TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Tarkvarateaduse instituut

Kätlin Raja 142744IABB

PÄRINGU FILTRI PREDIKAADI AUTOMAATNE LIHTSUSTAMINE KAHE SQL-ANDMEBAASISÜSTEEMI NÄITEL

Bakalaureusetöö

Juhendaja: Erki Eessaar

Doktor

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kätlin Raja

22.05.2017

Annotatsioon

Käesoleva bakalaureusetöö eesmärkideks on lähemalt uurida, kui palju oskavad kaks populaarset SQL-andmebaasisüsteemi (Oracle ja PostgreSQL) päringute (*SELECT* lause) filtri predikaate e tingimusi automaatselt lihtsustada ning kui palju mõjutab lihtsustamata tingimuste kasutamine päringute töökiirust.

Eesmärkide saavutamiseks projekteeriti lihtne parkimiste teemaline andmebaas ning realiseeriti see kahes SQL-andmebaasisüsteemis: Oracle (versioon Database 12c Enterprise edition Release 12.1.0.1.0) ja PostgreSQL (versioon 9.5.3). Andmebaasidesse genereeriti testandmeid.

Kokku katsetati töös kaheksat erinevat päringu tüüpi ning 26 erinevat loogikaavaldise (filtri predikaadi) tüüpi. Nendest moodustati kokku 154 päringut, kus oleks võimalik filtri predikaati lihtsustada. Lihtsustamise kindlakstegemiseks uuriti päringute täitmisplaane. Nendest katsetatud lausetest suutis Oracle lihtsustad 88 ja PostgreSQL 46 lause filtri predikaati. Teise eesmärgi saavutamiseks mõõdeti lausete täitmisaega, kui päringusse kirjutatud tingimus on lihsustamata ja lihtsustatud.

Töökiiruse uurimise tulemusena leian, et lihtsustamata filtri predikaatidega päringud suurendavad selle täitmisele kuluvat aega. Seega peaksid andmebaasisüsteemides PostgreSQL ja Oracle päringute kirjutajad pöörama rõhku koostatud lausetes kasutatavatele filtri predikaatidele, et lausete täitmiseks kuluks võimalikult vähe aega. Andmebaasisüsteem ei pruugi lihtsustamisel alati appi tulla. Lisaks sellele halvendavad lihtsustamata predikaadid inimeste jaoks koodi arusaadavust ning seega ka koodi hallatavust ning edasiarendatavust.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 38 leheküljel, 6 peatükki, 17 joonist, 16 tabelit.

Abstract

Automatic Simplification of Query Filter Predicate in the Example of Two SQL Database Management Systems

A goal of the bachelor's thesis is to examine how much can two popular SQL Database Management Systems (DBMSs) automatically simplify query (*SELECT* statement) filter predicates. If one uses un-simplified filter predicates, then to what extent does it affect the overall query execution speed? Thus, the second goal is to examine as to whether the un-simplified query filter predicate reduces query execution time or not.

In order to achieve the established goals, a simple database of parkings was constructed. It was implemented in two separate SQL DBMSs: Oracle (version Database 12c Enterprise Edition Release 12.1.0.1.0) and PostgreSQL (version 9.5.3). Test data was generated.

Eight different query types and 26 types of different boolean expressions (filter predicate types) were considered in the study. From those – 154 queries with un-simplified filter predicates were created. Execution plans of the queries were investigated to find out as to whether the DBMS simplifies the predicates. In case of these queries Oracle was able to simplify the predicate of 88 and PostgreSQL 46 queries.

Experiments with three different sets of rows were conducted to achieve the second goal. In case of the biggest set, the cardinality of the biggest table was 1 million rows. Based on the findings, I conclude that un-simplified filter predicates increase the query execution time. Hence, it is advisable to place more emphasis on query filter predicates to make query execution faster. A DBMS might help in simplifying the predicate but it is not guaranteed. Moreover, un-simplified predicates make it harder to understand the code by humans and it makes it more difficult to manage and modify the code.

The thesis is in Estonian and contains 38 pages of text, 6 chapters, 17 figures, 16 tables.

Lühendite ja mõistete sõnastik

Alternatiivvõti — ***Unique***

Alternatiivvõti on ühest või rohkemast veerust koosnev veergude hulk, mille väärtus identifitseerib unikaalselt tabeli ridu. Selle SQL-andmebaasi baastabelis jõustamiseks tuleb kasutada UNIQUE ning NOT NULL kitsendusi. Igas tabelis on null või rohkem alternatiivvõtit.

Andmekäitluskeel — ***Data Manipulation Language***

Andmebaasikeele alamkeel, mis on mõeldud andmete otsimiseks e pärimiseks ning muutmiseks.

Baastabel — ***Base table***

Nimega tabel, mis ei ole defineeritud teiste tabelite põhjal. Baastabelites on alusfaktid, millest saab päringutega hakata tuletama uusi fakte.

CSV — ***Comma Separated Values***

CSV on komadega eraldatud väärtuste fail, mis võimaldab andmeid salvestada tabeli struktrureeritud formaadis [1].

Disjunktiivne normaalkuju — ***Disjunctive normal form***

„Antud valemiga loogiliselt samaväärne valem, mis kujutab endast elementaarkonjunktsioonide disjunktsiooni [2].“ Disjunktsioon on loogiline liitmine (loogikaoperaator OR).

Geomeetriline keskmine — ***Geometric mean***

„Positiivsete suuruste korrutis, mis on astendatud nende suuruste arvu pöördväärtusega [2].“

Indeks — ***Index***

Indeks on andmebaasi siseskeemi element, mis võimaldab andmebaasisüsteemil osadele päringutele kiiremini vastata.

Konjunktiivne normaalkuju — ***Conjunctive normal form***

„Antud valemiga loogiliselt samaväärne valem, mis kujutab endast elementaardisjunktsioonide konjunktsiooni [2].“ Konjuktsioon on loogiline korrutamine (loogikaoperaator AND).

Loogikaalgebra — ***Boolean algebra***

Loogikaalgebra koosneb loogikaväärtuste hulgast {true; false}, millel on defineeritud kolm elementaarset loogikatehet: unaarne

tehe inversioon ehk eitus ja binaarsed tehted konjunktsioon ja disjunktsioon [3, p. 21].

Predikaat ***Predicate***

Predikaat on tõeväärtusfunktsioon, millel on null või rohkem parameetrit.

Päringu filtri predikaat ***Query filter predicate***

(päringu tingimus) ***(Query condition)***

Päringu (*SELECT* lause) *WHERE* klauslis olev loogikaavaldis on olemuselt predikaat. Selle parameetriteks on veergude nimed. Tingimuse kontrollimisel asendatakse iga kontrollitava rea korral veeru nimi sellele veerule vastavas väljas oleva väärtusega. Tulemuseks saadakse väide, mille tõesust saab andmebaasisüsteem kontrollida [4]. Sellise predikaadi saab kirjutada ka *HAVING* klauslisse, kuid *SELECT* lause saab alati kirjtuada ümber nii, et *HAVING* klausli asmel kasutatakse *WHERE* klauslit. Selleks on vaja kasutada *FROM* klauslis olevat alampäringut.

Primaarvõti ***Primary key***

Primaarvõti on ühest või rohkemast veerust koosnev veergude hulk, mille väärtus identifitseerib unikaalselt tabeli ridu. Selle SQL-andmebaasi baastabelis jõustamiseks tuleb kasutada PRIMARY KEY kitsendust. Igas baastabelis on maksimaalselt üks primaarvõti.

SQL **SQL – *Structure Query Language***

Andmebaasikeel, mis on mõeldud andmete haldamiseks, admestrutkuuride haldamiseks ning neile juurdepääsu reguleerimiseks. See keel põhineb relatsioonilisel andmemudelil.

SQL-andmebaasisüsteem ***SQL Database Management System***

Andmebaasisüsteem, kus saab andmetega töötamiseks kasutada SQL andmebaasikeelt.

Välisvõti ***Foreign Key***

Välisvõti on veergude hulk, milles olevad väärtused loovad seosed samas või teises tabelis olevate andmetega.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Andmebaas on korrastatud infokogum, kust on võimalik vajalikke andmeid otsida. Andmebaasisüsteemide oluliseks ülesandeks on info tagastamine võimalikult kiiresti ning ilma ressursse raiskamata. Seejuures on oluline, et süsteem oskaks kirjutatud päringut hästi optimeerida, st muuta seda võimalusel loogiliselt samaväärseks kuid lihtsamaks ja leida selle täitmiseks võimalikult hea täitmisplaan. Täitmisplaan paneb paika lause täitmiseks kasutatava algoritmi. Päringu optimeerimine andmebaasisüsteemi poolt vähendab selle täitmiseks kuluvat aega.

SQL-andmebaasisüsteemides on üheks probleemiks andmekäitluskeele lausete tingimustes (*WHERE* või *HAVING* klauslis) olevate filtri predikaatide e tingimuste lihtsustamine. Nende abil filtreeritakse lauses käsitletavaid ridu. Kui tegemist on andmete otsimise lausega e päringuga, siis on tulemuses vaid read, mis rahuldavad predikaadiga määratud tingimust. Kui tegemist on andmete muutmise lausega, siis piirab tingimus muutuse ulatust. Andmekäitluskeele lause kirjutajal on vabadus kirjutada lausesse filtri predikaat, mille saaks asendada loogiliselt samaväärse, kuid lihtsama predikaadiga. Antud töös käsitletakse fraase „filtri predikaat“ ja „tingimus“ sünonüümidena.

Käesoleva bakalaureusetöö üheks eesmärgiks on uurida, kui palju oskavad kaks populaarset SQL-andmebaasisüsteemi (Oracle ja PostgreSQL) päringute (*SELECT* lausete) filtri predikaate automaatselt lihtsustada ning kui palju mõjutab lihtsustamata tingimuste kasutamine päringute töökiirust. Esimese eesmärgi täitmiseks uuritakse SQL lausete täitmisplaane. Teise eesmärgi täitmiseks mõõdetakse lausete täitmisaegu, kui päringusse kirjutatud tingimus on lihsustamata ja lihtsustatud. Töös uuritakse kahte SQL-andmebaasisüsteemi: Oracle (versioon Database 12c Enterprise Edition Release 12.1.0.1.0) ja PostgreSQL (versioon 9.5.3). Töö tulemused esitavad võimalikult täieliku hulga päringute tingimustest, mida mõlemad süsteemid oskasid või ei osanud lihtsustada. Samuti otsitakse vastust küsimusele, kas lihtsustamata filtri predikaadid päringutes omavad mõju nende täitmise kiirusele.

Antud töö jaguneb neljaks suuremaks osaks. Kõigepealt antakse lühiülevaade SQL andmekäitluskeele lausete töötlemisest. Lisaks selgitatakse, mis on päringute lihtsustamine ning miks tekivad lihtsustamata päringud. Teises peatükis kirjeldatakse eksperimendi läbiviimiseks loodud andmebaasi ning esitatakse katsete täpsed spetsifikatsioonid: ehk selgitatakse mida ja kuidas tehti, et huvilistel oleks võimalus eksperimenti korrata. Kolmandas peatükis esitatakse kõik katsete tulemused koondtabelitena ja analüüsitakse huvipakkuvamaid käsitletud päringuid eraldiseisvalt. Neljandas peatükis esitatakse järeldused.

2 SQL andmekäitluskeel

Andmekäitluskeel on üks SQLi alamkeeltest. See on standardne keel andmete lisamiseks, muutmiseks, kustutamiseks ja andmebaasist pärimiseks e otsimiseks [5]. See tegeleb andmete kasutamisega kõige kõrgemal loogilisel tasemel – puhta andmetöötlusega. SQL andmekäitluskeele lause on deklaratiivne – see ütleb andmebaasisüsteemile, milliseid andmeid on vaja leida või milliseid andmeid on vaja lisada, muuta või kustutada. Andmekäitluskeele lause alusel koostab andmebaasisüsteem imperatiivse programmi (päringu täitmise sammsammulise käigu), mis realiseerib soovitud tulemusteni jõudmiseks kasutava algoritmi. Sellist programmi nimetatakse füüsiliseks täitmisplaaniks [6].

Andmebaasisüsteemi ülesandeks on leida võimalikult hea e võimalikult optimaalne täitmisplaan. Üks võimalus optimaalsuse hindamiseks on lause täitmise kiiruse järgi – mida kiiremini täitmisplaani kirjeldatav programm lõplikult täidetakse ning lause eesmärk saavutatakse, seda parem [6]. Andmebaasisüsteemi võimetus leida head täitmisplaani on oluliseks andmekäitluskeele lausete kauakestva täitmise põhjuseks. Sõltuvalt valitud täitmisplaanist võib sama lause täitmise kiirus kordades erineda [7]. Andmebaasisüsteemi osaks olevale andmekäitluskeele lausete töötlemise moodulile on lause töötlemisel sisendiks töödeldav lause, andmebaasi skeem (andmebaasi tabelite ja indeksite definitsioonid) ja andmebaasis olevaid andmeid iseloomustav statistika [8].

SQLi probleemiks on liigne võimsus – ühte ja sama ülesannet saab lahendada paljudel erinevatel viisidel, sh ühte ja sama andmeid piiravat tingimust saab kirja panna paljudel erinevatel viisidel. See muudab andmebaasisüsteemi jaoks lausete optimeerimise raskemaks ning paneb suurema vastuse lausete kirjutaja õlgadele. Selleks, et SQL andmekäitluskeele lausete töötlemise protsess oleks võimalikult kiire ja tulemuslik, peaks lausete kirjutaja pöörama tähelepanu kirja pandud lausele ning viima selle võimalikult lihtsale loogiliselt samaväärsele kujule [7].

2.1 Tingimused andmekäitluskeele lausetes

Üheks andmekäitluse operatsiooniks on piirangu operatsioon, mis piirab andmekäitluskeele lause käsitletavate ridade hulka. Võib öelda, et see operatsioon filtreerib ridu ja jätab sõelale vaid piirangu operatsiooni tingimusele vastavad read.

SQL keeles realiseeritakse piirangu operatsioon *WHERE* ja *HAVING* klauslis olevate tingimuste abil [9]. *HAVING* klauslit sisaldava lause saab alati ümber kirjutada nii, et see sisaldaks selle asemel *WHERE* klauslit. Tingimus on loogikaavaldis, mis võib olla atomaarne tingimus või atomaarsetest tingimustest loogikaoperaatorite (AND, OR, NOT) abil moodustatud liittingimus [4]. Tingimus on predikaat e tõeväärtusfunktsioon. Tingimuses kasutatavad veergude nimed on parameetrid, mille kõigi asendamisel tabeli reas olevate väärtustega saadakse väide. Sellise väite tõesust on võimalik kontrollida. Tingimusele vastavad kõik need read, mille puhul saadud väite kontrollimise tulemus on TRUE.

Jaotises 2.2 kirjutatakse täpsemalt, kuidas on võimalik lause koostajal andmebaasisüsteemi töövaeva vähendada. Jaotises 2.4 kirjutatakse põhjustest, miks tekivad keerulised, lihtsustamata laused, mis suurendavad nende töötlemiseks kuluvat aega. Käesoleva töö eksperimendi osas vaadeldakse ainult *SELECT* lauseid e päringuid ja ainult tingimusi, mis on *WHERE* klauslis. Seetõttu kasutan ka järgmistes jaotistes terminit „päring“, sest see on lühem ja suupärasem kui „andmekäitluskeele lause“.

Ingliskeelses kirjanduses viitab sõna „query“, mis eesti keelde tõlgitakse kui „päring“, sageli mistahes andmekäitluskeele lausele. Eesti keeles tähendab päring andmete otsimist. Üldised põhimõtted andmete otsimise, lisamise, muutmise ja kustutamise lausete kirjutamisel ning töötlemisel on ühesugused.

2.2 Tingimuste analüüsimine

Päringu töötlemine on mitmesammuline protsess. Üks osa sellest on tingimuste analüüsimine ja optimeerimine. SQL-andmebaasisüsteemides tugineb see suuresti loogikaalgebrale. Loogikaalgebra, ka Boole’i algebra, avaldiste lihtsustamine on üks arvutiteadusele huvi pakkuvatest probleemidest. Paljud algoritmid eeldavad, et loogikaavaldis on juba kas konjunktiivsel või disjunktiivsel normaalkujul. Filtri

predikaadi lihtsustamisel ei ole aga garantiid, et see oleks kirjutatud juba normaalkujul [10].

Andmebaasisüsteem võib enne päringule täitmisplaani otsimist või selle käigus asendada selle lihtsama, kuid loogiliselt samaväärse päringuga. Antud tegevust võib nimetada ka päringu lihtsustamiseks. Sealhulgas toimub ka filtri predikaadi lihtsustamine, st asendamine andmebaasisüsteemi jaoks lihtsamini töödeldava, kuid loogiliselt samaväärse predikaadiga [10].

Tingimuste analüüsimisel tuvastatud vead võib jagada kaheks: süntaktilisised ja semantilised. Süntaktiline viga tähendab, et sisestatud märgijada ei vasta SQL keele nõuetele. Sellistel juhtudel väljastavad kõik andmebaasisüsteemid veasõnumi. Semantiline viga tähendab, et koostatud tingimus on süntaktiliselt korrektne, kuid ebaloogiline. Näiteks on päringus (vt Joonis 1) kirjutatud tingimused vastuolus, mis tähendab, et see päring tagastab tühja tulemuse. Semantiliste vigade korral andmebaasisüsteemid tüüpiliselt veasõnumeid ei väljasta [11, pp. 1, 14].

```
SELECT *
FROM Parkimine
WHERE kestus>8
  AND kestus>10;
```

Joonis 1. Vastuolulise päringu näide.

Soovitud andmete leidmiseks andmebaasist tuleb päringu tulemust sobilike tingimustega piirata. Sellest tulenevalt tekivad tihti pikad liittingimused, mida on sageli võimalik lihtsustada. Kuigi SQL keel on deklaratiivne, on päringu kirjutaja ülesandeks aidata süsteemil päringut efektiivselt täita. Võib juhtuda, et süsteem ei oska tingimust lihtsustada ja see võib muuta lause täitmise aeglasemaks [12, pp. 631; 636-637].

Päringu töötlemise käigus toimuva päringu tingimuse kontrollimise tüüpilised sammud on [13, pp. 609-613]:

- analüüs – kontrollitakse, et tingimuses nimetatud tabelid ja veerud oleksid andmebaasis olemas. Samuti kontrollitakse, et kõik andmetega tehtavad operatsioonid oleksid andmetüüpe arvestades võimalikud;
- normaliseerimine – tingimus lihtsustatakse kas konjunktiivsele või disjunktiivsele normaalkujule;
- semantiline analüüs – ülesandeks on tuvastada vastuolulised alamtingimused;

- lihtsustamine – eesmärgiks on leida ja eemaldada ülearused alamtingimused ning saada tulemuseks loogiliselt samaväärne, kuid lihtsam tingimus. Näiteks võidakse elimineerida korduvaid alampäringuid;

Antud töö puhul pakub huvi kas ja kui hästi oskavad andmebaasisüsteemid läbi viia lihtsustamise sammu. Päringu töötlemise käigus koostab andmebaasisüsteem täitmisplaani. Jaotises 2.3 kirjutatakse täitmisplaanidest täpsemalt.

2.3 Täitmisplaanid

Peale seda kui andmebaasisüsteem on lauset süntaktiliselt ja semantiliselt kontrollinud toimub selle täitmisplaani koostamine. Täitmisplaan annab andmebaasisüsteemi andmekäitluskeele lausete täitmise moodulile täpsed päringu täitmise juhised. Täitmisplaani koostamiseks on olemas kaks äärmuslikku lähenemist: staatiline ja dünaamiline. Võimalik on ka hübriidne lähenemine. Staatilise optimeerimise käigus koostatakse täitmisplaan üks kord, see salvestatakse ning sama päringu uuesti käivitamisel seda taaskasutatakse. Alternatiiviks on dünaamiline optimeerimine. Selle korral leitakse lause käivitamisel iga kord täitmisplaan uuesti [13, pp. 608-609]. Hübriidse lähenemise korral koostab andmebaasisüsteem vajadusel (nt andmete hulk on võrreldes eelmise täitmisplaani koostamisel ajal muutunud) plaani uuesti või teeb plaanis käigupealt muudatusi, kui plaani täitmise käigus selgub, et andmed on vahepeal märkimisväärselt muutunud [14].

Lausete täitlmisplaani koostamine jaguneb loogilise ja füüsilise täitmisplaani koostamiseks. Esmalt koostab andmebaasisüsteem loogilise täitmisplaani, mis on relatsioonialgebra operatsioonide puu. Selle puu lehtedeks on relatsioonid (tabelid), millest saadakse lähteandmed ning ülejäänud sõlmed esitavad relatsioonialgebra operatsioone. Koostatud relatsioonialgebra puu on sisendiks füüsilise täitmisplaani koostamisele [6]. Andmebaasisüsteem kasutab loogilist täitmisplaani ja andmebaasi statistikat, et leida päringu läbi viimiseks parim võimalik füüsiline täitmisplaan. Andmebaasisüsteem genereerib mitmeid füüsilisi täitmisplaane, arvutab igaühele neist andmebaasi statistika põhjal maksumuse ja valib neist kõige madalama maksumusega plaani. Sellist protsessi nimetatakse maksumuspõhiseks optimeerimiseks. SQL-andmebaasisüsteemid võimaldavad kasutajatel andmekäitluskeele lausete füüsilisi

täitmisplaane vaadata [15]. Seda võimalust kasutatakse käesolevs töös, et näha, kas andmebaasisüsteem on päringu tingimust lihtsustanud või mitte.

Päringu täitmisplaani koostamisel toetub andmebaasisüsteem andmebaasi statistikale tabelite veergudes ja indeksites olevate andmete kohta. Statistika kogumisega tegeleb andmebaasisüsteem, inimkasutaja saab anda korralduse statistikat värskendada [15]. Hea täitmisplaani koostamiseks peab statistika olema värske. Käesolevas töös värskendati statistikat päringute töökiiruse mõõtmise käigus, iga kord, kui tabelites olevate ridade hulka muudeti.

SQL-andmebaasisüsteemid kasutavad indekseid, et mõningaid päringuid kiiremini täita. Piirangu operatsiooni läbi viimisel peab andmebaasisüsteem otsustama, kas kasutada tingimusele vastavate andmete otsimiseks indeksit või mitte. Piirangu operatsiooni läbiviimiseks võib andmebaasisüsteem valida tabeli täieliku läbiskaneerimise, ridade indeksi abil otsimise või ainult indeksi alusel otsimise vahel. Nii nagu raamatu indeksi kasutamine on mõnikord otstarbekas ja mõnikord mitte, on sama lugu ka andmebaasi indeksi kasutamisega. Selleks, et andmebaasisüsteem üldse saaks kaaluda päringu täitmiseks indeksi kasutamist, peab päringu tingimustes kasutatud veergudele olema loodud indeks [16]. SQL-andmebaasisüsteemid loovad tavaliselt primaarvõtme ja unikaalsuse indeksi alusel indeksi automaatselt. Kasutaja ei peaks ise looma indekseid, mis selliseid indekseid dubleerivad [17]. Käesolevas töös loodi testandmebaasi samuti mõned täiendavad indeksid.

Andmebaasisüsteemid hindavad statistikat kasutades, milline on iga operatsiooni tulemuseks olev ridade arv. Ridade arvu hinnang on oluline, et valida parimad sisemise taseme operatsioonid ja nende kõige parem järjekord. Teoreetiliselt, nii kaua kuni ridade arvu hinnang ja koostatud maksumuse hinnang on täpsed, leitakse andmebaasisüsteemi poolt päringule optimaalne täitmise plaan. Tegelike andmebaaside ja andmebaasisüsteemide korral on need hinnangud mõnikord ebatäpsed. Sellisel juhul kasutab andmebaasisüsteem otsuste tegemisel valesid eelduseid, mis viib mitteoptimaalsete täitmisplaanideni ning päringu aeglase täitmiseni [18, p. 206]. Käesoleva töö kontekstis pakub huvi, kui palju mõjutavad lihtustamata tingimused andmebaasisüsteemi hinnangut ridade hulgale ning kas see mõjutab täitmisplaani koostamist.

2.4 Lihtsustamata tingimuste tekkimise põhjused

Selleks, et andmebaasisüsteem täidaks päringut kiiremini, on vajalik, et päring, sh ka selle tingimus, oleks kirja pandud võimalikult lihtsalt ning loogilisel kujul. Põhjuseid, miks tekivad lihtsustamata tingimustega päringud, on mitmeid. Päringu kirjutajal võib olla SQL lausete kirjutamises vähe kogemust. Arvan, et see võib olla üheks peamiseks lihtsustamata tingimuste tekkimise põhjuseks. Lihtsustamata tingimused võivad tekkida ka olukorras, kus andmeid piiravate tingimuste hulk kasvab väga suureks. Seejuures on lihtne jätta märkamata, et kirja pandud tingimusele ei saa vastata ükski rida või vastavad kõik read [12, p. 631]. Samuti võib jääda kahe silma vahele, et alamtingimused kordavad üksteist või sisaldub üks alamtingimus teises.

Lihtsustamata tingimused võivad tekkida ka SQL lausete generaatori kasutamisel. Võib juhtuda, et generaatori koostajad ei olnud teadlikud lihtsustamise võimalustest või töötab genereeraatori kood valesti.

Joonis 2 on näide lihtsustamata tingimusega päringust. Ilmselt sunnib see andmebaasisüsteemi valima täitmisplaani, kus esmalt koostatakse ning materialiseeritakse alampäringu tulemused ning rakendatakse sellele siis täiendav kitsendus.

```
SELECT parkimine_id
FROM
  (SELECT parkimine_id,
          registri_nr
   FROM Parkimine
   WHERE registri_nr IN
       (SELECT registri_nr
        FROM Teisaldamine
        WHERE parkimine_id>100))
WHERE registri_nr LIKE '2%';
```

Joonis 2. Ebaefektiivse sõnastuse näide.

Joonis 3 on näide lihtsustatud tingimusega päringust. Andmebaasisüsteem ei pea alampäringu tulemusi materialiseerima. Lisaks on lihtsustatud päring kompaktsemalt kirjutatud ning ka inimkasutajale loetavam [6].

```
SELECT parkimine_id
FROM Parkimine
WHERE registri_nr LIKE '2%'
  AND parkimine_id IN
    (SELECT parkimine_id
     FROM Teisaldamine
     WHERE parkimine_id>100);
```

Joonis 3. Lihtsustatud päringu näide.

3 Eksperimendi kirjeldus

Eksperimendi eesmärgiks on leida andmebaasisüsteemide võimekus ja erinevused päringu filtri predikaatide automaatse lihtsustamise osas ning võrrelda päringute täitmise aega lihtsustatud ja lihtsustamata filtri predikaatide korral.

SQL-andmebaasisüsteeme on kokku väga palju: andmebaasi populaarsuse indeksi tabelis 2017. aasta maikuu seisuga kokku 131. Seega testida kõiki neid süsteemi käesoleva bakalaureusetöö raames oleks liiga mahukas. Eksperimendis kasutavate andmebaasisüsteemide valimisel lähtusin süsteemide populaarsusest ning minu võimalustest ja oskustest süsteeme kasutada. Valisin nimetatud süsteemidest Oracle ja PostgreSQLi. Oracle oli 2017. aasta veebruaris kõige populaarsem andmebaasisüsteem, PostgreSQL oli populaarsuselt neljas [19].

Kasutatavad andmebaasisüsteemide versioonid olid järgmised:

- Oracle Database 12c Enterprise Edition Release 12.1.0.1.0
- PostgreSQL 9.5.3

Oracle ja PostgreSQLi kasutamist võimaldati Tallinna Tehnikaülikooli poolt.

3.1 Andmebaasi projekteerimine

Kasutasin väljavalitud andmebaasisüsteemide päringute lihtsustamise tingimuste ja täitmisplaanide analüüsimiseks andmebaasi, mis oleks lihtne, aga samas võimaldaks piisavalt päringuid koostada.

3.1.1 Andmebaasi loodavad tabelid

Lõin andmebaasi kolm tabelit: *Klient*, *Parkimine*, *Teisaldamine*. Nende tabelite kõik veerud on kohustuslikud, st ei luba NULLe. Kuna SQL kasutab tänu NULLidele kolmevalentset loogikat, siis mõjutab see valik päringute tingimusi. Tänu veergude kohustuslikkusele ei saa ühegi töös käsitletud tingimuse korral olla mõne rea korral selle kontrollimise tulemus UNKNOWN. Tabel 1 esitab andmebaasi veergude näiteväärtuseid.

Olgu öeldud, et indekseeritud on veerud, mida kasutatakse piirangu operatsioonides (*SELECT* lause *WHERE* klauslis). Samuti on indekseeritud ka välisvõtmed.

Tabel 1. Tabelite veerud.

Tabeli nimi	**Veeru nimi**	**Näite väärtus**
Klient	kliendi_kood	1
Klient	eesnimi	Tiiu
Klient	perenimi	Kask
Parkimine	parkimine_id	1
Parkimine	kliendi_kood	1
Parkimine	algus	2017-01-17 07:55:28
Parkimine	kestus	6
Parkimine	registri_nr	467KLO
Teisaldamine	parkimine_id	1
Teisaldamine	teisaldamise_aeg	2017-01-19 09:15:45

3.1.2 Andmebaasi realisatsioon Oracle näitel

Mõlemad töös kasutavad süsteemid on SQL-andmebaasisüsteemid. Seega on nendes andmebaasi loomine sarnane. Erinevusi esineb eelkõige SQL lausete süntaksis ja andmetüüpides (nii nimedes kui nende tähistatavates väärtuste hulkades). Eelnevalt kirjeldatud asjaoludest tulenevalt kirjeldan andmebaasi loomist ühe andmebaasisüsteemi näitel ning toon suurimad erisused lühidalt välja.

Oracle korral lõin tabelid eelnevalt loodud skeemi *C##TUD3*. PostgreSQLis lõin esmalt andmebaasi ning tabelid läksid selles automaatselt loodvasse skeemi *PUBLIC*. Oracles käivitasin tabelite loomiseks Joonis 4 olevad laused.

```
CREATE TABLE Klient (
      Kliendi_kood NUMBER (10,0) GENERATED AS IDENTITY START WITH 1,
      Eesnimi VARCHAR2 (255 BYTE) NOT NULL
      Perenimi VARCHAR2 (255 BYTE) NOT NULL,
      CONSTRAINT PK_kliendi_kood PRIMARY KEY (kliendi_kood)
);

CREATE TABLE Parkimine (
      Parkimine_id NUMBER (10,0) GENERATED AS IDENTITY START WITH 1,
      Kliendi_kood NUMBER (10,0) NOT NULL,
      ALGUS TIMESTAMP (6) NOT NULL,
      Kestus NUMBER (5,0) NOT NULL,
      Registri_nr VARCHAR2(255 BYTE),
      CONSTRAINT PK_parkimine_id PRMARY KEY(parkimine_id),
      CONSTRAINT AK_kliendikood_algus UNIQUE (kliendi_kood, algus),
      CONSTRAINT FK_kliendi_kood FOREIGN KEY (kliendi_kood) REFERENCES
      Klient (kliendi_kood)
);
CREATE TABLE Teisaldamine (
      Parkimine_id NUMBER(10,0) NOT NULL,
      Teisaldamise_aeg TIMESTAMP (6) NOT NULL,
      PRIMARY KEY (parkimine_id),
      CONSTRAINT FK_parkimine_id REFERENCES Parkimine (parkimine_id)
);
```

Joonis 4. Tabelite loomise laused Oracle andmebaasisüsteemis.

Lisa 1 (Joonis 18) kirjeldab PostgreSQLi tabelite loomise lauseid. Antud töös kasutatavad andmebaasisüsteemid loovad automaatselt indeksid primaarvõtmetele ning unikaalsuse kitsendusega hõlmatud veergudele, kuid mitte välisvõtmetele. Välisvõtmetesse kuuluvatele veergudele lõin eraldi indeksid. Lisaks indekseerin veerud, mida kasutan töös uuritavates kitsendustes (*SELECT* lause *WHERE* klauslis) – *kestus* ja *registri_nr*. Joonis 5 esitab indeksite loomise laused. Nii PostgreSQL kui Oracle loovad selliste lausetega B-puu (tasakaalustatud puu) indeksi.

```
CREATE INDEX ixfk_kliendi_kood ON Parkimine (kliendi_kood);
CREATE INDEX ix_registri_nr ON Parkimine (registri_nr);
CREATE INDEX ix_kestus ON Parkimine (kestus);
```

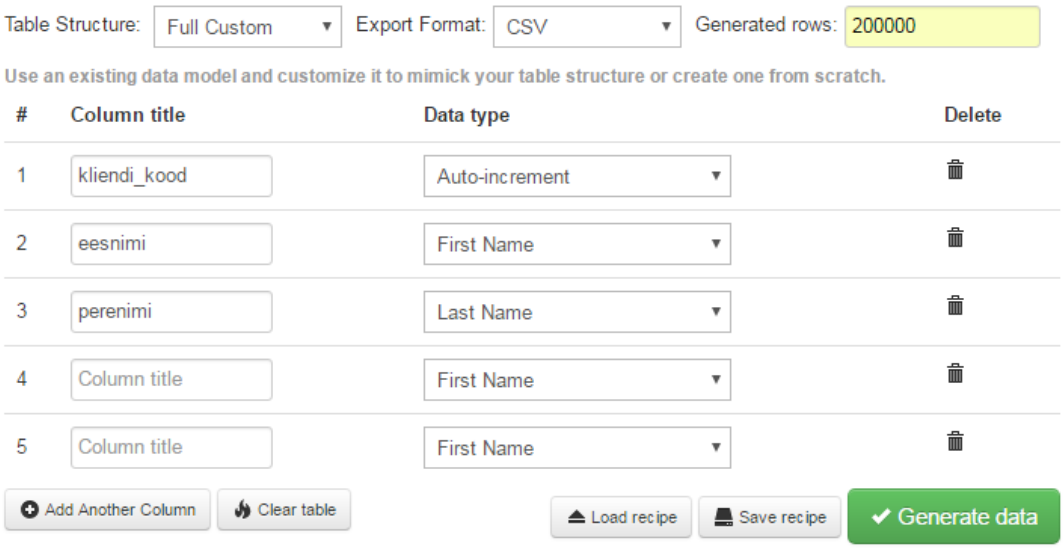
Joonis 5. Indeksite loomise laused Oracle ja PostgreSQL andmebaasisüsteemid.

3.2 Testandmete genereerimine

Töö eesmärkide saavutamiseks viin läbi eksperimendi andmebaasisüsteemides päringu filtri predikaadi (tingimuse) lihtsustamise uurimiseks. Selleks, et katsetest saadavad tulemused oleksid andmebaasisüsteemide vahel võrreldavad, peavad kõigis andmebaasides olema samasugused testandmed.

Testandmete genereerimiseks on mitmeid erinevaid võimalusi. Otsustasin tabelisse *Klient* testandmete genereerimiseks kasutada veebipõhist testandmete generaatorit [20]. Valisin selle lihtsuse tõttu, sest see ei vaja eraldi tarkvara paigaldamist.

Testandmete genereerimiseks kasutatud leheküljel on võimalik määrata andmetüübid ning soovitud testandmete ridade arv (vt Joonis 6). Valida oli võimalik testandmete ekspordi väljundite vahel, vastavalt JSON, CSV või XML. Valisin CSV-kuju.



Joonis 6. Tabelisse Klient testandmete genereerimine.

Kasutatud leheküljel puudus aga mugav võimalus tabelisse *Parkimine* vaja mineva registri numbri genereerimiseks. Samuti jooksis leht suure hulga andmete genereerimisel pidevalt kokku. Seega leidsin, et tabelisse *Parkimine* ja *Teisaldamine* on testandmete genereerimine Excelis mugavam.

Kasutan Excelit oma igapäevatöös ning leian, et see oli tabelisse *Parkimine* ja *Klient* andmete genereerimiseks minu jaoks parim ning mugavaim võimalus. Testandmete genereerimisel kasutasin kõikide andmehulkade korral sama loogikat. Lisa 2 kirjeldab testandmete genereerimist Excelis.

3.2.1 Ridade arv andmebaasi tabelites

Töökiiruse uurimiseks genereerin tabelitesse erinev arv testandmeid. Töökiirust uuritakse erinevate andmehulkade korral. Tabel 2 esitab ülevaate andmebaasi lisatavate testandmete hulgast.

Suurima hulga ridade genereerimisel arvestasin asjaoluga, et Tallinna linnas elab 2017. aastal umbes 440 000 inimest [21]. Statistikaameti andmetel oli Eesti 2015. aastal 1000 elaniku kohta 548 autot [22]. Sellest eeldusest lähtudes on Tallinnas umbes 440 000 inimese kohta 241 120 autot. Iga auto pargib päevas mitu korda. Tabelisse *Parkimine* maksimum ridade arvu lisamisel võeti aluseks hinnanguline Tallinna linna nädalase perioodi parkimiste arv. Tegeliku teisaldamiste arvu kohta mul andmed puuduvad, kuid parkimisprobleeme arvestades peaks neid kindlasti üksjagu toimuma.

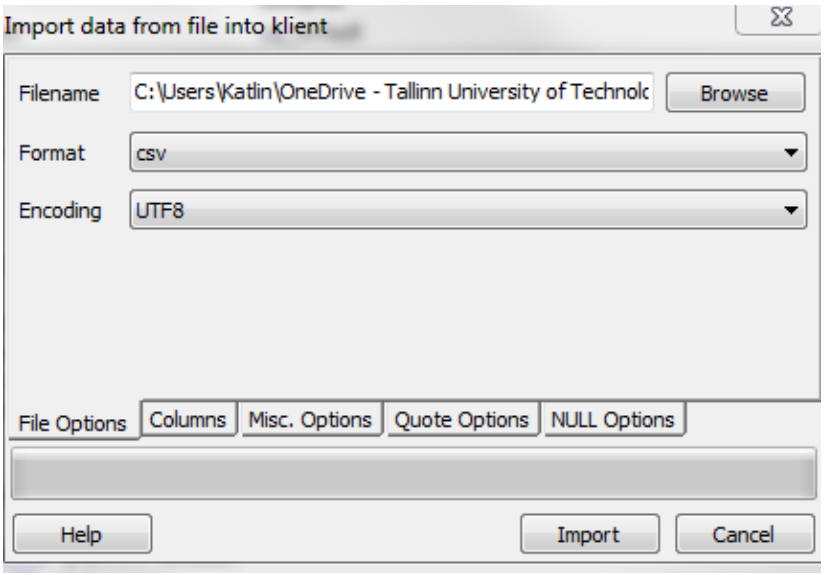
Tabel 2. Ridade arv andmebaasi tabelites.

Tabeli nimi	**Suurim ridade arv**	**Kaks korda väiksem ridade arv**	**Neli korda väiksem ridade arv**
Klient	200 000	100 000	50 000
Parkimine	1 000 000	500 000	250 000
Teisaldamine	10 000	5000	2500

3.2.2 Andmete ülekandmine andmebaasidesse

Antud töös käsitlevatel andmebaasisüsteemidel on mõlemal olemas graafilist kasutajaliidest pakkuvad programmid (Oracle SQL Developer, pgAdmin III), mille kaudu on võimaldatud ka andmete importimine failist. Kõikide saadud testandmete andmebaasi ülekandmiseks kasutasin süsteemide poolt pakutavaid võimalusi (vt Joonis 7).

PostgeSQLi korral peavad väärtused failis olema samas järjekorras nagu on veergude järjekord tabelis, kuhu need andmed kantakse. Oracle korral on olemas võimalus failis kirjutatud veergude vastavusse viimiseks tabelis olevate veergudega. Lisaks selgus, et Excelis genereeritud väärtused ei ole alati unikaalsed ning läksid vastuollu tabelis *Parkimine* defineeritud unikaalsuse kitsendusega. Tegin vigade tuvastamisel parandusi käsitsi.



Joonis 7. Andmete importimine PostrgeSQLi.

Andmebaasis oleks saanud jõustada mitmeid täiendavaid kitsendusi (näiteks teisaldamise aeg ei tohi olla väiksem kui parkimise alguse aeg), kuid seda otsustati mitte teha kuna töö eksperimendi seisukohalt ei ole antud täiendavad kitsendused määrava tähtsusega. Töös kasutavatesse andmebaasisüsteemides loodud andmebaasidesse imporditi täpselt samad andmed.

3.3 Testpäringud

Selleks, et saada ülevaadet päringute tingimuste lihtsustamisest andmebaasisüsteemides, peab neis käivitama erinevaid päringuid, mille puhul on andmebaasisüsteemil võimalik päringu tingimust lihtsustada.

Tabel 3 esitab kõik antud töös testitavad lihtsustamata tingimuste tüübid. Iga tüübi korral esitatakse ka oodatav lihtsustamise tulemus. A, B ja C on loogikatingimused (nt kestus=6). A, B ja C võivad omakorda sisaldada täpset otsingut, mustri järgi otsingut, aga ka alampäringut. Tingimusi võib kasutada põhipäringutes ja alampäringutes. a tähistab konkreetset veergu (nt kestus). Testitavate tingimuste tüübid leidsin allikatest [6] ja [23]. Tingimuste tüüpides kasutatavad arvulised ja tekstilised konstandid (nt 6, '1%') on näited ja nende asemel võib olla ka mõni muu konstant, mis on tingimust arvestades sobivat tüüpi.

Tabel 3. Testitavate tingimuste tüübid.

Lihtsustamata tingimus	**Oodatav lihtsustamise tulemus**
A AND A	A
A AND (false)	False (tingimusele ei vasta ühtegi rida)
A AND (true)	A
A AND NOT A	False (tingimusele ei vasta ühtegi rida)
A AND (A OR B)	A
A OR A	A
A OR (false)	A
A OR (true)	True (tingimusele vastavad kõik read)
A OR NOT (A)	True (tingimusele vastavad kõik read)
A OR (A AND B)	A
a>2 AND a>3	a>3
a LIKE '12%' AND a LIKE '1%'	a LIKE '12%'
a>2 OR a>3	a>2
a LIKE '12%' OR a LIKE '1%'	a LIKE '1%'
NOT (a>6)	a<=6
NOT (NOT(a))	a
a>6 AND 6<a	a>6
a +6-6>6	a>6
NOT(A OR B)	NOT(A) AND NOT(B)
NOT(A AND B)	NOT(A) OR NOT(B)
A OR (NOT(A) AND B)	A OR B
A AND NOT(B) OR B	A OR B
A AND B OR A AND NOT (B)	A
(A OR B) AND (A OR NOT(B))	A
A AND B OR A AND C	A AND (A OR B)
(A OR B) AND (A OR C)	A OR (A AND B)

Antud töös tahan uurida lihtsustamata tingimuste mõju päringute täitmisele ning andmebaasisüsteemide oskust tingimusi lihtsustada. Selleks, et see töö oleks võimalikult põhjalik, pean ma ka oma katsetes proovima erinevat tüüpi tingimusi *SELECT* lausete erinevates osades. Kõik tingimused on *WHERE* klauslites. Katsetan tingimuste

lihtsustamist nii siis, kui tingimus on põhipäringus kui ka siis, kui tingimus on alampäringus. Tabel 4 seab igale katsetes kasutatavale päringu tüübile vastavusse koodi, millele edaspidi tööd viidatakse.

Tabel 4. Testitavad päringute tüübid.

Tüübi kood	**Tüübi kirjeldus**
P1	Tingimus on põhipäringus ja sisaldab täpset otsingut (kestus=6)
P2	Tingimus on põhipäringus ja sisaldab mustri järgi otsingut (registri_nr LIKE ’1%’)
P3	Tingimus on põhipäringus ja sisaldab alampäringut (parkimine_id IN (SELECT parkimine_id FROM Teisaldamine))
P4	Tingimus on põhipäringus. Tegemist on liittingimusega, mille mõnda osa (lihttingimust) saaks süsteem lihtsustada.
P5	Tingimus on alampäringus ja sisaldab täpset otsingut (kestus=6)
P6	Tingimus on alampäringus ja sisaldab mustri järgi otsingut (registri_nr LIKE ’1%’)
P7	Tingimus on alampäringus ja sisaldab alampäringut (parkimine_id IN (SELECT parkimine_id FROM Teisaldamine))
P8	Tingimus on alampäringus. Tegemist on liittingimusega, mille mõnda osa (lihttingimust) saaks süsteem lihtsustada.

3.4 Andmebaasisüsteemile antavad korraldused

Järgnevalt antakse ülevaade eksperimendi käigus andmebaasisüsteemides kasutatud käskudest ning põhjendatakse lühidalt nende olulisust antud töös.

3.4.1 Statistika värskendamine

Tabel 5 esitab töös kasutavate statistika kogumise käsud. Värskendasin statistikat peale iga andmemuudatust, tagamaks, et andmebaasisüsteemidel on täitmisplaanide koostamiseks käepärast kõige värskem info.

Oracles kasutasin statistika kogumiseks *DBMS_STATS* paketis olevaid protseduure. Tabel 5 oleva näite korral analüüsitakse tabelit *Parkimine* ja kogutakse selles olevate andmete kohta statistikat (palju ridu, palju unikaalseid väärtuseid jne). Parameetrile

cascade väärtuse *true* andmine tähendab, et statistikat kogutakse ka tabelitega seotud indektiste kohta [24].

PostgreSQL korral kasutasin statistika kogumiseks käsku *VACUUM ANALYZE*. *VACUUM* eemaldab andmebaasist nö prügi ja *ANALYZE* värskendab statistikat [25]. Statistikat värskendatakse kogu andmebaasi kohta.

Tabel 5. Statistika kogumise käsud.

Andmebaasisüsteem	**Statistika kogumise käsk**
PostgreSQL	`VACUUM ANALYZE;`
Oracle	`Exec DBMS_STATS.GATHER_TABLE_STATS (ownname => 'C##TUD3' , tabname => 'Parkimine',cascade => true);`

3.4.2 Täitmisplaanide vaatamine

Tegemaks kindlaks, kas andmebaasisüsteem lihtsustas päringu korral tingimust või mitte on mul vaja vaadata selle päringu täitmisplaani. Täitmisplaanide vaatamiseks on võimalik kasutada nii andmebaasisüsteemidega suhtlemiseks mõeldud programmide graafilist kasutajaliidest kui ka andmebaasisüsteemi enda pakutavaid võimalusi.

Täitmisplaani vaatamiseks kasutatud andmebaasisüsteemide käsud on välja toodud järgnevas tabelis (vt Tabel 6).

Tabel 6. Täitmisplaani vaatamise käsud.

Andmebaasisüsteem	**Täitmisplaani vaatamise käsk**	**Selle käsu andmise järel väljastatakse**
PostgreSQL	`EXPLAIN ANALYZE {päring}`	SQL lause täitmiseks kasutatud täitmisplaan; plaani koostamiseks kulunud aeg; lause täitmiseks kulunud aeg
Oracle	`SET AUTOTRACE ON EXPLAIN; {päring}`	SQL lause tulemus; SQL lause täitmisplaan

Päringute filtri predikaadi automaatse lihtsustamise täitmisplaanide analüüsimisest kirjutatakse täpsemalt peatükis 4.

3.4.3 Hinnang tingimusele vastavate ridade arvule

Eraldi eksperimendina viin läbi töökiiruse uuringu. Selleks koostan keerukate liitingimustega päringud, milles toimub ka tabelite ühendamine. Liittingimustes kasutatavad alamtingimused valin järgmisel viisil. Kõikide tingimuste korral, mida *mõlemad* andmebaasisüsteemid ei lihtsusta, uurin andmebaasisüsteemi hinnangut tingimuse rakendamise operatsiooni tulemuses olevale ridade arvule. Hüpotees on, et lihtsustamata tingimused ajavad andmebaasisüsteemi segadusse, mistõttu hindab see tingimusele vastavate ridade arvu valesti. See võib omakorda mõjutada töökiirust, kui andmebaasisüsteem teeb valedest eeldustest lähtudes täitmisplaani koostamisel ebamõistlikke valikuid.

PostgresSQLis on *EXPLAIN ANALYZE* tulemustes näha nii andmbaasisüsteemi hinnang ridade arvule kui ka tegelik ridade arv.

Oracles kasutan *SELECT* klauslis vihjet. Seejärel käivitan teise lause, mis väljastab lause täitmisplaani ja statistika (sh ridade hinnanguline arv ja tegelik arv). Lauseid tuleb käivitada järjest [26].

Tabel 7 esitab ridade arvu hinnangu analüüsiks kasutatavad täitmisplaani vaatamise käsud.

Tabel 7. Ridade hinnangu täitmisplaani käsud.

Andmebaasisüsteem	**Täitmisplaani vaatamise käsk**
PostgreSQL	`EXPLAIN ANALYZE {päring};`

Oracle	`SELECT /*+ GATHER_PLAN_STATISTICS*/ *`
	`FROM tabelinimi`
	`WHERE tingimus;`
	`SELECT *`
	`FROM TABLE`
	`(DBMS_XPLAIN.DISPLAY_CURSOR(FORMAT=>'ALLSTATS LAST'));`

Ridade hinnangu alusel koostan töökiiruse uurimise päringud – kasutan nendes päringutes tingimusi, mille korral andmebaasisüsteemi hinnang sellele vastavale ridade arvule ning tegelik ridade arv on kõige rohkem erinevad. Mõõtmise eesmärgiks on välja selgitada, kas päringu lihtsustamata jätmine omab mõju töökiirusele.

3.4.4 Päringute täitmiseks kuluv aeg

Tabel 8 esitab andmebaasisüsteemides päringu täitmise aja mõõtmiseks kasutatavad käsud. Mõlemal juhul kasutasin käske, mille tulemusena ei tooda ekraanile päringu tulemust ning seega ei mõõda ajavõtt ka lause tulemuste väljastamiseks kulunud aega.

Tabel 8. Ajamõõtmise käsud.

Andmebaasisüsteem	**Aja mõõtmise käsk**
PostgreSQL	`EXPLAIN ANALYZE {päring};`

Oracle	`SET TIMING ON;`
	`SET AUTOTRACE TRACEONLY;`
	`{päring}`

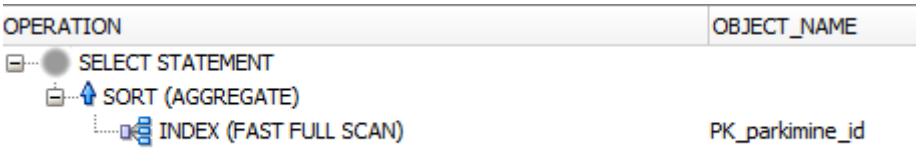
PostgreSQL korral väljastab *EXPLAIN ANALYZE* eraldi täitmisplaani koostamise ning täitmise ajad. Liidan need väärtused kokku.

Töökiiruse uurimisel käivitan igat uuritavat päringut kuus korda, millest esimest tulemust ei võta ma arvesse. Ülejäänud viie mõõtmise tulemustest arvutan nende geomeerilise keskmise. Kasutan geomeetrilist keskmist, sest nende mõõtmise tulemused on üksteisest sõltuvad. Operatsiooni korduva täitmise tulemusel valmistab süsteem ette ja jätab meelde lause täitmisplaani. Samuti loetakse mällu andmeid, mida pole järgnevatel täitmistel enam vaja teha [27]. Kirjeldatud asjaolud on ka põhjuseks, miks esimesel käivitamisel tulemust arvesse ei võeta – süsteem ei ole veel „soojaks“ läinud. Samuti ei tühjenda ma päringute käivitamise vahel muutmälu, et mitte tulemust moonutada, sest normaalolukorras süsteem seda samuti ei tee [28]. Mõõtmistulemustest kirjutatakse peatükis 4.

4 Eksperimendi tulemused ja järeldused

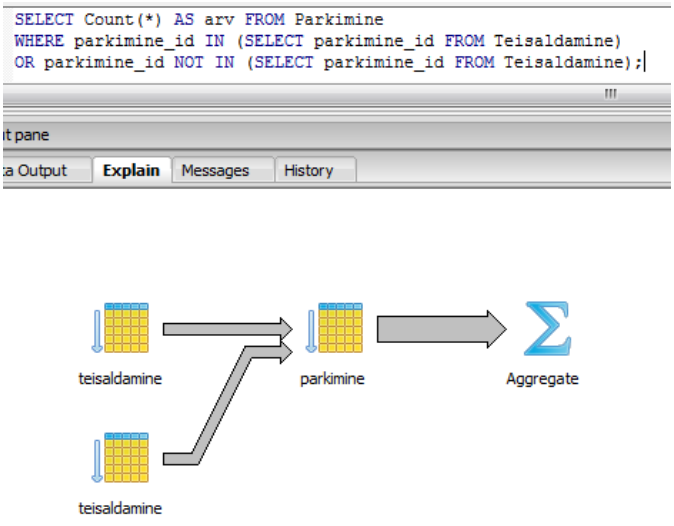
Käesolevas peatükis tuuakse välja, milliseid filtri predikaate e tingimusi oskasid andmebaasisüsteemid automaatselt lihtsustada. Eksperimendis kasutatavad lihtsustamata päringute tingimused on esitatud Lisa 3 (Tabel 17).

Joonis 8 on näide täitmisplaanist Oraclest (vt päringut Joonis 9). Oracle optimeerimismoodul on predikaati kirjutatud filtrit osanud lihtsustada ning päringu täitmisel on pöördutud ainult tabeli *Parkimine* poole.



Joonis 8. Lihtsustatud päringu tööplaan Oracles.

Joonis 9 esitab näite PostgreSQLi sama lause täitmisplaanist. Sellel korral ei osanud andmebaasisüsteem lause predikaadis olevat filtrit lihtsustada ning töötles seetõttu tabelit *Teisaldamine* ebavajalikult kaks korda.



Joonis 9. Lihtsustama päringu tööplaan PostgreSQLis.

4.1 Tingimuste lihtsustamine

Testpäringute (vt Lisa 3 Tabel 17) käivitamise tulemused on kokku kogutud üldistavatesse tabelitesse.

Tabel 9 kirjeldab andmebaasisüsteemides katsetatud päringute hulka erinevate päringu tüüpide korral. Samuti esitatakse iga päringu tüübi korral mitu protsenti moodustavad lihtsustatud päringud kokku katsetatud päringutest.

Tabel 9. Tingimuste lihtsustamise koondtulemused.

Tüübi kood	**Kokku katsetatud päringuid**	**PostgreSQL lihtsustas**	**Oracle lihtsustad**
P1	24	8 **(33%)**	15 **(63%)**
P2	21	6 **(29%)**	11 **(52%)**
P3	19	5 **(26%)**	8 **(42%)**
P4	13	4 **(31%)**	10 **(77%)**
P5	24	8 **(33%)**	15 **(63%)**
P6	21	6 **(29%)**	11 **(52%)**
P7	19	5 **(26%)**	8 **(42%)**
P8	13	4 **(31%)**	10 **(77%)**

Tabel 9 esitatud tulemustest on näha, et Oracle suutis päringu filtri predikaati kirjutatud filtrit automaatselt optimeerida poole rohkematel kordadel, kui seda suutis PostgreSQL. Kõige enam lihtsustas Oracle P4 ja P8 tüüpi päringusse kirjutatud tingimusi (77%) ja kõige vähem lihtsustati P7 tüüpi päringusse kirjutatud tingimusi (42%). PostgreSQL lihtsustas kõige rohkem P1 ja P5 tüüpi päringusse kirjutatud tingimusi (33%), kuid tulemused jäävast kõvasti alla Oracle poolt lihtsustatud tingimuste arvule. See, et andmebaasisüsteemid üldse tingimusi lihtustavad, näitab samas, et andmebaasisüsteemide arendajad peavad antud teemat oluliseks.

Jaotises 4.1.1 on kirjeldatud täpsemalt, milliseid tingimusi oskasid/ei osanud andmebaasisüsteemid lihtsustada.

4.1.1 Andmebaasisüsteemide tingimuste lihtsustamise võrdlemine

Antud töö tulemused on kasulikud andmebaasisüsteemide PostgreSQL ja Oracle andmebaasides päringute kirjutajale ning nendes päringute kirjutamise õppijatele ja õpetajatele. Sellest tulenevalt koostati tulemuste tabelid andmebaasisüsteemide kaupa.

Tabel 10 esitab tingimused, mida oskavad nii Oracle kui ka PostgreSQL lihtsustada. Tabelist on näha, et kõikide päringu tüüpide korral tundsid andmebaasisüsteemid ära DeMorgani seaduse. Samuti on tabelist näha, et tingimuse A OR A lihtsustamist esineb kõige rohkem.

Tabel 10. Tingimused, mida mõlemad andmebaasisüsteemid lihtsustasid.

P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**
A AND A	A OR A	A OR A	NOT(a>6)	A AND A	A OR A	A OR A	NOT(a>6)
A OR A	A OR (A AND B)	NOT (A OR B)	NOT (A OR B)	A OR A	A OR (A AND B)	NOT (A OR B)	NOT (A OR B)
A OR (A AND B)	NOT(NOT(a))	NOT (A AND B)	NOT(A AND B)	A OR (A AND B)	NOT(NOT(a))	NOT(A AND B)	NOT (A AND B)
NOT (a>6)	NOT (A OR B)			NOT (a>6)	NOT(A OR B)		A AND B OR A AND C
NOT (NOT(a))	NOT (A AND B)			NOT (NOT(a))	NOT (A AND B)		
NOT (A OR B)	A AND B OR A AND C			NOT (A OR B)	A AND B OR A AND C		
NOT (A AND B)				NOT (A AND B)			
A AND B OR A AND C				A AND B OR A AND C			

Tabel 11 kirjeldab tingimusi, mida PostgreSQL ja Oracle võrdluses ainult Oracle oskab lihtsustada. Tabelist on näha, et läbivalt tundis Oracle ära hästi tuntud loogikaavaldiste reegleid, näiteks A OR (true), A OR (false) ja A AND A.

Tabel 11. Tingimused, mida ainult Oracle lihtsustas.

P1	P2	P3	P4	P5	P6	P7	P8
A AND (true)	A AND A	A AND A	A AND A	A AND (true)	A AND A	A AND A	A AND (true)
A AND (A OR B)	A AND (true)	A AND (true)	A AND (true)	A AND (A OR B)	A AND (true)	A AND (true)	A OR A
A OR (false)	A AND (A OR B)	A OR (false)	A OR A	A OR (false)	A AND (A OR B)	A OR (false)	A OR (false)
A OR (true)	A OR (false)	A OR (true)	A OR (false)	A OR (true)	A OR (false)	A OR (true)	A OR (true)
a>2 AND a>3	A OR (true)	A OR NOT(A)	A OR (true)	a>2 AND a>3	A OR (true)	A OR NOT(A)	a>2 OR a>3
a>2 OR a>3			a>2 OR a>3	a>2 OR a>3			a>6 AND 6<a
a>6 AND 6<a			a>6 AND 6<a	a>6 AND 6<a			

Tabel 12 kirjeldab tingimusi, mida PostgreSQL ja Oracle võrdluses ainult PostgreSQL oskab lihtsustada. Tabelist on näha, et tingimuste arv, mida ainult PostgreSQL oskas lihtsustada, on võrreldes Oraclega palju väiksem. Siiski võrreldes Oraclega oskas PostgreSQL päringu tüübi P3 ja P7 korral lisaks lihtsustada loogikaalgebras tuntud neeldumisseadust: A OR (A AND B) ja päringu tüübi P3, P4 ja P7 korral distributiivuse seost A AND B OR A AND C.

Tabel 12. Tingimused, mida ainult PostgreSQL oskas lihtsustada.

P1	P2	P3	P4	P5	P6	P7	P8
-	-	A OR (A AND B)	A AND B OR A AND C	-	-	A OR (A AND B)	-
		A AND B OR A AND C				A AND B OR A AND C	

Tabel 13 esitab tingimused, mida mõlemad süsteemid ei osanud lihtsustada. Tingimustele, mida andmebaasisüsteemid lihtsustasid erinevalt oodatavast tulemusest, on lisatud juurde *.

Tabel 13. Tingimused, mida mõlemad andmebaasisüsteemid ei osanud lihtsustada.

P1	P2	P3	P4	P5	P6	P7	P8
A AND (false)	A AND (false)	A AND (false)	A OR (A AND B)	A AND (false)	A AND (false)	A AND (false)	A OR (A AND B)
A AND NOT A	A AND NOT A	A AND NOT A	A AND B OR A AND NOT(B)*	A AND NOT A	A AND NOT A	A AND NOT A	A AND B OR A AND NOT(B)*
A OR NOT (A)	A OR NOT (A)	A AND (A OR B)		A OR NOT (A)	A OR NOT (A)	A AND (A OR B)	
a +6-6>6	a LIKE '12%' AND a LIKE '1%'	NOT (NOT(a))		a +6-6>6	a LIKE '12%' AND a LIKE '1%'	NOT (NOT(a))	
A OR (NOT(A) AND B)	a LIKE '12%' OR a LIKE '1%'	A OR (NOT(A) AND B)		A OR (NOT(A) AND B)	a LIKE '12%' OR a LIKE '1%'	A OR (NOT(A) AND B)	
A AND NOT(B) OR B	A OR (NOT(A) AND B)	A AND NOT(B) OR B		A AND NOT(B) OR B	A OR (NOT(A) AND B)	A AND NOT(B) OR B	
(A OR B) AND (A OR NOT(B))	A AND NOT(B) OR B	A AND B OR A AND NOT(B)		(A OR B) AND (A OR NOT(B))	A AND NOT(B) OR B	(A OR B) AND (A OR C)	
(A OR B) AND (A OR C)	(A OR B) AND (A OR NOT(B))	(A OR B) AND (A OR NOT(B))		(A OR B) AND (A OR C)	(A OR B) AND (A OR NOT(B))	A AND B OR A AND NOT (B)*	
A AND B OR A AND NOT(B)*	(A OR B) AND (A OR C)	(A OR B) AND (A OR C)		A AND B OR A AND NOT(B)*	(A OR B) AND (A OR C)		
	A AND B OR A AND NOT(B)*				A AND B OR A AND NOT (B)*		

Tabel 13 on näha, et lihtsustamisel erinevate päringu tüüpide korral esineb sarnasusi. Filtri predikaadi tingimusi, mida andmebaasisüsteem ei optimeerinud põhipäringus (päringu tüübid P1-P4), ei optimeeritud ka alampäringutes (päringu tüübid P5-P8).

Mõlemad andmebaasisüsteemid ei osanud lihtsustada suurel hulgal loogikaavaldisi, mis põhjustab lausete täitmisel ebavajalike andmete lugemist. Lihtsustamise tulemused

päringu tüüpide kaupa on välja toodud Lisas 4 (vt Tabel 18–Tabel 25). Lihtsustamata päringute mõju töökiirusele on esitatud jaotises 4.2.

4.2 Päringute töökiirus

Töökiiruse päringutes kasutatakse liittingimusi, sh tingimusi, mis sisaldavad alampäringuid. Töökiiruse mõõtmise ülesande lahendamine koosneb järgmistest sammudest.

- Samm 1: Leia tingimused, millest panna kokku töökiiruse uurimise lausete liittingimused (vt jaotis Töökiiruse testpäringute tingimuste valimine 4.2.1).
- Samm 2: Koosta töökiiruse testpäringud (vt jaotis 4.2.1).
- Samm 3: Mõõta sammul 2 koostatud päringute töökiiruseid (vt jaotis 4.2.3).

4.2.1 Töökiiruse testpäringute tingimuste valimine

Töökiiruse mõõtmiseks kasutan ainult tingimusi, mida mõlemad andmebaasisüsteemid ei oska lihtsustada. Seega analüüsin ainult Tabel 13 esitatud tingimusi. Võtan aluseks täitmisplaanides esitatud hinnangud tingimusele vastavate ridade arvule ja tegeliku tingimusele vastava ridade arvu. Mida suurem on erinevus, seda rohkem ajab lihtsustamata tingimus andmebaasisüsteemi segadusse ja seda mõttekam on kasutada seda tingimust töökiiruse uurimisel.

Ridade arvu analüüsimisel leian tegeliku ja hinnangulise ridade arvu erinevuse absoluutväärtuse protsendi, arvutatuna sellest väärtusest (tegelik või hinnanguline suurus), mis parajastasti on suurem. Valem (1) esitab ridade arvu analüüsimisel kasutatud valemit.

$$\frac{Ridade\ hinnangu\ absoluutväärtus \times 100}{MAX(hinnang\ ridade\ arvule; tegelik\ ridade\ arv)} \quad (1)$$

Mida suurem on väärtus, seda suurem on suhteliselt erinevus tegelikkuse ja hinnangu vahel. Arvutuse võimalikud väärtused jäävad vahemikku 0 ja 100. Postgre ja Oracle ridade hinnangu arvulised väärtused on esitatud Lisa 5 (Tabel 26) PostgeSQLi protsentuaalne analüüs on esitatud Lisas 6 (Tabel 27) ja Oracle analüüs Lisa 7 (Tabel 28).

Tingimuste valimisel töökiiruse katsetamise päringutesse arvestasin asjaoluga, et *false* tingimusele vastavate ridade arv on alati null, seega Lisa 6 (Tabel 27) ja Lisa 7 (Tabel 28) esitatud protsentuaalseid erinevusi arvesse võtta ei olnud võimalik (nende erinevus on alati null). Seetõttu vaatasin selliseks lihtsustuva tingimusega päringu hinnangut ridade arvule ja valisin välja töökiiruse uurimiseks tingimuse, mille korral erinevus oli suurim.

Tingumuste välja valimisel arvestasin ka asjaoluga, et andmebaasid hindasid tingimustele vastavate ridade arvu erinevalt. Näiteks päringu `SELECT Count(*) AS arv FROM Parkimine WHERE registri_nr LIKE '12%' AND registri_nr LIKE '1%';` tegeliku ja hinnanguliste väärtuste protsentuaalne erinevus oli Postgres 82%, aga Oracles 0%. Seega ei valitud seda tingimust töökiiruse uurimiseks. Tabel 14 on välja toodud töökiiruse uurimiseks valitud lihtsustamata tingimused ning ridade hinnangu analüüsimisel kasutatud testpäringuid. Tabel esitab päringu tüübi, lihtsustamata päringu ning oodatava lihtsustamise tulemuse.

Tabel 14. Töökiiruse päringute tingimuste otsimiseks kasutatud päringud.

Päringu tüüp	**Lihtsustamata tingimus**	**Lihtsustamata päring**	**Oodatav tulemus**

P1	A AND (false)	`SELECT Count(*) AS arv`
		`FROM Parkimine`
		`WHERE kestus=6`
		`  AND (kestus<6`
		`  AND kestus>8`
			`SELECT Count(*) AS arv`
			`FROM Parkimine`
			`WHERE FALSE;`

P1	A OR NOT (A)	`SELECT Count(*) AS arv`
		`FROM Parkimine`
		`WHERE kestus=6`
		`  OR kestus<>6;`
			`SELECT Count(*) AS arv`
			`FROM Parkimine`
			`WHERE FALSE;`

P1	a+6-6>6	`SELECT Count(*) AS arv`
		`FROM Parkimine`
		`WHERE kestus +6-6>6;`
			`SELECT Count(*) AS arv`
			`FROM Parkimine`
			`WHERE kestus>6;`

P1	(A OR B) AND (A OR NOT(B))	`SELECT Count(*) AS arv`
		`FROM Parkimine`
		`WHERE (kestus=6`
		`  OR kestus=9)`
		`  AND (kestus=6`
		`  OR kestus<>9);`
			`SELECT Count(*) AS arv`
			`FROM Parkimine`
			`WHERE kestus=6;`

P2	a LIKE '12%' OR a LIKE '1%'	`SELECT Count(*) AS arv`
		`FROM Parkimine`
		`WHERE registri_nr LIKE '12%'`
		`  OR registri_nr LIKE '1%';`
			`SELECT Count(*) AS arv`
			`FROM Parkimine`
			`WHERE registri_nr LIKE '1%';`

4.2.2 Töökiiruse testpäringute koostamine

Töökiiruse uurimisel koostati testpäringud järgmise struktuuriga:

```
SELECT kliendi_kood,
       K.perenimi,
       P.alguse_aeg
FROM Klient AS K
INNER JOIN Parkimine AS P USING(kliendi_kood)
WHERE (mingid tingimused,
       mis piiravad kliendi andmeid)
  AND (mingid tingimused,
       mis piiravad parkimise andmeid);
```

Joonis 10. Töökiiruse testpäringu näide.

Andmebaasisüsteemide strateegia sellise lause täitmisel on rakendada kõigepealt *WHERE* klauslis olevad piirangud tabelite *Klient* ja *Parkimine* ridadele ning allesjäänud read ühendatakse kokku. Sellega viiakse kokkuühendatavate ridade arv võimalikult väikeseks. Töökiiruse lihtsustamata päringud ning nende oodatavad tulemused on kirjeldatud Lisas 8 (Tabel 29). Kasutasin nendes kombineeritult erinevaid tingimusi, mida andmebaasisüsteemid ei suutnud lihtsustada.

4.2.3 Päringu täitmiseks kuluv aeg

Päringute täitmise aja testi tulemused maksimum ridade arvu korral on kirjeldatud Tabel 15. Tabeli legend on järgmine: **L** – lihtsustamata päringu kood; **O** – lihtustamise tulemusest oodatava päringu kood (vt Lisa 8).

Mõõdeti ainult päringu täitmisele sh täitmisplaani koostamisele kulunud aega, st tähendab, et ei arvestatud tulemuste printimiseks kulunud aega. Printimisele kulunud aeg on lihtsustamata ja lihtsustatud tingimuse korral sama, sest tulemuseks on sama hulk andmeid. Seega uuriti ainult päringu täitmisele kulunud aega.

Tabel 15. Päringute täitmisele kulunud aeg.

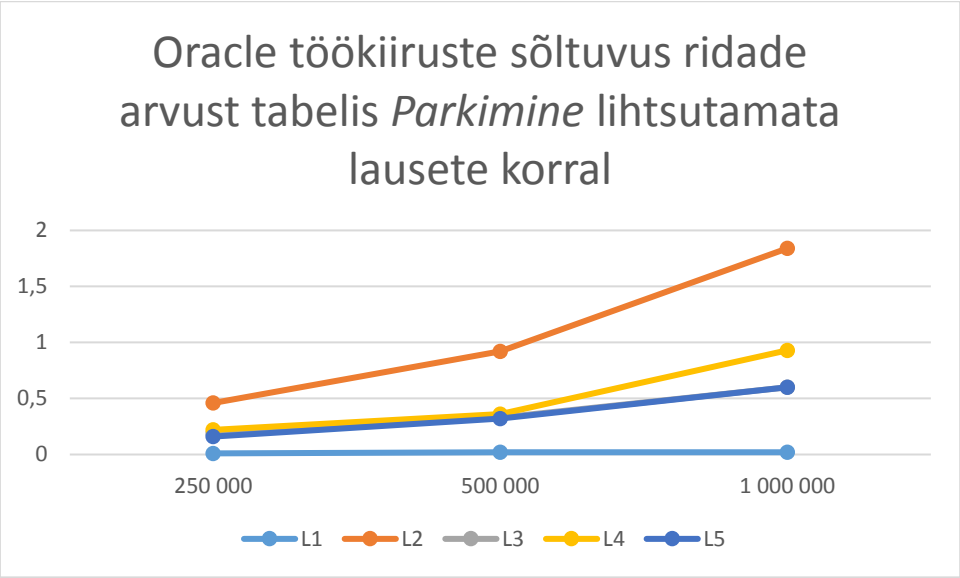
Lihtsustamata päringu kood	**Lihtsustamata päringu aeg (s)**	**Oodatava tulemuse kood**	**Ootatud tulemuse aeg (s)**	**Päringu täitmise aja vahe (s) lihtsustatud**
Oracle				
L1	0,02	**O1**	0,01	**0,01**
L2	1,84	**O2**	1,75	**0,09**
L3	0,60	**O3**	0,46	**0,14**

Lihtsustamata päringu kood	Lihtsustamata päringu aeg (s)	Oodatava tulemuse kood	Ootatud tulemuse aeg (s)	Päringu täitmise aja vahe (s) lihtsustatud
L4	0,93	**O4**	0,40	**0,53**
L5	0,60	**O5**	0,46	**0,14**
PostgreSQL				
L1	0,46	**O1**	0,0005	**0,4595**
L2	1,52	**O2**	1,12	**0,40**
L3	0,08	**O3**	0,08	**0,00**
L4	0,66	**O4**	0,13	**0,53**
L5	0,70	**O5**	0,57	**0,13**

Tulemustest on näha, et pea kõikide päringute täitmisajad on lihtsustatud päringu korral kiiremad, kui lihtsustamata päringu korral. Erandiks on PostgreSQLis lihtsustamata päring L3 – lihtsustatud päringu täitmiseks kulus keskmiselt sama palju aega. Mõlemas andmebaasisüsteemis oli kõige suurem erinevus päringute L4 ja O4 vahel (0,53 s). Kasutasin täitmisaegade vahet, et valida päringuid, mille täitmisplaane lähemalt uurida. Täitmisplaanide analüüsimisest on kirjutatud jaotises 4.3.

Mõõtsin töökiirust erinevate andmehulkade korral, eemärgiga välja selgitada, kas sõltuvus tabelis *Parkimine* oleva andmehulga ja töökiiruse vahel on lineaarne või mitte. Jämeda hinnangu saab anda graafikutelt (vt Oracle graafikut Joonis 11 ja Postgre graafikut Lisa 10 Joonis 23). Täpsema hinnangu andmiseks arvutati välja Pearsoni korrelatsiooni koefitsent (vt Tabel 16). Töökiiruste tulemused kaks ja neli korda väiksemate andmehulkade korral on välja toodud Lisas 9 (Tabel 30 ja Tabel 31).

Joonis 11 esitab seosed lihtsustamata päringute töökiiruste vahel Oracles. Jooniselt on näha, et tabelis *Parkimine* oleva andmehulga ja töökiiruse vahel on lineaarne seos. Sama kehtib ka PostgreSQL korral (vt Lisa 10 Joonis 23).



Joonis 11. Oracle töökiiruse seosed.

Tabel 16 esitatud tulemustest on näha, et tabelis *Parkimine* oleva andmehulga ja töökiiruse vahel on tugev lineaarne seos. Tugevaks lineaarseks seoseks loetakse, kui kõik korrelatsioonikordaja väärtused jäävad vahemikku [0,5; 1] või [-0,5; 1.0] – mida lähemal on väärtused ühele, seda tugevamalt on tunnused omavahel seotud [29].

Tabel 16. Lihtsustamata päringute töökiiruste seosed.

		Ridade arv tabelis Parkimine			
		250 000	**500 000**	**1 000 000**	**Korrelatsioonikordaja**
	Oracle				
Aeg (s)	**L1**	0,01	0,02	0,02	**0,755929**
	L2	0,46	0,92	1,84	**1**
	L3	0,19	0,33	0,60	**0,999959**
	L4	0,22	0,36	0,93	**0,989325**
	L5	0,16	0,32	0,60	**0,999424**
	PostgreSQL				
Aeg (s)	**L1**	0,13	0,25	0,46	**0,999424**
	L2	0,50	0,90	1,52	**0,997807**
	L3	0,02	0,04	0,08	**1**
	L4	0,18	0,36	0,66	**0,998906**
	L5	0,17	0,45	0,70	**0,975284**

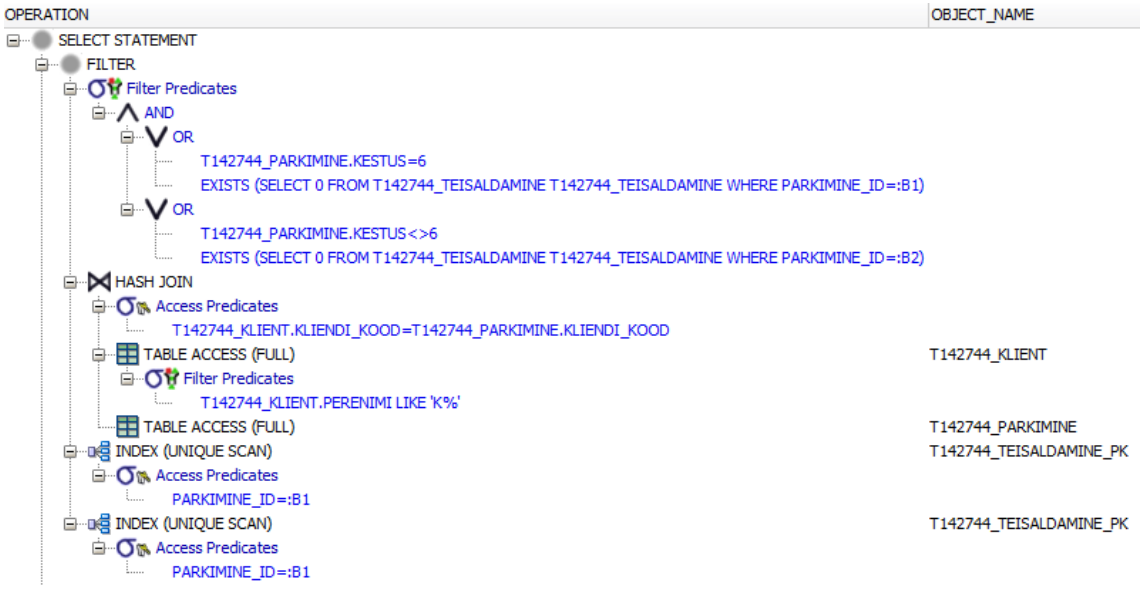
4.3 Täitmisplaanide analüüs

Vaatasin täitmisplaane lähemalt kahe päringute paari korral – ühte PostgreSQLis ja ühte Oracles. Lähtusin valikul töökiiruse uuringust kõige suurema ridade arvu korral (vt Tabel 15). Valisin uurimiseks päringu L4, mille korral nii PostgreSQLis kui Oracles täitmisaeg lihtsustatud ning lihtsustamata päringu korral erines kõige rohkem. Mõlemal juhul on lihtsustamata ja lihtsustatud lausete täitmisplaanid erinevad – ja mitte ainult päringu filtri predikaadi poolest, st andmebaasisüsteem kasutab lihtsustamata ja lihtsustatud tingimustega lausete puhul erinevates tabelites olevate andmete kokkupanemiseks erinevat strateegiat.

Lihtsustamata päring L4 (Joonis 12)**:**

```
SELECT kliendi_kood,
       T142744_Klient3.perenimi,
       T142744_Parkimine3.algus
FROM T142744_Klient3
INNER JOIN T142744_Parkimine3 USING(kliendi_kood)
WHERE T142744_Klient3.perenimi LIKE 'K%'
  AND ((T142744_Parkimine3.parkimine_id IN
         (SELECT parkimine_id
          FROM T142744_Teisaldamine3))
       OR T142744_Parkimine3.kestus=6)
  AND ((T142744_Parkimine3.parkimine_id IN
         (SELECT parkimine_id
          FROM T142744_Teisaldamine3))
       OR T142744_Parkimine3.kestus<>6);
```

Joonis 12. Lause L4.



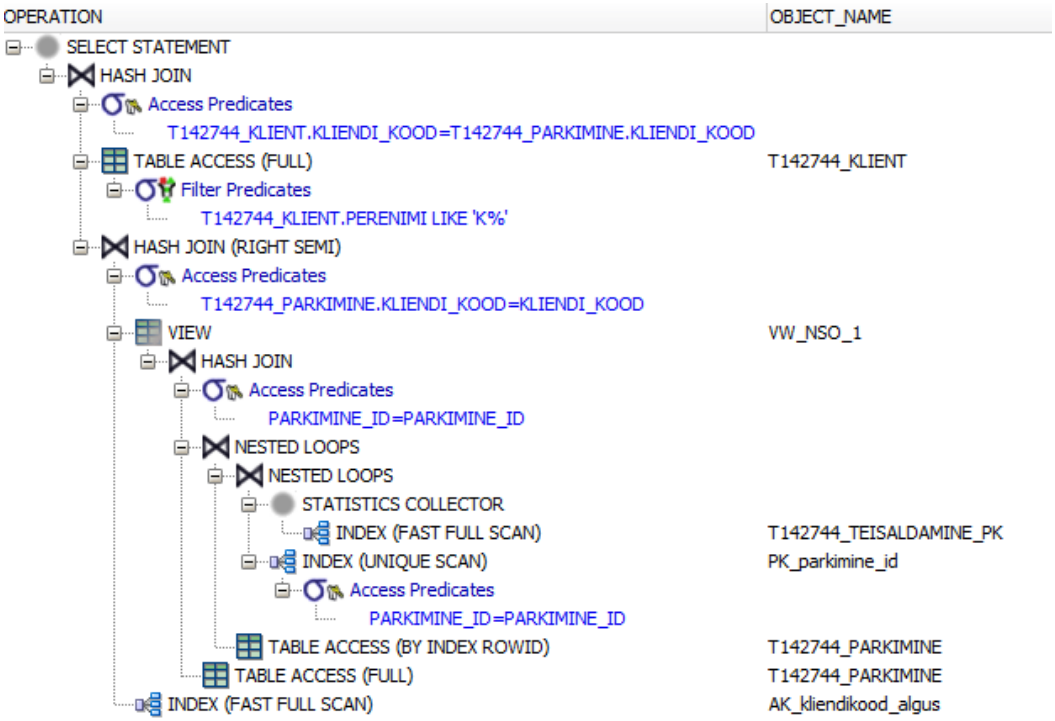
Joonis 13. Oracle täitmisplaan lihtsustamata päringu korral.

Oracle täitmisplaani selgitus (Joonis 13)**:** Joonis 12 esitatud päringu alusel leitakse kliendi perenime tingimusele vastavate parkimiste hulk. Tulemust piiratakse tabelile *Parkimine* rakendatud tingimuse alusel. Sealjuures loetakse tabelile *Teisaldamine* loodud indeksit kaks korda.

Oodatav tulemus O4 (Joonis 14)**:**

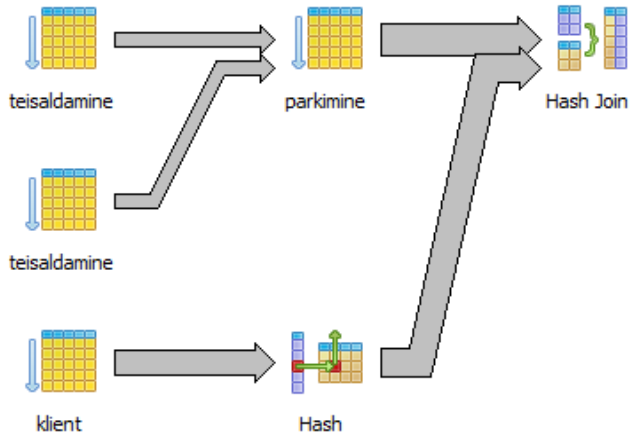
```
SELECT kliendi_kood,
       T142744_Klient3.perenimi,
       T142744_Parkimine3.algus
FROM T142744_Klient3
INNER JOIN T142744_Parkimine3 USING(kliendi_kood)
WHERE T142744_Klient3.perenimi LIKE 'K%'
  AND T142744_Parkimine3.parkimine_id IN
    (SELECT parkimine_id
     FROM T142744_Teisaldamine3);
```

Joonis 14. Lause O4.



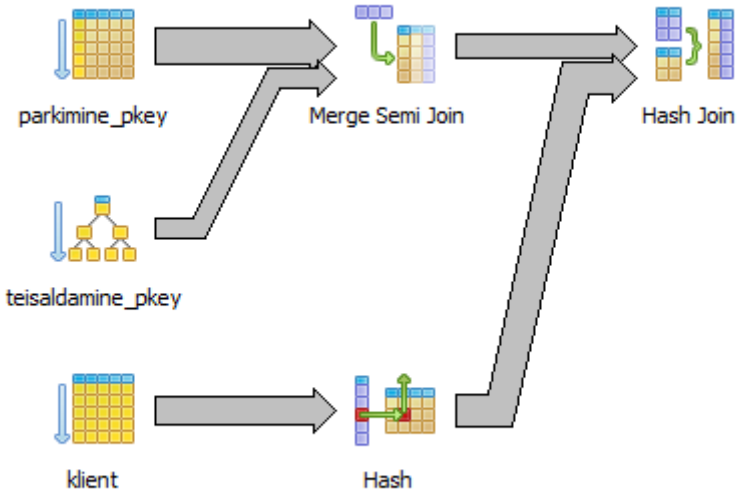
Joonis 15. Oracle täitmisplaan lihtsustatud päringu korral.

Oracle täitmisplaani selgitus (Joonis 15)**:** Joonis 14 esitatud päringu alusel leitakse teisaldamisega parkimised ning tulemus ühendatakse kokku perenime tingimusele vastavate klientidega. Erinevalt lihtsustamata lausest pole vaja tabelit *Teisaldamine* kaks korda lugeda. Lihtsustamine ei muutnud seda, et parkimiste ja klientide andmed ühendati kokku hash join algoritmi alusel. Seega antud juhul ei sundinud lihtsustamata tingimus süsteemi teistsugust tabelite ühendamise algoritmi valima, kuid muutus ühendamiste järjekord.



Joonis 16. PostgreSQLi täitmisplaan lihtsustamata päringu korral.

PostgreSQLi täitmisplaani selgitus (Joonis 16): Joonis 12 esitatud päringu alusel leitakse perenime tingimusele vastavad kliendid. Tabelist *Parkimine* tingimusele vastavate ridade otsimisel loetakse tabelit *Teisaldamine* kaks korda. Tulemus ühendatakse kokku.



Joonis 17. PostgreSQL täitmisplaan lihtsustatud päringu korral.

PostgreSQL täitmisplaani selgitus (Joonis 17): Joonis 14 esitatud päringu alusel leiab süsteem teisaldamisega parkimised ning tulemus ühendatakse kokku perenime tingimustele vastavate klientidega. Seejuures on näha, et erinevalt lihtsustamata lausest pole vaja tabelit *Teisaldamine* kaks korda lugeda. Tabelist *Teisaldamine* ja *Parkimine* andmete lugemise asemel loeb süsteem nendele tabelitele loodud indekseid (justkui „kõhnem“ versioon tabelitest), mitte ei kasuta tabelite täielikku läbiskaneerimist nagu lihtsustamata variandi korral. Lihtsustamine ei muutnud seda, et parkimiste ja klientide andmed ühendati kokku hash join algoritmi alusel. Seega antud juhul ei sundinud lihtsustamata tingimus süsteemi teistsugust tabelite ühendamise algoritmi valima.

5 Tulemuste analüüs

Käesolevas peatükis võetakse lühidalt kokku peatükis 4 leitud tulemused. Mõlemad andmebaasisüsteemid oskasid mingil määral päringu filtri predikaati kirjutatud tingimusi lihtsustada, koondtabelid on välja toodud jaoises 4.1.1. Iga katsetatud päringu tüübi P1-P8 korral oskas Oracle võrreldes PostgreSQLga päringuid lihtsustada poole rohkem.

Lihtsustamata tingimusi katsetati nii põhipäringutes kui ka alampäringutes. Tulemuste analüüsimisel selgus, et kui andmebaasisüsteem oskas põhipäringusse kirjutatud filtri predikaati e tingimust lihtsustada, siis lihtsustati seda ka alampäringus. Kui põhipäringusse kirjutatud filtri predikaati lihtsustada ei osatud, siis seda ei tehtud ka alampäringus.

PostgeSQLi korral katsetasin ka tingimuste lihtsustamise võimekust, kui juhtparameetri *constraint_exclusion* väärtus oli *on.* See aga ei omanud tulemustele suurt mõju. Iga päringu tüübi korral lisandus *constraint_exclusion* väärtusele *on* andmine juurde maksimaalselt 1-2 tingimust, mida PostgreSQL oskas lihtsustada. Tulemused on esitatud Lisas 4, kus on esitatud Oraclet ja PostgeSQLi võrdlevad tabelid päringu tüüpide kaupa.

Käesoleva töö tulemuste põhjal ei saa väita, et Oracle optimeeriks päringuid paremini kui PostgreSQL, sest valdkonda on uuritud liiga kitsalt. Lihtsustatud päringute korral on nende täimiseks kulunud aeg üldiselt väiksem, kui lihtsustamata päringute korral. Seega võib öelda, et päringu filtri predikaadi lihtsustamine on kasulik. Nagu öeldud, paraku ei pruugi andmebaasisüsteemid lihtsustamisel appi tulla. Lihtsustamata predikaadid halvendavad inimeste jaoks koodi arusaadavust ning seega ka koodi hallatavust ning edasiarendatavust. Seega peaks päringute kirjutajad üritama ikkagi juba ise kirjutada võimalikult lihtsat ja puhast koodi.

Lihtsustamata ja lihtsustatud päringu täitmisaegade tulemustest oli kõige suurem ajavahe loogikaalgebras tuntud neeldumise *(x v xy = x)* lihtsustamisel. Töö tulemustest selgub, et mõlemad andmebaasisüsteemid jäid hätta mitmete filtri predikaati kirjutatud tingimuste lihtsustamisel, mis omakorda suurendas seda predikaati kasutava päringu töökiirust.

Antud töö käigus mõõdeti lihtsustatud ja lihtsustamata päringute töökiiruseid erinevate andmehulkade korral. Selgus, et ridade hulga suurenemisel, suureneb ka päringu töötlemisele kuluv aeg lineaarselt.

6 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli lähemalt uurida, kui palju oskavad kaks populaarset SQL-andmebaasisüsteemi (Oracle ja PostgreSQL) päringu (*SELECT* lausete) filtri predikaate automaatselt lihtsustada ning kui palju mõjutab lihtsustamata tingimuste kasutamine päringute töökiirust. Filtri predikaadid kirjutatakse päringus *WHERE* või *HAVING* klauslisse, kuid viimase saab lause ümberkirjutamise järel samuti asendada *WHERE* klausliga.

Eesmärkide saavutamiseks projekteeriti lihtne parkimiste teemaline andmebaas ning realiseeriti see kahes SQL-andmebaasisüsteemis: Oracle (versioon Database 12c Enterprise edition Release 12.1.0.1.0) ja PostgreSQL (versioon 9.5.3). Andmebaasisüsteemid valiti, kuna need on populaarsed, autorile kättesaadavad ning autoril on nendega varasemaid kogemusi. Andmebaasidesse genereeriti testandmeid, kasutades nii olemasolevat veebipõhist tarkvara kui ka MS Excelit.

Kokku katsetati töös kaheksat erinevat päringu tüüpi ning 26 erinevat loogikaavaldise (filtri predikaadi) tüüpi. Nendest moodustati kokku 154 päringut, kus oleks võimalik filtri predikaati lihtsustada. Lihtsustamise kindlakstegemiseks uuriti päringute täitmisplaane. Nendest katsetatud lausetest suutis Oracle lihtsustad 88 ja PostgreSQL 46 lause filtri predikaati.

Teise eesmärgi saavutamiseks mõõdeti lausete täitmisaegu, kui päringusse kirjutatud tingimus on lihsustamata ja lihtsustatud. Selleks hinnati esmalt nii lihtsustatud kui ka lihtsustamata tingimuste korral andmebaasisüsteemide hinnangut tingimustele vastavate ridade arvule. Tingimustest, mille korral hinnanguline ridade arv ja tegeliku ridade arvu erinevus oli suurim, moodustati töökiiruse uurimiseks testpäringud. Töökiiruse katsed viidi läbi viie päringuga. Ootuspäraselt selgus, et lihtsustamata tingimus päringu *WHERE* klauslis suurendab päringu täitmisele kuluvat aega.

Töökiirust mõõdeti erinevate andmehulkade juures. Eesmärgiks oli teada saada, kas sõltuvus tabelis *Parkimine* oleva andmehulga ja töökiiruse vahel on lineaarne või mitte. Selgus, et andmete hulga kasvamisel, kasvab ka päringutele kuluv aeg lineaarselt.

Töö tulemusena leian, et lihtsustamata tingimused päringutes suurendavad nende päringute täitmisele kuluvat aega. Seega peaksid andmebaasisüsteemides PostgreSQL ja Oracle päringute kirjutavad pöörama tähelepanu loodud lausete tingimustele, et lausete täitmiseks kuluks võimalikult vähe aega.

Edasisteks uurimissuundadeks võiks olla uuringu laiendamine suuremale hulgale andmebaasisüsteemidele ning põhjalikumad töökiiruse uuringud, kui päringud peavad küsima andmeid veel suuremast hulgast tabelitest, kuid päringutes on lihtsustamata tingimused.

Kasutatud kirjandus

[1] Bigcommerce, „What is a CSV file and how do I save my spreadsheet as one?,“ [Võrgumaterjal]. Available: https://support.bigcommerce.com/articles/Public/What-is-a-CSV-file-and-how-do-I-save-my-spreadsheet-as-one. [Kasutatud 21.05.2017.].

[2] Ü. Kaasik, Matemaatikaleksikon, Eesti Entsüklopeediakirjastus, 2001.

[3] H. Lensen ja M. Kruus, Diskreetne matemaatika, Tallinn: Tallinna Raamatutrükikoda, 2012.

[4] E. Eessaar, „Teema 3. Relatsioonialgebra. Sissejuhatus SQL keelde,“ Tallinn, 2012.

[5] w3schools, „SQL Tutorial,“ [Võrgumaterjal]. Available: https://www.w3schools.com/sql/DEfaULT.asP. [Kasutatud 10.05.2017.].

[6] E. Eessaar, „Teema7. SQL andmekäitluskeele lausete töötlemine ja optimeerimine,“ Tallinn, 2016.

[7] Basis International, „SQL Optimization,“ [Võrgumaterjal]. Available: http://legacy.basis.com/support/tips/sqloptimization.html. [Kasutatud 08.05.2017.].

[8] Microsoft, „SQL Statement Processing,“ [Võrgumaterjal]. Available: https://technet.microsoft.com/en-us/library/ms190623(v=sql.105).aspx. [Kasutatud 10.05.2017.].

[9] TechOnTheNet, „SQL: Comparison Operators,“ [Võrgumaterjal]. Available: https://www.techonthenet.com/sql/comparison_operators.php. [Kasutatud 11.05.2017.].

[10] K. Vorwerk ja G. N. Paulley, „On Implicate Discovery and Query Optimization,“ *Database Engineering and Applications Symposium*, 2002.

[11] S. Brass ja C. Goldberg, „Detecting Logical Errors in SQL Queries,“ Grundlagen von Datenbanken, Halle, 2004.

[12] S. Brass ja C. Goldberg , „Semantic errors in SQL queries: A quite complete list.,“ *Journal of Systems and Software 79.5,* pp. 630-644, 2006.

[13] T. Connolly ja C. Begg, Database systems : a practical approach to design, implementation, and management, 3rd toim., Harlow: Addison-Wesley: Pearson Education, 2002.

[14] P. Belknap, „Hybrid optimization strategies in automatic SQL tuning,“ U.S. Patent No. 7, 2011.

[15] G. Fritchery, „Simple Talk,“ 11.05.2008.. [Võrgumaterjal]. Available: https://www.simple-talk.com/sql/performance/execution-plan-basics/. [Kasutatud 12.05.2017.].

[16] Microsoft, „Which is Faster: Index Access or Table Scan?,“ [Võrgumaterjal]. Available: https://technet.microsoft.com/en-us/library/aa224773(v=sql.80).aspx. [Kasutatud 12.02.2017.].

[17] Microsoft, „Create Clustered Indexes,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/create-clustered-indexes. [Kasutatud 17.04.2017.].

[18] V. e. a. Leis, „How good are query optimizers, really?,“ Proceedings of the VLDB Endowment 9.3, 2015.

[19] DB-Engines, „DB-Engines Ranking of Relational DBMS,“ 2017. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking/relational+dbms. [Kasutatud 02.02.2017.].

[20] „Generate test data for your database,“ [Võrgumaterjal]. Available: http://www.databasetestdata.com/. [Kasutatud 20.04.2017.].

[21] Tallinn, „Tallinna elanike arv,“ [Võrgumaterjal]. Available: http://www.tallinn.ee/est/Tallinna-elanike-arv. [Kasutatud 20.04.2017.].

[22] Statistikaamet, „Sõiduautode arv 1000 elaniku kohta,“ [Võrgumaterjal]. Available: https://www.stat.ee/34300. [Kasutatud 20.05.2017.].

[23] ElectronicsTutorials, „Laws of Boolean Algebra,“ [Võrgumaterjal]. Available: http://www.electronics-tutorials.ws/boolean/bool_6.html. [Kasutatud 02.02.2017.].

[24] Burleson Consulting, „dbms_stats.gather_table_stats tips,“ Burleson Consulting, 17.05.2015. [Võrgumaterjal]. Available: http://www.dba-oracle.com/t_dbms_stats_gather_table_stats.htm. [Kasutatud 01.05.2017.].

[25] PostgreSQL, „VACUUM,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/9.5/static/sql-vacuum.html. [Kasutatud 01.05.2017.].

[26] M. Colgan, „How do I know if the cardinality estimates in a plan are accurate?,“ Oracle, 03.08.2011.. [Võrgumaterjal]. Available: https://blogs.oracle.com/optimizer/how-do-i-know-if-the-cardinality-estimates-in-a-plan-are-accurate. [Kasutatud 01.05.2017.].

[27] C. Gallant, „What is the difference between arithmetic and geometric averages?,“ 2017.. [Võrgumaterjal]. Available: http://www.investopedia.com/ask/answers/06/geometricmean.asp. [Kasutatud 05.17.2017.].

[28] T. Kyte, „On Procedures, Flushes, and Writes,“ July/August 2003. [Võrgumaterjal]. Available: http://www.oracle.com/technetwork/issue-archive/o43asktom-094944.html. [Kasutatud 01.05.2017.].

[29] „Pearson Correlation: Definition and Easy Steps for Use,“ [Võrgumaterjal]. Available: http://www.statisticshowto.com/what-is-the-pearson-correlation-coefficient/. [Kasutatud 01.05.2017.].

[30] ExtendOffice, „How To Quickly Generate Random Time In Excel?,“ [Võrgumaterjal]. Available: https://www.extendoffice.com/documents/excel/1781-excel-generate-random-time.html. [Kasutatud 20.04.2017.].

[31] ExtendOffice, „How To Generate Random Character Strings In A Range In Excel?,“ [Võrgumaterjal]. Available: https://www.extendoffice.com/documents/excel/642-excel-generate-random-string.html. [Kasutatud 20.04.2017.].

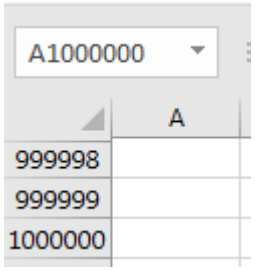
Lisa 1 – PostgreSQLi tabelite loomise laused

```
CREATE TABLE Klient(
  kliendi_kood SERIAL NOT NULL,
  eesnimi VARCHAR(255) NOT NULL,
  perenimi  VARCHAR(255) NOT NULL,
  CONSTRAINT PK_kliendi_kood PRIMARY KEY (kliendi_kood)
);

CREATE TABLE Parkimine(
  parkimine_id SERIAL NOT NULL,
  kliendi_kood integer NOT NULL,
  algus    timestamp    without    time    zone    NOT    NULL    DEFAULT
('now'::text)::timestamp(0) without time zone,
  kestus integer NOT NULL,
  registri_nr  VARCHAR(255) NOT NULL,
  CONSTRAINT PK_parkimine_id PRIMARY KEY (parkimine_id),
  CONSTRAINT AK_kliendi_kood UNIQUE (kliendi_kood, algus),
  CONSTRAINT  FK_kliendi_kood  FOREIGN  KEY  (kliendi_kood)  REFERENCES
Klient (kliendi_kood)

);

CREATE TABLE Teisaldamine
(
  parkimine_id integer NOT NULL,
  teisaldamise_aeg  timestamp  without  time  zone  NOT  NULL  DEFAULT
('now'::text)::timestamp(0) without time zone,
  PRIMARY KEY (parkimine_id),
  CONSTRAINT  FK_parkimine_id  FOREIGN  KEY  (parkimine_id)  REFERENCES
Parkimine (parkimine_id)
);
```

Joonis 18. PostgreSQL tabelite loomise laused

Lisa 2 – Testandmete genereerimine Excelis

Andmete genereerimist alustasin veerust *parkimine_id*, kus on väärtused vahemikus 1-1 000 000. Selleks tuli esmalt liikuda Excelis reale 1 000 000 (vt Joonis 19). Seejärel on vajutati klahvikombinatsiooni *ctrl+shift* , millega aktiveeriti read üks kuni miljon. Peale ridade aktiveerimist võib vajutada mistahes klahvile, seejärel kombinatsioni *ctrl+enter,* et paljundada ajutisi väärtuseid ridadesse üks kuni miljon. Antud toiming on vajalik, et hiljem oleks võimalik mugavalt valemeid paljundada ülevalt alla ilma käsitsi miljoninda rea otsimist.



Joonis 19. Excelis reale 1 000 000 liikumine.

Veergu *parkimine_id* jaoks sisestati lahtrisse järjestikused väärtused 1 kuni 1 000 000. Seda võib teha järgmiselt: esmalt lisada väärtused 1 ja 2 lahtritesse käsitsi, seejärel liita alati eelmisele lahtrile +1 juurde ning paljundada valemit kuni reani 1 000 000.

Veergu *kliendi_kood* lisati esmalt väärtused 1st kuni 200 000ni järjest, seejärel kasutati Exceli funktsiooni *randbetween* (vt Joonis 20).

```
RANDBETWEEN(1;200000)
```

Joonis 20. Randbetween funktsioon.

Väiksemate ridade arvude korral tuleb tabelisse *Parkimine* ja *Teisaldamine* valida veeru *kliendi_kood* väärtused vastavalt tabelis *Klient* olevatele väärtusetele.

Parkimise algus kuupäeva genereeriti valemiga [30] (vt Joonis 21):

```
TEXT(
(RAND()*("2017-04-10 00:00:00"-"2017-04-3 00:00:00")+"2017-0-3
00:00:00");"yyyy-mm-dd h:m:s")
```

Joonis 21. Kuupäeva genereerimine Excelis.

Joonis 21 esitatud valemis tähistab alguse kuupäeva 2017-04-3 ning lõpu kuupäeva 2017-04-10.

Sarnaselt kliendi_koodile leidmisele, genereeriti veergu *kestus* väärtused *randbetween* funktsiooniga. Väärtused valiti vahemikus 1 kuni 72, mis tähendab, et üks parkimine ei saa kesta vähem kui tund aega ega rohkem kui 72 tundi. Reaalses elus võib parkimine kesta ka vähem või rohkem, kui ette antud piirangud, kuid käesoleva töö raames ei ole see oluline.

Joonis 22 esitab valemi veergu *registri_nr* väärtuste genereerimise jaoks [31]. Lähtuti lihtsustatud eeldusest, et kõik registri numbrid on Eestis välja antud ja järgivad sama formaati. Funktsioon *randbetween(100;999)* on juhusliku kolme numbri genereerimise jaoks, millele liidetakse juurde kolm juhuslikult genereeritud tähte. Funktsioon *char(randbetween(65;90))* on juhusliku tähestiku tähe vahemikus A-Z genereerimise jaoks.

```
RANDBETWEEN(100;999)&CHAR(RANDBETWEEN(65;90))&CHAR(RANDBETWEEN(65;90))
&CHAR(RANDBETWEEN(65;90))
```

Joonis 22. Veergu registri_nr väärtuse genereerimine

Lisa 3 – Töös uuritavad päringud

Kui tingimus oli põhipäringus (päringu tüübid P1-P4), siis kasutati testimiseks lauset üldkujul: `SELECT Count(*) AS arv FROM Parkimine WHERE <lihtsustamata tingimus>;`.

Kui tingimus oli alampäringus (päringu tüübid P5-P8), siis kasutati testimiseks lauset üldkujul: `SELECT Count(*) AS arv FROM Klient WHERE kliendi_kood IN (SELECT kliendi_kood FROM Parkimine WHERE <lihtsustamata tingimus>);`

Tabel 17. Töös uuritavad päringud.

Lihtsustamata tingimus (üldkuju kohta tuuakse välja ka lihtsustus)
A AND A => A
`kestus=6 AND kestus=6`
`registri_nr LIKE '1%' AND registri_nr LIKE '1%'`
`parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)`
`(kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)) AND (kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine))`
A AND (false) => false
`(kestus<6 AND kestus>8);`
`registri_nr LIKE '1%' AND (kestus<6 AND kestus>8);`
`parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND (kestus<6 AND kestus>8);`
A AND (true) =>A
`kestus=6 AND (kestus>=10 OR kestus<10);`
`registri_nr LIKE '1%' AND (kestus>=10 OR kestus<10);`
`parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND (kestus>=10 OR kestus<10);`
A AND NOT A => false
`kestus=6 AND kestus<>6`
`registri_nr LIKE '1%' AND registri_nr NOT LIKE '1%'`
`parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND parkimine_id NOT IN (SELECT parkimine_id FROM Teisaldamine)`
A AND (A OR B) => A
`kestus=6 AND (kestus=6 OR kestus=8)`
`registri_nr LIKE '1%' AND (registri_nr LIKE '1%' OR registri_nr LIKE '2%')`

parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND ((parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR kestus=6))
A OR A => A
kestus=6 OR kestus=6
registri_nr LIKE '1%' OR registri_nr LIKE '1%'
parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)
(kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)) OR (kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine))
A OR (false) => A
kestus=6 OR (kestus<6 AND kestus>8)
registri_nr LIKE '1%' OR (kestus<6 AND kestus>8)
parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR (kestus<6 AND kestus>8)
(kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)) OR (kestus<6 AND kestus>7))
A OR (true) => true
kestus=6 OR (kestus>=10 OR kestus<10)
registri_nr LIKE '1%' OR (kestus>=10 AND kestus<10)
parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR (kestus>=10 OR kestus<10)
(kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)) OR (kestus>=10 OR kestus<10)
A OR NOT (A) => true
kestus=6 OR kestus<>6
registri_nr LIKE '1%' OR registri_nr NOT LIKE '1%'
parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR parkimine_id NOT IN (SELECT parkimine_id FROM Teisaldamine)
A OR (A AND B) => A
kestus=6 OR (kestus=6 AND kestus=8)
registri_nr LIKE '1%' OR (registri_nr LIKE '1%' AND registri_nr LIKE '2%'
parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR (parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND kestus=6);
((kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine))) OR ((kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine))AND (kestus=9 OR registri_nr LIKE '7%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)))
a>2 AND a>3 => a>2
kestus>2 AND kestus>3
(kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine WHERE parkimine_id>2 AND parkimine_id>3))
a>2 OR a>3 => a>3
kestus>2 OR kestus>3
(kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine WHERE parkimine_id>2 OR parkimine_id>3))

NOT (a>6) => a<=6
NOT (kestus>6)
parkimine_id IN (SELECT parkimine_id FROM Teisaldamine WHERE NOT parkimine_id>6)
NOT (NOT(a)) => a
NOT (NOT(kestus>6))
NOT registri_nr NOT LIKE '1%'
parkimine_id IN (SELECT parkimine_id FROM Teisaldamine WHERE parkimine_id NOT IN (SELECT parkimine_id FROM Teisaldamine WHERE parkimine_id NOT IN (SELECT parkimine_id FROM Teisaldamine)))
a>6 AND 6<a => a>6
kestus>6 AND 6<kestus
(kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM T142744_Teisaldamine WHERE parkimine_id>6 AND 6<parkimine_id))
a +6-6>6 => a>6
kestus +6-6>6
a LIKE '12%' AND a LIKE '1%' => a LIKE '12%'
registri_nr LIKE '12%' AND registri_nr LIKE '1%'
a LIKE '12%' OR a LIKE '1%' => '1%'
registri_nr LIKE '12%' OR registri_nr LIKE '1%'
NOT(A OR B) => NOT(A) AND NOT(B)
NOT(kestus=6 AND kestus=9)
NOT (registri_nr LIKE '1%' OR registri_nr LIKE '2%')
NOT (parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR kestus=8)
NOT ((kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)) OR (kestus=9 OR registri_nr LIKE '7%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)))
NOT(A AND B) => NOT(A) OR NOT(B)
NOT(kestus=6 OR kestus=9)
NOT(registri_nr LIKE '1%' AND registri_nr LIKE '2%')
NOT (parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND kestus=8)
NOT ((kestus=8 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)) AND (kestus=9 OR registri_nr LIKE '7%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)))
A OR (NOT(A) AND B) => A OR B
kestus=6 OR (kestus<>6 AND kestus=10)
registri_nr LIKE '1%' OR (registri_nr NOT LIKE '1%' AND registri_nr LIKE '2%')
parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR ((parkimine_id NOT IN (SELECT parkimine_id FROM Teisaldamine)) AND kestus=6)
A AND NOT(B) OR B => A OR B
kestus=6 AND kestus<>10 OR kestus=10
registri_nr LIKE '1%' AND registri_nr NOT LIKE '2%' OR registri_nr LIKE '2%'
parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND kestus<>6 OR kestus=6

A AND B OR A AND NOT (B) => A
`kestus=6 AND kestus=9 OR kestus=6 AND kestus<>9`
`registri_nr LIKE '1%' AND registri_nr LIKE '2%' OR registri_nr LIKE '1%' AND registri_nr NOT LIKE '2%'`
`parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND kestus=6 OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND kestus<>6;`
(A OR B) AND (A OR NOT(B)) => A
`(kestus=6 OR kestus=9) AND (kestus=6 OR kestus<>9)`
`(registri_nr LIKE '1%' OR registri_nr LIKE '2%') AND (registri_nr LIKE '1%' OR registri_nr NOT LIKE '2%')`
`(parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)OR kestus=6) AND (parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR kestus<>6)`
A AND B OR A AND C => A AND (B OR C)
`kestus=6 AND registri_nr='111KLO' OR kestus=6 AND registri_nr='316CID'`
`registri_nr LIKE '1%' AND registri_nr LIKE '2%' OR registri_nr LIKE '1%' AND registri_nr LIKE '3%'`
`parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)AND kestus=6 OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND kestus=8`
`(kestus=6 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)) AND (kestus=9 OR registri_nr LIKE '7%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)) OR (kestus=6 OR registri_nr LIKE '1%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)) AND (kestus=10 OR registri_nr LIKE '3%' OR parkimine_id IN (SELECT parkimine_id FROM Teisaldamine))`
(A OR B) AND (A OR C) => A OR (B AND C)
`(kestus=6 OR registri_nr='111KLO') AND (kestus=6 OR kestus=9)`
`(registri_nr LIKE '1%' OR registri_nr LIKE '2%') AND (registri_nr LIKE '1%' OR registri_nr LIKE '3%')`
`(parkimine_id IN (SELECT parkimine_id FROM Teisaldamine)OR kestus=6) AND (parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) OR registri_nr= registri_nr='111KLO')`

Lisa 4 – Lihtsustamise tulemused päringu tüüpide kaupa

Tabelite legend on järgmine: **X** – süsteem lihtsustas päringut; **O** – süsteem ei lihtsustanud päringut; **O*** – PostgreSQL lihtsustas päringut kui *constrain exclusioni* väärtus oli määratud *on;* **V** – süsteem lihtsustas päringut oodatust teisiti; '**-** ' – antud tingimusest ei moodustatud liittingimust, sest eelnevalt ei osanud andmebaasisüsteem tingimimust lihtsustada.

Tabel 18. Tulemused päringu tüübi P1 korral

Päringu tüüp P1			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A AND A	A	**X**	**X**
A AND (false)	FALSE	O*	O
A AND (true)	A	O	**X**
A AND NOT (A)	FALSE	O	O
A AND (A OR B)	A	O	**X**
A OR A	A	**X**	**X**
A OR (false)	A	O	**X**
A OR (true)	TRUE	O	**X**
A OR NOT (A)	TRUE	O	O
A OR (A AND B)	A	**X**	**X**
a>2 AND a>3	a>3	O	**X**
a>2 OR a>3	a>2	O	**X**
NOT (a>6)	a<=6	**X**	**X**
NOT (NOT(a))	a	**X**	**X**
a>6 AND 6<A	a>6	O	**X**
a+6-6>6	a>6	O	O
NOT(A OR B)	NOT(A) AND NOT(B)	**X**	**X**
NOT(A AND B)	NOT(A) OR NOT(B)	**X**	**X**

Päringu tüüp P1			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A OR (NOT(A) AND B)	A OR B	O	O
A AND NOT(B) OR B	A OR B	O	O
A AND B OR A AND NOT (B)	A	V	V
(A OR B) AND (A OR NOT(B))	A	O	O
A AND B OR A AND C	A AND (A OR B)	**X**	**X**
(A OR B) AND (A OR C)	A OR (A AND B)	O	O

Tabel 19. Tulemused päringu tüübi P2 korral

Päringu tüüp P2			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A AND A	A	O	X
A AND (false)	FALSE	O*	O
A AND (true)	A	O	X
A AND NOT (A)	FALSE	O*	O
A AND (A OR B)	A	O	X
A OR A	A	X	X
A OR (false)	A	O	X
A OR (true)	TRUE	O	X
A OR NOT (A)	TRUE	O	O
A OR (A AND B)	A	X	X
a LIKE '12%' AND a LIKE '1%'	a LIKE '12%'	O	O
a LIKE '12%' OR a LIKE '1%'	a LIKE '1%'	O	O
NOT (NOT(a))	a	X	X
NOT(A OR B)	NOT(A) AND NOT(B)	X	X
NOT(A AND B)	NOT(A) OR NOT(B)	X	X
A OR (NOT(A) AND B)	A OR B	O	O
A AND NOT(B) OR B	A OR B	O	O

Päringu tüüp P2			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A AND B OR A AND NOT (B)	A	V	V
(A OR B) AND (A OR NOT(B)	A	O	O
A AND B OR A AND C	A AND (B OR C)	X	X
(A OR B) AND (A OR C)	A OR (B AND C)	O	O

Tabel 20. Tulemused päringu tüübi P3 korral

Päringu tüüp P3			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A AND A	A	O	**X**
A AND (false)	FALSE	O*	O
A AND (true)	A	O	**X**
A AND NOT (A)	FALSE	O	O
A AND (A OR B)	A	O	O
A OR A	A	**X**	**X**
A OR (false)	TRUE	O	**X**
A OR (true)	TRUE	O	**X**
A OR NOT (A)	A	O	**X**
A OR (A AND B)	A	**X**	O
NOT (NOT(a))	a	O	O
NOT(A OR B)	NOT(A) AND NOT(B)	**X**	**X**
NOT(A AND B)	NOT(A) OR NOT(B)	**X**	**X**
A OR (NOT(A) AND B)	A OR B	O	O
A AND NOT(B) OR B	A OR B	O	O
A AND B OR A AND NOT(B)	A	O	O
(A OR B) AND (A OR NOT(B))	A	O	O
A AND B OR A AND C	A AND (B OR C)	**X**	O
(A OR B) AND (A OR C)	A OR (B AND C)	O	O

Tabel 21. Tulemused päringu tüübi P4 korral

Päringu tüüp P4			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A AND A	A	O	**X**
A AND (true)	A	O	**X**
A OR A	A	O	**X**
A OR (false)	A	O	**X**
A OR (true)	TRUE	O	X
A OR (A AND B)	A	O	O
NOT(a>6)	a<=6	**X**	**X**
NOT(A OR B)	NOT(A) AND NOT(B)	**X**	**X**
NOT(A AND B)	NOT(A) OR NOT(B)	**X**	**X**
A AND B OR A AND C	A AND (B OR C)	**X**	-
a>2 AND a>3	a>3	-	O
a>2 OR a>3	a>2	-	**X**
a>8 AND 8<a	a>8	-	**X**

Tabel 22. Tulemused päringu tüübi P5 korral

Päringu tüüp P5			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A AND A	A	**X**	**X**
A AND (false)	FALSE	O*	O
A AND (true)	A	O	**X**
A AND NOT A	FALSE	O*	O
A AND (A OR B)	A	O	**X**
A OR A	A	**X**	**X**
A OR (false)	A	O	**X**
A OR (true)	TRUE	O	**X**
A OR NOT (A)	TRUE	O	O
A OR (A AND B)	A	**X**	**X**
a>2 AND a>3	a>3	O	**X**
a>2 OR a>3	a>2	O	**X**

Päringu tüüp P5			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
NOT (a>6)	a<=6	**X**	**X**
NOT (NOT(a))	a	**X**	**X**
a>6 AND 6<a	a>6	O	**X**
a +6-6>6	a>6	O	O
NOT(A OR B)	NOT(A) AND NOT(B)	X	X
NOT(A AND B)	NOT(A) OR NOT(B)	X	X
A OR (NOT(A) AND B)	A OR B	O	O
A AND NOT(B) OR B	A OR B	O	O
A AND B OR A AND NOT (B)	A	O	V
(A OR B) AND (A OR NOT(B))	A	O	O
A AND B OR A AND C	A AND (A OR B)	X	X
(A OR B) AND (A OR C)	A OR (A AND B)	O	O

Tabel 23. Tulemused päringu tüübi P6 korral

Päringu tüüp P6			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A AND A	A	O	**X**
A AND (false)	FALSE	O*	O
A AND (true)	A	O	**X**
A AND NOT A	FALSE	O*	O
A AND (A OR B)	A	O	**X**
A OR A	A	**X**	**X**
A OR (false)	A	O	**X**
A OR (true)	TRUE	O	**X**
A OR NOT (A)	TRUE	O	O
A OR (A AND B)	A	**X**	**X**
a LIKE '12%' AND a LIKE '1%'	a LIKE '12%'	O	O
a LIKE '12%' OR a LIKE '1%'	a LIKE '1%'	O	O
NOT (NOT(a))	a	**X**	**X**

Päringu tüüp P6			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
NOT(A OR B)	NOT(A) AND NOT(B)	**X**	**X**
NOT(A AND B)	NOT(A) OR NOT(B)	**X**	**X**
A OR (NOT(A) AND B)	A OR B	O	O
A AND NOT(B) OR B	A OR B	O	O
A AND B OR A AND NOT (B)	A	V	V
(A OR B) AND (A OR NOT(B)	A	O	O
A AND B OR A AND C	A AND (B OR C)	X	X
(A OR B) AND (A OR C)	A OR (B AND C)	O	O

Tabel 24. Tulemused päringu tüübi P7 korral

Päringu tüüp P7			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A AND A	A	O	**X**
A AND (false)	FALSE	O*	O
A AND (true)	A	O	**X**
A AND NOT (A)	FALSE	O	O
A AND (A OR B)	A	O	O
A OR A	A	**X**	**X**
A OR (false)	A	O	**X**
A OR (true)	TRUE	O	**X**
A OR NOT (A)	A	O*	**X**
A OR (A AND B)	A	**X**	O
NOT (NOT(a))	a	O	O
NOT(A OR B)	NOT(A) AND NOT(B)	**X**	**X**
NOT(A AND B)	NOT(A) OR NOT(B)	**X**	**X**
A OR (NOT(A) AND B)	A OR B	O	O
A AND NOT(B) OR B	A OR B	O	O
A AND B OR A AND NOT(B)	A	V	O

Päringu tüüp P7			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
(A OR B) AND (A OR NOT(B)	A	V	O
A AND B OR A AND C	A AND (B OR C)	**X**	O
(A OR B) AND (A OR C)	A OR (B AND C)	O	O

Tabel 25. Tulemused päringu tüübi P8 korral

Päringu tüüp P8			
Lihtsustamata tingimuse tunnus	**Oodatava tulemuse tunnus**	**PostgreSQL**	**Oracle**
A AND A	A	-	O
A AND (true)	A	-	**X**
A OR A	A	O	**X**
A OR (false)	A	-	**X**
A OR (true)	TRUE	-	**X**
A OR (A AND B)	A	O	O
NOT(a>6)	a<=6	**X**	**X**
NOT(A OR B)	NOT(A) AND NOT(B)	**X**	**X**
NOT(A AND B)	NOT(A) OR NOT(B)	**X**	**X**
A AND B OR A AND C	A AND (B OR C)	**X**	**X**
a>2 AND a>3	a>3	-	O
a>2 OR a>3	a>2	-	**X**
a>6 AND 6<a	a>8	-	**X**

Lisa 5 – Ridade hinnangu arvuline analüüs

Tabel 26. Ridade hinnangu analüüs

Päringu tunnus	Lihtsustamata tingimus	Hinnang ridade arvule Postgres	Hinnang ridade arvule Oracle	Oodatav lihtustus	Hinnang ridade arvule Postgres	Hinnang ridade arvule Oracles	Tegelik ridade arv
P1	A AND (false)	82	1	False	1	0	0
P1	A AND NOT(A)	16132	15016	False	1	1	0
P1	A OR NOT(A)	985354	984984	True	1000000	1000000	1000000
P1	a+6-6>6	333333	50000	a>6	911433	909439	909439
P1	A OR (NOT(A) AND (B))	28583	29682	A OR B	28790	30133	3133
P1	A AND NOT(B) OR B	28583	29682	A OR B	28790	30133	30133
P1	(A OR B) AND (A OR NOT(B))	29739	29562	A	14867	15194	15194
P1	A AND B OR A AND NOT (B)	14639	14969	A	14867	15194	15194
P2	a LIKE '12%' AND a LIKE '1%	1127	11738	a LIKE '12%'	10101	11738	10890
P2	a LIKE '12%' OR a LIKE '1%	120552	116821	a LIKE '1%	111578	110 481	110481
P3	A AND (A OR B)	5074	10000	A	10000	10000	10000
P3	NOT(NOT(a))	5000	10000	a	10000	10000	10000
P7	(A OR B) AND (A OR C)	175224	10001	A OR (B AND C)	175224	10001	10001

Lisa 6 – PostgreSQLi ridade hinnangu protsentuaalne analüüs

Tabel 27. PostgreSQLi ridade hinnangu protsentuaalne analüüs

Päringu tunnus	Lihtsustamata tingimus	Erinevus protsentides	Oodatav lihtustus	Erinevus protsentides	Protsendide erinevus
P1	A AND (false)	**100%**	False	100%	**0**
P1	A AND NOT(A)	**100%**	False	100%	**0**
P1	A OR NOT(A)	1%	True	0%	**1**
P1	a+6-6>6	63%	a>6	0%	**63**
P1	A OR (NOT(A) AND (B))	5%	A OR B	5%	**0**
P1	A AND NOT(B) OR B	5%	A OR B	5%	**0**
P1	(A OR B) AND (A OR NOT(B))	49%	A	2%	**47**
P1	A AND B OR A AND NOT (B)	4%	A	2%	**2**
P2	a LIKE '12%' AND a LIKE '1%	90%	a LIKE '12%'	7%	**83%**
P2	a LIKE '12%' OR a LIKE '1%	8%	a LIKE '1%	1%	**7**
P3	A AND (A OR B)	50%	A	0%	**50**
P3	NOT(NOT(a))	50%	a	0%	**0**
P7	(A OR B) AND (A OR C)	94%	A OR (B AND C)	94%	**0**

Lisa 7 – Oracle ridade hinnangu protsentuaalne analüüs

Tabel 28. Oracle ridade hinnangu protsentuaalne analüüs

Päringu tunnus	Lihtsustamata tingimus	Erinevus protsentides	Oodatav lihtustus	Erinevus protsentides	Protsendide erinevus
P1	A AND (false)	**100%**	False	100%	**0**
P1	A AND NOT(A)	**100%**	False	100%	**0**
P1	A OR NOT(A)	2%	True	0%	**2**
P1	a+6-6>6	95%	a>6	0%	**0**
P1	A OR (NOT(A) AND (B))	2%	A OR B	0%	**0**
P1	A AND NOT(B) OR B	2%	A OR B	0%	**2**
P1	(A OR B) AND (A OR NOT(B))	49%	A	0%	**0**
P1	A AND B OR A AND NOT (B)	1%	A	0%	**1**
P2	a LIKE '12%' AND a LIKE '1%	7%	a LIKE '12%'	7%	**0**
P2	a LIKE '12%' OR a LIKE '1%	5%	a LIKE '1%	4%	**1**
P3	A AND (A OR B)	0%	A	0%	**0**
P3	NOT(NOT(a))	0%	a	0%	**0**
P7	(A OR B) AND (A OR C)	0%	A OR (B AND C)	0%	**0**

Lisa 8 – Töökiiruse testpäringud

Tabel 29. Töökiiruste testpäringud.

Lihtsustamata päringu kood:	Lihtsustamata päring:	Oodatava tulemuse kood	Oodatav tulemuse päring
L1	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE K.perenimi LIKE 'K%' AND P.parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) **AND P.kestus<6** **AND P.kestus>6;**	**O1**	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE FALSE;

Lihtsustamata päringu kood:	Lihtsustamata päring:	Oodatava tulemuse kood	Oodatav tulemuse päring
L2	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE K.perenimi LIKE 'K%' AND **(P.parkimine_id IN** **(SELECT parkimine_id** **FROM Teisaldamine)** **OR P.parkimine_id NOT IN** **(SELECT parkimine_id** **FROM Teisaldamine));**	**O2**	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE K.perenimi LIKE 'K%';
L3	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE K.perenimi LIKE 'K%' AND P.parkimine_id IN (SELECT parkimine_id FROM Teisaldamine) AND **P.kestus+6-6>6 ;**	**O3**	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE K.perenimi LIKE 'K%' AND P.kestus>6 AND P.parkimine_id IN (SELECT parkimine_id FROM Teisaldamine);

Lihtsustamata päringu kood:	Lihtsustamata päring:	Oodatava tulemuse kood	Oodatav tulemuse päring
L4	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE K.perenimi LIKE 'K%' AND **((P.parkimine_id IN** **(SELECT parkimine_id** **FROM Teisaldamine))** **OR P.kestus=6)** **AND ((P.parkimine_id IN** **(SELECT parkimine_id** **FROM Teisaldamine))** **OR P.kestus<>6);**	**O4**	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE K.perenimi LIKE 'K%' AND P.parkimine_id IN (SELECT parkimine_id FROM Teisaldamine);
L5	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE K.perenimi LIKE 'K%' AND **(P.registri_nr LIKE '12%'** **OR P.registri_nr LIKE '1%');**	**O5**	SELECT kliendi_kood, K.perenimi, P.algus FROM Klient AS K INNER JOIN Parkimine AS P USING(kliendi_kood) WHERE K.perenimi LIKE 'K%' AND P.registri_nr LIKE '1%';

Lisa 9 – Töökiiruse tulemused väiksemate ridade arvu korral

Tabel 30. Töökiiruse uurimise tulemused suurimast ridade arvust kaks korda väiksema ridade arvu korral

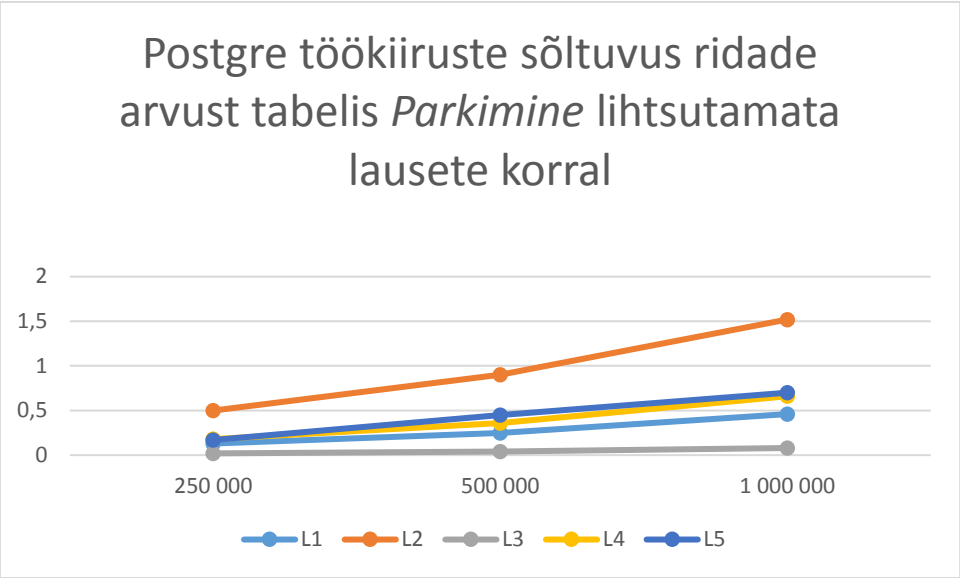
Lihtsustamata päringu kood	Lihtsustamata päringu täitmise aeg (s)	Lihtsustatud päringu kood	Lihtsustatud päringu täitmise aeg (s)	Päringute täitmise aja vahe (s)
Oracle				
L1	0,02	**O1**	0,01	**0,01**
L2	0,92	**O2**	0,89	**0,03**
L3	0,33	**O3**	0,22	**0,11**
L4	0,36	**O4**	0,22	**0,14**
L5	0,32	**O5**	0,25	**0,07**
PostgreSQL				
L1	0,25	**O1**	0,0005	**0,2495**
L2	0,90	**O2**	0,65	**0,25**
L3	0,04	**O3**	0,04	**0,00**
L4	0,36	**O4**	0,04	**0,32**
L5	0,45	**O5**	0,29	**0,16**

Tabel 31. Töökiiruse uurimise tulemused suurimast ridade arvust neli korda väiksema ridade arvu korral

Lihtsustamata päringu kood	Lihtsustamata päringu täitmise aeg (s)	Lihtsustatud päringu kood	Lihtsustatud päringu täitmise aeg (s)	Päringute täitmise aja vahe (s)
Oracle				
L1	0,01	**O1**	0,01	**0,00**
L2	0,46	**O2**	0,43	**0,03**
L3	0,19	**O3**	0,11	**0,08**
L4	0,22	**O4**	0,10	**0,12**
L5	0,16	**O5**	0,13	**0,03**
PostgreSQL				
L1	0,13	**O1**	0,0005	**0,1295**
L2	0,50	**O2**	0,33	**0,17**

Lihtsustamata päringu kood	Lihtsustamata päringu täitmise aeg (s)	Lihtsustatud päringu kood	Lihtsustatud päringu täitmise aeg (s)	Päringute täitmise aja vahe (s)
L3	0,02	**O3**	0,02	**0,00**
L4	0,18	**O4**	0,03	**0,15**
L5	0,17	**O5**	0,12	**0,05**

Lisa 10 – PostgreSQL töökiiruste seosed



Joonis 23. Postgre töökiiruse seosed.