VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Duomenų transformacijų vykdymas naudojant dalykinės srities kalbą

Execution of Data Transformations Using Domain Specific Language

Magistro baigiamasis darbas

Atliko:	Mantas Gedrimas	(parašas)
Darbo vadovas:	partn. doc. Andrius Adamonis	(parašas)
Darbo recenzentas:	partn. doc. Vaidas Jusevičius	(parašas)

Vilnius – 2018

SANTRAUKA

Šiame magistro baigiamajame darbe nagrinėjamas dalykinės srities kalbos naudojimas duomenų transformacijoms reliacinėse SQL duomenų bazėse atlikti. Išanalizuota dalykinė sritis: įvardinti ir apibūdinti jos komponentai, aprašyti vykdomi procesai, išnagrinėta, kaip sprendžiami ETL uždaviniai. Išskirti įrankiai, kurie galėtų palengvinti DSL kūrimą – analizatoriai ir ETL bibliotekos. Remiantis išskirtomis dalykinės srities esybėmis, sudarytas reikalavimų rinkinys ir pagal jį sukonstruota kalba – Relational DataRules. Nusakyta kalbos gramatika, sintaksė, apibrėžtos naudojimo taisyklės, pateikti konstrukcijų pavyzdžiai. Įgyvendintas dalykinės srities kalbos vykdymo modelis, t. y. aprašyti algoritmai, kaip turėtų būti vykdomos transformacijos. Taip pat sukurtas prototipas, gebantis vykdyti Relational DataRules aprašytas transformacijas.

Raktiniai žodžiai: dalykinės srities kalba, DSL, ETL, duomenų transformacijos, duomenų valymas, duomenų validavimas, kalbos vykdymo modelis.

SUMMARY

In this paper Domain specific language usage for data transformations in relational SQL databases is investigated. Mentioned domain is analysed – main components are identified and described alongside with necessary processes. ETL tasks and execution is analysed. Possible tools which would benefit in creating the language are also identified and anlysed including analizators and ETL libraries. Based on identified entities of domain the requirements of a domain specific language are created and based on it a language – Relational DataRules – is constructed. Language's grammar, syntax, ruleset and usage examples are provided. Domain specific language execution model is created by providing algorithms of how data transformations should be executed. Also, a prototype is created which executes Relational DataRules transformations.

Keywords: domain specific language, DSL, ETL, data transformation, data cleaning, data validation, language execution model.

TURINYS

ĮVADAS

Duomenų transformavimas, valymas ir klaidų paieška – uždavinys, su kuriuo kasdien susiduria programų sistemų inžinieriai. Duomenis reikia perkelti iš vienos saugyklos į kitą, o dėl skirtingų duomenų bazių, gali prireikti vykdyti transformacijas. Taip pat dėl to atsiranda įvairios duomenų anomalijos, kurias reikia sutvarkyti ir išvalyti. Pasidomėjus, kokie galimi šios problemos sprendimai, galima pastebėti, kad pasitelkiami įvairiausi metodai – programuotojų rašomos programos, ETL (angl. „Extract, Transform, Load“) įrankiai, grįsti tiek grafinės sąsajos naudojimu (pvz. „Informatica“ [I18]), tiek ir programinio kodo rašymu (pvz. „petl“ [P17]).

Šiame magistro darbe nagrinėjama dalykinė sritis – duomenų transformacijos, valymas ir klaidų paieška (toliau visi veiksmai apibendrintai vadinami tiesiog duomenų transformacijomis) reliacinėse SQL duomenų bazėse. Šios dalykinės srities uždaviniams įgyvendinti pasirinktas sprendimas kitoks, nei prieš tai minėti sprendimai – naudoti dalykinės srities kalbą (sutr. DSL), kuri būtų interpretuojama ir tiktų spręsti transformavimo, klaidų paieškos ir duomenų valymo uždavinius reliacinėse SQL duomenų bazėse saugomiems duomenims.

Duomenų transformacijų atlikimas naudojant DSL turi savų privalumų lyginant su minėtais analogais (remiantis [DS15], [Hud97], [Tub15], [HK14]):

- Kalbinės išraiškos yra ekspresyvesnės nei bendros paskirties ar vidinės dalykinės srities kalbų, todėl tai padeda sutrumpinti tokių programų rašymą – tai daryti paprasčiau bei pačios programos trumpesnės. Remiantis [Her07] minimu „Boilio dėsniu dalykinės srities kalboms“ (angl. „Boyle’s law for DSLs”), kuo kalba mažesnė, tuo ji gali būti ekspresyvesnė, o išorinė DSL maža, kadangi apima tik dalykinei sričiai reikalingas konstrukcijas;
- Aprašytas transformacijas galima pakartotinai panaudoti daug kartų, kai, pavyzdžiui, naudojant grafinės sąsajos įrankius to padaryti apskritai nepavyktų. Dalykinės srities kalba nusakomos dalykinės srities žinios, kurios vėliau gali būti pakartotinai panaudotos;
- Mažesni DSL parašytų programų palaikymo kaštai, lyginant su bendros paskirties kalbomis parašytų programų palaikymu. DSL išraiškos deklaratyvios ir pačios save specifikuoja, programos paprastesnės, todėl ir palaikymas tampa paprastesnis ir pigesnis;
- Programas dalykinės srities kalba galėtų rašyti (arba bent jau skaityti ir suprasti, ką jos daro) ir žmonės, kurie neturi daug patirties programavime, tačiau išmano dalykinę sritį ir, šiuo atveju, tos srities kalbą (t. y. duomenų transformacijų reliacinėse duomenų bazėse DSL);
- Tokios kalbos naudojimas galėtų būti patogus žmonėms, įpratusiems rašyti programinį kodą, vietoje grafinę sąsają turinčių ETL įrankių naudojimo.

Nagrinėjant prieinamus literatūros šaltinius, nebuvo atrasta jokia pilnai išbaigta kalba, kuri spręstų visus tris pagrindinius šios dalykinės srities uždavinius. Artimiausia šiai sričiai aprašyta DSL – DataRules [Tub15]. Išnagrinėjus jos savybes, nustatyta, kad kalbos dalykinės sritis panaši (DataRules skirta struktūrizuotų duomenų transformacijoms ir validavimui aprašyti), konstrukcijos ekspresyvios, tačiau visiškai neapibrėžtas jos vykdymo modelis. Dėl dalykinės srities panašumo nuspręsta naujos kalbos nekurti, o sudaryti kriterijų rinkinį ir kalbą modifikuoti pritaikant šio magistro darbo dalykinei sričiai. Taip pat, nusakytas tokios kalbos vykdymo modelis – t. y. pseudokodu ir tekstu nusakyti algoritmai, kurie turėtų būti atliekami vykdant aprašytas transformacijas. Galiausiai sukurtas prototipas, gebantis vykdyti sukurtos kalbos konstrukcijas pagal apibrėžtą vykdymo modelį. Kuriama DSL pavadinta Relational DataRules.

Verta pastebėti, kad apibrėžiant dalykinės srities kalbos vykdymo modelį, buvo siekiama, kad jis būtų nepriklausomas nuo konkrečios (šiuo atveju Relational DataRules) DSL gramatikos ar sintaksės. Dėl to buvo sukurtas toks dalykinės srities kalbos vykdymo modelis, kuris yra tinkamas bet kokiai DSL, kuri tenkina sudarytus reikalavimus kalbai ir turi visas konstrukcijas, kuriomis būtų galima aprašyti transformacijos – priklausomai nuo kalbos reikėtų modifikuoti tik pačių konstrukcijų atpažinimą ir jas susieti su atitinkamais algoritmais.

Pagrindinis magistro baigiamojo darbo tikslas – sukurti duomenų transformavimo dalykinės srities kalbos vykdymo modelį, nusakant algoritmus, apibrėžiančius duomenų transformacijas ir jų vykdymą.

Tikslui pasiekti sudaryti uždaviniai:

1. Ištirti duomenų transformacijų, klaidų paieškos ir duomenų valymo dalykinės srities ypatumus ir išsiaiškinti, kokias savybes privalo turėti šios dalykinės srities uždaviniams spręsti naudojama DSL.

2. Sukonstruoti dalykinės srities kalbą pagal sudarytą reikalavimų rinkinį.

3. Apibrėžti DSL vykdymo modelį nusakant algoritmus, kaip turėtų būti vykdoma DSL parašyta programa.

4. Sukurti prototipą (programą), gebantį vykdyti dalykinės srities kalba aprašytas transformacijas pagal apibrėžtą DSL vykdymo modelį.

1. DALYKINĖ SRITIS

Šiame magistro baigiamajame darbe nagrinėjamas dalykinės srities kalbos naudojimas duomenų transformacijoms vykdyti, todėl pirmiausia reikia tiksliai apsibrėžti, kas yra nagrinėjama dalykinė sritis ir kokia ji. Šiame darbe, trumpai tariant – tai duomenų transformacijos reliacinėse duomenų bazėse. Norint geriau suprasti, kas tai yra, reikėtų įsigilinti, ką konkrečiai ši sritis apima.

Pirmiausia, reikia pabrėžti, kad kalbama apie duomenis, kurie saugomi reliacinėse SQL duomenų bazėse. Tai duomenų bazės, kurios apibrėžiamos naudojant reliacinį modelį, tai yra, jose duomenys saugomi lentelėse, sudarytose iš eilučių ir stulpelių. Tai reiškia, kad tie duomenys yra struktūrizuoti, o duomenų bazė turi apibrėžtą schemą. Jų duomenys valdomi pasitelkiant reliacinių duomenų bazių valdymo sistemas (RDBVS), rašant SQL (angl. „Structured Query Language“) užklausas. Taip pat, tokių sistemų skiriamasis bruožas – stabilių ACID transakcijų palaikymas – tai reiškia, kad duomenys jose visada darnūs. Šios sistemos šiuo metu yra plačiausiai naudojamos, lyginant su kitomis DBVS – remiantis db-engines.com populiarumo reitingu [DBE18], net pirmas 4 vietas užima RDBVS.

Kalbant apie pačias duomenų transformacijas, būtų galima išskirti tris aspektus, kurie apibūdina šiame darbe nagrinėjamą dalykinę sritį. Tai duomenų transformavimas, valymas ir klaidų paieška.

Be pačių transformacijų vykdymo, dalykinė sritis apima ir technines veiklas: duomenų ištraukimą iš šaltinių duomenų bazių, ir įkėlimą tiek į tarpinę aplinką, kur gali būti vykdomos transformacijos, tiek į adresatų duomenų bazes. Dar kitaip visas šis procesas vadinamas ETL procesu.

1.1. Duomenų transformavimas

Remiantis [T17], duomenų transformavimas yra duomenų ar informacijos konvertavimo procesas, kurio metu šie duomenys iš formato, kuriuo jie laikomi esančioje saugykloje, verčiami į formatą, kuriuo bus saugomi kitoje saugykloje. Tai galima iliustruoti keliais pavyzdžiais:

1. Tarkime, lentelės stulpelyje saugomi duomenys su datomis, kurios užrašytos JAV naudojamu datos formatu (mm.dd.yyyy) ir norime juos transformuoti į Lietuvoje naudojamą datos formatą (yyyy-mm-dd).

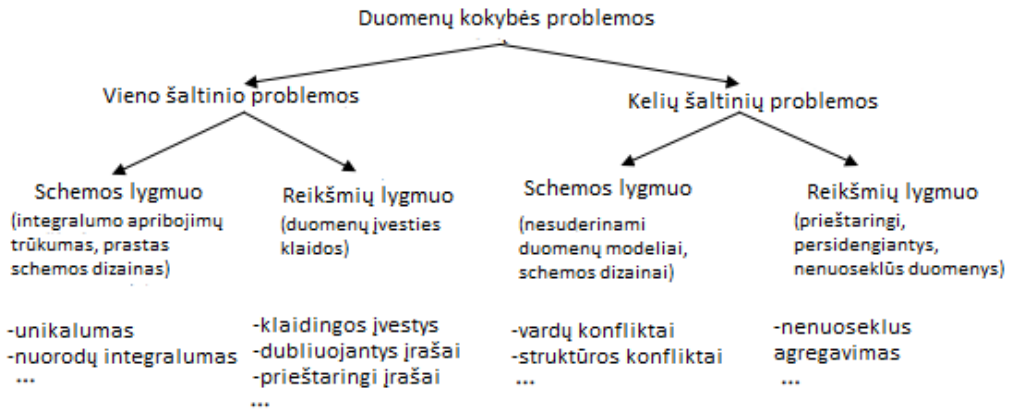
2. Tarkime, turime dvi lenteles, kuriose yra surašyti sistemos naudotojai. Mes norime migruoti naudotojų duomenis į kitą sistemą, kurioje visi naudotojai turėtų būti vienoje lentelėje (ir galbūt skirsis naujos lentelės sandara – transformuojant reikės pridėti ir išmesti tam tikrus atributus ir pan.).

Visgi, šiame darbe nagrinėjamas duomenų transformavimas platesne prasme nei minėtas apibrėžimas. Tai, be prieš tai apibrėžto funkcionalumo, remiantis [DI17], taip pat turėtų apimti įvairių taisyklių pritaikymą tuos duomenis transformuojant – apjungti duomenis, skaičiuoti tam tikras apibendrintas reikšmes, kurti raktus, rikiuoti duomenis pagal nurodytas reikšmes, kurti naujus duomenis (pagal nuskaitytuosius), taikyti validacijos taisykles, filtruoti duomenis.

1.2. Duomenų valymas ir klaidų paieška

Visgi, pats duomenų transformavimas nėra paprastas uždavinys ir tai sąlygoja, kad atlikus transformacijas, naujai suformuoti duomenys gali būti nevalidūs – tai yra galimos tam tikros duomenų anomalijos. Be to, ir pačiuose šaltiniuose esantys duomenys jau gali būti klaidingi.

Pagal [RD00] duomenų anomalijos skirstomos į 2 rūšis – vieno duomenų šaltinio problemas ir kelių šaltinių problemas (pvz. kai integruojami duomenys iš kelių šaltinių). Abi iš jų dar išsiskaido į schemos lygmens anomalijas ir reikšmių lygmens anomalijas (žr. 1 pav.).



1 pav. Duomenų kokybės problemos. Adaptuota pagal [RD00].

Remiantis [RD00], galima išskirti tokias vieno duomenų šaltinio schemos lygmens anomalijas:

- negalimos atributo reikšmės (pvz.: skaičius privalo būti teigiamas, bet randamas nulis – „0“),
- pažeistos atributų priklausomybės (pvz.: jei asmens kodas prasideda skaitmeniu „3”, tai tada žmogaus lytis turi būti „vyras“, o randama, kad tai „moteris“),
- unikalumo pažeidimas (pvz.: negali būti besidubliuojančių raktų, o randami du tokie patys raktai skirtingiems įrašams),
- nuorodų integralumo pažeidimas (pvz.: prie studento įrašo nurodyta universiteto, kuriame jis mokosi identifikatorius, tačiau tokio universiteto įrašo nėra).

Remiantis [RD00], galima išskirti tokias vieno duomenų šaltinio įrašų lygmens anomalijas:

- trūkstamos reikšmės (rastos „NULL“ reikšmės, kur jų būti neturėtų),
- gramatinės klaidos („pvz.: Vilnius po transformacijos įvardijamas kaip „Vylnius“),
- įvairūs netinkamai panaudoti sutrumpinimai (pvz.: duomenų bazių administratoriaus pareigybė atvaizduojama kaip „DB adminas“),
- neteisingai įterptas tekstas (pvz.: atribute „Vardas“, kur turėtų būti tik žmogaus vardas, įvestos reikšmės, susijusios su laisva forma įvedamais duomenimis – „Mantas, mėgsta krepšinį, studentas“),
- sumaišytos reikšmės (pvz.: atribute „Miestas“ įrašyta „Lietuva“),
- pažeistos atributų priklausomybės (analogiškai anksčiau minėtam šios anomalijos pavyzdžiui, tik šiuo atveju, tai užtikrinama duomenų bazės schema),
- netinkamai išdėliotos reikšmės (pvz.: atribute „Vardas“ sutinkamos reikšmės „Jonas Jonaitis“ ir „Petraitis Petras“, nors visada pirma turėtų eiti vardas),
- besidubliuojantys įrašai (du sykius pasitaiko toks pat įrašas – nebūtinai identiškas savo reikšmėmis, bet identiškas prasmės požiūriu, pvz.: „Jonas Jonaitis“ ir „J. Jonaitis“),
- netinkamos nuorodos (pvz.: studento įrašas turi turėti atributą, kuriame patalpintas jo universiteto identifikatorius, bet jis žymi netinkamą universitetą).

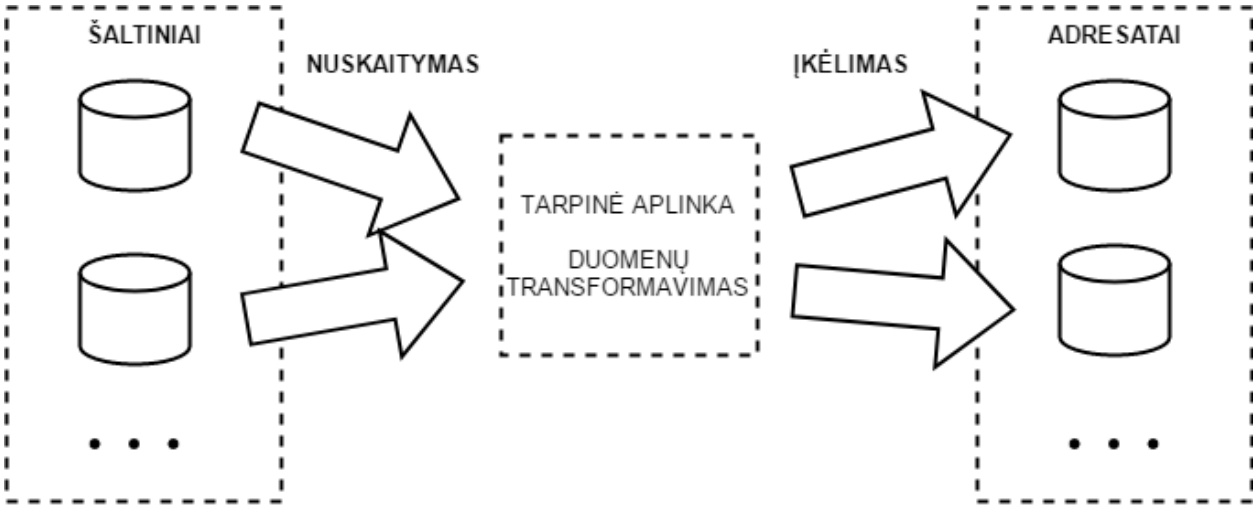
Kalbant apie kelių šaltinių problemas, pabrėžtina, kad pačios anomalijos iš esmės yra tokios pat, kaip kad aptartos vieno duomenų šaltinio problemos. Visgi, galima pastebėti, kad esminis skirtumas yra priežastis, kodėl anomalijos atsiranda. Tokių anomalijų pavyzdžiu galėtų būti ši situacija: iš skirtingų duomenų šaltinių integruojami „Naudotojų“ įrašai. Nors šaltiniuose duomenų struktūra skirtinga, bet atliekamas korektiškas susiejimas ir tikimasi, kad duomenys bus transformuoti teisingai. Visgi kai kurių naudotojų identifikatoriai sutampa, dėl to transformuoti įrašai turi besidubliuojančių identifikatorių reikšmių.

Dėl šios priežasties, reikalingas šių duomenų sutvarkymas. Tam pasitelkiamas duomenų valymas. Duomenų valymas (angl. „data cleaning“ arba „data cleansing“) padeda atrasti ir išvalyti klaidas ir neatitikimus duomenyse, taip kartu pagerindamas tų duomenų kokybę. Pagal [RD00], valymą sudaro šie etapai:

- duomenų analizė – tiek rankinė, tiek ir automatinė anomalijų paieška,
- valymo transformacijų eigos ir susiejimo nusakymas,
- transformacijų verifikavimas, dažnai su testiniais duomenimis, siekiant įsitikinti transformacijų efektyvumu,
- transformacijų vykdymas,
- šaltinių duomenų tvarkymas, t. y. rastos anomalijos turėtų būti sutvarkytos ir šaltiniuose.

2. DUOMENŲ TRANSFORMAVIMO PROCESAS

Bendrai, duomenų transformavimo procesas taip pat gali būti vadinamas ETL (angl. sutr. “Extract, Transform, Load”) procesu. Kaip jau matoma iš pavadinimo, visas procesas susideda iš 3 žingsnių: duomenų nuskaitymo, transformavimo ir įkėlimo (žr. 2 pav).



2 pav. ETL procesas. Adaptuota pagal [DI17].

2.1. Nuskaitymas

Pirmasis žingsnis – nuskaitymas. Jo tikslas, nuskaityti duomenis, kuriuos norima transformuoti, iš nurodytų šaltinių. Šaltinių kiekis bei vieta nėra svarbi. Iš jų nuskaityti galima tiek visus duomenis, tiek ir dalį (pvz. pasirinktus lentelės stulpelius, arba pagal tam tikrą kriterijų išfiltruotas eilutes, kas iš esmės galėtų sutaupyti nemažai laiko esant didesniam duomenų kiekiui). Duomenų nuskaitymas, remiantis [ED17], gali būti vykdomas šiais trimis būdais:

1. Informacija apie atnaujinimą. Jei nuskaitymas atliekamas nuolatos, kaskart skaityti visus duomenis būtų neefektyvu. Kai kurios saugyklos palaiko pranešimų apie atnaujinimus išsiuntimą, todėl nuskaitymas atliekamas tik tada, kai duomenys pasikeičia.

2. Inkrementinis nuskaitymas. Sistemos ne visada turi galimybę išsiųsti atnaujinimų pranešimus. Tačiau jos gali suteikti informaciją apie įrašus, kurie buvo atnaujinti, taigi galima nuskaityti tik juos. Visgi, tada prarandama informacija apie ištrintus įrašus.

3. Pilnas nuskaitymas. Šiuo būdų paprasčiausiai nuskaitomi visi duomenys. Tai gali būti atliekama tiek pirmą kartą, kai dar neturima duomenų, arba tada, kai sistema negali identifikuoti pasikeitimų. Visgi, tada reikėtų išsaugoti ir senų nuskaitytų duomenų kopiją, norint identifikuoti pasikeitimus.

Šiame darbe kuriama kalba pati savaime neturėtų turėti galimybės aprašyti, kaip bus sekami duomenys, todėl jos pagalba atliekamą nuskaitymą būtų galima laikyti kaip „pilną“. Tačiau su kitų

sistemų (ar žmonių) įsikišimu, kuriama DSL galėtų įgyvendinti dalį 1 ir 2 punktuose aprašyto funkcionalumo, t. y. tada kalbai turėtų būti paduodama informacija apie pasikeitimus bei nurodyta, kokiems duomenims reikalingos transformacijos.

2.2. Transformavimas

Antrasis žingsnis – transformavimas. Jis įprastai susideda iš 2 dalių – paties duomenų transformavimo ir valymo. Žingsnis ypatingas tuo, kad nuskaityti duomenys jau būna taip vadinamoje „tarpinėje aplinkoje“ [DI17], kur jie transformuojami ir taisomos duomenų anomalijos. Tai reiškia, kad duomenys iš šaltinių nėra sugadinami ir neįkeliami į paskirtą vietą nesėkmės atveju. Ši aplinka gali būti įvairi – tiek tam tikslui skirta kita reliacinė duomenų bazė, arba duomenys galėtų būti laikomi ir operatyviojoje atmintyje, jei jų kiekis nėra toks didelis. Visgi, tarpinė aplinka nėra būtinybė, transformuoti ir įkelti duomenis galima ir be jos, tačiau tai be abejonės didina klaidos tikimybę ir mažina duomenų kokybę, be to, kai kuriais atvejais, duomenis galima pasiekti tik trumpą laiką arba gali nutrūkti ryšys [Tub15].

Naudojantis šiame darbe kuriama kalba, tarpine aplinka galės būti nurodyta reliacinė SQL duomenų bazė arba tokia aplinka gali būti ir praleista.

Kalbant apie antrąjį ETL žingsnį, taip pat derėtų pabrėžti, kad jį gali sudaryti ir tik viena iš minėtų dviejų dalių – transformavimo ir valymo.

Šiame žingsnyje, remiantis [ED17], [O17], [ET17], [Tub15], gali būti vykdomos tokios transformacijos:

- Kopijavimas – transformacija, kuri nukopijuoja reikšmę iš vienos vietos į kitą;
- Apdorojimas, pasitelkus tam tikrą funkciją, t. y. nukopijuoti duomenų gali neužtekti – tuos duomenis gali reikėti ir transformuoti atliekant funkciją pasinaudojus šaltinio duomenimis kaip įvestimi;
- Numatytųjų reikšmių priskyrimas – transformacija, kuri priskiria nurodytas reikšmes adresatui;
- Identifikatorių generavimas – transformacija, kuri automatiškai sukuria identifikatorius įrašams;
- Filtravimas – transformacija, kuri išfiltruoja duomenis pagal tam tikras logines sąlygas su tais duomenimis;
- Agregavimas – transformacija, kurios tikslas, agreguoti duomenis pagal pasirinktus kriterijus pritaikant nurodytą funkciją;
- Rikiavimas – ši transformacija surikiuoja duomenis pagal nurodytus atributus;

- Apjungimas – ši transformacija apjungia duomenis (tiek kalbant apie apjungimą atributų lygmenyje, tiek apie apjungimą įrašų lygmenyje) pagal nurodytus kriterijus;
- Išskleidimas (atskyrimas) – transformacija, kuri priešingai nei apjungimas, atskiria atributus ar įrašus;
- Valymas – transformacija, kuri pagal apibrėžtas taisykles išvalo duomenų anomalijas.

2.3. Įkėlimas

Trečiasis žingsnis – įkėlimas. Kai jau duomenys transformuoti ir išvalyti, juos galima įkelti į nurodytas duomenų bazes. Kaip ir nuskaitymo, taip ir įkėlimo atveju nei duomenų bazių skaičius, nei vieta nesvarbi. Kalbant apie įkėlimą, apie šį žingsnį galima galvoti dvejopai. Viena vertus, jo paskirtis visų pirma yra įrašyti transformuotus duomenis į nurodytas saugyklas, kita vertus, gali būti ir taip, kad įrašant duomenis, tos duomenų bazės neturi tokių lentelių ar stulpelių, kurie nurodyti kaip įkėlimo vieta. Tada galimi trys pasirinkimai – paskelbti operaciją nepavykusia, ignoruoti tokias vietas arba sukurti naujas lenteles (ar stulpelius). Tad trečiuoju atveju šiame žingsnyje taip pat atliekamas ir duomenų bazės struktūros modifikavimo veiksmas.

[ED17] išskiriami du įkėlimo metodai:

- Pilnas įkėlimas: visi transformuoti duomenys yra įkeliami į nurodytus adresatus.
- Inkrementinis įkėlimas: jei duomenys nuskaitomi ir transformuojami dalimis, tada dalimis juos galima ir įkelti. Tai ypač aktualu, kai reikia duomenis nuolatos sinchronizuoti.

Be to, norint, kad įkėlimas būtų spartus, taip pat galima vykdyti įkėlimą lygiagrečiai arba naudoti masinį įkėlimą (angl. „bulk load“) [ET17], užuot įrašinėjus po vieną eilutę arba pasinaudoti lygiagretaus programavimo galimybėmis.

Pritaikant šias galimybes kuriamai kalbai, reikia pažymėti, kad duomenų bazės struktūros modifikavimas neturėtų būti atliekamas – modifikuojami tik patys duomenys. Įkėlimo aspektu, turėtų būti galimi abu minėti metodai. Visgi, masinis įkėlimas negalimas, kadangi SQL tokios galimybės nesuteikia. Tačiau neturėtų kilti kliūčių pasitelkiant lygiagretų programavimą šiam veiksmui atlikti.

2.4. ELT procesas

Vis dėlto, ETL nėra vienintelis duomenų transformavimo proceso atlikimo būdas. Kitas – ELT (angl. sutr. „Extract, Load, Transform“). Esminis jo skirtumas yra tas, kad visi duomenys yra nukopijuojami iš šaltinių į adresatą ir transformacijos atliekamos jau pačiame adresate, taip išnaudojant paties adresato resursus transformacijoms atlikti [Spe15].

Įprastai ELT pasirenkamas tada, kai reikia apdoroti didelį duomenų kiekį [Ser12], šaltinio ir adresato duomenų bazė yra ta pati [Ser12] arba adresato saugykla turi didelę skaičiuojamąją galią ir pakankamai atminties [Spe15].

Tam tikras ELT savybes galima panaudoti ir kuriant Relational DataRules, kai tam tikrus veiksmus galima vykdyti pačioje duomenų bazėje, o ne atskirai nuskaičius ir apdorojant atmintyje (pvz. agreguojant duomenis).

3. ETL PASKIRTIS

Kalbant apie duomenų transformavimo procesą, natūraliai kyla klausimas, kokiose srityse ETL (arba ir ELT) yra taikomas ir kaip tai gali įtakoti duomenų transformacijų vykdymą. Nagrinėjant literatūrą, buvo pastebėti tam tikri pasitaikantys naudojimo būdai.

Duomenų migravimas – tai procesas, kurio metu duomenys perkeliami iš vienos sistemos į kitą. Šis procesas gali būti taikomas tokiais atvejais [DI17]:

- kai sistema pasikeičia (naujos technologijos ir pan.), dėl to duomenis reikia perkelti,
- kai vienoje sistemoje prireikia duomenų iš kitos sistemos ir kt.

Duomenų sinchronizavimas – nuolat vykdomas procesas, kurio tikslas užtikrinti duomenų nuoseklumą tarp kelių sistemų, tas sistemas atnaujinant [DI17]. Kaip jau matoma iš apibrėžimo, tai nėra vienkartinis procesas. Šį procesą galima iliustruoti pavyzdžiu: kelios sistemos, esančios skirtingose vietose, atlieka tą pačią paskirtį – pardavinėja bilietus. Norint užtikrinti, kad bilietų nebūtų parduota per daug, būtina, kad duomenys tarp šių sistemų būtų darnūs – juos reikia sinchronizuoti.

Duomenų integravimas – procesas, kurio tikslas – apjungti duomenis iš skirtingų šaltinių [ET17]. Paprasčiausias tokio naudojimo pavyzdys – dviejų kompanijų susijungimas, po kurio abiejų sistemų duomenys turi būti integruoti į vieną. Kitas, labai plačiai naudojamas šio proceso pavyzdys – duomenų sandėliavimas, kai duomenys apjungiami, kur vėliau gali būti analizuojami, standartizuojami ir pan.

Duomenų valymas – procesas, kurio metu siekiama įvertinti saugyklų kokybę, surasti anomalijas ir jas išvalyti [Tub15]. Anomalijos nebūtinai gali atsirasti dėl to, kad atliekamos sudėtingos transformacijos, dėl ko atsiranda didesnė žmogiškosios klaidos tikimybė. Taip pat ir pačių šaltinių duomenys dėl vienokių ar kitokių priežasčių gali būti netvarkingi ir juos reikia išvalyti.

Šiame darbe kuriamą Relational DataRules kalbą turėtų būti galima panaudoti bet kurioje minėtoje srityje.

4. ETL SPRENDIMAI

Pasidomėjus, kokie galimi ETL uždavinių sprendimai, galima pastebėti, kad pasitelkiami įvairiausi metodai – programuotojų rašomos programos bendros paskirties kalbomis, ETL (angl. „Extract, Transform, Load“) įrankiai, grįsti tiek grafinės sąsajos naudojimu, tiek ir programinio kodo rašymu naudojant įvairias ETL bibliotekas. Iš esmės, [KC04] išskiriami 2 būdai, kaip spręsti šiuos uždavinius.

4.1. ETL sprendimo būdai

Pirmasis būdas – tiesiog patiems rašyti savo ETL programas. Būdo principas toks – jei kyla tam tikra problema (šiuo atveju duomenų transformavimo uždavinys), visada galima sukurti programą jai spręsti. Šiam tikslui įgyvendinti galima pasitelkti bendros paskirties programavimo kalbą (pvz. Java, C#, C++ ir pan.) ir suprogramuoti sprendimą. Remiantis [KC04] galima išskirti tokio pasirinkimo privalumus:

- Parašytas programinis kodas yra tinkamas testavimui, kas sumažina klaidos (ir duomenų anomalijų pasitaikymo) tikimybę;
- Sukurta ETL programa leidžia pasiekti didžiulį lankstumą – programa rašoma specifiniam tikslui, specifinei dalykinei sričiai (tai jau siauresnė sritis nei duomenų transformavimas apskritai), todėl būtent šiam uždaviniui spręsti ji veikiausiai puikiai atliks savo funkciją;
- Kadangi programą kurtų savi programuotojai, jie geriau galėtų pritaikyti dalykinės srities žinias norimoms transformacijoms atlikti, bei geriau išnaudotų savo stiprybes kaip programuotojai – t. y. naudotų įrankius (ir programavimo kalbas), kuriuos geriausiai patys išmano.

Kitas būdas – naudoti jau sukurtus ETL įrankius. Remiantis [KC04], galima išskirti šio pasirinkimo privalumus:

- Įrankiams (šiuo atveju, kalbant apie įrankius su grafine sąsaja) naudotis, nereikia programavimo žinių, siekiant atlikti ETL uždavinius – norimus veiksmus galima nurodyti grafiškai, pasirenkant meniu pasirinkimuose.
- Dauguma įrankių užtikrina aukštą greitaveiką, kadangi jie buvo kuriami būtent šiam tikslui.
- Kadangi įrankiai kuriami šiam tikslui, daugeliu tų įrankių aspektų naudotojui rūpintis nereikia, nes tai atliekama automatiškai (t. y. didelė vykdymo logikos dalis paprasčiausiai paslėpta nuo naudotojo).

4.2. ETL įrankiai

Pačius ETL įrankius būtų galima išskirstyti į 3 rūšis (remiantis [Tub15]):

- Tekstiniai ETL įrankiai,
- Grafiniai ETL įrankiai,
- Mokslo tiriamieji įrankiai.

Tekstinių ETL įrankių grupei būtų galima priskirti įrankius, kurie naudoja tekstinę programavimo kalbą. Tai gali būti įrankiai, kurie remiasi bendros paskirties kalba (Java, C++, ir pan.), vidine dalykinės srities kalba (pvz. petl – Python kalbos biblioteka [P17]) ar išorine DSL (platesnio naudojimo dalykinei sričiai, įskaitant duomenų transformacijas – SQL, tik duomenų transformacijoms – DataRules [Tub15] (kalba nerealizuota) ir kt.).

Grafinės sąsajos ETL įrankiai pasižymi tuo, kad jie turi grafinę naudotojo sąsają, kurios pagalba galima modeliuoti norimas duomenų transformacijas, įvairių mygtukų paspaudimais, išsiskleidžiančių meniu naudojimu, schemų braižymu ir pan. [Tub15] išskiriami populiaresni atstovai – komerciniai IBM InfoSphere DataStage, Microsoft Integration Services, Oracle Warehouse Builder, Informatica PowerCenter, Ab Initio, bei nemokami Apatar, Clover.ETL, KETL, Kettle/Pentaho Data Integration, Talend Open Studio.

Mokslo tiriamiesiems įrankiams priskiriami įrankiai, kurie pasižymi specifinėmis, vien tik šiems įrankiams būdingomis savybėmis [Tub15]. Tokio įrankio pavyzdys galėtų būti Potter's Wheel [RH01]. Tai pirmasis tokio tipo įrankis, kuris buvo sukurtas 2001 metais, ir kurio veikimo principas pasižymi tuo, kad naudotojas gali kurti transformacijas aprašytas dalykinės srities kalba, pasirinkdamas komandas iš meniu, bei gali matyti greitus rezultatus. Šis įrankis vėliau įkvėpė jo kūrėjus sukurti komercinį įrankį Trifacta [HK14], kuris skirtas transformuoti nestruktūrizuotus duomenis (dėl šios priežasties labiau tinkamas analizei, o ne ETL užduotims) rašant ekspresyvias komandas tam sukurta dalykinės srities kalba ir gaunant greitus rezultatus. Šis įrankis taip pat gali apsimokyti (numato, kokias funkcijas norės atlikti naudotojas), bei yra laikomas itin patogiu – laimėjo daugelį apdovanojimų kaip patogiausias įrankis duomenų transformacijoms atlikti [Dor16].

5. DALYKINĖS SRITIES KALBA

Dalykinės srities kalba (arba sutrumpintai DSL, pagal angl. „Domain specific language“) yra kompiuterinė kalba, skirta tam tikros dalykinės srities uždaviniams spręsti, skirtingai nei bendros paskirties kalbos, tinkamos atlikti bet kokios srities uždavinius. Kitaip sakant, tai kalba, kuri siekia iškelti dalykinės srities ekspresyvumą ir paprastumą tos srities uždaviniams atlikti (kartu aukojant tinkamumą kitiems uždaviniams). Visgi „DSL kūrimas yra sudėtingas uždavinys“, reikalaujantis tiek dalykinės srities, tiek kalbos kūrimo žinių, todėl „dažnai atidedamas vėlesniam laikui“ arba apskritai nepradedamas [MHS05].

Siekiant suprasti, kas yra dalykinės srities kalba, ją galima palyginti su bendros paskirties kalba. Bendros paskirties kalba būtinai turi būti vykdoma [Wil01], ir ji nėra skirta tam tikram specifiniam uždaviniui spręsti. Bendros paskirties kalbos pavyzdžiu galėtų būti tokios žinomos kalbos kaip C, C++, Java, C#, Python ir t. t., o keletu žymesnių DSL atstovių galima įvardinti HTML (kalba skirta aprašyti puslapius hipertekstu), SQL (kalba, skirta rašyti užklausas duomenų bazėms), LaTeX (LaTeX sistemos kalba, skirta dokumentų paruošimui) ir kt.

Vis dėlto, griežtos ribos, kada galima įvardinti, kad kalba yra bendros paskirties, o kada ne, nėra. Kalbant apie šias kalbas, galima įsivaizduoti graduotą skalę, kur krašte prie DSL būtų HTML kalba, o krašte prie bendros paskirties kalbų – C++. Pavyzdžiui, kai kurie žmonės ir Cobol laiko dalykinės srities kalba, skirtą verslo programoms, tad ji galbūt galėtų būti jau kiek arčiau vidurio minėtoje skalėje. [MHS05]

Pačios dalykinės kalbos gali būti skirstomos į dvi rūšis pagal įterpimo būdą – vidines ir išorines. Vidinė DSL yra tokia kalba, kuri išpildoma bendros paskirties kalbos elementų (gramatikos, raktinių žodžių ir bibliotekinių funkcijų) poaibiu. Tokios kalbos pavyzdžiu gali būti Python biblioteka, skirta ETL funkcijoms atlikti – petl [P17]. Šių kalbos išraiškos apibrėžtos per funkcijas, kurios, atlikdamos paprastus uždavinius, gali būti gana ekspresyvios, pvz. petl atveju, rikiavimo transformacija iš vienos lentelės į kitą galėtų būti užrašyta taip:

table2 = etl.sort(table1, 'foo')

Išorinė DSL, skirtingai nei vidinė, nėra priklausoma nuo bendros paskirties kalbos – tai kalba su būtent tai kalbai skirtu transliatoriumi. Ji gali būti simuliuojama ar vykdoma tai kalbai skirtų transliatorių, interpretatorių, kompiliatorių. [MHS05] Išorinių DSL pavyzdžiai yra jau minėtos SQL, HTML.

6. IŠORINIŲ DSL DUOMENŲ TRANSFORMACIJOMS ATLIKTI APŽVALGA

Siekiant sukurti dalykinės srities kalbą, derėtų paieškoti, kokios kalbos jau sukurtos šiai ar panašiai dalykinei sričiai tam, kad būtų galima pasinaudoti jų savybėmis. Paieškojus, ar yra jau sukurtų išorinių dalykinės srities kalbų duomenų transformacijoms atlikti, buvo rastos tam tikros tokio pobūdžio kalbos. Jos nagrinėtos atsižvelgiant į tai, kokiems duomenims apdoroti skirtos, kokius veiksmus (transformacijas) galima atlikti, ar kalba užbaigta ir palaikoma, kokios jos kalbinės savybės (paprastumas, ekspresyvumas), kiek kalba dokumentuota.

Viena, gana populiari kalba – **Trifacta** (kartu su pačiu Trifacta įrankiu). Ši DSL skirta apdoroti nestruktūrizuotiems duomenims. Jos paskirtis – transformacijų vykdymas įvairiems analizės tikslams. Tai DSL, pasižyminti dideliu ekspresyvumu ir paprastumu [HK14]. Kalbos gramatika ir sintaksė dokumentuota, tačiau vykdymas – komercinė paslaptis, kuri neskelbiama. Trifacta užbaigta ir palaikoma. Apibendrinus, matoma, kad ši kalba netinkama ETL vykdymui (skirta duomenų analizei nestruktūrizuotiems duomenims), o jos konstrukcijomis galima pasinaudoti nebent siekiant idėjų, kaip patogiau aprašyti transformacijas, kadangi kalba ekspresyvi ir paprasta.

Kita DSL – **HiperFuse** [HQC14]. Kalba skirta darbui su įvairiais duomenimis. Jos paskirtis – integruoti duomenis iš skirtingų duomenų šaltinių (jei reikia atliekant transformacijas). Ši kalba remiasi tam tikrų direktyvų naudojimu, o tos direktyvos vykdo PDDL (angl. sutr. „Planning Domain Definition Language“) kalba užrašytas funkcijas. Visgi, kalba dar nėra baigta ir nerasta informacijos, kad jos kūrimas būtų tęsiamas. Kalbines savybes taip pat sunku įvertinti, nes kalbos gramatika ir sintaksė niekur nepaviešinta. Apibendrinus, galima teigti, kad kalbos savybėmis pasinaudoti negalima, nes kalbos gramatika ir sintaksė nėra vieša, o HiperFuse nebeplėtojama.

Dar viena dalykinės srities kalba – **XClean** [WM07]. Ji skirta darbui su XML failuose saugomais duomenimis, o jos paskirtis kiek siauresnė – duomenų anomalijų valymas. Taigi, lyginant su šiame darbe nagrinėjama dalykine sritimi, sutampa tik pati anomalijų paieška ir valymas, bet ir ji atliekama tik XML failuose. XClean gramatika ir sintaksė nusakyta. Kalba kaip ir jos dalykinė sritis nedidelė, bet jau būdingos specifinės šios dalykinės srities konstrukcijos, kurios vėliau verčiamos į XQuery konstrukcijas. Apibendrinus, galima teigti, kad kalbos konstrukcijomis darbe nagrinėjamai dalykinei sričiai pasinaudoti būtų sudėtinga, kadangi XClean būdingos specifinės savybės, kurių reikia XML duomenų apdorojimui.

DataRules – dalykinės srities kalba, kurios dalykinė sritis panaši į nagrinėjamą šiame darbe – ji apima duomenų transformavimą ir validavimą struktūrizuotiems duomenims [Tub15], kurie saugomi įvairiose saugyklose (nebūtinai reliacinėse duomenų bazėse). Ši kalba pritaikyta atlikti

visus tris ETL žingsnius – nuskaitymą, transformavimą ir įkėlimą. Išnagrinėjus DataRules transformacijų aibę, galima pastebėti, kad ji taip pat geba vykdyti visas 2 skyriuje nusakytas transformacijas. Be to, kalba kurta, siekiant kuo didesnio ekspresyvumo, paprastumo ir artumo žmogiškąjai kalbai (pvz. panaudojant įvairius jungtukus, prielinsksnius).

Vis dėl to, jeigu pati DataRules kalba apibrėžta (nusakyta jos gramatika, sintaksė), tai nesudarytas tos kalbos vykdymo modelis – neapibrėžta, kaip turėtų būti vykdomos transformacijos – kokia tvarka, kokie yra pačių transformacijų algoritmai ir pan. Be to, kalba teorinė ir jos įgyvendinimas neplanuojamas.

Yra ir kitų dalykinės srities kalbų, skirtų tam tikros duomenų transformacijoms atlikti, tačiau tos kalbos akivaizdžiai stokoja šiai dalykinei sričiai reikalingo ekspresyvumo, todėl jos nenagrinėtos (pvz. AJAX naudojama kalba SQL99 [GFS+01], kuri tiesiog praplečia SQL).

Reziumuojant, kalba, kurios savybėmis ar konstrukcijomis kuriant dalykinės srities kalbą šiame darbe nagrinėjamai dalykinei sričiai būtų galima pasinaudoti labiausiai – DataRules, kadangi jos nagrinėjama dalykinė sritis panaši į šio darbo dalykinę sritį, konstrukcijos dokumentuotos, kalba ekspresyvi ir paprasta. Tuo bus galima įsitikinti išnagrinėjus, kokius reikalavimus dalykinės srities kalbai ji tenkintų (žr. 10 skyrių).

7. DSL VYKDYMO BŪDAI

Viena vertus, kuriant dalykinės srities kalbą, reikia apibrėžti, kokia tos kalbos sintaksė, tačiau kita kūrimo proceso dalis – nuspręsti kaip ta kalba bus vykdoma. Pagal [MHS05] išskiriami tokie dalykinės srities kalbų vykdymo būdai:

- **Interpretatorius** (angl. „interpreter“). Tai kompiuterinė programa, kuri iš karto vykdo šiuo atveju DSL parašytą programą, nekurdama jos dvejetainio vykdomojo failo [LZ18]. Pasitelkiant interpretatorių, programų vykdymas tampa paprastesnis, nes jos vykdomos be jokio parengiamojo etapo – kompiliavimo. Vis dėlto, pats vykdymas kiek lėtesnis lyginant su kompiliatoriumi, nes kaskart reikia analizuoti parašytos programos tekstą.
  Esminiai privalumai:
  - Paprastumas lyginant su kompiliatoriumi, nes nereikia kompiliuoti kodo į mašininį,
  - Didesnė vykdymo kontrolė, lyginant su kompiliatoriumi,
  - Taip pat lyginant su kompiliatoriumi, galimas didesnis lankstumas, plečiamumas,
  - DSL sintaksė gali būti tokia, kokia tinkama dalykinės srities ekspertams,
  - Gera klaidų valdymo galimybė,
  - Dalykinės srities analizės, verifikavimo, optimizavimo, transformavimo galimybė.

  Trūkumai:
  - Sudėtingesnis kūrimas (lyginant su vidinėmis DSL), kadangi reikia sukurti sudėtingą kalbos procesorių. Visgi, tai galima kiek palengvinti jau naudojant tam tikslui sukurtus įrankius, kurie palengvintų kalbos atpažinimo procesą.
  - Kiek lėtesnis vykdymas (lyginant su kompiliatoriumi) dėl tos pačios kalbos konstrukcijų analizės vykdant DSL parašytas programas.
- **Kompiliatorius/kodo generatorius** (angl. „compiler/application generator“). Tai programa, verčianti programos tekstą iš vienos programavimo kalbos (šiuo atveju dalykinės srities kalbos) į kitą (dažnai iš aukštesnio lygio į žemesnio) [LZ18].
  Privalumai ir trūkumai panašūs į interpretatoriaus, skirtinga tik tai, kad programų vykdymas gali būti kiek greitesnis, nes nereikia kaskart analizuoti DSL teksto, bet šiek tiek mažesnis lankstumas, plečiamumas.
- **Preprocesorius** (angl. „preprocessor). Tai programa, kuri įvesties duomenis (DSL konstrukcijas) paverčia įvesties duomenimis kitai programai, kuri galėtų parašytą programą vykdyti. Šis būdas panašus į kompiliatorių, tik kad statinė DSL analizė apribota ir vykdoma tik bendros paskirties kalbos procesoriaus [MHS05].
- **Įterpimas** (angl. „embedded“). Kitaip sakant, tai vidinės DSL realizacija, kai kalba išreikšta per bendros paskirties kalbą.

- **Praplečiamas kompiliatorius/interpretatorius** (angl. „extensible compiler/interpreter). Tai būdas, kai jau esamos kalbos kompiliatorius ar interpretatorius praplečiamas dalykinės srities esybėmis, taisyklėmis ir kt. Interpretatoriaus praplėtimas galėtų būti sąlyginai paprastas, bet kompiliatoriaus įprastai gana sudėtingas (nebent kuriant jį buvo galvojama apie būsimą praplėtimą).
- **Egzistuojantys įrankiai.** Šiuo atveju tam tikri jau sukurti įrankiai pritaikomi dalykinei sričiai.
- **Hibridinis.** Šio būdo esmė ta, kad dalykinės srities kalbos vykdymas susideda iš kelių anksčiau paminėtų būdų.

Kadangi šiame darbe nagrinėjamas būtent išorinės dalykinės srities kalbos naudojimas duomenų transformacijoms reliacinėse duomenų bazėse atlikti, reiškia, kad DSL reikia arba interpretuoti, arba generuoti kitos kalbos programinį kodą iš jos. Kadangi žinoma, kad bus dirbama su reliacinėmis SQL duomenų bazėmis, savaime atrodo, kad galbūt vertėtų generuoti SQL užklausas. Vis dėlto, pastebima, kad ne visas funkcionalumas gali būti įgyvendinamas (pvz. prisijungimai prie duomenų bazių, ar validavimo galimybės), be to, transformacijos taip pat gali būti sudėtingesnės ir natūraliai vien tik SQL užklausom jų aprašyti nepavyktų (pvz. įvairių funkcijų pagal verslo logikos taisykles atlikimas su duomenimis). Taip pat, nepavyktų užtikrinti deramo klaidų valdymo mechanizmo.

Tada galima apsvarstyti kitus du būdus – interpretavimą ir kompiliavimą/kodo generavimą į bendros paskirties kalbą. Jų veikimas iš dalies panašus – nuskaitomas DSL aprašytas programos tekstas ir jis vykdomas – interpretavimo atveju, iškart skaitant tekstą, kompiliavimo – po patiės kompiliavimo proceso. Interpretatoriaus pagrindiniai privalumai prieš kompiliatorių galėtų būti paprastesnis įgyvendinimas, lankstumas ir plečiamumas (ši savybė galėtų būti gana svarbi, nes kalbos įgyvendinimas yra kūrimo fazėje, daug kas gali kisti, todėl paprasčiau tai suvaldyti) bei didesnė vykdymo kontrolė.

Savo ruožtu kompiliuotas programinis kodas vykdomas greičiau. Tai galėtų būti svarbus aspektas, bet prisiminus, kad dalykinė sritis – duomenų transformacijos, galima pastebėti, kad pačios transformacijos užtruks neabejotinai ilgiau nei DSL teksto skaitymas, todėl nauda šiuo aspektu veikiausiai tokia didelė nėra.

Visgi, jei interpretatorius ar kompiliatorius/kodo generatorius kuriamas pasitelkiant jau egzistuojančius kalbos analizės įrankius, tam tikri sudėtingesnės įgyvendinimo detalės tiek vienu, tiek kitu atveju minimizuojamos, todėl iš esmės abu sprendimai atrodo geri ir galutinį nuosprendį galėtų padėti išsiaiškinti naudojami įrankiai – t. y. vienu ar kitu įrankiu geriau pasirinkti vieną iš šių būdų dėl geresnio palaikymo, greitaveikos ar pan.

Be šių dalykų, taip pat galima pasinaudoti jau sukurtas įrankiais, kurie galėtų būti pritaikomi įgyvendinant patį DSL aprašomą transformacijų vykdymą – pvz. kviesti tam tikras komandas iš tam tikslui sukurtų ETL bibliotekų. Tokiu būdu, būtų pasitelkiamas ir būdas „Egzistuojantys įrankiai“.

Apibendrinus, galima pastebėti, kad savų privalumų turi DSL vykdymo įgyvendinimas tiek pasitelkiant interpretatorių, tiek ir kompiliatorių/kodo generatorių, kadangi jų įgyvendinimas gali būti palengvintas jau sukurtų kalbos analizatorių. Taip pat galima pasinaudoti tam tikromis jau sukurtomis ETL bibliotekomis atliekant transformacijas. Reiškia, kad dalykinės kalbos vykdymas galėtų būti įgyvendintas pagal aprašytą interpretatoriaus ar kompiliatoriaus/kodo generatoriaus būdą, arba hibridinį (interpretatoriaus ar kompiliatoriaus/kodo generatoriaus, ir egzistuojančių įrankių).

8. KALBOS ANALIZAVIMO ĮRANKIAI

Dalykinės srities kalbos interpretatoriaus ar kompiliatoriaus/kodo generatoriaus kūrimas yra sudėtingas. Vis dėl to, sukurta įvairių įrankių, kurie pastebimai palengvina šį uždavinį. Dėl šios priežasties šiame skyrelyje nagrinėjami kalbos analizavimo įrankiai, kurie galėtų padėti įgyvendinti dalykinės srities kalbos konstrukcijų atpažinimą.

8.1. ANTLR

ANTLR (ANother Tool for Language Recognition) – analizatoriaus kūrimo įrankis, kuriuo naudojantis galima skaityti, vykdyti, versti tam tikrą struktūrizuotą tekstą ar dvejetainius failus. Šis įrankis labai populiarus ir greitas (pvz. Twitter jį naudoja paieškos užklausoms analizuoti – kasdien išanalizuojama po daugiau nei 2 milijardus užklausų; NetBeans naudoja įrankį C++ kalbai analizuoti ir kt.).

Šio įrankio veikimo principas remiasi taip vadinamų „gramatikų“ naudojimu, kurios paremtos Bekaus ir Nauro forma (sutr. BNF). BNF – metakalba, formalios kalbos sintaksei apibrėžti [KZ18]. Pagal parašytą teksto (šiuo atveju dalykinės srities kalbos) gramatiką ANTLR sukuria analizatorių, kuris sukonstruoja analizavimo medžius (angl. „parse trees“), kurie savo ruožtu susieja aprašytą gramatiką su įvestimi. Taip pat įrankis sukuria ir taip vadinamus „ėjikus“ (angl. „tree walkers“), kurie geba aplankyti tų medžių viršūnes ir taip vykdyti tam tikrą bendros paskirties kalba parašytą programinį kodą.

ANTLR pagrindiniai privalumai yra tai, kad įrankis yra galingas ir greitas, leidžiantis valdyti visą teksto analizavimo procesą. Be to, tai gana patogus įrankis, leidžiantis aprašyti gramatikas tam tikslui sukurta dalykinės srities kalba.

Šiuo metu naujausia ANTLR versija – 4.7. Ji leidžia generuoti Java, C#, C++ , JavaScript, Python2, Python3, Swift, ir Go programinį kodą.

Informacija apie įrankį parengta pagal [A18].

8.2. Xtext

Xtext – programavimo ir dalykinių sričių kalbų kūrimo karkasas. Tam tikra prasme, šis įrankis gana panašus į ANTLR, bet be analizatoriaus kūrimo, taip pat suteikia ir patogią programavimo aplinką sukurtai kalbai.

Pats Xtext, kaip teigiama [X17], pagal nutylėjimą naudoja ANTRL analizatoriaus generatorių, todėl teksto analizė veikia iš esmės taip pat. Be to, Xtext (kartu su kitu įrankiu Xtend) sudaro sąlygas efektyviai kurti naujos kalbos kompiliatorių/kodo generatorių. Taip pat, kaip ir ANTLR, Xtext naudoja gana panašų būdą gramatikoms aprašyti.

Xtext, skirtingai nei ANTLR, sukuria analizatorių Java kalbai, tai yra sugeneruotos klasės aprašytos Java. Tai netrukdo konstruoti kitų kalbų kodo, bet toks konstravimas tampa tiesiog teksto konstravimu, nes Xtext tinkamiausiai veikia konstruojant Java programinį kodą.

Vis dėlto, jeigu norima rinktis interpretavimo vykdymo būdą, šis įrankis gali būti ne pats tinkamiausias tai atlikti, nors informacija apie tai dvejopa. Oficialioje įrankio svetainėje ([X17]) plačiai nekalbama apie interpretavimo galimybę, tačiau panagrinėjus Eclipse bendruomenės atsiliepimus ir diskusijas [E17], galima pastebėti, kad nors interpretavimo galimybė yra, bet ji yra gana apribota – todėl geriau rinktis kompiliavimo ar kodo generavimo būdą.

Esminis Xtext privalumas – programavimo aplinkos sukūrimas naujai sukurtai (dalykinės srities) kalbai. Tai galima padaryti įrašant papildinį į „Eclipse“ ar „IntelliJ“ įrankį. Taip pat Xtext neseniai skelbė, kad su tam tikrais apribojimais, kaip programavimo aplinkos galės būti panaudojamos ir įvairios naršyklės (pvz. „Chrome“, „Firefox“ ir pan.).

Informacija apie įrankį parengta remiantis [X17].

8.3. Kiti įrankiai

Iš tiesų, įrankių palengvinančių dalykinės srities kalbos kūrimo procesą yra gana daug ir visus aprašyti būtų gana sudėtinga, todėl be 2 populiarių anksčiau paminėtų įrankių, programuotojų bendruomenė naudoja ir kitus įrankius, kurių keli paminėti šiame poskyryje.

textX – Python karkasas, įkvėptas Xtext įrankio. Šiame įrankyje galima aprašyti gramatikas labai panašiai, kaip tai atliekama ir Xtext. textX negeneruoja kodo, bet išnaudoja Python metaprogramavimo galimybes ir apibrėžia klases operatyviojoje atmintyje. Esminis skirtumas nuo Xtext – šis įrankis nesukuria programavimo aplinkos. [Tom18]

Spoofax – (dalykinės srities) kalbų kūrimo platforma. Ji suteikia galimybę kurti kalbas remiantis šio įrankio tam tikslui sukurtomis dalykinės srities kalbomis. Įrankis geba sukurti interpretatorius, kompiliatorius, analizatorius pagal apibrėžtas kalbas. Taip pat, kaip ir Xtext, sukuria papildinius Eclipse ar IntelliJ (eksperimentinė versija), kurie gali būti panaudoti programavimo aplinkos pritaikymui sukurtai kalbai. [S18]

Jetbrains MPS – labai galingas (ir kartu sudėtingas) įrankis įvairių kalbų (ne tik tekstinių, bet ir grafinių ar kt.) kūrimui. Šis įrankis naudojamas, kai norima sukurti galingą ir labai spartų kalbos analizatorių ir vykdytoją [Tom18]

8.4. Tinkamiausias įrankis kuriamai kalbai

Išanalizavus analizatorius galima pastebėti, kad vieni ar kiti turi savų privalumų ar trūkumų, tinkamesni programuojant viena ar kita bendros paskirties kalba. Vis dėlto dėl plataus bendros paskirties kalbų palaikymo, detalios dokumentacijos, daugelio dalykinės srities kalbų, kurios

naudoja įrankį ir kurių pavyzdžiais galima remtis analizuojant Relational DataRules konstrukcijas, taip pat dėl BNF gramatikos naudojimo, kuri supaprastina kalbos interpretatoriaus ar kodo generatoriaus kūrimą nuspręsta naudoti ANTLR įrankį.

9. ETL BIBLIOTEKOS

Prieš konstruojant DSL vykdymo modelį, galima pastebėti, kad dalis duomenų transformavimo vykdymo logikos įgyvendinta jau egzistuojančiose ETL bibliotekose ir naudojantis tuo būtų galima supaprastinti to modelio kūrimą. Dėl šios priežasties, nagrinėjamas galimas jau sukurtų ETL bibliotekų funkcionalumo pritaikymas kuriant Relational DataRules vykdymo modelį.

Kadangi ETL bibliotekų kiekis didžiulis, nuspręsta apžvelgti populiaresnes C#, Java, Python kalbų bibliotekas. Šios kalbos suderinamos su anksčiau įvardintais analizatorių kūrimo įrankiais (ANTRL, Xtext arba textX). Taip pat, norint panaudoti tam tikras ETL bibliotekų funkcijas (ar pasisemti idėjų iš jų vykdymo), tą galima padaryti tik žinant, kaip tos bibliotekų funkcijos yra įgyvendintos – t. y. bibliotekos privalo būti atviro kodo. Be to, atsižvelgiama į bibliotekų palaikymą, atitikimą šio darbo dalykinei sričiai ir galimą funkcionalumą, dokumentaciją.

9.1. C# ETL bibliotekos

Pats populiariausias .NET karkaso įrankis ETL operacijoms C# kalba atlikti – **RhinoETL** (remiantis [N17] atsisiuntimų skaičiumi). Pasinaudojant juo, galima ištraukti duomenis iš bet kokios reliacinės duomenų bazės (taip pat ir iš kitokių šaltinių), juos transformuoti ir įkelti į nurodytą vietą. Tai galima atlikti tiek rašant C# klases, tiek ir pasitelkiant panašią sintaksę į C# turinčią dalykinės srities kalbą. Operacijos vykdomos manipuliuojant nuskaitytais duomenimis atmintyje, dažnai pasitelkiant komandų grandines, kai operacija vykdomos ne iš karto jas perskaičius, bet vykdoma visa grandinė, kas pagreitina atlikimo laiką. Visgi pati transformacijų aibė nėra plati, kitaip sakant pats programuotojas turi suprogramuoti tam tikras transformacijas, pasitelkdamas bendros paskirties kalbos konstrukcijas (ciklus, kintamuosius, sąlygos sakinius ir t.t.). Taip pat, RhinoETL nėra detaliai dokumentuotas, pateikiami keli pavyzdžiai, kaip reikėtų naudotis biblioteka. Šiam įrankiui reikalingas .NET 4.0 karkasas. [HR17]

Kitas nagrinėtas įrankis – **ReactiveETL**. Iš esmės, ši biblioteka yra perrašyta pagal RhinoETL pritaikant tam tikrus pagerinimus greitaveikai naudojant Reactive Extensions biblioteką. Tai reiškia, kad jos funkcionalumas turėtų būti toks pat kaip ir RhinoETL. Visgi, panašu, kad pats projektas yra apleistas (nekeistas nuo 2012 m.), todėl seniau buvusios klaidos neištaisytos, operacijos gali būti pasenusios, be to, galimai net ir greitaveikos prasme pats RhinoETL šiuo metu jau gali būti optimizuotas ir greitesnis. [R17]

Dar kelios nagrinėtos bibliotekos – **Cinchoo ETL**, **NDataflow**, **FluentETL**. Nustatyta, kad pirmoji tinkama tik darbui su tekstiniais failais, tokiais kaip CSV, XML ir pan. [C17], antroji nebevystoma jau nuo 2010 metų, be to, visiškai nedokumentuota, o trečioji iš esmės yra

supaprastinta RhinoETL versija, kurios pagalba galima perkopijuoti duomenis (kartu ir modifikuojant), bet ir ji nebepalaikoma (išleista tik BETA versija) [F17].

Apibendrinus, galima pastebėti, kad ETL įrankių pritaikytų C# kalbai nėra daug, arba jie visai nepopuliarūs (ieškant bibliotekų iš esmės išsiskyrė tik RhinoETL). Kaip ir populiarumo aspektu, taip ir funkcionalumo prasme – RhinoETL, lyginant su kitomis C# bibliotekomis, galėtų duoti daugiausiai naudos sprendžiant tam tikrus šiame darbe nagrinėjamos dalykinės srities uždavinius. Veikiausiai pagrindinis šios bibliotekos privalumas – komandų grandinių pritaikymo galimybė vykdant ETL operacijas, kas duotų naudos greitaveikos aspektu.

9.2. Java ETL bibliotekos

Vienintelė rasta atviro kodo universali Java (netgi tiksliau – Groovy) ETL biblioteka – **GETL** (pagal Groovy ETL). Kaip teigiama jos aprašyme [G17], ji geba vykdyti įvairias ETL operacijas kviečiant funkcijas iš Groovy ar Java klasių. Vis dėlto, detaliau biblioteka nedokumentuota, todėl paties programinio kodo neištyrus, sunku spręsti apie šios ETL bibliotekos panaudojimo galimybes.

Kitos paieškos metu rastos bibliotekos apėmė mažesnę dalykinę sritį arba nebuvo atviro kodo, todėl neįvardijamos ir detaliau nenagrinėtos.

9.3. Python ETL bibliotekos

C# ir Java ETL bibliotekų kiekis sąlyginai nedidelis, tačiau Python šiuo aspektu situacija geresnė. Ieškant šiai kalbai pritaikytų bibliotekų buvo detaliau išnagrinėtos bent 4 (tačiau galima rasti ir daugiau) kiek daugiau vertos dėmesio – Parade, Bubbles, pandas ir petl.

Parade – įrankis, skirtas atlikti tokius su duomenimis susijusius procesus kaip ETL, duomenų analizė, verslo analizės raportų sudarinėjimas ir t. t. Sprendžiant iš bibliotekos saugyklos puslapyje esančios istorijos, tai ganėtinai nauja biblioteka, lyginant su kitomis nagrinėtomis – pirmasis kodo įkėlimas atliktas 2017 m. kovo 3 d. Kaip teigiama Parade saugyklos puslapyje, šis įrankis veikia Linux, Windows, Mac OSX ir BSD operacinėse sistemos ir reikalauja Python 3.3 arba naujesnės versijos. [Par17]

Išnagrinėjus dokumentacijoje [Par17] pateiktus bibliotekos naudojimo pavyzdžius, galima pastebėti, kad Parade leidžia atlikti tam tikras transformacijas (įskaitant ir tam tikrų anomalijų valymą arba filtravimą, pvz. NULL reikšmėms), skaito ir įkelia duomenis. Vis dėlto, dokumentacijos lygis gana žemas, nelabai aišku, ar galima nuskaityti duomenis iš reliacinės duomenų bazės, nors teigiama, kad įkelti į tokią galima. Matyti, kad Parade neseniai sukurta ir šiuo metu plėtojama biblioteka, gebanti iš dalies atlikti tam tikrus uždavinius šio darbo dalykinės srities problemoms spręsti ir šiuo metu esanti pradinėje kūrimo stadijoje.

Bubbles – Python karkasas, skirtas duomenų apdorojimui ir kokybei užtikrinti. Kaip rašoma įrankio svetainėje, ši biblioteka optimizuota ir automatiškai pasirenka, kaip vykdys operacijas priklausomai nuo duomenų šaltinio (pvz., jei tai reliacinė duomenų bazė, naudos SQL užklausas, kur tai įmanoma padaryt). Pati operacijų aibė nėra didelė (galima vykdyti tam tikras standartines funkcijas – filtruoti, rikiuoti duomenis, ieškoti besidubliuojančių įrašų ir kt.) ir verslo logikos transformacijoms atlikti nepritaikyta. Taip pat, panašu, kad naudojantis Bubbles, duomenis galima tik nuskaityti ir apdoroti, bet įkėlimo į adresatą galimybės nėra. [B17]

Šis įrankis reikalauja Python 3.3 arba naujesnės versijos. Visgi, kaip teigiama Bubbles saugyklos puslapyje, biblioteka nebepalaikoma, todėl jos funkcijų aibė ateityje praplėsta nebebus. [BR17]

Trečioji nagrinėta Python ETL biblioteka – **pandas**. Šis įrankis, remiantis nagrinėtų bibliotekų GitHub saugyklų duomenimis yra populiariausias tarp šiame skyrelyje nagrinėjamų atstovų. Kaip teigiama bibliotekos aprašyme, tai lanksti ir galinga biblioteka duomenų analizės ir manipuliavimo uždaviniams spręsti. Nors pandas puslapyje rašoma, kad pagrindinė įrankio paskirtis yra duomenų analizė (panašiai kaip naudojant įrankį R, tik kad Python kalba), bet taip pat, biblioteka suteikia galimybę skaityti ir rašyti duomenis į įvairius duomenų šaltinius (įskaitant ir reliacines duomenų bazes) bei atlikti įvairias operacijas su pačiais duomenimis (pvz. grupavimas, suliejimas, indeksavimas ir kt.). [Pan17]

Kaip nurodoma pandas aprašyme, pagrindinės stiprybės yra greitis ir naudojimo paprastumas, o tikslas – užpildyti spragą Python pasaulyje, kuri susijusi su duomenų analizės įrankių stoka. Taigi, kalbant apie šiame darbe nagrinėjamą dalykinę sritį, pandas daugiausiai naudos galėtų duoti atliekant analitinio pobūdžio duomenų transformacijas, tačiau neduotų tiek daug naudos kopijuojant duomenis ar bandant pritaikyti verslo logikos taisykles. Ši biblioteka tinka tiek Python 2.7, tiek Python 3.4-3.6 versijoms. [Pan17]

Ketvirtoji apžvelgta Python ETL biblioteka – **petl**. Šios bibliotekos dalykinė sritis labai panaši į nagrinėjamą šiame darbe – petl ji kiek platesnė – apima duomenų transformacijas (įskaitant ir validavimą bei valymą), duomenų struktūros keitimą duomenų saugyklose, kurios gali būti ne tik reliacinės duomenų bazės, bet ir kitokio pobūdžio saugyklos, pvz. XML failai. Iš esmės, šios bibliotekos paskirtis yra paprastesnis (šiuo atveju petl galime laikyti vidine dalykinės srities kalba) ETL uždavinių sprendimas naudojant Python kalbą. [P17]

petl funkcijų aibė gana didelė ir leidžia spręsti įvairiausius ir šiame darbe nagrinėjamus dalykinės srities uždavinius. Didžiausi bibliotekos privalumai – paprastumas (bibliotekos funkcijos gana ekspresyvios, paprastesnės transformacijos aprašomos gana nesudėtingai) ir plati funkcijų aibė. Dar vienas privalumas – ši biblioteka taip pat naudojasi ir tam tikru kitų bibliotekų galimybėmis, pvz. PyTables ar pandas. Visgi vertėtų paminėti, kad šis įrankis labiau skirtas

manipuliuoti duomenimis, neišnaudojant operatyviosios atminties stiprybių spartos aspektu, todėl jis gali efektyviai vykdyti ETL užduotis (taip pat ir dideliam duomenų kiekiui), jeigu sparta nėra pagrindinis prioritetas ir nesiekiama efektyvaus darbinės atminties išnaudojimo (ypač kalbant apie didesnį duomenų kiekį). Be to, sudėtingesnės transformacijos nebėra tokios ekspresyvios ir užrašomos kur kas sudėtingiau. Biblioteka reikalauja Python 2.6, 2.7 arba 3.4 versijos ir buvo ištestuota Linux ir Windows operacinėse sistemose. [P17]

Be šių keturių apžvelgtų bibliotekų, buvo rasta ir daugiau, bet jos savo funkcinėmis savybėmis, plėtojimu ar kitais aspektais buvo arba panašios, arba kiek mažiau tinkamos, todėl detaliau neapžvelgiamos. Tarp tokių galima priskirti – Odo (biblioteka skirta duomenų migravimui) [ODO17], PyTables (biblioteka darbui su HDF5 saugomiems duomenims) [PT17].

Apibendrinus, galima teigti, kad daugiausiai naudos kuriant dalykinės srities kalbos vykdymo modelį duotų petl, kadangi jos apimama dalykinė sritis taip pat apima ir šio darbo dalykinę sritį, paprastos transformacijos užrašomos gana ekspresyviai. Jei dalykinės srities kalba būtų įgyvendinama Python kalba, šią biblioteką būtų galima panaudoti kviečiant jos funkcijas. Kitu atveju, galima pasisemti idėjų, kaip galėtų būti atliekamas transformacijų vykdymas.

10. REIKALAVIMAI DALYKINĖS SRITIES KALBAI

Šiame darbe nagrinėjama dalykinė sritis – duomenų, saugomų reliacinėse SQL duomenų bazėse, transformavimas, klaidų tuose duomenyse paieška ir jų valymas. Norint sukonstruoti dalykinės srities kalbą, tinkamą šios srities uždaviniams spręsti, būtina sudaryti reikalavimų rinkinį, kurį turi tenkinti kuriama kalba.

Pirmiausia, literatūros apžvalgoje buvo nustatyta, kad bendrai šie dalykinės srities uždaviniai vadinami ETL procesu. Siekiant vykdyti šį procesą, būtina turėti galimybę pasiekti jame naudojamus duomenis, t. y. juos nuskaityti, transformuoti ir įkelti. Reiškia, pasinaudojant kalbos konstrukcijomis, turi būti galimybė aprašyti, kaip prisijungiama prie duomenų bazių. Taip išryškėja pirmasis reikalavimas:

1. Turi būti galima nurodyti prisijungimo duomenis prie duomenų šaltinių, tarpinės aplinkos (kurioje būtų galima vykdyti transformacijas prieš įkėlimą) ir adresatų reliacinių duomenų bazių.

Kai duomenys pasiekiami, jau galima vykdyti ETL procesą. Kalbant apie nuskaitymą ir įkėlimą – tai iš esmės yra vienas ir tas pats veiksmas – duomenų perkopijavimas iš vienos vietos į kitą. Tai reiškia, kad kalba privalo įgyvendinti ir šį reikalavimą:

2. Galima kopijuoti duomenis iš vienos vietos į kitą. Tai yra iš vienos lentelės į kitą, iš vienos eilutės į kitą, iš vienos duomenų bazės į kitą ir t. t.

Vis dėl to, kaip buvo nustatyta 1, 2 ir 3 skyriuje, ETL procesas neapsiriboja duomenų perkėlimu iš vienos vietos į kitą. Tuos duomenis taip pat prireikia transformuoti. 1.1. poskyryje buvo nustatyta, kad į transformacijos apibrėžimą įeina tiek duomenų vertimas iš vieno formato į kitą, tiek įvairių taisyklių pritaikymas jiems – apjungimas, apibendrintų reikšmių skaičiavimas, raktų kūrimas, rikiavimas, naujų duomenų kūrimas. Visgi, kalbant apie reliacines SQL duomenų bazes, verta pastebėti, kad čia transformacijų aibė kiek mažesnė nei kalbant apskritai apie transformacijas bet kur saugomiems duomenims, nes kai kurios iš jų neįgyvendinamos arba neprasmingos (pvz., duomenų rikiavimas, t. y. kokia eile jie išsaugoti duomenų bazėje, nes tai neefektyvu – išrinkti duomenis eilės tvarka galima ir SQL užklausomis). Iš to išplaukia reikalavimai:

3. Galima transformuoti duomenis naudojant įvairias funkcijas. Čia turima omenyje, kad galima atlikti tam tikrą veiksmų su duomenimis seką, kuria būtų galima apdoroti įvesties duomenis ir grąžinti tos funkcijos rezultatus bei taip įvykdyti transformacijas.

4. Galima parinkti numatytąsias reikšmes. Tai reiškia, kad galima iš anksto reikšmę apibrėžti kaip konstantą, kur ta reikšmė turi būti priskiriama transformacijų metu.

5. Galima sugeneruoti identifikuojančias reikšmes (raktus). Reikalavimo esmė tokia, kad turi būti galima sukurti unikalius raktus, kurie galėtų būti generuojami tiek eilės tvarka ir sudarytų seką, tiek atsitiktiniai identifikuojantys raktai (pvz. UUID [LMS05]).

6. Galima agreguoti duomenis. Pagal tam tikrų atributų reikšmes galima agreguoti kitų atributų iš tų eilučių duomenis, pritaikant norimą agregavimo funkciją, ir gauti galutinį rezultatą.

7. Galima apjungti (sulieti) duomenis iš kelių skirtingų šaltinių (tiek duomenų bazių, tiek lentelių) į tam tikrą nurodytą vietą. Taip pat galima apjungti kelių stulpelių duomenis į vieną stulpelį.

8. Galima atskirti (išskaidyti) duomenis, t. y. atvirkščias veiksmas apjungimui. Pvz., iš vieno stulpelio su data išskaidymas į kelis stulpelius su metais, mėnesiu ir diena.

Transformuojant duomenis galima gauti anomalijas, t. y. duomenis, kurie yra nevalidūs. Taip pat, jau ir pačiuose šaltiniuose esantys duomenys gali būti klaidingi. Vykdant transformacijas, reikia išlaikyti aukštą duomenų kokybę, t. y. dėl transformacijų ji neturėtų suprastėti. Dėl šios priežasties kalba turėtų turėti konstruktus, kuriais naudojantis būtų galima duomenis validuoti ir sutvarkyti. Be to, kalba turėtų turėti konstruktus, kuriais būtų galima nurodyti kokybės režimą – t. y. kaip elgsis programa esant nenumatytiems atvejams (kilus klaidoms, pasitaikius neleidžiamoms reikšmėms ir kt.), kas taip pat padėtų pagerinti duomenų kokybę. Tai atsispindi reikalavimuose:

9. Galima validuoti duomenis – išsiaiškinti, kur matomos anomalijos, bei pažymėti jas nurodytame faile. Validuoti galima pasitelkiant įvairias funkcijas (pvz., ieškant NULL reikšmių, neatitikimo verslo logikos taisyklėms ir kt.)

10. Galima išvalyti (sutvarkyti) duomenis, tai yra pašalinti ar sutvarkyti įvairias duomenų anomalijas – formato klaidas, NULL reikšmes, besidubliuojančius įrašus, raktų ryšių klaidas, verslo taisyklių neatitikimus ir kt.

11. Galima nurodyti kokybės režimą. Tai reiškia, kad atliekant transformacijas, kylant tam tikroms klaidoms ar vykstant nenumatytoms situacijoms, būtų galima nurodyti, kaip turėtų toliau būti vykdomos transformacijos. Pvz., jei kyla klaida dėl to, kad norima transformuoti kokią nors reikšmę (t. y. pirminė reikšmė privalo egzistuoti), bet randama „NULL“ reikšmė. Tokiu atveju, turi būti galima nurodyti, ar bandyti problemą išspręsti, ar praleisti eilutę, ar apskritai nutraukti procesą.

12. Galima nurodyti, kur saugoma pagalbinė registruojama informacija (angl. „logs“), kuri gali būti panaudota tiek sekti proceso eigai, tiek validacijos tikslais, klaidoms registruoti ir pan.

Tiek atliekant transformacijas, tiek validuojant duomenis, svarbu, kad kalba suteiktų galimybę aprašyti, kokie duomenys konkrečiai turėtų būti naudojami tiek kalbant apie jų struktūrą, tiek apie reikšmes. Tai apibrėžiama reikalavimuose:

13. Galima atlikti transformacijas konkrečiai pasirinktiems duomenims. Šis reikalavimas reiškia, kad nėra iš anksto numatyto transformacijų vykdymo tam tikrai duomenų aibei, bet naudotojas gali pats pasirinkti, ar transformuos pasirinktas eilutes, stulpelius, pavienes reikšmes ir t. t.

14. Galima filtruoti duomenis. Tai yra, galima išfiltruoti tiktai reikalingus duomenis pagal nurodytas logines sąlygas su tais duomenimis. Tai turėtų būti pasiekiama pasitelkiant logines operacijas.

15. Galima surasti duomenis pagal tam tikrus atributus, reikšmes juose, tiek siekiant atlikti transformacijas, tiek ir validuojant ar valant duomenis.

Galiausiai, reikia pastebėti, kad didelė dalis transformacijų (net apie 80% [ET17]) remiasi verslo logikos taisyklių taikymu, todėl kalba turi suteikti galimybę aprašyti transformacijas naudojant jas. Tai išreiškiama reikalavimu:

16. Galima įkelti papildomų funkcijų paketą (išorinę biblioteką, kuri pati savaime nėra kalbos dalis), kurio funkcijas būtų galima panaudoti norimoms transformacijoms atlikti. Reikalavimo esmė ta, kad vienu vertus, nors integruotų funkcijų aibė ir pati savaime turėtų būti pakankamai plati, bet dažnai reikalingos funkcijos remiasi tam tikra verslo logika, todėl turi būti sudaryta galimybė įkelti ir naudoti pačių apsibrėžtas funkcijas transformacijoms atlikti.

10.1. DataRules atitikimas reikalavimams

6 skyriuje pastebėta, kad kuriant kalbą, galima pasinaudoti DataRules konstrukcijomis. Dėl šios priežasties nagrinėjama, ar tai tikrai galima padaryti remiantis tuo, kiek ši kalba tenkintų sudarytų reikalavimų. Pastebima, kad ši kalba tenkintų (bent jau dalinai):

- 1 („Define data store“ konstrukcija) reikalavimą,
- 2 („Copy“ konstrukcija) reikalavimą,
- 3 (dalinai – ribotas transformacijų kiekis ir nenumatytos priemonės jam plėsti, ‚DataRules aprašytos transformacijos) reikalavimą,
- 4 („Default“ konstrukcija) reikalavimą,
- 5 (dalinai – negalimas UUID naudojimas, „Sequence“ konstrukcija) reikalavimą,
- 6 („Aggregate“ konstrukcija) reikalavimą,

- 7 (dalinai – negalimas apjungimas į vieną atributą, „Union“, „Join“, „Map“ konstrukcijos) reikalavimą,
- 8 (dalinai – galima tik iš vienos lentelės atributų juos išskaidyti į skirtingas lenteles, „Map“ konstrukcija) reikalavimą,
- 9 (dalinai – ribotas validavimo taisyklių kiekis ir nenumatytos priemonės jį praplėsti, „Validate“ konstrukcija) reikalavimą,
- 10 (DataRules transformacijos) reikalavimą,
- 12 (dalinai – galima nurodyti tik lentelę, kur saugomos anomalijos, bet negalima saugoti informacijos apie vykdymo klaidas, „Validate“ konstrukcija) reikalavimą,
- 13 (DataRules transformacijoms galima nurodyti transformuojamus atributus) reikalavimą,
- 14 („Filter“ konstrukcija) reikalavimą,
- 15 („Lookup“, „Match“ konstrukcijos) reikalavimą.

Dėl to, kad nemaža dalis reikalavimų tenkinama, nuspręsta pasiremti šia kalba ir modifikuoti bei papildyti ją pagal sudarytus reikalavimus bei, kur galima, konstrukcijas padaryti paprastesnėmis, kadangi tai viena svarbiausių DSL savybių, kuri įprasmina jų naudojimą lyginant su bendros paskirties kalbomis. Kuriama kalba pavadinta Relational DataRules.

11. DALYKINĖS SRITIES KALBOS KONSTRUKCIJOS

Šiame skyriuje aprašomos Relational DataRules dalykinės srities kalbos konstrukcijos. Paaiškinta, kaip jos turėtų būti naudojamos, kokie galimi apribojimai.

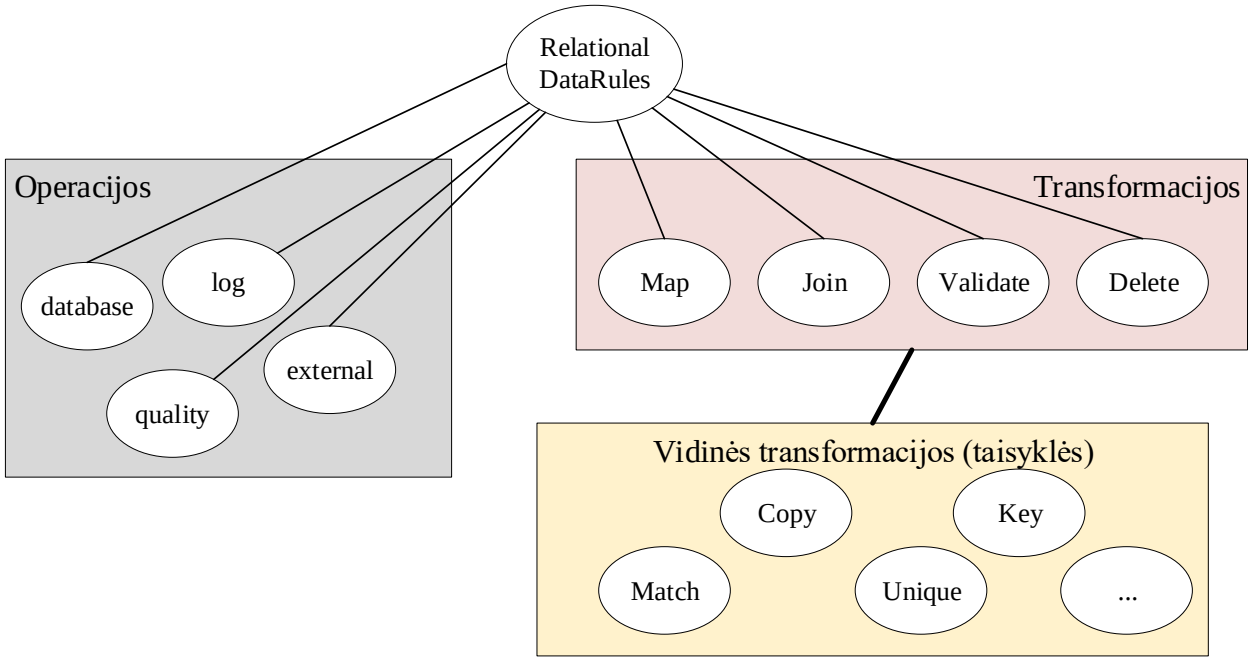
Pirmiausia, norint suprasti, kokio lygmens konstrukcijos ir kokiose situacijose jos naudojamos, išskiriamos esybės, kuriomis tos DSL konstrukcijos apibrėžiamos.

Transformacijos – tai funkcijos, kurios skirtos aprašyti, kaip duomenys turi būti transformuojami, validuojami ir valomi. Pvz.: „Map“, „Validate“ ir kt.

Vidinės transformacijos (taisyklės) – tai funkcijos, kurios aprašo transformacijų detales, specifiką. Jos aprašomos transformacijų viduje ir be jų negali būti naudojamos. Pvz.: „Copy“, „Filter“, „Unique“ ir kt.

Operacijos – tai funkcijos, skirtos aprašyti pagalbinę informaciją apie tai, kaip konfigūruojama transformacijų vykdymo aplinka. Pvz.: „database“, „log“ ir kt.

Transformacijų, vidinių transformacijų ir operacijų tarpusavio ryšys gali būti pavaizduotas paveikslėliu (žr. 3 pav.):



3. pav. Relational DataRules kalbos konstrukcijų klasifikacija.

11.1. Prisijungimas prie duomenų bazės

Prisijungimas prie duomenų bazės – operacija, skirta aprašyti duomenis, kurie reikalingi vykdant transformacijas prisijungimui prie duomenų bazės. Ji užrašoma taip:

- **database *myDatabase* set connection “*myDatabaseConnectionString*“**

Šiuo atveju „myDatabase“ yra bet koks pasirinktas duomenų bazės vardas, kuris bus naudojamas transformacijose norint identifikuoti duomenų bazę. Pasirinktas duomenų bazės

vardas turi būti unikalus kitų duomenų bazių atžvilgiu. Savo ruožtu „myDatabseConnectionString“ yra prisijungimo prie duomenų bazės nuoroda (angl. „connection string“).

Pvz.: „*database BankingDB set connection “server=127.0.0.1;User Id=user;password=superHardPwd;database=bankingDB”“*.

11.2. Įvykių registravimo failų nurodymas

Įvykių registravimas – operacija, kuria aprašomi registravimo failai, kurie skirti tam, kad juose būtų įrašomi klaidų pranešimai arba anomalijos, vykdant validavimo transformaciją. Jie nustatomi taip:

- **log *logType* set path “*myLogPath*“**

Šiuo atveju „logType“ įvykių registravimo failo paskirtis, t. y. kokia informacija bus registruojama. Galimos reikšmės:

- „errors“ – registruojamos vykdymo metu įvykusios klaidos,
- „validation“ – registruojamos validavimo metu rastos anomalijos.

„myLogPath“ nurodoma įvykių registravimo tekstinio (.txt) failo vieta kompiuteryje.

Pvz.: „*log errors set path “C:\Logs\transformationErrors.txt”“*.

Siekiant atlikti validavimą, būtina nurodyti anomalijų failą. Klaidoms registruoti, to atlikti nebūtina – tada klaidų pranešimai pagal nutylėjimą bus saugomi ten, iš kur leidžiama programa, su numatytuoju failo pavadinimu: „errors“ ir data.

11.3. Kokybės režimas

Kokybės režimas – operacija, kuria aprašoma, kaip turėtų elgtis programa klaidų metu. Ji atvaizduojama taip:

- **quality set *qualityMode***

Šiuo atveju „qualityMode“ yra kokybės režimas – tai yra, kaip elgsis programa ištikus klaidoms, kurių reikšmės gali būti:

- „skip“ – transformuojami duomenys praleidžiami, vykdymas tęsiamas,
- „fix“ – iškilusi problema bandoma spręsti (vykdoma alternatyvi transformacija, įskaitant numatytosios reikšmės priskyrimą, jei tokia nurodyta),
- „stop“ – transformacijos stabdomos, vykdymas nutraukiamas.

Pvz.: „*quality set skip*“.

Jei kokybės režimas nenurodomas, pagal nutylėjimą laikoma, kad pasirinktas režimas „stop“, tam, kad klaidos atveju būtų nutrauktas vykdymas ir naudotojas iškart, o ne tik po vykdymo pabaigos sužinotų, kad transformacijos (ar dalis jų) buvo įvykdytos nesėkmingai.

11.4. Funkcijų paketo įkėlimas

Funkcijų paketo įkėlimas – operacija, kuria nurodoma, kur randasi funkcijų bibliotekos failas, kuriame saugomos norimos funkcijos (vidinės transformacijos), reikalingos transformacijoms atlikti. Ji užrašoma taip:

- **external functions set path “*myFunctionsPath*”**

„myFunctionsPath“ nurodoma funkcijų bibliotekos (pvz., „dll“ failas, jei kalba įgyvendinta naudojant .NET technologijas) failo vieta kompiuteryje.

Pvz.: „*external functions set path “C:\Libraries\myFunctions.dll”*“.

Norint naudoti šias funkcijas, prieš funkcijos vardą reikia nurodyti žodį „external“. Funkcijų bibliotekoje funkcijos turi būti aprašytos analogiškai kaip kitos kalbos konstrukcijos, kurių vykdymas nusakytas 14 skyriuje.

11.5. Susiejimo transformacija

Susiejimo transformacija skirta susieti dviejų lentelių duomenis, perkeliant ir transformuojant juos iš vienos lentelės į kitą. Ji apibrėžiama taip:

- **Map *table1* to *table2 innerTransformations* end**

Šiuo atveju „table1” ir „table2” yra lentelių pavadinimai, kurios susiejamos. „innerTransformations“ – vidinės transformacijos, kurios vykdomos susiejimo metu. Tai gali būti „Default“, „Key“, „Copy“, „Filter“, „Aggregate“, „Match“ bei išorinės funkcijos.

Pvz.:

„*Map UserAddress to UserStreets*

Key Id

Copy UserId to UserId

Copy Street to Street

end“.

Transformacija reikalinga siekiant atlikti veiksmus įrašų lygmenyje. Ji negali būti naudojama be vidinių transformacijų.

Kai naudojama agregavimo transformacija, kartu su ja gali būti naudojamos tik „Default“ ir „Filter“ vidinės transformacijos.

11.6. Kopijavimas

Kopijavimas – viena iš galimų „Map“ transformacijos vidinių transformacijų. Ji užrašoma taip:

- **Copy *attribute1* to *attribute2 alternative transformation***

Šiuo atveju „attribute1“ yra lentelės, iš kurios reikmės nukopijuojamos, atributas, o „attribute2“ yra lentelės, į kurią reikšmės nukopijuojamos, atributas. „alternative“ rodo, kad bus nurodoma alternatyvi transformacija, o „transformation“ ta transformacija pateikiama su parametrais. Alternatyvi transformacija neprivaloma.

Pvz.: „*Copy UserName to Name*“.

11.7. Rakto priskyrimas

Rakto priskyrimas – viena iš galimų „Map“ transformacijos vidinių transformacijų. Ji užrašoma taip:

- **Key *keyType startNumber* to *attribute alternative transformation***

Šiuo atveju, „keyType“ yra rakto tipas, kuris gali įgyti dvi reikšmės:

- „number“ – raktas nusakomas skaičiumi (iš eilės),
- „uuid“ – raktas nusakomas universaliu unikaliu identifikatoriumi (UUID).

Kitos kintamos dalys:

- „startNumber“ – jei pasirenkamas „number“ rakto tipas, galima nurodyti nuo kokio skaičiaus pradedamas numeravimas. Ši konstrukcijos dalis gali būti praleista.
- „attribute“ – atributas, kuriam priskiriama reikšmė.
- „alternative“ rodo, kad bus nurodoma alternatyvi transformacija. Neprivaloma.
- „transformation“ alternatyvi transformacija pateikiama su parametrais.

Pvz.: „*Key number Id*“.

11.8. Numatytosios reikšmės priskyrimas

Numatytosios reikšmės priskyrimas – viena iš galimų „Map“ transformacijos vidinių transformacijų. Ji užrašoma taip:

- **Default *“value”* to *attribute alternative transformation***

Čia „value“ yra numatytoji reikšmė, o „attribute“ yra atributo vardas, kuriam ta reikšmė bus priskiriama. „alternative“ rodo, kad bus nurodoma alternatyvi transformacija, o „transformation“ ta transformacija pateikiama su parametrais.

Pvz.: „*Default “Male” to Gender*“.

11.9. Susiejimas pagal paiešką

Susiejimas pagal paiešką – viena iš vidinių transformacijų, kurios pagalba galima priskirti naujas reikšmes toms reikšmėms, kurios randamos adresato lentelėje pagal atitinkamus atributus. Ji užrašoma taip:

- **Match *attribute1* with *attribute2***

  Čia „attribute1“ yra atributas iš šaltinio lentelės, o „attribute2“ – iš adresato lentelės.

  Pvz.: „*Match Id with Id*“.

11.10. Filtravimas

Filtravimas – viena iš vidinių transformacijų, kuria naudojantis galima susieti tik filtruotus duomenis. Ji užrašoma taip:

- **Filter “*logicOperationWithParams”***

Čia „logicOperationWithParams“ – loginė operacija su reikalingais parametrais, kuri grąžina loginę reikšmę. Tai tokios operacijos, kaip „=“ (lygu), „>“ (daugiau), „<“ (mažiau), „AND“ (ir) ir kt.

Pvz.: „*Filter ‘‘Name = ‘John’’*“.

11.11. Suliejimas

Suliejimas – viena iš „Map“ vidinių transformacijų, kuria naudojantis galima sulieti kelių atributų reikšmes į vieną. Ji užrašoma taip:

- **Merge “*separator*” *attributes* to *attribute alternative transformation***

Čia „separator“ yra skirtukas, kuris įterpiamas suliejant reikšmes (jei nenorima įterpti skirtuko, reikėtų nurodyti jį tuščią). „attributes“ – atributai, kurie suliejami. „attribute“ – atributas, kuriame patalpinamos sulietos reikšmės. „alternative“ rodo, kad bus nurodoma alternatyvi transformacija, o „transformation“ ta transformacija pateikiama su parametrais.

Pvz.: „*Merge “ ” Name, Surname to FullName*“.

11.12. Atskyrimas

Atskyrimas – viena iš vidinių „Map“ transformacijų, atvirkštinė suliejimo transformacijai. Ji užrašoma taip:

- **Split “*separator*” *attribute* to *attributes* end alternative transformation**

Čia „separator“ – skirtukas, pagal kurį atskiriamos reikšmės. „attribute“ – atributos, iš kurio atskiriamos reikšmės. „attributes“ – atributai, į kuriuos patalpinamos atskirtos reikšmės.

„alternative“ rodo, kad bus nurodoma alternatyvi transformacija, o „transformation“ ta transformacija pateikiama su parametrais.

Pvz.: „*Split “ ” FullName to Name, Surname end*“.

11.13. Agregavimas

Agregavimas – viena iš vidinių „Map“ transformacijų, kuri pagal pasirinktą funkciją agreguoja duomenis ir tuo būdu keičia įrašų skaičių. Ji užrašoma taip:

- **Aggregate *aggregateFunctionWithParams* to *attributeDest***

Čia „aggregateFunctionWithParams“ – funkcija su jai reikalingais parametrais, pagal kurią galima agreguoti duomenis. Tarp tokių funkcijų – „Sum“ (sumai), „Count“ (pasikartojimų skaičiui), „Average“ (vidurkiui), „Min“ (minimaliai reikšmei), „Max“ (maksimaliai reikšmei). Taip pat prie jų prisiskaito ir „on“, kuri nurodo, pagal kurį atributą agreguojama. T. y. susiejimo transformacija naudojanti agregavimą privalo nurodyti tiek agreguojamus atributus, tiek atributus, pagal kuriuos agreguojama. „attributeDest“ – atributas, kuriame išsaugomi rezultatai.

Pvz.: „*Aggregate Sum Money to Expenses*

Aggregate on Month to Month“.

Jei naudojama agregavimo transformacija, kartu su ja gali būti naudojamos tik „Filter“ ir „Default“ transformacijos.

11.14. Sujungimas

Sujungimas – transformacija, kurios paskirtis sujungti dvi lenteles į vieną. Ji užrašoma taip:

- **Join *table1* with *table2* to *table3 attributeJoins* Match *Merge attributeMatches end***

Čia „table1“ ir „table2“ – lentelės, kurios sujungiamos. „table3“ – lentelė, kurioje patalpinama jungtinė lentelė. „attributeJoins“ – nurodoma, kokie atributai vienoje lentelėje atitinka atributus kitoje. Jei tuose atributuose skirtingos reikšmės, paimama to atributo reikšmė, kuri yra ne „null“ arba kuris atributas aprašytas pirmiau. Ši dalis gali būti praleista, tada bus apjungiami visi atributai. „Merge“ – žodis, kuriuo nurodoma, ar norima pašalinti dublikatus, jei tokių yra. Šį žodį galima praleisti, jei dublikatų šalinti nenorima. „attributeMatches“ – atributai, pagal kuriuos žiūrima, kurie įrašai vienoje lentelėje atitinka įrašus kitoje. Ši dalis taip pat gali būti praleista – tada laikoma, kad jokie įrašai nepersidengia.

Pvz.:

„*Join ShopA.Client with ShopB.User to Shop.User*

ShopA.Client.Name with ShopB.User.Name to Shop.User.Name

ShopB.User.CardNo with ShopB.User.CardNo to ShopB.User.CardNo

Match

ShopA.Client.Id with ShopB.User.Id

end“.

11.15. Validavimas

Validavimas – transformacija, kurios paskirtis validuoti duomenis tikrinant, ar nėra nenumatytų anomalijų. Šiai transformacijai kaip ir „Map“ reikalingos vidinės transformacijos (taisyklės) pagal kokius kriterijus reikės duomenis validuoti. Rezultatai išsaugomi faile, kuris nurodytas išsaugant įvykių registravimo failą. Validavimo transformacija užrašoma taip:

- **Validate *MyTable Rules* end**

Čia „MyTable“ – pasirinktos lentelės, kuri validuojama, vardas. „Rules“ – taisyklės, pagal kurias lentelė validuojama. Jų turi būti bent viena. Galimos taisyklės – „Unique“ (unikalumui patikrinti), „NotNull“ (patikrinti, ar nėra „null“ reikšmių) ir t. t. Galima naudoti ir išorines („external“) funkcijas.

Pvz.: „*Validate User Unique IdentificationNumber end*“.

11.16. Unikalumas

Unikalumas – vidinė validavimo transformacija, kurios paskirtis patikrinti, ar visos atributo reikšmės unikalios. Ji užrašoma taip:

- **Unique *attribute***

Čia „attribute“ – atributo vardas, kuris tikrinamas.

Pvz.: „*Unique Id*“.

11.17. Nuorodos patikrinimas

Nuorodos patikrinimas – validavimo vidinė transformacija, kurios paskirtis patikrinti, ar egzistuoja reikšmė kitoje lentelėje pagal nurodytus atributus. Ji užrašoma taip:

- **Reference *attribute* in *otherTableAttribute***

Čia „attribute“ – tikrinamos lentelės atributas, pagal kurio reikšmę turi būti rastas įrašas kitoje lentelėje. „otherTableAttribute“ – kitos lentelės atributas, kuris užrašomas nurodant duomenų bazę, lentelė ir atributą atskiriant taškais.

Pvz.: „*Reference AddressId in MyDB.Address.Id*“.

11.18. Reikšmės egzistavimo patikrinimas

Reikšmės egzistavimo patikrinimas – validavimo vidinė transformacija, kurios paskirtis patikrinti, ar atributas turi kokią nors reikšmę. Ji užrašoma taip:

- **NotNull *attribute***

  Čia „attribute“ – atributo vardas, kuris tikrinamas.

  Pvz.: „*NotNull Name*“.

11.19. Trynimas

Trynimas – transformacija, kurios paskirtis ištrinti nurodytus duomenis. Tai gali būti tiek tiek lentelės, tiek eilučių, tiek atributų trynimas. Tai turi būti užrašoma taip:

- **Delete *type name filters***

Čia „type“ – trynimo tipas, kuris gali būti „table“ (kai norima ištrinti visą lentelę), „rows“ (kai norima ištrinti lentelės reikšmes) ir „attribute“ (kai norima ištrinti pasirinkto atributo reikšmės). „name“ šiuo atveju gali būti lentelės ar atributo pavadinimas. „filters“ – „Filter“ transformacijos, skirtos išfiltruoti, kurie duomenys ištrinami. Šis parametras neprivalomas.

Pvz.: „*Delete rows UserDB.Users Filter “Active = 0”*“.

12. DALYKINĖS SRITIES KALBOS NAUJOVĖS

Magistro darbo literatūros apžvalgos metu buvo pastebėta, kad nagrinėjamai dalykinei sričiai nebūtina kurti visiškai naujos kalbos – galima pasinaudoti jau sukurta DataRules [Tub15] kalba, ją modifikuoti pagal sudarytą reikalavimų rinkinį ir pritaikyti duomenų transformavimui reliacinėse duomenų bazėse saugomiems duomenims. Taip pat, pastebėta, kad kai kurios kalbos konstrukcijos gali būti pagerintos (supaprastintos). Šiame skyriuje aprašomi svarbiausi dalykai, kurie buvo pakeisti ar pridėti kuriant Relational DataRules kalbą.

Svarbiausios Relational DataRules naujovės lyginant su DataRules:

- Pasikeitė prisijungimo duomenų prie duomenų šaltinio (šiuo atveju SQL duomenų bazės) nurodymas. Didžiausi skirtumai pastebimi tame, kad nereikia nurodyti duomenų saugyklos tipo (nes dirbama su reliacinėmis SQL duomenų bazėmis). Taip pat nereikia nurodyti duomenų bazės lentelių ir atributų, kadangi duomenų saugyklos turi apibrėžtą schemą. Tai ženkliai supaprastina operaciją.
- Relational DataRules nuspręsta registravimo failus nurodyti atskira komanda, ko nebuvo galima padaryti DataRules. Ši operacija pridėta, kadangi siekiama naudoti daugiau registravimo galimybių (t. y. ne vien registruojant anomalijas validavimo metu, kaip tai daroma, naudojant DataRules, bet ir klaidas).
- Atsirado kokybės režimo nurodymo galimybė. Šios galimybės DataRules neturėjo, tačiau išnagrinėjus dalykinę sritį, nuspręsta, kad kokybės režimo nurodymas yra reikalinga operacija atliekant transformacijas ir norint užtikrinti aukštą duomenų kokybę.
- Taip pat kartu su kokybės režimo nurodymo galimybe, atsirado ir galimybė nurodyti alternatyvias transformacijas, jei pagrindinės būtų nesėkmingos ir pasirinktas toks režimas, kuris bandytų problemą išspręsti.
- Atsirado galimybė naudoti išorines („external“) funkcijas. Ji reikalinga, nes kaip buvo nustatyta sudarant reikalavimus kalbai, dažnai transformacijos remiasi verslo logikos taisyklėmis ir numatytų transformacijų nepakanka. Dėl to, pasinaudojant šia galimybe Relational DataRules aprėmiama dalykinė sritis gali būti stipriai praplėsta įtraukiant daugybę reikalingų vidinių transformacijų.
- Modifikuota „Sequence“ transformacija. Jai suteiktas naujas pavadinimas – „Key“. Esminis skirtumas yra tai, kad naudojant „Sequence“ transformaciją buvo galima nurodyti, kad atribute bus saugomi skaičiai eilės tvarka. „Key“ ši savybė išliko, tačiau taip pat pridėta galimybė naudoti unikalius identifikatorius (UUID).
- DataRules kalboje buvo naudojama transformacija „Lookup“, tačiau pastebėta, kad jos sudėtingumas perteklinis (naudojamos kelios transformacijos, kai užtenka vienos tam

pačiam veiksmui įvykdyti). Todėl nuspręsta „Lookup“ atsisakyti ir naudoti supaprastintą transformaciją „Match“, kuri funkcionalumo prasme geba atlikti tuos pačius dalykus.

- Pasikeitė „Filter“ transformacijos naudojimas. DataRules kalboje, filtravimą reikėjo naudoti kartu su loginėmis funkcijomis, kurios užrašomos žodžiais, net jei transformacija paprasčiau ir aiškiau galėtų būti užrašyta pasitelkiant loginius operatorius („<“, „>“, „=” ir kt.). Dėl to, Relational DataRules nuspręsta naudoti juos.
- „Join“ – dar viena modifikuota transformacija. Lyginant su DataRules, tai „Union“ ir „Join“ transformacijų mišinys. Pastebėta, kad „Join“ ir „Union“ sintaksiškai užrašomos labai panašiai, todėl nuspręsta, kad šiam tikslui užtenka vienos transformacijos. Naujoje „Join“ transformacijoje galima nurodyti, kuris atributas turi viršenybę, galimas lankstesnis atributų nurodymas. Nors transformacija vizualiai panašesnė į DataRules aprašytą „Union“, parinktas „Join“ vardas, kadangi tai veiksmažodis, kaip ir kitos kalboje naudojamos transformacijos, todėl natūraliau naudojamas aprašant taisykles. Ši transformacija geba atlikti duomenų apjungimą tiek į plotį (kaip „Join“), tiek į ilgį (kaip „Union“).
- Šiek tiek pasikeitė ir validavimo transformacija. Esminis skirtumas tarp šios transformacijos ir DataRules aprašyto validavimo yra tai, kad nebereikia nurodyti, kur saugomos anomalijos (tai daroma naudojant „Log“ operaciją). Be to, nereikia bereikalingo „Validate“ žodžio kartojimo aprašant vidines transformacijas, kas tik apsunkina skaitomumą, bet naudos neatneša.
- Pasikeitė „Uniqueness“ transformacija – jos pavadinimas pakeistas į „Unique“ bei atsisakyta palyginimo funkcijos (todėl, nes pagal nutylėjimą naudojamas SQL palyginimas).
- „Reference“ taip pat supaprastinta pašalinant palyginimo funkciją.
- Skirtingai nei DataRules, kurioje naudojamos „Clean“ (visos lentelės duomenų išvalymui) ir „Drop“ (lentelės ištrynimui) transformacijos, čia pasirinkta naudoti vieną transformaciją „Delete“, kuri geba atlikti abiejų transformacijų funkciją. Taip pat, ji funkcionalumo prasme yra netgi platesnė, nes naudojant ją galima pasitelkti vidinę filtravimo transformaciją ir išrinkti, ką norima ištrinti (tiek eilučių, tiek atributų lygmenyje), kas padeda lengviau sutvarkyti duomenis ar išvalyti anomalijas.
- Pridėta nauja transformacija „Merge“ (atributų suliejimui).
- Pridėta nauja transformacija „Split“ (atributų atskyrimui).
- Pridėta nauja transformacija „NotNull“ (reikšmės egzistavimo patikrinimui).
- Atsisakyta transformacijų, kurių, remiantis sudarytu reikalavimų rinkiniu nereikia – „Expression“, „Fold“, „Unfold“.

12.1. Relational DataRules gramatikos palyginimas

Siekiant išsiaiškinti, kokios Relational DataRules kalbos gramatikos savybės ir kaip jos skiriasi lyginant su kitomis kalbomis, bei norint įsitikinti, kad kalbai būdingos gerosios DSL savybės (pvz. paprastumas, išmokstamumas ir kt.), ji buvo ištirta pagal [CČF+09] nusakytą metodiką naudojant „Grammar Metrics tool“ (arba sutr. „gMetrics“). 1 lentelėje pavaizduota, kokie gauti Relational DataRules gramatikos įverčiai bei kaip jie skiriasi nuo kitų kalbų gramatikų. Relational DataRules įverčiai gauti „gMetrics“ įrankiui paduodant ANTLR gramatikos failą bei norimą parametrą (pvz. „TERM“). „LAR/LRS“ įvertis gautas „LAT“ normalizuojant „LRS“ įverčiu.

Pateiktų metrikų reikšmės:

- TERM – gramatikos dydį rodantis rodiklis, kuris rodo, kiek terminių simbolių kalboje;
- VAR – gramatikos dydį nusakantis rodiklis, kuris rodo, kiek neteriminių simbolių kalboje. Didesnis skaičius rodo, kad kalbos palaikymas sudėtingesnis;
- MCC – McCabe‘o ciklinis sudėtingumas, rodiklis, nusakantis alternatyvų kiekį neterminiams simboliams. Didesnis skaičius rodo, kad daugiau pastangų reikia gramatikos testavimui atlikti bei didesnė analizavimo klaidų tikimybė;
- AVS – vidutinis dešinės pusės taisyklių dydis (ANTLR gramatikos taisyklėse, tai, kas yra po dvitaškio). Didesnis skaičius rodo, kad gramatika sunkiau skaitoma, o konstrukcijos analizatoriaus atpažįstamos lėčiau;
- HAL – „Halstead effort“, rodiklis, kuris rodo kiek pastangų reikia sukurti ir modifikuoti gramatiką. Mažesnis skaičius rodo, kad tai padaryti lengviau;
- LRS – metrika, nusakanti būsenų skaičių LR automatone (t. y. analizatoriuje, kuris atpažįsta konstrukcijas „iš apačios į viršų“). Didesnis skaičius rodo didesnį gramatikos sudėtingumą;
- LTPS – rodiklis, kuris nusako, kiek skirtingų terminių simbolių porų galima sudaryti kalboje. Didesnis skaičius rodo didesnį sudėtingumą;
- LAT/LRS – rodiklis, gautas LAT (vidutinis LR automatono būsenų skaičius, galintis pakeisti esamo terminio simbolio reikšmę) normalizuojant LRS reikšme. Jis rodo kalbos konstrukcijų apribojimus, t. y. kaip skirtingos konstrukcijos gali būti naudojamos kartu. Didesnis skaičius rodo, kad skirtingi terminiai simboliai gali būti naudojami kartu su mažiau apribojimų.
- SS – kalbos daugiažodiškumas. Didesnis skaičius rodo, kad kalba turi daugiau žodžių.

1 lentelė. Relational DataRules gramatikos palyginimas. Kitų kalbų duomenys paimti iš [CČF+09].

	Kalba	TERM	VAR	MCC	AVS	HAL	LRS	LTPS	LAT/LRS	SS
DSL kalbos	Relational DataRules	38	24	1,5	5,67	13,81	298	118	0,11	3,72
	FDL	14	6	2,17	6,5	2,63	115	47	0,18	2,53
	EBNF	12	7	1,71	3,29	1,17	129	102	0,49	1,59
Bendros paskirties kalbos	Ruby 1.8.5	88	83	2,61	4,74	54,44	13474	3200	0,26	1,47
	ANSI C	83	66	2,21	5,09	42,34	2512	1777	0,18	1,73
	Python 2.5	85	86	2,22	4,93	63,41	3909	1576	0,17	1,73
	Java 1.6	98	110	2,46	5,96	122,66	6244	2691	0,18	2,04

Kaip matoma 1 lentelėje, terminių (TERM) ir neterminių (VAR) simbolių skaičius daug mažesnis lyginant su bendros paskirties kalbomis (nors kiek ir didesnis nei kitų DSL kalbų), kas reiškia, jog kalba nedidelė, o jos kalbos gramatikos palaikymas paprastas, kas būdinga visoms DSL.

Gramatikos McCabe‘o ciklinis sudėtingumas (MCC), kuris rodo, kiek egzistuoja galimų alternatyvų neterminiams simboliams, yra pats mažiausias tarp palygintų gramatikų, o tai reiškia, kad gramatiką lengva ištestuoti ir egzistuoja mažiausia analizavimo klaidų tikimybė.

Gramatikos skaitomumas ir potenciali analizavimo sparta (AVS) lyginant su kitomis gramatikomis panaši. Čia nepastebima jokia tendencija tarp dalykinės ir bendros paskirties kalbų.

Savo ruožtu „Halstead effort“ (HAL) rodiklis rodo, kad gramatiką sukurti ir modifikuoti tiek pastangų, kiek reikia bendros paskirties kalbos gramatikai sukurti ir palaikyti nereikia, bet tai kiek sudėtingiau padaryti lyginant su kitomis DSL.

LR pasikeitimo (būsenų) sudėtingumas (LRS, LTPS) daug kartų mažesnis nei bendros paskirties kalbų, kas rodo teigia, kad kalba paprasta. O LAT/LRS metrika rodo, kad kalba pati griežčiausia terminių simbolių naudojimo aspektu. Viena vertus, tai rodo kad kalba mažiau lanksti, tačiau kita vertus, ją išmokti lengviausia, kadangi tie patys terminiai simboliau rečiau naudojami skirtingose taisyklėse, todėl reikia atsiminti mažiau taisyklių, kada kas gali būti panaudotas.

Taip pat galima pastebėti, kad kalba daugiažodiškiausia (SS). Nors tai galėtų rodyti, kad kalba galimai sudėtingesnė ar neoptimizuota, tačiau šį rodiklį galima paaiškinti tuo, kad, viena vertus, pačių dalykinės srities esybių daug, kita vertus, kalboje naudojamos įvairūs jungtukai ar prielinksniai (pvz. „to“, „from“ ir kt.), kurios kaip tik suteikia panašumo į žmogiškąją kalbą ir supaprastina naudojimą ta kalba rašančiam asmeniui.

Pastebima, kad į palyginimą būtų buvę tikslinga įtraukti ir DataRules, kadangi pagal ją buvo kuriama Relational DataRules kalba ir tai padėtų ištirti kaip Relational DataRules kalbinės savybės skiriasi nuo savo pirmtakės, t. y. ar atliktos modifikacijos nesuprastino kalbos savybių. Vis dėlto,

to nebuvo galima atlikti, nes [Tub15] pateikiama kalbos gramatika ANTLR formatu turi tam tikrų netikslumų – joje visos taisyklės apibrėžiamos kaip lekserio taisyklės (nėra analizatoriaus taisyklių), todėl ir tokios gramatikos įverčiai negali būti vertinami.

Apibendrinus, galima pastebėti, kad kalbos gramatikai būdingos savybės panašesnės į kitų dalykinės srities kalbų gramatikų savybes. Ją lengva palaikyti, analizuoti. Kai kurie įverčiai didesni nei kitų DSL, tad tai gali rodyti, kad pati dalykinė sritis sudėtingesnė (ar platesnė). Verta pažymėti, kad kalbą lengva išmokti, kas yra svarbu, nes ji skirta specifinei dalykinei sričiai. Kadangi negalimas tiesioginis gramatikos palyginimas su DataRules, sunku nustatyti, kiek naujos konstrukcijos ar senų konstrukcijų pagerinimas, modifikavimas ar supaprastinimas galėjo pasiekti geresnius ar prastesnius įverčius kuriant Relational DataRules gramatiką, tačiau lyginant su kitų kalbų gramatikomis, galima teigti, kad gerosios DSL savybės išlaikomos.

13. DALYKINĖS SRITIES KALBOS VYKDYMAS

13.1. Kalbos konstrukcijų atpažinimas

Relational DataRules bus vykdoma naudojant šios dalykinės srities kalbos interpretatorių. Pats vykdymas susideda iš 2 dalių: kalbos konstrukcijų atpažinimo ir aprašytų taisyklių vykdymo.

Kalbos konstrukcijų atpažinimui naudojamas ANTLR (v4.7) [A18] įrankis. Šis įrankis pagal sudarytą gramatiką sukuria kalbos lekserį (angl. „lexer“), analizatorių (angl. „parser“), o remdamasis jais, sudarinėja abstrakčius sintaksės medžius (angl. „abstract syntax trees“ arba AST), kuriuos „apeinant“ galima vykdyti atpažintą bendros paskirties kalba aprašytą programinį kodą su reikalingais parametrais. Tai galima iliustruoti supaprastintu Relational DataRules pavyzdžiu.

Tarkime aprašome kopijavimo taisyklę naudodami „Copy“ funkciją:

- **Copy myDB1.Name to myDB2.FullName**

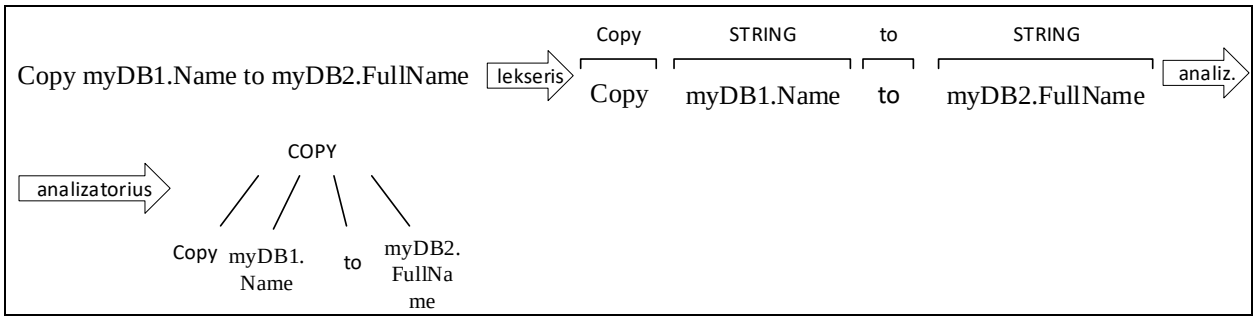
Supaprastintas gramatikos failas gali būti aprašytas taip:

```
/*
 * Parser Rules
 */
COPY: ‘Copy’ STRING ‘to’ STRING;
/*
 * Lexer Rules
 */
STRING : [a-z]+;
WHITESPACE : ‘ ‘ -> skip;
```

Taigi, remiantis šiuo pavyzdžiu, pirmiausiai kalbos lekseris skaito simbolius, o aptikdamas tarpus žino, kad tai atskiri žodžiai. Kitaip sakant, nagrinėdamas nurodytą tekstą, lekseris išskiria žodžius: „Copy“, „myDB1.Name“, „to“, „myDB2.FullName“. Tada analizatorius suformuoja abstraktų sintaksės medį (žr. 4 pav.). Šis medis „pereinamas“ naudojant ANTLR sugeneruotą klausytoją (angl. „listener“) arba lankytoją (angl. „visitor“) ir vykdomas bendros paskirties programinis kodas.

Klausytojo atveju (kuris ir naudojamas Relational DataRules, nes jo įgyvendinimas paprastesnis, o funkcionalumas pakankamas) „einant“ per kiekvieną medžio viršūnę iškviečiama tam skirta funkcija, kuri turi būti įgyvendinta priklausomai nuo to, kokią funkciją turi atlikti. „Copy“ atveju iškviečiama šiam tikslui aprašyta funkcija, kuri įdės ją į vidinių transformacijų

sąrašą, kurį vykdant bus sukonstruota SQL užklausa „myDB1.Name“ išrinkimui ir SQL užklausa įterpimui į „myDB2.FullName“.



4 pav. Copy taisyklės atpažinimas naudojant ANTLR.

Tokiu būdu pagal aprašytą lekserį ir analizatorių gramatikoje, konstruojami medžiai ir atpažįstamos visos Relational DataRules kalba aprašytos konstrukcijos ir vykdomos transformacijos (bendros paskirties kalba aprašytas programinis kodas panaudojant parametrus). Pilna šios kalbos gramatika nurodyta 1 priede.

Taip pat, nors literatūros analizės metu buvo pastebėta, kad egzistuoja ETL bibliotekos, kurias būtų galima panaudoti, kuriant algoritmus pastebėta, kad bibliotekų funkcijų kodo generavimas pagal DSL konstrukcijas pernelyg sudėtingas bei prarandama kodo, kuris vykdomas bibliotekos viduje, kontrolė, todėl nuspręsta jas naudoti tik kaip žinių šaltinį.

13.2. Susiejimo transformacijos vykdymas

Kai interpretatorius perskaito žodį „Map“, žinoma, kad bus vykdoma susiejimo transformacija. Šią transformaciją galima vykdyti dalimis lygiagrečiai.

Pirmiausia, naudojantis kalbos atpažinimo įrankiu, nustatoma, kokios vidinės transformacijos bus vykdomos (atpažįstant vidines transformacijas, jos patalpinamos į sąrašą, per kurį bus pereinama). Jei nebus vykdoma agregavimo transformacija, gaunama, kiek įrašų yra lentelėje, kad būtų žinoma, kiek iteracijų reikės atlikti nuskaitant po 1000 įrašų.

Tada lygiagrečiai pradedamos vykdyti iteracijos. Lygiagretus vykdymas saugus, nes skaitomi skirtingi duomenys, o po to ir įterpiami. Kiekvienos iteracijos metu pirmiausiai iš duomenų bazės išrenkama tiek įrašų, koks dalies dydis nurodytas ir tie įrašai patalpinami objektuose (turima omenyje objektinės kalbos objektus). Tada išrinktiems įrašams vykdomos vidinės transformacijos. Galiausiai po vidinių transformacijų naujai sukurti objektai po vieną įkeliami į kitą lentelę (masinis įkėlimas viena SQL užklausa negalimas).

Pseudokodu tai atrodytų taip:

```
currentConstruction := "Map“
partSize := 1000;
recordCount := partSize;
hasAggregate := innerTransformations.Contains(“Aggregate“);
if (not hasAggregate)
begin
   recordCount := executeSql(“SELECT Count(*) FROM {table1} {filters}”)
end

for (i := 0; i < recordCount; i += partSize)
   runInParallel
   begin
      newObjects := [];
      if (hasAggregate)
         then
         begin
            newObjects := applyAggregation(table1);
         end
         else
         begin
            objects := executeSql(“SELECT * FROM {Table1} {filters} LIMIT {partSize}
                                    OFFSET {i}”);
            newObjects := applyInnerTransformations(objects);
         end
   foreach object in newObjects
   begin
   try
      executeSql(“INSERT {object.main} INTO {table2}”)
   catch (error)
      begin
         errorLogger.Log(error, object.main);
         if (qualityMode = “stop”) then terminate();
        if (qualityMode = “skip”) then continue;
        try
         executeSql(“INSERT {object.alternate} INTO {table2}”)
```

```
        catch
        begin
            errorLogger.Log(error, object.alternate);
        end
    end
  end
end
currentConstruction := null;
```

Jeigu naudojama „Filter“ vidinė transformacija, tada modifikuojamos SQL užklausos pridedant „WHERE“ sąlygą (pseudokodu atitinka „filters“).

Kai susiejimo transformacija naudoja agregavimą, tada ji vykdoma kiek kitaip, kaip tai jau matoma pseudokodu aprašytame algoritme. Tam pasitelkiamas ELT pobūdžio veikimas, kai transformacijos vykdomos pačioje duomenų bazėje, išskyrus tai, kad įkėlimo procesas remiasi ETL, nes rezultatai vis tiek įkeliami paskiausiai. Kitaip tariant, atliekant agregavimą, nereikia ištraukti duomenų ir atlikti skaičiavimus, nes šie veiksmai jau gali būti atlikti pačioje duomenų bazėje. Taigi, kadangi visi skaičiavimai atliekami duomenų bazėje, vykdymas tampa nuoseklus (vykdoma tik viena iteracija).

Kai vykdoma „applyAggretation()“ ar „applyInnerTransformations()“ vykdomos vidinės transformacijos pereinant per jų sąrašą ir kviečiant jų funkcijas (žr. vidinių transformacijų vykdymą) su reikalingais parametrais, kurie gaunami nuskaičius su ANTLR.

Kalbant apie lygiagretų transformacijos vykdymą, „runInParallel“ žodis reiškia, kad tai, kas yra jo veikimo lauko viduje (prasideda „begin“ ir baigiasi „end“), turėtų būti vykdoma lygiagrečiai visoms iteracijoms tarpusavyje. Pvz.: C# atveju, tam galima pasitelkti „Parallel.For()“ funkciją ir reikalingą vykdymo logiką aprašyti tarp „{}“ (žr. priedą nr. 2); jei kalba neturi tokios funkcijos, galima kiekvieną iteraciją vykdyti skirtingoje gijoje apribojant gijų skaičių remiantis tuo, kiek branduolių turi procesorius, t. y. gijų skaičius neturėtų viršyti branduolių skaičiaus. Jei kalbos bibliotekos neturi nei specialiai tam sukurtos funkcijos, nei galimybės nustatyti branduolių skaičių, tada tokia kalba netinkama Relational DataRules įgyvendinimui. Taip pat reikia pažymėti, kad transformacijos lygiagretus vykdymas saugus, nes kiekviena iteracija dirba su skirtingais duomenimis, todėl papildomų sinchronizacijos taisyklių nereikia. Vis dėlto, naudojamų duomenų bazių prieiga kitiems klientams turėtų būti apribota ir užklausos, keičiančios duomenis, neleidžiamos, tam, kad būtų nuskaitomi visi egzistuojantys duomenys, transformacijų vykdymo metu jie nepasikeistų bei būtų pašalinta aklavietės (angl. „deadlock“) galimybė.

Vykdymas „fix“ režimu: jei vykdant SQL užklausas kyla klaida dėl pertraukimo (angl. „timeout“), užklausa pakartojama. Jei klaida kilo dėl kitos priežasties, tai buvo įterpimo ar atnaujinimo užklausa, bei nurodytos alternatyvios vidinės transformacijos, bandoma įterpti alternatyviomis transformacijomis gautus objektus.

13.3. Kopijavimo transformacijos vykdymas

Kai aptinkama „Copy“ transformacija, į vidinių transformacijų rinkinį įdedamas transformacijos objektas, nurodant jos vardą, t. y. „Copy“, bei reikalingus duomenis – parametrų vardus, kurie aprašyti DSL transformacijoje. Taip pat, jei tai alternatyvi transformacija, nurodomas numeris, kad būtų žinoma, kuriai vidinei transformacijai ji alternatyvi.

Pseudokodu tai atrodo taip:

```
transformation := new MapInnerTransformation();
transformation.Name := “Copy”;
transformation.Params.Add(name);
transformation.Params.Add(newName);
if (isAlternative)
then
begin
   transformation.Number := innerTransformations.Count;
   alternativeTransformations.Add(transformation);
end
else
   innerTransformations.Add(transformation);
```

Kopijavimo transformacija vykdoma atmintyje, kai vieno objekto parametro reikšmė priskiriama kito objekto reikiamo parametro reikšmei. Kopijavimo funkcijai paduodamos objektų nuorodos, pagal tas nuorodas ir parametrų vardus gaunamos reikšmės ir naujo objekto parametrui priskiriama nauja reikšmė.

Pseudokodu tai galėtų būti pavaizduota taip:

```
function Copy(paramName, newParamName, object, newObject, isAlternative)
begin
   value := object.getParameter(paramName).value;
   if (isAlternative)
      newObject.Alternative.getParameter(newParamName).value := value;
```

```
   else
      newObject.Main.getParameter(newParamName).value := value;
end
```

13.4. Rakto priskyrimo transformacijos vykdymas

Kai aptinkama „Key“ transformacija, į vidinių transformacijų rinkinį įdedamas transformacijos objektas, nurodant jos vardą, t. y. „Key“, bei reikalingus duomenis – parametro vardą, rakto tipą ir pradinį numerį, jei toks yra, kurie aprašyti DSL transformacijoje. Taip pat, jei tai alternatyvi transformacija, nurodomas numeris, kad būtų žinoma, kuriai vidinei transformacijai ji alternatyvi.

Pseudokodu tai atrodo taip:

```
transformation := new MapInnerTransformation();
transformation.Name := “Key”;
transformation.Params.Add(keyType);
transformation.Params.Add(paramName);
if (startNumber != null)
   transformation.Params.Add(startNumber);
if (isAlternative)
then
begin
   transformation.Number := innerTransformations.Count;
   alternativeTransformations.Add(transformation);
end
else
   innerTransformations.Add(transformation);
```

Rakto priskyrimo transformacija vykdoma atmintyje, kai nurodyto objekto parinktam parametrui priskiriamas raktas.

Pseudokodu tai galėtų būti pavaizduota taip:

```
function Key (index, keyType, paramName, newObject, startNumber, isAlternative)
begin
   if (keyType = “number”)
      then
      begin
```

```
    if (isAlternative)
      then newObject.Alternative.getParameter(newParamName).value := startNumber +
                                                                    index;
      else newObject.Main.getParameter(newParamName).value := startNumber +index;
    end
    else
    begin
    if (isAlternative)
      then newObject.Alternative.getParameter(newParamName).value := new UUID();
      else newObject.Main.getParameter(newParamName).value := new UUID();
    end
end
```

13.5. Numatytosios reikšmės priskyrimo transformacijos vykdymas

Kai aptinkama „Default“ transformacija, į vidinių transformacijų rinkinį įdedamas transformacijos objektas, nurodant jos vardą, t. y. „Default“, bei reikalingus duomenis – parametro vardą ir numatytąją reikšmę, kurie aprašyti DSL transformacijoje. Taip pat, jei tai alternatyvi transformacija, nurodomas numeris, kad būtų žinoma, kuriai vidinei transformacijai ji alternatyvi.

Pseudokodu tai atrodo taip:

```
transformation := new MapInnerTransformation();
transformation.Name := “Default”;
transformation.Params.Add(defaultValue);
transformation.Params.Add(paramName);
if (isAlternative)
then
begin
  transformation.Number := innerTransformations.Count;
  alternativeTransformations.Add(transformation);
end
else
  innerTransformations.Add(transformation);
```

Numatytosios reikšmės priskyrimo transformacija vykdoma atmintyje, kai nurodyto objekto parinktam parametrui priskiriamas reikšmė.

Pseudokodu tai galėtų būti pavaizduota taip:

```
function Default (paramName, defaultValue, newObject, isAlternative)
begin
   if (isAlternative)
      then newObject.Alternative.getParameter(paramName).value := defaultValue;
      else newObject.Main.getParameter(paramName).value := defaultValue;
end
```

13.6. Suliejimo transformacijos vykdymas

Kai aptinkama „Merge“ transformacija, į vidinių transformacijų rinkinį įdedamas transformacijos objektas, nurodant jos vardą, t. y. „Merge“, bei reikalingus duomenis – parametrų vardus ir skirtuką, kurie aprašyti DSL transformacijoje. Taip pat, jei tai alternatyvi transformacija, nurodomas numeris, kad būtų žinoma, kuriai vidinei transformacijai ji alternatyvi.

Pseudokodu tai atrodo taip:

```
transformation := new MapInnerTransformation();
transformation.Name := “Merge”;
transformation.Params.Add(separator);
for attributeName in attributes
   transformation.Params.Add(attributeName);
if (isAlternative)
then
begin
   transformation.Number := innerTransformations.Count;
   alternativeTransformations.Add(transformation);
end
else
   innerTransformations.Add(transformation);
```

Suliejimo transformacija vykdoma atmintyje, kai nurodyto objekto parinktam parametrui priskiriama nauja reikšmė.

Pseudokodu tai galėtų būti pavaizduota taip:

```
function Merge (separator, attributeNames, object, newOjbect, isAlternative)
begin
   mergedValue := “”;
```

```
   i := 1;
   for i to attributeNames.Length
      begin
      if (i = attributeNames.Length)
         then mergedValue += newObject.getParameter(attributeNames[i]).value;
         else mergedValue += newObject.getParameter(attributeNames[i]).value + separator;
      end
   if (isAlternative)
      then newObject.Alternative.getParameter(attributeNames[0]).value := mergedValue;
      else newObject.Main.getParameter(attributeNames[0]).value := mergedValue;
end
```

13.7. Atskyrimo transformacijos vykdymas

Kai aptinkama „Split“ transformacija, į vidinių transformacijų rinkinį įdedamas transformacijos objektas, nurodant jos vardą, t. y. „Split“, bei reikalingus duomenis – parametrų vardus ir skirtuką, kurie aprašyti DSL transformacijoje. Taip pat, jei tai alternatyvi transformacija, nurodomas numeris, kad būtų žinoma, kuriai vidinei transformacijai ji alternatyvi.

Pseudokodu tai atrodo taip:

```
transformation := new MapInnerTransformation();
transformation.Name := “Split”;
transformation.Params.Add(separator);
for attributeName in attributes
   transformation.Params.Add(attributeName);
if (isAlternative)
then
begin
   transformation.Number := innerTransformations.Count;
   alternativeTransformations.Add(transformation);
end
else
   innerTransformations.Add(transformation);
```

Atskyrimo transformacija vykdoma atmintyje, kai nurodyto objekto parinktam parametrui priskiriama nauja reikšmė.

Pseudokodu tai galėtų būti pavaizduota taip:

```
function Split (separator, attributeNames, object, newOjbect, isAlternative)
begin
   splittedValues := object.getParameter(attributeNames[0]).value.split(separator);
   i := 1;
   for i to splittedValues.length
      begin
         if (isAlternative)
            then newObject.Alternative.getParameter(attributeNames[i]).value :=
               splittedValues[i-1];
            else newObject.Main.getParameter(attributeNames[i]).value := splittedValues[i-1];
      end
end
```

13.8. Susiejimo pagal paiešką transformacijos vykdymas

Kai aptinkama „Match“ transformacija, į vidinių transformacijų rinkinį įdedamas transformacijos objektas, nurodant jos vardą, t. y. „Match“, bei reikalingus duomenis – parametrų vardus, kurie aprašyti DSL transformacijoje.

Transformacija išsaugoma analogiškai kitoms transformacijoms:

```
transformation := new MapInnerTransformation();
transformation.Name := “Match”;
transformation.Params.Add(firstParam);
transformation.Params.Add(secondParam);
innerTransformations.Add(transformation);
```

Susiejimo pagal paiešką transformacija modifikuoja „Map“ transformacijos vykdymą taip, kad naujų įrašų įterpimo metu patikrinama, ar jau įrašas pagal numatytus atributus egzistuoja ir tada, jei egzistuoja, vykdoma atnaujinimo operacija, jei ne – įterpimo.

Pseudokodu tai atrodo taip:

```
foreach object in newObjects
   runInParallel
   begin
      if (innerTransformations.Contains(match) AND
         executeSql(“SELECT (*) FROM newTable WHERE
```

```
        innerTransformations.Get("Match").Param =
        object.getParameter(innerTransformations.Get("Match").Param).value") is not null)
    then executeSql("UPDATE {newTable} SET VALUES {object} WHERE ...")
    else executeSql("INSERT {object} INTO {newTable}")
end
```

13.9. Filtravimo transformacijos vykdymas

Filtravimo transformacija, kaip jau buvo minėta „Map“ transformacijos vykdymo skiltyje, modifikuoja SQL užklausas į duomenų bazę, kur išrenkami ar agreguojami objektai.

Transformacijos filtravimo parametras išsaugomas filtrų lauke:

```
if (Filters = null)
   then Filters := filterParam;
   else Filters = Filters + "AND" + filtersParam;
```

13.10. Agregavimo transformacijos vykdymas

Kai aptinkama „Aggregate“ transformacija, į vidinių transformacijų rinkinį įdedamas transformacijos objektas, nurodant jos vardą, t. y. „Aggregate“, bei reikalingus duomenis – agreguojamo parametro vardą ir agregavimo funkciją, kurie aprašyti DSL transformacijoje.

Pseudokodu tai atrodo taip:

```
transformation := new MapInnerTransformation();
transformation.Name := "Aggregate";
transformation.Params.Add(aggregateFunction);
transformation.Params.Add(attributeName);
transformation.Params.Add(attributeNameDest);
innerTransformations.Add(transformation);
```

Priklausomai nuo to, ar ši transformacija naudojama, keičiasi ir „Map“ transformacijos vykdymas, kaip tai buvo nurodyta aprašant jos vykdymo algoritmą. Kadangi siekiama agregavimą atlikti pačioje duomenų bazėje, išrenkami jau agreguoti objektai. Taip pat, jei nurodyta „Filter“ transformacija, modifikuojama agregavimo užklausa.

Pseudokodu agregavimo transformacijų vykdymas užrašomas taip:

```
query := "SELECT {aggregateFunctions}, {aggregateAttributes} FROM {MyTable} WHERE
        {filter} GROUP BY {aggregateAttributes}";
newObjects := executeSql(query);
```

13.11. Sujungimo transformacijos vykdymas

Sujungimo transformacijos vykdymas pradedamas lentelių atributų išsaugojimu, kurie bus apjungiami, jei tokie yra. Po to, išsaugomi atributai, pagal kuriuos susiejamos lentelės. Atlikus šiuos veiksmus, pažiūrima, ar egzistuoja apjungiami atributai – jei ne, laikoma, kad naudojami visi lentelių atributai ir naujiems atributams suteikiami pavadinimai principu: *duomenų bazės pavadinimas_lentelės pavadinimas _atributo pavadinimas.*

Kai įvykdomas pasiruošimas, galima pradėti vykdyti pačią transformaciją. Ji gali būti vykdoma lygiagrečiai, panašiai kaip „Map“ transformacija. t. y. nuskaitytos dalys vykdomos lygiagrečiai. Verta pastebėti, kad dalių dydžiai turėtų būti dvigubai mažesni, nes reikės išrinkti reikšmes iš dviejų lentelių. Objektų sudarymas atliekamas su reikšmėmis iš pirmos nurodytos lentelės. Kai tai atlikta, išrenkamos reikšmės iš antros lentelės, kurios neturėjo atitikmenų pirmoje ir taip pat patalpinamos naujojoje lentelėje. Čia jau dalių dydis nebeturi būti dvigubai mažesnis, nes atitikmenų nebėra.

Pseudokodu tai galima užrašyti taip:

```
currentConstruction := "Join“;
innerJoins := getInnerJoins(); // from ANTLR
matches := getMatches(); // from ANTLR
partSize := 500;
count := executeSql(“SELECT Count(*) FROM {transformation.Table1}”)

for (i := 0; i < count; i := i + partSize)
runInParallel
begin
   objects1 := executeSql(“SELECT {innerJoins.table1Params} FROM
                              {transformation.Table1} LIMIT {partSize} OFFSET {i}”);
   newObjects := null;
   objects2 := null;
   if (matches = null)
   begin
      newObjects := joinObjects(objects1, innerJoins);
   end
   else
   begin
      objects2 := executeSql(“SELECT {innerJoins.table2Params} FROM
```

```
                                {transformation.Table1}, {transformation.Table2}
                                WHERE {match(matches, objects1)}");
        newObjects := joinObjects(objects1, objects2, innerJoins, matches);
    end
    for each object in newObjects
    runInParallel
    begin
        executeSql("INSERT {object} INTO table3");
    end
end
count := executeSql("SELECT Count(*) FROM {transformation.Table2}
                        WHERE NOT ({matches})");
partSize := partSize + partSize;
for (i := 0; i < count; i := i + partSize)
runInParallel
begin
    objects := executeSql("SELECT TOP({partSize}) {innerJoins.table2Params} FROM
                            {transformation.Table2} WHERE NOT ({matches})");
    newObjects := joinObjects(objects, innerJoins);
    for each object in newObjects
    runInParallel
    begin
        executeSql("INSERT {object} INTO {transformation.Table3}");
    end
end
currentConstruction := null;
```

Čia „getInnerJoins“ ANTLR pagalba gaunami atributų pavadinimai, kurie apjungiami, „getMatches“ ANTLR pagalba gaunami atributų vardai, kurie susiejami. „joinObjects“ vykdymas gali būti atvaizduotas funkcijomis:

```
function joinObjects(objects1, objects2, innerJoins, matches)
begin
    newObjects := [];
    for each object in objects1
    begin
```

```
      newObject := null;
      for each join in innerJoins
      begin
         matchObject := getMatchObject(object, objects2, matches);
         newObject.getParam(join.Param3) := getJoinValue(object, matchObject,
                                                join.Param1, join.Param2, join.ParamOrder);
      end
   end
end

function joinObjects(objects, innerJoins)
begin
   newObjects := [];
   for each object in objects
   begin
      newObject := null;
      for each join in innerJoins
      begin
         newObject.getParam(join.Param3) := getJoinValue(object,
                                                join.Param1, join.Param2);
      end
   end
end
```

Savo ruožtu „getMatchObject“ iš antrojo objekto sąrašo surandamas atitikmuo paduotam objektui pagal nurodytus parametrus iš „matches“ objekto. „getJoinValue“ grąžina reikšmę, pagal tokią logiką: priklausomai nuo to, kurios lentelės stulpelis pirmiau užrašytas Relation DataRules konstrukcijoje (tai išsaugota „ParamOrder“ parametre), paimama reikšmė iš to objekto parametro, arba, jei reikšmė „null“, iš kito.

13.12. Validavimo vykdymas

Validavimo vykdymas susideda iš vidinių validavimo taisyklių patikrinimo. Tų taisyklių vykdymas iš esmės susideda iš duomenų skaitymo – modifikuoti nieko nereikia. Dėl šios priežasties, beveik visas procesas gali būti vykdomas duomenų bazėje išrenkant duomenis, kur pastebimos anomalijos ir tos anomalijos surašomos į numatytą failą. Be to, kadangi nėra

modifikavimo, aprašytas taisykles galima vykdyti lygiagrečiai be jokių apribojimų, kas paspartintų atlikimą.

Kai pradedama vykdyti validavimo funkcija, panašiai kaip ir „Map“ transformacijos metu, vidinės transformacijos, interpretatoriui jas identifikavus, patalpinamos į sąrašą, kuris gali būti vykdomas lygiagrečiai – patikrinama ar yra anomalijų ir jei yra, rastos anomalijos įrašomos į failą.

Pseudokodu tai atrodo taip:

```
currentConstruction := "Validate“
innerRules := getInnerRules();
foreach rule in innerRules
   runInParallel
   begin
      rule.Execute();
   end
currentConstruction := null;
```

13.13. Unikalumo patikrinimo vykdymas

Unikalumo patikrinimo transformacija išsaugoma taip pat kaip ir kitos vidinės transformacijos. Sukuriamas objektas, kuriam priskiriamas vardas ir parametrai – atributų vardai, kurių reikšmės turėtų būti unikalios.

Pseudokodu tai užrašoma taip:

```
transformation := new InnerTransformation();
transformation.Name := “Unique”;
transformation.Params.Add(attributeName1);
...
transformation.Params.Add(attributeNameN);
rules.Add(transformation);
```

Vykdant validavimo vidines transformacijas, unikalumo patikrinimas vykdomas išrenkant eilutes, kurios sulaužė šią taisyklę. Jos užfiksuojamos įrašant duomenis apie tai pagalbiniame faile.

Algoritmas pseudokodu:

```
function Unique(table, attributes)
begin
   duplicates := executeSql(“SELECT * FROM {table} GROUP BY {attributes} HAVING
                              COUNT(*) > 1”);
```

```
    validationLogger.Log("Duplicates in {table}: \r\n");
    for duplicate in duplicates
        validationLogger.Log({duplicate});
end
```

13.14. Reikšmės egzistavimo patikrinimas

Reikšmės egzistavimo patikrinimo transformacija išsaugoma taip pat kaip ir kitos vidinės transformacijos. Sukuriamas objektas, kuriam priskiriamas vardas ir parametras – atributo vardas, kurio reikšmės turėtų egzistuoti.

Pseudokodu tai užrašoma taip:

```
transformation := new InnerTransformation();
transformation.Name := "NotNull";
transformation.Params.Add(attributeName);
rules.Add(transformation);
```

Vykdant validavimo vidines transformacijas, reikšmės egzistavimo patikrinimas vykdomas išrenkant eilutes, kurios sulaužė šią taisyklę. Jos užfiksuojamos įrašant duomenis apie tai pagalbiniame faile.

Algoritmas pseudokodu:

```
function NotNull(table, attribute)
begin
    nullRows := executeSql("SELECT * FROM {table} WHERE {attribute} IS NULL");
    validationLogger.Log("Null values in {table} found in rows: \r\n");
    for row in nullRows
        validationLogger.Log({row});
end
```

13.15. Nuorodos patikrinimo vykdymas

Nuorodos patikrinimo transformacija išsaugoma taip pat kaip ir kitos vidinės transformacijos. Sukuriamas objektas, kuriam priskiriamas vardas ir parametrai – atributo vardas, pagal kurio reikšmes turėtų būti galima rasti eilutes kitoje lentelėje, ir kelio iki kito lentelės pirminio rakto parametras.

Pseudokodu tai užrašoma taip:

```
transformation := new InnerTransformation();
transformation.Name := “Reference”;
transformation.Params.Add(attributeName);
transformation.Params.Add(otherTableKeyPath);
rules.Add(transformation);
```

Vykdant validavimo vidines transformacijas, nuorodos patikrinimas vykdomas išrenkant eilutes, kurios sulaužė šią taisyklę. Jos užfiksuojamos įrašant duomenis apie tai pagalbiniame faile.

Algoritmas pseudokodu:

```
function Reference(table1, table2, attribute1, attribute2)
begin
   referenceNotFound := executeSql(“SELECT * FROM {table1} WHERE (SELECT
      COUNT(*) FROM {table2} WHERE {attribute1} = {attribute2}) = 0”);
   validationLogger.Log(“Null values in {table} found in rows: \r\n”);
   for row in nullRows
      validationLogger.Log({row});
end
```

13.16. Trynimo vykdymas

Trynimas transformacija vienu metu apima vienos lentelės duomenis bei analogiški veiksmai gali būti užrašomi SQL kalba. Dėl šios priežasties, vykdymas atliekamas pačioje duomenų bazėje sukuriant tam skirtas užklausas. Jei trinama lentelė, sukuriama „DROP“ užklausa. Jei trinamos eilutės ar atributai – „DELETE“ užklausa su reikalingomis „WHERE“ sąlygomis priklausomai nuo to, ar naudojama filtravimo vidinė transformacija.

Pseudokodu tai užrašoma taip:

```
currentConstruction := "Delete“;
type := getDeleteType();
if (type = “table”)
   then
      executeSql(“DROP {getName()}”);
   else if (type = “rows”)
   then
     executeSql(“DELETE FROM {getName()} {getFilters()}”);
```

```
    else
        executeSql("UPDATE {getName()} SET {getAttributes() = null} {getFilters()}");
currentConstruction := null;
```

13.17. Prisijungimas prie duomenų bazės

Vykdant operaciją „database“ išsaugomi duomenys, kurie reikalingi prisijungiant prie duomenų bazių. Tai atliekama išsaugojant prisijungimo prie duomenų bazės nuorodą lauke (angl. „field“), kuris pasiekiamas vykdant transformacijas. Šiame lauke patalpinamos visos reikalingos prisijungimų nuorodos nurodant duomenų bazės vardą, kuris turi būti unikalus ir bus naudojamas programoje, bei pačią prisijungimo nuorodą.

Pseudokodu tai galėtų būti užrašyta taip:

```
if (ConnectionStrings.Has(dbName))
    then ConnectionStrings[dbName] := connectionString;
    else ConnectionStrings.Add(dbName, connectionString);
```

Prieš kiekvieną transformaciją, pasitelkiant šiame lauke išsaugotas reikšmes, atliekamas prisijungimas prie duomenų bazės.

Pseudokodu tai užrašoma taip:

```
connectionString := ConnectionStrings[dbName];
connection := new SqlConnection(connectionString);
connection.Open();
```

13.18. Įvykių registravimo failų nurodymas

Vykdant operaciją „log“ išsaugomos nuorodos į failus, kuriuose įrašoma informacija apie klaidas ir anomalijas. Kaip ir prisijungimo nuorodų prie duomenų bazių atveju, nuorodos į failus išsaugomos tam skirtuose laukuose tik šiuo atveju jų gali būti tik po vieną.

Pseudokodu tai užrašoma taip:

```
ValidationLog.FilePath := validationFilePath;
ErrorLog.FilePath := errorsFilePath;
```

Kviečiant šiuos objektus išsaugoma reikalinga informacija apie įvykius:

```
ValidationLog(message);
ErrorLog(message);
```

13.19. Kokybės režimo nurodymas

Vykdant operaciją „quality“ nustatoma, kokiu kokybės režimu veiks programa. Ši informacija patalpinama tam skirtame lauke, kuris pasiekiamas vykdant transformacijas.

Pseudokodu tai užrašoma taip:

```
qualityMode := qualityModeParam;
```

13.20. Funkcijų paketo įkėlimas

Kai programos vykdymo pradžioje sutinkama operacija „external“, informacija apie tai, kur saugoma išorinė funkcija biblioteka panaudojama užkraunant biblioteką ir ją priskiriant tam skirtam laukui.

Pseudokodu tai užrašoma taip:

```
currentConstruction := "external“
externalLibrary := loadLibrary(libraryFilePath);
currentConstruction := null;
```

Čia „loadLibrary“ funkcija konkrečioje kalboje turėtų būti vykdoma, kaip išorinių bibliotekų užkrovimo funkcija. Pvz.: C# ateju tam galėtų būti panaudota „Assembly.LoadFile(filepath)“ funkcija.

13.21. Išorinių funkcijų vykdymas

Tam, kad geriau suprasti, kaip turėtų būti vykdomos išorinės funkcijos, kurios užkraunamos nurodant bibliotekos failą, pateikiamas pavyzdys, kaip turėtų būti vykdoma datos formato keitimo funkcija.

Tarkime, kad šią funkciją Relational DataRules kalboje naudosime taip:

```
Map MyDB.AmericanDates to MyDB.LithuanianDates
   external FormatDate
      US to LT
      Date to Data
   end
end
```

Kai vykdoma susiejimo transformacija, pastebima, kad naudojama išorinė funkcija. Ši funkcija panašiai kaip kitos vidinės transformacijos išsaugoma, tik šiuo atveju pridedant parametrų sąrašą, kadangi jis dinamiškas:

```
transformation := new InnerTransformation();
transformation.Name := “external”;
transformation.Params.Add(params);
innerTransformations.Add(transformation);
```

Tada ši vidinė transformacija vykdoma taip – įkeltoje bibliotekoje surandama reikalinga funkcija, pagal nurodytą pavadinimą, tada ta funkcija iškviečiama paduodant parametrus, o gautas rezultatas grąžinamas.

Pseudokodu tai atrodo taip:

```
function External(params, object, newObject)
begin
   functionName := params[0];
   externalFunction := externalLibrary.getFunction(functionName);
   result := externalFunction.Invoke(params, object, newObject);
   return result;
end
```

Savo ruožtu datos keitimo funkcija bibliotekoje gali būti įgyvendinta kaip tik norima, tik svarbu paisyti trijų taisyklių – funkcija turi būti pavadinta taip pat, kaip kviečiama Relational DataRules kalboje, priimtų parametrus „params“, „object“ ir „newObject“ ir atliktų reikiamą veiksmą (šiuo atveju priskirtų naujam objektui reikiamą datą). Pvz.:

```
class MyLibrary
begin
   function FormatDate(params, object, newObject)
   begin
      if (params[1] = “US” AND params[3] “LT”)
      begin
         oldDateArray := object.getParameter(params[4]).value.Split(“/”);
         newObject.getParameter(params[6]).value := oldDateArray[2] + “-” +
            oldDateArray[0] + “-“ + oldDateArray[1];
      end
   end
end
```

13.22. Klaidų valdymas

Vykdant Relational DataRules parašytas programas, gali kilti klaidų, nebūtinai susijusių su pačių transformacijų vykdymu, t. y. galimos ir gramatinės ar sintaksinės klaidos. Dėl šios priežasties, kiekviena kalbos konstrukcija apgaubiama „try/catch“ bloku, o įvykus nenumatytai situacijai (angl. „exception“), į klaidų registravimo failą įrašoma, kokią transformaciją vykdant atsirado klaida (jei tai įmanoma padaryti, kadangi neatpažinto teksto atveju, programa nežinotų to) ir kokioje eilutėje. Ši informacija saugoma vykdant kiekvieną konstrukciją, t. y. prieš skaitant ją (ANTLR analizatoriumi perskaičius „token“), išsaugomas eilutės numeris, o konstrukcijos skaitymo metu išsaugomas jos pavadinimas. Jeigu klaida gramatikoje ar sintaksėje, ANTLR pagalba ji sugaunama ir išsaugoma faile. Tai atliekama pasitelkiant klaidų „klausytoją“ (angl. „listener“), kuris iškviečiamas aptikus klaidą.

Pseudokodu tai užrašoma taip:

```
currentLine := token.getLine();
try
begin
   // start executing transformation or operation
   currentConstruction := “construction name” // construction name is replaced by real
                                                // transformation/operation name
   // continue executing
end
catch (exception)
begin
   errorLog.Log(“error in line: ” + currentLine + “, “ + currentConstruction);
end
```

13.23. Vykdymo pastabos ir apribojimai

Įgyvendinant vykdymo modelį bendros paskirties kalba ir vykdant transformacijas, reikėtų atkreipti dėmesį į šiuos aspektus:

- Vykdant susiejimo („Map“) ir sujungimo („Join“) transformacijas, kai lygiagrečiai dalimis išrenkamos eilutės iš šaltinio duomenų bazės, priklausomai nuo duomenų bazės tipo turėtų būti pakoreguota SQL užklausa, kadangi SQL neturi galimybės išrinkti įrašus nuo tam tikros eilutės, jei nežinomas raktas;

- Jei duomenų bazė, kuriai atliekamos transformacijos, nepalaiko lygiagretaus vykdymo (pvz. SQLite lygiagrečiai galima tik nuskaityti duomenis), transformacijas reikėtų vykdyti nuosekliai.
- Pradedant vykdyti transformacijas, reikia apriboti duomenų bazių naudojimą kitiems klientams, kad jis nekliudytų transformacijų vykdymui. Modifikavimo užklausų uždraudimas turėtų būti pakankamas sprendimas.

14. RELATIONAL DATARULES KALBĄ VYKDANTI PROGRAMA

Vienas darbo uždavinių buvo sukurti programą (prototipą), gebantį vykdyti Relational DataRules kalba parašytas duomenų transformacijas. Šio uždavinio tikslas pavaizduoti, kad kalbą galima įgyvendinti pasinaudojant bendros paskirties kalba ir tą DSL panaudoti nagrinėjamos dalykinės srities uždaviniams spręsti vykdant transformacijas, o aprašytas vykdymo modelis teisingas ir jį galima įgyvendinti naudojant bendros paskirties kalbą.

Prototipas parašytas C# kalba. Visi darbe aprašyti algoritmai suprogramuoti ir adaptuoti kalbai, kur galima pasinaudojant .NET karkaso funkcionalumu (pvz. LINQ naudojimu, sisteminių .NET funkcijų naudojimu ir t. t.).

Programos struktūra:

- Kalbos gramatika ANTLR v4.7 formatu;
- Sugeneruotos ANTLR klasės (klausytojo, analizatoriaus, lekserio klasės ir ženklų (angl. „tokens“ failas)), pagal sukurtą kalbos gramatiką pasitelkiant ANTLR Language Support įrankį [H16];
- Vykdymui pritaikytas Relational DataRules „klausytojas“, kuris pagal atpažintas konstrukcijas vykdo reikiamus algoritmus;
- Relational DataRules klaidų „klausytojas“, kuris atpažįsta sintaksės klaidas ir informaciją apie tai išsaugo tam skirtame klaidų faile;
- Atskiros klasės kiekvienai transformacijai ir operacijai, kuriose aprašytos funkcijos, skirtos transformacijų vykdymui (pvz.: vidinių transformacijų funkcijos, funkcijų susiejimas pagal atpažintą konstrukciją ir kt.) bei reikalingi laukai (pvz. vidinių transformacijų sąrašas, vykdomos konstrukcijos pavadinimas ir kt.);
- Pradinė vykdymo klasė (Program.cs), kuri nuskaito Relational DataRules programinį kodą ir pradeda konstrukcijų atpažinimą ir vykdymą.

Prototipo apribojimai:

- Neišbaigtas klaidų valdymas ir įvesties validavimas – prototipas turėtų veikti gerai, kai Relational DataRules transformacijos aprašytos korektiškai ir nėra išorinių trikdžių vykdant SQL užklausas;
- Neįgyvendinta sujungimo („Join“) transformacija bei išorinių funkcijų („external“) naudojimas;
- Transformacijas galima atlikti tik Microsoft SQL Server duomenų bazėse saugomiems duomenims.

Prototipo veikimas: programai paduodamas parametras – Relational DataRules programinio kodo failas, kuriame aprašytos transformacijos; transformacija ar operacija viena po kitos atpažįstama ir vykdomos atitinkamos funkcijos.

Programos kūrimas padėjo detaliau patikrinti vykdymo modelį, ištaisyti rastus netikslumus (angl. „bugs“), optimizuoti kai kurių veiksmų atlikimą (pvz. filtravimo vykdymą, kurio loginės operacijos gali būti išsaugomos tam skirtame lauke, o ne vidinių transformacijų sąraše) ir pavaizduoti, kaip transformacijų algoritmai gali būti įgyvendinti ir susiejami su aprašytomis konstrukcijomis, pasinaudojant ANTLR įrankiu, naudojant bendros paskirties kalbą – C#. Taip pat, kadangi parašyta programa geba įvykdyti paduotas Relational DataRules transformacijas, tai rodo, kad nusakytus vykdymo modelio algoritmus galima įgyvendinti bendros paskirties kalba, juos vykdyti ir jie yra teisingi.

Suprogramavus likusį funkcionalumą (t. y. pašalinus dabartinius apribojimus) ir kruopščiai ištestavus, programą būtų galima pilnai naudoti atliekant duomenų transformacijų uždavinius, pasinaudojant kalbos transformacijomis ir, jei reikia, įkeltomis išorinėmis funkcijomis.

Programinį Relational DataRules transformacijas vykdančios programos išeities kodą galima rasti antrame priede.

REZULTATAI IR IŠVADOS

Pagrindiniai darbo metu gauti rezultatai:

1. Išanalizuota duomenų transformacijų (ETL) dalykinė sritis ir pagal tai sudarytas reikalavimų rinkinys dalykinės srities kalbai.

2. Sukonstruota Relational DataRules kalba – nusakyta jos gramatika ir sintaksė, apibrėžtos naudojimo taisyklės, pateikti pavyzdžiai. Gramatika taip pat aprašyta ANTLR v4.7 formatu.

3. Sudarytas dalykinės srities kalbos vykdymo modelis – pseudokodu ir tekstu aprašyti algoritmai, kurie turi būti atliekami vykdant DSL konstrukcijas.

4. Gautas Relational DataRules gramatikos palyginimas su kitų kalbų gramatikomis.

5. Suprogramuotas prototipas, gebantis vykdyti Relational DataRules kalba aprašytas transformacijas pagal sudarytą DSL vykdymo modelį.

Esminės darbo išvados:

1. Pasinaudojant apžvelgta literatūra apie duomenų transformavimo proceso atlikimą, galima sudaryti reikalavimus šios srities kalbai, o pagal sudarytus reikalavimus apibrėžti DSL konstrukcijas, kuriomis aprašomos transformacijos.

2. Duomenų transformacijos gali būti aprašytos Relational DataRules kalba, o dalykinės srities kalbos vykdymo modelio algoritmai gali būti įgyvendinti bendros paskirties kalba. Tuo įsitikinta, sukūrus Relational DataRules konstrukcijas vykdantį prototipą, kuris pagal sudaryto DSL vykdymo modelio algoritmus geba vykdyti transformacijas.

3. DSL vykdymo modelyje aprašyti algoritmai nepriklauso nuo Relational DataRules konstrukcijų, todėl pakeitus DSL konstrukcijų atpažinimą, transformacijas aprašyti ir vykdyti galima bet kokia kita dalykinės srities kalba, kuri tenkina sudarytą reikalavimų rinkinį.

Su vykdymo modelio kūrimu susijusi išvada:

4. Nagrinėjant, kaip galima įgyvendinti dalykinės srities kalbos interpretatorių nustatyta, kad jo kūrimas gali būti palengvintas pasinaudojant jau egzistuojančiais analizatoriais, kurie padeda atpažinti DSL konstrukcijas ir jas susieti su bendros paskirties kalbos konstrukcijomis. Šiame darbe tam tikslui buvo panaudotas ANTLR įrankis.

ŠALTINIAI

[A18] ANTLR. About The ANTLR Parser Generator. [Žiūrėta 2018-04-15]. Prieiga per internetą: <http://www.antlr.org/about.html>.

[B17] Bubbles puslapis. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://bubbles.databrewery.org/index.html>.

[BR17] Bubbles saugyklos puslapis. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://github.com/stiivi/bubbles>.

[C17] Cinchoo ETL dokumentacija. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://www.cinchoo.com/category/etl>.

[CČF+09] J. Cervelle, M. Črepinšek, R. Forax, T. Kosar, M. Mernik, G. Roussel. On Defining Quality Based Grammar Metrics. [Žiūrėta 2018-04-15]. Prieiga per internetą: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5352768>.

[DBE18] DB-Engines. DB-Engines Ranking. 2018. [Žiūrėta 2018-04-15]. Prieiga per internetą: <http://db-engines.com/en/ranking>.

[Dor16] I. Dorbian. Trifacta scores $35 mln. 2016. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://www.pehub.com/2016/02/trifacta-scores-35-mln/#>.

[DI17] Data Integration Information. ETL (Extract-Transform-Load). [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://www.dataintegration.info/etl>.

[DS15] I. Damyanov, M. Sukalinska. Domain Specific Languages in Practice. 2015. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://research.ijcaonline.org/volume115/number2/pxc3902205.pdf>.

[E17] Eclipse Community Forums. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://www.eclipse.org/forums/>.

[ED17] ETL Database. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://etldatabase.com>.

[ET17] ETL Tools. Data transformation. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://www.etltools.org/data-transformation.html>.

[F17] FluentETL - Data automation made easy for coders. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://fluentetl.codeplex.com>.

[G17] GETL saugyklos puslapis. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://github.com/ascrus/getl>.

[GFS+01] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C. A. Saita. Declarative Data Cleaning: Language, Model, and Algorithms. Proceedings of the 27th VLDB Conference. Roma. 2001. [Žiūrėta 2017-04-01]. Prieiga per internetą: <http://www.vldb.org/conf/2001/P371.pdf>.

[H16] S. Harwell. ANTLR Language Support. [Žiūrėta 2018-05-08]. Prieiga per internetą: <https://marketplace.visualstudio.com/items?itemName=SamHarwell.ANTLRLanguageSupport>.

[Her07] J. Heering. Domain-Specific Languages in Perspective. 2007. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://swerl.tudelft.nl/twiki/pub/MoDSE/Colloquium/MoDSE-JanHeering.pdf>.

[HK14] J. Hellerstein, S. Kandel, Trifacta. Advantages of a Domain Specific Language Approach to Data Transformation. 2014. [Žiūrėta 2017-05-02]. Prieiga per internetą: <https://www.youtube.com/watch?v=eXCiBLtet-4>.

[HQC14] E. Huang, A. Quiroz, L. Ceriani. Automating Data Integration with HiperFuse. 2014. [Žiūrėta 2016-12-08]. Prieiga per internetą: <http://ieeexplore.ieee.org/document/7004316/?part=1

[HR17] Rhino ETL puslapis. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://hibernatingrhinos.com/oss/rhino-etl>.

[Hud97] P. Hudak. Domain Specific Languages. 1997. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://haskell.cs.yale.edu/wp-content/uploads/2011/01/DSEL-Little.pdf>.

[I18] Informatica. About. 2018. [Žiūrėta 2018-05-07]. Prieiga per internetą: <https://www.informatica.com/about-us.html>.

[KC04] R. Kimball, J. Caserta. The Data Warehouse ETL Toolkit, Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. Wiley Publishing, Inc., Indianapolis. 2004.

[KZ18] Enciklopedinis kompiuterijos žodynas. [Žiūrėta 2018-05-24]. Prieiga per internetą: <http://ims.mii.lt/ALK%C5%BD>.

[LMS05] P. Leach, M. Mealling, R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. Request for Comments: 4122. 2005. [Žiūrėta 2018-01-06]. Prieiga per internetą: <https://www.ietf.org/rfc/rfc4122.txt>.

[LZ18] Terminų žodynas. [Žiūrėta 2018-04-15]. Prieiga per internetą: <http://www.lietuviuzodynas.lt/terminai>.

[MHS05] M. Mernik, J. Heering, A. M. Sloane. When and how to develop domain-specific languages. 2005. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://dl.acm.org/citation.cfm?id=1118892>.

[N18] NuGet gallery. Rhino-Etl. Žiūrėta [2018-04-15]. Prieiga per internetą: <https://www.nuget.org/packages/Rhino-Etl>.

[O17] Oracle. Overview of Transforming Data. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://docs.oracle.com/cd/E11882_01/owb.112/e10935/transformdata_intro.htm#WBETL04000>.

[ODO17] Odo dokumentacija. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://odo.readthedocs.io/en/latest/index.html>.

[P17] Petl dokumentacija. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://petl.readthedocs.io/en/latest/>.

[Pan17] Pandas puslapis. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://pandas.pydata.org/>.

[Par17] Parade saugyklos puslapis. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://github.com/bailaohe/parade/wiki/Tutorials>.

[PT17] PyTables puslapis. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://www.pytables.org>.

[R17] ReactiveETL puslapis. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://reactiveetl.codeplex.com/>.

[RD00] E. Rahm, H. H. Do. Data Cleaning: Problems and current approaches. IEEE Bulletin of the Technical Commitee on Data Engineering, 2000, 24, 4. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://www.academia.edu/download/41858217/A00DEC-CD.pdf#page=5>.

[RH01] V. Raman, J. M. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. 2001. [Žiūrėta 2017-05-02]. Prieiga per internetą: <http://control.cs.berkeley.edu/pwheel-vldb.pdf>.

[S18] Spoofax dokumentacija. [Žiūrėta 2018-04-15]. Prieiga per internetą: <http://www.metaborg.org/en/latest>.

[Ser12] J. Serra. Difference between ETL and ELT. 2012. [Žiūrėta 2017-06-17]. Prieiga per internetą: <http://www.jamesserra.com/archive/2012/01/difference-between-etl-and-elt/>.

[Spe15] G. Speare. ETL vs. ELT – What's the Big Difference? 2015. [Žiūrėta 2017-06-17]. Prieiga per internetą: <https://www.ironsidegroup.com/2015/03/01/etl-vs-elt-whats-the-big-difference/>.

[T17] Technopedia. Data Transformation. [Žiūrėta 2017-04-22]. Prieiga per internetą: <https://www.techopedia.com/definition/6760/data-transformation>.

[Tom18] F. Tomassetti. The complete guide to (external) Domain Specific Languages. [Žiūrėta 2018-04-15]. Prieiga per internetą: <https://tomassetti.me/domain-specific-languages/>.

[Tub15] A. Tubelevičiūtė. Automatizuoto duomenų validavimo ir transformavimo taisyklių apibrėžimas dalykinės srities kalba. Vilnius, 2015.

[Wil01] D. S. Wile. Supporting the DSL Spectrum. Journal Of Computing and Information Technology, 9(4), 2001, p. 263-287. [Žiūrėta 2017-06-11]. Prieiga per internetą: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.469.8992&rep=rep1&type=pdf>.

[WM07] M. Weis, I. Manolescu. Declarative XML Data Cleaning with XClean. 2007. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://link.springer.com/chapter/10.1007/978-3-540-72988-4_8>.

[X17] Xtext puslapis. [Žiūrėta 2017-06-11]. Prieiga per internetą: <https://eclipse.org/Xtext/>.

```
grammar RelationalDataRules;
program : externalFunctions? database+ log? quality? (rule | log)+;
rule : join | delete | validate | map;
transform1 : defaultT | key | copy | filter | aggregate | matchT | merge | split | external;
transform2 : unique | reference | notNull | external;
externalFunctions : 'external' 'functions' 'set' 'path' Value;
database : 'database' String 'set' 'connection' Value;
log : 'log' LogType 'set' 'path' Value;
quality : 'quality' 'set' QualityMode;
join : 'Join' String 'with' String 'to' String (String 'with' String 'to' String)+ 'Match' ('Merge')? (String 'with'
String)+ 'end';
map : 'Map' String 'to' String (transform1)+ 'end';
defaultT : 'Default' Value 'to' String ('alternative' transform1)?;
key : 'Key' KeyType (Number)? 'to' String ('alternative' transform1)?;
copy : 'Copy' String 'to' String ('alternative' transform1)?;
filter : 'Filter' Value;
aggregate : 'Aggregate' AggregateFunction String 'to' String;
matchT : 'Match' String 'with' String;
merge : 'Merge' Value String (',' String)* 'to' String ('alternative' transform1)?;
split : 'Split' Value String 'to' String (',' String)* 'end' ('alternative' transform1)?;
external : 'External' String (',' String)* 'end' ('alternative' transform1)?;
validate : 'Validate' String transform2+ 'end';
delete : 'Delete' DeleteType String (filter)*;
unique : 'Unique' String;
reference : 'Reference' String 'in' String;
notNull : 'NotNull' String;

LogType : 'error' | 'validation';
QualityMode : 'skip' | 'fix' | 'stop';
KeyType : 'number' | 'uuid';
AggregateFunction: 'Sum' | 'Count' | 'Average' | 'Min' | 'Max' | 'on';
DeleteType : 'table' | 'rows' | 'attribute';
Number : Digit+;
String : (Symbol|Digit)+;
Value : ""String"";
Symbol : 'A'..'Z'|'a'..'z'|'_'|'-'|','|'.'|'`'|'*'|'('|')'|'{'| '}'|'['|']'|':'|';'|'|'|'?'|'!'|'@'|'#'|'$'|'%'|'^'|'&'|'+'|'='|'<'|'>'|'~'|'/';
Digit : '0'..'9';
Whitespace : (' ' | '\t' | '\r'| '\n') -> skip;
```