20	20.00	100.00	0.20	20.00

Viso			**93.13**			**84.56**

Atlikus eksperimentą pastebėta, kad sukurti duomenų modeliai apriboja ar patikrina kardinalumus taip pat, kaip ir transformacijos metu gautas duomenų modelis. Atributai taip pat atpažinti ir teisingai susieti su lentelėmis. Antro duomenų modelio atveju sukurta labiau specializuota žanrams saugoti skirta lentelė.

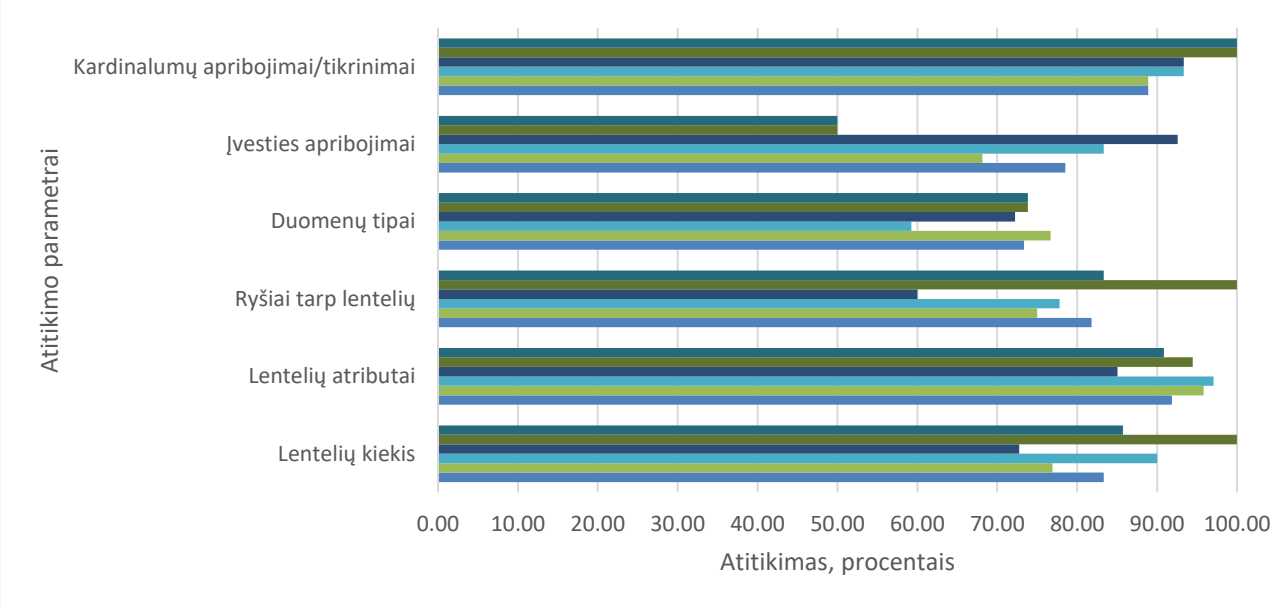
5.3. **Sprendimo veikimo ir savybių analizė, kokybės kriterijų įvertinimas**

Metodo veikimo įvertinimas pasirinktoms dalykinėms sritims

Klaida! Nerastas nuorodos šaltinis. lentelėje, bei 5.7 pav. esančioje diagramoje pateikiami bendri transformacijos metodo sugeneruotų duomenų modelių atitikimai žmogaus sukurtiems modeliams.

5.11 lentelė. Sukurtų duomenų modelių atitikimas sugeneruotam pagal parametrus, procentais

	Darbuotojų DB1	Darbuotojų DB2	Renginių DB1	Renginių DB2	Knygų DB1	Knygų DB2
Lentelių kiekis	83.33	76.92	90.00	72.73	100.00	85.71
Lentelių atributai	91.88	95.83	97.06	85.03	94.44	90.87
Ryšiai tarp lentelių	81.82	75.00	77.78	60.00	100.00	83.33
Duomenų tipai	73.33	76.67	59.26	72.22	73.81	73.81
Įvesties apribojimai	78.52	68.15	83.33	92.59	50.00	50.00
Kardinalumų apribojimai, tikrinimai	88.89	88.89	93.33	93.33	100.00	100.00



5.7 pav. Generuotų duomenų modelių atitikimas sukurtiems duomenų modeliams pagal parametrus, procentais

Iš gautų rezultatų galime daryti šias prielaidas:

- lentelių kiekis – matome, kad pagrindinės sukurtos lentelės atitinka kituose duomenų modeliuose esančias lenteles, kadangi jos dažnu atveju atitinka veiklos žodyne išskirtus daiktavardinius konceptus. Priklausomai nuo duomenų bazės normalizavimo rezultatų duomenų bazės lentelės gali būti sujungtos ar atskirtos tam, kad užtikrinti vieno fakto būvimą vienoje vietoje ir ryšiai tarp faktų būtų tvarkingi. Didžiausi skirtumai rasti tarpinių lentelių kūrimo sprendime, kai vienas daiktavardinis konceptas siejasi keliais kito daiktavardinio koncepto objektais ir atvirkščiai. Taip pat atsiranda skirtumų kai norima sukurti lentelės atributui reikšmių lentelę, saugant esamoje išorinį raktą. Lentelės pavadinimui konkrečių taisyklių nėra, todėl rezultatas priklauso nuo projektuotojo pasirinkimo;
- lentelių atributai – pagal veiklos žodyną rasti konceptai, kurie susieti su lentelėmis transformuotais konceptais ryšiu *property_association.* Visose duomenų modeliuose atitikimas yra aukštas (daugiau nei 85%). Skirtumai atsiranda dėl atributų pasirinkimo saugoti atskiroje lentelėje (išorinis laukas gali pakeisti tekstinį lauką);
- ryšiai tarp lentelių – šis parametras artimai susijęs su pirmuoju lentelių kiekio parametru. Jeigu pasirenkama lenteles skaidyti smulkiau, atsiranda daugiau ryšių, jeigu jos grupuojamos, automatiškai sumažėja sukurtų ryšių kiekis;
- duomenų tipai – vienas iš mažiausiai atitikimo procentų surinkęs parametras. Veiklos žodyne koncepto tipų aibė susideda iš kelių elementų, t.y. tekstas, skaičius, data ir realusis skaičius, o duomenų bazėje šių tipų yra daug daugiau. Taip pat kadangi saugomos reikšmės ilgis negali būti nurodomas veiklos žodyne, metodas parenka universaliausius variantus. Projektuotojas gali daug tiksliau nuspėti būsimų duomenų tipą ar ilgį, dėl šios priežasties generuojant duomenų modelį būtų naudinga peržvelgti ir papildomai modifikuoti gautas reikšmes;
- įvesties apribojimai – tai veiklos taisyklėse atributam nurodyti reikšmių intervalai arba aibės. Taip pat segmentavimo ar kategorizavimo būdu aprašyti koncepto reikšmių variantai, kurie gali būti transformuoti tiek į įvedamo stulpelio tekstinės reikšmės apribojimą, tiek į kategorijos lentelės reikšmių įvedimą. Dėl keleto transformavimo variantų bendras šio parametro atitikimo procentas krenta;
- kardinalumų apribojimai, tikrinimai – veiklos taisyklių transformavimo rezultatai. Atitikimo procentas pakankamai didelis (visose daugiau nei 88%). Pagrindiniai trūkumai tai konjukcijos ir disjunkcijos nutransformavimas, bei perteklinis trigerių kūrimas (jeigu taisyklė parašyta ryšiui „ne daugiau kaip vienas elementas“).

Bendras vertinimas

Bendrai vertinant duomenų modelių atitikimų procentus, atsižvelgus į kiekvieno palyginimo parametro atitikimo vertinimą, galime daryti išvadą, kad sistema ganėtinai tiksliai atlieka veiklos žodynų ir taisyklių transformavimą į duomenų modelius. Bendras atitikimo procentas įvertinus svorį siekia nuo 78% iki 93%. Rezultatai pateikti 5.12 lentelėje.

5.12 lentelė. Duomenų modelių atitikimas įvertinus svorį, procentais

Duomenų modelis	Atitikimas įvertinus svorį
Darbuotojų DB1	84.09%

Darbuotojų DB2	80.03%
Renginių DB1	87.34%
Renginių DB2	78.13%
Knygų DB1	93.13%
Knygų DB2	84.56%

Reiktų pabrėžti, kad tiek sukurti žmogaus duomenų modeliai, tiek sugeneruoti sukurto metodo modeliai yra korektiški ir tinkami naudoti kuriant informacinę sistemą.

5.4. **Sprendimo taikymo rekomendacijos**

1. Tolimesniam sistemos vystymui rekomenduojama papildyti *SBVR* metamodelį duomenų tipais tam, kad būtų galima tiksliau apibrėžti konceptą. Taip pat perkelti koncepto apibrėžimo reikšmę į *XMI* schemą.
2. Papildyti veiksmažodinio koncepto transformavimą į reliacinių lentelių ryšius pritaikant koncepto pavadinimą ryšio pavadinimui. Taip pat papildyti tarpinės lentelės kūrimo sąlygą taip, kad kuriama ji būtų tik kai veiklos taisyklėse nurodyti apribojimai reikalauja daugiau nei vieno įrašo.
3. Atlikus transformaciją projektuotojui reiktų papildomai peržiūrėti laukų pavadinimų bei tipų reikšmes, kadangi kiekvienai dalykinei sričiai atsiranda tik jai būdingų reikalavimų.

6. REZULTATŲ APIBENDRINIMAS IR IŠVADOS

1. Specifikuojant kuriamos sistemos reikalavimus gali atsirasti komunikacijos problemų tarp sistemos kūrėjų ir užsakovų, kurie gali nevienodai suprasti dalykinės srities modelius ir veiklos taisykles. Dėl šios priežasties sukurta informacinė sistema nevisiškai atitiks užsakovo poreikius. Vienas iš būdų siekiant išvengti tokių problemų - reikalavimus užrašyti struktūrizuota natūralia kalba, naudojant veiklos žodyną ir taisykles.
2. Išnagrinėtas *SBVR* standartas, kuris leidžia aprašyti veiklos žodyną ir taisykles visiems suprantamu būdu. *SBVR* standartu užrašytas tekstas gali būti interpretuojamas kompiuteriu ir transformuojamos į programinį kodą.
3. Atlikus įrankių analizę matoma, jog nemažai nagrinėta kaip patikrinti duomenų nuoseklumą, taisyklių laikymąsi ir pan., tačiau šie įrankiai skirti patikrinti, ar duomenų bazėje saugomi duomenys tenkina taisykles, bet netikrina įvedamų duomenų teisingumo.
4. Šiame darbe sukurtas metodas leidžia automatiškai transformuoti veiklos žodyną ir taisykles į duomenų modelius. Metodas realizuotas modeliais grindžiamos inžinerijos pagrindu, naudojant *SBVR* ir šiame darbe aprašytą *DDL* metamodelį.
5. Išanalizavus *SBVR* ir *DDL* metamodelius sukurtos taisyklės, kurios transformuoja veiklos žodyno konceptus ir struktūrines veiklos taisykles į duomenų bazės schemą. Atlikus duomenų modelių kūrimo analizę nustatyta, kad ne visos taisyklės gali būti realizuotos duomenų bazės schemos apribojimais. Likę apribojimai transformuoti į užklausas, kurios patikrina duomenų atitikimą veiklos taisyklėms.
6. Transformavimo taisyklių realizacijai panaudota modelių transformavimo kalba ATL. *DDL* kodo generavimui panaudotas modelių transformavimo į tekstą *Acceleo* įrankis. Sprendimas integruotas į *SBVR* veiklos žodyno ir taisyklių redaktorių.
7. Eksperimentu metu buvo siekiama išsiaiškinti ar sukurtas metodas tinkamas duomenų modelių kūrimui. Ištirtos trys skirtingos dalykinės sritys ir skaičiuotas metodo rezultato atitikimas žmogaus sukurtiems duomenų modeliams. Kadangi atitikimo procentas svyruoja nuo 78 iki 93 procentų, galima teigti, kad sukurtas metodas praplečia modelių kūrimo galimybes.
8. Plėtojant sprendimą ateityje rekomenduojama praplėsti *SBVR* žodyną duomenų tipais, bei įtraukti konceptų aprašymus į *XMI* schemos generavimą. Šiame darbe sukurtas transformacijos metodas galėtų palaikyti didesnį kiekį duomenų bazių, bei turėti automatinį sujungimą su pasirinkta duomenų bazių valdymo sistema.

7. LITERATŪRA

[1] Ž VAIRA. *Programinės Įrangos Kūrimo Technologijos* . VšĮ Socialinių mokslų kolegija. , 2013 Prieiga per: http://www.esparama.lt/es_parama_pletra/failai/ESFproduktai/2013_Programines_irangos_kurimo_technologijos.pdf.pdf.

[2] MCGRAW, G. Making Essential Software Work. *Citgal, Inc.Mar*, 2003.

[3] *Database Design and System Design.* Prieiga per: http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/0303011/1146161367915/1146161680673/1146161836562/1146161929756.html.

[4] R. G. Ross. "The Business Rules Manifesto" , 2003.

[5] Donald R. Chapin. *SBVR: What is Now Possible and Why?* Prieiga per: https://www.brcommunity.com/articles.php?id=b407.

[6] KARPOVIC, J. and NEMURAITE, L. Transforming SBVR Business Semantics into Web Ontology Language OWL2: Main Concepts. *Information Technologies*, 2011. pp. 27-29.

[7] RAVENTÓS PAGÈS, R. SBVR Meanings and Representations Metaschemas, 2008.

[8] *Daiktavardiniai Ir Veiksmažodiniai SBVR Konceptai.* Prieiga per: https://www.researchgate.net/figure/Noun-and-verb-concepts-in-SBVR_fig1_261436985.

[9] *Semantics of Business Vocabulary and Rules Specification 1.0.* Prieiga per: http://www.omg.org/spec/SBVR/1.0/About-SBVR/.

[10] *Duomenų Bazė.* Prieiga per: https://lt.wikipedia.org/wiki/Duomen%C5%B3_baz%C4%97.

[11] *Data Definition Language.* , 2019Prieiga per: https://en.wikipedia.org/wiki/Data_definition_language.

[12] *Data Manipulation Language.* , 2019Prieiga per: https://en.wikipedia.org/wiki/Data_manipulation_language.

[13] *Stack Overflow Survey.* Prieiga per: https://insights.stackoverflow.com/survey/2019.

[14] *Sql.* Prieiga per: https://en.wikipedia.org/wiki/SQL.

[15] *SQL Constraints.* Prieiga per: https://www.w3schools.com/sql/sql_constraints.asp.

[16] *ATLAS Transformation Language.* Prieiga per: https://en.wikipedia.org/wiki/ATLAS_Transformation_Language.

[17] *ATL/Concepts.* Prieiga per: http://wiki.eclipse.org/ATL/Concepts.

[18] MISO - Uniandes. *ATL: ATLAS Transformation Language.*

[19] Wikipedia. *Qvt.* Prieiga per: https://en.wikipedia.org/wiki/QVT.

[20] VAN AMSTEL, M., BOSEMS, S., KURTEV, I. and PIRES, L.F. *Performance in Model Transformations: Experiments with ATL and QVT.* Springer, 2011.

[21] NATALI, V. and LIEM, I. *Automated Data Consistency Checking using SBVR: Case Study: Academic Data in a University.* IEEE, 2015.

[22] MARINOS, A., MOSCHOYIANNIS, S. and KRAUSE, P.J. An SBVR to SQL Compiler. *RuleML Challenge*, 2010, vol. 649.

[23] NEMURAITE, L., et al. VETIS Tool for Editing and Transforming SBVR Business Vocabularies and Business Rules into UML&OCL Models. *Information Technologies*, 2010. pp. 21-23.

[24] *Oracle DDL Generation and Customization.* Prieiga per: https://docs.nomagic.com/display/MD182/Oracle+DDL+generation+and+customization.
[25] ŠUKYS, A., ABLONSKIS, L., NEMURAITĖ, L. and PARADAUSKAS, B. A Grammar for Advanced SBVR Editor. *Information Technology and Control*, 2016, vol. 45, no. 1. pp. 27-41.

8. PRIEDAI

8.1. **priedas.** Darbuotojų informacinės sistemos veiklos žodynas ir taisyklės

```
person
first_name
last_name
person has first_name
            Concept_type: property_association
person has last_name
            Concept_type: property_association
gender
            Concept_type: categorization_type
            Necessity: is_for general_concept person
Person_by_gender
            Necessity: segmentation for general_concept person that subdivides person by
gender
male
            General_concept: person
            Necessity: is_included_in Person_by_gender
            Description: "Male"
female
            General_concept: person
            Necessity: is_included_in Person_by_gender
            Description: "Female"

Jonas
            General_concept: person

employee
            General_concept: person
            Definition: "Information_to_store_about_employee"
hire_date
            General_concept: datetime
employee has hire_date
            Concept_type: property_association
employee_no
employee has employee_no
            Concept_type: property_association
experience
            Concept_type: categorization_type
            Necessity: is_for general_concept employee
Experience_type
            Necessity: segmentation for general_concept employee that subdivides
employee by experience
junior
            General_concept: employee
            Necessity: is_included_in Experience_type
middle
            General_concept: employee
            Necessity: is_included_in Experience_type
senior
            General_concept: employee
            Necessity: is_included_in Experience_type

customer
```

```
            General_concept: person
email
customer has email
            Concept_type: property_association

project
name
project has name
            Concept_type: property_association
price
            General_concept: number
project has price
            Concept_type: property_association

customer orders project
            Synonymous_form: project has customer
employee is_involved_in project
            Synonymous_form: project has employee

salary
employee gets salary
amount
            General_concept: number
from_date
            General_concept: datetime
to_date
            General_concept: datetime
salary has from_date
            Concept_type: property_association
salary has to_date
            Concept_type: property_association
salary has amount
            Concept_type: property_association


title
employee has title
title has from_date
            Concept_type: property_association
title has to_date
            Concept_type: property_association
title has title
            Concept_type: property_association


department_manager
department_manager has from_date
            Concept_type: property_association
department_manager has to_date
            Concept_type: property_association
employee has department_manager


department
department_no
department has department_no
            Concept_type: property_association
```

```
department has name
            Concept_type: property_association
department has department_manager
First_department
            Concept_type: individual_concept
            General_concept: department
Second_department
            Concept_type: individual_concept
            General_concept: department


department_employee
department_employee has from_date
            Concept_type: property_association
department_employee has to_date
            Concept_type: property_association
employee has department_employee
            Synonymous_form: department_employee part_of employee

department has department_employee

It is necessary that salary has amount and salary has from_date and salary has to_date.
It is necessary that title has from_date and title has to_date.
It is necessary that customer orders at_least 1 project.
It is necessary that project has exactly 1 customer.
It is necessary that project has at_least 2 employee.
It is necessary that employee gets at_least 1 salary.
It is necessary that salary has at_most 1 employee.
It is necessary that employee has at_least 1 and at_most 5 title.
It is necessary that department_employee part_of at_most 1 employee.
```

8.2. **priedas.** Renginio bilietų informacinės sistemos veiklos žodynas ir taisyklės

```
organizer
name
phone_no
            General_concept: text
organizer has name
            Concept_type: property_association
organizer has phone_no
            Concept_type: property_association
event
title
event has title
            Concept_type: property_association
date
            General_concept: datetime
event has date
            Concept_type: property_association
event_type
            Concept_type: role
event has event_type
            Concept_type: property_association
ticket
price
            General_concept: number
ticket has price
```

```
            Concept_type: property_association
purchase_date
            General_concept: datetime
ticket has purchase_date
            Concept_type: property_association
owner_name
ticket has owner_name
            Concept_type: property_association
event_venue
capacity
            General_concept: integer
event_venue has capacity
            Concept_type: property_association
            Synonymous_form: capacity of event_venue
location
address
location has name
            Concept_type: property_association
location has address
            Concept_type: property_association

event has ticket
organizer organizes event
event_venue hosts event
location contains event_venue

payment
            Concept_type: categorization_type
            Necessity: is_for general_concept event
Event_by_payment
            Necessity: segmentation for general_concept event that subdivides event by
payment
free_event
            General_concept: event
            Necessity: is_included_in Event_by_payment
            Definition: event that has price equals 0
paid_event
            General_concept: event
            Necessity: is_included_in Event_by_payment
            Definition: event that has price greater_than 0

event_venue_type
            Concept_type: categorization_type
            Necessity: is_for general_concept event_venue
Event_venue_by_type
            Necessity: categorization_scheme for general_concept event_venue that
subdivides event_venue by event_venue_type
sports_arena
            General_concept: event
            Necessity: is_included_in Event_venue_by_type
park
            General_concept: event
            Necessity: is_included_in Event_venue_by_type

Concert
            Concept_type: individual_concept
            General_concept: event
```

```
Exhibition
            Concept_type: individual_concept
            General_concept: event
Lecture
            Concept_type: individual_concept
            General_concept: event
Sport
            Concept_type: individual_concept
            General_concept: event

team_member
team_member has name
            Concept_type: property_association
organizer consist_of team_member
            Synonymous_form: team_member is_part_of organizer

seller
            Synonym: ticket_seller
            Description: "A place, where you can buy tickets"
seller_name
seller has seller_name
            Concept_type: property_association
seller can_sell ticket
            Synonymous_form: ticket is_sold_by seller
Tiketa

            General_concept: seller

It is necessary that capacity of event_venue greater_than 123
It is necessary that event has at_most 1000 ticket.
It is necessary that organizer consist_of at_least 2 and at_most 7 team_member.
It is necessary that event has event_type.
It is necessary that team_member is_part_of exactly 1 organizer.
It is necessary that ticket is_sold_by at_least 1 seller.
It is necessary that seller can_sell at_least 10 ticket.
```

8.3. **priedas**. Knygų sistemos veiklos žodynas ir taisyklės

```
author
author_of_a_book
            See: author
author_name
author has author_name
  Concept_type: property_association
book
author writes book
book belongs_to author
            Synonymous_form: author writes book
pages
            General_concept: integer
            Description: "Number of pages in a book"
book has pages
  Concept_type: property_association
price_type
            Concept_type: categorization_type
            Necessity: is_for general_concept book
Book_by_price
```

```
            Necessity: segmentation for general_concept book that subdivides book by
price_type
cheap
            General_concept: book
            Necessity: is_included_in Book_by_price
expensive
            General_concept: book
            Necessity: is_included_in Book_by_price

chapter
book contains chapter
Ayn_Rand
            General_concept: author
Ernest_Hemingway
            General_concept: author
            Description: "Author of: The Old Man and the Sea, A Farewell to Arms and so
on."
title
            General_concept: text
chapter has title
            Concept_type: property_association
book has title
            Concept_type: property_association
words
            General_concept: integer
            Concept_type: role
chapter has words
       Concept_type: property_association
words of chapter
       See: chapter has words
genre
            Concept_type: categorization_type
            Necessity: is_for general_concept book
Genres
            Necessity: categorization_scheme for general_concept book that subdivides
book by genre
art
            General_concept: book
            Necessity: is_included_in Genres
autobiography
            General_concept: book
            Necessity: is_included_in Genres
cookbook
            General_concept: book
            Necessity: is_included_in Genres
science
            General_concept: book
            Necessity: is_included_in Genres
romance
            General_concept: book
            Necessity: is_included_in Genres
publisher
book has publisher
            Synonymous_form: publisher publishes book
name
publisher has name
            Concept_type: property_association
```

```
The_Hunger_Games
            Concept_type: individual_concept

            General_concept: book

It is necessary that author writes at_least 7 book.
It is necessary that book belongs_to exactly 2 author.
It is necessary that book contains at_least 5 and at_most 9 chapter.
It is necessary that book has title.
It is necessary that book has at_most 2 publisher.
It is necessary that publisher has at_least 1 book.
It is necessary that book contains at_least 5 and at_most 9 chapter.
```

8.4. **priedas** Modelių transformacijos *ATL* kodas

```
-- @path SBVR=/lt.ktu.isd.semantika.sbvr2ddl.atl/metamodels/SBVR.ecore
-- @path SQL=/lt.ktu.isd.semantika.sbvr2ddl.atl/metamodels/SQL.ecore
---- @atlcompiler emftvm

module sbvr2sql;
create OUT: SQL from IN: SBVR;

-- INSTANCE SELECTORS
helper def: termAllInstances: Set(SBVR!Term) = SBVR!Term.allInstances();
helper def: nameAllInstances: Set(SBVR!Name) = SBVR!Name.allInstances();
helper def: script: Set(SQL!SQLScript) = OclUndefined;
helper def: tableAllInstances: Sequence(SQL!Table) = Sequence{};
helper def: columnAllInstances: Sequence(SQL!Column) = Sequence{};
helper def: referentialColumnConstraintAllInstances:
Sequence(SQL!ReferentialColumnConstraint) = Sequence{};
helper def: checkConstraintAllInstances: Sequence(SQL!CheckColumnConstraint) =
Sequence{};

helper def: schemaName: String = 'dbo';
helper def: categoryTableName: String = 'CATEGORY';

helper def: moreThen: String = 'moreThen';
helper def: lessThen: String = 'lessThen';
helper def: equals: String = 'equals';
helper def: number: Integer = 1;


-- CONCEPT NAME EXTRACTORS
helper def: ObjectTypeText(o: SBVR!ObjectType): SBVR!Text =
   thisModule.termAllInstances -> select(t | t.meaning = o) -> asSequence() -> first().
        expression;

helper def: RoleTypeText(o: SBVR!Role): SBVR!Text =
   thisModule.termAllInstances -> select(t | t.meaning = o) -> asSequence() -> first().
        expression;

helper context SBVR!ObjectType def: getCategorizationFactTypeSet():
      Set(SBVR!CategorizationFactType) =
   SBVR!CategorizationFactType -> allInstances() -> select(icft | icft.role -> last().
        objectType -> first() = self);

helper def: primitiveTypeMap: Map(String, String) =
```

```
   Map{
      ('number', 'http://www.w3.org/2001/XMLSchema#decimal'),
      ('text', 'http://www.w3.org/2001/XMLSchema#string'),
      ('nonnegative_integer', 'http://www.w3.org/2001/XMLSchema#nonNegativeInteger'),
      ('positive_integer', 'http://www.w3.org/2001/XMLSchema#positiveInteger'),
      ('boolean', 'http://www.w3.org/2001/XMLSchema#boolean'),
      ('datetime', 'http://www.w3.org/2001/XMLSchema#dateTime'),
      ('something', ''),
      ('number1', 'http://www.w3.org/2001/XMLSchema#decimal'),
      ('number2', 'http://www.w3.org/2001/XMLSchema#decimal')};

helper context SBVR!ObjectType def: isPrimitiveDataType(): Boolean =
   if
thisModule.primitiveTypeMap.get(self.getClassName(self.getObjectTypeTerm())).oclIsUndefi
ned() then
      false
   else
      true
   endif;

helper context SBVR!ObjectType def: getClassName(term: SBVR!Term): String =
   let text: SBVR!Text =
      SBVR!Text.allInstances() -> select(i | i = term.expression) -> first()
   in
      text.value;

helper context SBVR!ObjectType def: getObjectTypeTerm(): SBVR!Term =
   let term : SBVR!Term = thisModule.termAllInstances -> select(i | i.meaning = self and
i.preferred = true) -> first() in term;

helper context SBVR!ObjectType def: getObjectTypeTermCount(): Integer =
   let size : Integer = thisModule.termAllInstances -> select(i | i.meaning =
self).size() in size;

helper context SBVR!ObjectType def: getObjectTypeTermNotPreferred(): SBVR!Text =
    thisModule.termAllInstances -> select(i | i.meaning = self and i.preferred = false) -
> first().expression;

-------------------------------------------------------------------------------
rule ObjectType2Table {
   from
      o: SBVR!ObjectType ( thisModule.itIsObjectForTable(o) and not
o.isPrimitiveDataType()
      )
   using {
      tableName : String = thisModule.ObjectTypeText(o).value.toString().toUpper();
      generalName: String = OclUndefined;
      canCreate: Boolean = true;
   }
   do {
      -- eliminate segmentation and categorization
      for (cs in SBVR!CategorizationScheme.allInstances()){
         for(category in cs.containsCategory){
            if (thisModule.ObjectTypeText(category).value.toUpper() = tableName){
               canCreate <- false;
            }
         }
```

```
      }
      if (canCreate){
         thisModule.createTable(tableName);
         if (not o.general.isEmpty()){
            generalName <-
o.general.first().getConceptDesignationTextValue().toString().toUpper();
            thisModule.makeReferentialColumnConstraint(tableName, generalName);
         }
         if (o.getObjectTypeTermCount() > 1){
            if (not o.getObjectTypeTermNotPreferred().oclIsUndefined()) {
               thisModule.createSynonim(thisModule.getTableByName(tableName),
o.getObjectTypeTermNotPreferred().value.toString().toUpper());
            }
         }
      }
   }
}


rule createTable(tableName: String) {
   to
      t: SQL!Table (
            name <- tableName,
            schemaName <- thisModule.schemaName,
            comment <- 'Table ' + tableName,
            owner <- thisModule.script

         )
   do {
      thisModule.makeColumnID(t);
      tableName.debug('CREATED TABLE (ObjectType)');
      thisModule.tableAllInstances <- thisModule.tableAllInstances.append(t);
   }
}

rule createSynonim(table: SQL!Table, synonimName: String) {
   to
      t: SQL!Synonim (
            table  <- table,
            synonimName <- synonimName
         )
}


helper def: objectIsNotANumber(ot: SBVR!ObjectType): Boolean =
   let ipt: Boolean = not
thisModule.ObjectTypeText(ot).value.toString().toUpper().startsWith('NUMBER')
   in ipt;


helper def: itIsObjectForTable(ot: SBVR!ObjectType): Boolean =
   let ipt: SBVR!IsPropertyOfFactType =
      if (SBVR!IsPropertyOfFactType.allInstances().size() > 0 and
SBVR!IsPropertyOfFactType.allInstances() -> select(i | i.role -> first().objectType ->
first() = ot).size() > 0) then
            SBVR!IsPropertyOfFactType.allInstances() -> select(i | i.role ->
first().objectType -> first() = ot) -> first()
```

```
        else
           OclUndefined
        endif
   in ipt.oclIsUndefined();


helper def: getTableByName(name: String): SQL!Table =
   let ipt: SQL!Table =
      if (thisModule.tableAllInstances.notEmpty() and thisModule.tableAllInstances ->
select(i | i.refGetValue('name') = name).notEmpty()) then
        thisModule.tableAllInstances -> select(i | i.refGetValue('name') = name) ->
first()
      else OclUndefined endif
   in ipt;


-------------------------------------------------------------------------------------
rule AssociativeFactType2Constraint {
   from
      s: SBVR!FactType (
        not s.oclIsTypeOf(SBVR!IsPropertyOfFactType) and s.
             oclIsTypeOf(SBVR!AssociativeFactType) and
SBVR!SententialForm.allInstances() -> select(tsf | tsf.meaning = s).size() = 1
             and thisModule.objectIsNotANumber(s.role -> last().objectType -> first())
      )
   using {
        firstRole : String = thisModule.getRoleObjectString(s.role -> first());
        lastRole : String = thisModule.getRoleObjectString(s.role -> last());
   }
   to
      c: SQL!Column (
        name <- firstRole + '_ID',
        constraint <- Sequence{con}
      ),
      con: SQL!ReferentialColumnConstraint (
        referencedTable <- tr,
        referencedColumns <- Sequence{'ID'},
        name <- 'FK_' + lastRole + '_' + firstRole + '_' + thisModule.number,
        owner <- c
      ),
      tr: SQL!TableReference (
        schemaName <- thisModule.schemaName,
        referenceTableName <- firstRole
      )
   do {
      thisModule.referentialColumnConstraintAllInstances <-
thisModule.referentialColumnConstraintAllInstances.append(con);
      thisModule.columnAllInstances <- thisModule.columnAllInstances.append(c);
      c.owner <- thisModule.getTableByName(lastRole);
      thisModule.setExactNumericIntType(c);
      thisModule.number <- thisModule.number + 1;
   }
}

rule makeReferentialColumnConstraint(table1: String, table2: String) {
   to
      c: SQL!Column (
```

```
      owner <- thisModule.getTableByName(table1),
      name <- table2 + '_ID',
      constraint <- Sequence{con}
   ),
   con: SQL!ReferentialColumnConstraint (
      referencedTable <- tr,
      referencedColumns <- Sequence{'ID'},
      name <- 'FK_' + table2 + '_' + table1 + '_' + thisModule.number,
      owner <- c
   ),
   tr: SQL!TableReference (
      schemaName <- thisModule.schemaName,
      referenceTableName <- table2
   )
do {
   thisModule.referentialColumnConstraintAllInstances <- thisModule.
        referentialColumnConstraintAllInstances.append(con);
   thisModule.setExactNumericIntType(c);
   thisModule.columnAllInstances <- thisModule.columnAllInstances.append(c);
   thisModule.number <- thisModule.number + 1;
}
}

rule AssociativeFactType2Table {
from
   s: SBVR!FactType (
      not s.oclIsTypeOf(SBVR!IsPropertyOfFactType) and s.
           oclIsTypeOf(SBVR!AssociativeFactType) and
SBVR!SententialForm.allInstances() -> select(tsf | tsf.meaning = s).size() > 1
   )
using {
      firstRole : String = thisModule.getRoleObjectString(s.role -> first());
      lastRole : String = thisModule.getRoleObjectString(s.role -> last());
      tableName : String = firstRole + '_' + lastRole;
}
to
   t: SQL!Table (
      name <- tableName,
      schemaName <- thisModule.schemaName,
      comment <- 'Table ' + tableName,
      owner <- thisModule.script,
      elements <- Sequence{c1,c2}
      --elements <- thisModule.Objects2Column(o)
   ),
   c1: SQL!Column (
      owner <- t,
      name <- firstRole + '_ID',
      constraint <- Sequence{con1}
   ),
   c2: SQL!Column (
      owner <- t,
      name <- lastRole + '_ID',
      constraint <- Sequence{con1}
   ),
   con1: SQL!ReferentialColumnConstraint (
      referencedTable <- tr1,
      referencedColumns <- Sequence{'ID'},
```

```
      name <- 'FK_' + firstRole + '_' + thisModule.number,
      owner <- c1
    ),
    con2: SQL!ReferentialColumnConstraint (
      referencedTable <- tr2,
      referencedColumns <- Sequence{'ID'},
      name <- 'FK_' + lastRole + '_' + thisModule.number,
      owner <- c2
    ),
    tr1: SQL!TableReference (
      schemaName <- thisModule.schemaName,
      referenceTableName <- firstRole
    ),
    tr2: SQL!TableReference (
      schemaName <- thisModule.schemaName,
      referenceTableName <- lastRole
    )
  do {
    thisModule.makeColumnID(t);
    tableName.debug('CREATED TABLE (FactType)');
    thisModule.tableAllInstances <- thisModule.tableAllInstances.append(t);
    thisModule.referentialColumnConstraintAllInstances <-
thisModule.referentialColumnConstraintAllInstances.append(con1);
    thisModule.referentialColumnConstraintAllInstances <-
thisModule.referentialColumnConstraintAllInstances.append(con2);
    thisModule.addNotNullConstraint(c1);
    thisModule.addNotNullConstraint(c2);
    thisModule.setExactNumericIntType(c1);
    thisModule.setExactNumericIntType(c2);
    thisModule.columnAllInstances <- thisModule.columnAllInstances.append(c1);
    thisModule.columnAllInstances <- thisModule.columnAllInstances.append(c2);
    thisModule.number <- thisModule.number + 1;
  }

}


helper context SBVR!FactType def: getFactTypeSententialForm():
    SBVR!SententialForm =
  let sf : SBVR!SententialForm = SBVR!SententialForm.allInstances() -> select(tsf | tsf.
      meaning = self) -> first() in sf;

helper def: getRoleObjectString(role: SBVR!Role): String =
  let text: SBVR!Text =
    thisModule.ObjectTypeText(role.objectType -> first())
  in
    text.value.toString().toUpper();

-------------------------------------------------------------------------------
rule CategorizationSchemeAndSegmentationTrasformation {
  from
    ic: SBVR!CategorizationScheme
  using {
    tableName: String = ic.isFor.first().getConceptDesignationTextValue().toUpper();
    newCollumnName: String = ic.isBasedOn.getConceptDesignationTextValue().toUpper();

    table: SQL!Table = OclUndefined;
```

```
      categoryColumnTitle: String = 'NAME';
   }
   do {
      table <- thisModule.getTableByName(tableName);

      -- create CATEGORY table, column, ref, insert statement.
      if (ic.oclIsTypeOf(SBVR!CategorizationScheme)){
         -- CATEGORY TABLE
         if (thisModule.getTableByName(thisModule.categoryTableName).oclIsUndefined()){
            thisModule.createCategorizationTable();
         }

         if (newCollumnName.endsWith('TYPE')){
            newCollumnName <- newCollumnName + '_ID';
         } else {
            newCollumnName <- newCollumnName + '_TYPE_ID';
         }

         -- make column
         thisModule.makeColumn(table, newCollumnName, true);
         -- ref
         thisModule.makeRefToCategoryTable(thisModule.getColumnByName(newCollumnName,
table), tableName);
         -- inserts
         for(category in ic.containsCategory){
            thisModule.addInsertStatement(thisModule.categoryTableName,
   Sequence{categoryColumnTitle}, Sequence{thisModule.ObjectTypeText(category).value});
         }
      }
      -- create column, check constraint.
      if (ic.oclIsTypeOf(SBVR!Segmentation)){
         -- column
         thisModule.makeColumn(table, newCollumnName, false);
         for(category in ic.containsCategory){

   thisModule.addcheckConstraintToColumn(thisModule.ObjectTypeText(category).value,
thisModule.getColumnByName(newCollumnName, table), '');
         }
      }
   }
}

helper def: isCategoryTableAlreadyCreated(): Boolean =
   let ipt: SQL!Table = thisModule.getTableByName(thisModule.categoryTableName)
   in (not ipt.oclIsUndefined());


rule makeColumn(t: SQL!Table, title: String, isID: Boolean) {
   to
      c: SQL!Column (
         owner <- t,
         name <- title
      )
   do {
      if (isID){
         thisModule.setExactNumericIntType(c);
      } else {
```

```
        thisModule.setCharacterStringType(c);
      }
      thisModule.columnAllInstances <- thisModule.columnAllInstances.append(c);
   }
}

rule makeRefToCategoryTable(column: String, tableName: String) {
   to
      con: SQL!ReferentialColumnConstraint (
         referencedTable <- tr,
         referencedColumns <- Sequence{'ID'},
         name <- 'FK_' + tableName + '_' + thisModule.categoryTableName + '_' +
thisModule.number,
         owner <- column
      ),
      tr: SQL!TableReference (
         schemaName <- thisModule.schemaName,
         referenceTableName <- thisModule.categoryTableName
      )
   do {
      thisModule.referentialColumnConstraintAllInstances <-
thisModule.referentialColumnConstraintAllInstances.append(con);
      thisModule.number <- thisModule.number + 1;
   }
}

rule createCategorizationTable() {
   using {
      tableName : String = thisModule.categoryTableName;
   }
   to
      t: SQL!Table (
         name <- tableName,
         schemaName <- thisModule.schemaName,
         comment <- 'Table ' + tableName,
         owner <- thisModule.script
      )
   do {
      thisModule.makeColumnID(t);
      thisModule.makeColumnName(t);
      tableName.debug('CREATED TABLE (CategorizationType)');
      thisModule.tableAllInstances <- thisModule.tableAllInstances.append(t);
   }
}

helper context SBVR!Segmentation def: getName(): SBVR!Name =
   let name : SBVR!Name = SBVR!Name.allInstances() -> select(n | n.refersTo ->
        includes(self)).first() in name;



helper def: IndividualConceptText(i: SBVR!IndividualConcept): SBVR!Text =
   thisModule.nameAllInstances -> select(n | n.meaning = i) -> asSequence() -> first().
        expression;
```

```
helper context SBVR!IndividualConcept def: getIndividualInstantiationFormulation():
SBVR!InstantiationFormulation =
   let sif: SBVR!InstantiationFormulation =
      if (SBVR!InstantiationFormulation.allInstances() -> select(isif |
isif.bindableTarget = self).size() > 0) then
      SBVR!InstantiationFormulation.allInstances() -> select(isif | isif.bindableTarget =
self) -> first()
      else
         OclUndefined
      endif
   in sif;


helper context SBVR!IndividualConcept def: isInstanceOfRole(): Boolean =
   let sif : SBVR!InstantiationFormulation = self.getIndividualInstantiationFormulation()
in
      if (sif.oclIsUndefined()) then
         false
      else
         if (sif.concept.oclIsTypeOf(SBVR!Role)) then
            true
         else
            false
         endif
      endif;

helper context SBVR!Concept def: getConceptDesignationTextValue(): String =
   let designation: SBVR!Designation =
      SBVR!Designation.allInstances() -> select(di | di.meaning = self) -> first()
   in
      thisModule.getDesignationTextValue(designation);

helper def: getDesignationTextValue(designation: SBVR!Designation): String =
   let text: SBVR!Text =
      SBVR!Text.allInstances() -> select(ti | ti = designation.expression) -> first()
   in
      text.value;

--------------------------------------------------------------------------------
rule makeColumnID(t: SQL!Table) {
   to
      id: SQL!Column (
         owner <- t,
         name <- 'ID',
         dataType <- dt,
         constraint <- Sequence{con2},
         constraint <- Sequence{con},
         defaultOption <- defOp
      ),
      dt: SQL!ExactNumericType (
         kind <- #INT
      ),
      con: SQL!UniqueColumnConstraint (
         kind <- #PRIMARY_KEY
      ),
      con2: SQL!NotNullColumnConstraint (
         owner <- id
```

```
      ),
      defOp: SQL!AutoIncrementOption (
         owner <- id
      )
   do {
      thisModule.columnAllInstances <- thisModule.columnAllInstances.append(id);
   }
}

rule makeColumnTypeId(t: SQL!Table) {
   to
      c: SQL!Column (
         owner <- t,
         name <- 'TYPE_ID'
      )
   do {
      thisModule.setExactNumericIntType(c);
      thisModule.columnAllInstances <- thisModule.columnAllInstances.append(c);
   }
}

rule makeColumnName(t: SQL!Table) {
   to
      c: SQL!Column (
         owner <- t,
         name <- 'NAME'
      )
   do {
      thisModule.setCharacterStringType(c);
      thisModule.columnAllInstances <- thisModule.columnAllInstances.append(c);
   }
}

-------------------------------------------------------------------------------------
helper def: ValuePrimitiveTypeText: String ='TEXT';
helper def: ValuePrimitiveTypeNumber: String ='NUMBER'; -- su kableliu
helper def: ValuePrimitiveTypeInteger: String ='INTEGER';
helper def: ValuePrimitiveTypeNonnegativeInteger: String ='NONNEGATIVE_INTEGER';
helper def: ValuePrimitiveTypePositiveInteger: String ='POSITIVE_INTEGER';
helper def: ValuePrimitiveTypeBoolean: String ='BOOLEAN';
helper def: ValuePrimitiveTypeDatetime: String ='DATETIME';

rule IsPropertyOfFactType2Column {
   from
      s: SBVR!FactType (
         s.oclIsTypeOf(SBVR!IsPropertyOfFactType) and not s.
              oclIsTypeOf(SBVR!AssociativeFactType)
      )
   using {
      typeName: String = thisModule.getRoleObjectString(s.role -> first());
      tableName: String = thisModule.getRoleObjectString(s.role -> last());
   }
   to
      c: SQL!Column (
         name <- s.getPropertyName(s.getPropertyNamePlaceholder()).toUpper()
         --owner <- s.role -> last().objectType -> first()
      )
```

```
   do {
        c.debug('Table: ' + tableName + '. Created column: ');
        c.owner <- thisModule.getTableByName(tableName);
        if (typeName = thisModule.ValuePrimitiveTypeText){
           thisModule.setCharacterStringType(c);
        } else if (typeName = thisModule.ValuePrimitiveTypeInteger){
           thisModule.setExactNumericIntType(c);
        } else if (typeName = thisModule.ValuePrimitiveTypeDatetime){
           thisModule.setDateTimeType(c);
        } else if (typeName = thisModule.ValuePrimitiveTypeNumber){
           thisModule.setExactNumericDecimalType(c);
        }

        thisModule.columnAllInstances <- thisModule.columnAllInstances.append(c);
        -- ADD constraints
        if (not thisModule.getCheckConstraintByColumnName(c.name).oclIsUndefined()){
           for(constraint in thisModule.getCheckConstraintByColumnName(c.name)){
              constraint.owner <- c;
           }
        }
     }
}

rule IndividualConcept2Insert {
   from
      ic: SBVR!IndividualConcept ((not
ic.getIndividualInstantiationFormulation().oclIsUndefined())
      )
   using {
      sif: SBVR!InstantiationFormulation = ic.getIndividualInstantiationFormulation();
      individualName: String = ic.getConceptDesignationTextValue();
      tableName: String = sif.concept.getConceptDesignationTextValue().toUpper();
      table: SQL!Table = OclUndefined;
      column: SQL!Column = OclUndefined;
   }
   do {
      table <- thisModule.getTableByName(tableName);
      for (col in table.elements){
         if (col.refGetValue('name').toString() = 'NAME'){
            column <- col;
         }
      }
      if (column.oclIsUndefined()){
         thisModule.makeColumnName(table);
      }
      thisModule.addInsertStatement(tableName, Sequence{'NAME'},
Sequence{individualName});
        }
}

helper def: getCheckConstraintByColumnName(name: String):
     Sequence(SQL!CheckColumnConstraint) =
   let ipt: SQL!CheckColumnConstraint =
      if (thisModule.checkConstraintAllInstances.notEmpty() and thisModule.
           checkConstraintAllInstances -> select(i | i.refGetValue('columnName') =
           name).notEmpty()) then
         thisModule.checkConstraintAllInstances -> select(i | i.
```

```
            refGetValue('columnName') = name)
      else OclUndefined endif
   in ipt;

rule setCharacterStringType(c: SQL!Column){
   to
      dt: SQL!CharacterStringType (
         owner <- c,
         kind <- #VARCHAR
      )
}

rule setExactNumericIntType(c: SQL!Column){
   to
      dt: SQL!ExactNumericType (
         owner <- c,
         kind <- #INT
      )
}

rule setExactNumericDecimalType(c: SQL!Column){
   to
      dt: SQL!ExactNumericType (
         owner <- c,
         kind <- #DECIMAL
      )
}

rule setDateTimeType(c: SQL!Column){
   to
      dt: SQL!DateType (
         owner <- c,
         kind <- #DATETIME
      )
}

helper context SBVR!FactType def: getPropertyNamePlaceholder(): SBVR!Placeholder =
   let placeh : SBVR!Placeholder = SBVR!Placeholder.allInstances() -> select(i | i.
         meaning = self.role -> first()) -> first() in placeh;

helper context SBVR!FactType def: getPropertyName(placeh: SBVR!Placeholder): String =
   let text: SBVR!Text =
      SBVR!Text.allInstances() -> select(i | i = placeh.expression) -> first()
   in
      text.value;

-------------------------------------------------------------------------------------
rule ModalFormulation2Constraints {
   from
      s: SBVR!ModalFormulation
   do {
      if (s.oclIsTypeOf(SBVR!NecessityFormulation) or
s.oclIsTypeOf(SBVR!ObligationFormulation)){
         thisModule.LogicalFormulation2Constraints(s.logicalFormulation);
      }
      else {
         s.debug('NEAPRASYTA');
```

```
      }
   }
}

rule LogicalFormulation2Constraints(lf: SBVR!LogicalFormulation) {
   do {
         if (lf.oclIsTypeOf(SBVR!UniversalQuantification)){
            --AtLeastNQuantification (bent n) at_least SUGENERUOTI SELECT + NOT NULL
            if (lf.scopeFormulation.oclIsTypeOf(SBVR!AtLeastNQuantification)){
               thisModule.AtLeastNQuantification(lf);

            --ExistentialQuantification (bent 1) at_least
            } else if (lf.scopeFormulation.oclIsTypeOf(SBVR!ExistentialQuantification)){
               thisModule.ExistentialQuantification(lf);


            --ExactlyNQuantification (exactly N) ir ExactlyOneQuantification
            }  else if (lf.scopeFormulation.oclIsTypeOf(SBVR!ExactlyNQuantification)){
               thisModule.ExactlyNQuantification(lf);


            --AtMostNQuantification ir AtMostOneQuantification
            }  else if (lf.scopeFormulation.oclIsTypeOf(SBVR!AtMostNQuantification)){
               thisModule.AtMostNQuantification(lf);
               -- TRIGER AFTER INSERT

            --NumericRangeQuantification (mazesne uz x  didesne uz y)
            }  else if (lf.scopeFormulation.oclIsTypeOf(SBVR!NumericRangeQuantification)){
               thisModule.NumericRangeQuantification(lf);

            --AtomicFormulation (mazesne uz x  didesne uz y)
            }  else if (lf.scopeFormulation.oclIsTypeOf(SBVR!AtomicFormulation)){
               thisModule.AtomicFormulation2Constraint(lf.scopeFormulation);

            }  else {
               lf.scopeFormulation.debug('ERROR. NEAPRASYTAS TIPAS: ' +
lf.scopeFormulation.oclType());
            }

         }  else if (lf.oclIsTypeOf(SBVR!Conjunction)){
               'NOT IMPLEMENTED'.debug();
               --thisModule.ModalFormulationConjunction2Constraints(lf);
            }
   }
}


helper def: getAtLeastNQuantificationIntegerValue(int: SBVR!AtLeastNQuantification):
Integer =
   let i: Integer =
      int.minimumCardinality.value
   in
      i;

helper def: getExactlyNQuantificationIntegerValue(int: SBVR!ExactlyNQuantification):
Integer =
   let i: Integer =
```

```
      int.cardinality.value
   in
      i;

helper def: getAtMostNQuantificationIntegerValue(int: SBVR!AtMostNQuantification):
Integer =
   let i: Integer =
      int.maximumCardinality.value
   in
      i;

helper def: getNumericRangeQuantificationMinIntegerValue(int:
SBVR!NumericRangeQuantification): Sequence(Integer) =
   let i: Integer =
      int.minimumCardinality.value
   in
      i;

helper def: getNumericRangeQuantificationMaxIntegerValue(int:
SBVR!NumericRangeQuantification): Sequence(Integer) =
   let i: Integer =
      int.maximumCardinality.value
   in
      i;


rule addSelectCheck(parent: String, child: String, n: Integer, sign : String) {
   to
      s: SQL!selectChecker (
         parentTable <- parent,
         childTable <- child,
         valueToCheck <- n,
         owner <- thisModule.script
      )
   do {
      ''.debug('Created SelectCheck for tables: ' + parent + ', ' + child + '. Value to
check: '+ n + '. Sign: ' + sign);
      if (sign.startsWith(thisModule.moreThen)){
         s.comparison <- #GREATER_THAN;
      } else if (sign.startsWith(thisModule.lessThen)){
         s.comparison <- #LESS_THAN;
      } else if (sign.startsWith(thisModule.equals)){
         s.comparison <- #EQUALS;
      }

   }
}

rule addSelectCheckBetween(parent: String, child: String, n1: Integer, n2 : Integer) {
   to
      s: SQL!selectCheckerBetween (
         parentTable <- parent,
         childTable <- child,
         valueFrom <- n1,
         valueTo <- n2,
         owner <- thisModule.script
      )
```

```
   do {
     ''.debug('Created SelectCheckBetween for tables: ' + parent + ', ' + child + '.
Value to check between: '+ n1 + ' and ' + n2);
   }
}

-- SELECT + NOT NULL
rule AtLeastNQuantification(lf: SBVR!LogicalFormulation){
   using {
      quan : SBVR!AtLeastNQuantification = lf.scopeFormulation;
      object : SBVR!ObjectType =
thisModule.ObjectTypeText(lf.variable.rangedOverConcept);
      elementFrom : String = object.value.toString().toUpper();
      elementTo : String = OclUndefined;
   }
   do {
      '------------'.debug('------------  AtLeastNQuantification');
      if (quan.variable.rangedOverConcept.oclIsTypeOf(SBVR!ObjectType)) {
         elementTo <-
thisModule.ObjectTypeText(quan.variable.rangedOverConcept).value.toString().toUpper();
         if (not thisModule.getTableByName(elementFrom +'_' +
elementTo).oclIsUndefined()){
            elementTo <- elementFrom +'_' + elementTo;
         } else if (not thisModule.getTableByName(elementTo +'_' +
elementFrom).oclIsUndefined()){
            elementTo <- elementTo +'_' + elementFrom;
         }
         thisModule.addSelectCheck(elementFrom, elementTo ,
thisModule.getAtLeastNQuantificationIntegerValue(quan), thisModule.lessThen);

         for(t in thisModule.referentialColumnConstraintAllInstances){
              if (t.refGetValue('name').toString().startsWith('FK_' + elementTo +'_' +
elementFrom)){
                 thisModule.addNotNullConstraint(t.refGetValue('owner'));
                 ''.debug('Created NOT NULL constraint for collumn: '+
t.refGetValue('name').toString());
              }
            }
      }
      else {
         quan.debug('NEAPRASYTA');
      }
   }
}

-- NOT NULL
rule ExistentialQuantification(lf: SBVR!LogicalFormulation){
   using {
      quan : SBVR!ExistentialQuantification =lf.scopeFormulation;
      object : SBVR!ObjectType =
thisModule.ObjectTypeText(lf.variable.rangedOverConcept);
      elementFrom : String = object.value.toString().toUpper();
      elementTo : String = OclUndefined;
      column : SQL!Column = OclUndefined;
   }
   do {
      '------------'.debug('------------  ExistentialQuantification');
```

```
      --  lentelei (TURI BUTI FK NOT NULL)
      if (quan.variable.rangedOverConcept.oclIsTypeOf(SBVR!ObjectType)) {
         elementTo <-
thisModule.ObjectTypeText(quan.variable.rangedOverConcept).value.toString().toUpper();
         for(t in thisModule.referentialColumnConstraintAllInstances){
            if (t.refGetValue('name').toString().startsWith('FK_' + elementTo +'_' +
elementFrom)){
               thisModule.addNotNullConstraint(t.refGetValue('owner'));
               ''.debug('Created NOT NULL constraint for collumn: '+
t.refGetValue('name').toString());
            }
         }
      -- stulpeliui (TURI STULPELIS TAPTI NOT NULL)
      } else if (quan.variable.rangedOverConcept.oclIsTypeOf(SBVR!Role)) {
         elementTo <-
thisModule.RoleTypeText(quan.variable.rangedOverConcept).value.toString().toUpper();
         for (table1 in thisModule.tableAllInstances) {
            for (element in table1.elements){
               if (element.refGetValue('name').toString() = elementTo and
table1.refGetValue('name').toString() = elementFrom){
                  column <- element;
               }
            }
         }
         thisModule.addNotNullConstraint(column);
         ''.debug('Created NOT NULL constraint for collumn: '+ column.name);
      }  else {
         quan.variable.rangedOverConcept.debug('ERROR. NEAPRASYTAS TIPAS2: ');
      }
   }
}

-- NOT NULL + TRIGGER + SELECT
rule ExactlyNQuantification(mf: SBVR!LogicalFormulation){
   using {
      quan : SBVR!ExactlyNQuantification = mf.scopeFormulation;
      object : SBVR!ObjectType =
thisModule.ObjectTypeText(mf.variable.rangedOverConcept);
      elementFrom : String = object.value.toString().toUpper();
      elementTo : String = OclUndefined;
      column : SQL!Column = OclUndefined;
   }
   do {
      '------------'.debug('------------  ExactlyNQuantification');
      --  lentelei (TURI BUTI FK NOT NULL)
      if (quan.variable.rangedOverConcept.oclIsTypeOf(SBVR!ObjectType)) {
         elementTo <-
thisModule.ObjectTypeText(quan.variable.rangedOverConcept).value.toString().toUpper();
         for(t in thisModule.referentialColumnConstraintAllInstances){
            if (t.refGetValue('name').toString().startsWith('FK_' + elementTo +'_' +
elementFrom)){
               thisModule.addNotNullConstraint(t.refGetValue('owner'));
               ''.debug('Created NOT NULL constraint for collumn: '+
t.refGetValue('name').toString());
            }
         }
```

```
        if (not thisModule.getTableByName(elementFrom +'_' +
elementTo).oclIsUndefined()){
           elementTo <- elementFrom +'_' + elementTo;
        } else if (not thisModule.getTableByName(elementTo +'_' +
elementFrom).oclIsUndefined()){
           elementTo <- elementTo +'_' + elementFrom;
        }
        thisModule.addSelectCheck(elementFrom, elementTo ,
thisModule.getExactlyNQuantificationIntegerValue(quan), thisModule.equals);
        thisModule.addTrigerSelectChecker('TRIGGER ExactlyNQuantification ne daugiau nei
value: ' +
           thisModule.getExactlyNQuantificationIntegerValue(quan).toString(),
elementFrom, elementTo ,
           thisModule.getExactlyNQuantificationIntegerValue(quan), thisModule.moreThen);

     -- stulpeliui (TURI STULPELIS TAPTI NOT NULL)
     } else if (quan.variable.rangedOverConcept.oclIsTypeOf(SBVR!Role)) {
        elementTo <-
thisModule.RoleTypeText(quan.variable.rangedOverConcept).value.toString().toUpper();
        column <- thisModule.columnAllInstances -> select(i | i.refGetValue('name') =
elementTo and i.refGetValue('owner').refGetValue('name') = elementFrom);
        if (column.notEmpty()){
           thisModule.addNotNullConstraint(column -> first());
           ''.debug('Created NOT NULL constraint for collumn: '+ column -> first().name);
        }
     }
   }
}


-- NOT NULL + TRIGGER
rule AtMostNQuantification(mf: SBVR!LogicalFormulation){
   using {
      quan : SBVR!AtMostNQuantification = mf.scopeFormulation;
      object : SBVR!ObjectType =
thisModule.ObjectTypeText(mf.variable.rangedOverConcept);
      elementFrom : String = object.value.toString().toUpper();
      elementTo : String = OclUndefined;
      column : SQL!Column = OclUndefined;
   }
   do {
      '------------'.debug('------------  AtMostNQuantification');
      --  lentelei (TURI BUTI FK NOT NULL)
      if (quan.variable.rangedOverConcept.oclIsTypeOf(SBVR!ObjectType)) {
        elementTo <-
thisModule.ObjectTypeText(quan.variable.rangedOverConcept).value.toString().toUpper();
        for(t in thisModule.referentialColumnConstraintAllInstances){
           if (t.refGetValue('name').toString().startsWith('FK_' + elementTo +'_' +
elementFrom)){
              thisModule.addNotNullConstraint(t.refGetValue('owner'));
              ''.debug('Created NOT NULL constraint for collumn: '+
t.refGetValue('name').toString());
           }
        }
        if (not thisModule.getTableByName(elementFrom +'_' +
elementTo).oclIsUndefined()){
           elementTo <- elementFrom +'_' + elementTo;
```

```
        } else if (not thisModule.getTableByName(elementTo +'_' +
elementFrom).oclIsUndefined()){
           elementTo <- elementTo +'_' + elementFrom;
        }
        thisModule.addTrigerSelectCheckMax('TRIGGER AtMostNQuantification: ' +
thisModule.getAtMostNQuantificationIntegerValue(quan).toString(),
           elementFrom, elementTo ,
thisModule.getAtMostNQuantificationIntegerValue(quan));
     -- stulpeliui (TURI STULPELIS TAPTI NOT NULL)
     } else if (quan.variable.rangedOverConcept.oclIsTypeOf(SBVR!Role)) {
        elementTo <-
thisModule.RoleTypeText(quan.variable.rangedOverConcept).value.toString().toUpper();
        column <- thisModule.columnAllInstances -> select(i | i.refGetValue('name') =
elementTo and i.refGetValue('owner').refGetValue('name') = elementFrom);
        if (column.notEmpty()){
           thisModule.addNotNullConstraint(column -> first());
           ''.debug('Created NOT NULL constraint for collumn: '+ column.name);
        }
     }
   }
}

-- NOT NULL + TRIGGER +SELECT
rule NumericRangeQuantification(mf: SBVR!LogicalFormulation){
   using {
      quan : SBVR!NumericRangeQuantification = mf.scopeFormulation;
      object : SBVR!ObjectType =
thisModule.ObjectTypeText(mf.variable.rangedOverConcept);
      elementFrom : String = object.value.toString().toUpper();
      elementTo : String = OclUndefined;
      column : SQL!Column = OclUndefined;
   }
   do {
      '------------'.debug('------------  NumericRangeQuantification');
      --  lentelei (TURI BUTI FK NOT NULL)
      if (quan.variable.rangedOverConcept.oclIsTypeOf(SBVR!ObjectType)) {
        elementTo <-
thisModule.ObjectTypeText(quan.variable.rangedOverConcept).value.toString().toUpper();
        for(t in thisModule.referentialColumnConstraintAllInstances){
           if (t.refGetValue('name').toString().startsWith('FK_' + elementTo +'_' +
elementFrom)){
              thisModule.addNotNullConstraint(t.refGetValue('owner'));
              ''.debug('Created NOT NULL constraint for collumn: '+
t.refGetValue('name').toString());
           }
        }
        if (not thisModule.getTableByName(elementFrom +'_' +
elementTo).oclIsUndefined()){
           elementTo <- elementFrom +'_' + elementTo;
        } else if (not thisModule.getTableByName(elementTo +'_' +
elementFrom).oclIsUndefined()){
           elementTo <- elementTo +'_' + elementFrom;
        }
        thisModule.addSelectCheckBetween(elementFrom, elementTo ,
thisModule.getNumericRangeQuantificationMinIntegerValue(quan),
thisModule.getNumericRangeQuantificationMaxIntegerValue(quan));
```

```
        thisModule.addTrigerSelectCheckMax('TRIGGER NumericRangeQuantification: iki ' +
thisModule.getNumericRangeQuantificationMaxIntegerValue(quan).toString(),
           elementFrom, elementTo ,
thisModule.getNumericRangeQuantificationMaxIntegerValue(quan));
      -- stulpeliui (TURI STULPELIS TAPTI NOT NULL)
      } else if (quan.variable.rangedOverConcept.oclIsTypeOf(SBVR!Role)) {
         elementTo <-
thisModule.RoleTypeText(quan.variable.rangedOverConcept).value.toString().toUpper();
        column <- thisModule.columnAllInstances -> select(i | i.refGetValue('name') =
elementTo and i.refGetValue('owner').refGetValue('name') = elementFrom);
        if (column.notEmpty()){
           thisModule.addNotNullConstraint(column -> first());
           ''.debug('Created NOT NULL constraint for collumn: '+ column.name);
        }
      }
   }
}

helper def: contrainingFacetTypeMap: Map(String, String) =
   Map{
      ('is_less_than_or_equal_to', #LESS_THAN_OR_EQUAL_TO),
      ('is_less_than', #LESS_THAN),
      ('is_greater_than_or_equal_to', #GREATER_THAN_OR_EQUAL_TO),
      ('is_greater_than', #GREATER_THAN),
      ('is_not_greater_than', #NOT_GREATER_THAN),
      ('is_not_less_than', #NOT_LESS_THAN),
      ('', #EQUALS)};


rule AtomicFormulation2Constraint(af: SBVR!AtomicFormulation){
   using {
      restriction: SBVR!Variable = af.getRoleBinding(1).bindableTarget;
      concept: SBVR!Variable = af.getRoleBinding(2).bindableTarget;
      secondAtomicForm: SBVR!AtomicFormulation = restriction.restrictingFormulation;
      factSymbol: String =
secondAtomicForm.factType.getFactTypeTextValue(secondAtomicForm.factType.getFactTypeFact
Symbol());
      column: SQL!Column = OclUndefined;
      columName: String =
thisModule.ObjectTypeText(restriction.rangedOverConcept).value.toUpper();
      tableName: String =
thisModule.ObjectTypeText(concept.rangedOverConcept).value.toUpper();
   }
   do {
      '------------'.debug('------------  AtomicFormulation');
      for (table1 in thisModule.tableAllInstances) {
         for (element in table1.elements){
            if (element.refGetValue('name').toString() = columName and
table1.refGetValue('name').toString() = tableName){
               column <- element;
            }
         }
      }
      thisModule.addcheckConstraintToColumn(

   thisModule.IndividualConceptText(secondAtomicForm.getRoleBinding(2).bindableTarget).va
lue.toString(),
```

```
        column,
        factSymbol);
   }

}

helper context SBVR!FactType def: getFactTypeFactSymbol(): SBVR!FactSymbol =
     let fsymbol : SBVR!FactSymbol = SBVR!FactSymbol.allInstances() -> select(i | i.
        meaning = self) -> first() in fsymbol;

helper context SBVR!FactType def: getFactTypeTextValue(fsymbol: SBVR!FactSymbol): String
=
   let text: SBVR!Text = SBVR!Text.allInstances() -> select(ti | ti = fsymbol.expression)
-> first()
   in text.value;

helper def:  FactTypeText(f:SBVR!FactSymbol):SBVR!Text = SBVR!FactSymbol.allInstances()-
>select(fs|fs.meaning=f)->asSequence()->first().expression;

helper context SBVR!AtomicFormulation def: getRoleBinding(roleNo: Integer):
SBVR!RoleBinding =
   let rb: SBVR!RoleBinding = SBVR!RoleBinding.allInstances() -> select(irb |
irb.atomicFormulation = self)
   in
      if (roleNo = 1) then
        rb -> first()
      else
        rb -> last()
      endif;

rule addNotNullConstraint(r: SQL!Column) {
   to
      con: SQL!NotNullColumnConstraint (
        owner <- r
      )
}


helper def: getColumnByName(name: String, table: SQL!Table): SQL!Column =
   let ipt: SQL!Column =
      if (thisModule.columnAllInstances.notEmpty() and thisModule.columnAllInstances ->
select(i | i.refGetValue('name') = name and i.refGetValue('owner') = table).notEmpty())
then
        thisModule.columnAllInstances -> select(i | i.refGetValue('name') = name and
i.refGetValue('owner') = table).first()
      else
        OclUndefined
      endif
   in ipt;

rule addTrigerSelectChecker(t: String, parent: String, child: String, n: Integer, sign :
String) {
   to
      s: SQL!triggerChecker (
        owner <- thisModule.script,
        parentTable <- parent,
        childTable <- child,
```

```
        valueToCheck <- n,
        owner <- thisModule.script
      )
   do {
      if (sign.startsWith(thisModule.moreThen)){
         s.sign <- '>';
      } else if (sign.startsWith(thisModule.lessThen)){
         s.sign <- '<';
      } else if (sign.startsWith(thisModule.equals)){
         s.sign <- '=';
      }
      ''.debug('Created Trigger for tables: ' + parent + ', ' + child + '. Value to
check: '+ n + '. Sign: ' + sign);

   }
}

rule addTrigerSelectCheckMax(t: String, parent: String, child: String, max : Integer) {
   to
      s: SQL!triggerCheckerMax (
         parentTable <- parent,
         childTable <- child,
         valueTo <- max,
         owner <- thisModule.script
      )
   do {
      ''.debug('Created Trigger for tables: ' + parent + ', ' + child + '. Value to check
(max): '+ max);
   }
}

rule addInsert(t: String) {
   to
      s: SQL!insert (
         text <- t.debug('insert: '),
         owner <- thisModule.script
      )
}

rule addInsertStatement(table: String, columns: Sequence(String), values:
Sequence(String)) {
   to
      s: SQL!insert (
         tableName <- table.debug('Created insert statement. COL: ' +columns.first() +'
VALUE: ' + values.first() + ' TABLE'),
         column <- columns,
         value <- values,
         owner <- thisModule.script
      )
}

rule addcheckConstraintToColumn(value: String, tableColumn: SQL!Column, compare: String)
{
   to
      s: SQL!CheckColumnConstraint (
         comparison <- thisModule.contrainingFacetTypeMap.get(compare),
         value <- value,
```

```
        columnName <- tableColumn.name,
        owner <- tableColumn
      )
   do {
      thisModule.checkConstraintAllInstances <-
thisModule.checkConstraintAllInstances.append(s);
      ''.debug('Created Check Constraint for column: ' + tableColumn.name + ' -> ' +
compare + ' '+ value);
   }
}

entrypoint rule TransformationEntryPoint() {
   to
      o: SQL!SQLScript
   do {
      ''.debug('------------------------------------------------------------------------
--------------------');
      thisModule.script <- o;
   }

}
```

8.5. **priedas.** Modelių transformacijos *Acceleo* kodas

```
[comment encoding = UTF-8 /]
[module generateDDL('http://www.eclipse.org/sql/')]
[template public generateDDL(script : SQLScript)]
[comment @main/]
[file ('tables.sql', false, 'UTF-8')]
---------------------- TABLES ------------------------------------------------------
------------
[for (t : Table | tables)]
CREATE TABLE [t.name/] (
   [for (col : Column | elements) separator (', ' + lineSeparator())]
   [col.name/] [getColumnDataType(col)/] [getDefaultOption(col)/][for (con :
ColumnConstraint |
constraint)][getConstraintNull(con)/][getConstraintUnique(con)/][/for][/for] [for (col :
Column | elements)][if (ifCheckConstraint(col))],
   CONSTRAINT CHK_[col.name/] CHECK ([for (con : CheckColumnConstraint | constraint)
separator (' OR ')][con.columnName/] [con.comparison/] '[con.value/]'[/for])[/if][/for]
[for (col : Column | elements)][if (ifContainsReferentialColumnConstraint(col))],
   [for (con : ColumnConstraint | constraint)][getReferencialConstraint(con, col,
t)/][/for][/if] [/for]
);
[if (not t.synonim.oclIsUndefined())]CREATE SYNONYM [t.synonim.synonimName/] FOR
[t.name/];
[/if]

[/for]


[/file]
[/template]


[query private getColumnDataType(c : Column) : String =
   if (not c.dataType.oclIsUndefined() and c.dataType.oclIsTypeOf(CharacterStringType))
```

```
     then c.dataType.oclAsType(CharacterStringType).kind
     else if (c.dataType.oclIsTypeOf(ExactNumericType))
          then c.dataType.oclAsType(ExactNumericType).kind
          else    if (c.dataType.oclIsTypeOf(DateType))
               then c.dataType.oclAsType(DateType).kind
               else ''
               endif
          endif
   endif

/]

[query private getDefaultOption(c : Column) : String =
   if (not c.defaultOption.oclIsUndefined() and
c.defaultOption.oclIsTypeOf(AutoIncrementOption)) then
     'IDENTITY(1,1) ' else '' endif
/]

[query private getConstraintNull(con : ColumnConstraint) : String =
   if con.oclIsTypeOf(NotNullColumnConstraint)
     then 'NOT NULL '
     else ''
   endif
/]

[query private getConstraintUnique(con : ColumnConstraint) : String =
   if con.oclIsTypeOf(UniqueColumnConstraint)
     then con.oclAsType(UniqueColumnConstraint).kind.toString() + ' '
     else ''
   endif
/]

[query private ifCheckConstraint(c : Column) : Boolean =
   if (c.constraint->asSequence()-> select (i | i.oclIsTypeOf(CheckColumnConstraint)) ->
isEmpty()) then false
   else true
   endif
/]

[query private ifContainsReferentialColumnConstraint(c : Column) : Boolean =
   if (c.constraint->asSequence()-> select (i |
i.oclIsTypeOf(ReferentialColumnConstraint)) -> isEmpty()) then false
   else true
   endif
/]

[query private getReferencialConstraint(con : ColumnConstraint, c : Column, t : Table) :
String =
   if (con.oclIsTypeOf(ReferentialColumnConstraint)) then
          ('CONSTRAINT ' + con.name + ' FOREIGN KEY ' + '(' + c.name + ')' + '
REFERENCES '
          +
con.oclAsType(ReferentialColumnConstraint).referencedTable.referenceTableName
          + '(' +
con.oclAsType(ReferentialColumnConstraint).referencedColumns.tokenizeLine() + ')')
     else ''
   endif
```

```
/]

[comment encoding = UTF-8 /]
[module generateConstraints('http://www.eclipse.org/sql/')]
[template public generateConstraints(script : SQLScript)]
[comment @main/]
[file ('constraints.sql', false, 'UTF-8')]
---------------------- CONSTRAINTS ----------------------------------------------------
-----------------
-- CHECK CONSTRAINTS
[for (t : Table | tables)]
   [for (col : Column | elements)]
      [if (ifCheckConstraint(col))]
ALTER TABLE [t.name/] ADD CONSTRAINT CHK_[col.name/] CHECK ([for (con :
CheckColumnConstraint | constraint) separator (' OR ')][con.columnName/]
[con.comparison/] '[con.value/]'[/for]);
      [/if]
   [/for]
[/for]

[/file]
[/template]


[query private ifCheckConstraint(c : Column) : Boolean =
   if (c.constraint->asSequence()-> select (i | i.oclIsTypeOf(CheckColumnConstraint)) ->
isEmpty()) then false
   else true
   endif
/]

[comment encoding = UTF-8 /]
[module generateInsert('http://www.eclipse.org/sql/')]
[template public generateInsert(script : SQLScript)]
[comment @main/]
[file ('inserts.sql', false, 'UTF-8')]
---------------------- INSERTS  -------------------------------------------------------
--------------
   [for (s : dml::insert | statements)]
INSERT INTO [s.tableName/] ([for (c : String | s.column)separator (', ' +
lineSeparator())][c/][/for]) VALUES ([for (v : String | s.value)separator (', ' +
lineSeparator())]'[v/]'[/for]);
   [/for]


[/file]
[/template]

[comment encoding = UTF-8 /]
[module generateSelectCheck('http://www.eclipse.org/sql/')]
[template public generateSelectCheck(script : SQLScript)]
[comment @main/]
[file ('checkRules.sql', false, 'UTF-8')]
---------------------- CHECKS  --------------------------------------------------------
-------------
[for (s : dml::selectChecker | statements)]
[if (s.eClass().name = 'selectChecker')]
```

```
SELECT *, ([getCountSelect(s)/]) AS [s.childTable/]_COUNT_NOW, [s.valueToCheck/] AS
[s.childTable/]_COUNT_MUST_BE
FROM [s.parentTable/] WHERE ([getCountSelect(s)/]) [getSign(s)/] [s.valueToCheck/];

[/if]
[/for]
[for (s : dml::selectCheckerBetween_|_statements)]
[if (s.eClass().name = 'selectCheckerBetween')]
SELECT *, ([getCountSelect(s)/]) AS [s.childTable/]_COUNT_NOW, [s.valueFrom/] AS
[s.childTable/]_COUNT_MUST_BE_FROM, [s.valueTo/] AS [s.childTable/]_COUNT_MUST_BE_TO
FROM [s.parentTable/] WHERE ([getCountSelect(s)/]) > [s.valueTo/] OR
([getCountSelect(s)/]) < [s.valueFrom/];

[/if]
[/for]


[/file]
[/template]

[query private getCountSelect(s : selectChecker) : String =
   'SELECT COUNT(*) FROM '+s.childTable+' WHERE '+s.parentTable+'_ID =
'+s.parentTable+'.ID'
/]

[query private getCountSelect(s : selectCheckerBetween) : String =
   'SELECT COUNT(*) FROM '+s.childTable+' WHERE '+s.parentTable+'_ID =
'+s.parentTable+'.ID'
/]

[query private getSign(s : selectChecker) : String =
   if s.comparison.toString().equalsIgnoreCase('less')
      then '<'
      else if s.comparison.toString().equalsIgnoreCase('more')
         then '<'
         else s.comparison.toString()
         endif
   endif
/]

[comment encoding = UTF-8 /]
[module generateTriggers('http://www.eclipse.org/sql/')]
[template public generateTriggers(script : SQLScript)]
[comment @main/]
[file ('triggers.sql', false, 'UTF-8')]
---------------------- TRIGGERS  ------------------------------------------------------
[for (s : dml::triggerChecker_|_statements)]
CREATE OR ALTER TRIGGER [s.parentTable/]_[s.childTable/]_COUNT_TRIGGER ON
[s.childTable/]
AFTER INSERT
AS
   DECLARE @COUNT int;
   DECLARE @NEW_ROW_ID int = (SELECT [s.parentTable/]_ID FROM INSERTED);
   SELECT @COUNT = COUNT(*) FROM [s.childTable/] INNER JOIN [s.parentTable/] ON
[s.parentTable/]_ID = [s.parentTable/].ID AND [s.parentTable/]_ID = @NEW_ROW_ID;

   IF @COUNT [s.sign/] [s.valueToCheck/]
```

```
   BEGIN
       rollback
       RAISERROR ('Violated business rules', 16, 1);
   END
GO

[/for]
[for (s : dml::triggerCheckerMax_|_statements)]
CREATE OR ALTER TRIGGER [s.parentTable/]_[s.childTable/]_COUNT_TRIGGER ON
[s.childTable/]
AFTER INSERT
AS
   DECLARE @COUNT int;
   DECLARE @NEW_ROW_ID int = (SELECT [s.parentTable/]_ID FROM INSERTED);
   SELECT @COUNT = COUNT(*) FROM [s.childTable/] INNER JOIN [s.parentTable/] ON
[s.parentTable/]_ID = [s.parentTable/].ID AND [s.parentTable/]_ID = @NEW_ROW_ID;

   IF @COUNT > [s.valueTo/]
   BEGIN
       rollback
       RAISERROR ('Violated business rules', 16, 1);
   END
GO

[/for]


[/file]
[/template]
```

8.6. **priedas.** Transformacijos metų gautų modelių palyginimo reikšmės

Darbuotojų informacinės sistemos rezultatai duomenų modelių kūrimo scenarijai ir rezultatai:

```
---------------------- TABLES ------------------------------------------------------
CREATE TABLE PERSON (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            GENDER BIT ,
            FIRST_NAME VARCHAR(255) ,
            LAST_NAME VARCHAR(255) ,
            );

CREATE TABLE EMPLOYEE (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            PERSON_ID INT ,
            EXPERIENCE_ID INT,
            HIRE_DATE DATETIME ,
            CONSTRAINT FK_PERSON_EMPLOYEE_1 FOREIGN KEY (PERSON_ID) REFERENCES PERSON(ID),
            CONSTRAINT   FK_EXPERIENCE_TYPE_EMPLOYEE_1   FOREIGN   KEY   (EXPERIENCE_TYPE)
REFERENCES EXPERIENCE_TYPE(ID));

INSERT INTO EXPERIENCE_TYPE (NAME) VALUES ('Junior'); -- new
INSERT INTO EXPERIENCE_TYPE (NAME) VALUES ('Middle');-- new
INSERT INTO EXPERIENCE_TYPE (NAME) VALUES ('Senior');-- new

CREATE TABLE EXPERIENCE_TYPE (-- new
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
```

```
            NAME VARCHAR(255));

CREATE TABLE CUSTOMER (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            PERSON_ID INT ,
            EMAIL VARCHAR(255) ,
            CONSTRAINT FK_PERSON_CUSTOMER_2 FOREIGN KEY (PERSON_ID) REFERENCES PERSON(ID);

CREATE TABLE PROJECT (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            NAME VARCHAR(255) ,
            CUSTOMER_ID INT NOT NULL , -- new column
            PRICE DECIMAL(25, 2) , --changed datatype
            CONSTRAINT FK_CUSTOMER_8 FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER(ID)));
-- new fk

CREATE TABLE SALARY (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            EMPLOYEE_ID INT NOT NULL ,
            FROM_DATE DATETIME NOT NULL, -- added NOT NULL
            TO_DATE DATETIME NOT NULL,  -- added NOT NULL
            AMOUNT DECIMAL(18, 2) NOT NULL, -- added NOT NULL
            CONSTRAINT   FK_SALARY_EMPLOYEE_3   FOREIGN   KEY   (EMPLOYEE_ID)   REFERENCES
EMPLOYEE(ID));

CREATE TABLE TITLE (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            EMPLOYEE_ID INT NOT NULL ,
            FROM_DATE DATETIME NOT NULL, -- added NOT NULL
            TO_DATE DATETIME NOT NULL, -- added NOT NULL
            TITLE VARCHAR(255) ,
            CONSTRAINT   FK_TITLE_EMPLOYEE_4   FOREIGN   KEY   (EMPLOYEE_ID)   REFERENCES
EMPLOYEE(ID));

CREATE TABLE DEPARTMENT (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            DEPARTMENT_NO VARCHAR(255) ,
            NAME VARCHAR(255) );

CREATE TABLE DEPARTMENT_MANAGER (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            EMPLOYEE_ID INT ,
            DEPARTMENT_ID INT ,
            FROM_DATE DATETIME ,
            TO_DATE DATETIME ,
            CONSTRAINT   FK_DEPARTMENT_MANAGER_EMPLOYEE_5   FOREIGN   KEY   (EMPLOYEE_ID)
REFERENCES EMPLOYEE(ID),
            CONSTRAINT  FK_DEPARTMENT_MANAGER_DEPARTMENT_6  FOREIGN  KEY  (DEPARTMENT_ID)
REFERENCES DEPARTMENT(ID));

CREATE TABLE DEPARTMENT_EMPLOYEE (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            DEPARTMENT_ID INT ,
            FROM_DATE DATETIME ,
            TO_DATE DATETIME ,
            EMPLOYEE_ID INT NOT NULL , --new column
```

```
            CONSTRAINT FK_EMPLOYEE_9 FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEE(ID), -
- new fk
            CONSTRAINT  FK_DEPARTMENT_EMPLOYEE_DEPARTMENT_7  FOREIGN  KEY  (DEPARTMENT_ID)
REFERENCES DEPARTMENT(ID));


CREATE TABLE EMPLOYEE_PROJECT (
            EMPLOYEE_ID INT NOT NULL ,
            PROJECT_ID INT NOT NULL ,
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            CONSTRAINT FK_EMPLOYEE_9 FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEE(ID),
            CONSTRAINT FK_PROJECT_9 FOREIGN KEY (PROJECT_ID) REFERENCES PROJECT(ID));




---------------------- TRIGGERS  ------------------------------------------------------
CREATE OR ALTER TRIGGER EMPLOYEE_TITLE_COUNT_TRIGGER ON TITLE
AFTER INSERT
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT EMPLOYEE_ID FROM INSERTED);
            SELECT  @COUNT  =  COUNT(*)  FROM  TITLE  INNER  JOIN  EMPLOYEE  ON  EMPLOYEE_ID  =
EMPLOYEE.ID AND EMPLOYEE_ID = @NEW_ROW_ID;

            IF @COUNT > 5
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

CREATE OR ALTER TRIGGER DEPARTMENT_EMPLOYEE_EMPLOYEE_DEPARTMENT_EMPLOYEE_COUNT_TRIGGER ON
EMPLOYEE_DEPARTMENT_EMPLOYEE
AFTER INSERT
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT DEPARTMENT_EMPLOYEE_ID FROM INSERTED);
            SELECT  @COUNT  =  COUNT(*)  FROM  EMPLOYEE_DEPARTMENT_EMPLOYEE  INNER  JOIN
DEPARTMENT_EMPLOYEE    ON    DEPARTMENT_EMPLOYEE_ID    =    DEPARTMENT_EMPLOYEE.ID    AND
DEPARTMENT_EMPLOYEE_ID = @NEW_ROW_ID;

            IF @COUNT > 1
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

---------------------- INSERTS  -------------------------------------------------------
INSERT INTO PERSON (FIRST_NAME) VALUES ('Jonas');
INSERT INTO DEPARTMENT (NAME) VALUES ('First_department');
INSERT INTO DEPARTMENT (NAME) VALUES ('Second_department');

---------------------- CHECKS  --------------------------------------------------------
SELECT  *,  (SELECT  COUNT(*)  FROM  CUSTOMER_PROJECT  WHERE  PROJECT_ID  =  PROJECT.ID)  AS
CUSTOMER_PROJECT_COUNT_NOW, 1 AS CUSTOMER_PROJECT_COUNT_MUST_BE
```

```
FROM PROJECT WHERE (SELECT COUNT(*) FROM CUSTOMER_PROJECT WHERE PROJECT_ID = PROJECT.ID)
= 1;

SELECT *, (SELECT COUNT(*) FROM EMPLOYEE_PROJECT WHERE PROJECT_ID = PROJECT.ID) AS
EMPLOYEE_PROJECT_COUNT_NOW, 2 AS EMPLOYEE_PROJECT_COUNT_MUST_BE
FROM PROJECT WHERE (SELECT COUNT(*) FROM EMPLOYEE_PROJECT WHERE PROJECT_ID = PROJECT.ID)
< 2;

SELECT *, (SELECT COUNT(*) FROM TITLE WHERE EMPLOYEE_ID = EMPLOYEE.ID) AS TITLE_COUNT_NOW,
1 AS TITLE_COUNT_MUST_BE_FROM, 5 AS TITLE_COUNT_MUST_BE_TO
FROM EMPLOYEE WHERE (SELECT COUNT(*) FROM TITLE WHERE EMPLOYEE_ID = EMPLOYEE.ID) > 5 OR
(SELECT COUNT(*) FROM TITLE WHERE EMPLOYEE_ID = EMPLOYEE.ID) < 1;
```

Parametras	Kriterijus	Sutampa	Skiriasi	Atitikimas, procentais
Lentelės	Duomenų bazės lentelių skaičius	10	2	83.33
	Pirminiai raktai	8	2	80.00
	Stulpeliai	21	3	87.50
	Sinonimai	0	0	100.00
	Lentelių ar stulpelių aprašymai	0	1	100.00
Ryšiai	Ryšiai tarp lentelių, išoriniai raktai	9	2	81.82
Duomenų tipai	Skaitiniai duomenų tipai	1	1	50.00
	Datos duomenų tipai	9	0	100.00
	Tekstiniai duomenų tipai	7	3	70.00
Įvesties apribojimai	Privalomi laukai	5	4	55.56
	Galimų įvedamų reikšmių apribojimas	2	0	100.00
	Segmentavimo, kategorizavimo, individualių konceptų transformavimo sprendimas	4	1	80.00
Ryšių apribojimai	Kardinalumo minimalios reikšmės apribojimas	7	0	100.00
	Kardinalumo minimalios reikšmės patikrinimo užklausa	3	0	100.00
	Kardinalumo maksimalios reikšmės apribojimas	2	1	66.67

```
---------------------- TABLES ------------------------------------------------------------
CREATE TABLE PERSON (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            GENDER VARCHAR(10) ,--changed datatype
            FIRST_NAME VARCHAR(255) ,
            LAST_NAME VARCHAR(255) ,
            CONSTRAINT CHK_GENDER CHECK (GENDER = 'male' OR GENDER = 'female'));

--description: "Information_to_store_about_employee"
CREATE TABLE EMPLOYEE (
            PERSON_ID INT ,
            EXPERIENCE VARCHAR(15) ,--changed datatype
            HIRE_DATE DATETIME ,
            EMPLOYEE_NO VARCHAR(20) PRIMARY KEY, -- changed to primary key --changed
datatype
            CONSTRAINT CHK_EXPERIENCE CHECK (EXPERIENCE = 'junior' OR EXPERIENCE = 'middle'
OR EXPERIENCE = 'senior'),
            CONSTRAINT FK_PERSON_EMPLOYEE_1 FOREIGN KEY (PERSON_ID) REFERENCES
PERSON(ID));
```

```
CREATE TABLE CUSTOMER (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            PERSON_ID INT ,
            EMAIL VARCHAR(255) ,
            CONSTRAINT FK_PERSON_CUSTOMER_2 FOREIGN KEY (PERSON_ID) REFERENCES PERSON(ID);

CREATE TABLE PROJECT (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            NAME VARCHAR(255) ,
            CUSTOMER_ID INT NOT NULL , -- new column
            PRICE DECIMAL(25, 2) , --changed datatype
            CONSTRAINT FK_CUSTOMER_8 FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER(ID)));

CREATE TABLE SALARY (
            EMPLOYEE_ID INT NOT NULL ,
            FROM_DATE DATETIME NOT NULL, -- added NOT NULL
            TO_DATE DATETIME NOT NULL,  -- added NOT NULL
            AMOUNT DECIMAL(18, 2) NOT NULL, -- added NOT NULL
            CONSTRAINT PK_SALARY PRIMARY KEY (EMPLOYEE_ID, FROM_DATE) -- changed primary
key
            CONSTRAINT   FK_SALARY_EMPLOYEE_3   FOREIGN   KEY   (EMPLOYEE_ID)   REFERENCES
EMPLOYEE(ID));

CREATE TABLE TITLE (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            EMPLOYEE_ID INT NOT NULL ,
            FROM_DATE DATETIME NOT NULL, -- added NOT NULL
            TO_DATE DATETIME NOT NULL, -- added NOT NULL
            TITLE VARCHAR(255) ,
            CONSTRAINT   FK_TITLE_EMPLOYEE_4   FOREIGN   KEY   (EMPLOYEE_ID)   REFERENCES
EMPLOYEE(ID));

CREATE TABLE DEPARTMENT (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            DEPARTMENT_NO VARCHAR(255) ,
            NAME VARCHAR(255) );

CREATE TABLE DEPARTMENT_MANAGER (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            EMPLOYEE_ID INT ,
            DEPARTMENT_ID INT ,
            FROM_DATE DATETIME ,
            TO_DATE DATETIME ,
            CONSTRAINT   FK_DEPARTMENT_MANAGER_EMPLOYEE_5   FOREIGN   KEY   (EMPLOYEE_ID)
REFERENCES EMPLOYEE(ID),
            CONSTRAINT  FK_DEPARTMENT_MANAGER_DEPARTMENT_6  FOREIGN  KEY  (DEPARTMENT_ID)
REFERENCES DEPARTMENT(ID));

CREATE TABLE DEPARTMENT_EMPLOYEE (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            DEPARTMENT_ID INT ,
            FROM_DATE DATETIME ,
            TO_DATE DATETIME ,
            EMPLOYEE_ID INT NOT NULL , --new column
            CONSTRAINT FK_EMPLOYEE_9 FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEE(ID), -
- new fk
```

```
            CONSTRAINT FK_DEPARTMENT_EMPLOYEE_DEPARTMENT_7 FOREIGN KEY (DEPARTMENT_ID)
REFERENCES DEPARTMENT(ID));


CREATE TABLE EMPLOYEE_PROJECT (
            EMPLOYEE_ID INT NOT NULL ,
            PROJECT_ID INT NOT NULL ,
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            CONSTRAINT FK_EMPLOYEE_9 FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEE(ID),
            CONSTRAINT FK_PROJECT_9 FOREIGN KEY (PROJECT_ID) REFERENCES PROJECT(ID));


---------------------- TRIGGERS  ------------------------------------------------------
CREATE OR ALTER TRIGGER EMPLOYEE_TITLE_COUNT_TRIGGER ON TITLE
AFTER INSERT
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT EMPLOYEE_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM TITLE INNER JOIN EMPLOYEE ON EMPLOYEE_ID =
EMPLOYEE.ID AND EMPLOYEE_ID = @NEW_ROW_ID;

            IF @COUNT > 5
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

CREATE OR ALTER TRIGGER DEPARTMENT_EMPLOYEE_EMPLOYEE_DEPARTMENT_EMPLOYEE_COUNT_TRIGGER ON
EMPLOYEE_DEPARTMENT_EMPLOYEE
AFTER INSERT
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT DEPARTMENT_EMPLOYEE_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM EMPLOYEE_DEPARTMENT_EMPLOYEE INNER JOIN
DEPARTMENT_EMPLOYEE ON DEPARTMENT_EMPLOYEE_ID = DEPARTMENT_EMPLOYEE.ID AND
DEPARTMENT_EMPLOYEE_ID = @NEW_ROW_ID;

            IF @COUNT > 1
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO


---------------------- INSERTS  -------------------------------------------------------
INSERT INTO PERSON (FIRST_NAME) VALUES ('Jonas'); -- changed from NAME to FIRST_NAME
INSERT INTO DEPARTMENT (NAME) VALUES ('First_department');
INSERT INTO DEPARTMENT (NAME) VALUES ('Second_department');


---------------------- CHECKS  --------------------------------------------------------
SELECT *, (SELECT COUNT(*) FROM CUSTOMER_PROJECT WHERE PROJECT_ID = PROJECT.ID) AS
CUSTOMER_PROJECT_COUNT_NOW, 1 AS CUSTOMER_PROJECT_COUNT_MUST_BE
FROM PROJECT WHERE (SELECT COUNT(*) FROM CUSTOMER_PROJECT WHERE PROJECT_ID = PROJECT.ID)
= 1;
```

```
SELECT *, (SELECT COUNT(*) FROM EMPLOYEE_PROJECT WHERE PROJECT_ID = PROJECT.ID) AS
EMPLOYEE_PROJECT_COUNT_NOW, 2 AS EMPLOYEE_PROJECT_COUNT_MUST_BE
FROM PROJECT WHERE (SELECT COUNT(*) FROM EMPLOYEE_PROJECT WHERE PROJECT_ID = PROJECT.ID)
< 2;

SELECT *, (SELECT COUNT(*) FROM TITLE WHERE EMPLOYEE_ID = EMPLOYEE.ID) AS TITLE_COUNT_NOW,
1 AS TITLE_COUNT_MUST_BE_FROM, 5 AS TITLE_COUNT_MUST_BE_TO
FROM EMPLOYEE WHERE (SELECT COUNT(*) FROM TITLE WHERE EMPLOYEE_ID = EMPLOYEE.ID) > 5 OR
(SELECT COUNT(*) FROM TITLE WHERE EMPLOYEE_ID = EMPLOYEE.ID) < 1;
```

Parametras	Kriterijus	Sutampa	Skiriasi	Atitikimas, procentais
Lentelės	Duomenų bazės lentelių skaičius	10	3	76.92
	Pirminiai raktai	10	0	100.00
	Stulpeliai	20	4	83.33
	Sinonimai	0	0	100.00
	Lentelių ar stulpelių aprašymai	0	1	100.00
Ryšiai	Ryšiai tarp lentelių, išoriniai raktai	9	3	75.00
Duomenų tipai	Skaitiniai duomenų tipai	1	1	50.00
	Datos duomenų tipai	9	0	100.00
	Tekstiniai duomenų tipai	8	2	80.00
Įvesties apribojimai	Privalomi laukai	4	5	44.44
	Galimų įvedamų reikšmių apribojimas	2	0	100.00
	Segmentavimo, kategorizavimo, individualių konceptų transformavimo sprendimas	3	2	60.00
Ryšių apribojimai	Kardinalumo minimalios reikšmės apribojimas	7	0	100.00
	Kardinalumo minimalios reikšmės patikrinimo užklausa	3	0	100.00
	Kardinalumo maksimalios reikšmės apribojimas	2	1	66.67

Bilietų sistema

```
---------------------- TABLES ---------------------------------------------------------
CREATE TABLE ORGANIZER (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            NAME VARCHAR(255) ,
            PHONE_NO VARCHAR(12) ); -- changed datatype

CREATE TABLE EVENT (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            ORGANIZER_ID INT ,
            EVENT_VENUE_ID INT ,
            PAYMENT VARCHAR(50) , -- changed datatyp
            TITLE VARCHAR(255) ,
            DATE DATETIME ,
            EVENT_TYPE_ID INT NOT NULL ,
            CONSTRAINT CHK_PAYMENT CHECK (PAYMENT = 'free_event' OR PAYMENT =
'paid_event'),
            CONSTRAINT FK_EVENT_ORGANIZER_2 FOREIGN KEY (ORGANIZER_ID) REFERENCES
ORGANIZER(ID),
```

```
            CONSTRAINT   FK_EVENT_CATEGORY_2   FOREIGN   KEY   (EVENT_TYPE_ID)   REFERENCES
CATEGORY(ID), -- added new fk
            CONSTRAINT  FK_EVENT_EVENT_VENUE_3  FOREIGN  KEY  (EVENT_VENUE_ID)  REFERENCES
EVENT_VENUE(ID));

CREATE TABLE TICKET (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            EVENT_ID INT NOT NULL ,
            PRICE DECIMAL(18, 2) ,
            PURCHASE_DATE DATETIME ,
            OWNER_NAME VARCHAR(255),
            CONSTRAINT FK_TICKET_EVENT_1 FOREIGN KEY (EVENT_ID) REFERENCES EVENT(ID));

CREATE TABLE EVENT_VENUE (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            LOCATION_ID INT ,
            EVENT_VENUE_TYPE_ID INT ,
            CAPACITY INT ,
            CONSTRAINT CHK_CAPACITY CHECK (CAPACITY > 123), -- removed ''
            CONSTRAINT  FK_EVENT_VENUE_LOCATION_4  FOREIGN  KEY  (LOCATION_ID)  REFERENCES
LOCATION(ID),
            CONSTRAINT   FK_EVENT_VENUE_CATEGORY_7   FOREIGN   KEY   (EVENT_VENUE_TYPE_ID)
REFERENCES CATEGORY(ID));

CREATE TABLE LOCATION (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            NAME VARCHAR(255) ,
            ADDRESS VARCHAR(255) );

CREATE TABLE TEAM_MEMBER (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            ORGANIZER_ID INT NOT NULL, -- addded fk
            NAME VARCHAR(255),
            CONSTRAINT    FK_ORGANIZER_5    FOREIGN    KEY    (ORGANIZER_ID)    REFERENCES
ORGANIZER(ID)); -- added fk

CREATE TABLE SELLER ( -- add Description "A place, where you can buy tickets"
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            SELLER_NAME VARCHAR(255));
CREATE SYNONYM TICKET_SELLER FOR SELLER;

CREATE TABLE SELLER_TICKET (
            SELLER_ID INT NOT NULL ,
            TICKET_ID INT NOT NULL ,
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            CONSTRAINT FK_SELLER_6 FOREIGN KEY (SELLER_ID) REFERENCES SELLER(ID),
            CONSTRAINT FK_TICKET_6 FOREIGN KEY (TICKET_ID) REFERENCES TICKET(ID));

CREATE TABLE CATEGORY (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            NAME VARCHAR(255) );


---------------------- TRIGGERS  ------------------------------------------------------
CREATE OR ALTER TRIGGER EVENT_TICKET_COUNT_TRIGGER ON TICKET
AFTER INSERT
AS
```

```
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT EVENT_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM TICKET INNER JOIN EVENT ON EVENT_ID = EVENT.ID
AND EVENT_ID = @NEW_ROW_ID;

            IF @COUNT > 1000
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

CREATE    OR    ALTER    TRIGGER    ORGANIZER_ORGANIZER_TEAM_MEMBER_COUNT_TRIGGER    ON
ORGANIZER_TEAM_MEMBER
AFTER INSERT
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT ORGANIZER_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM ORGANIZER_TEAM_MEMBER INNER JOIN ORGANIZER ON
ORGANIZER_ID = ORGANIZER.ID AND ORGANIZER_ID = @NEW_ROW_ID;

            IF @COUNT > 7
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO



---------------------- INSERTS  ----------------------------------------------------------
INSERT INTO CATEGORY (NAME) VALUES ('sports_arena');
INSERT INTO CATEGORY (NAME) VALUES ('park');
INSERT INTO CATEGORY (NAME) VALUES ('Concert'); -- changed from EVENT to CATEGORY
INSERT INTO CATEGORY (NAME) VALUES ('Exhibition');-- changed from EVENT to CATEGORY
INSERT INTO CATEGORY (NAME) VALUES ('Lecture');-- changed from EVENT to CATEGORY
INSERT INTO CATEGORY (NAME) VALUES ('Sport');-- changed from EVENT to CATEGORY
INSERT INTO SELLER (SELLER_NAME) VALUES ('Tiketa');-- changed from NAME to SELLER_NAME

---------------------- CHECKS  -----------------------------------------------------------
SELECT   *,   (SELECT   COUNT(*)   FROM   ORGANIZER_TEAM_MEMBER   WHERE   TEAM_MEMBER_ID   =
TEAM_MEMBER.ID)         AS         ORGANIZER_TEAM_MEMBER_COUNT_NOW,         1         AS
ORGANIZER_TEAM_MEMBER_COUNT_MUST_BE
FROM TEAM_MEMBER WHERE (SELECT COUNT(*) FROM ORGANIZER_TEAM_MEMBER WHERE TEAM_MEMBER_ID =
TEAM_MEMBER.ID) = 1;

SELECT  *,  (SELECT  COUNT(*)  FROM  SELLER_TICKET  WHERE  SELLER_ID  =  SELLER.ID)  AS
SELLER_TICKET_COUNT_NOW, 10 AS SELLER_TICKET_COUNT_MUST_BE
FROM SELLER WHERE (SELECT COUNT(*) FROM SELLER_TICKET WHERE SELLER_ID = SELLER.ID) < 10;

SELECT *, (SELECT COUNT(*) FROM ORGANIZER_TEAM_MEMBER WHERE ORGANIZER_ID = ORGANIZER.ID)
AS ORGANIZER_TEAM_MEMBER_COUNT_NOW, 2 AS ORGANIZER_TEAM_MEMBER_COUNT_MUST_BE_FROM, 7 AS
ORGANIZER_TEAM_MEMBER_COUNT_MUST_BE_TO
FROM ORGANIZER WHERE (SELECT COUNT(*) FROM ORGANIZER_TEAM_MEMBER WHERE ORGANIZER_ID =
ORGANIZER.ID) > 7 OR (SELECT COUNT(*) FROM ORGANIZER_TEAM_MEMBER WHERE ORGANIZER_ID =
ORGANIZER.ID) < 2;
```

Parametras	Kriterijus	Sutampa	Skiriasi	Atitikimas, procentais
Lentelės	Duomenų bazės lentelių skaičius	9	1	90.00
	Pirminiai raktai	9	0	100.00
	Stulpeliai	15	2	88.24
	Sinonimai	1	0	100.00
	Lentelių ar stulpelių aprašymai	0	1	100.00
Ryšiai	Ryšiai tarp lentelių, išoriniai raktai	7	2	77.78
Duomenų tipai	Skaitiniai duomenų tipai	2	2	50.00
	Datos duomenų tipai	2	2	50.00
	Tekstiniai duomenų tipai	7	2	77.78
Įvesties apribojimai	Privalomi laukai	1	0	100.00
	Galimų įvedamų reikšmių apribojimas	1	0	100.00
	Segmentavimo, kategorizavimo, individualių konceptų transformavimo sprendimas	1	1	50.00
Ryšių apribojimai	Kardinalumo minimalios reikšmės apribojimas	4	1	80.00
	Kardinalumo minimalios reikšmės patikrinimo užklausa	3	0	100.00
	Kardinalumo maksimalios reikšmės apribojimas	2	0	100.00

```
---------------------- TABLES ---------------------------------------------------------
CREATE TABLE ORGANIZER (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            NAME VARCHAR(255) ,
            PHONE_NO VARCHAR(12) ); -- changed datatype

CREATE TABLE EVENT (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            ORGANIZER_ID INT ,
            EVENT_VENUE_ID INT ,
            IS_FREE BIT,
            TITLE VARCHAR(255) ,
            DATE DATETIME ,
            EVENT_TYPE_ID INT NOT NULL ,
            CONSTRAINT  FK_EVENT_ORGANIZER_2  FOREIGN  KEY  (ORGANIZER_ID)  REFERENCES
ORGANIZER(ID),
            CONSTRAINT  FK_EVENT_CATEGORY_2  FOREIGN  KEY  (EVENT_TYPE_ID)  REFERENCES
EVENT_TYPE(ID),
            CONSTRAINT  FK_EVENT_EVENT_VENUE_3  FOREIGN  KEY  (EVENT_VENUE_ID)  REFERENCES
EVENT_VENUE(ID));

CREATE TABLE TICKET (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            EVENT_ID INT NOT NULL ,
            PRICE DECIMAL(18, 2) ,
            PURCHASE_DATE DATETIME ,
            OWNER_NAME VARCHAR(255),
            CONSTRAINT FK_TICKET_EVENT_1 FOREIGN KEY (EVENT_ID) REFERENCES EVENT(ID));

CREATE TABLE EVENT_VENUE (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            LOCATION_ID INT ,
            EVENT_VENUE_TYPE_ID INT ,
```

```
            CAPACITY INT ,
            CONSTRAINT CHK_CAPACITY CHECK (CAPACITY > 123), -- removed ''
            CONSTRAINT  FK_EVENT_VENUE_LOCATION_4  FOREIGN  KEY  (LOCATION_ID)  REFERENCES
LOCATION(ID),
            CONSTRAINT        FK_EVENT_VENUE_EVENT_VENUE_TYPE_7        FOREIGN        KEY
(EVENT_VENUE_TYPE_ID) REFERENCES EVENT_VENUE_TYPE(ID));

CREATE TABLE LOCATION (
            NAME VARCHAR(255) PRIMARY KEY,
            ADDRESS VARCHAR(255) );

CREATE TABLE TEAM_MEMBER (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            ORGANIZER_ID INT NOT NULL, -
            NAME VARCHAR(255),
            CONSTRAINT    FK_ORGANIZER_5    FOREIGN    KEY    (ORGANIZER_ID)    REFERENCES
ORGANIZER(ID));

CREATE TABLE SELLER ( -- add Description "A place, where you can buy tickets"
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            SELLER_NAME VARCHAR(255));
CREATE SYNONYM TICKET_SELLER FOR SELLER;

CREATE TABLE SELLER_TICKET (
            SELLER_ID INT NOT NULL ,
            TICKET_ID INT NOT NULL ,
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            CONSTRAINT FK_SELLER_6 FOREIGN KEY (SELLER_ID) REFERENCES SELLER(ID),
            CONSTRAINT FK_TICKET_6 FOREIGN KEY (TICKET_ID) REFERENCES TICKET(ID));

CREATE TABLE EVENT_TYPE (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            NAME VARCHAR(255) );

CREATE TABLE EVENT_VENUE_TYPE (
            ID INT IDENTITY(1,1) PRIMARY KEY ,
            NAME VARCHAR(255) );


---------------------- TRIGGERS  ------------------------------------------------------
CREATE OR ALTER TRIGGER EVENT_TICKET_COUNT_TRIGGER ON TICKET
AFTER INSERT
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT EVENT_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM TICKET INNER JOIN EVENT ON EVENT_ID = EVENT.ID
AND EVENT_ID = @NEW_ROW_ID;

            IF @COUNT > 1000
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

CREATE    OR    ALTER    TRIGGER    ORGANIZER_ORGANIZER_TEAM_MEMBER_COUNT_TRIGGER    ON
ORGANIZER_TEAM_MEMBER
```

```
AFTER INSERT
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT ORGANIZER_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM ORGANIZER_TEAM_MEMBER INNER JOIN ORGANIZER ON
ORGANIZER_ID = ORGANIZER.ID AND ORGANIZER_ID = @NEW_ROW_ID;

            IF @COUNT > 7
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

---------------------- INSERTS  ------------------------------------------------------------
INSERT INTO EVENT_VENUE_TYPE (NAME) VALUES ('Sports rena');
INSERT INTO EVENT_VENUE_TYPE (NAME) VALUES ('Park');
INSERT INTO EVENT_TYPE (NAME) VALUES ('Concert'); -- changed from EVENT to EVENT_TYPE
INSERT INTO EVENT_TYPE (NAME) VALUES ('Exhibition');-- changed from EVENT to EVENT_TYPE
INSERT INTO EVENT_TYPE (NAME) VALUES ('Lecture');-- changed from EVENT to EVENT_TYPE
INSERT INTO EVENT_TYPE (NAME) VALUES ('Sport');-- changed from EVENT to EVENT_TYPE
INSERT INTO SELLER (SELLER_NAME) VALUES ('Tiketa');-- changed from NAME to SELLER_NAME

---------------------- CHECKS  -------------------------------------------------------------
SELECT  *,  (SELECT  COUNT(*)  FROM  ORGANIZER_TEAM_MEMBER  WHERE  TEAM_MEMBER_ID  =
TEAM_MEMBER.ID)         AS         ORGANIZER_TEAM_MEMBER_COUNT_NOW,         1         AS
ORGANIZER_TEAM_MEMBER_COUNT_MUST_BE
FROM TEAM_MEMBER WHERE (SELECT COUNT(*) FROM ORGANIZER_TEAM_MEMBER WHERE TEAM_MEMBER_ID =
TEAM_MEMBER.ID) = 1;

SELECT  *,  (SELECT  COUNT(*)  FROM  SELLER_TICKET  WHERE  SELLER_ID  =  SELLER.ID)  AS
SELLER_TICKET_COUNT_NOW, 10 AS SELLER_TICKET_COUNT_MUST_BE
FROM SELLER WHERE (SELECT COUNT(*) FROM SELLER_TICKET WHERE SELLER_ID = SELLER.ID) < 10;

SELECT *, (SELECT COUNT(*) FROM ORGANIZER_TEAM_MEMBER WHERE ORGANIZER_ID = ORGANIZER.ID)
AS ORGANIZER_TEAM_MEMBER_COUNT_NOW, 2 AS ORGANIZER_TEAM_MEMBER_COUNT_MUST_BE_FROM, 7 AS
ORGANIZER_TEAM_MEMBER_COUNT_MUST_BE_TO
FROM ORGANIZER WHERE (SELECT COUNT(*) FROM ORGANIZER_TEAM_MEMBER WHERE ORGANIZER_ID =
ORGANIZER.ID) > 7 OR (SELECT COUNT(*) FROM ORGANIZER_TEAM_MEMBER WHERE ORGANIZER_ID =
ORGANIZER.ID) < 2;
```

Parametras	Kriterijus	Sutampa	Skiriasi	Atitikimas, procentais
Lentelės	Duomenų bazės lentelių skaičius	8	3	72.73
	Pirminiai raktai	7	4	63.64
	Stulpeliai	13	4	76.47
	Sinonimai	1	0	100.00
	Lentelių ar stulpelių aprašymai	0	1	100.00
Ryšiai	Ryšiai tarp lentelių, išoriniai raktai	6	4	60.00
Duomenų tipai	Skaitiniai duomenų tipai	6	3	66.67
	Datos duomenų tipai	2	0	100.00
	Tekstiniai duomenų tipai	2	2	50.00
	Privalomi laukai	7	2	77.78

Įvesties apribojimai	Galimų įvedamų reikšmių apribojimas	1	0	100.00
	Segmentavimo, kategorizavimo, individualių konceptų transformavimo sprendimas	0	2	100.00
Ryšių apribojimai	Kardinalumo minimalios reikšmės apribojimas	4	1	80.00
	Kardinalumo minimalios reikšmės patikrinimo užklausa	3	0	100.00
	Kardinalumo maksimalios reikšmės apribojimas	2	0	100.00

Knygų sistema

Parametras	Kriterijus	Sutampa	Skiriasi	Atitikimas, procentais
Lentelės	Duomenų bazės lentelių skaičius	7	0	100.00
	Pirminiai raktai	7	0	100.00
	Stulpeliai	7	2	77.78
	Sinonimai	1	0	100.00
	Lentelių ar stulpelių aprašymai	0	1	100.00
Ryšiai	Ryšiai tarp lentelių, išoriniai raktai	6	0	100.00
Duomenų tipai	Skaitiniai duomenų tipai	2	2	50.00
	Datos duomenų tipai	0	0	100.00
	Tekstiniai duomenų tipai	5	2	71.43
Įvesties apribojimai	Privalomi laukai	1	1	50.00
	Galimų įvedamų reikšmių apribojimas	1	1	50.00
	Segmentavimo, kategorizavimo, individualių konceptų transformavimo sprendimas	1	1	50.00
Ryšių apribojimai	Kardinalumo minimalios reikšmės apribojimas	5	0	100.00
	Kardinalumo minimalios reikšmės patikrinimo užklausa	3	0	100.00
	Kardinalumo maksimalios reikšmės apribojimas	2	0	100.00

```
---------------------- TABLES ---------------------------------------------------------
CREATE TABLE AUTHOR (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            AUTHOR_NAME VARCHAR(255) ,
            NAME VARCHAR(MAX) ); -- changed datatype
CREATE SYNONYM AUTHOR_OF_A_BOOK FOR AUTHOR;

CREATE TABLE BOOK (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            PRICE_TYPE VARCHAR(50), -- changed datatype
            GENRE_TYPE_ID INT ,
            PAGES INT , -- descr:  "Number of pages in a book"
            TITLE VARCHAR(255) NOT NULL ,
            CONSTRAINT  CHK_PRICE_TYPE  CHECK  (PRICE_TYPE  =  'cheap'  OR  PRICE_TYPE  =
'expensive'),
            CONSTRAINT FK_BOOK_GENRE_4 FOREIGN KEY (GENRE_TYPE_ID) REFERENCES GENRE(ID));

CREATE TABLE CHAPTER (
```

```
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            BOOK_ID INT NOT NULL ,
            TITLE VARCHAR(255) ,
            WORDS INT ,
            CONSTRAINT FK_CHAPTER_BOOK_1 FOREIGN KEY (BOOK_ID) REFERENCES BOOK(ID));

CREATE TABLE PUBLISHER (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            NAME VARCHAR(255) );

CREATE TABLE BOOK_AUTHOR (
            BOOK_ID INT NOT NULL ,
            AUTHOR_ID INT NOT NULL ,
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            CONSTRAINT FK_BOOK_2 FOREIGN KEY (BOOK_ID) REFERENCES BOOK(ID),
            CONSTRAINT FK_AUTHOR_2 FOREIGN KEY (AUTHOR_ID) REFERENCES AUTHOR(ID));

CREATE TABLE BOOK_PUBLISHER (
            BOOK_ID INT NOT NULL ,
            PUBLISHER_ID INT NOT NULL ,
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            CONSTRAINT FK_BOOK_3 FOREIGN KEY (BOOK_ID) REFERENCES BOOK(ID),
            CONSTRAINT    FK_PUBLISHER_3    FOREIGN    KEY    (PUBLISHER_ID)    REFERENCES
PUBLISHER(ID));

CREATE TABLE GENRE (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            NAME VARCHAR(255) );

---------------------- INSERTS  ---------------------------------------------------------
INSERT INTO GENRE (NAME) VALUES ('art'); -- from CATEGORY to GENRE
INSERT INTO GENRE (NAME) VALUES ('autobiography');-- from CATEGORY to GENRE
INSERT INTO GENRE (NAME) VALUES ('cookbook');-- from CATEGORY to GENRE
INSERT INTO GENRE (NAME) VALUES ('science');-- from CATEGORY to GENRE
INSERT INTO GENRE (NAME) VALUES ('romance');-- from CATEGORY to GENRE
INSERT INTO AUTHOR (NAME) VALUES ('Ayn_Rand');
INSERT INTO AUTHOR (NAME) VALUES ('Ernest_Hemingway');
INSERT INTO BOOK (TITLE) VALUES ('The_Hunger_Games'); -- changed from name to title

---------------------- TRIGGERS  ---------------------------------------------------------
CREATE OR ALTER TRIGGER BOOK_BOOK_AUTHOR_COUNT_TRIGGER ON BOOK_AUTHOR
AFTER INSERT
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT BOOK_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM BOOK_AUTHOR INNER JOIN BOOK ON BOOK_ID = BOOK.ID
AND BOOK_ID = @NEW_ROW_ID;

            IF @COUNT > 2
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

CREATE OR ALTER TRIGGER BOOK_CHAPTER_COUNT_TRIGGER ON CHAPTER
AFTER INSERT
```

```
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT BOOK_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM CHAPTER INNER JOIN BOOK ON BOOK_ID = BOOK.ID AND
BOOK_ID = @NEW_ROW_ID;

            IF @COUNT > 9
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

---------------------- CHECKS ------------------------------------------------------------
SELECT *, (SELECT COUNT(*) FROM BOOK_AUTHOR WHERE AUTHOR_ID = AUTHOR.ID) AS
BOOK_AUTHOR_COUNT_NOW, 7 AS BOOK_AUTHOR_COUNT_MUST_BE
FROM AUTHOR WHERE (SELECT COUNT(*) FROM BOOK_AUTHOR WHERE AUTHOR_ID = AUTHOR.ID) < 7;

SELECT *, (SELECT COUNT(*) FROM BOOK_AUTHOR WHERE BOOK_ID = BOOK.ID) AS
BOOK_AUTHOR_COUNT_NOW, 2 AS BOOK_AUTHOR_COUNT_MUST_BE
FROM BOOK WHERE (SELECT COUNT(*) FROM BOOK_AUTHOR WHERE BOOK_ID = BOOK.ID) = 2;

SELECT *, (SELECT COUNT(*) FROM CHAPTER WHERE BOOK_ID = BOOK.ID) AS CHAPTER_COUNT_NOW, 5
AS CHAPTER_COUNT_MUST_BE_FROM, 9 AS CHAPTER_COUNT_MUST_BE_TO
FROM BOOK WHERE (SELECT COUNT(*) FROM CHAPTER WHERE BOOK_ID = BOOK.ID) > 9 OR (SELECT
COUNT(*) FROM CHAPTER WHERE BOOK_ID = BOOK.ID) < 5;
```

Parametras	Kriterijus	Sutampa	Skiriasi	Atitikimas, procentais
Lentelės	Duomenų bazės lentelių skaičius	6	1	85.71
	Pirminiai raktai	6	1	85.71
	Stulpeliai	7	2	77.78
	Sinonimai	1	0	100.00
	Lentelių ar stulpelių aprašymai	0	1	100.00
Ryšiai	Ryšiai tarp lentelių, išoriniai raktai	5	1	83.33
Duomenų tipai	Skaitiniai duomenų tipai	2	2	50.00
	Datos duomenų tipai	0	0	100.00
	Tekstiniai duomenų tipai	5	2	71.43
Įvesties apribojimai	Privalomi laukai	1	1	50.00
	Galimų įvedamų reikšmių apribojimas	1	1	50.00
	Segmentavimo, kategorizavimo, individualių konceptų transformavimo sprendimas	1	1	50.00
Ryšių apribojimai	Kardinalumo minimalios reikšmės apribojimas	5	0	100.00
	Kardinalumo minimalios reikšmės patikrinimo užklausa	3	0	100.00
	Kardinalumo maksimalios reikšmės apribojimas	2	0	100.00

```
---------------------- TABLES ------------------------------------------------------------
CREATE TABLE AUTHOR (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            AUTHOR_NAME VARCHAR(255) ,
```

```
            NAME VARCHAR(MAX) );
CREATE SYNONYM AUTHOR_OF_A_BOOK FOR AUTHOR;

CREATE TABLE BOOK (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            PRICE_TYPE VARCHAR(50),
            GENRE_TYPE_ID INT ,
            PAGES INT , -- descr:  "Number of pages in a book"
            TITLE VARCHAR(255) NOT NULL ,
            CONSTRAINT  CHK_PRICE_TYPE  CHECK  (PRICE_TYPE  =  'cheap'  OR  PRICE_TYPE  =
'expensive'),
            CONSTRAINT FK_BOOK_GENRE_4 FOREIGN KEY (GENRE_TYPE_ID) REFERENCES GENRE(ID));

CREATE TABLE CHAPTER (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            BOOK_ID INT NOT NULL ,
            TITLE VARCHAR(255) ,
            WORDS INT ,
            CONSTRAINT FK_CHAPTER_BOOK_1 FOREIGN KEY (BOOK_ID) REFERENCES BOOK(ID));

CREATE TABLE PUBLISHER (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            NAME VARCHAR(255) );

CREATE TABLE BOOK_AUTHOR (
            BOOK_ID INT NOT NULL ,
            AUTHOR_ID INT NOT NULL ,
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            CONSTRAINT FK_BOOK_2 FOREIGN KEY (BOOK_ID) REFERENCES BOOK(ID),
            CONSTRAINT FK_AUTHOR_2 FOREIGN KEY (AUTHOR_ID) REFERENCES AUTHOR(ID));

CREATE TABLE BOOK_PUBLISHER (
            BOOK_ID INT NOT NULL ,
            PUBLISHER_ID INT NOT NULL ,
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            CONSTRAINT FK_BOOK_3 FOREIGN KEY (BOOK_ID) REFERENCES BOOK(ID),
            CONSTRAINT    FK_PUBLISHER_3    FOREIGN    KEY    (PUBLISHER_ID)    REFERENCES
PUBLISHER(ID));

CREATE TABLE CATEGORY (
            ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
            NAME VARCHAR(255) );

---------------------- INSERTS  ------------------------------------------------------
INSERT INTO GENRE (NAME) VALUES ('art');
INSERT INTO GENRE (NAME) VALUES ('autobiography');
INSERT INTO GENRE (NAME) VALUES ('cookbook');
INSERT INTO GENRE (NAME) VALUES ('science');
INSERT INTO GENRE (NAME) VALUES ('romance');
INSERT INTO AUTHOR (NAME) VALUES ('Ayn_Rand');
INSERT INTO AUTHOR (NAME) VALUES ('Ernest_Hemingway');
INSERT INTO BOOK (TITLE) VALUES ('The_Hunger_Games');

---------------------- TRIGGERS  ------------------------------------------------------
CREATE OR ALTER TRIGGER BOOK_BOOK_AUTHOR_COUNT_TRIGGER ON BOOK_AUTHOR
AFTER INSERT
AS
```

```
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT BOOK_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM BOOK_AUTHOR INNER JOIN BOOK ON BOOK_ID = BOOK.ID
AND BOOK_ID = @NEW_ROW_ID;

            IF @COUNT > 2
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

CREATE OR ALTER TRIGGER BOOK_CHAPTER_COUNT_TRIGGER ON CHAPTER
AFTER INSERT
AS
            DECLARE @COUNT int;
            DECLARE @NEW_ROW_ID int = (SELECT BOOK_ID FROM INSERTED);
            SELECT @COUNT = COUNT(*) FROM CHAPTER INNER JOIN BOOK ON BOOK_ID = BOOK.ID AND
BOOK_ID = @NEW_ROW_ID;

            IF @COUNT > 9
            BEGIN
                rollback
                RAISERROR ('Violated business rules', 16, 1);
            END
GO

---------------------- CHECKS  ------------------------------------------------------
SELECT  *,  (SELECT  COUNT(*)  FROM  BOOK_AUTHOR  WHERE  AUTHOR_ID  =  AUTHOR.ID)  AS
BOOK_AUTHOR_COUNT_NOW, 7 AS BOOK_AUTHOR_COUNT_MUST_BE
FROM AUTHOR WHERE (SELECT COUNT(*) FROM BOOK_AUTHOR WHERE AUTHOR_ID = AUTHOR.ID) < 7;

SELECT  *,  (SELECT  COUNT(*)  FROM  BOOK_AUTHOR  WHERE  BOOK_ID  =  BOOK.ID)  AS
BOOK_AUTHOR_COUNT_NOW, 2 AS BOOK_AUTHOR_COUNT_MUST_BE
FROM BOOK WHERE (SELECT COUNT(*) FROM BOOK_AUTHOR WHERE BOOK_ID = BOOK.ID) = 2;

SELECT *, (SELECT COUNT(*) FROM CHAPTER WHERE BOOK_ID = BOOK.ID) AS CHAPTER_COUNT_NOW, 5
AS CHAPTER_COUNT_MUST_BE_FROM, 9 AS CHAPTER_COUNT_MUST_BE_TO
FROM BOOK WHERE (SELECT COUNT(*) FROM CHAPTER WHERE BOOK_ID = BOOK.ID) > 9 OR (SELECT
COUNT(*) FROM CHAPTER WHERE BOOK_ID = BOOK.ID) < 5;
```