KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Darius Kasiulevičius

xBase komandų pakeitimo SQL užklausomis tyrimas

Magistro darbas

Darbo vadovas
Prof. R. Butleris

KAUNAS, 2009

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Darius Kasiulevičius

xBase komandų pakeitimo SQL užklausomis tyrimas

Magistro darbas

Vadovas
Prof. R. Butleris
2009-05-22

Konsultantas
Lekt. T. Danikauskas
2009-05-22

Recenzentas
Doc. dr. V. Pilkauskas
2009-05-21

Atliko
IFM 3/1 gr. stud.
D. Kasiulevičius
2009-05-18

KAUNAS, 2009

Turinys:

1. Įvadas

Tobulėjant šiuolaikinėms technologijoms, sparčiai auga poreikis senų kompiuterizuotų sistemų tobulinimui, atnaujinimui. Senos sistemos pasitarnauja kuriant naująsias šiuolaikines sistemas. Naujos sistemos gali būti kuriamos perpanaudojimo principu, imama palikuoninė sistema, peržiūrimi, išgaunami senieji reikalavimai, funkcionalumas, ir tada papildžius naujais reikalavimais, pakeičiama programinės įrangos naudojama technologija ir kuriama nauja sistema. Naujos sistemos kūrimui panaudojamos taip pat senas programinis kodas, kuris pasitarnauja kaip senus reikalavimus atitinkantis šablonas pagal kurį yra toliau kuriama naujoji sistema.

Pasaulyje bene pirmoji plačiai paplitusi duomenų bazių programavimo kalba ir technologija dBase gyvuoja dar ir iki šių dienų. Šios programavimo kalbos pagrindinis principas yra darbas su dideliais kiekiais duomenų, kuriuos pradėta saugoti lentelėmis į atskiras rinkmenas. Šitoks duomenų saugojimo ir naudojimo būdas buvo pavadintas - navigacinėmis duomenų bazėmis. Esamose duomenų bazėse buvo galima manipuliuoti visais duomenis su nedideliu kiekiu komandų, kurių įvairios kombinacijos buvo suprantamos ir lengvai perprantamos. Šiuo metu šitos programavimo kalbos ir duomenų bazių valdymo technologijos klonai praminta xBase. Po šiuo pavadinimu slepiasi komerciniai produktai, tokie kaip: Clipper iš GrafxSoft, Vulcan.NET iš GrafX Software, dBase iš dBASE Inc., FlagShip iš multisoft GmbH, xBase++ iš Alaska Software, xHarbour Builder iš xHarbour.com Inc., taip pat ir atviro kodo produktai: FoxPro iš Microsoft, Harbour Project, xHarbour.

Šiuo metu plačiausiai paplitusios yra reliacinės duomenų bazės, nes jų pagalba yra pasiekiami aukšti darbo našumo rezultatai. Reliacinės duomenų bazės įgalina naudotis kliento – serverio modeliu, kur duomenis apdoroja serveris, o užklausas serveriui pateikia klientas. Naudojantis navigacinėmis duomenų bazių sistemomis toks modelis negalimas, todėl kompiuteriniai tinklai yra apkraunami duomenimis, kuriuos apdoroja vartotojų kompiuteriai. xBase programavimo kalbose nesenai atsirado galimybė realizuoti kliento – serverio modeliu pagrįstą programinę įrangą.

Yra sukurta keletas sprendimų kaip xBase komandomis parašyta programinė įrangą priversti dirbti kliento – serverio modeliu. Tai Advantage Database Server programinė įranga serveriui, kuri optimizuoja komandas ir jas vykdo serveryje, grąžindamos tik rezultatus, bet naudojasi vis tiek tomis pačiomis komandomis ir rinkmenomis kaip navigacinės duomenų bazės. Antras sprendimas - xHarbour SQLRDD biblioteka, kuri programinės įrangos veikimo metu xBase komandas keičia į SQL užklausas ir jas siunčia reliacinių duomenų bazių valdymo serverinei programinei įrangai.

Šio darbo tikslas – sukurti metodą kaip efektyviai ir greitai transformuoti Harbour programavimo kalbos programinį kodą pritaikant naujas technologijas. Esminis darbo uždavinys –

sukurti programą, kuri padėtų pereiti iš xBase tekstinių duomenų bazių komandų prie SQL reliacinės duomenų bazės užklausų ir optimizuoti užklausas. Norint pasiekti išsikeltą tikslas ir įgyventinti suformuluotą uždavinį reikia išanalizuoti dažniausiai pasitaikančias xBase komandų konstrukcijas, išanalizuoti SQL užklausų optimizavimo būdus. Analizė rodo, kad iš pačių paprasčiausių xBase komandų konstrukcijų yra kuriamos programinės įrangos kodas, todėl užtenka aprašyti tik šių komandų transformavimo būdus.

Išanalizavus prieš tai paminėtus sprendimo būdus pasiūlytas metodas xBase komandas transformuoti į SQL užklausas, taip tiesiogiai pereiti prie reliacinių duomenų bazių nesinaudojant papildomomis programomis. Siūlomas metodas padeda programuotojams transformuoti esamą xBase komandomis pagrįsta programinį kodą automatizuojant transformavimą.

Projektavimo etape pagal pasiūlytą metodą buvo sukurta programinės įrangos prototipas, kuris leido automatizuoti programinio kodo transformavimo darbus. Taip pat įdiegtas papildomas funkcionalumas, kuris leidžia perkelti duomenis iš navigacinės duomenų bazės į reliacines duomenų bazes.

Eksperimento metu buvo tiriama ar xBase komandos yra vykdomos lėčiau nei SQL užklausos. Naudojantis lokalia duomenų baze, kurioje nebuvo naudojami indeksai xBase komandos buvo vykdomos greičiau. Perkėlus duomenų bazes į kitą tinklinį kompiuterį rezultatai pasikeitė, xBase komandų vykdymo laikas labai išaugo, o SQL užklausų nežymiai padidėjo. Eksperimentas buvo pakartotas imituojant realią duomenų bazę, kurioje buvo panaudoti indeksai, raktiniai laukai, rezultatas gautas kaip ir tikėtasi SQL užklausos vykdymo laikas buvo trumpesnis nei xBase komandų.

Darbo rezultatų patikrai, skaičiavimams ir programinės įrangos prototipo kūrimui, panaudotos nemokamo atviro kodo programos Harbour 1.0.1 versija [11], Borland C++ Compiler 5.5 - versija [10] ir atviro kodo duomenų bazių valdymo sistema MySQL 5.1 versija [12].

Darbo rezultatai pristatyti ir aprobuoti 14-joje tarpuniversitetinėje magistrantų ir doktorantų mokslinėje konferencijoje „Informacinės technologijos“

- Kasiulevičius D. Danikauskas T., xBase komandų keitimas SQL užklausomis, 14-osios tarpuniversitetinės magistrantų ir doktorantų mokslinės konferencijos „Informacinės technologijos“ pranešimų medžiaga, gegužės 8 d., Kaunas, 2009.

2. xBase programiniai sprendimai

2.1. xBase duomenų bazės istorija ir vizija

dBase buvo pirma pažangi duomenų bazės programa personaliniams kompiuteriams, todėl dBase rinkmenos formatas tapo standartu. Pirmoji versija buvo sukurta CP/M operacinei sistemai, o vėliau pritaikyta ir DOS sistemai. Šią programinę įrangą sukūrė 1978 metais C. Wayne Ratliff, kuris dirbo Jet Propulsion Laboratories Pasadenoje, Kalifornija atvaizduotas 2.1 paveiksle. Programos pirmas pavadinimas buvo Vulcan, kuri turėjo didelį pasisekimą, nes galėjo sukurti lenteles, išsaugoti ir atvaizduoti duomenis, o svarbiausia sugebėjo sukurti grynas ASCII programas, kaip DOS komandų failai.

Vėliau, 1981 metais, kai susikūrė Ashton-Tate kompanija, produktas pervadintas į dBase II. dBase III PLUS versijoje atsirado pirmą kartą iškrentantis meniu, kuris padėjo pasiekti visą programinės įrangos funkcionalumą. Milijonai žmonių naudojo tai kasdien. dBase nebuvo tiktai duomenų bazių valdymo programa, tai buvo taip pat ir programavimo kalba. Su jos dideliu komandų kiekiu, buvo kuriamos netik grynos duomenų bazių aplikacijos, bet ir buhalterinės ar net mokslinės aplikacijos. Kadangi dBase programavimo kalbą lengva išmokti, tai milijonai žmonių tapo programuotojais, to patys nežinodami. Šis populiarumas leido atsirasti dBase kompiliatoriams, kurie buvo kuriami namų sąlygomis. Taip atsirado dBXL, QuickSilver, Arago, Clipper, Force ir kt., kurie leido sutransliuoti dBase III PLUS programas į DOS EXE rinkmenas.

Lygiagrečiai dBase projektui atsirado FoxBase klonas, kuris buvo greitesnis nei originali dBase sistema. Ashton-Tate (A-T) bandė nupirkti, bet derybos žlugo, po to padavė į teismą dėl programinės įrangos autorinių teisių pažeidimo. Kovodama teismuose dBase atsiliko nuo konkurentų, ir 1991 metų rudenį kompaniją buvo parduota Borland‘ui. O 1999 metais Borland produktą pardavė vienam iš savo didžiausių klientų, kuris pelnėsi iš programinės įrangos kūrimo su dBase programine įranga. Buvo sukurta bendravimo sistema, internetinis biuletenis, tam kad būtų užtikrintas tolimesnis programinės įrangos palaikymas, o nesenai buvo išleista nauja versija, kuri taip pat pritaikyta naujai operaciniai sistemai Vista. Programos detali kitimo raida pateikiama 2.1 lentelėje, kur matome, kad dBase gyvuoja iki šiol.

1985 metais Nantucket korporacija sukūrė Clipper kaip dBase III kompiliatorių, kuris vėliau parduotas Computer Associates. Clipper buvo tiesiog įrankis padedantis greičiau parašyti dBase programas, bet Clipper 5 buvo esminis persilaužimas xBase kalbos istorijoje. Clipper 5 tapo Application Development System, tikra programavimo kalba. Clipper 5 koncepcija buvo pasiskolinta iš C ir kitų programavimo kalbų. Sukurtos programos kompiliuojamos į vieną

monolitinę vykdomąją rinkmeną EXE, tokiu būdu programos kodas yra saugus ir gali būti vykdomas įvairiose operacinėse sistemose: DOS, Windows, OS/2 ir kt. Sukurtai programai kompiuterio atminties trūkumas nekelia problemų, nes turi pažangią virtualios atminties valdymo sistemą, kuri panaši į Windows. Programos gali dirbti, tiek seniausiuose kompiuteriuose XT („IBM Personal Computer XT“ sukurtas 1983 metais), tiek naujausiuose Pentium. Toje pačioje versijoje buvo įgyvendinta „Replaceable Database Driver“ (RDD) technologija, kuri leido kūrėjams įjungti skirtingus duomenų apdorojimo variklius nuo senųjų DBF rinkmenų iki naujausių SQL duomenų bazių serveriuose. Naujoji technologija leido tuo pačiu metu naudoti kelis skirtingus duomenų apdorojimo variklius.

2.1 lentelė **dBase ir Clipper vystimosi raida**

Pavadinimas	**Bitai**	**Metai**	**OS**
dBASE II	16	1981	CP/M
dBASE II	16	1982	DOS
dBASE III	16	1984	DOS
dBASE III PLUS	16	1986	DOS
dBASE IV	16	1988	DOS
dBASE 5	16	1994	Windows
Visual dBASE 5.5	16	1995	Windows
Visual dBASE 5.7	16	1996	Windows
Visual dBASE 7.0	32	1997	Windows
Visual dBASE 7.5	32	1999	Windows
dB2K	32	2000	Windows
dBASE Plus	32	2002	Windows
dBASE Plus 2.2v	32	2003	Windows
dBASE Plus 2.5v	32	2004	Windows
dBASE Plus 2.6v	32	2005	Windows
dBASE Plus 2.61.5v	32	2008	Windows

Pavadinimas	**Metai**
Nantucket Clipper Winter'84	1985
Nantucket Clipper Summer'85	1985
Nantucket Clipper Winter'85	1986
Nantucket Clipper Autumn'86	1986
Nantucket Clipper Summer'87	1987
Nantucket Clipper 5.00	1990
Nantucket Clipper 5.01	1991
Nantucket Clipper 5.01a	1992
CA Clipper 5.20	1993
CA Clipper 5.2a	1993
CA Clipper 5.2b	1993
CA Clipper 5.2c	1993
CA Clipper 5.2d	1994
CA Clipper 5.2e	1995
CA Clipper 5.30	1995
CA Clipper 5.3a	1996
CA Clipper 5.3b	1997

Clipper kalba toliau yra plėtoja daug organizacijų, vienos kuria GNU Bendros Viešosios Licencijos pagrindu, tokios kaip CLIP, Harbour, xHarbour, kitos kuria komercinius projektus XBase++, FlagShip. Daugelis projektų yra pritaikyti DOS, Windows, Linux, Unix ir Mac OS X platformoms, kurie palaiko daug RDD, tokius kaip DBF, DBTNTX, DBFCDX (kuriuos naudoja FoxPro, SuccessWare, Comix), MachSix (SIx driver ir Apollo), SQL ir daug kitų. Nauji kūrėjai stengiasi palaikyti pilną suderinamumą su dBase/xBase sintaksės standartu, siūlydami objektiškai orientuotą programavimą.

Harbour yra nemokamos programinės įrangos kompiliatorius xBase komplekto kalbai, dažnai vadintai Clipper. Kompiliatorius geba kompiliuoti ir veikti tokiose operacinėse sistemose, kaip MS DOS, MS Windows, OS/2, GNU/Linux ir Mac OSX. Tolimesniam tyrimui bus naudojamas Harbour kompiliatorius, nes jis yra nemokamas ir grindžiamas xBase programavimo kalba.

2.2. Reliacinės duomenų bazės istorija ir vizija

Kiekvienas verslas turi duomenis, kuriems reikalingas organizuotas metodas ar mechanizmas tam, kad gerai ir greitai ekpslotuoti duomenis. Šis mechanizmas vadinamas duomenų bazių valdymo sistema (DBVS, database management system - DBMS. Duomenų bazių valdymo sistemos buvo pradėtos kaip plokščios rinkmenos išdėstymas centriniame kompiuteryje. Šiuolaikinė informacijos vadybos banga yra pirmiausiai pritaikyta vartojimui reliacinės duomenų bazės valdymo sistema (RDBVS, relational database management system - RDBMS), gautas iš tradicinio DBVS. Šiuolaikinė duomenų bazė remiasi kliento - serverio ir interneto technologijomis, kurios yra tipiškos kombinacijos naudojamos šiuolaikinėse firmose, kad sėkmingai valdytų duomenis.

1974 metais IBM sukūrė prototipinę reliacinę duomenų bazės valdymo sistemą System R. 1974 metais System/R projektas buvo užbaigtas, bet projekte buvo įgyvendintu du reikšmingi pasiekimai. Buvo įrodyta pasauliui, kad reliacinis duomenų modelis yra įgyvendinamas. Taip pat projektas pastūmėjo struktūrizuotos užklausų kalbos plėtojimuisi. Artėjant System/R projekto pabaigai, IBM įgyvendino kalbą, kuri palaikė kelių lentelių užklausas ir daugialypę vartotojų prieigą. Naująją kalbą pavadino struktūrizuota angliškų užklausų kalba - Structured English Query Language - SEQUEL. Vardas vėliau buvo sutrumpintas į struktūrizuota užklausų kalba - Structured Query Language - SQL.

Grupė inžinierių, stebėjusių System/R, numatė reliacinių duomenų bazių potencialą, sukūrė kompanija, pavadinta Relational Software, Inc. 1979, jie pagamino pirmą komercinę reliacinės duomenų bazės valdymo sistemą, kurioje realizavo SQL kalbą. Sistema buvo pavadinta Oracle.
SQL kaip kalba turėjo konkurentus žinomus kaip QUEL, kurią naudojo Ingres RDBVS. 1980 metais Ingres nebeišlaikė konkurencijos sąlygų ir pradėjo naudoti SQL kalbą. Tuo tarpu IBM toliau tęsė savo tyrimo darbus System/R pagrindu, kurios pasėkoje atsirado SQL/Data System (SQL/DS), o vėliau ir Database 2 (DB2). Kur pastaroji versija tapo SQL kalbos standartu.

2.2 lentelė **SQL standartų priėmimo istorija**

Metai	Pavadinimas	Pastaba
1986	SQL-86	Priimtas ANSI, 1987 patvirtintas ISO.
1989	SQL-89	Nedideli pataisymai
1992	SQL-92	Dideli pakeitimai, naujas ISO standartas 9075.
1999	SQL:1999	Pridėtos rekursinės užklausos, trigeriai, neskaliariniai tipai ir kai kurios į objektą orientuotos ypatybės.
2003	SQL:2003	Įvestos XML-susijusios ypatybės, lango funkcijos, standartizavo sekos ir skiltis (stulpelius) su automatiškai sukurtomis vertėmis.
2006	SQL:2006	9075-14:2006 ISO/IEC apibrėžia būdus, kuriuose SQL gali būti panaudotas sujungiant su XML. Apibrėžia būdus importuoti ir sukaupti XML duomenis SQL duomenų bazėje.

Nuo tada rinkoje pasirodė daug reliacinių duomenų bazių valdymo sistemų, kurių pagrindinė kalba buvo SQL. Kai tik tapo matoma, kad reliacinės duomenų bazės neapseina be SQL, ANSI (American National Standards Institute) pradėjo darbą standarto kūrime. 1986 metais IBM sukurta SQL kalba tapo standartine reliaciniu duomenų bazių komunikavimo kalba. 1987 metais tarptautinė standartų organizacija – ISO (International Standards Organization) priėmė ANSI SQL standartą. Po to standartas buvo keičiamas ir papildomas, dabar naujausias standartas vadinasi SQL:2006. Detalus standartų priėmimo sąrašas pateikiamas 2.2 lentelėje.

2.3. xBase duomenų bazės problematika

xBase yra bendras terminas visoms programavimo kalboms, kurios kilo iš originalios dBase (Aston-Tate) programavimo kalbos. xBase ir kitų panašių produktų esminė problema tai kad jie nebuvo pagrįsti kliento-serverio modeliu. Tai yra kai duomenų bazę naudoja tinkle daugybė vartotojų, sistema parsisiunčia visas rinkmenas į vartotojo kompiuterį, kur ir atlieka užklausas. Taip tinklas yra labai apkraunamas [4], nes kiekvieną kartą atliekant vis tas pačias užklausas rinkmenos yra siunčiamos per tinklą. Kliento – serverio modelyje, klientas siunčia nedideles užklausų komandas į serverį, kur serveris įvykdęs užklausas gražina tik rezultatą. Taip tinklo apkrovimas yra sumažinamas.

Ed Esber ir Bill Gates Niujorko spaudos konferencijoje pristatė SQL Serverį - „Advantage Database Server“ pasauliui, kurio pagrindinė idėja buvo panaudoti SQL Serverį kaip galinę programinę įrangą ir xBase kaip pradinę programinę įrangą. Taip leisdama egzistuojančiai xBase rinkai panaudoti jau sukurtas formas ir programuoti SQL pagrindu.

„Advantage Database Server“ programinė įranga buvo suprojektuota kompanijos Sybase, kurios dėka programos pradėjo dirbti remiantis serverio - kliento modeliu. Produktas buvo kuriamas nepriklausomiems programinės įrangos pardavėjams, komerciniams programinės įrangos kūrėjams ir mažoms - ir vidutinio dydžio firmoms. Pabrėžiamos produkto savybės:

- Lanksti duomenų prieiga per reliacines SQL užklausas, ar per tiesioginės navigacijos duomenų bazių komandas.
- Optimizuota duomenų prieiga visoms kūrimo aplinkoms: Delphi, Visual Studio, Visual Objects, Visual Basic, FoxPro ir kt.
- Nėra arba minimalus administravimas, lengva įdiegti ir sukonfigūruoti — nereikalingas duomenų bazės administratorius, kurio dėka sumažėja išlaidos.
- Veikia serverio transakcijų apdirbimas, taip mažindamas duomenų bazių gedimus ir išlaidas priežiūrai.
- Įgalina serverio pagrįstą transakcijų apdirbimą, mažindamas duomenų bazės gedimus, drastiškai mažindamas palaikymo kainas.

- Palaiko duomenų bazių saugumą ir šifravimą.
- Leidžia naudotis lokaliai, tiesioginės prieigos (P2P), kliento – serverio modeliais pagrįsta aplinka – vienos programinės įrangos veikimo metu.

Serveris palaiko visus xBase rinkmenų formatus, kurie yra toliau aptariami. xBase turi keturis skirtingus rinkmenų tipus:

- Duomenų failas – *.DBF, pagrindinis duomenų rinkmena. Be duomenų rinkmenos - nėra duomenų bazės.
- Indekso rinkmenos – *.IDX, *.NDX, *.CDX.
- Pilno teksto rinkmenos – *.DBT. Tai Memo laukų rinkmenos.
- Aprašomosios rinkmenos – įvairios rinkmenos aprašančios etiketes (labels), kintamuosius, serverio išraiškoms ir kt.

Kiekviena lentelė yra saugoma atskirose *.DBF rinkmenose. Kiekviena lentelė gali turėti keletą indeksų pagal skirtingus raktus, vienos lentelės visi indeksai saugomi vienoje rinkmenoje *.CDX. Dažniausiai vienos lentelės duomenų, indeksų ir pilno teksto rinkmenos turi tokius pat pavadinimus. Panagrinėkime, kodėl xBase duomenų bazės yra ribotų galimybių. Vienas iš pagrindinių argumentų yra tai, kad duomenų ir indeksų rinkmenų struktūros yra griežtai apibrėžtos[3]. Kiekvienas saugomas bitukas duomenų rinkmenose turi savo reikšmę ir įtaką duomenims. Niekaip neišeis praplėsti laukų vardus, nes kiekvienam lauko pavadinimui yra skiriama 10 baitų. Tokie apribojimai labai sumažina galimybes visapusiškai dirbti ir plėsti programinės įrangos funkcionalumą.

Programinės įrangos kūrėjas xHarbour siūlo SQLRDD biblioteką, kurios pagalba xBase komandos yra keičiamos į SQL kodą programinės įrangos vykdymo metu – realiu laiku. Programinės įrangos gamintojų teigimu SQLRDD bibliotekos privalumai [14]:

- SQLRDD vienintelis įrankis rinkoje, kuris xBase komandomis parašytai programinei įrangai leidžia jungtis prie įvairių duomenų bazių, tokių kaip: MySQL, Postgres, Firebird, Oracle, Microsoft SQL, Cache, ADABAS, Sybase, IBM.
- Nereikalingos papildomos programos serverio ar kliento pusėje. Taip pat tas pats programinis kodas veikia Linux ir Windows aplinkose.
- Reikalingi tik nedideli pakeitimai programiniame kode: deklaruoti RDD, apibrėžti prisijungimo funkciją prie duomenų bazės.
- Supranta visas tradicines xBase komandas, indeksavimo ir lentelių sukūrimo.
- Naudojami gimtieji duomenų tipai ir indeksai.
- Duomenų bazės gali būti prieinamos su kitomis kalbomis ir aplikacijomis.

2.4. Duomenų bazių valdymo sistemos parinkimas

Prieš bandant pereiti prie SQL užklausų reiktų palyginti duomenų bazės valdymo sistemas, kuri prilygtų ar net pralenktų xBase galimybes. Daug darbų [1][2] nagrinėja vis tuos pačius duomenų bazių skirtumus ir panašumus, todėl tik trumpai 2.3 lentelėje bus pateikti esminiai skirtumai tarp xBase, MySQl ir MS-SQL. Kaip matoma iš lentelės bet kuri nauja SQL duomenų bazių valdymo sistema lenkia xBase galimybes.

2.3 lentelė ***MySQL ir MS-SQL palyginimas***

Ypatybės	**Microsoft SQL Server 2005**	**MySQL 5.0**	**xBase**
Operacinė sistema	Windows XP, Windows 2000+	Windows, Linux, Unix, Mac	Visos
Licenzija	Komercinis produktas – uždaras projektas.	GPL Atviro kodo, komercinis.	Atviro kodo
Daliniai indeksai – pvz. tik tiems įrašams kurių reikšmes ne NULL	Ne	Ne	Taip
Išoriniai raktai - Cascade Update/Delete	Taip	Taip (InnoDB),bet ne MyISAM	Ne
Nejautrus didžiosioms raidėms	Pagal nutylėjimą nejautrus, bet gali būti pakeistas.	Pagal nutylėjimą nejautrus	Nejautrus
Eilutės /Automatinė numeracija	Taip	Taip	Ne
Stulpelio pavadinimo ilgis	128	64	10
Indexo vardo ilgis	128	64	10
Lentelės vardo ilgis	128	64	8 – suderinimui su DOS
Maksimalus stulpelių skaičius lentelėje	1024	3398	1024

Duomenų bazės yra skirstomos į:

- mažas DB (~10 000 įrašų arba > 1 GB);
- vidutines DB ($1*10^7$ įrašų arba < 10 GB);
- dideles DB ($1*10^9$ įrašų arba 1 TB);
- labai dideles DB ($1*10^9$ įrašų arba 100 TB);
- duomenų saugyklos (>$1*10^9$ įrašų ir > 100 TB);

Šiomis dienomis Lietuvos rinkoje dominuoja vidutinės duomenų bazės. Todėl patenkinti poreikiams pakanka MySQL duomenų bazės valdymo sistemos [13].

2.5. Reliacinės duomenų bazės užklausų optimizavimas

Duomenų bazės struktūra turi įtakos užklausos vykdymo laikui, bet tuo pačiu ir pačios užklausos struktūra įtakoja vykdymo laiką. Užklausų optimizavimas yra svarbus veiksnys užtikrinantis greitą duomenų išrinkimą [6]. Panaudojant įvairios sintaksės užklausas pasiekiamas skirtingas užklausos vykdymo laikas. Dinamiškai kuriamos užklausos dažnai neprilygsta užklausoms, kurias sukuria programuotojai. Statinės užklausos dažnai būna ištestuotos įvairiais aspektais ir pritaikytos konkrečiai duomenų bazei taip, kad veiktų greičiausiai.

Kiekvienos duomenų bazės valdymo sistemos turi savo SQL užklausų vykdymo metodus [7] [8], bet patys pagrindiniai žingsniai yra šie:

- SQL užklausos gramatinis nagrinėjimas. Užklausa yra dalina į individualius žodžius ir tikrinama jų sintaksė. Taip surandama loginės ir gramatinės klaidos.
- Toliau užklausoje yra tikrinama pagal informacine duomenų bazės schema. Patikrinama ar visos lentelės egzistuoja duomenų bazėje ar turi tam reikalingas privilegijas. Taip pat peržiūrima ar visi laukai paminėti užklausoje egzistuoja toje lentelėje.
- Kitas žingsnis tai užklausos optimizavimas. Kiekviena skirtinga duomenų bazių valdymo sistema turi savo optimizavimo metodus, bet visos laikosi pagrindinės idėjos. Yra apsprendžia kuria tvarka geriau sujungti lenteles, kada naudoti ir kuriuos indeksus. Jei užklausa parašyta tinkamai tai šita optimizacija pagerina užklausos vykdymą.
- Toliau sistema sukuria vykdymo planą, kuris yra taikomas atsižvelgiant į prieš tai žingsnyje padarytus sprendimus. Vykdymo planas tai binarinis optimizuoto komandų vykdymo atvaizdas.
- Paskutinis žingsnis tai prieš tai žingsnyje sukurtų binarinių sekų vykdymas ir rezultatų grąžinimas.

SQL užklausa tai taisyklių rinkinys kurį supranta duomenų bazių valdymo sistemos ir pagal tai išrenka reikalingus duomenis iš duomenų bazės. SQL užklausa nusako kokie duomenys turėtų būti išrinkti, bet nenusako sistemai kokiais metodais duomenys turi būti ieškomi duomenų bazėje. Vadinasi, kad duomenų bazių valdymo sistema parinktų optimalią strategiją įgyvendinti užklausai, būtina atlikti užklausos pertvarkymą. Kai kuriais atvejais duomenų valdymo sistema negali nustatyti optimalios strategijos. Prielaidos daromos iš realiame gyvenime pasitaikomų užduočių ir užklausų joms įgyvendinti. Tai leidžia suprasti, kad iš esmės panašios užklausos, kuriose padaryti tik nedideli sintaksės ir struktūros pakeitimai, gali būti vykdomas skirtingą laiko tarpą. Viena iš paprasčiausių užklausų vykdymo laiko sutrumpinimo strategija tai naudoti duomenų bazės lentelių laukų indeksus, taip pat ir sudėtinius laukų indeksus. Indeksai iš esmės pagreitina duomenų atrinkimą iš lentelės. Tai taikoma bet kokiose duomenų bazėse tiek tekstinėse, tiek reliacinėse. Bet tai taip įprasta, kad negalvojama jog tai leidžia koreguoti užklausų vykdymo laiką.

Duomenų bazių darbo našumą taip pat lemia ir teisingai užrašyta užklausa. Apžvelgsime kelis užklausų optimizavimo aspektus. Vienas iš pagrindinių metodų tai kurti kuo paprastesnes užklausas. Jei užklausos bus paprastesnės tai ir vykdymas bus kur kas greitesnis. Paprasta užklausa leidžia sistemai priimti palankiausia vykdymo eigą. Bet paprastumas turi ir trūkumų: paprastos užklausos bus vykdomos greitai, bet kiekvieną kartą sistemai reiks prisijungti atsijungti nuo duomenų bazės, o tai yra labai neefektyvu [7][4]. Todėl turėtų būti pasirinktas tarpinis variantas.

Užklausa SELECT susideda iš kelių dalių, pradžioje nurodoma kokį rezultatą norime matyti, po to nurodome lentelę iš kurios rinksime ir tada sąlygas pagal kurias rinkti. Pati paprasčiausia užklausa atrodo taip:

```
SELECT * FROM valstybes;
```

Tačiau literatūroje rašoma [8] kad užklausa bus vykdoma greičiau, jei išvardinsime visus laukus:

```
SELECT pavadinimas, plotas, gyv_sk, valiuta FROM valstybes;
```

Jei rezultate norime matyti ne visus laukus tai užklausa tikrai bus įvykdyta greičiau. Jei išvardinti laukai dar turės sudėtinį indeksą, tai duomenis rinks ne iš lentelės, o iš indekso, kur veiksmai atliekami kur kas greičiau.

Naudojant filtravimo sąlygas, reiktų stengtis kad užduodamos sąlygos būtų vykdomos indeksų pagalba:

```
SELECT * FROM valstybes WHERE pavadinimas LIKE `L%`;
SELECT * FROM valstybes WHERE pavadinimas NOT LIKE `L%`;
```

Pirmoji užklausa bus vykdoma kur kas greičiau, nes užklausa eis per indeksus ir tikrins sąlygą, o antruoju atveju eis per visą lentelę ir atrinks tik tinkančias eilutes. Jei naudojami laukai visi turi indeksus tai geriausia pradžiai parašyti tą sąlygą, kurios rezultatas bus mažiausia aibė:

```
SELECT * FROM pacientai WHERE lytis = `M` AND amzius = `80`;
SELECT * FROM pacientai WHERE amzius = `80` AND lytis = `M`;
```

Tarkime šiuo atveju lentelė „pacientai“ turi indeksus pagal lytį ir pagal amžių, bet užklausų vykdymo laikas skirsis. Ta užklausa kurios pirmoji išrinkta aibė bus mažesnė ta ir bus greičiau atlikta. Tarkim, kad pacientų pagal lytį turime po lygiai 50%, o ligonių kurių amžius yra 80 turime tik 5%. Tai iš duomenų bazės išrinkus pirmiausia tuos įrašus kurie yra pagal amžių, o po to iš atrinktų įrašu išrinkti tinkančius pagal lytį.

Dažnai rezultatą norime matyti surikiuota tam tikra tvarka, todėl naudojam ORDER BY. Jei rezultate norėsime matyti kelis laukus, tai indeksas nepadės. O jei norėsime matyti tik tą lauką, kuris yra indeksuotas tai nereikės net rikiuoti nes ir taip rezultatas bus renkamas iš indekso, o jis visada būna surikiuotas.

```
SELECT pavadinimas, plotas FROM valstybes ORDER BY pavadinimas;
SELECT pavadinimas FROM valstybes ORDER BY pavadinimas;
```

Pirmoji užklausa bus rikiuojama, nes išrenkamos visos eilutės, o antroji užklausa nebūtina net rikiuoti nes indeksas ir taip būna surikiuotas.

Rašant užklausas reikia stengtis naudoti kuo paprastesnes konstrukcijas:

SELECT pirkėjo_nr, SUM(balansas) FROM pirkėjai GROUP BY pirkėjo_nr HAVING pirkėjo_nr < 10;

Šiuo atveju iš lentelės bus renkami visi įrašai, sumuojami, grupuojami ir pabaigoje atrenkami tik tie kurių pirkėjo numeris mažiau už dešimt. Tą patį galima užrašyti pasinaudojant paprasčiausia WHERE sąlyga:

SELECT pirkėjo_nr, SUM(balansas) FROM pirkėjai WHERE pirkėjo_nr < 10 GROUP BY pirkėjo_nr;

Šiuo atveju sistema net neįtrauks tokių įrašų kurių pirkėjo numeris didesnis už dešimt. Užklausa bus įvykdyta greičiau, nes nereikės bereikalingo sumos skaičiavimo. Taip sutrumpinti užklausos mums neišeis jei norėsime parodyti tuos pirkėjus kurių balansas buvo virš šimto litų, nes reikės pradžiai vis tiek suskaičiuoti visų pirkėjų balanso sumą, o tik po to atrinkti pagal sumas, kuriuos įrašus palikti.

Naudojant funkcijas reikia stengtis taip kad jos nebūtų naudojamos laukų skaičiavimui:

SELECT * FROM pirkėjai WHERE pašto_kodas = TO_NUMBER('94002');

SELECT * FROM pirkėjai WHERE TO_CHAR(pašto_kodas) = '94002';

Pirmoji užklausa bus vykdoma greitai, nes prieš jai pradedant vykdyti yra įvykdoma vieną sykį pateikta funkcija. O antruoju atveju funkcija turės būti vykdoma kiekvienos eilutės laukui, o tai užtruks daugiau laiko.

JOIN sakinys gali būti pakeičiamas į WHERE sakinį, kai iš antrosios lentelės nereikia nieko grąžinti [8][9]:

SELECT r.region_id, r.region FROM region r JOIN country c ON(c.region_id = r.region_id);

SELECT r.region_id, r.region FROM region r WHERE EXISTS (SELECT region_id FROM country WHERE region_id = r.region_id);

Padarius net nedidelius pakeitimus pastebėsime kaip užklausos pradės veikti greičiau. Nedidelėse duomenų bazėse iki kelių milijonų įrašų, optimizavimas nesijaus. Todėl šitie patarimai taikytini, arba didelėms užklausoms, arba didelėms duomenų bazėms.

2.6. Įrankio kūrimo perspektyvos analizė

xBase komandų pakeitimo SQL užklausomis įrankis turėtų padėti greitai ir patogiai pakeisti senas, bet dar plačiai naudojamas xBase komandas naujesnėmis, patogesnėmis ir populiariomis SQL užklausomis. Šiuolaikinės didelės sistemos remiasi duomenų bazėmis, o dirbti su xBase kaip serverine duomenų bazę yra sudėtinga, nepatogu ir neleidžia vienu metu dirbti dideliam kiekiui vartotojų. SQL duomenų bazės yra pritaikytos dirbti tinkle su daugybę vartotojų. Įrankio pagalba būtų patogu, lengva, greitai pakeisti xBase komandas į SQL užklausas. Norint įgyvendinti

užsibrėžtą tikslą verta atlikti SWOT (SSGG – stiprybių, silpnybių, galimybių, grėsmių) analizę [5]. Ji leis nustatyti, kokia yra sėkmės ar nesėkmės galimybė.

2.6.1. Įrankio SWOT analizė

SWOT analizė – tai metodas, skirtas įvertinti verslo ar individualias stiprybes, silpnybes, galimybes ir grėsmes. Tiriami išoriniai ir vidiniai aplinkos faktoriai. Išoriniams priklauso galimybės ir grėsmės, o vidiniams – silpnybės ir stiprybės. Prieš pradedant SWOT analizę būtina apsibrėžti tikslą, kuris padės tolimesnėje eigoje. Suformulavus projekto tikslą, reikia surasti, sugalvoti, atpažinti projektą įtakojančius faktorius, kurie nustatomi remiantis keliais punktais:

- Stiprybės – organizacijos ar asmeninės savybės, padedančios pasiekti tikslą;
- Silpnybės – organizacijos ar asmeninės savybės, trukdančios siekti tikslo;
- Galimybės – išorinės sąlygos, padedančios siekti tikslo;
- Grėsmės – išorinės sąlygos, trukdančios pasiekti tikslą.

Remiantis aprašytais punktais, pirma nustatome tikslą – sukurti gerą, suprantamą įrankį keitimams atlikti. Remiantis pradinėmis nuostatomis, galima sakyti, jog šiai analizei atlikti bus remiamasi intelektinėmis savybėmis, kuriamo įrankio nauda kūrėjui bei įmonei užsakiusiai įrankį. Dėl to galime išskirti šias stiprybes, silpnybes, galimybes ir grėsmes:

Stiprybės:

- Įdomi naujojo programinio produkto tema;
- Programuotojo didelis noras, kurti naujus produktus;
- Pagalba iš įmonės darbuotojų;

Silpnybės:

- Maža programuotojo patirtis, kuriant aukšto lygio programas;
- Naujojo programinio produkto kūrimas nauja programavimo kalba;
- Mažas literatūros kiekis;
- Trumpas laiko tarpas, produkto įgyvendinimui;

Galimybės:

- Produktas pagelbėtų sutaupyti laiko keičiant programinį kodą iš xBase komandų į SQL užklausas;
- Galimybė parduoti – platinti nemokamai, tam kad programuotojas – įmonė taptų žinoma;
- Senų projektų atnaujinimas ir pardavimas kaip naujesnės versijos su spartesniu darbu, naujomis galimybėmis;
- Programuotojo tapimas naujosios srities ekspertu;

Grėsmės:

- Keičiamame programiniame kode, gal atsirasti sunkiai surandamų klaidų, kurios prailgins kodo keitimo darbus;
- Sudėtingumas greitai perprasti ir koreguoti kodą;
- Panašių produktų atsiradimas, konkurencingumo didėjimas;

Visi veiksniai yra patalpinami į SWOT analizės lentelę, jiems priskiriami tikrumo laipsniai - μ, įtakos koeficientai – c, bei įtakų dydžiai tarpusavyje. Visiems dydžiams išreikšti naudojami dydžiai pasiskirstę pagal normalinį dėsnį. Visi duomenys pateikiami 2.4 - oje lentelėje.

Įtakos laipsnis:

1 – labai stipri; 0,75 – stipri; 0,5 – vidutinė; 0,25 – maža; 0 – nėra įtakos.

2.4 lentelė **Veiksniai ir jų įtakos projektui (1)**

					Stiprybės			Silpnybės			
					1	2	3	1	2	3	4
					Įdomi tema	Noras kurti	Pagalba	Maža patirtis	Nauja kalba	Literatūros trūkumas	Laiko trūkumas
			μ		0,50	0,50	0,75	0,75	0,75	0,50	0,25
				c	0,25	0,75	0,75	0,50	0,25	0,75	0,75
Galimybės	1	Laiko sutaupymas	0,50	1,00		0,75	0,50				-0,50
	2	Galimybė parduoti	0,50	0,25		0,75	0,25				-0,25
	3	Projektų atnaujinimas	0,75	0,75	1,00		0,50	-0,25	-0,50		
	4	Eksperto tapimas	0,50	0,25	0,75	0,50		-0,50		-0,25	-0,25
Grėsmės	1	Klaidų atsiradimas	0,75	0,75	-0,25		-0,75	0,75		0,75	
	2	Sudėtinga koreguoti	0,50	0,50		-0,75	-0,25		0,50		
	3	Konkurencija	0,25	0,25				0,25			0,50

Lentelės pagalba yra suskaičiuojamos bendros projekto galimybės ir grėsmės. Suminės įtakos projektui padeda įvertinti bendrą projekto būklę ir tolimesnį jo vystimąsi. Sumines įtakas nesunku atvaizduoti grafike, pakeitus įtakų koeficientų reikšmes grafike galime pamatyti kaip kinta projekto grėsmės ir galimybės. Pirmosios ir antrosios formulių pagalba apskaičiuojamos suminės projekto grėsmės ir galimybės.

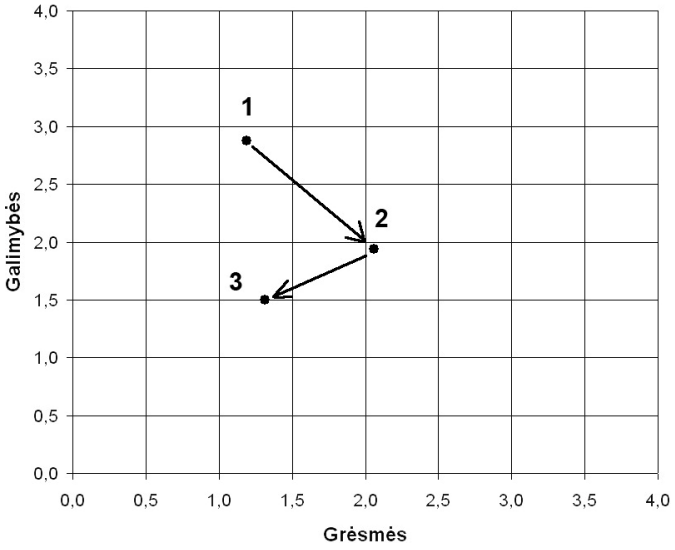
$$OP_{\Sigma} = \sum_{o=1}^{O} \left\{ \left(\mu_o + \sum_{s=1}^{S} ST_{os} + \sum_{w=1}^{W} WK_{ow} \right) C_o \right\}, \quad (1)$$

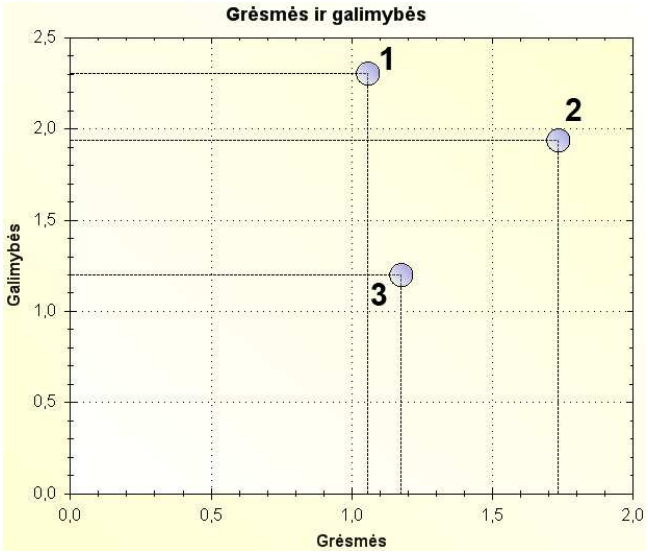
$OP_{\Sigma} = 2{,}88;$

$$TH_{\Sigma} = \sum_{t=1}^{T} \left\{ \left(\mu_t + \sum_{s=1}^{S} ST_{ts} + \sum_{w=1}^{W} WK_{tw} \right) C_t \right\}, \quad (2)$$

$TH_{\Sigma} = 1{,}19;$

Apskaičiuotos reikšmės yra atvaizduojamos 2.3 paveiksle. Paveikslai buvo sukurti naudojantis Microsoft Excel 2.3 paveiksle a) ir studentų Kęstučio Malinausko, Vytenio Bivainio sukurta programine įranga 2.3 paveikslas b). Keičiant projekto veiksnius matosi kaip keičiasi suminės projekto grėsmės ir galimybės. 2.5 ir 2.6 lentelėse pateikiamos projekto SWOT koeficientai su pakeistomis veiksnių įtakomis. Pakeistos įtakos dalyvauja skaičiuojant dinamines projekto grėsmes ir galimybes. Visi pakeitimai apskaičiuoti ir atvaizduoti grafikuose 2.2 pav. Visi grafikai ir lentelės leidžia nuspręsti ar programinį įrankį gaminti yra nenuostolinga.





a) Pagal pateiktas formules **b) pasinaudojant SWOT įrankiu**

2.2 pav. **Projekto grėsmės ir galimybės**

2.5 lentelė Veiksniai ir jų įtakos projektui (2)

			μ	c	Stiprybės 1 Įdomi tema	Stiprybės 2 Noras kurti	Stiprybės 3 Pagalba	Silpnybės 1 Maža patirtis	Silpnybės 2 Nauja kalba	Silpnybės 3 Literatūros trūkumas	Silpnybės 4 Laiko trūkumas
			μ		0,50	0,50	0,75	0,75	0,75	0,50	0,25
				c	0,25	0,75	0,75	0,50	0,25	0,75	0,75
Galimybės	1	Laiko sutaupymas	0,50	1,00		0,75	0,50	-0,50			-0,75
	2	Galimybė parduoti	0,50	0,25	0,25	0,50			-0,50		-0,50
	3	Projektų atnaujinimas	0,75	0,75	0,75		1,00	-0,25	-0,25	-0,25	-0,25
	4	Eksperto tapimas	0,50	0,25	0,50	0,75	0,25	-0,50		-0,50	
Grėsmės	1	Klaidų atsiradimas	0,75	0,75	-0,50	-0,25	-0,25	1,00	0,50	0,50	0,25
	2	Sudėtinga koreguoti	0,50	0,50		-0,50		0,25	0,50		0,25
	3	Konkurencija	0,25	0,25	-0,25			0,25			

2.6 lentelė Veiksniai ir jų įtakos projektui (3)

			μ	c	Stiprybės 1 Įdomi tema	Stiprybės 2 Noras kurti	Stiprybės 3 Pagalba	Silpnybės 1 Maža patirtis	Silpnybės 2 Nauja kalba	Silpnybės 3 Literatūros trūkumas	Silpnybės 4 Laiko trūkumas
			μ		0,50	0,50	0,75	0,75	0,75	0,50	0,25
				c	0,25	0,75	0,75	0,50	0,25	0,75	0,75
Galimybės	1	Laiko sutaupymas	0,50	1,00	0,75	0,25	0,50	-0,25	-0,50	-0,50	-0,25
	2	Galimybė parduoti	0,50	0,25		0,25		-0,50			-0,25
	3	Projektų atnaujinimas	0,75	0,75	0,75	0,50	0,25	-0,25	-0,75		
	4	Eksperto tapimas	0,50	0,25	0,50	0,25		-0,50	-0,25	-0,25	
Grėsmės	1	Klaidų atsiradimas	0,75	0,75	-0,50	-0,50	-1,00	1,00	0,50	0,75	0,25
	2	Sudėtinga koreguoti	0,50	0,50	-0,25	-0,50	-0,50	0,25	0,75	0,25	0,25
	3	Konkurencija	0,25	0,25		-0,25	-0,25	0,25			

Didžiausios projekto įgyvendinimo grėsmės yra nauja programavimo sritis, naujos literatūros trūkumas, kurias laikui bėgant kompensuoja programuotojų patirtis panašiose srityse. Dinaminė SWOT analizė parodo, kad laikui bėgant programuotojai tobulėja naujoje programavimo kalboje, todėl projekte neatsiranda naujų grėsmių ir tuo pačiu nedidėja esamų įtaka.

2.7. xBase komandos ir jų atitikmenys SQL

Naudojant DBF lenteles nereikia prisijungti prie serverio ar duomenų bazės, nes lentelės yra tiesiog atskiros rinkmenos. Programos pradžioje reiktų patikrinti ar įmanoma pasiekti aplankalą ar rinkmenas galima redaguoti (nėra tik skaitymo rėžime). SQL atveju reikia prisijungti prie serverio ir išsirinkti pačią duomenų bazę. Todėl toliau nagrinėjant pavyzdžius nenaudosime prisijungimo ir atsijungimo prie MySQL bazės, todėl pateiksime tik pirmą pavyzdį su prisijungimo ir atsijungimo komandomis.

2.7.1. Naujo įrašo įterpimas

Viena iš svarbių duomenų bazių galimybių tai naujų įrašų įterpimas. Įterpiant naujus įrašus duomenų bazės lentelė yra rakinama, tam kad keli naudotojai vienu metu nepradėtu rašyti duomenų į tą pačią vietą. Lentelės užrakinimas turi trukti kuo trumpesnį laiką, kad kiti vartotojai kuo greičiau galėtų prieiti prie duomenų.

```
FIELD cLaukas, nLaukas, dLaukas, lLaukas, mLaukas   //išvardiname visų naudojamų laukų pavadinimus
LOCAL lNewArea := .T., cRddName := "DBFCDX", cDatabase := "C:\bazes\lentele.dbf", cAlias := "AliasLentele", lShared := .T., lReadonly := .F.   //kintamieji kurių pavadinimai nusako prasmę
DbUseArea(lNewArea, cRddName, cDatabase, cAlias, lShared, lReadonly) //duomenų bazės lentelės atidarymas, su indekso ir memo laukų failais
IF FLock()      //užrakina duomenų bazės lentelės failą
  DbAppend()    //įterpia naują tuščią įrašą
//priskiriam laukams reikšmes
  cLaukas := "Pirmoji eilutė"
  nLaukas := 10.25
  dLaukas := STOD("20080529")
  lLaukas := .T.
  mLaukas := "Labai ilga eilutė ..."
  DbCommit()   //iš operatyviosios atminties surašome duomenis į rinkmeną
  DbUnlock()   //atrakiname duomenų bazės lentelės rinkmeną
ENDIF
DbCloseArea()  //uždaroma duomenų bazės lentelė
```

Pateiktas programinio kodo fragmentas gali būti užrašomas sutrumpinta forma nenaudojant kintamųjų:

```
DbUseArea(.T., „DBFCDX“, „C:\bazes\lentele.dbf“, „AliasLentele“, .T.,.F.)
IF FLock()
  DbAppend()
```

```
  FIELD -> cLaukas := "Pirmoji eilutė"
  FIELD -> nLaukas := 10.25
  FIELD -> dLaukas := STOD("20080529")
  FIELD -> lLaukas := .T.
  FIELD -> mLaukas := "Labai ilga eilutė ..."
  DbCommit()
  DbUnlock()
ENDIF
DbCloseArea()
```

Pateiktas programinio kodo fragmentas pasinaudojant SQL užklausas užrašomas taip:

```
LOCAL cBaseType := „MYSQL“, cHost := „www.serveris.lt“, cUser := "Vartotojas",
cPass := "Slaptazodis", cBase := "duombaze", cRddName := "SQLBASE", hConnection
//kintamieji kurių pavadinimai nusako prasmę
IF RDDINFO(RDDI_CONNECT, {cBaseType, cHost, cUser, cPass, cBase}, cRddName,
@hConnection) //prisijungimas prie duomenų bazės
RDDINFO(RDDI_EXECUTE, ;   /*Nurodom kad įvykdytų užklausą*/
    "set autocommit=0;" +;   /*Automatiškai nesurašyti duomenų iš buferio į
atmintį*/
    "start transaction;" +;   /*Pradedam transakciją*/
    "lock table lentele Write;" +;   /*Užrakiname lentelę*/
    "insert into lentele () Values();" +;   /*Įterpia naują tuščią įrašą*/
    "DO (select @id:=Last_insert_ID());" +;   /*Gauname paskutinį ID*/
/*Priskiriam laukams naujas reikšmes*/
    "update lentele set cLaukas='Pirmoji eilutė' where ID=@id;" +;
    "update lentele set nLaukas=10.25 where ID=@id;" +;
    "update lentele set dLaukas='20080529' where ID=@id;" +;
    "update lentele set lLaukas=1 where ID=@id;" +;
    "update lentele set mLaukas='Labai ilga eilutė ...' where ID=@id;" +;
    "commit;" +;   /*Patvirtiname transakciją*/
    "unlock tables;", ;   /*atrakiname lenteles*/
    cRddName, hConnection)
RDDI_INFO(RDDI_DISCCONECT, hConnection, cRddName)   //Atsijungiame nuo duomenų
bazės
```

Aukščiau pateiktas SQL kodo sprendimas atitinka tiesioginį xBase komandų pakeitimą. Pateiktas programinio kodo SQL sprendimas yra sutrumpinamas iki vienos užklausos, kurios supratimas yra lengvesnis, aiškesnis užrašymas, lengviau koreguojama, lengviau surandamos klaidos.

```
LOCAL cRddName := "SQLBASE", hConnection
IF RDDINFO(RDDI_CONNECT, {"MYSQL", „www.serveris.lt“, „Vartotojas“,
„Slaptazodis“, „duombaze“}, cRddName, @hConnection)
```

```
RDDINFO(RDDI_EXECUTE, " autocommit=1;" +;
„Insert Into `lentele` (cLaukas, nLaukas, dLaukas, lLaukas, mLaukas)
Values('Pirmoji eilutė', 11.25, ,20080529', 1, 'Labai ilga eilutė ...');“
cRddName, hConnection)
RDDI_INFO(RDDI_DISCCONECT, hConnection, cRddName)
```

2.7.2. Duomenų lentelės sukūrimas

Duomenys yra laikomi lentelėse, iš jų yra atrenkami reikalingi įrašai ir pateikiamos ataskaitos. Kiekvienos lentelės duomenys yra unikalūs ir būdingi tik tai lentelei. Trumpai aprašysim kaip sukuriamos naujos lentelės.

```
LOCAL aStruct := { ;
  { "cLaukas", "C", 50, 0 }, ;
  { "nLaukas", "N",  8, 2 }, ;
  { "dLaukas", "D",  8, 0 }, ;
  { "lLaukas", "L",  1, 0 }, ;
  { "mLaukas", "M", 10, 0 }}
DbCreate("lentele", aStruct, "DBFCDX", .T., "AliasLentele" )
```

Komandos DbCreate(): 1-as parametras nurodo lentelės pavadinimą, 2-asis parametras nurodo lentelės struktūrą, 3-asis parametras nurodo kokia naudojama RDD biblioteka, 4-tas parametras rodo ar atverti lentelę sukūrus, jei nėra parametro – neatidaro, jei .T. – atidaro naujoje srityje, jei .F. - atidaro einamoje srityje ir 5-asis parametras nurodo naujai atidaromos lentelės pavadinimą, kuriuo bus prieinama lentelė.

Pateiktas programinio kodo fragmentas SQL užklausomis užrašomas sekančiai. SQL kuriant lentelę nurodoma daugiau parametrų, kurie vėliau palengvina darbą su sukurta lentele koreguojant, įterpiant ir ieškant įrašų.

```
RDDINFO(RDDI_EXECUTE, "DROP TABLE IF EXISTS `lentele`;" +;
"CREATE TABLE `lentele` (" +;
"  `ID` int(10) unsigned NOT NULL auto_increment, " +;
"  `cLaukas` varchar(50) default NULL, " +;
"  `nLaukas` float default NULL, " +;
"  `dLaukas` date default NULL, " +;
"  `lLaukas` tinyint(1) default NULL, " +;
"  `mLaukas` blob, " +;
"  PRIMARY KEY  (`ID`)" +;
") ENGINE=InnoDB AUTO_INCREMENT=0 DEFAULT CHARSET=cp1257 " +;
"COLLATE=cp1257_lithuanian_ci; ") //nebūtina nurodyti komandos RDDINFO 3-io ir
4-to parametro. Trečią parametrą nustatome programos pradžioje su
RDDSETDEFAULT("SQLBASE"), o prisijungimą naudoją einamąjį.
```

2.7.3. Indekso kūrimas

Duomenų bazių valdymo sistemose būtina naudoti indeksus. Todėl kaip jie kuriami aprašysoma toliau. xBase komandomis indekso kūrimas atrodo taip:

```
OrdCreate("lentele", "IndeksasA", "DTOS(dLaukas) + cLaukas", {|| DTOS(dLaukas) + cLaukas }, .F.)   //4-tasis parametras parodo ar indeksas unikalus
```

SQL užklausa indekso sukūrimas atrodytų taip:

```
RDDINFO(RDDI_EXECUTE, "CREATE INDEX IndeksasA ON lentele (dLaukas, cLaukas);")
```

2.7.4. Įrašo koregavimas

Įrašų reikšmių koregavimas dažniausiai atliekamas tik tam tikrą sąlygą tenkinantiems įrašams. Filtravimo sąlyga nurodo kuriems įrašams turėtų būti pakeista reikšmė. Tarkim norime pakeisti visų įrašų datą iš 2008 05 29 į 2008 05 30. Duomenų lentelė jau atidaryta, tereikia perjungti sritį, nes komandų veiksmus reikia atlikti aktyvioje srityje.

```
DBSELECTAREA(AliasLentele)   //pasirenkame sritį, kurioje bus atliekami veiksmai
DBEVAL({|| dLaukas := STOD(20080530)}, {|| dLaukas == STOD(20080529)})
//padaromi keitimai
```

Pateiktas programinio kodo fragmentas SQL užklausa atrodys taip:

```
RDDINFO(RDDI_EXECUTE, "UPDATE lentele SET dLaukas = '20080530' WHERE dLaukas = '20080529';")
```

Naudojant indeksuotus laukus pakeitimų kodo fragmentas skirsis nuo prieš tai pateikto. xBase kalbose negalima keisti to lauko reikšmės, kurio indeksas bus naudojamas. Komanda ORDSCOPE() naudojasi indeksais, kurių pagalba apriboją įrašų priėjimą. Pakeitus naudojamo indeksuoto lauko reikšmę indeksas yra perkuriamas ir po to bandant pereiti prie kito įrašo nebeaišku kurį kitą įrašą rodys rodyklė. Tokia pat logika galioja ir su trinamais įrašais. Pavyzdyje pateikiama lauko „cLaukas“ reikšmės keitimas nauja reikšme tik tiems įrašams, kurių data yra 2008 05 30.

```
DBSELECTAREA("AliasLentele")
ORDSETFOCUS("IndeksasA")
ORDSCOPE(BOTTOMSCOPE, STOD(20080530))
ORDSCOPE(TOPSCOPE, STOD(20080530))
DBGOTOP()
DBEVAL({|| cLaukas := "Pakeitėm"})
ORDSCOPE(BOTTOMSCOPE, NIL)
ORDSCOPE(TOPSCOPE, NIL)
```

Pateiktam kodo fragmentui SQL užklausos pavyzdys išlieka lygiais toks pat kaip ir pirmuoju atveju. Duomenų bazių valdymo sistemos visada stengiasi naudoti indeksus, jei tik jie yra sukurti. Todėl užklausos užrašymas nepasikeičia.

Dažnai xBase kalboje norima keisti indeksuoto lauko reikšmes, bet tuo pačiu ir pasinaudoti indeksuotu lauku apribojant priėjimą prie įrašų. Šiai situacijai išspręsti yra naudojamas papildomas kintamasis - masyvas, kur susikaupiami visų redaguojamų įrašu eilės numeriai, kurie yra unikalūs kiekvienam įrašui (kitaip vadinamas eilės numeris).

```
LOCAL aMasyvas, aI
DBSELECTAREA("AliasLentele")
ORDSETFOCUS("IndeksasA")
ORDSCOPE(BOTTOMSCOPE, STOD(20080529))
ORDSCOPE(TOPSCOPE, STOD(20080529))
aMasyvas := ARRAY(ORDKEYCOUNT())   //pasidarom reikalingą masyvo dydį, taip
veikia greičiau nei dinamiškai plėsti masyvą
DBGOTOP()
DBEVAL({|| AINS(aMasyvas, 1, RECNO())})   //į masyvą susidedame visų eilučių
eilės numerius
ORDSCOPE(BOTTOMSCOPE, NIL)
ORDSCOPE(TOPSCOPE, NIL)
FOR EACH aI IN aMasyvas   //pereinam visus masyvo elementus
  DBGOTO(aI)
  FIELD -> dLaukas := STOD(20080530)
NEXT
```

2.7.5. Įrašo trynimas

Įrašo ištrynimas xBase kalbose reiškia įrašą pažymėti kaip ištrintą. SQL kalboje įrašas yra išmetamas be galimybės prie jo prieiti. xBase įrašo išmetimo algoritmas gali būti skirstomas į tris grupes. Pirmas variantas – tai esamus įrašus komandos DBDELETE() pagalba pažymima kaip ištrintus, bet laukų reikšmės išlieka ir po to skaitant įrašus tikrinama ar jis yra nepažymėtas kaip ištrintas - .NOT. DELETED(), pvz. DBEVAL({|| atliekami veiksmai su neištrintais įrašais}, {|| .NOT. DELETED()}). Antras variantas – kuris dažniau naudojamas tai yra pažymėti įrašą ištrintu ir tuo pačiu visų laukų reikšmes paversti į NIL. Trečias variantas tai pažymėtus ištrintais įrašus pašalinti iš duomenų lentelės, tai atliekama pasinaudoti komanda PACK. Naudojant PACK duomenų lentelė būtinai turi būti atidaryta EXCLUSIVE rėžimu, tai rėžimas kai prie duomenų lentelės gali prieiti tik vienas vartotojas. Paminėti variantai xBase komandomis užrašomi sekančiai.

Žemiau pateikiamas kodo fragmentas yra panaudotas visuose toliau nagrinėjamuose įrašų ištrynimo pavyzdžiuose.

```
  LOCAL aI, aJ
  DBUSEAREA(.T., "DBFCDX", "split",, .F., .F.)
  ORDSETFOCUS("number")
  ORDSCOPE(BOTTOMSCOPE, 16102)
  ORDSCOPE(TOPSCOPE, 16102)
  aJ := ARRAY(ORDKEYCOUNT())
  DBGOTOP()
  DBEVAL({|| AINS(aJ, 1, RECNO())})
```

1) Įrašų ištrynimas, kai įrašai pažymimi kaip ištrinti, bet laukų reikšmės paliekamos.

```
  ...
  FOR EACH aI in aJ
    DBGOTO(aI)
    DBDELETE()
  NEXT
  DBEVAl({|| <Veiksmas su likusiais įrašais>}, {|| .NOT. DELETED() .AND. FIELD -
> number == 16102})
```

2) Įrašų trynimas, kai įrašai pažymimi kaip ištrinti ir laukų reikšmės pakeičiamo tuščiomis.

```
  ...
  FOR EACH aI in aJ
    DBGOTO(aI)
    null_rec() //pasinaudojam papildoma procedūra
  NEXT
  DBEVAl({|| <Veiksmas su likusiais įrašais>}, {|| FIELD -> number == 16102})
PROC null_rec()
  LOCAL nI, xI, cFile
  FOR nI := 1 TO FCOUNT()
    SWITCH DBFIELDINFO(DBS_TYPE, nI)
    CASE "C"; CASE "M"; xI := ""      ; EXIT
    CASE "D"          ; xI := STOD(""); EXIT
    CASE "L"          ; xI := .F.     ; EXIT
    CASE "N"          ; xI := 0       ; EXIT
    DEFAULT           ; xI := NIL
    END
    FIELDPUT(nI, xI)
  NEXT
  DBDELETE()
RETURN
```

3) Įrašų ištrynimas, kai įrašus panaikiname iš duomenų lentelės.

```
  ...
  FOR EACH aI in aJ
    DBGOTO(aI)
    DBDELETE()
  NEXT
  PACK
  DBEVAl({|| <Veiksmas su likusiais įrašais>}, {|| FIELD -> number == 16102})
```

Visų pateiktų xBase kodo fragmentų pavyzdžiai SQL užklausomis yra išverčiamas visada vienodai. Žemiau pateikiama kaip tai atrodo.

```
  RDDINFO(RDDI_CONNECT, {"MYSQL", "192.168.1.4", "root", "Pedalai", "test"},
"SQLBASE", @hConnection)
  RDDINFO(RDDI_EXECUTE, "SET CHARACTER SET cp1257", "SQLBASE", hConnection)
  RDDINFO(RDDI_EXECUTE, "DELETE FROM split WHERE number=16102;", "SQLBASE",
hConnection)
  RDDINFO(RDDI_DISCONNECT, hConnection, "SQLBASE")
```

Išnagrinėjus charakteringuosius xBase komandų atvejus pastebėta, kad dauguma rezultatų išgaunama pasinaudojant tomis pačiomis komandomis. Nedidelis kiekis skirtingų komandų užrašytų kitokia tvarka išgauna skirtingą rezultatą, bet ta pati komandą visada grąžiną tą patį

rezultatą. Todėl yra įmanoma sukurti taisykles, kurių pagalba galima pateiktas xBase komandas pakeisti SQL užklausomis. Skirtinga komandų tvarka grąžina skirtingą rezultatą, todėl užklausa turėtų būti formuojama atsižvelgiant į visas komandas, o ne pavienes. Viena komanda niekada neatstos visos užklausos, todėl užklausos bus lipdomos iš atskirų gabalėlių, kuriuos surinksime paeiliui transformuojant xBase komandas.

2.8. Analizės išvados

1. Atlikta SWOT analizė parodė daromo darbo silpnąsias ir stipriąsias vietas. Suskaičiuota bendra projekto galimybių reikšmė yra didesnė nei bendra projekto grėsmių reikšmė, todėl programinį įrankį verta kurti.
2. Atlikta analizė parodė, kad pasinaudojant xHarbour SQLRDD bibliotekos kodo transformavimo realiu laiku daromais sprendimais, galima kurti metodą kuris leistų transformuoti programinį kodą prieš sutransliuojant programinę įrangą.
3. Analizės metu nustatyta, kad transformuojant programinės įrangos kodą, nereikia keisti duomenų bazės struktūros, nes pasikeičia tik naudojama duomenų bazės valdymo sistema.

3. Transformavimo metodo kūrimas

3.1. xBase komandų transformavimo į SQL komandas taisyklės

xBase komandas galima užrašyti pasinaudojant aibių teorija. Aibių teorijos pagalba lengviau suprasti xBase komandomis atliekamus veiksmus ir juos konvertuoti į SQL užklausas. Skirtingos xBase komandos atliekančios tokius pat veiksmu aibių teorijos pagalba bus užrašomos vienodai, todėl nekils problemų keičiant į SQL užklausas.

Tarkime duomenų bazę žymėsime *DB*. Duomenų bazė sudaryta iš baigtinio skaičiaus lentelių T_l, $l=\overline{1,t}$ čia $t-$lentelių skaičius duomenų bazėje. Tada duomenų bazės aibė užrašoma $DB=\bigcup_{l=1}^{t} T_l$. Lenteles sudaro įrašų aibės $T_l=\{x^l_{\overline{1,r}} : x^l_n \neq x^l_m\}$, čia $n,m=\overline{1,r}$, $n\neq m$, $r-$eilučių skaičius duomenų lentelėje. Vieną įrašą sudaro keletas skirtingų laukų, kurie apibrėžiami kaip vektorius $x^l_{\overline{1,r}}=\left(x^l_{\overline{1,r},1}, x^l_{\overline{1,r},2}, x^l_{\overline{1,r},3},..., x^l_{\overline{1,r},c}\right)$, čia $c-$lentelės l stulpelių skaičius. Visų lentelių duomenys yra būdingi tik tai lentelei, todėl aibės yra nesikertančios $T_i \cap T_j=\varnothing$, čia $i,j=\overline{1,t}$, $i\neq j$. Išskirsim keletą žymėjimų: lentelė *D*, lentelė *I* ir lentelė *U*. Aibė *DBA* – atidarytų lentelių aibė. Tai tos pačios lentelės T_l, tik atlikus tam tikrą veiksmą. Lentelė *D* tai kai iš lentelės T_l pašalinami įrašai, $D=T_l \setminus \{x^l_{\overline{1,d}} : x^l_n \neq x^l_m\}$, čia $n,m=\overline{1,d}$, $n\neq m$, l – konkreti lentelė, $d-$pašalintų įrašų skaičius. Lentelė I tai lentelė T_l po įrašų įterpimo, $I=T_l \cup \{x^l_{\overline{1,i}} : x^l_n \neq x^l_m, x^l_{\overline{1,i}} \notin T_l\}$, čia $n,m=\overline{1,i}$, $n\neq m$, i – įterpiamų įrašų skaičius. Lentelė U tai lentelė T_l po įrašų pakeitimo, $U=(T_l \setminus D)\cup I$.

Pasinaudojant aukščiau apibrėžtomis pradinėmis sąlygomis užrašomos xBase komandos panaudojant aibių teorija. xBase komandos ir jų atvaizdavimas aibių teorijos elementais pateikiamas 3.1 lentelėje. Šis aprašas padės programuojant realizuoti xBase komandų transformavimo į SQL konstrukcijas.

3.1 lentelė **xBase komandos ir atitikmenys pasinaudojant aibėmis**

xBase komanda	**Atitikmuo pasinaudojant aibėmis**
DbAppend()	$I=T_l \cup (...)$ - nurodoma, kad prie aibės bus pridedamas naujas elementas – naujo įrašo įterpimas.
DbCloseArea()	$DBA = DBA \setminus T_l$ - nurodoma, kad iš atidarytų lentelių aibės reikia atimti

	tą lentelę, kurią uždarom.
DbCommit()	Visų padarytų pakeitimų su duomenų lentelėmis ir indeksais įrašymas iš laikinosios atminties į pastoviąją atmintį. Šis veiksmas nėra aprašomas aibėmis, nes SQL užklausose nėra naudojamas joks alternatyvus veiksmas.
DbCreate(pr_1, pr_2, pr_3, pr_4, pr_5)	$DB = DB \cup T_{pr_1}$, $T_{pr_1} = \{\varnothing : (pr_2)\}$, $DBA = \begin{cases} DBA \cup T_{pr_1}, jei \quad pr_4 = .T. \\ DBA, jei \quad pr_4 = .F. \end{cases}$. Naujos lentelės sukūrimas ir įtraukimas į reikiamas aibes.
DbDelete()	$D = T_l \setminus \{x_d^l\}$ - iš aibės išmetamas vienas elementas – įrašo ištrynimas.
DbEval({|| pr_1})	$x_{\overline{1,r}}^l$ - pereiti visus aibės elementus ir įvykdyti pr_1 komandą.
DbSeek(pr_1) taip pat ir komanda OrdScope(pr_2, pr_1)	$(..., x_{\overline{1,r,c}}^l = pr_1, ...)$ - šis vektorius bus naudojamas atrenkant įrašus pagal lauką c kuriems atlikti veiksmus.
DbSkip()	Parodo kad, veiksmai bus atliekami ne su vienu aibės elementu, o su keliais.
DbUseArea(pr_1, pr_2, pr_3, pr_4, pr_5, pr_6, pr_7, pr_8)	$DBA = DBA \cup T_{pr_3}$ - atidaromas duomenų lentelė, įtraukiam lentelę į atidarytų lentelių aibę.
LastRec()	$\|T_l\|$ - grąžiną aibės elementų skaičių.
RecNo()	x_r^l – galim apibrėžti kaip grąžinamas einamasis įrašas – aibės elementas, nes SQL neturi tokio dalyko kaip eilės numerio, tai jei reiks identifikuoti, bus galima pagal visą įrašą.
FIELD *laukas1*, *laukas2, laukas3, ... , laukasC*	Komanda FIELD nurodoma, kad kintamieji yra laukai turintys pavadinimus: *laukas1, laukas2. laukas3, ..., laukasC*. Visose taisyklėse bus naudojami ir tai yra aibės elemento $x_{\overline{1,r,c}}^l$ - antrasis indeksas *c*.

Toliau nagrinėjama kaip aibės atsivaizduojamos SQL užklausomis. Jei buvo atliktas naujo elemento įterpimas $I = T_l \cup \{x_{\overline{1,i}}^l : x_n^l \neq x_m^l, x_{\overline{1,i}}^l \notin T_l\}$, tai SQL užklausa užsirašo INSERT INTO `$T_l$` $\left(1^l, 2^l, 3^l ... c^l\right)$ VALUES $\left(x_{1,1}^l, x_{1,2}^l, x_{1,3}^l, ..., x_{1,c}^l\right)$, ..., $\left(x_{i,1}^l, x_{i,2}^l, x_{i,3}^l, ..., x_{i,c}^l\right)$, $l = \overline{1,t}$, čia T_l – duomenų bazės lentelė, t – lentelių skaičius duomenų bazėje, i – įterpiamų įrašų skaičius, c – lentelės l stulpelių skaičius. Esamo įrašo atnaujinimo atveju $U = \left(T_l \setminus D\right) \cup I$ užsirašo, UPDATE `$T_l$` SET $1^l = x_{\overline{1,i,1}}^I, 2^l = x_{\overline{1,i,2}}^I, ..., c^l = x_{\overline{1,i,c}}^I$ WHERE $1^l = x_{\overline{1,i,1}}^D$ *AND* $2^l = x_{\overline{1,i,2}}^D$ *AND...AND* $c^l = x_{\overline{1,i,c}}^D$, čia

$x^{D}_{1,i,c}$ - lentelės T_l reikšmė, kuri turi būti pakeista, $x^{I}_{1,i,c}$ - lentelės T_l reikšmė, į kurią bus keičiama, i - keičiamų įrašų skaičius. Įrašo išmetimo atveju $D = T_l \setminus \{x^{l}_{1,d} : x^{l}_{n} \neq x^{l}_{m}\}$, SQL užklausa užrašoma DELETE FROM `$T_l$` WHERE $1^l = x^{l}_{1,d,1}$ *AND* $2^l = x^{l}_{1,d,2}$ *AND...AND* $c^l = x^{l}_{1,d,c}$, čia d – trinamų įrašų skaičius.

Panaudojus 3.1 lentelėje pateiktus xBase komandų aprašus panaudojant aibių teorijos elementus ir bendruosius SQL komandų formavimo principus išreikštus aibių elementais sudarytos xbase komandų transformavimo į SQL konstrukcijas taisyklės, kurios pateikiamos 3.2 lentelėje. Pateiktos taisyklės naudojamos kitame skyrelyje pateiktame transformavimo proceso algoritme.

3.2 lentelė **Transformavimo taisyklės**

Eil. Nr.	**xBase komanda**	**SQL užklausa**	**Žodinis paaiškinimas**
1.	DbAppend()	INSERT INTO `Einamoji()`	DbAppend() komanda, kuri neturi parametrų. SQL užklausa susidarys iš INSERT INTO, kurios kūnas bus sudarytas iš žemiau xBase kode nurodytų laukų. Funkcija Einamoji() grąžina duomenų lentelės pavadinimą, kuri yra einamoji sritis iškviečiant xBase komandą.
2.	DbCloseArea()	Pabaiga(.T.)	DbCloseArea() komanda, kuri neturi parametrų, parodo kad SQL užklausa, kuri buvo formuojama iki šitos komandos einamajai sričiai, turi būti užbaigta ir įrašyta į transformuoto kodo rinkmeną. Toliau SQL užklausos su einamąją sritimi negalimi, nes ši komanda uždaro lentelę. Funkcija Pabaiga(), nurodo, kad užklausos formavimas turi būti baigtas, jei parametras .T. tada yra uždaroma lentelė, jei .F. toliau veiksmai su lentele yra galimi.
3.	DbCommit()	Pabaiga(.F.)	DbCommit() komanda, kuri neturi parametrų, atitinką tuos pačius veiksmus kaip ir DbCloseArea() atveju, tik lentelė yra neuždaroma, ir toliau SQL užklausos gali būti vykdomos su einamosios srities lentele.
4.	DbCreate(pr_1, pr_2, pr_3, pr_4, pr_5)	CREATE TABLE `$pr_1$`	DbCreate() komanda turi penkis parametrus. Parametrai žymimi pr_N, čia N – parametro eilės

		(Struktūra(pr_2))	numeris. Komandos pirmasis parametras pereina į SQL užklausos lentelės pavadinimą, o antrasis parametras yra perduodamas funkcijai Struktūra(), kuri iš xBase struktūros sugeneruoja SQL užklausos struktūra, naujai lentelei sukurti.
5.	DbDelete()	DELETE FROM `Einamoji()` WHERE	DbDelete() komanda neturi parametrų, kuri yra keičiama į SQL užklausą DELETE, pasinaudojant funkcija Einamoji(), o prie sąlygos WHERE prirašoma konstrukcija, kuri buvo sukurta prieš tai xBase komandos eilutėse.
6.	DbSeek(pr_1, pr_2, pr_3)	WHERE Sąlyga(pr_1)	DbSeek() komanda turi tris parametrus, kuri keičiama į sąlygą WHERE, kurią suformuoja funkcija Sąlyga() pasinaudojant pirmuoju xBase komandos parametru. Po šitos komandos būtinai turės eiti komandos iš kurių bus toliau formuojama SQL užklausa iš kurios nusprendžiamas užklausos tipas (SELECT, UPDATE, DELETE).
7.	OrdScope(pr_1, pr_2)	WHERE Sąlyga(pr_1, pr_2)	OrdScope() komanda turi du parametrus, kuriuos panaudojame suformuoti WHERE sąlygai, o tai atlieka funkcija Sąlyga(). Po šitos komandos būtinai turės eiti komandos iš kurių bus toliau formuojama SQL užklausa iš kurios nusprendžiamas užklausos tipas (SELECT, UPDATE, DELETE).
8.	LastRec()	SELECT COUNT(*) FROM `Einamoji()`	LastRec() komanda, neturi parametru ir yra keičiama į SELECT sakinį, kuris suskaičiuoja lentelės eilučių skaičių. Lentelė yra nurodoma pasinaudojant funkcija Einamoji(), kuri grąžina einamosios srities pavadinimą, kuris ir yra lentelės pavadinimas.
9.	OrdKeyCount()	SELECT COUNT(*) FROM `Einamoji()` WHERE	OrdKeyCount() komanda, neturi parametru ir yra keičiama į SELECT sakinį, kuris grąžina eilučių skaičių atitinkantį filtrą, kuris buvo nurodytas su komandomis OrdScope().
10.	FIELD ->	`FieldName` =	Pateikta xBase struktūra yra keičiama į pateiktą

	FieldName := Value	`Value`	SQL struktūrą, kuri naudojama INSERT arba UPDATE sakiniuose. Kuris sakinys priklauso nuo prieš tai buvusių komandų, kurios apsprendžia sakinio tipą.
11.	Variable := FieldName	SELECT `FieldName` FROM `Einamoji()` WHERE	Pateikta xBase struktūra yra keičiama į SELECT užklausą, kurioje nurodome lauką ir lentelės pavadinimą, kurį grąžina funkcija Einamoji(). Filtravimo sakinys yra užpildomas prieš tai transformuotomis komandomis.

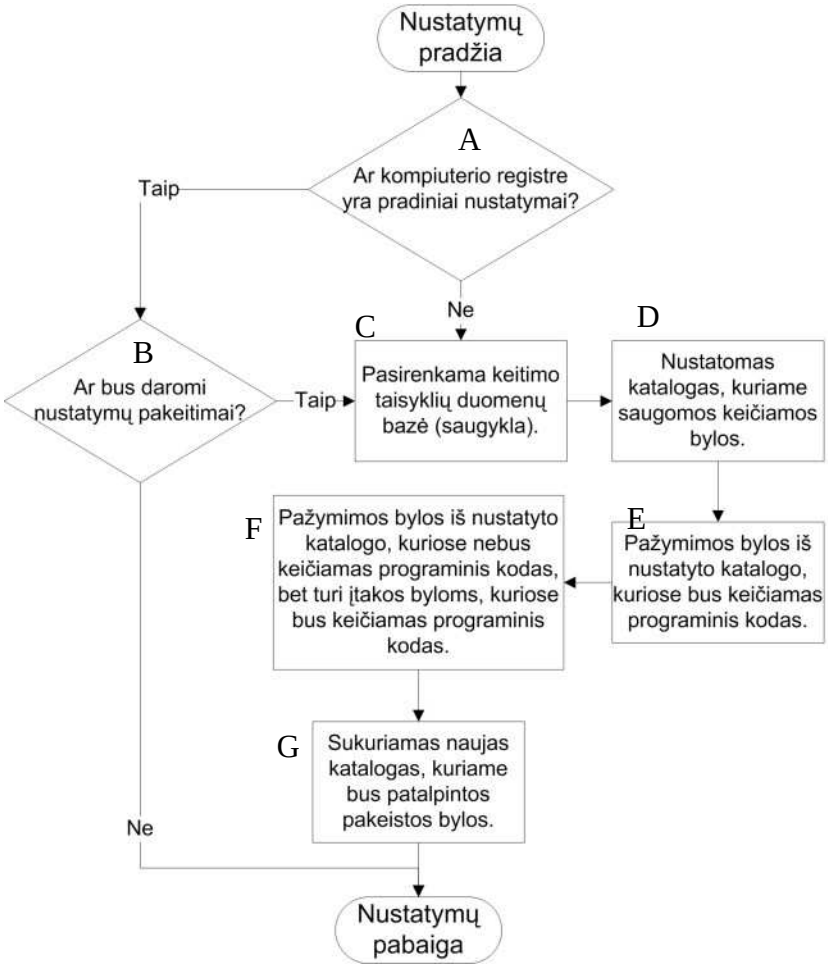
Pasinaudojus 3.2 lentelė pateikiamas pavyzdys 3.3 lentelėje, kurioje matosi xBase programinio kodo eilutės, kurioms buvo pritaikyta nurodyta taisyklė ir SQL kodas pritaikius taisyklę.

3.3 lentelė **Kodo transformavimo pavyzdys**

Programinio kodo fragmentas	Taisyklės Nr.	SQL
LOCAL aI, aJ		
DBUSEAREA(.T., "DBFCDX", "split",, .F., .F.)		
ORDSETFOCUS("number")		
ORDSCOPE(BOTTOMSCOPE, 16102)	7.	WHERE `number` <= `16102`
ORDSCOPE(TOPSCOPE, 16102)	7.	WHERE `number` >= `16102`
aJ := ARRAY(ORDKEYCOUNT())	9.	SELECT COUNT(*) FROM `split` WHERE
DBGOTOP()		
DBEVAL({||AINS(aJ,1,RECNO())})		
FOR EACH aI in aJ		
DBGOTO(aI)		
DBDELETE()	5.	DELETE FROM `split` WHERE
NEXT		
ORDSCOPE(BOTTOMSCOPE)	7.	WHERE
ORDSCOPE(TOPSCOPE)	7.	WHERE
DBCLOSEAREA()	2.	Bus suformuota galutinė užklausa: DELETE FROM `split` WHERE `number` <= `16102` AND `number` >= `16102`;

3.2. Xbase komandų transformavimo modulio algoritmai

Prieš pradedant darbą programoje būtina nustatyti keletą pirminių nustatymų, kurie bus naudojami tolimesniam programos veikimui užtikrinti, o svarbiausi ir programinio kodo transformavimo metu. Pasinaudojus „Įvesti pradinius nustatymus“ panaudos atveju sukurtas detalus programos veikimo algoritmas, kuris pateikiamas 3.1 paveiksle.



3.1 pav. **Programinės įrangos nustatymų algoritmas**

Pateikto algoritmo detalus žodinis aprašymas padeda suprasti kiekvieno žingsnio svarbą algoritme:

A. žingsnis - pradedant vykdyti nustatymų algoritmą būtina patikrinti ar kompiuterio registre yra saugomi prieš tai buvę nustatymai. Kad vartotojui pasileidus programą kiekvieną kartą nereiktų suvedinėti nustatymų programinė įranga kompiuterio registruose saugo paskutinių nustatymų duomenis. Jei nustatymai yra saugomi, tada pereiname prie B punkto. Jei sistema pradėdama darbą neranda išsaugotų nustatymų kompiuterio registre, tada paprašo vartotojo suvesti pradinius nustatymus, kurie bus naudojami tolimesniame programinės įrangos darbe ir programa pereina prie C punkto.

B. – „ar bus daromi nustatymų pakeitimai?“. Atidarius programą ne pirmą sykį vartotojas nori iškarto dirbti ir nebenori nustatinėti tuos pačius nustatymus, kuriuos padarė prieš tai. Taip pat vartotojas tuo metu turi galimybę pasirinkti vieną iš penkių paskutinių išsaugotų nustatymų. Vartotojui atsakius, kad nustatymų nekeis arba pasirinkus viena iš penkių išsaugotų nustatymų programinė įranga nusiskaito kompiuterio registruose saugomus duomenis ir užbaigia darbą.

C. Programinė įranga paprašo vartotojo nurodyti, kur saugoma transformavimo taisyklių duomenų bazę. Vartotojas neturi kitokio pasirinkimo kaip tik išsirinkti ko prašo sistema arba visiškai nutraukti darbą, išeinant iš programos.

D. žingsnyje nustatomas katalogas, kuriame yra saugomos bylos kurios bus transformuojamos, programa taip pat sugeba surasti ir visas bylas pakatalogiuose. Programa ieško bylų, kurių praplėtimas yra PRG, tai praplėtimas, kurį naudoja Harbour kompiliatorius. Vieno projekto bylos turėtų būti saugomos viename kataloge, kur jos gali būti suskirstytos į pakatalogius.

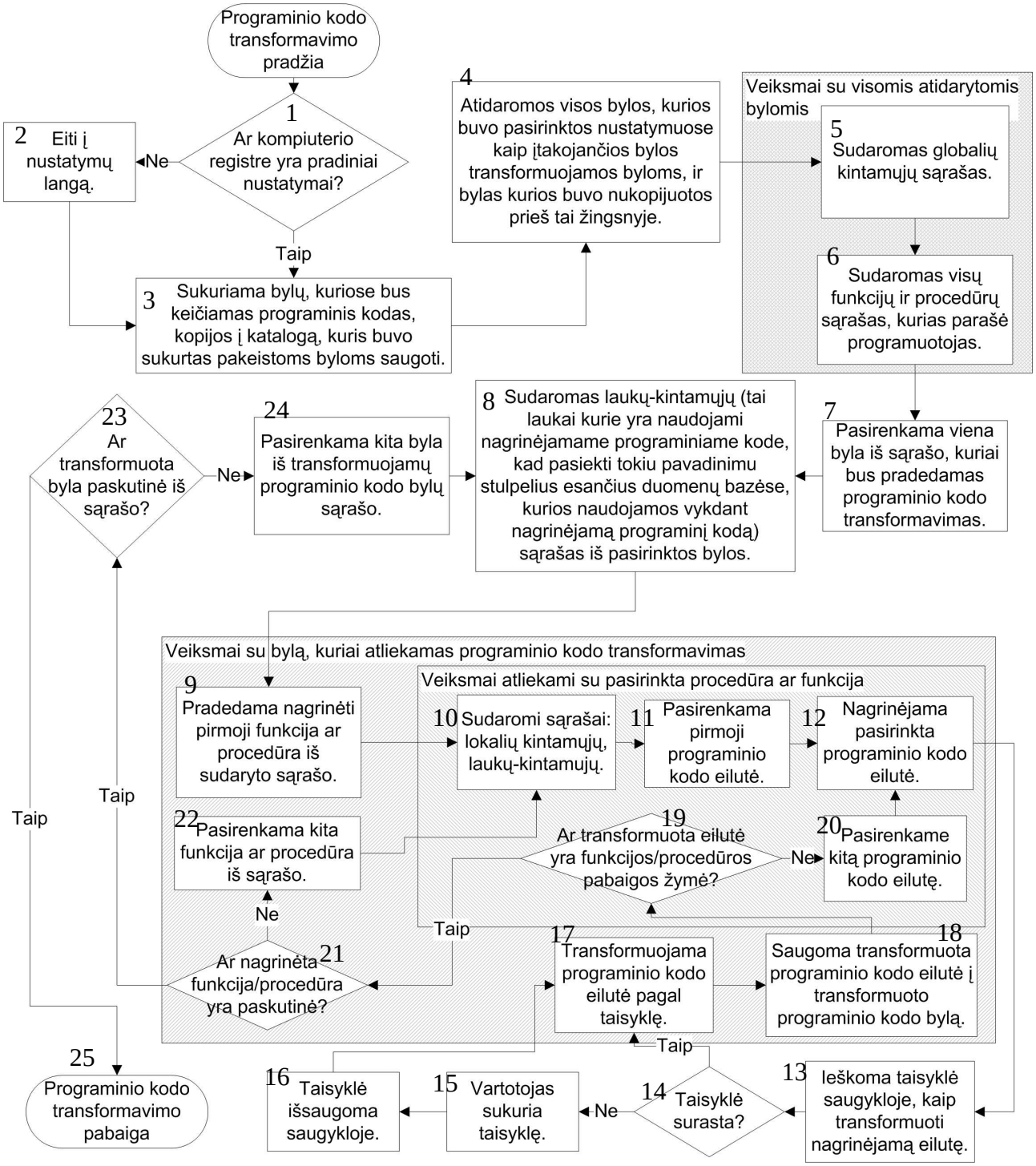
E. Atlikus D veiksmą programa peržiūri katalogus ir pateikia rastų bylų sąrašą. Vartotojas iš pateikto sąrašo išsirenka bylas, kurios bus transformuotos. Iš pateikto sąrašo būtinai turi būti pasirinkta bent viena byla. Jei vartotojas suklydo nurodydamas katalogą, gali grįžti vį D žingsnį ir nurodyti kitą katalogą.

F. Po to pateikiamas antras sąrašas, kuriame nebėra prieš tai pasirinktų bylų. Pateiktame sąraše vartotojas pažymi tas bylas, kurios turi įtakos transformuojant programinį kodą. Šiuo atveju gali būti pažymėta nei viena byla. Tarp pažymėtų bylų pavyzdžiui gali būti byla kurioje surašytos konstantos.

G. Sistema pasiūlo vartotojui katalogą, kuriame bus išsaugotos bylos su transformuotu programiniu kodu. Vartotojas taip pat turi galimybę nurodyti norimą katalogą. Sistema sukuria nurodytą katalogą ir baigia nustatymus.

Vartotojui suvedus visus reikalingus nustatymus programa pasiūlo pradėti programinio kodo transformavimą. Detalus „Transformuoti kodą iš xBase į SQL užklausas“ panaudojimo atvejo algoritmas pateikiamas 3.2 paveiksle. Algoritmo pagalba detaliai aprašoma kokius veiksmus sistema atlieka transformuodama programinį kodą.

1. Algoritmo pirmajame žingsnyje yra patikrinama ar kompiuterio registre yra išsaugoti nustatymai reikalingi programos veikimui. Jei programa neranda nustatymų tada pereina prie antrojo žingsnio. Jei kompiuterio registre programa randa tinkamus nustatymus, tada pereina prie 3 žingsnio.
2. Programa iškviečia „Įvesti pradinius nustatymus“ panaudojimo atvejį, kur vartotojas yra priverstas suvesti reikalingus nustatymus.



3.2 pav. **Programinio kodo transformavimo algoritmas [autoriaus]**

3. Programos pasirengiamieji darbai programinio kodo transformavimui atlikti. Programa padaro transformuojamų bylų kopijas į katalogą, kur bus saugomos bylos su transformuotu programiniu kodu.
4. Ketvirtu etapu programa atsidaro visas bylas , kurios buvo pažymėtos nustatymų algoritmo F žingsnyje ir tas kurias sistema sukūrė trečiame etape. Toliau penktas ir šeštas veiksmai yra atliekami su visomis atidarytomis bylomis.

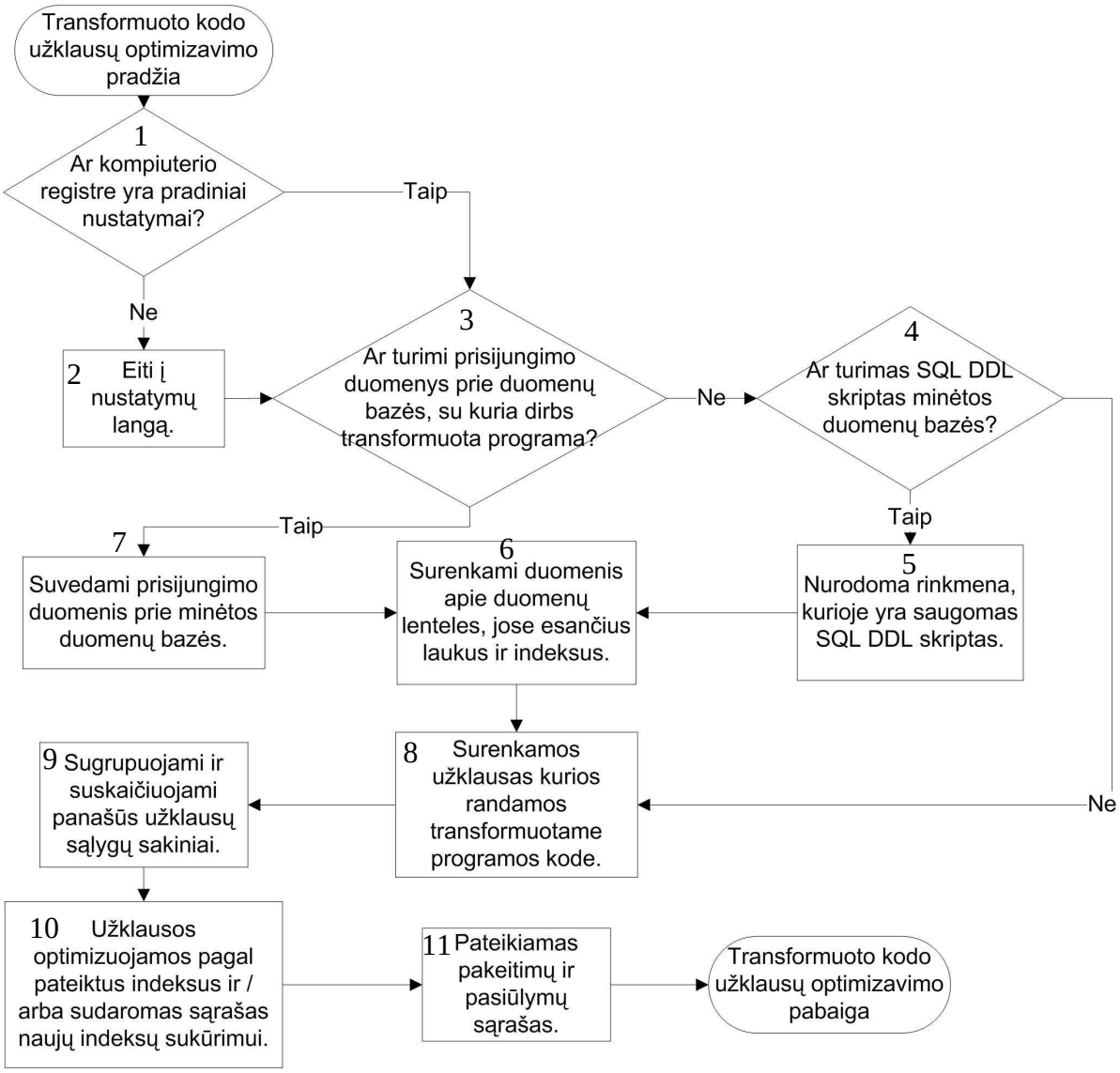
5. Sistema susidaro globalių kintamųjų sąrašą, tam kad galėtų atskirti kur kintamieji kur programavimo kalbos funkcijos ir procedūros.
6. Sistema susidaro funkcijų ir procedūrų sąrašą, kurios buvo parašytos programuotojo transformuojamame programiniame kode. Sudarytas sąrašas toliau bus naudojamas transformuojant programinį kodą.
7. Septintuoju žingsniu pasirenkame vieną bylą iš nustatymų algoritmo E etapo sąrašo. Tai bus pirmoji byla, kurioje bus atliktas programinio kodo transformavimas.
8. Toliau yra būtina surinkti laukų-kintamųjų sąrašą iš pasirinktos bylos. Laukai-kintamieji tai programinio kodo kintamieji kurie nurodo kokie yra laukų pavadinimai duomenų bazėse. Jų pagalba galima kreiptis į stulpelį įrašant ar skaitant informaciją. Tolimesni veiksmai yra atliekami tik su byla kuriai atliekamas programinio kodo transformavimas.
9. Devintas žingsnis – atsirenkame funkcijas ir procedūras, kurios yra aprašytos naudojamojoje byloje iš sąrašo sudaryto šeštajame žingsnyje ir pradedama nagrinėti pirmoji funkcija/procedūra iš sąrašo.
10. Pasirinktos funkcijos/procedūros nagrinėjimo pradžioje yra sudaromas lokalių kintamųjų ir laukų-kintamųjų sąrašas.
11. Pasirenkama pirmoji programinio kodo eilutė, nuo kurios prasideda kodo transformavimo darbai.
12. Pasirinktoji eilutė yra skaidoma į funkcijas, jų parametrus, atliekamus veiksmus tarp kintamųjų, laukų-kintamųjų naudojimas, priskyrimas. Eilutės struktūra yra toliau perduodama paieškai.
13. Eilutės struktūra bandoma atpažinti peržiūrint taisyklių rinkinį pasirinktą nustatymų algoritmo C etape.
14. Jei taisyklė kaip transformuoti programinio kodo eilutę yra nesurandama, tada vartotojui pateikiama galimybė sukurti taisyklę dar nenumatytam atvejui, programa pereina prie 15 žingsnio. Jei taisyklė buvo surasta programa pereina prie 17 žingsnio.
15. Vartotojas sukuria transformavimo taisyklę pateiktai programinio kodo eilutei. Taisyklių kūrimas aptartas 3.1 skyriuje.
16. Sukūrus taisyklę sistema išsaugo esamoje taisyklių saugykloje.
17. Sukūrus naujai ar atradus jau esamą taisyklę programinio kodo eilutė yra pakeičiama pritaikant taisyklę.
18. Transformuota eilutė yra saugoma transformuotoje byloje, kuri buvo sukurta 3 - čiame programos algoritmo žingsnyje.

19. Po eilutės transformavimo yra tikrinama ar transformuota eilutė yra funkcijos/procedūros pabaigos žymė, jei ne tada programa pereina prie 20 žingsnio. Jei tai buvo pabaigos žymė programa pereina prie 21 žingsnio.
20. Pasirenkama kitą iš eilės einanti programinio kodo eilutė ir pradedamas kartoti algoritmas nuo 12 - to žingsnio.
21. žingsnyje tikrinama ar tai buvo paskutinė funkcija/procedūra transformuojamoje byloje. Jei tai buvo paskutinė transformuota funkcija/procedūra transformuojamos bylos sąraše, tai sistema pereina prie 23 – čio žingsnio. Jei programa atranda, kad tai nėra paskutinė funkcija/procedūra pereina prie 22 – o žingsnio.
22. Sąrašui nepasibaigus sistema parenką kitą iš eilės einančią funkciją/procedūrą ir pradeda kartoti algoritmą nuo 10 - tojo žingsnio.
23. tikriną transformuojamų bylų sąrašą. Jei sąraše dar yra bylų, kurias reikia transformuoti programa pereina prie 24 – tojo žingsnio. Jei transformuota byla buvo paskutinė iš sąrašo tai sistema užbaigia savo darbą.
24. Programa pasirenka kitą bylą ir pradeda kartoti algoritmą nuo 8 – tojo žingsnio.

Atlikus programinio kodo transformavimo darbus galima pratęsti programinio kodo keitimo darbus transformuoto kodo optimizavimu. Programinio kodo optimizavimas atliekamas transformavus programinį kodą. Užklausų optimizavimas yra naudingas tuo atveju, jei programuotojas nenori gaišti laiko ir tikrinti transformuotos programos veikimo spartos. Detalus panaudojimo atvejo „Optimizuoti užklausas“ programinis algoritmas pateikiamas 3.3 paveiksle, kurio pagalba matoma kaip programinė įranga optimizuoja užklausas. Užklausų detalūs optimizavimo metodai aptarti 2.5 skyriuje.

Programinės įrangos užklausų optimizavimo algoritmo detalus aprašymas:

1. Programa kompiuterio registre ieško ar yra išsaugoti nustatymai, jei taip tai pereina prie 3 – čio žingsnio, jei ne tai pereina prie 2 – ojo žingsnio.
2. Programa nerado reikalingu nustatymų, todėl vykdo „Įvesti pradinius nustatymus“ panaudos atvejį.
3. Programa patikrina ar turimi duomenys prisijungimui prie SQL duomenų bazės. Jei ne tai pereiname prie 4 – tojo žingsnio, jei taip pereiname prie 7 – tojo užklausų optimizavimo algoritmo žingsnio.
4. Programa teiraujasi vartotojo ar turima SQL DDL scenarijaus byla, pagal kurią būtų galima surinkti informaciją apie duomenų bazės lenteles ir indeksus. Tai byla, kurią galime gauti eksportuojant duomenų bazę į tekstinę rinkmeną. Jei vartotojas nurodo, programa pereina prie 5 – tojo žingsnio, jei ne tai pereinama prie 8 – tojo žingsnio.

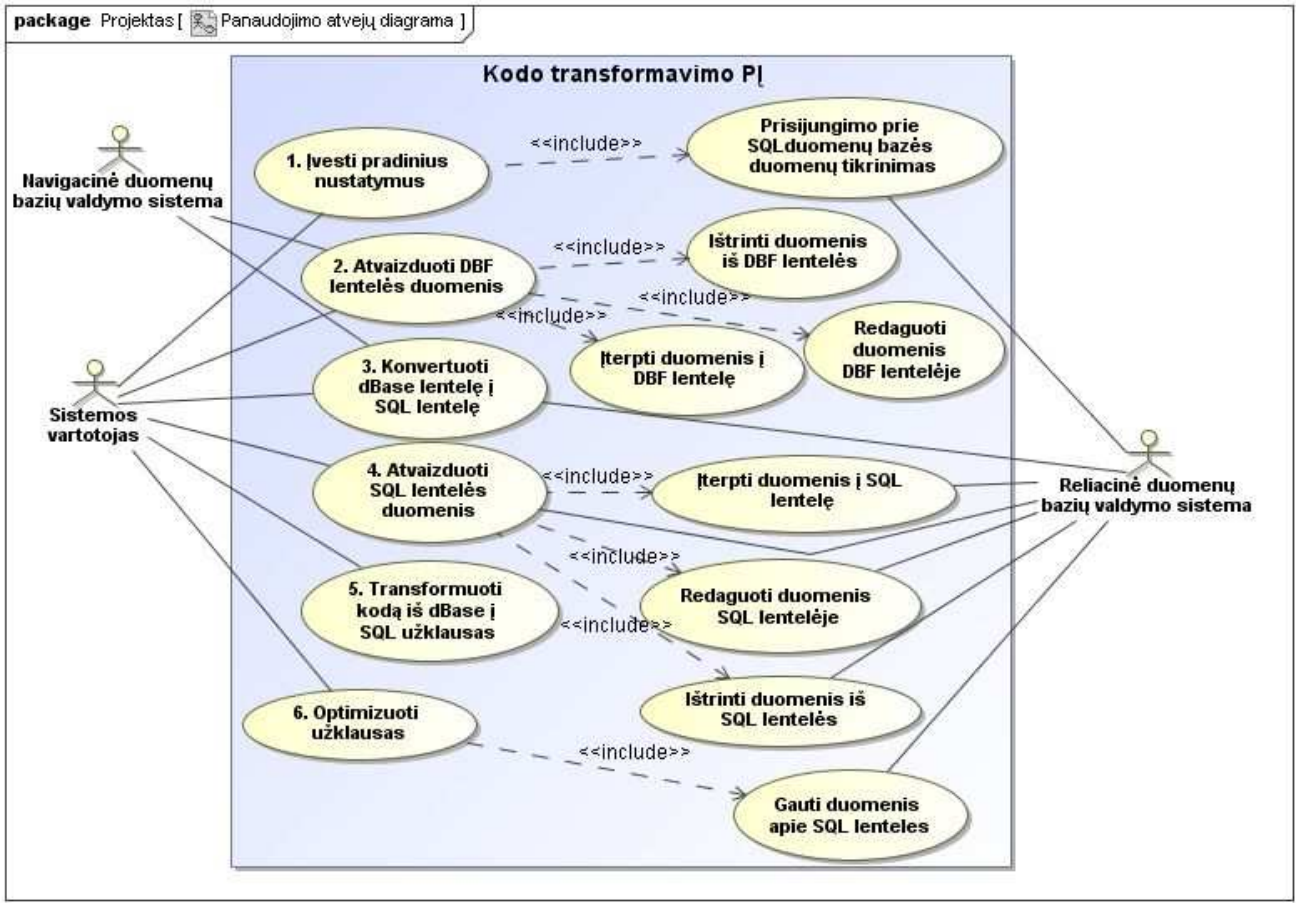


3.3 pav. **Užklausų optimizavimo algoritmas**

5. Vartotojas nurodo kelia iki SQL DDL rinkmenos.
6. Programinė įranga sukaupia duomenis apie duomenų lentelėse esančius laukus ir indeksus. Pasinaudojusi prisijungimo duomenimis prie duomenų bazės arba nurodyta DDL byla.
7. Vartotojas suveda prisijungimo duomenis prie duomenų bazės ir programa prisijungia prie nurodytos duomenų bazės.
8. Programa surenka visas užklausas, kurios buvo sukurtos transformuojant programinį kodą.
9. Programa sugrupuoja ir suskaičiuoja kaip dažnai yra naudojami vienodi sąlygų sakiniai.
10. Surinktos užklausos yra optimizuojamos pasinaudojant 2.5 skyriuje aptartais metodais. Jei programai nepavyko prieš tai gauti jokių duomenų apie realią duomenų bazę, tada yra sudaromas sąrašas indeksų sukūrimui.
11. Programa pateikia, padarytus pakeitimus ir pasiūlymus, kaip reiktų pakeisti duomenų bazę, kad nagrinėjamo programinio kodo užklausos būtų vykdomos sparčiau.

4. XBase komandų transformavimo modulis (programinė įranga)

Programa turi sugebėti analizuoti programinį kodą. Duomenų bazėje saugomos struktūros, kurias reikia surasti ir pakeisti pateiktomis kitomis struktūromis. Programa automatizuotų būdu keičia programinį kodą, tik jei atsiranda dviprasmybės ar neaiškumai, programa pasiūlo vartotojui išspręsti iškilusią problemą. Programa po programinio kodo transformavimo turi peržiūrėti sukurtas užklausas ir jas optimizuoti, arba pateikti vartotojui tai padaryti rankiniu būdu. 4.1 paveiksle pateikiama programinės įrangos panaudojimo atvejų diagrama, kurios pagalba toliau bus projektuojama programinė įranga.



4.1 pav. **Programinės įrangos panaudojimo atvejai**

4.1. XBase komandų transformavimo modulio projektinė specifikacija

Projektuojamos programinės įrangos pagrindinės funkcijos: transformuoti programinį kodą iš xBase komandų į SQL užklausas, optimizuoti užklausas. Programa taip pat gali atlikti šias funkcijas: atvaizduoti SQL lenteles, atvaizduojamose lentelėse pakeisti, ištrinti ir įterpti naujus

įrašus; atvaizduoti DBF lentelių duomenis, įterpti, redaguoti, ištrinti DBF lentelių duomenis; konvertuoti xBase lenteles į SQL lenteles.

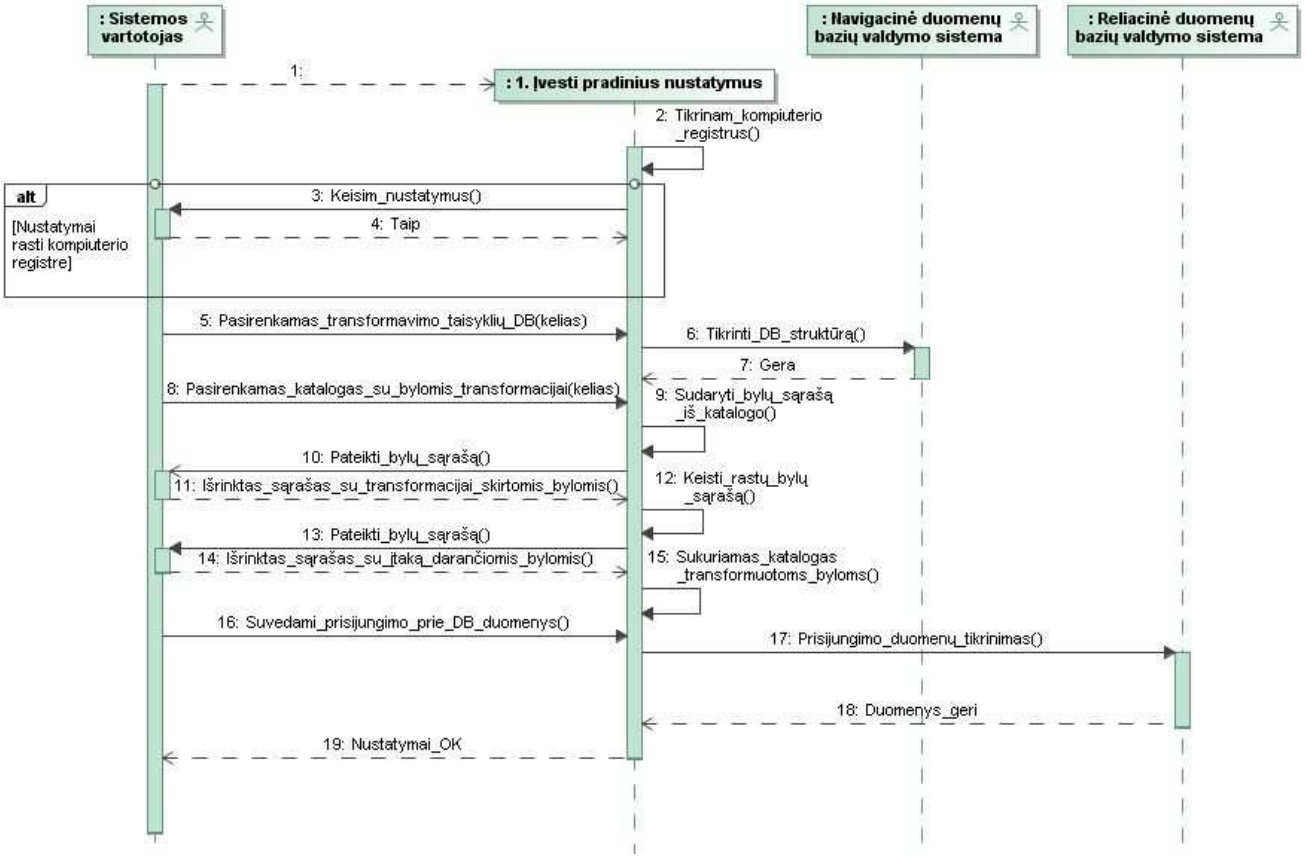
Pirmas žingsnis pradedant dirbti su programa tai yra įvesti nustatymus. Nustatymai naudojamai visuose likusiuose veiksmuose, todėl svarbu teisingai užpildyti pateiktas lenteles, kad vėliau nereikėtų grįžti. Projektuojamos programinės įrangos detali pradinių nustatymų įvedimo specifikacija pateikiama 4.1 lentelėje, kurios pagalba detalizuojamas „įvesti pradinius nustatymus“ panaudos atvejis.

4.1 lentelė. Programinės įrangos pradinių nustatymų panaudos atvejų specifikacija

Panaudos atvejis 1. „Įvesti pradinius nustatymus“	
Prieš sąlyga	Nėra
Sužadinimo sąlyga	1. Vartotojas nori įvesti nustatymus
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
Vartotojas atliko sužadinimo sąlygas	2. Tikrina vietinio kompiuterio registrus. Jei neranda išsaugotų nustatymų sistema pereina prie 5 punkto.
	3. Sistema klausia vartotojo ar keisim rastus nustatymus?
4. Vartotojo atsakymas [taip/ne]	Jei atsakymas buvo taip tai pereinam prie 5 punkto, jei ne tai pereinam prie 19 punkto.
5. Vartotojas pasirenka transformavimo taisyklių duomenų bazę	6. Sistema užklausia duomenų lentelės struktūros navigacinės duomenų bazės valdymo sistemos. Jei struktūra gera sistema pereina prie 8 punkto, jei ne kartoja 5 punktą.
8. Vartotojas pasirenka kelią iki katalogo, kuriame yra bylos kuriose bus transformuojamas programinis kodas.	9. Sistema patikrina ar nurodytu keliu egzistuoja bylos su plėtiniais *.prg. Jei sistema rado bylų, tai sudaro nurodytam kataloge surastų bylų sąrašą. Jei sistema nerado tinkamų bylų vykdo 7 punktą.
	10. Sistema pateikia vartotojui surastų bylų sąrašą.
11. Vartotojas pažymi bylas iš pateikto sąrašo, kuriose reikia transformuoti programinį kodą.	12. Sistema koreguoja prieš tai sudarytą sąrašą palikdama tik tas bylas, kurios nebuvo pažymėtos kodo transformavimui.
	13. Sistema pateikia vartotojui pakeistą bylų sąrašą()
14. Vartotojas pažymi bylas, kurios turi įtakos transformuojamam programiniam kodui.	15. Sistema sukuria katalogą, kur bus talpinamos bylas su transformuotu programiniu kodu.

16. Vartotojas suveda reliacinės duomenų bazės prisijungimo duomenis.	17. Sistema bando prisijungti prie duomenų bazės.
18. Duomenų bazių valdymo sistemos atsakymas į bandymą prisijungti	19. Jei atsakymas teigiamas, tai sistema išsaugo nustatymus kompiuterio registre ir baigia darbą. Jei atsakymas neigiamas sistema pereina prie 15 punkto.
Po sąlyga	Kompiuterio registre išsaugoti nustatymų duomenys.

Pasinaudojant pradinių nustatymų panaudos atvejo specifikacijos 4.1 lentele, sukuriamas pradinių nustatymų panaudos atvejo specifikacijos sekų diagrama, kuri pateikiama 4.2 paveiksle. Diagramoje vaizdžiai matosi sistemos gyvavimo laikas, vartotojo ir sistemos sąveikos, sistemos sąveika su išorine sistema.

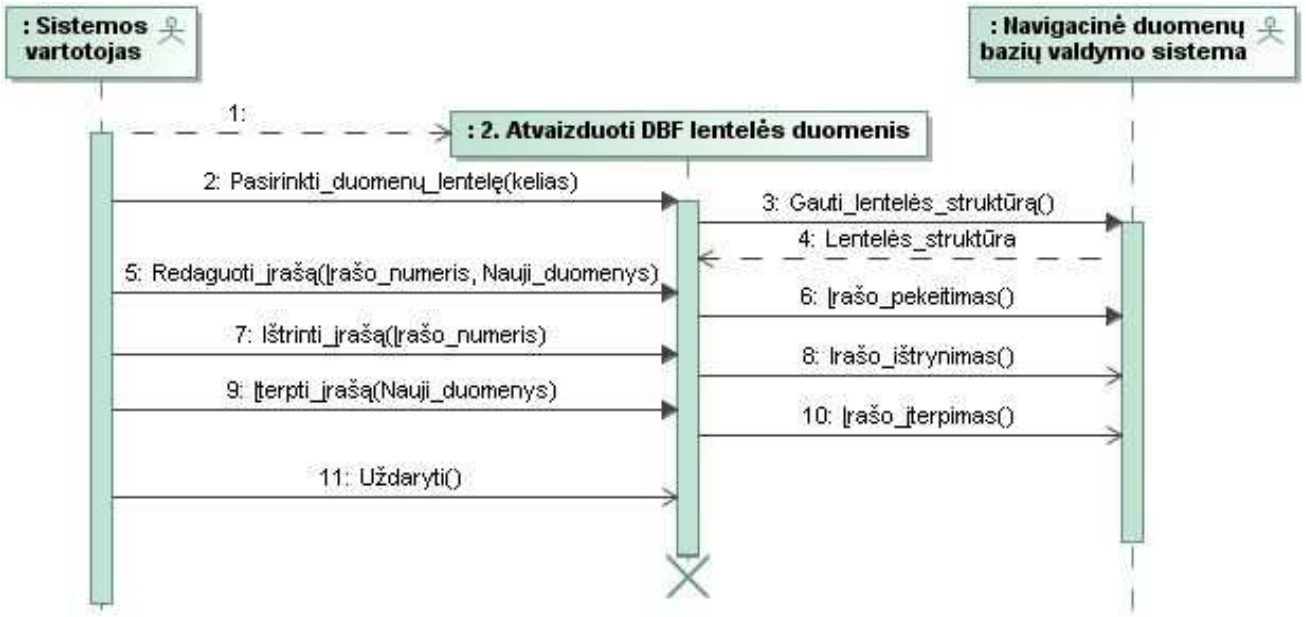


4.2 pav. **Panaudojimo atvejo „Įvesti pradinius duomenis“ specifikacijos sekų diagrama**

4.2 lentelė. Programinės įrangos DBF lentelės duomenų atvaizdavimo panaudos atvejų specifikacija

Panaudos atvejis 2. „Atvaizduoti DBF lentelės duomenis“	
Prieš sąlyga	Pradiniai nustatymai išsaugoti kompiuterio registre.
Sužadinimo sąlyga	1. Vartotojas nori peržiūrėti, redaguoti, įvesti, ištrinti duomenis

	iš DBF lentelės.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
2. Vartotojas nurodo rinkmeną, kurią nori atverti	3.Sistema klausia navigacinės duomenų bazių valdymo sistemos duomenų lentelės struktūros.
	4. Navigacinė duomenų bazių valdymo sistema grąžina duomenų lentelės struktūrą.
5. Vartotojas redaguoja įrašą.	6. Sistema perduoda įrašo pakeitimus navigacinei duomenų bazių valdymo sistemai.
7. Vartotojas trina įrašą.	8. Sistema perduoda trinamus įrašus navigacinei duomenų bazių valdymo sistemai.
9. Vartotojas įterpia naują įrašą	10. Sistema perduoda naujus duomenis navigacinei duomenų bazių valdymo sistemai.
11. Vartotojas užbaigia darbą.	Sistema uždaro rinkmeną ir sunaikiną duomenų atvaizdavimo langą.
Po sąlyga	Duomenų lentelė su pakeistais duomenimis, jei buvo redaguota arba ištrinta arba įterpta duomenų.



4.3 pav. **Panaudojimo atvejo „Atvaizduoti DBF lentelės duomenis“ specifikacijos sekų diagrama**

Dirbant su konvertavimo programine įranga iškyla poreikis peržiūrėti saugomus duomenis DBF lentelėse. Peržiūros metu vartotojas gali lengviau perprasti konvertuojamos programos algoritmo subtilybes. Detalizuojam panaudos atvejį „Atvaizduoti DBF lentelės duomenis“, pasinaudojant specifikacijos pagalba, kuri pateikiama 4.2 lentelėje.

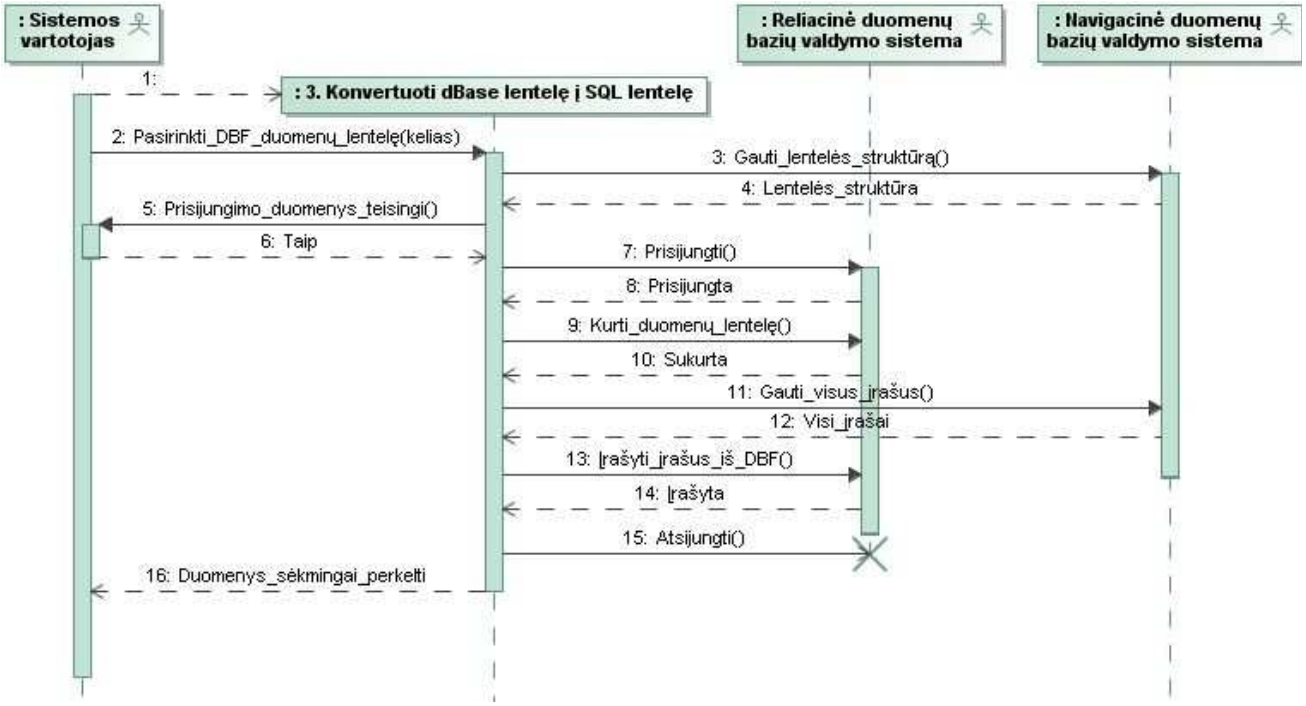
Pasinaudojant DBF lentelės duomenų atvaizdavimo panaudos atvejo specifikacijos 4.2 lentele, sukuriamas DBF lentelės duomenų atvaizdavimo panaudos atvejo specifikacijos sekų diagrama, kuri pateikiama 4.3 paveiksle. Diagramoje vaizdžiai matosi sistemos gyvavimo laikas, vartotojo ir sistemos sąveikos, sistemos sąveika su išorine sistema.

Konvertavus programinį kodą į SQL užklausas, vartotojai taip pat nori toliau dirbti su esamais duomenimis, todėl numatyta xBase lentelių konvertavimas į SQL lenteles galimybė. Detali lentelių konvertavimo panaudos atvejo specifikacija pateikiama 4.3 lentelėje. Taip pat apsinaudojant specifikacija paruošta sekų diagrama, kuri pateikiama 4.4 paveiksle.

4.3 lentelė. Programinės įrangos xBase lentelių konvertavimo į SQL lenteles panaudos atvejų specifikacija

Panaudos atvejis 3. „Konvertuoti xBase lentelę į SQL lentelę“	
Prieš sąlyga	DBF duomenų lentelė. Nustatymai kompiuterio registre.
Sužadinimo sąlyga	1. Vartotojas nori duomenis iš DBF lentelės perkelti į SQL lentelę.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
2. Vartotojas nurodo DBF duomenų lentelės rinkmeną	3. Sistema prašo navigacinės duomenų bazių valdymo sistemos lentelės struktūros.
	4. Navigacinė duomenų bazių valdymo sistema grąžina duomenų lentelės struktūrą.
	5. Sistema klausia vartotojo ar prisijungimo duomenys prie SQL bazės yra teisingi?
6. Vartotojo atsakymas į pateiktą klausimą.	7. Jei vartotojas atsakė taip tai sistema bando prisijungti prie SQL duomenų bazės. Jei vartotojas atsako ne sistema vykdo panaudos atvejį: 1. „Įvesti pradinius nustatymus“.
8. RDBVS atsako apie prisijungimo būklę.	9. Jei prisijungė sistema siunčia užklausą lentelės sukūrimui prieš tai sukauptai struktūrai. Jei neprisijungė, sistema vykdo A1.
10. RDBVS atsako apie duomenų lentelės sukūrimą.	11. Jei lentelė sukurta sėkmingai sistema siunčiu komandas navigacinei duomenų bazių valdymo sistemai visų įrašų gavimui. Jei lentelė nesukurta, sistema vykdo A1.
	12. Navigacinė duomenų bazių valdymo sistema grąžina įrašus.
	13.Sistema siunčia įterpimo užklausas RDBVS
	14. RDBVS atsako apie duomenų įterpimo veiksmo pasisekimą.
	15. Jei duomenys sėkmingai įtepti, sistema siunčia atsijungimo

	komandą. Jei įvyko klaida, tai vykdo A1.
	16. Sistema praneša apie tai, kad duomenys sėkmingai įkelti.
Po sąlyga	Sukelti duomenys iš DBF lentelės į SQL lentelę.
Alternatyvūs scenarijai	
A1. DBVS grąžino klaidos kodą.	A1.1 Sistema nutraukia scenarijaus vykdymą ir parodo klaidos kodą vartotojui.



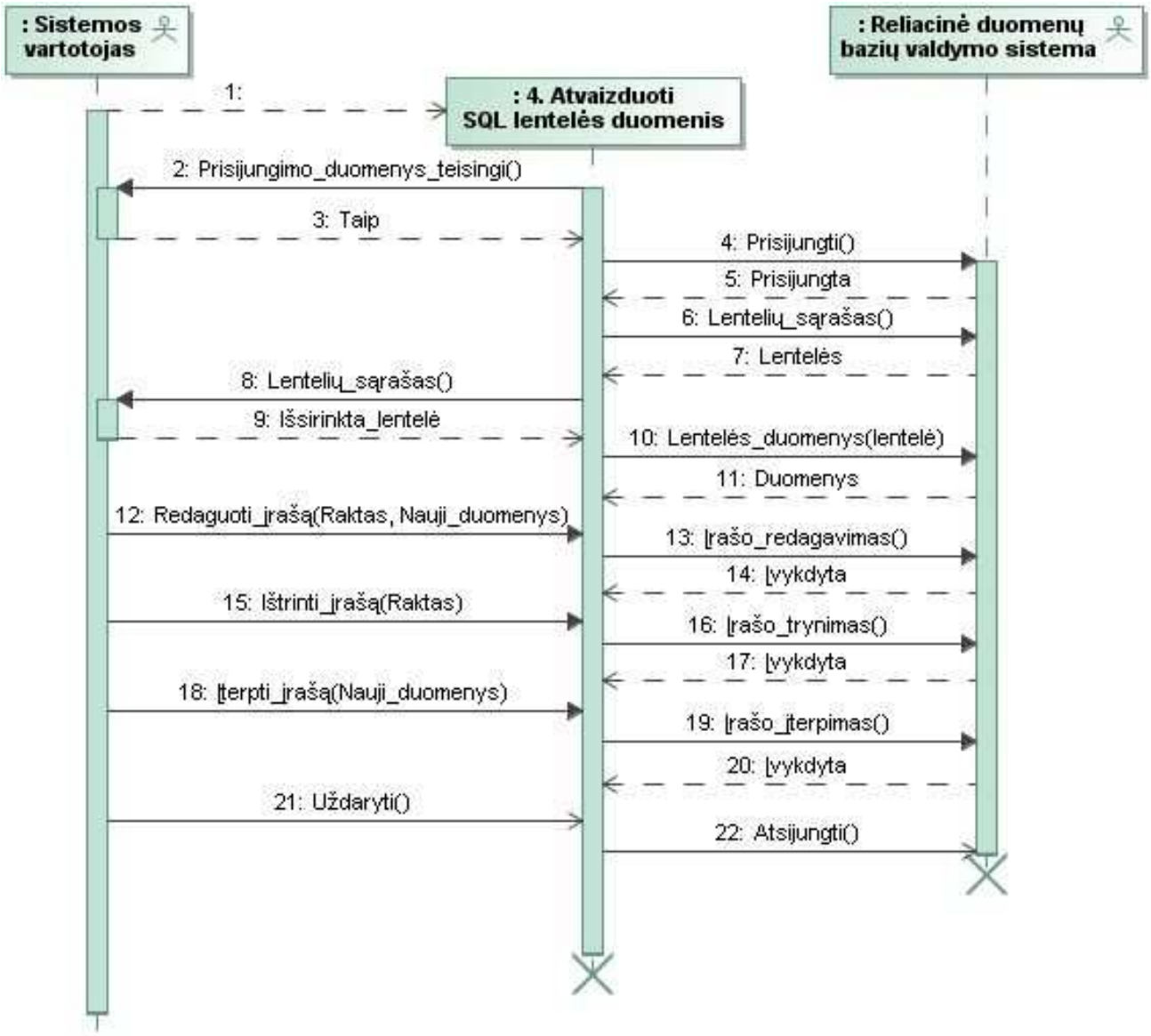
4.4 pav. **Panaudojimo atvejo „Konvertuoti xBase lentelę į SQL lentelę“ specifikacijos sekų diagrama**

Konvertavus duomenis iš xBase į SQL lenteles. Taip pat iškyla poreikis SQL lentelių peržiūrai, įrašų koregavimui. Detaliau aprašysime „Atvaizduoti SQL lentelės duomenis“ panaudos atvejį specifikacija, kuri pateikiama 4.4 lentelėje. Pasinaudojus specifikacija sukuriama ir SQL lentelės duomenų atvaizdavimo panaudos atvejo specifikacijos sekų diagrama 4.5 paveiksle. Sekų diagrama toliau bus naudojama kuriant programinę įrangą.

4.4 lentelė. Programinės įrangos SQL lentelės duomenų atvaizdavimo panaudos atvejų specifikacija

Panaudos atvejis 4. „Atvaizduoti SQL lentelės duomenis“	
Prieš sąlyga	Nustatymai išsaugoti kompiuterio registre.
Sužadinimo sąlyga	1. Vartotojas nori peržiūrėti, redaguoti, įterpti arba ištrinti duomenis.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
	2. Sistema klausia vartotojo ar prisijungimo duomenys prie

	SQL bazės yra teisingi?
3. Vartotojo atsakymas į pateiktą klausimą.	4. Jei taip tai sistema jungiasi prie SQL duomenų bazės. Jei ne tai vykdo A1.
5. RDBVS atsako apie prisijungimo būklę.	6. Jei prisijungė, tada siunčia užklausą gauti lentelių sąrašui. Jei ne vykdo A2 veiksmą.
7. RDBVS įvykdo užklausą ir grąžina rezultatą.	8. Jei neįvyko klaida sistema pateikia vartotojui lentelių sąrašą. Jei įvyko klaida, vykdo A2 veiksmą.
9. Vartotojas išsirenka lentelę.	10. Sistema siunčia užklausą pasirinktos lentelės duomenims gauti.
	11. RDBVS įvykdo užklausą ir grąžina rezultatą. Jei buvo grąžinti duomenys, tada sistema atvaizduoja duomenis. Jei įvyko klaida, vykdo A2 veiksmą.
12. Vartotojas redaguoja įrašą.	13. Sistema suformuoja užklausą ir siunčia serveriui.
	14. RDBVS įvykdo užklausą ir grąžina rezultatą. Jei redagavimas pavyko pereinam toliau. Jei įvyko klaida sistema vykdo A2 veiksmą.
15. Vartotojas ištrina įrašą.	16. Sistema suformuoja trynimo užklausą ir siunčia RDBVS.
	17. RDBVS įvykdo užklausą ir grąžina rezultatą. Jei įrašas ištrintas pereinam toliau. Jei įvyko klaida sistema vykdo A2 veiksmą.
18. Vartotojas suveda naujus duomenis.	19. Sistema suformuoja įterpimo užklausą ir siunčia RDBVS.
	20. RDBVS įvykdo užklausą ir grąžina rezultatą. Jei įrašas sėkmingai įterptas pereinam toliau. Jei įvyko klaida sistema vykdo A2 veiksmą.
21. Vartotojas paspaudžia uždaryti.	22. Sistema siunčia atsijungimo užklausą RDBVS. Sistema sunaikina duomenų atvaizdavimo langą.
Po sąlyga	Pakeista duomenų lentelė jei buvo redaguota, ištrinta arba įterpta duomenų.
Alternatyvūs scenarijai	
A1. Vartotojas nori pakeisti prisijungimo duomenis prie SQL duomenų bazės.	A1.1 Sistema vykdo panaudos atvejį 1. „Įvesti pradinius nustatymus“
A2. DBVS grąžino klaidos pranešimą.	A2.1 Sistema nutraukia darbą, vartotojui parodo klaidos pranešimą.



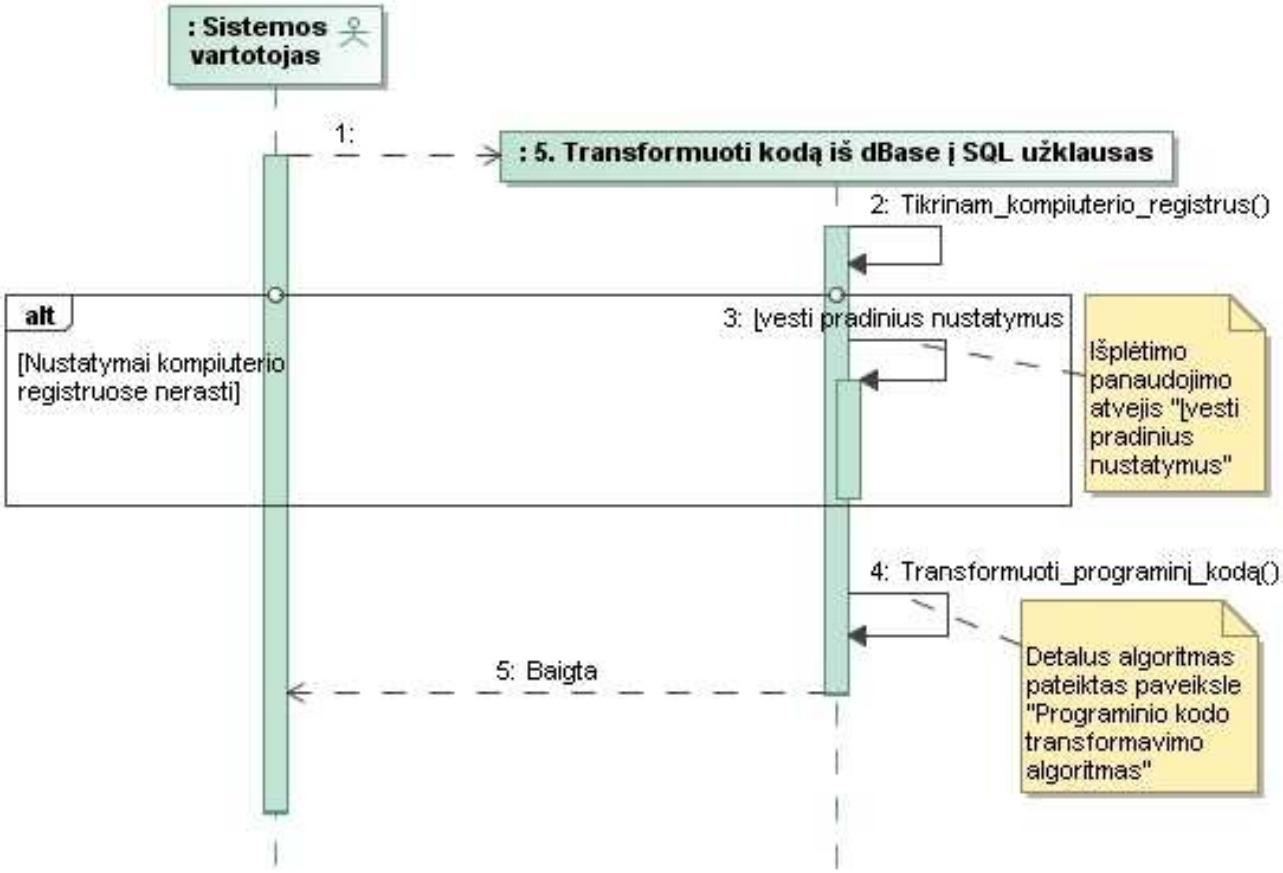
4.5 pav. **Panaudojimo atvejo „Atvaizduoti SQL lentelės duomenis“ specifikacijos sekų diagrama**

Pagrindinė ir svarbiausia projektuojamos programinės įrangos funkcija tai programinio kodo transformavimas iš xBase komandų į SQL užklausas. Detali specifikacija pateikiama 4.5 lentelėje, pagal kurią sukurtas kodo transformavimo algoritmas. Panaudos atvejo specifikacijai vaizdiniam apipavidalinimui panaudota sekų diagrama 4.6 paveikslas, kurios pagalba lengviau suprasti kada kokius veiksmus atlieka vartotojas, o kada sistema.

4.5 lentelė. Programinės įrangos xBase kodo transformacijos į SQL užklausas panaudos atvejų specifikacija

Panaudos atvejis 5. „Transformuoti kodą iš xBase į SQL užklausas“	
Prieš sąlyga	Nustatymai išsaugoti kompiuterio registre.

Sužadinimo sąlygos	1. Vartotojas nori konvertuoti programinį kodą.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
	2. Sistema tikrina kompiuterio registrus.
	3. Jei rado nustatymus pereina toliau. Jei nerado tada vykdo panaudos atvejį 1. „Įvesti pradinius nustatymus“.
	4. Sistema transformuoja programinį kodą. (detalus algoritmas pateiktas 3.9 paveiksle)
	5. Sistema praneša vartotojui apie baigtą transformavimą.
Po sąlyga	Transformuotas programinis kodas.

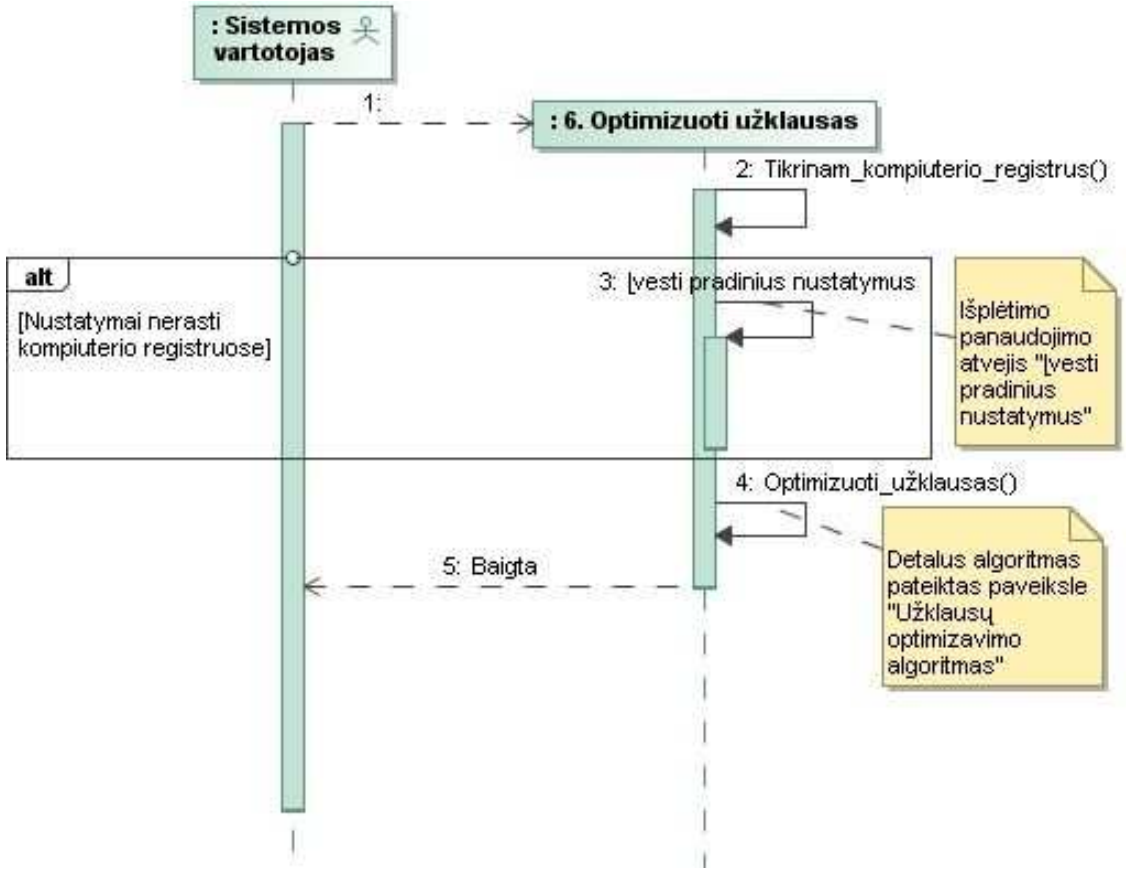


4.6 pav. **Panaudojimo atvejo „Transformuoti kodą iš xBase į SQL užklausas“ specifikacijos sekų diagrama**

Transformavus programinį kodą, naujoji programa turėtų dirbti greičiau, nei buvusi prieš tai, bet tam pasiekti dar reikia atlikti užklausų optimizaciją. Optimizacijos dėka užklausos yra pakoreguojamos, kad DBVS serveris lengviau ir greičiau atliktu užklausą. Pateikiame kodo optimizavimo panaudos atvejo specifikaciją 4.6 lentelė, kurioje matosi kaip programinė įranga apdoroja transformuota programinį kodą. Taip pat specifikacijos vizualizavimui sukurta specifikacijos sekų diagrama 4.7 paveikslas.

4.6 lentelė. Programinės įrangos užklausų optimizavimo panaudos atvejų specifikacija

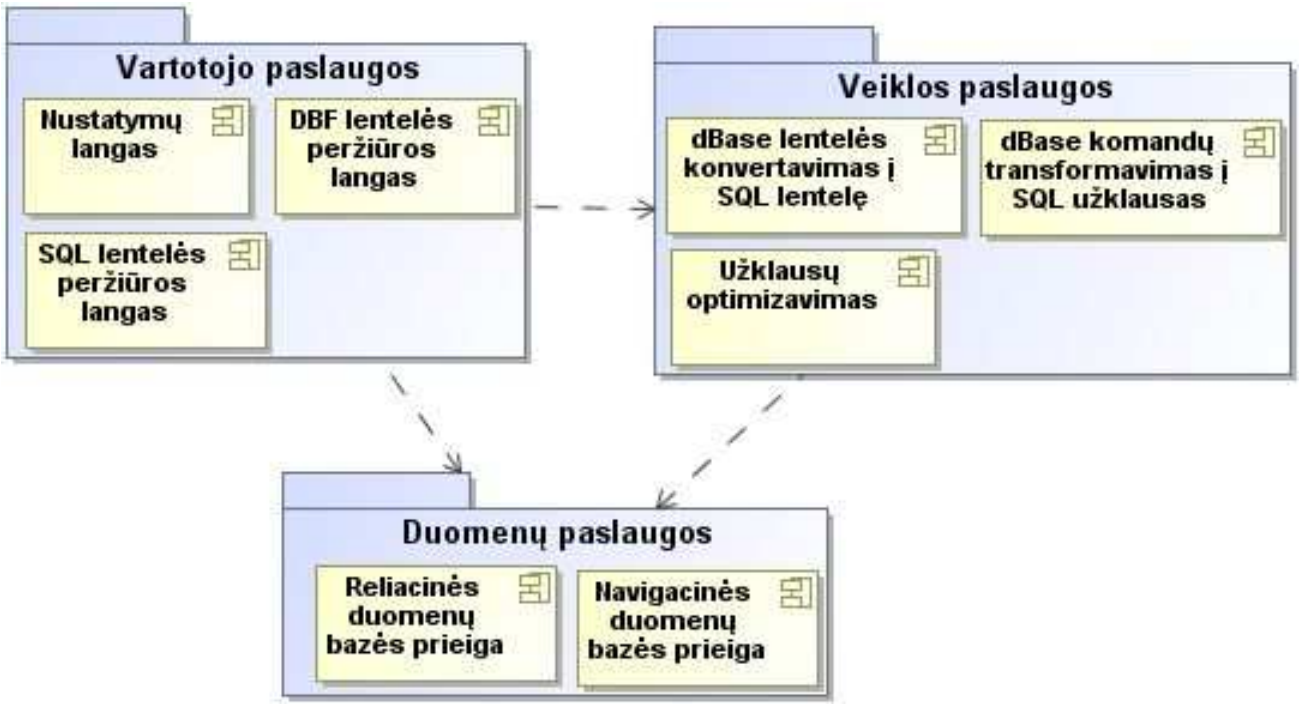
Panaudos atvejis 6. „Optimizuoti užklausas“	
Prieš sąlyga	Nustatymai išsaugoti kompiuterio registre. Transformuotas programinis kodas.
Sužadinimo sąlyga	1. Vartotojas nori optimizuoti transformuotas užklausas
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
	2. Sistema tikrina kompiuterio registrus.
	3. Jei rado nustatymus pereina toliau. Jei nerado tada vykdo panaudos atvejį 1. „Įvesti pradinius nustatymus“.
	4. Sistema optimizuoja transformuoto kodo užklausas (detalus algoritmas pateiktas 3.10 paveiksle)
	5. Sistema praneša vartotojui apie baigtą užklausų optimizaciją.
Po sąlyga	Optimizuotos užklausos transformuotame programiniame kode.



4.7 pav. **Panaudojimo atvejo „Optimizuoti užklausas“ specifikacijos sekų diagrama**

Pasinaudojus prieš tai sukurtomis panaudojimo atvejų specifikacijomis sukurta programinės įrangos trijų lygių architektūros diagrama, kuri pateikiama 4.8 paveiksle. Diagrama naudojama

kuriant programinės įrangos prototipą, padeda skaidyti sistema į mažesnes dalis ir realizuoti programiškai.



4.8 pav. Programinės įrangos architektūros diagrama

4.2. Kodo transformavimo taisyklių saugykla

Pasiūlytas programinio kodo transformavimo metodas, naudojasi transformavimo taisyklių saugykla, kurioje saugomos taisyklės kaip programinis kodas turi būti keičiamas. Programinio Kodo keitimas remiasi prieš tai sukurtomis taisyklėmis, kurių struktūra susideda iš dviejų dalių: xBase komandos šablonas ir SQL komandos šablonas. Pasinaudojant sukurtomis taisyklėmis suprojektuota duomenų bazė, kurioje saugomos taisyklės. Duomenų bazės schema pateikta 4.9 paveiksle, detalus schemos struktūros aprašymas pateiktas 4.7 lentelėje.

Taisykle		
PK	NUM	COUNTER
	KOMANDA UZKLAUSA	CHAR(100) CHAR(100)

4.9 pav. Transformavimo taisyklių duomenų bazės schema

4.7 lentelė Transformavimo taisyklių duomenų bazės aprašas

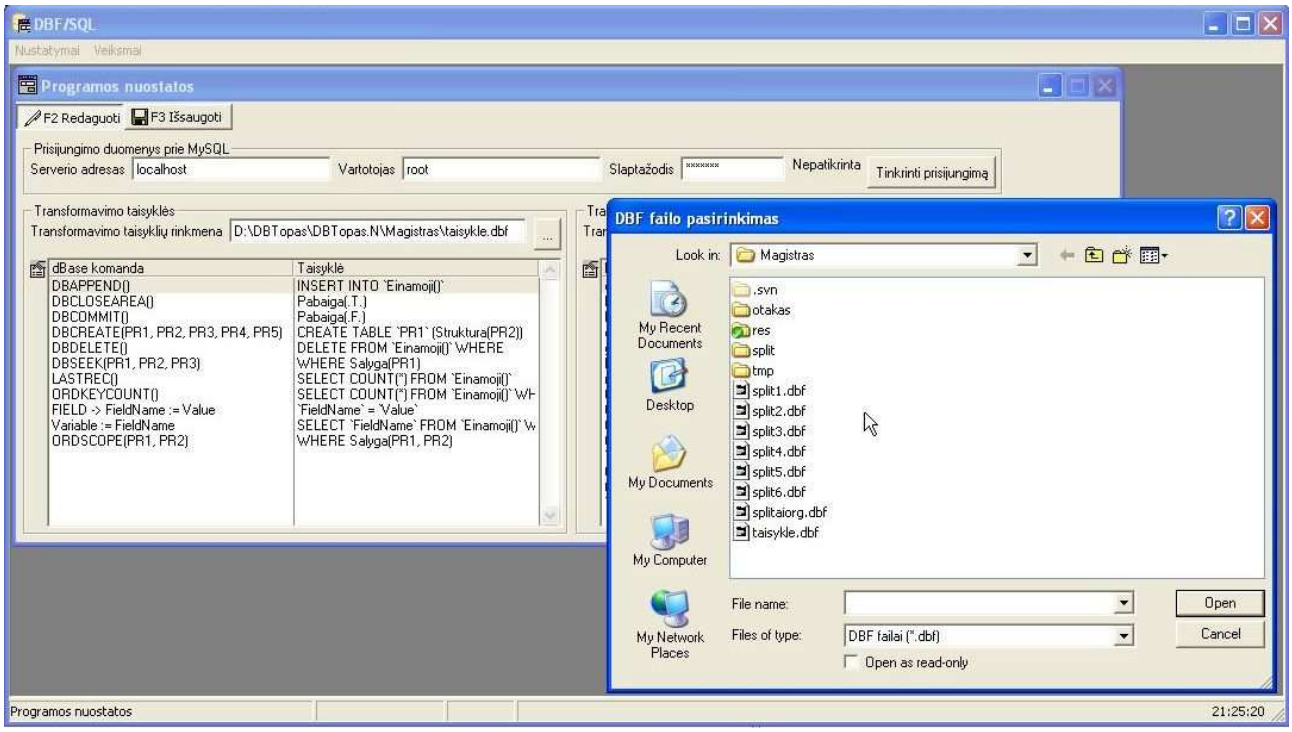
ATRIBUTAS	TIPAS	PASKIRTIS
NUM	NUMBER(10)	Pirminis lentelės raktas.
KOMANDA	CHAR(100)	xBase komandų šablonai.
UZKLAUSA	CHAR(100)	SQL užklausų šablonai.

4.3. Programinės įrangos prototipas

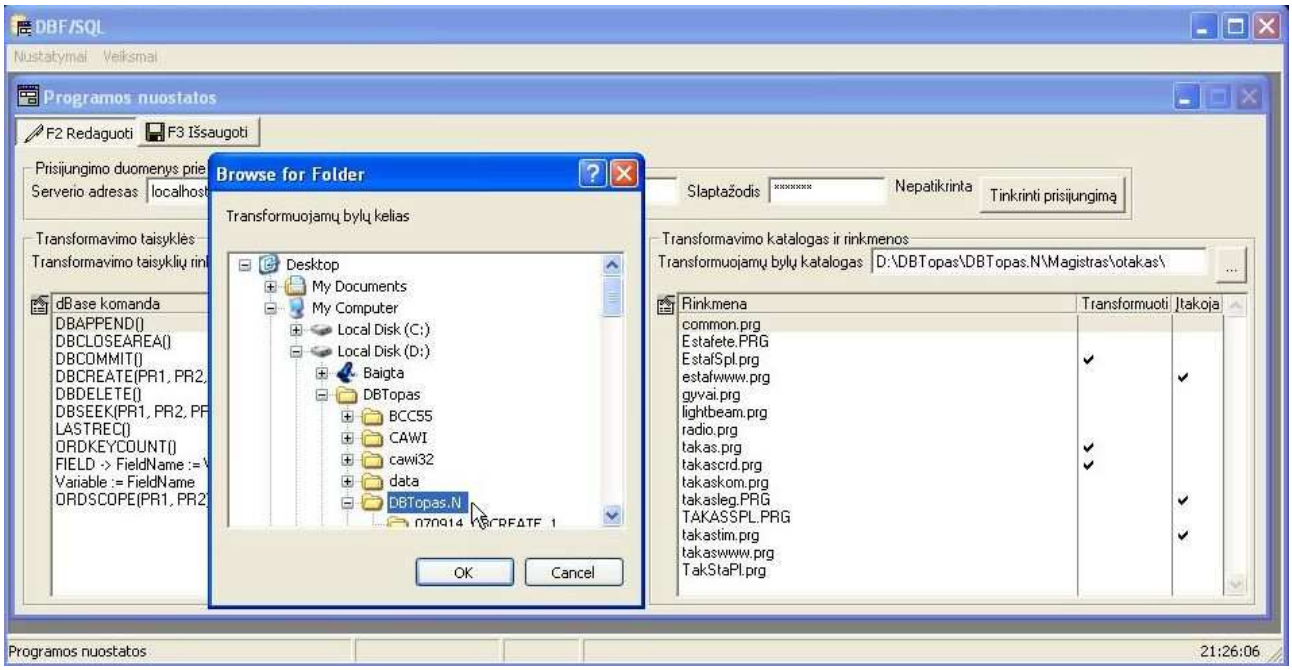
xBase komandų transformavimui į SQL užklausas buvo sukurtas metodas ir transformavimo taisyklių saugykla. Pagal pasiūlytą modelį sukurta programinė įranga, kuri leidžia atlikti xBase komandų transformavimą į SQL užklausas. Taip pat sukurtoje programinėje įrangoje įdiegtas ir papildomas funkcionalumas: duomenų bazės lentelių duomenų peržiūra, koregavimas, duomenų perkėlimas iš senų *.dbf lentelių į serverines duomenų bazės lenteles. Sukurtas sprendimas palengvina sistemos perkėlimą iš navigacinės duomenų bazių sistemos į šiuolaikines kliento serverio architektūros sistemas veikiančias DBVS pagrindu. Programinės įrangos prototipo išvaizda su išskleistais meniu punktais pateikiama 4.10 paveiksle.

4.10 pav. Programinės įrangos prototipo išvaizda su išskleistais meniu punktais

Pirmas žingsnis pradedant dirbti su prototipu tai yra nustatymai, taip kaip buvo numatyta specifikacijoje. Programoje pasirenkame nustatymai – programos nustatymai ir atsidariusiame lange spaudžiame mygtuką redaguoti. Redagavimo rėžime galima keisti transformavimo taisyklių saugyklos rinkmenos pasirinkimą, įvesti prisijungimo duomenis prie MySQL duomenų bazės ir aptikrinti ar įmanoma prisijungti. Paspaudus mygtuką su trimis taškiukais šalia transformavimo taisyklių rinkmenos įvedimo lauko atsidaro papildomas langas, kurio pagalba lengvai galime nurodyti rinkmeną. Taip pat paspaudus tris taškiukus šalia transformuojamų bylų katalogo įvesties lauko atsidaro papildomas langas, kuriame nurodome katalogą, kur randasi rinkmenos kurias žadame transformuoti. Aprašyti programos prototipo vaizdai pateikiami 4.11 ir 4.12 paveiksluose. Atlikus norimus pakeitimus būtina nepamiršti išsaugoti atliktus veiksmus.

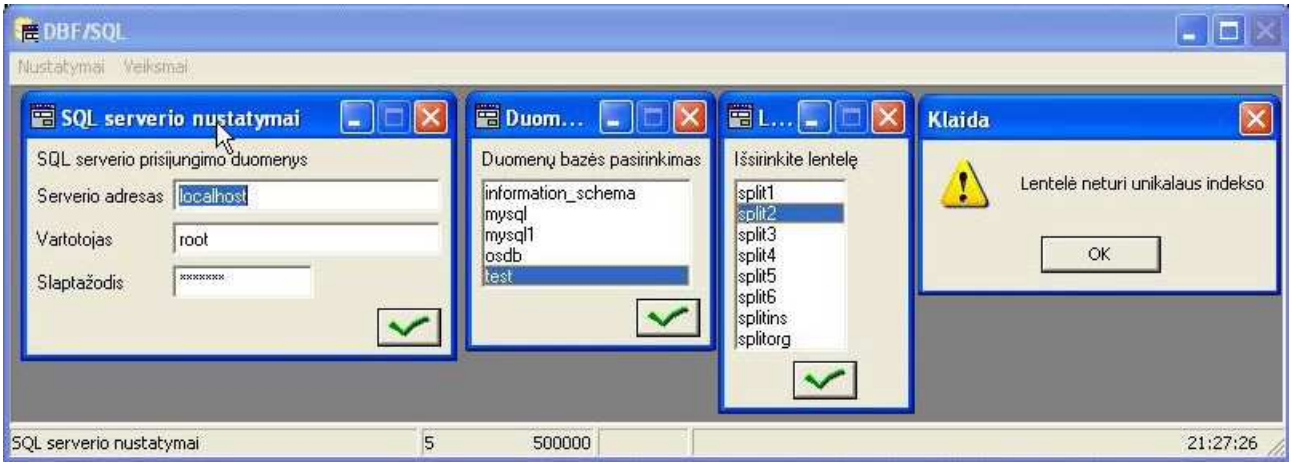


4.11 Transformavimo taisyklių rinkmenos pasirinkimas



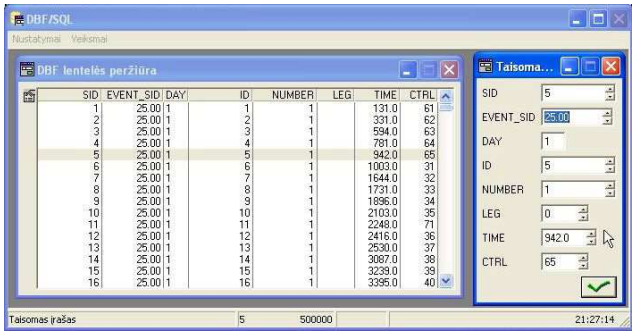
4.12 Transformuojamų rinkmenų katalogo pasirinkimas

Norint peržiūrėti duomenis iš MySQL lentelės reikia prisijungti prie duomenų bazės. Nurodomi duomenys: prisijungimo serverio adresas, vartotojo vardas, slaptažodis, jei duomenys buvo įvesti prieš tai pradiniuose nustatymuose, jų nebereikės įvedinėti užteks patvirtinti. Po to kai prisijungiama prie MySQL duomenų bazės programos prototipas pateikia duomenų bazių sąrašą, kuriame pasirenkama viena duomenų bazė, iš kurios pateikiamas lentelių sąrašas. Pasirinkus lentelę programa atidaro ją jei randa unikalų indeksą, kitu atveju grąžinamas klaidos pranešimas. Prisijungimo langų eiliškumo tvarka pateikiami 4.13 paveiksle.



4.13 Prisijungimo prie MySQL duomenų bazės lentelės dialogų eiliškumas

Atlikus visus reikiamus nustatymus programa leidžia peržiūrėti tiek MySQL lenteles tiek DBF duomenų lentelės rinkmenas, jas redaguoti, įterpti naujus įrašus, ištrinti nereikalingus. Įrašo redagavimo pavyzdys pateikiamas 4.14 paveiksle. Naujo įrašo įtraukimas vizualiai labai panašus veiksmas todėl kartu buvo ištrinta keletą įrašų. Programa ištrintus įrašus pradžiai atvaizduoja su perbraukta linija, vėliau kai uždarinėjamas langas arba nuspaudžiama klavišų kombinacija Ctrl+X yra fiziškai ištrinama, norint atšaukti įrašo ištrynimą reikia pakartotinai paspausti Delete klavišą, kuris panaikiną perbrauktą raudoną liniją. Aptartų veiksmų pavyzdys pateikiamas 4.15 paveiksle.



4.14 Įrašo redagavimas

4.15 Naujo įrašo sukūrimas ir įrašų trynimas

5. xBase komandų ir SQL komandų našumo tyrimas

5.1. Įrašų redagavimo našumas

Programinės įrangos palyginimui naudosime testus, kuriuose bus keičiamas vienodas kiekis įrašų skirtingo dydžio duomenų bazėse. xBase komandų ir SQL užklausų vykdymo spartai patikrinti bus matuojamos laiko sąnaudos kiek užtrunka pakeisti įrašus. Testavimo programos pagalba buvo atliekamas įrašų pakeitimas. Testavimo programos veikimo principas: pereina visą duomenų lentelę ir jei dalyvio numeris yra 16102 tai pakeičia varžybų numerį. Šis eksperimentas buvo atliekamas norint palyginti vykdymo laiką atliekant net pačią paprasčiausią užduotį. Panaudota duomenų lentelę susidedanti iš aštuonių stulpelių, kur buvo pakeistas 61 įrašas. Eksperimentas buvo pakartotas kelis sykius, norint įsitikinti, kad rezultatai yra korektiški. Pirmasis eksperimentas atliktas su lentelėmis, kurios neturėjo indeksinių laukų, testavimo programos kodas pateiktas žemiau, testo rezultatai pateikti 5.1 lentelėje.

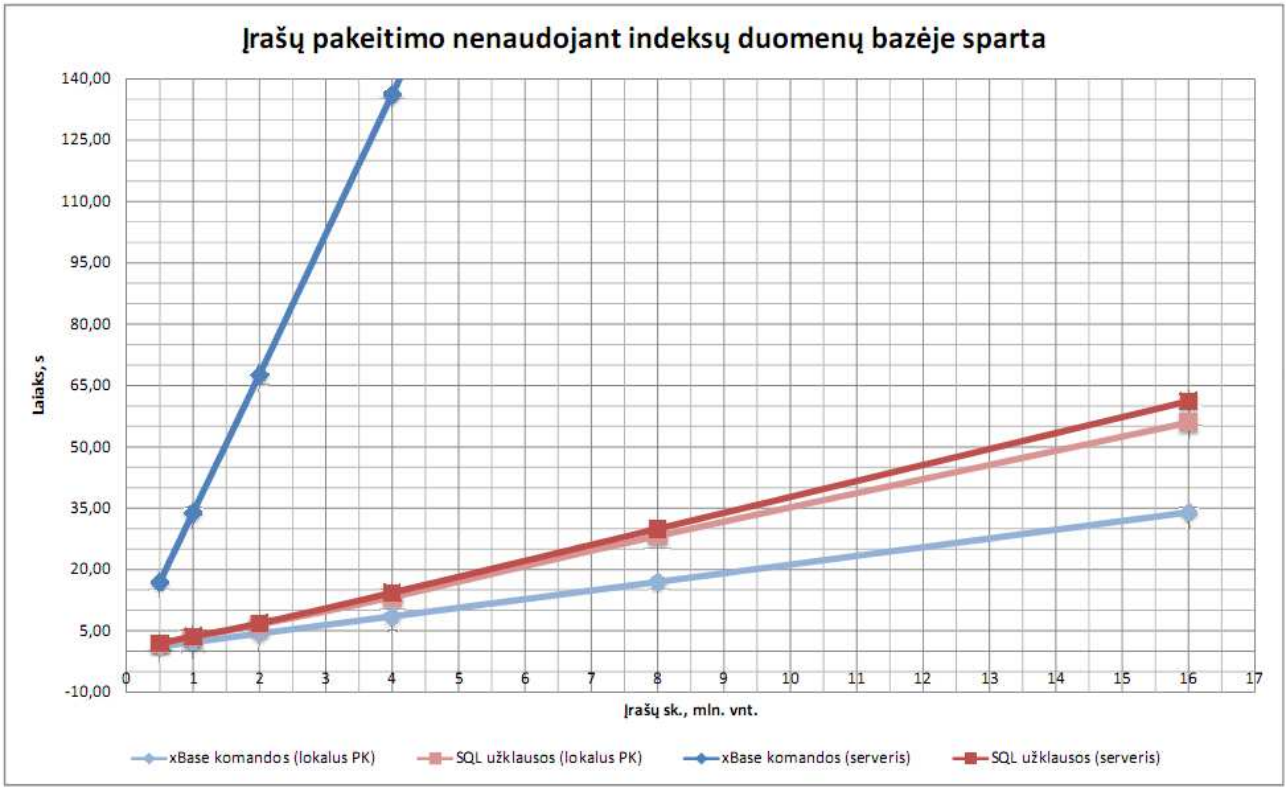
```
aSrit := {"split1", "split2", "split3", "split4", "split5", "split6"}
  FOR EACH aI IN aSrit
    goApp:StatusBarText({0, aI})
    nI := time()
    nI := 900 + val(SUBSTR(nI, 3, 2)) + val(SUBSTR(nI, 5, 2))
    DBUSEAREA(.T., "DBFCDX", aI,, .F., .F.)
    i := Seconds()
    DBEVAL({|| FIELD->EVENT_SID := nI}, {|| FIELD->NUMBER == 16102})
    debug("DBF", aI, Seconds()-i)
    (aI)->(DBCLOSEAREA())
    RDDINFO(RDDI_CONNECT, {"MYSQL", "localhost", "root", "Pedalai", "test"},
"SQLBASE", @hConnection)
    i:=Seconds()
    RDDINFO(RDDI_EXECUTE, "UPDATE `" + aI + "` set `EVENT_SID`=" + STR(nI, 4, 0)
+ " where `NUMBER`=16102;", "SQLBASE", hConnection)
    debug("SQL", aI, Seconds()-i)
    RDDINFO(RDDI_DISCONNECT, hConnection, "SQLBASE")
  NEXT
```

Nesinaudojant indeksais xBase komandos įvykdytos greičiau tik todėl, kad eksperimentas buvo vykdomas vieno kompiuterio pagalba. Dirbant tinklinėje sistemoje xBase komandos vykdomos lėčiau. Dirbti su duomenų baze, kuri yra lokaliam kompiuteryje ir kuri vienoje lentelėje turi nedaugiau kaip milijoną įrašų xBase komandomis išeitų daug greičiau. Darbo tikslas buvo

transformuoti xBase komandas į SQL užklausas, todėl tik didesnės sistemos turi būti transformuojamos ir pritaikomos dirbti su reliacinėmis duomenų bazėmis. Grafinė rezultatų analizė pateikiama 5.1 paveiksle, kurios pagalba matome, kad SQL užklausų veikimas kai serveris yra nutolęs vyksta sparčiau nei xBase komandos.

5.1 lentelė xBase ir SQL įrašų pakeitimo duomenų lentelėse vykdymo sparta, nenaudojant indeksų

Lentelė	Įrašų sk. lentelėje	Pakeistų įrašų sk.	Duomenys lokaliam kompiuteryje		Duomenys serveryje	
			xBase komanda įvykdyta per s	SQL užklausa įvykdyta per s	xBase komanda įvykdyta per s	SQL užklausa įvykdyta per s
Split1	500000	61	1,20	1,81	16,94	1,91
Split2	1000000	61	2,19	3,44	33,88	3,60
Split3	2000000	61	4,30	6,55	67,64	6,82
Split4	4000000	61	8,51	13,22	136,12	14,28
Split5	8000000	61	16,94	28,14	271,04	29,93
Split6	16000000	61	33,95	55,91	545,48	61,13



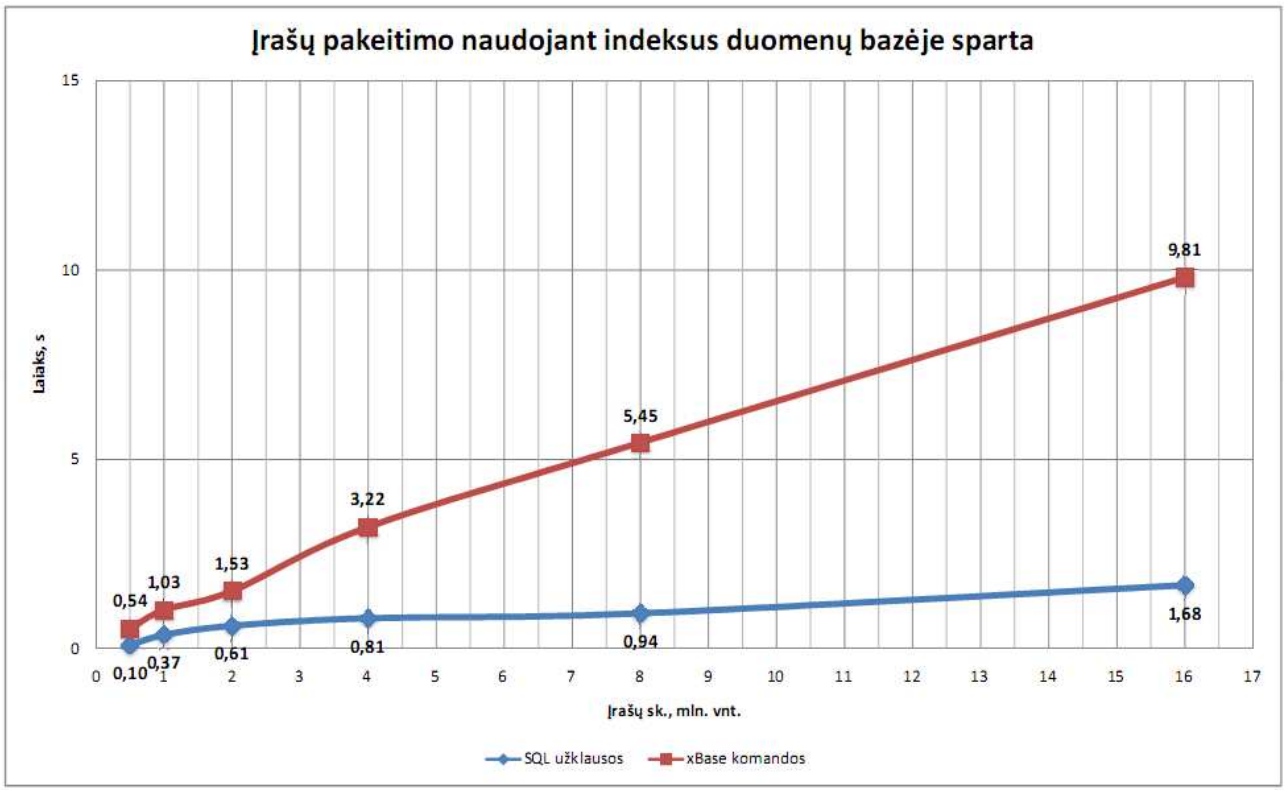
5.1 pav. xBase ir SQL įrašų pakeitimo duomenų lentelėse vykdymo sparta, nenaudojant indeksų

Buvo imituota reali duomenų bazė, kurioje yra indeksiniai laukai, raktiniai laukai, trigeriai, procedūros. Pakartojus prieš tai aprašytą bandymą, visos komandos ir užklausos užtruko mažiau kaip vieną šimtąją sekundės dalį, nes buvo keičiami tik 61 įrašai. Bandymas buvo pakeistas, kad užklausos kuo didesnėje bazėje pakeistų kuo daugiau įrašų. Bandymo programinis kodas, pateikiamas žemiau, o rezultatai 5.2 lentelėje.

```
aSrit := {"split1", "split2", "split3", "split4", "split5", "split6"}
  nI := 123456
  FOR EACH aI IN aSrit
    goApp:StatusBarText({0, aI})
    DBUSEAREA(.T., "DBFCDX", aI,, .F., .F.)
    ORDSETFOCUS("event")
    i := Seconds()
    ORDSCOPE(TOPSCOPE, 998)
    ORDSCOPE(BOTTOMSCOPE, 998)
    DBEVAL({|| FIELD->LEG := nI})
    ORDSCOPE(BOTTOMSCOPE, NIL)
    ORDSCOPE(TOPSCOPE, NIL)
    debug("DBF", aI, Seconds()-i)
    (aI)->(DBCLOSEAREA())
    RDDINFO(RDDI_CONNECT, {"MYSQL", "localhost", "root", "Pedalai", "test"},
"SQLBASE", @hConnection)
    i:=Seconds()
    RDDINFO(RDDI_EXECUTE, "UPDATE `" + aI + "` set `LEG`=" + STR(nI, 6, 0) +
" where `EVENT_SID`=998;", "SQLBASE", hConnection)
    debug("SQL", aI, Seconds()-i)
    RDDINFO(RDDI_DISCONNECT, hConnection, "SQLBASE")
  NEXT
```

5.2 lentelė xBase ir SQL įrašų pakeitimo duomenų lentelėse vykdymo sparta, naudojant indeksus

Lentelė	Įrašų sk. lentelėje	Pakeistų įrašų sk.	Pakeistų įrašų procentinė dalis	xBase komanda įvykdyta per s	SQL užklausa įvykdyta per s
Split1	500000	76754	15,4 %	0,54	0,10
Split2	1000000	148234	14,8 %	1,03	0,37
Split3	2000000	215211	10,8 %	1,53	0,61
Split4	4000000	458748	11,5 %	3,22	0,81
Split5	8000000	769262	9,6 %	5,45	0,94
Split6	16000000	1390290	8,7 %	9,81	1,68



5.2 pav. xBase ir SQL įrašų pakeitimo duomenų lentelėse vykdymo sparta, naudojant indeksus

Naudojantis indeksuotais laukais SQL užklausos veikimo sparta padidėja ir aplenkia xBase komandų veikimo spartą. Naudojant dideles duomenų bazių lenteles tinklas daug mažiau apkraunamas, naudojantis SQL užklausomis, todėl komandų transformavimas yra vertingas žingsnis, norint toliau palaikyti produkto gyvavimą.

5.2. Programinio kodo transformavimo bandymas

Sukurtame programos prototipe realizuota pagrindinė kodo transformavimo funkcija, kurios pagalba xBase komandos pakeičiamos SQL užklausomis. Taip pat programinės įrangos projekte buvo numatytas papildomas funkcionalumas, kuris dalinai buvo įgyvendintas programinės įrangos prototipe: prisijungimas prie MySQL duomenų bazės, prisijungimas prie xBase lentelių, duomenų konvertavimas iš DBF rinkmenų į SQL lenteles, koreguoti, įrašyti, ištrinti bet kurios lentelės įrašus.

Pasinaudojus programinės įrangos prototipu buvo atliktas eksperimentas konvertuojant programinį kodą iš xBase komandų į SQL užklausas. Bandymas atliktas su viena rinkmena, kuri neturėjo įtakojančių rinkmenų. Prototipo konvertuotas programinio kodo fragmentas pateikiamas žemiau.

```
STATIC PROC radio_control_punch(pnECNum, pnTime, pnControl)
  FIELD DIENA, E_CARD, KP, LAIKAS, NUMERIS
  LOCAL nLine
  IF FilLock()
```

```
        NaujasId("id")
        RDDINFO(RDDI_EXECUTE, "INSERT INTO `" + soBrRadio:nArea + "` `E_CARD` VALUES (" + pnECNum + ")", "SQLBASE", hConnection)
        RDDINFO(RDDI_EXECUTE, "INSERT INTO `" + soBrRadio:nArea + "` `LAIKAS` VALUES (" + pnTime + IF(pnTime < ghVar["SR_" + gcDiena], 12 * 3600, 0) + ")", "SQLBASE", hConnection)
        RDDINFO(RDDI_EXECUTE, "INSERT INTO `" + soBrRadio:nArea + "` `KP` VALUES (" + pnControl + ")", "SQLBASE", hConnection)
        RDDINFO(RDDI_EXECUTE, "INSERT INTO `" + soBrRadio:nArea + "` `DIENA` VALUES (" + gcDiena + ")", "SQLBASE", hConnection)
        FilUnLock()
      ENDIF
      IF (nLine := soBrRadio:nRowPos + soBrRadio:nCount - soBrRadio:nPos) <= soBrRadio:nRowCount
        soBrRadio:aId[nLine] := RECNO()
        soBrRadio:FreshLine(nLine)
      ENDIF
      soBrRadio:nCount += 1
      IF soBrRadio:lEOF; soBrRadio:Down()
      ENDIF
      soBrRadio:FreshStatus()
    RETURN
```

Konvertuotas programinis kodas turi keletą užklausų, kurios gali būti apjungiamos į vieną. Konvertavimo metu tokias užklausas programinės įrangos prototipas, kuria kiekvienai pažystamai eilutei, todėl rekomenduotina praleisti konvertuotas programinės įrangos rinkmenas pro transformuoto kodo optimizavimo įrankį, kuris tokias užklausas turėtų apjungti į vieną visumą. Šita galimybė nėra įgyvendinta programinės įrangos prototipe, bet 3.2 skyriuje yra pateikiamas algoritmas, kaip optimizacija turėtų būti vykdoma. Žemiau pateikiamas transformuotas kodas, kuris buvo pataisytas programuotojų į tokį koks turėtų būti po programinės įrangos optimizavimo.

```
    STATIC PROC radio_control_punch(pnECNum, pnTime, pnControl)
      LOCAL nLine
      RDDINFO(RDDI_EXECUTE, "INSERT INTO `" + soBrRadio:nArea + "` (`ID`, `E_CARD`, `LAIKAS`, `KP`, `DIENA`) VALUES (0, " + pnECNum + ", " + + IF(pnTime < ghVar["SR_" + gcDiena], 12 * 3600, 0) + ", " + pnControl + ", " + gcDiena + ")", "SQLBASE", hConnection)
      IF (nLine := soBrRadio:nRowPos + soBrRadio:nCount - soBrRadio:nPos) <= soBrRadio:nRowCount
        soBrRadio:aId[nLine] := RECNO()
        soBrRadio:FreshLine(nLine)
      ENDIF
      soBrRadio:nCount += 1
      IF soBrRadio:lEOF; soBrRadio:Down()
      ENDIF
      soBrRadio:FreshStatus()
    RETURN
```

6. Išvados

1. Išanalizavus šiuo metu pateikiamus sprendimus padedančius paspartinti xBase komandų darbo spartą nustatyta, kad vienintelis būdas patikimai ir saugiai pagreitinti programinės įrangos veikimą tai pakeisti programinės įrangos kodą iš xBase komandų į SQL komandas.
2. Literatūros šaltinių ir egzistuojančių interneto paslaugų sistemų analizė parodė, kad programinė įranga sukurta remiantis xBase duomenų bazės valdymo sistema yra valdoma nedideliu kiekiu elementarių komandų kombinacijų, kurias galima transformuoti į SQL užklausas.
3. Atlikus xBase komandų analizę ir jų atitikmenų SQL užklausų tyrimą, nuspręsta, kad xBase komandų transformavimas į SQL konstrukcijas gali būti automatizuotas. Pasiūlytas algoritmas ir taisyklių rinkinys xBase komandų transformavimui į SQL užklausas.
4. Pasiūlytas metodas buvo įgyvendintas realizuojant kodo transformavimo programinės įrangos prototipą, kuris automatizuoja kodo transformavimo algoritmą. Algoritmo automatizavimas padeda sparčiau transformuoti programinės įrangos kodą, kuris yra pritaikomas naudotis reliacinėmis duomenų bazėmis.
5. Eksperimento metu pastebėta, kad atliktas kodo transformavimas palengvina atpažinti ir modifikuoti nekorektiškus rezultatus transformuotame kode, kurie atsiranda šabloniško transformavimo metu.
6. Eksperimento metu nustatyta, kad transformuotas kodas turi būti optimizuojamas, užklausų vykdymo pagreitinimui ir supaprastinimui, nes programinės įrangos prototipas šabloniškai transformuoja komandas į užklausas, kurios skirtingais atvejais gali būti užrašomos skirtingai.
7. Pasiūlytas sprendimas buvo pristatytas ir aprobuotas 14-toje tarpuniversitetinėje magistrantų ir doktorantų mokslinėje konferencijoje „Informacinės technologijos“.

Literatūra

[1] Paulson, L.D. „Open source databases move into the marketplace“, Computer, Volume 37, July 2004, 13 – 15;

[2] Ramana, U.V.; Prabhakar, T.V., „Some experiments with the performance of LAMP architecture“, Computer and Information Technology, 2005, The Fifth International Conference, 916 – 920;

[3] Jain, A. Doan, A. Gravano, L. Columbia Univ., New York, NY; „SQL Queries Over Unstructured Text Databases“, Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference, Istanbul.

[4] Jain, Alpa Doan, AnHai Gravano, Luis, Computer Science Department, Columbia University „Optimizing SQL Queries over Text Databases“, Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference, Cancun, Mexico.

[5] R. Jasinevičius, V. Petrauskas, Miglotieji pažintiniai planai: teorija ir praktika (108 – 111 psl.), Informacijos Mokslai 26 tomas, Vilniaus universiteto leidykla, 2003.

[6] R. Motwani, S.U. Nabar, D. Thomas, „Auditing SQL Queries“, Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference, April 2008, 287 – 296.

[7] Dušan Petković, „Microsoft SQL server 2008. A Begginer‘s guide“, The McGraw-Hill Companies, 2008.

[8] Ken England, Gavin Powell, „Microsoft® SQL Server 2005 Performance Optimization and Tuning Handbook“, Digital Press, Elsevier, USA, 2007.

[9] Paul DuBois, „MySQL developer‘s library“, ketvirtas leidimas, Addison-Wesley, USA, 2008.

[10] Nemokamas Borland C++ Compiler 5.5 [interaktuvus] [žiūrėta 2009-05-12]. Prieiga per internetą: <http://www.codegear.com/downloads/free/cppbuilder>

[11] Nemokama atviro kodo programa Harbour 1.0.1, [interaktuvus] [žiūrėta 2009-05-12]. Prieiga per internetą: <http://www.harbour-project.org/>.

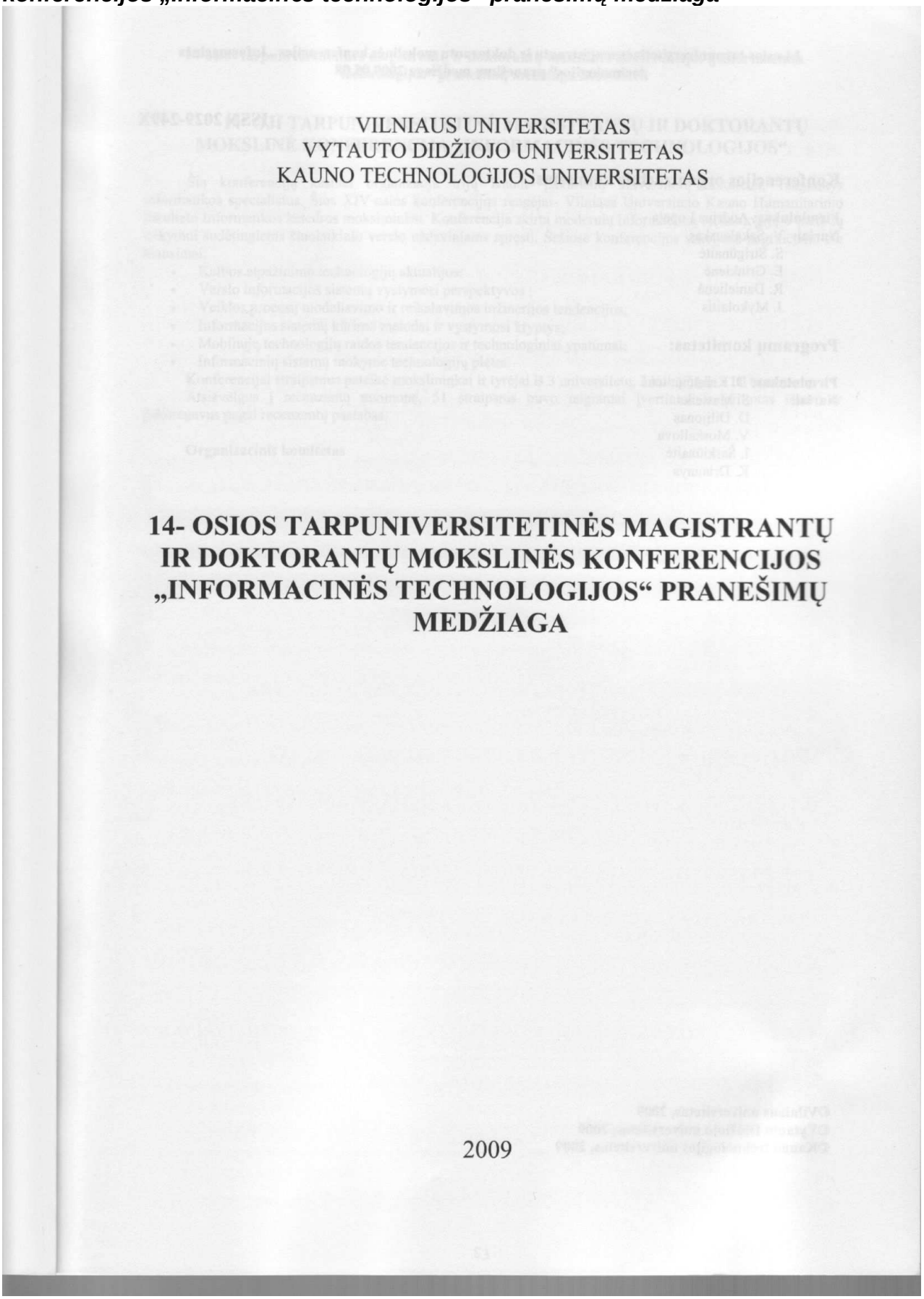
[12] Atviro kodo MySQL 5.1 duomenų bazių valdymo sistema, [interaktuvus] [žiūrėta 2009-05-12]. Prieiga per internetą: <http://www.mysql.com/>.

[13] Jakšta Deividas, SQL serverių efektyvumo tyrimai vidutinio duomenų bazių projektuose, Magistro darbas, Šiauliai 2006/2007 m.m.

[14] SQLRDD bibliotekos aprašymas , [interaktyvus] [žiūrėta 2009-04-22]. Prieiga per internetą: <http://www.xharbour.com/index.asp?page=products_addon_sqlrdd_2>.

Priedai

A. Publikacija leidinyje „14-osios tarptautinės magistrantų ir doktorantų mokslinės konferencijos „Informacinės technologijos“ pranešimų medžiaga“



VILNIAUS UNIVERSITETAS
VYTAUTO DIDŽIOJO UNIVERSITETAS
KAUNO TECHNOLOGIJOS UNIVERSITETAS

14- OSIOS TARPUNIVERSITETINĖS MAGISTRANTŲ IR DOKTORANTŲ MOKSLINĖS KONFERENCIJOS „INFORMACINĖS TECHNOLOGIJOS“ PRANEŠIMŲ MEDŽIAGA

2009

XBASE KOMANDŲ KEITIMAS SQL UŽKLAUSOMIS

Darius Kasiulevičius[1], Tomas Danikauskas[1]

[1]Kauno technologijos universitetas, informatikos fakultetas, informacijos sistemų katedra, Studentų g. 50 – 313a, LT-51368, Kaunas, Lietuva, dariustri@centras.lt, danitoma@soften.ktu.lt

Anotacija. Darbe analizuojama xBase komandų transformavimo į SQL užklausas problema. Pateikiamas konceptualus transformavimo problemos sprendimas, kurį sudaro dvi dalys: transformavimo algoritmas ir transformavimo taisyklių saugykla. Pagal pasiūlytą modelį sukurta programinė įranga, kuri leidžia atlikti xBase komandų transformavimą į SQL užklausas. Taip pat sukurtoje programinėje įrangoje įdiegtas ir papildomas funkcionalumas: duomenų bazės lentelių duomenų peržiūra, koregavimas, duomenų perkėlimas iš senų *.dbf lentelių į serverinės duomenų bazės lenteles. Sukurtas sprendimas palengvina sistemos perkėlimą iš navigacinės duomenų bazių sistemos į šiuolaikines kliento serverio architektūros sistemas veikiančias serverinių dbvs pagrindu.

PAGRINDINIAI ŽODŽIAI: xBase, SQL, transformavimas, aibės.

1 Įvadas

Tobulėjant šiuolaikinėms technologijoms, sparčiai auga poreikis senų kompiuterizuotų sistemų atnaujinimui. Atliekant esamų sistemų atnaujinimą dažniausia siekiama perkelti visa jose esamą funkcionalumą. Tuo tikslu esama (palikuoninė) sistema yra išanalizuojama iš jos išgaunami senieji reikalavimai, funkcionalumas ir atnaujinus reikalavimus kuriama nauja sistema. Atsižvelgus į esamos sistemos privalumus ir trūkumus visada galima pasiekti kokybiškesnio naujos sistemos sukūrimo rezultato [1].

Šio darbo tikslas – sukurti metodą kaip efektyviai ir greitai transformuoti Harbour programavimo kalbos xBase komandas į SQL užklausas, siekiant senas programas pritaikyti šiuolaikinėms duomenų bazių valdymo sistemoms (DBVS). Pagrindinis darbo uždavinys – sukurti programinį įrankį, kuris padėtų pereiti iš xBase tekstinių duomenų bazių komandų prie reliacinės duomenų bazės SQL užklausų. Atlikus analizę nustatyta, kad esamos programos sukurtos su xBase komandomis naudojasi nedideliu kiekiu paprastų komandų, kurios nesudėtingai gali būti pakeistos SQL užklausomis.

2 xBase komandų transformavimas

xBase yra bendras terminas visoms programavimo kalboms, kurios kilo iš originalios dBase (Aston-Tate) programavimo kalbos. xBase ir kitų panašių produktų esminė problema tai kad jie nebuvo pagrįsti kliento - serverio modeliu. Tai yra kai duomenų bazę naudoja tinkle daugybė vartotojų, sistema parsisiunčia didelę dalį rinkmenos į vartotojo kompiuterį, kur ir atlieka užklausas. Taip tinklas yra labai apkraunamas, nes kiekvieną kartą atliekant vis tas pačias užklausas rinkmenos iš naujo yra siunčiamos per tinklą. Kliento – serverio modelyje, klientas siunčia nedideles užklausų komandas į serverį, kur serveris įvykdęs užklausas gražina tik rezultatą. Taip tinklo apkrovimas yra sumažinamas.

xBase kalboje parašytos komandos dirba su pavieniais failais, kurie dažniausiai būna lokaliame kompiuteryje, todėl tokios programos būna dažniausiai orientuotos į vieno vartotojo programas. Pasitelkus SQL užklausas programos kuriamos didesnei vartotojų grupei, kuri dirbs su duomenimis esančiais serveryje. Priėjimas prie lokalių rinkmenų yra patikrinamas programos paleidimo metu, naudojantis xBase komandomis. Tikrinimas atliekamas tokių būdų: pasinaudojant operacinės sistemos komandomis yra patikrinama ar egzistuoja aplankalas, ar aplankalo teisėse yra nurodyta galimybė rašyti. Tada patikrinamas aplankalo turinys, patikrinamos visos duomenų lentelės, ar turi rašymo teises. SQL atveju yra sukuriamas prisijungimas prie serverio, atidaroma norima duomenų bazė ir toliau vyksta darbas [2]. Prisijungimo metu yra patikrinama ar egzistuoja norima duomenų bazė, ar tinkami prisijungimo duomenys. SQL lentelių ir duomenų bazės paprastas vartotojas, kuris naudojasi tik programa, neištrins, o naudojantis rinkmenų (*.dbf) duomenų baze, vartotojas gali nesuprasdamas ką daro ištrinti visus duomenis, nes visos lentelės saugomos lokaliai ir laisvai prieinamos. Todėl SQL duomenų bazėse saugomi duomenys yra saugesni.

2.1 Komandų ir užklausų atitikmenys

Šiame skyrelyje pateikiamas xBase komandų ir SQL konstrukcijų atitikmenų analizė. Tarkime kiekviena duomenų lentelė tai atskira aibė $T_l = \{x^l_{\overline{1,r}} : x^l_n \neq x^l_m\}$, čia $n, m = \overline{1, r}$, $n \neq m$, $l = \overline{1, t}$, t – lentelių skaičius duomenų bazėje, r – eilučių skaičius duomenų lentelėje, x^l_1 – duomenų lentelės l, 1-oji eilutė, kurios

išraiška $x^{l}_{\overline{1,r}} = \left(x^{l}_{\overline{1,r},1}, x^{l}_{\overline{1,r},2}, x^{l}_{\overline{1,r},3}, ..., x^{l}_{\overline{1,r},c}\right)$, c – lentelės l stulpelių skaičius. Visų lentelių duomenys yra būdingi tik tai lentelei, todėl $T_i \not\subset T_j$, čia $i, j = \overline{1,t}$, $i \neq j$, t – lentelių skaičius duomenų bazėje. Visą duomenų bazę sudaro baigtinis skaičius lentelių $DB = \bigcup_{l=1}^{t} T_l$, čia DB – duomenų bazės aibė.

Visas xBase komandas išreikšime per apibrėžtas aibes [3], kurių pagalba po to užrašysime SQL užklausą. Programinėje įrangoje taisyklės aprašomos taip pat naudojantis šitomis pradinėmis duomenų bazės, lentelių, laukų apibrėžimais. Pagrindiniai veiksmai, kurie atliekami naudojantis duomenų bazėmis tai: duomenų sukūrimas, duomenų redagavimas, duomenų šalinimas ir duomenų atrinkimas. Pateiksime keletą pavyzdžių, kuriais remiantis pailiustruosime kaip aprašyti xBase komandas aibėmis ir predikatais. Pavyzdžiuose pateiksime įrašo įterpimo ir išrinkimo komandas. Duomenų redagavimo ir šalinimo kodo transformavimas yra analogiškas pateiktiems pavyzdžiams.

2.1.1 Įrašo įterpimas

Naujo įrašo įterpimas, tai naujos reikšmės pridėjimas esamai aibei: $T_l^{'} = T_l \cup z$, čia $T_l^{'}$ – duomenų bazės lentelė l, po naujo elemento (eilutės) įterpimo, z – naujas elementas, eilutė. Pagal pradines sąlygas naujas elementas gali būti įtrauktas tada ir tik tada jei tokio elemento dar nebuvo: $z \notin T_l$, tuomet $T_l^{'} = \{x^{l}_{\overline{1,r}}, z : x^{l}_{n} \neq x^{l}_{m} \neq z\}$, čia $n, m = \overline{1,r}$, $n \neq m$, $l = 1,2,...,n$, toliau $T_l^{'}$ žymėsime T_l, tiesiog pasikeitė elementų skaičius, bet jis neturi įtakos programinio kodo transformavimui. Kodo transformavimo pavyzdys pateiktas 1 lentelėje. Komanda *DbUseArea()* atitinka konkrečios lentelės T_l išrinkimą duomenų bazėje $DB = \left(\bigcup_{i=1}^{l-1} T_i\right) \bigcup T_l \bigcup \left(\bigcup_{i=l+1}^{t} T_i\right)$. Komanda *DbAppend()* atitinka aibės sąjungos operaciją. FIELD -> cLaukas tai būtų konkrečios reikšmės priskyrimas naujame įraše, aibėmis apsirašytų taip: $z = (z_1, z_2, z_3, ..., z_c)$, čia c – įterpiamų laukų skaičius, kuris turi būti mažesnis arba lygus lentelės l laukų skaičiui. Naujos aibės atsiradimas – *DbCommit()*. SQL užklausomis atitinkamai lentelės išsirinkimas *RDDI_CONNECT*, sąjungos operaciją ir naują reikšmę – *Insert Into `lentele` (cLaukas, nLaukas, dLaukas, lLaukas, mLaukas) Values('Pirmoji eilutė', 10.25, '20080529', 1, 'Labai ilga eilutė ...')*, naujos aibės atsiradimas *RDDI_DISCCONECT*.

Lantelė 1. Naujo įrašo įterpimas.

xBase komandomis [4]	SQL užklausomis

`DbUseArea(.T., "DBFCDX", "C:\lentele.dbf", "AliasLentele", .T., .F.)`
`FLock()`
`DbAppend()`
`FIELD -> cLaukas := "Pirmoji eilutė"`
`FIELD -> nLaukas := 10.25`
`FIELD -> dLaukas := STOD("20080529")`
`FIELD -> lLaukas := .T.`
`FIELD -> mLaukas := "Labai ilga eilutė ..."`
`DbCommit()`
`DbUnlock()`
`DbCloseArea()`
	`LOCAL cRddName := "SQLBASE", hConnection`
	
	`RDDINFO(RDDI_CONNECT, {"MYSQL", "www.serveris.lt", "Vartotojas", "Slaptažodis", "duombazė"}, cRddName, @hConnection)`
	
	`RDDINFO(RDDI_EXECUTE, "autocommit=1;" +; "Insert Into `lentele` (cLaukas, nLaukas, dLaukas, lLaukas, mLaukas) Values('Pirmoji eilutė', 10.25, ,20080529', 1, 'Labai ilga eilutė ...');" cRddName, hConnection)`
	`RDDI_INFO(RDDI_DISCCONECT, hConnection, cRddName)`

2.1.2 Įrašų išrinkimas

Naudojantis duomenų bazėmis dažniausiai atliekamas veiksmas tai duomenų peržiūra, jų analizė, ataskaitų spausdinimas. Visiems išvardintiems veiksmams atlikti naudojama įrašų paieška pagal tam tikrą vieno ar kelių laukų filtrą. Visus veiksmus aprašant aibėmis atrodytų taip: $R = \{x^{l}_{\overline{1,r}} : x^{l}_{n} \in T_l, x^{l}_{nk} \in "Filtras"\}$, $R \subset T_l$, $n = \overline{1,p}$, $|r| \leq |p|$ čia R – rezultato aibė, p – rezultato aibės eilučių skaičius, kuris yra mažesnis arba lygus r, r – eilučių skaičius duomenų lentelėje T_l, k – stulpelio numeris lentelėje, pagal kurio požymį išrenkame tinkamus duomenis. Transformavimo pavyzdys pateikiamas 2-oje lentelėje. Komanda *DBEVAL()* pereina visus lentelės duomenis ir sukaupia tinkamas eilutes į aibę – kintamąjį *Rezultatai*. Komandos pirmasis parametras įterpia duomenis į aibę *Rezultatai*, antrasis parametras nurodo kada galima vykdyti pirmojo

parametro veiksmus, tai yra kurias eilutes galima įtraukti. SQL užklausos atveju kintamasis *Rezultatas* yra aibė, kurioje yra mūsų atrinktos eilutės. SQL serveriui nusiunčiam užklausą, o rezultato aibę grąžiną per kintamąjį *Rezultatai*.

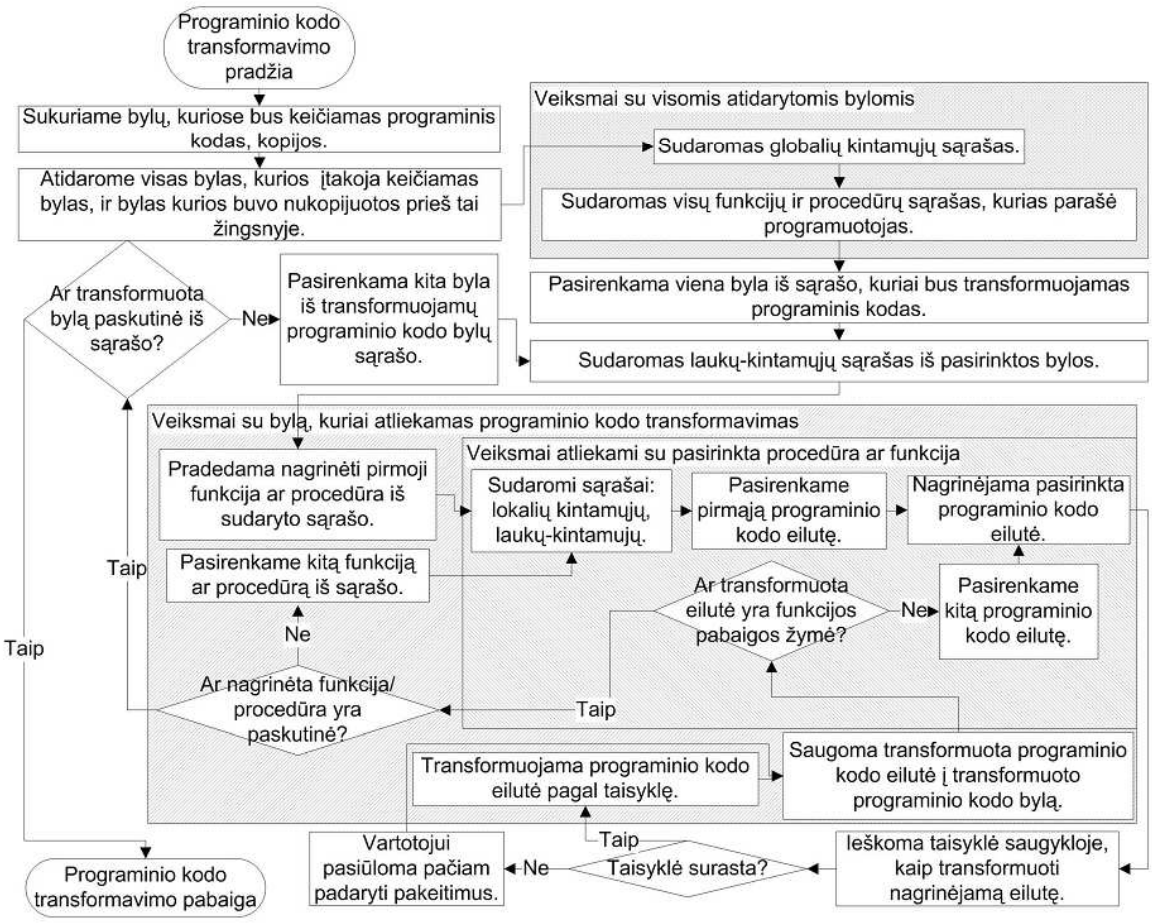
Lantelė 2. Įrašų išrinkimas.

xBase komandomis [4]	SQL užklausomis

`LOCAL Rezultatas := Array()`
`FIELD miestas, vastybe, id`
`DbUseArea(.T., "DBFCDX",`
`"C:\lentele.dbf",`
`"AliasLentele",.T.,.F.)`
`DBEVAL({|| AADD(Rezultatas, {id,`
`valstybe, miestas})},{|| miestas ==`
`"Filtras"})`
`DbCloseArea()`
	`LOCAL cRddName := "SQLBASE", hConnection`
	`RDDINFO(RDDI_CONNECT, {"MYSQL", "www.serveris.lt",`
	`"Vartotojas", "Slaptažodis", "duombazė"}, cRddName,`
	`@hConnection)`
	`DbUseArea(.T., cRddName, "Select id, valstybe,`
	`miestas From `lentele` Where miestas = `Filtras`",`
	`"Rezultatas", , hConnection)`
	`RDDI_INFO(RDDI_DISCCONECT, hConnection, cRddName)`

2.2 xBase komandų transformavimo į SQL konstrukcijas procesas

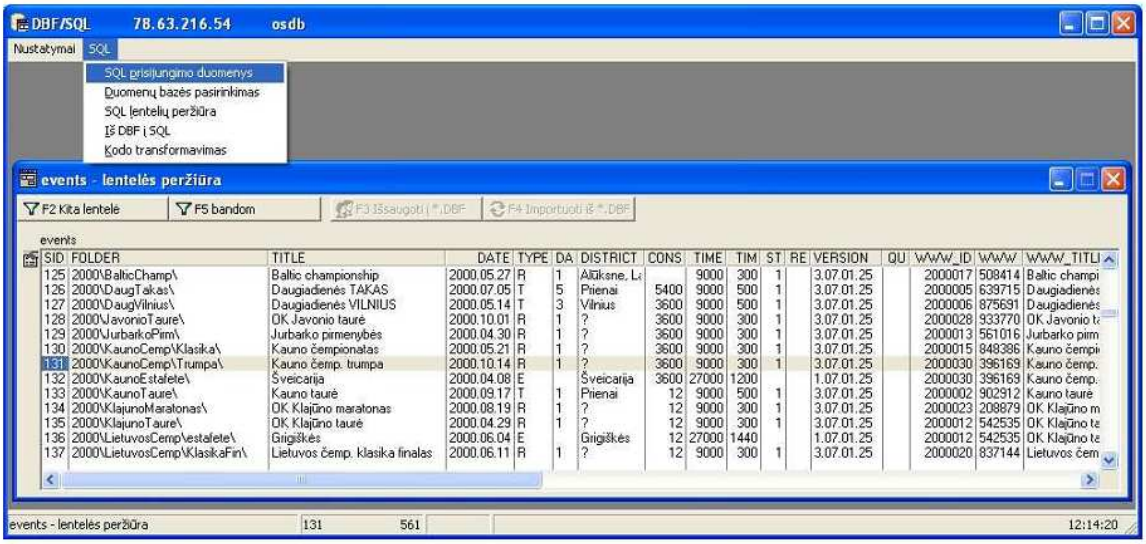
Automatizuotam transformavimui sudarytas algoritmas pateiktas 1 paveiksle, kuris realizuotas, kaip transformavimo įrankio prototipas. Pagrindinis programinio kodo transformavimo algoritmo tikslas teisingai atpažinti ir transformuoti komandas į SQL užklausas. Tai pasiekiama dirbant su visomis transformuojamo programinio kodo rinkmenomis, kurios priklauso transformuojamam projektui. Programinis kodas gali būti transformuojamas ne visose rinkmenose, tai yra pasirenkama. Darbas prasideda nuo visų rinkmenų globalių kintamųjų ir laukų pavadinimų surinkimo, po to pereinamos paeiliui transformuojamos rinkmenos ir peržiūrėjus lokalius kintamuosius pakeičiamas kodas atsižvelgiant į pateiktas taisykles. Transformavimo taisyklės yra aprašomos remiantis aibių teorija, pradiniu duomenų bazės ir lentelių apibrėžimu. Panašiai kaip pateikti pavyzdžiai ankstesniame skyrelyje.



Paveikslas 1. Programinės įrangos programinio kodo transformavimo algoritmas

3 xBase komandų transformavimo eksperimentinis įrankis

Programinio kodo transformavimui iš xBase komandų į SQL užklausas suprojektuota originali programinė įranga. Programinės įrangos pagalba galima senus projektus transformuoti ir panaudoti šiuolaikinėms problemoms spręsti. Programos pagalba yra surandamos eilutės su komandomis ir pasinaudojus taisyklių duomenų bazę yra transformuojamos į užklausas. Programos pagalba galime senus duomenis perkelti į naujas SQL duomenų bazes, kur toliau duomenys tarnauja naujai programai. Programa taip pat leidžia peržiūrėti duomenis iš duomenų bazės lentelių, juos koreguoti, įterpti naujus įrašus, ar išmesti nereikalingus. Programos išvaizda pateikta 2 paveiksle.



Paveikslas 2. Kodo transformavimo programinė įranga (projektas)

Programinei įrangai įgyvendinti naudojamas nemokamas kompiliatorius Borland C++ Compiler 5.5, nemokama atviro kodo programa Harbour 1.0.1 ir atviro kodo MySQL 5.1 duomenų bazių valdymo sistema. Pasinaudojant kuriama programine įranga buvo atliktas eksperimentas, kurio metu nustatyta, kad SQL užklausų panaudojimas paprastuose veiksmuose pagreitino programos veiksmų atlikimą tris kartus. Užklausų optimizavimas vykdymo laikui sutrumpinti būtų tolimesnis žingsnis tobulinant programinės įrangos projektą. Tikėtina, kad po užklausų optimizavimo programų veiksmų atlikimo laikas turėtų sutrumpėti.

4 Išvados

Atlikus xBase komandų analizę ir jų atitikmenų SQL užklausų tyrimą, nuspręsta, kad xBase komandų transformavimas į SQL konstrukcijas gali būti automatizuotas.

Atlikta xBase komandų analizė parodė, kad įmanoma sukurti kompiuterinę programinę įrangą, kuri palengvintų ir pagreitintų dBase komandų transformavimą į SQL užklausas.

Eksperimento metu pastebėta, kad atliktas kodo transformavimas palengvina atpažinti ir modifikuoti nekorektiškus rezultatus transformuotame kode.

Eksperimento metu nustatyta, kad transformuojamas kodas tiksliau transformuojamas jei naudojamos visos rinkmenos kurios įtakoja transformuojamą kodą.

Literatūra

[1] **Jim-Min L.** „Cross-platform software reuse by functional integration approach“. Computer Software and Applications Conference, 1997, COMPSAC '97, The Twenty-First Annual International, 402 – 408 p.

[2] **DuBois P.**, „MySQL developer‘s library“, ketvirtas leidimas, Addison-Wesley, USA, 2008.

[3] **Lopes, S., Petit, J.-M., Lakhal, L.**, „A framework for understanding existing databases“. Database Engineering & Applications, 2001 International Symposium on, 16-18 July 2001, 330 – 336 p.

[4] **Computer Associates International Inc.**, „CA-Clipper Programming and Utilities Guide“, June 1995, USA.