KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Anna TRUNCAITĖ
Sigitas PAULAVIČIUS

IŠSAMIOS LOGINĖS SCHEMOS ATSTATYMAS IŠ LIKTINIŲ INFORMACIJOS SISTEMŲ

Tiriamasis magistro darbas

Darbo vadovas: Prof. dr. Bronius PARADAUSKAS

Kaunas, 2009

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Anna TRUNCAITĖ
Sigitas PAULAVIČIUS

IŠSAMIOS LOGINĖS SCHEMOS ATSTATYMAS IŠ LIKTINIŲ INFORMACIJOS SISTEMŲ

Tiriamasis magistro darbas

Recenzentas
Prof. dr. Vacius JUSAS
2009-01-13

Vadovas
Prof. dr. Bronius PARADAUSKAS

Konsultavo
Mag. Aurimas LAURIKAITIS
2009-01-13

Atliko
IFM-3/4 grupės studentai:
Anna TRUNCAITĖ
Sigitas PAULAVIČIUS
2009-01-13

Kaunas, 2009

TURINYS

SUMMARY **5**

ĮVADAS **6**

1. ATVIRKŠTINĖS DUOMENŲ INŽINERIJOS ANALIZĖ **10**
1.1. Atvirkštinė ir tiesioginė duomenų inžinerijos 10
1.2. ADI aktualumas 13
1.3. ADI aktualumo praktikoje tyrimas 14
1.3.1. Tyrimo formulavimas 14
1.3.2. Tyrimo eiga 14
1.3.3. Tyrimo išvados 15
1.4. ADI problematika 17
1.5. Atvirkštinės duomenų inžinerijos kokybės kriterijai 18
1.6. ADI metodologija 19
1.7. Duomenų struktūros išgavimas 21
1.7.1. DDL teksto analizė 23
1.7.2. Fizinė integracija 24
1.7.3. Schemos išgryninimas 24
1.7.4. Schemos valymas 25
1.7.5. Neišreikštosios struktūros ir apribojimai 26
1.7.6. Duomenų struktūros išgavimo informacijos šaltiniai 28
1.7.7. Duomenų struktūros išgavimo metodikos 30
1.8. Programų supratimas ADI 33
1.8.1. Programų supratimas 33
1.8.2. Programų supratimas atvirkštinės duomenų inžinerijos kontekste 36
1.8.3. Programų supratimo sunkumai 38
1.8.4. ADI programų supratimo metodikos 39
1.9. Duomenų schemos specifikacija 40
1.9.1. Konceptuali specifikacija 40
1.9.2. Loginė specifikacija 42
1.9.3. Fizinė specifikacija 42
1.9.4. Reliacinis modelis 43
1.10. Esamų ADI metodų analizė 45
1.10.1. Metodų aprašymai 47
1.10.2. Metodų palyginimas 53
1.11. Analizės išvados 57

2. SUDARYTOS ADI METODIKOS APIBRĖŽIMAS IR ESMĖ **59**
2.1. ADI metodikos aplinka, sąlygos ir prielaidos 59
2.2. ADI metodikos vartotojų analizė 62
2.3. ADI metodika ir jos etapai 62
2.3.1. Duomenų žodynų išgavimas 64
2.3.2. Abstrakčios sintaksės medžio sudarymas 65
2.3.3. Neišreikštųjų apribojimų atributams nustatymas 66
2.3.4. Neišreikštųjų identifikatorių nustatymas 66
2.3.5. Apjungtų esybių nustatymas 67
2.3.6. Neišreikštųjų ryšio ir lygybės apribojimų nustatymas 69

2.3.7. Kitų neišreikštųjų apribojimų ir priklausomybių nustatymas ... 70
2.4. Metodologijos rezultatų duomenų modelis ... 71
3. ADI ĮRANKIO PROJEKTAS ... 72
3.1. ADI eksperimentinio įrankio projektas ... 72
3.1.1. Duomenų žodyno išgavimas ... 72
3.1.2. Abstrakčios sintaksės medžio sudarymas ... 73
3.1.3. Neišreikštųjų apribojimų atributams nustatymas ... 74
3.1.4. Neišreikštųjų identifikatorių nustatymas ... 77
3.1.5. Apjungtų esybių nustatymas ... 78
3.1.6. Neišreikštųjų ryšio ir lygybės apribojimų nustatymas ... 80
3.1.7. Kitų neišreiktųjų apribojimų ir priklausomybių nustatymas ... 84
3.2. Kuriamo įrankio architektūros pasirinkimas ... 86
3.3. Reikalavimų analizė ... 87
4. REALIZACIJA IR EKSPERIMENTINIS METODIKOS TYRIMAS ... 88
4.1. Realizacijos aprašymas ... 88
4.2. Kontrolinis pavyzdys ... 89
4.2.1. Duomenų žodyno išgavimas ... 90
4.2.2. Abstrakčios sintaksės medžio sudarymas ... 91
4.2.3. Neišreikštųjų apribojimų atributams nustatymas ... 92
4.2.4. Neišreikštųjų identifikatorių nustatymas ... 94
4.2.5. Apjungtų esybių nustatymas ... 95
4.2.6. Neišreikštųjų ryšio ir lygybės apribojimų nustatymas ... 96
4.2.7. Kitų neišreikštųjų apribojimų ir priklausomybių nustatymas ... 100
4.3. Sudarytos ADI metodikos įvertinimas ... 101
4.3.1. ADI metodikos įvertinimas pagal išgaunamas struktūras ir apribojimus ... 101
4.3.2. ADI metodikos kokybinis įvertinimas apklausiant ekspertus ... 105
5. IŠVADOS ... 107
6. LITERATŪRA ... 108
7. TERMINŲ IR SUTRUMPINIMŲ ŽODYNAS ... 110
8. PRIEDAI ... 111
A PRIEDAS. C KODO GRAMATIKA ASM SUDARYTI ... 111
B PRIEDAS. APRIBOJIMŲ IR PRIKLAUSOMYBIŲ APTIKIMO UŽKLAUSOS ... 116
C PRIEDAS. STRAIPSNIS ... 120

RECOVERY OF COMPLETE LOGICAL SCHEMA FROM LEGACY INFORMATION SYSTEMS

SUMMARY

In order for maintenance or migration of a legacy system to be efficient and safe, complete documentation is required, which might be non-existent, lost or out-dated. Recovery of this information from legacy system itself is possible through the proceses of reverse engineering. This work is concerned with reverse data engineering to recover of complete logical schema by first extracting explicitly declared database structure and constraints and then completing it by extarcting implicit constructs and constraints from source code and data analysis. The methology described in this work is sought to be simple to use and oriented mainly towards data migration efforts, which was found to be the field where reverse data engineering is used the most. The methodology is concerned with extraction of complete logical schema, although it is not always required for data migration proccess in its full extent.

An issue of discovery and separation of joined entities is adressed in this work. Joined entity can be defined as multiple conceptual entities that were joined into single database table due to certain external limitations on number of entities. This situation is not rare in legacy systems especially and is not found to be adressed by any other method of database reverse engineering.

ĮVADAS

Programinės įrangos inžinerija daugelį metų intensyviai nagrinėjo naujų sistemų vystymo klausimus. Pagrindinės mokslo ir verslo pastangos skirtos naujoms efektyvesnėms programų sistemų vystymo metodologijoms kurti ir procesams, kurie pagerina programų kokybę, sumažina kūrimo trukmę ir padeda vystyti vartotojų poreikius atitinkančias programas, tobulinti. Susitelkus į šiuos aspektus, programinės įrangos palaikymas, kuris yra viena svarbiausių (laiko ir kaštų aspektu) programinės įrangos ciklo veiklų, nustumtas į antrą planą. Palaikymas sudaro apie 70% sistemos gyvavimo ciklo kaštų [32]. Visos didelės programos modifikuojamos, pakeitimai įkeliami į sistemą, todėl jos struktūra ima blogėti. Pasikeičia nariai iš pradinių ir palaikančių komandų, dokumentacija (jei egzistuoja) palaipsniui pasensta. Tokios sistemos vadinamos liktinėmis sistemomis, jos apima ypač svarbias veiklos žinias ir kritinius duomenis.

Liktinė sistema – tai bet kokia informacinė sistema (IS), kuri pastebimai priešinasi modifikavimui ar evoliucijai, skirtų atitikti naujus, nuolat kintančius verslo reikalavimus [5]. Liktinės sistemos paprastai apima neįtikėtinai detalias veiklos taisykles ir sudaro organizacijos informacijos srautų pagrindą, kuris sutelkia verslo procesų informaciją [7]. Vienos iš šių sistemų sutrikimas gali turėti rimtų padarinių verslui. Liktinės sistemos sukelia nemažai problemų jas eksploatuojančioms organizacijoms. Svarbiausios liktinių sistemų problemos:

1) sistemas paprastai palaiko techniškai pasenusi, pernelyg lėta aparatūra, kurios palaikymo kaštai pakankamai dideli;
2) programinės įrangos palaikymas yra brangus – trūksta dokumentacijos ir bendro sistemos vidinio funkcionalumo supratimo, todėl klaidų aptikimas būna brangus ir trunka ilgai;
3) nėra aiškiai apibrėžtų sistemos sąsajų, todėl sunku integruoti;
4) liktinių sistemų funkcionalumą yra labai sunku arba net neįmanoma praplėsti;
5) neišsaugojami duomenų objektų kodifikatoriai ir objektų identifikatoriuose „paslepiami“ objektų klasifikavimo principai.

Siekiant išspręsti šias problemas, IS pakartotinėje inžinerijoje žinomi metodai gali būti skirstomi į tris klases [37]: perkūrimas, įpakavimas ir migravimas. Perkūrimo procesą sudaro egzistuojančių programų sukūrimas nuo nulio, panaudojant naują aparatūros platformą, architektūrą, įrankius ir duomenų bazes. Įpakavimas apima programos komponentų, vadinamų apvalkalais, kūrimą, kurie kitiems komponentams leidžia prieiti prie egzistuojančių programinės įrangos komponentų. Migravimas leidžia liktines sistemas perkelti į naujas aplinkas, kurios leistų lengviau

palaikyti informacines sistemas ir jas pritaikyti naujiems verslo reikalavimams, tačiau tuo pat metu turi būti išlaikytas pradinių liktinių sistemų funkcionalumas, nereikia visiškai perkurti jų. Gana dažnai liktinės sistemos keičiamos, naudojant visas tris metodų klases.

Bet kuriuo atveju, sprendžiant liktinių sistemų problemas, visų pirma reikia aiškiai suvokti bendrą informacijos sistemos vaizdą, t.y. suprasti sistemos struktūrą ir vidinius ryšius. Kadangi šis vaizdas yra nepilnas, jį būtina atstatyti iš turimų šaltinių, ir visų pirma iš pačios veikiančios sistemos. Šis atstatymo procesas yra vadinamas atvirkštine inžinerija.

Čikovskis (Chikofsky) ir Krosas (Cross) atvirkštinę inžineriją apibrėžia [12] kaip „procesą, skirtą suprasti tiriamos sistemos struktūrą ir vidinius ryšius. Atvirkštinės inžinerijos tikslas – atvaizduoti tiriamą sistemą ir palengvinti supratimo procesą. Atvirkštinę inžineriją galima pritaikyti trims pagrindiniams sistemos aspektams: duomenims, procesams ir valdymui.“

Informacijos sistemų arba duomenų manipuliavimo programų, t.y. programų, kurių svarbiausia dalis yra duomenų bazė, atvirkštinės inžinerijos proceso sudėtingumą galima sumažinti, nes duomenų bazes galima ištirti beveik nepriklausomai nuo procedūrinės dalies.

- Atvirkštinėje inžinerijoje DB loginę schemą galima gauti išsamesnę, tarpusavyje derinant duomenų struktūrines ir procedūrines savybes.
- Semantiniai skirtumai tarp duomenų konceptualių specifikacijų ir fizinio realizavimo dažniausiai yra daug mažesni nei procedūrinio kodo (duomenų reikšmė DB tikslesnė, nei programos kode).
- Statinės duomenų struktūros paprastai yra stabiliausia IS dalis.

Todėl kur kas efektyviau pirmiausia susitelkti į programos duomenų komponentų atvirkštinę inžineriją, o ne analizuoti visą programą. Čikovskis ir Krosas atvirkštinę duomenų inžineriją (ADI) apibrėžia [12] kaip „procesą, kuris susitelkia į sistemos duomenų struktūros aspektus. Tai yra metodų ir įrankių rinkinys, padedantis organizacijai suprasti jos duomenų struktūrą, funkcijas ir reikšmę“. Hainautas (Hainaut) ADI apibrėžia [22] kaip „programos duomenų bazės (DB) schemos atstatymas iš duomenų bazės valdymo sistemos (DBVS) duomenų aprašymo kalbos (DDL) teksto ir programos išeities kodo, siekiant suprasti tikrą schemų struktūrą ir reikšmę“.

Šio darbo **tyrimo sritis** atvirkštinė duomenų inžinerijos metodologija ir praktinis taikymas. **Tyrimo objektas:** atvirkštinės duomenų inžinerijos procesas.

Problema. ADI yra sudėtingas procesas, reikalaujantis ekspertų gilesnių žinių ir patirties nei tiesioginė duomenų inžinerija. Esami formalūs ADI metodai nepateikia nurodymų dėl jų taikymo praktikoje arba šie metodai yra per sudėtingi netgi nedidelės apimties ADI projektams. Dėl to

praktikoje ADI dažniausiai taikomas ekspertinis būdas ir tik pačia būtiniausia apimtimi, kad išgauti bazines struktūrines duomenų savybes - nuosavus, svetimus raktus, privalomus ir neprivalomus vienareikšmius atributus, išorinius ref- ir equ- ryšius, kurie liktinėje DB schemoje nebuvo specifikuoti ir juos tenka išgauti neformaliais metodais. Darbo dėstyme išgaunamos struktūros arba apribojimais įvardinti neišreikštomis struktūromis ir apribojimais.

Darbo tikslas – išnagrinėti bendrąjį atvirkštinės duomenų inžinerijos (ADI) procesą, sudaryti ADI metodiką, kuri būtų skirta loginės schemos atstatymui, būtų lengvai pritaikoma praktikoje, būtų automatizuota, orientuota į bazines struktūrines savybes, ir orientuta į nedidelę patirtį turintį ADI vykdytoją. Atstatyta loginė schema turi būti orientuota į duomenų migraciją, kad perkelti duomenys į naujas aplinkas būtų palaikomi standartinėmis RDBVS priemonėmis.

Duomenų migracijai aktualu atstatyti visus bazinius dalykine sritimi sąlygojamus duomenų apribojimus – formaliai specifikuotas ir neišreikštąsias duomenų struktūras.

Darbo uždaviniai:

1. Išanalizuoti atvirkštinės duomenų inžinerijos procesą;
2. Atlikti ADI aktualumo tyrimą;
3. Išanalizuoti esamus ADI metodus ir juos palyginti pagal pasirinktus kriterijus;
4. Sudaryti ADI metodiką, kurioje numatoma išgauti neišreikštąsias struktūras fizinės DB duomenų ir DML kodo analizės bei ekspertinio įvertinimo būdu;
5. Suprojektuoti ir realizuoti ADI įrankio prototipą, kuris palaiko šią metodiką;
6. Atlikti eksperimentinį metodikos tyrimą;
7. Atlikti metodikos įvertinimą apklausiant ekspertus.

Tyrimo metodologija

Pagrindinė darbo metodologija yra konstruktyvusis tyrimas, kurio tikslas yra sukurti ADI metodiką. Tyrimo objekto ir esamų metodų analizei atlikti buvo naudojamas literatūros analizės ir apibendrinimo metodas. Sprendimo konstravimui buvo taikomas objektinio sistemų projektavimo ir realizavimo metodas. Eksperimentui atlikti buvo taikomi šie metodai: natūralus eksperimentas ir laboratorinis tyrimas.

Projektavimui atlikti pasirinktas objektinis sistemų projektavimo metodas, taikant RUP projektavimo procesą ir UML projektavimo kalbą. Įrankio projektavimui buvo naudojamas Magic Draw CASE įrankis, o realizavimui Eclipse platforma.

Darbo naujumas

Sudaryta ADI metodika, kurioje numatyta svarbiausias (bazines) neišreikštąsias struktūras išgauti iš fizinės DB ir DML kodo, pasitelkiant ekspertinį - patyriminį įvertinimo būdą. Metodikoje įvestas žingsnis, kuris neminimas nei viename iš išnagrinėtų ADI metodų - apjungtų esybių aptikimas.

Apjungtomis esybėmis darbe įvardintos esybės, kurių egzempliorių identifikavimui naudojami agreguoti domenai su „paslėtomis“ klasifikavimo savybėmis, kurias gali ekspertas nagrinėti histograminiu būdu.

Darbe aprašyta ADI metodika nereikalaujanti gilių ADI ir DB teorijos žinių, akcentuojant praktinį jos taikymą.

Darbo struktūra:

Šis darbas susideda iš tokių pagrindinių dalių: atvirkštinės duomenų inžinerijos procesų ir metodų analizės, ADI metodikos konceptualaus apibrėžimo, ADI įrankio projekto, realizacijos ir eksperimentinio metodikos tyrimo ir metodikos įvertinimo, bei rezultatų apibendrinimo.

1 skyriuje pateikta atvirkštinės duomenų inžinerijos analizė. Pateikiamos ADI metodų problemos. Pateikiama probleminės srities analizė, kurioje aprašytas ADI aktualumo praktikoje tyrimas. Pristatomi esami ADI metodai, atlikta šių metodų lyginamoji analizė. Detalesnė šio skyriaus struktūra pateikta 1 skyriaus įžangoje.

2 skyriuje remiantis analizės rezultatais sudaryta ADI metodika, aprašyta jos aplinka ir vartotojai. Reikalavimų specifikacijomis detaliai aprašyti ADI metodikos etapai. Kiekvienas etapas detalizuojamas šio skyriaus poskyriuose.

3 skyriuje pateiktas ADI įrankio prototipo projektas, kuris skirtas metodikos eksperimentiniam tyrimui atlikti.

4 skyriuje aprašyta ADI įrankio prototipo realizacija ir pateiktas kontrolinis pavyzdys, su kuriuo atlikome eksperimentinį sudarytos ADI metodikos tyrimą. Tai pat pateiktas sudarytos ADI metodikos įvertinimas lyginant metodiką su kitais metodais ir kokybinis įvertinimas apklausiant ekspertus.

5 skyriuje pateiktos išvados.

1. ATVIRKŠTINĖS DUOMENŲ INŽINERIJOS ANALIZĖ

Analizės tikslai – išanalizuoti atvirkštinės duomenų inžinerijos procesą. Išanalizuoti esamus ADI metodus ir atlikti šių metodų lyginamąją analizę pagal pasirinktus kriterijus. Išanalizuoti programų supratimo metodikas, pritaikomas ADI procese.

Analizės metodai. Tyrimo objekto analizei atlikti bus naudojamas teorinės analizės ir apibendrinimo metodas. Esamų ADI metodų analizei bus taikomas mokslinės literatūros analizės ir apibendrinimo metodas.

Skyriaus struktūra:

1.1 skyrius aprašo bendrą atvirkštinės duomenų bazės inžinerijos metodologiją kaip duomenų bazės tiesioginei inžinerijai priešingą procesą.

1.2 aprašomas ADI aktualumas.

1.3 skyriuje pateiktas ADI aktualumo praktikoje tyrimas.

1.4 aprašomos problemos su kuriomis tenka susidurti vykdant ADI.

1.5 skyriuje pateikiami atvirkštinės duomenų bazės inžinerijos kokybės kriterijai.

1.6 skyriuje aprašoma bendra atvirkštinės duomenų inžinerijos metodologija ir jos procesai.

1.7 skyriuje detaliai aprašomas duomenų struktūros išgavimo procesas. Kiekvienas etapas detalizuojamas poskyriuose. Taip pat aprašomi pagrindiniai neišreikštųjų apribojimų ir struktūrų tipai. Pristatomi galimi informacijos šaltiniai ir metodikos duomenų struktūrai atstatyti.

1.8 skyriuje aprašomas programų supratimo metodų panaudojimas atvirkštinei duomenų inžinerijai atlikti.

1.9 skyriuje aprašomas bendras duomenų struktūros modelis, kuris gali tiksliai išreikšti skirtingų abstrakcijos lygių schemas. Aprašyta, kaip šis modelis gali išreikšti fizinės, loginės ir konceptualios schemų sąvokas.

1.10 skyriuje pristatyti esami ADI metodai, atlikta šių metodų lyginamoji analizė.

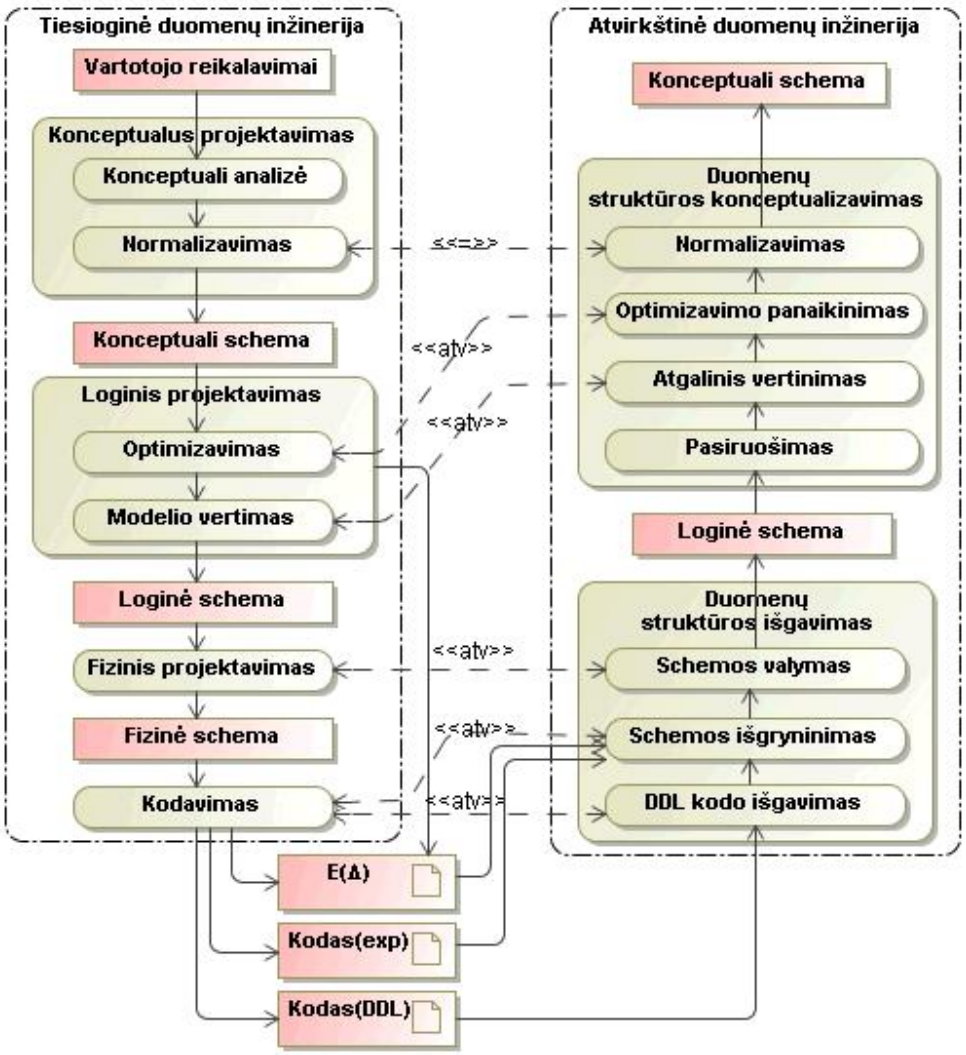
1.11 skyriuje pateiktos analizės išvados.

1.1. Atvirkštinė ir tiesioginė duomenų inžinerijos

Kadangi atvirkštinė inžinerija atstato konceptualiąją schemą iš veikiančio kodo, todėl galima teigti, kad šis procesas yra tiesiog atvirkštinis veiksmas tiesioginei inžinerijai [6].

Siekiant apsukti hierarchiškai išskaidomą procesą, reikia jo sudėtinius procesus išdėlioti atvirkštine tvarka, o tuomet kiekvieną procesą pakeisti jam atvirkščiu. Duomenų bazės, kuri

išvystyta pagal idealų metodą, tiesioginė inžinerija apima trijų rūšių rezultatus [20] (atvirkštinės inžinerijos proceso šaltinius): $Kodas_{ddl}$ atvaizduoja DDL kodą, $Kodas_{ext}$ atvaizduoja ne DDL kodą (struktūros ir apribojimų dalį, kurios neišreiškia DDL kodas) ir $E(\Delta)$ yra specifikacijos dalis, kuris prarasta atliekant tiesioginę inžineriją. Tiesioginė inžinerija susideda iš šių procesų (kairė pusė (1.1 pav.)



1.1 pav. Tiesioginė ir atvirkštinė inžinerija

- Konceptualus projektavimas. Pateikia konceptualiąją schemą, nuo aparatūros nepriklausantį DB informacijos struktūrų aprašymą. Susideda iš dviejų procesų:
    - Konceptuali analizė vartotojo reikalavimus paverčia į formalias specifikacijas.

- Normalizavimas konceptualiajai schemai suteikia kokybę, pvz. normalizavimas, minimizavimas, skaitomumas, aiškumas ir suderinamumas su įmonės standartais.

- Loginis projektavimas. Konceptualiąją schemą transformuoja į su DBVS suderintą loginę schemą. Schemą išreiškia pasirinkto DBVS duomenų modelis, schema atitinka vykdymo kriterijus, pvz. vietos ir laiko našumo klausimai. Susideda iš dviejų procesų:
  - Optimizavimas pakeičia schemą, kad suteiktų didesnį našumą.
  - Modelio vertimas pakeičia schemą į duomenų struktūras, kurios suderinamos su DBVS modeliu.

Kai kurių konceptualios schemos struktūrų neverčia į loginę schemą ($E(\Delta)$). Šie semantikos praradimai įvyksta, nes DBVS modelis silpnesnis nei konceptualus modelis ir tam tikros struktūros yra per sudėtingos (brangios) išreikšti ar programa tiesiog jų nepalaiko.

- Fizinis projektavimas. Nustato techninius parametrus ir fizines struktūras, kad sugeneruotų fizinę schemą.
- Kodavimas. Pateikia vykdomą DB versiją. Verčia DBVS struktūras į DBVS-DDL aprašymą ($Kodas_{ddl}$) ir ne DBVS struktūras pvz. į procedūrinę dalį ($Kodas_{exp}$). Kai kurių struktūrų neišverčia (sąmoningai ar netyčia) į kodą ($E(\Delta)$).

ADI procesų sąryšiai su tiesioginės inžinerijos procesais pateikti (1.1 pav.) Tiesioginės/atvirkštinės inžinerijos procesų sąsaja „*atv*“ reiškia, kad procesai vienas kitam atvirkštiniai, o „=“ nurodo, kad jie sutampa. ADI susideda iš dviejų žingsnių:

- Duomenų struktūros išgavimas. Atstato loginę schemą iš veikiančio kodo ($Kodas_{ddl}$ ir $Kodas_{exp}$). Susideda iš $Kodas_{ddl}$ išgavimo (DDL kodo išgavimas) ir $Kodas_{exp}$ išgavimo (schemos išgryninimas). Galima atstatyti kai kurias nerealizuotas specifikacijos dalis ($E(\Delta)$), analizuojant kitus informacijos šaltinius, pvz. duomenys, vartotojo ar programuotojo apklausa ir t.t. Siekiant išgauti loginę schemą, reikia panaikinti fizinio projektavimo proceso rezultatus. Tai gana paprasta, nes tiesioginės inžinerijos procesas prie loginių struktūrų prideda technines struktūras; atvirkštinis procesas vadinamas schemos valymu.
- Duomenų struktūros konceptualizavimas. Atstato konceptualiąją schemą iš loginės schemos. Pašalina ir transformuoja optimizavimą (optimizavimo panaikinimas – optimizavimui atvirkščias veiksmas). Interpretuoja logines struktūras konceptualių

struktūrų sąvokomis (atgalinis vertimas – modelio transformacijai atvirkščias veiksmas). Konceptualiajai schemai suteikia kokybės (normalizavimas).

Metodas yra teisingas, jei programa, kuriai atliekama atvirkštinė inžinerija, suprojektuota atsižvelgiant į idealų metodą [23] . Bet daugelis programų suprojektuota atsižvelgiant į empirinius metodus arba palaikymo etape buvo vystomos. Tokioms programoms pasiūlyta metodologija sunkiai pritaikoma.

ADI tikslas ne atstatyti schemą, kurią naudojo projektuojant DB, bet išgauti galimą konceptualiąją schemą, kuri išreiškia dabartinę DB semantiką. Tą pačią DB gali specifikuoti daugiau nei viena konceptuali schema. Visos schemos turi tą pačią semantiką, t.y. jos atvaizduoja tą pačią sritį.

ADI yra sudėtingas procesas, nes dalis semantikos egzistuoja už sistemos ribų, t.y. nebuvo realizuota sistemos kodo dalyje ($E(\Delta)$). Sistema (DBVS, programa) nežino apie šių struktūrų egzistavimą. Šią semantiką galima rasti kitur, pvz. programos aplinkoje, dokumentacijoje ar duomenyse. Todėl išgaunant reikia išanalizuoti kitus informacijos šaltinius, ne tik programos kodą.

1.2. ADI aktualumas

Prarastą IS duomenų struktūros aprašymo informaciją tenka atstatyti, nes jos niekada nebuvo, ji prarasta arba nebeatitinka realybės dėl nuolatinio programos palaikymo. Aktuali dokumentacijos versija reikalinga įvairiais atvejais [15], [20]:

- Palaikymas. Siekiant efektyviai palaikyti programą, reikia gerai suprasti esamą sistemą, tuomet teisingai įvertinami ir reikalaujamo pakeitimo kaštai. Į sistemą žiūrint globaliai, numatomi pašaliniai pakeitimo efektai.
- Duomenų bazės administravimas. Kad įvertintų reikalingą saugojimo vietą ir nuspėtų duomenų bazės evoliuciją, administratorius turi gerai pažinti duomenų bazę. Gerai žinant duomenis naudojančią programą, galima optimizuoti duomenų bazę ir pasiekti optimalų reakcijos laiką.
- Duomenų formatų keitimas. Norint atlikti tam tikrų duomenų vertimą (pvz. 2000 metų problema arba euro įvedimas), reikia tiksliai žinoti kiekvieno lauko semantiką.
- Duomenų migravimas. Sėkmingai migruoti duomenis iš vienos duomenų bazės į kitą galima tik turint nuodugnias programos žinias. Duomenys migruojami, kai keičiasi aparatūros ar programinės įrangos aplinkos, siekiant paskelbti duomenis internete, įdiegti įmonės išteklių valdymo sprendimus ar apjungti kelias IS.

- Duomenų išgavimas. Kad dabartinės duomenų bazės duomenimis galėtų naudotis kitos IS, reikia turėti duomenų semantikos žinias. Duomenų saugojimo ir duomenų išgavimo įrankiai išgauna šias žinias, todėl gali palaikyti strateginius sprendimus. Duomenų semantikos žinios taip pat palaiko e-verslo ir B2B programas.
- Pakartotinis panaudojimas. Analitikams reikalaujant naujų funkcijų, tačiau siekiant išvengti duomenų struktūrų dubliavimo, svarbu žinoti, kokios duomenų struktūros egzistuoja.
- Esamos programinės įrangos įvertinimas. ADI galima naudoti, kad įvertinti bendrą programinės įrangos sistemų kokybę. Duomenų struktūra, turinti žymių projektavimo trūkumų, nurodo prastai realizuotą programinę įrangą.

1.3. ADI aktualumo praktikoje tyrimas

1.3.1. Tyrimo formulavimas

Kad nustatyti pasirinktos temos aktualumą praktikoje buvo atliktas aktualumo tyrimas. Pasirinkta tyrimo forma buvo interviu, apklausiant srities praktikus. Tokia forma buvo pasirinkta dėl to, kad sudarius pirminę anketą ir atlikus pilotinę apklausą pastebėta, jog anketa negalės parodyti nemažos dalies aktualių aspektų jos smarkiai neišplečiant.

Tyrimui buvo iškelti tokie uždaviniai:

- Tyrimas turi parodyti, kokios technologijos, produktai buvo plačiai naudojami praktikoje kuriant sistemas įvairiais laikotarpiais, tiek sistemas pagal individualius užsakymus, tiek kaip produktus laisvai parduodamus rinkai, įskaitant šių technologijų pasirinkimo motyvus.
- Tyrimas turi apibūdinti praktiką susijusią su sistemų gyvavimo ciklo galutiniais etapais, perėjimais iš vienos sistemos į kitą, su tuo susijusius priimamus sprendimus ir jų motyvus.
- Tyrimas turėtų parodyti ADI taikymo praktiką, taikymo ar netaikymo motyvus.
- Tyrimas turėtų parodyti kokios šiuo metu praktikoje naudojamos technologijos ir sistemos labiausiai tikėtina labiausiai gresia ateityje tapti liktinėmis.

1.3.2. Tyrimo eiga

Tyrimas buvo atliktas apklausiant 16 informacijos technologijų srities specialistų, kurie perteikė savo patirtį. Stengtasi apklausti kuo įvairesnės patirties specialistų.

3 apklaustieji – asmenys, konsultavę ar vadovavę IT sistemų diegimo ir kūrimo projektams, turintys daugelio projektų patirtį.

5 apklaustieji – įmonių, kuriančių ir diegiančių savo produktus Lietuvos rinkoje, atstovai.

8 iš apklaustųjų – IT specialistai, turintys įvairios patirties: kaip sistemos kūrėjai ir vystytojai įmonės viduje (kaip įmonės darbuotojai), ir kaip vykdantys sistemų priežiūros ir administravimo funkcijas.

1.3.3. Tyrimo išvados

Negalima teigti, kad tyrimo imtis yra reprezentatyvi ir kad pateikti skaičiai savo absoliučiais dydžiais parodo bendrą situaciją. Atlikto tyrimo apimtis buvo beveik maksimali, kokią leido tyrimo autorių ištekliai. Visgi galima teigti, kad tyrimas parodo tam tikras tendencijas.

Atlikus interviu, buvo atskleistos tokios tendencijos. Iki 1990 metų visi informacijos sistemų kūrimo žmoniškieji ištekliai buvo sukoncentruoti stambių pramonės įmonių ir valstybinių įstaigų taip vadinamų automatizuoto valdymo sistemų (AVS) vystymo ir priežiūros padaliniuose. Po maždaug 1990 metų Lietuvą pasiekusi personalių kompiuterių banga smarkiai pakeitė šią situaciją. Daugelis AVS padalinių buvo reformuoti. Dalis jų virto nepriklausomomis įmonėmis, kurios pradėjo kurti sistemas, kaip produktus vietinei rinkai. Šie produktai paprastai apėmė tik standartinę įmonės veiklą: buhalterinę apskaitą, sandėlių apskaitą, darbo užmokesčio apskaitą ir pan. Kita AVS personalo dalis liko susijusi su pačiomis įmonėmis ir kūrė tiek bendruosius tiek specifinius įmonės procesus apimančias sistemas. Šių sistemų nebūtų galima pavadinti pilnavertėmis sistemomis, kadangi didžiąja dalimi tai buvo smulkios, gana izoliuotos, siauro koncentruoto funkcionalumo programos, kurios buvo vadinamos „uždaviniais“. Tiek šios sistemoms, tiek sistemoms kaip produktams buvo kuriamos tomis naujomis technologijomis ir įrankiais, kurie atėjo su personalinių kompiuterių banga: DOS operacinėje sistema, FoxPro, dBase, kiek vėliau Clarion specializuotais įrankiais.

Situacija iš esmės nesikeitė iki maždaug 1995, kai atsirado nauji sistemų kūrimo įrankiai Microsoft Windows terpei. Panašiai tais pačiais metais Lietuvos rinkai pradėti siūlyti ir tarptautiniai produktai: Scala, Navision ir kiti. Be to ir vietiniai produktai buvo jau žymiai geriau išvystyti, ir jų kaina santykinai mažėjo. Dėl to bendrą įmonių veiklą apimančios sistemos vis daugiau buvo keičiamos iš savo gamybos į pirktinius produktus. Ir tik specifinę veiklą apimančios vienetinės sistemos buvo vystomos ir migruojamos į naujas technologijas.

Laikotarpyje nuo 1995 pradėti naudoti tokie įrankiai: Microsoft Access, Visual FoxPro, Oracle, Borland Delphi ir kiti. Naujuose produktuose ėmė vyrauti dviejų lygių architektūra. Taipogi būtina paminėti tuo metu išplitusį Microsoft Excel įrankį, kadangi jis buvo labai plačiai naudotas ir net iki šiol naudojamas kaip alternatyva elementarioms apskaitos sistemoms, sistemų užuomazgų kūrimui. Daugelio tuo laikotarpiu susikūrusių smulkių įmonių kompiuterizuota apskaita būdavo pradedama arba vietiniu produktu, arba būtent šiuo įrankiu.

Nuo 2000-2002 metų naudojamų įrankių, technologijų ir produktų spektras tapo dar įvairesnis. Šį spektrą papildė žiniatinklio technologijos, trijų lygių architektūra, atviras kodas.

Tyrime nustatyta, kad praktikoje labiausiai ADI yra pritaikytina duomenų migracijai. T.y. duomenų perkėlimui iš savo gamybos sistemos į įsigytą produktą, arba keičiant vieną produktą kitu. Abiem atvejais būtent naujos sistemos autoriai būna šio proceso vykdytojai.

Tyrimas parodė, kad didžiajai daliai duomenų migracijos atvejų ADI nėra aštri problema. Tam nustatytos tokios priežastys.

1. Veiklos apskaitai būdingas periodiškumo principas. t.y. visa veiklos apskaita paprastai yra suskirstoma į laikotarpius. Laikotarpio pabaigoje sudaromos suvestinės, kurios perkeliamos į naujo laikotarpio pradžią. Migracija iš vienos sistemos į kitą atliekama būtent tokių laikotarpių sandūroje. Tokiu būdu perkeliamų duomenų kiekis sumažėja tiek, kad duomenis pakanka perkelti rankiniu ar pusiau rankiniu būdu iš standartinių ataskaitų, o ne duomenų bazės. Tokių sistemų migracijų atveju, kaip taisyklė, senoji sistema nėra demontuojama, o paliekama kaip archyvinės informacijos šaltinis.

2. Sistemos yra nesudėtingos duomenų struktūros, dėl to migracija nereikalauja gilių srities žinių ir formalių ADI metodų. Dalies duomenų perkėlimą (klasifikatoriai, nomenklatūra, pagrindiniai sąrašai) pakankamai efektyviai gali atlikti IT specialistas su minimaliomis probleminės srities žiniomis, pasitelkęs pusiau rankinius būdus. Su šia išvada susietini tokie

- Duomenų migracija yra visiškai standartinė procedūra ir paslauga, kurią teikia naujos sistemos kūrėjas ar diegėjas.
- Diegėjas yra linkęs duomenų migraciją atlikti prastesne kokybe, jei migruojama iš vienetinės sistemos, ir geresne – jei iš kito produkto. t.y. migracijos kokybei bus skiriama daugiau dėmesio, jei migracijos metu sukauptos žinios apie konkrečia sistemą gali būti pakartotinai panaudotos.
- Migracija absoliučia dauguma atveju vykdoma iš vienos reliacinės schemos į kitą reliacinę schemą. Todėl schemos konceptualizavimas didžiąja dalimi atvejų nėra tikslingas.

3. Labai dažnai senos sistemos autoriai būna prieinami ir gali pateikti pakankamai informacijos apie sistemos veikimą ir duomenų struktūrą.

4. Tarp senos ir naujo sistemų duomenų struktūrų gali būti nesuderinamų skirtumų, dėl kurių visiškai automatizuotas visų duomenų perkėlimas tampa neįmanomas. Tokiu atveju perkeliama tik esminė informacija, kuri vėliau tikslinama rankiniu būdu iš pirminių šaltinių.

Tyrimas parodė, kad tam tikrais atvejais ADI visgi tampa labai aktualia problema. Pirmoji sąlyga yra ta, kad sistemos autoriai dėl kokių nors priežasčių negali ar nesutinka bendradarbiauti ir nėra išsamios sistemos dokumentacijos.

Antra sąlyga yra ta, kad sistema turi pasižymėti kažkuriomis iš šių savybių:

- Sistema yra unikali tiek savo veikla, tiek duomenų struktūra. Pvz.: valstybinis nekilnojamojo turto registras.
- Sistemos paskirtis grindžiama didelio kiekio ir neriboto senumo duomenų saugojimu ir analize. Pvz.: Draudimo kompanijos klientų ir sutarčių registras.

Įvertinus tai, jog šiuo metu ADI problema aštriausia yra vienetinių arba retų sistemų, sukurtų „nuo nulio“ migracijos metu, darytina išvada, jog ateityje ši tendencija nepasikeis. Įvertinus kokiomis technologijomis ir įrankiais tokios sistemos kuriamos šiuo metu, darytina išvada, kad būtent tokių technologijų sistemoms bus taikoma ADI. Tokios technologijos tai: dviejų lygių, trijų lygių architektūros, reliacinis duomenų modelis, žiniatinklio technologijos, atviro kodo technologijos.

1.4. ADI problematika

Duomenų struktūros išgavimas yra kur kas sudėtingesnis procesas, nei paprasta DB DDL teksto analizė. Duomenų aprašymo kalba yra DBVS įrangos dalis, skirta DB struktūroms deklaruoti. Siekiant suprasti sistemą iš veikiančių komponentų, susiduriama su įvairiomis problemomis, pvz.: neišverčiamos duomenų struktūros ir apribojimai, nestandartiniai realizavimo metodai, senos ir specializuotos DBVS, blogai suprojektuotos duomenų struktūros. Kadangi DDL nėra vienintelis informacijos šaltinis, reikia ištirti kitus dokumentus ir sistemos komponentus, kuriuos kur kas sudėtingiau analizuoti ir kurie mažiau patikimi. Dažniausiai pasitaikančios problemos su kuriomis tenka susidurti vykdant ADI, yra šios [4], [8], [22], [39]:

- DBVS modelių silpnumas. Kai kurių DBVS techninis modelis gali išreikšti tik mažą konceptualios schemos duomenų struktūrų ir apribojimų poaibį. Atmestus struktūras

apdoroja procedūriniai programos komponentai: programos, dialogo procedūros, trigeriai ir t.t. Juos galima atstatyti analizuojant procedūras.

- Numanomos struktūros. Kai kurios struktūros apgalvotai (arba netyčia) nėra deklaruoti DB DDL specifikacijoje, siekiant optimizuoti, apsirikus ar norint suderinti su senesnėmis DBVS versijomis. Jie realizuojami tuo pačiu būdu, kaip ir atmestos struktūros dėl DBVS modelių silpnumo.
- Optimizuotos struktūros. Dėl techninių priežasčių, pvz. laiko/vietos optimizavimo, daugelis duomenų bazių struktūrų apima nesemantines struktūras. Be to, perteklinės ir nenormalizuotos struktūros pridedamos, siekiant pagerinti sistemos reakcijos laiką.
- Nevykęs projektas. Ne visas DB sukūrė patyrę projektuotojai. Naujokai arba nepakankamai apmokyti programuotojai, neišmanantys DB teorijos ir metodologijos, dažnai sukuria prastas arba netgi klaidingas struktūras.
- Pasenusios struktūros. Dabartinės programos gali palikti ir ignoruoti kai kurias DB dalis.
- Nenuoseklus standartas. Liktines sistemas vystė ir palaikė daug metų, per tą laiką pasikeitė programavimo standartai ir metodologijos, taip pat programinė ir techninė įranga. Sistema nebėra vientisa, tai greičiau mažų posistemių rinkinys, kiekvienas su savo savybėmis. Kai kuriais itin nepalankiais atvejais, sistema naudoja kelias programavimo kalbas arba daugiau nei vieną DBVS.
- Sistemos dydis. Sistemos integruoja vis daugiau verslo procesų, jos vystomos daugelį metų. Tokios sistemos gali būti labai didelės. Pvz.: keletas milijonų kodo eilučių ir daugiau nei 500 lentelių nėra išimtiniai atvejai. Taigi, smulkiems projektams naudojami metodai nebetinka vidutinio dydžio ar dideliems projektams.

Didžioji neišreikštųjų struktūrų ir apribojimų dalis paslepiama programos išeities kode, todėl jis tampa patikimiausia vieta, kur galima juos rasti. Programos analizė reikalauja sudėtingų metodikų, priklausančių programos supratimo sričiai.

1.5. Atvirkštinės duomenų inžinerijos kokybės kriterijai

Pagrindinis atvirkštinės duomenų bazės inžinerijos tikslas – gauti geros kokybės duomenų bazę [38]. Schemos transformavimo kokybė skiriasi nuo schemų ar duomenų modeliavimo kokybės, kuri dažniausiai remiasi intuityviomis sąvokomis – skaitomumas, aiškumas ar išraiškingumas. Šios

sąvokos svarbios konceptualiajame duomenų bazių projektavime. Atvirkštinei duomenų inžinerijai būdingi kiti kriterijai:

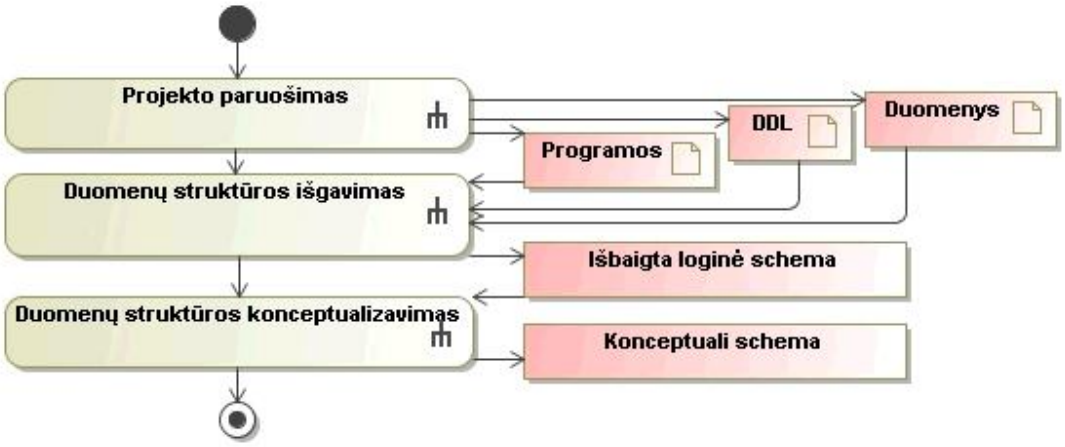
- Teisingumas: schema yra sintaksiškai teisinga, jei visos schemos sąvokos yra tinkamai apibrėžtos. Schema yra semantiškai teisinga, jei visos jos sąvokos vartojamos pagal jų apibrėžimą, pavyzdžiui, asociacija nėra modeliuojama kaip apibendrinimas ar atvirkščiai.
- Primityvumas: schema yra primityvi, jei visos schemos struktūros priskirtos ne daugiau kaip vienai realaus pasaulio abstrakcijai.
- Minimalumas: schema yra minimali, jei negalima pašalinti nė vieno schemos elemento, nepraradus informacijos. Schemos, kurios nėra minimalios, dažnai yra daug sunkiau suprantamos.
- Tinkamumas: schema yra tinkama, jei visi jos objektai ir ryšiai yra aiškiai ir tiksliai atvaizduoti duomenų bazės schemoje, naudojant atitinkamo modeliavimo sąvokas.
- Normiškumas: duomenų bazės schema turi tenkinti tam tikrą norminę formą.

Svarbiausias kokybės kriterijus yra semantinis teisingumas. Minimalumo ir primityvumo kriterijai yra glaudžiai susiję su semantinės abstrakcijos lygiu, kadangi pertekliškumas ir paslėpti objektai mažina abstrakcijos lygį. Tinkamumo kriterijus svarbus tuo, kad leidžia visus ryšius ir objektus tiksliai aprašyti modeliavimo sąvokomis, o ne paslėptais ryšiais ar loginiu suvokimu.

1.6. ADI metodologija

Atvirkštinė duomenų inžinerija (ADI) – analitinių metodų taikymas vienam ar daugiau duomenų šaltinių, siekiant iš pastarųjų išgauti struktūrinę informaciją, kad būtų pagerintas duomenų bazės projektas arba pateikta trūkstama schemų dokumentacija [15].

Bendra atvirkštinės duomenų inžinerijos metodologija susideda iš šių procesų [20]: schemos paruošimas, duomenų struktūros išgavimas ir duomenų struktūros konceptualizavimas (1.2 pav.). Paruošimo procesas ne visai duomenų atvirkštinės inžinerijos procesas, jo tikslas yra surinkti ir įvertinti būtinų informacijos šaltinių aktualumą ir susipažinti su sritimi. Paskutiniai du procesai atkuria dvi skirtingas schemas ir reikalauja skirtingų koncepcijų, pagrindimų ir įrankių.



1.2 pav. Pagrindiniai atvirkšinės duomenų inžinerijos procesai

Projekto paruošimas yra pradinis žingsnis, kurio tikslas nustatyti ir įvertinti analizuotinus komponentus, įvertinti būtinus išteklius ir apibrėžti operacijų planą. Net jei šis žingsnis nėra griežtai atvirkštinės inžinerijos, o greičiau valdymo užduotis, viso projekto valdymas ir būtinų informacijos šaltinių nustatymas nėra paprasta užduotis. Šis žingsnis yra esminis teisingam projekto vystymui. Projekto paruošimas susideda iš šių procesų:

- Būtinų komponentų ir jų kokybės nustatymas. Nustatomi ir įvertinami failai, programos, langai, formos, ataskaitos, duomenų žodynai, saugyklos, programiniai šaltiniai, duomenys ir dokumentacija. Ne visi komponentai duoda tą pačią informacijos kokybę ir kiekybę. Pvz. DDL pateikia labai tikslią informaciją ir ją lengvą analizuoti, procedūrinis išeities kodas taip pat pateikia tikslią informaciją, bet jo analizė yra labai brangi. Dokumentacija, jei aktuali ir kruopščiai paruošta, taip pat gali būti labai naudinga ir lengvai analizuojama (netektų prasmės netgi ADI procesas). Kada dokumentacija pasenusi ir/arba restruktūrizuota, jos analizė užtrunka ir gali vesti prie neteisingų prielaidų.
- Architektūros atstatymas. Išgauna pagrindinius sistemos procedūrinius ir duomenų komponentus bei jų ryšius.
- Projekto apibrėžimas. Tiksliai apibrėžia, kokios programos dalys analizuojamos ir kokie tikėtini rezultatai. Paprastai ADI vykdoma tik programos daliai. Svarbu tiksliai apibrėžti analizuojamų programų ribas, nes programos smarkiai sąveikauja ir gali būti sunku žinoti, pvz. ar esybė priklauso ar nepriklauso analizuojamai programai. Reikia nustatyti rezultato pobūdį (tikėtinas rezultatas: konceptuali schema, loginė schema ar

nauja duomenų bazė) ir privalumus bei trūkumus (pvz. išgaunama tik esybių tipų struktūra).

- Išteklių nustatymas. Įvertina reikalingus įgūdžių, darbo jėgos, terminų, aparatūros, įrankių ir biudžeto išteklius.
- Operacijų planavimas. Kiekvienam projekto žingsniui numatoma data ir biudžetas.

Duomenų struktūros išgavimas yra svarbiausia ir sunkiausia ADI dalis. Duomenų struktūros išgavimo proceso tikslas – atstatyti išbaigtą loginę schemą, kur pavaizduotos išreikštos ir neišreikštos struktūros bei savybės. Pagrindinė problema, kad didelė dalis struktūrų ir apribojimų neišreikšti, t.y. jie ne deklaruoti, bet realizuoti procedūrinėse programų dalyse. Kad išgauti išreikštąsias ir neišreikštąsias struktūras ir apribojimus, reikia panaudoti visus įmanomus informacijos šaltinius. 1.7 skyriuje plačiau aprašysime duomenų struktūros išgavimo procesą, kadangi šis procesas yra svarbiausia ir sunkiausia ADI dalis.

Duomenų struktūros konceptualizavimo procesas semantines loginės schemos struktūras nustato kaip konceptualios schemos struktūras. Kai kurias struktūras pakankamai lengva interpretuoti (pvz. standartinis išorinis raktas yra vienas su daug ryšio tipo realizavimas), kitiems reikia sudėtingų realizavimo ir optimizavimo metodikų. Šis etapas nepatenka į šio darbo apimtį, dėl šių priežasčių detaliau jo neanalizuosime.

1.7. Duomenų struktūros išgavimas

Duomenų struktūros išgavimo etapas analizuoja egzistuojančią liktinę sistemą, kad atstatytų išbaigtą loginę schemą. Šiame skyriuje detalizuojamas duomenų struktūros išgavimo procesas. Duomenų struktūros išgavimo procese yra analizuojami visi galimi informacijos šaltiniai, tam kad išgauti neišreikštuosius apribojimus ir struktūras.

1.7.1 – 1.7.4 poskyriuose detaliai aprašyti duomenų struktūros išgavimo etapai.

1.7.5 poskyryje aprašomi pagrindiniai ieškomi neišreikštieji apribojimai ir struktūros.

1.7.6 poskyryje pristatomi galimi informacijos šaltiniai duomenų struktūrai atstatyti.

1.7.7 poskyryje pateiktos metodikos duomenų struktūrai atstatyti, naudojant skirtingus informacijos šaltinius.

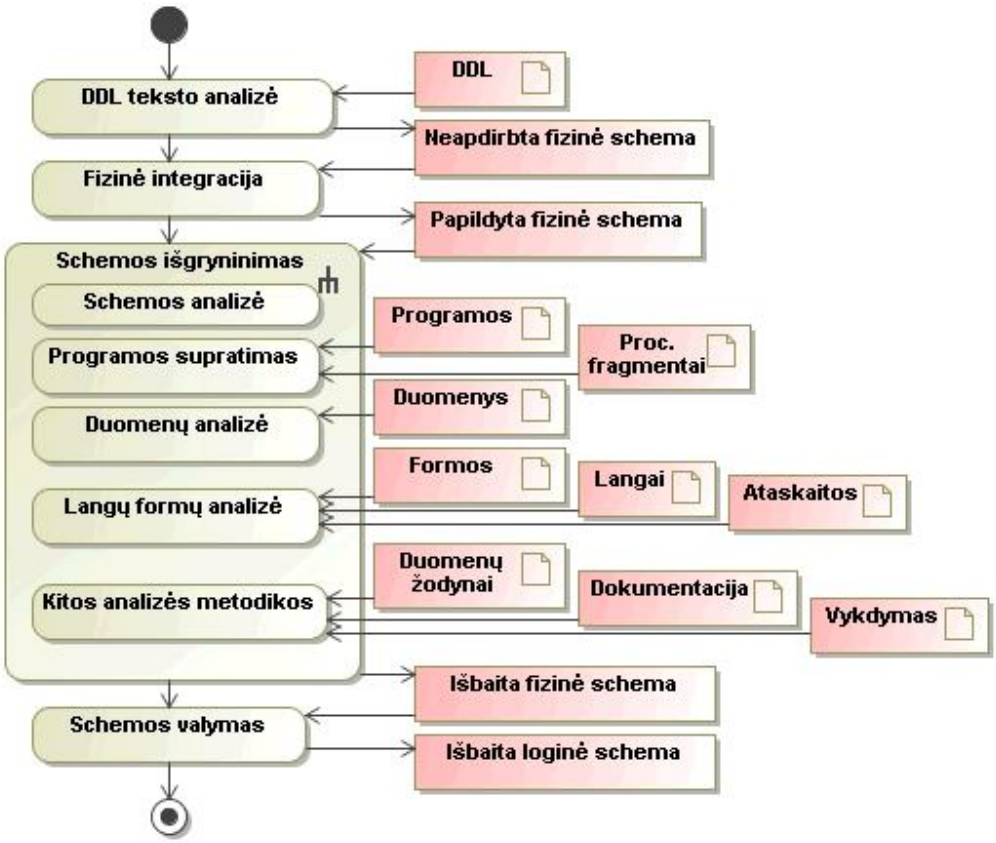
Duomenų struktūros išgavimo etapas atstato išbaigtą schemą, kuri apima tiek išreikštąsias tiek neišreikštąsias struktūras ir apribojimus. Ši schema, tai tokia schema, kurią programuotojas turi pilnai suvokti, kad galėtų teisingai kurti ir vystyti sistemą. Joje aprašomi visi rinkiniai, DB esybių

tipai ir atributai jų prasmingais pavadinimais ir visais apribojimais, tiek išreikštaisiais, tiek neišreikštaisiais, kuriuos turi atitikti duomenys [20].

DB sistemos paprastai pateikia schemos aprašymą tam tikra skaitoma ir apdorojama forma (duomenų žodyno turinys, DDL tekstas ir t.t.). Nors šioje schemoje gali trūkti esminės informacijos, tačiau tai gali būti pradinis taškas, kurį vėliau galima išgryninti analizuojant kitus programos komponentus (vaizdinius, poschemes, lango ir ataskaitų išdėstymą, procedūras, dokumentacijos fragmentus, DB turinį, programos vykdymą ir t.t.). Daugelį realių programų reikia analizuoti detaliau, ne tik nustatyti programoje deklaruotus esybių tipus.

Pagrindiniai duomenų struktūros išgavimo proceso etapai (1.3 pav.):

- DDL teksto analizė. Analizuoja fizinę schemą, DDL kodą ar vartotojo rodinius, kad išgauti išreikštuosius apribojimus ir struktūras. Išgauna neapdirbtą fizinę schemą. Schema apima tik visas deklaruotas duomenų struktūras ir apribojimus.
- Fizinė integracija. Jei apdorotas daugiau nei vienas šaltinis, tuomet išgaunamos kelios schemos. Šios schemos apjungiamos į vieną bendrą – papildytą fizinę schemą.
- Schemos išgryninimas. Papildyta fizinė schema praturtinama neišreikštaisiais apribojimais, kurie aptinkami analizuojant įvairius informacijos šaltinius, pvz. procedūrinis kodas, lango išdėstymas ir pan. Tokiu būdu sudaroma išbaigta fizinė schema. Schema sukaupia viso duomenų struktūros išgavimo etapo duomenis, ji apima visas aptiktas duomenų struktūras ir apribojimus. Schema apima fizinio DB realizavimo detales (indeksai, puslapio dydis ir pan.) ir duomenų struktūrą.
- Schemos valymas. Pašalina fizines struktūras, kurių nebereikia, kad pateikti išbaigtą loginę schemą.



1.3 pav. Duomenų struktūros išgavimas

Išbaigta loginė schema apima visas duomenų struktūros išgavimo etapo duomenų struktūras ir apribojimus. Ši schema gali būti nesuderinama su DBVS dėl dviejų priežasčių. Ji yra išgryninimo proceso, kuris pagerina schemą atrastais apribojimais, rezultatas. Tarp šių apribojimų daugiausia tokių, kurių negali išreikšti DBVS. Išbaigta loginė schema aprašo nuolatinių programos duomenų struktūrą, o tam tikros programos naudoja daugiau nei vieną DBVS. Jei šios DBVS turi skirtingus duomenų modelius, tuomet išgauta loginė schema nesuderinama su nei viena iš jų. Schema vis dar pakankamai artima dabartinei realizacijai. Ji yra tarsi programuotojo požiūris į DB su visais apribojimais ir detalėmis, kad parašyti ar pakeisti programas, kurios naudoja DB.

1.7.1. DDL teksto analizė

Pats paprasčiausias duomenų struktūros išgavimo proceso etapas. Šis procesas analizuoja duomenų struktūros deklaracijos sakinius (konkrečios DDL), pateiktus schemos sukūrimo scenarijuose ir/arba programos deklaracijose. Išgauna fizinę schemą, kuri vadinama neapdirbta fizine schema. Fizinių specifikacijų išgavimas iš sistemos duomenų žodyno, pvz. iš DBVS sistemos katalogo yra tas pats kaip DDL analizė.

Iš kiekvienos DDL išraiškos išveda fizines abstrakcijas. Taisyklių rinkinius lengva formuoti daugeliui DBVS, išgautas abstraktus fizinis modelis apima pakankamai turtingą savybių rinkinį. Kiekviena DDL išraiška apima logines sąvokas (esybių tipai, atributai ir integralumo apribojimai), konceptualias sąvokas (identifikatoriai) ir sakinius, skirtus fizinėms struktūroms (pvz. indeksai) išreikšti.

Beveik visi CASE įrankiai pateikia DDL kodo analizės įrankius populiariausioms DBVS. Kai kurie taip pat gali išgauti reliacines specifikacijas iš sistemos duomenų žodyno.

1.7.2. Fizinė integracija

Kai apdorojamas daugiau nei vienas DDL šaltinis, tuomet išgaunamos kelios schemos. Toks atvejis galimas, jei programa naudoja kelias skirtingas DB, galbūt net skirtingas DBVS. Išgautos schemos turi bendrų elementų, tačiau kiekviena schema gali turėti tik jai būdingų elementų. Atlikus analizę, visas išgautas fizines schemas reikia apjungti. Galutinė loginė schema privalo apimti visų dalinių schemų specifikacijas, todėl reikia atlikti fizinės integracijos procesą. Procesas skiriasi nuo literatūroje siūlomų metodų konceptualioms schemoms integruoti [19]. Fizinių schemų integravimo savybės:

- Kiekviena fizinė schema yra unikalaus ir pilnai identifikuoto fizinio objekto (liktinės DB) rodinys. Sintaksiniai ir semantiniai konfliktai parodo ne skirtingus vartotojo rodinius, bet greičiau nepakankamą analizę. Sintaksinis konfliktas kyla, jei dvi to paties fizinio objekto deklaracijos yra nesuderinamos.
- Atitikimo euristikai galima naudoti fizinius ir techninius duomenų aspektus, pvz. duomenų lauko tipas ir ilgis.
- Rodinių gali būti labai daug. Išbaigtą bendrą schemą galima gauti tik integravus visus dalinius rodinius.

Procesas integruoja tik skirtingų neapdirbtų fizinių schemų, kurios yra skirtingi to paties fizinio objekto rodiniai, elementus. Procesas neintegruoja skirtingų fizinių objektų, kurie turi tą pačią semantiką.

1.7.3. Schemos išgryninimas

Pagrindinė duomenų struktūros išgavimo etapo problema – naudojant išgryninimo procesą atrasti ir išreikšti struktūras ir apribojimus, kurie buvo netiesiogiai realizuoti arba atmesti vystant programą. Yra daug neišreikštųjų apribojimų, pagrindiniai iš jų: esybės tipo ir atributo išskaidymas, identifikatorius, integralumo apribojimas, funkcinė priklausomybė, reikšmingi pavadinimai [20].

Schemos išgryninimo procesas yra sudėtingas, nes reikia apžiūrėti įvairius informacijos šaltinius, kad nustatyti neišreikštuosius arba prarastus apribojimus. Išgauta fizinė schema papildoma šiais apribojimais ir tokiu būdu sudaroma išbaigta fizinė schema. Proceso sudėtingumas priklauso nuo informacijos šaltinių sudėtingumo. Neišreikštieji apribojimai paslėpti programos procedūrinėse dalyse, sąsajos procedūrose, languose, formose, ataskaitose, trigeriuose ir išsaugotose procedūrose. Kad atlikti vienos programos atvirkštinę inžineriją, paprastai reikia išanalizuoti daugiau nei vieną galimą šaltinį, ir kiekvienu atveju apribojimas gali būti išreikštas daugiau nei vienu būdu.

Kad išgauti prarastus apribojimus, reikia išanalizuoti ne tik užkoduotą sistemos dalį. Kita sistemos dalis apima failų turinį (duomenis), egzistuojančią dokumentaciją, eksperimentus (programos vykdymą), vartotojų ir programuotojų apklausą, aplinkos elgesį. Aplinkos elgesys yra apribojimai, kuriuos valdo programos aplinka. Pvz. pirkėjų sąrašą pateikia kita programa, kuri patikrina visus apribojimus, todėl analizuojama programa netikrina, ar pirkėjo numeris yra unikalus (identifikatorius). Tokiu atveju neįmanoma atrasti pirkėjo identifikatorių tiriant tik analizuojamą sistemą.

1.7.4. Schemos valymas

Procesas pašalina arba pakeičia visas fizines struktūras loginiais atitikmenimis ir tokiu būdu išbaigtą fizinę schemą transformuoja į išbaigtą loginę schemą. Nuo šiol galima atmesti visas fizines struktūras, nes jos nepateikia jokios informacijos apie DB loginę struktūrą. Fizinės struktūros yra naudingos dėl techninių priežasčių, pvz. DB našumo optimizavimas ar tam tikrų mechanizmų realizavimas dėl DBVS apribojimų. Pagrindinės naudojamos transformacijos:

- Prieigos raktų pašalinimas.
- Rinkinių pašalinimas.

Išbaigta loginė schema apima visas duomenų struktūras ir apribojimus, kurie atrasti atliekant duomenų struktūros išgavimą. Schema turi likti artima dabartinei realizacijai, ji yra programuotojo požiūris į DB. Siekiant atitikti šiuos kriterijus, išgaunant išbaigtą loginę schemą fizinės schemos negalima keisti iš esmės. Pvz. negalima pervadinti elementų, kadangi programuotojas negalės kurti programų, kurios naudoja DB. Perteklius reikia pažymėti, tačiau negalima pašalinti, kadangi keisdamas programą programuotojas užmirš palaikyti perteklinius elementus.

Struktūros, kurios atliekant schemos valymą galima keisti ar pašalinti, priklauso nuo naudojamos DBVS. Reliacinėms DBVS galima pašalinti indeksus, nes programuotojas neprivalo žinoti indeksų, kad galėtų kurti programas, kurios naudoja duomenis.

1.7.5. Neišreikštosios struktūros ir apribojimai

Kai kurios pagrindinės neišreikštosios struktūros ir apribojimai[22]:

- Neprivalomų (null – „jokios reikšmės“) atributų suradimas. Struktūra gali būti deklaruota taip, kad kiekvieno esybės tipo kiekvienas atributas privalo turėti reikšmę. Jei atributas neturi reikšmės, dažniausiai tai reiškia, kad suteikiama ypatinga reikšmė, kuri atitinka null reikšmę. Kadangi nėra standartinio būdo realizuoti šį apribojimą, reikia analizuoti programą ir duomenis. Programuotojas null reikšmės atvaizdavimui skaitmeniniuose atributuose gali naudoti anomalinę atributo semantikos prasme reikšmę. O simboliniuose atributuose – tarpą arba kitą ypatingą simbolį.
- Atributų agregavimo suradimas. Tariamai nepriklausomų atributų seka atsirado iš šaltinio sudėtinio atributo, kuris buvo išskaidytas. Šį sudėtinį atributą reikia atstatyti. Tai įprasta reliacinių DB problema.
- Daugiareikšmių atributų suradimas. Atributas deklaruotas kaip vienareikšmis gali būti sujungtas iš daugiareikšmio atributo reikšmių. Kitas galimas variantas, kai to paties tipo ir ilgio vienareikšmių atributų sąrašas materializuoja daugiareikšmius atributus. Reikia aptikti pasikartojančią struktūrą ir išreikšti daugiareikšmį atributą.
- Daugialypių atributų ir esybių tipų struktūrų suradimas. Toje pačioje atributo arba esybės tipo struktūroje galima išsaugoti įvairių tipų reikšmes.
- Daugiareikšmių atributų identifikatorių suradimas. Struktūriniai esybių tipai dažnai apima sudėtingus daugiareikšmius sudėtinius atributus. Gana dažnas atvejis, kad šios reikšmės turi numanomą identifikatorių.
- Ryšio apribojimų suradimas. Tarp esybių gali egzistuoti ryšiai, kuriuos atvaizduoja ryšio apribojimai, t.y. atributai, kurių reikšmės nurodo į esybės tipą. Įprastą ryšio apribojimo formą (vadinamą standartiniu išoriniu raktu) sudaro vienas ar keletą privalomų atributų. Ji rodo į esybės tipo identifikatorių, abu integralumo apribojimo galai yra apibrėžiami toje pačioje dalykinėje srityje. Liktinės sistemos apima ne tik standartinius išorinius raktus, bet ir daug sudėtingų šablonų, pvz. neprivalomi, rekursiniai ar suskaičiuojami išoriniai raktai.
- Funkcinių priklausomybių suradimas. Reliacinių DB metodologijos rekomenduoja atlikti normalizaciją, todėl idealiose programose neturėtų būti funkcinių priklausomybių. Kad turėtų didesnį našumą, dauguma tikrų DB apima nenormalizuotas struktūras. Funkcinės priklausomybės trumpina priėjimo laiką ir

mažina skaičiavimus, tačiau padidina reikalingos saugojimo vietos apimtis ir padidina valdymo taisyklių, kurios palaiko duomenų nuoseklumą, sudėtingumą. Perteklius yra speciali funkcinių priklausomybių rūšis, kur funkcija yra tapatybė. Funkcinės priklausomybės paprastai naudojamos kartu su ryšio apribojimais. Kad žinotų, kuris esybės tipas yra priklausomybės šaltinis, programa tiria ryšio apribojimą. Jei išanalizavus programos fragmentą, aptinkama priklausomybė tarp dviejų atributų, tai dar nebūtinai reiškia, kad duomenų priklausomybė visuomet patvirtinta. Duomenų priklausomybė gali būti teisinga tam tikru momentu, bet ne visą laiką. Tokiu atveju įvardinama veiklos taisyklė, kurią reikia išsaugoti. Svarbu atskirti duomenų priklausomybes ir veiklos taisykles, nes atributus, kurie priklausomi nuo duomenų, reikia pašalinti atliekant konceptualizavimą, o atributų, kurie sudaro veiklos taisyklių dalį, pašalinti negalima.

- Egzistavimo apribojimų suradimas. Atributų ir/arba vaidmenų rinkiniai gali bendrai egzistuoti, t.y. kiekvienam esybių tipui jie arba visi turi reikšmę, arba visi yra nuliniai. Yra ir kitų panašių apribojimų, pvz. išskirtinis (tik vienas atributas gali nebūti nulinis). Šie apribojimai vieninteliai gali padėti atskleisti įterptų agregatų arba potipio realizacijas.
- Išvardintų reikšmių srities suradimas. Daugelis atributų privalo naudoti iš anksto nustatytos reikšmių aibės reikšmes.
- Reikšmių srities apribojimų suradimas. Dažnai DBVS deklaruojamos duomenų struktūrų reikšmių sritis yra labai skurdi. Leidžiamoms reikšmėms gana dažnai taikomi griežti apribojimai.
- Prasmingų vardų suradimas. Kai kurie programavimo metodai ar techniniai apribojimai nurodo naudoti prasmingus vardus. Iš kitos pusės, tam tikros programos neišvystė jokių metodų, todėl įvardinimas skurdus ir prieštaringas. Išgaunant duomenų struktūrą, objektų negalima pervardinti, kitu atveju išbaigta loginė schema neatvaizduos dabartinio realizavimo. Jei randami prasmingesni vardai, juos reikia pasižymėti, kad konceptualizuojant duomenų struktūrą juos būtų galima pervadinti.
- Esybių apjungimo suradimas. Tam tikrais atvejais realizuojant fizinę schemą labai panašios arba identiškos struktūros tačiau skirtingos semantikos esybės gali būti apjungtos į vieną ir realizuotos kaip viena lentelė. Šitokia neišreikštoji struktūra ypač plačia naudojama, kai esybių kiekis yra ribojamas techninėmis sąlygomis ir ši riba yra

sąlyginai nedidelė. Pvz.: DOS operacinėje sistemoje yra nustatoma riba kiek failų gali būti atidaryta vienu metu. Taipogi duomenų bazių talpyklos paslaugų tiekėjai tam tikram paslaugų planui gali taikyti apribojimus lentelių kiekiui duomenų bazėje.

1.7.6. Duomenų struktūros išgavimo informacijos šaltiniai

Siekiant atrasti neišreikštąsias struktūras ir apribojimus, reikia analizuoti daugiau nei vieną informacijos šaltinį. Reikia remtis visais galimais informacijos šaltiniais, pvz. programomis, duomenimis, vartotojo sąsajos procedūriniais fragmentais, ekrano ir ataskaitos išdėstymu, bendrais DBVS kodo fragmentais, egzistuojančia dokumentacija, apklausomis, srities žiniomis, veikimo aplinkos žiniomis ir panašiai [20].

Reikia išanalizuoti keletą šaltinių, nes nė vienas iš jų nenurodo visų apribojimų visas žymes. Pvz. tam tikri apribojimai nerealizuoti programoje, nes juos patvirtina tam tikros aplinkos savybės (įėjimo duomenys visuomet teisingi, juos pateikia kita visiškai patikima programa). Duomenų analizė nepateikia visų apribojimų, nes gali būti tam tikrų klaidingų duomenų. Tiriant duomenis galima rasti netikrų apribojimų, nes duomenų aibė per maža.

Pagrindiniai šaltiniai:

- DBVS-DDL (schemos ir rodiniai). DB deklaravimo sakiniai, kurie nustato išreikštas struktūras ir apribojimus. DB gali apimti tam tikro procedūrinio kodo fragmentus, trigerius, tikrinimo ar saugomas procedūras, kuriuos reikia išanalizuoti, kad atrasti tam tikrus netiesioginius apribojimus.
- Duomenų žodynas/fizinė schema. Duomenų žodynas aprašo aktualias DB struktūras ir apribojimus. Paprastai DBVS pati atnaujina duomenų žodyną ir jis yra vienas iš naujausių informacijos šaltinių. Kaip ir DBVS-DDL sakiniai, jis apima išreikštas duomenų struktūras, taip pat tam tikrą procedūrinį kodą.
- Bendri DBVS kodo fragmentai. Šiuolaikinės DB gali naudoti kodo dalis, kurios stebi DB elgesį, pvz.: tikrinimo/teigimo predikatai, trigeriai ir išsaugotos procedūros. Bendru atveju šie kodo fragmentai patikrina duomenų struktūras ir integralumo apribojimus. Kaip ir bet kokį kodą, juos žymiai sunkiau analizuoti, kadangi nėra standartinio būdo integralumo apribojimams realizuoti. DBVS kodo fragmentai realizuoja neišreikštus apribojimus.

- Programos išeities kodas. Didžioji tiesiogiai neišreikštų duomenų struktūrų ir apribojimų dalis pateiktos programos procedūrinėse dalyse. Dėl to vienas iš svarbiausių informacijos šaltinių yra programos kodas.
- Ekrano ir ataskaitos išdėstymas. Ekrano formą arba struktūrinę ataskaitą galima traktuoti kaip išvestinius duomenų rodinius. Išėjimo duomenų išdėstymas su pavadinimais ir komentarais gali suteikti esminės informacijos apie duomenis.
- Einamoji dokumentacija. Tam tikri atvirkštinės inžinerijos projektai apima įvairias dokumentacijos rūšis. Nors tokie dokumentai dažnai yra daliniai, pasenę ar netgi klaidingi, jie gali pateikti naudingos informacijos. Programos komentarai taip pat gali būti turtingas informacijos šaltinis. Didžioji DBVS dalis leidžia administratoriams pridėti trumpus kiekvienos schemos komentarus.
- Išoriniai duomenų žodynai ir CASE saugyklos. Trečios šalies duomenų žodynų sistemos leidžia duomenų administratoriams įrašyti ir palaikyti svarbiausių organizacijos informacijos išteklių, pvz. failų ir DB struktūrų, aprašymus. Gali pateikti neformalų, bet labai naudingą duomenų aprašymą, kuris leidžia geriau perprasti semantiką. Pagrindinė šių šaltinių problema, kad jie neturi abipusio ryšio su DB, todėl gali apimti nepilną, pasenusią ar klaidingą informaciją. Tas pats ir su CASE įrankiais, kurie gali aprašyti DB struktūras skirtingais abstrakcijos lygiais. Nors šie įrankiai gali generuoti DB aprašymo kodą, paprastai juos sunku panaudoti tiesioginiam DB struktūrų modifikavimui.
- Srities žinios. Neįmanoma pradėti atvirkštinės inžinerijos projektą, neturint jokių taikomosios srities žinių. Turint pradinį tikslų ir pagrindinių programos koncepcijų modelį, egzistuojančią sistemą galima traktuoti kaip šio modelio realizaciją. Šį pradinį modelį reikia išgryninti ir patvirtinti. Apklausus dabartinius ar buvusius vartotojus, vystytojus ar srities žinių ekspertus galima sukurti pirmą srities modelį ar patvirtinti išgautą modelį.
- Duomenys. Gali atskleisti reguliarius šablonus, unikalumo ar įjungimo savybes, kuriuos galima panaudoti struktūrinėms hipotezėms patvirtinti ar paneigti. Galima rasti žymes, kurios nurodo identifikatorius, ryšio apribojimus, atributų išskaidymą, neprivalomus atributus, funkcines priklausomybes, egzistavimo apribojimus, ar kurios apriboja atributo reikšmių sritį.

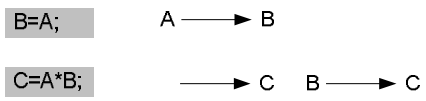
- Ne DB šaltiniai. Mažus duomenų kiekius gali realizuoti bendros paskirties programinė įranga, pvz. skaičiuoklė ir teksto redaktorius. Be to, pusiau struktūriniai dokumentai nagrinėjami kaip sudėtingi duomenų šaltiniai, kuriems taip pat reikia atlikti atvirkštinę inžineriją.
- Programos vykdymas. Dinaminis su duomenimis dirbančios programos elgesys nurodo reikalavimus, kuriuos turi atitikti duomenys, ir ryšius tarp saugomų duomenų. Užpildytų formų ir ataskaitų analizė kartu su duomenų analize leidžia aptikti duomenų struktūras ir savybes.
- DBVS žurnalai. Kai kurios DBVS į žurnalą įrašo visus duomenų priėjimo ir užklausų atlikimo atvejus. Žurnalų analizė leidžia nustatyti naudojamas užklausas.
- Aplinkos savybės. Aplinkos savybės (pvz. DBVS), vystymo įrankiai, vystymo kalba, programavimo principai, naudojama aparatūra gali išreikšti tam tikrus nefunkcinius reikalavimus. Šie reikalavimai gali įtakoti tam tikrų apribojimų patvirtinimo būdą ar netgi juos atmesti.
- Programos istorija. Kokie analitikai sukūrė ir palaikė programą, kokios skirtingos DBVS ir programavimo kalbos naudotos? Ši informacija gali paaiškinti, kodėl apribojimams ir duomenų struktūroms buvo taikomos tam tikros metodikos.
- Organizacijos praktika. Kai kurios organizacijos turi vidinę metodologiją ir įpročius. Žinios apie juos gali palengvinti duomenų struktūros išgavimą. Pvz. visi identifikatorių pavadinimai prasideda simboliais „*id*“ ar kiekvienos procedūros pradžioje pateikiamas komentaras, aprašantis procedūros uždavinius.

1.7.7. Duomenų struktūros išgavimo metodikos

Nors egzistuoja gana didelis neišreikštų apribojimų ir informacijos šaltinių rinkinys, tačiau analizės metodikų rinkinys pakankamai mažas [21]. Toliau aprašomos kai kurios iš jų.

- Duomenų srautų analizė. Ištyrus duomenų reikšmių srautus, galima nustatyti struktūrinius ar tikslinius panašumus tarp kintamųjų. Pvz. jei kintamajam *B*, kurio struktūra *Sb*, priskiriama kintamojo *A*, kurio struktūra *Sa*, reikšmė, ir jei *Sb* yra tikslesnis nei *Sa* (*Sb* turi smulkesnį suskaidymą), tuomet kintamajam *A* galima suteikti *Sb* struktūrą. Srauto sąvoka naudojama platesne reikšme: jei du kintamieji priklauso tam pačiam duomenų srauto keliui, tam tikru laiku ir esant tam tikroms apibrėžtoms

aplinkybėms, jų reikšmės gali sutapti arba vienas iš jų yra tiesioginė kito funkcija. Žemiau esančiame 1.4 paveiksle pateiktas paprasto duomenų srauto pavyzdys.



1.4 pav. Paprasto duomenų srauto pavyzdys

Galima naudoti sudėtingesnį ar ne tokį griežtą ryšį, pvz. `if (A==B)` ir `C=A*B`. Tokie šablonai nenurodo lygybės tarp *A* ir *B* reikšmių, greičiau nusako tam tikros rūšies panašumus. Priklausomybė gali reikšti, kad *A* ir *B* turi suderinamas reikšmių sritis. Svarbus ne tik tiesioginis ryšys tarp kintamųjų, bet ir pereinamas ryšys. Pvz. `B=A` ir `C=B` nurodo ryšį tarp *A* ir *B* ir tarp *B* ir *C*, taip pat ir pereinamą ryšį tarp *A* ir *C*.

- Programavimo šablonai. Drausmingi programuotojai spręsdami panašias problemas naudoja panašius standartinius šablonus. Apibrėžtai problemai nustačius šabloną (vadinamą programavimo klišę), šio šablonų galima ieškoti programoje ar kitos rūšies procedūriniuose fragmentuose, kad surasti vietas, kur sprendžiamos tokios pačios rūšies problemos [24], [40], [42], [45].
- Programos dalinimas. Labai efektyvi metodika, kuri didelę programą padalina pagal nurodytą kriterijų [47]. Programai *P*, programos taškui *p* (pvz. komanda) ir objektui *V* (kintamasis ar įrašas) atgalinis programos *P* padalinimas pagal dalinimo kriterijus <*p, V*> yra visų *P* sakinių, kurie susiję su *V* būsena *p* taške, rinkinys. Kitais žodžiais tariant, vykdant *P* ir vykdant padalinimą, gaunama ta pati *V* reikšmė, esant bet kokioms išorinėms vykdymo sąlygoms.
- Pavadinimų analizė. Patyrę programuotojai kruopščiai parenka esybių tipų, atributų ir kintamųjų pavadinimus, kad palengvintų programos palaikymą ir vystymą. Jie suteikia reikšmingus vardus, kurie nurodo objektų semantiką ir funkcijas. Objektų pavadinimų analizė gali suteikti labai naudingų užuominų apie semantiką ir duomenų struktūrą. Be to, ši analizė gali aptikti sinonimus (kelis to paties objekto pavadinimus) ir homonimus (tuos pačius skirtingų objektų pavadinimus.
- Fizinė struktūra. Kai kurios fizinės struktūros (adreso išlyginimas, esybės tipo atšakos, neįprastai ilgi atributai, priėjimo raktai ir pan.) gali teikti užuominas apie logines struktūras ir apribojimus.

- Langų ir ataskaitų analizė. Langai ir ataskaitos skirti duomenims pateikti ir tam tikrais atvejais juos modifikuoti. Jie yra duomenų rodiniai, todėl jų struktūra nurodo svarbias siunčiamų duomenų struktūros ir semantikos užuominas. Ištyrus langus ir ataskaitas galima išvesti trijų rūšių informaciją:
    - Erdvinius ryšius tarp duomenų laukų. Laukų išdėstymo būdas ekrane gali nurodyti netiesioginius ryšius.
    - Lange nurodyti pavadinimai ir komentarai. Jie pateikia kiekvieno lango lauko reikšmę, vaidmenį, prasmingą pavadinimą ir apribojimus.
    - Atmesti atributai. Jei atributas nepateiktas ekrane, tuomet:
        - Atributas pasenęs ir daugiau nebenaudojamas.
        - Atributas gali būti laisvai pasirenkamas ir šiame kontekste neturi reikšmės.
        - Atributas perteklinis kartu su kitu jau pateiktu formoje.
        - Gali reikšti, kad atributas nurodo informaciją, kurią galima suteikti pagal kontekstą, pvz. apie užsakymo pirkėją.

Lango išdėstymą aukščiau pateiktu būdu galima ištirti kaip atskirą komponentą. Taip pat galima išanalizuoti šaltinio/tikslo programos duomenų struktūras.

Duomenų ataskaitos nagrinėjamos ir kaip duomenų struktūros, ir kaip nuolatinių duomenų rodiniai. Pirmas aspektas panašus į lango išdėstymą: ataskaita yra hierarchinė duomenų struktūra, kuri išreiškia duomenų sąryšius. Antras aspektas susijęs su duomenų analizės euristika.

- Duomenų analizė. Analizuojant DB turinį, galima nustatyti savybes arba palaikyti, įrodyti ar atmesti hipotezes.
- Programos vykdymo analizė. Analizuoja programos reakcijas į pasirinktą dirgiklį, pvz. įėjimo duomenų ir užklausų atnaujinimo priėmimas bei atmetimas. Veikianti programa taip pat naudoja langus ir ataskaitas, todėl ši analizė stipriai susijusi su langų ir ataskaitų analize.
- Dokumentacijos analizė. Jei vis dar egzistuoja ir galima remtis, dokumentacija yra pirmas naudojamas informacijos šaltinis. Paprastai dokumentacijos turi apimti lentelių, atributų, trigerių, procedūrų ir pan. aprašymus. Tačiau prieš ją naudojant, reikia įvertinti dokumentacijos kokybę. Ar ji atnaujinta? Ar ji aprašo dabartinę sistemos versiją? Kokius formalumus ji naudoja?

- Schemos analizė. Išanalizavus darbinę schemą, t.y. ištobulintą schemą, galima nustatyti, kuriuos apribojimus vis dar reikia surasti. Pvz. jei esybės tipas ryšio apribojimais ar ryšių tipais nesujungtas su likusia schema, galima spėti, kad trūksta ryšio apribojimo, kuris apima šį esybės tipą kaip nuorodą arba šaltinį. Schemos analizę pakankamai lengva atlikti, nes schema yra DB struktūros abstrakcija, o CASE įrankis, kuris pateikia analizės įrankius, saugo ją.

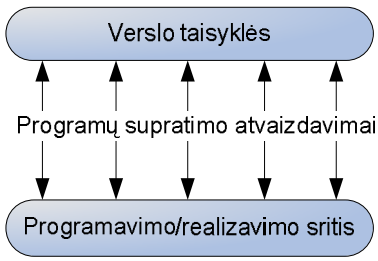
1.8. Programų supratimas ADI

Iš pirmo žvilgsnio keistokai atrodo programų supratimo metodologijos panaudojimas atvirkštinei duomenų bazių inžinerijai atlikti. ADI išgauna duomenų struktūrą, kuri daugiau ar mažiau priklausoma nuo programos. Išeities kodas yra vienas iš tiksliausių ir aktualiausių informacijos išteklių netiesioginiams apribojimams atstatyti. Tačiau išeities kodo analizė yra sudėtinga ir brangi veikla.

Šiame skyriuje trumpai apžvelgsime programų supratimo metodologijos (-ų) panaudojimą ADI atlikti. Pristatysime programų supratimo metodus.

1.8.1. Programų supratimas

Programų supratimas yra žinių apie kompiuterinę programą įgijimo procesas [30], [34], [44]. Šios žinios leidžia atlikti atvirkštinę inžineriją, dokumentavimą, klaidų taisymą, programų pagerinimą, pakartotinį panaudojimą ir kt. veiklas. Programų supratimas yra ne tik kodo supratimas, bet ir atvaizdavimas tarp programos ir dalykinės sričių (1.5 pav.).



1.5 pav. Programų supratimas

Nors supratimo procesą ir siekiama automatizuoti, tačiau reikalingi tokie dideli žinių ir analitiniai pajėgumai, kad didžioji programų supratimo dalis atliekama rankiniu būdu. Programų supratimas yra sudėtingas procesas, kadangi jis privalo apjungti skirtingas konceptualias sritis. Svarbiausi egzistuojantys atotrūkiai [44]:

1. Taikymo sritis ir programa. Programos yra tam tikros taikymo srities probleminių situacijų sprendimai. Programų supratimo užduotis yra atstatyti taikomosios srities atvaizdavimus į programą.
2. Fizinės mašinos ir abstraktūs aprašymai. Kompiuterinės programos yra neįtikėtinai detalios. Viena iš užduočių yra nuspręsti, kurios sąvokos svarbiausios (abstrakcijos procesas) iš visų programos detalių.
3. Susiję modeliai ir nesusiję artefaktai. Atliekant projektavimą, programa sudaroma kaip susijusių detalių rinkinys. Programos palaikymas, pvz. klaidų taisymas ar naujų funkcijų realizavimas, gali sugadinti pradinę programos struktūrą. Reikia aptikti aukšto lygio programos struktūrą, atsižvelgiant į galimą pradinės programos paskirties pasikeitimą.
4. Hierarchinė programos struktūra ir asociatyvus žmogaus pažinimas. Kompiuterinės programos yra ypač formalios. Jos atitinka griežtas taisykles, kurios apriboja idėjų išraišką ir valdo tų idėjų vykdymą. Žmogaus pažinimas veikia asociatyviai. Pažinimo procesą valdo iš taikomosios srities ir programavimo žinių kilę lūkesčiai. Programą reikia suprasti iki tokio lygio, kad būtų galima atkurti teisingas aukšto lygio dalis iš aiškių žemesnio lygio programos detalių.

Didžiąją programų analizės dalį šiuo metu atlieka žmonės. Siekiant suprasti programą, galimi trijų rūšių veiksmai [14]:

1. Skaityti apie programą: analitikas gali skaityti (arba analizuoti) bet kokios rūšies prieinamą dokumentaciją apie programą. Pagrindinė problema – tokia dokumentacija ne visuomet egzistuoja, o kai ji yra, tai nebūtinai nauja, teisinga ar gerai parašyta.
2. Peržiūrėti programą: analitikas gali peržiūrėti (arba analizuoti) kodą pats. Paprastai kodas yra pirminis informacijos šaltinis, kadangi tai vienintelis tikrai tikslus sistemos atvaizdavimas.
3. Vykdyti programą: įdomus informacijos apie programą šaltinis yra jos vykdymas siekiant analizuoti (sekant tam tikrus duomenis) programos darbą su realiais duomenims.

Kadangi programinės įrangos inžinierių produktyvumas ženkliai skiriasi, tai sėkmingai taikomos specialistų strategijos labai domina programų supratimo metodologijos, įrankių ir metodikų kūrėjus. Šie įrankiai pradeda nuo tiesioginės teksto analizės ir pereina į programų vykdymo dinaminę analizę. Pagrindinės programų analizės metodikos yra šios:

1. Tekstinė analizė. Vienas iš paprasčiausių būdų suprasti programą yra rankinis išeities kodo peržiūrėjimas arba reikalingos tekstinės eilutės paieška.

2. Sintaksinė analizė. Sintaksinę analizę atlieka nagrinėtojas, kuris padalina programą į išraiškas ir sakinius. Nagrinėtojo rezultatai saugomi struktūroje, kuri vadinama abstrakčios sintaksės medžiu. Abstrakčios sintaksės medis yra sudėtingiausių programų analizės įrankių pagrindas. Kadangi abstrakčios sintaksės medis yra medžio struktūra, tai jį galima pereiti arba atlikti jo užklausas.
3. Valdymo srautų analizė. Valdymo srautų analizė turi dvi formas. Vidinė procedūros analizė nustato kokia tvarka vykdomi programos sakiniai (sekos, sąlygos, kilpos ir pan.). Tarp procedūrinė analizė nustato iškvietimų ryšius tarp programos modulių, pvz. iškvietimų grafas.
4. Duomenų srautų analizė. Duomenų srautų analizė yra kintamųjų reikšmių srauto tarp programos sakinių analizė. Siekiant įvertinti programos duomenų srautus, reikia žinoti kiekvieno sakinio nustatomus ir nurodomus kintamuosius. Sakinys nustato kintamąjį, kai sakinys pakeičia kintamojo reikšmę (pvz. priskyrimas). Sakinys nurodo kintamąjį, kai sakinys naudoja kintamojo reikšmę (pvz. sąlygos sakinio kintamasis). Duomenų srautų analizė sprendžia su nurodymų tėkme susijusius programos klausimus.
5. Padalinimas. Programos padalinimas atsižvelgiant į programos tašką p ir kintamąjį v susideda iš visų programos kintamųjų ir predikatų, kurie gali įtakoti v reikšmę p taške. Šią koncepciją pirmas aptarė Veizeris [47].
6. Programavimo šablonų atpažinimas. Šių šablonų ieškoma programos išeities kode. Programavimo šablono pavyzdys yra kilpų šablonas tiesinei paieškai atlikti.
7. Abstraktus interpretavimas. Pagrindinė abstraktaus interpretavimo idėja yra savybių priartinimas panaudojant abstrakčią sritį vietoje tikros skaičiavimų srities. Tokiu būdu, visai programai galima suteikti apytikslę reikšmę, išgaunant tik reikalingas savybes, o į nereikšmingas detales neatsižvelgiant.
8. Dinaminė analizė. Aukščiau pateiktos analizės metodikos yra statinės, t.y. jos atliekamos naudojant programos išeities kodą. Supratimo procesą įmanoma pagerinti sistemiškai vykdant programą. Šis procesas vadinamas dinamine analize.

Tokie palaikantys mechanizmai gali valdyti programos supratimo sudėtingumą padėdami išgauti aukšto lygio informaciją iš žemo lygio kodo. Šie palaikantys mechanizmai išvaduoja iš nuobodžių, rankinių ir į klaidas linkusių užduočių, pvz. kodo peržiūrėjimas, ieškojimas ir šablonų lyginimas.

1.8.2. Programų supratimas atvirkštinės duomenų inžinerijos kontekste

ADI lengviau atlikti nei procedūrinės dalies atvirkštinę inžineriją. Duomenų bazės supratimas palengvina visos programos supratimą. Kai kurie autoriai atlikdami ADI naudoja DDL, fizinę schemą arba pačius duomenis. Šis metodas tinka, jei DBVS yra pakankamai veiksminga visiems apribojimams išreikšti, ir vystydamas programą programuotojas panaudojo visas DBVS išraiškos priemones. Tokiu atveju visi apribojimai aiškiai deklaruoti ir nėra jokių netiesioginių apribojimų. Tokias sąlygas galima patenkinti kai kurioms šiuolaikinėms gerai suprojektuotoms duomenų bazėms. Bet jų neįmanoma užtikrinti bendru atveju, o ypač liktinėse sistemose.

Liktinė DBVS neturi gausaus apribojimų rinkinio, todėl programuotojas sudėtingus apribojimus privalo išreikšti kaip išorinius apribojimus, duomenų priklausomybes ir pan. Taigi, programuotojas šiuos apribojimus realizuoja kaip netiesioginius apribojimus (ne deklaratyvia struktūra). Net jei DBVS leidžia tiesiogiai deklaruoti apribojimus ir duomenų struktūras, tačiau kartais jie vis tiek ne deklaruojami, o realizuojami netiesiogiai (struktūros paslėpimas). Egzistuoja įvairios priežastys, kodėl apribojimai nedeklaruojami tiesiogiai DBVS: pakartotinis panaudojimas, bendrumas, paprastumas, efektyvumas, prasti programavimo įgūdžiai, šių apribojimų nepalaikymas ankstesnėse DBVS versijose, netvarka, kuri kyla dėl pernelyg ilgo palaikymo [46].

Visus netiesioginius apribojimus galima atkurti vykdant duomenų struktūros išgavimą. Šie apribojimai tampa dar tikslesni atlikus schemos išgryninimą. Neišgavus netiesioginių apribojimų ADI gali tik pateikti kitą (grafinį) fizinės schemos vaizdą. ADI tampa reikšminga, jei ji tiek pagerina fizinės schemos semantiką, kad būtų galima pateikti konceptualią schemą.

Visas ADI procesas, išskyrus schemos išgryninimą, yra gerai žinomas. DDL analizė nagrinėta nuo IX dešimtmečio pradžios, todėl egzistuoja daug komercinių įrankių jai atlikti. Duomenų struktūros konceptualizavimas dėstomas mokyklose ir universitetuose, jį palaiko nemažai komercinių įrankių. Tačiau egzistuoja labai mažai tyrimų, sprendžiančių schemos išgryninimo klausimus. Vienintelė kai kuriose metodologijose siūloma netiesioginių apribojimų aptikimo metodika yra analitiko srities žinios arba tam tikros žinios apie programą. Deja, jos nepataria kaip šias žinias išgauti.

Didžioji realių projektų išgryninimo proceso dalis remiasi analitiko žiniomis ir didžiuliu rankiniu darbu.

Schemos išgryninimo procesas prideda labai svarbias papildomas loginės schemos reikšmes. Nuo 1992 m. daugelis autorių suvokė, kad procedūrinė programos dalis yra pagrindinis informacijos šaltinis duomenų struktūroms išgauti [4], [22], [28], ir kad tam tikrų programų aspektų supratimas

yra labai svarbus siekiant visiškai suprasti duomenų struktūrą. Didžioji tiesiogiai nedeklaruotų duomenų struktūrų ir apribojimų dalis yra užkoduotos procedūrinėje programos dalyje. Kodas yra vienintelis tikrai tikslus sistemos atvaizdavimas.

Duomenų bazės duomenys yra programos, kuri atnaujina duomenų bazę, vykdymo rezultatas. Todėl visus netiesioginius apribojimus galima išvesti iš programos veiksmų atnaujinant duomenų bazę. Jei duomenų bazė atitinka tam tikrus apribojimus prieš programos vykdymą ir vis dar atitinka tuos apribojimus po vykdymo, tada programa turi patvirtinti, kad modifikuoti duomenys nepažeidžia apribojimų. Taigi, programa turi patikrinti visus apribojimus prieš keisdama duomenis. Kai kuriuos apribojimus galima atrasti analizuojant programas, kurios tik prieina prie duomenų, tačiau nekeičia jų. Kai programa nuskaito duomenis, kad atspausdintų ataskaitą arba pateiktų rezultatus ekrane, ji naudoja tam tikrus duomenų bazės apribojimus. Pvz. jei egzistuoja ryšio apribojimas tarp esybės tipo „*Klientas*“ ir esybės tipo „*Užsakymas*“, tai spausdinant ataskaitą šis ryšio apribojimas bus naudojamas klientui atrasti.

Programos išeities kodas yra tikslus ir aktualus informacijos šaltinis. Programos yra vienintelis būdas vartotojams prieiti prie duomenų, taigi visus reikalingus apribojimus reikia realizuoti kode. Programavimo kalbos yra labai tikslios ir apibrėžtos, todėl yra tik viena atliekamų fragmento veiksmų interpretacija. Analizuojant kodą gautos žinios yra pakankamai patikimos. Kadangi išeities kodas naudojamas programai generuoti, tai jis yra aktualus informacijos šaltinis.

Išeities kodo analizė yra sudėtinga ir brangi užduotis. Ji sudėtinga, nes programos parašytos naudojant liktines kalbas, kurias analitikas turi įvaldyti. Analitikas privalo turėti nuodugnias kalbos žinias, kad suprastų kitų programuotojų parašytas programas. Analitikas taip pat privalo gerai suprasti tiesioginės inžinerijos procesą, kad suprastų kitų programuotojų sukurtą kodą. Programos dydis taip pat yra problemų šaltinis. Neretai programos turi kelis šimtus tūkstančių išeities kodo eilučių.

Programos išeities kodas yra labai naudingas informacijos šaltinis, iš kurio atliekant duomenų struktūrų išgavimą galima atkurti daug neišreikštųjų apribojimų. Tačiau siekiant šį informacijos šaltinį panaudoti efektyviai, reikia pritaikyti įvairias programų supratimo metodikas ir įrankius.

Procedūrinio kodo analizė padeda suprasti duomenų semantiką. Veiklos taisyklių supratimui duomenų manipuliavimo algoritmai duoda svarbias duomenų reikšmių užuominas, todėl pagilina dalykinės srities žinias.

1.8.3. Programų supratimo sunkumai

Programos išeities kodo analizė yra sudėtinga ir nuobodi užduotis. Kadangi procedūriškai užkuoduotos duomenų struktūros yra paskleistos dideliuose išeities kodo failų kiekiuose, vadinasi jie gali dubliuotis, be to nėra standartinio metodo struktūroms ar apribojimams aprašyti.

Pvz. yra tik vienas būdas ryšio apribojimams tiesiogiai deklaruoti SQL-DDL (... `foreign key <column> reference <table>`), tai atliekama tik vieną kartą (deklaruojant duomenų bazę), o apribojimas visada patenkinamas. Šį deklaravimą lengva aptikti DDL kode. Tai įrodymas, kad apribojimas egzistuoja ir jį galima pridėti į schemą be jokio patvirtinimo.

Iš kitos pusės, yra daug skirtingų būdų realizuoti netiesioginį ryšio apribojimą. Kiekvienam kartui, kai reikia modifikuoti, įterpti arba pašalinti vieno iš apribojimo esybių tipo atvejus, reikia sukurti apribojimą patikrinantį kodą. Kodas išskaidomas programoje ir gali būti realizuotas skirtingai. Norint patvirtinti apribojimą, reikia patikrinti visus kodo fragmentus, kurie modifikuoja, įterpia arba pašalina esybių tipus.

Reikia surinkti susijusius sakinius, išanalizuoti tūkstančius kodo eilučių ir sekti valdymo srautą (`if`, `while`). Sudėtingumas didėja ir dėl vidinių DBVS užklausų panaudojimo (SQL). Reikia suprasti (nagrinėti) dvi skirtingas kalbas. Kita problema norint suprasti programas su vidinėmis DBVS užklausomis – DBVS schema nedeklaruojama programose. Fizinę schemą reikia išgauti iš DBVS DDL, po to surišti fizinės schemos ir programos kintamuosius.

Kiekvienas programuotojas turi savo asmeninį būdą apribojimams (kintamųjų pavadinimai, komentarų naudojimas, algoritmas, kodo pateikimas) išreikšti, kuris priklauso nuo įgūdžių, programavimo patirties, nuotaikos. Šie veiksniai padidina supratimo sudėtingumą, kadangi pirmiausia reikia nustatyti programuotojo darbo stilių. Pvz. jei vartotojas naudoja kintamąjį „*kliento_vardas*“ dvejopai skirtingai informacijai saugoti (kliento vardas ir produkto vardas), kad sutaupytų vietos atmintyje, tuomet gali būti labai sunku peržiūrėti (suprasti) kodą. Apribojimų kodavimas taip pat priklauso nuo programavimo kalbos, paskirties DBVS, organizacijos taisyklių (vardų susitarimas, komentarų naudojimas, naudojamų DBVS savybės), reikalingo optimizavimo lygio, programos istorijos (palaikymo procesas, migracijos), naudotų įrankių. Pvz. jei ryšio apribojimo šaltinio pavadinimas apima tikslo atributo arba esybės tipo pavadinimą, tuomet ryšio apribojimus galima rasti žymiai lengviau. Bet jei atributai turi bereikšmius vardus, tuomet gali būti labai sudėtinga suprasti ryšio apribojimo patikrinimo kodą.

Tą pačią kodo dalį galima naudoti keliems apribojimams patvirtinti. Pvz.: esybės tipas „*Užsakymas*“ apima daugiareikšmį atributą užsakomiems produktams saugoti, vienu užsakymu

galima užsakyti tik vieną tokį patį produktą. Kodo dalis, kuri patikrina naują į užsakymą pridėtą produktą, privalo patikrinti, kad naujas produktas yra galiojantis, ir kad produktas dar nebuvo įtrauktas į einamąjį užsakymą. Reikia atskirti skirtingus apribojimų patikrinimus.

Jei apribojimai nėra tiesiogiai deklaruoti duomenų bazėje, tuomet kiekvienas modulis (arba funkcija), kuri modifikuoja duomenis, turi patvirtinti juos. Taigi, patikrinimo kodas dubliuojamas, o kiekvienas patikrinimo atvejis gali būti realizuotas skirtingai. Gana dažnai bent vienas patikrinimo kodas nepatvirtina tam tikrų apribojimų kaip kiti. Šį atvejį galima skirtingai interpretuoti. Gali būti, kad apribojimas nesuprastas, todėl reikia surasti kitą interpretaciją, kuri apima visą kodo fragmentą.

1.8.4. ADI programų supratimo metodikos

Programų supratimą vystė programavimo inžinerijos bendruomenės, kad gautų žinių, programai derinti, palaikyti, pagerinti ir pakartotinai panaudoti. Didžioji metodikų dalis buvo išvystyta, kad padėtų suprasti egzistuojančias programas.

ADI procesui nereikia išgauti pilnos programos specifikacijos, jam tereikia tik svarbių požymių, kurie padėtų atrasti neišreiktąsias nuolatinių duomenų struktūras ir apribojimus. Jei tiksliau, tai reikia surasti neišreikštųjų struktūrų ir apribojimų požymius, pvz. atributų išgryninimas ir agregavimas, integralumo apribojimai, tikslūs kardinalumai ir t.t. Programų supratimo metodikos palengvina neišreikštųjų apribojimų atradimą.

Kad atlikti atvirkštinę DB inžineriją, nebūtina suprasti visus programos aspektus, tačiau nuolatinių duomenų panaudojimą būtina atvaizduoti į duomenų bazės schemą. Šis skyrius pristato skirtingas programų supratimo metodikas, kurias galima panaudoti, siekiant geriau suprasti, kaip naudojami nuolatiniai duomenys, ir kuriuos šių duomenų apribojimus programa užtikrina. Siekiant geriau suprasti, kaip valdomi pastovūs duomenys, galima naudoti skirtingas programų supratimo metodikas:

- Sistemos priklausomybių grafo analizė [25]. Sistemos priklausomybių grafas yra programų atvaizdavimas, kurį naudoja programų dalinimas. Sistemos priklausomybių grafas, tai grafas, kur mazgai atitinka programos kintamuosius, o lankai – ryšius (paprastai duomenų srautų) tarp kintamųjų.
- Programų dalinimas. Skaidymo metodika, kuri išgauna tam tikram skaičiavimui aktualius programų sakinius t.y. būtinų ir pakankamų programos eilučių išgavimas, siekiant suprasti duoto sakinio kintamojo reikšmę.
- Šablonų lyginimas. Šablonų paieška išeities kode.

1.9. Duomenų schemos specifikacija

Atvirkštinė duomenų inžinerija (ADI) nagrinėja schemos išgavimą, analizę ir transformavimą. Kaip ir bet toks kitas DB inžinerijos procesas, ji turi remtis modeliu, kuris apima platų duomenų struktūrų rinkinį. Šis modelis privalo suteikti galimybę aprašyti duomenų struktūras skirtingais abstrakcijos lygiais, pradedant nuo fizinės, baigiant konceptualiąja, ir atsižvelgti į skirtingas modeliavimo paradigmas. Atliekant ADI įprasta skirtingas schemos dalis atvaizduoti skirtingais abstrakcijos lygiais. Pasirinktas modelis privalo atvaizduoti tokias situacijas.

DB vystymo metodologijas galima supaprastinti nustatant tris abstrakcijos lygius. Tarp šių lygių paskirstomi įvairūs inžinerijos reikalavimai, pradedant teisingumu, baigiant efektyvumu. Konceptualiajame lygyje projektuotojas pateikia nuo technologijos nepriklausančią informacijos specifikaciją. Konceptuali schema remiasi specialiu formalizmu, pavadintu konceptualusis modelis. Loginiame lygyje informacija išreiškiama į modelį, kuriam egzistuoja technologija. Pvz. reikalinga informacija atvaizduota reliacine arba objektine logine schema. Loginė schema paremta DBVS modelių šeima, o fizinė schema skirta konkrečiai DBVS. Be loginių apribojimų, ji apima technines specifikacijas, kurios valdo duomenų saugyklą, priėjimo mechanizmus, lygiagretumo protokolus ar atstatymo parametrus.

Toliau yra pristatomas bendras modelis atsižvelgiant į tris skirtingus abstrakcijos lygius: konceptualų, loginį ir fizinį.

1.9.1. Konceptuali specifikacija

Konceptualus modelis leidžia grafiškai vaizduoti modelio konstrukcijas, jis plačiai naudojamas ir automatizuotose programų kūrimo priemonėse [38].

Konceptuali schema specifikuoja esybių tipus (arba objektų klases), ryšių tipus ir atributus. Esybės tipas, arba tiesiog esybė, - tai konkrečių arba abstrakčių realaus pasaulio daiktų ar objektų aibė, apie kurią turi būti surinkta ir saugoma informacija. Konkreti interpretacija priklauso nuo schemos abstrakcijos lygio. Esybės tipui atstovauja jos vardas. Objektų aibė turi būti atvaizduojama tik viena esybe. Esybė apibūdinama atributais ir gali turėti vidinių vientisumo apribojimų.

Esybių tipus galima sukurti pagal isa hierarchijas (sukuriant supertipus ir potipius): pagal potipių apribojimus, žyminčius visumą ir suskaidymą, hierarchija gali būti pilnoji arba dalinė ir nesikertančioji arba susikertančioji. Pilnoji hierarchija (T) reiškia, kad supertipą privalo realizuoti bent vienas potipis (potipių aibė pilnoji ir susikertančioji), išskirtinumo hierarchija (D) reiškia, kad supertipą privalo realizuoti ne daugiau nei vienas potipis (potipių aibė dalinė ir nesikertančioji),

suskaidymo hierarchija (P) reiškia, kad supertipą privalo realizuoti tiksliai vienas potipis (potipių aibė pilnoji ir nesikertančioji), laisvoji hierarchija reiškia, kad supertipo realizavimui nėra jokių apribojimų (potipių aibė dalinė ir susikertančioji). Esybės tipas gali paveldėti daugiau nei vieno esybės tipo (supertipo) savybes, tai vadinama daugialypiu paveldėjimu.

Atributas – tai esybės elementas, kuris išreiškia kokybines arba kiekybines jos savybes. Atributai turi savo reikšmes, kuriomis esybės egzemplioriai identifikuojami ir klasifikuojami. Atributai savo reikšmėmis apibūdina esybės būseną tam tikru laiko momentu. Atributai gali būti atominiai arba sudėtiniai. Atominiai atributai pažymi vieną elementarią esybės savybę. Sudėtinis atributas sudarytas iš paprastųjų, t.y. jį galima išskaidyti į bent vieną atributą (atominį arba sudėtinį).

Ryšio tipas, arba tiesiog ryšys,– tai prasminis esybių santykis. Ryšio tipą sudaro esybių tipai su vaidmenimis, o vaidmenys sieja ryšio tipą su esybe. Ryšio tipas su dviem vaidmenimis vadinamas dvinariu, o ryšio tipas su $n>2$ vaidmenimis vadinamas daugianariu (n-nariu). Ryšio tipai taip pat gali turėti atributų. Paprastai viena esybė susieta vienu vaidmeniu su ryšio tipu, tačiau galimas toks variantas, kai kelios esybės tuo pačiu vaidmeniu yra susietos su tuo pačiu ryšio tipu, tokie vaidmenys yra daugialypiai. Vaidmenis apibūdina kardinalumo laipsnis $[i\text{-}j]$, $0\leq i<N$ ir $0<j\leq M$ ir $i\leq j$. Ši vaidmens savybė nurodo, kiek kartų (nuo i iki j) esybės egzempliorius gali būti susietas šiuo vaidmeniu su ryšio tipu.

Esybių tipai ir ryšių tipai gali turėti apribojimus (pvz. identifikatorius), kurie susideda iš atributų ir/arba nuotolinių vaidmenų. Šiuos apribojimus išreiškia grupės sąvoka. Grupę sudaro komponentai, kurie yra atributai, vaidmenys ir/arba kitos grupės. Grupė atvaizduoja struktūrą, priskirtą tėviniam objektui (esybės tipas, ryšio tipas arba sudėtinis atributas). Naudojamas atvaizduoti vaizdinius, pvz. identifikatorius, nesuderinimo ar sambūvio apribojimus:

- Pirminis identifikatorius (*id*). Grupės komponentai sudaro pagrindinį tėvinio objekto identifikatorių. Tėvinis objektas gali turėti tik vieną pirminį *id*, visų jo komponentų reikšmės privalomos.
- Antrinis identifikatorius (*id'*). Grupės komponentai sudaro antrinį tėvinio objekto identifikatorių. Tėvinis objektas gali turėti bet kokį antrinių identifikatorių kiekį.
- Sambūvis (*coex*). Grupės komponentų reikšmės turi būti nustatytos kartu arba visai nenustatytos bet kokiam tėvinio objekto egzemplioriui.
- Išskirtinis (*excl*). Tarp grupės komponentų ne daugiau vienos reikšmės gali būti nustatyta bet kokiam tėvinio objekto egzemplioriui.

- Bent 1 (*at-lst-1*). Tarp grupės komponentų bent viena reikšmė turi būti nustatyta bet kokiam tėvinio objekto egzemplioriui.
- Tiksliai 1 (*exact-1*). Tarp grupės komponentų, viena ir tik viena reikšmė turi būti nustatyta bet kokiam tėvinio objekto egzemplioriui (=*išskirtinis*+*bent 1*).

Apdorojantis modulis yra bet koks dinaminis ar loginis aprašytos sistemos komponentas, kurį galima susieti su schema, esybės tipu ar ryšio tipu. Pvz. apdorojantis modulis gali atvaizduoti išsaugotą procedūrą, programą, trigerį, veiklos taisyklę ar metodą.

1.9.2. Loginė specifikacija

Loginė schema apima duomenų struktūrų apibrėžimus pagal vieną iš įprastų modelių šeimų: reliacinis modelis, tinklinis modelis, hierarchinis modelis, invertuotų failų modelis, standartinių failų modelis, objektinis modelis, objektinis-reliacinis modelis ir t.t.

Loginė specifikacija naudoja tuos pačias bendro modelio struktūras, kaip ir konceptuali specifikacija. Kai kurie loginiai modeliai nenaudoja tam tikrų konceptualios specifikacijos struktūrų. Reliacinis modelis nenaudoja ryšių tipų, sudėtinių atributų, daugiareikšmių atributų, isa hierarchijos. Šias sąvokas reikia išreikšti lygiaverčiomis modelio struktūromis. Kai kurių konceptualios schemos dalių negalima versti į loginės schemos atitikmenis, nes loginis modelis turi daugiau apribojimų. Šiuos apribojimus reikia pateikti kaip teksto išnašas po schema [38].

Naujos loginio modelio struktūros:

- Ryšio apribojimas (*ref*). Tarp grupinis apribojimas, kur pradinė grupė yra nuorodos grupė, o tikslo grupė yra nurodyta grupė. Nurodyta grupė turi būti identifikatorius (pirminis arba antrinis). Kiekvienas pirmos grupės atvejis privalo būti iš antros grupės panaudotų reikšmių aibės.
- Įjungimo apribojimas (*incl*). Tarp grupinis apribojimas, kai kiekvienas pirmos grupės atvejis privalo būti iš antros grupės panaudotų reikšmių aibės. Antra grupė nebūtinai identifikatorius, įjungimo apribojimas yra ryšio apribojimo apibendrinimas.
- Lygybės apribojimas (*equ*). Kaip ir ryšio apribojimas, tačiau dar nurodo įjungimo apribojimą iš antros grupės į pirmą.

1.9.3. Fizinė specifikacija

Fizinė schema išreiškia fizines specifikacijas. Kadangi egzistuota platus nuo konkrečios DBVS priklausančių savybių spektras, nėra lengva pateikti bendrą techninių struktūrų modelį. Atvirkštinės inžinerijos atveju dvi struktūros pateikia esmines struktūrines ar semantines užuominas:

- Rinkinys. Rinkinys yra lentelių vietos, saugojimo srities ar DB vietos saugykla, kur išsaugomi nuolatiniai duomenys.
- Priėjimo raktas (*acc*). Priėjimo raktas yra grupė, kuri atitinka bet kokį kelią, kuris suteikia greitą ir atrankinį priėjimą prie esybių tipų, kurie atitinka apibrėžtą kriterijų, pvz. indeksai.

Fizinė specifikacija prie loginės specifikacijos prideda informaciją, skirtą efektyviai realizuoti DB, naudojant konkrečią DBVS. Fizinė specifikacija turi būdingas savybes, susijusias su sintakse, apribojimais ir našumu.

1.9.4. Reliacinis modelis

Šiame skyriuje pateikiama, kaip bendras modelis atvaizduoja reliacinio modelio duomenų struktūras. Aprašomas vertimas iš reliacinio modelio į bendrą modelį: kaip specifikuojamos duomenų struktūros ir kokie skirtumai tarp loginės ir fizinės schemos. Taip pat nurodoma, kaip specifikuoti skirtingus neišreikštąsias duomenų struktūras ir apribojimus.

Reliacinis modelis nenaudoja ryšių tipo. Iš atributų naudoja tik atominius vienareikšmius privalomus arba neprivalomus atributus, kurie vadinami stulpeliais. Pagrindiniai išreikšti apribojimai: pirminis identifikatorius (vadinamas pirminiu raktu), antrinis identifikatorius (vadinamas unikaliu indeksu) ir ryšio apribojimas (vadinamas išoriniu raktu). Priėjimo raktus galima deklaruoti su dublikatais arba ne (atitinkamai vadinamas indeksu arba unikaliu indeksu). Kiekvieną identifikatorių palaiko priėjimo raktas (indeksas). Esybės tipas vadinamas lentele. Rinkinys priklausomai nuo DBVS vadinamas lentelės vieta, saugojimo sritimi arba DB vieta, gali apimti daugiau nei vieną esybės tipą.

Lentelėje 1.1 pateikti bendro ir reliacinio modelių struktūrų atitikmenys.

1.1 lentelė. Reliacinio modelio struktūrų atitikmenys

Reliacinis modelis	Bendras modelis
Lentelės vieta, saugojimo sritis, DB vieta	Rinkinys
Lentelė	Esybės tipas
Stulpelis	Atominis vienareikšmis atributas (.-1)
Nenulinis stulpelis	Privalomas atributas (1-1)
Nulinis stulpelis	Neprivalomas atributas (0-1)
Pirminis raktas	Pirminis identifikatorius
Indeksas	Priėjimo raktas
Unikalus indeksas	Antrinis identifikatorius ir priėjimo raktas
Išorinis raktas	Ryšio grupė

Reliacinis modelis turi nedidelį duomenų struktūrų ir apribojimų rinkinį. DBVS modelis daugelio struktūrų neišreikia, pvz. nėra ryšių tipo. Joks DBVS modelis nesiūlo mechanizmo dubliavimui nurodyti. DBVS neturi tam tikrų apribojimų ne todėl, kad jų nereikia. Tokiose situacijose programuotojas programoje turi netiesiogiai realizuoti trūkstamus apribojimus.

Jei atliekant ADI nustatomas apribojimas ar duomenų struktūra, kuri nepriklauso DBVS modeliui, tuomet schemoje reikia išreikšti šį apribojimą. Jei apribojimas egzistuoja kitoje DBVS ar konceptualiame modelyje, tuomet reikia naudoti kito modelio notaciją. Pvz. reliacinės DB atributas, kurį galima išskaidyti į kelis atributus (adresą galima išskaidyti į namą, gatvę, miestą), atvaizduojamas kaip sudėtinis atributas.

Būtina apibrėžti metodikas, kad atvaizduoti nestandartinius apribojimus, t.y. apribojimus, kurių DBVS modeliai neatvaizduoja, bet programuotojai naudoja. Kai kurie iš apribojimų:

- Pertekliaus apribojimas (*rd*). Įprasta optimizavimo metodika nukopijuoja tam tikrą informaciją iš vieno įrašo į kitą. Pertekliaus apribojimas pateiktas kaip tarpgrupinis apribojimas, nurodantis, kad pradinės grupės reikšmė yra tikslo grupės kopija. Tikslo grupė neturi ypatingo tipo.
- Duomenų priklausomybės apribojimas (*dd*). Duomenų priklausomybė yra tarpgrupinis apribojimas, kuris nurodo, kad kiekvienas pradinės grupės atvejis priklauso nuo tikslo grupės reikšmės.
- Pasenusi/nenaudojama duomenų struktūra. Atliekant DB palaikymo darbus ir vykstant evoliucijai, gali nutikti, kad kai kurie atributų ar esybių tipai sukurti, tačiau daugiau nebenaudojami. Svarbu pažymėti šias duomenų struktūras kaip nebenaudojamas, kad išvengti tolimesnės analizės, o vėliau pašalinti atliekant palaikymo ar migravimo darbus.
- Darbinės duomenų struktūros. DB randami esybių tipai arba atributai gali neturėti jokio ryšio su programos sritimi, kadangi jie yra darbinės duomenų struktūros. Pvz. vartotojo vardas ir slaptažodis, paskutinis naudotas pirkėjo numeris gali būti saugomi kaip esybės tipai. Tokios duomenų struktūros priklauso nuo programos realizavimo būdo, o ne nuo taikymo srities. Pvz. jei operacinė sistema pasiūlo tam tikrą programos priėjimo kontrolę, nebūtina palaikyti esybės tipo su vartotojo vardu ir slaptažodžiu.

Taip pat yra ir kitų tipų apribojimų, kuriems standartinis atvaizdavimas neapibrėžtas.

1.10. Esamų ADI metodų analizė

Atvirkštinės duomenų inžinerijos sritis yra žinoma ir analizuojama seniai. Pirmieji ADI metodai buvo sukurti 1980 metais ir kuriami ir tobulinami iki šiol. Visi šie metodai siekia išgauti konceptualią schemą iš veikiančios liktinės sistemos. Konceptualią schemą galima išreikšti tam tikru esybių-ryšių modelio variantu arba objektiniu modeliu (ODMG, UML, OMT).

Kiekvienas metodas pristato savas taisykles ir euristiką, pateikia savo rezultatus ir kelia savitus reikalavimu duomenims ir būtinumo sąlygas. Iš reliacinių schemų konceptualius modelius išgaunančių metodų kiekis gerokai viršija kiekį tų metodų, kurie konceptualų modelį išgauna iš kitų schemų. Tam yra keletas priežasčių. Paskutinis kelis dešimtmečius reliacinė teorija gerokai ištobulinta: formalizuota (matematiškai) teorija, sukurta metodologija efektyvioms reliacinėms DB projektuoti, aprašytas konceptualios schemos transformavimas į reliacines struktūras. Toks teorinis pagrindas neegzistuoja tinklo, hierarchiniams modeliams ar paprastiems failams. Reliacinės DB projektavimo mokoma mokyklose, universitetuose ir profesionaliuose DB apmokymo kursuose, taigi reliacinės DB yra geriau suprojektuotos nei kitos DB. Modernios reliacinės DB leidžia įdiegti beveik visas konceptualios schemos struktūras, naudojant išorinius raktus, pirminius raktus, indeksus (unikalius arba ne) ir nulines reikšmes. Apribojimai, kurie nėra išreikšti DDL kalba, realizuojami programos procedūrinėje dalyje, naudojant duomenų manipuliavimo kalbą (DML). Naudojant DML galima išreikšti sudėtingas užklausas, taigi atstatyti neišreikštuosius apribojimus ir struktūras galima tik analizuojant DML fragmentus ir netiriant procedūrinės programos dalies. Kai kuriais palankiais atvejais fizinė DB schema apima visus apribojimus ir struktūras, todėl nereikia analizuoti nei programos išeities kodo, nei duomenų.

Reliacinės DB paprastai veikia moderniuose kompiuteriuose, kurie yra galingi ir turi labai trumpą disko kreipties laiką, dėl ko detalus duomenų struktūros išskaidymas sukelia labai nereikšmingą sistemos produktyvumo sumažėjimą. Didesnioji reliacinės DB kursų dalis pristato normalizavimo koncepciją ir skatina kurti normalizuotas DB. Tačiau liktinės DB, priešingai, yra stipriai optimizuotos, nes jų projektavimo metu kompiuterių techniniai ištekliai buvo brangūs ir labai riboti, kas iš projektuotojo reikalavo taupyti vietą ir mažinti disko kreipties laiką. Programuotojai nebuvo apmokomi DB projektavimo, todėl jie nenaudojo standartizuotų projektavimo metodų.

Šios priežastys ir tyrėjų siekis dirbti ir studijuoti modernias kalbas, paaiškina, kodėl didžioji ADI metodų dalis analizuoja atvirkštinę reliacinių DB inžineriją.

Naudojami įvairūs kriterijai, siekiant klasifikuoti ADI metodus:

- DBVS palaikymas. Daugelis metodų būdingi tam tikram DBVS modeliui.

- Rezultato modelis. Metodai išreiškia konceptualią schemą atsižvelgiant į konceptualų modelį. Šis modelis gali būti esybių ryšių modelio variacija (EER, ERC+) arba tam tikras objektinis modelis (ODMG, UML, OMT).
- Būtinos sąlygos. Kai kurie metodai nustato reikalavimus DB, kurioms taikoma ADI, pvz. schema turi būti trečios normalinės formos, atributų vardai turi sietis (du atributai su tuo pačiu vardu atvaizduoja tą patį daiktą, o du atributai su skirtingais vardais turi skirtingą semantiką), neturi būti duomenų klaidų.
- Srities semantikos perpratimo nuodugnumas. Kai kurie metodai reikalauja, kad fizinė schema apimtų visą semantiką (reikšmingi vardai, pavaizduoti visi identifikatoriai) ir analitikas žinotų sritį, kad galėtų interpretuoti trūkstamą informaciją. Kiti metodai analizuoja papildomus informacijos šaltinius, kad išgautų fizinėje schemoje trūkstamą semantiką.
- Naudojama euristika ir metodikos. Euristikos ir metodikų aprašymas, kurias naudoja atvirkštinės inžinerijos procesas. Kai kurie autoriai tik paaiškina tam tikrą abstrakčią euristiką, kiti pateikia algoritmus ir įrankius, kurie siūlomi ar jau yra realizuoti.
- Išbaigtumas ir tvirtumas. Liktinės DB yra tikros DB, kurias sukūrė ir palaiko tikri programuotojai. Taigi šios DB retai suprojektuotos atsižvelgiant į knygų taisykles ir metodus. Dėl palaikymo veiklos, DB objektų įvardinimas tampa nenuoseklus, kai kurios duomenų struktūros nebenaudojamos, jos optimizuotos, atsiskleidžia projektavimo klaidos ir pan. Bet kokį atvirkštinės inžinerijos procesą reikia įvertinti dėl šių įprastų savybių. Metodo išbaigtumas pasiekiamas, jei procesas nagrinėja galimas klaidas. Metodo tvirtumas nusako, kaip metodas elgiasi, kai tokia klaida aptinkama.
- Automatizavimas/vartotojo sąveika. Siekiant panaudoti metodą realiam projektui, svarbu žinoti, kokią metodo dalį galima automatizuoti ir kada būtina sąveika su vartotoju.
- Informacijos ištekliai. Yra labai daug informacijos šaltinių, kuriuos galima analizuoti (fizinė schema, programa, duomenys, dokumentacija, vartotojo žinios). Didžioji metodų dalis naudoja tik kai kuriuos iš jų.

Pirmosios tyrinėjimo pastangos apėmė reliacinės schemos transformavimą su žinomais pirminiais raktais ir išoriniais raktais į konceptualią schemą. Šie metodai naudojo duomenis apie lenteles, stulpelių pavadinimus ir pirminius raktus. Jie nenurodė informacijos šaltinių (fizinė schema

duota) ir nustatė griežtus reikalavimus fizinei schemai (trečia normalinė forma, reikšmingi vardai). Kai kurie tyrimai mėgino atstatyti objektinę konceptualią schemą, todėl telkėsi į apibendrinimo hierarchijų atradimą.

Paskutiniu metu išryškėjo tendencija pabrėžti informacijos įgijimo fazę. Tyrinėjami skirtingi informacijos šaltiniai raktams, išoriniams raktams ir įjungimo priklausomybėms atrasti.

1.10.1.Metodų aprašymai

[Hainut – 1981]

Hainuto metodas [21] nagrinėja loginę schemą, kurioje išreikšti visi DB apribojimai. Rezultatas – dvinaris esybių-ryšių modelis.

Straipsnis atvirai neišreiškia atvirkštinės DB inžinerijos metodo. Aprašo grįžtamąsias transformacijas iš dvinario esybių-ryšių modelio į DBVS modelį. Kad paaiškintų pavyzdžius, naudoja reliacinį ir bendrą DBVS modelį. Parodo, kad siūlomos transformacijos yra abipusės, taigi esybių-ryšių modelį galima pakeisti į DBVS modelį, o iš DBVS modelio galima vėl išvesti esybių-ryšių modelį. Formaliai aprašo kiekvieną transformaciją, kurias galima lengvai automatizuoti.

[Dumpala – 1983]

Dumpalos metodas [18] yra vienas ankstyviausių darbų, kur aprašyta, kaip iš reliacinio, tinklinio ir hierarchinio modelio atvaizduoti į konceptualų modelį. Šis metodas nagrinėja trečios normalinės formos loginę schemą kartu su duomenimis apie pirminius raktus ir išorinius raktus. Rezultatas – esybių-ryšių modelis su atributais ir ryšių tipais.

Metodologija pateikia algoritmą loginiam modeliui transformuoti. Algoritmas pagrįstas ryšių ir raktų (pirminių, išorinių) klasifikavimu. Apibrėžiamos skirtingos klasės, bet algoritmas šiam klasifikavimui automatizuoti nepateiktas, vartotojas pats turi atlikti rankiniu būdu.

[Navathe – 1987]

Navato metodas [36] yra pagerinta Dumpalos atvaizdavimo algoritmų versija. Metodas nagrinėja trečios normalinės formos ryšius, o raktiniai atributai, kuriuos naudoja daugiau nei vienas ryšys, privalo turėti tą patį pavadinimą visoje schemoje. Rezultatas – konceptuali schema, išreikšta pagerinta esybių-ryšių modelio versija, pavadinta esybių-kategorijos-ryšių modeliu (ECR), kuris įveda poklasio ir apibendrinimo hierarchijų sąvokas.

Metodologija klasifikuoja ryšius ir atributus. Konceptualią schemą generuoja panašiai kaip Dumpala, išskyrus etapų eiliškumą ir intensyvią sistemos sąveikauja su vartotoju. Esybių tipams, kurie turi tuos pačius pirminius raktus, sukuria supertipus.

[Casanova – 1983]

Kazanovos metodas [9] nagrinėja reliacinę schemą su pirminiais raktais ir išoriniais raktais. Šią schemą turi pateikti vartotojas. Gauta konceptuali schema yra esybių-ryšių modelis be sudėtingų objektų ar apibendrinimo.

Metodas nagrinėja pirminius raktus ir išorinius raktus. Lenteles padalina ir apjungia į esybių tipus, kiekvienas iš jų atvaizduoja vieną konceptualios schemos elementą. Algoritmas, kuris sugeneruoja schemą yra pateiktas kartu su formaliu įrodymu, kad sugeneruota schema yra teisinga.

[Markowitz – 1990]

Markovičiaus metodas [35] tęsia formalų Kazanovos metodą. Metodas apdoroja reliacinę schemą, raktines priklausomybes ir raktais pagrįstas įjungimo priklausomybes, t.y. integralumo apribojimus. Ryšius reikia pateikti Boyce-Codd normaline forma. Metodo rezultatas – išplėstas esybių-ryšių (EER) modelis.

Metodologija apima keturis žingsnius. Pirmas žingsnis reliacinę schemą transformuoja į formą, tinkamą nustatyti EER objekto struktūras. Antras metodologijos žingsnis ištiria transformuojant išgautas reliacines schemas, funkcines priklausomybes ir įjungimo priklausomybes, kad nustatytų, ar jie atitinka konkrečias savybes. Trečias žingsnis kiekvienai įjungimo priklausomybei apibrėžia objekto sąveikų tipą, pvz. silpna esybė ir specializacija. Ketvirtas žingsnis pagal tam tikras atvaizdavimo taisykles sudaro EER schemą, po to šią schemą patikrina.

Metodas labai priklauso nuo pateiktų duomenų. Pagrindinis darbo indėlis yra nepriklausomumas nuo atributų pavadinimų ir atvaizdavimo tarp schemų formalizavimas.

[Davis – 1987]

Deivio metodas [16], [17] nagrinėja trečios normalinės formos reliacinę schemą. Kaip šią schemą išgauti, specifikuota nėra. Rezultatas yra esybių-ryšių modelis be sudėtingų objektų ar apibendrinimo.

Lentelę su vienu atributu kaip raktu ir lentelę su keletą atributų apimančiais raktais (jei visi raktiniai atributai yra arba nei vienas iš jų nėra integralumo apribojimai) verčia į esybių tipus. Lenteles su kabančiais atributais (tik dalis raktinio atributo yra integralumo apribojimas) verčia į

silpnus esybių tipus kartu su priskirtais integralumo apribojimais, kurie nustato, kad silpnos esybės atvejis negali egzistuoti be atitinkamo stiprios esybės tipo, nuo kurio jis priklauso. Lentelę su raktu, kuris yra į esybių tipus paverstų lentelių pirminių raktų sąryšis, verčia į daug su daug ryšio tipą. Valdo ryšių tipus tarp daugiau nei 2 esybių. Jei lentelės, kurią verčia į esybės tipą, raktiniai atributai yra kitos lentelės, kurią verčia į esybės tipą, neraktiniai atributai, tuomet sukuriamas vienas su daug ryšio tipas.

DB-MAIN [Hainut – 1991]

DB-MAIN metodas [20] nagrinėja visą informaciją apie liktinę DB, pvz. DDL, išeities kodą, duomenis, dokumentaciją ir pan. Metodas išgauna dvi schemas: loginę schemą, kurią privalo suprasti programuotojas, kad galėtų pakeisti liktinę DB ir duomenis modifikuojančias programas, ir konceptualią schemą – esybių-ryšių schemą.

Metodas nedaro jokių prielaidų nei dėl liktinės sistemos naudojamos kalbos, nei dėl optimizavimo lygio. Metodą sudaro du pagrindiniai procesai: duomenų struktūros išgavimas ir duomenų struktūros konceptualizavimas. Duomenų struktūros išgavimas analizuoja visus galimus informacijos šaltinius ir pateikia loginę schemą. Procesas susideda iš dviejų dalių: DDL analizės proceso, kuris išgauna visus DDL kode deklaruotus apribojimus, ir schemos išgryninimo proceso, kuris, analizuodamas kitus informacijos šaltinius, išgauna visus kitus apribojimus. Duomenų struktūros konceptualizavimas loginę schemą transformuoja į konceptualiąją schemą. Procesas pašalina schemos optimizavimus, konkrečiai DBVS būdingas struktūras transformuoja į jų konceptualius atitikmenis ir konceptualiai normalizuoja schemą.

Šį metodą palaiko DB-MAIN įrankis.

[Premerlani – 1993]

Premerlanio metodas [39] analizuoja DDL duomenų žodyną ir programos duomenis, analitikas turi žinoti programos semantiką. Jokių prielaidų dėl ryšių nedaroma, metodas susidoroja su projekto optimizavimu ir blogai suprojektuota realizacija. Metodas naudoja OMT notaciją konceptualiajai schemai modeliuoti.

Metodas remiasi ne pirminiais, o pretenduojančiais (angl. Candidate) raktais. Metodas identifikuoja pretenduojančius raktus: analizuoja unikalius indeksus, automatiškai skenuoja duomenis ir išgauna vartotojų žinias. Modernių sistemų DBVS gali pateikti išorinius raktus. Jei jų nėra, tuomet ištiria atitinkančius pavadinimus, sritį bei duomenų tipus ir aptinka išorinius raktus.

Vaizdo jungimo sąlygos apima išorinių raktų ir antrinių indeksų informaciją. Ieško išorinių atributų grupių, analizuoja DB praplėtimą, kad atliktų įjungimo analizę.

Išanalizuoja didelį išeities schemos duomenų struktūrų kiekį ir verčia į rezultato schemą. Klasės, kurios susijusios vienas su vienas ryšiu, nurodo apibendrinimą. Apibendrinimai gali būti realizuoti perkeliant supertipo atributus potipiui. Palaiko didelę transformacijų dalį po vertimo, t.y. apjungia vertikaliai fragmentuotas klases.

Šis metodas pripažįsta, kad ne visi apribojimai išreikšti fizinėje schemoje. Todėl reikia analizuoti kitus informacijos šaltinius, pvz. DB praplėtimą, vartotojo žinias, programas.

[Johanesson – 1994]

Johanesono metodas [29] nagrinėja reliacinę schemą, funkcines priklausomybes ir įjungimo priklausomybes. Analizuoja trečios normalinės formos ryšius. Rezultatas – esybių-ryšių modelis be sudėtingų objektų, bet su apibendrinimu.

Metodas apibrėžia transformacijų aibę, naudojamų padalinti ryšius, kurie atvaizduoja daugiau nei vieną objekto tipą. Apjungia skirtingus tą patį objekto tipą atvaizduojančius ryšius į vieną ryšį. Schemos atvaizdavimo algoritmo principas toks: atvaizduoti kiekvieną ryšį į objekto tipą ir kiekvieną įjungimo priklausomybę į apibendrinimo apribojimą arba ryšio tipą. Parodo, kad sugeneruota konceptuali schema gali atvaizduoti tiek pat informacijos, kiek pradinė reliacinė schema. Metodas pagrįstas nusistovėjusiomis reliacinės DB teorijos sąvokomis. Išbaigtas metodas atvirkštinės inžinerijos žingsnių aprašymo prasme, bet turi trūkumų, nes reikalauja visų raktų ir įjungimo priklausomybių.

[Signore – 1994]

Signorės metodas [45] nagrinėja reliacinę schemą, SQL ir programos išeities kodą. Rezultatas – esybių-ryšių modelis, apibrėžiantis sudėtingus daugiareikšmius atributus ir apibendrinimą.

Metodologija apima tris etapus. Pirmu etapu identifikuoja pirminius raktus. Jei reliacinė schema jų neatvaizduoja, tuomet išgauna iš išeities kodo. Analitikas turi patvirtinti pirminių raktų teisingumą. Antru etapu aptinka sinonimų ir integralumo apribojimų indikatorius. Šių indikatorių ieško išeities kode (programavimo kalba ir SQL užklausos), jei reliacinė schema jų neatvaizduoja, o po to išeities kode patikrina pagal integralumo apribojimus ir pateikia vartotojui patvirtinti. Paskutinis etapas – konceptualizavimas. Naudodamas ankstyvesnių etapų indikatorius, išveda konceptualų modelį.

Pažymėtina, kad metodas pagrįstas įrankiais. Įrankiai adaptuoti, kad susidorotų su neįprastomis realizavimo metodikomis, optimizavimo pasirinkimais, silpna DDL, išeities kodo klaidomis ir pan. Trumpai aprašytas galimas šį metodą palaikantis įrankis.

[Chiang – 1995]

Čiango metodas [10] nagrinėja naudojamas DB. Ryšiai turi būti trečios normalinės formos ir raktiniai atributai turi būti vienodai pavadinti visuose lentelėse, kadangi metodas naudoja raktinių atributų vardus, kad išvestų ryšius tarp lentelių. Jei ne trečios normalinės formos struktūra negali atvaizduoti daugiau nei vieną tipą, tuomet ją reikia rankiniu būdu išskaidyti į detalesnes modeliavimo struktūras. DB apima bet kokius raktinių atributų klaidingus duomenų atvejus. Rezultatas – išplėstas esybių-ryšių modelis su apibendrinimo hierarchija.

Metodologija turi tris pagrindinius žingsnius [11]. Pirmas žingsnis klasifikuoja ryšius ir atributus, priklausomai nuo reliacinės schemos ir jos pirminių raktų. Antras žingsnis suranda įjungimo priklausomybes (integralumo apribojimai). Automatiškai aptinka galimas įjungimo priklausomybes, pagal identifikatorius ir atributų pavadinimus. Tuomet šias įjungimo priklausomybes patikrina užklausomis iš DB. Galiausiai, nustato EER komponentus pagal taisyklių sąrašą.

Metodas skiriasi nuo aukščiau pateiktų, nes jis sprendžia informacijos įgijimo problemą ir nagrinėja įjungimo priklausomybes. Pateikia EER modelį, kuris semantiškai turtingesnis nei reliacinė schema.

Metodas pateikia įrankį Knowledge Extraction System. Įrankis sukurtas Prolog ir C kalbomis ir vykdo Oracle DB užklausas.

[Ramanathan – 1996]

Ramanatano metodas [43] nagrinėja trečios normalinės formos reliacinę schemą kartu su pirminiais raktais ir išoriniais raktais. Rezultatas – schema, naudojanti OMT notaciją.

Metodologija padalinta į tris žingsnius. Pirmas žingsnis nustato lenteles, kurios atitinka objektus-klases. Antras žingsnis nustato ryšius. Apibrėžia tris ryšių tipus: asociacija, apibendrinimas/specializacija ir agregavimas. Paskutinis žingsnis nustato tikslius asociacijų kardinalumus. Beveik visą procesui reikalingą informaciją pateikia pirminiai ir išoriniai raktai. Metodas turi didelį automatizavimo potencialą, bet joks įrankis nepateiktas.

Metodas nurodo kaip susidoroti su ne trečios normalinės formos schema, kada šis denormalizavimas įvyko optimizuojant programą. Pasiūlo, kaip išspręsti horizontalų ir vertikalų optimizavimo padalinimą.

[Petit – 1996]

Petito metodas [41] reikalauja reliacinės schemos su unikaliais ir nenuliniais apribojimais, duomenimis ir išeities kodu (SQL užklausos). Rezultatas – EER modelis. Kad atstatytų išorinius raktus ir funkcines priklausomybes, metodologija analizuoja užklausų sąlygas ir vaizdus. Tuomet šią schemą restruktūrizuoja, kad išgautų trečios normalinės formos loginę schemą. Galiausiai verčia schemą į konceptualią schemą, kurią turi patvirtinti srities ekspertas.

Petito metodas praplėstas [33]: analizuojant loginius perėjimus atstatomos įjungimo priklausomybės. Loginis perėjimas – jungimo stulpelių panaudojimas reliacinei DB pereiti. Pristatytas įrankis DBA companion, kuris atranda tokias struktūras.

MeRCI [Comyn – 1996]

MeRCI metodas [13] nėra charakteringas konkrečiam DBVS modeliui, tačiau metodui paaiškinti naudojamas reliacinis modelis. Šis metodas nagrinėja programos išeities kodą: DB deklaraciją (SQL DDL) ir programos procedūrinį kodą (su įterpta SQL). Rezultatas – išplėstas esybių-ryšių modelis su sudėtingais atributais iš apibendrinimu/specializacija.

Metodas susideda iš 5 žingsnių. Pirmas žingsnis iš duomenų žodyno, DDL ir rodinių išgauna fizinę schemą. Antras žingsnis pritaiko fizinės atvirkštinės inžinerijos taisyklių rinkinį, kad pašalintų fizinės schemos optimizavimus. Kad aptiktų optimizavimus, metodas ištiria DDL, SQL užklausas ir duomenų bazės praplėtimą. Trečias žingsnis analizuoja įterptą SQL, sinonimus ir integralumo apribojimus ir nustato esybių tipus, ryšių tipus ir kardinalumus. Jie papildo indikatorių taisykles [45]. Ketvirtas žingsnis analizuoja užklausas bei duomenis ir atstato apibendrinimo/specializaciją. Paskutinis žingsnis nustato daugiareikšmius atributus, esybių tipus ir ryšių tipus. MeRCI metodą realizuoja sistema [2].

Varlet [Jahnke – 1999]

Varlet metodas [26], [27] nagrinėja visus įmanomus informacijos apie reliacinę DB išteklius: reliacinės DB deklaraciją (SQL DDL), procedūrinį kodą (įterptą SQL), duomenų bazės praplėtimą (duomenys), dokumentaciją ir t.t. Rezultatas – objektinė konceptuali schema (OMT).

Metodas susideda iš dviejų pagrindinių etapų: schemos analizės ir konceptualios schemos migravimo. Schemos analizės etapas analizuoja skirtingas DB dalis, kad išgautų nuoseklią ir išbaigtą loginę duomenų struktūrą. Konceptualaus migravimo etapas loginę duomenų struktūrą transformuoja į konceptualią schemą.

Pagrindinis šio metodo įnašas – grafinė kalba, pavadinta GFRN, skirta atvaizduoti analitiko loginės schemos žinias. GFRN specifikacijos atskiria deklaratyvias žinias nuo vykdymo aspektų. Metodas leidžia sutvarkyti neaiškius ir prieštaringus analizės rezultatus. Išvystyta prototipinė CARE aplinka, kuri naudoja GFRN specifikacijas ir konfigūruojamą sąsają.

[Alhaij – 2001]

Alhajo metodas [3] nagrinėja fizinę schemą su pirminiais raktais ir išoriniais raktais bei vartotojo žinias apie liktinę DB. Rezultatas – objektinė schema, tačiau konkretus modelis nenurodytas. Metodas verčia reliacinę schemą į tarpinį modelį RIDG. RIDG yra grafas, kur kiekvienas mazgas atvaizduoja lentelę, o briaunos rodo, kad tarp dviejų lentų yra išorinis raktas. Metodas RIDG transformuoja į klases.

Metodas taip pat pristato algoritmą duomenims migruoti iš reliacinės DB į objektinę DB.

1.10.2.Metodų palyginimas

Išanalizavus atvirkštinės duomenų inžinerijos metodus, sudaryta jų palyginimo lentelė. Šioje lentelėje metodai yra lyginami pagal šiuos kriterijus:

Rezultato modelis – nurodo rezultato konceptualųjį modelį.

Būtinos sąlygos – išvardina metodo reikalavimus DB, kuriai taikoma atvirkštinė inžinerija. Naudojami trumpiniai reiškia: PR – išgauti pirminiai raktai; IR – išgauti išoriniai raktai, 3NF – atitikimas 3-ai normalinei formai

Semantikos atstatymas – pateikia semantikai (kuri nėra išreikšta fizinėje schemoje) atstatyti naudojamą metodiką.

Euristika – nurodo metodo naudojamas euristikas – „Transf“ reiškia, kad konceptualiąją schemą metodas išgauna transformuodamas fizinę schemą. Didžioji autorių dalis nenaudoja termino „transformavimas“, vietoje jo naudoja terminą „perrašymas“ – konceptualios schemos sukūrimas, pridedant į schemą objektus, kurie atitinka fizinės schemos elementus.

Įrankis – pateikia įrankio, kuris palaiko metodą, pavadinimą, jei toks įrankis egzistuoja.

Informacijos šaltinis nurodo informacijos šaltinius, kuriuos naudoja metodas.

Metodų lyginamoji analizė pateikta 1.2 lentelėje.

1.2 lentelė. ADI metodų palyginimas

Nr.	Metodas [autorius – metai]	Rezultato modelis	Būtinos sąlygos	Semantikos atstatymas	Euristika	Įrankis	Informacijos šaltinis
•	[Hainut – 1981]	ER	Loginė schema	-	Transf	-	Loginė schema
•	[Dumpala – 1983]	ER	PER, IR	-	Transf	Pasiūlytas	Fizinė schema
•	[Navathe – 1987]	ECR	3NF, pavadinimas	-	Transf	-	Fizinė schema
•	[Casanova – 1983]	ER	PER, IR	-	Transf	-	Fizinė schema
•	[Markowitz – 1987]	ER	PER, IR	-	Transf	-	Fizinė schema
•	[Davis – 1987]	ER	3NF	-	-	-	Fizinė schema
•	[Premerlani – 1987]	OMT	-	Vartotojo žinios	Schemos ir duomenų analizė, transf	OM įrankis	DDL, duomenys, srities žinios
•	[Johannesson – 1994]	ER	3NF su įjungimo priklausomybėmis	-	Transf	-	Fizinė schema
•	[Signore – 1994]	ER	-	Kodo analizė	Kodo analizė, transf	Pasiūlytas	DDL kodas ir DML.
•	[Chiang – 1995]	EER	3NF, pavadinimas, jokių klaidų duomenyse	Schemos analizė, pavadinimas	Schemos ir duomenų analizė, transf	Ekspertinė sistema	Fizinė schema, duomenys, srities žinios
•	[Ramanathan – 1996]	OMT	3NF, jokio optimizavimo, PR, IR	-	Transf	Pasiūlytas	Fizinė schema
•	[Petit – 1996]	EER	-	Užklausų analizė	Užklausų analizė, transf	DBA companion	DDL, DML, duomenys
•	MeRCI [Comyn – 1996]	EER	-	Kodo ir duomenų analizė	Kodo ir duomenų analizė, transf	Prototipas	DDL, DML, duomenys
•	Varlet [Jahnke – 1999]	OMT	-	Kodo ir duomenų analizė	Schemos analizė, transf	Prototipas	DDL, DML, duomenys, dokumentacija
•	DB-MAIN [Hainut – 1999]	ER	-	Kodo ir duomenų analizė	Kodo ir duomenų analizė, transf	DB-MAIN	DDL, kodas, duomenys, srities žinios
•	[Alhaij – 2001]	OMT	PER, IR	-	Transf	-	Fizinė schema, srities žinios

Lentelėje pateikta, kad Hainauto, Dumpalos, Navato, Kazanovos, Markovičiaus, Deivio, Johanesono, Ramanatano ir Alhajo metodai skirti tik išgautos fizinės arba loginės schemos konceptualizavimui. Visi jie reikalauja išgautos fizinės schemos su visais pirminiais ir išoriniais raktais, dalis jų reikalauja dar ir schemos atitikimo 3 normalinei formai. Visi šie metodai analizuoja tik pačią schemą ir nenaudoja jokių kitų informacijos šaltinių. Kadangi šio tyrimo objektas yra loginės schemos išgavimo metodika, šie metodai toliau nebus nagrinėjami.

Visais šiais kriterijais Čango metodas yra panašus į atmestuosius. Skirtumas tas, kad Čango metodas analizuoja ne tik fizinę schemą, bet ir duomenų bazėje esančius duomenis.

Likusieji metodai (DB-MAIN, Premerlanio, Signorės, Petito, MeRCI ir Varlet metodai) nekelia jokių būtinų sąlygų duomenų bazės struktūrai. Visi jie analizuoja DDL kodą t.y. duomenų struktūros deklaracijų sakinius. Praktikoje DDL kodo išgavimas iš duomenų bazės nesudaro jokių problemų, kadangi yra daug metodų kaip pagal turimos duomenų bazės struktūrą suformuoti DDL išraiškas. DDL analizė nagrinėta nuo IX dešimtmečio pradžios, todėl egzistuoja daug komercinių įrankių jai atlikti. DDL kodo analizė išgauna visas išreikštąsias struktūras ir apribojimus.

Neišreikštosios struktūros ir apribojimai išgaunami kituose šaltiniuose t.y. visų pirma duomenyse esančiuose duomenų bazėje, programos kode ir ypač jame esančiose DML išraiškose. Be to visi šie metodai reikalauja, kad analitikas vykdantis atvirkštinę inžineriją, būtų probleminės srities ekspertas. Kadangi tuo atveju, jei sistema negali savarankiškai priimti vienareikšmio sprendimo, sprendimą turi priimti ekspertas pagal srities žinias. Neišreikštųjų struktūrų ir apribojimų išgavimo etapas yra pats sudėtingiausias visame ADI procese. Taip yra todėl, kad neišreikštosios struktūros ir apribojimai gali būti realizuojami labai įvairiuose sistemos dalyse. Apie neišreikštųjų apribojimų egzistavimą gali byloti iš įvairių šaltinių surinkti požymiai. Kai kurie apribojimai netgi gali būti realizuoti už sistemos ribų pvz.: vartotojo instrukcijose ar susiformavusioje darbo su sistema praktikoje. Kiekvienas metodas kiek skirtingai atlieka šį etapą ir išgauna skirtingo tipo apribojimus.

Toliau atliekama tik pasirinktųjų ADI metodų lyginamoji analizė. Šie metodai bus lyginami pagal tai iš kokio šaltinio ir kokio tipo struktūras ir apribojimus jie išgauna. Metodų palyginimas pagal skirtingų struktūrų ir apribojimų išgavimą pateiktas tolimesnėje 1.3 lentelėje.

1.3 lentelė. ADI metodų palyginimas pagal išgaunamas struktūras ir apribojimus

Eil. Nr.	Metodo pavadinimas [autorius -metai]	Išgaunamos struktūros ir apribojimai												
		Esybės	Atributai	Pirminiai identifikatoriai	Antriniai identifikatoriai	Neprivalomi atributai	Išvardintų reikšmių sritis	Reikšmių srities apribojimai	Ryšio apribojimai	Ryšio kardinalumas	Funkcinės priklausomybės	Pertekliaus apribojimai	Egzistavimo apribojimai	Apjungtos esybės
1	DB-MAIN [Hainaut -1991]	+	+	+	+	+	+	+	+	+	+	+	+	–
2	[Premerlani - 1993]	1	1	1,2	1,2	–	–	–	1,2	–	–	–	–	–
3	[Signore - 1994]	1	1	1,3	–	–	–	–	1,3	–	–	–	–	–
4	[Chiang - 1995]	1	1	1	–	–	–	–	1,2	–	–	–	–	–
5	[Petit - 1996]	1	1	1	–	–	–	–	1,3	–	1,3	–	–	–
6	MeRCI [Comyn -1996]	1	1	1	–	–	–	1,2	1,2,3	1,2,3	–	–	–	–
7	Varlet [Jahnke -1999]	1	1	1	–	1,2	1,2	1,2	1,2,3	1,2,3	–	–	–	–

Analizuojami šaltiniai:

1 – DDL kodas; 2 – duomenys; 3 – DML, programos kodas; „+“ – visi įmanomi šaltiniai, „–“ – neišgauna

Apžvelgus ADI metodus, matome kad nėra nei vieno tokio metodo, kuris išgautų visas analizuojamas struktūras ir apribojimus. Iš visų metodų akivaizdžiai išsiskiria DB-MAIN. DB-MAIN tai labiau metodika nei metodas, kadangi tai labai išsamus, sudėtingas visus šaltinius analizuojantis ir didžiąją dalį apribojimų išgaunantis metodas. Jis apima tiek atvirkštinę, tiek tiesioginę inžineriją ir analizuoja visus pagrindinius duomenų modelius. Dėl savo sudėtingumo šio metodas naudojimui reikalingos gilios ADI ir DB teorijos žinios. Tai įvardintina kaip pagrindinis šio metodo trūkumas. Jis tinkamas dideliems ir sudėtingiems ADI projektams, tačiau netinkamas mažesniems projektas, kuriuos atlieka tik pagrindines ADI ir DB teorijos žinias turintys vykdytojai.

Kiti lentelėje pateikti metodai yra gerokai siauresni ir labiau koncentruoti. Jie orientuoti į tam tikrų struktūrų ir apribojimų išgavimą ir/arba į tam tikrų šaltinių analizę. Juose nekeliamas tikslas analizuoti visus įmanomus šaltinius ir išgauti visas įmanomus struktūras ir apribojimus.

Visi lentelėje pateikti metodai išgauna duomenų bazėje deklaruotą struktūra ir neišreikštuosius ryšio apribojimus. Tik dalis jų atlieka neišreikštųjų apribojimų atributams paiešką. Ir visgi nei vienas iš jų nemėgina atrasti esybių, kurios fizinėje schemoje yra apjungtos į viena lentelę.

1.11. Analizės išvados

1. Atlikus atvirkštinės duomenų inžinerijos proceso analizę, prieita išvados jog visas ADI procesas, išskyrus neišreikštųjų struktūrų ir apribojimų išgavimą, yra gerai žinomas. Šis procesas yra sudėtingas ir nevienareikšmis, reikalaujantis papildomų eksperto įsikišimų ir sprendimų. Dėl savo sudėtingumo šio proceso neįmanoma pilnai automatizuoti.
2. Atlikus esamų ADI metodų analizę, nustatyta kad didžioji dalis metodų skirti konkretiems apribojimams iš konkrečių šaltinių išgauti ir neapima viso ADI proceso. Vienas metodas yra skirtas visų įmanomų apribojimų suradimui analizuojant visus įmanomus šaltinius, apima tiek tiesioginę duomenų inžineriją tiek ADI, tačiau šis metodas pasižymi aukštu sudėtingumo laipsniu.
3. Atlikus ADI aktualumo praktikoje tyrimą prieita išvados, jog dėl ADI proceso sudėtingumo, pilna apimtimi šis procesas naudojamas ypač retai. Absoliučia dauguma ADI taikymo atvejų atliekama tik dalinė ADI, su tikslu atstatyti duomenų schemą tiek, kad jos pakaktų tik aktualių duomenų migracijai atlikti. Dažnai vietoj ADI yra pasirenkama alternatyva: duomenų migraciją daryti rankiniu arba pusiau rankiniu būdu ir palaikyti liktinės sistemos kaip archyvo veikimą t.y. nevykdant jokios migracijos ar ADI.

4. Atlikus ADI informacijos šaltinių analizę prieita išvados, jog patys svarbiausi ADI šaltiniai neišreikštosioms struktūroms ir apribojimams atrasti yra duomenų bazėje sukaupti duomenys bei DML išraiškos išgautos iš programos kodo ir DBVS.
5. Apibendrinus analizės išvadas, pastebėta jog nėra metodo, kuris būtų orientuotas „į viduriuką“. T.y. į tik pagrindines ADI ir DB teorijos žinias turinčiam analitiką, kuris ADI naudoja išimtinai duomenų semantikai atstatyti, kuri būtų pakankama duomenų migracijai. Duomenų migracijai nėra aktualu atstatyti absoliučiai visus schemos apribojimus, o tik tam tikrus. Todėl nusprendėme kad būtų tikslinga sudaryti metodiką, kurią būtų lengva pritaikyti praktikoje.

2. SUDARYTOS ADI METODIKOS APIBRĖŽIMAS IR ESMĖ

Tikslai. Šio skyriaus tikslas remiantis analizės dalyje padarytomis išvadomis, suformuoti ADI metodiką. Aprašyti ADI metodikos aplinką, vartotojus ir procesus.

Metodai. Sprendimo konstravimui bus taikomas objektinio sistemų projektavimo ir realizavimo metodas. Projektavimui atlikti pasirinktas objektinis sistemų projektavimo metodas, taikant RUP projektavimo procesą ir UML projektavimo kalbą. Projektavimui bus naudojamas Magic Draw įrankis. RUP ir UML pagalba galima tiksliai atskleisti dinamines ir statines metodikos savybes.

Skyriaus struktūra:

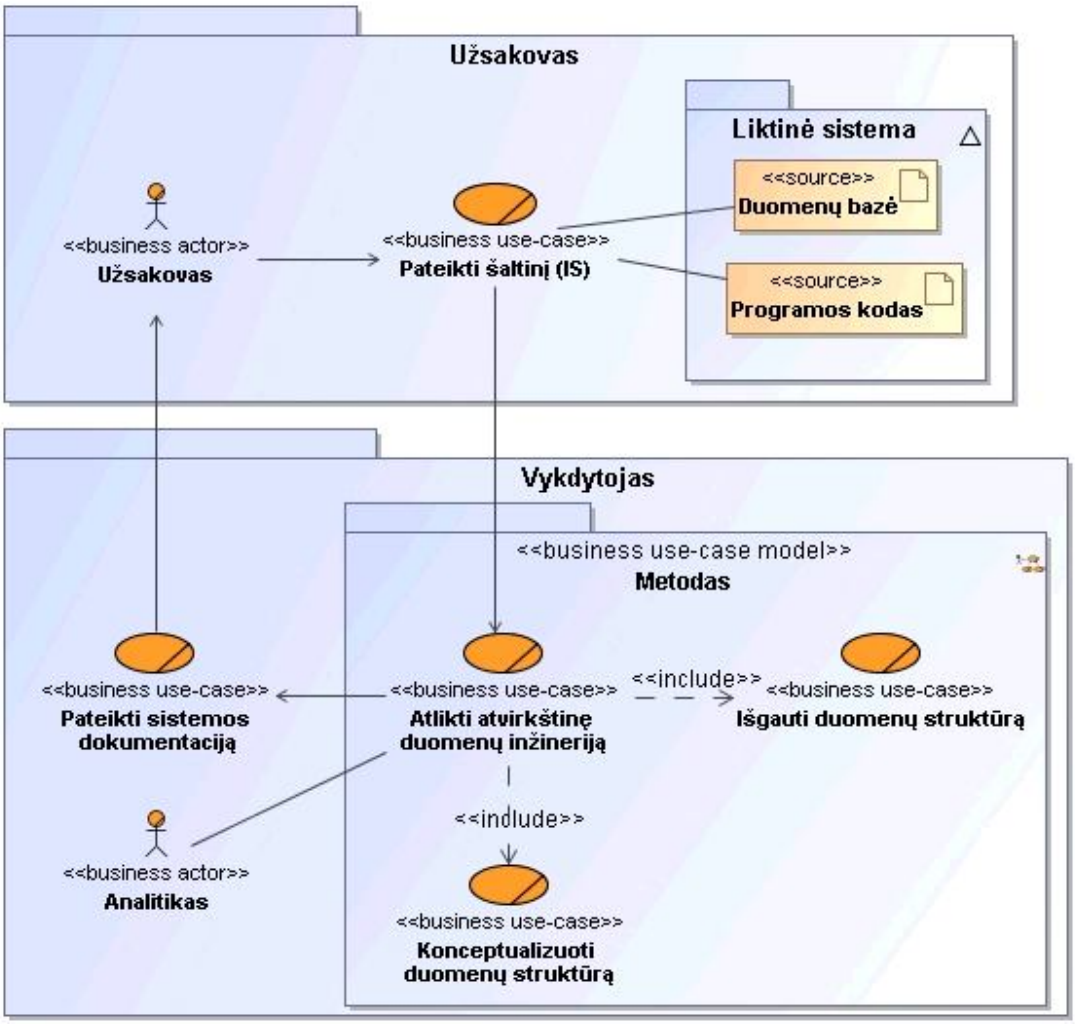
2.1 skyriuje aprašyta ADI metodikos aplinka.

2.2 skyriuje aprašyti vartotojai kurie taikys šią metodiką.

2.3 skyriuje pristatyta ADI metodika. Šio skyriaus poskyriuose reikalavimų specifikacijomis detaliai aprašyti ADI metodikos etapai.

2.1. ADI metodikos aplinka, sąlygos ir prielaidos

Sukurta metodika išgauti loginę duomenų schemą iš veikiančios informacijos sistemos. Ši metodika yra sudėtinė atvirkštinės duomenų inžinerijos dalis. Todėl pirmiausiai bus pateikta metodikos taikymo aplinka. Kontekstinė viso ADI proceso diagrama pateikta panaudojimo atvejų diagramoje (2.1 pav.).



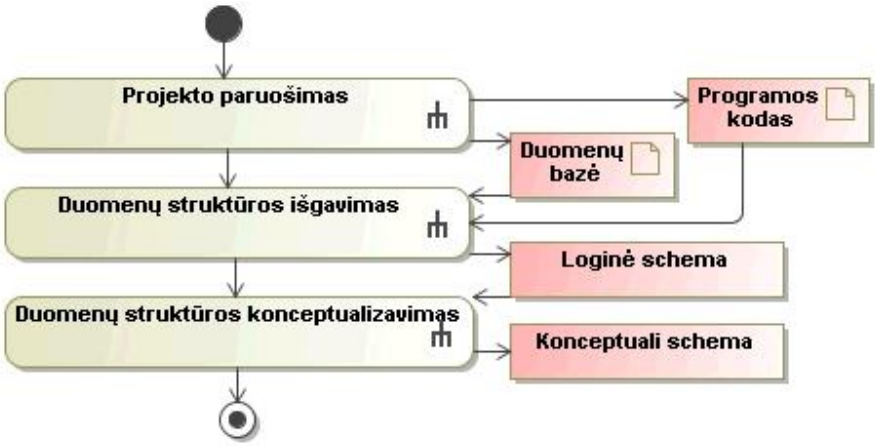
2.1 pav. Kontekstinė veiklos diagrama

Visas procesas prasideda nuo to, kad užsakovas pateikia užsakymą analitikui atvirkštinei duomenų inžinerijai atilikti. Kartu užsakovas pateikia ir ADI šaltinį – sistemą su tokiais pagrindiniais jos komponentais: duomenų bazė (DB struktūra ir duomenys) ir procedūrinė dalis (vykdomąją dali ir sistemos programinį kodą). Šaltinio pateikimas nėra suprantamas kaip visos sistemos veikiančios kopijos pateikimas, bet prieigos prie originalios sistemos užtikrinimas. Be to sistemos programinis kodas pateikiamas tik jei toks šaltinis yra turimas.

Visas ADI atlikimas suprantamas kaip du pagrindiniai procesai: duomenų struktūros išgavimas ir struktūros konceptualizavimas. Pirmasis procesas yra būtinas bet kuriuo atveju, antrasis – tik jei užsakovas to pageidauja. Sukurta ADI metodika yra skirta pirmajam procesui, t.y. duomenų struktūros išgavimui, realizuoti.

Atlikus ADI, tiek loginė schema tiek konceptuali schema dokumentacijos forma pateikiama užsakovui.

ADI procesas be pagrindinių dviejų turi vieną paruošiamąjį procesą (2.2 pav.), kurio tikslas yra surinkti ir įvertinti būtinų informacijos šaltinių aktualumą ir susipažinti su sritimi. Kaip parodoma schemoje, šis procesas paruošia šaltinius tolimesniam žingsniui – duomenų struktūros išgavimui. Pastarojo etapo rezultatas yra išbaigta loginė schema, kurios pagrindu paskutiniame etape yra sudaroma konceptuali duomenų schema.



2.2 pav. Konceptuali ADI proceso diagrama

Metodika remiasi tokiomis sąlygomis ir prielaidomis. Laikoma, kad duomenų bazėje esančių duomenų kiekis yra pakankamas, ir juose atvaizduojama visų įmanomų tipinių (arba užsakovui aktualių) dalykinės srities atvejų.

Procedūrinės dalies analizės metu išgaunamos DML išraiškos. Jei priėjimo prie programos kodo nėra, DML išraiškos gali būti surinktos iš kitų šaltinių: iš DBVS žurnalų, kurie kaupiami sistemos eksploatacijos metu, DBVS procedūrinių dalių (be abejo, su sąlyga kad DBVS turi tokį funkcionalumą).

Net ir turint prieigą prie programos kodo, jo analizė yra labai sudėtingas procesas. Visų pirma reikia suformuoti programavimo kalbos leksinį modelį, sudaryti leksinį analizatorių, tuomet sugeneruoti sintaksės analizatorių konstruojantį abstrakčios sintaksės medį. Net ir atlikus šiuos veiksmus gali kilti problemų su DML išraiškų išgavimu, kadangi išraiškos gali būti formuojamos dinamiškai programos vykdymo metu, gali būti naudojamos įvairios bibliotekos, bendrinės funkcijos, automatinės transformacijos iš/į objektinį modelį ir kito priemonės kurios suskaido, išbarsto ir visiškai ar iš dalies paslepia DML išraiškas iš pagrindinio sistemos kodo. Dėl to šios

analizės gali būti atsisakoma, kaip reikalaujančios daug sąnaudų ir susijusios su nemaža nesėkmės tikimybe.

2.2. ADI metodikos vartotojų analizė

Vienintelis šios metodikos vartotojas, tai analitikas. Visa metodikos naudojimo, atskirų etapų atsisakymo, sprendimų ir galutinių rezultatų kokybė yra šio vartotojo atsakomybėje. Galima išskirti dvi šio vartotojo roles ar kompetencijos sritis.

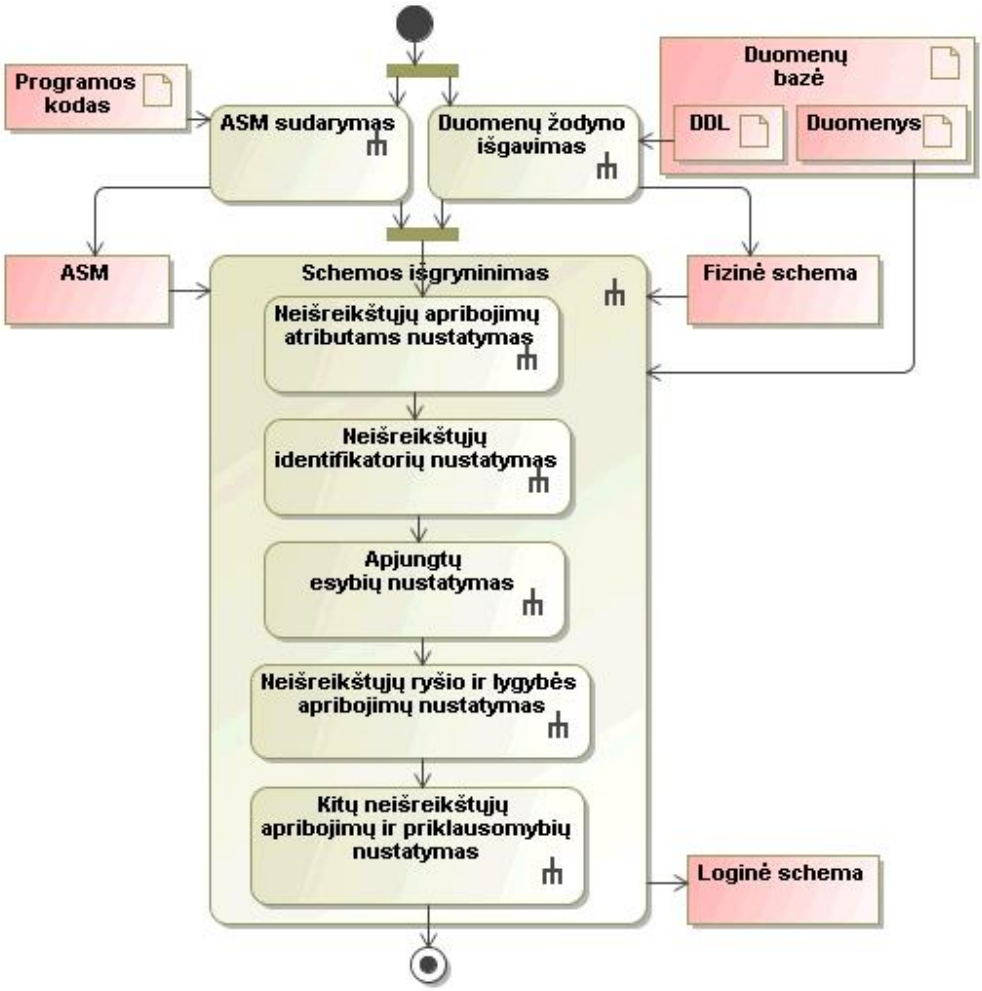
ADI analitikas – metodika besinaudojantis asmuo turi giliai išmanyti ADI: reliacinį modelį, duomenų struktūras ir apribojimus, proceso problematiką.

Dalykinės srities ekspertas – kai kuriais atvejais, jei grynai metodikos pagalba galutinio sprendimo priimti negalima, sprendimas turi būti priimamas pagal dalykinės srities žinias.

2.3. ADI metodika ir jos etapai

ADI metodikos tikslas yra atstatyti išsamią loginę schemą, jos išgryninimo etape interaktyviajame režime panaudojant skirtingų šaltinių informacijos vientisumo patikrinimą.

Pradinė schemos informacija išgaunama iš sistemos duomenų bazių valdymo sistemoje (DBVS, angl. database managment system) deklaruotos duomenų struktūros. Po to šią schemą galima semantiškai pagerinti analizuojant detales iš duomenų rinkinio ir taikomosios programos kodo (2.3 pav.).



2.3 pav. Konceptuali ADI metodikos diagrama

ADI metodika susideda iš septynių pagrindinių etapų, kurie apimą pagrindinį ir sunkiausią atvirkštinės duomenų inžinerijos etapą t.y. duomenų struktūros išgavimo etapą

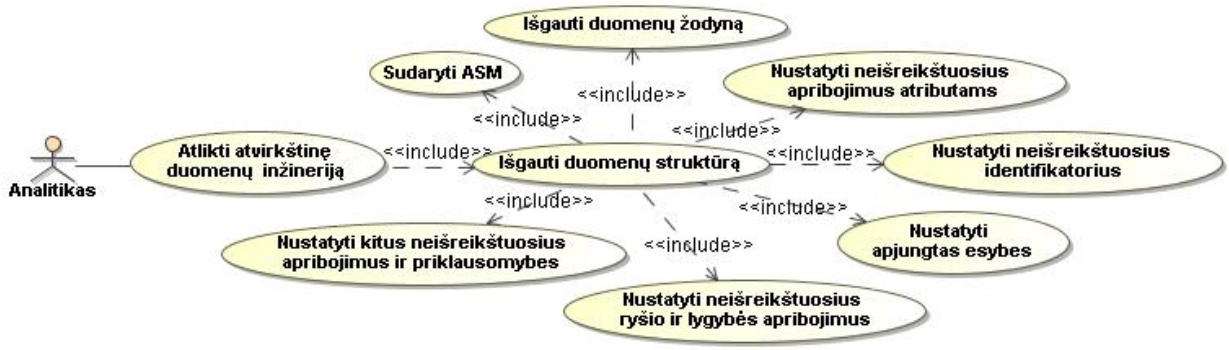
Duomenų struktūros išgavimas yra svarbiausia ir sunkiausia ADI dalis. Analizuoja egzistuojančią liktinę sistemą, kad atstatytų išbaigtą loginę schemą, kuri apimtų išreikštąsias ir neišreikštąsias duomenų struktūras bei savybes (ir apribojimus). Neišreikštosioms struktūroms bei apribojimams išgauti naudojamos įvairios programų supratimo ir duomenų analizės metodikos.

Duomenų struktūros išgavimo dalis, susideda iš dviejų procesų: duomenų žodyno ir duomenų schemos išgryninimo proceso.

Duomenų žodyno išgavimas yra pats paprasčiausias duomenų struktūros išgavimo proceso etapas. Išgauna neapdirbtą fizinę schemą, kuri apima tik visas išreikštąsias duomenų struktūras ir apribojimus.

Schemos išgryninimas yra pagrindinis duomenų struktūros išgavimo proceso etapas. Pagrindinė problema, kad didelė dalis struktūrų ir apribojimų ir sąvokų neišreikšti, t.y. jie ne deklaruoti, bet realizuoti procedūrinėse programų dalyse. Yra daug neišreikštųjų struktūrų, pagrindinės iš jų: pirminiai ir antriniai identifikatoriai, ryšio, lygybės ir kt. apribojimai, funkcinės priklausomybės, prasmingi pavadinimai. Šiame etape fizinė schema praturtinama neišreikštosiomis struktūromis ir apribojimais ir sudaroma išbaigta loginė schema.

Toliau aprašomas kiekvienas ADI metodikos etapas ir susijusios veiklos. ADI metodikos etapai aprašyti panaudojimo atvejų diagrama ir jos specifikacijomis (2.4 pav.).



2.4 pav. Metodikos funkcinių reikalavimų aprašymas panaudojimo atvejų diagrama

Reikalavimų panaudojimo atvejų specifikacijos sudaromos kiekvienam panaudojimo atvejui atskirai. Reikalavimų specifikacijomis detaliai aprašyti ADI metodikos etapai. Kiekvienas etapas detalizuojamas sekančiuose poskyriuose. Visos reikalavimų specifikacijos pateiktos 2.5 – 2.11 paveiksluose.

2.3.1. Duomenų žodynų išgavimas

Duomenų žodyno išgavimas yra pats paprasčiausias duomenų struktūros išgavimo proceso etapas. Šis etapas turi išgauti fizinę schemą, kuri apima visas išreikštąsias duomenų struktūras ir apribojimus:

- Rinkiniai (pavadinimas);
- Esybės (pavadinimas, aprašymas);
- Atributai (pavadinimas, esybė, duomenų tipas, maksimalus ilgis, reikšmė pagal nutylėjimą, aprašymas, būtinumo požymis);
- Identifikatoriai (pavadinimas, esybė, atributai, unikalumas);

- Išreikštieji ryšio apribojimai (pavadinimas, esybė, atributai, pirminė esybė, pirminiai atributai).

2.5 pav. PA „Išgauti duomenų žodyną“ specifikacija veiklos diagrama

2.3.2. Abstrakčios sintaksės medžio sudarymas

Siekiant panaudoti programų supratimo metodikas, pirmiausia reikia sugeneruoti liktinės programos kodo abstrakčios sintaksės medį (ASM, angl. *abstract syntax tree*). ASM palaiko keletą schemos išgryninimo etapo procesų (2.6 pav.).

2.6 pav. PA „Sudaryti ASM“ specifikacija veiklos diagrama

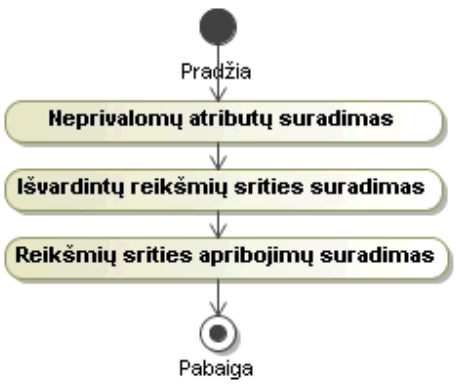
Siekiant sudaryti kodo ASM, reikia atlikti leksinę analizę (angl. *lexical analysis*) ir nagrinėjimą (angl. *parsing*). Programos kodo leksinis analizatorius skaito išeities kodą eilutė po eilutės ir padalina jį į leksemas (angl. *token*). Nagrinėtojas nuskaito šias leksemas ir sukuria kalbos gramatiką (A priede pateikta C kodo gramatika ASM sudaryti) atitinkantį išeities kodo ASM. Šis metodas puikiai veikia procedūrinėms kalboms, pvz. C. Tačiau metodo tiesiogiai taikyti objektinėms

kalboms (pvz. Java) negalima, nes dėl įvairių priežasčių (daugialypio paveldėjimo nevienareikšmiškumo, klasių ir objektų specializacijos skirtingumo ir pan.) ženkliai padidėja problemos sudėtingumas.

2.3.3. Neišreikštųjų apribojimų atributams nustatymas

Šio etapo tikslas yra iš liktinių šaltinių duomenų bazės duomenų išgauti struktūras ir apribojimus, kurių nepavyko atrasti duomenų žodyno išgavimo etape (2.7 pav.):

- Neprivalomų atributų suradimas. Duomenų žodyno išgavimo etape išgautos atributų būtinumo savybės, tačiau gali egzistuoti neprivalomi atributai, kurie deklaruoti kaip privalomi atributai. Šiems atributams suteikta ypatinga reikšmė, kuri nurodo trūkstamą arba nežinomą reikšmę. Kadangi nėra standartinio būdo šiam apribojimui realizuoti, todėl reikia analizuoti duomenis. Atributams, kurie nėra pirminiai identifikatoriai ir yra privalomi, patikrinti.
- Išvardintų reikšmių srities suradimas. Daugelis atributų privalo naudoti iš anksto nustatytos reikšmių aibės reikšmes. Atributams, kurie nėra pirminiai identifikatoriai ar išoriniai raktai, nustatyti visas galimas reikšmes.
- Reikšmių srities apribojimų suradimas. Dažnai DBVS deklaruojamos duomenų struktūrų reikšmių sritis yra labai skurdi. Leidžiamoms reikšmėms gana dažnai taikomi griežti apribojimai.

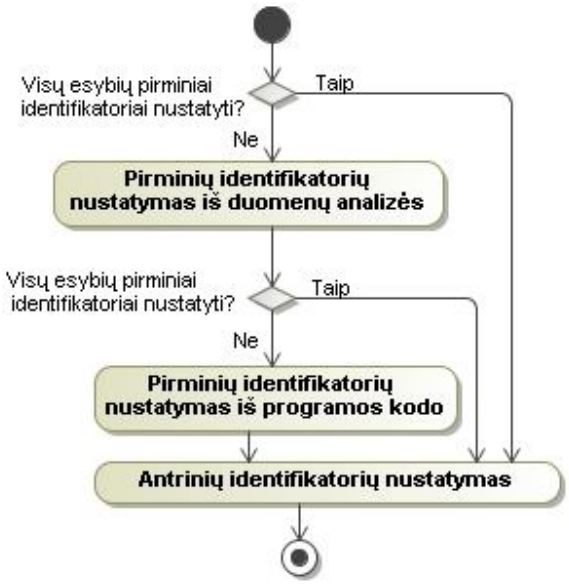


2.7 pav. PA „Nustatyti neišreikštuosius apribojimus atributams“ specifikacija veiklos diagrama

2.3.4. Neišreikštųjų identifikatorių nustatymas

Etapo tikslas yra iš liktinių šaltinių išgauti pirminius identifikatorius toms lentelėms, kurių nepavyko išgauti duomenų žodyno išgavimo etapu. Etape pirmiausiai pirminius identifikatorius

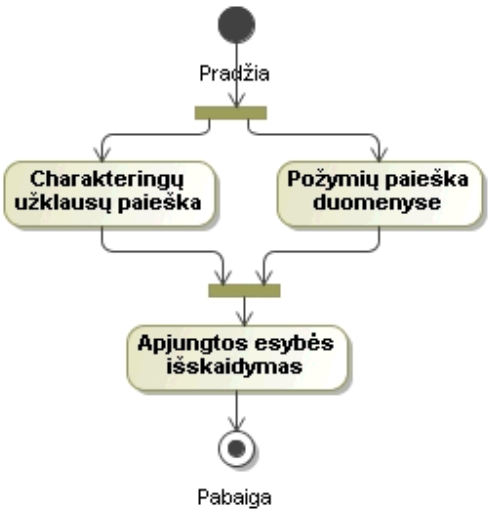
bandoma nustatyti iš duomenų analizės, vėliau iš programos kodo. Etape taipogi nustatomi antriniai identifikatoriai, kurie taip kaip ir pirminiai gali būti naudojami ryšio apribojimuose (2.8 pav.).



2.8 pav. PA „Nustatyti neišreikštuosius identifikatorius“ specifikacija veiklos diagrama

2.3.5. Apjungtų esybių nustatymas

Šio etapo tikslas yra nustatyti ar fizinėje lentelėje nėra apjungta kelių panašios struktūros bet skirtingų semantikos esybių. Esybės gali būti apjungtos dėl techninių ar kitų išorinių apribojimų. Tokie apjungimai nustatomi analizuojant esybių pirminių identifikatorių reikšmių sritį, bei atrandant charakteringas DML išraiškas programos kode.



2.9 pav. PA „Nustatyti apjungtas esybes“ specifikacija veiklos diagrama

Formaliai apjungtas esybes galima apibrėžti tokiu būdu.

Sakykime, esybės tipas $R = \{A_1, A_2, \ldots, A_n\}$, kur kiekvieną atributą A_i atitinka domenas – netuščia ir suskaičiuojama atributo reikšmių aibė $D_i = dom(A_i)$. Sakykime, kad $D = D_1 \cup D_2 \cup \ldots \cup D_n$. Esybės R egzempliorių aibė $e(R)$ – tai aibės R atvaizdų aibėje D rinkinys $\{t_1, t_2, \ldots, t_p\}$, $e(R) = \{t_1, t_2, \ldots, t_p\}$. Čia p – esybių egzempliorių $e(R)$ aibės dydis. Kiekvienas egzempliorius $t_j \in e(R)$ tenkina apribojimą $t_j(A_i) \in D_i$; $1 \le i \le n$; $1 \le j \le p$. Egzemplioriuje t_j atributas A_i turi reikšmę $t_j(A_i) = d_k$, kuri yra k-toji reikšmių aibėje D_i.

Tarkime, kad esybė R tai jungtinė esybė, sudaryta iš apjungtųjų esybių $R_1, R_2, ..., R_m$. Čia m – apjungtųjų esybių kiekis. Apjungtoji esybė R_i $(1 \le i \le m)$ sudarytas ir ni kiekio atributų $R_i = \{A_{1,i}, A_{2,i}, \ldots, A_{ni,i}\}$ ir susijusi su pi kiekiu egzempliorių $e(R_i) = \{t_{1,i}, t_{2,i}, \ldots, t_{pi,i}\}$. $null(R_i)$ – tai atributų, kurie visuose esybės R_i egzemplioriuose neįgyja reikšmės, aibė. Kiekvienas jungtinės esybės egzempliorius priklauso tik kažkuriai vienai iš apjungtųjų esybių:

$e(R) = e(R_1) \cup e(R_2) \cup ... \cup e(R_m)$;

$e(R_i) \cap e(R_j) = Ø$; $1 \le i \le m$; $1 \le j \le m$; $i \ne j$

Sakykime esybės egzempliorių identifikuojančių atributų aibė $I_R \in R$. $I_R = I_{disk} \cup I_{id}$, kur I_{disk} identifikuojančių atributų aibė, kurie apjungtos esybės atveju atlieka esybės tipo diskriminatorių rolę, o I_{id} – apjungtosios esybė tipo egzempliorių identifikatorių aibė.

Elementų kiekį aibėje Z parodo $count(Z)$.

Tuo atveju, jei aibę I_R sudaro tik vienas atributas A_x, tuomet

$I_R = \{A_x\} \rightarrow ($

$I_{disk} = \emptyset$; $I_{id} = \{A_x\}$;

$R_i = R - null(R_i)$;

$I_{Ri} = I_R = \{A_x\}$;

$dom(A_x) = dom(A_{x,1}) \cup dom(A_{x,2}) \cup ... \cup dom(A_{x,m})$;

$dom(A_{x,j}) \cap dom(A_{x,k}) = \emptyset$; $1 \le j \le m$; $1 \le k \le m$; $k \ne j$;

$e(R_i) = \{t_1, t_2, ... , t_{pi}\}$;

$\forall j(1 \le j \le pi; t_j(A_x) \in dom(A_{x,i}))$

)

Atributinių reikšmių aprašytas identifikacinis atributų reikšmių grupavimas būdingas grupiniam kodavimo būdui.

Tuo atveju, jei aibe I_R sudaro daugiau kaip vienas atributas ir $I_{disk} = \{A_y\}$

$I_{id} \ne \emptyset$; $I_{disk} = \{A_y\} \rightarrow$ (

$R_i = R - I_{disk} - null(R_i)$;

$I_{Ri} = I_R - I_{disk}$;

$m = count(dom(A_y))$;

$e(R_i) = \{t_1, t_2, ... , t_{pi}\}$;

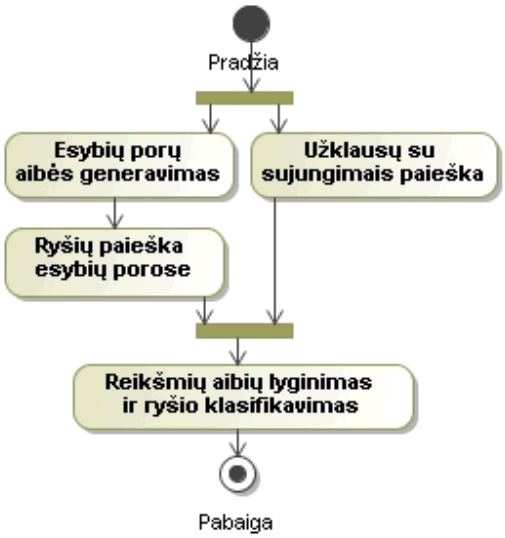
$\forall j\ (1 \le j \le pi; t_j(A_y) = d_i; d_i \in dom(A_y))$

)

Kodavimas naudojant diskriminatorių būdingas dešimtainiam kodavimui.

2.3.6. Neišreikštųjų ryšio ir lygybės apribojimų nustatymas

Etapo tikslas yra nustatyti apribojimus, kurie padėtų klasifikuoti išgautas esybes, atvaizduojančias tiek realaus pasaulio esybes, tiek ir ryšius tarp jų. Kad realizuoti šį tikslą, naudojami ryšio ir lygybės apribojimai, parodantys tarpryšinių apribojimų egzistavimą, tarp jų ir klasės/poklasio ryšius.



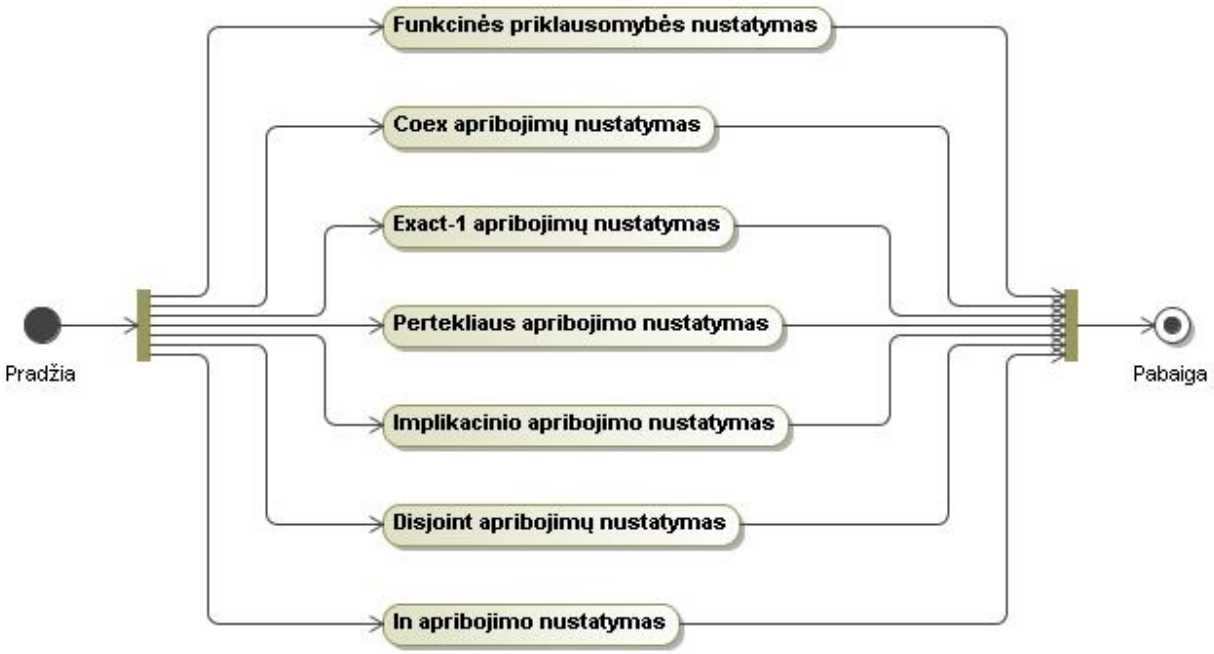
2.10 pav. PA „Nustatyti ryšio ir lygybės apribojimus“ specifikacija veiklos diagrama

2.3.7. Kitų neišreikštųjų apribojimų ir priklausomybių nustatymas

Etapo tikslas – išgauti likusius neišreikštuosius apribojimus. Šiame žingsnyje analizuojami duomenų bazėje esantys duomenys (2.11 pav.):

- **Coex** apribojimas nurodo, kad esybės tipo atributai turi būti nustatyti kartu arba visai nenustatyti. *Coex* apribojimus reikia apjungti.
- **Exact-1** Apribojimas nurodo, kad iš esybės tipo atributų turi būti nustatytas vienas ir tik vienas atributas.
- **Disjoint** apribojimas, *disjoint(R1.A1,R2.A1)* nurodo, kad esybių tipų pirminiai raktai nepersikerta. *disjoint* apribojimus reikia apjungti, pvz. jei egzistuoja apribojimai *disjoint(R1.A1,R2.A1)*, *disjoint(R1.A1,R3.A1)*, *disjoint(R2.A1,R3.A1)*, tuomet reikia apjungti į *disjoint(R1.A1,R2.A1,R3.A1)*.
- **Funkcinė priklausomybė** *R1∪R2:A1-->A2* nurodo, kad jei esybių tipų pirminiai raktai sutampa, tuomet sutampa ir jų bendri atributai. Kad nustatyti, ar egzistuoja funkcinė priklausomybė tarp esybių `R1` ir `R2` pirminių identifikatorių `A1` ir esybių atributų `A2`, reikia įvykdyti duomenų užklausą kiekvienai esybių, kurios neturi *disjoint* apribojimo, pirminių identifikatorių ir esybių atributų porai.
- **In** apribojimas *R1.A1in(R2.A1∪R3.A1)* nurodo, kad bendros esybės pirminis raktas sutampa su bent vieno esybės tipo pirminiu raktu.

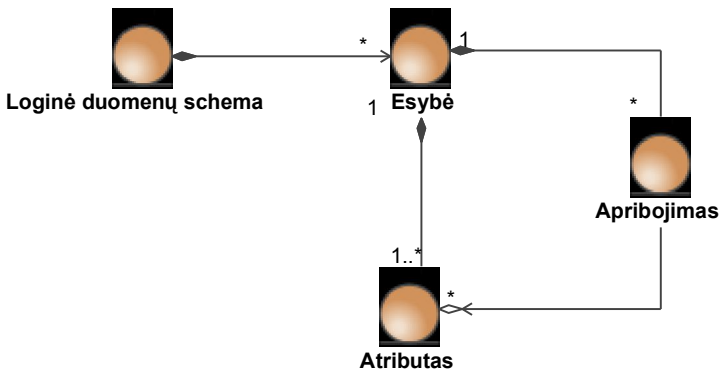
- **Implikacinis** apribojimas *R:BA1==>(BA1=A1)* nurodo, kad jei esybės tipo antrinis identifikatorius turi reikšmę, tai ji sutampa su pirminio identifikatoriaus reikšme.
- **Pertekliaus** apribojimas *rd* nurodo, kad esybės tipo atributas yra kitos esybės atributo kopija.



2.11 pav. PA „Nustatyti kitus apribojimus“ specifikacija veiklos diagrama

2.4. Metodologijos rezultatų duomenų modelis

Dalykiniai sričiai perteikti pasirinktas reikalavimų esybių klasių modelis (2.12 pav.). Reikalavimų esybių klasių modelis pavaizduoja sistemos reikalavimų objektus bei ryšius tarp jų.



2.12 pav. Loginės duomenų schemos metamodelis

3. ADI ĮRANKIO PROJEKTAS

Tikslai. Šio skyriaus tikslas pateikti ADI įrankio prototipo projektą, kuris skirtas metodikos eksperimentiniam tyrimui atlikti. Taip pat aprašyti pagrindines ADI procesą palaikančias technologijas.

Metodai. Sprendimo konstravimui bus taikomas objektinio sistemų projektavimo ir realizavimo metodas. Projektavimui atlikti pasirinktas objektinis sistemų projektavimo metodas, taikant RUP projektavimo procesą ir UML projektavimo kalbą. Įrankio projektavimui bus naudojamas Magic Draw įrankis, o realizavimui Eclipse platforma.

Skyriaus struktūra:

3.1 skyriuje pateikiamas įrankio prototipo projektas, kuris palaiko ADI metodiką.

3.2 skyriuje aprašytos ADI procesą palaikančios technologijos. Įrankio architektūra.

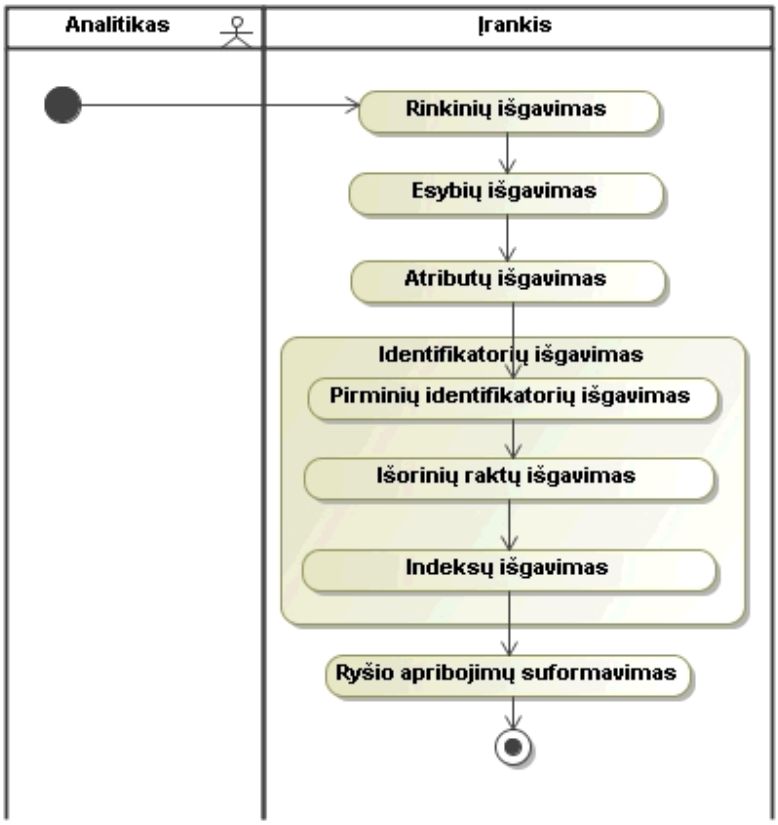
3.1. ADI eksperimentinio įrankio projektas

Šiame skyriuje pateikiamas ADI įrankio prototipo projektas. Projektas pateikia įrankio projektą, pagal analizės ir reikalavimų dalyse suformuotus reikalavimus. Šio skyriaus poskyriuose pateikiami detalūs kiekvieno ADI metodikos etapo procesai.

3.1.1. Duomenų žodyno išgavimas

Etapo tikslas yra iš liktinių šaltinių duomenų žodyno išgauti reliacines specifikacijas. Pats paprasčiausias duomenų struktūros išgavimo proceso etapas. Beveik visi CASE įrankiai pateikia DDL kodo analizės įrankius populiariausioms DBVS. Kai kurie taip pat gali išgauti reliacines specifikacijas iš sistemos duomenų žodyno. Standartizuota Java kalbos sąsaja su reliacinėmis duomenų bazėmis JDBC naudoja reliacinį SQL duomenų modelį ir tiesiogiai leidžia vykdyti SQL užklausas. JDBC meta duomenų sąsaja leidžia išgauti duomenų bazės struktūrą ir savybes (3.1 pav.):

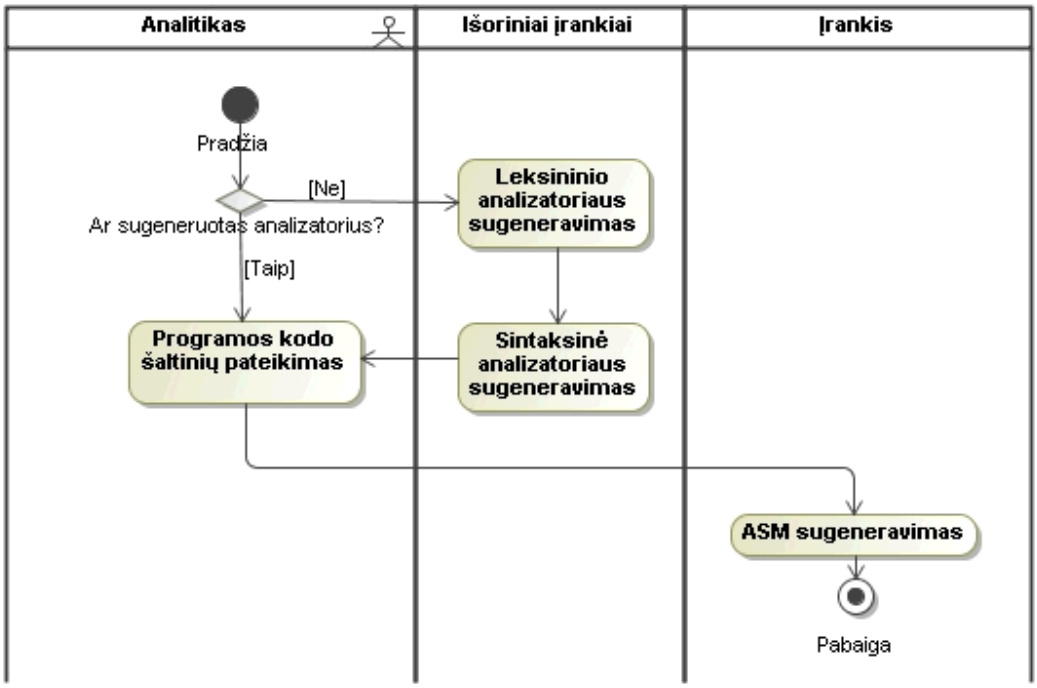
- Rinkiniai (pavadinimas);
- Esybės (pavadinimas, aprašymas);
- Atributai (pavadinimas, esybė, duomenų tipas, maksimalus ilgis, reikšmė pagal nutylėjimą, aprašymas, būtinumo požymis);
- Pirminiai identifikatoriai (pavadinimas, esybė, atributai);
- Išoriniai raktai (pavadinimas, esybė, atributai, pirminė esybė, pirminiai atributai);
- Priėjimo raktai (pavadinimas, esybė, atributas, unikalumas).



3.1 pav. Žodyno išgavimas

3.1.2. Abstrakčios sintaksės medžio sudarymas

Siekiant panaudoti programų supratimo metodikas, pirmiausia reikia sugeneruoti liktinės programos kodo abstrakčios sintaksės medį (ASM, angl. *abstract syntax tree*). ASM palaiko keletą schemos išgryninimo etapo procesų. ASM sudarymo etapo išplėtimas pateiktas (3.2 pav.).



3.2 pav. ASM sudarymas

Siekiant sudaryti kodo ASM, reikia atlikti leksinę analizę (angl. *lexical analysis*) ir nagrinėjimą (angl. *parsing*). Programos kodo leksinis analizatorius skaito išeities kodą eilutė po eilutės ir padalina jį į leksemas (angl. *token*). Nagrinėtojas nuskaito šias leksemas ir sukuria kalbos gramatiką (A priede pateikta C kodo gramatika ASM sudaryti) atitinkantį išeities kodo ASM. Šis metodas puikiai veikia procedūrinėms kalboms, pvz. C. Tačiau metodo tiesiogiai taikyti objektinėms kalboms (pvz. Java) negalima, nes dėl įvairių priežasčių (daugialypio paveldėjimo nevienareikšmiškumo, klasių ir objektų specializacijos skirtingumo ir pan.) ženkliai padidėja problemos sudėtingumas.

3.1.3. Neišreikštųjų apribojimų atributams nustatymas

Etapo tikslas yra iš liktinių šaltinių duomenų bazės duomenų išgauti struktūras ir apribojimus, kurių nepavyko atrasti duomenų žodyno išgavimo etape:

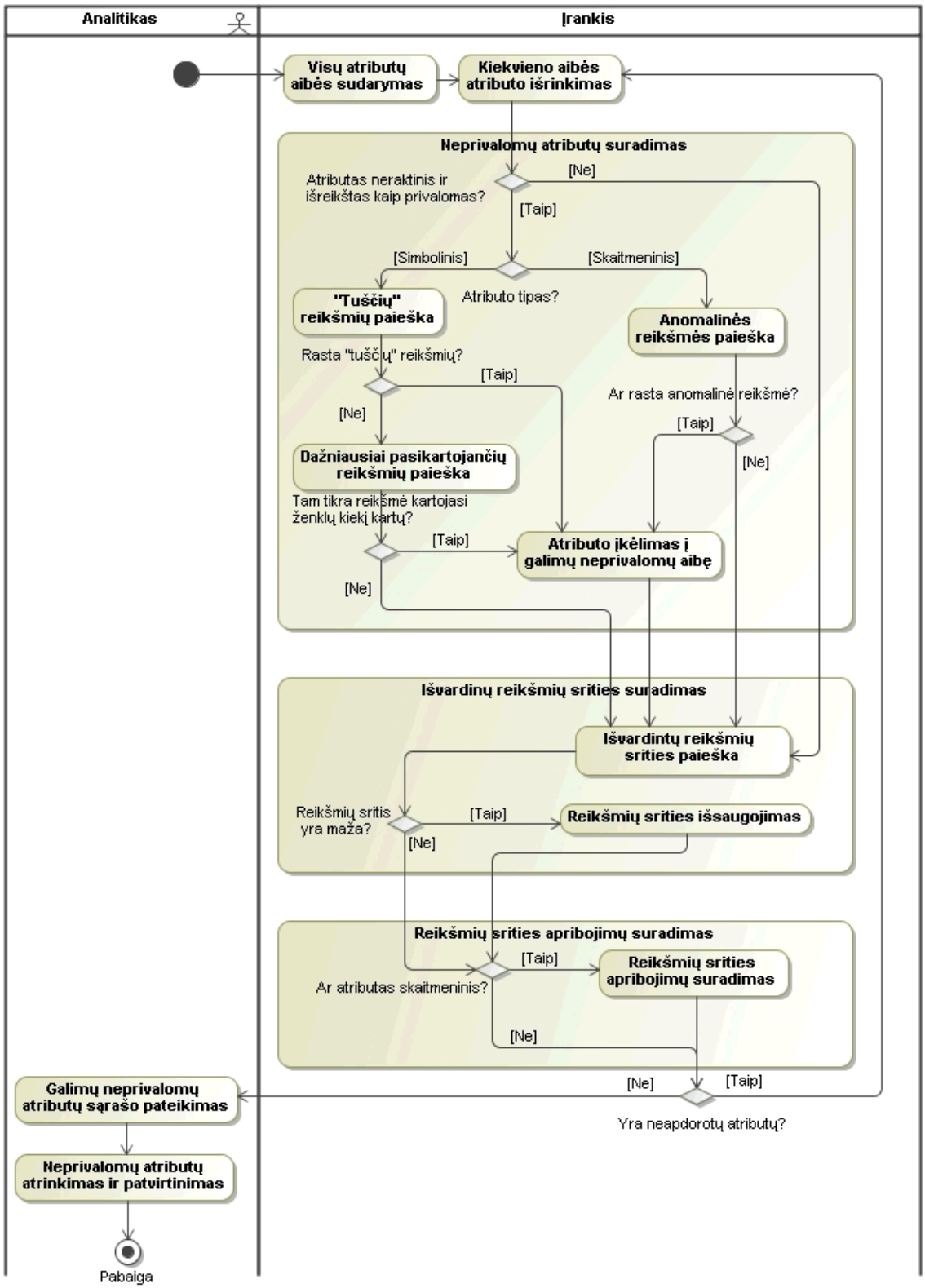
- Neprivalomų atributų suradimas. Duomenų žodyno išgavimo etape išgautos atributų būtinumo savybės, tačiau gali egzistuoti neprivalomi atributai, kurie deklaruoti kaip privalomi atributai. Šiems atributams suteikta ypatinga reikšmė, kuri nurodo trūkstamą arba nežinomą reikšmę. Kadangi nėra standartinio būdo šiam apribojimui

realizuoti, todėl reikia analizuoti duomenis. Atributams, kurie nėra pirminiai identifikatoriai ir yra privalomi, patikrinti:

- Surasti maksimalią ir minimalią skaitmeninių atributų reikšmę. Jei rasta reikšmė yra daug kartų didesnė (mažesnė) už antrą maksimalią (minimalią) reikšmę (pvz. > 100 kartų) ir nemaža įrašų dalis turi tą reikšmę (pvz. > 5%), tuomet atributą pažymėti kaip pretenduojantį į neprivalomus atributus, išsaugoti reikšmę.
- Ištirti, ar yra simbolinių atributų reikšmių su tarpo simboliu. Jei tokių reikšmių yra, tuomet atributą pažymėti kaip pretenduojantį į neprivalomus atributus, išsaugoti reikšmę.
- Surasti dažniausiai pasitaikančią simbolinių atributų reikšmę (nekreipiant dėmesio į didžiąsias ir mažąsias raides). Jei nemaža įrašų dalis turi tą reikšmę (pvz. > 5%), tuomet atributą pažymėti, kaip pretenduojantį į neprivalomus atributus, išsaugoti reikšmę.

- Išvardintų reikšmių srities suradimas. Daugelis atributų privalo naudoti iš anksto nustatytos reikšmių aibės reikšmes. Atributams, kurie nėra pirminiai identifikatoriai ar išoriniai raktai, nustatyti visas galimas reikšmes (nekreipiant dėmesio į mažąsias ir didžiąsias raides). Jei skirtingų reikšmių yra mažai (pvz. < 1%), tuomet išsisaugoti galimas atributo reikšmes.
- Reikšmių srities apribojimų suradimas. Dažnai DBVS deklaruojamos duomenų struktūrų reikšmių sritis yra labai skurdi. Leidžiamoms reikšmėms gana dažnai taikomi griežti apribojimai. Skaitmeniniams atributams, kurie nėra pirminiai identifikatoriai ar išoriniai raktai, surasti minimalią ir maksimalią reikšmes. Jei minimali ir maksimali reikšmės daugiau už 0, tuomet atributo leidžiamas reikšmių intervalas tik teigiami skaičiai (>0). Jei maksimali reikšmė teigiama, o minimali reikšmė lygi nuliui, tuomet leidžiamas reikšmių intervalas apima teigiamus skaičius ir nulį (>=0). Jei minimali ir maksimali reikšmės mažiau už 0, tuomet atributo leidžiamas reikšmių intervalas tik neigiami skaičiai (<0). Jei minimali reikšmė neigiama, o maksimali reikšmė lygi nuliui, tuomet leidžiamas reikšmių intervalas apima neigiamus skaičius ir nulį (<=0). Išsisaugoti atributo reikšmių intervalą.

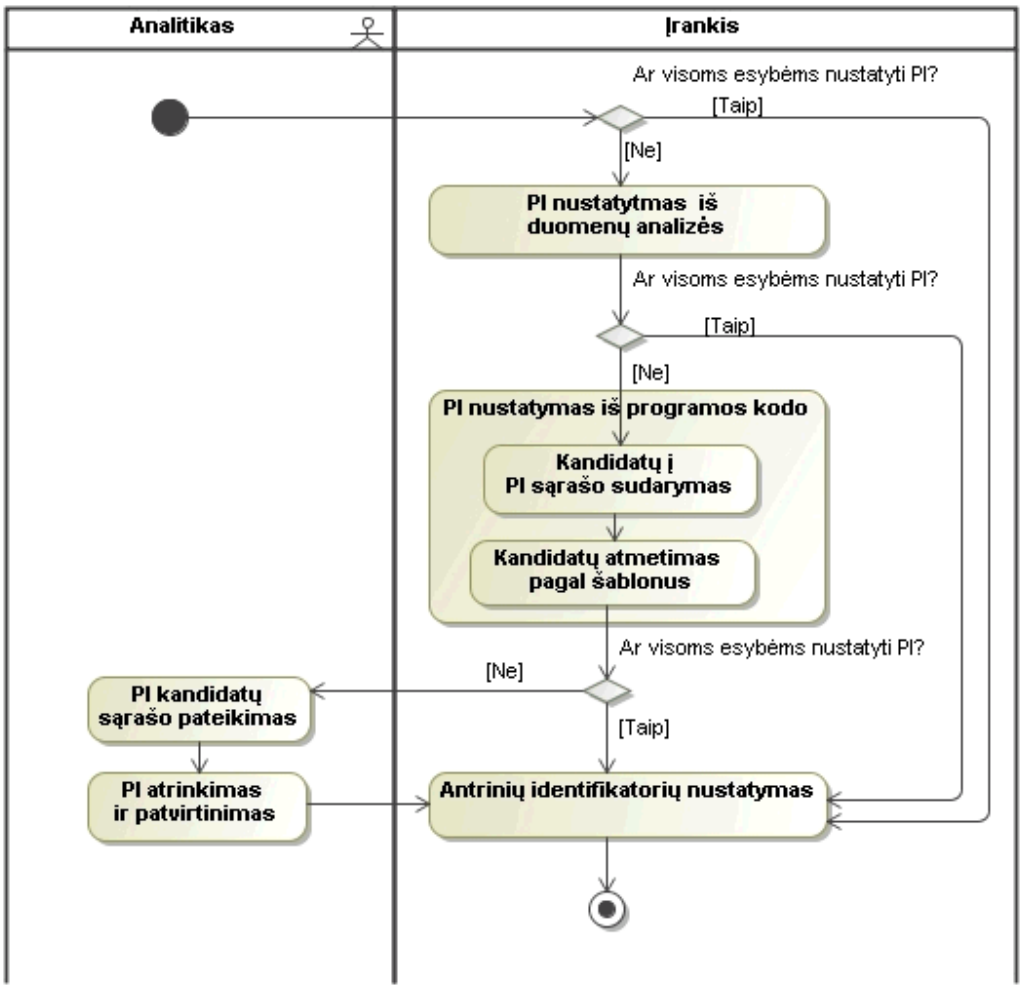
Jei yra atributų, pretenduojančių į neprivalomus atributus, tuomet pateikti šių atributų sąrašą su galimomis neprivalomomis reikšmėmis, kad vartotojas patvirtintų neprivalomus atributus.



3.3 pav. Neišreikštųjų apribojimų atributams nustatymas

3.1.4. Neišreikštųjų identifikatorių nustatymas

Etapo tikslas yra iš liktinių šaltinių išgauti pirminius identifikatorius toms lentelėms, kurių nepavyko išgauti duomenų žodyno išgavimo etapu. Šio žingsnio detalės pateiktos (3.4 pav.)



3.4 pav. Neišreikštųjų identifikatorių nustatymas

Raktinių atributų nustatymo algoritmas: kiekvienai esybei pagal priėjimo raktus pirmiausia identifikuojama pretenduojančių atributų (privalomi ir unikalūs) aibė. Jei yra tik vienas pretenduojantis raktas esybei, tuomet tai ir yra pirminis identifikatorius. ASM reikia ieškoti atmetimo (angl. *rule-out*) šablonų ir pašalinti atmetimo šablone esančius atributus iš pretendentų rinkinio. Kai programuotojas tikisi pasirinkti kortežų rinkinį (angl. *set of tuples*), tuomet programos kode naudoja SQL užklausas, kurias galima panaudoti kaip atmetimo šablonus. Atsižvelgiant į

pirminio identifikatoriaus apibrėžimą, šios taisyklės atmeta galimybę, kad atributai $a_1 \ldots a_n$ sudarytų pirminį identifikatorių. Trys pavyzdiniai atmetimo šablonai yra:

```
SELECT DISTINCT <rinkinys>
      FROM LENTELĖ
      WHERE a1=<išraiška1> AND a2=<išraiška2> AND … AND an=<išraiškan>

SELECT <rinkinys>
      FROM LENTELĖ
      WHERE a1=<išraiška1> AND a2=<išraiška2> AND … AND an=<išraiškan>
      GROUP BY ...

SELECT <rinkinys>
      FROM LENTELĖ
      WHERE a1=<išraiška1> AND a2=<išraiška2> AND … AND an=<išraiškan>
      ORDER BY ...
```

Jei pirminis identifikatorius vis dar nenustatytas, tuomet pateikti sumažintą pretenduojančių raktų rinkinį, kad vartotojas pasirinktų pirminius raktus.

Pagal priėjimo raktus, kurie atvaizduoja ne pirminius identifikatorius ir yra unikalūs, sudaryti antrinių identifikatorių sąrašą (pavadinimas, esybė, atributai).

3.1.5. Apjungtų esybių nustatymas

Savybių apjungimas nustatomas analizuojant priminių identifikatorių reikšmių sritį ir programos kode ieškant charakteringų DML išraiškų. Esybių apjungimas gali būti realizuotas tokiais būdais.

Jei apjungtos esybės identifikatorius yra sudarytas iš kelių atributų, tuomet viena dalis tų atributų atliks esybės tipo diskriminatorių rolę, o kita dalis bus esybės egzemplioriaus identifikatoriai konkrečiame esybės tipe. Atributų-diskriminatorių reikšmių aibė privalo būti baigtinė, t.y. išvardintų reikšmių aibė.

Programos kode, DML išraiškose atributams-diskriminatoriams palyginimo sąlygos arba jungimo sąlygos turi būti suformuotos naudojant konstantas ir konkrečias reikšmes, o ne kintamuosius. Išraiškos turi būti tokio pavidalo (da – diskriminuojantys atributai, ia – identifikuojantys atributai, ira – išorinio rakto atributai, da + ia sudaro pilną identifikatorių):

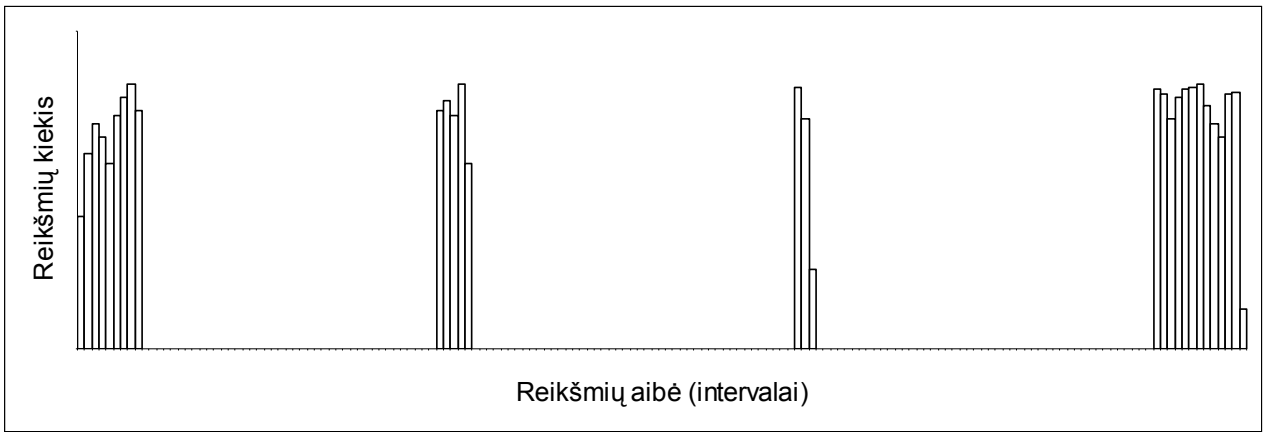
```
SELECT <rininkys>
      FROM lent1
      WHERE da1=<konstanta> AND … AND dan=<konstanta>
        AND ia1=<kintamasis> AND … AND ian=<kintamasis>

SELECT <rininkys>
      FROM lent1, lent2
```

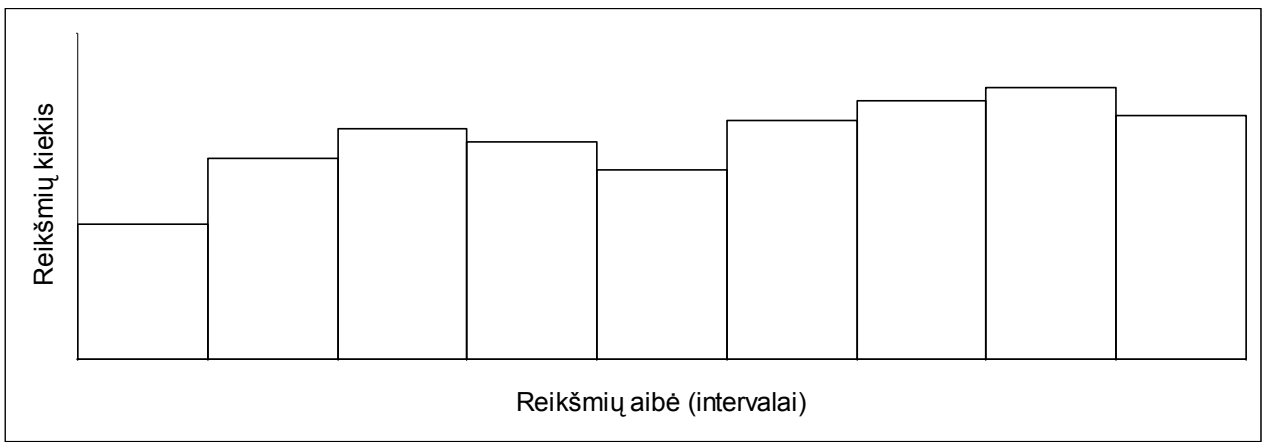
```
      WHERE lent1.da1=<konstanta> AND … AND lent1.dan=<konstanta>
        AND lent1.ia1=lent2.ira1 AND … AND lent1.ian=lent.iran

SELECT <rininkys>
      FROM lent2
      JOIN lent1 ON lent2.ira1=lent1.ia1,… , lent2.iran=lent1.ian
      WHERE lent1.pr1=<konstanta> AND … AND lent1.prn=<konstanta>
```

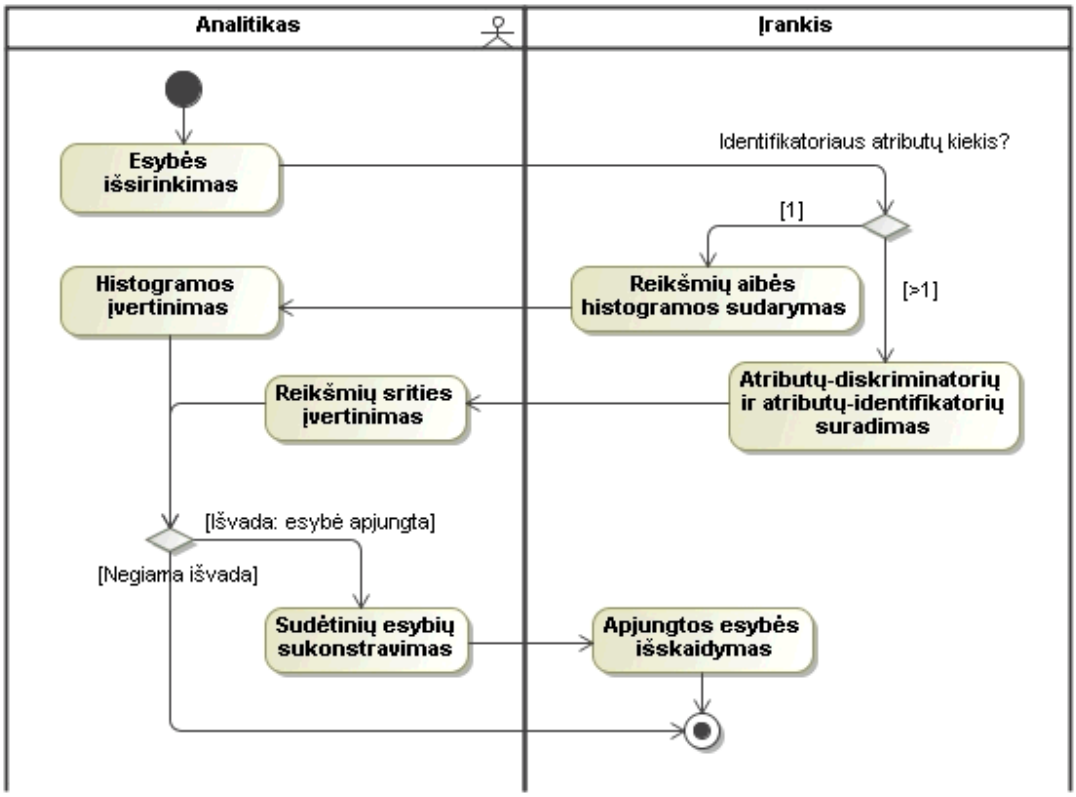
Jei apjungtos esybės pirminis identifikatorius yra vienas atributas, tuomet esybių tipų išskyrimas gali būti realizuotas kiekvienam esybių tipui suteikiant atskirą identifikatoriaus reikšmių sritį. Taipogi vieno identifikatorius atveju, atributas gali turėti sudėtinę paslėptą struktūrą. Taip realizuoto esybių apjungimo nustatymui, reikia atlikti sudėtingesnę reikšmių aibės analizę sudarant reikšmių aibės histogramą. Ši analizė parodo ar reikšmės yra išsibarščiusios visame reikšmių intervale (3.6 pav.), ar atskiruose nutolusiuose intervaluose (3.5 pav.).



3.5 pav. Sujungtoms esybėms būdinga identifikatoriaus reikšmių histograma



3.6 pav. Identifikatoriaus reikšmių histograma be sujungtų esybių požymio



3.7 pav. Apjungtų esybių nustatymas

3.1.6. Neišreikštųjų ryšio ir lygybės apribojimų nustatymas

Etapo tikslas yra nustatyti apribojimus, kurie padėtų klasifikuoti išgautas esybes, atvaizduojančias tiek realaus pasaulio esybes, tiek ir ryšius tarp jų. Kad realizuoti šį tikslą, naudojami ryšio ir lygybės apribojimai, parodantys tarpryšinių apribojimų egzistavimą, tarp jų ir klasės/poklasio ryšius.

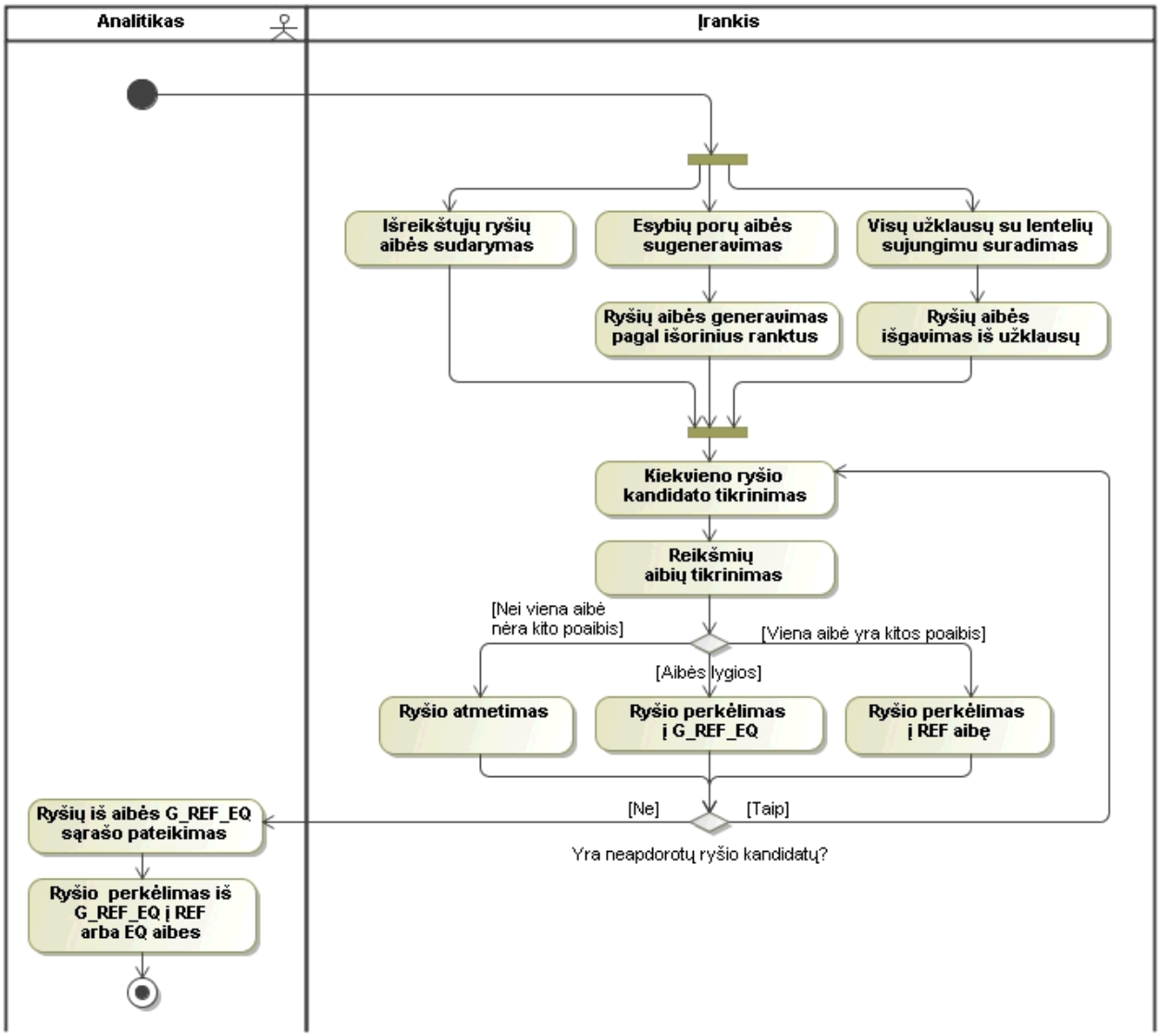
Ryšio apribojimai yra apibrėžiami tokiu būdu. Tegul `R1` ir `R2` yra 2 esybės, o `A` ir `B` yra atributai arba atitinkamai `R1` ir `R2` atributų aibė. Ryšio apribojimas `R1.A<<R2.B` reiškia, kad `R1.A` reikšmių aibė yra `R2.B` poaibis (`R2.B` privalo būti pirminis arba antrinis identifikatorius). Ryšio apribojimus galima nustatyti ištyrus visus įmanomus poaibio ryšius tarp bet kokių dviejų liktinių šaltinių esybių `R1` ir `R2`. Šiems apribojimams atrasti naudojama nuodugni paieška: kiekvienai liktinių šaltinių schemos esybių porai `R1` ir `R2` reikia palyginti kiekvienos neraktinių atributų kombinacijos `A` iš `R1` reikšmes su kiekvienos pirminio (arba antrinio) identifikatoriaus kombinacijos `B` iš `R2` reikšmėmis (pažymėtina, kad `A` ir `B` gali būti atskiri atributai). Ryšio apribojimas `R1.A<<R2.B` gali egzistuoti, jei:

1. `A` ir `B` apima tą patį atributų skaičių.
2. `A` ir `B` apima tą pačią dalykinę sritį (tas pats duomenų tipas ir toks pat maksimalus atributų ilgis).
3. `R1.A⊆R2.B`.

Jei papildomai egzistuoja poaibio priklausomybė `R2.B⊆R1.A`, tuomet gali egzistuoti lygybės apribojimas.

Kaip parodyta, yra trys ryšio ir lygybės apribojimų aptikimo informacijos šaltiniai: papildyta fizinė schema pateikia esybes, atributus, pirminius ir antrinius identifikatorius bei išorinius raktus (jei prieinami), lygių sujungimų (angl. *equi-join*) užklausų ieškojimo žingsnis ištiria ASM ir pateikia programos kodo lygių sujungimų užklausose kartu pasitaikančių esybių ir atitinkamų atributų poras (kadangi lygių sujungimo operacijoje yra naudojamos dvi esybės, tai akivaizdu, kad tarp jų yra ryšio arba lygybės apribojimas), poaibių priklausomybių testavimas atlieka liktinės duomenų bazės poaibio testus. Ryšio ir lygybės apribojimų aptikimo algoritmas yra toks:

1. Sukurti esybių porų aibę `X`: pvz. jei yra esybės `P`, `Q`, `R` ir `S`, tada `X={(P,Q), (P,R), (P,S), (Q,R), (Q,S), (R,S)}`. Ši aibė apima esybių poras, kurioms dar nenustatyti ryšio arba lygybės apribojimai. Be to, palaikomos galimų (`G_REF`, `G_EQU`) ir galutinių (`REF`, `EQU`) apribojimų aibės (jos iš pradžių tuščios).



3.8 pav. Neišreikštųjų ryšio ir lygybės apribojimų nustatymas

2. Jei išoriniai raktai buvo sėkmingai išgauti, tuomet kiekvienam išorinio rakto apribojimui atlikti:
    a. Nustatyti dalyvaujančių esybių porą, t.y. esybę, kuriai išorinis raktas priklauso, ir esybę, į kurią raktas nurodo.
    b. Įvykdyti duomenų užklausą ir nustatyti poaibio priklausomybės kryptį. Poaibio priklausomybės kryptis tikrinama naudojant B priede aprašytą poaibio testą.
    c. Pašalinti nustatytą porą iš x aibės.

d. Pridėti šį išorinį raktą apimantį ryšio arba lygybės apribojimą atitinkamai į `REF` arba `EQU` aibę.

3. Jei iš kodo išgautos lygių sujungimų užklausos, tuomet kiekvienai lygių sujungimų užklausai atlikti:
    a. Nustatyti dalyvaujančių esybių poras;
    b. Įvykdyti duomenų užklausą ir nustatyti poaibio priklausomybės kryptį. Poaibio priklausomybės kryptis tikrinama naudojant B priede aprašytą poaibio testą.
    c. Jei išgautas ryšio apribojimas, tada nustatytą porą pašalinti iš X aibės ir ryšio apribojimą pridėti į REF aibę.
    d. Jei išgautas lygybės apribojimas, tada pridėti lygybės apribojimą į G_EQU aibę ir abu galimus ryšio apribojimus į G_REF aibę.
4. Kiekvienai `X` aibėje paliekančiai porai `p` nustatyti tą patį duomenų tipą turinčius atributus ar atributų kombinacijas. Patikrinti poaibio priklausomybės egzistavimą, naudojant B priede aprašytą poaibio testą. Jei priklausomybė egzistuoja, tada pridėti ryšio ar lygybės apribojimus į atitinkamai `G_REF` arba `G_EQU` aibę. Jei po 4 žingsnio, jokie apribojimai nėra patalpinti į pretenduojančių apribojimų aibes, tuomet `p` pašalinti iš `X`; priešingu atveju, `p` palikti aibėje `X`, kad vartotojas patikrintų.
5. Kiekvienam `G_REF` ir `G_EQU` aibių apribojimui atlikti:
    a. Jei atributų pavadinimai abiejose pusėse yra vienodi, tada priskirti aukštą reitingą.
    b. Jei apribojimų kairės pusės atributo pavadinimas yra susijęs (pagrįsta paprastomis poeilutėmis) su dešinės pusės esybės pavadinimu, tada priskirti aukštą reitingą.
    c. Jei nei viena sąlyga netenkinama, tada priskirti žemą reitingą.
6. Kad vartotojas galutinai nuspręstų, kiekvienai `X` porai pateikti įjungimo priklausomybes ir jų `G_REF` bei `G_EQU` aibių reitingus. Priklausomai nuo vartotojo pasirinkimo, priskirti galiojančius apribojimus prie `REF`, `EQU` aibių.

Siekiant sumažinti poaibio priklausomybėms tarp atributų patikrinti reikalingų duomenų bazės užklausų skaičių (4 žingsnis), gali būti naudojami papildomi veiksmai, t.y. duomenų tipų ir maksimalaus atributų ilgio (pvz. `varchar(5)`) patikrinimas.

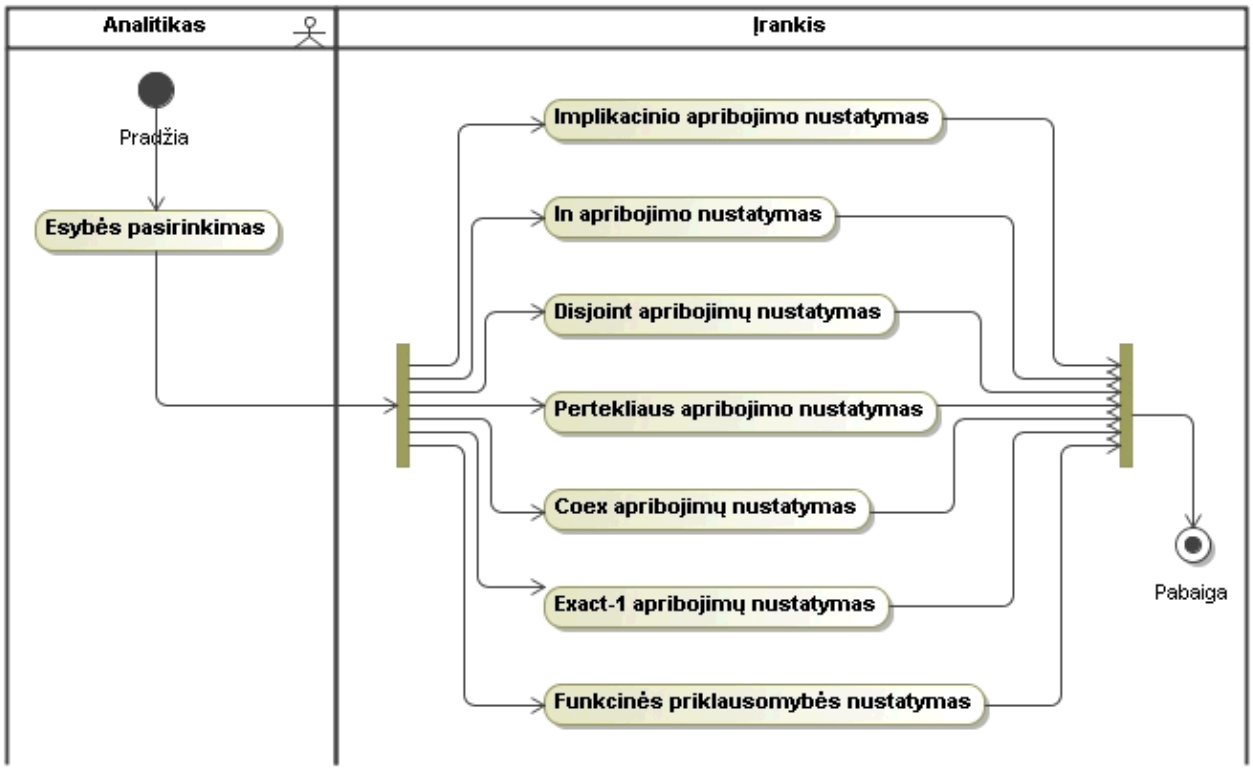
3.1.7. Kitų neišreiktųjų apribojimų ir priklausomybių nustatymas

Etapo tikslas – išgauti netiesiogines struktūras t.y. apribojimus ir priklausomybes. Siekiant atrasti šiuos apribojimus ir priklausomybes, vykdomos liktinės duomenų bazės užklausos:

- Apribojimas *coex* nurodo, kad esybės tipo atributai turi būti nustatyti kartu arba visai nenustatyti. Kad nustatyti, ar egzistuoja apribojimas tarp esybės `R` atributų `A`, `B` ir `D`, reikia įvykdyti duomenų užklausą kiekvienos esybės kiekvienai neprivalomų atributų kombinacijai. *coex* apribojimas tikrinamas naudojant B priede aprašytą *coex* apribojimo užklausą.
- Apribojimas *exact-1* nurodo, kad iš esybės tipo atributų turi būti nustatytas vienas ir tik vienas atributas. Kad nustatyti, ar egzistuoja apribojimas tarp esybės `R` atributų `A`, `B` ir `D`, reikia įvykdyti duomenų užklausą kiekvienos esybės kiekvienai neprivalomų atributų kombinacijai. *exact-1* apribojimas tikrinamas naudojant B priede aprašytą *exact-1* apribojimo užklausą.
- Apribojimas *disjoint(R1.A1,R2.A1)* nurodo, kad esybių tipų pirminiai raktai nepersikerta. Kad nustatyti, ar egzistuoja apribojimas tarp esybės `R1` pirminio identifikatoriaus `A1` ir esybės `R2` pirminio identifikatoriaus `A1`, reikia įvykdyti duomenų užklausą kiekvienai esybių pirminių identifikatorių porai. *disjoint* apribojimas tikrinamas naudojant B priede aprašytą *disjoint* apribojimo užklausą. *disjoint* apribojimus reikia apjungti, pvz. jei egzistuoja apribojimai *disjoint(R1.A1,R2.A1)*, *disjoint(R1.A1,R3.A1)*, *disjoint(R2.A1,R3.A1)*, tuomet reikia apjungti į *disjoint(R1.A1,R2.A1,R3.A1)*.
- Funkcinė priklausomybė *R1∪R2:A1-->A2* nurodo, kad jei esybių tipų pirminiai raktai sutampa, tuomet sutampa ir jų bendri atributai. Kad nustatyti, ar egzistuoja funkcinė priklausomybė tarp esybių `R1` ir `R2` pirminių identifikatorių `A1` ir esybių atributų `A2`, reikia įvykdyti duomenų užklausą kiekvienai esybių, kurios neturi *disjoint* apribojimo, pirminių identifikatorių ir esybių atributų porai. Funkcinė priklausomybė tikrinama naudojant B priede aprašytą funkcinės priklausomybės užklausą. Funkcines priklausomybes reikia apjungti, pvz. jei egzistuoja funkcinės priklausomybės *R1∪R2:A1-->A2*, *R1∪R3:A1-->A2* arba *R1∪R2:A1-->A2*,

R2∪R3:A1-->A2 arba *R1∪R3:A1-->A2*, *R2∪R3:A1-->A2*, tuomet reikia apjungti į *R1∪R2∪R3:A1-->A2*.

- Apribojimas *R1.A1in(R2.A1∪R3.A1)* nurodo, kad bendros esybės pirminis raktas sutampa su bent vieno esybės tipo pirminiu raktu. Kad nustatyti, ar egzistuoja apribojimas tarp esybės `R1` pirminio identifikatoriaus `A1` ir bendros esybių `R2` bei `R3` pirminių identifikatorių `A1` aibės, reikia įvykdyti duomenų užklausą kiekvienai esybių tipų (ne mažiau dviejų), kurių pirminis identifikatorius turi ir ryšio apribojimą, nurodantį į bendrą esybės tipą, pirminių identifikatorių ir bendro esybės tipo pirminio identifikatoriaus kombinacijai. *in* apribojimas tikrinamas naudojant B priede aprašytą *in* apribojimo užklausą.
- Apribojimas *R:BA1==>(BA1=A1)* nurodo, kad jei esybės tipo antrinis identifikatorius turi reikšmę, tai ji sutampa su pirminio identifikatoriaus reikšme. Kad nustatyti, ar egzistuoja apribojimas tarp esybės `R` pirminio identifikatoriaus `A1` ir antrinio identifikatoriaus `BA1`, reikia įvykdyti duomenų užklausą kiekvienos esybės kiekvienai privalomo pirminio identifikatoriaus ir neprivalomo antrinio identifikatoriaus porai. Implikacinis apribojimas tikrinamas naudojant B priede aprašytą implikacinio apribojimo užklausą.
- Pertekliaus apribojimas *rd* nurodo, kad esybės tipo atributas yra kitos esybės atributo kopija. Kad nustatyti, ar egzistuoja *rd* apribojimas tarp esybės `R1` su pirminiu identifikatoriumi `A1` ir `R2` esybės su išoriniu raktu `A1` atributų `A2`, reikia įvykdyti duomenų užklausą kiekvienai esybių, kurios susijusios ryšio apribojimu, ne pirminių identifikatorių ir ne ryšio apribojimų atributų porai. Pertekliaus apribojimas tikrinamas naudojant B priede aprašytą pertekliaus apribojimo užklausą.



3.9 pav. Kitų apribojimų nustatymas

7 ADI metodikos etapai iš liktinės informacijos sistemos leidžia išgauti išsamę loginę schemą, kurioje yra tokia informacija:

- Esybes (pavadinimai, aprašymas);
- Atributus (pavadinimas, duomenų tipas, maksimalus ilgis, reikšmė pagal nutylėjimą, aprašymas, semantinė prasmė);
- Atributų apribojimus (būtinumas, unikalumas, bendras egzistavimas);
- Pirminius ir antrinius identifikatorius;
- Ryšius (kardinalumas);

3.2. Kuriamo įrankio architektūros pasirinkimas

Įrankis realizuotas trijų lygių architektūroje. Griežtai išskirtos vartotojo sąsajos, apdorojimo ir duomenų klasės.

Siekiant turėti universalų ir nuo platformos nepriklausantį įrankį, realizacijai pasirinkta Java programavimo kalba. Ši kalba buvo pasirinkta, nes tai yra aukšto lygio, neutralios platformos, paprasta, pernešama *(angl. portable)* ir objektinė kalba. Leidžiant dauguma programinių kalbų sukurtas programas, jas reikia arba perkompiliuoti, arba interpretuoti iš naujo. Java šiuo atžvilgiu yra

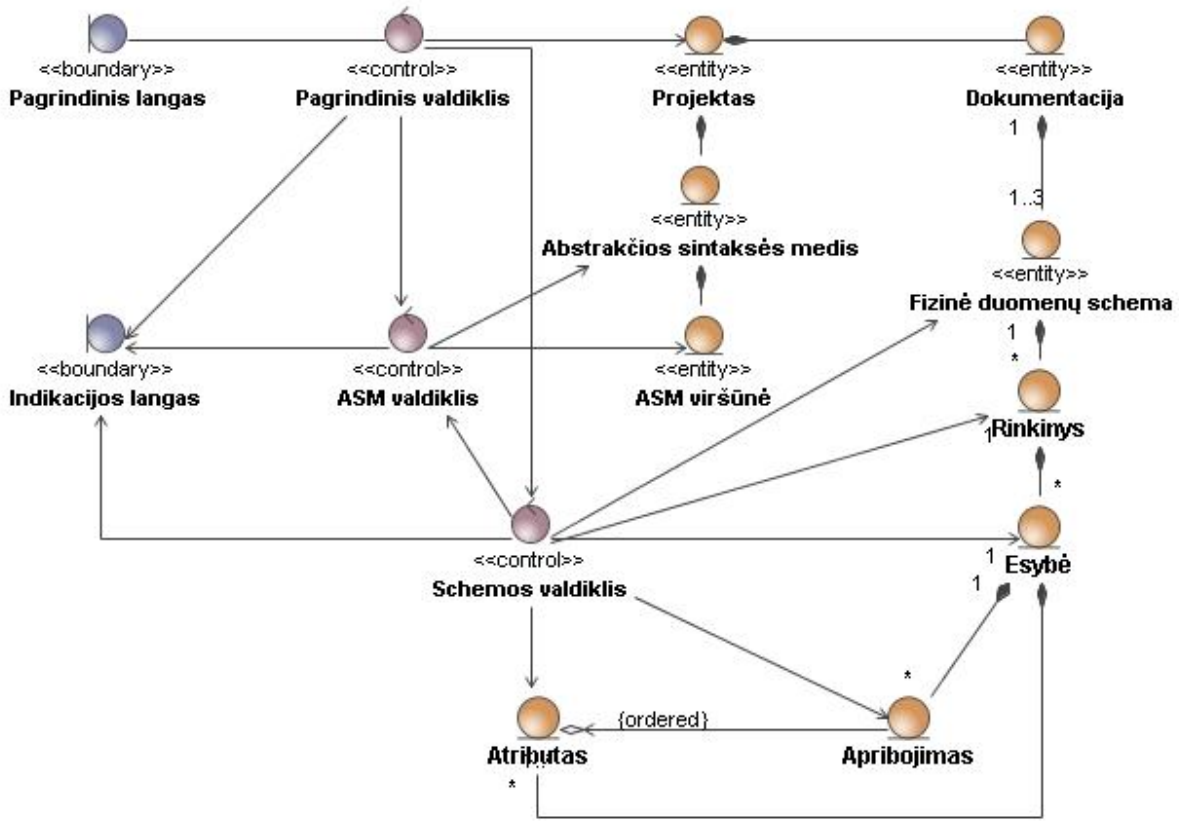
netradicinė, nes programa yra tuo pačiu metu yra ir kompiliuojama, ir interpretuojama. Kompiliatorius verčia programą į tarpinę kalbą, Javos baitų kodus *(angl. Java bytecodes)* – nuo platformos nepriklausomas kodo instrukcijas. Kiekvieną kartą leidžiant programą interpretatorius vykdo Javo baitų kodo instrukcijas.

Sąsajai su reliacine duomenų baze pasirinkta JDBC sąsaja. Abstrakčios sintaksės medžio generavimui bus naudojama JFlex leksinių analizatorių generatorius [31] ir CUP nagrinėtojo generatorius.

Tolimesnis išgautų modelių išsaugojimas šiame įrankyje nėra aktualus.

3.3. Reikalavimų analizė

Reikalavimų analizė pateikiama analizės klasių diagrama. Šioje diagramoje vaizduojamos ribinės klasės, valdikliai ir esybės, kurios sudarytos pagal reikalavimus sistemai. Diagrama atskiria sistemos dalis į langus, juos valdančius komponentus ir duomenų esybes. Sistema padalinta į tris dalis: vartotojo paslaugas, veiklos paslaugas ir duomenų paslaugas (3.10 pav.).



3.10 pav. Analizės klasių diagrama

4. REALIZACIJA IR EKSPERIMENTINIS METODIKOS TYRIMAS

Tikslai. Aprašyti ADI įrankio prototipo realizaciją, atlikti eksperimentinį metodikos tyrimą ir atlikti metodikos įvertinimą lyginant sukurtos metodikos savybes su kitų nagrinėtų metodų savybėmis ir atlikti kokybinį metodikos įvertinimą apklausiant ekspertus.

Metodai. Eksperimentui atlikti bus taikomi šie metodai: natūralus eksperimentas ir laboratorinis tyrimas. Realizavimui Eclipse platforma.

Skyriaus struktūra:

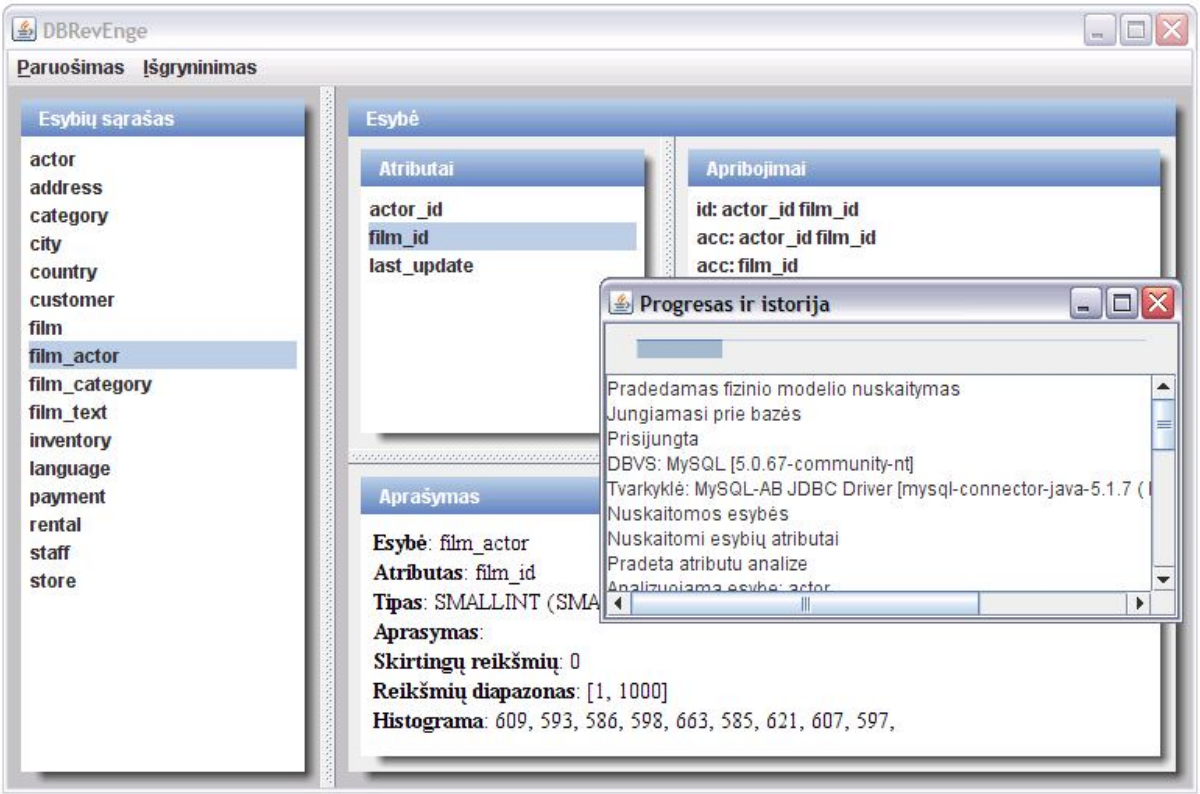
4.1 skyriuje aprašyta realizacija.

4.2 skyriuje pateikiamas kontrolinis pavyzdys, su kuriuo atlikome eksperimentinį ADI metodikos tyrimą.

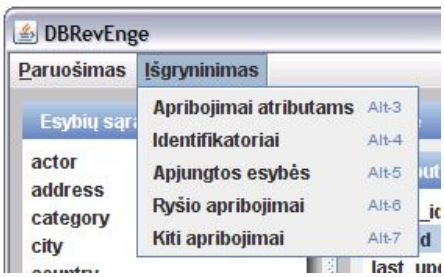
4.3 skyriuje pateikiamas ADI metodikos įvertinimas lyginant metodiką su kitais metodais ir kokybinis metodikos įvertinimas apklausiant ekspertus.

4.1. Realizacijos aprašymas

Buvo sukurtas elementarios vartotojo sąsajos sandaros eksperimentinis įrankio prototipas. Sąsaja sudaryta iš lango, kuriame matomi esybių sąrašas. Pažymėjus esybę matomi jos atributai ir apribojimai. Pažymėjus atributą arba apribojimą, informacija apie jį pateikiama aprašymo lange (4.1 pav.). Visi žingsniai atliekami tiesiogiai iš pagrindinio meniu (4.2 pav.). Žingsnio vykdymo progresas ir istorija matomi atskirame lange.



4.1 pav. Bendras įrankio vaizdas



4.2 pav. Žingsnių vykdymas įrankyje

4.2. Kontrolinis pavyzdys

Šiame skyriuje pateikiame atliktą eksperimentinį ADI metodikos veiksmingumo tyrimą. Siekiant paaiškinti kiekvieną iš ADI metodikoje pateiktų žingsnių ir jų susijusias veiklas, naudojamas trivialus liktinės sistemos pavyzdys. Tarkime, kad yra liktinė duomenų bazė DB_L, kurią valdo reliacinių duomenų bazių valdymo sistema. Kad būtų paprasčiau nagrinėti šį pavyzdį, tarkime, kad egzistuoja šie DB_L ryšiai:

```
KLIENTAS [KID, PAVADINIMAS, ADR_MIESTAS, ADR_GATVĖ, ADR_NAMO_NR, ADR_BUTAS]
TELEFONAS [KID, TELEFONAS]
```

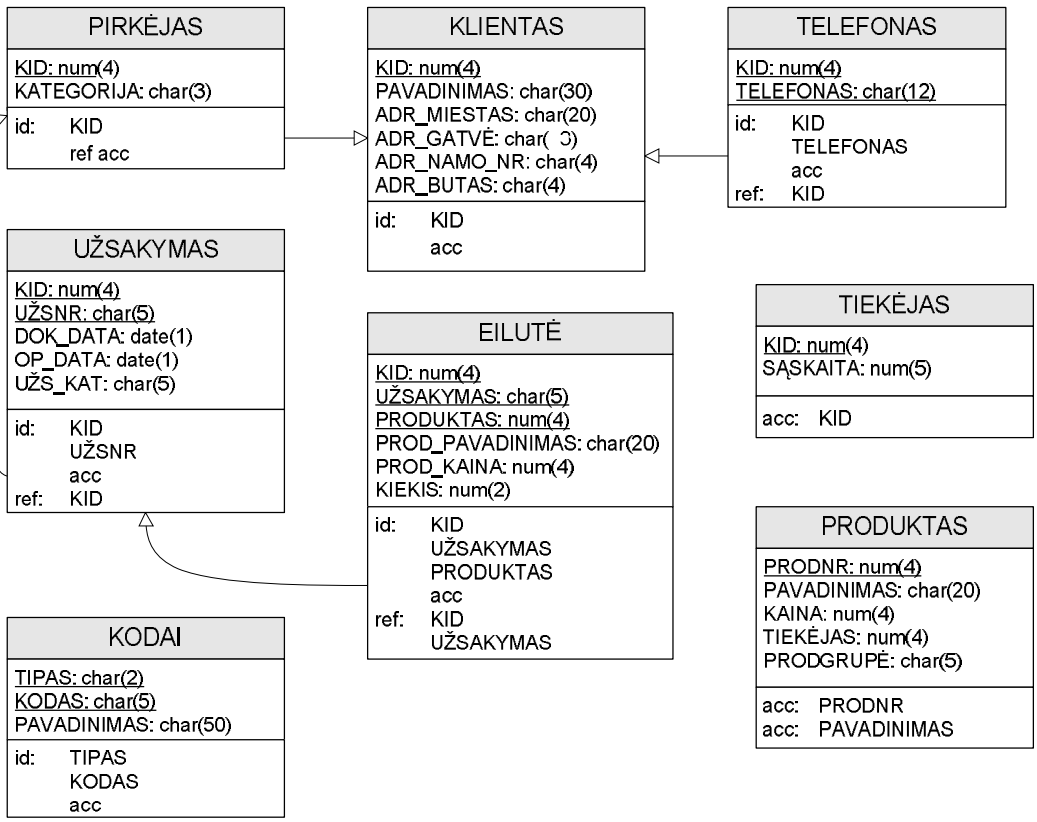
```
TIEKĖJAS [KID, SĄSKAITA]
PRODUKTAS [PRODNR, PAVADINIMAS, KAINA, TIEKĖJAS]
UŽSAKYMAS [KID, UŽSNR, DOK_DATA, OP_DATA]
KODAI [TIPAS, KODAS, TEKSTAS]
EILUTĖ [KID, UŽSAKYMAS, PRODUKTAS, PROD_PAVADINIMAS, PROD_KAINA, KIEKIS]
```

Siekiant paaiškinti kodo analizę ir parodyti, kaip ji pagerina duomenų struktūros išgavimą, naudojamas C kodo fragmentas, kuris atspindi paprastą, hipotetinę sąveiką su liktine duomenų baze (pateikti tik pavyzdžiui aktualūs sakiniai):

```
char *aReiksme, *cReiksme;
int bReiksme = 100;
..........
/* daugiau kodo */
..........
EXEC SQL SELECT DOK DATA, OP DATA INTO :aReiksme,:cReiksme
FROM UŽSAKYMAS WHERE UŽSNR = :bReiksme;
..........
/* daugiau kodo */
..........
if (*cReiksme < *aReiksme)
{ cReiksme = aReiksme; }
..........
/* daugiau kodo */
..........
printf("Dokumento data %s", aReiksme);
printf("Operacijos data %s", cReiksme);
```

4.2.1. Duomenų žodyno išgavimas

Iš sistemos duomenų žodyno išgaunama duomenų bazės struktūra ir savybės: rinkiniai, esybės, atributai, pirminiai identifikatoriai, išoriniai raktai, priėjimo raktai (4.3 pav.).

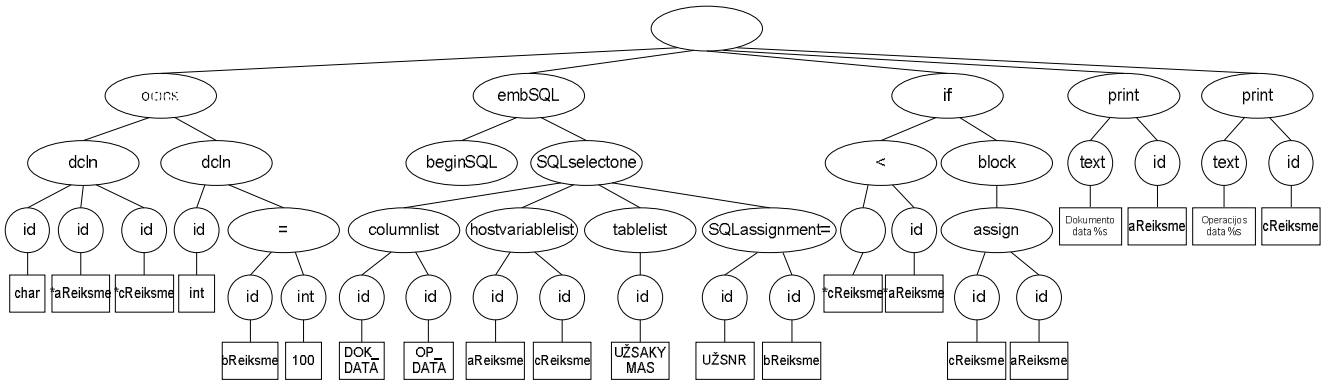


4.3 pav. Neapdirbta fizinė schema

Toliau yra atrandamos ir išreiškiamos struktūros bei apribojimai, kurios buvo netiesiogiai realizuotos arba atmestos vystant programą. Yra daug netiesioginių struktūrų, pagrindinės iš jų: pirminiai ir antriniai identifikatoriai, ryšio, lygybės ir kt. apribojimai, funkcinės priklausomybės, prasmingi pavadinimai.

4.2.2. Abstrakčios sintaksės medžio sudarymas

Iš liktinės programos kodo sugeneruojamas abstrakčios sintaksės medis (4.4 pav.).



4.4 pav. Liktinės sistemos abstrakčios sintaksės medis

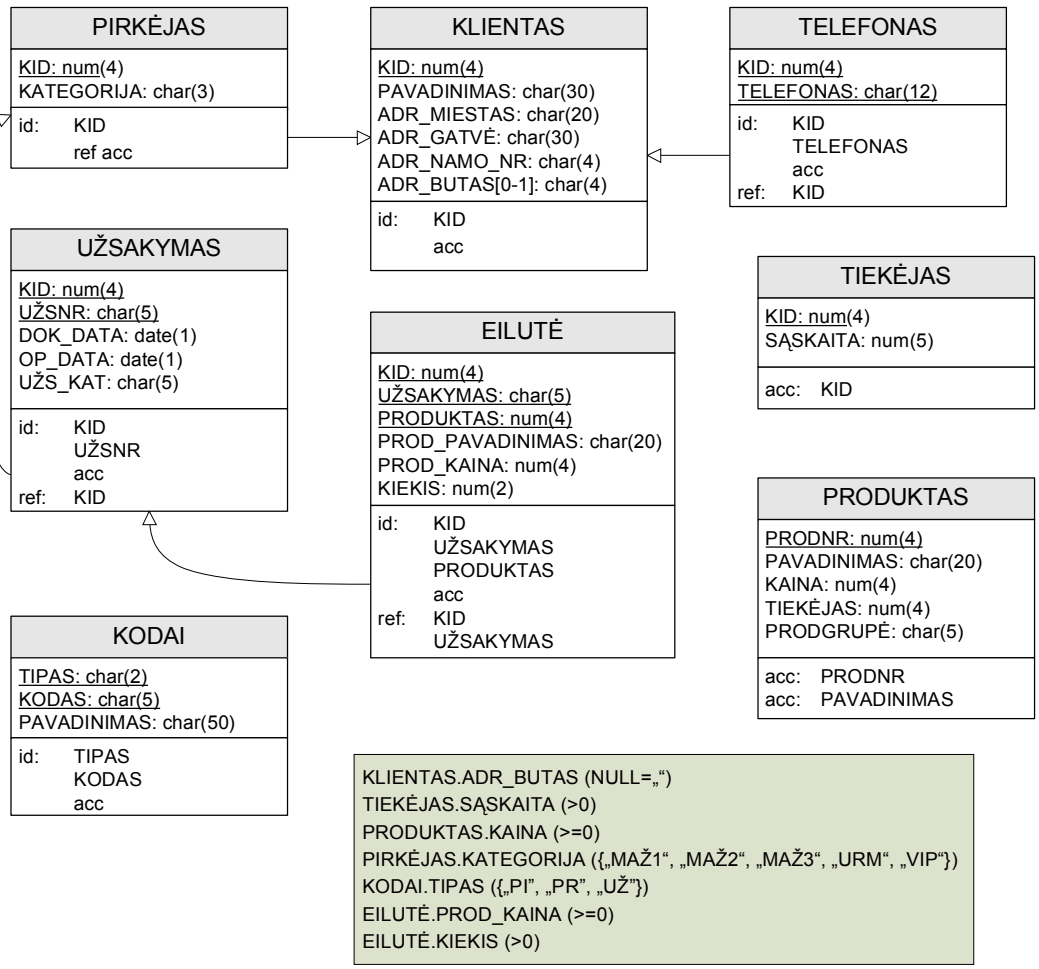
4.2.3. Neišreikštųjų apribojimų atributams nustatymas

Iš liktinių šaltinių duomenų bazės duomenų rinkinio išgaunamos struktūros ir apribojimai, kurių nepavyko atrasti duomenų žodyno išgavimo etapu:

- Neprivalomų atributų suradimas:
  - Nustatomos maksimalios ir minimalios skaitmeninių atributų reikšmės. **Rezultatas**: reikšmės nėra daug kartų didesnės (mažesnės) už antras maksimalias (minimalias) reikšmes. Skaitmeninių atributų, pretenduojančių į neprivalomus atributus, nenustatyta.
  - Ištiriama, ar yra simbolinių atributų reikšmių su tarpo simboliu. **Rezultatas**: atributas „*KLIENTAS.ADR_BUTAS*“ turi reikšmių su tarpo simboliu. Atributą pažymėti kaip pretenduojantį į neprivalomus atributus, išsaugoti reikšmę („ “).
  - Nustatomos dažniausiai pasitaikančios simbolinių atributų reikšmės. **Rezultatas**: atributo „*PIRKĖJAS.KATEGORIJA*“ dažniausiai pasitaikanti kategorija „*URM*“, ją turi 40% įrašų. Atributą pažymėti, kaip pretenduojantį į neprivalomus atributus, išsaugoti reikšmę („*URM*“).
- Išvardintų reikšmių srities suradimas. **Rezultatas**:
  - atributas „*PIRKĖJAS.KATEGORIJA*“ turi mažai galimų skirtingų reikšmių. Išsaugoti galimas atributo reikšmes {„*MAŽ1*“, „*MAŽ2*“, „*MAŽ3*“, „*URM*“, „*VIP*“}.
  - atributas „*KODAI.TIPAS*“ turi mažai galimų skirtingų reikšmių. Išsaugoti galimas atributo reikšmes {„PI“, „PR“, „UŽ“}.
- Reikšmių srities apribojimų suradimas. **Rezultatas**:

- Atributo „*TIEKĖJAS.SĄSKAITA*“ minimali ir maksimali reikšmės daugiau už 0, todėl atributo leidžiamas reikšmių intervalas tik teigiami skaičiai (>0).
- Atributo „*PRODUKTAS.KAINA*“ minimali reikšmė lygi nuliui, o maksimali reikšmė daugiau už 0, todėl atributo leidžiamas reikšmių intervalas apima teigiamus skaičius ir nulį (>=0).
- Atributo „*EILUTĖ.PROD_KAINA*“ minimali reikšmė lygi nuliui, o maksimali reikšmė daugiau už 0, todėl atributo leidžiamas reikšmių intervalas apima teigiamus skaičius ir nulį (>=0).
- Atributo „*EILUTĖ.KIEKIS*“ minimali ir maksimali reikšmės daugiau už 0, atributo leidžiamas reikšmių intervalas tik teigiami skaičiai (>0).

Analitikui pateikiamas atributų, pretenduojančių į neprivalomus atributus, sąrašas. Analitikas patvirtina, kad „*KLIENTAS.ADR_BUTAS*“ yra neprivalomas atributas, ir atmeta, kad „*PIRKĖJAS.KATEGORIJA*“ yra neprivalomas atributas (4.5 pav.).



4.5 pav. Atributų apribojimais papildyta fizinė schema

4.2.4. Neišreikštųjų identifikatorių nustatymas

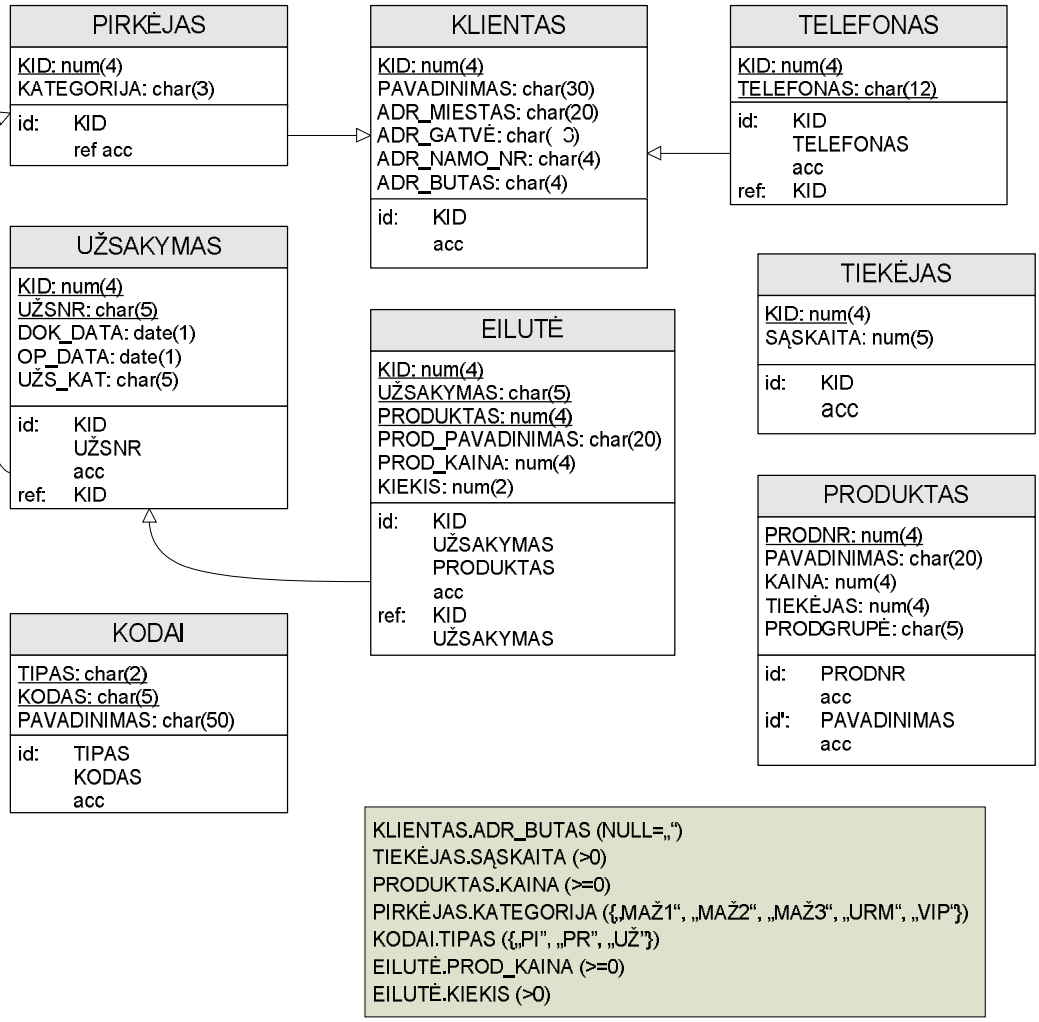
Iš liktinių šaltinių išgaunami pirminiai identifikatoriai toms lentelėms, kurių nepavyko išgauti duomenų žodyno išgavimo etapu:

- Esybė „*TIEKĖJAS*“ turi tik vieną pretenduojantį atributą (privalomas ir unikalus), todėl „*TIEKĖJAS.KID*“ yra pirminis identifikatorius.
- Esybė „*PRODUKTAS*“ turi du pretenduojančius atributus „*PRODUKTAS.PRODNR*“ ir „*PRODUKTAS.PAVADINIMAS*“. Šablonų atmetimas neaptiko nė vieno iš šių atributų tarp atmetimo šablonų, pirminį identifikatorių turi pasirinkti vartotojas.

Analitikui pateikiamas esybės „*PRODUKTAS*“ pretenduojančių atributų sąrašas. Analitikas pasirenka, kad „*PRODUKTAS.PRODNR*“ yra esybės „*PRODUKTAS*“ pirminis identifikatorius.

Esybės „*PRODUKTAS*“ atributas „*PRODUKTAS.PAVADINIMAS*“ nėra pirminis identifikatorius, tačiau yra unikalus. Šis atributas yra esybės „*PRODUKTAS*“ antrinis identifikatorius.

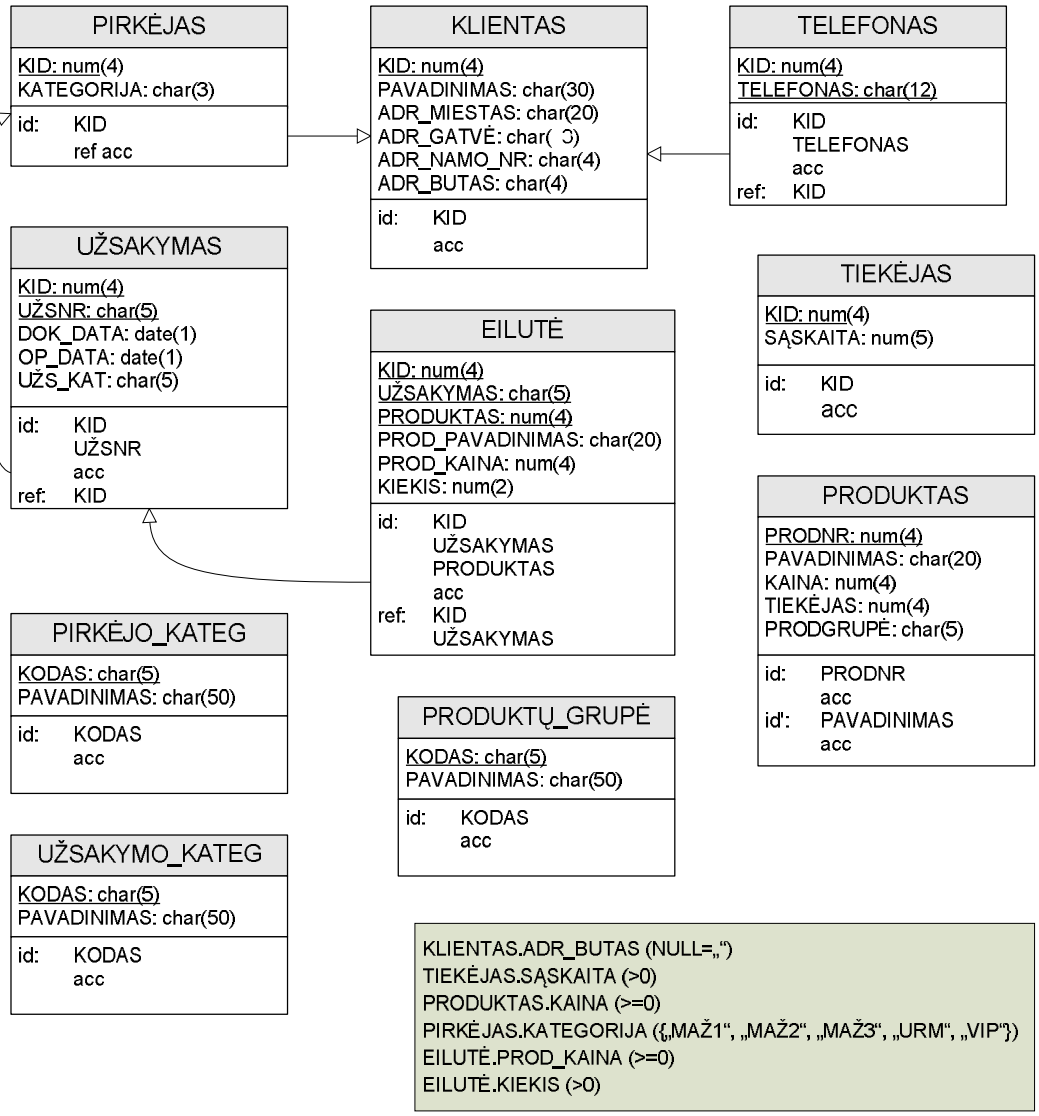
Po šio etapo pirminių ir identifikatorių aibė nustatoma galutinai (4.6 pav.).



4.6 pav. Identifikatoriais papildyta fizinė schema

4.2.5. Apjungtų esybių nustatymas

Analizuojant liktinius šaltinius nustatoma, jog esybė KODAI yra apjungta iš kelių esybių. Nustatoma, jog atributas TIPAS yra diskriminatorius, kuris atskiria kokiai esybei priklauso konkretus įrašas. Pagal diskriminatoriaus reikšmių aibę sudaromos 3 naujos esybės: PIRKĖJO_KATEG, UŽSAKYMO_KATEG, PRODUKTŲ_GRUPĖ. Naujose esybės paveldi visą struktūrą išskyrus atributą-diskriminatorių (4.7 pav.).



4.7 pav. Fizinė schema po apjungtos esybės išskyrimo

4.2.6. Neišreikštųjų ryšio ir lygybės apribojimų nustatymas

Nustatomi ryšio ir lygybės apribojimai, kurie padeda klasifikuoti realaus pasaulio esybes:

1. Sukuriama esybių porų aibė
    a. X = {(KLIENTAS, PIRKĖJAS), (KLIENTAS, TIEKĖJAS), (KLIENTAS, UŽSAKYMAS), (KLIENTAS, PRODUKTAS), (KLIENTAS, TELEFONAS), (KLIENTAS, PIRKĖJO_KATEG), (KLIENTAS, UŽSAKYMO_KATEG), (KLIENTAS, PRODUKTŲ_GRUPĖ), (PIRKĖJAS, TIEKĖJAS), (PIRKĖJAS, UŽSAKYMAS), (PIRKĖJAS, PRODUKTAS), (PIRKĖJAS, TELEFONAS), (PIRKĖJAS, PIRKĖJO_KATEG), (PIRKĖJAS,

UŽSAKYMO_KATEG), (PIRKĖJAS, PRODUKTŲ_GRUPĖ), (TIEKĖJAS, UŽSAKYMAS), (TIEKĖJAS, PRODUKTAS), (TIEKĖJAS, TELEFONAS), (TIEKĖJAS, PIRKĖJO_KATEG), (TIEKĖJAS, UŽSAKYMO_KATEG), (TIEKĖJAS, PRODUKTŲ_GRUPĖ), (UŽSAKYMAS, PRODUKTAS), (UŽSAKYMAS, TELEFONAS), (UŽSAKYMAS, PIRKĖJO_KATEG), (UŽSAKYMAS, UŽSAKYMO_KATEG), (UŽSAKYMAS, PRODUKTŲ_GRUPĖ), (PRODUKTAS, TELEFONAS), (PRODUKTAS, PIRKĖJO_KATEG), (PRODUKTAS, UŽSAKYMO_KATEG), (PRODUKTAS, PRODUKTŲ_GRUPĖ), (TELEFONAS, PIRKĖJO_KATEG), (TELEFONAS, UŽSAKYMO_KATEG), (TELEFONAS, PRODUKTŲ_GRUPĖ), (PIRKĖJO_KATEG, UŽSAKYMO_KATEG), (PIRKĖJO_KATEG, PRODUKTŲ_GRUPĖ), (UŽSAKYMO_KATEG, PRODUKTŲ_GRUPĖ)}.

b. Apribojimų aibės: G_REF ={}, G_EQU={}, REF={}, EQU={}.

2. Jei išoriniai raktai prieinami, tuomet ištiriami ryšiai, kuriems tie raktai priklauso arba į kuriuos nurodo, ir nustatomi ryšio bei lygybės apribojimai:
   a. *REF = {TELEFONAS.KID << KLIENTAS.KID, PIRKĖJAS.KID << KLIENTAS.KID, UŽSAKYMAS.KID << PIRKĖJAS.KID},*
   b. *EQU={ EILUTĖ.KID, EILUTĖ.UŽSAKYMAS << UŽSAKYMAS.KID, UŽSAKYMAS.NR}*.
   c. Sumažinama esybių porų aibė *X = {(KLIENTAS, TIEKĖJAS), (KLIENTAS, PRODUKTAS), (KLIENTAS, EILUTĖ), (KLIENTAS, UŽSAKYMAS), (TELEFONAS, TIEKĖJAS), (TELEFONAS, PRODUKTAS), (TELEFONAS, EILUTĖ), (TELEFONAS, UŽSAKYMAS), (TELEFONAS, PIRKĖJAS), (TIEKĖJAS, PRODUKTAS), (TIEKĖJAS, EILUTĖ), (TIEKĖJAS, UŽSAKYMAS), (TIEKĖJAS, PIRKĖJAS), (PRODUKTAS, EILUTĖ), (PRODUKTAS, UŽSAKYMAS), (PRODUKTAS, PIRKĖJAS), (EILUTĖ, PIRKĖJAS), (KLIENTAS, PIRKĖJO_KATEG), (PIRKĖJAS, PIRKĖJO_KATEG), (TIEKĖJAS, PIRKĖJO_KATEG), (UŽSAKYMAS, PIRKĖJO_KATEG), (PRODUKTAS, PIRKĖJO_KATEG), (TELEFONAS, PIRKĖJO_KATEG), (KLIENTAS, UŽSAKYMO_KATEG), (PIRKĖJAS, UŽSAKYMO_KATEG), (TIEKĖJAS, UŽSAKYMO_KATEG), (UŽSAKYMAS,*

UŽSAKYMO_KATEG), (PRODUKTAS, UŽSAKYMO_KATEG), (TELEFONAS, UŽSAKYMO_KATEG), (PIRKĖJO_KATEG, UŽSAKYMO_KATEG), (KLIENTAS, PRODUKTŲ_GRUPĖ), (PIRKĖJAS, PRODUKTŲ_GRUPĖ), (TIEKĖJAS, PRODUKTŲ_GRUPĖ), (UŽSAKYMAS, PRODUKTŲ_GRUPĖ), (PRODUKTAS, PRODUKTŲ_GRUPĖ), (TELEFONAS, PRODUKTŲ_GRUPĖ), (PIRKĖJO_KATEG, PRODUKTŲ_GRUPĖ), (UŽSAKYMO_KATEG, PRODUKTŲ_GRUPĖ)}.

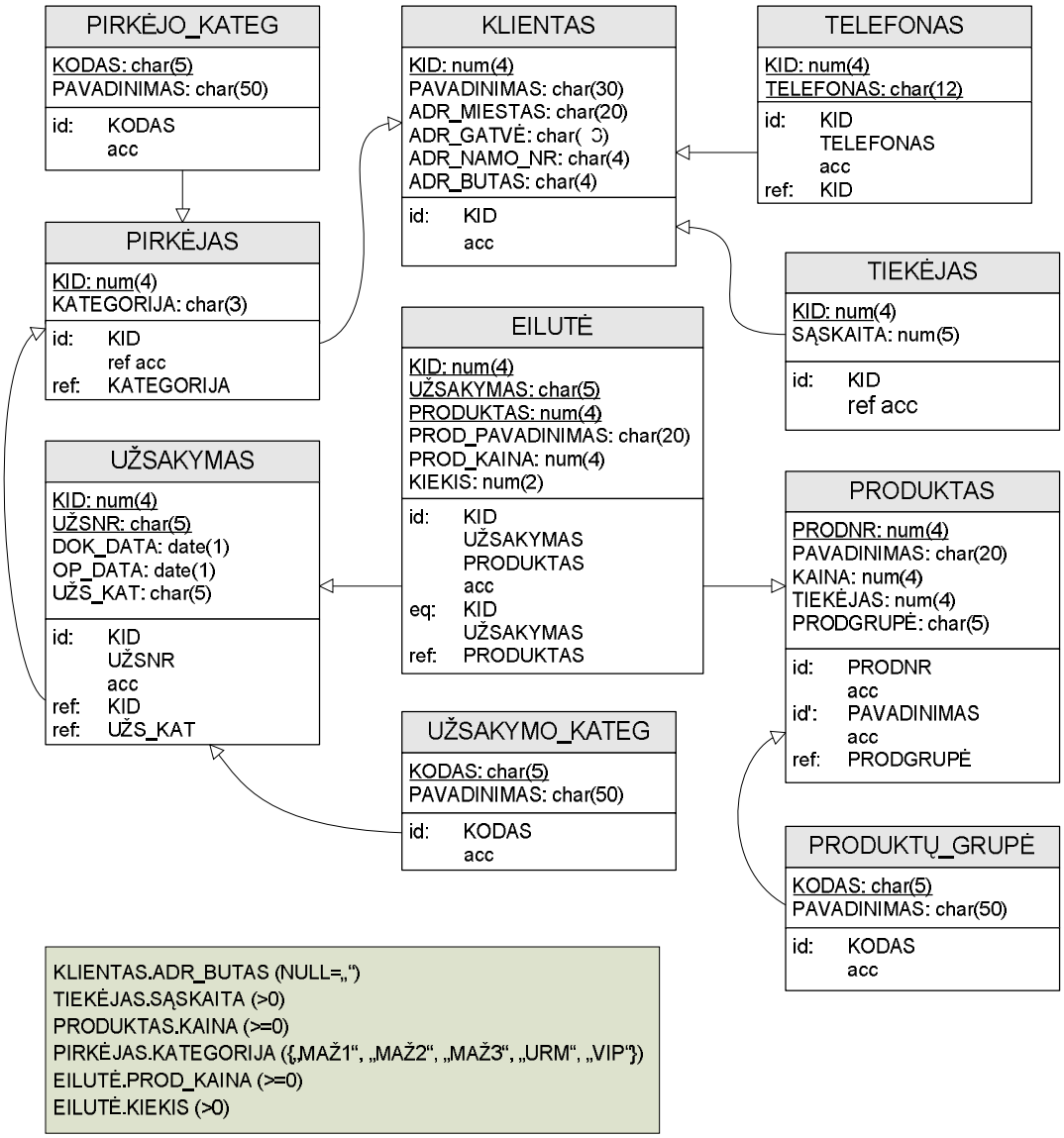
3. Nustatoma, ar lygių sujungimų užklausos įterptos į programą, ir papildomi egzistuojantys ryšio ir lygybės apribojimai:
    a. REF = {TELEFONAS.KID << KLIENTAS.KID, PIRKĖJAS.KID << KLIENTAS.KID, UŽSAKYMAS.KID << PIRKĖJAS.KID, **TIEKĖJAS.KID << KLIENTAS.KID, UŽSAKYMAS.UŽS_KAT << UŽSAKYMO_KATEG.KODAS, PRODUKTAS.PRODGRUPĖ << PRODUKTŲ_GRUPĖ.KODAS, PIRKĖJAS.KATEGORIJA << PRIKĖJO_KATEG.KODAS**},
    b. EQU = {EILUTĖ.KID, EILUTĖ.UŽSAKYMAS << UŽSAKYMAS.KID, UŽSAKYMAS.NR}.
    c. Sumažinama esybių porų aibė X = {(KLIENTAS, PRODUKTAS), (KLIENTAS, EILUTĖ), (KLIENTAS, UŽSAKYMAS), (TELEFONAS, TIEKĖJAS), (TELEFONAS, PRODUKTAS), (TELEFONAS, EILUTĖ), (TELEFONAS, UŽSAKYMAS), (TELEFONAS, PIRKĖJAS), (TIEKĖJAS, PRODUKTAS), (TIEKĖJAS, EILUTĖ), (TIEKĖJAS, UŽSAKYMAS), (TIEKĖJAS, PIRKĖJAS), (PRODUKTAS, EILUTĖ), (PRODUKTAS, UŽSAKYMAS), (PRODUKTAS, PIRKĖJAS), (EILUTĖ, PIRKĖJAS), (KLIENTAS, PIRKĖJO_KATEG), (TIEKĖJAS, PIRKĖJO_KATEG), (UŽSAKYMAS, PIRKĖJO_KATEG), (PRODUKTAS, PIRKĖJO_KATEG), (TELEFONAS, PIRKĖJO_KATEG), (KLIENTAS, UŽSAKYMO_KATEG), (PIRKĖJAS, UŽSAKYMO_KATEG), (TIEKĖJAS, UŽSAKYMO_KATEG), (PRODUKTAS, UŽSAKYMO_KATEG), (TELEFONAS, UŽSAKYMO_KATEG), (PIRKĖJO_KATEG, UŽSAKYMO_KATEG), (KLIENTAS, PRODUKTŲ_GRUPĖ), (PIRKĖJAS, PRODUKTŲ_GRUPĖ), (TIEKĖJAS, PRODUKTŲ_GRUPĖ), (UŽSAKYMAS,

PRODUKTŲ_GRUPĖ), (TELEFONAS, PRODUKTŲ_GRUPĖ), (PIRKĖJO_KATEG, PRODUKTŲ_GRUPĖ), (UŽSAKYMO_KATEG, PRODUKTŲ_GRUPĖ)}.

4. Ištestuojami visi įmanomi apribojimai tarp vis dar nesusietų ryšių porų ir nustatomi galimi ryšio ir lygybės apribojimai:
    a. G_REF = {PRODUKTAS.TIEKĖJAS << KLIENTAS.KID, EILUTĖ.KID << KLIENTAS.KID, UŽSAKYMAS.KID << KLIENTAS.KID, PRODUKTAS.TIEKĖJAS << TIEKĖJAS.KID EILUTĖ.PRODUKTAS << PRODUKTAS.PRODNR, EILUTĖ.KID << PIRKĖJAS.KID},
    b. G_EQU={}.
5. Priskiriami galimų ryšio ir lygybės aibių apribojimų reitingai
    a. G_REF = {PRODUKTAS.TIEKĖJAS<<KLIENTAS.KID - Ž, EILUTĖ.KID<<KLIENTAS.KID - A, UŽSAKYMAS.KID<<KLIENTAS.KID - A, PRODUKTAS.TIEKĖJAS<<TIEKĖJAS.KID – A, EILUTĖ.PRODUKTAS<<PRODUKTAS.PRODNR - A, EILUTĖ.KID<<PIRKĖJAS.KID - A},
    b. G_EQU={}.
6. Analitikui pateikiamos galimų apribojimų aibės. Analitikas patvirtina, kad visi galimi apribojimai teisingi. Papildomos ryšio ir lygybės apribojimų aibės:
    a. *REF={TELEFONAS.KID<<KLIENTAS.KID, PIRKĖJAS.KID<<KLIENTAS.KID, UŽSAKYMAS.KID<<PIRKĖJAS.KID, TIEKĖJAS.KID<<KLIENTAS.KID,* ***PRODUKTAS.TIEKĖJAS<<TIEKĖJAS.KID,*** ***EILUTĖ.PRODUKTAS<<PRODUKTAS.PRODNR****},*
    b. *EQU={ EILUTĖ.KID, EILUTĖ.UŽSAKYMAS<<UŽSAKYMAS.KID, UŽSAKYMAS.NR}*.

Apribojimus *PRODUKTAS.TIEKĖJAS<<KLIENTAS.KID, EILUTĖ.KID<<KLIENTAS.KID, UŽSAKYMAS.KID<<KLIENTAS.KID, EILUTĖ.KID<<PIRKĖJAS.KID* galima pašalinti, kadangi kiti apribojimai per tranzityvumo ryšį juos netiesiogiai apima.

Po šio etapo ryšių ir lygybės apribojimų aibė nustatoma galutinai (4.8 pav.).



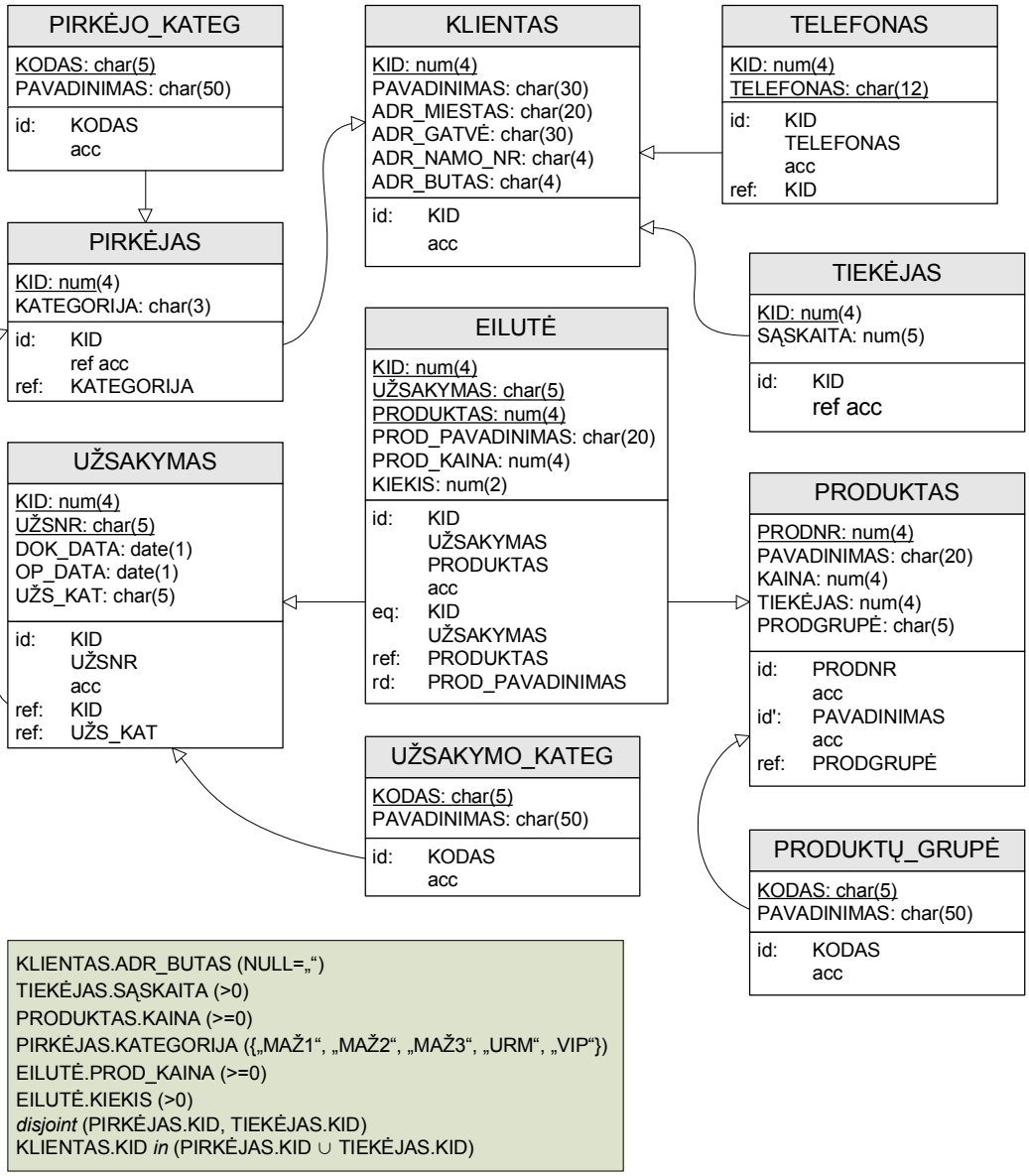
4.8 pav. Ryšio apribojimais papildyta fizinė schema

4.2.7. Kitų neišreikštųjų apribojimų ir priklausomybių nustatymas

Iš liktinių šaltinių duomenų bazės duomenų rinkinio išgaunami likę apribojimai ir priklausomybės. Egzistuoja šie apribojimai:

- *disjoint(PIRKĖJAS.KID, TIEKĖJAS.KID)*.
- *KLIENTAS.KIDin(PIRKĖJAS.KID∪TIEKĖJAS.KID)*.
- *rd(EILUTĖ.PROD_PAVADINIMAS<<PRODUKTAS.PAVADINIMAS)*.

Po šio etapo sudaroma išbaigta loginė schema (4.9 pav.).



4.9 pav. Išbaigta loginė schema

4.3. Sudarytos ADI metodikos įvertinimas

Šiame skyriuje pateikiamas ADI metodikos įvertinimas lyginant metodiką su kitais metodais ir pateikiamas atliktas kokybinis metodikos įvertinimas apklausiant ekspertus.

4.3.1. ADI metodikos įvertinimas pagal išgaunamas struktūras ir apribojimus

Sudarytą ADI metodiką įvertinome, palygindami sukurtos metodikos savybes su kitų nagrinėtų metodų savybėmis. Sudaryta palyginamoji metodų vertinimo lentelė. Šioje lentelėje

metodai yra lyginami pagal analizės dalyje apibrėžtas išgaunamas struktūras ir apribojimus ir iš kokio šaltinio ši informacija išgaunama.

Metodikų vertinamoji lentelė, kurioje pavaizduota sukurtos ADI metodikos ir esamų metodų palyginimas pagal skirtingų struktūrų ir apribojimų išgavimą, pateikta 4.1 lentelėje.

4.1 lentelė. Sudarytos ADI metodikos palyginimas su esamais ADI metodais pagal išgaunamas struktūras ir apribojimus

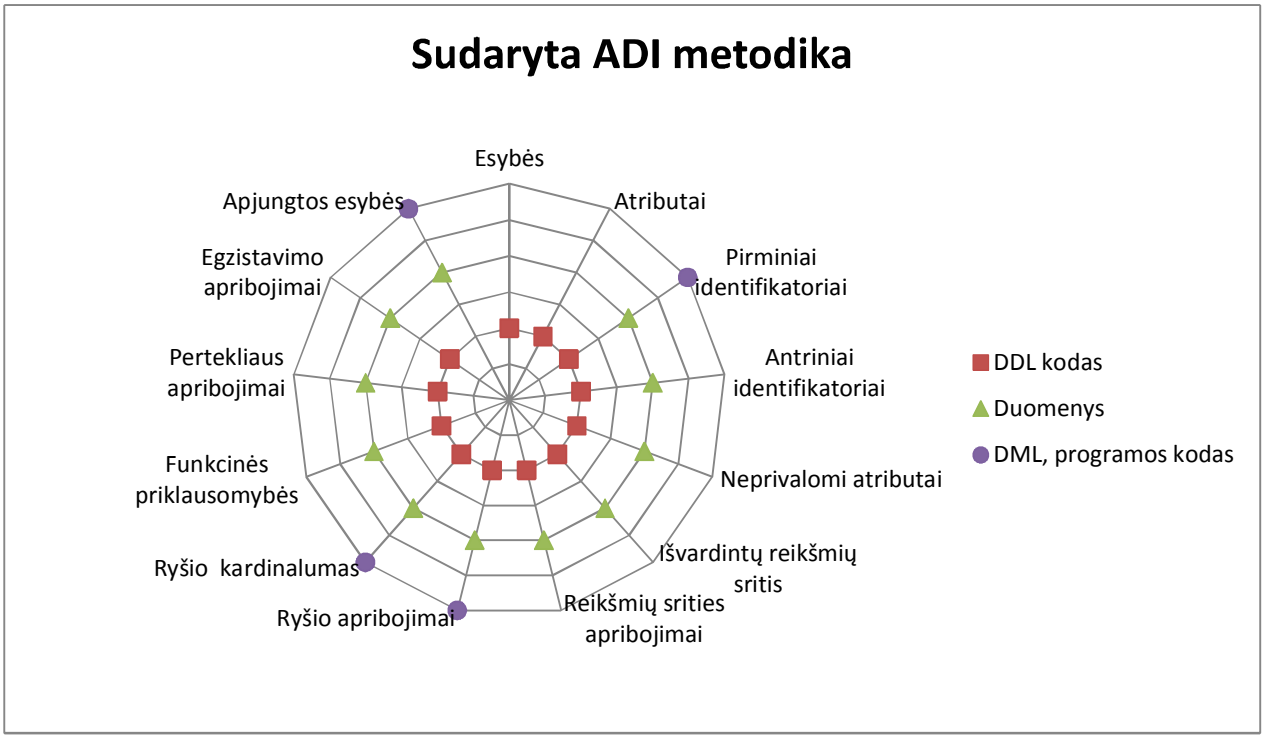
Eil. Nr.	Metodo pavadinimas [autorius -metai]	Išgaunamos struktūros ir apribojimai												
		Esybės	Atributai	Pirminiai identifikatoriai	Antriniai identifikatoriai	Neprivalomi atributai	Išvardintų reikšmių sritis	Reikšmių srities apribojimai	Ryšio apribojimai	Ryšio kardinalumas	Funkcinės priklausomybės	Pertekliaus apribojimai	Egzistavimo apribojimai	Apjungtos esybės
1	DB-MAIN [Hainaut -1991]	+	+	+	+	+	+	+	+	+	+	+	+	–
2	[Premerlani - 1993]	1	1	1,2	1,2	–	–	–	1,2	–	–	–	–	–
3	[Signore - 1994]	1	1	1,3	–	–	–	–	1,3	–	–	–	–	–
4	[Chiang - 1995]	1	1	1	–	–	–	–	1,2	–	–	–	–	–
5	[Petit - 1996]	1	1	1	–	–	–	–	1,3	–	1,3	–	–	–
6	MeRCI [Comyn -1996]	1	1	1	–	–	–	1,2,3	1,2,3	1,2,3	–	–	–	–
7	Varlet [Jahnke -1999]	1	1	1	–	1,2	1,2	1,2,3	1,2,3	1,2,3	–	–	–	–
8	Sudaryta ADI metodika	1	1	1,2,3	1,2	1,2	1,2	1,2	1,2,3	1,2,3	1,2	1,2	1,2	2,3

Analizuojami šaltiniai:

1 – DDL kodas; 2 – duomenys; 3 – DML, programos kodas; „+“ – visi įmanomi šaltiniai, „–“ – neišgauna

Iš 4.1 lentelės matome, kad sudaryta ADI metodika išgauna visas analizuojamas struktūras ir apribojimus. Šie apribojimai ir struktūros buvo sudarytos pagal tai kokios struktūros ir apribojimai reikalingi, kad rezultate butų išauta išsami loginė schema ir skirtos naujai sukurtai metodikai įvertinti.

Matome, kad sudaryta metodika analizuoja DDL kodą, duomenis ir DML programos kodą ir išgauna visas įmanomas struktūras ir apribojimus (4.10 pav.).



4.10 pav. Sudarytoje ADI metodikoje išgaunamos struktūros ir apribojimai ir analizuojami šaltiniai

Apžvelgus ADI metodus, matome kad nėra nei vieno tokio metodo, kuris išgautų visas analizuojamas struktūras ir apribojimus. Iš visų metodų akivaizdžiai išsiskiria DB-MAIN. DB-MAIN tai labiau metodika nei metodas, kadangi tai labai išsamus, sudėtingas visus šaltinius analizuojantis ir didžiąją dalį apribojimų išgaunantis metodas. Dėl savo sudėtingumo šio metodas naudojimui reikalingos gilios ADI ir DB teorijos žinios. Tai įvardintina kaip pagrindinis šio metodo trūkumas. Jis tinkamas dideliems ir sudėtingiems ADI projektams, tačiau netinkamas mažesniems projektas, kuriuos atlieka tik pagrindines ADI ir DB teorijos žinias turintys vykdytojai.

Kiti 4.1 lentelėje pateikti metodai yra gerokai siauresni ir labiau koncentruoti. Jie orientuoti į tam tikrų struktūrų ir apribojimų išgavimą ir/arba į tam tikrų šaltinių analizę.

Visi 4.1 lentelėje pateikti metodai išgauna duomenų bazėje deklaruotą struktūra ir neišreikštuosius ryšio apribojimus. Tik dalis jų atlieka neišreikštųjų apribojimų atributams paiešką.

Ir visgi nei vienas iš jų nemėgina atrasti esybių, kurios fizinėje schemoje yra apjungtos į viena lentelę.

4.3.2. ADI metodikos kokybinis įvertinimas apklausiant ekspertus

Apklausdami ekspertus sudarytą ADI metodiką įvertinome, pagal atvirkštinės duomenų inžinerijos kokybės kriterijus. Rezultatus pateikėme 4.2 lentelėje. Šioje lentelėje ADI metodika yra vertinama pagal atvirkštinei duomenų inžinerijai būdingus kokybės kriterijus:

- Teisingumas: schema yra sintaksiškai teisinga, jei visos schemos sąvokos yra tinkamai apibrėžtos. Schema yra semantiškai teisinga, jei visos jos sąvokos vartojamos pagal jų apibrėžimą.
- Primityvumas: schema yra primityvi, jei visos schemos struktūros priskirtos ne daugiau kaip vienai realaus pasaulio abstrakcijai.
- Minimalumas: schema yra minimali, jei negalima pašalinti nė vieno schemos elemento, nepraradus informacijos. Schemos, kurios nėra minimalios, dažnai yra daug sunkiau suprantamos.
- Tinkamumas: schema yra tinkama, jei visi jos objektai ir ryšiai yra aiškiai ir tiksliai atvaizduoti duomenų bazės schemoje, naudojant atitinkamo modeliavimo sąvokas.
- Normiškumas: duomenų bazės schema turi tenkinti tam tikrą norminę formą.

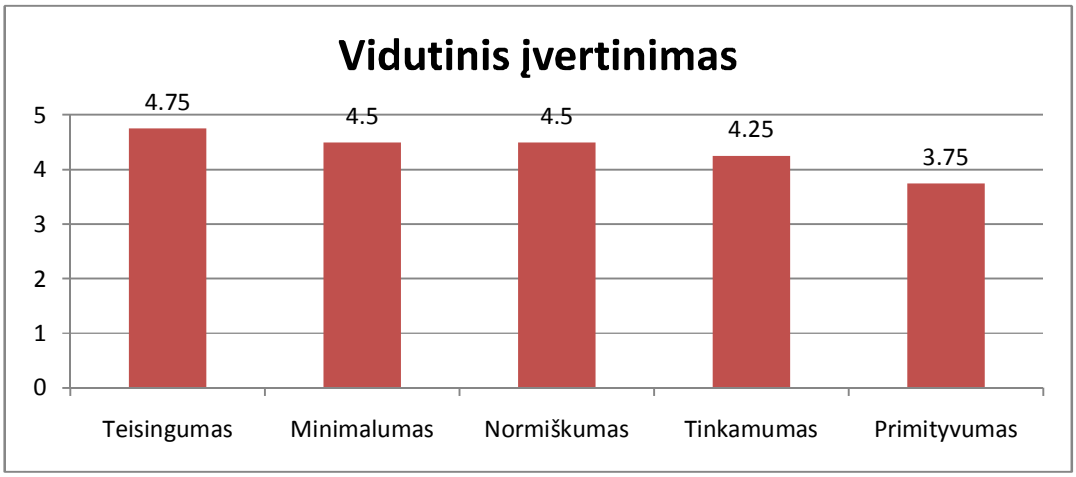
Svarbiausias kokybės kriterijus yra semantinis teisingumas. Minimalumo ir primityvumo kriterijai yra glaudžiai susiję su semantinės abstrakcijos lygiu, kadangi pertekliškumas ir paslėpti objektai mažina abstrakcijos lygį. Tinkamumo kriterijus svarbus tuo, kad leidžia visus ryšius ir objektus tiksliai aprašyti modeliavimo sąvokomis, o ne paslėptais ryšiais ar loginiu suvokimu.

ADI metodikos kokybiniame tyrime dalyvavo 4 ekspertai: 2 docentai, technikos mokslų daktarai; profesorius; doktorantas.

Kiekvienas kokybės kriterijus įvertintas 5-balėje sistemoje, kur 5 – geriausias įvertinimas, 1 – blogiausias. Apklausos rezultatai pateikti 4.2 lentelėje.

4.2 lentelė. ADI metodikos vertinimas pagal ADI kokybės kriterijus

Kriterijus	1 ekspertas	2 ekspertas	3 ekspertas	4 ekspertas	Vidutinis įvertinimas
Teisingumas	4	5	5	5	4,75
Primityvumas	3	4	4	4	3,75
Minimalumas	4	4	5	5	4,5
Tinkamumas	4	4	5	4	4,25
Normiškumas	4	5	4	5	4,5



4.11 pav. Kokybinio metodikos vertinimo ekspertų apklausos rezultatai

Galima teigti, kad bendrai ekspertai metodiką įvertino teigiamai. Nebuvo nei vieno įvertinimo mažesnio už vidutinį (3). Geriausiai įvertintas metodikos teisingumo kriterijus. Labai gerai įvertinti minimalumo, normiškumo ir tinkamumo kriterijai.

5. IŠVADOS

1. Apibendrinus analizės išvadas, pastebėta jog nėra metodų ir metodikų, kurios būtų orientuotos „į viduriuką“:
    - ADI analitikas turi bazines DB žinias ir gali įvertinti loginės schemos praktinį pritaikymą – ADI procesą naudoja duomenų semantikai atstatyti, kuri reikalinga duomenų migracijai atlikti;
    - duomenų migracijai aktualu atstatyti visus bazinius dalykine sritimi sąlygojamus duomenų apribojimus – formaliai specifikuotas ir neišreikštąsias duomenų struktūras, kurios būdingos liktinių IS nedokumentuotiems DB projektams;
    - po migracijos duomenų palaikymui naudojamos standartinės RDBVS.
2. Sudaryta ADI metodika, su kuria išgaunama išsami loginė schema, kurioje svarbiausios (bazinės) neišreikštosios struktūras išgaunamos iš fizinės DB ir DML kodo, pasitelkiant ekspertinį - patyriminį įvertinimo būdą.Šie šaltiniai pasirinkti todėl, kad DB duomenys yra visuomet prieinami ir lengvai analizuojami, o DML kodas – pats turtingiausias šaltinis.
3. Metodikoje įvestas žingsnis kuris neminimas nei viename ADI metode – apjungtų esybių aptikimas, kuriuo fiziniame modelyje identifikuojamos apjungtas esybes. Apjungtas esybės būdingos liktinėms duomenų bazėms, kai objektų identifikavimui naudojami agreguoti domenai su „paslėptomis“ klasifikavimo savybėmis; kurios nagrinėja ekspertai histograminiu būdu.
4. Naujų subesybių išskyrimas pasiūlytu histograminiu būdu atstatomos 1NF RDB schemos t.y. loginės schemos su jų egzempliorių atominėmis, nedalomomis reikšmėmis; atstatytos 1 NF schemos sudaro prielaidas pakartotinėje inžinerijoje sunorminti DB iki aukštesnių norminių formų.
5. Sudaryta ADI metodika išbandyta eksperimentiškai ir patvirtintas jos veiksmingumas. Ekspertai metodiką įvertino teigiamai.
6. Toliau darbas galėtų būti tobulinamas tokiomis kryptimis:
    - įtraukti papildomi neišreikštųjų apribojimų šaltiniai;
    - įvesta duomenų analizės tolerancija tam tikram klaidų kiekiui;
    - numatyta galimybė grįžti į tam tikrą žingsnį.

6. LITERATŪRA

[1] **P. Aiken, A. Muntz, R. Richards**. DoD Legacy Systems: Reverse Engineering Data Requirements. *Communications of the ACM*, vol. 37, p. 26-41, 1994.

[2] **J. Akoka, I. Comyn-Wattiau**. MeRCI: An Expert System for Software Reverse Engineering. *Proceedings of the 4th World Congress on Expert System*, 1998.

[3] **R. Alhajj, F. Polat**. Reengineering Relational Databases to Object-Oriented: Constructing the Class Hierarchy and Migrating the Data. *Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'2001)*, p. 335-344, 2001.

[4] **M. Anderson**. Reverse Engineering of Legacy Systems: From Value-Based to Object-Based Models. *PhD thesis*, *EPFL,* 1996.

[5] **G. Bakehouse ir T. Wakefield.** Legacy Information Systems. *ACCA services, 2005, prieiga per Internetą:* <http://www.accaglobal.com/>.

[6] **I. Baxter, M. Mehlich**. Reverse Engineering is Reverse Forward Engineering. *Proceedings of 4th Working Conference on Reverse Engineering (WCRE'97)*, 1997.

[7] **J. Bisbal, D. Lawless, B. Wu, J. Grimson**. Legacy Information System Migration: A Brief Review of Problems, Solutions and Research Issues. *Technical Report TCD-CS1999-38*, 1999, prieiga per Internetą: <https://www.cs.tcd.ie/>.

[8] **M. Blaha**. A Catalog of Object Model Transformations. *Proc. of the 3rd Working Conf. on Reverse Engineering (WCRE'96)*, 1996.

[9] **M. A. Casanova, J. E. A. de Sa**. Designing Entity-Relationship Schemas for Conventional Information Systems. *Third International Conference on Entity-Relationship Approach*, 1983.

[10] **R. H. Chiang**. A Knowledge-Based System for Performing Reverse Engineering of Relational Database. *Decision Support Systems*, 1995, 13, p. 295-312.

[11] **R. H. L. Chiang, T. M. Barron, V. C. Storey**. Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database. *Data and Knowledge Engineering*, 1994, 12, p. 107-142.

[12] **E. J. C. Chikofsky**. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, vol. 7, p. 13-17, 1990.

[13] **I. Comyn-Wattiau, J. Akoka**. Reverse Engineering of Relational Database Physical Schema. *Proceedings of the International Entity-Relationship Conference (ER'96)*, p. 372-391, 1996.

[14] **T. A. Corbi**. Program Understanding: Challenge for the 1990s. *IBM System Journal*, 28(2), 1989.

[15] **H. Dayani-Fard, I. Jurisica**. Reverse Engineering: A History - Where We've Been and What We've Done. *IEEE Fifth Working Conference on Reverse Engineering*, 1998.

[16] **K. H. Davis, P. Aiken**. Data Reverse Engineering: A Historical Survey. *IEEE Seventh Working Conference on Reverse Engineering*, 2000.

[17] **K. H. Davis, A. K. Arora**. Converting a Relational Database Model into an Entity-Relationship Model. *Sixth International Conference on Entity-Relationship Approach*, p. 271-285, 1987.

[18] **S. R. Dumpala, S. K. Arora**. Schema Translation using the Entity-Relationship Approach. *Proceedings of the International Conference on Entity-Relationship Approach (ER'83)*, p. 337-356, 1983.

[19] **M. Garcia-Solaco, F. Saltor, M. A. Castellanos**. Structure Based Schema Integration Methodology. *Proceedings of the 11th International Conference of Interoperable Database Systems, IEEE CS Press*, p. 505-512, 1995.

[20] **J. L. Hainaut**. Database Reverse Engineering: Models, techniques, and strategies. *10th International Conference on Entity-Relationship Approach*, 1991.

[21] **J. L. Hainaut**. Theoretical and Practical Tools for Data Base Design. *Proceedings of the Very Large Data Bases, 7th International Conference*, p. 216-224, 1981.

[22] **J. L. Hainaut, M. Chandelon, C. Tonneau, M. Joris**. Contribution to a Theory of Database Reverse Engineering. *Proceedings of the 1993 Working Conference on Reverse Engineering*, 1993, prieiga per Internetą: <http://citeseer.ist.psu.edu/>.

[23] **J. L. Hainaut, J. M. Hick, J. Henrard, V. Englebert, D. Roland**. The Concept of Foreign key in Reverse Engineering: A Pragmatic Interpretative Taxonomy. *Technical report, Computer Science Departement, University of Namur*, 1997.

[24] **J. Henrard, V. Englebert, J. M. Hick, D. Roland, J. L. Hainaut**. Program understanding in databases reverse engineering. *Proceedings of DEXA'98,* 1998.

[25] **S. Horwitz, T. Reps**. The Use of Program Dependence Graphs in Software Engineering. *Proceedings of the Fourteenth International Conference on Software Engineering*, 1992, prieiga per Internetą: <http://www.cs.wisc.edu/>.

[26] **J. H. Jahnke**. Managing Uncertainty and Inconsistency in Database Reengineering Processes. *PhD Thesis*, 1999.

[27] **J. H. Jahnke, J. P. Wadsack**. Varlet: Human-Centered Tool Support for Database Reengineering. *Proceedings of Workshop on Software-Reengineering*, 1999.

[28] **M. Joris, R. Van Hoe, J. L. Hainaut, M. Chandelon, C. Tonneau, F. Bodart**. PHENIX: Methods and Tools for Database Reverse Engineering. *In Proc 5th International Conf. on Software Engineering and Applications*, 1992.

[29] **P. Johannesson**. A Method for Transforming Relational Schemas into Conceptual Schemas. *Proceedings of the 10th Int. Conf. on Data Engineering*, p. 190-201, 1994.

[30] **P. Young**. Program Comprehension. *Technical report*, *Center for Software Maintenance*, 1996.

[31] **G. Klein**. JFlex User's Manual. 2005, prieiga per Internetą: <http://www.jflex.de/>.

[32] **B. P. Leintz, E. F. Swanson**. Software maintenance Management. Addison-Wesley, 1980.

[33] **S. Lopes, J. M. Petit, F. Toumani**. Discovering interesting inclusion dependencies: application to logical database tuning. *Information Systems, 27:1-19*, 2002.

[34] **A. von Mayrhauser, A. M. Vans**. From Program Comprehension to Tool Requirements for an Industrial Environment. *In Proc. of Second Workshop on Program Comprehension*, 1993.

[35] **V. M. Markowitz, J. A. Makowsky**. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE Transactions on Software Engineering*, vol. 16, p. 777-790, 1990.

[36] **S. B. Navathe, A. M. Awong**. Abstracting Relational and Hierarchical Data with a Semantic Data Mode. *Proceedings of the 6th International Conference on Entity-Relationship Approach (ER'87)*, p. 305-333, 1987.

[37] **J. L. Nhampossa**. Strategies to Deal with Legacy Information Systems: A Case Study From the Mozambican Health Sector. *IRMA*, 2004, prieiga per Internetą: <http://heim.ifi.uio.no/>.

[38] **B. Paradauskas, L. Nemuraitė**. Duomenų bazės ir semantiniai modeliai: Monografija. ISBN 9955-09-436-2. Kaunas, Technologija, 2002.

[39] **W. J. Premerlani, M. R. Blaha**. An approach for Reverse Engineering of Relational Databases. *Proc. of the Working Conf. on Reverse Engineering (WCRE'93)*, 1993.

[40] **J. M. Petit, J. Kouloumdjian, J. F. Bouliaut, F. Toumani**. Using Queries to Improve Database Reverse Engineering. *Proceedings of the 13th int. Conf. on ER Approach (ER'94)*, 1994.

[41] **J. M. Petit, F. Toumani, J. F. Boulicaut, J. Kouloumdjian**. Towards the Reverse Engineering of Denormalized Relational Databases. *Proceedings of the Twelfth International Conference on Data Engineering (ICDE)*, 1996, p. 218-227.

[42] **A. Quilici, S. Woods, Y. Zhang**. New Experiments With A Constraint-Based Approach To Program Plan Matching. *Proceeding of the fourth IEEE Working Conference on Reverse Engineering (WCRE'97)*, 1997.

[43] **S. Ramanathan, J. Hodges**. Reverse Engineering Relational Schemas to Object-Oriented Schemas. *Techreport 960701, Department of Computer Science, Mississippi State University*, 1996.

[44] **S. Rugaber**. Program Comprehension. *Technical Report*, 1995, prieiga per Internetą: <http://coblitz.codeen.org:3125/>.

[45] **O. Signore, M. Loffredo, M. Gregori, M. Cima.** Using Procedural Patterns in Abstracting Relational Schemata. *Proceedings 3rd Workshop on Program Comprehension*, 1994.

[46] **S. R. Tilley**. Coming Attractions in Program Understanding II: Highlights of 1997 and Opportunities in 1998. *Technical Report CMU/SEI-98-TR-001*, 1998, prieiga per Internetą: <http://www.sei.cmu.edu/>

[47] **M. Weiser**. Program Slicing. *Proceedings on the 5th International Conference on Software Engineering*, 1981, p. 439-449.

7. TERMINŲ IR SUTRUMPINIMŲ ŽODYNAS

7.1 lentelė. Darbe naudojami sutrumpinimai ir jų paaiškinimai

Naudojamas sutrumpinimas, sąvoka	Aprašymas
1NF	pirmoji normalinė forma
ADI	atvirkštinė duomenų inžinerija (angl. *Data Reverse Engineering*)
ASM	abstrakčios sintaksės medis (angl. *Abstract Syntax Tree*)
CASE	automatizuotas kompiuterinis programinės įrangos kūrimas (angl. *Computer-Aided Software Engineering*)
CUP	efektyvių nagrinėtojų konstruktorius (angl. *Constructor of Useful Parsers*)
DB	duomenų bazė (angl. *database*)
DBVS	duomenų bazės valdymo sistema (angl. *Data Base Management System*)
DDL	duomenų aprašymo kalba (angl. *Data Definition Language*)
DML	duomenų manipuliavimo kalba (angl. *Data Manipulation Language*)
ER	esybių ryšių modelis (angl. *Entity Relationship*)
EER	išplėstas esybių ryšių modelis (angl. *Extended Entity-Relationship Model*)
ECR	esybių kategorijos ryšių modelis (angl. *Entity-Category-Relationship model*)
ERC+	išplėstas esybių ryšių modelis
IS	informacijos sistema (angl. *Information System*)
JDBC	Java duomenų bazės sąsaja (angl. *Java Database Connectivity*)
ODMG	objektinių duomenų valdymo grupė (angl. *Object Data Management Group*)
OMT	objektinio modeliavimo metodika (angl. *Object Modeling Technique*).
RDBVS	Reliacinė duomenų bazės valdymo sistema (angl. Relational *Data Base Management System*)
RIDG	reliacinis pereinamasis tiesioginis grafas (angl. *Relational Intermediate Direct Graph*)
SQL	struktūrinė užklausų kalba (angl. *Structured Query Language*)
UML	vieninga modeliavimo kalba (angl. *Unified Modeling Language*)
XML	išplėstinė dokumentų aprašų kalba (angl. *Extensible Markup Language*)

8. PRIEDAI

A PRIEDAS. C kodo gramatika ASM sudaryti

```
CProgram    ->    Consts Forwards Dclns Function+    => "program";

Includes    ->    ('#include' '"' <filename> '"'     => "include";
                  ';')*
Consts      ->    (Const ';')+                       => "consts"
      ->                                             => "consts";
Const       ->    `#define' Name                     => "const";
Forwards    ->    (Forward ';')+                     => "forwards"
      ->                                             => "forwards";
Forward     ->    ^' Type Name Params                => "forward";
Dclns       ->    (DclnList ';')+                    => "dclns"
      ->                                             => "dclns";
Type  ->          Id;
DclnList    ->    Type Dcln list ','                 => "dcln";
      ->          `struct' Type Dcln list `,'        => "structdcln";
Dcln  ->          Id '=' Expression                  => "="
      ->          Id;
Function    ->    Type Name Params `{' Dclns         => "function";
                  Statement+ '}'
Params      ->    '(' DclnList ? ')'                 => "params";
Block       ->    '{' Statement* '}'                 => "block";
Statement   ->    Assignment ';'
      ->          Name '(' (Expression list ',')?    => "call"
                  ')' ';'
      ->          'printf' '(' String? Expression    => "print"
                  list ',' ')' ';'
      ->          'printf' '(' String? ')' ';'       => "emptyprint"
      ->          'scanf' '(' String ? Id list ','   => "scanf"
                  ')' ';'
      ->          'if' '(' Expression ')'            => "if"
                  Statement ('else' Statement)?

      ->          'while' '(' Expression ')'         => "while"
                  Statement
      ->          'for' '(' Assignment ';'           => "for"
                  Expression ';'
                  Assignment ')' Statement
      ->          'for' '(' ';' ';' ')' Statement    => "for"

      ->          'do' Statement 'while'             => "do"
                  Expression ';'
      ->          'switch' '(' Term ')' '{' Case+    => "switch"
                  'default' ':' Block '}'

      ->          Block
      ->          SQLprefix SQLstatement             =>"embSQL"
                  SQLterminator?
      ->          (DclnList ';')+                    => "dclns"
      ->          Primary `++'                       => "++"
      ->          Primary `--'                       => "--"
      ->          ;
SQLprefix   ->    EXEC SQL DBclause?                 => "beginSQL"
SQLterminator     END-EXEC                           => "endSQL"
```

```
      ->
      ->            ;
SQLstatement ->     'SELECT' columnlist 'INTO'          => 'SQLselectone'
                    hostvariablelist ‘FROM’
                    tablelist
                    'SELECT' columnlist 'INTO'          => 'SQLselectone'
                    hostvariablelist ‘FROM’
                    tablelist 'WHERE' SQLExpression
      ->            'SELECT' columnlist 'INTO'          => ‘SQLselectone’
                    hostvariablelist 'FROM'
                    tablelist 'WHERE' ‘EXISTS’
                    SQLExpression
      ->            'SELECT' columnlist 'INTO'          => ‘SQLselectone’
                    hostvariablelist 'FROM'
                    tablelist 'WHERE' ‘NOT EXISTS’
                    SQLExpression
      ->            'SELECT' ‘COUNT’ ‘(‘ ‘*’ ‘)’        => ‘SQLselectonecount’
                    columnlist 'INTO'
                    hostvariablelist 'FROM'
                    tablelist 'WHERE' SQLExpression
      ->            'SELECT' ‘DISTINCT’ columnlist      => ‘SQLselectonedistinct’
                    'INTO' hostvariablelist 'FROM'
                    tablelist 'WHERE' SQLExpression

      ->            'SELECT' columnlist 'INTO'          =>’SQLselectonegroupby’
                    hostvariablelist 'FROM'
                    tablelist 'WHERE' SQLExpression
                    ‘GROUP’ ‘BY’ columnlistgroupby
      ->            'SELECT' columnlist 'INTO'          =>’SQLselectonegroupby’
                    hostvariablelist 'FROM'
                    tablelist 'WHERE' SQLExpression
                    ‘ORDER’ ‘BY’ columnlistgroupby
      ->            'SELECT' columnlist ‘FROM’          => 'SQLselecttwo'
                    tablelistmod
                    'SELECT' columnlist ‘FROM’          => 'SQLselecttwo'
                    tablelistmod 'WHERE'
                    SQLExpression
      ->            'SELECT' columnlist 'FROM'          => ‘SQLselecttwo’
                    tablelistmod 'WHERE' ‘EXISTS’
                    SQLExpression
      ->            'SELECT' columnlist 'FROM'          => ‘SQLselecttwo’
                    tablelistmod 'WHERE' ‘NOT
                    EXISTS’ SQLExpression
      ->            'SELECT' ‘COUNT’ ‘(‘ ‘*’ ‘)’        => ‘SQLselecttwocount’
                    columnlist ‘FROM’ tablelistmod
                    'WHERE' SQLExpression
      ->            'SELECT' ‘DISTINCT’ columnlist      => ‘SQLselecttwodistinct’
                    ‘FROM’ tablelistmod 'WHERE'
                    SQLExpression
      ->            'SELECT' columnlist ‘FROM’          =>’SQLselecttwogroupby’
                    tablelistmod 'WHERE'
                    SQLExpression ‘GROUP’ ‘BY’
                    columnlistgroupby
```

```
      ->          'SELECT' columnlist 'FROM'          =>'SQLselecttwogroupby'
                  tablelistmod 'WHERE'
                  SQLExpression 'ORDER' 'BY'
                  columnlistgroupby
      ->          'INSERT' 'INTO'                     =>'SQLinsert'
                  tablelist'VALUES'
                  '('hostvariablelist ')'
      ->          'DELETE' Id 'FROM' tablelist        =>'SQLdelete'
                  'WHERE' SQLExpression
      ->          'UPDATE' tablelist 'SET'            =>'SQLupdate'
                  (SQLAssignment ',') list 'WHERE'
                  SQLExpression
      ->          ;                                   => 'SQLselect'
      ->          ( Name list ',')                    => 'tablelist'
tablelistmod      ( tablename list ',')               =>'tablelist'
      ->
      ->          Id Id                               =>'tablename'
      ->          ( Term list ',')                    =>'columnlist"
columnlistgroupb  ( Name list ',')                    =>'columnlistgroupby"
y
 ->
Hostvariablelist  (Variable list ',')                 => 'hostvariablelist"
      ->
      ->                                              => 'hostvariablelist"
Variable    ->    ':' Name ;
SQLExpression     SQLExpression 'AND'                 => "SQLExpression"
      ->          SQLAssignment
                  SQLExpression 'OR' SQLAssignment    => "SQLExpression"

                  SQLAssignment;
SQLAssignment     Id '=' Name                         => "SQLAssignment="
      ->
      ->          Id '>' Name                         => "SQLAssignment>"
      ->          Id '<' Name                         => "SQLAssignment<"
      ->          Id '>=' Name                        => "SQLAssignment>="
      ->          Id '<=' Name                        => "SQLAssignment<="
      ->          Id '<>' Name                        => "SQLAssignment<>"
      ->          Id '=' Name '(' (Expression list    => "SQLAssignment="
                  ',')? ')' ';'
      ->          Id '>' Name '(' (Expression list    => "SQLAssignment>"
                  ',')? ')' ';'
      ->          Id '<' Name '(' (Expression list    => "SQLAssignment<"
                  ',')? ')' ';'
      ->          Id '>=' Name '(' (Expression        => "SQLAssignment>="
                  list ',')? ')' ';'
      ->          Id '<=' Name '(' (Expression        => "SQLAssignment<="
                  list ',')? ')' ';'
      ->          Id '<>' Name '(' (Expression        => "SQLAssignment<>"
                  list ',')? ')' ';'
      ->          Id 'LIKE' String                    => "SQLAssignmentLIKE"
      ->          Id '=' SQLStatement                 => "SQLAssignment="
      ->          Id '=' 'ANY' SQLStatement           => "SQLAssignment="
      ->          Id '>' 'ANY' SQLStatement           => "SQLAssignmen>"
      ->          Id '<' 'ANY' SQLStatement           => "SQLAssignment<"
      ->          Id '<=' 'ANY' SQLStatement          => "SQLAssignment<="
      ->          Id '>=' 'ANY' SQLStatement          => "SQLAssignment>="
      ->          Id '<>' 'ANY' SQLStatement          => "SQLAssignment<>"
      ->          Id '=' 'ALL' SQLStatement           => "SQLAssignment="
      ->          Id '>' 'ALL' SQLStatement           => "SQLAssignment>"
```

```
      ->          Id '<' 'ALL' SQLStatement          => "SQLAssignment<"
      ->          Id '<=' 'ALL' SQLStatement         => "SQLAssignment<="
      ->          Id '>=' 'ALL' SQLStatement         => "SQLAssignment>="
      ->          Id '<>' 'ALL' SQLStatement         => "SQLAssignment<>"
      ->          Id '=' 'IN' SQLStatement           => "SQLAssignment="
      ->          Id '>' 'IN' SQLStatement           => "SQLAssignment>"
      ->          Id '<' 'IN' SQLStatement           => "SQLAssignment<"
      ->          Id '<=' 'IN' SQLStatement          => "SQLAssignment<="
      ->          Id '>=' 'IN' SQLStatement          => "SQLAssignment>="
      ->          Id '<>' 'IN' SQLStatement          => "SQLAssignment<>"
DB clause   ->    'BEGIN' 'DECLARE' 'SECTION'        => "DBclause"
      ->          'END' 'DECLARE' 'SECTION'          => "DBclause"
      ->          'WHENEVER' 'SQL' 'WARNING'         => "DBclause"
                  'CALL' Name '(' (Expression list
                  ',')? ')'
      ->          'WHENEVER' 'SQL' 'NOT' 'FOUND'     => "DBclause"
                  'CALL' Name'(' (Expression list
                  ',')? ')'
      ->          'WHENEVER' 'SQL' 'NOT' 'FOUND'     => "DBclause"
                  'DO' 'BREAK'
      ->          'WHENEVER' 'SQL' 'NOT' 'FOUND'     => "DBclause"
                  'DO' 'CONTINUE'
      ->          'COMMIT' 'WORK'                    => "DBclause"
      ->          'WHENEVER' 'SQL' 'ERROR' 'CALL'    => "DBclause"
                  Name '(' (Expression list ',')?
                  ')'
      ->          'DISCONNECT' 'ALL'                 => "DBclause"
      ->          'USE' Id                           => "DBclause"
      ->          'CONNECT' Name 'IDENTIFIED' 'BY'   => "DBclause"
                  Name
      ->          'COMMIT'                           => "DBclause"
      ->          'COMMIT' 'WORK' 'RELEASE'          => "DBclause"
      ->          'COMMIT' 'WORK'                    => "DBclause"
      ->          'OPEN' Name                        => "DBclause"
      ->          'CLOSE' Name                       => "DBclause"
      ->          'DECLARE' Name 'FOR'               => "DBclause"
      ->          'FETCH' Name 'INTO'                => "DBclause"
                  hostvariablelist
Case  ->          'case' '<integer>' ':' Block       => "case";
Assignment  ->    Id '=' Expression                  => "assign";
      ->          Id '+''=' Expression               => "assign";
      ->          Id '-''=' Expression               => "assign";
Expression  ->    LExpression '?' LExpression ':'    => "?"
                  LExpression
      ->          LExpression;
LExpression ->    LExpression '&&' Comparison        => "and"
      ->          LExpression '||' Comparison        => "or"
      ->          LExpression '~' Comparison         => "xor"
      ->          Comparison;
Comparison  ->    Term '<=' Term                     => "<="
      ->          Term '==' Term                     => "=="
      ->          Term '>=' Term                     => ">="
      ->          Term '!=' Term                     => "!="
      ->          Term '<' Term                      => "<"
      ->          Term '>' Term                      => ">"
      ->          Term;
Term  ->          Term '+' Factor                    => "+"
      ->          Term '-' Factor                    => "-"
```

```
      ->          Factor;
Factor      ->    Exp '*' Factor                    => "*"
      ->          Exp '/' Factor                    => "/"
      ->          Exp '%' Factor                    => "%"
      ->          Exp ;
Exp   ->          Primary '**' Exp                  => "**"
      ->          Primary;
Primary     ->    '-' Primary                       => "-"
      ->          '+' Primary
      ->          '!' Primary                       => "!"
      ->          '++' Primary                      => "++"
      ->          '--' Primary                      => "--"
      ->          Primary ‘++’                      => "++"
      ->          Primary ‘--’                      => "--"
      ->          Atom;
Atom  ->          'eof'                             => "eof"
      ->          '<integer>'
      ->          Id
      ->          '(' Expression ')';
      ->          Name '(' (Expression list ',')?   => "rhscall"
                  ')' ';'
Initializer ->    '<integer>'
      ->          '&' Name                          => "&";
Id    ->          '*' Name                          => "*"
      ->          '&' Name                          => "&"
      ->          Name;
Name  ->          ‘<identifier>';
String      ->    ‘<text>’ ;
```

B PRIEDAS. Apribojimų ir priklausomybių aptikimo užklausos

- **Ryšio ir lygybės apribojimų aptikimo poaibio užklausa**

Kad nustatyti, ar egzistuoja ryšio arba lygybės apribojimai tarp esybės `R1` atributo (ar atributų rinkinio) `A` ir esybės `R2` pirminio (arba antrinio) identifikatoriaus (arba jų rinkinio) `B`, naudojama poaibio užklausa. Pažymėtina, kad `A` ir `B` privalo turėti tą patį duomenų tipą ir apimti tą patį atributų skaičių. Tikrinant poaibio kriterijų, kiekvienai liktinių šaltinių esybių vienos esybės pirminio (arba antrinio) identifikatoriaus bei kitos esybės atributų kombinacijų porai reikia įvykdyti SQL užklausas, kurių šablonai tokie:

```
C1 = select count (*) from R1
where A not in
(select B from R2);
C2 = select count (*) from R2
where B not in
(select A from R1);
```

Jei `C1` yra nulis, tada galima daryti išvadą, kad gali egzistuoti ryšio apribojimas `R1.A<<R2.B`, ir atvirkščiai, jei `C2` yra nulis, tada gali egzistuoti ryšio apribojimas `R2.B<<R1.A`. Pažymėtina, kad galimi atvejai, kai ir `C1`, ir `C2` yra nulis. Tuomet galima daryti išvadą, kad dvi atributų aibės `A` ir `B` yra lygios, t.y. egzistuoja lygybės apribojimas.

- **coex apribojimų aptikimo užklausa**

Kad nustatyti, ar egzistuoja *coex* apribojimas tarp esybės `R` atributų `A`, `B` ir `D`, naudojama užklausa. Pažymėtina, kad `A`, `B` ir `D` yra neprivalomi atributai. Tikrinant *coex* apribojimą, kiekvienos liktinių šaltinių esybės kiekvienai neprivalomų atributų kombinacijai reikia įvykdyti SQL užklausą, kurios šablonas toks:

```
C = select count (*) from R
WHERE not((A is null and B is null and D is null)
or (A is not null and B is not null and D is not null));
```

Jei `C` yra nulis, tada galima daryti išvadą, kad *coex* apribojimas tarp atributų `A`, `B` ir `D` egzistuoja, jei `C` daugiau už nulį, tuomet *coex* apribojimas atributams `A`, `B` ir `D` neegzistuoja.

- **exact-1 apribojimų aptikimo užklausa**

Kad nustatyti, ar egzistuoja *exact-1* apribojimas tarp esybės `R` atributų `A`, `B` ir `D`, naudojama užklausa. Pažymėtina, kad `A`, `B` ir `D` yra neprivalomi atributai. Tikrinant *exact-1* apribojimą,

kiekvienos liktinių šaltinių esybės kiekvienai neprivalomų atributų kombinacijai reikia įvykdyti SQL užklausą, kurios šablonas toks:

```
C = select count (*) from R
where (A is not null and B is not null)
or (A is not null and D is not null)
or (B is not null and D is not null)
or (A is null and B is null and D is null);
```

Jei `C` yra nulis, tada galima daryti išvadą, kad *exact-1* apribojimas tarp atributų `A`, `B` ir `D` egzistuoja, jei `C` daugiau už nulį, tuomet *exact-1* apribojimas atributams `A`, `B` ir `D` neegzistuoja.

- **disjoint(R1.A1,R2.A1) apribojimų aptikimo užklausa**

Kad nustatyti, ar egzistuoja *disjoint* apribojimas tarp esybės `R1` pirminio identifikatoriaus `A1` ir esybės `R2` pirminio identifikatoriaus `A1`, naudojama užklausa. Pažymėtina, kad `R1.A1` ir `R2.A1` privalo turėti tą patį duomenų tipą. Tikrinant *disjoint* apribojimą, kiekvienai liktinių šaltinių esybių pirminių identifikatorių porai reikia įvykdyti SQL užklausas, kurių šablonai tokie:

```
C1 = select count (*) from R1
where A1 in
(select A1 from R2);
C2 = select count (*) from R2
where A1 in
(select A1 from R1);
```

Jei `C1` ir `C2` yra nulis, tada galima daryti išvadą, kad *disjoint* apribojimas egzistuoja. Jei `C1` ir `C2` daugiau už nulį, tuomet *disjoint* apribojimas tarp `R1` atributo `A1` ir `R2` atributo `A1` negalioja.

- **(R1.A1∪R2.A1): A1-->A2 funkcinių priklausomybių aptikimo užklausa**

Kad nustatyti, ar egzistuoja funkcinė priklausomybė tarp esybių `R1` ir `R2` pirminių identifikatorių `A1` ir esybių atributų `A2`, naudojama užklausa. Pažymėtina, kad `R1.A1` ir `R2.A1` bei `R1.A2` ir `R2.A2` privalo turėti tą patį duomenų tipą. Tikrinant funkcines priklausomybes, kiekvienai liktinių šaltinių esybių, kurios neturi *disjoint* apribojimo, pirminių identifikatorių ir esybių atributų kombinacijų porai reikia įvykdyti SQL užklausas, kurių šablonai tokie:

```
C = select count (*) from R1, R2
where R1.A1=R2.A1 and R1.A2<>R2.A2;
```

Jei `C` yra nulis, tada galima daryti išvadą, kad funkcinė priklausomybė egzistuoja. Jei `C` daugiau už nulį, tuomet funkcinė priklausomybė tarp esybių `R1` ir `R2` pirminių identifikatorių `A1` ir esybių atributų `A2` negalioja.

- **R1.A1 in (R2.A1∪R3.A1) apribojimų aptikimo užklausa**

Kad nustatyti, ar egzistuoja *in* apribojimas tarp esybės `R1` pirminio identifikatoriaus `A1` ir bendros esybių `R2` bei `R3` pirminių identifikatorių `A1` aibės, naudojama užklausa. Pažymėtina, kad `R1.A1`, `R2.A1` ir `R3.A1` privalo turėti tą patį duomenų tipą. Tikrinant *in* apribojimą, kiekvienai liktinių šaltinių esybių tipų (ne mažiau dviejų), kurių pirminis identifikatorius turi ir ryšio apribojimą, nurodantį į bendrą esybės tipą, pirminių identifikatorių ir bendro esybės tipo pirminio identifikatoriaus kombinacijai reikia įvykdyti SQL užklausas, kurių šablonai tokie:

```
C = select count (*) from R1
where A1 not in
(select A1 from R2
union
select A1 from R3);
```

Jei `C` yra nulis, tada galima daryti išvadą, kad *in* apribojimas egzistuoja. Jei `C` daugiau už nulį, tuomet *in* apribojimas tarp bendros esybės `R1` pirminio identifikatoriaus `A` ir esybių `R2` bei `R3` atributų `A1` aibės negalioja.

- **A:BA1==>(BA1=A1) implikacinių apribojimų aptikimo užklausa**

Kad nustatyti, ar egzistuoja implikacinis apribojimas tarp esybės `R` pirminio identifikatoriaus `A1` ir antrinio identifikatoriaus `BA1`, naudojama užklausa. Pažymėtina, kad `A1` ir `BA1` privalo turėti tą patį duomenų tipą, `A1` turi būti privalomas atributas, `BA1` – neprivalomas atributas. Tikrinant implikacinius apribojimus, kiekvienos liktinių šaltinių esybės kiekvienai privalomo pirminio identifikatoriaus ir neprivalomo antrinio identifikatoriaus porai reikia įvykdyti SQL užklausas, kurių šablonai tokie:

```
C = select count (*) from R
where BA1 is not null and BA1<>A1;
```

Jei `C` yra nulis, tada galima daryti išvadą, kad implikacinis apribojimas egzistuoja. Jei `C` daugiau už nulį, tuomet implikacinis apribojimas tarp esybės `R` pirminio identifikatoriaus `A1` ir antrinio identifikatoriaus `A2` negalioja.

- **Pertekliaus apribojimų aptikimo užklausa**

Kad nustatyti, ar egzistuoja pertekliaus apribojimas tarp esybės `R1` su pirminiu identifikatoriumi `A1` ir `R2` esybės su išoriniu raktu `A1` atributų `A2`, naudojama užklausa. Pažymėtina, kad `R1.A2` ir `R2.A2` privalo turėti tą patį duomenų tipą. Tikrinant pertekliaus apribojimus, kiekvienai

liktinių šaltinių esybių, kurios susijusios ryšio apribojimu, ne pirminių identifikatorių ir ne ryšio apribojimų atributų porai reikia įvykdyti SQL užklausas, kurių šablonai tokie:

```
C = select count (*) from R1, R2
where R1.A1=R2.A1 and R1.A2<>R2.A2;
```

Jei `C` yra nulis, tada galima daryti išvadą, kad pertekliaus apribojimas egzistuoja. Jei `C` daugiau už nulį, tuomet pertekliaus apribojimas tarp esybių `R1` ir `R2` atributų `A1` ir `A2` negalioja.

C PRIEDAS. Straipsnis

Loginės DB schemos atstatymas naudojant JDBC

Bronius Paradauskas, Aurimas Laurikaitis, Sigitas Paulavičius, Anna Truncaitė
Kauno technologijos universitetas, Informacijos sistemų katedra
Studentų g. 50, 51368 Kaunas

Straipsnyje aprašomas duomenų bazės metaduomenų išgavimas, panaudojant Java kalbos technologijas. Aptariami JDBC metodai ir savybės. Pateikiama ADI metodika lentelėms, atributams ir jų savybėms, pirminiams ir išoriniams raktams, ryšiams išgauti. Pristatomas repozitorius duomenų bazės metaduomenims saugoti.

1. Įvadas

Liktinė sistema – tai bet kokia informacinė sistema, kuri pastebimai priešinasi modifikavimui ar evoliucijai, skirtų atitikti naujus, nuolat kintančius verslo reikalavimus [3]. Liktinės sistemos paprastai apima neįtikėtinai detalias veiklos taisykles ir sudaro organizacijos informacinių srautų pagrindą, kuris sutelkia verslo procesų informaciją [1]. Vienos iš šių sistemų sutrikimas gali turėti rimtų padarinių verslui. Liktinės sistemos sukelia nemažai problemų jas eksploatuojančioms organizacijoms. Svarbiausios liktinių sistemų problemos:

- sistemas paprastai palaiko techniškai pasenusi, pernelyg lėta aparatūra, kurios palaikymo kaštai pakankamai dideli;
- programinės įrangos palaikymas yra brangus; trūksta dokumentacijos ir bendro sistemos vidinio funkcionalumo supratimo, todėl klaidų aptikimas brangus ir ilgai trunka;
- nėra aiškiai apibrėžtų sistemos sąsajų, todėl sunku integruoti;
- palikuoninių sistemų funkcionalumą yra labai sunku arba net neįmanoma praplėsti.

Siekiant sėkmingai išspręsti liktinių sistemų problemas, reikia turėti aiškų informacinės sistemos vaizdą. Todėl pirmiausia reikia įvykdyti atvirkštinę duomenų inžineriją, kurios tikslas atrasti ir išgauti kiek įmanoma detalesnes veiklos žinias iš liktinių šaltinių.

Atvirkštinė duomenų inžinerija nagrinėja liktinę sistemą, siekiant:

- Nustatyti sistemos komponentus ir jų tarpusavio ryšius;
- Sukurti sistemos atvaizdavimus kita forma arba aukštesniu abstrakcijos lygiu;
- Sukurti fizinį sistemos atvaizdavimą;
- Nustatyti kritinius komponentus, kuriuos reikia peržiūrėti, siekiant įgyvendinti verslo pokyčius.

Atvirkštinė duomenų inžinerija (ADI) – analitinių metodų taikymas vienam ar daugiau palikuoninių duomenų šaltinių, siekiant iš pastarųjų išgauti struktūrinę informaciją (pvz. sąvokų

apibrėžimus, schemų aprašymus), kad būtų pagerintas duomenų bazės projektas arba pateikta trūkstama schemų dokumentacija. Šiame darbe nagrinėjama ADI metodika skirta tik liktinėms sistemoms su reliacinėmis duomenų bazėmis. Kadangi reliacinis modelis turi ribotą semantinį išraiškingumą, tai sudaryta ADI metodika prie schemos papildomai pateikia liktinėje schemoje neišreikštąsias struktūras ir apribojimus t.y. esybes ir ryšius. Tam kad išbandyti metodikos veiksmingumą, šiame darbe buvo sukurtas įrankis, kuris palaiko šią metodiką. Siekiant turėti universalų ir nuo platformos nepriklausantį įrankį, realizacijai pasirinkta Java programavimo kalba. Ši kalba buvo pasirinkta, nes tai yra aukšto lygio, neutralios platformos, paprasta, pernešama *(angl. portable)* ir objektinė kalba. Leidžiant dauguma programinių kalbų sukurtas programas, jas reikia arba perkompiliuoti, arba interpretuoti iš naujo. Java šiuo atžvilgiu yra netradicinė, nes programa yra tuo pačiu metu yra ir kompiliuojama, ir interpretuojama. Kompiliatorius verčia programą į tarpinę kalbą, Javos baitų kodus *(angl. Java bytecodes)* – nuo platformos nepriklausomas kodo instrukcijas. Kiekvieną kartą leidžiant programą interpretatorius vykdo Javo baitų kodo instrukcijas. Sąsajai su reliacine duomenų baze pasirinkta JDBC sąsaja. Šiame straipsnyje detaliai aprašysime duomenų bazės metaduomenų išgavimą ir liktinių informacijos sistemų, panaudojanta JAVA kalbos technologijas, JDBC sąsają ir klases.

2 skyriuje apžvelgiamos JDBC galimybės, pagrindinės klasės ir sąsajos. 3. skyriuje nagrinėjama programa metaduomenims išgauti. 3.1 skyriuje aprašomi metaduomenų išgavimo žingsniai 3.2 skyriuje apžvelgiamos pagrindinės klasės DatabaseMetaDate metodai. 0 skyriuje pateikiamos pagrindinės straipsnio išvados.

2. JDBC galimybių apžvalga

Java programavimo kalba yra viena populiariausių objektinių programavimo kalbų pasaulyje, pasižyminti tokiomis savybėmis kaip paprastumas, nepriklausomumas nuo platformos, tinkamumas paskirstytoms sistemoms ir lygiagrečių procesų realizacijai. Parašytų programų duomenis ir rezultatus efektyviausia saugoti duomenų bazėse. Didžiausią rinkos dalį užima reliacinės duomenų bazės, kurių pagrindas yra reliacinis duomenų modelis. Java programavimo kalba yra pagrįsta objektiniu Java duomenų modeliu [2], todėl tiesiogiai saugoti duomenis reliacinėje duomenų bazėje yra neįmanoma. Duomenų saugojimui ir nuskaitymui yra naudojamos specialios taikomųjų programų sąsajos (*API*) kaip JDO, SQLJ. JDBC (*Java DataBase Connectivity*) yra populiariausios iš šiuo metu sukurtų Java taikomųjų programų sąsajų su reliacinėmis duomenų bazėmis.

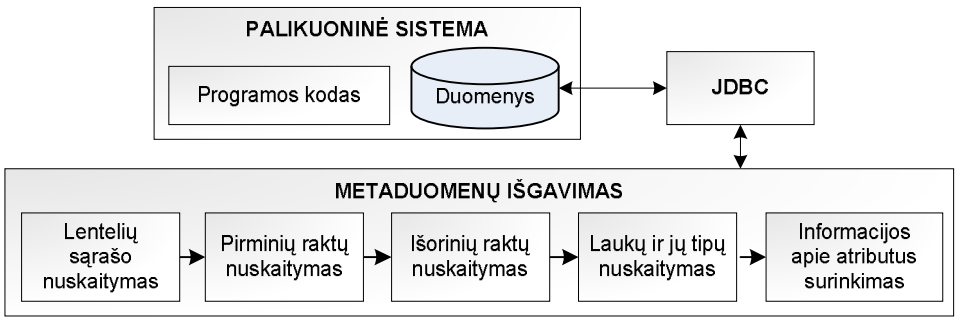
JDBC yra standartizuota Java kalbos sąsaja su reliacinėmis duomenų bazėmis. JDBC naudoja reliacinį SQL duomenų modelį ir tiesiogiai leidžia vykdyti SQL užklausas. Gauti rezultatai saugomi Java klasės *ResultSet* objekte, kuris leidžia iteratyviai gauti konkrečias atributų reikšmes. SQL kalba turi aibę dialektų tokių kaip PL/SQL, Transact-SQL, kadangi JDBC sąsaja leidžia tiesiogiai vykdyti SQL užklausas, yra galimybė naudoti egzistuojančius SQL dialektus. Tai leidžia efektyviau išnaudoti reliacinės duomenų bazės galimybes, bet tokia taikomoji programa veikia tik su vieno gamintojo reliacinėmis duomenų bazėmis.

JDBC sudaro tokios pagrindinės klasės ir sąsajos:

- *DriverManager* - registruoja tvarkykles (driver), pagal nurodytą URL iš užregistruotų tvarkyklių parenka tinkamą tvarkyklę ir tos tvarkyklės priemonėmis sukuria ir grąžina *Connection* objektą.
- *Driver* - tvarkyklės sąsaja, apibrėžianti metodus duomenų bazės URL interpretavimui ir *Connection* objektų sukūrimui.
- *Connection* - ryšio su duomenų baze sąsaja atsakinga už įvairių vykdomųjų sakinių (*Statement*, *PreparedStatement*, *CallableStatement*) objektų sukūrimą, ryšio nutraukimą, duomenų bazės meta duomenų objekto pateikimą. Praktiškai, *Connection* tipo objektas yra priėjimo prie duomenų bazės abstrakcija.
- *DatabaseMetaData* - duomenų bazės meta duomenų sąsaja, leidžianti sužinoti duomenų bazės struktūrą (katalogus, schemas, lenteles) ir savybes (ar palaiko transakcijas, duomenų tipus, maksimalų prisijungimų kiekį ir pan.).
- *Statement* - paprasto vykdomojo sakinio, pagrįsto SQL, sąsaja.
- *PreparedStatement* - paruošto vykdomojo sakinio, pagrįsto SQL su nustatomais parametrais, sąsaja.
- *CallableStatement* - iššaukiamo vykdomojo sakinio sąsaja, skirta darbui su vidinėmis duomenų bazės procedūromis (stored procedure).
- *ResultSet* - rezultatų rinkinio sąsaja, leidžianti perrinkti (iterate) vykdomųjų sakinių grąžinamus rezultatus.
- *ResultSetMetaData* – rezultatų meta duomenų sąsaja.
- *SQLException* - specifinė SQL išimčių klasė skirta duomenų bazių klaidų valdymui.

3. Metaduomenų išgavimas naudojant JDBC

Bendru atveju metaduomenys yra duomenys apie duomenis. Mūsų atveju metaduomenys tai yra informacija apie duomenų bazės serverio savybes, DB schemą arba *ResultSet* struktūrą. Konceptualioje metaduomenų išgavimo schemoje pavaizduota, kaip metaduomenų išgavimo įrankis sąveikauja su palikuoninės sistemos DB, JDBC bibliotekos pagalba. Prisijungus prie duomenų bazės pradedamas metaduomenų išgavimas, pirmiausia nuskaitomas lentelių sarašas su lentelių pavadinimais, toliau nuskaitomas kiekvienos lentelės pirminių raktų sąrašas, nuskaitomas kiekvienos lentelės indeksų sąrašas, nuskaitomas kiekvienos lentelės laukų sąrašas ir jų tipai, galų gale sudaroma visos išgautos informacijos suvestinė.



1 pav. Konceptuali metaduomenų išgavimo schema

3.1. Metaduomenų išgavimo žingsniai

Prieš pradedant bet kokią programos sąveiką su duomenų bazės serveriu, programa privalo prisijungti prie serverio. Kad atlikti prisijungimą, būtina užkrauti tinkamą tvarkyklę ir iškviesti metodą *driverManager.getConnection*. Užkraunama tvarkyklė turi atitikti duomenų bazės serverį.

Prisijungimui prie palikuoninės sistemos DB naudojama JDBC bibliotekos klasė *Connection*. *Connection* - ryšio su duomenų baze sąsaja atsakinga už įvairių vykdomųjų sakinių (*Statement*, *PreparedStatement*, *CallableStatement*) objektų sukūrimą, ryšio nutraukimą, duomenų bazės meta duomenų objekto pateikimą. Praktiškai, *Connection* tipo objektas yra priėjimo prie duomenų bazės abstrakcija.

Metaduomenų išgavimui naudojama ši klasė *DatabaseMetaData* - duomenų bazės meta duomenų sąsaja, leidžianti sužinoti duomenų bazės struktūrą ir savybes.

Toliau esančioje programos atkarpoje sukuriamas prisijungimo *Connection* klasės objektas pavadinimu con ir šio prisijungimo metaduomenų *DatabaseMetaData* klasės objektas dbmd. Šie objektai bus naudojami tolimesnėse programinio kodo atkarpose. (tolimesniuose programinio kodo segmentuose).

```
Connection con = DriverManager.getConnection(url, username, password);
DatabaseMetaData dbmd = con.getMetaData();
```

Lentelių sąrašas yra nuskaitomas *DatabaseMetaData* klasės metodu *getTables*. Šio metodo parametrai tai dominančių lentelių atrankos kriterijai. Metodas gražina *ResultSet* klasės objektą su lentelių informacija. Žemiau esančiame kodo segemente parodoma, kaip suformuojami atrankos kriterijai, atliekamas metodas *getTables* bei iteraciniu būdu nuskaitomas kiekvienos lentelės pavadinimas. Detaliau metodas getTable aprašomas 3.4 skyriuje.

```
String [] tableTypes = new String[1];
tableTypes[0] = "TABLE";
ResultSet rs = dbmd.getTables(null, null, null, tableTypes);
rsmd = rs.getMetaData();
while (rs.next()) {
                ...
      // lenteles pavadinimas
      String tableName = rs.getString(”TABLE_NAME”);
                ...
}
```

Pirminiai raktai ir indeksai nuskaitomi atitinkamai *getPrimaryKeys* ir *getIndexInfo* metodais. Tolimesniame programinio kodo atkarpoje pademonstruojama, kaip šie metodai iškviečiami, ir cikle nuskaitomi raktinių laukų pavadinimai.

```
ResultSet rs = dbmd.getPrimaryKeys(null, null, tableName);
while (rs.next()) {
                ...
      // pirminio rakto laukas
      String pkColumnName = rs.getString("COLUMN NAME");
                ...
}
           ...
ResultSet rs = dbmd.getIndexInfo(null, null, tableName, false, true);
while (rs.next()) {
                ...
      // indekso laukas
      String idxColumnName = rs.getString("COLUMN_NAME");
                ...
}
```

Paskutinis metaduomenų išgavimo žingsnis tai DB laukų, jų tipų bei kai kurios kitos informacijos apie laukus nuskaitymas. Tolimesnėje programinio kodo atkarpoje iškviečiamas metodas *getColumns*, kuris gražina informaciją apie laukus, atitinkančius parametruose nurodytus atrankos kriterijus. Šiuo atveju vienintelis atrankos kriterijus tai kintamajame tableName įrašytas lentelės pavadinimas. Užrašas “%” tai SQL kalbos šablono išraiška, reiškianti bet kokio ilgio ir bet

kokių simbolių seką, t.y. “bet kas”. Šiuo užrašu atrenkami laukų pavadinimai. Detaliau *getColumns* metodo parametrai ir gražinamos reikšmės aprašomos tolimesniam poskyryje. Kodo atkarpoje cikliškai nuskaitomas kiekvieno lauko pavadinimas, nustatomas lauko tipas, bei požymis, ar laukas gali įgyti reikšmę NULL.

```
rs = dbmd.getColumns(null, null, tableName, "%");
while (rs.next()) {
                  ...
      // atributo pavadinimas
      String columnName = rs.getString("COLUMN_NAME");

      // atributo SQL tipas
      String columnType;
      int sqlColType = new Integer(rs.getString("DATA_TYPE")).intValue();
      switch(sqlColType) {
            case 1:
                  columnType = "String";
            break;
            case 91:
                  columnType = "Date";
            break;
            case 4 :
                  columnType = "Integer";
            break;
            case 1111:
                  columnType = "Float";
            break;
                  default:
                  columnType = "Unknown";
      }

      // ar atributas gali netureti reiksmes
      bool canBeNull = getString("IS NULLABLE").equals("YES");
                  ...
}
```

3.2. Pagrindinių klasės DatabaseMetaData metodų metaduomenims išgauti apžvalga

DatabaseMetaData tai JDBC bibliotekos klasė, leidžianti nuskaityti duomenų bazės struktūrą (katalogus, schemas, lentelių struktūrą) ir savybes (ar palaiko transakcijas, duomenų tipus, maksimalų prisijungimų kiekį ir pan.). Metaduomenų nuskaitymui šioje klasėje gali būti naudojami du metodai: *getTables*, kuris gražina DB lentelių sąrašą, ir *getColumns*, kuris nuskaito informaciją apie lentelės laukus. Dabar detaliau apie kiekvieną.

Metodas *DatabaseMetaData.getTables* gražina *ResultSet* objektą su lentelių sąrašu ir kai kuria informacija apie jas. Metodo parametrai tai lentelių atrankos kriterijai, t.y. yra gražinama informacija tik apie tas lenteles, kurios atitinka parametruose pateikiamus kriterijus. Paminėtina, kad

šablonai yra užrašomi SQL kalbos šablono sintakse. Užrašas "%" arba reikšmė NULL reiškia „bet kuris". Taigi metodo parametrai yra šie. (1 lent.):

1 lent. Metodo *getTables* parametrai

Paramentras	Tipas	Aprašymas
catalog	*String*	Katalogo pavadinimas
schemaPattern	*String*	Schemos pavadinimo šablonas
tableNamePattern	*String*	Lentelės pavadinimo šablonas
types	*String[]*	Atrenkamų lentelių tipų masyvas. Galimi lentelių tipai yra šie: TABLE, VIEW, SYSTEM TABLE, GLOBAL TEMPORARY, LOCAL TEMPORARY, ALIAS ir SYNONYM

Kiekviename gražinamojo *ResultSet* įraše patalpinamama informacija apie vieną konkrečią lentelę. Gražinamas *ResultSet* objektas būna sudaryti iš 10 stulpelių, iš kurių svarbūs yra šie (2 lent.)

2 lent. Pagrindiniai Metodo *getTables* rezultatų stulpeliai

Stulpelio pavadinimas	Tipas	Aprašymas
TABLE_CAT	*String*	Lentelės katalogas
TABLE_SCHEM	*String*	Lentelės schema
TABLE_NAME	*String*	Lentelės pavadinimas
TABLE_TYPE	*String*	Lentelės tipas. Galimos reikšmės atitinka galimas parametro *types* reiškmes
REMARKS	*String*	Pastabos prie lentelės

Kitas *DatabaseMetaData* klasės metodas, *getColumns,* gražina objektą *ResultSet* su informacija apie lentelės struktūrą, t.y. informaciją apie lentelės laukus. Kaip ir ankstesnio metodo, šio metodo parametrai tai atrankos šablonai (3 lent.):

3 lent. Metodo *getColumns* parametrai

Paramentras	Tipas	Aprašymas
catalog	*String*	Katalogo pavadinimas
schemaPattern	*String*	Schemos pavadinimo šablonas
tableNamePattern	*String*	Lentelės pavadinimo šablonas
columnNamePattern	*String*	Lauko pavadinimo šablonas

Gražinamas *ResultSet* objektas būna sudarytas iš daugiau kaip 20 stulpelių, iš kurių praktikoje dažniausiai naudingi gali būti šie (4 lent.).

4 lent. Pagrindiniai Metodo *getColumns* rezultatų stulpeliai

Stulpelio pavadinimas	Tipas	Aprašymas
TABLE_CAT	*String*	Lentelės katalogas
TABLE_SCHEM	*String*	Lentelės schema
TABLE_NAME	*String*	Lentelės pavadinimas
COLUMN_NAME	*String*	Lauko pavadinimas

DATA_TYPE	*int*	SQL duomenų tipo kodas. Kodus galima sužinoti bibliotekoje *java.sql.Types*
COLUMN_SIZE	*int*	Lauko ilgis. Eilutės tipo laukams – maksimalus galikų saugoti simbolių skaičius, skaitinio tipo laukams – tikslumas
DECIMAL_DIGITS	*int*	Dešimtainių skilčių skaičiuje kiekis
REMARKS	*String*	Pastabos, lauko aprašas
IS_NULLABLE	*String*	Gražinta reikšmė "NO" reiškia, kad laukas tikrai negali įgyti reikšmės NULL. "YES" – kad laukas gali įgyti NULL reikšmę. Tuščia – šios lauko savybės nustatyti nepavyko

3.3. Pirminių raktų, indeksų ir išorinių raktų nuskaitymas

Pirminių raktų nuskaitymui yra skirtas *DatabaseMetaData* klasės metodas *getPrimaryKeys*, indeksų nuskaitymui – *getIndexInfo*, o išorinių raktų ir su jais susijusių pirminių raktų nuskaitymui – *getCrossReference*. Dabar detaliau apie kiekvieną iš jų.

DatabaseMetaData klasės metodas *getPrimaryKeys* gražina nurodytos lentelės pirminį raktą. Skirtingai nuo anksčiau aprašytų metodu, šio metodo paramaterai yra ne šablonai, apraštantys lentelių grupę, o konkrečią lentelę nurodančios reikšmės (5 lent.).

5 lent. Metodo *getPrimaryKeys* parametrai

Paramentras	Tipas	Aprašymas
catalog	*String*	Katalogo pavadinimas
schema	*String*	Schemos pavadinimo šablonas
table	*String*	Lentelės pavadinimo šablonas

Metodas gražina *ResultSet* tipo objektą, kuriame kiekvienas pirminio rakto laukas atitinką vieną rezultato lentelės eilutę. Jei pirminis raktas sudarytas iš vieno lauko, tuomet *ResultSet* bus sudarytas iš vienintelio įrašo (6 lent.).

6 lent. Pagrindiniai Metodo *getPrimaryKeys* rezultatų stulpeliai

Stulpelio pavadinimas	Tipas	Aprašymas
TABLE_CAT	*String*	Lentelės katalogas
TABLE_SCHEM	*String*	Lentelės schema
TABLE_NAME	*String*	Lentelės pavadinimas
COLUMN_NAME	*String*	Lauko pavadinimas
KEY_SEQ	*short*	Atributo eilės numeris rakte
PK_NAME	*String*	Pirminio rakto pavadinimas

Metodas *getIndexInfo* yra skirtas nuskaityti lentelės indeksus. Šiam metodui reikia tų pačių parametrų kaip ir pirminio rakto nuskaitymo metodui, ir dar dviejų papildomų (7 lent.).

7 lent. Metodo *getIndexInfo* parametrai

Paramentras	Tipas	Aprašymas
catalog	*String*	Katalogo pavadinimas
schema	*String*	Schemos pavadinimo šablonas

table	*String*	Lentelės pavadinimo šablonas
unique	*boolean*	Jei nustatyta *true*, gražinami tik unikalumą garantuojantys indeksai
approximate	*boolean*	Jei nustatyta true, rezultatai gali būti apytikriai, jei nustatyta false, rezultatai turi būti tikslūs

Šis metodas gražina *ResultSet* tipo objektą, kuriame įrašais saugoma informacija apie indesų laukus. Jei indeksas sudarytas iš kelių laukų, kiekviena laukas bus išvestas atskirame įraše, kuriuose sutaps indekso pavadinimas ir kartu bus nurodyta lauko eilės numeris tame indekse (8 lent.).

8 lent. Pagrindiniai Metodo *getIndexInfo* rezultatų stulpeliai

Stulpelio pavadinimas	Tipas	Aprašymas
TABLE_CAT	*String*	Lentelės katalogas
TABLE_SCHEM	*String*	Lentelės schema
TABLE_NAME	*String*	Lentelės pavadinimas
NON_UNIQUE	*boolean*	Parodo ar indeksas neužtikrina unikalumo
INDEX_NAME	*String*	Indekso pavadinimas
TYPE	*short*	*tableIndexStatistic* – įraše su toia reikšme šiame lauke yra gražinama statistinę informaciją apie lentelę *tableIndexClustered* – klasterinis ideksas *tableIndexHashed* – maišos indeksas *tableIndexOther* – kitas indeksas
ORDINAL_POSITION	*short*	Lauko eilės numeris indekse
COLUMN_NAME	*String*	Lauko pavadinimas

Metodas *getCrossReference* yra skirtas ryšiams tarp dviejų lentelių nuskaityti. Šiam metodui būtina nurodyti abi lenteles, tarp kurių esančius ryšius reikia nuskaityti – būtina nurodyti kurioje lentelėje yra pirminis raktas ir kurioje išoriniai (9 lent.).

9 lent. Metodo *getCrossReference* parametrai

Paramentras	Tipas	Aprašymas
primaryCatalog	*String*	Lentelės su pirminiu raktu katologas
primarySchema	*String*	Lentelės su pirminiu raktu schema
primaryTable	*String*	Lentelės su pirminiu raktu pavadinimas
foreignCatalog	*String*	Lentelės su išoriniais raktais katologas
foreignSchema	*String*	Lentelės su išoriniais raktais schema
foreignTable	*String*	Lentelės su išoriniais raktais pavadinimas

Šis metodas gražina *ResultSet* tipo objektą, kuriame įrašais saugoma informacija apie ryšius, t.y. vienos lentelės išorinius raktų susiejimą su kitos lentelės pirminiu raktu. Dažniausiai ryšiai realizuojami vienu lauku, todėl ir ryšio atvejį dažniausiai atitiks vienas rezultato įrašas. Tačiau jei ryšys sudarytas iš daugiau nei vieno lauko, kiekvieną ryšio lauką atitinka vienas įrašas. Vieno ryšio atvejo įrašuose sutaps pirminio ir išorinio rakto pavadinimai, ir bus nurodytas lauko eilės numeris rakte (10 lent.).

10 lent. Pagrindiniai Metodo *getCrossReference* rezultatų stulpeliai

Stulpelio pavadinimas	Tipas	Aprašymas
PKTABLE_CAT	*String*	Pirminio rakto lentelės katalogas
PKTABLE_SCHEM	*String*	Pirminio rakto lentelės schema
PKTABLE_NAME	*String*	Pirminio rakto lentelė
PKCOLUMN_NAME	*String*	Pirminio rakto laukas
PK_NAME	*String*	Pirminio rakto pavadinimas
FKTABLE_CAT	*String*	Išorinio rakto lentelės katalogas
FKTABLE_SCHEM	*String*	Išorinio rakto lentelės schema
FKTABLE_NAME	*String*	Išorinio rakto lentelė
FKCOLUMN_NAME	*String*	Išorinio rakto laukas
FK_NAME	*String*	Išorinio rakto pavadinimas
KEY_SEQ	*short*	Lauko eilės numeris rakte

4. Išvados

1. Siekiant sėkmingai spręsti liktinių sistemų problemas, būtina turėti išsamią sistemos dokumentaciją. Jei ši dokumentacija prarasta, pasenusi arba jos niekada nebuvo, ją būtina atstatyti vykdant atvirkštinę duomenų inžineriją.
2. Pati svarbiausia ir pagrindinė sistemos dokumentacijos dalis yra duomenų struktūra, kadangi į ją remiasi visi kiti dokumentuojami sistemos aspektai. Duomenų struktūrą įprasta išreikšti logine schema.
3. JDBC bibliotekoje yra realizuotos visos būtinos priemonės loginei schemai nuskaityti. Metodai *getTables* ir *getColumns* naudojami visų duomenų bazėje esančių lentelių ir jų atributų nuskaitymui. Metodai *getPrimaryKeys, getIndexInfo* ir *getCrossReference* – lentelių priminiams raktams, idenksams ir ryšiams tarp lentelių nustatyti.

5. Literatūra

[1] **J. Bisbal, D. Lawless, B. Wu ir J. Grimson**. Legacy Information System Migration: A Brief Review of Problems, Solutions and Research Issues. *Technical Report TCD-CS1999-38*, 1999, prieiga per Internetą: <https://www.cs.tcd.ie/>.

[2] **V. Kontrimas** Objektinės orientacijos duomenų bazės, *Kompiuterija. 2001*, Nr. 2, p.39-40

[3] **G. Bakehouse ir T. Wakefield**. Legacy Information Systems. *ACCA services*, 2005, prieiga per Internetą: <http://www.accaglobal.com>.

[4] **Sun Microsystems**, JavaTM 2 Platform, Standard Edition, v 1.4.2 API Specification, prieiga per Internetą: <http://java.sun.com/j2se/1.4.2/docs/api/overview-summary.html>

[5] **Thomas B. Gendreau,** Computer Science Department University of Wisconsin - La Crosse, Using Java and JDBC in an introductory database course,