KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Rūta Vairaitė

Duomenų filtravimo ir atrankos sprendimų analizė

Magistro darbas

Darbo vadovas prof. R. Butleris

Konsultantas lekt. T.Danikauskas

Kaunas, 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Rūta Vairaitė

Duomenų filtravimo ir atrankos sprendimų analizė

Magistro darbas

Vadovas
prof. R. Butleris
2008-05-26

Recenzentas
doc. dr. Algimantas Venčkauskas
2008-05-26

Atliko
IFM – 2/1 gr. stud.
Rūta Vairaitė
2008-05-26

Kaunas, 2008

Turinys

Santrauka

Esant dideliems saugomų duomenų kiekiams, yra svarbus našus jų apdorojimas, taigi, vartotojams reikia vis didesnio duomenų bazių našumo. Šiame darbe sprendžiama problema, kaip paskatinti duomenų bazes veikti greičiau, kai duomenų bazių lentelės turi labai daug įrašų. Todėl skiriamas dėmesys duomenų bazių spartos derinimui, ar duomenų bazių spartos optimizavimui.

Išnagrinėjus duomenų bazių esamus spartinimo metodus ir priežastis, kurios mažina našumą, yra siūlomas metodas, kuris leidžia sparčiau apdoroti ir filtruoti duomenis bei greičiau pateikti vartotojui užklausos rezultatą.

Darbui atlikti pasirinkta MS SQL Server duomenų bazių valdymo sistema. Eksperimento metu atliktas užklausų greičio tyrimas, palyginant sudarytą metodą su virtualių lentelių metodu.

Summary

The analysis of data filtration and selection solutions

When the amount of stored data is growing, it is very important to get them fast and users are expecting to see how database performance is rising. Using database performance tuning, or database performance optimization, it is possible to make a database system run faster.

In this paper after analysis of database performance optimization and performance tuning methods was suggested a method which enables to process data from database more quick and to user to get query result faster.

To perform the research the MS SQL Server Database Management System was chosen. The experiment was performed in order to evaluate how method works. The experiment results show that compared with views, this method has better query performance.

1. Įvadas

Reliacinės duomenų bazių valdymo sistemos (RDBVS) praktiškai tapo standartu. Jos tokios pagrindinės stiprybės, kaip lengvas naudojimasis ir užklausų sudarymo pajėgumai, vertinamos labiau, nei jos techninės įrangos našumas ir sistemos išlaidos. Tačiau problema iškyla, kai nuolat augant organizacijose kaupiamų duomenų kiekiams ir jų apdirbimui, duomenų bazių našumas filtruojant ir atrenkant duomenis nėra didelis. Tuo tarpu vartojo reikalavimai ir lūkesčiai pastoviai auga, o užklausos atsakymo laikas įtakoja įvairias organizacijų operacijas. Duomenų bazių spartos derinimas ir duomenų bazių spartos optimizavimas yra vienos iš tų veiklų, kurios skatina duomenų bazes veikti greičiau. SQL optimizavimas siekia patobulinti užklausas taikymo lygmenyje ir turi didelį potencialą pagerinant duomenų bazių veikimą. Viena iš pirmųjų duomenų bazių (DB) derinimo užduočių yra suprasti problemų priežastis ir rasti esamas kliūtis. Yra nustatyta, kad SQL derinimas turi didžiausią įtaką našumui, kuri gali siekti daugiau negu 50%.

Remiantis duomenų bazių administratorių atlikta apklausa, kuriose jie nurodė, kad SQL derinimas turi didelę įtaką pagerinti DB našumą. Dalis nurodė, kad aptikti problemas susijusias su technine įranga kartais yra lengviau, nei kitas priežastis. Galiausiai, neteisingas arba vos optimalus indeksavimas ir netinkamas blokavimo procedūrų naudojimas, taip pat nurodyti kaip galimos priežastys, kurios įtakoja veikimo problemas[1].

Taigi, pirmoji faktorių kategorija, susijusi su nepakankamu DB našumu, gali būti sudaryta remiantis technine įranga (procesorius, atmintis, disko ir tinklo darbas). Antroji kategorija yra labiau susijusi su duomenų bazių sistemos charakteristikomis, (DB schemos kokybė, indeksavimas, suskirstymas ar blokavimas). Trečioji faktorių kategorija siejasi su problemomis, iškylančiomis taikomajame lygyje. Toliau SQL užklausų derinimo ir optimizavimo būdų analizės metu aptarsime antrosios ir trečiosios kategorijos faktorių įtaką.

Šio darbo tikslas yra, atsižvelgiant į įvairius siūlomus metodus, pasiūlyti ir pademonstruoti tokį metodą, kuris suformuotų užklausą pagal vartotojo poreikius, atrinkdamas ir filtruodamas jam reikiamus įrašus, bei parinktų geriausią variantą užklausai įvykdyti, tenkinant našumo sąlygas.

Nustačius kokie veiksniai labiausiai įtakoja užklausų našumą, yra patobulinamas užklausų formavimo metodas, kurio metu yra atrenkami duomenys pagal iš anksto apibrėžtas taisykles, panaudojami lentelių apjungimo šablonai. Eksperimento metu pasiūlytas metodas yra įvertinamas, stebint užklausų greitį, faktorius įtakojančius užklausos našumą bei sulyginama su analizės dalyje aptartu virtualių lentelių metodu. Eksperimentiniai duomenys yra iš Lietuvos Miškotvarkos Informacinės Sistemos duomenų bazės.

2. SQL užklausų derinimo ir optimizavimo būdų analizė

SQL yra deklaratyvi kalba, vartotojas tik turi nurodyti kokių duomenų jis nori. Duomenų bazių valdymo sistemos (DBVS) užklausų optimizatorius „nusprendžia“, kokiu informacijos išrinkimo keliu naudotis. Sprendimas gali būti labai sunkus, nes kartais įmanomi šimtai ar tūkstančiai skirtingų kelių sėkmingai įvykdyti užklausą. Užklausų generatorius parenka geriausią įvykdymo planą, kuris yra vadinamas užklausos vykdymo planu.

Yra labai svarbu parašyti užklausą taip, kad būtų grąžinti tik reikalaujami stulpeliai ir eilutės. DBVS užklausų optimizatorius turi galėti surasti geriausią užklausos vykdymo planą, kuris yra paremtas matematiniu modeliu su daugeliu prielaidų ir parametrais [1, 15].

Jeigu problema susijusi su blogai parašytu SQL kodu, tai reiktų atlikti tokius žingsnius:

- Stebėti ir aptikti našumo problemas.
- Identifikuoti kur problemos yra.
- Pasinaudoti informacijos išrinkimo analizavimo įrankiu.
- Pasistengti pagerinti užklausas išspręsti iškilusioms problemas.

2.1. Užklausų optimizatorius

Vienas iš labiausiai svarbiausių SQL serverio duomenų bazės komponentų yra užklausų optimizatorius. Jis reikalingas plano sudarymui, kaip SQL serveris turėtų įvykdyti užklausą. Į planą įeina visi žingsniai, kuriuos serveris turi įvykdyti, bei papildoma informacija apie bet kokį indeksą ar indeksus, kuriuos serveris turi naudoti išrinkdamas informaciją iš lentelės ar lentelių, kurios susijusios su parašyta užklausa. Taip pat tame plane yra informacija kaip surūšiuoti, sugrupuoti duomenis remiantis „GROUP BY“ sakiniu, ir kaip išrinkti duomenis iš sujungtų lentelių.

Užklausų optimizatorius įvertina skirtingus planus kaip pasiekti duomenis, remiantis kaina ir greičiu, kuriuo jis gali pateikti duomenis.

Skaičiavimai, kuriuos atlieka užklausų optimizatorius:

- Identifikuoja kokie indeksai yra ir ar jie sumažina užklausos įvykdymo laiką.
- Nustato indeksus ir stulpelius, kuriuos užklausų optimizatorius gali naudoti apribojant eilučių skaičių, kurį SQL serveris turi išanalizuoti, kad įvykdytų užklausą. Apriboja eilučių skaičių, kurį SQL serveris turi išanalizuoti, sumažina disko įvedimo/ išvedimo operacijų skaičių, kurį serveris turi įvykdyti, ir visa tai pagerina viso serverio spartą. Taip pat sudaro reikalingą stulpelių statistiką užklausos veikimui pagerinti.

- Parenka patį efektyviausią metodą sujungiant lenteles. Į šį skaičiavimą įeina ir lentelių apjungimo tvarkos pasirinkimas.

Tačiau užklausų optimizatorius turi limitą užklausų vykdymo planams, nes generuojant begalinį skaičių vykdymo planų, gali būti sulėtintas serverio veikimas.

SQL serverio darbai, kai gauna užklausą:

1. Tikrina užklausos sintaksę, tada suskaldo užklausą į smulkesnius komponentus, kuriuos duomenų bazės variklis galėtų vykdyti. Šis žingsnis vadinamas *nagrinėjimu*, ir galutinis rezultatas yra vadinamas *išnagrinėtos užklausos medžiu*.
2. SQL serveris verifikuoja objektų, kurie naudojami užklausoje vardus ir patikrina ar yra leidimas naudoti šiuos objektus. Taip pat patikrina perteklinius sintaksės sakinius ir juos pašalina, standartizuoja įdėtines užklausas. Šis žingsnis vadinamas *standartizavimo žingsniu*, ir jo rezultatas yra *standartizuotas užklausos medis*.
3. SQL serveris toliau analizuoja užklausą nuspręsdamas kuriuos indeksus, jei jų yra, jis turėtų naudoti, kad pagreitintų užklausą ir kaip jis sujungs lenteles, jei tai būtina. Šis žingsnis vadinamas *optimizavimo žingsniu*.
4. SQL serveris konvertuoja užklausą į vykdomą kalbą ir įtraukia identifikatorius, kurias lenteles ir indeksus naudos išrenkant duomenis. Šis žingsnis vadinamas *kompiliavimo žingsniu*.
5. Galiausiai, SQL serveris siunčia sukompiliuotą užklausą įvykdymui. Tai vadinama *įvykdymo žingsniu* užklausų apdorojime.

Kaip matome, yra tiek daug žingsnių, kuriuos SQL serveris turi padaryti vykdydamas užklausą, tai jis išsaugo užklausos sukompiliuotą vykdymo planą procedūrinėje atmintyje. Vykdymo planų saugojimas pagerina pasikartojančių užklausų įvykdymą [2, 21].

2.2. Indeksai

Didinant serverio spartą, duomenų bazių inžinieriui yra patariama atkreipti dėmesį į indeksavimą. Atsižvelgus į WHERE dėmenį SQL sakiniuose, teisingas indeksų parinkimas lentelei, kad užklausų optimizatorius parinktų geriausią strategiją, gali turėti žymius rezultatus.

Taigi, kodėl indeksai yra tokie svarbūs?

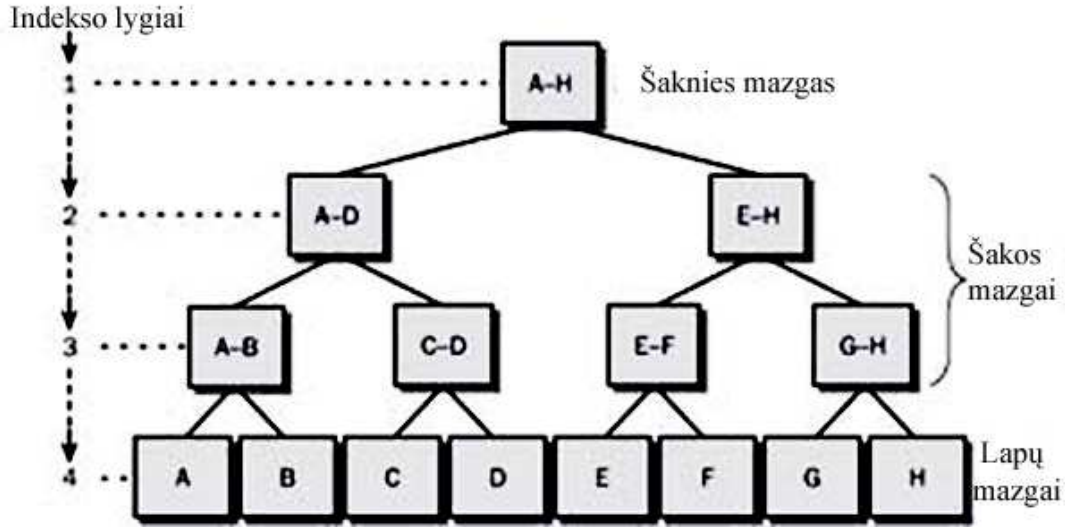
2.2.1. Indeksų principai

Visų pirma indeksas yra pagalbinė SQL serverio duomenų struktūra naudojama išrinkti informacijai. SQL serveryje indeksai yra naudojami kuriant lenteles ir virtualias lenteles. Viena lentelė ar virtuali lentelė gali turėti daugiau nei vieną indeksą. Priklausomai nuo indekso tipo, indeksuoti duomenys gali būti saugomi arba duomenų lentelėje arba atskirai nuo jos.

Be indeksų visi duomenys yra randami atliekant lentelės pilną peržiūrą, tai reiškia, kad visi duomenys lentelėje turi būti nuskaityti ir sulyginti su duomenimis, kurių prašoma. Kaip matome, geriau vengti lentelės pilnos peržiūros, nes ji generuoja daug disko įvedimo/ išvedimo operacijų, o didelės lentelės sunaudoja labai daug sistemos resursų. Tad naudojant tinkamą indeksą, galima žymiai sumažinti įvedimo/ išvedimo operacijų skaičių surandant reikalingų duomenų eilutes.

Taigi, indeksavimas gali paspartinti SELECT užklausas, tačiau INSERT, DELETE ir UPDATE užklausos bus lėtesnės indeksuotuose stulpeliuose.

Indeksai paprastai būna sudaryti kaip B-medžiai (1 pav.).



1 pav. Paprastas indeksas [3]

2.2.2. Indeksų tipai

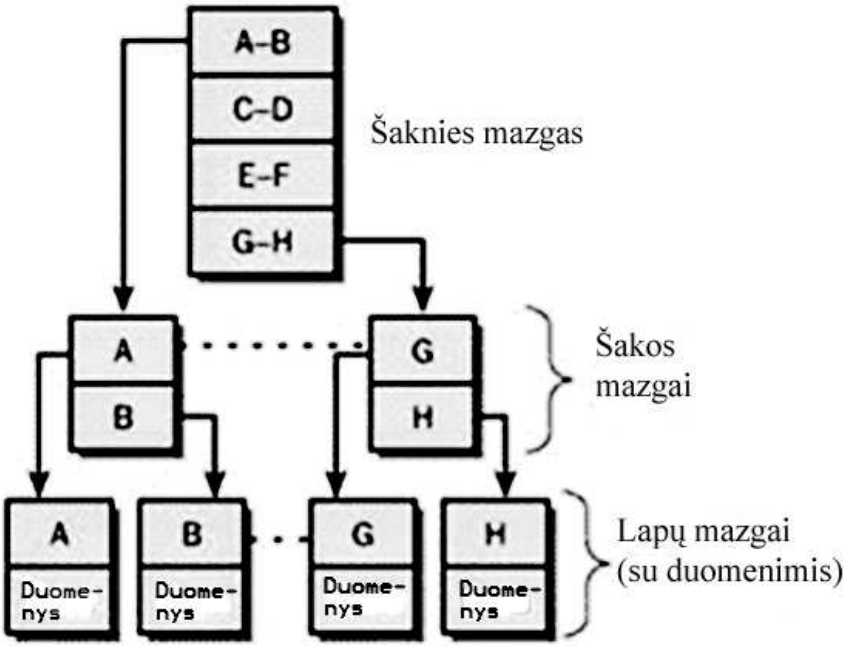
SQL serveris palaiko du pagrindinius indeksų tipus : grupuotus (clustered) (dar kartais vadinamais pirminiais indeksais) ir negrupuotus (nonclustered) (dar vadinamais antraeiliais indeksais).

2.2.2.1.Grupuotas indeksas

Grupuotas indeksas (2 pav.) žymi tvarką, kuria lentelės duomenys yra fiziškai saugomi. Lentelės duomenys yra sugrupuoti ir saugomi pagal stulpelio raktą ar stulpelį apibrėžtą grupuotam indeksui.

Grupuoti indeksai yra labai efektyvūs kada yra sukurti stulpeliams, kurie bus apieškomi dėl tam tikrų duomenų. Taip pat jie gali būti efektyvūs, kai dažnai reikia rūšiuoti išrinktus duomenis, arba kai dažnai tenka ieškoti įrašo, kuris turi specifinę reikšmę stulpeliui, kurį sudaro tik unikalios reikšmės.

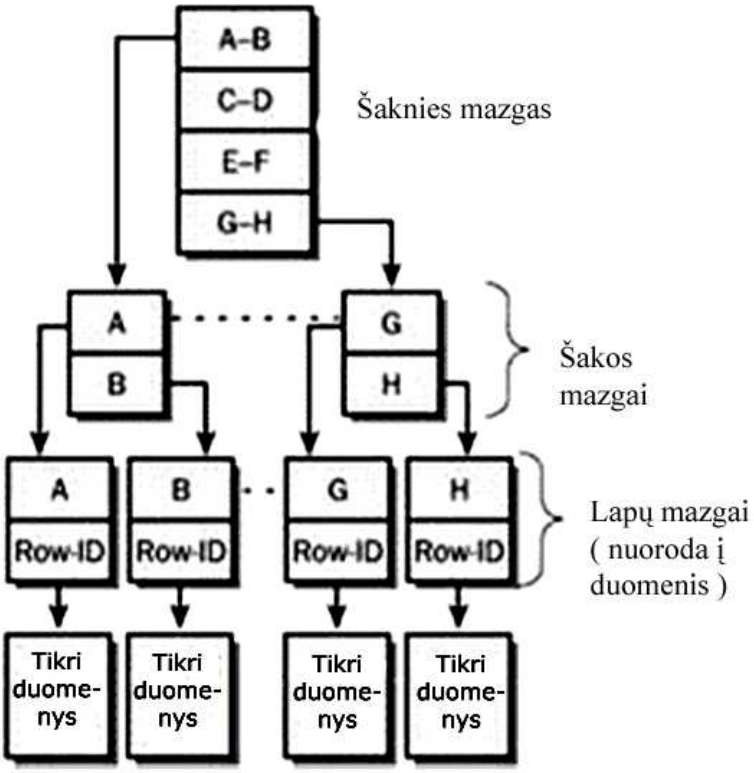
Vienai lentelei galima sudaryti tik vieną grupuotą indeksą, todėl reikia įvertinti tokias sąlygas kaip: kokios paieškos sąlygos bus dažniausiai naudojamos, kaip dažnai tai bus atliekama ir kiek duomenų įtraukta kiekvienoje užklausoje.



2 pav. Grupuoto indekso struktūra [3]

2.2.2.2. Negrupuotas indeksas

Kitaip nei grupuotas indeksas, negrupuotas indeksas neturi dabartinės lentelės duomenų savo lapų mazguose (3 pav.). Vietoj to, indeksas pats atskirtas nuo duomenų. Lapų mazgai sudaryti iš indekso eilutės, kuri saugo indekso duomenis ir informaciją, kuri apibrėžia tikslią eilutės vietą. Ji gali būti dviejų tipų, nustatoma pagal grupuotą indeksą, jei jis yra, tai tada kiekvienai eilutei grupuoto indekso rakto reikšmė yra saugoma negrupuoto indekso lapų mazguose. Jei grupuotas indeksas nėra unikalus, tai SQL serveris priskiria vidinę reikšmę kiekvienai identiškai rakto reikšmei, taip padarydamas juos unikalius naudojant negrupuotus indeksus. Jei nėra grupuoto indekso lentelei, tai tada kiekvienas lapo mazgas saugo eilutės ID (Row - ID) eilutei nustatyti, vietoje grupuoto indekso rakto reikšmės.



3 pav. Negrupuoto indekso struktūra [3]

2.2.3. Indeksų derinimas

Išsiaiškinus indeksų struktūrą, yra lengviau įgyvendinti indeksų realizavimą. Kaip minėta anksčiau - veiksmingi indeksai padeda rasti duomenis su keliais įvedimais/ išvedimais ir mažesniu sistemos resursų panaudojimu, nei atliekant pilną lentelės peržiūrą. Todėl, kad indeksai įtraukia B- medžio peržiūrą ieškant reikiamos reikšmės. Yra neefektyvu naudoti indeksą, kai reikia išrinkti didelį kiekį duomenų. Pagrindinė taisyklė užklausoms yra tokia, kad norint išrinkti daugiau nei 20% eilučių iš lentelės, geriau naudoti lentelės pilną peržiūrą nei naudoti indeksus.

SQL serveris nusprendžia naudoti ar nenaudoti indeksus, ir kokius indeksus naudoti iš esamų galimų (jeigu efektyviau įvykdyti užklausą nenaudojant indeksų, tai SQL serveris ir jų ir nenaudos). Taip pat SQL serveris gali sujungti 2 indeksus, kurie charakterizuoja lentelę, vykdydamas užklausą.

Yra keli pagrindiniai nurodymai kuriant indeksus:

- Reikia nustatyti ar dauguma užklausų yra SELECT sakiniai, ir koks INSERT, UPDATE, DELETE sakinių skaičius. Kai yra daug užklausų su INSERT, UPDATE, DELETE sakiniais, reikia atsargiai parinkti lentelėms kuriamų indeksų skaičių. Jeigu dauguma užklausų yra su SELECT sakiniais, tada daugiau indeksų gali leisti geresnį

veikimą, nes SQL serveris turi daugiau indeksų, iš kurių gali pasirinkti užklausų įvykdymo planams. Paprastai reikia rasti balansą tarp dviejų šių atvejų.

- Reikia sukurti indeksą taip, kad jis padėtų specifinėms užklausoms ne tik apibendrinti ar spėti kokius stulpelius naudoti kaip indeksų raktus. Sukūrus indeksą, geriau yra patikrinti įvykdymo planus užklausai ar užklausoms, kurioms jie buvo sukurti, patikrinti ar tikrai tas indeksas SQL serverio buvo panaudotas kaip numatyta.
- Indeksuojant mažas lenteles galima sulėtinti serverio veikimą, todėl gali būti greičiau ir efektyviau atlikti pilną lentelės peržiūrą.
- Reikia vengti indekse turėti labai daug indekso raktų, nes kuo daugiau duomenų indekse, tuo daugiau jų reikia atnaujinti, kai kokie nors pasikeitimai atsiranda lentelėje. Indeksas su vienu ar keliais raktų stulpeliais vadinamas siauru indeksu; indeksas su daug raktų stulpelių vadinamas plačiu indeksu. Siauras indeksas užima mažiau vietos kaip platus indeksas ir yra labiau tinkamesnis [3,16,19].

2.2.4. Indeksai ir užklausos

Indeksai taip pat turi būti apgalvotai naudojami visiems stulpeliams, kurie yra dažnai naudojami WHERE, ORDER BY, GROUP BY, TOP, ir DISTINCT sakinių. Be indekso kiekviena iš šių operacijų reikalauja pilnos lentelės peržiūros, o tai smarkiai gali įtakoti darbo greitį.

Nereikia automatiškai pridėti indeksus lentelėms, jei tik atrodo, kad tai geriausias būdas. Indeksus reikia pridėti tik tada, kai tikrai yra žinoma, kad jie bus panaudoti užklausose.

Taško užklausos, tai užklausos, kurios grąžina vieną eilutę, yra kaip tik greitesnės naudojant grupuotą indeksą, nei negrupuotą. Tačiau kuriant indeksą, kad pagreitinti vieno įrašo išrinkimą nėra naudinga, geriau naudoti negrupuotą indeksą, pataupant grupuotą sudėtingesnei užklausai.

Reikia vengti atsitiktinai antrą kartą uždėti tą patį indeksą tai pačiai lentelei. Tai yra labai lengvai įvykdoma, nes galima duoti indeksams skirtingus vardus ir SQL serveris leis kurti juos vėl ir vėl.

Geriau ištrinti indeksus, kurie niekada nenaudojami užklausų optimizatoriaus. Nenaudojami indeksai lėtina duomenų atnaujinimą, sukelia nereikalingus disko nuskaitymus ir švaisto duomenų bazių vietą, padidina laiką, kuomet dubliuojamos ir atkuriamos duomenų bazės. Galima pasinaudoti „Index Wizard“, kuris padėtų nustatyti, kurie indeksai nėra naudojami.

Dar vienas svarbus momentas, jeigu yra dvi ar daugiau lentelių, kurios yra dažnai sujungiamos, tada stulpeliai, kurie yra naudojami sujungimams turi turėti atitinkamą indeksą.

Jei stulpeliai naudojami sujungimams nėra kompaktiški, tai atsižvelgiant į tai pridedami pakaitiniai raktai lentelei, kurie yra kompaktiški, tam, kad sumažintų raktų skaičių, visa tai sumažina disko įvedimo/ išvedimo operacijas per sujungimo procesą, padidindami visą darbo greitį [4].

2.3. Virtuali lentelė

Virtualios lentelės yra naudingas įrankis, jos patogesnės nei funkcijos, lankstesnės nei saugomos procedūros, jos suteikia duomenų bazei daugiau saugumo, našumo ir paprastumo naudojantis. Todėl atkreipsime dėmesį į jų veikimą, naudojimą ir pasiūlymus, kaip jų dėka paspartinti užklausas.

„Virtualios lentelės turi tokią tendenciją, kad yra arba per daug naudojamos arba nepakankamai, ir labai retai būtent tiek, kiek reikia“ [13].

Virtuali lentelė naudojama:

- Sukurti vartotojui tam tikrą duomenų bazės sandaros įspūdį.
- Paslėpti nuo vartotojo nereikalingas lentelių eilutes ir stulpelius.
- Palengvinti vartotojo darbą, nes taip suprastėja duomenų struktūra.
- Kaupti duomenis našumui.
- Pridėti papildomą indeksavimą duomenų bazei, užklausos našumui padidinti.

Virtuali lentelė neturi savo duomenų, ji remiasi tik kitų realių lentelių duomenimis, realiai egzistuoja tik virtualios lentelės apibrėžimas, bet ne jos duomenys. Virtualių lentelių greitis tiesiogiai priklauso nuo užklausos greičio, todėl reikia stebėti pirminius ir išorinius raktus, per kuriuos yra jungiamos lentelės, ar jie turi indeksus, taip pat ir tuos laukus, kuriems taikomos filtravimo sąlygos.

Šalia virtualių lentelių pliusų iškyla ir minusai, kartais našumas tik sumažėja. DBVS užklausas virtualiai lentelei keičia paprastomis lentelėmis. Taigi, jeigu virtuali lentelė jungia daug lentelių, tai užklausa tampa sudėtingu lentelių sujungimu, ir tam reikia daug kompiuterio resursų. Todėl konkrečiu atveju reikia rinktis tarp patogumo vartotojui filtruojant duomenis ir našumo [13].

2.4. Indeksuota virtuali lentelė

Kitas būdas paspartinti duomenų atrinkimą ir filtravimą, yra indeksuotų virtualių lentelių naudojimas. Kadangi, be indeksų dabartinės SQL duomenų bazių sistemos taip pat kuria ir palaiko indeksuotas virtualias lenteles. Indeksuota virtuali lentelė yra realizuojama virtualiai lentelei sukuriant unikalų grupuotą indeksą. Tinkamai indeksuota virtuali lentelė

gali žymiai padidinti duomenų bazės našumą, tačiau teisingai realizuoti šį potencialų indeksuotų virtualių lentelių pasirinkimą yra sunku.

Konceptualiai, tiek indeksai, tiek indeksuotos virtualios lentelės yra fizinės struktūros, kurios žymiai pagreitina spartą. Jie yra panašūs, tik indeksuotos virtualios lentelės struktūros požiūriu yra labiau gausesnės, nei indeksai, nes jos gali nusakyti daugialypes lenteles, gali turėti išrinkimus ir GROUP BY daugialypiams stulpeliams.

Indeksuojant virtualias lenteles kyla galimybė padidinti disko įvedimo/išvedimo operacijų skaičių, bet taip pat galima išspręsti daugelį užklausos problemų.

Norint realizuoti indeksuotų virtualių lentelių pajėgumą, reikia rasti sprendimą šioms 3 svarstomoms problemoms:

- *Virtualių lentelių projektavimas:* nustatyti kurias virtualias lenteles reikia naudoti, kaip jas saugoti ir indeksuoti.
- *Virtualių lentelių palaikymas:* efektyvus realizuotų virtualių lentelių atnaujinimas, kai atnaujinama lentelė.
- *Virtualių lentelių eksploatavimas:* užklausų vykdymo pagreitinimui panaudoti realizuotų virtualių lentelių efektyvumą [5, 6].

Naudojant virtualias lenteles, kurios grąžina labai didelį rezultatų sąrašą, gali sulėtėti sparta, nes rezultatų sąrašas nėra indeksuotas ir visi rezultatai turi būti gaunami pilnai peržiūrint lentelę, jei virtualios lentelės naudojamos lentelių sujungimuose ar įdėtinėse užklausose. SQL serverio užklausų optimizatorius pabandys pasinaudoti indeksuota virtualia lentele, net jei ji nėra nurodyta SQL užklausoje.

Kada grupuotas indeksas yra sukuriamas indeksuotai virtualiai lentelei, tai SQL serveris iškarto rezervuoja vietą saugoti virtualios lentelės rezultatams.

Indeksuotai virtualiai lentelei yra sudaryti tokie reikalavimai:

1. Virtualios lentelės apibrėžimas visada turi grąžinti tuos pačius rezultatus iš tų pačių esančių duomenų.
2. Virtualios lentelės negali naudoti nedeterminuotąsias funkcijas.
3. Pirmas indeksas virtualiojoje lentelėje turi būti grupuotas, unikalus indeksas.
4. Jeigu naudojamas GROUP BY, tada virtuali lentelė negali turėti HAVING, CUBE ar ROLLUP.
5. Virtualios lentelės apibrėžimas negali turėti:
   a. TOP sakinio

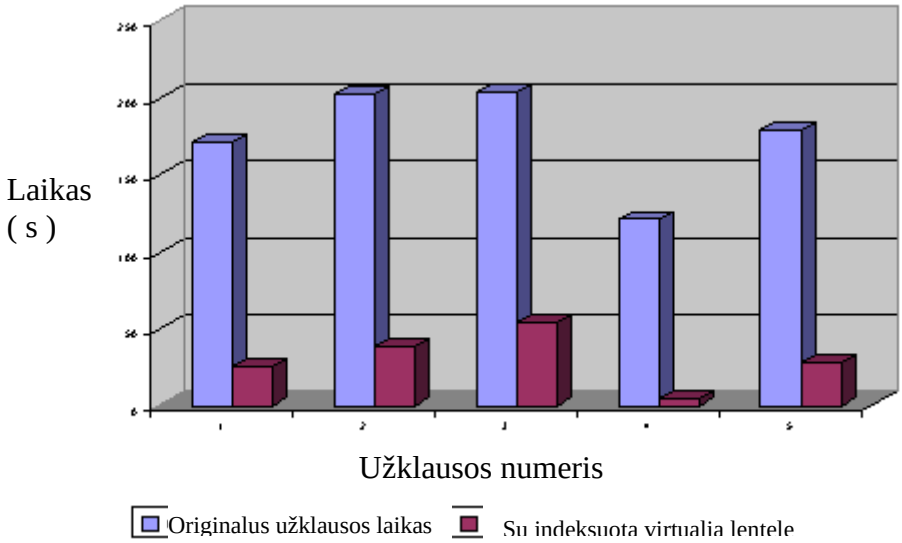
b. Teksto ar paveikslo stulpelių.
c. DISTINCT
d. MIN, MAX, COUNT, STDEV, VARIANCE, AVG sakinių.
e. SUM nulinėse išraiškose
f. Rowset funkcijos
g. Kitos virtualios lentelės
h. UNION sakinio
i. Įdėtinės užklausos, OUTER JOIN, SELF JOIN sakinių.
j. COMPUTE ar COMPUTE BY sakinių
k. Negali turėti ORDER BY virtualios lentelės aprašyme.

Microsoft raportavo, kad spartą galima padidinti nuo 10 iki 100 kartų naudojant kreipinius, kurie naudoja indeksuotas virtualias lenteles vietoj pagrindinių lentelių. Taip pat naudojant užklausas, kurios tiesiogiai nėra susietos su indeksuotomis virtualiomis lentelėmis, galima padidinti spartą, nes užklausų optimizatorius įtraukia indeksuotas virtualias lenteles tarp galimų įvykdymo planų [7].

Kaip matome, naudojant indeksuotas virtualias lenteles galima papildomai gauti naudos, ko negalima būtų pasiekti naudojant tik standartinius indeksus. Indeksuotos virtualios lentelės gali paspartinti užklausos darbą šiuose atvejuose:

- Agregatai gali būti iš anksto įvertinti ir saugomi indekse, kad sumažintų brangius skaičiavimus vykdant užklausą.
- Lentelės gali būti sujungtos ir rezultatų duomenys gali būti saugomi.
- Sujungimų ir agregavimo duomenys gali būti saugomi.

4 paveiksle galima pamatyti kaip našumas gali išaugti, kai užklausų optimizatorius naudoja indeksuotas virtualias lenteles. Kiekviena užklausa skiriasi sudėtingumu (pavyzdžiui, agregatų skaičiavimų kiekiu, naudojamų lentelių kiekiu, ar predikatų kiekiu) ir naudoja dideles daugiamilijonines įrašų lenteles.



4 pav. Užklausų vykdymo laikas [8]

Kitas variantas, kad virtualiose lentelėse naudojant negrupuotus indeksus galima taip pat papildomai padidinti spartą atrenkanti ir filtruojant duomenis. Panašiai kaip negrupuoti indeksai lentelėse, taip ir čia virtualiose lentelėse gali suteikti daugiau galimybių užklausų optimizatoriui pasirinkti, kol yra atliekamas kompiliavimo procesas. Pavyzdžiui, jei užklausa apima stulpelius, kurie neturi grupuoto indekso, tai optimizatorius gali pasirinkti vieną ar daugiau antrinių indeksų plane ir išvengti laiką ryjančios pilnos lentelės ar indeksuotos virtualios lentelės peržiūros.

Dažnai naudojami agregatai ir sujungimai yra geriausi kandidatai indeksuotoms virtualioms lentelėms. Ar dažnai ar retai naudojama užklausa, taip pat gali būti įgyvendinta su indeksuota virtualia lentele, jei tik prireikia didelės dalies laiko atsakyti į užklausą, ir yra svarbu gauti atsakymą labai greitai [8, 9].

2.5. Dalinai indeksuota virtuali lentelė

Dar vienas būdas, kurį išskirsime, norėdami paspartinti duomenų atrinkimą ir filtravimą, pasitelkiant virtualių lentelių principus yra dalinai indeksuotos virtualios lentelės. Tai yra naujas indeksuotų virtualių lentelių tipas, kuris kuriamas tik kai kuriems įrašams, pavyzdžiui, labiausiai naudojamiems įrašams; tai sumažina užimamą atminties vietą ir reikia mažiau resursų virtualių lentelių atnaujinimui. Viena ar kelios valdymo lentelės yra susiejamos su virtualia lentele, jose nurodoma, kurie virtualios lentelės įrašai yra saugomi ir eksploatuojami. Galima lengvai keisti, kurie virtualios lentelės įrašai yra saugomi ir eksploatuojami, paprasčiausiai modifikuojant duomenis valdymo lentelėje. Microsoft eksperimentai su MS SQL serveriu rodo, kad lyginant su pilnai indeksuotomis virtualiomis

lentelėmis, dalinai indeksuotos virtualios lentelės reikalauja mažiau vietos, jų geresnis veikimas, ir žymiai mažesni išlaikymo kaštai [11].

Neribojant atminties resursų, užklausos įvykdymas su dalinai indeksuotomis virtualiomis lentelėmis pagerėja, jei virtuali lentelė apima didesnę dalį užklausų, nes kelios užklausos greičiausiai gali naudoti labiau brangesnį atsarginį planą. Kuo daugiau apimama užklausų, tuo dalinai indeksuotų virtualių lentelių našumas panašėja į pilnai indeksuotas virtualias lenteles. Tačiau, kai yra riboti atminties resursai, tai didesnė dalis dalinai indeksuotų virtualių lentelių yra talpinama atmintyje, dėl to sumažėja disko įvedimo/išvedimo operacijų greitis. Net jei virtuali lentelė pilnai neapima visų užklausų, užklausos vykdymas naudojant dalinai indeksuotas virtualias lenteles gali būti veiksmingesnis, nei naudojant pilnai indeksuotas virtualias lenteles. Tačiau kartais duomenų bazėms netinka dalinai indeksuotos virtualios lentelės, tada tikslinga arba naudoti pilnai indeksuotas virtualias lenteles arba nenaudoti jų iš viso [10, 11].

2.6. Sujungimai

Lentelių sujungimai gali būti svarbus veiksnys sukeliantis SQL serverio spartos problemas, ypatingai, kai jungiama daugiau nei 2 lentelės arba lentelės yra labai didelės. Deja, sujungimai yra duomenų bazių svarbus elementas, todėl be jų apseiti neįmanoma. Kadangi jie yra labai paplitę, todėl jiems reikia skirti daug laiko, kad jie veiktų optimaliai, todėl svarbu atkreipti dėmesį į patarimus sujungimams.

Patarimai:

- jei yra kelios lentelės ar daugiau lentelių, kurios dažnai yra sujungiamos, tada stulpeliai, kurie panaudojami sujungimuose turėtų būti indeksuoti.
- vykdymo spartai padidinti, stulpeliai, naudojami sujungimuose, turėtų būti to paties tipo. Jei yra įmanoma, geriau jei jie skaitmeniai, o ne tekstinio tipo.
- Geriau vengti sujunginėti lenteles remiantis stulpeliais, kurie turi kelias unikalias reikšmes. Jei naudojami stulpeliai sujungimui nėra daugiausiai unikalūs, tada SQL serverio optimizatorius sujungimui atliks lentelės pilną peržiūrą, net jeigu indeksas stulpeliams egzistuoja. Optimizavimo tikslais, sujungimai turėtų būti atliekami su tais stulpeliais, kurie turi unikalų indeksą.
- Jeigu reguliariai reikia sujungti 4 ar daugiau lentelių, kad gautume įrašų sąrašą, tai reikia apgalvoti lentelių denormalizavimą, kad sujungiamų lentelių skaičius sumažėtų. Dažnai, pridedant vieną ar kelis stulpelius nuo vienos lentelės prie kitos, sujungimai gali būti sumažinti.

Lentelių sujungimai yra naudojami gauti rezultatus iš kelių ar daugiau lentelių. Dėl didesnės SQL serverio spartos, reikėtų riboti sujungiamų lentelių kiekį, nes kuo daugiau lentelių sujungiama, tuo ilgiau vykdoma užklausa, bandant rasti geriausią vykdymo planą. Lentelės yra jungiamos remiantis stulpeliu ar stulpeliais, kuriuos turi abi lentelės.

SQL serveris palaiko 3 tipų sujungimus: inner, outer ir cross.

Inner Naudojamas sujungti 2 lenteles, kurios turi bendras reikšmes viename ar daugiau stulpelių. Rezultatų sąrašas yra tik sutampantys įrašai. Pagal nutylėjimą yra naudojamas Inner sujungimas.

Outer Naudojamas kai reikia sujungti dvi lenteles, bet ir reikalingas rezultatas sudarytas ne tik iš įrašų, kurie tenkina sujungimo sąlygas, bet ir iš tų įrašų, kurie netenkina sąlygų.

Cross Naudojamas, kai norima sujungti vienos lentelės kiekvieną įrašą su kitos lentelės kiekvienu įrašu [2].

Užklausos, kurios sujungia dvi ar daugiau lentelių gali būti vykdomos įvairiais būdais. Kuo daugiau lentelių yra apjungiama, tuo daugiau sujungimo būdų SQL serveris gali pateikti. 1 lentelėje galime pamatyti, kaip priklauso sujungiamų lentelių skaičius ir galimų sujungimų variantų skaičius, galimų sujungimų variantų skaičius yra sujungiamų lentelių skaičiaus faktorialas.

1 lentelė. Lentelių skaičiaus ir sujungimo variantų ryšys [18]

Sujungiamų lentelių skaičius	Galimų sujungimų variantų skaičius
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
10	3 628 800
15	1 307 674 368 000
16	20 922 789 888 000 Net jeigu pavyktų per 1 sekundę apskaičiuoti milijoną skirtingų apjungimo variantų, tai surasti geriausią vykdymo planą užtruktų 242 dienas apjungiant 16 lentelių.

Yra rekomenduojama nesujunginėti daugiau nei 5 lentelių, net jei naudojama geriausia techninė įranga ir naujausia programinė įranga [18].

Kelios pagrindinės taisyklės mėginant pagerinti SQL kodo sujungimus:

- Naudoti didžiausius filtrus pirma: filtruoti didžiausią lentelę pirmiau, kad sumažintume įrašų sujungimą. Išrinkti lenteles tokia tvarka – nuo labiausiai išfiltruotos lentelės žemyn, geriau didžiausia lentelė būna labiausiai filtruojama. Yra svarbu sumažinti įrašų skaičių iš didžiausių lentelių, kiek įmanoma prieš sujungiant jas su kitomis lentelėmis.
- Naudoti indeksus: rašant kodą kiek įmanoma labiau naudoti indeksus, išskyrus mažas lenteles. Mažos lentelės dažnai nuskaitomos ignoruojant indeksus, kartais netgi dideles lenteles yra geriau nuskaityti neatsižvelgiant į indeksus, ypatingai, kai didesnė dalis lentelės yra nuskaitoma.
- Lizdiniai įdėtinių užklausų daliniai sujungimai (Nested subquery semi-joins): yra įmanoma pagerinti arba tiktai supaprastinti sujungimus, naudojant įdėtinės užklausos „lizdinius“ sluoksnius. Šis derinimo tipas yra dažniau naudojamas smarkiai normalizuotose duomenų bazių modeliuose, ir nenaudojamas denormalizuotose duomenų bazių modeliuose [12, 20].

Užklausų optimizavimo problema yra „NP-complete“ uždavinys. Taigi, neįmanoma surasti tokio optimizavimo algoritmo su polinominiu laiko sudėtingumu, kad pavyktų sugeneruoti optimalų įvykdymo planą kiekvienai užklausai. Dinaminio programavimo technika, kuri pritaikyta komercinėse DBVS, yra pati populiariausia technika ieškant optimalaus plano užklausai. Bet kaip bebūtų, blogiausia tai, kad laiko sudėtingumas yra eksponentinis $O(2^n)$. Dėl to, kai užklausa yra labai didelė (pavyzdžiui, reikia daugiau nei 50 sujungimų), dinaminio programavimo technika gali užimti mėnesius, netgi metus suoptimizuoti užklausą. Kada DBVS resursai išsenka, tai optimizatorius gali nutraukti dinaminį programavimą ir pasirinkti kokią nors godų išskaičiavimą. Tada įvykdymo planas yra dalinai optimalus [22].

2.7. Užklausų sakinių optimizavimo principai

Yra labai svarbu atkreipti dėmesį ir į tai, kaip rašomos užklausos. Tolesniuose poskyriuose aptarsime sakinių optimizavimo principus.

Viena iš tokių dėmesio reikalaujančių vietų, yra DISTINCT parametro naudojimas. Reikia atsargiai įvertinti ar SELECT užklausai reikia DISTINCT parametro ar ne. Jis turi būti naudojamas tik tada, kai yra žinoma, kad bus pasikartojančių įrašų, ir jie yra nepageidaujami. DISTINCT parametras suteikia daug darbo SQL serveriui, sumažina fizinius resursus, kuriais naudojasi kiti SQL sakiniai.

2.7.1. WHERE dalies formavimas

WHERE sakinyje įvairūs operatoriai tiesiogiai įtakoja, kiek greitai užklausa yra vykdoma. Todėl jei yra galimybė, geriau rinktis tuos, su kuriais duomenys išrenkami greičiau. Šie išvardinti operatoriai, naudojami WHERE sakinyje, surikiuoti pagal savo darbo greitumą:

- =
- >, >=, <, <=
- LIKE
- <>

Operatoriai, kurie yra viršuje, veikia greičiau, nei tie kur apačioje. Taigi geriau naudoti, jei įmanoma daugiau = nei <>.

Daugelis rašydami užklausas daro klaidą WHERE sakinyje kada lygina:

```
SELECT column_name FROM table_name
WHERE LOWER(column_name) = 'name'
```

Šiuo atveju yra elgiamasi, lyg SQL serveris būtų jautrus raidžių dydžiui. Jeigu SQL serveris nėra sukonfigūruotas būti jautrus, tada nereikia naudoti LOWER ar UPPER**.** Nereikalingas jų naudojimas tik sulėtins darbą.

Tačiau ką daryti, jeigu serveris sukonfigūruotas būti jautriu? Naudojimas LOWER ar UPPER vis tiek gali lėtinti darbą. Tada galima pabandyti naudoti tokį variantą, kartu su atitinkamais indeksais stulpeliuose naudojamuose užklausoje:

```
SELECT column_name FROM table_name
WHERE column_name = 'NAME' or column_name = 'name'
```

Šis variantas bus greitesnis, nei pirmasis variantas.

Jeigu reikia paspartinti užklausas su AND operatoriais WHERE sakiniuose reikia, kad:

- Bent vienas iš paieškos kriterijų WHERE sakiniuose, turi būti pagrįstas gerai išrenkamu stulpeliu, kuris turi indeksą.
- Jeigu, bent vienas iš paieškos kriterijų WHERE sakiniuose nėra gero išrenkamumo, atsižvelgti pridedant indeksus visiems stulpeliams paminėtiems WHERE sakiniuose.
- Jeigu nė vienas stulpelis WHERE sakinyje nėra pakankamai aukšto išrenkamumo, kad naudotume jiems indeksą, vadinasi reikia apgalvoti dengiamojo indekso sukūrimą šiai užklausai.

Užklausa su vienu ar daugiau OR operatorių kartais gali būti parašyta kaip serija užklausų, kurios yra sujungtos su UNION sakiniais, kad pagreitintų užklausos veikimą. Pavyzdžiui:

```
SELECT employeeID, firstname, lastname
FROM names
WHERE dept = 'prod' or city = 'Orlando' or division = 'food'
```

Ši užklausa turi 3 skirtingas sąlygas WHERE sakinyje. Norint naudoti indeksą, kiekvienas iš 3 stulpelių, kurie yra WHERE sakinyje, tūrėtų būti su indeksu. Ta pati užklausa gali būti perrašyta ir kitaip, naudojant UNION operatorių vietoj OR operatoriaus. Pavyzdžiui:

```
SELECT employeeID, firstname, lastname FROM names WHERE dept = 'prod'
UNION ALL
SELECT employeeID, firstname, lastname FROM names WHERE city = 'Orlando'
UNION ALL
SELECT employeeID, firstname, lastname FROM names WHERE division = 'food'
```

Kiekviena iš šių užklausų pateiks tokius pat rezultatus. Ir jeigu čia yra tik indeksas *dept* stulpeliui WHERE sakinyje, tai pirmoje versijoje nebus naudojamas joks indeksas, ir bus peržiūrima visa lentelė. Tačiau atlikus pakeitimą, užklausa naudos indeksą daliai užklausos, bet ne visai užklausai.

Tik testuojant galima atrasti tinkamiausią variantą, ar naudoti UNION ar OR operatorius [4].

Reiktų vengti tokių WHERE sakinių, kurie lygina stulpelį su konkrečia reikšme. Tokie paieškos elementai kaip IS NULL", "<>", "!=", "!>", "!<", "NOT", "NOT EXISTS", "NOT IN", "NOT LIKE", ir "LIKE '%500'" neleidžia dažniausiai užklausų optimizatoriui naudoti indekso, kad įvykdytų paiešką.

Tačiau kai kuriais atvejais yra įmanoma perrašyti tokius sakinius WHERE sakiniuose. Pavyzdžiui:

```
WHERE SUBSTRING(firstname,1,1) = 'm'
```

Gali būti perrašomas į tokį:

```
WHERE firstname like 'm%'
```

Naudojant WHERE sakinį, kuriame yra kelios išraiškos sujungtos AND operatoriais, SQL serveris jas vykdys iš kairės į dešinę kaip jos parašytos. Jokie skliaustai nėra naudojami, kad pakeistų vykdymo tvarką, taigi reiktų atkreipti dėmesį:

- Geriausia yra nustatyti mažiausiai tikėtiną AND išraišką. Šiuo atveju, jei AND išraiška yra neteisinga, sakinys bus nutrauktas, taip sutaupant laiko.
- Jeigu abi AND išraiškos pusės gali vienodai būti neteisingos, tai pirmiau reiktų rašyti mažiausiai sudėtingą išraišką. Nes jeigu ji neteisinga, bus mažiau darbo atlikto, kad ją įvykdyt.

Jeigu SELECT užklausa turi HAVING sakinį, tai užklausą reiktų parašyti taip, kad WHERE sakinys atliktų didesnę dalį darbo (atrinktų tik reikiamas eilutes), negu tai darytų HAVING. Taip pat naudojant WHERE sakinį galima eliminuoti nereikiamas eilutes prieš rūšiuojant su GROUP BY ir HAVING sakiniais, taip apsaugant nuo nereikalingo darbo ir pagreitinant užklausas.

2.7.2. Rūšiavimas

Kai tik SQL serveris turi įvykdyti rūšiavimo operaciją, tai papildomi resursai yra naudojami atlikti šiai užduočiai. Rūšiavimas naudojamas, kai yra šie operatoriai:

- ORDER BY
- GROUP BY
- SELECT DISTINCT
- UNION
- CREATE INDEX

Daugeliu atvejų šių komandų neįmanoma neišvengti, tačiau yra būdų kaip sumažinti jų daromą įtaką darbo greičiui. Tai galima padaryti:

- Rūšiuojamų eilučių skaičių sumažinti iki minimumo. Tai galima padaryti, tik grąžinant tik tas eilutes, kurios yra būtinos.
- Rūšiuojamų stulpelių skaičių sumažinti iki minimumo, nerūšiuoti daugiau stulpelių nei reikia.

- Stulpelių pločio (fizinį dydį) rūšiavimą sumažinti iki minimumo.
- Geriau rūšiuoti stulpelius su skaitiniais tipais, nei raidiniais.

GROUP BY sakinys gali būti pagreitintas tokiais būdais:

- Stengtis, kad užklausa grąžintų eilučių skaičių kiek galima mažesnį.
- Negrupuoti perteklinių stulpelių.
- Jeigu yra JOIN tame pačiame SELECT sakinyje, kuriame yra ir GROUP BY, tai geriau perrašyti užklausą naudojant įdėtines užklausas, vietoj JOIN. Jeigu tai įmanoma, tai darbo greitis padidės.
- Kartais verta apgalvoti naudoti ORDER BY sakinį SELECT sakinyje, kuris rūšiuoja tą patį stulpelį kaip ir GROUP BY. Tai gali leisti GROUP BY veikti greičiau [14].

2.7.3. Užuominų taikymas

Reikėtų vengti užklausose naudoti užuominas optimizatoriui, nes yra labai sunku pranokti patį užklausų optimizatorių. Optimizatoriaus užuominos yra specialūs raktažodžiai, kurie yra įrašomi užklausoje, nurodant ką optimizatorius turi daryti. Taigi, naudojant užuominas užklausų optimizatorius tampa statiniu, jis negali keisti plano pagal susidariusią situaciją. Tai dažniausiai tik kenkia, nei padeda pagretinti užklausas.

Tačiau jeigu, manoma, kad užuomina pagerins situaciją, tai pirmiausiai reikia padaryti:

- atnaujinti naudojamų lentelių statistiką.
- Jeigu problemiška užklausa yra saugomoje procedūroje, ją perkompiliuoti.
- Peržiūrėti ieškojimo argumentų ar jie yra tinkamiausi, jei ne, tai pasistengti juos perrašyti.
- Peržiūrėti indeksus, ir atlikti pakeitimus, jei reikia.

Ir tik tada kai viskas atlikta, ir užklausa vis tiek neveikia, kaip reikia, tada galima bandyti naudoti tam tikras užuominas optimizatoriui. Ir naudojant užuominas, periodiškai reikia peržiūrėti ar jos veikia taip, kaip yra tikimąsi [21].

2.7.4. Įdėtinių užklausų optimizavimas

Geriausias būdas yra visas įdėtines užklausas pakeisti sujungimais. Optimizatorius kartais pats gali suprastinti įdėtines užklausas ir pakeisti jas įprastais ar išoriniais sujungimais (outer joins). Bet ne visada jis tai padaro geriausiu būdu. Tikslūs sujungimai optimizatoriui suteikia daugiau galimybių pasirinkti lentelių tvarką ir rasti geriausią planą. Taigi, optimizuojant reikia įvertinti ar atsisakius įdėtinių užklausų atsiranda skirtumas [20].

2.7.5. Operatorių UNION ir UNION ALL taikymas

Jeigu tik įmanoma geriau naudoti UNION ALL operatorių nei UNION operatorių. Skirtumas yra tas, kad UNION turi „šalutinį efektą“ eliminuoja identiškas eilutes ir rūšiuoja rezultatus, ko UNION ALL visai nedaro. Kai renkami skirtingi rezultatai, reikia sukurti laikiną darbo lentelę, kurioje saugomi visi įrašai ir rūšiuojami prieš pateikiant išvestį. Taigi, jei nesitikima identiškų įrašų, UNION ALL daugeliu atveju yra geresnis ir efektyvesnis variantas. Dar viena problema su UNION operatoriumi, kad galima „užkimšti“ laikiną DB su didele darbo lentele, tai gali atsitikti, kai laukiama labai didelių rezultatų iš UNION užklausos [17].

2.8. SQL užklausų derinimo ir optimizavimo metodų išvados

Kad serveris greitai ir efektyviai dirbtų, reikia ne tik gerai suprojektuoti duomenų bazę, suprasti jos struktūrą, bet ir gerai optimizuoti SQL užklausas. Buvo išnagrinėtos priežastys, kurios pagreitina ir sulėtina serverio darbą:

1) Veiksmingi indeksai padeda rasti duomenis su keliais įvedimais/ išvedimais ir mažesniu sistemos resursų panaudojimu, nei atliekant pilną lentelės peržiūrą. Tačiau neefektyvu naudoti indeksą, kai reikia išrinkti didesnę dalį įrašų iš lentelės. O tuos indeksus, kurie niekada nenaudojami užklausų optimizatoriaus yra geriau ištrinti, nes jie lėtina duomenų atnaujinimą, sukelia nereikalingus disko nuskaitymus ir švaisto duomenų bazių vietą
2) Jeigu virtuali lentelė jungia daug lentelių, tai užklausa tampa sudėtingu lentelių sujungimu, ir tam reikia daug kompiuterio resursų ir sugaištama daug laiko, kol gaunami reikalingi įrašai.
3) Kartais užklausų spartą galima padidinti nuo 10 iki 100 kartų naudojant užklausas su indeksuotomis virtualiomis lentelėmis vietoj pagrindinių lentelių. Tačiau kartais jos grąžina labai didelį rezultatų sąrašą, taip gali sulėtėti sparta, nes rezultatų sąrašas būna neindeksuotas ir visi įrašai turi būti gaunami pilnai peržiūrint lentelę.
4) Dalinai indeksuotos virtualios lentelės yra patogios, nes galima lengvai keisti, kurie virtualios lentelės įrašai yra saugomi ir eksploatuojami, taip sutaupoma vieta, skirta duomenims saugoti ir užklausa yra įvykdoma greičiau.
5) Sujungiamų lentelių kiekis turėtų būti ribojamas, nes kuo daugiau lentelių sujungiama, tuo ilgiau vykdoma užklausa, bandant rasti geriausią vykdymo planą. Yra svarbu sumažinti įrašų skaičių iš didžiausių lentelių, kiek įmanoma

prieš sujungiant jas su kitomis lentelėmis, tad geriausia, kai didžiausia lentelė būna pirmiausiai filtruojama.

6) Užklausų sakinių rašymas taip pat turi įtakos užklausos vykdymui, todėl svarbu atkreipti dėmesį kada kokius parametrus naudoti ir kokia tvarka juos išdėstyti.

3. Užklausų formavimo metodas, panaudojant lentelių apjungimo šablonus

3.1. Taikymo sritis, sąlygos ir prielaidos

Šiame darbe sprendžiama problema susijusi su duomenų atrinkimo moduliu, kuriame duomenys atrinkami pagal iš anksto apibrėžtas taisykles, kurios yra aprašomos atskiroje duomenų bazėje. Taigi, problemą sudaro duomenų bazių, kurių lentelės turi didelius kiekius įrašų, naudojimas ir veikimas. Duomenų atrinkimo modulis tam tikrus sistemos objektus transformuoja į duomenų bazės virtualias lenteles, su kuriomis yra formuojamos užklausos ir atrenkami reikia duomenys. Tačiau apdorojant lenteles su dideliais kiekiais įrašų, duomenų apdorojimo sparta maža, kuo daugiau įrašų, tuo daugiau laiko sugaištama ir sunaudojama resursų.

Taigi, šiame skyriuje siūlomas metodas, kaip atrinkti reikiamus duomenis nenaudojant virtualių lentelių, o parenkant tinkamą lentelių apjungimo šabloną pagal atributus ir lenteles, bei kaupiamą statistiką. Nagrinėjami reikalavimai metodo įgyvendinimui, taip pat susiję duomenų struktūros ypatumai, pagrindiniai veiksniai, lemiantys duomenų apdorojimo spartą.

Darbui atlikti pasirinkta MS SQL Server duomenų bazių valdymo sistema.

3.2. Funkciniai reikalavimai siūlomam sprendimui

Reikalavimai užklausose naudojamų kriterijų įvedimui

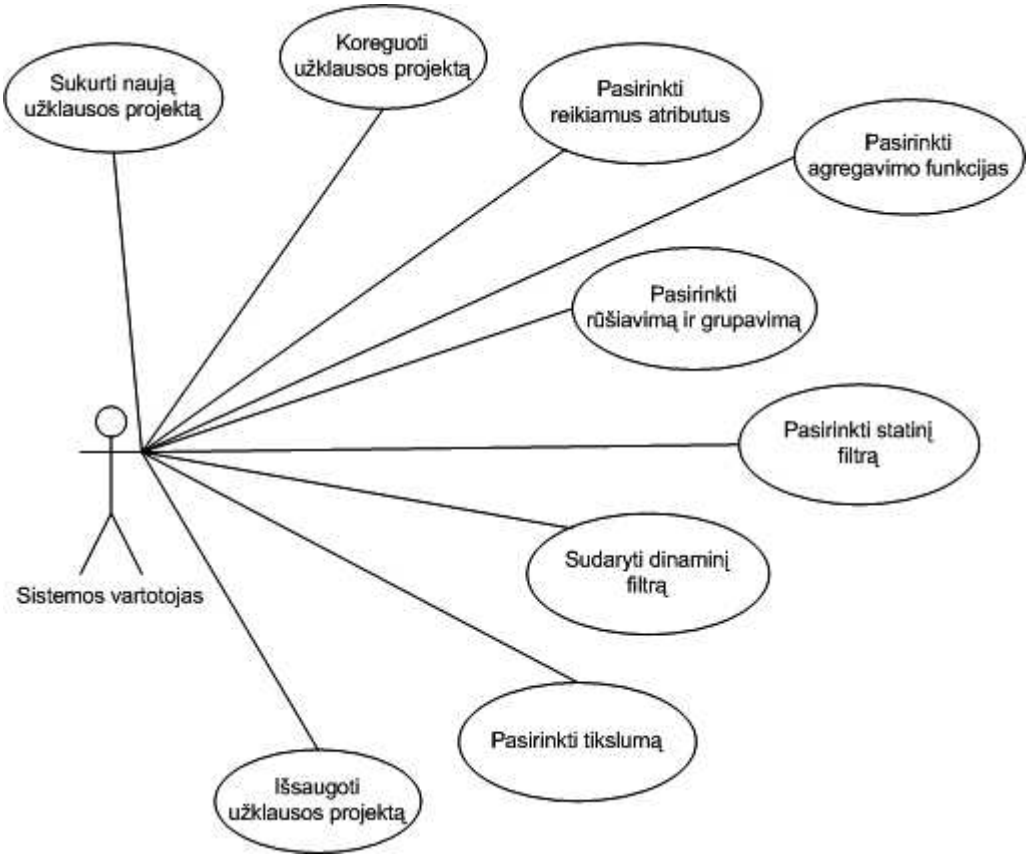
1) Sistemos vartotojas gali naudoti bet kokį skaičių atributų.
2) Sistemos vartotojas gali pasirinkti naudoti statinį filtrą.
3) Sistemos vartotojas gali pasirinkti sudaryti dinaminį filtrą.
4) Sistemos vartotojas gali pasirinkti jam reikiamas agregavimo funkcijas.
5) Sistemos vartotojas gali sugrupuoti ar surūšiuoti duomenis pagal jam reikiamą lauką.
6) Sistemos vartotojas gali pasirinkti laukų tikslumą.
7) Turi būti leidžiama koreguoti įvestus kriterijus.
8) Sistemos vartotojas gali išsaugoti savo sudarytą užklausos projektą.

Reikalavimai užklausos sudarymui, kriterijų panaudojimui

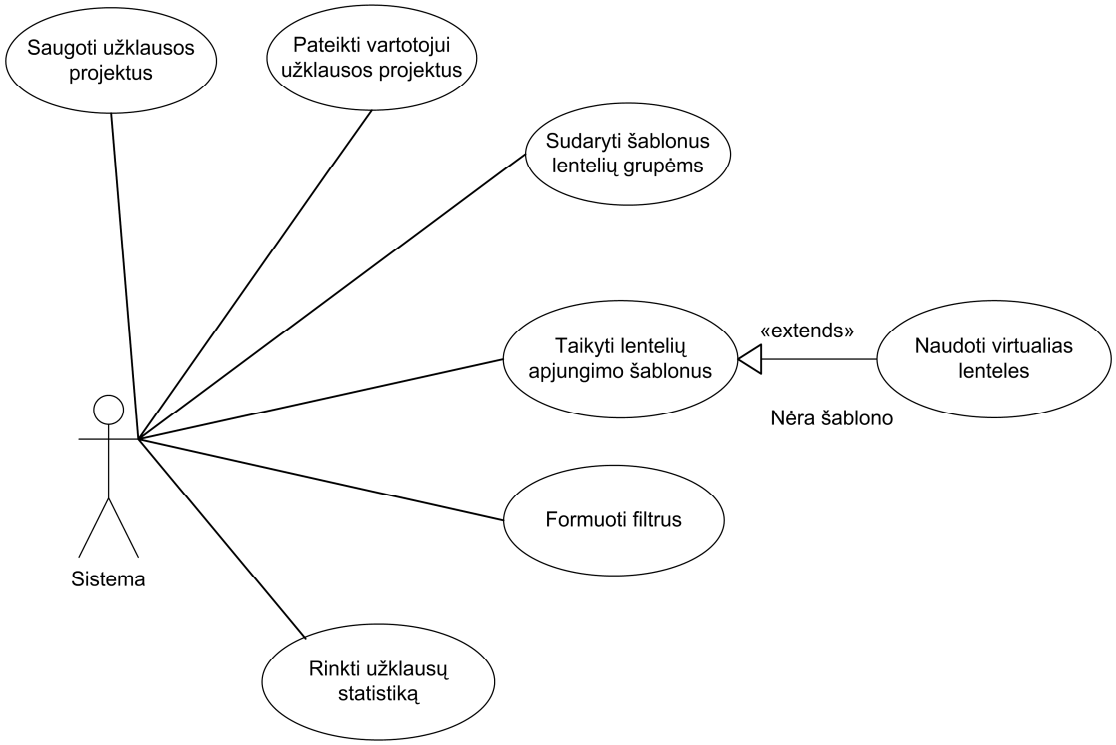
9) Duomenų atrankos modulis pagal pasirinktus kriterijus turi sudaryti užklausą.
10) Pagal pasirinktus atributus surasti labiausiai tinkantį lentelių apjungimo šabloną.
11) Neradus tinkamiausio šablono atributų išrinkimui naudoti virtualias lenteles.
12) Kaupti statistiką apie dažniausiai vartojamas atributų grupes ir lenteles.

13) Lentelių grupėms sukurti lentelių apjungimo šablonus.
14) Saugoti vartotojo sukurtus užklausos projektus.

Panaudojimo atvejų diagrama aprašo duomenų atrinkimo modulio funkcinius reikalavimus. 7 paveiksle pavaizduoti vartotojo poreikiai susiję su duomenų atrinkimo komponentu, o 8 paveiksle sistemos veiklos panaudojimo atvejai.



7 pav. Panaudos atvejų diagrama vartotojo poreikiams.



8 pav. Sistemos veiklos panaudojimo atvejai

3.3. Nefunkciniai reikalavimai siūlomam sprendimui

1) Efektyvumas - didelėje duomenų bazėje turinčioje daug lentelių su dideliais įrašų kiekiais užklausos turi būti vykdomos sparčiai.
2) Patikimumas – pagal vartotojo duotus kriterijus atrinkti reikiamus teisingus duomenis.
3) Tikslumas – tinkamai panaudoti agregavimo funkcijas, jos turi operuoti teisingais duomenimis ir pateikti tikslų rezultatą.
4) Patogumas – vartotojui turi būti patogu ir suprantama kaip pasirinkti kriterijus užklausai suformuoti.

3.4. Duomenų modelis

3.4.1. Įėjimai ir išėjimai

Kada vartotojas sudaro naują ar redaguoja esamą užklausos projektą:

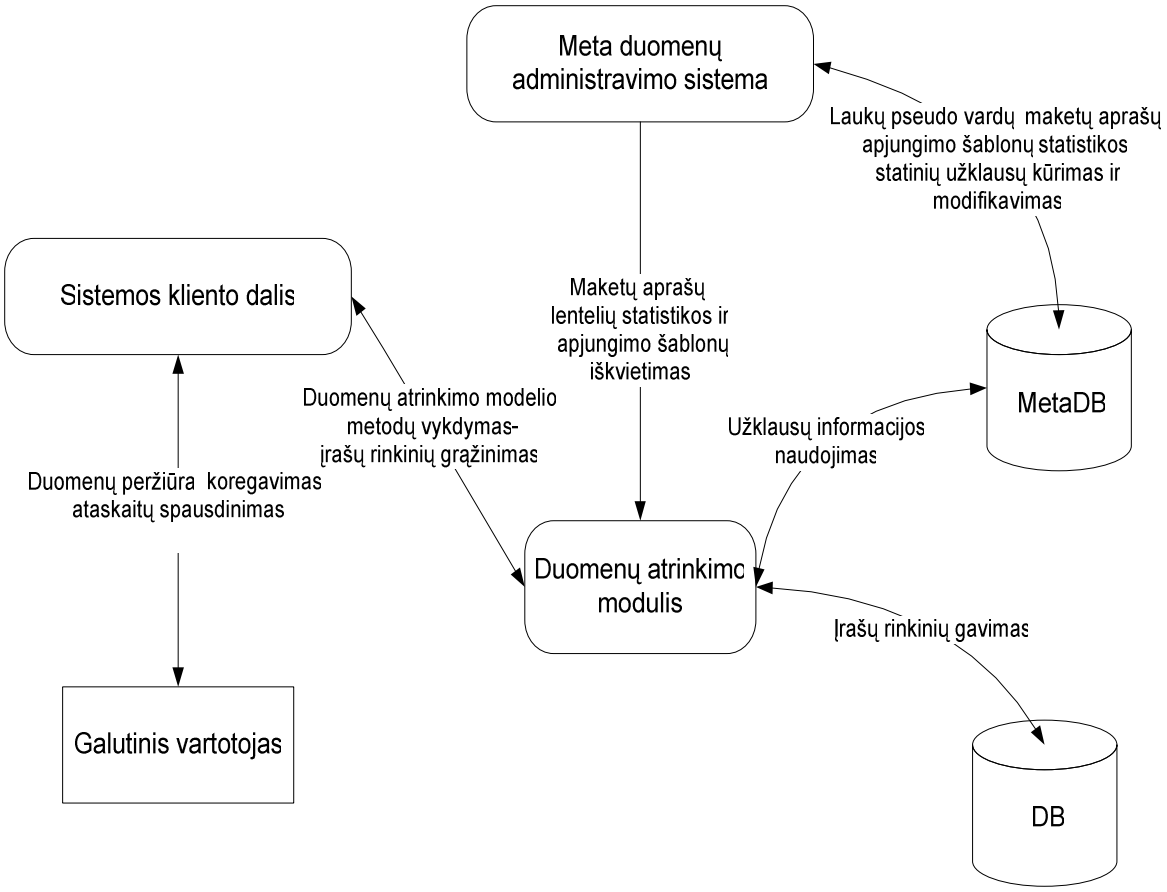
Įeinantys duomenys: užklausos vardas, komentaras, užklausą sudarantys laukai (atrenkami atributai), lentelių, maketų pavadinimai, agregavimo funkcijos, filtrų laukai, grupavimo, rūšiavimo požymiai.

Tada yra suformuojama SQL užklausa pagal šiuos duomenis, ir su ja išrenkami duomenys iš duomenų bazės.

Išeinantys duomenys: atrinkta informacija pagal sudarytą užklausą iš duomenų bazės.

3.4.2. Duomenų atrinkimo sistemos struktūrinė schema

Schema (9 pav.) vaizduoja ryšius tarp visų sistemos komponentų: sistemos duomenų šaltinius, duomenų aprašymą, duomenų pateikimą vartotojui, procesus.



9 pav. Struktūrinė schema

Su meta duomenų administravimo sistema galima administruoti meta duomenų bazę:

1. Atlikti pagrindinės ir meta duomenų bazių skirtumų analizę, išanalizuoti pakeitimus ir jei reikia, pasinaudojus analizės metu sukaupta informacija, atnaujinti meta duomenų bazę.
2. Sukurti naują sistemos objekto (maketo) aprašą, panaikinti nereikalingą maketą.
3. Redaguoti norimus maketų aprašus: priskirti ar panaikinti laukus, suteikti laukams pseudovardus.
4. Generuoti maketus atitinkančius lentelių apjungimo šablonus.

Su duomenų atrinkimo moduliu galima:

1. Atrinkti duomenis, kurių pageidauja vartotojas, pagal apibrėžtas taisykles saugomas Meta DB.
2. Sukurti naują užklausos projektą, kuris skirtas ataskaitoms reikalingų įrašų atrinkimui.
3. Užklausos projekte nurodyti jį sudarančius laukus, agreguojamas funkcijas, grupavimo ir rūšiavimo, apvalinimo atributus, laukų pseudovardus, atrenkamų įrašų unikalumo požymius.
4. Priskirti statinį filtrą, kuris iš anksto apriboja atrenkamų duomenų aibę, patikrinti filtro korektiškumą filtro transformavimo priemonėmis.
5. Kaupti užklausų statistiką: kokios lentelės ir kokie lentelių apjungimo šablonai kartu yra naudojami.

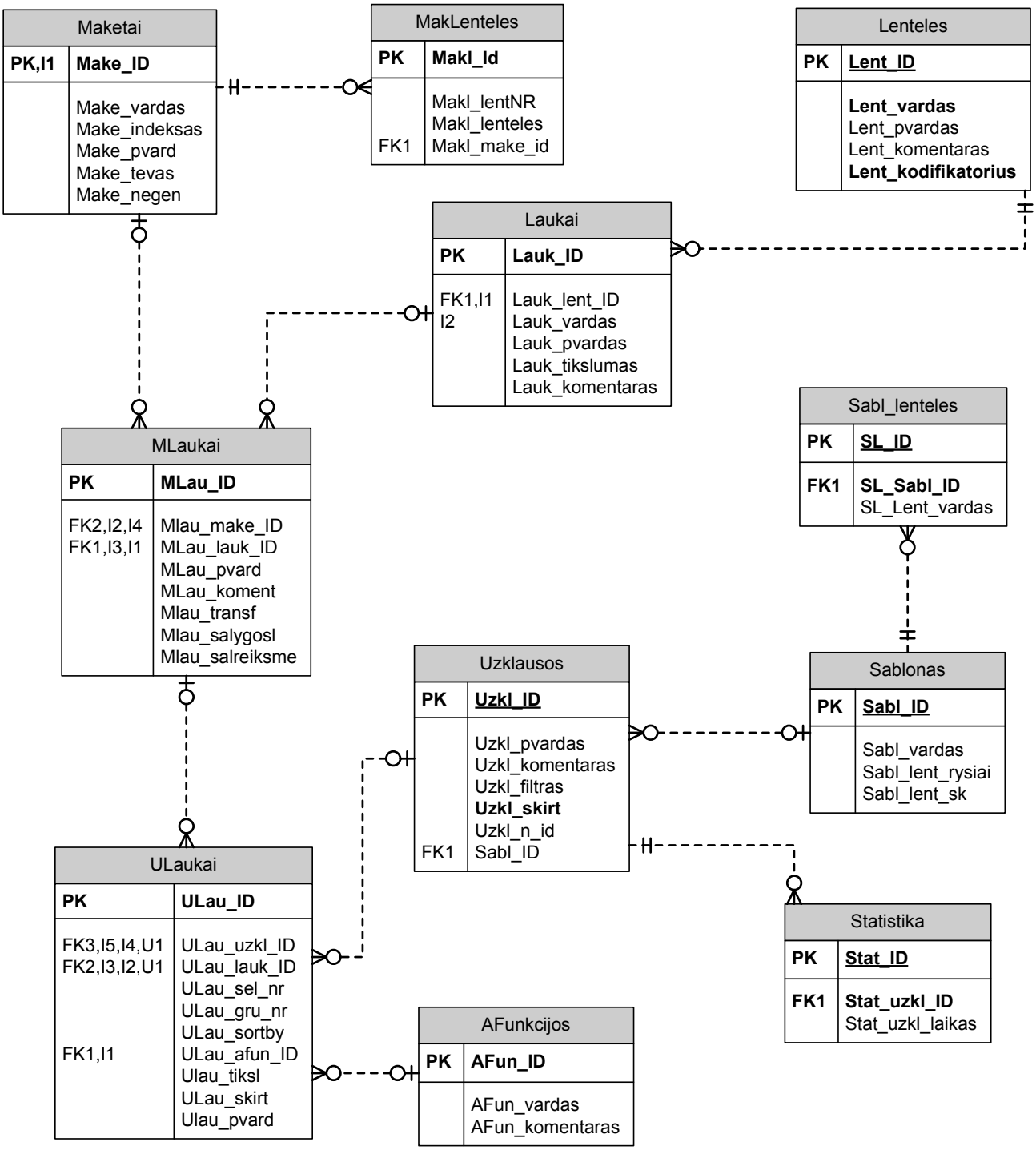
3.5. Meta DB struktūra

Metaduomenys yra svarbus kiekvienos sistemos komponentas. Metaduomenys (kur graikiškai meta reiškia „kitimas“) yra duomenys apie duomenis arba plačiau – tai yra informacija apibrėžianti duomenų turinį, kokybę, būseną ir kitas charakteristikas, o šiuo atveju tai duomenys, aprašantys pačią DB.

Ši meta duomenų bazė (10 pav.) skirta saugoti:

1. Pagrindinės duomenų bazės lentelių ir laukų informaciją;
2. Sistemos objektų (maketų) aprašus;
3. Statinių užklausų specifikacijas;
4. Papildomą informaciją, susijusią su statinių užklausų specifikacijų kūrimu.
5. Lentelių apjungimo šablonų informaciją.
6. Labiausiai vykdomų užklausų ir lentelių statistikos informaciją.

Meta duomenų bazę sudarančių lentelių sąrašas yra pateikiamas 2 lentelėje, o šių lentelių struktūra aprašyta 3-13 lentelėse.



10 pav. Meta duomenų bazės schema

2lentelė. Meta duomenų bazės lentelių sąrašas

Lentelės vardas	Paskirtis	Struktūra
LENTELES	Pagrindinės DB lentelių sąrašas	3 lentelė
LAUKAI	DB lentelių laukų sąrašas	4 lentelė
MAKETAI	Pagrindinė informacija apie maketus	5 lentelė
MAKLENTELES	Maketus sudarančių lentelių ryšių specifikacija	6 lentelė
MLAUKAI	Maketus sudarančių laukų sąrašas	7 lentelė

Lentelės vardas	Paskirtis	Struktūra
UZKLAUSOS	Pagrindinė informacija apie statines užklausas	8 lentelė
ULAUKAI	Statines užklausas sudarančių laukų sąrašas	9 lentelė
SABLONAS	Lentelių apjungimo ryšių specifikacija	10 lentelė
STATISTIKA	Užklausų naudojimo informacija	11 lentelė
AFUNKCIJOS	Agregavimo funkcijų sąrašas	12 lentelė
SABL_LENTELES	Šabloną sudarančių lentelių sąrašas	13 lentelė

3 lentelė. DB lentelės **LENTELES** struktūra

Lauko vardas	Paskirtis
Lent_ID	Lentelių sąrašo pirminis raktas, automatiškai numeruojamas
Lent_vardas	Lentelės vardas
Lent_pvardas	Lentelės vardas programose
Lent_komentaras	Lentelės paskirties paaiškinimas
Lent_kodifikatorius	Parodo, ar atitinkama lentelė yra normatyvinė ar paprasta (apskaitos duomenų).

4 lentelė. DB lentelės **LAUKAI** struktūra

Lauko vardas	Paskirtis
Lauk_ID	Lentelės pirminis raktas, numeruojamas automatiškai.
Lauk_lent_ID	Konkrečios lentelės, kuriai priklauso laukas, unikalus identifikatorius.
Lauk_pvardas	Lauko vardas programose
Lauk_komentaras	Lauko paskirties paaiškinimas
Lauk_tikslumas	Lauko tikslumas skaičiais po kablelio.

5 lentelė. DB lentelės **MAKETAI** struktūra

Lauko vardas	Paskirtis
Make_ID	Maketų unikalus identifikatorius.
Make_vardas	Maketo vardas.
Make_indeksas	Maketo indeksas, skirtas maketų eiliškumui išsaugoti, jei maketai trinami, įterpiami nauji.
Make_pvard	Maketo vardas filtruose (m#0, m#10 ir t.t.)
Make_tevas	Parodo, ar maketas yra aukščiausio lygio maketas-antraštė. Jei 0- tai antraštė, jei 1- tai paprastas maketas.
Make_negen	Jei 0- tai maketo vaizdai bus generuojami, jei 1- negeneruojami. Skirtas išimtiniams atvejams, kai maketo struktūros neįmanoma aprašyti sukurtomis priemonėmis ir jį tenka kurti standartiniu būdu.

6 lentelė. DB lentelės **MAKLENTELES** struktūra

Lauko vardas	Paskirtis
Makl_ID	Maketus sudarančių lentelių ryšių unikalus identifikatorius.
Makl_lentNR	Maketus sudarančių lentelių eilės numeris.
Makl_lenteles	Maketą sudarančių lentelių ryšiai. Tai tiesiog užklausos dalis, kurioje nurodomi ryšiai tarp lentelių.
Makl_make_ID	Konkretaus maketo identifikatorius.

7 lentelė. DB lentelės **MLAUKAI** struktūra

Lauko vardas	**Paskirtis**
Mlau_ID	Maketą sudarančių laukų unikalus identifikatorius
Mlau_make_ID	Konkretaus maketo, kuriam priklauso laukas, identifikatorius
Mlau_lauk_ID	Konkretaus lentelės lauko, kuris įtraukiamas į maketą, identifikatorius.
Mlau_pvard	Maketo lauko pseudovardas.
Mlau_koment	Maketo lauko paskirtis
Mlau_transf	Parodo ar laukas yra transformuotas iš paprasto įrašo pagal nurodytą sąlygą.
Mlau_salygosl	Sąlygos laukas, naudojamas aukščiau paminėtam transformacijos vykdymui.
Mlau_salreiksme	Sąlygos reikšmė, naudojama transformuojant įrašus į laukus.

8 lentelė. DB lentelės **UZKLAUSOS** struktūra

Lauko vardas	**Paskirtis**
Uzkl_ID	Unikalus statinės užklausos identifikatorius (pirminis raktas).
Uzkl_pvardas	Statinės užklausos programinis vardas
Uzkl_komentaras	Statinės užklausos paskirtis.
Uzkl_filtras	Statinei užklausai priskirtas statinis filtras.
Uzkl_skirt	Šis parametras parodo, ar užklausa turi atrinkti skirtingas, ar visas užklausos rezultato eilutes. Jei 0- tai visas eilutes, jei 1- tik skirtingas eilutes.
Uzkl_n_id	Statinės užklausos autoriaus ID.

9 lentelė. DB lentelės **ULAUKAI** struktūra

Lauko vardas	**Paskirtis**
Ulau_ID	Unikalus statinės užklausos identifikatorius
Ulau_lauk_ID	Statinės užklausos lauko identifikatorius surišimui su lentelių laukai.
Ulau_sel_nr	Lauko eilės numeris SELECT dalyje
Ulau_gru_nr	Jei pagal konkretų lauką grupuojama, tai Ulau_gru_nr=1.
Ulau_sort_nr	Paliktas atvejui, jei reikės skirtingos rūšiavimo tvarkos nei SELECT dalies laukų sąraše.
Ulau_sortby	Rūšiavimo požymis. Jei NULL tai nerūšiuojama, jei “ASC”- didėjimo tvarka, jei “DESC”- tai mažėjimo tvarka.
Ulau_afun_ID	Agreguojamos funkcijos identifikatorius (jei laukui taikoma agreguojama funkcija).
Ulau_tiksl	Parodo, ar bus panaudojamas tikslumas skaičiais po kablelio, jei jis buvo nurodytas laukui laukų lentelėje. Jei 0- tikslumas nebus naudojamas, jei 1- tikslumas bus naudojamas, bet tik esant tikslumui laukų lentelėje.
Ulau_skirt	Šis laukas skirtas nurodyti ar agreguojama funkcija turi atrinkti tik skirtingas ar visas reikšmes savo skaičiavimams. Jei 0- tai visos reikšmės, jei 1- tai skirtingos reikšmės. Naudojama tada, kai reikia skaičiuoti objektus, objektų hierarchijoje aukštesnius už sklypą.
Ulau_pvard	Užklausos lauko pseudovardas, kuris būtų panaudojamas sugeneruotame vaizde. Jei jis tuščias, tai vardas vaizde išlieka toks pats kaip ir lauko pseudovardas maketo apraše. Jei laukui taikoma agreguojama funkcija, būtina nurodyti pseudovardą.

10 lentelė. DB lentelės **SABLONAS** struktūra

Lauko vardas	**Paskirtis**
Sabl_ID	Šablonų unikalus identifikatorius.
Sabl_vardas	Šablono vardas
Sabl_lenteles	Šabloną sudarančių lentelių ryšiai. Tai tiesiog užklausos dalis, kurioje nurodomi ryšiai tarp lentelių.
Sabl_lent_sk	Šabloną sudarančių lentelių kiekis.

11 lentelė. DB lentelės **STATISTIKA** struktūra

Lauko vardas	**Paskirtis**
Stat_ID	Statistikos unikalus identifikatorius.
Stat_uzkl_laikas	Laikas, per kurį įvykdoma užklausa.

12 lentelė. DB lentelės **AFUNKCIJOS** struktūra

Lauko vardas	**Paskirtis**
Afun_ID	Agreguojamos funkcijos unikalus identifikatorius
Afun_vardas	Agreguojamos funkcijos vardas (AVG, SUM ir t.t)
Afun_komentaras	Funkcijos paskirtis.

13 lentelė. DB lentelės **SABL_LENTELES** struktūra

Lauko vardas	**Paskirtis**
SL_ID	Šabloną sudarančios lentelės unikalus identifikatorius
SL_Sabl_ID	Konkretaus šablono identifikatorius.
SL_Lent_vardas	Šabloną sudarančios lentelės vardas

3.6. Lentelių apjungimo šablonų formavimas

Lentelių apjungimo šablonai formuojami lentelių grupėms, kurios yra dažniausiai naudojamos užklausose. Vartotojai dažnai naudoja tas pačias užklausas nekeisdami kriterijų, todėl vieną kartą apibrėžus lentelių apjungimo šabloną galima jį daugelį kartų naudoti, arba pritaikyti kitai užklausai. Kai vykdant užklausas yra naudojami maketai ir jiems sudarytos virtualios lentelės, taip atrenkant tam tikrus atributus gali būti panaudojamos nereikalingos lentelės, ir gali tekti filtruoti didesnį kiekį duomenų, taip išauga laiko kiekis, panaudojamas užklausai įvykdyti. 2.3 ir 2.4 poskyriuose buvo išanalizuota, kad virtualių lentelių ir indeksuotų virtualių lentelių silpnosios vietos pasireiškia, kai reikia apjungti kelias virtualias lenteles grąžinamas labai didelis rezultatų sąrašas, kuris nėra indeksuotas ir visi rezultatai gaunami pilnai peržiūrint lentelę, ir dar, kai tenka virtualias lenteles jungti per laukus, kurie neturi indeksų, šiais abiem atvejais žymiai sumažėja užklausos sparta, labiau naudojami sistemos resursai. Todėl užklausai parinkus tikslų arba su minimaliai perteklinių lentelių kiekiu apjungimo šabloną, sutaupoma laiko, kuris skiriamas perteklinių lentelių sujungimui.

3.6.1. Šablono sudarymas

Su Meta duomenų administravimo sistema yra sudaromas lentelių apjungimo šablonas, kuris yra panaudojamas atrenkant ir filtruojant duomenis.

1) Kai vartotojas užklausos projekte nurodo kriterijus užklausos sudarymui, tai jie yra surenkami ir išanalizuojami: kokie atributai turi būti atrinkti, kokioms lentelėms jie priklauso ir koks tų lentelių ryšys su pagrindine lentele ***Geo_objektai***.

2) Kai vartotojas nurodo filtrą, taip pat reikia atkreipti dėmesį į filtruojančius laukus, kokioms lentelėm jie priklauso ir koks jų ryšys su pagrindine lentele ***Geo_objektai***.

3) Atlikus pirmus du veiksmus gaunamas sąrašas tų lentelių, kurios naudojamos užklausoje, ir tų lentelių ryšiai su pagrindine ***Geo_objektai*** lentele. Tarp jų nebūtinai turi būti tiesioginis ryšys, lentelės gali sietis ir per tarpines lenteles.

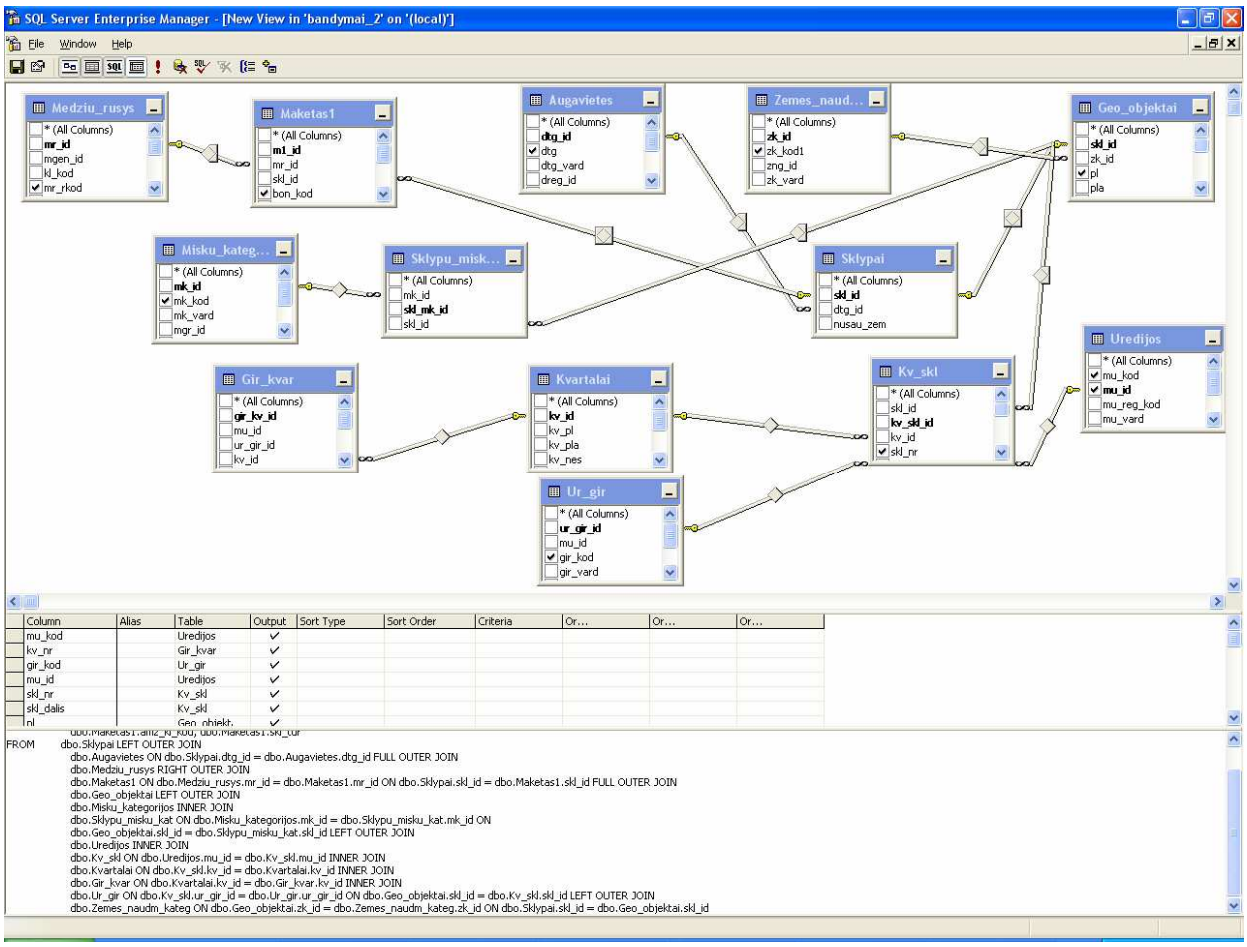
Kadangi aprašant ryšių kelius tarp lentelių gali pasitaikyti alternatyvūs keliai, tai geriau pasirinkti tokį kelią:

- Kurio ryšys tarp lentelių stipresnis;
- Kurio ryšys užmezgamas tik tarp privalomų lentelių atributų;
- Kuris saugomos informacijos požiūriu niekada nenutrūksta.

4) Aprašius visas užklausos projekte naudojamas lenteles ir ryšius galima atlikti lentelių apjungimą šiai išskirtai grupei.

Patogiausia tai atlikti, pavyzdžiui, su MS SQL Enterprise Manager paketu. Pasileidus paketą, pasirenkama duomenų bazė, joje pasirenkamas skirsnis *Views*. Pasirenkama kurti naują virtualią lentelę – *New view*, tada kontekstiniame meniu pasirinkamas punktas įtraukti lentelę – *Add Table.* Atsidariusiame dialogo pirmajame puslapyje pasirenkamos visos į šabloną norimos įtraukti lentelės ir tarpinės lentelės, kurios reikalingos ryšio užmezgimui tarp posistemės grupės lentelių ir ***Geo_objektai*** lentelės. Įkėlus visas reikiamas lenteles patikrinama, ar visas grupės lenteles galima susieti su lentele ***Geo_objektai***. Jei reikia į šabloną įtraukiamos papildomos lentelės. Toliau nustatoma įrašų atrinkimo “kryptis”. Krypties atrinkimui taip pat patogu naudoti grafinį Enterprise Manager redaktorių. Reikia laikytis pagrindinės taisyklės nustatinėjant atrinkimo kryptį :

- Visiems įrašams atrinkimo kryptis nurodoma nuo ***Geo_objektai*** lentelės link nagrinėjamų posistemės grupės lentelių. Jei visas lenteles esančias “kelyje” nuo ***Geo_objektai*** lentelės link nagrinėjamos lentelės būtų išdėstytos į horizontalią eilę iš dešinės į kairę (11 pav.) ir įrašų atrinkimo kryptis bus laikoma iš dešinės į kairę, tai visi įrašai turi būti atrenkami iš lentelės esančios dešinėje įrašu atrinkimo krypčiai ir tik tiek kurių reikia iš lentelės esančios kairėje pusėje.



11pav. Įrašų atrinkimo krypties nustatymas

5) Gautas pilnas naudojamų lentelių sąrašas yra išsaugomas Meta duomenų bazėje ***Sabl_lenteles*** lentelėje, kiekvienai lentelei priskiriant šablono identifikatorių, bei nurodant lentelės pavadinimą. Taip pat ***Sablonas*** lentelėje **Sabl_lent_rysiai** lauke yra išsaugoma užklausos FROM dalis, kuri ir galės būti panaudojama užklausose, kai reikės atrinkti atributus, kurie priklauso šablone esančioms lentelėms.

Lentelių apjungimo šablonas – FROM dalis:

```
dbo.Krumu_rusys INNER JOIN
dbo.Vyr_krumu_rusys ON dbo.Krumu_rusys.kr_id = dbo.Vyr_krumu_rusys.mr_id
RIGHT OUTER JOIN
dbo.Maketas19 ON dbo.Vyr_krumu_rusys.m19_id = dbo.Maketas19.m19_id RIGHT
OUTER JOIN
dbo.Sklypai ON dbo.Maketas19.skl_id = dbo.Sklypai.skl_id LEFT OUTER JOIN
dbo.Trako_tankumo_tipai ON dbo.Maketas19.tan_id =
dbo.Trako_tankumo_tipai.tan_id
```

3.7. Užklausų formavimas

Sistemos vartotojas gali arba sukurti naują savo užklausos projektą, arba panaudoti esamą užklausos projektą, jį redaguoti pagal savo poreikius, atrenkant reikiamus duomenis

ataskaitai. Jeigu kuriama nauja užklausa, tai turi būti įvedamas unikalus užklausos vardas, bei komentaras aprašantis užklausos paskirtį.

Užklausos projekto sudarymo ir vykdymo eiga susideda iš šių 4 punktų**:**

1) Užklausos atributų sąrašo formavimas – skirtas užklausos laukams nurodyti.
2) Užklausos konkrečių laukų formavimas – skirtas užklausos laukų rūšiavimo, grupavimo, eilės tvarkai nustatyti, agreguojamoms funkcijoms priskirti.
3) Lentelių apjungimo šablono paieška – skirta surasti geriausią lentelių apjungimo šabloną.
4) Užklausos statistikos kaupimas – skirtas kaupti statistiką apie užklausos įvykdymą.

3.7.1. Užklausos atributų sąrašo formavimas

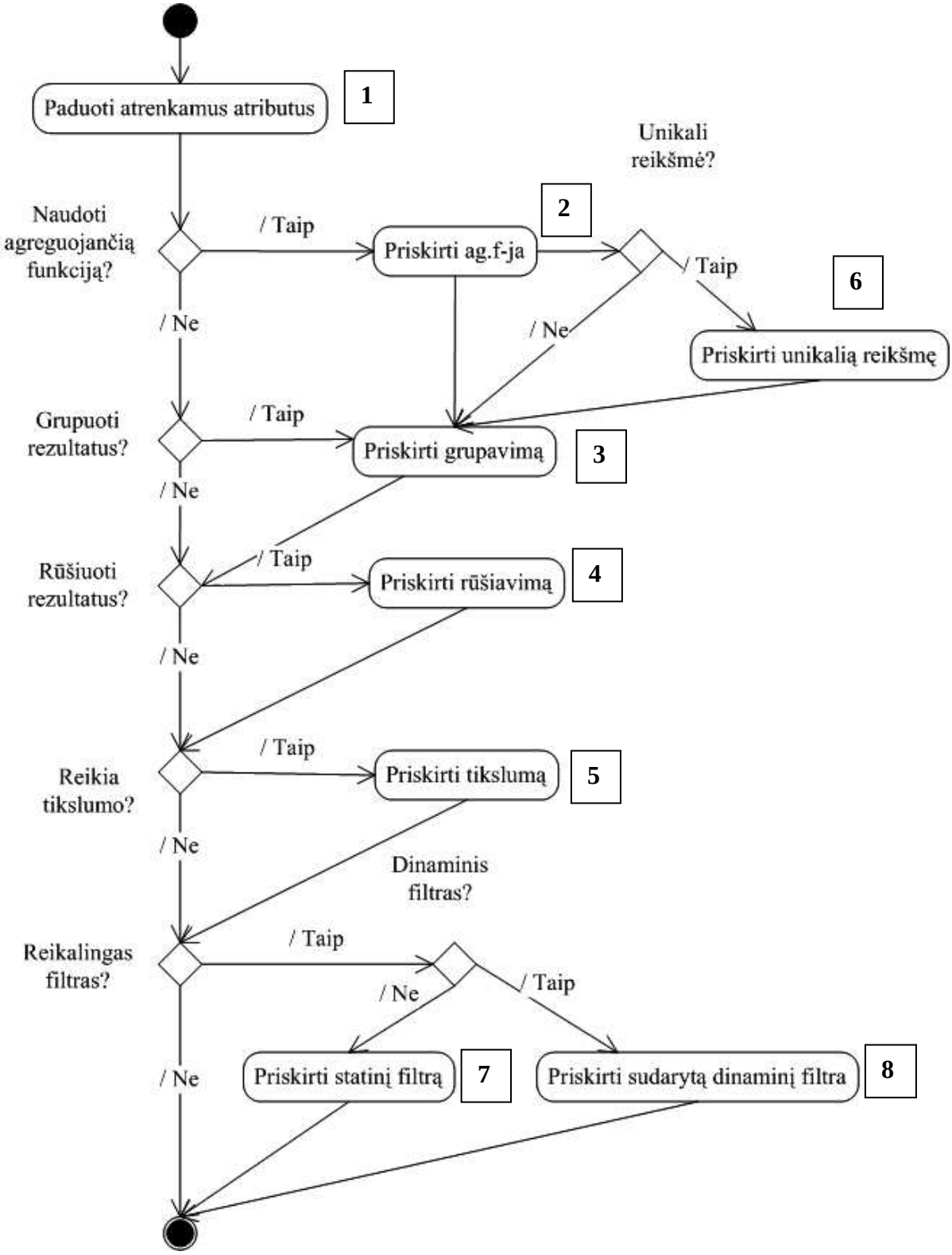
Formuojant užklausą, pirmiausia reikia apibrėžti ją sudarančius laukus. Vartotojas pasirinkęs maketą, iš maketą sudarančio laukų sąrašo gali pasirinkti jam reikalingus laukus. Maketo laukų pavadinimai, kuriuos mato vartotojas, skiriasi nuo lentelių laukų pavadinimų, todėl sudarant užklausos atributų sąrašą reikia maketų laukų pavadinimus (pseudovardus) pakeisti į lentelių laukų pavadinimus. Turint maketo pavadinimą, Meta duomenų bazėje (aprašyta 3.5. poskyryje) esančioje lentelėje ***Maketai,*** yra surandamas maketo unikalus identifikatorius **Make_ID**, su kuriuo lentelėje ***MLaukai*** yra surandamas ***Laukai*** lentelės unikalus identifikatorius. Jis nurodo lentelės lauko pavadinimą. Tokiu būdu galima sudaryti užklausą sudarančių atributų sąrašą.

3.7.2. Užklausos konkrečių laukų formavimas

Veiksmai, kuriuos galima atlikti su konkrečiais užklausos laukais (12 paveikslas):

1) Jau sudarytą užklausos atributų sąrašą paduoti formavimui.
2) Jei reikia, galima priskirti pasirinktą agreguojamą funkciją konkrečiam užklausos laukui iš pateikto funkcijų sąrašo. Priskyrus agreguojamą funkciją, automatiškai visiems laukams, kurie neturi agreguojamų funkcijų uždedamas grupavimo požymis. Priskyrus laukui agreguojamą funkciją reikia įvesti laukui pseudovardą.
3) Jei reikia, galima grupuoti užklausos rezultatus pagal pasirinktą lauką.
4) Jei reikia, galima rūšiuoti užklausos rezultatus pagal pasirinktą lauką.
5) Jei reikia, galima pasirinkti konkrečiam laukui tikslumą. Jei laukui buvo priskirta kokia nors agreguojama funkcija, tai tikslumas taikomas laukui, o ne agreguojamos funkcijos rezultatui.
6) Jei reikia galima nurodyti, kad rezultatai butų skaičiuojami tik su unikaliomis laukų reikšmėmis, kai laukas turi agreguojamą funkciją.
7) Jei reikia, galima užklausai priskirti statinį filtrą.

8) Jei reikia, galima užklausai sudaryti dinaminį filtrą pagal filtrų sudarymo taisykles.



12 pav. Užklausos vykdymas

3.7.3. Lentelių apjungimo šablono paieška

Pagal užklausą sudarančius laukus, statinį ir dinaminį filtrą yra sudaromas naudojamų užklausoje lentelių sąrašas. Pagal jį Meta duomenų bazėje yra ieškomas lentelių apjungimo

šablonas (13 paveikslas), kuris arba visai atitiktų, arba savyje turėtų minimaliai perteklinių kitų užklausoje nenaudojamų lentelių.

Šablonas Meta duomenų bazėje yra ieškomas pagal tokį užklausos formatą, kuris sudaromas remiantis 2.7 poskyryje aprašytais užklausų optimizavimo principais:

```
SELECT     TOP 10 COUNT(*) AS Lent_kiekis, Sabl_lenteles.SSabl_ID,
Sablonas.Sabl_lent_sk
FROM         Sabl_lenteles INNER JOIN
             Sablonas ON Sabl_lenteles.SSabl_ID = Sablonas.Sabl_ID
WHERE       (Sabl_lenteles.SL_Lent_vardas LIKE 'Lenteles_pavad1') OR
            (Sabl_lenteles.SL_Lent_vardas LIKE 'Lenteles_pavad2') OR
                                          …
            (Sabl_lenteles.SL_Lent_vardas LIKE 'Lenteles_pavadN')
GROUP BY Sabl_lenteles.SSabl_ID, Sablonas.Sabl_lent_sk
HAVING      (COUNT(*) = Kiekis)
ORDER BY Sablonas.Sabl_lent_sk
```

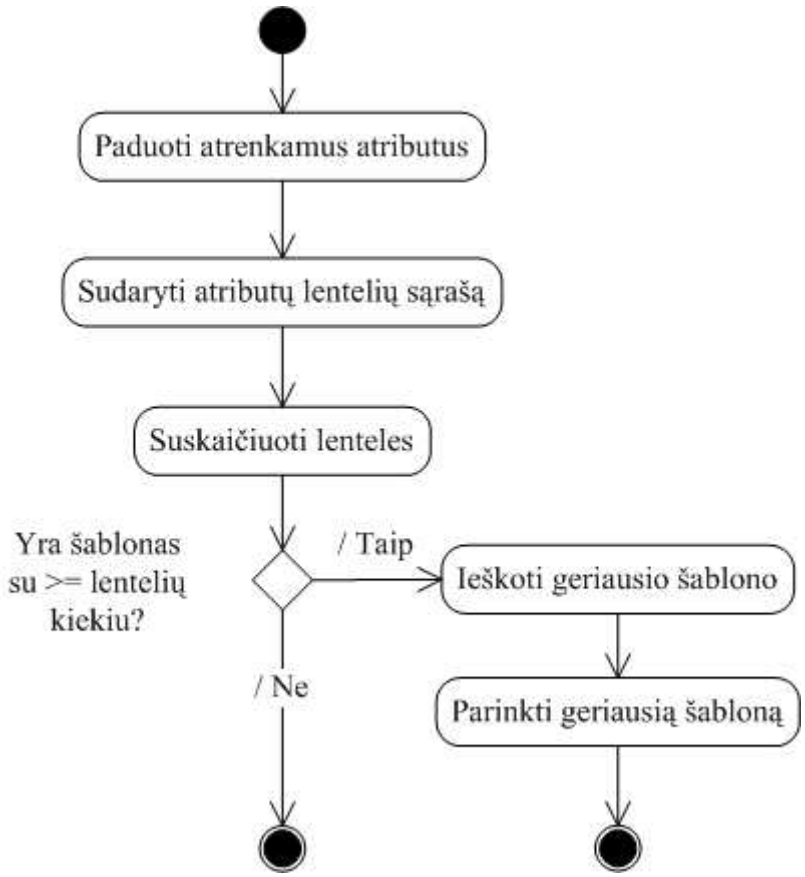
Kur *Lent_kiekis* – parodo koks lentelių skaičius atitiko užklausą.

Lenteles_pavad1,…, Lenteles_pavadN – lentelės, kurioms ieškomas bendras šablonas. Jų turi būti tiek, kiek suformuotame lentelių sąraše.

Kiekis – Lentelių kiekis, kuriam ieškomas šablonas. Jis padeda atfiltruoti tuos šablonus, kurie neturi tų lentelių, kurioms ieškomas šablonas.

Taip gaunamas maksimalus 10 šablonų sąrašas, kuriame be atitikusio užklausą lentelių kiekio, pateikiamas šablono identifikatorius ir tą šabloną sudarančių lentelių kiekis didėjimo tvarka. Taip pirmuoju numeriu sąraše yra šablonas, kuris neturi perteklinių lentelių arba jų turi minimaliai. Jis ir priskiriamas prie vykdomos užklausos.

Jeigu užklausa nauja ir/ arba tinkamiausio šablono nėra, tada vykdomos užklausos FROM dalyje panaudojami virtualių lentelių sujungimai. Kai surandamas tinkamas lentelių apjungimo šablonas, Meta duomenų bazėje su šablonu yra susiejama užklausa, kad kitą kartą vykdant tą pačią užklausą, nereikėtų per naują ieškoti šablono. Jeigu atnaujinami šablonai, tai vykdant užklausas, kiekvienai užklausai per naują ieškomas lentelių sujungimo šablonas, nes gali būti sudarytas dar efektyvesnis.



13 pav. Lentelių apjungimo šablono paieška

3.7.4. Užklausos statistikos kaupimas

Vartotojui sudarant ir vykdant užklausas, apie kiekvieną užklausą Meta duomenų bazėje yra kaupiama statistika, kad būtų galima matyti, kurios užklausos dažnai vykdomos ir koks jų įvykdymo laikas.

3.8. Užklausų formavimo metodo, kai panaudojami lentelių apjungimo šablonai išvados

1. Specifikuoti funkciniai ir nefunkciniai reikalavimai siūlomam užklausų formavimo metodui, taip pat apibrėžti duomenų įėjimai, išėjimai.
2. Papildyta Meta duomenų bazė 3 lentelėmis, kurios skirtos informacijai susijusiai su šablonais saugoti, kuri panaudojama užklausos vykdymo metu.
3. Sudaryta lentelių apjungimo šablonų formavimo metodika, kurios metu lentelių grupėms suformuojami šablonai, kurie panaudojami užklausose, kad sparčiau atrinktų ir filtruotų reikiamus duomenis.
4. Patobulintas užklausų formavimo metodas, kuris leidžia parinkti tokį lentelių apjungimo šabloną, kuriame nebūtų perteklinių lentelių, arba jų būtų minimaliai.

5. Pasinaudojus analizės dalyje esančiais užklausų derinimo ar optimizavimo būdais, suformuotos užklausos reikiamiems įrašams filtruoti ir atrinkti.

4. Užklausų vykdymo spartos eksperimentinis tyrimas

4.1. Eksperimento tikslas ir uždaviniai

Atliktas eksperimentas su užklausomis, stebint dviejų metodų spartos ir tikslumo charakteristikas filtruojant ir atrenkant reikiamus duomenis iš kelių duomenų bazių. Buvo stebima kaip kinta sparta atrenkant ir filtruojant duomenis:

1) naudojant virtualias lenteles;
2) tiesiogiai iš lentelių;
3) kartu su perteklinėmis lentelėmis.

Eksperimentinis tyrimas atliktas su MS SQL Server 2000 DBVS. Eksperimentiniai duomenys yra iš Lietuvos Miškotvarkos Informacinės Sistemos duomenų bazės. Eksperimentui atlikti buvo pasirinkta panaši į vartotojo aplinka: geografiškai nuo serverio nutolęs kompiuteris, interneto greitis – 4 Mb/s. Kadangi užklausos apdorojimo greičiui turi įtakos interneto kanalo pralaidumas, jo ir serverio apkrautumas, kiekviena užklausa buvo vykdoma po kelis ir daugiau kartų skirtingų laikotarpiu ir stebimas laikų vidurkis.

4.2. Užklausų vykdymo spartos tyrimas nenaudojant perteklinių lentelių.

Visų pirma buvo atliktas eksperimentas su užklausomis, kuriose atributai atrenkami naudojant virtualias lenteles, užklausas pakeičiant taip, kad būtų naudojamos tik tos lentelės, kurios susijusios su atrenkamais atributais. (A) užklausa, atrenkanti 17 atributų iš lentelių, panaudojant 3 virtualias lenteles. (B) užklausa, atrenkanti tuos pačius 17 atributų, tik iš pačių 15 lentelių tiesiogiai.

A užklausa:

```
SELECT  DISTINCT  top 100 percent
[m#0.UR],[m#0.GIR],[m#0.KV],[m#1.SKNR],[m#1.SD],[m#1.ZK],[m#1.PL],[m#1.STUR
],[m#1.BON],[m#1.DTG],[m#1.VMR],[m#2.UP],[m#2.PROC],[m#2.eilt_id],
[m#1.MK1],[m#1.AKL],[m#0.MUID1]
FROM  metadbgenAntraste LEFT OUTER JOIN  metadbgenMaketas1 ON
[m#0.ID]=[m#1.ID]
LEFT OUTER JOIN  metadbgenMaketas2 ON [m#0.ID]=[m#2.ID]
ORDER BY  [m#0.UR] ASC, [m#0.GIR] ASC, [m#0.KV] ASC, [m#1.SKNR] ASC,
[m#1.SD] ASC
```

B užklausa:

```
SELECT  DISTINCT  top 100 percent dbo.Uredijos.mu_kod, dbo.Gir_kvar.kv_nr,
dbo.Ur_gir.gir_kod, dbo.Uredijos.mu_id, dbo.Kv_skl.skl_nr,
dbo.Kv_skl.skl_dalis, dbo.Geo_objektai.pl,
                      dbo.Zemes_naudm_kateg.zk_kod1,
dbo.Medziu_rusys.mr_rkod, dbo.Maketas1.bon_kod, dbo.Augavietes.dtg,
dbo.Misku_kategorijos.mk_kod, dbo.Maketas1.amz_kl_kod,
dbo.Maketas1.skl_tur, dbo.Maketas2.procent, dbo.Ukines_priemones.up_kod,
dbo.Maketas2.eilt_id
FROM         dbo.Maketas1 LEFT OUTER JOIN
                      dbo.Medziu_rusys ON dbo.Maketas1.mr_id =
dbo.Medziu_rusys.mr_id FULL OUTER JOIN
                      dbo.Sklypai LEFT OUTER JOIN
                      dbo.Augavietes ON dbo.Sklypai.dtg_id =
dbo.Augavietes.dtg_id ON dbo.Maketas1.skl_id = dbo.Sklypai.skl_id FULL
OUTER JOIN       dbo.Misku_kategorijos INNER JOIN
                      dbo.Sklypu_misku_kat ON dbo.Misku_kategorijos.mk_id =
dbo.Sklypu_misku_kat.mk_id RIGHT OUTER JOIN
dbo.Geo_objektai ON dbo.Sklypu_misku_kat.skl_id = dbo.Geo_objektai.skl_id
LEFT OUTER JOIN
                      dbo.Uredijos INNER JOIN
                      dbo.Kv_skl ON dbo.Uredijos.mu_id = dbo.Kv_skl.mu_id
INNER JOIN
                      dbo.Kvartalai ON dbo.Kv_skl.kv_id =
dbo.Kvartalai.kv_id INNER JOIN
                      dbo.Gir_kvar ON dbo.Kvartalai.kv_id =
dbo.Gir_kvar.kv_id INNER JOIN
                      dbo.Ur_gir ON dbo.Kv_skl.ur_gir_id =
dbo.Ur_gir.ur_gir_id ON dbo.Geo_objektai.skl_id = dbo.Kv_skl.skl_id LEFT
OUTER JOIN
                      dbo.Zemes_naudm_kateg ON dbo.Geo_objektai.zk_id =
dbo.Zemes_naudm_kateg.zk_id LEFT OUTER JOIN
                      dbo.Maketas2 LEFT OUTER JOIN
                      dbo.Ukines_priemones ON dbo.Maketas2.up_id =
dbo.Ukines_priemones.up_id ON dbo.Geo_objektai.skl_id = dbo.Maketas2.skl_id
ON
                      dbo.Sklypai.skl_id = dbo.Geo_objektai.skl_id
ORDER BY  Uredijos.mu_kod ASC, Ur_gir.gir_kod ASC, Gir_kvar.kv_nr ASC,
Kv_skl.skl_nr  ASC, Kv_skl.skl_dalis ASC
```

Šios A ir B užklausos įvykdytos trijose Lietuvos Miškotvarkos Informacinės Sistemos duomenų bazėse:

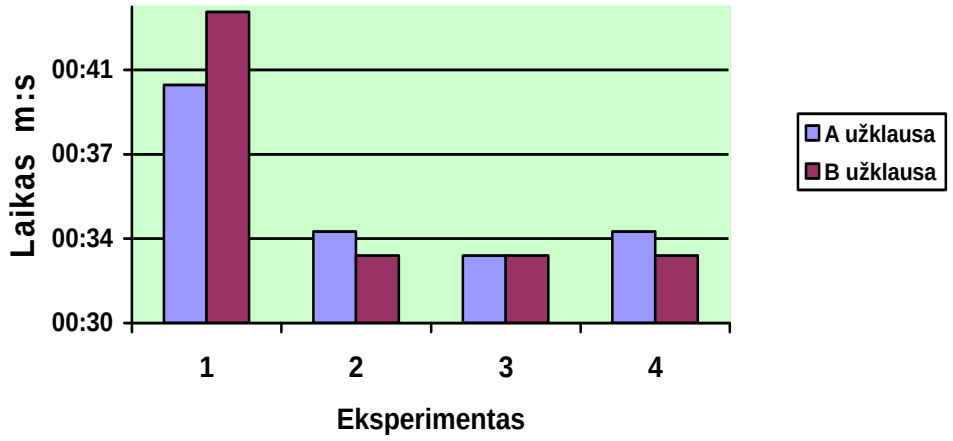
- PVP – 145 370 įrašų.
- Bandymai2 – 250 000 įrašų
- RMKAD – 2 128 624 įrašų.

14 –oje lentelėje pateikiamas laikas, reikalingas užklausai įvykdyti ir pateikti rezultatus, bei atrinktų įrašų skaičius. O šio tyrimo rezultatai pavaizduoti 14 -17 paveiksluose.

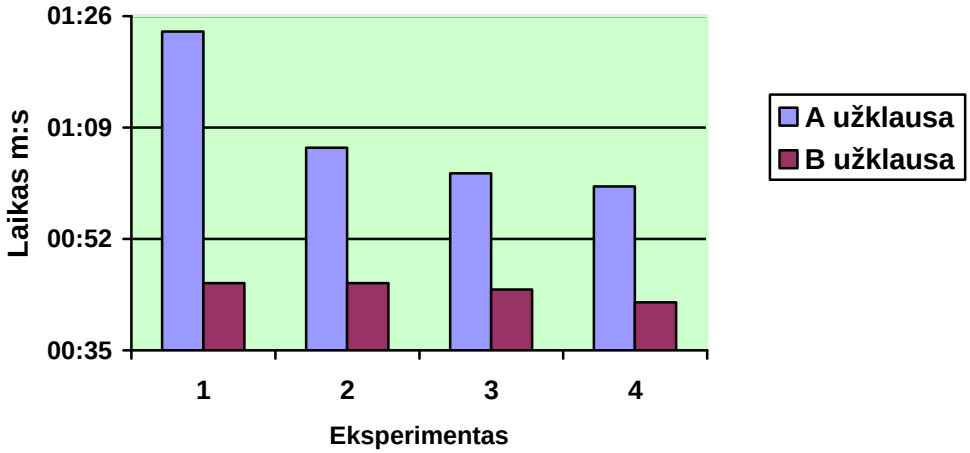
14 lentelė. Užklausų įvykdymo laikas

	PVP A	**PVP B**	**Bandymai2 A**	**Bandymai2 B**	**RMKAD A**	**RMKAD B**
Laikas	40s	43 s	1m 24s	45s	35m 58 s	12m 49s
Laikas	34s	33 s	1m 06s	45s	29m 39s	7m 51s
Laikas	33s	33s	1m 02s	44s	27m 56s	10m 31s
Laikas	34s	33s	1m00s	42s	35m 56s	8m 00s

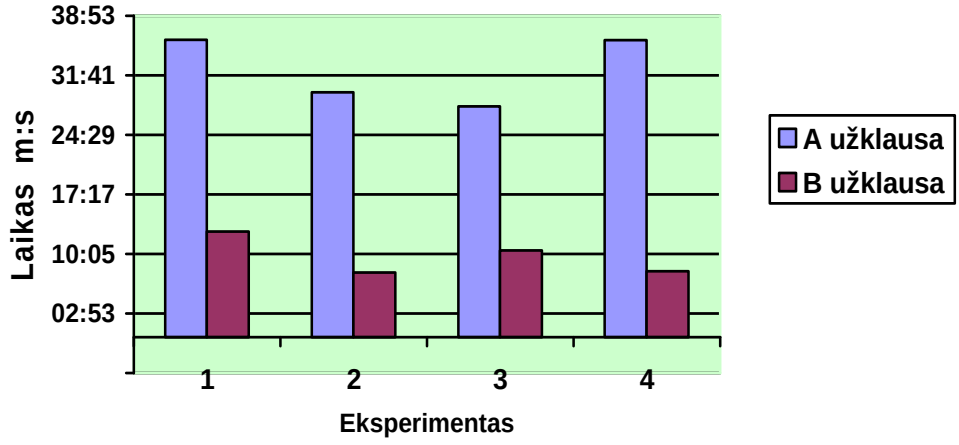
	PVP A	PVP B	Bandymai2 A	Bandymai2 B	RMKAD A	RMKAD B
Vidutinis laikas	35s	36s	1m 08s	44s	32m 22s	9m 48s
Atrinkta	141514 įrašų	141514 įrašų	189010 įrašų	189010 įrašų	2147386 įrašų	2147386 įrašų



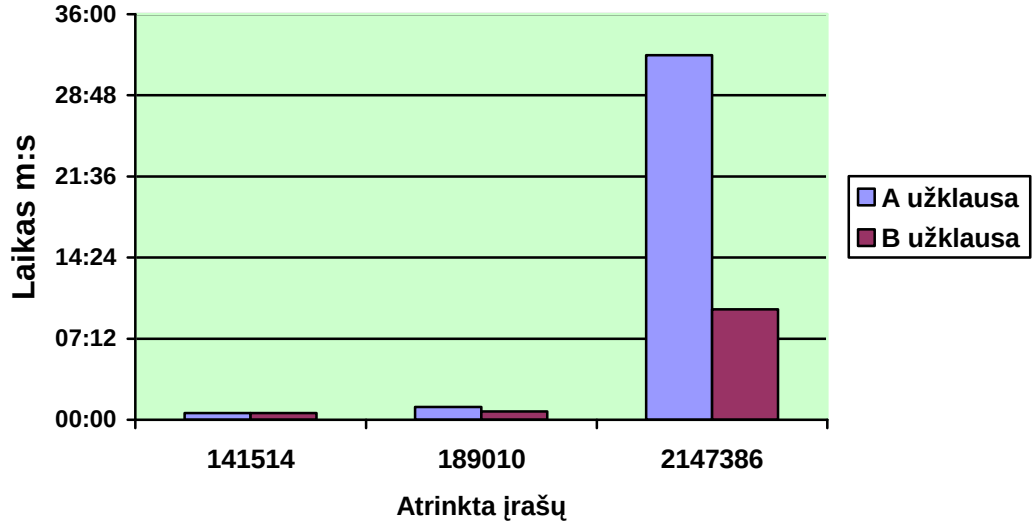
14 pav. PVP duomenų bazėje užklausų įvykdymo laikas.



15 pav. Bandymai2 DB užklausų įvykdymo laikas



16 pav. RMKAD duomenų bazėje užklausų įvykdymo laikas



17 pav. Laiko ir atrinktų įrašų priklausomybė

Kaip matyti iš 14-oje lentelėje esamų duomenų ar 14-16 paveiksluose esančių grafikų, didėjant įrašų kiekiui didėja ir laikas, sunaudojamas užklausos įvykdymui. Ypač daug laiko sugaištama, kada reikia atrinkti reikiamus atributus iš tų lentelių, kurios turi milijoninius skaičius įrašų, ir atrenkama virš 76% lentelėse esančių įrašų, tai puikiai atspindi 17 paveiksle esantis grafikas. **Bandymai2** ir **RMKAD** duomenų bazėje įvykdytos užklausos naudojančios lentelių apjungimo šabloną įvykdytos nuo 1,5 iki 3,3 karto greičiau, nei naudojant metodą, pagal kurį apjungiamos suformuotos maketams virtualios lentelės. Kadangi virtualiose lentelėse yra perteklinės informacijos, ji padidina užklausos vykdymo laiką, sumažindama spartą. Todėl verta naudoti tokį sprendimą, kuris formuojamoje užklausoje naudoja šabloną su minimaliu kiekiu nenaudojamų perteklinių lentelių.

4.3. Užklausų vykdymo spartos tyrimas, kai naudojamos kelios perteklinės lentelės.

Taip pat buvo stebima kaip veikia užklausos turinčios filtrą, kai kinta atrenkamų atributų kiekis, perteklinių lentelių skaičius ir sujungiamų virtualių lentelių skaičius.

Šios užklausos įvykdytos dvejose Lietuvos Miškotvarkos Informacinės Sistemos duomenų bazėse:

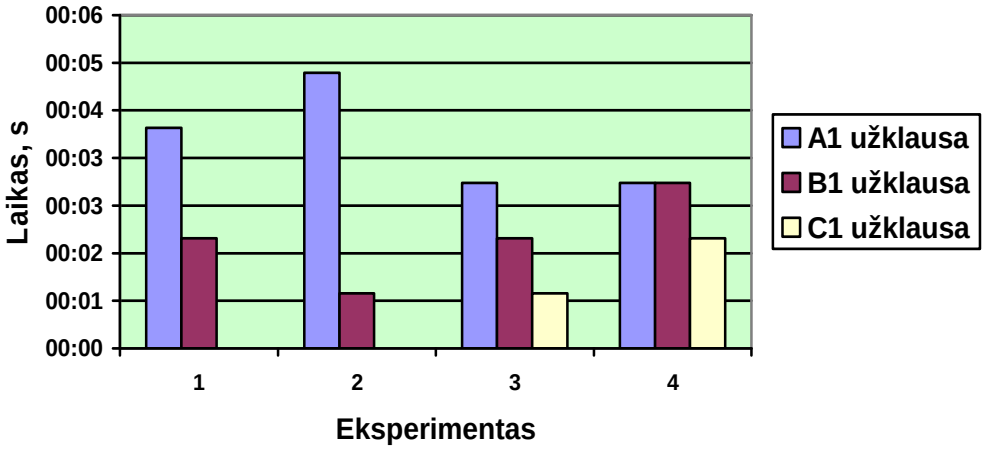
- RMKAD – 2 128 624 įrašų.
- MKAD2 – 2 834 894 įrašų.

Toliau 15 -18 lentelėse pateikiamas laikas, reikalingas užklausai įvykdyti ir pateikti rezultatus, bei atrinktų įrašų skaičius. O šio tyrimo rezultatai pavaizduoti 18 -25 paveiksluose.

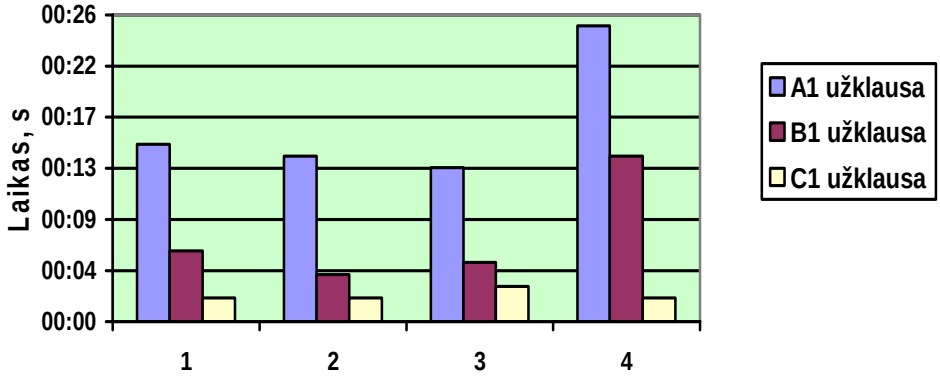
(A1) užklausa, atrenkanti 3 atributus iš lentelių, panaudojant 1 virtualią lentelę. (B1) užklausa, atrenkanti tuos pačius 3 atributus, iš pačių lentelių tiesiogiai su perteklinėmis lentelėmis. (C1) užklausa, atrenkanti tuos pačius 3 atributus, tik iš pačių lentelių tiesiogiai. Užklausų vykdymo laikas 15- oje lentelėje, o užklausų kodas 1 priede.

15 lentelė. Užklausų įvykdymo laikas

	RMKAD A1	**MKAD2 A1**	**RMKAD B1**	**MKAD2 B1**	**RMKAD C1**	**MKAD2 C1**
Laikas	4s	15s	2s	06s	0s	02 s
Laikas	5s	14s	1s	04s	0s	02s
Laikas	3s	13s	2s	05s	1s	03s
Laikas	3s	25s	3s	15s	2s	02
Atrinkta	20 įrašų	37 įrašų	20 įrašų	37 įrašų	20 įrašų	37 įrašų



18pav. RMKAD duomenų bazėje užklausų įvykdymo laikas



19 pav. MKAD2 duomenų bazėje užklausų įvykdymo laikas

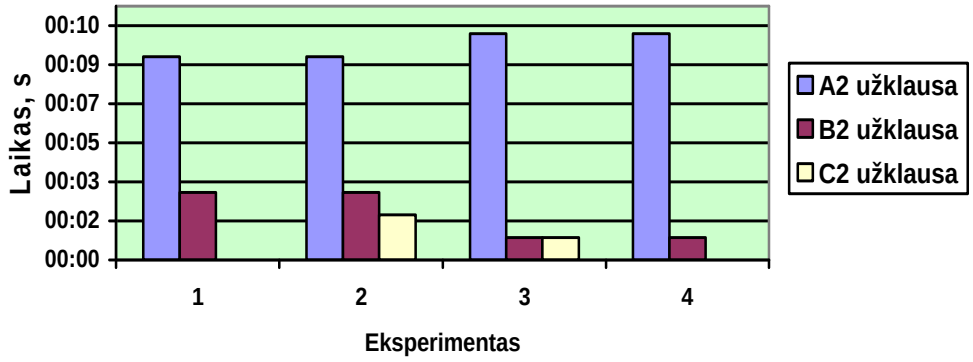
Kaip matyti iš grafikų, esančių 18 ir 19 paveiksluose, užklausa (A1) naudojanti virtualias lenteles yra įvykdoma pakankamai greitai, tačiau perrašius užklausą, kad atrinktų ir

filtruotų tuos pačius atributus su perteklinėmis lentelėmis (B1) ar pašalinus perteklines lenteles (C1) užklausa dar labiau pagreitėja.

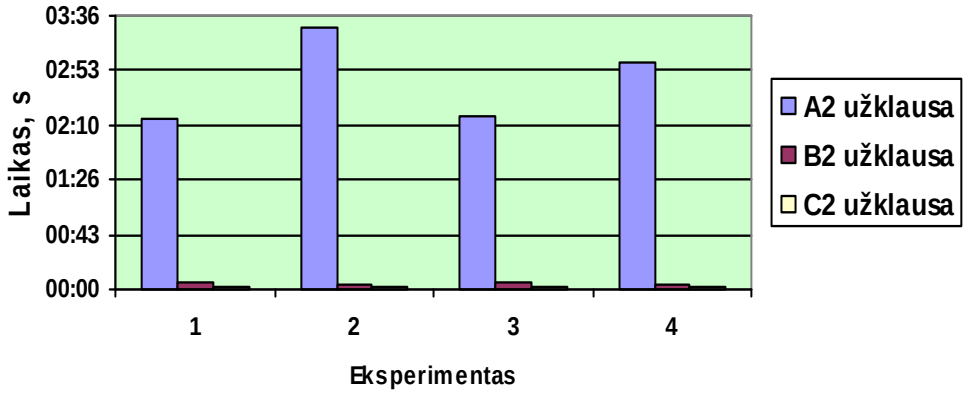
(A2) užklausa, atrenkanti 5 atributus iš lentelių, panaudojant 2 virtualias lenteles. (B2) užklausa, atrenkanti tuos pačius 5 atributus, iš pačių lentelių tiesiogiai su 6 perteklinėmis lentelėmis. (C2) užklausa, atrenkanti tuos pačius 5 atributus, tik iš pačių lentelių tiesiogiai. Užklausų kodas 1 priede.

16 lentelė. Užklausų įvykdymo laikas

	RMKAD A2	**MKAD2 A2**	**RMKAD B2**	**MKAD2 B2**	**RMKAD C2**	**MKAD2 C2**
Laikas	9s	02:14s	3s	2s	0	2s
Laikas	9s	03:27s	3s	2s	2s	2s
Laikas	10s	02:16s	1s	2s	1s	2s
Laikas	10s	02:59s	1s	4s	0s	2s
Atrinkta	20 įrašų	37 įrašų	20 įrašų	37 įrašų	20 įrašų	37 įrašų



20 pav. RMKAD duomenų bazėje užklausų įvykdymo laikas



21 pav. MKAD2 duomenų bazėje užklausų įvykdymo laikas

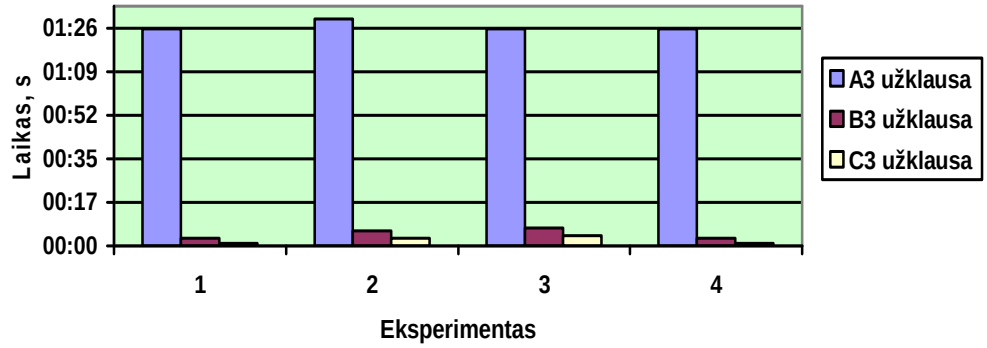
Kaip matyti iš šio eksperimento – 16 lentelės, abejose duomenų bazėse įvykdyta (A2) užklausa vykdoma lėčiau, nei (B2 ir C2) užklausos. Todėl atsisakius virtualių lentelių, šiuo

konkrečiu atveju, galima keliasdešimt kartų paspartinti užklausos veikimą. Ypač tai akivaizdu MKAD2 duomenų bazėje, kai atrenkant 5 atributus virtualių lentelių pagalba sugaištama virš 2 minučių, o be jų, net su 6 perteklinėmis lentelėmis iki 4 sekundžių (21 paveikslas).

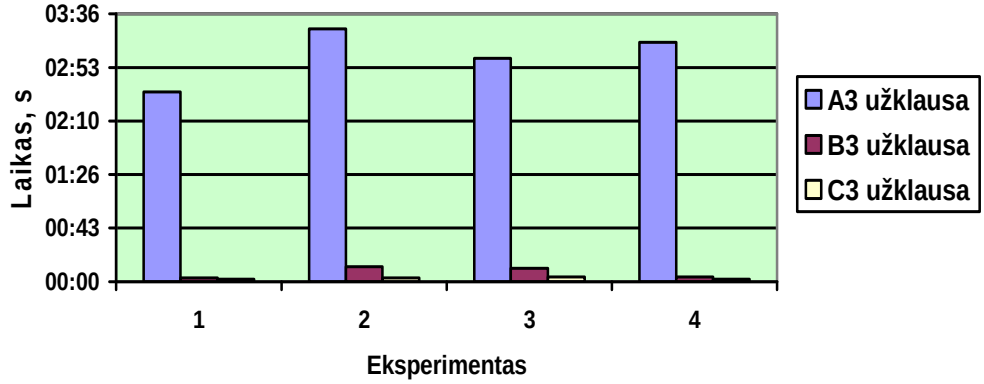
(A3) užklausa, atrenkanti 8 atributus iš lentelių, panaudojant 3 virtualias lenteles. (B3) užklausa, atrenkanti tuos pačius 8 atributus, iš pačių lentelių tiesiogiai su 5 perteklinėmis lentelėmis. (C3) užklausa, atrenkanti tuos pačius 8 atributus, tik iš pačių lentelių tiesiogiai. Užklausų kodas 1 priede.

17 lentelė. Užklausų įvykdymo laikas

	RMKAD A3	**MKAD2 A3**	**RMKAD B3**	**MKAD2 B3**	**RMKAD C3**	**MKAD2 C3**
Laikas	01:26s	2:33 s	3s	3s	1s	2s
Laikas	01:30s	3:24s	6s	12s	3s	3s
Laikas	01:26s	3:00s	7s	11s	4s	4s
Laikas	01:26s	3:13s	3s	4s	1s	2s
Atrinkta	123 įrašų	243 įrašų	123 įrašų	243 įrašų	123 įrašų	243 įrašų



22 pav. RMKAD duomenų bazėje užklausų įvykdymo laikas



23 pav. MKAD2 duomenų bazėje užklausų įvykdymo laikas

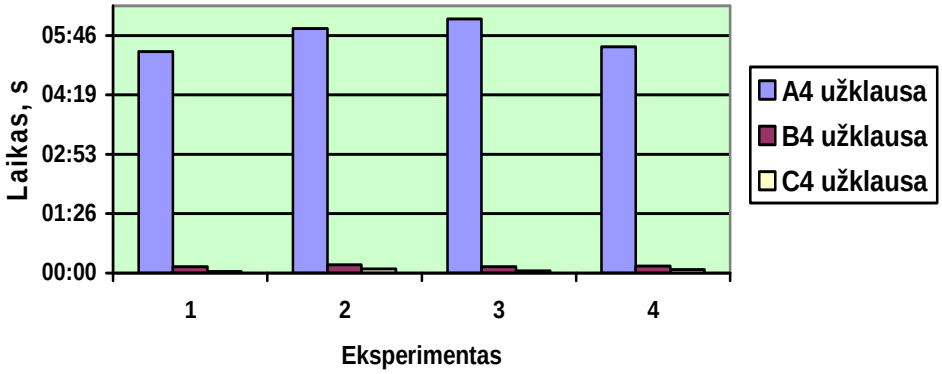
Kaip matyti iš šio eksperimento ir 17- oje lentelėje esančių užklausos įvykdymo duomenų, abejose duomenų bazėse įvykdyta (A3) užklausa taip pat vykdoma lėčiau, nei (B3

ir C3) užklausos. 22 ir 23 paveiksle esantys grafikai parodo, kad tiesioginis duomenų atrinkimas iš lentelių, nenaudojant virtualių lentelių, sutaupo brangaus laiko vartotojams.

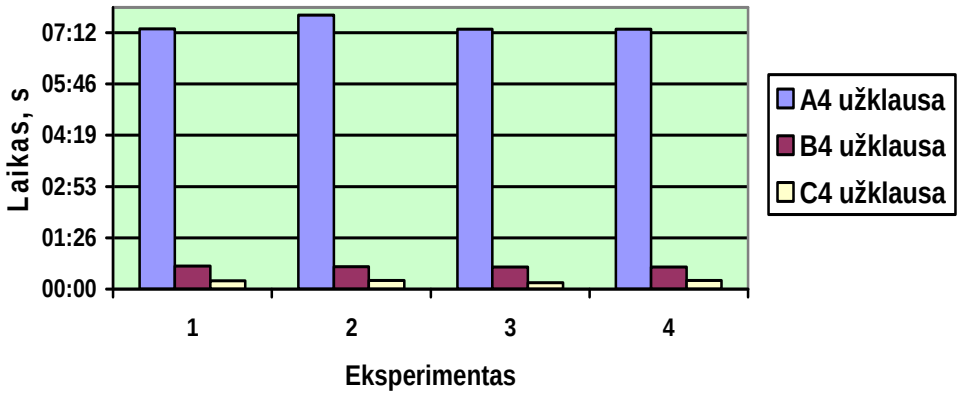
(A4) užklausa, atrenkanti 15 atributų iš lentelių, panaudojant 5 virtualias lenteles. (B4) užklausa, atrenkanti tuos pačius 15 atributų, iš pačių lentelių tiesiogiai su 5 perteklinėmis lentelėmis. (C4) užklausa, atrenkanti tuos pačius 15 atributų, tik iš pačių lentelių tiesiogiai. Užklausų kodas 1 priede.

18 lentelė. Užklausų įvykdymo laikas

	RMKAD A4	**MKAD2 A4**	**RMKAD B4**	**MKAD2 B4**	**RMKAD C4**	**MKAD2 C4**
Laikas	05:22 s	07:19s	9s	39s	2s	14s
Laikas	05:56s	07:42	12s	38s	6s	15s
Laikas	06:10s	07:18s	9s	37s	3s	11s
Laikas	05:29s	07:18s	10s	37s	5s	15s
Atrinkta	124 įrašų	244 įrašų	124 įrašų	244 įrašų	124 įrašų	244 įrašų



24 pav. RMKAD duomenų bazėje užklausų įvykdymo laikas



25 pav. MKAD2 duomenų bazėje užklausų įvykdymo laikas

Kaip matyti 18-oje lentelėje, vykdant (A4) užklausą abejose duomenų bazėse sugaištama nuo 5 iki 7 minučių, tiek trunka 5 virtualių lentelių sujungimai ir duomenų atrinkimas. Kai tuo tarpu atrenkant duomenis tiesiogiai net su 5 perteklinėmis lentelėmis

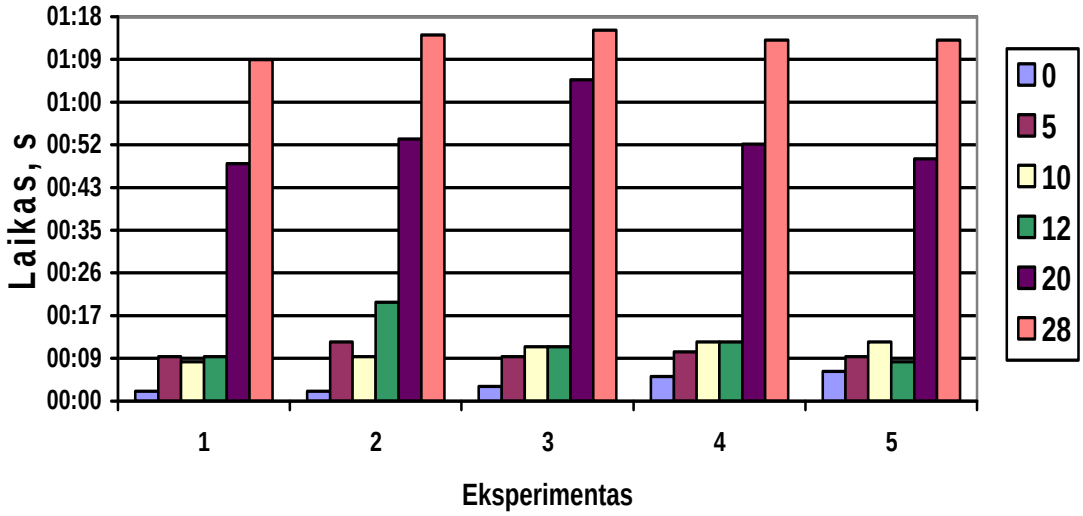
sugaištama iki 39 sekundžių. Be perteklinių lentelių užklausa (C4) pasižymi didesniu spartumu ir 15 atributų atrenka RMKAD duomenų bazėje atrenka per 2-6 sekundes, o MKAD2 per 11 – 15 sekundžių. Kaip matyti 24 paveiksle esančiame grafike RMKAD duomenų bazėje duomenys be perteklinių lentelių atrenkami ir filtruojami 65 kartus sparčiau, nei naudojant virtualias lenteles. O 25 paveiksle esantis grafikas parodo, kad MKAD2 duomenų bazėje duomenys be perteklinių lentelių atrenkami ir filtruojami apie 35 kartus sparčiau palyginus su virtualiomis lentelėmis. Taigi, kuo daugiau virtualių lentelių naudojama duomenis atrinkti, tuo ilgiau užtrunkama, kol gaunamas užklausos rezultatas.

4.4. Užklausų vykdymo spartos tyrimas, stebint perteklinių lentelių įtaką spartai.

Kaip buvo galima pastebėti iš aukščiau esančių grafikų, kad perteklinės lentelės įtakoja užklausos įvykdymo laiką. Buvo nuspręsta stebėti kaip ir koks perteklinių lentelių kiekis įtakoja spartą. Užklausos įvykdytos tose pačiose dvejose Lietuvos Miškotvarkos Informacinės Sistemos duomenų bazėse RMKAD ir MKAD2. Tyrimo rezultatus galime pamatyti 19 ir 20 lentelėse ir grafiškai pavaizduotus 26 -27 paveiksluose, kur užklausos, atrenka 15 atributų iš lentelių su skirtingu kiekiu perteklinių lentelių. Užklausų B4, C4, D,E, F ir G kodas 1 priede.

19 lentelė. Užklausų su perteklinėmis lentelėmis vykdymo laikas RMKAD duomenų bazėje

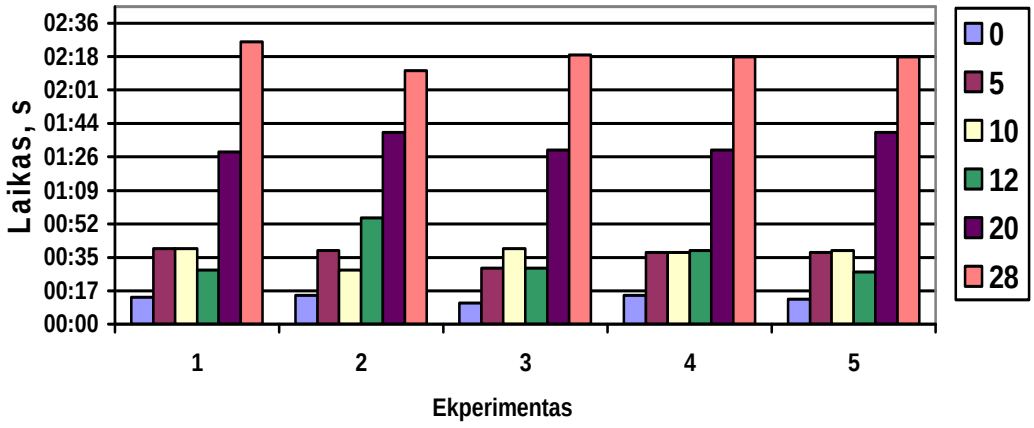
Perteklinės lentelės	**0 C4**	**5 B4**	**10 D**	**12 E**	**20 F**	**28 G**
Laikas	2s	9s	8s	9s	48s	01:09s
Laikas	2s	12s	9s	20s	53s	01:14s
Laikas	3s	9s	11s	11s	1:05s	01:15s
Laikas	5s	10s	12s	12s	52s	01:13s
Laikas	6s	9s	12s	8s	49s	01:13s
Atrinkta	124 įrašų					



26 pav. RMKAD duomenų bazėje užklausų įvykdymo laikas

20 lentelė. Užklausų su perteklinėmis lentelėmis vykdymo laikas MKAD2 duomenų bazėje

Perteklinė lentelės	**0** **C4**	**5** **B4**	**10** **D**	**12** **E**	**20** **F**	**28** **G**
Laikas	14s	39s	39s	28s	01:29s	02:26s
Laikas	15s	38s	28s	55s	01:39s	02:11s
Laikas	11s	29s	39s	29s	01:30s	02:19s
Laikas	15s	37s	37s	38s	01:30s	02:18s
Laikas	13s	37s	38s	27s	01:39s	02:18s
Atrinkta	244 įrašų					



27 pav. MKAD2 duomenų bazėje užklausų įvykdymo laikas

Kaip matyti, iš eksperimento metu gautų duomenų, 26 ir 27 grafikų, abejose RMKAD ir MKAD2 duomenų bazėse įvykdytų užklausų, tos užklausos, kurios neturi perteklinių lentelių yra vykdomos greičiausiai. Užklausos turinčios 5, 10, 12 perteklinių lentelių yra

įvykdomos per panašų laiką, jų vykdymui įtakos turi serverio ir interneto linijos apkrovimas, todėl kelios perteklinės lentelės smarkiai neįtakoja užklausos įvykdymo laiko. Kai užklausa turi 20 ar daugiau perteklinių lentelių užklausos įvykdymo laikas išauga gan smarkiai, t.y. kai užklausa neturi perteklinių lentelių iki, kai jų yra 28 vykdymo laikas pailgėja nuo 10 iki 20 kartų. Tačiau (A4) užklausa su virtualiomis lentelėmis įvykdyta RMKAD duomenų bazėje vidutiniškai vykdoma 5 su puse minutės, o MKAD2 virš 7 minučių, taigi užklausa suformuota pagal sudarytą metodą su 28 perteklinėmis lentelėmis įvykdoma nuo 3 iki 5 kartų sparčiau.

4.5. Eksperimento rezultatų apibendrinimas

Buvo atliktas eksperimentas keliose skirtingose Lietuvos Miškotvarkos Informacinės Sistemos duomenų bazėse, stengiantis pasirinkti laiką, kada serveris ir interneto ryšys mažiausiai apkrauti, kad gautume kiek įmanoma tikslesnį užklausų įvykdymo laiką, kuris reikalingas užklausų spartai įvertinti.

Užklausų vykdymo spartos tyrimas nenaudojant perteklinių lentelių parodė, kad užklausos greičiui daug įtakos turi įrašų kiekis esantis lentelėse ir koks jų kiekis turi būti atrinktas. Tyrimui panaudojus duomenų bazes, kurių lentelėse yra nuo 140 000 iki 2 128 624 įrašų matyti, kad kuo daugiau įrašų yra lentelėse ir kuo daugiau virtualių lentelių reikia sujungti, tuo daugiau laiko sugaišta reikiamiems duomenims atrinkti. Panaudojus lentelių apjungimo šabloną, užklausos paspartėja nuo 1,5 iki 3,3 karto.

Antras užklausų vykdymo spartos tyrimas, kai naudojamos kelios perteklinės lentelės parodė, kad atrenkamų atributų ir sujungiamų lentelių kiekis įtakoja užklausos įvykdymo laiką. Kuo labiau duomenys filtruojami, kuo mažiau atributų reikia atrinkti, ir kuo mažiau lentelių sujungti, tuo greičiau užklausos rezultatas yra gaunamas. Taigi, atrenkant ir filtruojant duomenis pagal tam tikrus kriterijus galima paspartinti užklausas iki 65 kartų virtualias lenteles keičiant paprastomis lentelėmis.

Trečias užklausų vykdymo spartos tyrimas, kai stebima perteklinių lentelių įtaką spartai parodė, kad nedidelis kiekis perteklinių lentelių 5-12 smarkiai neįtakoja užklausos vykdymo laiko. Tačiau didinant perteklinių lentelių kiekį iki 28 lentelių, užklausos vykdymas 10-20 kartų ilgėja, bet šiuo atveju užklausos sparta 3-5 kartus didesnė, nei duomenis atrinktume virtualių lentelių pagalba.

5. Išvados

1) Didėjant duomenų bazėse kaupiamų duomenų kiekiams, vis dažniau iškyla duomenų atrinkimo ir filtravimo našumo problema, todėl yra ieškoma būdų ir metodų kaip paspartinti duomenų atrinkimą ir filtravimą iš tų duomenų bazių, kurios turi lenteles su milijonais įrašų.
2) Buvo atlikta duomenų bazių spartos derinimo ir duomenų bazių spartos optimizavimo literatūros analizė, kurios metu nustatyti veiksniai skatinantys užklausas veikti greičiau, ir faktoriai mažinantys užklausų spartumą. Nustatyta, kad apjungiant kelias ir daugiau virtualių lentelių, ar naudojant indeksus, duomenų bazės sparta kai kuriais atvejais gali nepadidėti, o net sumažėti, todėl buvo ieškomas kitoks būdas užklausos spartai padidinti.
3) Duomenims filtruoti ir atrinkti buvo sudaryta lentelių apjungimo šablonų formavimo metodika, pagal kurią tam tikroms lentelių grupėms, labiausiai naudojamoms vartotojų užklausose, suformuojami šablonai, kurių dėka paspartėja užklausos rezultato pateikimas.
4) Užklausų formavimo metodas patobulintas taip, kad vartotojo suformuotai užklausai būtų parenkamas ir priskiriamas tinkamiausias lentelių apjungimo šablonas, kuris turi mažiausiai perteklinių lentelių. Buvo atsisakyta naudoti virtualias lenteles, su kuriomis užklausos vykdomos lėčiau, nes perteklinėms lentelėms ir jų duomenims apdoroti reikia papildomo laiko.
5) Meta duomenų bazė papildyta 3 lentelėmis, kurios skirtos lentelių apjungimo šablonams, su jais susijusiai informacijai bei užklausų statistikai saugoti. Šios lentelės užpildomos sukuriant naują lentelių apjungimo šabloną, taip pat vykdant naują užklausos projektą, tuomet priskiriamas šablonas bei visų užklausų vykdymo metu kaupiama užklausų vykdymo statistika, užklausų greičiui įvertinti. Šiose lentelėse esančios informacijos dėka vartotojams sparčiau pateikiamas užklausos rezultatas.
6) Atliktas užklausų vykdymo spartos eksperimentinis tyrimas su Lietuvos Miškotvarkos Informacinės Sistemos duomenų bazės duomenimis. Tyrimo metu pasiūlytas metodas įvertintas trimis būdais: naudojant virtualias lenteles; atrenkant duomenis tiesiogiai iš lentelių; atrenkant duomenis naudojant perteklines lenteles. Nustatyta, kad :
    a) atrenkant daugiau negu 76% įrašų iš lentelių turinčių apie 250 000 ir 2 128 624 įrašų ir naudojant virtualių lentelių metodą, negu sudarytą užklausų formavimo metodą su lentelių apjungimo šablonais, užklausų vykdymo laikas yra nuo 1,5 iki 3,3 kartų didesnis.

b) Atrenkant duomenis tiesiogiai iš lentelių ir filtruojant didžiąją dalį duomenų, reikiami įrašai gali būti gaunami iki 65 kartų sparčiau, nei su virtualių lentelių metodu. Tam įtakos turi perteklinių lentelių duomenys saugomi virtualiose lentelėse.

c) Atrenkant ir filtruojant didžiąją dalį duomenų, kai naudojamos 1 - 18 perteklinių lentelių, jų kiekis žymios įtakos užklausos rezultatui neturi. Naudojant 20 - 28 perteklines lenteles užklausos sparta sumažėja nuo 10 iki 20 kartų palyginus su užklausos sparta, kai duomenys atrenkami tiesiogiai iš lentelių, tačiau palyginus su virtualių lentelių metodu užklausos sparta yra 3 - 5 didesnė.

Pagal tyrimo rezultatus matyti, kad sudarytas užklausų formavimo metodas, panaudojant lentelių apjungimo šablonus, yra teisingas.

7) Magistrinio darbo tematika buvo perskaitytas pranešimas XIII tarpuniversitetinėje magistrantų ir doktorantų konferencijoje „Informacinės technologijos“, bei publikuotas straipsnis konferencijos pranešimų medžiagoje (8.2 priedas) .

6. Naudota literatūra:

[1] Charvet, F.; Pande, A. *Database Performance Study,* 2003 [žiūrėta 2008.05.09] . Prieiga per internetą: http://www.umsl.edu/divisions/business/mis/bov/TuningPaperV5.pdf

[2] *Microsoft SQL Server 2000 - DatabaseDesign* [žiūrėta 2008.02.25] Prieiga per internetą: http://portal.aauj.edu/portal_resources/downloads/database/microsoft_sql_server2000_database_design.pdf

[3] Whalen, E.; et al. *Microsoft SQL Server 2000 Performance Tuning Technical Reference.* Microsoft Press, 2001. 438p. ISBN 0-7356-1270-6.

[4] McGehee, B; *Tips on Optimizing SQL Server Indexes.* 2007, baandis [žiūrėta 2008.02.26] Prieiga per internetą: http://www.sql-server-performance.com/optimizing_indexes.asp

[5] Agrawal, S.; Chaudhuri, S.; Narasayya, V. *Automated Selection of Materialized Views and Indexes for SQL Databases***.** International Conference on VeryLarge Databases, Cairo, Egypt, *2000.* [žiūrėta 2008.04.21] Prieiga per internetą: http://citeseer.ist.psu.edu/699518.html

[6] Goldstein, J.; Larson, P. *Optimizing Queries Using Materialized Views:A Practical, Scalable Solution*. 2001 [žiūrėta 2008.04.25] Prieiga per internetą: http://portal.acm.org/citation.cfm?id=376284.375706&coll=&dl=acm&CFID=15151515&CFTOKEN=6184618

[7] Carpenter, D. *Indexed Views in SQL Server 2000*. [žiūrėta 2008.02.27] Prieiga per internetą: http://www.sqlteam.com/item.asp?ItemID=1015

[8] Kollar, L.; Ward, J. *Improving Performance with SQL Server 2000 Indexed Views*. [žiūrėta 2008.02.27] Prieiga per internetą: http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/indexvw.mspx

[9] Novick**,** A.*Indexed Views Basics in SQL Server 2000*. [žiūrėta 2008.02.27]. Prieiga per internetą: http://www.databasejournal.com/features/mssql/article.php/2119721

[10] Zhou, J.; Larson, P.; Goldstein, J. *Partially Materialized Views. 2005* [žiūrėta 2008.04.26] Prieiga per internetą: http://research.microsoft.com/research/pubs/view.aspx?type=Technical%20Report&id=931

[11] Valluri, S. R.; *Partially Materialized Partitioned Views.* 2005 [žiūrėta 2008.02.26] Prieiga per internetą: http://comad2005.persistent.co.in/COMAD2005Proc/pages046-057.pdf

[12] Powell, G.; *Beginning Database Design.* Wiley Publishing, Inc., 2006. 504 p. ISBN-13: 978-0-7645-7490-0

[13] Vieira, R. *Beginning SQL Server™ 2005 Programming.* Wiley Publishing, Inc, 2004. 720 p. ISBN-13: 978-0-7645-8433-6

[14] Krishnasamy, S. *Transact-SQL Query SQL Server Performance Tuning Tips.* [žiūrėta 2007.05.16] Prieiga per internetą: http://www.sql-server-performance.com/transact_sql.asp

[15] Agarwal, S.; et al. *Troubleshooting Performance Problems in SQL Server 2005*. Microsoft Corporation, 2005 [žiūrėta 2008.02.25] Prieiga per internetą: http://download.microsoft.com/download/1/3/4/134644fd-05ad-4ee8-8b5a-0aed1c18a31e/TShootPerfProbs.doc

[16] England K. *Microsoft SQL Server 2000 Performance Optimization and Tuning Handbook.* Butterworth-Heinemann, 2001. 390p. ISBN 1-55558-241-9.

[17] Kline, K.; Zanevsky, A.; Gould L. *Microsoft T-SQL Performance Tuning Part 3: Query Optimization Strategies.* O'Reilly & Associates. 2002 [žiūrėta 2008.02.25].Prieiga per internetą: http://reg.dlt.com/quest/pdf/microsoft%20infrastructure/microsoft%20sql%20server/tuning%20part%203.pdf

[18] Kline, K.; Zanevsky, A.; Gould L. *Microsoft T-SQL Performance Tuning Part 2: Index Tuning Strategies.* O'Reilly & Associates. 2002 [žiūrėta 2008.02.25]. Prieiga per internetą: http://reg.dlt.com/quest/pdf/microsoft%20infrastructure/microsoft%20sql%20server/tuning%20part%202.pdf

[19] Shasha, D.; Bonnet P. *Database tuning: Principles, Experiments, and Troubleshooting Techniques.* Morgan Kaufmann Publishers. 2003. 441 p. ISBN: 1558607536

[20] Gulutzan, P.; Pelzer T. *SQL Performance Tuning.* Addison Wesley. 2002. 373 p. ISBN: 0-201-79169-2

[21] Jones, D. *The Definitive Guide to SQL Server Performance Optimization.* 2002 [žiūrėta 2007.04.18]. Prieiga per internetą: http://nexus.realtimepublishers.com/DGSSPO.htm

[22] Tao, Y. et al. *Optimizing Large Star-Schema Queries with Snowakes via Heuristic-Based Query Rewriting***,** IBM Toronto Laboratory. 2003 [žiūrėta 2008.02.27] Prieiga per internetą: http://portal.acm.org/citation.cfm?id=961365

7. Terminų ir santrumpų žodynas

Duomenų bazė (**DB**)– tai duomenų apie konkrečius objektus tam tikroje dalykinėje srityje visuma.

DBVS – duomenų bazių valdymo sistema.

Užklausa – duomenų valdymo (įrašymo, atnaujinimo, šalinimo) ar išrinkimo iš duomenų bazės pagal užduotas sąlygas arbe be jų nurodymas.

SQL – *Structured Query Language*. Kalba, skirta užklausoms rašyti.

Įdėtinė užklausa – Užklausa suformuota kitos užklausos viduje.

Užuominos (ang. Hints) – nuorodos DBVS užklausų optimizatoriui.

8. Priedai

8.1. Užklausų kodas

Šiame priede pateikiamas 4.2 ir 4.3 poskyrių aprašytų eksperimente įvykdytų užklausų kodai.

A1 užklausa:

```
SELECT   top 100 percent [m#0.UR],[m#0.GIR],[m#0.KV] FROM metadbgenAntraste
WHERE ([m#0.ur]=1 AND [m#0.gir]=3 AND [m#0.kv]=173)
```

B1 užklausa:

```
SELECT top 100 percent  Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod
FROM  Rysiai_su_pos RIGHT OUTER JOIN
                      Geo_objektai LEFT OUTER JOIN
                      Vietoves_blokai INNER JOIN
                      Vietoves ON Vietoves_blokai.kdv_id = Vietoves.kdv_id
INNER JOIN
                      Kad_blokai ON Vietoves_blokai.kdb_id =
Kad_blokai.kdb_id INNER JOIN
                      Rysiai_su_blokais ON Kad_blokai.kdb_id =
Rysiai_su_blokais.kdb_id ON Geo_objektai.skl_id = Rysiai_su_blokais.skl_id
ON
                      Rysiai_su_pos.skl_id = Geo_objektai.skl_id LEFT OUTER
JOIN            Bloku_sklypai INNER JOIN
                      Rysiai_su_ks INNER JOIN
                      Kad_sklypai ON Rysiai_su_ks.kskl_id =
Kad_sklypai.kskl_id ON Bloku_sklypai.kskl_id = Kad_sklypai.kskl_id ON
                      Geo_objektai.skl_id = dbo.Rysiai_su_ks.skl_id LEFT
OUTER JOIN       Skl_naud_grup INNER JOIN
                      Naudotoju_grupes ON Skl_naud_grup.naud_id =
Naudotoju_grupes.naud_id ON
                      Geo_objektai.skl_id = Skl_naud_grup.skl_id LEFT OUTER
JOIN            Sklypu_taksatoriai INNER JOIN
                      Darbuotojai ON Sklypu_taksatoriai.darb_id =
Darbuotojai.darb_id ON
                      Geo_objektai.skl_id = Sklypu_taksatoriai.skl_id LEFT
OUTER JOIN       Sklypai_saug_teritorijos INNER JOIN
                      Saug_teritorijos ON Sklypai_saug_teritorijos.st_id =
Saug_teritorijos.st_id LEFT OUTER JOIN                  Funkcines_zonos ON
Sklypai_saug_teritorijos.fz_id = Funkcines_zonos.fz_id LEFT OUTER JOIN
Funkcines_pazones ON Sklypai_saug_teritorijos.fp_id =
Funkcines_pazones.fp_id ON
                      Geo_objektai.skl_id = Sklypai_saug_teritorijos.skl_id
LEFT OUTER JOIN
                      Rysiai_su_raj INNER JOIN
                      Rajonai ON Rysiai_su_raj.admr_id = Rajonai.admr_id ON
Geo_objektai.skl_id = Rysiai_su_raj.skl_id LEFT OUTER JOIN
Uredijos INNER JOIN
                      Kv_skl ON dbo.Uredijos.mu_id = Kv_skl.mu_id INNER
JOIN            Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN
Gir_kvar ON Kvartalai.kv_id = Gir_kvar.kv_id INNER JOIN            Ur_gir
ON Kv_skl.ur_gir_id = Ur_gir.ur_gir_id ON Geo_objektai.skl_id =
Kv_skl.skl_id LEFT OUTER JOIN
                      Rysiai_su_viet ON Geo_objektai.skl_id =
Rysiai_su_viet.skl_id  JOIN Ivykiai1 ON Geo_objektai.pr_id =
Ivykiai1.iv1_id
WHERE (Uredijos.mu_kod =1 AND Ur_gir.gir_kod=3 AND Gir_kvar.kv_nr =173)
```

C1 užklausa:

```
SELECT     Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod
FROM         Geo_objektai LEFT OUTER JOIN
                      Uredijos INNER JOIN
                      Kv_skl ON Uredijos.mu_id = Kv_skl.mu_id INNER JOIN
Ur_gir ON Kv_skl.ur_gir_id = Ur_gir.ur_gir_id INNER JOIN
Kvartalai INNER JOIN Gir_kvar ON Kvartalai.kv_id = Gir_kvar.kv_id ON
Kv_skl.kv_id = Kvartalai.kv_id ON Geo_objektai.skl_id = Kv_skl.skl_id
WHERE (Uredijos.mu_kod =1 AND Ur_gir.gir_kod=3 AND Gir_kvar.kv_nr =173)
```

A2 užklausa:

```
SELECT   top 100 percent [m#0.UR],[m#0.GIR],[m#0.KV],[m#1.SKNR],[m#1.SD]
FROM  metadbgenAntraste LEFT OUTER JOIN  metadbgenMaketas1 ON
[m#0.ID]=[m#1.ID] WHERE ([m#0.ur]=1 AND [m#0.gir]=3 AND [m#0.kv]=173)
```

B2 užklausa:

```
SELECT  TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Kv_skl.skl_dalis
                      FROM  Sklypai LEFT OUTER JOIN
                      Augavietes ON Sklypai.dtg_id = Augavietes.dtg_id FULL
OUTER JOIN
                      Medziu_rusys RIGHT OUTER JOIN
                      Maketas1 ON dbo.Medziu_rusys.mr_id = Maketas1.mr_id
ON Sklypai.skl_id = Maketas1.skl_id FULL OUTER JOIN
                      Geo_objektai LEFT OUTER JOIN
                      Misku_kategorijos INNER JOIN
                      Sklypu_misku_kat ON dbo.Misku_kategorijos.mk_id =
dbo.Sklypu_misku_kat.mk_id ON
                      Geo_objektai.skl_id = Sklypu_misku_kat.skl_id LEFT
OUTER JOIN       Uredijos INNER JOIN
                      Kv_skl ON dbo.Uredijos.mu_id = Kv_skl.mu_id INNER
JOIN            Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN
Gir_kvar ON Kvartalai.kv_id = Gir_kvar.kv_id INNER JOIN            Ur_gir
ON Kv_skl.ur_gir_id = Ur_gir.ur_gir_id ON Geo_objektai.skl_id =
Kv_skl.skl_id LEFT OUTER JOIN
                      Zemes_naudm_kateg ON Geo_objektai.zk_id =
Zemes_naudm_kateg.zk_id ON Sklypai.skl_id = Geo_objektai.skl_id
   WHERE (Uredijos.mu_kod =1 AND Ur_gir.gir_kod=3 AND Gir_kvar.kv_nr =173)
```

C2 užklausa:

```
SELECT     TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Kv_skl.skl_dalis
FROM         Geo_objektai LEFT OUTER JOIN
                      Uredijos INNER JOIN
                      Kv_skl ON Uredijos.mu_id = Kv_skl.mu_id INNER JOIN
Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN            Gir_kvar
ON Kvartalai.kv_id = Gir_kvar.kv_id INNER JOIN            Ur_gir ON
Kv_skl.ur_gir_id = Ur_gir.ur_gir_id ON Geo_objektai.skl_id = Kv_skl.skl_id
WHERE     (Uredijos.mu_kod = 1) AND (Ur_gir.gir_kod = 3) AND
(Gir_kvar.kv_nr = 173)
```

A3 užklausa:

```
SELECT   top 100 percent
[m#0.UR],[m#0.GIR],[m#0.KV],[m#1.SKNR],[m#1.SN],[m#0.ID],[m#10.D],[m#10.H]
FROM  metadbgenAntraste LEFT OUTER JOIN  metadbgenMaketas1 ON
[m#0.ID]=[m#1.ID]  LEFT OUTER JOIN  metadbgenMaketas10 ON
[m#0.ID]=[m#10.ID] WHERE ([m#0.ur]=1 AND [m#0.gir]=3 AND [m#0.kv]=173)
```

B3 užklausa:

```
SELECT     TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Geo_objektai.sn, Geo_objektai.skl_id, Ardu_sudetis.d,
Ardu_sudetis.h
FROM         Maketas1 FULL OUTER JOIN
                      Maketas10 LEFT OUTER JOIN
                      Medziu_rusys INNER JOIN
                      Ardu_sudetis ON Medziu_rusys.mr_id =
Ardu_sudetis.mr_id ON Maketas10.m10_id = Ardu_sudetis.m10_id RIGHT OUTER
JOIN            Augavietes RIGHT OUTER JOIN
                      Sklypai ON Augavietes.dtg_id = Sklypai.dtg_id ON
Maketas10.skl_id = Sklypai.skl_id ON Maketas1.mr_id = Medziu_rusys.mr_id
AND Maketas1.skl_id = Sklypai.skl_id FULL OUTER JOIN
Geo_objektai LEFT OUTER JOIN
                      Misku_kategorijos INNER JOIN
                      Sklypu_misku_kat ON Misku_kategorijos.mk_id =
Sklypu_misku_kat.mk_id ON Geo_objektai.skl_id = Sklypu_misku_kat.skl_id
LEFT OUTER JOIN
                      Uredijos INNER JOIN
                      Kv_skl ON Uredijos.mu_id = Kv_skl.mu_id INNER JOIN
Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN            Gir_kvar
ON Kvartalai.kv_id = Gir_kvar.kv_id INNER JOIN            Ur_gir ON
Kv_skl.ur_gir_id = Ur_gir.ur_gir_id ON Geo_objektai.skl_id = Kv_skl.skl_id
LEFT OUTER JOIN
                      Zemes_naudm_kateg ON Geo_objektai.zk_id =
Zemes_naudm_kateg.zk_id ON Sklypai.skl_id = Geo_objektai.skl_id
WHERE     (Uredijos.mu_kod = 1) AND (Ur_gir.gir_kod = 3) AND
(Gir_kvar.kv_nr = 173)
```

C3 užklausa:

```
SELECT     TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Geo_objektai.sn, Geo_objektai.skl_id, Ardu_sudetis.d,
Ardu_sudetis.h
FROM         Ardu_sudetis RIGHT OUTER JOIN
                      Maketas10 ON Ardu_sudetis.m10_id = Maketas10.m10_id
RIGHT OUTER JOIN
                      Sklypai ON Maketas10.skl_id = Sklypai.skl_id FULL
OUTER JOIN       Geo_objektai LEFT OUTER JOIN
                      Uredijos INNER JOIN
                      Kv_skl ON Uredijos.mu_id = Kv_skl.mu_id INNER JOIN
Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN            Gir_kvar
ON Kvartalai.kv_id = Gir_kvar.kv_id INNER JOIN            Ur_gir ON
Kv_skl.ur_gir_id = Ur_gir.ur_gir_id ON Geo_objektai.skl_id = Kv_skl.skl_id
ON Sklypai.skl_id = Geo_objektai.skl_id
WHERE     (Uredijos.mu_kod = 1) AND (Ur_gir.gir_kod = 3) AND
(Gir_kvar.kv_nr = 173)
```

A4 užklausa:

```
SELECT   top 100 percent
[m#0.UR],[m#0.GIR],[m#0.KV],[m#1.SKNR],[m#1.SN],[m#0.ID],[m#10.MR],[m#10.SK
AL],[m#13.T2],[m#13.T3],[m#13.T1],[m#14.T1],[m#14.T2],[m#14.T3],[m#14.T4]
FROM  metadbgenAntraste LEFT OUTER JOIN  metadbgenMaketas1 ON
[m#0.ID]=[m#1.ID]  LEFT OUTER JOIN  metadbgenMaketas10 ON
[m#0.ID]=[m#10.ID]  LEFT OUTER JOIN  metadbgenMaketas13 ON
[m#0.ID]=[m#13.ID]  LEFT OUTER JOIN  metadbgenMaketas14 ON
[m#0.ID]=[m#14.ID]  WHERE ([m#0.ur]=1 AND [m#0.gir]=3 AND [m#0.kv]=173)
```

B4 užklausa:

```
SELECT     TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Geo_objektai.sn, Geo_objektai.skl_id, Medziu_rusys.mr_rkod,
Maketas10.skal, Keliu_ypatybes.buk_kod, Keliu_dangos.dan_kod,
```

```
Maketas13.plotis, Augalu_rusys.aug_kod, Maketas14.sutink, Maketas14.tank,
Ukines_vertes_kodai.uk_vert_kod
FROM         Skl_lin_objektai RIGHT OUTER JOIN
                      Zemes_naudm_kateg RIGHT OUTER JOIN
                      Geo_objektai ON Zemes_naudm_kateg.zk_id =
Geo_objektai.zk_id LEFT OUTER JOIN
                      Sklypu_misku_kat INNER JOIN
                      Misku_kategorijos ON Sklypu_misku_kat.mk_id =
Misku_kategorijos.mk_id ON Geo_objektai.skl_id = Sklypu_misku_kat.skl_id
LEFT OUTER JOIN
                      Ur_gir INNER JOIN
                      Kv_skl ON Ur_gir.ur_gir_id = Kv_skl.ur_gir_id INNER
JOIN
                      Uredijos ON Kv_skl.mu_id = Uredijos.mu_id INNER JOIN
Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN            Gir_kvar
ON Kvartalai.kv_id = Gir_kvar.kv_id ON Geo_objektai.skl_id = Kv_skl.skl_id
LEFT OUTER JOIN
                      Maketas10 RIGHT OUTER JOIN
                      Augavietes RIGHT OUTER JOIN
                      Sklypai ON Augavietes.dtg_id = Sklypai.dtg_id ON
Maketas10.skl_id = Sklypai.skl_id LEFT OUTER JOIN
                      Maketas14 ON Sklypai.skl_id = Maketas14.skl_id ON
Geo_objektai.skl_id = Sklypai.skl_id ON
                      Skl_lin_objektai.skl_id = Geo_objektai.skl_id LEFT
OUTER JOIN       Augalu_rusys ON Maketas14.aug_id = Augalu_rusys.aug_id
LEFT OUTER JOIN
                      Ukines_vertes_kodai ON Maketas14.uk_vert_id =
Ukines_vertes_kodai.uk_vert_id LEFT OUTER JOIN
                      Maketas13 LEFT OUTER JOIN
                      Keliu_ypatybes ON Maketas13.buk_id =
Keliu_ypatybes.buk_id LEFT OUTER JOIN
                      Keliu_dangos ON Maketas13.dan_id =
Keliu_dangos.dan_id ON Skl_lin_objektai.skl_id = Maketas13.skl_id LEFT
OUTER JOIN       Ardu_sudetis ON Maketas10.m10_id = Ardu_sudetis.m10_id
LEFT OUTER JOIN
                      Maketas1 LEFT OUTER JOIN
                      Medziu_rusys ON Maketas1.mr_id = Medziu_rusys.mr_id
ON Ardu_sudetis.mr_id = Medziu_rusys.mr_id AND Sklypai.skl_id =
Maketas1.skl_id
WHERE     (Uredijos.mu_kod = 1) AND (Ur_gir.gir_kod = 3) AND
(Gir_kvar.kv_nr = 173)
```

C4 užklausa:

```
SELECT     TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Geo_objektai.sn, Geo_objektai.skl_id, Medziu_rusys.mr_rkod,
                      Maketas10.skal, Keliu_ypatybes.buk_kod,
Keliu_dangos.dan_kod, Maketas13.plotis, Augalu_rusys.aug_kod,
Maketas14.sutink, Maketas14.tank,
                      Ukines_vertes_kodai.uk_vert_kod
FROM         Skl_lin_objektai RIGHT OUTER JOIN
                      Ur_gir INNER JOIN
                      Kv_skl ON Ur_gir.ur_gir_id = Kv_skl.ur_gir_id INNER
JOIN            Uredijos ON Kv_skl.mu_id = Uredijos.mu_id INNER JOIN
Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN            Gir_kvar
ON Kvartalai.kv_id = Gir_kvar.kv_id RIGHT OUTER JOIN      Geo_objektai ON
Kv_skl.skl_id = Geo_objektai.skl_id LEFT OUTER JOIN
                      Medziu_rusys RIGHT OUTER JOIN
                      Maketas10 RIGHT OUTER JOIN
                      Sklypai ON Maketas10.skl_id = Sklypai.skl_id LEFT
OUTER JOIN       Ardu_sudetis ON Maketas10.m10_id = Ardu_sudetis.m10_id ON
Medziu_rusys.mr_id = Ardu_sudetis.mr_id LEFT OUTER JOIN
Maketas14 ON Sklypai.skl_id = Maketas14.skl_id ON Geo_objektai.skl_id =
Sklypai.skl_id ON
```

```
                     Skl_lin_objektai.skl_id = Geo_objektai.skl_id LEFT
OUTER JOIN       Augalu_rusys ON Maketas14.aug_id = Augalu_rusys.aug_id
LEFT OUTER JOIN
                     Ukines_vertes_kodai ON Maketas14.uk_vert_id =
Ukines_vertes_kodai.uk_vert_id LEFT OUTER JOIN
                     Maketas13 LEFT OUTER JOIN
                     Keliu_ypatybes ON Maketas13.buk_id =
Keliu_ypatybes.buk_id LEFT OUTER JOIN
                     Keliu_dangos ON Maketas13.dan_id =
Keliu_dangos.dan_id ON Skl_lin_objektai.skl_id = Maketas13.skl_id WHERE
(Uredijos.mu_kod = 1 AND Ur_gir.gir_kod = 3 AND Gir_kvar.kv_nr = 173)
```

D užklausa:

```
SELECT     TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Geo_objektai.sn, Geo_objektai.skl_id, Medziu_rusys.mr_rkod,
Maketas10.skal, Keliu_ypatybes.buk_kod, Keliu_dangos.dan_kod,
Maketas13.plotis, Augalu_rusys.aug_kod, Maketas14.sutink, Maketas14.tank,
Ukines_vertes_kodai.uk_vert_kod
FROM         Maketas10 RIGHT OUTER JOIN
                     Augavietes RIGHT OUTER JOIN
                     Sklypai ON Augavietes.dtg_id = Sklypai.dtg_id ON
Maketas10.skl_id = Sklypai.skl_id LEFT OUTER JOIN
                     Maketas14 ON Sklypai.skl_id = Maketas14.skl_id RIGHT
OUTER JOIN      Ivykiai1 INNER JOIN
                     Ivykiu1_tipai ON Ivykiai1.iv1_tip_id =
Ivykiu1_tipai.iv1_tip_id RIGHT OUTER JOIN
                     Ivykiai2 INNER JOIN
                     Ivykiu2_tipai ON Ivykiai2.iv2_tip_id =
Ivykiu2_tipai.iv2_tip_id RIGHT OUTER JOIN
                     Geo_objektai ON Ivykiai2.iv2_id =
Geo_objektai.keit_id LEFT OUTER JOIN
                     Ur_gir INNER JOIN
                     Kv_skl ON Ur_gir.ur_gir_id = Kv_skl.ur_gir_id INNER
JOIN            Uredijos ON Kv_skl.mu_id = Uredijos.mu_id INNER JOIN
Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN            Gir_kvar
ON Kvartalai.kv_id = Gir_kvar.kv_id ON Geo_objektai.skl_id = Kv_skl.skl_id
ON Ivykiai1.iv1_id = Geo_objektai.pr_id LEFT OUTER JOIN
                     Zemes_naudm_kateg ON Geo_objektai.zk_id =
Zemes_naudm_kateg.zk_id LEFT OUTER JOIN
                     Sklypu_misku_kat INNER JOIN
                     Misku_kategorijos ON Sklypu_misku_kat.mk_id =
Misku_kategorijos.mk_id ON Geo_objektai.skl_id = Sklypu_misku_kat.skl_id ON
                     Sklypai.skl_id = Geo_objektai.skl_id LEFT OUTER JOIN
Skl_lin_objektai ON Geo_objektai.skl_id = Skl_lin_objektai.skl_id LEFT
OUTER JOIN
                     Augalu_rusys ON Maketas14.aug_id =
Augalu_rusys.aug_id LEFT OUTER JOIN
                     Ukines_vertes_kodai ON Maketas14.uk_vert_id =
Ukines_vertes_kodai.uk_vert_id LEFT OUTER JOIN
                     Keliu_dangos RIGHT OUTER JOIN
                     Maketas13 LEFT OUTER JOIN
                     Keliu_kategorijos ON Maketas13.kkat_id =
Keliu_kategorijos.kkat_id LEFT OUTER JOIN
                     Keliu_ypatybes ON Maketas13.buk_id =
Keliu_ypatybes.buk_id ON Keliu_dangos.dan_id = Maketas13.dan_id ON
                     Skl_lin_objektai.skl_id = Maketas13.skl_id LEFT OUTER
JOIN           Ardu_sudetis ON Maketas10.m10_id = Ardu_sudetis.m10_id LEFT
OUTER JOIN
                     Maketas1 LEFT OUTER JOIN
                     Medziu_rusys ON Maketas1.mr_id = Medziu_rusys.mr_id
ON Ardu_sudetis.mr_id = Medziu_rusys.mr_id AND Sklypai.skl_id =
Maketas1.skl_id
```

```
WHERE     (Uredijos.mu_kod = 1) AND (Ur_gir.gir_kod = 3) AND
(Gir_kvar.kv_nr = 173)
```

E užklausa:

```
SELECT     TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Geo_objektai.sn, Geo_objektai.skl_id, Medziu_rusys.mr_rkod,
Maketas10.skal, Keliu_ypatybes.buk_kod, Keliu_dangos.dan_kod,
Maketas13.plotis, Augalu_rusys.aug_kod, Maketas14.sutink, Maketas14.tank,
                      Ukines_vertes_kodai.uk_vert_kod
FROM         Ukines_priemones RIGHT OUTER JOIN
                      Maketas1 LEFT OUTER JOIN
                      Misku_tipai ON Maketas1.mtip_id = Misku_tipai.mtip_id
ON Ukines_priemones.up_id = Maketas1.up_id LEFT OUTER JOIN
      Medziu_rusys ON Maketas1.mr_id = Medziu_rusys.mr_id RIGHT OUTER JOIN
                      Maketas10 RIGHT OUTER JOIN
                      Augavietes RIGHT OUTER JOIN
                      Sklypai ON Augavietes.dtg_id = Sklypai.dtg_id ON
Maketas10.skl_id = Sklypai.skl_id LEFT OUTER JOIN
                      Maketas14 ON Sklypai.skl_id = Maketas14.skl_id RIGHT
OUTER JOIN      Ivykiai1 INNER JOIN
                      Ivykiu1_tipai ON Ivykiai1.iv1_tip_id =
Ivykiu1_tipai.iv1_tip_id RIGHT OUTER JOIN
                      Ivykiai2 INNER JOIN
                      Ivykiu2_tipai ON Ivykiai2.iv2_tip_id =
Ivykiu2_tipai.iv2_tip_id RIGHT OUTER JOIN
                      Geo_objektai ON Ivykiai2.iv2_id =
Geo_objektai.keit_id LEFT OUTER JOIN
                      Ur_gir INNER JOIN
                      Kv_skl ON Ur_gir.ur_gir_id = Kv_skl.ur_gir_id INNER
JOIN            Uredijos ON Kv_skl.mu_id = Uredijos.mu_id INNER JOIN
Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN            Gir_kvar
ON Kvartalai.kv_id = Gir_kvar.kv_id ON Geo_objektai.skl_id = Kv_skl.skl_id
ON Ivykiai1.iv1_id = Geo_objektai.pr_id LEFT OUTER JOIN
                      Zemes_naudm_kateg ON Geo_objektai.zk_id =
Zemes_naudm_kateg.zk_id LEFT OUTER JOIN
                      Sklypu_misku_kat INNER JOIN
                      Misku_kategorijos ON Sklypu_misku_kat.mk_id =
Misku_kategorijos.mk_id ON Geo_objektai.skl_id = Sklypu_misku_kat.skl_id ON
                      Sklypai.skl_id = Geo_objektai.skl_id LEFT OUTER JOIN
Skl_lin_objektai ON Geo_objektai.skl_id = Skl_lin_objektai.skl_id LEFT
OUTER JOIN
                      Augalu_rusys ON Maketas14.aug_id =
Augalu_rusys.aug_id LEFT OUTER JOIN
                      Ukines_vertes_kodai ON Maketas14.uk_vert_id =
Ukines_vertes_kodai.uk_vert_id LEFT OUTER JOIN
                      Keliu_dangos RIGHT OUTER JOIN
                      Maketas13 LEFT OUTER JOIN
                      Keliu_kategorijos ON Maketas13.kkat_id =
Keliu_kategorijos.kkat_id LEFT OUTER JOIN
                      Keliu_ypatybes ON Maketas13.buk_id =
Keliu_ypatybes.buk_id ON Keliu_dangos.dan_id = Maketas13.dan_id ON
                      Skl_lin_objektai.skl_id = Maketas13.skl_id LEFT OUTER
JOIN            Ardu_sudetis ON Maketas10.m10_id = Ardu_sudetis.m10_id ON
Medziu_rusys.mr_id = Ardu_sudetis.mr_id AND
                      Maketas1.skl_id = Sklypai.skl_id
WHERE     (Uredijos.mu_kod = 1) AND (Ur_gir.gir_kod = 3) AND
(Gir_kvar.kv_nr = 173)
```

F užklausa:

```
SELECT     TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Geo_objektai.sn, Geo_objektai.skl_id, Medziu_rusys.mr_rkod,
```

```
Maketas10.skal, Keliu_ypatybes.buk_kod, Keliu_dangos.dan_kod,
Maketas13.plotis, Augalu_rusys.aug_kod, Maketas14.sutink, Maketas14.tank,
Ukines_vertes_kodai.uk_vert_kod
FROM         Zemes_naudm_kateg RIGHT OUTER JOIN
                      Geo_objektai LEFT OUTER JOIN
                      Skl_naud_grup INNER JOIN
                      Naudotoju_grupes ON Skl_naud_grup.naud_id =
Naudotoju_grupes.naud_id ON Geo_objektai.skl_id = Skl_naud_grup.skl_id LEFT
OUTER JOIN
                      Funkcines_zonos RIGHT OUTER JOIN
                      Sklypai_saug_teritorijos INNER JOIN
                      Saug_teritorijos ON Sklypai_saug_teritorijos.st_id =
Saug_teritorijos.st_id ON
                      Funkcines_zonos.fz_id =
Sklypai_saug_teritorijos.fz_id LEFT OUTER JOIN
                      Funkcines_pazones ON Sklypai_saug_teritorijos.fp_id =
Funkcines_pazones.fp_id ON
                      Geo_objektai.skl_id = Sklypai_saug_teritorijos.skl_id
LEFT OUTER JOIN
                      Rysiai_su_raj INNER JOIN
                      Rajonai ON Rysiai_su_raj.admr_id = Rajonai.admr_id ON
Geo_objektai.skl_id = Rysiai_su_raj.skl_id LEFT OUTER JOIN         Ivykiai2
INNER JOIN
                      Ivykiu2_tipai ON Ivykiai2.iv2_tip_id =
Ivykiu2_tipai.iv2_tip_id ON Geo_objektai.keit_id = Ivykiai2.iv2_id LEFT
OUTER JOIN    Ur_gir INNER JOIN
                      Kv_skl ON Ur_gir.ur_gir_id = Kv_skl.ur_gir_id INNER
JOIN             Uredijos ON Kv_skl.mu_id = Uredijos.mu_id INNER JOIN
Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN             Gir_kvar
ON Kvartalai.kv_id = Gir_kvar.kv_id ON Geo_objektai.skl_id = Kv_skl.skl_id
LEFT OUTER JOIN
                      Ivykiai1 INNER JOIN
                      Ivykiu1_tipai ON Ivykiai1.iv1_tip_id =
Ivykiu1_tipai.iv1_tip_id ON Geo_objektai.pr_id = Ivykiai1.iv1_id ON
                      Zemes_naudm_kateg.zk_id = Geo_objektai.zk_id LEFT
OUTER JOIN       Sklypu_misku_kat INNER JOIN
                      Misku_kategorijos ON Sklypu_misku_kat.mk_id =
Misku_kategorijos.mk_id ON Geo_objektai.skl_id = Sklypu_misku_kat.skl_id
LEFT OUTER JOIN
                      Maketas10 RIGHT OUTER JOIN
                      Augavietes RIGHT OUTER JOIN
                      Sklypai ON Augavietes.dtg_id = Sklypai.dtg_id ON
Maketas10.skl_id = Sklypai.skl_id LEFT OUTER JOIN
                      Maketas14 ON Sklypai.skl_id = Maketas14.skl_id ON
Geo_objektai.skl_id = Sklypai.skl_id LEFT OUTER JOIN
                      Skl_lin_objektai ON Geo_objektai.skl_id =
Skl_lin_objektai.skl_id LEFT OUTER JOIN
                      Augalu_rusys ON Maketas14.aug_id =
Augalu_rusys.aug_id LEFT OUTER JOIN
                      Ukines_vertes_kodai ON Maketas14.uk_vert_id =
Ukines_vertes_kodai.uk_vert_id LEFT OUTER JOIN
                      Keliu_dangos RIGHT OUTER JOIN
                      Maketas13 LEFT OUTER JOIN
                      Keliu_kategorijos ON Maketas13.kkat_id =
Keliu_kategorijos.kkat_id LEFT OUTER JOIN
                      Keliu_ypatybes ON Maketas13.buk_id =
Keliu_ypatybes.buk_id ON Keliu_dangos.dan_id = Maketas13.dan_id ON
                      Skl_lin_objektai.skl_id = Maketas13.skl_id LEFT OUTER
JOIN            Ardu_sudetis ON Maketas10.m10_id = Ardu_sudetis.m10_id LEFT
OUTER JOIN
                      Ukines_priemones RIGHT OUTER JOIN
                      Maketas1 LEFT OUTER JOIN
                      Misku_tipai ON Maketas1.mtip_id = Misku_tipai.mtip_id
ON Ukines_priemones.up_id = Maketas1.up_id LEFT OUTER JOIN
```

```
Medziu_rusys ON Maketas1.mr_id = Medziu_rusys.mr_id ON Ardu_sudetis.mr_id =
Medziu_rusys.mr_id AND Sklypai.skl_id = Maketas1.skl_id
WHERE     (Uredijos.mu_kod = 1) AND (Ur_gir.gir_kod = 3) AND
(Gir_kvar.kv_nr = 173)
```

G užklausa:

```
SELECT     TOP 100 PERCENT Uredijos.mu_kod, Gir_kvar.kv_nr, Ur_gir.gir_kod,
Kv_skl.skl_nr, Geo_objektai.sn, Geo_objektai.skl_id, Medziu_rusys.mr_rkod,
Maketas10.skal, Keliu_ypatybes.buk_kod, Keliu_dangos.dan_kod,
Maketas13.plotis, Augalu_rusys.aug_kod, Maketas14.sutink, Maketas14.tank,
Ukines_vertes_kodai.uk_vert_kod
FROM         Ivykiai2 INNER JOIN
                      Ivykiu2_tipai ON Ivykiai2.iv2_tip_id =
Ivykiu2_tipai.iv2_tip_id RIGHT OUTER JOIN
                      Geo_objektai LEFT OUTER JOIN
                      Sklypu_taksatoriai INNER JOIN
                      Darbuotojai ON Sklypu_taksatoriai.darb_id =
Darbuotojai.darb_id ON Geo_objektai.skl_id = Sklypu_taksatoriai.skl_id LEFT
OUTER JOIN
                      Ivykiai1 INNER JOIN
                      Ivykiu1_tipai ON Ivykiai1.iv1_tip_id =
Ivykiu1_tipai.iv1_tip_id ON Geo_objektai.pr_id = Ivykiai1.iv1_id ON
                      Ivykiai2.iv2_id = Geo_objektai.keit_id LEFT OUTER
JOIN            Bloku_sklypai INNER JOIN
                      Kad_sklypai ON Bloku_sklypai.kskl_id =
Kad_sklypai.kskl_id INNER JOIN
                      Rysiai_su_ks ON Kad_sklypai.kskl_id =
Rysiai_su_ks.kskl_id ON Geo_objektai.skl_id = Rysiai_su_ks.skl_id LEFT
OUTER JOIN       Ur_gir INNER JOIN
                      Kv_skl ON Ur_gir.ur_gir_id = Kv_skl.ur_gir_id INNER
JOIN            Uredijos ON Kv_skl.mu_id = Uredijos.mu_id INNER JOIN
                Kvartalai ON Kv_skl.kv_id = Kvartalai.kv_id INNER JOIN
Gir_kvar ON Kvartalai.kv_id = Gir_kvar.kv_id ON Geo_objektai.skl_id =
Kv_skl.skl_id LEFT OUTER JOIN
                      Kad_blokai INNER JOIN
                      Rysiai_su_blokais ON Kad_blokai.kdb_id =
Rysiai_su_blokais.kdb_id INNER JOIN
                      Vietoves_blokai ON Kad_blokai.kdb_id =
Vietoves_blokai.kdb_id INNER JOIN
                      Vietoves ON Vietoves_blokai.kdv_id = Vietoves.kdv_id
ON Geo_objektai.skl_id = Rysiai_su_blokais.skl_id LEFT OUTER JOIN
Rysiai_su_raj INNER JOIN
                      Rajonai ON Rysiai_su_raj.admr_id = Rajonai.admr_id ON
Geo_objektai.skl_id = Rysiai_su_raj.skl_id LEFT OUTER JOIN
Skl_naud_grup INNER JOIN
                      Naudotoju_grupes ON Skl_naud_grup.naud_id =
Naudotoju_grupes.naud_id ON Geo_objektai.skl_id = Skl_naud_grup.skl_id LEFT
OUTER JOIN
                      Funkcines_zonos RIGHT OUTER JOIN
                      Sklypai_saug_teritorijos INNER JOIN
                      Saug_teritorijos ON Sklypai_saug_teritorijos.st_id =
Saug_teritorijos.st_id ON
                      Funkcines_zonos.fz_id =
Sklypai_saug_teritorijos.fz_id LEFT OUTER JOIN
                      Funkcines_pazones ON Sklypai_saug_teritorijos.fp_id =
Funkcines_pazones.fp_id ON
                      Geo_objektai.skl_id = Sklypai_saug_teritorijos.skl_id
LEFT OUTER JOIN
                      Zemes_naudm_kateg ON Geo_objektai.zk_id =
Zemes_naudm_kateg.zk_id LEFT OUTER JOIN
                      Sklypu_misku_kat INNER JOIN
```

```
                      Misku_kategorijos ON Sklypu_misku_kat.mk_id =
Misku_kategorijos.mk_id ON Geo_objektai.skl_id = Sklypu_misku_kat.skl_id
LEFT OUTER JOIN
                      Maketas10 RIGHT OUTER JOIN
                      Augavietes RIGHT OUTER JOIN
                      Sklypai ON Augavietes.dtg_id = Sklypai.dtg_id ON
Maketas10.skl_id = Sklypai.skl_id LEFT OUTER JOIN
                      Maketas14 ON Sklypai.skl_id = Maketas14.skl_id ON
Geo_objektai.skl_id = Sklypai.skl_id LEFT OUTER JOIN
                      Skl_lin_objektai ON Geo_objektai.skl_id =
Skl_lin_objektai.skl_id LEFT OUTER JOIN
                      Augalu_rusys ON Maketas14.aug_id =
Augalu_rusys.aug_id LEFT OUTER JOIN
                      Ukines_vertes_kodai ON Maketas14.uk_vert_id =
Ukines_vertes_kodai.uk_vert_id LEFT OUTER JOIN
                      Keliu_dangos RIGHT OUTER JOIN
                      Maketas13 LEFT OUTER JOIN
                      Keliu_kategorijos ON Maketas13.kkat_id =
Keliu_kategorijos.kkat_id LEFT OUTER JOIN
                      Keliu_ypatybes ON Maketas13.buk_id =
Keliu_ypatybes.buk_id ON Keliu_dangos.dan_id = Maketas13.dan_id ON
                      Skl_lin_objektai.skl_id = Maketas13.skl_id LEFT OUTER
JOIN     Ardu_sudetis ON Maketas10.m10_id = Ardu_sudetis.m10_id LEFT OUTER
JOIN       Ukines_priemones RIGHT OUTER JOIN
                      Maketas1 LEFT OUTER JOIN
                      Misku_tipai ON Maketas1.mtip_id = Misku_tipai.mtip_id
ON Ukines_priemones.up_id = Maketas1.up_id LEFT OUTER JOIN
Medziu_rusys ON Maketas1.mr_id = Medziu_rusys.mr_id ON Ardu_sudetis.mr_id =
Medziu_rusys.mr_id AND Sklypai.skl_id = Maketas1.skl_id
WHERE     (Uredijos.mu_kod = 1) AND (Ur_gir.gir_kod = 3) AND
(Gir_kvar.kv_nr = 173)
```

8.2. Straipsnis

DUOMENŲ FILTRAVIMO IR ATRANKOS SPRENDIMŲ ANALIZĖ

Rūta Vairaitė
Rimantas Butleris
Kauno technologijos universitetas, Informacijos sistemų katedra, Studentų g. 50, Kaunas

Esant dideliems saugomų duomenų kiekiams, yra svarbus našus jų apdorojimas, taigi, vartotojams reikia vis didesnio duomenų bazių našumo. Pasitelkiant duomenų bazių spartos derinimą, ar duomenų bazių spartos optimizavimą, galima paskatinti duomenų bazes veikti greičiau. Išnagrinėjus duomenų bazių esamus spartinimo metodus ir priežastis, kurios mažina spartumą, yra siūlomas metodas, kuris leidžia sparčiau apdoroti duomenis ir greičiau pateikti vartotojui užklausos rezultatą. Siūlomas metodas yra patikrintas eksperimentu.

1 Įvadas

Organizacijose nuolat augant kaupiamų duomenų kiekiams, yra tikimasi jų spartaus apdorojimo. Be to, vartotojų reikalavimai ir lūkesčiai pastoviai auga, taigi, duomenų bazių našumo didinimas filtruojant ir atrenkant duomenis tampa vis svarbesnis. Duomenų bazių spartos derinimas ir duomenų bazių spartos optimizavimas yra vienos iš tų veiklų, kurios skatina duomenų bazes veikti greičiau. SQL optimizavimas siekia patobulinti užklausas taikymo lygmenyje ir turi didelį potencialą pagerinant duomenų bazių veikimą Duomenų bazių derinimas reikalauja įgūdžių ir susideda iš sistemos administravimo, duomenų bazių administravimo ir taikymo plėtojimo. O viena iš pirmųjų duomenų bazių derinimo užduočių yra suprasti problemų priežastis ir rasti esamas kliūtis. Yra nustatyta, kad SQL derinimas turi didžiausią įtaką našumui, kuri gali siekti daugiau negu 50% [1].

Pirmoji faktorių kategorija, susijusi su nepakankamu duomenų bazių našumu, gali būti sudaryta remiantis technine įranga (procesorius, atmintis, disko ir tinklo darbas). Antroji kategorija yra labiau susijusi su duomenų bazių sistemos charakteristikomis, (DB schemos kokybė, indeksavimas, suskirstymas ar blokavimas). Trečioji faktorių kategorija siejasi su problemomis, iškylančiomis taikomajame lygyje. Toliau aptarsime antrosios ir trečiosios kategorijos faktorių įtaką.

2 Problemos formulavimas ir egzistuojantys siūlymai

Sprendžiama problema susijusi su duomenų atrinkimo moduliu, kai duomenys atrenkami pagal iš anksto apibrėžtas taisykles, kurios yra aprašomos atskiroje duomenų bazėje. Taigi, problemą sudaro duomenų bazių, kurių lentelės turi didelius kiekius įrašų, naudojimas ir veikimas. Duomenų atrinkimo modulis tam tikrus sistemos objektus transformuoja į duomenų bazės virtualias lenteles (VIEWS), su kuriomis yra formuojamos užklausos ir atrenkami reikalingi duomenys. Tačiau apdorojant lenteles su dideliais kiekiais įrašų, duomenų apdorojimo sparta maža, kuo daugiau įrašų, tuo daugiau laiko sugaištama ir sunaudojama resursų. Todėl reikia metodo, galinčio padidinti spartą, turint duomenų bazę su dideliu kiekiu įrašų.

Kadangi SQL yra deklaratyvi kalba, vartotojas tik turi nurodyti kokių duomenų jis nori, o DBVS užklausų optimizatorius „nusprendžia“, kokiu informacijos išrinkimo keliu naudotis, todėl yra labai svarbu parašyti užklausą taip, kad būtų grąžinti tik reikalaujami stulpeliai ir eilutės [1, 2].

Vienas iš siūlymų, gerinant užklausų įvykdymo spartą, yra atkreipti dėmesį į indeksavimą. Teisingas indeksų parinkimas lentelei, atsižvelgiant į WHERE dėmenį SQL sakiniuose, gali turėti ryškius rezultatus, jų pagalba atrenkant duomenis sunaudojama mažiau sistemos resursų, negu atliekant pilną lentelės peržiūrą. Tačiau yra dar viena pagrindinė taisyklė, jei sudarant užklausą norima išrinkti daugiau nei 20% įrašų iš lentelės, geriau naudoti pilną lentelės peržiūrą nei naudoti indeksus [3].

Kadangi pradinis duomenų atrinkimo modelis naudoja virtualias lenteles, buvo atlikta jų naudojimo veiksmingumo analizė. Virtualios lentelės yra naudingas įrankis, atrenkant ir filtruojant duomenis, jos patogios, lanksčios, suteikia duomenų bazei daugiau saugumo, našumo ir paprastumo naudojantis. Tačiau virtualių lentelių greitis tiesiogiai priklauso nuo užklausos greičio, todėl reikia stebėti pirminius ir išorinius raktus, per kuriuos yra jungiamos lentelės, ar jie turi indeksus, taip pat ir tuos laukus, kuriems taikomos filtravimo sąlygos. Šalia virtualių lentelių pliusų iškyla ir minusai, kartais našumas tik sumažėja, nes DBVS užklausas virtualiai lentelei keičia paprastomis lentelėmis. Taigi, jeigu virtuali lentelė jungia daug lentelių, tai užklausa tampa sudėtingu lentelių sujungimu, ir tam reikia daug kompiuterio resursų. Todėl konkrečiu atveju reikia rinktis tarp patogumo vartotojui filtruojant duomenis ir našumo. Tada galima pasirinkti naudoti indeksuotą virtualią lentelę, virtualiai lentelei sukuriant unikalų grupuotą indeksą. Tinkamai indeksuota virtuali lentelė gali žymiai padidinti darbo spartą, ko negalima būtų pasiekti naudojant vien tik standartinius indeksus. Tačiau šio metodo realizavimas yra pakankamai sunkus, todėl kyla galimybė padidinti disko įvedimo/išvedimo operacijų skaičių, bei netinka naudoti, kai dažnai atnaujinama duomenų bazė.

Kitas būdas sparčiau atrinkti reikiamus duomenis yra dalinai indeksuotos lenteles. Tai naujas indeksuotų virtualių lentelių tipas, kuris kuriamas tik kai kuriems įrašams, pavyzdžiui, labiausiai naudojamiems įrašams; tai sumažina užimamą atminties vietą ir reikia mažiau resursų virtualių lentelių atnaujinimui. Viena ar kelios valdymo lentelės yra susiejamos su virtualia lentele, jose nurodoma, kurie virtualios lentelės įrašai yra

saugomi ir eksploatuojami. Įrašus galima lengvai keisti. Microsoft eksperimentai su MS SQL serveriu rodo, kad lyginant su pilnai indeksuotomis virtualiomis lentelėmis, dalinai indeksuotos virtualios lentelės reikalauja mažiau vietos, jų geresnis veikimas, ir žymiai mažesni išlaikymo kaštai [4]. Neribojant atminties resursų, užklausos įvykdymas su dalinai indeksuotomis virtualiomis lentelėmis pagerėja, jei virtuali lentelė apima didesnę dalį užklausų. Kuo daugiau apimama užklausų, tuo dalinai indeksuotų virtualių lentelių našumas panašėja į pilnai indeksuotas virtualias lenteles. Tačiau, kai yra riboti atminties resursai, tai didesnė dalis dalinai indeksuotų virtualių lentelių yra talpinama atmintyje, dėl to sumažėja disko įvedimo/išvedimo operacijų greitis. Net jei virtuali lentelė pilnai neapima visų užklausų, užklausos vykdymas naudojant dalinai indeksuotas virtualias lenteles gali būti veiksmingesnis, nei naudojant pilnai indeksuotas virtualias lenteles. Tačiau kartais duomenų bazėms netinka dalinai indeksuotos virtualios lentelės, tada tikslinga arba naudoti pilnai indeksuotas virtualias lenteles arba nenaudoti jų iš viso [4, 5].

Kitas svarbus veiksnys, lemiantis užklausos spartą atrenkant duomenis, yra lentelių sujungimai, ypatingai kai jungiama daugiau nei 2 lentelės arba lentelės yra labai didelės. Deja, sujungimai yra duomenų bazių svarbus elementas, todėl jiems reikia skirti daug laiko, kad jie veiktų optimaliai, todėl svarbu atkreipti dėmesį į patarimus sujungimams:

- jei yra kelios lentelės ar daugiau lentelių, kurios dažnai yra sujungiamos, tada stulpeliai, kurie panaudojami sujungimuose, turėtų būti indeksuoti.
- vykdymo spartai padidinti, stulpeliai, naudojami sujungimuose, turėtų būti to paties tipo. Jei yra įmanoma, geriau jei jie skaitmeniai, o ne tekstinio tipo.
- geriau vengti sujunginėti lenteles remiantis stulpeliais, kurie turi kelias unikalias reikšmes, nes SQL serverio optimizatorius sujungimui gali atlikti pilną lentelės peržiūrą, net jeigu egzistuoja indeksas stulpeliams. Optimizavimo tikslais, sujungimai turėtų būti atliekami su tais stulpeliais, kurie turi unikalų indeksą.
- Jeigu reguliariai reikia sujungti 4 ar daugiau lentelių, norint gauti įrašų sąrašą, reikia apgalvoti lentelių denormalizavimą, kad sujungiamų lentelių skaičius sumažėtų. Dažnai, pridedant vieną ar kelis stulpelius nuo vienos lentelės prie kitos, sujungimai gali būti sumažinti.

Dėl didesnės SQL serverio spartos, reikėtų riboti sujungiamų lentelių kiekį, nes kuo daugiau jų sujungiama, tuo ilgiau vykdoma užklausa, bandant rasti geriausią vykdymo planą. 1 lentelėje galima pamatyti, kaip priklauso sujungiamų lentelių skaičius ir galimų sujungimų variantų skaičius [6].

1 lentelė. Lentelių skaičiaus ir sujungimo variantų ryšys [6].

Sujungiamų lentelių skaičius	Galimų sujungimų variantų skaičius
1	1
4	24
7	5040
10	3 628 800
15	1 307 674 368 000
16	20 922 789 888 000 Net jeigu pavyktų per 1 sekundę apskaičiuoti milijoną skirtingų apjungimo variantų, tai surasti geriausią vykdymo planą užtruktų 242 dienas apjungiant 16 lentelių.

3 Metodas

Metodo pagal nagrinėtus reikalavimus įgyvendinimui, taip pat atsižvelgus į duomenų struktūros ypatumus ir pagrindinius veiksnius, lemiančius duomenų apdorojimo spartą, buvo pasiūlytas metodas, kaip filtruoti ir atrinkti reikiamus duomenis nenaudojant virtualių lentelių, o parenkant tinkamą sujungimų variantą pagal atributus ir lenteles, bei kaupiamą statistiką, ir iš anksto apibrėžtas taisykles, kurios yra aprašomos atskiroje meta duomenų bazėje.

3.1 Meta DB struktūra

Metaduomenys yra svarbus kiekvienos sistemos komponentas. Tai yra informacija apibrėžianti duomenų turinį, kokybę, būseną ir kitas charakteristikas, o mūsų atveju tai duomenys, aprašantys pačią DB.

Taigi naudojama meta duomenų bazė, kuri skirta saugoti:

6. Pagrindinės duomenų bazės lentelių ir laukų informaciją;
7. Sistemos objektų (maketų) aprašus;
8. Statinių užklausų specifikacijas;
9. Papildomą informaciją, susijusią su statinių užklausų specifikacijų kūrimu.
10. Lentelių apjungimo šablonų informaciją.
11. Labiausiai naudojamų atributų grupių ir lentelių statistikos informaciją.

Meta duomenų bazę sudaro 10 lentelių (2 lentelė). Jose, be saugomos informacijos, yra apibrėžtos tokios taisyklės, kaip, pavyzdžiui, koks turi būti maketo indeksas, skirtas maketų eiliškumui išsaugoti, jei maketai trinami, įterpiami nauji; kaip susiejamas konkretus lentelės laukas ir maketas; koks tikrasis lauko vardas ir koks

lauko vardas pateikiamas vartotojui; kokia turi būti statinių užklausų struktūra; pagal kokį konkretų lauką grupuoti, rūšiuoti; koks turi būti lauko tikslumas skaičiais po kablelio.

2 lentelė. Meta duomenų bazės lentelių sąrašas.

Lentelės vardas	Paskirtis
LENTELES	Pagrindinės duomenų bazės lentelių sąrašas
LAUKAI	Duomenų bazės lentelių laukų sąrašas
MAKETAI	Pagrindinė informacija apie maketus
MAKLENTELES	Maketus sudarančių lentelių ryšių specifikacija
MLAUKAI	Maketus sudarančių laukų sąrašas
UZKLAUSOS	Pagrindinė informacija apie statines užklausas
ULAUKAI	Statines užklausas sudarančių laukų sąrašas
SABLONAS	Lentelių apjungimo ryšių specifikacija
AFUNKCIJOS	Agregavimo funkcijų sąrašas
STATISTIKA	Užklausų naudojimo informacija

3.2 Lentelių apjungimo šablonai

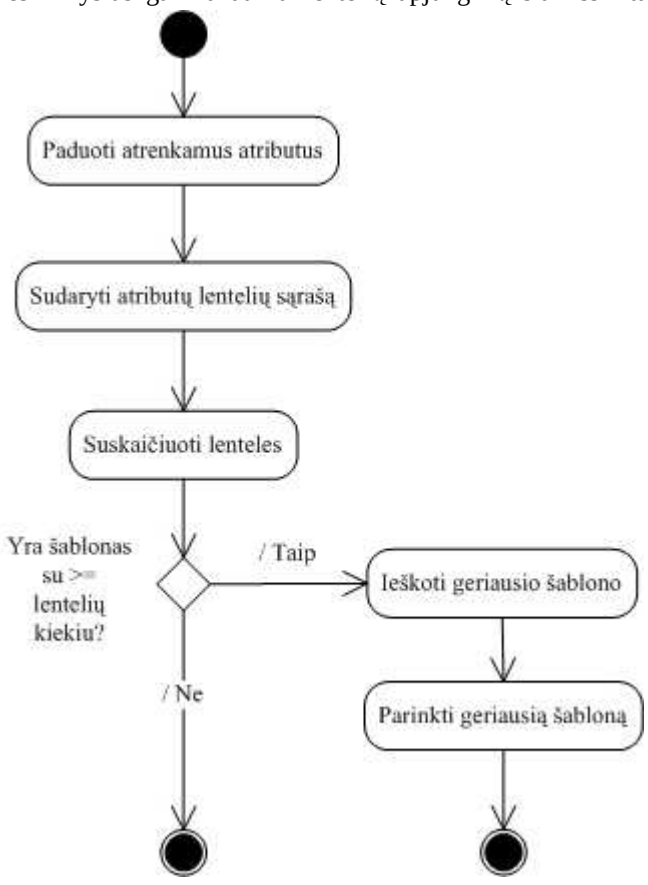
Lentelių apjungimo šablonai gali būti formuojami lentelių grupėms, kurios yra dažniausiai naudojamos užklausose. Vartotojai dažnai naudoja tas pačias užklausas duomenų atrinkimui nekeisdami kriterijų, todėl vieną kartą apibrėžus lentelių apjungimo šabloną galima jį daugelį kartų naudoti, arba pritaikyti kitai užklausai. Kai vykdant užklausas yra naudojami maketai ir jiems sudarytos virtualios lentelės, tai atrenkant tam tikrus atributus gali būti panaudojamos nereikalingos lentelės, ir gali tekti filtruoti didesnį kiekį duomenų, tokiu atveju išauga laiko kiekis, panaudojamas užklausai įvykdyti.
Duomenų atrankos ir filtravimo kriterijai, kuriuos dažniausiai vartotojai užklausos projekte nurodo, yra surenkami ir išanalizuojami. Taip pagal atrenkamus atributus gaunamas sąrašas tų lentelių, kurios naudojamos užklausoje, ir tų lentelių ryšiai su pagrindine lentele. Tarp jų nebūtinai turi būti tiesioginis ryšys, lentelės gali sietis ir per tarpines lenteles. Kadangi aprašant ryšių kelius tarp lentelių gali pasitaikyti alternatyvūs keliai, tai geriau pasirinkti tokį kelią:

- Kurio ryšys tarp lentelių stipresnis;
- Kurio ryšys užmezgamas tik tarp privalomų lentelių atributų;
- Kuris saugomos informacijos požiūriu niekada nenutrūksta.

Turint visas užklausos projekte naudojamas lenteles ir ryšius galima atlikti lentelių apjungimą šiai išskirtai grupei, taip sudarant lentelių apjungimo šabloną.

3.3 Užklausos projekto sudarymas

Sudarant užklausą pagal ataskaitai reikiamus duomenis, iš sudarytų maketų parenkami laukai iš maketą sudarančių laukų sąrašo. Jei reikia, galima priskirti pasirinktą agreguojamą funkciją konkrečiam užklausos laukui, grupuoti, rūšiuoti užklausos rezultatus pagal pasirinktą lauką, pasirinkti konkrečiam laukui tikslumą, užklausai priskirti statinį filtrą arba sudaryti dinaminį filtrą pagal filtrų sudarymo taisykles. Vartotojui sudarant ir vykdant užklausas meta duomenų bazėje yra kaupiama statistika apie labiausiai naudojamus atributus ir lenteles. Taip pat pagal užklausą sudarančius laukus, statinį ir dinaminį filtrą yra sudaromas naudojamų užklausoje atributų ir lentelių sąrašas. Pagal kurį yra sprendžiama, kokį būdą naudoti: ar virtualias lenteles ar sudarytą lentelių apjungimo šabloną. Pagal užklausoje naudojamas lenteles meta duomenų bazėje yra ieškomas lentelių apjungimo šablonas (1 pav.), kuris arba visai atitiktų, arba savyje turėtų minimaliai perteklinių kitų užklausoje nenaudojamų lentelių. Jeigu tokio šablono nėra, tada užklausos FROM dalyje naudojamos virtualios lentelės ir jų sujungimai. Kai



1 pav. Šablono parinkimas.

surandamas tinkamas lentelių apjungimo šablonas, su juo yra susiejama užklausa, kad kitą kartą ją vykdant, nereikėtų iš naujo ieškoti šablono. Jeigu atnaujinami šablonai, tai vykdant užklausas, kiekvienai užklausai iš naujo ieškomas lentelių sujungimo šablonas, nes gali būti sudarytas dar efektyvesnis.

4 Eksperimentas

Buvo atliktas eksperimentas su užklausa, kurioje atributai atrenkami naudojant virtualias lenteles, užklausa pakeičiama taip, kad būtų naudojamos tik tos lentelės, kurios susijusios su atrenkamais atributais. (A) užklausa, atrenkanti 17 atributų iš lentelių, panaudojant 3 virtualias lenteles. (B) užklausa, atrenkanti tuos pačius 17 atributų, tik iš pačių lentelių tiesiogiai.

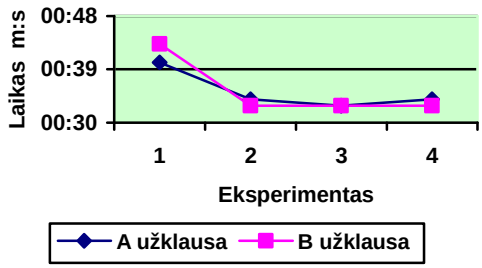
Šios A ir B užklausos įvykdytos trijose duomenų bazėse:

- PVP – 140 000 įrašų.
- Bandymai2 – 250 000 įrašų
- RMKAD – 3 000 000 įrašų.

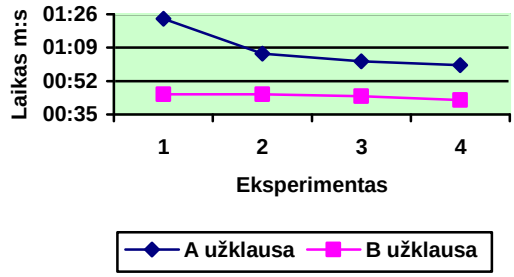
Lentelėje pateikiamas laikas, reikalingas užklausai įvykdyti ir pateikti rezultatus, bei atrinktų įrašų skaičius.

3 lentelė Užklausų įvykdymo laikas

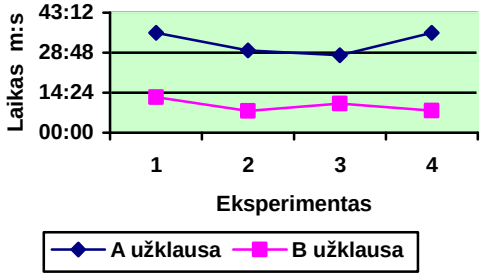
	PVP A	*PVP B*	*Bandymai2 A*	*Bandymai2 B*	*RMKAD A*	*RMKAD B*
Laikas	40s	43 s	1m 24s	45s	35m 58 s	12m 49s
Laikas	34s	33 s	1m 06s	45s	29m 39s	7m 51s
Laikas	33s	33s	1m 02s	44s	27m 56s	10m 31s
Laikas	34s	33s	1m00s	42s	35m 56s	8m 00s
Vidutinis laikas	35s	36s	1m 08s	44s	32m 22s	9m 48s
Atrinkta	141514 įrašų	141514 įrašų	189010 įrašų	189010 įrašų	2147386 įrašų	2147386 įrašų



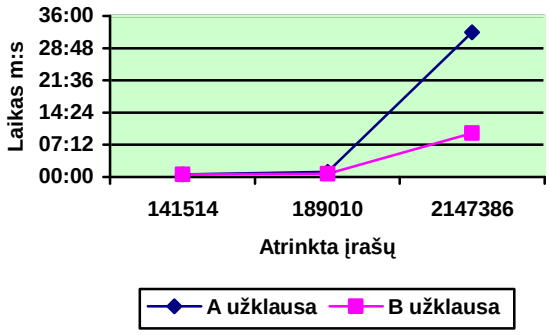
2 pav. PVP duomenų bazėje užklausų įvykdymo laikas.



3 pav. Bandymai2 DB užklausų įvykdymo laikas



4 pav. RMKAD duomenų bazėje užklausų įvykdymo laikas



5 pav. Laiko ir atrinktų įrašų priklausomybė

Kaip matyti iš lentelėse esamų duomenų, didėjant įrašų kiekiui didėja ir laikas, sunaudojamas užklausos įvykdymui. Ypač daug laiko sugaištama, kada reikia atrinkti reikiamus atributus iš tų lentelių, kurios turi milijoninius skaičius įrašų. Naudojant metodą, kuris apjungia suformuotas maketams virtualias lenteles, sugaištama labai daug laiko, todėl verta naudoti tokį sprendimą, kuris užklausoje naudoja minimaliai nenaudojamų lentelių.

5 Išvados

1. Buvo atlikta duomenų bazių spartos derinimo ir duomenų bazių spartos optimizavimo analizė, kurios metu atkreiptas dėmesys į indeksavimą ir virtualių lentelių veikimą.
2. Nustatyta, kad apjungiant kelias ir daugiau virtualių lentelių, ar naudojant indeksus, duomenų bazės sparta gali nepadidėti, o net sumažėti.
3. Duomenims filtruoti ir atrinkti sudarytas lentelių apjungimo šablonų formavimo modelis.
4. Taip pat sudarytas užklausų formavimo modelis, kuris vartotojo suformuotai užklausai parenka optimalų lentelių apjungimo šabloną.
5. Atliktas eksperimentas, kurio metu sulyginami du galimi sprendimai: virtualių lentelių naudojimas ir siūlomas metodas. Eksperimento metu nustatyta, kad sudarytas užklausų formavimo modelis teisingas, užklausos vykdomos sparčiau.
6. Ateityje numatoma ištirti perteklinių lentelių, esančių užklausos apjungimo šablone, įtaką duomenų bazės spartai.

Literatūros sąrašas

[1] **F. Charvet, A. Pande** "Database Performance Study". [žiūrėta 2008.04.09]. Prieiga per internetą: http://www.umsl.edu/divisions/business/mis/bov/TuningPaperV5.pdf

[2] **S. Agarwal, B. Baryshnikov, T. Davidson, K. Elmore, D. Ribeiro, J. Thomas** "Troubleshooting Performance Problems in SQL Server 2005", Microsoft Corporation, 2005. [žiūrėta 2008.04.09]. Prieiga per internetą: http://download.microsoft.com/download/1/3/4/134644fd-05ad-4ee8-8b5a-0aed1c18a31e/TShootPerfProbs.doc

[3] **E. Whalen, M. Garcia, S. Adrien DeLuca, D. Thompson** "Microsoft SQL Server 2000 Performance Tuning Technical Reference", Microsoft Press, 2001, ISBN 0-7356-1270-6. 438p.

[4] **Satyanarayana R Valluri** „Partially Materialized Partitioned Views", 2005. [žiūrėta 2008.04.09]. Prieiga per internetą: http://comad2005.persistent.co.in/COMAD2005Proc/pages046-057.pdf

[5] **J. Zhou, P.Larson, J. Goldstein** "Partially Materialized Views" 2005" [žiūrėta 2008.04.09]. Prieiga per internetą: http://research.microsoft.com/research/pubs/view.aspx?type=Technical%20Report&id=931

[6] **K. Kline, A. Zanevsky, L.** Gould "Microsoft T-SQL Performance Tuning Part 2: Index Tuning Strategies". O'Reilly & Associates, 2002, ISBN: 1565924010

The analysis of data filtration and selection solutions

When the amount of stored data is growing, it is very important to get them fast and users are expecting to see how database performance is rising. Using database performance tuning, or database performance optimization, it is possible to make a database system run faster. After analysis of database performance optimization and performance tuning methods was suggested a method which enables to process data from database more quick and to user to get query result faster. The experiment was performed in order to evaluate how method works.