TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Piret Näppi 182859

REA- JA VEERUPÕHISE ANDMEBAASISÜSTEEMI VÕRDLUS JAEKAUBANDUSE ANALÜÜTIKA NÄITEL

Magistritöö

Juhendaja: Innar Liiv
PhD

Tallinn 2020

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: Piret Näppi

11.05.2020

Annotatsioon

Käesoleva töö eesmärgiks on võrrelda rea- ja veerupõhise andmete salvestamise mõju jaekaubanduse valdkonnas tüüpiliste andmepäringute töökiirustele. Töö alguses leitakse 12 jaekaubanduse peamist infovajadust, millele vastamiseks defineeritakse sobivad andmepäringud. Järgmises etapis luuakse identsete andmetega andmebaasid kahes erinevas SQL-andmebaasisüsteemis. Andmebaasidel on sarnane loogiline disain, kuid erinevus sisemisel tasemel, kus toimub andmete füüsiline salvestamine andmekandjale. Veerupõhise salvestamise realiseerimiseks osutus analüütiliste hierarhiate meetodil valituks andmebaasisüsteem MariaDB, milles asub Columnstore andmebaasimootor. Seda võrreldakse reapõhise PostgreSQL andmebaasisüsteemiga. Andmepäringuid käivitatakse andmehaldusvahendis DBeaver ja käsureal, kus väljastatakse lisaks töökiirustele ka täitmisplaanid, mida analüüsida. Kirjeldatakse ka andmebaasidesse andmete lisamist ning nendega töötamisel tekkinud kitsaskohti. Kõikide andmebaasipäringute töökiirused 40 GB andmemahu korral olid paremad veerupõhisel salvestamisel. Ärilise konteksti ilmestamiseks tuuakse eksperimendis vaadeldud andmepäringutega rakendamise näide jaekaubanduse analüütikas kaupluste profileerimisel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 77 leheküljel, 7 peatükki, 13 joonist, 10 tabelit.

Abstract

A Comparison of Row-Oriented and Column-Oriented Database Management Systems on the example of Retail Analytics

The aim of this research is to compare row-oriented data storage database management system with column-oriented database management system on the basis of typical data queries performance in the retail sector.

At the beginning of the thesis, 12 main retail information needs are identified, for which suitable data queries are defined. In the next step, databases with identical data are created in two different SQL database management systems. Databases have similar logical design, but differ at the internal level, where data is physically stored on a storage medium. Column-oriented storage is implemented in MariaDB database management system that uses Columnstore storage engine, which achieved the best rating in AHP (*Analytic Hierarchical Process*). It is compared to a row-oriented storage model in the PostgreSQL database management system. The comparison of two database management systems is based on execution time of the queries. Also, data import and bottlenecks are described. Queries are executed in the DBeaver data management tool and on the command line, where execution plans are issued. The execution time of all queries with 40 GB data capacity were better in columnar storage. At the end, an example of application in retail analytics in store profiling is provided to demonstrate advantages of selected queries in commercial context.

The thesis is in Estonian and contains 77 pages of text, 7 chapters, 13 figures, 10 tables.

Lühendite ja mõistete sõnastik

ACID	*Atomicity*, *Consistency*, *Isolation*, *Durability*, andmebaasitehingute usaldusväärseks töötluseks vajalikud omadused (atomaarsus, konsistentsus, isoleeritus, püsivus)
AHP	*Analytic Hierarchical Process*, analüütiliste hierarhiate meetod
BLOB	*Binary Large Object*, andmetüüp binaarsetele andmetele
BPS	*Batch Primitive Step,* andmete skaneerimise operatsioon Columnstore andmebaasimootori täitmisplaanis
CPU	*Central Processing Unit,* protsessor
CSV	*Comma Separated Value*, tekstifaili tüüp
DML	*Data Manipulation Language*, andmete manipuleerimiskeel
DRAM	*Dynamic Random Access Memory*, dünaamiline muutmälu
DSM	*Decomposition Storage Model*, lehekülje formaat veerupõhisel andmete salvestamisel
DSS	*Dictionary Structure Step,* operatsioon Columnstore andmebaasimootori täitmisplaanis stringi väärtuse sõnastikust otsimiseks
ERP	*Enterprise Resource Planning*, ettevõtte ressursside plaanimine
FPGA	*Field-Programmable Gate Array*, programmeeritav ventiilimaatriks
NewSQL-andmebaasisüsteem	Andmebaasisüsteemide kontseptsioon, kus on kombineeritud relatsiooniliste ja NoSQL-andmebaasisüsteemide eelised
NoSQL-andmebaasisüsteem	*„Not only SQL“*, andmebaasisüsteemide kontseptsioon suurandmete töötluse hõlbustuseks ja pilvetöötlusega sobituseks
NSM	*N-Ary Storage or Slotted Pages*, lehekülje formaat reapõhisel andmete salvestamisel
OLAP	*Online Analytical Processing*, sidusanalüüs
OLTP	*Online Transaction Processing*, sidus tehingutöötlus
PCA	*Principal Component Analysis,* peakomponentide meetod
POS	*Point Of Sale*, müügipunkt - kassa, kassaterminal, kassaarvuti
Puukaart	*Treemap*, hierarhiliste andmete esitusviis - puuskeemi harusid esindavate pesastatud ristkülikutega
RAM	*Random Access Memory*, muutmälu

RLE	*Run-Length Encoding,* andmete kodeerimise meetod
Soojakaart	*Heatmap*, maatriksdiagramm, milles andmete väärtusi esitatakse värvustena või halliastmiku toonidena
SQL	*Structured Query Language*, struktureeritud päringukeel
SSD	*Solid-State Drive*, pooljuhtketas - kõvaketta funktsioonidega mahukas välkmälu
TAS	*Tuple Aggregation Step,* operatsioon Columnstore andmebaasimootori täitmisplaanis vahepealsete tulemuste vastu võtmiseks
TNS	*Tuple Annexation Step,* operatsioon Columnstore andmebaasimootori täitmisplaanis päringu tulemuse viimistlemiseks
UML	*Unified Modeling Language*, unifitseeritud modelleerimiskeel

Lühendite ja mõistete sõnastiku koostamisel on kasutatud Andmekaitse ja infoturbe leksikoni (AKIT) [1].

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva magistritöö teemaks on rea- ja veerupõhise salvestusmudeliga SQL-andmebaasisüsteemide võrdlemine jaekaubanduse valdkonna näitel koos tehnilise lahenduse ja eksperimentidega.

1.1 Taust ja probleem

Igapäevaselt tekitatakse väga palju ostuandmeid. Neid andmeid kasutatakse inimeste ostuharjumuste analüüsimiseks, mille põhjal teha korrektuure kaupluse riiulitel ning personaalseid pakkumisi. Jaekaubanduses tegeletakse pidevalt andmete analüüsimisega ning selle pealt otsitakse vastuseid erinevatele küsimustele, mille eesmärgiks on kasumi ja kliendibaasi suurendamine, hetkeolukorra hindamine ning varasemaga võrdlemine. Mustrite ja trendide märkamiseks on vajalik andmeanalüüs pikema perioodi andmetega, mis suurendab päringutes kasutatavat andmemahtu kordades.

Enim kasutatavad andmebaasid on reapõhise salvestusmudeliga, kus sisemiselt salvestatakse ühele leheküljele andmeid ühest tabelist ridade kaupa. Suurte andmehulkadega jääb tavapärane reapõhine andmebaasisüsteem hätta, sest selle väljatöötamise hetkel ei olnud andmemahud nii suured ning probleemi iga rea lugemisel ei nähtud. Üha suurenevate andmemahtudega toimetulekuks ning tõhusaks andmeanalüüsiks on vajalik tehnoloogiliste uuendustega kaasas käimine. Kitsaskohtadest üle saamiseks tuleks andmebaasisüsteemide sisemistes tööpõhimõtetes teha muudatusi.

Probleemi lahendamiseks on loodud NoSQL-andmebaasisüsteemid („*Not only SQL*“), mis on mõeldud tulema toime suurte andmemahtudega ja kiirendama OLAP (*Online Analytical Processing*) päringuid. Nende populaarsus on aastatega järk-järgult tõusnud, kuid nad ei edasta traditsioonilisi andmebaasisüsteeme, sest enamus neist hülgab tuntud relatsioonilise andmemudeli ning SQL-andmebaasikeele (*Structured Query Language*).

Alternatiivina võiks kaaluda veerupõhisel salvestamisel töötavaid andmebaasisüsteeme, kus kasutatakse endiselt SQL-andmebaasikeelt ning muutus viiakse läbi andmebaasi sisemisel tasandil. Lihtne lahendus, millest tekib küsimus, kas sellise tehnilise lahenduse eelised on piisavad ning annavad märkimisväärseid tulemuste erinevusi OLAP päringute käivitamisel.

1.2 Ülesande püstitus

Käesoleva magistritöö eesmärgiks on võrrelda andmepäringute töökiirust kahes SQL-andmebaasisüsteemis, mis kasutavad erinevaid sisemisi salvestusmudeleid. Reapõhine salvestamine teostatakse kasutades PostgreSQL andmebaasisüsteemi ning veerupõhine kasutades MariaDB Columnstore andmebaasisüsteemi. Võrreldavate andmepäringute aluseks on jaekaubanduse valdkonna peamised infovajadused, mis on kombineeritud kirjandusest ning suhtlusest valdkonnas tegutsevate inimestega. Andmepäringuid rakendatakse kaupluste profileerimisel Jupyter Notebook tarkvaras. Eesmärgi saavutamiseks on vaja uurida teoreetilisi aspekte veerupõhisest salvestusmeetodist ning jaekaubanduse analüütikast, tuleb luua kaks andmebaasi, kuhu importida ostuandmed ning käivitada ja mõõta andmepäringute töökiirust.

1.3 Metoodika

Andmebaasisüsteemides kasutatavate salvestusmeetodite ja nende teoreetilise taustaga tutvutakse läbi avaldatud artiklite ja raamatute mõistmaks ajaloolisi aspekte ning hetke trende. Eksperimendi jaoks luuakse identse andmestikuga andmebaasid PostgreSQL ja MariaDB Columnstore andmebaasisüsteemides. Ühe aasta ostuandmete põhjal koostatud andmestik on päritud valdkonnas tegutsevalt ettevõttelt, kes eelistab jääda anonüümseks.

Jaekaubandus valdkonna peamised 12 küsimust tulevad samuti kirjandusest, kuid on kombineeritud küsimustega, mis on tekkinud kokkupuutest valdkonnas tegutsevate inimestega, kellel tuleb igapäevatöös tegeleda kauba tellimuste esitamisega analüüsides varasemaid ostuandmeid ja poe sortimenti. Infovajadustele vastuste leidmiseks kirjutatakse vastavad päringud. Leitud päringuid käivitatakse kahes andmebaasis ning võrreldakse nende jõudlust. Sobivate päringutega tuuakse rakenduse näide kaupluste profileerimisel, mille aluseks on kategooriad.

1.4 Ülevaade tööst

Magistritöö on jagatud viieks suuremaks osaks. Esimeses osas uuritakse jaekaubanduse valdkonda laiemalt. Tuuakse ülevaade hetke trendidest ning varasematest uurimustest.

Töö teises osas vaadeldakse veerupõhise salvestamise kujunemise ajalugu ning SQL-andmebaasisüsteemides kasutatavaid salvestusmudeleid. Kirjeldatakse veerupõhise andmebaasisüsteemi valikut ning tuuakse välja mõlema andmebaasisüsteemi eelised ja puudused. Peatüki lõpus tehakse kokkuvõte varasematest uurimustest.

Kolmandas osas kirjeldatakse eksperimendi ülesehitust. Detailsemalt kirjutatakse lahti andmebaaside loomise protsess koos andmete importimisega, selgitatakse uuritavaid päringuid ning eksperimendi sooritamist. Eraldi peatükina tuuakse välja tulemused.

Eksperimendile järgneb tulemuste analüüsi peatükk, kus võrreldakse andmebaasisüsteemide täitmisplaane, kirjeldatakse andmebaasidega töötamist ning andmete lisamist.

Magistritöö viiendas osas tuuakse välja üks näide päringute rakendamisest. Päringuid rakendatakse kaupluste grupeerimisel kategooriate käibe protsentide alusel k-keskmiste meetodit kasutades. Peale seda tehakse tööst kokkuvõte, kus tuuakse välja peamised järeldused ning pakutakse välja mõned edasiarendused.

2 Jaekaubanduse analüütika

Järjest enam on ettevõtted hakanud tegelema detailsema andmeanalüüsiga, sest on arusaam, et parem kliendi tundmine viib õigete toodete pakkumiseni õigel ajal ja õiges kohas. Jaekaubanduses on analüüsi aluseks andmed, mis on seotud klientidega, toodetega, asukohaga, ajaga ning kanalitega. Peamisteks uurimussuundadeks on klientide profileerimine, ostukorvi analüüs ning erinevad tehnoloogilised lahendused nagu toidurobotid ja iseteeninduskassad. Käesolevas peatükis antakse ülevaade hetke trendidest jaekaubanduses ning tuuakse välja varasemad uurimused.

2.1 Jaekaubanduse ülevaade

Erinevad tehnoloogiad nagu lojaalsuskaardid, POS (*Point Of Sale*) terminalid, veebisaitide küpsised, peaksid andma ettevõtetele arusaama kliendi käitumisest, kuid olenemata suurtest andmehulkadest, ei osata kaevandada vajalikku informatsiooni mustrite märkamiseks. Tooted muutuvad sageli ja on levinud arvamus, et ajaloolised andmed ei anna informatsiooni hetke trendide kohta, kuid samas sisaldavad nad endas stabiilseid parameetreid nagu enim ostetud tooted kindlas kaupluses, suuruste vahemikud ja kategooriate kasvutrendid, mille pealt ennustada harva muutuvat tarbijakäitumist. Mustrite märkamiseks tuleks sarnaseid kauplusi ja ladusid grupeerida, sest ühe kaupluse pealt pole võimalik teha üldistusi. Tarneahela toimimiseks tuleks valida vajalikud mõõdikud. Laos olevad tooted ei näita tegelikku müüki, sest pole potentsiaalse kliendi vaateväljas ning sel asjaolul peaks tellimustes arvestama laoseisu asemel müüdud toodete arvu. Lühem tarneaeg kasutades õhutransporti võimaldab paremini toime tulla nõudluse ebakindlusega, sest prognoosiviga on väiksem [2].

Iseteeninduskassad, puldikassad, targad riiulite tehnoloogiad, on vaid mõned näited jaekaubandussektori tehnoloogilistest arengutest, mida klientide meelitamiseks ja hoidmiseks kasutusele võetakse. Kliendi rahulolu tõstmiseks kasutatakse ennustavat analüütikat, mille sisendiks on infrapuna sensorid kaupluste uste ja kassade juures eesmärgiga mõõta järjekordade pikkust, millega on lühendatud ooteaega 4-lt minutilt 30-le sekundile. Jätkuvalt arenevaks tehnoloogiaks peetakse erinevaid kassasid, mille

näiteks võib tuua Amazon Go mobiilirakenduse, mis tuvastab riiulitelt võetud tooted kasutades masinõppe tehnoloogiaid [3]. Toodete riiulitele paigutamisel on samuti väga suur osakaal. Läbi viidud uurimuses tuli välja, et vertikaalselt asetatud toodete võtmine riiulilt oli 90% suurem kui horisontaalselt [4]. Kauba otsa lõppemist on jaekaubanduses analüüsitud palju, kuid see on endiselt asjakohane, sest jaemüüjad kaotavad aastas keskmiselt 3,9% teenitavast tulust üle maailma [5].

2.2 Varasemad uurimused jaekaubanduses

Jaekaubanduse valdkonnas magistritööd tehes on mitmeid erinevaid uurimissuundi ja artiklite valik lai, kuid töömahtu ja kasutatavat ressurssi arvestades oli mõistlik valikut kitsendada. Autor valis artiklite leidmiseks 2 kindlat lähenemist – klientide profileerimine, mis annaks suuniseid kaupluste profileerimiseks ja ostukorvi analüüs, sest saadud andmestiku alusel oli võimalik neid valdkondi detailsemalt käsitleda.

2018.aastal valminud magistritöös [6] on analüüsitud avalikku e-poe andmestikku graafipõhises andmebaasisüsteemis Neoj4. Töö eesmärgiks on modelleerida kohandatud süsteem, mis soovitaks mingi toote ostmist. Valitud on graafipõhine andmete hoiustamine, sest see on uuenduslik ning annab võimaluse analüüsida suuri andmehulki. Kirjeldatud on graafiteooria põhimõtteid ning välja on toodud põhilised erinevused SQL- ja NoSQL-andmebaasisüsteemide vahel. Samuti on kirjeldatud andmestiku üldisi omadusi ning omavahelisi seoseid. Kitsaskohad, mis välja tulid – juhtumipõhised alampäringud üle suurte võrkude vajavad palju RAM mälu (*Random Access Memory*) ning Neo4j omadus organiseerida sõlmedele määratud atribuudid andmetüübina BLOB *(Binary Large Object)*, mis on salvestatud ühte faili. Graafipõhine andmebaas, Neo4j, toimis hästi suurel ja ühendatud andmekogumil, mis toetas sõlmede tõhusat otsimist. Autor soovitab edasiarendusteks mitmeid töid, näiteks luua mitme erineva ettevõtte andmete pealt eraldi graafi osad ning võrrelda neid omavahel. Veel leitakse, et magistritöö tulemusena arendatud süsteemist võiks luua pilveteenuse.

Artiklis [7] on lahti mõtestatud jaekaubanduse suurandmeid, mis on jaotatud viide erinevasse gruppi: kliendiga, toodetega, asukohaga, ajaga ning kanaliga seotud andmed. Andmehulkade tõhusamaks analüüsimiseks on vajalik andmete pakkimine ning statistilise meetodite loomine. Jaemüügi küsimustele vastuste otsimise tähtsus on jäänud madalaks isegi kui suurandmete osatähtsus ning nende pealt ostuharjumuste

ennustamise osatähtsus on suurenenud. Üha olulisemaks saab ennustav analüütika, mille on teinud võimalikuks uued andmeallikad ja korrelatsioonitehnikad teooria, valdkonna teadmiste ja statistiliste meetodite kaasamisega. Eksperimendis on kasutatud prognoosivat analüüsi, mille valimi moodustavad sarnaste tunnustega kauplused. Loodud mudel täiustas kaupluste marginaale läbi hinnastamise muutuste nii statistiliselt kui ka halduslikult. Suuremat tähelepanu peaks pöörama andmete kvaliteedile ja mitte mahule. Arutletud on Bayes'i analüüsimeetodite (andmete laenamine, värskendamine, suurendamine ja hierarhiline modelleerimine) kasutamist jaemüügi kontekstis. Tähelepanu pööratakse olulisematele privaatsuse ja eetilistele probleemidele, mida ilmestatakse ettevõtte Target näitel.

2019.aastal avaldatud artiklis [8] tuuakse visuaalseid näiteid toodete profileerimiseks jaekaubanduses ning esitatakse tegevuses ettetulevate probleemide lahendused. Masinõppe meetodid vajavad treenimiseks andmeid, mis on vigadest vabad. Andmed peavad olema puhastatud ja kategoriseeritud, et tõsta nende kvaliteeti, kuid samas nõuab see laialdast andmete profileerimist. Probleemi lahenduseks on artiklis välja pakutud mudel, mis kiirendab oluliselt toodete profileerimist. Autor eristab toodete kirjeldavaid ja protsessipõhiseid atribuute ERP (*Enterprise Resource Planning*) süsteemis, mis annab aimu informatsiooni asukohast. Lisaks tuuakse välja kahe ettevõtte arhitektuur SAP Retail süsteemis, mis ilmestab andmete struktuuri. Kokku puututakse järgmiste tehniliste väljakutsetega:

- suur hulk toodete spetsiifilisi andmeid, näiteks asukoht, mis erineb kaupluste lõikes;
- palju kategooriaid, mis muudab vigade leidmise keerulisemaks, sest atribuutide hulk on erinev;
- erinevad toote tüübid, mille alusel kategooriaid liigendada;
- süsteemides hoitakse vanu andmeid;
- äriprotsesside optimeerimisega ei värskendata aegunud toodete andmeid;
- keeruline andmemudel;
- uue versiooni kasutuselevõtuga muutub andmemudel.

Tulemusena selgitatakse lahti loodud mudeli arhitektuur ning millised on ekspertide poolt välja pakutud parimad lahendused toodete profileerimiseks. Soovitatakse kasutada soojakaarti (ingl *Heatmap*) ja puukaarti (ingl *Treemap*) parema ülevaate saamiseks ning

kindlasti peaks andmetega koos olema muutmiskuupäev, mis võimaldaks elimineerida aegunud tooted, sest töös avastati, et kaupluste sortimentides oli ainult 20% kogu toodete mahust. Mudeli eesmärgiks on vähendada ressursse, mis kuluvad info hankimiseks ja säilitamiseks näitamaks, et andmete profileerimist ja uurimist on võimalik teha palju väiksema hulga, kuid kvaliteetsema informatsiooniga.

Artiklis [9] on käsitletud klientide profileerimist tekkinud emotsioonide alusel. Ettevõtted tahavad saavutada konkurentsieelist väärtuspakkumistega, mille saavutamiseks luuakse töös skaala kliendi väärtuse mõõtmiseks ja modelleerimiseks. Uurimuses on palutud Ameerika, Soome ja Jaapani jaekaubanduse klientidel hinnata moe või elektroonika kaupluste külastamist kuues erinevas kategoorias (majanduslik, funktsionaalne, emotsionaalne, sümboolne, rahulolu, suusõnaline), mida uuriti struktuurvõrrandi modelleerimisega. Analüüsis toodi välja erinevused riigiti ning ka kanali kaudu koos arvatavate põhjustega. Tööst järeldati nii oodatavaid kui ka uusi teadmisi, näiteks oli tulemusena oodatav, et kõrgem kliendi väärtus majandusliku, funktsionaalse, emotsionaalse ja sümboolse väärtuse osas oli seotud kõrgema rahulolu ja suusõnaliste väärtustega. Kolmel riigi erinev kultuur tõi analüüsi erinevused kliendi väärtuste profileerimisel ja nende rollil. Näiteks ameeriklased tajuvad rohkem emotsionaalseid ja sümboolseid väärtuseid võrreldes soomlaste ja jaapanlastega.

2018.aastal avaldatud artiklis [10] on välja pakutud ärianalüütika lähenemine, mis kasutab klasterdamise (ingl *clustering*) tehnikaid kaupluste külastuste segmenteerimiseks. Tarbijate andmeid on varasemalt analüüsitud ka klasterdamise ja assotsiatsioonireeglite kaevandamisega. Ostukorvi andmed grupeeritakse tootekategooriate järgi, mida igal külastusel osteti ja selle pealt tuvastatakse külastuse segmendid. Oluliseks pidepunktiks osutus toote kategooria detailsuse valimine hierarhiast. Kitsaskohtade lahendusena leiti, et dimensionaalsuse vähendamine (ingl *dimensionality reduction*) klasterdamisel ja peakomponentanalüüsis (PCA - *Principal Component Analysis*), mille peakomponendid olid tootekategooriad, on oluline. Autorid on arvamusel, et nende töö võiks olla aluseks muutustele jaekaubanduses. Näiteks toodete ümber korraldamine kaupluses või laos, nii et kaup oleks organiseeritud külastussegmentide järgi. Loodud lähenemisviis annaks lisafunktsionaalsuse turul olevatele ärilistele tööriistadele nagu IBM WATSON, SAP HANA.

Artiklis [11] kirjutatakse suurandmetest, hetkel jaemüüjate poolt kasutatavatest meetoditest kliendibaasi säilitamiseks ning inimeste mõjutamisviisidest, millest kliendid ei pruugi arugi saada. Peamiseks eesmärgiks on teadvustamine, kuidas jaemüüjad kliente kasumi teenimise eesmärgil mõjutavad ning arutletakse, kas teguviis on eetiline. Väljend „*nudge*“ tähendab selles kontekstis, mis tahes valikuvõimaluse aspekti, mis muudab inimeste käitumist prognoositaval viisil igasuguseid valikuid või majanduslikke stiimuleid muutmata. Artiklis käsitletakse, kuidas klientide kaasamiseks kasutatakse suurandmeid ning mis on kasutatavate meetodite mõju. Töös viiakse läbi intervjuud kahte gruppi jaotatud inimestega: 21 indiviidi, kes nägid suurandmete kasutamist jaekaubanduses ja 5 tarbijagruppi, kes ostlesid kauplustes. Töö on organiseeritud kolme gruppi: tooted, kaubandus ja mustrid.

Artiklis [12] on välja pakutud mudel, mille eesmärk on näidata, et toodete naabertooted kaupluse riiulil mõjutavad tarbija ostukorvi. Kategooriajuhid peavad kaupa tellides arvestama ka teistele kategooriatele minevate ressurssidega, näiteks ruum. Artiklis näidatakse, et eksisteerib kategooriate vaheline sõltuvus, mis on ajendatud korrelatsioonita kategooriate tarbimisest. Uurimuses tuleb välja, et fookus kindlalt kategoorialt kaob ning ostu tõenäosus väheneb olenevalt laiema valikuga teine kategooria lähedusest. Hüpotees sai tõestatud silmajälgimis uuringuga. Uurimus aitab jaekaubandus ettevõtetel riiuliruumi ümber korrigeerida. Esitatud mudelit on võimalik kasutada ka väljaspool jaekaubandust, näiteks toitlustuse menüüdes.

Töö [13] esitab ostukorvi analüüsimiseks algoritmi, mis põhineb assotsiatsiooni reeglitel ja ühisel filtreerimisel. Algoritm lahendab assotsiatsioonireeglite probleemi, mis ei loo reeglit ebapopulaarsete tarbekaupade vahel ehk ei arvestata kaupade mitmekesisust, vaid kirjeldatakse kõige populaarsemate kaupade seoseid. Kirjeldatakse kahte tüüpi probleeme, millele tuleb tähelepanu pöörata: transaktsioonide andmed on kõige täpsemad ning ühele ebapopulaarsele kategooriale assotsiatsioonireegli määramine vähendab mudeli täpsust. Selles artiklis on võetud tarbekaupade andmed, kus on soojade kaupade osakaal märkimisväärselt suurem kui külmade. Mudelit on testitud ning hinnatud on selle täpsust ja jõudlust.

3 SQL-andmebaasisüsteemid

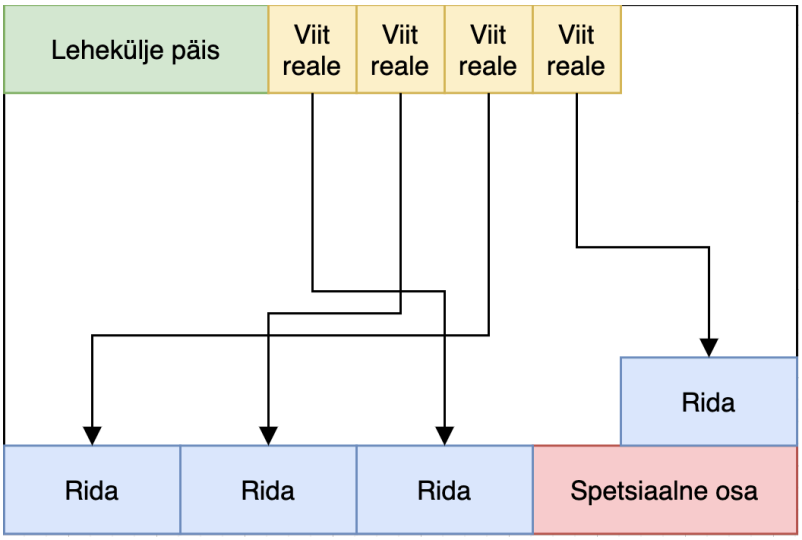
Andmebaasisüsteemid erinevad andmemudeli poolest, millel nad põhinevad. Olenevalt andmemudelist toetavad andmebaasisüsteemid ühte või mitut andmebaasikeelt. Tänapäeval kasutavad enamus andmebaasisüsteemid SQL-andmebaasikeelt, mis on SQL standardi üks osa. SQL standard põhineb Edgar F. Codd poolt 1969.aastal välja pakutud relatsioonilisel andmemudelil, kuid mitmetes punktides esineb erinevusi, mistõttu on korrektsem eristada ideaalset relatsioonilist mudelit ja SQL-mudelit [14]. Antud töös kasutatakse SQL-andmemudelil põhinevaid andmebaasisüsteeme– PostgreSQL ja MariaDB Columnstore, mida autor nimetab SQL-andmebaasisüsteemideks.

3.1 Reapõhine andmete salvestamine PostgreSQL näitel

Reapõhiseks andmete salvestamiseks valiti PostgreSQL mitmel põhjusel. PostgreSQL on oma populaarsuselt hetkel 4. kohal ning tõusutrendis, mis viitab suurele kasutajahulgale ja tugevale kogukonnale [15]. Vabavaralistest andmebaasisüsteemidest on ees vaid MySQL, milles päringute jõudlus jääb PostgreSQL-le alla [16]. Andmebaasisüsteemi peamisteks eelisteks on tugev vastavus SQL standardile ning võimekus töödelda keerukamaid päringuid. Suurte andmemahtudega töötamisel peetakse plussideks ka struktureerimata andmete toetust läbi XML, JSON ja HStore andmetüüpide ning päringute paralleeltöötlust. Lisaks on PostgreSQL-l andmete genereerimise võimalus kasutades funktsioone random ja generate_series. PostgreSQL-l on ka puuduseid, mis annavad võimaluse ka teistele andmebaasisüsteemidele. Võrreldes analüütiliste andmebaasisüsteemidega ei paku PostgreSQL andmete pakkimist ja veerupõhiseid tabeleid, mistõttu võib olla vajalik suuremate tabelite vertikaalne killustamine [17]. Töös kasutatakse PostgreSQL 12.2 versiooni vaikimisi seadistustega.

Reapõhisel andmete salvestamisel sisemisel tasandil salvestatakse tabeli 0 või rohkem rida ühele leheküljele. PostgreSQL, nagu ka mitmed teised tuntumad reapõhist salvestamist kasutavad andmebaasisüsteemid, kasutab lehekülje paigutusel NSM (*N-Ary Storage*) salvestamismudelile sarnast lähenemist.

PostgreSQL salvestusmudelist on toodud pilt joonisel 1.



Joonis 1. PostgreSQL salvestamine leheküljel [18].

Joonis 1 illustreerib PostgreSQL andmebaasisüsteemis realiseeritud lehekülje salvestusmudelit. Iga lehekülg on jagatud viieks osaks: lehekülje päis, väärtuste identifikaatorid, vaba ruum, väärtused ja spetsiaalne sektsioon. Iga lehekülg algab lehekülje päisega, kus igal baidil on oma otstarve näitamaks lehekülje üldist informatsiooni. Sellele järgnevad väärtuste identifikaatorid, mis sisaldavad viita väärtuse algusesse, väärtuse pikkust ja väärtuse interpreteerimise infot. Väärtused ise salvestatakse tagantpoolt ettepoole, mis tekitab vaba ruumi identifikaatorite ja väärtuste vahel. Lehekülje lõpus asub spetsiaalne osa, kuhu salvestatakse vajalik informatsioon juurdepääsuks, mida tavaliselt pole vaja, kuid näiteks B-puu indeksite korral kasutatakse struktuuri taastamiseks [18].

3.1.1 PostgreSQL laiendus cstore-fdw

Kaasajal on mitmetele traditsioonilistele andmebaasisüsteemidele lisatud juurde laiendusi ja strateegiaid, mille eesmärgiks on realiseerida veerupõhist salvestamist. PostgreSQL jaoks on loodud vabavaraline laiendus cstore_fdw, mis pakub eeliseid andmeanalüüsiks, kus kasutatakse palju agregatsiooni funktsioone. Nii nagu teistes veerupõhistes andmebaasides toimub ka selles andmete tihendamine, mis vähendab andmemahtu mitu korda. Mälust loetakse ainult vajalikud veerud ning indeksite asemel salvestatakse reagruppide miinimum ja maksimum väärtused [19].

3.2 Veerupõhine andmete salvestamine MariaDB Columnstore näitel

Kaasajal on reapõhistele süsteemidel lisandunud veerupõhise salvestamisega andmebaasisüsteemid, mille kujunemist ja valikut järgnevas peatükis uuritakse. Veerupõhist salvestamist teostatakse erinevatel viisidel. Tuntud andmebaasisüsteemis, Microsoft SQL Server, on loodud näiteks veerupõhised indeksid [20], kuid MariaDB on loonud eraldi andmebaasimootori Columnstore.

3.2.1 Ajalugu

Veerupõhiste andmebaasisüsteemide juured ulatuvad 1970-ndatesse, kuid laiemalt hakkasid levima 2000-ndatel. Cantor oli esimene andmebaasisüsteem, mis sarnases tänapäevase veerupõhise andmebaasisüsteemiga, sest kasutas samu andmete pakkimise tehnikaid nagu kaasaegsed süsteemid. 1985. aastal pakkusid Copeland ja Khoshafian NSM salvestusmudeli kõrvale välja DSM (*Decomposition Storage Model*) salvestusmudeli. DSM eripära seisneb iga veeru eraldi plokki salvestamisel, kus iga kirjega selles veerus salvestatakse ka kirje surrogaatvõti, mille miinuseks on kasvav salvestusmaht. Edasise tehnoloogia arenguga ilmus 1990-ndatel akadeemilise taustaga MonetDB, mida on uuritud ja edasi arendatud väga paljudes akadeemilistes ringkondades. Esimene äriline veerupõhine andmebaasisüsteem, SybaseIQ, ilmus 1996. aastal. See ei leidnud laiemat kasutust, sest tuli turule kui riistvara ei soosinud veel veerupõhist andmemudelit ning selles puudusid olulised arhitektuurilised aspektid nagu näiteks hiline materialiseerimine ja operatsioonid pakitud andmetel. Tänu paremale riistvarale, mis võimaldas koguda ja analüüsida suuremaid andmahte, arendati välja veerupõhiste andmebaasisüsteemide pioneerid, C-Store ja VectorWise, milles on ühendatud peamised tehnikad protsessorite ja salvestusmeediumite tõhusamaks kasutamiseks. Uusi veerupõhise salvestusmudeliga andmebaasisüsteeme ilmus väga palju ning tehnikaid hakati järjest rohkem kombineerima ärilistesse traditsioonilistesse süsteemidesse. Näiteks SAP HANA, mis töötleb hästi nii OLAP kui ka OLTP (*Online Transaction Processing*) päringuid [21].

3.2.2 Veerupõhise andmebaasisüsteemi erinevusi

- Hiline materialiseerimine (ingl *late materialization*)
  Ainult valitud väärtustega töötamiseks on veerupõhistes andmebaasides kasutusele võetud hiline materialiseerimine. Meetod tähendab individuaalsete

veergudega töötamist, kus iga veeru väärtused filtreeritakse eraldi. Selle eesmärgiks on võimalikult hilises staadiumis panna otsitud veergude väärtustest kokku read [22].

- Andmete pakkimine (ingl *compression*)
  Veerupõhisel andmete salvestamise eeliseks on asjaolu, et sama andmetüübiga andmed on võimalik tihedamalt kokku pakkida kui mitme andmetüübiga ridu. Sellega ei loeta ülearuseid bitivektoreid. Vähendamaks veelgi andmete kettalt lugemise ja kirjutamise aega saab mõningate pakkimisalgoritmide kasutamisel töötada otse pakitud andmetel. Eriti hästi töötab eelnev eelis RLE (*Run-Length Encoding*) algoritmiga, kus järjestikused identsed väärtused asendatakse väärtuse ja selle esinemiste arvu paarina [21].
- Andmete lisamine, muutmine ja kustutamine
  Tulenevalt salvestusmudelist on OLTP päringud veerupõhises andmebaasis aeglasemad, sest andmeid tuleb ühe faili asemel kirjutada igasse veeru faili [23]. DML keelt kasutavate käskude aegluse põhjuseks on ka andmete pakkimine, sest uue kirje lisamine tähendab kolme sammu: andmete lahti pakkimist, kirje uuendamist ning andmete uuesti pakkimist [21].

3.2.3 Saaty otsustusmudel sobivaima veerupõhise andmebaasisüsteemi valikul

Veerupõhise andmebaasisüsteemi valikul kasutati varasemalt õpitud analüütiliste hierarhiate meetodit ehk Saaty otsustusmeetodit. Meetod on nime saanud Thomas L. Saaty järgi, kes 1970-ndatel selle välja arendas. Meetod põhineb paaritisel võrdlusel ja on mõeldud subjektiivsete hinnangute alusel objektiivse hinnangu saamiseks. Meetodi struktuur jaguneb kolmeks: eesmärk, kriteeriumid ja alternatiivid. Eesmärgi saavutamiseks on seatud kriteeriumid, mille alusel alternatiive paarikaupa võrreldakse. Meetod annab lõpptulemuseks välja kõige sobilikuma alternatiivi kasutaja poolt seatud kriteeriumitele [24].

Antud töös on kasutusel Saaty skaala arvulised hinnangud 1 kuni 9 ning nende pöördväärtused, sest meetodi idee on kriteeriumite paarikaupa võrdlemine. Kui elemendil A on elemendi B suhtes hinnang H, siis elemendil B on elemendi A suhtes hinnang 1/H. Saaty mudelis antakse suhtelised hinnangud sõnaliselt: võrdtähtis, mõõdukas paremus, oluline paremus, väga tugev paremus ning ekstreemne paremus, millele vastavad numbrilised hinnangud 1, 3, 5, 7 ja 9. Hinnangud 2, 4, 6, 8 on

kompromissid kahe naaberhinnangu vahel. Töös kasutatakse tasuta tarkvara SuperDecisions ning hinnangud on antud ainult autori enda poolt. Hinnatavad kriteeriumid on toodud tabelis 1, kus igale kriteeriumile on lisatud identifikaator ning selgitus. Tabelis 2 on toodud alternatiivid koos identifikaatoriga.

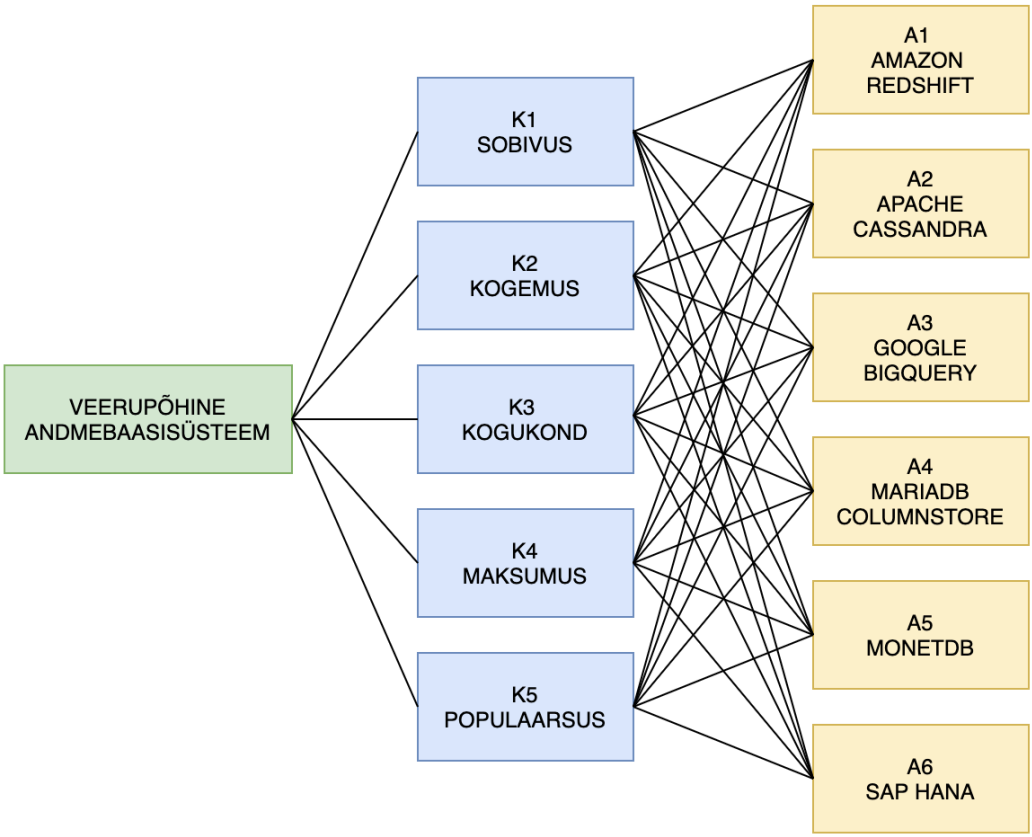
Tabel 1. Saaty mudelis hinnatavad kriteeriumid.

Identifikaator	**Kriteerium**	**Selgitus**
K1	Sobivus	Alternatiivide hindamisel antud kriteeriumi suhtes tuleb arvestada jaekaubanduse andmete eripäradega, näiteks palju erinevaid tooteid, kuid ostukorvis keskmiselt ainult 7 toodet.
K2	Kogemus	Kindla eelise annab autori varasem kogemus andmebaasisüsteemiga, millel valitav põhineb (MySQL, PostgreSQL).
K3	Kogukond	Probleemide tekkimise korral peaks vastuse leidma kiirelt dokumentatsioonist või foorumist, kus mõni teine kasutaja on varasemalt juba sama probleemi tõstatanud.
K4	Maksumus	Oluline eelis on vabavaralistel andmebaasisüsteemidel. Alternatiivide hindamisel on arvesse võetud prooviperioodi ja selle sobivust lõputöö praktilise osa täitmisel.
K5	Populaarsus	Kriteerium on oluline, sest valitav andmebaasisüsteem peaks olema laiemalt kasutusel, et analüüs annaks edaspidiseks kasutatava tulemi. Populaarsuse hindamisel vaadatakse alternatiivide järjestust veebilehelt, mida uuendatakse igakuiselt. Paremusjärjestuse koostamisel arvestatakse mitmete parameetritega, näiteks mainimine veebilehtedel ja tehniliste arutelude sagedus [15].

Tabel 2. Andmebaasisüsteemide alternatiivid Saaty mudelis.

Identifikaator	**Alternatiiv**
A1	Amazon Redshift
A2	Apache Cassandra
A3	Google BigQuery
A4	MariaDB Columnstore
A5	MonetDB
A6	SAP HANA

Saaty mudeli eesmärgi, kriteeriumite ja alternatiivide mudel on toodud joonisel 2.



Joonis 2. Saaty mudel veerupõhise andmebaasisüsteemi valikul.

Jooniselt võib näha vasakul olevat eesmärki, keskel kriteeriume koos nende tähistega ning paremal hinnatavaid alternatiive koos identifikaatoriga.

Tabelis 3 on toodud kõigi 5 kriteeriumi paaritiste võrdluste tulemused ning lõpptulemuses arvestatavad kaalud.

Tabel 3. Kriteeriumite paaritine võrdlustabel.

	K1	**K2**	**K3**	**K4**	**K5**	**Kaalud**
K1	1,00	3,00	7,00	3,00	5,00	**0,468**
K2	0,33	1,00	3,00	0,33	0,33	**0,093**
K3	0,14	0,33	1,00	0,20	0,20	**0,040**
K4	0,33	3,00	5,00	1,00	3,00	**0,248**
K5	0,20	3,00	5,00	0,33	1,00	**0,150**

Kõige olulisemaks kriteeriumiks on hinnatud sobivus, mille kaal on 0.468. Kõrgelt on hinnatud ka maksumust, sest hind on oluline tegur valides andmebaasisüsteemi pikemaks ajaks. Kõige vähem olulisem on kogukond hinnanguga 0.040.

Tabelis 4 on kombineeritud iga alternatiivi paaritine võrdlus teiste alternatiividega igas kriteeriumis. Tabel on koostatud ülevaatlikult, et mõista lõpptulemuses esinevaid kaale. Eraldi tabelid alternatiivide hinnangutele kõigis kriteeriumites on kirjeldatud töö lisades (Lisa 3).

Tabel 4. Alternatiivide kokkuvõtte kaalud.

	A1	**A2**	**A3**	**A4**	**A5**	**A6**
K1	0,100	0,027	0,100	0,421	0,291	0,061
K2	0,084	0,067	0,173	0,316	0,246	0,114
K3	0,065	0,115	0,061	0,528	0,193	0,040
K4	0,042	0,292	0,042	0,292	0,292	0,042
K5	0,052	0,441	0,083	0,254	0,025	0,144

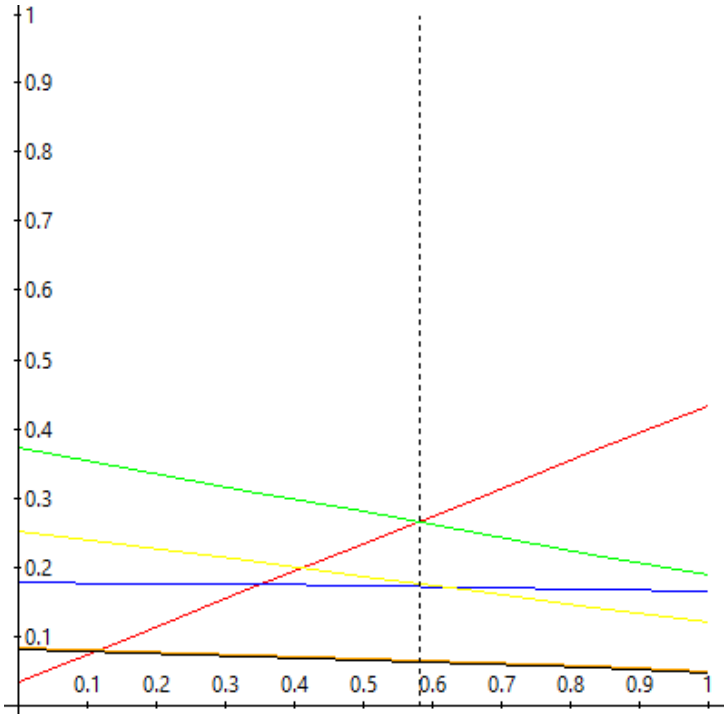
Tabelis 5 on kokkuvõtlikult toodud iga alternatiivi hinnang igas kriteeriumis, mis on läbi korrutatud kriteeriumi enda kaaluga.

Tabel 5. Saaty meetodi lõpptulemus.

	A1	**A2**	**A3**	**A4**	**A5**	**A6**
K1	0,047	0,012	0,047	0,197	0,136	0,029
K2	0,008	0,006	0,016	0,029	0,022	0,011
K3	0,003	0,005	0,002	0,021	0,008	0,002
K4	0,010	0,072	0,010	0,072	0,072	0,010
K5	0,008	0,066	0,012	0,038	0,004	0,022
Kokku	**0,076**	**0,161**	**0,077**	**0,357**	**0,242**	**0,074**

Saaty mudeli alusel on parimateks alternatiivideks MariaDB Columnstore ja MonetDB. Valituks osutus MariaDB Columnstore, sest saavutas mitmes kriteeriumis kõige parema tulemuse ning oli stabiilselt parimate seas ka vähem olulistes kriteeriumites.

Hindamaks mudeli sobivust on tehtud ka tundlikkuse analüüs vaatamaks, kas mõne muudatuse tegemine on otstarbekas. Joonisel 3 on toodud üks tundlikkuse analüüsi graafik.



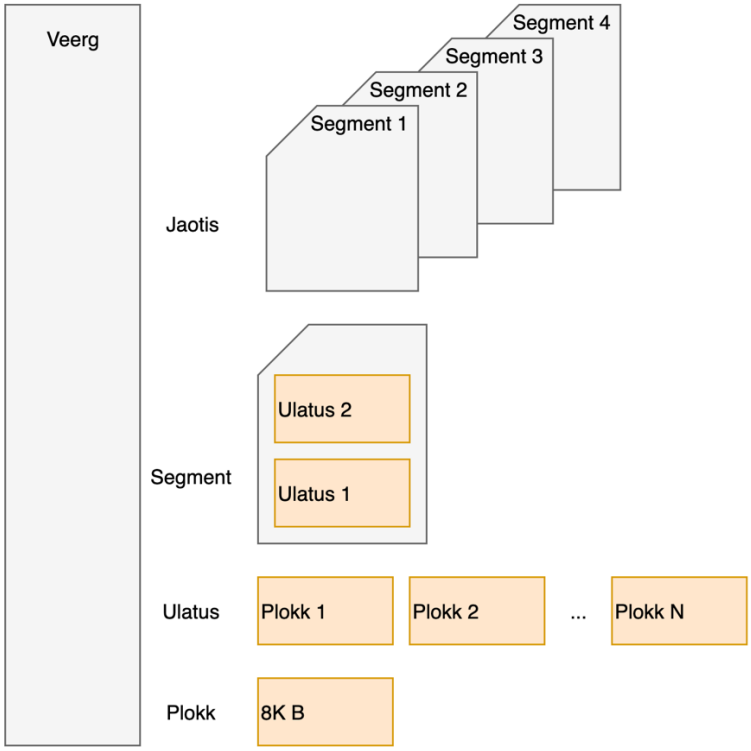
Joonis 3. Tundlikkuse analüüs kriteeriumi sobivus kohta.

Tundlikkuse analüüsist sobivuse kriteeriumi kohta on näha, et kui kriteeriumi kaalu tõsta 0.58-ni, siis võiks parimaks osutuda Amazon Redshift, mis on graafikul punase joonena. Nii suure osakaalu andmine sobivuse kriteeriumile ei ole mõistlik, sest sellisel juhul tulekski suures osas lähtuda ainult antud kriteeriumist.

MariaDB Columnstore sai valitud veerupõhise salvestuse demonstreerimiseks, sest saavutas kõrge hinnangu Saaty meetodi kasutamisel. MariaDB Columnstore on veerupõhine andmebaasisüsteem, mis on mõeldud suurandmetega töötamiseks, sest koondab veerupõhise salvestuse, andmete pakkimise ja oskusliku horisontaalse ning vertikaalse jaotamise. Andmebaasisüsteem on vabavaraline, põhineb MySQL-il, laia kasutajaskonnaga ning väga populaarne, sest on andmebaasisüsteemidest 12. kohal [15]. Töös kasutatakse MariaDB Columnstore 1.2.2 versiooni vaikimisi seadistustega.

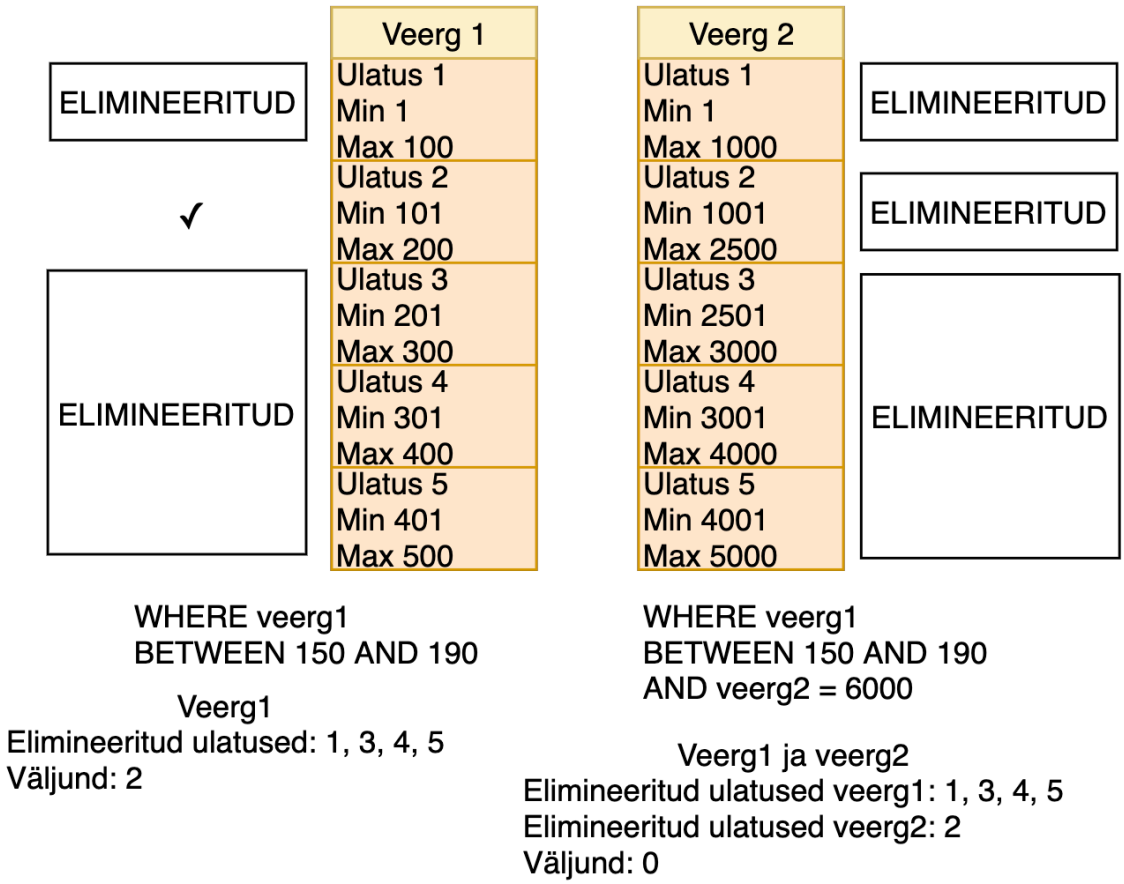
3.2.4 MariaDB salvestusmudel

Tabeli loomisega MariaDB Columnstore andmebaasisüsteemis kasutatakse vertikaalselt jaotamist ning luuakse igale tabeli veerule vähemalt üks salvestusfail. Iga tabeli iga veeru väärtused andmebaasis salvestatakse eraldi plokki, mille suurus on kuni 8 KB. Plokk on ketta sisend-väljund ühik. Kui veerus on rohkem andmeid, siis luuakse sama veeru kohta uus plokk. Plokkide kogum moodustab ulatuse (ingl *extent*). Füüsiliselt salvestatakse kettale segmentide failid, mis koosnevad ulatuste failidest. Ühe veeru segmendid moodustavad horisontaalse jaotise [25] – [26]. Joonis 4 illustreerib MariaDB Columnstore salvestamist andmebaasi sisemisel tasandil.



Joonis 4. Salvestusmudel MariaDB Columnstore andmebaasisüsteemis [26].

Ulatuste kaart (ingl *Extent map*) on püsiv loogiline andmestruktuur, mida süsteem hooldab, et kindlustada vajalikud metaandmed ulatuse tasemel. Kaardi eesmärgiks on kaotada vajadus kasutada indekseerimist, tabeli manuaalset jaotamist, materialiseeritud vaadete ja koondtabelite loomist ning teisi struktuure, mis kiirendaksid päringu jõudlust. Ulatused salvestatakse plokkidesse koos minimaalse ja maksimaalse väärtusega, mida jälgides mõistab kaart, millised plokid ei vasta päringu tingimustele ja tuleks elimineerida. Meetod sobib hästi näiteks sorteeritud või klasterdatud andmetele, kus toimub sage andmete laadimine ning päringus viidatakse ajale [26]. Joonisel 5 on toodud näide ulatuste elimineerimisest.



Joonis 5. Ulatuste elimineerimine [26].

Joonisel 5 on toodud 2 veergu ning näitena 2 tingimust. Kui arvestada ainult esimese rea tingimusega, siis väljastataks ulatus 2. Luues tingimuse ka teisele veerule ei väljastata midagi, sest teise veeru tingimus elimineerib esimese veeru poolt sobiva väljundi.

MariaDB Columnstore kasutab andmete pakkimisel ettevõtte Google poolt 2011.aastal loodud Snappy teeki, mis on kirjutatud programmeerimiskeeles C++. Snappy teegi eelisteks on kiirus, stabiilsus, robustsus ja vabavaralisus. Eripäraks on väga kiire protsess, kuid mõõdukas andmete tihendamine. Teek kasutab baitidele orienteeritud kodeeringut. Põhineb LZ77 algoritmil [27], mis kuulub kadudeta andmete pakkimise meetodite hulka ja on kaasaja süsteemides laialt levinud. Snappy tihendamise kiirus ühe tuumalisel 64-bitisel seitsmenda generatsiooni protsessoril on 250 MB/s ja hõrendamise kiirus 500 MB/s, kuid võrreldes gZIP meetodiga on tihendatud failid 20%–100% suuremad [28].

3.2.5 Mõned MariaDB Columnstore eripärad

Ridade lugemine (*SELECT COUNT(*)*) on andmebaasisüsteemis optimeeritud kujul *SELECT COUNT(COL_N),* kus *COL-N* tähistab veergu, mis kasutab kõige vähem baite salvestamiseks. Erinevus seisneb NULL väärtuste lugemisel, mida *SELECT COUNT(COL-N)* ei tee [25].

Sorteerimist (*ORDER BY*) ja limiteerimist (*LIMIT*) rakendatakse protsessi lõpus ajutisel tulemuste tabelil. See tähendab, et kõige pealt tuleb saada kõik sorteerimata andmed ja alles siis saab neid väljastuseks muuta. Suurte andmehulkade peal vähendab see märkimisväärselt päringu kiirust [25].

Columnstore kasutab räsi ühendamisoperatsioone (ingl *hash join*) vältimaks indeksite ja *nested loop* ühendamisoperatsioone ja samaaegselt optimeerida suuremahulisi ühendamisi [25].

MariaDB Columnstore andmebaasisüsteemis veergudele formaate andes tuleb lähtuda salvestusmudelist ja mõelda teksti formaatide kasutamise vajalikkusele. Stringid, mis on pikemad kui 8 baiti salvestatakse eraldi sõnastiku põhisesse ulatusesse ja veerupõhisesse ulatusesse salvestatakse viit sellele ulatusele. Parema jõudluse saavutamiseks tuleks vältida sõnastiku teatmeotsingut [25].

3.3 Varasemad uuringud andmebaasisüsteemide võrdlusest

Andmebaasisüsteemide võrdlustel põhinevaid artikleid on kaasajal väga palju, sest uusi süsteeme tuleb pidevalt juurde. Populaarseimates artiklites on võrreldavateks NoSQL-andmebaasisüsteemid, mis on viimasel ajal laialt levima hakanud. Artiklite leidmiseks kasutati Google Scholar platvormi, kus märksõnadeks kasutati järgnevaid fraase: *column-store, column-oriented dbms, column store vs row store.*

Veerupõhiste andmebaasisüsteemide jõudluse parendamiseks testitakse dünaamilisi muutmälusid. Artiklis [29] on loodud tehis veerupõhise andmebaasisüsteemi ja FPGA (*Field-Programmable Gate Array*) koostööl, mis kiirendab andmete eraldamist SQL-päringu töötlemiseks, sest võrreldes DRAM mälu (*Dynamic Random Access Memory*) SSD (*Solid-State Drive*) mäluga on viimane odavam ja energiasäästlikum. Töös loodi uus veerule orienteeritud andmeformaat, et FPGA tuleks toime suurt jõudlust nõudva

andmetöötlusega. Avastati, et materialiseerimine tarbib 34% protsessori jõudlusest, mille vähendamiseks võeti kasutusele uus strateegia. Lisaks jaotati rollid andmebaasisüsteemi ja tehise vahel, et saada üle FPGA mälupiirangutest. Autorid võrdlesid oma prototüüp süsteemi, FCAccel, andmebaasisüsteemidega, mis salvestasid andmeid erinevalt. Eksperimendis saavutas loodud tehis 0.77–1.79 korda parema jõudluse võrreldes andmebaasisüsteemiga MonetDB. Andmebaasisüsteem Impala oli 4.10–23.4 korda kehvema jõudlusega kui FCAccel.

2016.aastal tehtud magistritöös [30] on põhjalikult analüüsitud rea- ja veerupõhist andmete salvestamist kahes erinevas andmebaasisüsteemis. Teooria osas käsitletaks veerupõhist salvestamist C-store näitel. Võrreldud on andmekäitluse operatsioonide jõudlust ja andmebaaside andmemahtu. Autor valis võrreldavateks andmebaasisüsteemideks MonetDB ja Microsoft SQL Server. Andmebaasisüsteemides on realiseeritud neli erinevat andmebaasi. Esimeses reapõhine salvestus ning veerupõhine salvestus ja teises veerupõhine salvestus ja veerupõhine salvestus koos indeksitega. Neid võrreldakse omavahel külma ja sooja käivitamise korral. Hüpoteesid, mis on esitatud veerupõhisele salvestusmudelile, on leidnud tõestust. Andmete muutmispäringud on Microsoft SQL Server-i andmebaasides sama kiirusega, sest veerupõhisel salvestamisel on kasutusel *deltastore*. Töös on järeldatud, et veerupõhine salvestusmudel sobib paremini analüütiliste päringute sooritamiseks, milles töödeldakse rohkem kui 10% ridadest, kuid vähe veerge. Magistritööst võib järeldada, et veerupõhine andmete salvestamine võiks sobida jaekaubanduse valdkonda.

Artiklis [31] on võrreldud C-store jõudlust erinevate kommertslike andmebaasisüsteemidega, mis kasutavad tähtskeemi. Tuuakse välja kolm võimalust, kuidas traditsioonilises andmebaasis veerupõhise andmebaasi jäljendamist realiseerida - vertikaalse jaotamisega, indeksitega ja materialiseeritud vaadetega. Lähemalt uuritakse veerupõhiste andmebaaside jõudlust ning tuuakse välja tüüpilised optimeerimismeetodid–andmete pakkimine, hiline materialiseerimine, ploki iteratsioon ja nähtamatud ühendamisoperatsioonid. Tulemusena näidatakse töös, et veerupõhise salvestusmudeli jäljendamine indeksite kasutuselevõtuga ja vertikaalse jaotamisega traditsioonilises andmebaasisüsteemis ei anna head tulemust, sest andmete taastamisele kulub palju aega. Leiti, et hiline materialiseerimine ning andmete pakkimine parandavad kordades päringu jõudlust, mis pärib järjestatud andmetest. Lisaks on esitatud meetod nimega *„invisible joins“*, mis parandas jõudlust veel 50%.

4 Eksperiment

Antud peatükis on detailsemalt kirjeldatud eksperimendi eesmärk, eksperimendi disain ja sooritamine ning viimaks eksperimendi tulemused.

4.1 Eksperimendi eesmärk

Teostatud eksperimendi eesmärgiks on võrrelda, kuidas erineb rea- ja veerupõhisel salvestamisel päringute töökiirus SQL-andmebaasisüsteemides konkreetselt jaekaubanduse valdkonnas ostuandmete analüüsimisel. Eksperimendis on kirjeldatud andmete lisamine andmebaasidesse, kuid andmete lisamise, kustutamise ja muutmise operatsioonide töökiirust ei ole detailselt võrreldud.

4.2 Eksperimendi kirjeldus

Eksperimendi läbi viimisel oli oluline kasutada sama riist- ja tarkvara ning andmestikku identsete andmebaaside loomiseks vältimaks ressursside eraldamisel tekkivaid eeliseid. Järgnevalt on välja toodud eksperimendi ülesehitus ning eksperimendis osalevate andmebaaside disain. Esimeses jaotises kirjeldatakse kasutatavat riistvara ja tarkvara koos andmestiku iseloomustusega. Edasi tuuakse välja andmebaaside loogiline disain, mille loomine on oluline andmebaaside projekteerimisel. Viimaks kirjeldatakse andmebaaside realiseerimist ning andmete importimist.

4.2.1 Riistvara ja tarkvara

Testid viiakse läbi sülearvutil MacBook Pro, millel võeti kasutusele Docker, mis võimaldab luua andmebaasidele identse keskkonna päringute käivitamiseks. Docker on raamistik, mis virtualiseerib operatsioonisüsteemi ning võrreldes virtuaalmasinaga võtab vähem ruumi ressurssi [32]. Docker-i sees võib olla üks või mitu konteinerit, milles käitatakse spetsiaalset tarkvara. Käesolevas eksperimendis on loodud 2 konteinerit–üks, mis realiseerib andmebaasisüsteemi MariaDB Columnstore versiooni 1.2.2 vaikimisi seadistustega ja teine andmebaasisüsteemi PostgreSQL versiooni 12.2 vaikimisi seadistustega.

Tabel 6 illustreerib eksperimendis kasutatava ressursi. Mõlemal on kasutusel 6-tuumaline Intel protsessor. Süsteemil on 16 GB vahemälu, millest Docker-ile eraldati 8 GB.

Tabel 6. Ressursside jagunemine.

	Host	**Docker**
CPU	6-tuumaline	6-tuumaline
vahemälu	16 GB	8 GB
SSD	512 GB	132 GB

Andmebaasidega töötamiseks kasutati käsurida ning universaalset kasutajaliidest DBeaver, milles on toetatud mõlemad andmebaasisüsteemid.

4.2.2 Andmestiku iseloomustus

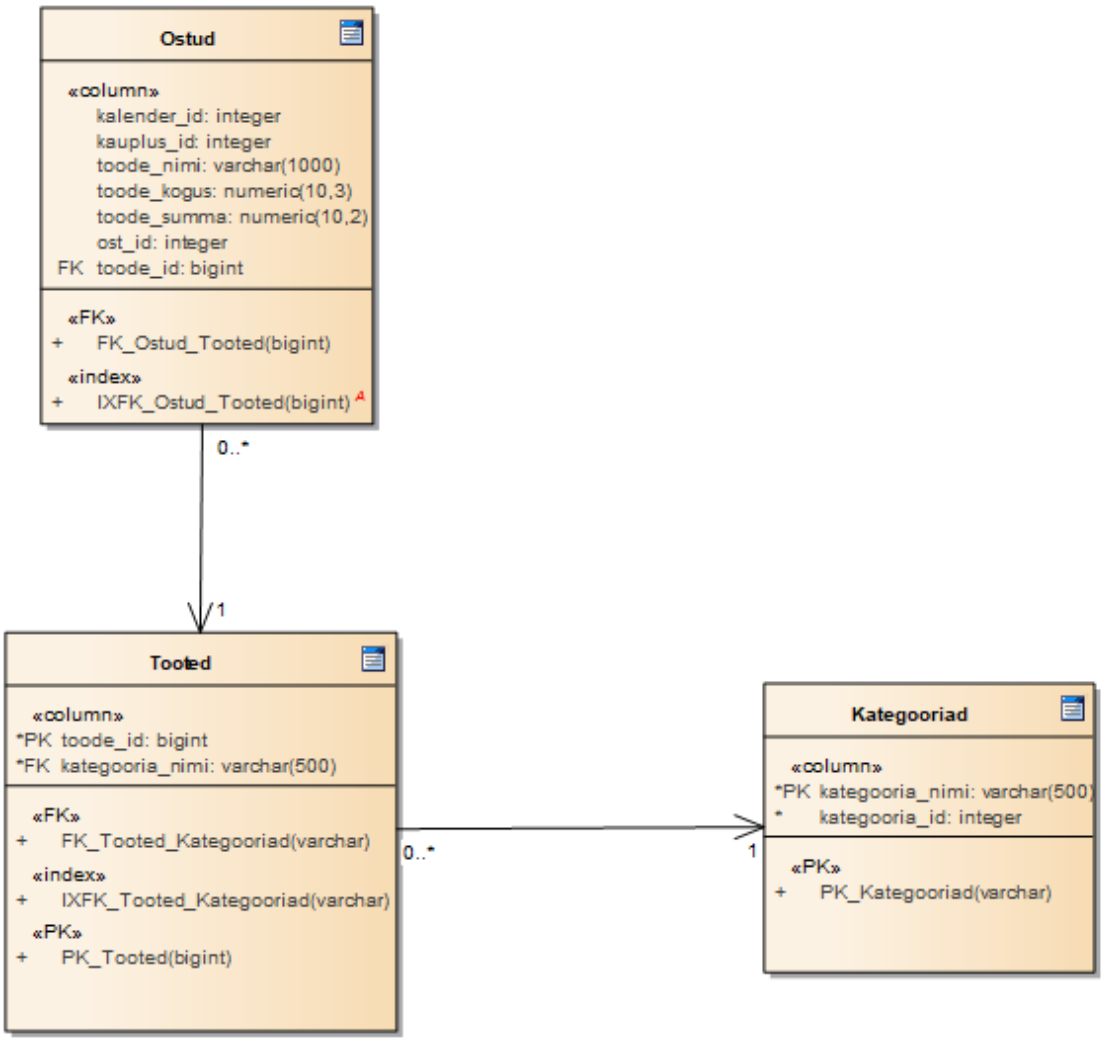
Töös on kasutatud reaalseid andmeid, mis on saadud koostöös Eesti jaekaubanduses tegutseva ettevõttega, mis eelistab jääda anonüümseks. Andmed imporditi rea- ja veerupõhisesse andmebaasi ettevõttelt saadud CSV *(Comma Separated Value)* failidest. Andmestiku suurus on ligikaudu 40 GB, mis sisaldab ühe aasta jooksul ettevõttes tekkinud ostuandmeid. Tabelis 7 on toodud puhastatud andmete arvuline jagunemine tabelistesse Ostud, Tooted ja Kategooriad. Kõikidele ostetud toodetele tabelis Ostud vastab kategooria nimi tabelis Tooted, kuid mitte vastupidi.

Tabel 7. Andmestiku üldiseloomustus.

Tabel	**Ridade arv**
Ridade arv tabelis Ostud	~ 420 000 000
Ridade arv tabelis Tooted	~ 700 000
Ridade arv tabelis Kategooriad	240
Ostude arv tabelis Ostud	~ 62 000 000
Unikaalsete toodete arv tabelis Ostud	~ 300 000

4.2.3 Andmebaaside loogiline disain

Joonisel 6 on toodud andmebaaside loogiline disain UML (*Unified Modeling Language*) skeemina. Skeemi jälgides on loodud andmebaaside struktuur.



Joonis 6. Andmebaasi loogiline disain.

Andmebaasi loogiline disain koosneb kolmest tabelist: Ostud, Kategooriad ja Tooted. Tabelis Ostud on ühe aasta ostutehingud. Tabelis Tooted on tootekood koos toote kategooria nimega. Antud tabeli välisvõtmeks on valitud veerg kategooria_nimi. Paremaks andmeanalüüsiks oli vajalik tähti sisaldavatele kategooriatele vastavusse seada numbriline väärtus ja selle realiseerimiseks on loodud tabel Kategooriad, kus igale kategooria nimele vastab kategooria identifikaator. Tabelis on primaarvõtmeks määratud kategooria_nimi, mis ühendatakse tabeliga Tooted. Andmebaasi loogiline disain on sarnane eksperimendis osalevates andmebaasisüsteemides, välja arvatud indeksid. Andmebaasisüsteemis MariaDB on välisvõtmeid võimalik luua ainult InnoDB

ja PBXT andmebaasimootorites [25]. MariaDB Columnstore andmebaasimootor on sisemiselt nii optimeeritud, et selles pole vaja luua indekseid [26]. Sel põhjusel on ka PostgreSQL-is loodud ainult primaar - ja välisvõtme indeksid.

4.2.4 Andmebaaside realiseerimine

Peale andmebaasi loogilist disaini loodi andmebaasid andmebaasisüsteemides PostgreSQL ja MariaDB Columnstore. PostgreSQL andmebaasi loomisel käivitatud SQL laused on toodud välja töö lisas. (Lisa 2) Andmebaasisüsteemis MariaDB Columnstore käivitatud laused on lisatud eraldi, sest selles andmebaasis pole lisatud indekseid, tuli täpsustada andmebaasimootor ning andmetüübid erinesid nime poolest. (Lisa 3)

4.2.5 Andmete lisamine

MariaDB Columnstore tabelitesse andmete importimiseks kasutati cpimport utiliiti, mis võimaldab kiirelt ja tõhusalt andmefailist andmed veerupõhiselt salvestavasse andmebaasisüsteemi sisse lugeda. Utiliit teostab andmete importimisel järgmised operatsioonid:

1. Andmete lugemine andmefailist.
2. Andmete transformeerimine sobitumaks veerupõhise salvestamise disainiga.
3. Liiasusega andmete sõnestus (ingl *tokenization*) ja loogiline kokku pakkimine.
4. Andmete kirjutamine kettale [25].

PostgreSQL andmebaasi andmete lisamiseks oli võimalik kasutada kahte meetodit– COPY käsk käsureal või importimine kasutades DBeaver kasutajaliidest. Autor kasutas DBeaver kasutajaliidest, mis oli aeglasem, kuid mugavam.

4.3 Eksperimendis uuritavad päringud

Andmepäringute valimi moodustavad peamised infovajadused, mis on teisendatud SQL-andmebaasikeelde. Valitud on 12 päringut, mille puhul võrreldakse aega tulemuse leidmiseks kahest andmebaasist, millest üks kasutab reapõhist salvestamist ja teine veerupõhist. Igale päringule on omistatud unikaalne identifikaator, mis aitab neile tekstis viidata. Välja on toodud küsimus, millele päring vastab, päringute väljundid ja selgitused.

3 päringut hõlmavad kõiki kauplusi, 6 päringut on koostatud ühe kaupluse kohta ning 3 päringut on käivitatud vaadeldes korraga viite kauplust. Kauplused on valitud juhuvalikuga, et nad erineksid käivete ja ostude arvude poolest. Lisaks on käivitatud ka päringud, mis on sisendiks Apriori algoritmile, et leida koos ostetavaid kategooriaid (P10). Peamised infovajadused ostuandmete pealt on leitud kasutades artikleid peatükist 2.2 Varasemad uurimused jaekaubanduses, suhtlusest valdkonna inimestega ning lisaks tutvudes erinevate äritarkvara pakkuvate ettevõtete lähenemistega [33] – [35].

4.3.1 Kaupuste käibed aasta jooksul

Identifikaator: P1

Küsimus: Mis on kõikide kaupluste käive aastas?

Väljund: kauplus_id, kaive

Selgitus/kirjeldus: Päring on ülevaatlik ja annab juhatusele aimduse, kuidas jaguneb kaupluste käive. Antud päring on sisendiks kaupluste grupeerimiseks käivete järgi.

Täitmisplaanid: Lisa 4, Lisa 5

4.3.2 Kaupluste käivete jagunemine protsentuaalselt kategooriate vahel

Identifikaator: P2

Küsimus: Kuidas jaguneb kaupluse käive kategooriate lõikes protsentides ning mis kategooriad moodustavad suurima osa käibest ja on kõige kasumlikumad?

Väljund: kauplus_id, kategooria_id, protsent

Selgitus/kirjeldus: Päringu eesmärgiks on saada teada, millised on ühe kaupluse 5 kõige kasumlikumat kategooriat. Päringu väljund edasisel analüüsil oleks üheks sisendiks riiulite ümberpaigutamisel ja ka sortimendi muutmisel. Päring võimaldab lihtsat klasterdamist protsentide alusel. Tingimuses võiks defineerida kaupluse koodi ning suurendada väljundis kuvatavate kategooriate arvu, et saada ühe kaupluse kategooriate täpsem jagunemine.

Täitmisplaanid: Lisa 6, Lisa 7

4.3.3 Keskmine ostukorvi maksumus ja toodete arv ostukorvis igas kaupluses kindlas kuus

Identifikaator: P3

Küsimus: Mis on keskmine ostukorvis olevate toodete arv ja summa?

Väljund: kuu, kauplus_id, avg_sum, avg_arv

Selgitus/kirjeldus: Kaupluste võrdlemisel ning grupeerimisel on keskmised parameetrid olulised, sest võimaldavad teha üldistusi ning profileerida kauplusi optimaalselt.

Täitmisplaanid: Lisa 8, Lisa 9

4.3.4 Kõikide kaupluste käivete ja ostude arv kvartalis

Identifikaator: P4

Küsimus: Millised olid kaupluste müügimahud eelmises kvartalis?

Väljund: kauplus_id, ost_arv, ost_sum

Selgitus/kirjeldus: Päring annab ülevaade kaupluste müügimahtudest, mida kombineerides teiste andmetega annab kvartalitulemused.

Täitmisplaanid: Lisa 10, Lisa 11

4.3.5 Ühe kaupluse ostude arv ja käive nädalate lõikes aasta jooksul

Identifikaator: P5

Küsimus: Kuidas on jagunenud ühe aasta käive ja ostude arv nädalate lõikes?

Väljund: nädal, ost_arv, ost_sum

Selgitus/kirjeldus: Päringu väljund aitab planeerida tööjõudu ning ka kaupade sisseostu, sest annab aimu, millistel nädalatel on nõudlus suurem.

Täitmisplaanid: Lisa 12, Lisa 13

4.3.6 Ühe kaupluse ostude arv ja käive kuude lõikes aasta jooksul

Identifikaator: P6

Küsimus: Kuidas on jagunenud ühe aasta käive ja ostude arv kuude lõikes?

Väljund: kuu, ost_arv, ost_sum

Selgitus/kirjeldus: Taaskord aitab päringu väljund planeerida tööjõudu ning määrata puhkuse andmist.

Täitmisplaanid: Lisa 14, Lisa 15

4.3.7 Ühe kaupluse ostude arv päevade lõikes igas tunnis

Identifikaator: P7

Küsimus: Kuidas on kaupluse külastused (ostude arv) päevade peale igas tunnis jagunenud? Mis nädalapäeval on kõige rohkem poekülastusi?

Väljund: päev_nr, tund, ost_arv

Selgitus/kirjeldus: Väljundandmed on paslik visualiseerida kasutades soojakaarti, sest see aitab koheselt tuvastada rahvarohkemad nädalapäevad ja kellaajad päevas. Graafik oleks abiks tööjõu paremaks planeerimiseks, sest teatud ajavahemikes on tööjõu vajadus suurem. Graafikult on näha, kuidas on jagunenud kaupluse külastused, sest loendatakse transaktsioonide arvu mitte ostukorvi suurust.

Täitmisplaanid: Lisa 16, Lisa 17

4.3.8 Ühes kaupluses müüdud toodete kogus nädala jooksul ühes kategoorias

Identifikaator: P8

Küsimus: Milline on ühe kaupluse kategooria ülevaade müüdud kogustest?

Väljund: kuupäev, toode_nimi, arv

Selgitus/kirjeldus: Kategooriajuhid, kes igapäevaselt kaupa tellivad, arvestavad tellimusi tehes eelneva nädala müügiga või kindla ajavahemikuga, sooduskampaaniatega ning lähenevate pühade ja tähtpäevadega. Olemasolevatest

andmetest on võimalik koostada päring, mis annaks väljundiks kaupluses juba ostetud kaupade ja koguste andmed, mida arvestades teha tellimus vältimaks kauba halvaks minemist ja samas ka kauba puudust enne järgmise tellimuse saabumist.

Täitmisplaanid: Lisa 18, Lisa 19

4.3.9 Uue toote populaarsus 5 kaupluses nädala jooksul

Identifikaator: P9

Küsimus: Kuidas muutub uue toote müük ühes nädalas?

Väljund: kuupaev, ost_arv, ost_sum

Selgitus/kirjeldus: Kategooriajuhid vahetavad pidevalt sortimente ning võtavad prooviks uudistooteid. Perioodiks valiti nädal, sest peale selle möödumist tuleb otsustada, kas tellida sama toodet juurde või vahetada see välja.

Täitmisplaanid: Lisa 20, Lisa 21

4.3.10 Ostudes sisalduvad unikaalsed kategooriad

Identifikaator: P10

Küsimus: Mis tootekategooriaid ostetakse koos?

Väljund: ost_id, kategooria_id

Selgitus/kirjeldus: Päring on vajalik hilisemaks Apriori algoritmi realiseerimiseks. Algoritmi rakendamisel soovitakse aru saada, milliseid kategooriaid ostetakse tihti koos ning sel juhul ei oma identsete kategooriate arv tähtsust. Algoritmi sisendiks antakse ühes ostukorvis olevate unikaalsete kategooriate list. Andmeid päritakse kindlate kaupluste kohta, sest finantsiliselt võib olla sarnane ruumipaigutus odavam ja kergesti jäljendatav ka teistes kauplustes.

Täitmisplaanid: Lisa 22, Lisa 23

4.3.11 Sortimendis olevate toodete arv kategooriate kaupa ühes kaupluses

Identifikaator: P11

Küsimus: Millise sortimendiga on kindel kauplus? Mis tootegruppides on enim tooteid?

Väljund: kategooria_nimi, toode_arv

Selgitus/kirjeldus: Päring annab ülevaate, milliste kategooriate sortiment on kõige laiem. Kui hakatakse sarnase pindalaga kauplust planeerima, siis osatakse arvestada riiuliruumi paigutusel eelnevate probleemidega.

Täitmisplaanid: Lisa 24, Lisa 25

4.3.12 Ühes kaupluses müüdud ühe ettevõtte tooted

Identifikaator: P12

Küsimus: Milliseid tooteid on ettevõtte sortimendist tellitud?

Väljund: toode_nimi, kategooria_nimi

Selgitus/kirjeldus: Päring oleks sisendiks uute toodete tellimisele, sest kategooriajuht on kursis kaupluses olnud või olevate toodetega.

Täitmisplaanid: Lisa 26, Lisa 27

4.4 Eksperimendi sooritamine

Kahe andmebaasi peal on käivitatud SQL-päringuid ja võrreldud andmebaaside töökiirust. Mõlemas andmebaasis käivitati päringuid DBeaver andmehaldusvahendis, kus väljundi ridade hulk oli piiratud 200 kirjega ning käsureal, kus väljastati hilisemaks analüüsiks vajalikud täitmisplaanid. Olenemata asjaolust, et mõlemad on SQL-andmebaasisüsteemid, on päringutes süntaks erinev, sest MariaDB põhineb MySQL-l. Tabelis 8 on välja toodud mõlemas andmebaasisüsteemis käivitatud SQL-päringud. Käivitatud andmepäringute täitmisplaanid on välja toodud töö lisades (Lisa 4–Lisa 27). Andmepäringute loetavuse lihtsustamiseks on kõigi päringute tabelite nimedest välja jäetud PostgreSQL andmebaasisüsteemi puhul skeemi nimi ning MariaDB Columnstore süsteemis samanimeline andmebaasi nimetus.

Tabel 8. PostgreSQL ja MariaDB Columnstore andmebaasisüsteemides käivitatud andmepäringud.

Päring	**PostgreSQL**	**MariaDB Columnstore**
P1	SELECT kauplus_id, SUM(toode_summa) AS ost_sum FROM ostud GROUP BY kauplus_id ORDER BY ost_sum DESC;	SELECT kauplus_id, SUM(toode_summa) AS ost_sum FROM ostud GROUP BY kauplus_id ORDER BY ost_sum DESC;
P2	SELECT t.kauplus_id, t.kategooria_nimi, t.protsent FROM (SELECT ostud.kauplus_id, tooted.kategooria_nimi, ROUND(SUM(toode_summa)*100 / t1.kaive,2) AS protsent, ROW_NUMBER() OVER (PARTITION BY ostud.kauplus_id ORDER BY ROUND(SUM(toode_summa)*100 / t1.kaive,2) DESC) AS rea_number FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id JOIN (SELECT kauplus_id, ROUND(SUM(toode_summa),0) AS kaive FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id WHERE ostud.kauplus_id IN (202304, 200001, 200840, 202516, 200631) GROUP BY ostud.kauplus_id) AS t1 ON t1.kauplus_id = ostud.kauplus_id WHERE ostud.kauplus_id IN (202304, 200001, 200840, 202516, 200631) GROUP BY kauplus_id, tooted.kategooria_nimi, t1.kaive ORDER BY kauplus_id, protsent ASC) AS t WHERE t.rea_number <= 5;	SELECT t.kauplus_id, t.kategooria_nimi, t.protsent FROM (SELECT ostud.kauplus_id, tooted.kategooria_nimi, ROUND(SUM(toode_summa)*100 / t1.kaive,2) AS protsent ,ROW_NUMBER() OVER (PARTITION BY ostud.kauplus_id ORDER BY ROUND(SUM(toode_summa)*100/ t1.kaive,2) DESC) AS rea_number FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id JOIN (SELECT kauplus_id, ROUND(SUM(toode_summa),0) AS kaive FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id WHERE ostud.kauplus_id IN (202304, 200001, 200840, 202516, 200631) GROUP BY ostud.kauplus_id) AS t1 ON t1.kauplus_id = ostud.kauplus_id WHERE ostud.kauplus_id IN (202304, 200001, 200840, 202516, 200631) GROUP BY ostud.kauplus_id, tooted.kategooria_nimi, t1.kaive ORDER BY kauplus_id, protsent desc) AS t WHERE t.rea_number <= 5;

Päring	PostgreSQL	MariaDB Columnstore
P3	SELECT t.kuu, t.kauplus_id, ROUND(AVG(ost_sum),2) AS avg_sum, ROUND(AVG(ost_arv),0) AS avg_arv FROM (SELECT EXTRACT(month FROM cast(left(cast(kalender_id AS varchar),6) AS date)) AS kuu, COUNT(toode_id) AS ost_arv, SUM(toode_summa) AS ost_sum, kauplus_id FROM ostud GROUP BY ost_id, kauplus_id, EXTRACT(month FROM cast(left(cast(kalender_id AS varchar),6) AS date)) HAVING EXTRACT(month FROM cast(left(cast(kalender_id AS varchar),6) AS date)) = 4) AS t GROUP BY t.kuu, t.kauplus_id;	SELECT t.kuu, t.kauplus_id, ROUND(AVG(ost_sum),2) AS avg_sum, ROUND(AVG(ost_arv),0) AS avg_arv FROM (SELECT month(date(left(kalender_id,6)))) AS kuu, COUNT(toode_id) AS ost_arv, SUM(toode_summa) AS ost_sum, kauplus_id FROM ostud GROUP BY ost_id, kauplus_id, kuu HAVING kuu = 4) AS t GROUP BY t.kuu, t.kauplus_id;
P4	SELECT kauplus_id, SUM(toode_summa) AS ost_sum, COUNT(DISTINCT ost_id) AS ost_arv FROM ostud WHERE EXTRACT(QUARTER FROM cast(left(cast(kalender_id AS varchar),6) AS date)) = 1 GROUP BY kauplus_id;	SELECT kauplus_id, SUM(toode_summa) AS ost_sum, COUNT(DISTINCT ost_id) AS ost_arv FROM ostud WHERE quarter(date(left(kalender_id,6))) = 1 GROUP BY kauplus_id;
P5	SELECT EXTRACT(week FROM cast(left(cast(kalender_id AS varchar),6) AS date)) AS nadal, SUM(toode_summa) AS ost_sum, COUNT(DISTINCT ost_id) AS ost_arv FROM ostud WHERE kauplus_id = 202304 GROUP BY nadal ORDER BY nadal ASC;	SELECT week(date(left(kalender_id,6))) AS nadal, SUM(toode_summa) AS ost_sum, COUNT(DISTINCT ost_id) AS ost_arv FROM ostud WHERE kauplus_id = 202304 GROUP BY nadal ORDER BY nadal ASC;

Päring	PostgreSQL	MariaDB Columnstore
P6	SELECT EXTRACT(month FROM cast(left(cast(kalender_id AS varchar),6) AS date)) AS kuu, SUM(toode_summa) AS ost_sum, COUNT(DISTINCT ost_id) AS ost_arv FROM ostud WHERE kauplus_id = 202304 GROUP BY kuu ORDER BY kuu ASC;	SELECT month(date(left(kalender_id,6))) AS kuu, SUM(toode_summa) AS ost_sum, COUNT(DISTINCT ost_id) AS ost_arv FROM ostud WHERE kauplus_id = 202304 GROUP BY kuu ORDER BY kuu ASC;
P7	SELECT EXTRACT(isodow FROM cast(left(cast(kalender_id AS varchar),6) AS date)) AS paev_nr, right(cast(kalender_id as varchar),2) AS tund, SUM(toode_summa) AS ost_sum, COUNT(DISTINCT ost_id) AS ost_arv FROM ostud WHERE kauplus_id = 202304 GROUP BY paev_nr, tund ORDER BY paev_nr ASC;	SELECT weekday(date(left(kalender_id,6))) AS paev_nr, right(kalender_id,2) AS tund, SUM(toode_summa) AS ost_sum, COUNT(DISTINCT ost_id) AS ost_arv FROM ostud WHERE kauplus_id = 202304 GROUP BY paev_nr, tund ORDER BY paev_nr ASC;
P8	SELECT cast(left(cast(kalender_id as varchar),6) as date) as kuupaev, toode_nimi, SUM(toode_kogus) as arv FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id WHERE kauplus_id = 202304 AND tooted.kategooria_nimi IN ('1PG01 Leib ja sai (LeSa)') GROUP BY kuupaev, tooted.kategooria_nimi, ostud.toode_nimi HAVING cast(left(cast(kalender_id AS varchar),6) AS date) BETWEEN '2019-03-04' AND '2019-03-10';	SELECT date(left(kalender_id,6)) AS kuupaev, toode_nimi, SUM(toode_kogus) AS arv FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id WHERE kauplus_id = 202304 AND tooted.kategooria_nimi IN ('1PG01 Leib ja sai (LeSa)') GROUP BY kuupaev, tooted.kategooria_nimi, ostud.toode_nimi HAVING kuupaev BETWEEN '2019-03-04' AND '2019-03-10';

Päring	PostgreSQL	MariaDB Columnstore
P9	SELECT kauplus_id, EXTRACT(week FROM cast(left(cast(kalender_id AS varchar), 6) AS date)) AS nadal, cast(left(cast(kalender_id AS varchar),6) AS date) AS kuupaev, SUM(toode_kogus) AS arv, ROUND(SUM(toode_summa),2) AS ost_sum FROM ostud WHERE kauplus_id IN (201826, 200635, 200840, 200327, 202808) AND toode_id = 4740113092139 GROUP BY nadal, kuupaev, ostud.kauplus_id HAVING EXTRACT(week FROM cast(left(cast(kalender_id AS varchar),6) AS date)) = 32 ORDER BY kauplus_id, kuupaev ASC;	SELECT kauplus_id, week(date(left(kalender_id,6))) AS nadal, date(left(kalender_id,6)) AS kuupaev, SUM(toode_kogus) AS arv, ROUND(SUM(toode_summa),2) AS ost_sum FROM ostud WHERE kauplus_id IN (201826, 200635, 200840, 200327, 202808) AND toode_id = 4740113092139 GROUP BY nadal, kuupaev, ostud.kauplus_id HAVING nadal = 32 ORDER BY kauplus_id, kuupaev ASC;
P10	SELECT DISTINCT ost_id, kategooria_id FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id JOIN kategooriad ON kategooriad.kategooria_nimi = tooted.kategooria_nimi WHERE kauplus_id IN (202304, 200001, 200840, 202516, 200631) ORDER BY ost_id DESC;	SELECT DISTINCT ost_id, kategooria_id FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id JOIN kategooriad ON kategooriad.kategooria_nimi = tooted.kategooria_nimi WHERE kauplus_id IN (202304, 200001, 200840, 202516, 200631) ORDER BY ost_id DESC;
P11	SELECT kategooria_nimi, COUNT(tooted.toode_id) AS toode_arv FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id WHERE kauplus_id = 202304 GROUP BY kategooria_nimi;	SELECT kategooria_nimi, COUNT(tooted.toode_id) AS toode_arv FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id WHERE kauplus_id = 202304 GROUP BY kategooria_nimi;

Päring	PostgreSQL	MariaDB Columnstore
P12	SELECT DISTINCT toode_nimi, kategooria_nimi FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id WHERE kauplus_id = 202304 AND toode_nimi LIKE '%Leibur%';	SELECT DISTINCT toode_nimi, kategooria_nimi FROM ostud JOIN tooted ON ostud.toode_id = tooted.toode_id WHERE kauplus_id = 202304 AND toode_nimi LIKE '%Leibur%';

4.5 Eksperimendi tulemused

Eksperimendis vaadeldud andmepäringute töökiirused on välja toodud alljärgnevas tabelis. Päringute töökiirused on mõõdetud DBeaver andmehaldus töövahendiga, kus on määratud väljundi ridade piirarvuks 200 ning käsureal täitmisplaanide leidmisel. Eksperimendis ei ole võrreldud andmete lisamise kiirust andmebaasidesse, sest need on lisatud kasutades erinevaid meetodeid. Täitmisplaanide nägemiseks on MariaDB Columnstore andmebaasisüsteemis käsureal käivitatud käske *select calSetTrace(1);* ja *select calGetTrace();* [25]. PostgreSQL puhul käivitatud päringute täitmisplaanide nägemiseks on käsureal kasutatud käsku *EXPLAIN ANALYZE* [18]. Võrreldud on serveripoolset päringu töökiirust ning ka kasutajapoolset esimese 200 rea näitamiseks. Tabelis 9 on välja toodud eksperimendis osalenud andmepäringute keskmised töökiirused 5 käivitamise kohta. Töös ei ole välja toodud päringute tulemusi, sest väljundite ridade arvud varieeruvad paarikümnest reast mitme saja tuhandeni.

Tabel 9. Eksperimendi tulemused.

Päring	**PostgreSQL käsureal (sek)**	**MariaDB Columnstore käsureal (sek)**	**PostgreSQL DBeaver, esimesed 200 rida (sek)**	**MariaDB Columnstore DBeaver, esimesed 200 rida (sek)**
P1	1722,2	16,1	156,3	19,4
P2	658,2	24,4	2466,0	185,5
P3	841,2	61,0	335,0	76,2
P4	2560,3	57,4	360,3	73,2
P5	173,6	7,8	93,0	10,1
P6	173,6	7,8	94,3	10,3
P7	172,5	8,2	97,6	10,8
P8	119,8	9,2	91,6	12,1
P9	116,3	6,9	96,4	8,9
P10	420,4	24,9	120,0	26,4
P11	147,5	7,4	91,2	10,3
P12	114,0	49,4	132,6	66,1

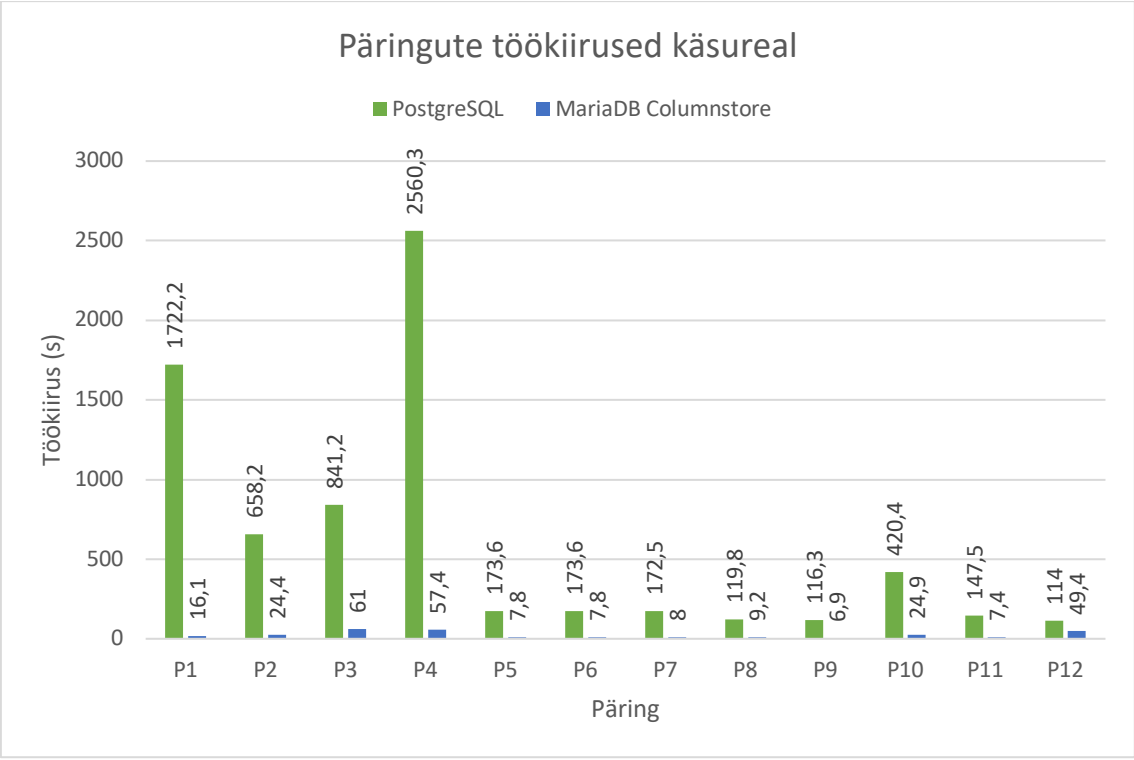
Tabelis 9 on ülevaatlikult toodud 12 päringu töökiirused sekundites kahes erinevas andmebaasis kahe erineva meetodi kasutamisel.

5 Tulemuste analüüs

Peatükis analüüsitakse lisaks eksperimendi tulemustele ka andmebaasidega töötamist ja kasutajakogemust. Andmete lisamist, kustutamist ja uuendamist ei ole detailselt võrreldud, kuid andmebaaside loomise käigus andmete importimise ligikaudne aeg on välja toodud koos tekkinud kitsaskohtadega.

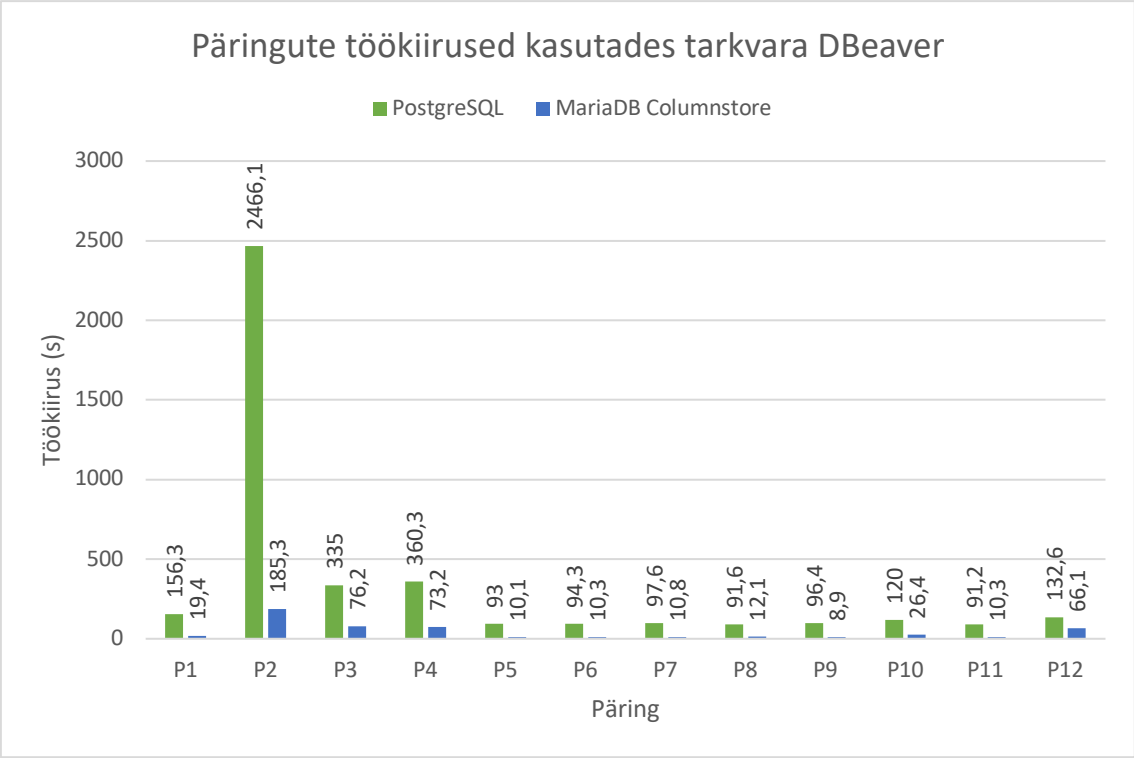
5.1 Eksperimendi tulemuste analüüs

Eksperimendis võrreldi 12 päringu töökiirust kahes erinevas andmebaasisüsteemis ning kasutades kahte erinevat meetodit. Joonis 7 illustreerib käsureal käivitatud päringute keskmised töökiirused, mille puhul on väljundis ka täitmisplaan. Täitmisplaanide võrdlemine põhjendab, miks üks süsteem on kiirem kui teine. Joonisel 8 on toodud esimeste tulemuste töökiirused kui väljundridade arv on piiratud.



Joonis 7. Päringute töökiirused käsureal.

Joonisel 7 on näha, et PostgreSQL on mitmeid kordi aeglasem kui MariaDB Columnstore.



Joonis 8. Päringute töökiirused andmehaldusvahendis DBeaver.

Joonis 8 illustreerib MariaDB ja PostgreSQL andmebaasisüsteemide töökiiruseid päringute käivitamisel kasutades DBeaver andmehaldusvahendit, kus ridade arv on piiratud 200-ga. Taaskord on näha, et päringute töökiirused PostgreSQL andmebaasisüsteemis on aeglasemad kui MariaDB Columnstore andmebaasimootoris.

Eksperimendis osutusid kõigi 12 päringu töökiirused paremaks veerupõhises andmebaasisüsteemis. Käsureal oli veerupõhises süsteemis kõige aeglasem päring P3 ja reapõhises osutus kõige aeglasemaks päring P4. Nii käsureal kui ka andmehaldusvahendis oli Columnstore andmebaasimootoris kõige kiirem päring P9. PostgreSQL puhul osutus käsurea kiireim P12 DBeaver keskkonnas aeglasemaks kui pooled käivitatud päringutest. Parima tulemuse saavutasid DBeaver keskkonnas P8 ja P11, mille tööaeg oli ligikaudu 91 sekundit. Kõige aeglasemaks päringuks jäi andmehaldusvahendis päring P2, mida andmehaldusvahendis ei piiratud 5 kaupluse tingimusega.

Järgnevalt uuritakse päringute täitmisplaane selgitamaks päringute töökiiruste erinevusi. Välja tuleb tuua asjaolu, et PostgreSQL täitmisplaanid on detailsemad, sest Columnstore andmebaasimootoris on mitmed tegevused ühe operatsiooni sisse

koondatud ja ei jälgita mysqld töötlusaega. Töökiiruseid võrreldakse kordades ning täitmisplaanis kirjeldatud operatsioone protsendina kogu tööajast.

Esimese päringu töökiiruste vahe on 100-kordne PostgreSQL kahjuks. MariaDB Columnstore andmebaasisüsteemis P1 päringu täitmise esimeses sammus - BPS (*Batch Primitive Step*) päritakse andmeid tabeli Ostud kahest veerust (kauplus_id, toode_summa) ning kõikidest ridadest. Järgmisena tehakse TAS (*Tuple Aggregation Step*) sõlmes *GROUP BY* funktsioon ning viimaks sorteeritakse väljund kasutaja jaoks sõlmes TNS (*Tuple Annexation Step*). PostgreSQL täitmisplaan on detailsem, sest välja on toodud isegi sorteerimismeetod, milleks on *quicksort*. Päringu P1 ajaerinevused tulevad välja kahes esimeses sammus. Kui PostgreSQL kulutab tabeli skaneerimiseks suurusjärk 800 sekundit, siis Columnstore andmebaasimootor suudab teha seda 16 sekundiga. Järgmiseks sammuks on grupeerimine, mille suurusjärk on PostgreSQL puhul sarnane eelneva sammuga, vastavalt 800 sekundit, kuid MariaDB puhul ainult 200 millisekundit. Kõikidele järgnevatele sammudele enne tulemi väljastamist läheb PostgreSQL-is ligikaudu 100 sekundit, kuid MariaDB-s on see enamasti paar millisekundit. Kuna päringus ei ole *WHERE* tingimus lauseid ja lugeda tuleb kõiki ridu, siis on selge veerupõhise andmebaasi paremus, mis vajab ainult kahte veergu tulemuse väljastamiseks vastupidiselt PostgreSQL-le, mis sisuliselt loeb terve andmebaasi.

Päring P2 on käivitatud käsureal lisatingimusega, kus võetakse andmed ainult 5 kaupluse kohta. DBeaver keskkonnas ei ole lisatingimust lisatud. See on ainus päring, kus töödeldakse aknafunktsiooni. Mõlemas andmebaasis kulub *PARTITION OVER* funktsioonile 20 millisekundit. Reapõhisel moodustab alampäringu täitmine 1/4 kogu ajast ning veerupõhisel 1/3 kogu ajast. PostgreSQL-is on tehakse mitmeid lisa sorteerimisoperatsioone enne grupeerimisi. Samuti kulub PostgreSQL-is rohkem aega funktsioonile *GROUP BY*.

Päringu P3 töökiiruste vahe on väiksem, sest vaadeldavate ridade arv on piiratud tingimuses ühe kuuga. P3 päring on veerupõhises keskkonnas kõige aeglasem kui DBeaver keskkonnas oleks samuti P2 päringule lisatud tingimus. Selle võimalik põhjus on *WHERE* tingimuse asemel *HAVING* operatsiooni kasutamine. MariaDB puhul võib täheldada, et pool päringu täitmisest kuluvat aega läheb andmete skaneerimisele, samal ajal kui PostgreSQL-is moodustab sama operatsioon 1/4 päringu töötlemise ajast, sest tulemusi filtreeritakse *HAVING* tingimuse järgi. PostgreSQL oskab tulemusi taas

kasutada ja see on üks põhjus, miks andmete lugemine on muutunud kiiremaks. PostgreSQL kulub ligikaudu 85% tööajast alampäringu töötlemisele. Columnstore andmebaasimootoris peaks alampäringu samm olema esimeses TAS operatsioonis, mis moodustab 100 millisekundit kogu tööajast.

Päringul P4 kulub veerupõhisel süsteemil üle 90% tööajast esimesele sammule BPS. PostgreSQL andmebaasisüsteemis on tabelist ridade leidmine kõige väiksema ajakuluga võrreldes teiste päringus läbiviidavate operatsioonidega, sest kasutusel on tingimus ning PostgreSQL saab osa andmeid vahemälust.

P5 ja P6 päringute ülesehitus on sarnane, mida on näha ka täitmisplaanidest. Erinevused tulevad sisse lõplikus väljundi ridade ja numbrite arvus, sest vaadeldakse erinevaid perioode. P5 päringus võetakse nädalad ning P6 päringus kuud. Reapõhises andmebaasisüsteemis on mõlemad 25 korda aeglasemad kui veerupõhises. Tingimuses on täpsustatud üks kaupluse kood, mis annab sarnase aja kulu ühes andmebaasisüsteemis mõlemas päringus. PostgreSQL lõpptulemuse sorteerimiseks kulub keskmiselt umbes 15% ajast, see vastu MariaDB-s on sorteerimisaeg paar millisekundit.

Päringus P7 võetakse kalender_id väärtusest päeva number ja tund ning tulemus sorteeritakse nende alusel. Kuna PostgreSQL-is kulub sorteerimisele ligikaudu 50 sekundit, siis selle ära jätmine mõjutaks tunduvalt päringu töökiirust. Sama ei kehti Columstore andmebaasimootori kohta, sest selles kulub sorteerimisele minimaalselt aega.

Päringus P8 on kitsendatud väljundit kahe *WHERE* tingimuse ja ühe *HAVING* tingimusega. PostgreSQL töötleb päringut taaskord paralleelselt ning vahetulemus sorditakse enne uueks vahetabeliks ühendamist. Selle päringu puhul tuleb veerupõhisesse täitmisplaani uus sõlm DSS (*Dictionary Structure Step*), kus otsitakse andmeid tingimuses toodud kategooria nime järgi, mis on salvestatud sõnastikuna. Kategooria nime piirang aeglustab päringut mõlemas andmebaasis.

P9 on MariaDB Columnstore andmebaasisüsteemis kõige kiirem ning PostgreSQL käsureal teisel kohal. Olenemata asjaolust, et tingimuses on toodud mitu kauplust, sarnaneb veerupõhise süsteemi aeg pigem päringutele, kus vaadeldakse ühte kauplust.

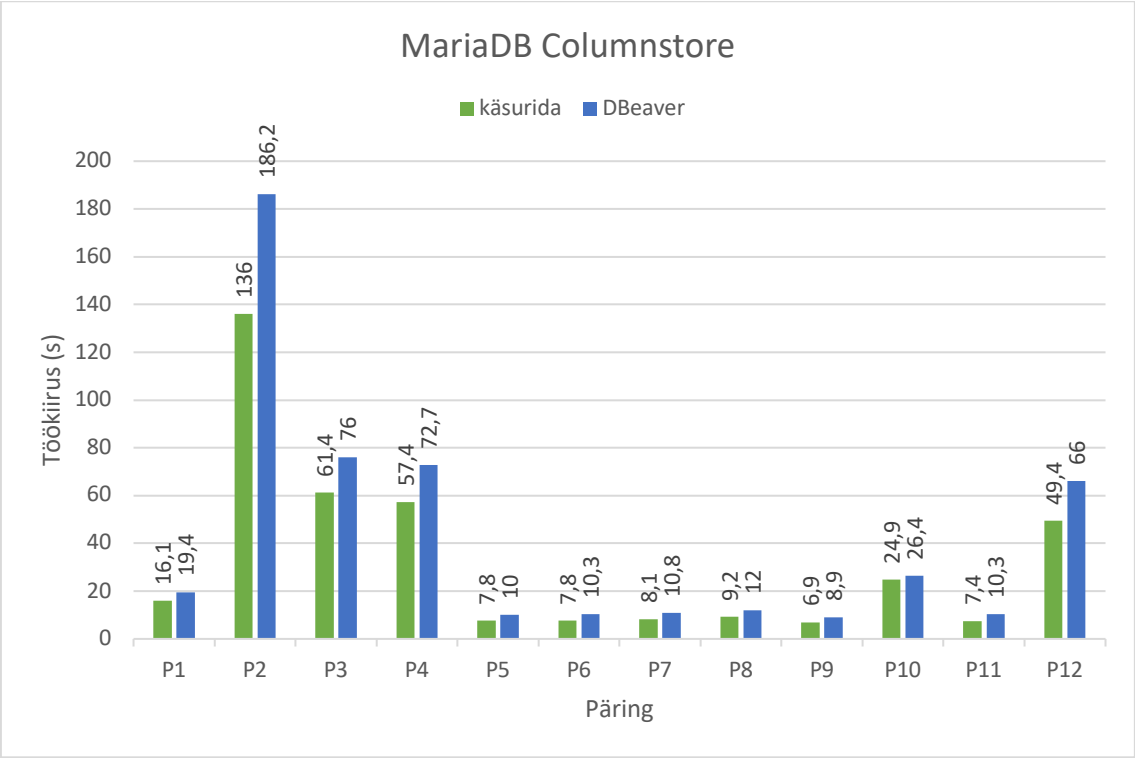
Selle põhjuseks on lisatingimus, et otsitakse ainult ühte toodet, mis suurendab vaadeldavate veergude arvu ühe võrra.

Antud päring P10 on mõlemas süsteemis keskmise kiirusega. Ainus päring, kus tehakse kaks ühendamisoperatsiooni, mis suurendab tööaega, sest üks ühendamine tehakse kategooria nime alusel. MariaDB täitmisplaanis pole välja toodud kui kaua võtavad ühendamised aega, kuid samas võib täheldada, et kuna väljundi ridade arv on suur, siis isegi selles süsteemis läheb ligikaudu 50% ajast väljundi töötlemisele. PostgreSQL puhul kulub 1/2 tööajast skaneerimisele ja ühendamisele, 1/4 sorteerimisele ning 1/4 unikaalsete ridade leidmisele.

Veerupõhises Columnstore andmebaasimootoris algab P11 päring vajalike veeru plokkide leidmisest Ostud tabelist. Reapõhises PostgreSQL-is võetakse enne andmed tabelist Tooted. Mõlemas andmebaasisüsteemis toimub järgmisena tabelite ühendamine. PostgreSQL täitmisplaanis on järgmiseks sammuks osaline grupeerimine, millele järgneb andmete sorteerimine *quicksort* meetodil, mida päringus ei ole välja toodud. Väljundis on üks veerg tekstiline ning sellest tingitult on grupeerimine ja andmete väljastamine ka MariaDB-s ajakulukamad kui eelnevates päringutes.

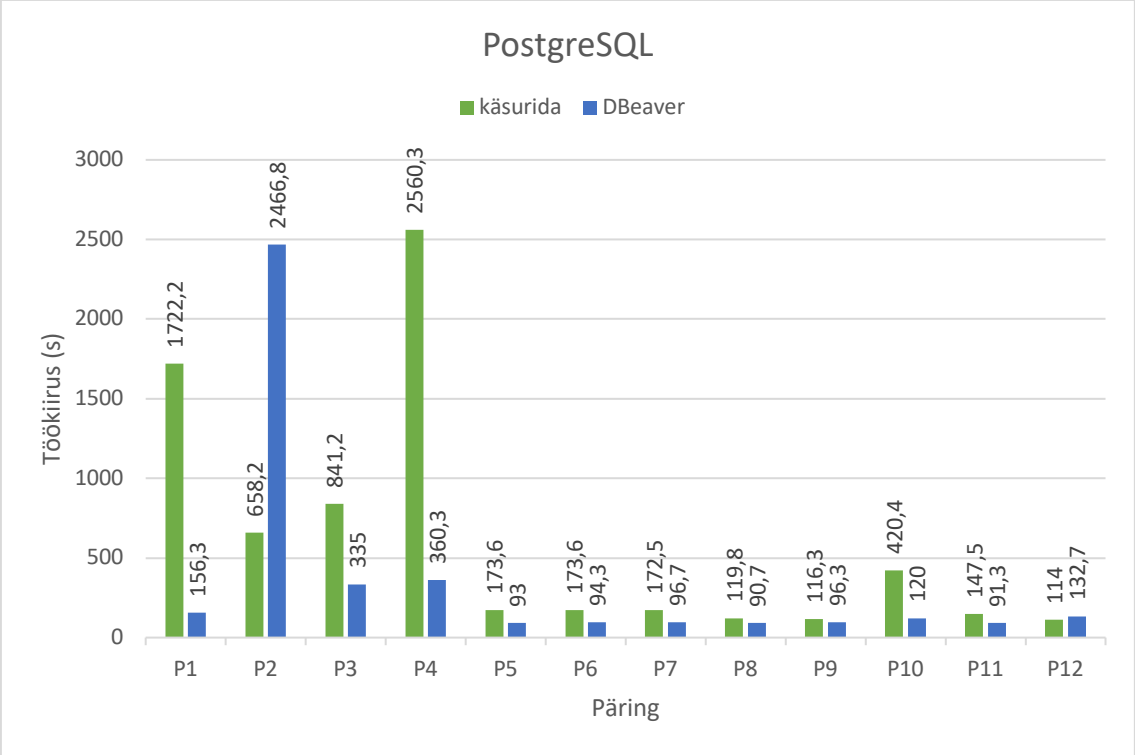
Päring P12 on PostgreSQL andmebaasisüsteemis kõige kiirem agregatsioon, kuid MariaDB andmebaasisüsteemis üks aeglasemaid. Olenemata asjaolust, et andmeid küsitakse ühe kaupluse kohta ja päringu kiirus võiks sarnaneda päringutele P5, P6, P7, P8 ja P11, kus ajakulu on alla 10 sekundi, on kiirus tingitud teisest tingimusest, mille alusel peab filtreerima andmed veerust, kus andmetüübiks on string. Columnstore andmebaasimootori jaoks tähendab see lisaoperatsiooni ja otsida tuleb andmetest, mis on sõnastiku kujul. Lisaks toimub P12 päringus ühendamisoperatsioon, mis võiks päringu aeglasemaks muuta, kuid sama ühendamine toimub ka päringus P11, mis on endiselt alla 10 sekundi. Kuna päring on PostgreSQL-is kõige kiirem, siis andmebaasisüsteem tuleb väga hästi toime tekstiliste andmetega.

Lisaks võrreldi samas andmebaasisüsteemis käivitatud päringute tagastatud töökiiruseid käsurealt ning andmehaldusvahendist. Joonisel 9 on toodud MariaDB Columnstore töökiirused ja joonisel 10 PostgreSQL tulemused.



Joonis 9. MariaDB Columnstore töökiirused.

Joonisel 9 on visualiseeritud päringute töökiirused MariaDB Columnstore andmebaasisüsteemis. Graafikult on selgelt näha, et andmehaldusvahendis DBeaver käivitatud päringud on kõik mitu sekundit aeglasemad kui käsureal käivitatud päringud.



Joonis 10. PostgreSQL töökiirused.

Joonis 10 illustreerib PostgreSQL andmebaasisüsteemi poolt tagastatud töökiirused käsureal ja kasutades tööriista DBeaver. Kui välja arvata teine (P2) ja viimane päring (P12), siis on kõik võrreldud päringud kiiremad andmehaldusvahendis. P2 päringu käivitamisel käsureal arvestati ainult 5 kauplusega, kuid DBeaver keskkonnas käivitati päring ilma piiranguteta, mis on vastuolu põhjuseks.

5.2 Eksperimendi tulemuste analüüsi järeldused

Eksperimendis, kus andmemahuks on ligikaudu 40 GB ja võrreldakse päringute töökiiruseid, on selgelt parem veerupõhise salvestusmudeliga MariaDB Columnstore edastades reapõhist PostgreSQL-i mõnel juhul isegi 100-kordselt. Veerupõhise andmebaasi päringud on kiiremad käsureal, kus reapõhise salvestamismudeliga andmebaasisüsteemi päringud on vastupidiselt aeglasemad. Täitmisplaanidelt võib näha, et enamasti kulub veerupõhises andmebaasis kõige rohkem aega andmete lugemiseks, mis on siiski kordades kiirem, sest loetakse ainult päringu täitmiseks vajalikke veerge. Reapõhises andmebaasis lugemine järk-järgult kiireneb, sest PostgreSQL loeb mõningaid andmeid ja indekseid vahemälust [18]. PostgreSQL täitmisplaanidelt loeb välja, et sellise andmemahuga töötleb ka reapõhine salvestusmudel operatsioone enne väljastamist paralleelselt. Kuigi eksperimendis defineeritud päringud on aja, kaupluste või kategooriate hulgaga piiratud ning see ei optimeeri veerupõhises andmebaasis loetavaid veerge maksimaalselt, on efektiivsem siiski veerupõhine MariaDB Columnstore.

Võrreldes eksperimendi tulemusi näiteks 2016.aastal kaitstud magistritöö tulemustega [30], siis töökiiruste erinevused andmebaasisüsteemide vahel antud eksperimendis on tunduvalt suuremad, kui paar sekundit. Võrreldavas töös on teised andmebaasisüsteemid ning väiksem andmehulk (suurimas tabelis 60 miljonit rida), mis on arvuti poolt genereeritud. Antud töös on eksperiment viidud läbi reaalsetel ettevõtte andmetel, mis annab täpsema ülevaate erinevustest jaekaubanduse valdkonnas, aga ka suurema kindluse, et päringute töökiirustel siiski on märgatav erinevus ning süsteemi muutmine on põhjendatud. Lisaks andmepäringute töökiirustele tuleks muidugi arvestada ka teiste aspektidega nagu näiteks ühilduvus olemasolevate süsteemidega.

5.3 Andmete lisamise võrdlus

MariaDB Columnstore andmebaasisüsteem on disainitud toetama täielikult ACID-omadustega *(Atomicity, Consistency, Isolation, Durability)* multiversioon konkurentjuhtimist, mis võimaldab samaaegselt käivitada päringuid ja muuta andmeid. Isegi kui SQL andmete manipulatsioonikeel, DML *(Data Manipulation Language)*, on realiseeritud, soovitatakse kiiremaks andmete lisamiseks kasutada cpimport utiliiti, mis võimaldab sisestada andmeid suurel hulgal. Parim jõudlus saavutatakse kui lisatavad andmed on aja järgi sorteeritud ja päringus kasutatakse ajalist tingimust, mis aitaksid elimineerida mittevajalikud ulatused [25].

Kuigi kõik kirjeldatud OLAP päringud olid kordades kiiremad veerupõhisel salvestamisel, võib kirjandusest lugeda, et andmete töötlemisel DML keelega esineb veerupõhisel salvestamisel puudujääke [31]. Tõesti ka veerupõhisesse andmebaasi andmete lisamisel oli esimeseks plaaniks importida need sarnaselt reapõhise andmebaasiga, kuid üsna pea sai selgeks, et lähenemine pole otstarbekas, sest ühe rea lisamine käsureal võttis ligikaudu 10 sekundit. PostgreSQL andmebaasisüsteemis võttis ühe osturea lisamine 5 millisekundit. Sellisel lähenemisel on PostgreSQL kindlalt eelisseisus, kuid kitsaskohast ülesaamiseks on mõlemas andmebaasisüsteemis realiseeritud hulga kaupa andmete import. Sel viisil toimetades sai andmed veerupõhisesse andmebaasi imporditud ligikaudu 5 minutiga.

5.4 Andmebaasidega töötamine

PostgreSQL on populaarne andmebaasisüsteem, mida arendatakse endiselt edasi. Kasutusmugavuse poolest oli parem PostgreSQL. Tuntud andmebaasina on PostgreSQL esindatud paljudes andmehaldusvahendites, näiteks antud töös kasutatavas DBeaver tarkvaras. Kuigi MariaDB on põhineb MySQL-il ja on DBeaver keskkonnas olemas, on Columnstore andmebaasimootori kasutamisel eripärad. DBeaver andmehaldusvahend loob küll muutmispäringu, kuid selle käivitamiseks on vajalikud modifikatsioonid.

Mitmete kitsaskohtade lahendamiseks on kasutusele võetud efektiivne JIRA süsteem, kus saab luua pileti probleemiga, mis ei eelda kasutajatoega ühenduse võtmist. Paljude probleemide lahenduseks pakutakse välja alternatiive, kuid on ka selliseid, mis on parandatud uues versioonis. Näiteks andmebaasimootori muutmine veerupõhiseks kui

töötav andmebaas on loodud kasutades reapõhist andmebaasimootorit. Selle ainus lahendus oli uue andmebaasi loomine veerupõhiselt, mis tähendas taaskord andmete importimist CSV failidest. Üheks probleemiks olid veel vaikimisi seadistuste piirangud. Näiteks tervet andmemahtu kaasavad päringud, mis sisaldasid *COUNT(DISTINCT)* funktsiooni, ületasid mälu, mille lahendamiseks oleks pidanud muutma vaikimisi seadistusi.

6 Rakenduse näide

Käesolev peatükk ei ole magistritöö peamise panuse osa, kuid saamaks paremat ettekujutust antud töö ärilisest kontekstist, siis on see lisatud demonstreerimaks eksperimendis kasutatud päringute kasu. Taoliste rakenduste regulaarsel läbiviimisel on kriitiline saada sisendandmete kättesaamine võimalikult kiireks, sest sellel on oluline osa tulemuslikus andmeanalüüsis.

Eksperimendis kasutatud päringutel on mitmeid kasutusalasid jaekaubanduse analüütikas. Kaasajal on olulisemaks muutunud klientide profileerimine, mille peamiseks eesmärgiks on personaalsete pakkumuste tegemine. Selline profileerimine ei ole ainult ostuandmete pealt mõistlik, sest arvestama peaks ka kliendikaardi olemasolu, mis on oluliseks parameetriks. Töös valiti rakendamise näiteks kaupluste profileerimine arvestades toodete suuremaid kategooriaid.

Antud töös kasutati andmeanalüüsiks Jupyter Notebook veebirakendust, kus on võimalik luua dokument, mis sisaldab programmikoodi koos jutustavate tekstilõikude ja graafikutega. Veebirakendus on avatud lähtekoodiga ning andmeteaduse valdkonnas väga populaarne tööriist võimaldades erinevaid kasutusvaldkondi, nagu näiteks masinõpe ja andmete visualiseerimine. Jupyter Notebook ühildub mõlema eksperimendis kasutatava andmebaasisüsteemiga. Programmikoodi kirjutamisel on kasutatud programmeerimiskeelt Python ning järgmisi teeke:

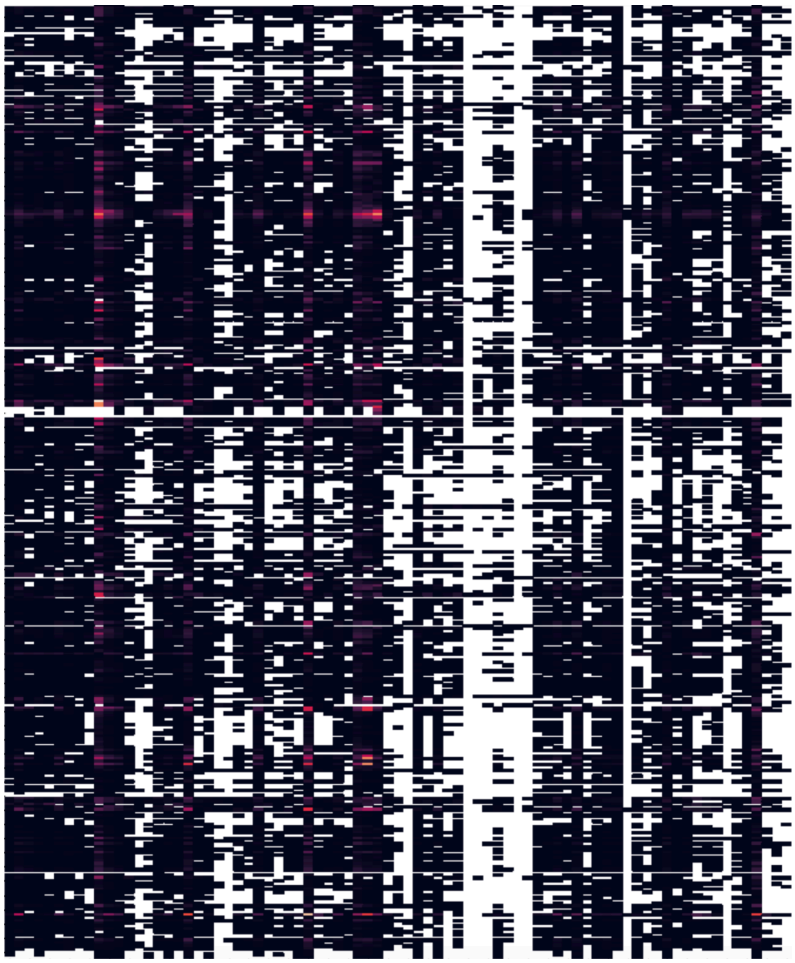
- pandas - teek tabeli kujul andmete töötlemiseks (Dataframe);
- sklearn - masinõppe tehnikad (klasterdamine, dimensionaalsuse vähendamine);
- seaborn/matplotlib - andmete visualiseerimine;
- numpy - teaduslikud arvutused N-dimensioonilistel andmetel.

6.1 Kaupluste profileerimine

Kaupluste klasterdamine on protsess, kus samas klastris olevate kaupluste võrreldavad parameetrid on sarnased. Kauplusi võib klasterdada kaupluse suuruse järgi, kus arvestatakse kaupluse ja lao üldpinda, riiulite kogust ning unikaalsete toodete arvu.

Teiseks meetodiks on kaupluste grupeerimine klientide profiili alusel, kus sortimendi planeerimises arvestatakse selliste mõõdikutega nagu kaupluse asukoht ning piirkonna elatustase. Veel võib kauplusi segmentideks jagada kategooriapõhiselt. Sellisel juhul vaadatakse iga kategooriat eraldi ning ühte klastrisse satuvad kauplused, millel on samas suurusjärgus käive või ostude arv. Samuti võib kõiki eelnevaid meetodeid koos vaadata ning klasterdada kauplusi multidimensionaalselt [35].

Andmetel viidi läbi esmane statistiline analüüs, mille eesmärgiks oli leida kaupluste erisusi käibe ja kategooriate lõikes. Kategooriate visualiseerimiseks kasutati soojakaarti, mis toetas edasist profileerimist. Soojakaart on suurepärane visualiseerimislahendus suurandmetega töötamisel. Töös on valitud vaikimisi seadistustega graafik, millel skaala suuremad väärtused kuvatakse tumedamates toonides ning väiksemad heledates. Visualiseerimine annab parema ettekujutuse andmetest ning andmevahemikest kui tavapärane tabel. Joonisel 11 on ühe soojakaardi pilt, mille järgi otsustati, et edasine profileerimine on mõistlik.



Joonis 11. Soojakaart kategooriate käivetega igas kaupluses.

Joonisel 11 on horisontaalteljel kategooriad ning vertikaalteljel kauplused. Üks ühikruut tähistab kategooria olemasolu kaupluses. Mida tumedam on ruut, seda suurem on kaupluse käive kategoorias. Valge ruut esitab kategooria puudumist kauplusest.

Käesolevas töös on kasutatud andmete klasterdamisel siluetimeetodit, mis realiseerib k-keskmiste klasterdamisalgoritmi [36]. Lisaks siluetimeetodile oli valikus tuntuim küünarnukimeetod. Siluetimeetod osutus valituks, sest numbrilised väärtused on lihtsamini mõistetavamad kui küünarnuki kujutis graafikul, millelt kõige optimaalsema klastrite arvu lugemine võib osutuda mõnevõrra keeruliseks kui naaberklastrite erinevus on väga väike. Siluetimeetod mõõdab punkti sarnasust enda klastriga arvestades sarnasust teiste klastritega. Tabelis 10 on töös kasutatud siluetimeetodi tulemuste numbriline väljund.

Tabel 10. Klastrite keskmised hinnangud.

Klastrite arv	**Siluetikordaja**
2	0.5357153819775969
3	0.4428005054463312
4	0.46203269412102643
5	0.4785473460177631
6	0.48358516782443145
7	0.46801548357275385
8	0.4554720752520366
9	0.47002830272517865

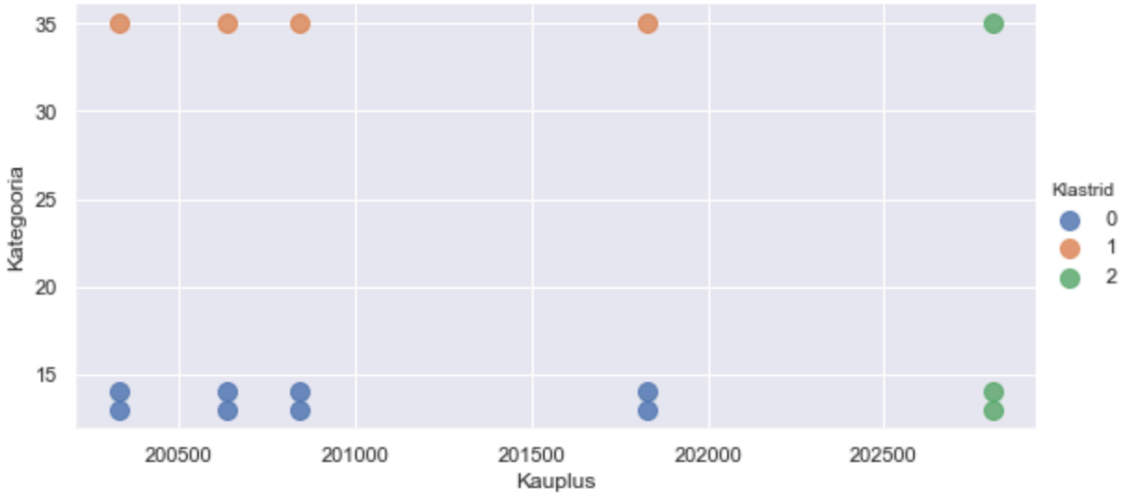
Tabelist 10 on näha, et parimaks klastrite arvuks osutub 6. Siluetikordaja arvutamisel arvestatakse kahe komponendiga: elemendi kaugus enda klastri keskpunktini ja elemendi kaugus lähima mitte-oma klastri keskpunktini. Parimal klastrite arvul on kõikide elementide kauguste summa tsentrini nullilähedane. Siluetikordaja skaala on -1 kuni 1, kus negatiivne tulemus tähendab klastri elementide omavahelist suuremat kaugust kui klastrite vahel [37].

Klastritesse jaotumist on parem vaadata graafikult kui tabelist ning joonis 12 illustreeribki klasterdatud kategooriaid igas kaupluses.

Joonis 12. Kaupluste kategooriate protsentide klasterdamine k-keskmiste meetodil.

Joonisel 12 on klasterdatud kauplused kategooria käibe protsendi alusel. Jooniselt saab eristada kauplused, mis on kategooriate jagunemise poolest sarnased. Ühte klastrisse võivad kuuluda erinevad kauplused ja erinevad kategooriad, sest klasterdatud on kategooriate käivete protsente kogu kaupluse käibest.

Ülal toodud näide on ülevaatlik ning detailsemaks kaupluste profileerimiseks võiks vaadata ühte või paari tooteliiki võrreldavates kauplustes. Joonisel 13 on toodud viie kaupluse klasterdamine kolmes kategoorias.



Joonis 13. 5 kaupluse 3 kategooria klasterdamine.

Joonisel 13 on näha detailsem kaupluste klasterdamine kolmes kategoorias. Parimaks klastrite arvuks osutus kolm, sest kategooriate protsendid on väga sarnased. Kategoorias 35 on nelja kaupluse protsentuaalne müük klastris 1 ning üks kauplus klastris 2. Kuna viimane kauplus kuulub teise klastrisse, siis selle toote valik peaks olema erinev, kuid

esimesed neli kauplust see vastu on ka kahes teises kategoorias samades klastrites. Sellest järeldus, et esimese 4 kaupluse profiil on sarnane ja sortimendi valik neis kategooriates võiks olla sarnane.

Selline profileerimine võimaldab pakkuda personaalsemat ostukogemust klientidele, mis on üks peamistest uurimissuundadest jaekaubanduse valdkonnas. Suurte jaekaubanduskettide puhul on kaupluste individuaalne vaatamine ajamahukas ja üldiselt suur väljakutse, kuid grupeerides kauplused klastritesse muutub see hoomatavamaks. Selline grupeerimine aitab ettevõtetel optimeerida sortimenti, mille tulemusena võiks kulud kahaneda, tulud kasvada ning ostukogemus paremaks muutuda. Ainult ostuandmeid arvestades on selline klasterdamine kõige mõistlikum, sest eelis seisneb mudeli lihtsuses. Toodud rakenduse seisukohalt on andmepäringute töökiirus oluline, sest edasisele analüüsile soovitakse kindlasti rohkem aega kulutada, et saada tulemusi, mis on aluseks tuleviku otsustele.

7 Kokkuvõte

Kaasajal on hakatud looma traditsioonilistest andmebaasisüsteemidest erinevaid andmebaasisüsteeme, sest suurenevate andmehulkade hoiustamiseks ning andmepäringute käivitamiseks ei ole mõnikümmend aastat turul olnud andmebaasisüsteemid enam optimaalsed. Uuendustega kaasas käies võib ettevõte valida andmete haldamiseks näiteks NoSQL-andmebaasisüsteemi, kuid enamus neist eeldavad tuntud SQL-andmebaasikeele hülgamist ning andmemudeli muutmist. Magistritöös on vaadeldud andmebaasisüsteeme, mis kasutavad andmete salvestamiseks sisemisel tasemel erinevaid salvestusmeetodeid, kuid võimaldavad SQL-andmebaasikeele kasutamist.

Käesoleva magistritöö peamiseks eesmärgiks on uurida, kuidas mõjutavad rea- ja veerupõhine salvestamine andmebaasides päringute jõudlust. Töö alguses toodi kirjanduse põhjal välja jaekaubanduse valdkonna peamised infovajadused, millele vastuse saamiseks koostati andmepäringud. Eksperimendi käigus loodud andmebaasid on identse andmestikuga ja sisaldavad ühe kaupluste keti aasta ostuandmeid. Reapõhine andmebaas loodi kasutades PostgreSQL andmebaasisüsteemi ja veerupõhine kasutades MariaDB andmebaasisüsteemis Columnstore andmebaasimootorit. Käivitatud andmepäringuid võrreldi töökiiruse alusel analüüsides päringute täitmisplaane, millega leiti andmebaasisüsteemide eelised ja puudused. Lisaks toodi andmepäringute rakendamise näide kaupluste profileerimisel ilmestamaks eksperimenti valitud päringute kasulikkust ärilises kontekstis.

Üldine tulemus näitab, et kõik 12 andmebaasipäringut olid tunduvalt kiiremad veerupõhisel salvestamisel. Antud töös ei ole detailselt uuritud andmete lisamise, muutmise ja kustutamise operatsioone, kuid andmebaaside loomise käigus lisamise käsku kasutati ja ligikaudseid tulemusi hinnati. Lähtudes OLAP päringute töökiirustest mõlemas andmebaasisüsteemis, on veerupõhisel salvestamisel selgelt eelis. Andmete lisamise aegluse probleemi lahenduseks on MariaDB Columnstore süsteemis realiseeritud hulga kaupa andmete laadimine. Kasutusmugavuse poolest vaadeldud andmehaldusvahendis on selge eelis tuntud PostgreSQL andmebaasisüsteemil.

Edasiarendusteks analüüsi poolelt tooks välja detailsema andmeanalüüsi, mis hõlmaks lisaks toodete ostuandmetele ja kategooriatele klientide andmeid, kaupluste asukohti ja laoseisu. Lisaks oleks huvitav analüüsi kaasata päevakajalised uudised, andmed ilma kohta ning kampaaniad, mis annaksid juurde uusi tahke. Kui lõputöö andmed oleks avalikud saaks vaadata käibe ja tulu muutumist, mis sisaldaks ka toodete sisseoste erinevate perioodide peale. Detailsem analüüs võimaldaks ettevõttel ka tööjõu tõhusamat kasutamist.

Lisaks oleks huvitav eksperimenteerida pilvepõhistes andmebaasisüsteemides, kuid kindlasti tuleks sel juhul arvestada välja umbkaudne maksumus. Üheks edasiarenduseks oleks ka eksperimendi sooritamine NoSQL-andmebaasisüsteemides, näiteks graafandmebaasis, kus võiks kasutada antud tööga sarnast andmemahtu tuvastamaks sobivus reaalsete andmetega. Kui NewSQL-andmebaasisüsteemid nagu näiteks VoltDB näitavad tõusutrendi, siis võimalus eksperimenteerida nendega.

Kasutatud kirjandus

[1] Cybernetica AS, „ANDMEKAITSE JA INFOTURBE LEKSIKON,“ [Võrgumaterjal]. Available: https://akit.cyber.ee. [Kasutatud 8 mai 2020].

[2] A. Raman ja F. Marshall, *The New Science of Retailing: How Analytics Are Transforming the Supply Chain and Improving Performance*, 2010.

[3] D. Grewal, A. Roggeveen ja J. Nordfält, „The Future of Retailing,“ *Journal of Retailing,* kd. 93, 2017.

[4] J. Nordfält, D. Grewal, A. Roggeveen ja K. Hill, „Review of Marketing Research: Shopper Marketing and the Role of In-Store Marketing,“ *Insights from In-store Experiments*, Emerald Books, 2014.

[5] L. Sanchez-Ruiz, . B. Blanco ja A. Kyguolienė, „A Theoretical Overview of the Stockout Problem in Retail: from Causes to Consequences,“ 2018.

[6] R. Priya, „Retail data analytics in graph database,“ 2018.

[7] E. T. Bradlow, M. Gangwar, P. K. Kopalle ja S. Voleti, „The Role of Big Data and Predictive Analytics in Retailing,“ *Journal of Retailing,* kd. 93, nr 1, pp. 79-95, 2017.

[8] R. Krieger ja C. Schorr, „A Reference Model for Product Data Profiling in Retail ERP Systems,“ *DATA*, 2019.

[9] T. Rintamäki ja K. Törnroos, „From perceptions to propositions: Profiling customer value across retail contexts,“ *Journal of Retailing and Consumer Services,* kd. 37, 2016.

[10] A. Griva, C. Bardaki, K. Pramatari ja D. Papakyriakopoulos, „Retail Business Analytics: Customer Visit Segmentation Using Market Basket Data,“ *Expert Systems with Applications,* kd. 100, 2018.

[11] M. Carolan, „Big data and food retail: Nudging out citizens by creating dependent consumers,“ *Geoforum,* kd. 90, pp. 142-150, 2018.

[12] S. Hong, K. Misra ja N. Vilcassim, „The Perils of Category Management: The Effect of Product Assortment on Multicategory Purchase Incidence 1,“ *Journal of Marketing,* kd. 80, nr 5, 2016.

[13] F. Wang, Y. Wen, J. Chen ja B.-Q. Cao, „Integrating Collaborative Filtering and Association Rule Mining for Market Basket Recommendation,“ *Web Information Systems Engineering – WISE 2018*, Dubai, 2018.

[14] E. Eessaar, Andmebaaside projekteerimine, Tallinn: Tallinna Tehnikaülikooli kirjastus, 2008.

[15] „DB-Engines Ranking,“ [Võrgumaterjal]. Available: https://db-engines.com/en/ranking. [Kasutatud 10 märts 2020].

[16] „Database Choice for Large Data Volume,“ [Võrgumaterjal]. Available: https://softmedialab.com/blog/database-choice-for-large-data-volume/. [Kasutatud 10 oktoober 2019].

[17] „Why You Should Learn PostgreSQL for Data Science,“ [Võrgumaterjal]. Available: https://www.dataversity.net/why-you-should-learn-postgresql-for-data-science/. [Kasutatud 8 veebruar 2020].

[18] T. P. G. D. Group, „PostgreSQL 12.2 Documentation,“ [Võrgumaterjal]. Available: https://www.postgresql.org/files/documentation/pdf/12/postgresql-12-A4.pdf. [Kasutatud 9 märts 2020].

[19] „cstore_dfw,“ [Võrgumaterjal]. Available: https://github.com/citusdata/cstore_fdw. [Kasutatud 14 märts 2020].

[20] „Columnstore indexes: Overview,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-ver15. [Kasutatud 1 mai 2020].

[21] D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos ja S. Madden, „The Design and Implementation of Modern Column-Oriented Database Systems,“ *Foundations and Trends•R in Databases,* kd. 5, nr 3, pp. 197-280, 2012.

[22] D. Abadi, P. Boncz ja S. Harizopoulos, „Column oriented Database Systems,“ *Proceedings of the VLDB Endowment 2(2):1664-1665*, 2009.

[23] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*, O'Reilly Media Inc., 2017, pp. 95-103.

[24] „Super Decisions,“ [Võrgumaterjal]. Available: https://www.superdecisions.com/. [Kasutatud 11 jaanuar 2020].

[25] „MariaDB Knowledge Base,“ [Võrgumaterjal]. Available: https://mariadb.com/kb/en/mariadb-columnstore/. [Kasutatud 3 mai 2020].

[26] „MariaDB,“ [Võrgumaterjal]. Available: https://mariadb.com/. [Kasutatud 3 mai 2020].

[27] D. Budhrani, „How data compression works: exploring LZ77,“ [Võrgumaterjal]. Available: https://towardsdatascience.com/how-data-compression-works-exploring-lz77-3a2c2e06c097. [Kasutatud 13 märts 2020].

[28] „Snappy,“ [Võrgumaterjal]. Available: https://github.com/google/snappy. [Kasutatud 27 märts 2020].

[29] S. Watanabe, K. Fujimoto, Y. Saeki, Y. Fujikawa ja H. Yoshino, „Column-oriented Database Acceleration using FPGAs,“ 2019.

[30] S. Puustusmaa, „Reapõhise ja veerupõhise andmete salvestamise võrdlus kahe SQL-andmebaasisüsteemi näitel,“ 2016.

[31] D. Abadi, S. Madden ja N. Hachem, „Column-Stores vs. Row-Stores: How Different Are They Really?,“ *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008.

[32] „Docker,“ [Võrgumaterjal]. Available: https://www.docker.com/. [Kasutatud 10 märts 2020].

[33] „6 Types of Inventory and Sales Reports to Use in Your Retail Store,“ [Võrgumaterjal]. Available: https://www.vendhq.com/blog/retail-inventory-sales-reports/. [Kasutatud 25 jaanuar 2020].

[34] „ERPLY,“ [Võrgumaterjal]. Available: https://erply.com/how-to-measure-retail-performance-5-essential-metrics/. [Kasutatud 15 jaanuar 2020].

[35] „dotActiv,“ [Võrgumaterjal]. Available: https://www.dotactiv.com/. [Kasutatud 20 jaanuar 2020].

[36] P. J. Rousseeuw, „Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,“ *Journal of Computational and Applied Mathematics,* kd. 20, pp. 53-65, 1987.

[37] „Sissejuhatus klasteranalüüsi,“ [Võrgumaterjal]. Available: http://samm.ut.ee/sissejuhatus-klasteranal%C3%BC%C3%BCsi. [Kasutatud 3 mai 2020].

Lisa 1 - Saaty mudeli otsustustabelid

Tabel 11. Kriteeriumi eesmärgipärasus hinnangud.

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**Kaalud**
A1	1,00	5,00	1,00	0,20	0,20	3,00	**0,100**
A2	0,20	1,00	0,20	0,14	0,14	0,20	**0,027**
A3	1,00	5,00	1,00	0,20	0,20	3,00	**0,100**
A4	5,00	7,00	5,00	1,00	3,00	5,00	**0,421**
A5	5,00	7,00	5,00	0,33	1,00	5,00	**0,291**
A6	0,33	5,00	0,33	0,20	0,20	1,00	**0,061**

Tabel 12. Kriteeriumi kogemus hinnangud.

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**Kaalud**
A1	1,00	1,00	1,00	0,20	0,20	1,00	**0,084**
A2	1,00	1,00	0,33	0,20	0,20	1,00	**0,067**
A3	1,00	3,00	1,00	1,00	1,00	1,00	**0,173**
A4	5,00	5,00	1,00	1,00	1,00	5,00	**0,316**
A5	5,00	5,00	1,00	1,00	1,00	1,00	**0,246**
A6	1,00	1,00	1,00	0,20	1,00	1,00	**0,114**

Tabel 13. Kriteeriumi kogukond hinnangud.

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**Kaalud**
A1	1,00	0,33	1,00	0,14	0,33	3,00	**0,065**
A2	3,00	1,00	3,00	0,14	0,33	3,00	**0,115**
A3	1,00	0,33	1,00	0,14	0,20	3,00	**0,061**
A4	7,00	7,00	7,00	1,00	5,00	7,00	**0,528**
A5	3,00	3,00	5,00	0,20	1,00	3,00	**0,193**
A6	0,33	0,33	0,33	0,14	0,33	1,00	**0,040**

Tabel 14. Kriteeriumi maksumus hinnangud.

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**Kaalud**
A1	1,00	0,14	1,00	0,14	0,14	1,00	**0,042**
A2	7,00	1,00	7,00	1,00	1,00	7,00	**0,292**
A3	1,00	0,14	1,00	0,14	0,14	1,00	**0,042**
A4	7,00	1,00	7,00	1,00	1,00	7,00	**0,292**
A5	7,00	1,00	7,00	1,00	1,00	7,00	**0,292**
A6	1,00	0,14	1,00	0,14	0,14	1,00	**0,042**

Tabel 15. Kriteeriumi populaarsus hinnangud.

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**Kaalud**
A1	1,00	0,14	0,33	0,20	5,00	0,20	**0,052**
A2	7,00	1,00	5,00	3,00	9,00	5,00	**0,442**
A3	3,00	0,20	1,00	0,20	5,00	0,33	**0,083**
A4	5,00	0,33	5,00	1,00	7,00	3,00	**0,254**
A5	0,20	0,11	0,20	0,14	1,00	0,20	**0,025**
A6	5,00	0,20	3,00	0,33	5,00	1,00	**0,144**

Lisa 2 - Tabelite loomise laused PostgreSQL andmebaasisüsteemis

```
CREATE SCHEMA loputoo AUTHORIZATION postgres;

CREATE TABLE loputoo.kategooriad (
	kategooria_nimi varchar(500) NOT NULL,
	kategooria_id int4 NOT NULL,
	CONSTRAINT PK_kategooriad PRIMARY KEY (kategooria_nimi)
);

CREATE TABLE loputoo.tooted (
	kategooria_nimi varchar(500) NOT NULL,
	toode_id int8 NOT NULL,
	CONSTRAINT PK_tooted PRIMARY KEY (toode_id),
	CONSTRAINT FK_tooted FOREIGN KEY (kategooria_nimi) REFERENCES
loputoo.kategooriad(kategooria_nimi)
);

CREATE TABLE loputoo.ostud (
	kalender_id int4 NOT NULL,
	kauplus_id int4 NOT NULL,
	toode_nimi varchar(1000) NULL,
	toode_kogus numeric(10,3) NULL,
	toode_summa numeric(10,2) NULL,
	ost_id int4 NULL,
	toode_id int8 NOT NULL,
	CONSTRAINT FK_ostud FOREIGN KEY (toode_id) REFERENCES
loputoo.tooted(toode_id)
);
```

Lisa 3 - Tabelite loomise laused MariaDB Columnstore andmebaasisüsteemis

```
CREATE TABLE kategooriad (
  kategooria_nimi varchar(500) COLLATE utf8mb4_estonian_ci NOT NULL,
  kategooria_id int(11) NOT NULL
) ENGINE=Columnstore DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_estonian_ci
```

```
CREATE TABLE tooted (
  kategooria_nimi varchar(500) COLLATE utf8mb4_estonian_ci NOT NULL,
  toode_id bigint(20) NOT NULL
) ENGINE=Columnstore DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_estonian_ci

CREATE TABLE ostud (
  kalender_id int(11) NOT NULL,
  kauplus_id int(11) NOT NULL,
  toode_nimi text COLLATE utf8mb4_estonian_ci DEFAULT NULL,
  toode_kogus decimal(10,3) DEFAULT NULL,
  toode_summa decimal(10,3) DEFAULT NULL,
  ost_id int(11) DEFAULT NULL,
  toode_id bigint(20) NOT NULL
) ENGINE=Columnstore DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_estonian_ci
```

Lisa 4 - P1 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                   QUERY PLAN
----------------------------------------------------------------------------------------------------------------
 Sort  (cost=8180633.75..8180634.57 rows=327 width=36) (actual time=1719256.023..1719258.127 rows=338 loops=1)
   Sort Key: (sum(toode_summa)) DESC
   Sort Method: quicksort  Memory: 40kB
   ->  Finalize GroupAggregate  (cost=8180534.79..8180620.09 rows=327 width=36) (actual time=1719230.981..1719252.714 rows=338 loops=1)
         Group Key: kauplus_id
         ->  Gather Merge  (cost=8180534.79..8180611.10 rows=654 width=36) (actual time=1719230.913..1719243.133 rows=1014 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Sort  (cost=8179534.77..8179535.59 rows=327 width=36) (actual time=1719159.073..1719161.174 rows=338 loops=3)
                     Sort Key: kauplus_id
                     Sort Method: quicksort  Memory: 51kB
                     Worker 0:  Sort Method: quicksort  Memory: 51kB
                     Worker 1:  Sort Method: quicksort  Memory: 51kB
                     ->  Partial HashAggregate  (cost=8179517.03..8179521.11 rows=327 width=36) (actual time=1719154.736..1719156.900 rows=338 loops=3)
                           Group Key: kauplus_id
                           ->  Parallel Seq Scan on "Ostud"  (cost=0.00..7329320.35 rows=170039335 width=10) (actual time=109.726..868284.354 rows=135717853 loops=3)
 Planning Time: 2.152 ms
 JIT:
   Functions: 24
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 3.197 ms, Inlining 99.987 ms, Optimization 127.720 ms, Emission 99.983 ms, Total 330.887 ms
 Execution Time: 1719262.129 ms
(22 rows)
```

Lisa 5 - P1 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table TableOID ReferencedColumns        PIO    LIO    PBE Elapsed Rows
BPS  PM   ostud 3018     (kauplus_id,toode_summa) 597464 608021 0   16.831  247172
TAS  UM   -     -        -                        -      -      -   16.695  338
TNS  UM   -     -        -                        -      -      -   0.000   338
 |
```

Lisa 6 - P2 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Subquery Scan on t  (cost=24682629.97..24798914.41 rows=2584099 width=61) (actual time=658405.957..658416.216 rows=25 loops=1)
   Filter: (t.rea_number <= 5)
   Rows Removed by Filter: 908
   ->  Sort  (cost=24682629.97..24702010.71 rows=7752296 width=133) (actual time=658404.324..658410.070 rows=933 loops=1)
         Sort Key: "Ostud".kauplus_id, (round((((sum("Ostud".toode_summa)) * '100'::numeric) / t1.kaive), 2))
         Sort Method: quicksort  Memory: 151kB
         ->  WindowAgg  (cost=21934360.42..22205690.78 rows=7752296 width=133) (actual time=658377.730..658397.455 rows=933 loops=1)
               ->  Sort  (cost=21934360.42..21953741.16 rows=7752296 width=125) (actual time=658377.614..658383.661 rows=933 loops=1)
                     Sort Key: "Ostud".kauplus_id, (round(((sum("Ostud".toode_summa) / t1.kaive) * '100'::numeric), 2)) DESC
                     Sort Method: quicksort  Memory: 125kB
                     ->  GroupAggregate  (cost=19246205.64..19536916.74 rows=7752296 width=125) (actual time=547425.919..658370.581 rows=933 loops=1)
                           Group Key: "Ostud".kauplus_id, "Tooted".kategooria_nimi, t1.kaive
                           ->  Sort  (cost=19246205.64..19265586.38 rows=7752296 width=67) (actual time=547424.013..603262.335 rows=7791195 loops=1)
                                 Sort Key: "Ostud".kauplus_id, "Tooted".kategooria_nimi, t1.kaive
                                 Sort Method: external merge  Disk: 392640kB
                                 ->  Hash Join  (cost=8486724.52..17723166.96 rows=7752296 width=67) (actual time=339170.729..487586.888 rows=7791195 loops=1)
                                       Hash Cond: ("Ostud".kauplus_id = t1.kauplus_id)
                                       ->  Gather  (cost=16031.67..9231860.59 rows=7752296 width=35) (actual time=156725.334..207202.076 rows=7791195 loops=1)
                                             Workers Planned: 2
                                             Workers Launched: 2
                                             ->  Parallel Hash Join  (cost=15031.67..8455630.99 rows=3230123 width=35) (actual time=156686.970..173331.795 rows=2597065 loops=3)
                                                   Hash Cond: ("Ostud".toode_id = "Tooted".toode_id)
                                                   ->  Parallel Seq Scan on "Ostud"  (cost=0.00..8392066.19 rows=3230123 width=18) (actual time=11.994..136123.515 rows=2597065 loops=3)
                                                         Filter: (kauplus_id = ANY ('{202304,200001,200840,202516,200631}'::integer[]))
                                                         Rows Removed by Filter: 133120788
                                                   ->  Parallel Hash  (cost=9312.41..9312.41 rows=281541 width=33) (actual time=3086.815..3086.822 rows=225232 loops=3)
                                                         Buckets: 65536  Batches: 16  Memory Usage: 3424kB
                                                         ->  Parallel Seq Scan on "Tooted"  (cost=0.00..9312.41 rows=281541 width=33) (actual time=240.432..1660.956 rows=225232 loops=3)
                                       ->  Hash  (cost=8470688.76..8470688.76 rows=327 width=36) (actual time=182445.335..182445.342 rows=5 loops=1)
                                             Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                             ->  Subquery Scan on t1  (cost=8470599.38..8470688.76 rows=327 width=36) (actual time=182444.980..182445.293 rows=5 loops=1)
                                                   ->  Finalize GroupAggregate  (cost=8470599.38..8470685.49 rows=327 width=36) (actual time=182444.965..182445.219 rows=5 loops=1)
                                                         Group Key: "Ostud_1".kauplus_id
                                                         ->  Gather Merge  (cost=8470599.38..8470675.68 rows=654 width=36) (actual time=182444.871..182590.649 rows=15 loops=1)
                                                               Workers Planned: 2
                                                               Workers Launched: 2
                                                               ->  Sort  (cost=8469599.35..8469600.17 rows=327 width=36) (actual time=182404.007..182404.062 rows=5 loops=3)
                                                                     Sort Key: "Ostud_1".kauplus_id
                                                                     Sort Method: quicksort  Memory: 25kB
                                                                     Worker 0:  Sort Method: quicksort  Memory: 25kB
                                                                     Worker 1:  Sort Method: quicksort  Memory: 25kB
                                                                     ->  Partial HashAggregate  (cost=8469581.61..8469585.70 rows=327 width=36) (actual time=182403.911..182403.950 rows=5 loops=3)
                                                                           Group Key: "Ostud_1".kauplus_id
                                                                           ->  Parallel Hash Join  (cost=13931.67..8453430.99 rows=3230123 width=10) (actual time=149615.632..166057.880 rows=2597065 loops=3)
                                                                                 Hash Cond: ("Ostud_1".toode_id = "Tooted_1".toode_id)
                                                                                 ->  Parallel Seq Scan on "Ostud" "Ostud_1"  (cost=0.00..8392066.19 rows=3230123 width=18) (actual time=10.898..129699.859 rows=2597065 loops=3)
                                                                                       Filter: (kauplus_id = ANY ('{202304,200001,200840,202516,200631}'::integer[]))
                                                                                       Rows Removed by Filter: 133120788
                                                                                 ->  Parallel Hash  (cost=9312.41..9312.41 rows=281541 width=8) (actual time=2892.727..2892.733 rows=225232 loops=3)
                                                                                       Buckets: 131072  Batches: 16  Memory Usage: 2720kB
                                                                                       ->  Parallel Seq Scan on "Tooted" "Tooted_1"  (cost=0.00..9312.41 rows=281541 width=8) (actual time=93.246..1497.769 rows=225232 loops=3)
 Planning Time: 4.541 ms
 JIT:
   Functions: 107
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 16.133 ms, Inlining 181.316 ms, Optimization 466.895 ms, Emission 348.238 ms, Total 1012.582 ms
 Execution Time: 658926.967 ms
(57 rows)
```

Lisa 7 - P2 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

Desc	Mode	Table	TableOID	ReferencedColumns	PIO	LIO	PBE	Elapsed	Rows
BPS	PM	ostud	3018	(kauplus_id,toode_id,toode_summa)	1051069	655770	0	9.476	7791225
BPS	PM	tooted	3028	(toode_id)	668	671	0	1.727	675833
HJS	UM	tooted-ostud	3028	-	-	-	-	-----	-
TAS	UM	-	-	-	-	-	-	1.663	5
TNS	UM	-	-	-	-	-	-	0.000	5
BPS	PM	tooted	3028	(kategooria_nimi,toode_id)	1337	1416	0	0.393	675833
BPS	PM	ostud	3018	(kauplus_id,toode_id,toode_summa)	997309	767962	0	12.536	377352
HJS	PM	ostud-tooted	3018	-	-	-	-	-----	-
HJS	PM	ostud-t1	3018	-	-	-	-	-----	-
TAS	UM	-	-	-	-	-	-	12.243	934
WFS	UM	-	-	-	-	-	-	0.002	934
TNS	UM	-	-	-	-	-	-	0.000	934
TNS	UM	-	-	-	-	-	-	0.000	25

Lisa 8 - P3 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=11535070.35..11560809.95 rows=40000 width=76) (actual time=776033.872..845768.110 rows=324 loops=1)
   Group Key: t.kuu, t.kauplus_id
   ->  Sort  (cost=11535070.35..11540058.27 rows=1995168 width=52) (actual time=775998.744..810527.277 rows=5260758 loops=1)
         Sort Key: t.kuu, t.kauplus_id
         Sort Method: external merge  Disk: 216176kB
         ->  Subquery Scan on t  (cost=10850061.03..11189900.19 rows=1995168 width=52) (actual time=290279.034..739448.623 rows=5260758 loops=1)
               ->  Finalize GroupAggregate  (cost=10850061.03..11169948.51 rows=1995168 width=56) (actual time=290279.019..674155.154 rows=5260758 loops=1)
                     Group Key: "Ostud".ost_id, "Ostud".kauplus_id, (date_part('month'::text, (("left"((("Ostud".kalender_id)::character varying)::text, 6))::date)::timestamp without time zone))
                     ->  Gather Merge  (cost=10850061.03..11084587.56 rows=1700394 width=56) (actual time=290278.978..560325.943 rows=11138909 loops=1)
                           Workers Planned: 2
                           Workers Launched: 2
                           ->  Partial GroupAggregate  (cost=10849061.01..10887319.87 rows=850197 width=56) (actual time=289953.148..462099.369 rows=3712970 loops=3)
                                 Group Key: "Ostud".ost_id, "Ostud".kauplus_id, (date_part('month'::text, (("left"((("Ostud".kalender_id)::character varying)::text, 6))::date)::timestamp without time zone))
                                 ->  Sort  (cost=10849061.01..10851186.50 rows=850197 width=30) (actual time=289953.040..362742.163 rows=11399288 loops=3)
                                       Sort Key: "Ostud".ost_id, "Ostud".kauplus_id
                                       Sort Method: external merge  Disk: 464640kB
                                       Worker 0:  Sort Method: external merge  Disk: 470032kB
                                       Worker 1:  Sort Method: external merge  Disk: 470856kB
                                       ->  Parallel Seq Scan on "Ostud"  (cost=0.00..10744985.50 rows=850197 width=30) (actual time=2112.111..208998.320 rows=11399288 loops=3)
                                             Filter: (date_part('month'::text, (("left"(((kalender_id)::character varying)::text, 6))::date)::timestamp without time zone) = '4'::double precision)
                                             Rows Removed by Filter: 124318564
 Planning Time: 21.345 ms
 JIT:
   Functions: 36
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 107.224 ms, Inlining 386.295 ms, Optimization 271.084 ms, Emission 226.595 ms, Total 991.199 ms
 Execution Time: 846071.608 ms
(27 rows)
```

Lisa 9 - P3 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

Desc	Mode	Table	TableOID	ReferencedColumns	PIO	LIO	PBE	Elapsed	Rows
BPS	PM	ostud	3018	(kalender_id,kauplus_id,ost_id,toode_id,toode_summa)	1254347	1250632	0	60.861	5281069
TAS	UM	-	-	-	-	-	-	60.763	5275634
TNS	UM	-	-	-	-	-	-	0.518	5275634
TAS	UM	-	-	-	-	-	-	0.519	324
TNS	UM	-	-	-	-	-	-	0.000	324

Lisa 10 - P4 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=11218748.53..11239157.33 rows=327 width=44) (actual time=1335915.751..2575075.161 rows=326 loops=1)
   Group Key: kauplus_id
   ->  Sort  (cost=11218748.53..11223849.71 rows=2040472 width=14) (actual time=1335695.384..1938958.951 rows=92416173 loops=1)
         Sort Key: kauplus_id
         Sort Method: external merge  Disk: 2350936kB
         ->  Gather  (cost=1000.00..10935154.25 rows=2040472 width=14) (actual time=115.750..697771.905 rows=92416173 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Parallel Seq Scan on "Ostud"  (cost=0.00..10730107.05 rows=850197 width=14) (actual time=102.553..306663.705 rows=30805391 loops=3)
                     Filter: (date_part('quarter'::text, (("left"(((kalender_id)::character varying)::text, 6))::date)::timestamp without time zone) = '1'::double precision)
                     Rows Removed by Filter: 104912462
 Planning Time: 1.776 ms
 JIT:
   Functions: 17
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 2.986 ms, Inlining 124.785 ms, Optimization 109.263 ms, Emission 69.777 ms, Total 306.811 ms
 Execution Time: 2576365.266 ms
(17 rows)
```

Lisa 11 - P4 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table TableOID ReferencedColumns                          PIO     LIO     PBE Elapsed Rows
BPS  PM   ostud 3018     (kalender_id,kauplus_id,ost_id,toode_summa) 1051582 1862891 0   56.876  13715004
TAS  UM   -     -        -                                          -       -       -   57.542  0
TNS  UM   -     -        -                                          -       -       -   0.000   326
 |
```

Lisa 12 - P5 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=8668255.24..8703912.95 rows=4723 width=48) (actual time=129694.544..170485.536 rows=52 loops=1)
   Group Key: (date_part('week'::text, (("left"(((kalender_id)::character varying)::text, 6))::date)::timestamp without time zone))
   ->  Sort  (cost=8668255.24..8677131.30 rows=3550421 width=18) (actual time=129176.328..149346.669 rows=3109129 loops=1)
         Sort Key: (date_part('week'::text, (("left"(((kalender_id)::character varying)::text, 6))::date)::timestamp without time zone))
         Sort Method: external merge  Disk: 91288kB
         ->  Gather  (cost=1000.00..8136349.27 rows=3550421 width=18) (actual time=159.520..105660.255 rows=3109129 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Parallel Seq Scan on "Ostud"  (cost=0.00..7780307.17 rows=1479342 width=18) (actual time=124.581..106494.518 rows=1036376 loops=3)
                     Filter: (kauplus_id = 202304)
                     Rows Removed by Filter: 134681476
 Planning Time: 3.514 ms
 JIT:
   Functions: 17
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 5.868 ms, Inlining 132.860 ms, Optimization 128.622 ms, Emission 80.307 ms, Total 347.658 ms
 Execution Time: 170507.061 ms
(17 rows)
```

Lisa 13 - P5 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table TableOID ReferencedColumns                          PIO     LIO    PBE Elapsed Rows
BPS  PM   ostud 3018     (kalender_id,kauplus_id,ost_id,toode_summa) 1047998 225367 0   7.586   323736
TAS  UM   -     -        -                                          -       -      -   7.576   0
TNS  UM   -     -        -                                          -       -      -   0.000   53
 |
```

Lisa 14 - P6 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                        QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=8668255.24..8703912.95 rows=4723 width=48) (actual time=133593.544..173180.011 rows=12 loops=1)
   Group Key: (date_part('month'::text, (("left"(((kalender_id)::character varying)::text, 6))::date)::timestamp without time zone))
   ->  Sort  (cost=8668255.24..8677131.30 rows=3550421 width=18) (actual time=131463.262..151630.829 rows=3109129 loops=1)
         Sort Key: (date_part('month'::text, (("left"(((kalender_id)::character varying)::text, 6))::date)::timestamp without time zone))
         Sort Method: external merge  Disk: 91288kB
         ->  Gather  (cost=1000.00..8136349.27 rows=3550421 width=18) (actual time=220.673..109058.129 rows=3109129 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Parallel Seq Scan on "Ostud"  (cost=0.00..7780307.17 rows=1479342 width=18) (actual time=215.837..109644.421 rows=1036376 loops=3)
                     Filter: (kauplus_id = 202304)
                     Rows Removed by Filter: 134681476
 Planning Time: 3.376 ms
 JIT:
   Functions: 17
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 3.701 ms, Inlining 333.611 ms, Optimization 148.479 ms, Emission 91.667 ms, Total 577.459 ms
 Execution Time: 173199.427 ms
(17 rows)
```

Lisa 15 - P6 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table TableOID ReferencedColumns                                PIO     LIO    PBE Elapsed Rows
BPS  PM   ostud 3018     (kalender_id,kauplus_id,ost_id,toode_summa) 1058750 225370 0   7.695   323736
TAS  UM   -     -        -                                                -       -      -   7.676   0
TNS  UM   -     -        -                                                -       -      -   0.000   12
 |
```

Lisa 16 - P7 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                        QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=8776433.31..8820978.88 rows=4723 width=56) (actual time=129753.622..171032.947 rows=109 loops=1)
   Group Key: (date_part('isodow'::text, (("left"(((kalender_id)::character varying)::text, 6))::date)::timestamp without time zone)), ("right"(((kalender_id)::character varying)::text, 2))
   ->  Sort  (cost=8776433.31..8785309.36 rows=3550421 width=50) (actual time=129753.569..150324.849 rows=3109129 loops=1)
         Sort Key: (date_part('isodow'::text, (("left"(((kalender_id)::character varying)::text, 6))::date)::timestamp without time zone)), ("right"(((kalender_id)::character varying)::text, 2))
         Sort Method: external merge  Disk: 97960kB
         ->  Gather  (cost=1000.00..8147444.34 rows=3550421 width=50) (actual time=228.776..105521.684 rows=3109129 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Parallel Seq Scan on "Ostud"  (cost=0.00..7791402.24 rows=1479342 width=50) (actual time=150.564..106678.752 rows=1036376 loops=3)
                     Filter: (kauplus_id = 202304)
                     Rows Removed by Filter: 134681476
 Planning Time: 172.761 ms
 JIT:
   Functions: 17
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 3.383 ms, Inlining 134.713 ms, Optimization 216.198 ms, Emission 95.609 ms, Total 449.904 ms
 Execution Time: 171061.642 ms
(17 rows)
```

Lisa 17 - P7 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table TableOID ReferencedColumns                             PIO     LIO    PBE Elapsed Rows
BPS  PM   ostud 3018     (kalender_id,kauplus_id,ost_id,toode_summa) 1085376 225410 0   8.188   324514
TAS  UM   -     -        -                                             -       -      -   8.190   0
TNS  UM   -     -        -                                             -       -      -   0.000   109
 |
```

Lisa 18 - P8 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                                                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------
 Finalize GroupAggregate  (cost=12866643.36..12866649.69 rows=42 width=96) (actual time=124328.434..124363.517 rows=621 loops=1)
   Group Key: (("left"((("Ostud".kalender_id)::character varying)::text, 6))::date), "Tooted".kategooria_nimi, "Ostud".toode_nimi
   ->  Gather Merge  (cost=12866643.36..12866648.19 rows=36 width=96) (actual time=124328.368..124351.915 rows=1040 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial GroupAggregate  (cost=12865643.34..12865644.02 rows=18 width=96) (actual time=124208.435..124223.341 rows=347 loops=3)
               Group Key: (("left"((("Ostud".kalender_id)::character varying)::text, 6))::date), "Tooted".kategooria_nimi, "Ostud".toode_nimi
               ->  Sort  (cost=12865643.34..12865643.39 rows=18 width=68) (actual time=124208.314..124214.419 rows=977 loops=3)
                     Sort Key: (("left"((("Ostud".kalender_id)::character varying)::text, 6))::date), "Ostud".toode_nimi
                     Sort Method: quicksort  Memory: 148kB
                     Worker 0:  Sort Method: quicksort  Memory: 206kB
                     Worker 1:  Sort Method: quicksort  Memory: 156kB
                     ->  Parallel Hash Join  (cost=10024.59..12865642.97 rows=18 width=68) (actual time=13039.040..124200.273 rows=977 loops=3)
                           Hash Cond: ("Ostud".toode_id = "Tooted".toode_id)
                           ->  Parallel Seq Scan on "Ostud"  (cost=0.00..12855598.74 rows=7397 width=51) (actual time=12692.635..123759.631 rows=13726 loops=3)
                                 Filter: ((kauplus_id = 202304) AND (("left"(((kalender_id)::character varying)::text, 6))::date >= '2019-03-04'::date) AND (("left"(((kalender_i
d)::character varying)::text, 6))::date <= '2019-03-10'::date))
                                 Rows Removed by Filter: 135704127
                           ->  Parallel Hash  (cost=10016.26..10016.26 rows=666 width=33) (actual time=337.190..337.196 rows=541 loops=3)
                                 Buckets: 2048  Batches: 1  Memory Usage: 208kB
                                 ->  Parallel Seq Scan on "Tooted"  (cost=0.00..10016.26 rows=666 width=33) (actual time=272.297..333.385 rows=541 loops=3)
                                       Filter: ((kategooria_nimi)::text = '1PG01 Leib ja sai (LeSa)'::text)
                                       Rows Removed by Filter: 224691
 Planning Time: 14.496 ms
 JIT:
   Functions: 60
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 35.378 ms, Inlining 221.872 ms, Optimization 331.300 ms, Emission 239.750 ms, Total 828.300 ms
 Execution Time: 124390.107 ms
(28 rows)
```

Lisa 19 - P8 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table         TableOID ReferencedColumns                                       PIO    LIO    PBE Elapsed Rows
BPS  PM   ostud         3018     (kalender_id,kauplus_id,toode_id,toode_kogus,toode_nimi) 820351 605067 0   9.451   41177
DSS  PM   tooted        3028     (kategooria_nimi)                                       0      1      -   0.000   1
BPS  PM   tooted        3028     (kategooria_nimi,toode_id)                              1337   1420   0   0.063   621
HJS  PM   tooted-tooted 3028     -                                                       -      -      -   ------  -
HJS  PM   tooted-ostud  3028     -                                                       -      -      -   ------  -
TAS  UM   -             -        -                                                       -      -      -   0.028   621
TNS  UM   -             -        -                                                       -      -      -   0.001   621
 |
```

Lisa 20 - P9 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------
-------------------------------------------------------------
 GroupAggregate  (cost=12218951.73..12218951.98 rows=4 width=80) (actual time=118564.466..118568.090 rows=26 loops=1)
   Group Key: kauplus_id, (("left"(((kalender_id)::character varying)::text, 6))::date), (date_part('week'::text, (("left"(((kalender_id)::character varying)::text, 6))::date)::timestamp without time zone)
)
   ->  Sort  (cost=12218951.73..12218951.74 rows=4 width=26) (actual time=118564.072..118565.819 rows=273 loops=1)
         Sort Key: kauplus_id, (("left"(((kalender_id)::character varying)::text, 6))::date)
         Sort Method: quicksort  Memory: 46kB
         ->  Gather  (cost=1000.00..12218951.69 rows=4 width=26) (actual time=67531.725..118568.847 rows=273 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Parallel Seq Scan on "Ostud"  (cost=0.00..12217951.29 rows=2 width=26) (actual time=67472.732..118508.442 rows=91 loops=3)
                     Filter: ((toode_id = '4740113092139'::bigint) AND (kauplus_id = ANY ('{201826,200635,200840,200327,202808}'::integer[])) AND (date_part('week'::text, (("left"(((kalender_id)::character
 varying)::text, 6))::date)
::timestamp without time zone) = '32'::double precision))
                     Rows Removed by Filter: 135717762
 Planning Time: 3.523 ms
 JIT:
   Functions: 17
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 11.192 ms, Inlining 149.049 ms, Optimization 183.165 ms, Emission 126.293 ms, Total 469.699 ms
 Execution Time: 118579.513 ms
(17 rows)
```

Lisa 21 - P9 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table TableOID ReferencedColumns                                        PIO    LIO    PBE Elapsed Rows
BPS  PM   ostud 3018     (kalender_id,kauplus_id,toode_id,toode_kogus,toode_summa) 494216 452904 0   7.453   47
TAS  UM   -     -        -                                                        -      -      -   7.422   20
TNS  UM   -     -        -                                                        -      -      -   0.000   20
 |
```

Lisa 22 - P10 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=10492256.48..10550398.70 rows=7752296 width=29) (actual time=254616.953..392255.764 rows=5555403 loops=1)
   ->  Sort  (cost=10492256.48..10511637.22 rows=7752296 width=29) (actual time=254616.938..307546.799 rows=7791195 loops=1)
         Sort Key: "Ostud".ost_id DESC, "Tooted".kategooria_nimi
         Sort Method: external merge  Disk: 293160kB
         ->  Gather  (cost=16039.07..9234188.80 rows=7752296 width=29) (actual time=152639.565..201500.349 rows=7791195 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Hash Join  (cost=15039.07..8457959.20 rows=3230123 width=29) (actual time=152563.858..201102.074 rows=2597065 loops=3)
                     Hash Cond: (("Tooted".kategooria_nimi)::text = ("Kategooriad".kategooria_nimi)::text)
                     ->  Parallel Hash Join  (cost=15031.67..8449322.99 rows=3230123 width=29) (actual time=152384.271..168735.546 rows=2597065 loops=3)
                           Hash Cond: ("Ostud".toode_id = "Tooted".toode_id)
                           ->  Parallel Seq Scan on "Ostud"  (cost=0.00..8392066.19 rows=3230123 width=12) (actual time=9.213..132337.897 rows=2597065 loops=3)
                                 Filter: (kauplus_id = ANY ('{202304,200001,200840,202516,200631}'::integer[]))
                                 Rows Removed by Filter: 133120788
                           ->  Parallel Hash  (cost=9312.41..9312.41 rows=281541 width=33) (actual time=2834.436..2834.442 rows=225232 loops=3)
                                 Buckets: 65536  Batches: 16  Memory Usage: 3424kB
                                 ->  Parallel Seq Scan on "Tooted"  (cost=0.00..9312.41 rows=281541 width=33) (actual time=0.099..1418.466 rows=225232 loops=3)
                     ->  Hash  (cost=4.40..4.40 rows=240 width=26) (actual time=178.888..178.894 rows=240 loops=3)
                           Buckets: 1024  Batches: 1  Memory Usage: 22kB
                           ->  Seq Scan on "Kategooriad"  (cost=0.00..4.40 rows=240 width=26) (actual time=175.798..177.328 rows=240 loops=3)
 Planning Time: 19.212 ms
 JIT:
   Functions: 58
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 7.966 ms, Inlining 111.713 ms, Optimization 248.064 ms, Emission 166.654 ms, Total 534.397 ms
 Execution Time: 426629.056 ms
(26 rows)
```

Lisa 23 - P10 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table              TableOID ReferencedColumns               PIO    LIO    PBE Elapsed Rows
BPS  PM   ostud              3018     (kauplus_id,ost_id,toode_id)    797131 651562 0   8.517   7791225
BPS  PM   kategooriad        3033     (kategooria_id,kategooria_nimi) 4      6      0   0.046   241
BPS  PM   tooted             3028     (kategooria_nimi,toode_id)      1337   1414   0   2.800   675830
HJS  PM   tooted-kategooriad 3028     -                               -      -      -   -----   -
HJS  UM   tooted-ostud       3028     -                               -      -      -   -----   -
TAS  UM   -                  -        -                               -      -      -   2.668   5555433
TNS  UM   -                  -        -                               -      -      -   9.511   5555433
 |
```

Lisa 24 - P11 täitmisplaan PostgreSQL andmebaasisüsteemis

```
                                                                     QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Finalize GroupAggregate  (cost=7795499.94..7795559.22 rows=234 width=33) (actual time=151865.511..151877.062 rows=208 loops=1)
   Group Key: "Tooted".kategooria_nimi
   ->  Gather Merge  (cost=7795499.94..7795554.54 rows=468 width=33) (actual time=151865.355..151901.012 rows=593 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Sort  (cost=7794499.91..7794500.50 rows=234 width=33) (actual time=151817.673..151818.921 rows=198 loops=3)
               Sort Key: "Tooted".kategooria_nimi
               Sort Method: quicksort  Memory: 41kB
               Worker 0:  Sort Method: quicksort  Memory: 41kB
               Worker 1:  Sort Method: quicksort  Memory: 41kB
               ->  Partial HashAggregate  (cost=7794488.36..7794490.70 rows=234 width=33) (actual time=151814.929..151816.181 rows=198 loops=3)
                     Group Key: "Tooted".kategooria_nimi
                     ->  Parallel Hash Join  (cost=15031.67..7787091.65 rows=1479342 width=33) (actual time=138773.485..145324.105 rows=1036376 loops=3)
                           Hash Cond: ("Ostud".toode_id = "Tooted".toode_id)
                           ->  Parallel Seq Scan on "Ostud"  (cost=0.00..7754418.69 rows=1479342 width=8) (actual time=28.147..129034.137 rows=1036376 loops=3)
                                 Filter: (kauplus_id = 202304)
                                 Rows Removed by Filter: 134681476
                           ->  Parallel Hash  (cost=9312.41..9312.41 rows=281541 width=33) (actual time=3008.736..3008.742 rows=225232 loops=3)
                                 Buckets: 65536  Batches: 16  Memory Usage: 3456kB
                                 ->  Parallel Seq Scan on "Tooted"  (cost=0.00..9312.41 rows=281541 width=33) (actual time=181.473..1590.639 rows=225232 loops=3)
 Planning Time: 0.235 ms
 JIT:
   Functions: 54
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 10.396 ms, Inlining 131.490 ms, Optimization 257.529 ms, Emission 153.206 ms, Total 552.622 ms
 Execution Time: 151913.175 ms
(26 rows)
```

Lisa 25 - P11 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table        TableOID ReferencedColumns          PIO    LIO    PBE Elapsed Rows
BPS  PM   ostud        3018     (kauplus_id,toode_id)      597464 605188 0   6.751   3109129
BPS  PM   tooted       3028     (kategooria_nimi,toode_id) 1337   1415   0   0.345   675833
HJS  UM   tooted-ostud 3028     -                          -      -      -   -----   -
TAS  UM   -            -        -                          -      -      -   0.322   208
TNS  UM   -            -        -                          -      -      -   0.001   208
 |
```

Lisa 26 - P12 täitmisplaan PostgreSQL andmebaasisüsteemis

```
QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=8206560.28..8206915.40 rows=47349 width=60) (actual time=118499.164..119033.386 rows=38 loops=1)
   ->  Sort  (cost=8206560.28..8206678.65 rows=47349 width=60) (actual time=118499.150..118768.766 rows=41324 loops=1)
         Sort Key: "Ostud".toode_nimi, "Tooted".kategooria_nimi
         Sort Method: external merge  Disk: 2680kB
         ->  Gather  (cost=16031.67..8202883.38 rows=47349 width=60) (actual time=117920.213..118233.510 rows=41324 loops=1)
               Workers Planned: 2
               Workers Launched: 2
               ->  Parallel Hash Join  (cost=15031.67..8197148.48 rows=19729 width=60) (actual time=117893.380..118030.920 rows=13775 loops=3)
                     Hash Cond: ("Ostud".toode_id = "Tooted".toode_id)
                     ->  Parallel Seq Scan on "Ostud"  (cost=0.00..8179517.03 rows=19729 width=43) (actual time=132.788..114761.961 rows=13775 loops=3)
                           Filter: (((toode_nimi)::text ~~ '%Leibur%'::text) AND (kauplus_id = 202304))
                           Rows Removed by Filter: 135704078
                     ->  Parallel Hash  (cost=9312.41..9312.41 rows=281541 width=33) (actual time=2995.402..2995.408 rows=225232 loops=3)
                           Buckets: 65536  Batches: 16  Memory Usage: 3424kB
                           ->  Parallel Seq Scan on "Tooted"  (cost=0.00..9312.41 rows=281541 width=33) (actual time=167.356..1574.958 rows=225232 loops=3)
 Planning Time: 4.427 ms
 JIT:
   Functions: 37
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 5.999 ms, Inlining 151.522 ms, Optimization 218.138 ms, Emission 131.043 ms, Total 506.701 ms
 Execution Time: 119051.164 ms
(21 rows)
```

Lisa 27 - P12 täitmisplaan MariaDB Columnstore andmebaasisüsteemis

```
|
Desc Mode Table        TableOID ReferencedColumns                 PIO     LIO     PBE Elapsed Rows
BPS  PM   ostud        3018     (kauplus_id,toode_id,toode_nimi) 2185580 2464810 0   49.359  41324
BPS  PM   tooted       3028     (kategooria_nimi,toode_id)        1181    681     0   0.060   38
HJS  PM   tooted-ostud 3028     -                                 -       -       -   -----   -
TAS  UM   -            -        -                                 -       -       -   0.027   38
TNS  UM   -            -        -                                 -       -       -   0.000   38
 |
```