TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kenn Jaakson 176660IAPM

AEGRIDADE ESITAMINE SQL-ANDMEBAASIDES OHLCV ANDMETE NÄITEL

Magistritöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2019

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: Kenn Jaakson

07.05.2019

Annotatsioon

Käesoleva magistritöö üheks eesmärgiks on kirjeldada erinevaid aegridade esitamiseks mõeldud disainilahendusi SQL-andmebaasisüsteemides, kasutades disainide parema loetavuse ja võrreldavuse huvides nende esitamiseks ühtset mustri formaati. Töö teiseks eesmärgiks on viia läbi eksperiment PostgreSQL (11) andmebaasisüsteemis erinevate disainilahenduste võrdlemiseks. Kolmandaks eesmärgiks on leida eksperimendis käsitletud disainilahenduste seast parim disainilahendus näiterakendusele, mis keskendub väärtpaberite hinna liikumiste statistikale.

Eesmärkide saavutamiseks koondatakse varasemalt aegridade esitamise kohta tehtud uuringutest disainilahendused kokku ning esitatakse kasutades ühtset mustri formaati. Kokku kirjeldatakse kaheksa erinevat disainilahendust. Peale disainilahenduse kirjelduse ja tugevate / nõrkade külgede tuuakse välja lühikesel aegreal põhinev näide, füüsilise disaini andmemudel PostgreSQL jaoks OHLCV andmete näitel ning disainilahenduse realiseerimiseks vajalikud SQL andmekirjelduskeele laused.

Järgmiseks kirjeldatakse eksperimenti – esitatakse ülevaade eksperimendi käigus võrreldavatest disainilahendustest, andmete otsimise ja lisamise ülesannetest, mida kasutatakse disainilahenduste kiiruslike omaduste võrdlemiseks ning andmehulkadest, millele tuginedes mõõtmised läbi viiakse. Eksperiment viiakse läbi nelja disainiga, erinevate andmehulkadega ning katsetades neid koos erinevate insekseerimise lahendustega. Tulemused dokumenteeritakse ning esitatakse.

Eksperimendi tulemuste põhjal analüüsitakse disainilahenduste sobivust näiterakendusele kasutades Saaty analüütiliste hierarhiate meetodit.

Kogu töö käigus loodud SQL kood on GitHubi hoidlas: https://github.com/176660/SQL_aegread

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 82 leheküljel, 6 peatükki, 41 joonist, 61 tabelit.

Abstract

Representing Time Series in SQL Databases in the Example of OHLCV Data

One of the goals of this Master's thesis is to present various time series database designs for SQL databases, by using a pattern format for the presentation for the sake of better readability and comparability of the designs. The second goal of the thesis is to carry out an experiment in the PostgreSQL (11) database management system (DBMS) to compare different designs. The third goal is to find the best design amongst the designs that were the objects of the experiment for an example application that focuses on calculating statistics on securities price movements.

To achieve these goals, designs are presented using a uniform pattern format based on articles, journals, and researches from the past about time series designs in SQL databases. For each design, in addition to the description and the strengths / weaknesses of the design, an example base on a short time series, the physical data model for PostgreSQL, and SQL data definition language statements that are needed to implement the design are also presented. There are eight designs presented in the work.

In the next chapter there is an overview of the designs being compared, data search and insertion tasks used in the experiment, and sizes of datasets where measurements are performed on. The results of the experiment are documented and presented. The experiment involves four designs, datasets with different sizes, and considers different indexing schemes in case of the designs. Based on the results of the experiment, the suitability of the designs for the exemplary application is analyzed using the analytic hierarchy process.

The results of the experiment showed that choosing the right design and index type can significantly reduce the amount of space used by the tables, by increasing query execution times only a little. In addition, it can be concluded that query execution time and speed of insertions depends on the number of rows in the table and the type of index used.

The results of the analytic hierarchy process showed that „vertical vectorization“ design would be the best database design choice for the exemplary application in case of using PostgreSQL (11). The main advantage that „vertical vectorization“ design had over others was the disk space used by tables – on average „vertical vectorization“ used three times less disk space than „naive method“ design. In case of design solution „vertical vectorization“ the best index type was B-tree index (compared with no additional indexes and BRIN indexes). In case of this index type query execution speeds where better compared to the alternatives. Data size and data insertion speed were not considerably worse in case of this index type.

All the SQL code that was created as the result of the work is in the GitHub repository: https://github.com/176660/SQL_aegread

The thesis is in Estonian and contains 82 pages of text, 6 chapters, 41 figures, 61 tables.

Lühendite ja mõistete sõnastik

API	*Application programming interface*, programmiliides
B-Puu	*Balanced Tree*, tasakaalustatud puu
BRIN	*Block Range Index*, plokivahemike indeks
CM	*consistency measure*, järjepidevuse mõõt
CTE	*Common Table Expression*, ühine tabeli avaldis
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
IoT	*Internet of Things*, nutistu
NoSQL	*Not only SQL*, mitte ainult SQL
OHLCV	*Open-High-Low-Close-Volume*
RRA	*Round Robin Archive*, järestikplaanuri arhiiv
SQL	*Structured Query Language*
XML	*Extensible Markup Language*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänu tehnoloogia arengule ning nutistu (IoT, *Internet of Things*) levikule tehakse maailmas üha enam mõõtmisi. Näiteks targa kodu lahendused mõõdavad õhuniiskust ja temperatuuri tubades ning automaatsed kauplemisalgoritmid jälgivad finanatsinstrumentide hinnaliikumisi. Need rakendused mõõdavad, kuidas mingit asja iseloomustavad väärtused aja jooksul muutuvad. Selliseid mõõtmisi, kus andmete peamiseks teljeks on aeg, nimetatakse aegridadeks. Selleks, et nendest mõõtmistest kasu oleks, tuleks neid analüüsida ehk teha aegridade analüüsi.

Aegridade analüüsi (näiteks aegridade põhjal statistika loomine) läbiviimiseks tuleks aegread salvestada. Üheks võimalikuks viisiks on aegridade salvestamine mõnes andmebaasisüsteemi abil loodud andmebaasis. Viimaste aastate jooksul on turule tekkinud palju aegridadele spetsialiseerunud andmebaasisüsteeme, mis salvestavad aegread kettaruumi efektiivselt kasutades ning pakuvad aegridade analüüsi lihtsustavaid funktsioone.

Siiski saab välja tuua mitmeid põhjuseid, miks eelistada aegridade salvestamist üldotstarbelise SQL-andmebaasisüsteemi (edaspidi SQL-andmebaasisüsteemi) abil loodud andmebaasis spetsialiseerunud andmebaasisüsteemi (mis võib kuid ei pruugi põhineda SQLi andmemudelil) andmebaasile. SQL-andmebaasisüsteemides on võimalik hallata ka aegridadest teistsuguse struktuuriga andmeid. Vastasel juhul tuleks panna andmeid kasutav rakendus suhtlema kahe erineva andmebaasisüsteemiga – üks on mõeldud aegridade andmete haldamiseks ja eraldi andmebaasisüsteem ülejäänud andmete haldamiseks, kuid see teeb andmete haldamise ja rakenduse loomise keerukamaks ning kallimaks. SQL-andmebaasisüsteemid on 2019. aasta mai seisuga oluliselt populaarsemad kui aegridadele spetsialiseerunud andmebaasisüsteemid [1], mis tähendab, et populaarsematel SQL-andmebaasisüsteemidel on suurem kogukond, rohkem materjale ning tööturul rohkem eksperte. 2019. aasta mai seisuga on andmebaasisüsteemide populaarsusedetabeli esikümnes kuus SQL-andmebaasisüsteemi. Samal ajal on kõige populaarsem aegridadele pühendunud süsteem (InfluxDB) alles 34.

kohal. Lisaks on populaarsemaid SQL-andmebaasisüsteeme (näiteks MySQL, PostgreSQL) kauem arendatud ning rohkem testitud kui spetsialiseerunud andmebaasisüsteeme, mis on turule ilmunud suhteliselt hiljuti. Lõpuks on aegridade salvestamisel SQL-andmebaasidesse mõõdetud konkurentsivõimelised, teatud juhtudel isegi paremad, päringute täitmise kiirused võrreldes aegridadele spetsialiseerunud andmebaasisüsteemides loodud andmebaasidega [2].

Aegridade salvestamiseks SQL-andmebaasis on olemas erinevaid lahendusi. Andmebaasi loomisel tuleb hoolikalt läbi mõelda, kuidas andmebaas disainida, kuid autor ei leidnud ühtegi varasemalt tehtud uuringut, kus oleks võrreldud mitmeid disainilahendusi ning toodud välja nende nõrgad / tugevad küljed ja erinevate ülesannete lahendamiseks mõeldud andmekäitluse operatsioonide täitmise kiirused.

Käesoleva magistritöö esimeseks eesmärgiks on esitada erinevaid aegridade salvestamiseks mõeldud disainilahendusi SQL-andmebaasides, kasutades disainide parema loetavuse ning võrreldavuse huvides nende esitamiseks ühtset mustri formaati.

Magistritöö teiseks eesmärgiks on viia läbi eksperiment PostgreSQL andmebaasisüsteemis erinevate disainilahenduste võrdlemiseks. Eksperimendi käigus otsitakse vastust järgnevatele küsimustele.

- Kuidas erinevad päringute kiirused erinevate disainilahenduste korral sama andmehulgaga?
- Kuidas erinevad päringute kiirused sama disainilahenduse, kuid erineva andmehulgaga?
- Kuidas erinevad päringute tegemiseks vajalike lausete keerukused erinevate disainilahenduste korral?
- Milline on erinevate disainilahenduste põhjal loodud tabelite andmemaht?

Magistritöö kolmandaks eesmärgiks on leida esitatud ning eksperimendis läbiproovitud disainilahenduste seast parim disainilahendus näiterakendusele, mis keskendub väärtpaberite hinna liikumiste statistikale. Parima disainilahenduse valimiseks kasutatakse töös Saaty analüütiliste hierarhiate (edaspidi Saaty) meetodit.

Töö metoodika põhineb disainiteaduse metoodika [3] soovitustel. Töö tulemuseks on tehnilised tehised – disainilahenduste kataloog ning parima disaini valik konkreetse konteksti korral. Kataloogi koostamisel arvestatakse maailmas selles valdkonnas tehtud töödega (esitakse disainilahendusi, mida ka varem on kirjanduses välja pakutud). Parima disainilahenduse valik näiterakendusele põhineb eksperimentidel.

Kõigepealt antakse ülevaade töö teoreetilisest taustast, kus kirjeldatakse aegridadega seotud mõisteid ning räägitakse lühidalt turu populaarseimatest aegridadele spetsialiseerunud andmebaasisüsteemidest. Järgmises peatükis antakse ülevaade varasematest uuringutest. Edasi tuuakse ühtses mustri formaadis välja erinevad disainilahendused. Viimased peatükid keskenduvad eksperimendi läbiviimisele, disainilahenduste võrdlemise eesmärgil ning näiterakendusele parima andmebaasi disainilahendusele leidmisele kasutades Saaty meetodit.

2 Teoreetiline taust

Selles peatükis antakse lühiülevaade aegridadest, nendega seotud mõistetest ja omadustest ning nende esitamise võimalustest erinevates andmebaasisüsteemides.

2.1 Aegread

Põhinedes aegrea klassikalisele definitsioonile on tegemist nähtuse ajalist muutumist iseloomustavate andmete reaga [4]. Aegrea elementideks on nähtust iseloomustavate tunnuste väärtused (üldjuhul arvulised) (Tabel 1 ühekordselt alla joonitud) ning nendega seotud ajamoment või -periood [5] (Tabel 1 kahekordselt alla joonitud). Tabel 1 tervikuna on näide aegreast. Aegrida iseloomustavaks tunnuseks on, et juba registreeritud andmeid muudetakse harva või üldse mitte, st andmebaasis toimub palju INSERT operatsioone, kuid vähe või üldse mitte UPDATE operatsioone. Kuna aegread koosnevad aegrea elementidest, mis iseloomustavad mingi nähtuse väärtusi mingil ajahetkel või ajavahemikul (näiteks aktsia hind 1. jaanuaril), siis ei ole võimalik, et hiljem see väärtus muutub. Erandiks on see kui väärtuse mõõtmisel või andmete sisestamisel tekkis mingisugune probleem. Aegridu iseloomustab ka see, et andmed saabuvad ajalises järjestuses ning andmete peamiseks teljeks on aeg. Aegridasid kasutatakse üldiselt igas valdkonnas, mis hõlmavad ajalisi mõõtmisi, näiteks statistika, fintantsmatemaatika ning ilmaprognoosimine. Aegread jagunevad omakorda moment- ja perioodridadeks.

Tabel 1 Näide pidevast momentreast.

ajatempel	**hind**
1. jaanuar 2019 00:00:00	120.79 €
2. jaanuar 2019 00:00:00	121.43 €
3. jaanuar 2019 00:00:00	125.11 €

Momentrida on aegrida, mille väärtused on seotud kindla ajamomendiga (kuupäev, aasta algus või lõpp), näiteks väärtpaberi väärtus 1. jaanuar 2019 kell 00:00:00 seisuga (Tabel 1). Momentread jagunevad omakorda võrdperioodseteks ning mittevõrdperioodseteks. Võrdperioodsete momentridade puhul on ajavahemikud ajamomentide vahel võrdsed ning mittevõrdperioodsete momentridade puhul ei ole ajavahemikud võrdsed.

Perioodrida on aegrida, mille väärtused on seotud kindla ajavahemikuga (näiteks minut, päev, kuu), näiteks ostetud väärtpaberite kogus 1. jaanuaril 2019 (antud juhul on perioodiks üks päev) (Tabel 2). Perioodread jagunevad omakorda pidevateks ja sõredateks perioodridadeks. Pidevate perioodridade puhul järgnevad perioodid vahetult üksteisele ning on kindla intervalliga. Sõredate perioodridade puhul võivad vaatlused olla üksteisest eraldatud ebakorrapäraste intervallidega ning mitte järgneda üksteisele.

Tabel 2 Näide pidevast perioodreast.

ajavahemiku algus	**ajavahemiku lõpp**	**keskmine hind**	**vahetatud kogus**
1. jaanuar 2019 00:00:00	1. jaanuar 2019 23:59:59	122.79 €	1 567
2. jaanuar 2019 00:00:00	2. jaanuar 2019 23:59:59	120.43 €	1 423
3. jaanuar 2019 00:00:00	3. jaanuar 2019 23:59:59	124.11 €	1 866

Selles töös keskendutakse eeskätt pidevatele perioodridadele, kuid disainilahendused on kasutatavad ka muude aegridade esitamiseks.

2.2 Aegridade esitamine andmebaasides

Aegridade analüüsi (näiteks aegridade põhjal statistika loomine) läbiviimiseks tuleks aegread salvestada. Üheks võimalikuks viisiks on aegridade salvestamine mõnes andmebaasisüsteemi abil loodud andmebaasis. Andmebaasisüsteemi kasutamise mõningateks eelisteks on nende pakutavad funktsionaalsused, mis võimaldavad andmeid arendajasõbralikult töödelda, toetavad kaugligipääsu ning võimaldavad andmetele samaaegset juurdepääsu. Selle töö raames jagan andmebaasisüsteemid kaheks: mitte-SQL (NoSQL) andmebaasisüsteemid ja SQL-andmebaasisüsteemid.

Järgnevalt antakse ülevaate kõige populaarsemast aegridadele spetsialiseerunud mitte-SQL ja SQL-andmebaasisüsteemist.

2.2.1 Mitte-SQL andmebaasid

Mitte-SQL ehk NoSQL (*not only SQL*) andmebaasisüsteemid on alternatiivid relatsioonilistele SQL-il põhinevatele andmebaasisüsteemidele. Mitte-SQL andmebaasisüsteemide andmebaasikeele aluseks on teistsugune andmemudel kui SQL andmebaasisüsteemide aluseks olev andmemudel. Sellest tulenevalt on teistsugused andmebaaside loomiseks kasutatavad ehitusplokid ja operatsioonid, mida saab teha plokkidest kokkupandud struktuuridega. Samuti saab ja tuleb struktuuridele defineerida teistsuguseid kitsendusi.

Ajalooliselt on sellisteks andmebaasisüsteemideks näiteks objektorienteeritud ja XML andmebaasisüsteemid, mis töö kirjutamise ajal on nišitooted. Uuesti tõusid mitte-SQL andmebaasisüsteemid tähelepanu keskpunkti 2010ndatel aastatel, kui esile kerkis uus laine selliseid süsteeme. Nendele viitamiseks hakati kasutama nime NoSQL. Käesolevas töös mõeldakse mitte-SQL andmebaasisüsteemide all NoSQL andmebaasisüsteeme. Samas on ajalooliselt ka näiteid püüdlustest kasutada aegridade haldamiseks objektorienteeritud andmebaasisüsteeme [6] ja XML andmebaasisüsteeme [7] [8].

Martin Fowler nimetab NoSQL-i olulisemateks iseloomustatavateks joonteks [9] järgmiseid.

- Ei kasutata relatsioonilist mudelit ega SQL andmebaasikeelt.
- Avatud lähtekoodiga (praeguseks on tekkinud ka mitmed kommerts mitte-SQL andmebaasisüsteeme).
- Sagedasti puudub ilmutatult kirjeldatud andmebaasi skeem, see tähendab et on võimalus lisada välju (*fields*) kirje põhiselt. Samas peab andmetel kasutamiseks skeem ehk struktuur olema. Mitte-SQL andmebaasis on tegemist lugemise skeemiga, mille puhul peavad andmete kasutajad (rakendused) olema skeemist teadlikud. See tähendab et skeemi kirjeldus on mingil kujul andmeid kasutava rakenduse koodis. Samas kirjutamise skeem, mille puhul kontrollitakse andmete struktuuri andmete salvestamisel, võib, kuid ei pruugi olla määratud [10], [11].

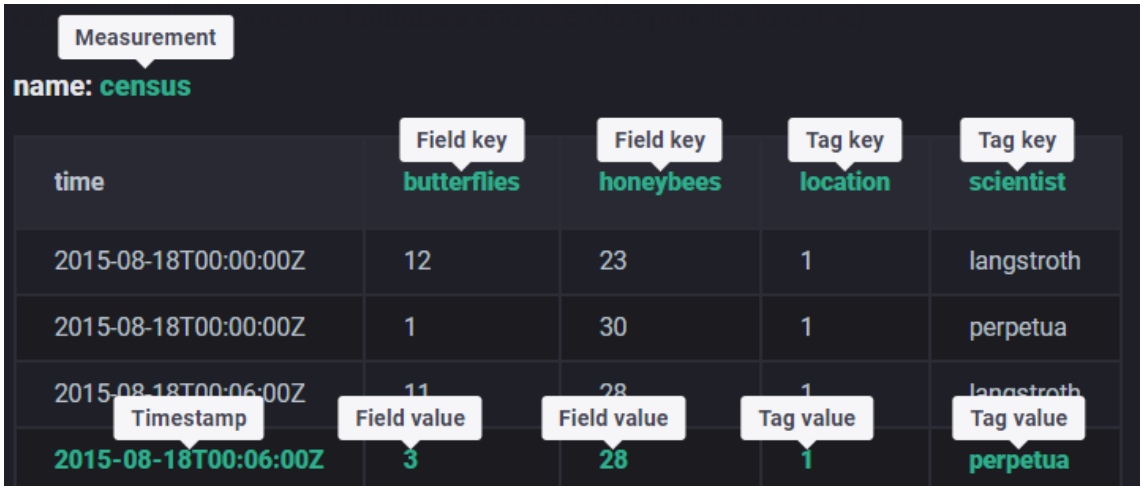
Andmebaasisüsteemide populaarsuse indeksi [1] kohaselt on mitte-SQL "vihmavarju" alla langevatest andmebaasisüsteemidest 2019. aasta maikuu seisuga populaarseimad MongoDB (5. koht), Redis (8. koht) ja Cassandra (10. koht). Aegridadele

spetsialiseerunud andmebaasisüsteemide seas populaarsemad mitte-SQL andmebaasisüsteemid on InfluxDB (üldedetabelis 34. koht), Kdb+ (55. koht) ja Graphite (82. koht). Järgnevalt uurin lähemalt neist kõige populaarsemat andmebaasisüsteemi InfluxDB. [1]

InfluxDB

InfluxDB on avatud lähtekoodiga Go programmeerimiskeeles kirjutatud aegridade esitamisele spetsialiseerunud andmebaasisüsteem. InfluxDB esimene versioon avalikustati 2013. aasta septembris. Süsteemi loojaks ja arendajaks on InfluxData. [12]

Järgnev ülevaade põhineb viidetel [13], [14]. InfluxDB andmebaasi olulisemateks ehitusplokkideks on mõõtmised (*measurement*), väljad (*fields*), aeg (*time*) ja sildid (*tags*). Mõõtmiste all mõeldakse konteinerit, mis on idee poolest sarnane SQL-andmebaasides kasutatava tabeliga. Igas andmebaasis on null või rohkem mõõtmist. Konteineri (mõõtmise) nimi võiks kirjeldada selle veergudesse salvestatud andmeid. Näide konteinerist on Joonis 1.



Joonis 1. Näide InfluxDB konteinerist. [13]

InfluxDB andmebaasis on igal konteineril kohustuslik ajaveerg (*time*), mis on ka ühtlasi konteineri primaarvõtmeks. Iga selles veerus olev väärtus esitab ajahetke. Vaikimisi talletatakse ajahetk nanosekundi täpsusega. Igas konteineris on kohustulik vähemalt üks väljaveerg. Väljavõti (*field key*) on nagu SQL tabeli veeru nimi ja väljaväärtus (*field value*) on nagu rea väljas olev väärtus. Seega on võimalik kõik väljaveerus olevad andmed esitada väljavõtme ja väljaväärtuse paaridena. Väljaväärtuse võimalikeks andmetüüpideks on sõne (*string*), ujukomaarv (*float*), täisarv (*integer*) või tõeväärtus

(*boolean*) ning väljad ei ole indekseeritud ehk väljaväärtuse järgi kirje leidmine nõuab kõigi mõõtmise kirjete läbivaatamist. [13] Väljaväärtustega tehakse päringutes sageli matemaatilisi operatsioone (nt summeerimine, keskmise leidmine). Väljade hulk (*field set*) on väljavõtmete ja väljaväärtuste paaride kollektsioon.

Vajadusel saab konteinerisse lisada siltveerge, mis on sisuliselt nagu indekseeritud veerud SQL-andmebaasi tabelis. Konteineris ei pea siltveerge olema, see tähendab nende lisamine on vabatahtlik. Siltvõti (*tag key*) on nagu SQL tabeli veeru nimi ja siltväärtus (*tag value*) on nagu rea väljas olev väärtus. Seega on võimalik kõik siltveerus olevad andmed esitada silvõtme ja siltväärtuse paaridena. Tänu indekseerimisele on siltveerud sobivad päringute WHERE ja GROUP BY klauslites kasutamiseks. InfluxDB andmebaasi disainer peab otsustama, millised väärtused salvestada väljaveerus ja millised siltveerus. Siltide hulk (*tag set*) on siltvõtmete ja siltväärtuste paaride hulk, kus ei ole korduseid.

Siltide eesmärk on sama ajatempliga (*timestamp*) aegridu eristada (näiteks erinevates asukohtades mõõdetud temperatuur), vastasel juhul toimuks sama ajatempliga andmete lisamisel hoopis andmete muutmine (*UPDATE*), sest InfluxDB ei toeta korduvaid (ühesuguste andmetega) kirjeid, millel on sama mõõtmine ja siltide hulk.

Igal mõõtmine on seotud ühe või mitme säilituspoliitikaga (*retention policy*), mis määrab andmebaasisüsteemi poolt andmete säilitamise pikkuse ning selleks tehtavate koopiate arvu. InfluxDB jaoks on aegrida andmete hulk, millel on sama säilituspoliitika, mõõtmine ja siltide hulk. Ajahetk (*point*) on nagu tabeli rida (Joonis 1) – see on väljade hulk aegreas, millel on sama aeg.

Järgnevalt toon välja mõned InfluxDB (versioon 1.7) märkimisväärsed omadused [15] [13].

1. InfluxDB poolt kasutatav andmebaasikeel InfluxQL on väga sarnane SQL-ile, mis tähendab väiksemat õppimiskõverat SQLi tundvatele arendajatele.

2. InfluxQLi funktsionaalsus sisaldab aegridade analüüsi lihtsustavaid funktsioone, näiteks standardhälve leidmiseks (kogu nimekirja saadaolevatest funktsioonidest leiab viitest [16]).

3. HTTP API, mis võimaldab andmebaasiga suhelda kasutades HTTP päringuid JSON formaadis [17].

4. Andmebaasisüsteemi arhitektuuri tõttu on muutmise (*UPDATE*) ja kustutamise (*DELETE*) lausete funktsionaalsus piiratud, et võimaldada paremat andmete lisamise (*CREATE*) ja lugemise (*READ*) kiirust. Näiteks ei ole võimalik kustutada ajahetki („ridu“) väljaväärtuse põhjal [18].

5. Andmebaasisüsteemi arhitektuuri tõttu on andmete, mille ajaviited (*timestamp*) ei ole kasvavalt järjestatud, sisetamine oluliselt aeglasem võrreldes kasvavalt järjestatud andmete sisestamisega.

6. Suure kasutuskoormuse korral ei pruugi päringute tulemused ehk sisaldada kõige viimasena lisatud andmeid.

7. Kohustuslik ajaväli ning sellega seotud primaarvõtme kitsendus piirab andmeid, mida saab andmebaasis efektiivselt hoiustada. See tähendab, et pole võimalik hoiustada andmeid, mille unikaalseks identifikaatoriks olevatel väärtustel on ajatemplist erinev andmetüüp.

8. Andmebaasisüsteemi poolt kasutatav andmemudel piirab funktsionaalsust – ei ole võimalik metaandmeid uuendada, teostada andmete valideerimist ning kasutada saab väikest hulka andmetüüpe

9. Võimalik on aegridadele sobilik andmete kokkupakkimine, et vähendada kettaruumi kasutust.

10. InfluxDB andmebaasisüsteemi klastri loomine (*clustering*) on saadaval ainult kommertsversioonis [19].

Autori arvates on InfluxDB sobilik valik süsteemidele, mille prioriteediks ei ole andmete käideldavus (võimalik lahendada kommertsversiooni eest makstes, mis võimaldab klastrite loomist) ning andmestruktuur vastab InfluxDB poolt pakutavale andmemudelile. Vastaseljuhul tuleks panna andmeid kasutav rakendus suhtlema kahe erineva andmebaasisüsteemiga – üks aegridade andmete haldamiseks ja eraldi andmebaasisüsteem ülejäänud andmete haldamiseks, kuid see teeb andmete haldamise ja rakenduse loomise keerukamaks ning kallimaks.

2.2.2 SQL-andmebaasid

SQL-andmebaasisüsteemi all mõeldakse andmebaasisüsteemi, mis võimaldab kasutada andmebaasikeelt SQL (*Structured Query Language*) ning SQL-andmebaasi all mõeldakse eelnevalt kirjeldatud anbmebaasisüsteemi abil loodud andmebaasi. SQL põhineb Edgar Frank Coddi poolt välja pakutud relatsioonilisel mudelil, kuid ei järgi seda täielikult [20].

SQL-andmebaasisüsteemide kasutamise teeb ahvatlevaks nende populaarsusest tingitud kogukonna tugi ja ekspertide kättesaadavus tööturul. Populaarseimatest andmebaasisüsteemidest on 2019. aasta maikuu seisuga esimesed neli (Oracle, MySQL, MSSQL Server, PostgreSQL) SQL-andmebaasisüsteemid [1]. Kõige populaarsemaks aegridadele spetsialiseerunud SQL-andmebaasisüsteemiks on üldedetabelis 137. kohal asuv TimescaleDB (aegridadele spetsialiseerunud andmebaasisüsteemide edetabelis 8. koht). Järgnevalt kirjeldan lühidalt TimescaleDB andmebaasisüsteemi.

TimescaleDB

TimescaleDB on avatud lähtekoodiga pistikprogramm (*extension*) PostgreSQL andmebaasisüsteemile [21]. TimescaleDB esimene versioon avalikustati 2017. aasta mais [22].

TimescaleDB olulisemateks mõisteteks on hüpertabel (*hypertable*) ja tükid (*chunks*). Hüpertabeliks kutsutakse TimescaleDB-s tükkide abstraktsiooni ehk vaadet (*view*), mis koosneb mitmest tükist. Iga tükk on sisuliselt üks tabel, mis hoiab andmeid. Tükid luuakse andmete sektsioonimise (*partitoning*) teel. Vaikimisi sätetes tükeldatakse andmed ajaintervalli põhjal, kuid on võimalik lisaks ajale tükeldada andmed ka metaandmete põhjal (näiteks mõõtmise teinud seadme unikaalne identifikaator). [21]

TimescaleDB mõned märkimisväärsed omadused [23] [24] on järgmised.

1. Arendaja jaoks tuttav andmebaasikeel – SQL.

2. Toetab kõiki PostgreSQL andmebaasisüsteemis pakutavaid võimalusi, sealhulgas andmetüüpe, indeksite tüüpe, trigereid, funktsioone ja muid tööriistu (näiteks andmebaasist koopiate loomine ja andmete taastamine).

3. Võimalik on hoida aegridade andmeid ja ülejäänuid andmebaasisüsteemi hallatavaid andmeid ühe ja sama andmebaasisüsteemi hoole all olevas

andmebaasis. See tähendab ei pea võtma kasutusele eraldi andmebaasisüsteemi aegridade andmete haldamiseks ja eraldi andmebaasisüsteemi ülejäänud andmete haldamiseks. Kuigi ühe ja sama rakenduse poolt erinevate andmebaasisüsteemide (mis võivad põhineda erinevatel andmemudelitel) kasutamine on võimalik (seda kutsutakse *polyglot persistence* [25]), muudab see süsteemi keerukamaks.

4. Ajapõhised funktsioonid, näiteks intervalli põhine andmete koondamine.

5. Automaatne aegridade säilitamise poliitika (*retention policy*). Näiteks soovi korral on võimalik hoiustada ainult viimase seitsme päeva andmed.

Autori arvates, põhinedes kasutuskogemusele, on TimescaleDB sobilik lahendus teisejärguliste süsteemide (näiteks hobiprojektid) puhul, sest tegu on uue ning aja jooksul veel testimata lahendusega. Sellise süsteemi nõrkuseks võib pidada kogukonna ning saadaolevate materjalide vähesust, mis aitaks probleemide ilmnemisel. Lisaks ei ole TimescaleDB kasutamine veel võimalik mõningate pilveteenusepakkujate juures.

Siiski on TimescaleDB märkimisväärsete omaduste juures välja toodud SQLi ja võimeka SQL-andmebaasisüsteemi kasutamisest tulenevad eelised (omadused 1–3) sellised, mis väärivad tähelepanu ja võiksid pakkuda palju kasu.

Käesoleva töö disainilahenduste konkreetses andmebaasisüsteemis realiseerimise ja eksperimendi tegemise osades keskendun SQL-andmebaasisüsteemidele. Täpsemalt esitatakse disainilahendused ning viiakse eksperimendid läbi PostgreSQL andmebaasisüsteemis ilma TimescaleDB pistikprogrammita. Järgnevalt põhjendan PostgreSQLi ning üleüldiselt SQL-andmebaasisüsteemi kasutamise valikut.

1. SQL-andmebaasisüsteemides on võimalik esitada ka aegridadest teistsuguse struktuuriga andmeid. Vastaseljuhul tuleks panna andmeid kasutav rakendus suhtlema kahe erineva andmebaasisüsteemiga – üks aegridade andmete haldamiseks ja eraldi andmebaasisüsteem ülejäänud andmete haldamiseks, kuid see teeb andmete haldamise ja rakenduse loomise keerukamaks ning kallimaks.

2. PostgreSQL on pikalt arenduses olnud projekt (esimene väljalase 1996. aastal, kuid eellaseks olnud süsteemi loomine algas juba 1986. aastal), mis teeb sellest usaldusväärse ning süvitsi testitud lahenduse [26].

3. PostgreSQL on 2019. aasta maikuu seisuga populaarsuselt teine avatud lähtekoodiga andmebaasisüsteem, mis tähendab materjalide, suure kogukonna ning tööturul ekspertide olemasolu [1].

4. Autoril on varasem kogemus PostgreSQL andmebaasisüsteemiga ning leiab, et andmebaasisüsteemi poolt pakutavad võimalused võiksid olla sobilikud aegridadega töötamiseks (näiteks akna (*window*) funktsioonid [27] ja massiivid).

5. PostgreSQL on avatud lähtekoodiga tasuta tarkvara ning toetab igat populaarsemat operatsioonisüsteemi [28].

6. PostgreSQLil on varem mõõdetud konkurentsivõimelised päringute täitmise kiirused võrreldes aegridadele spetsialiseerunud andmebaasiga (InfluxDB) [2].

2.3 Näiterakenduse kirjeldus

Käesolevas töös otsitakse parimat andmebaasi disainilahendust rakendusele, mille eesmärgiks on kasutajale anda informatsiooni väärtpaberiturul toimuvatest hinna muutustest ning kuidas need tema portfelli väärtust mõjutavad. Näiterakenduse teisi komponente (näiteks kasutajaliides) selle töö raames ei realiseerita. Portfelli all mõeldakse kasutaja poolt sisestatud väärtpaberitega seotud tehingute lõppseisu, mis väljendab kui palju väärtpabereid kasutaja mingil ajahetkel omas (Tabel 3). Järgnevalt põhjendatakse näiterakenduse valikut:

1. Väärtpaberituru hinna muutuste esitamiseks vajalike päringutega kaetakse põhilisemad aegridadega seotud päringud, mis tähendab, et lugejal on võimalik töös toodud päringutega seotud laused oma kasutusjuhuga lihtsalt sobitada. Päringute näideteks on aegrea elemendi väärtuse leidmine mingil ajahetkel või andmete koondamine kindla intervalliga vahemikeks.

2. Autor ei leidnud 2019. aasta maikuu seisuga GitHubist ühtegi avalikku hoidlat, kus oleks olemas kirjeldatud näiterakenduse andmebaasi disainilahendus.

Tabel 3. Näide kasutaja investeeringute portfelli lõppseisust mingil ajahetkel.

Aktsiat tähistav sümbol	**Kogus**	**Keskmine soetamise hind**
GOOG	2	1 100.00 €
FB	1	160.00 €

Disainilahenduste lihtsustamiseks vaadeldakse väärtpaberitest ainult aktsiate ehk ettevõtte osakute informatsiooni väljendamist, kuid disainilahendusi saab kasutada ka muude väärtpaberite ning aegridade hoiustamiseks. Aktsiate hinna liikumise iseloomustamiseks kasutatakse käesolevas töös OHLCV (*open-high-low-close-volume*) andmete struktuuri. OHLCV on turuandmete kokkuvõtlik vorm, mis sisaldab viite andmepunkti: avamishind (*open*) ja lõpphind (*close*) esindavad esimest ja viimast hinnataset teatud ajaperioodil, kõrgeim (*high*) ja madalaim hind (*low*) on sellel ajaperioodil kõrgeim ja madalaim saavutatud hind ning tehingute kogus (*volume*) on selle ajaperioodi jooksul kaubeldud väärtpaberite kogusumma. OHLCV andmetega on veel seotud ka ajaperiood, mille kohta andmed käivad. Tabel 4 on näide andmete struktuurist, mille hoiustamisest lähtudes luuakse disainilahenduste mustri osas andmemudel ning viiakse läbi eksperimendid.

Järgnevalt toon välja näiterakenduse osad, mille päringute kiirust eksperimendi osas mõõdetakse.

Aktsia hinna liikumise kujutamine küünlagraafina

Rakenduse üheks eesmärgiks on aktsia hinna liikumise kujutamine küünlagraafina. Näite sellisest graafist võib leida viitest [29] kui valida graafitüübiks „*Candle*“ (vaikimisi valik on *Line*) (Joonis 2).

Joonis 2 Küünlagraafi näide [29].

Küünlagraafi andmed sisaldavad vajalikke andmeid ka teiste viites toodud graafitüüpide esitamiseks. Küünlagraafi loomiseks vajalikud andmed on kujutatud Tabel 4 ning tegu on pideva perioodreaga. Perioodrea ajavahemikud peaks sõltuma graafile kuvatava ajavahemiku pikkusest, näiteks seitsme päeva puhul võiks perioodrea ajavahemikuks olla 15 minutit. Selline lähenemine aitab vähendada graafi loomiseks ning andmete edastamiseks kuluvat aega.

Tabel 4. Küünlagraafi loomiseks vajalikud (OHLCV) andmed.

Alguse ajatempel	**Lõpu ajatempel**	**Avamishind**	**Kõrgeim hind**	**Madalaim hind**	**Lõpphind**	**Kogus**
29.03.2019 11:00:01	29.03.2019 11:15:00	120.00	122.23	119.87	121.01	100
29.03.2019 11:15:01	29.03.2019 11:30:00	121.01	122.44	117.12	122.21	140

Järgnevalt selgitan Tabel 4 veerge.

- Alguse ajatempel – tähistab ajaperioodi algust.
- Lõpu ajatempel – tähistab ajaperioodi lõppu.
- Avamishind – tähistab ajaperioodi alguses olevat hinda, kusjuures eelmise ajaperioodi lõpphind on järgmise ajaperioodi avamishinnaks.
- Kõrgeim hind – tähistab ajaperioodi jooksul väärtpaberi kõrgeimat hinda.
- Madalaim hind – tähistab ajaperioodi jooksul väärtpaberi madalaimat hinda.
- Lõpphind – tähistab ajaperioodi lõpus olevat hinda.

- Kogus – tähistab ajaperioodi jooksul ostetud/müüdud väärtpaberite kogust.

Veergude kirjeldusest võib järeldada, et teades intervalli, millise vahega andmeid küsitakse, on võimalik rakenduse poolel tuletada alguse ajatempel. Lisaks on rakenduse poolel võimalik tuletada ka avamishind, sest tegu on eelmise perioodi lõpphinnaga. Kuna tuletavad veerud on lihtsasti leitavad rakenduse poolel, siis leian, et andmebaasist võiks pärida ainult vajalikku informatsiooni, et vähendada päringu keerukust ning andmebaasisüsteemilt oodatavat tööd. Tabel 5 esitab näite andmetest, mida oleks vaja küünlagraafi loomiseks andmebaasist küsida.

Tabel 5. Andmebaasist päritavad andmed küünlagraafi loomiseks.

Lõpu ajatempel	**Kõrgeim hind**	**Madalaim hind**	**Lõpphind**	**Kogus**
29.03.2019 11:15:00	122.23	119.87	121.01	100
29.03.2019 11:30:00	122.44	117.12	122.21	140

Kasutaja portfellis olevate väärtpaberite väärtused mingil ajahetkel

Rakenduse teiseks eesmärgiks on kasutajale kuvada kokkuvõte tema investeeringutest ning nende väärtustest praegusel ajahetkel. Oletame, et kasutaja investeeringud praegusel ajahetkel on kujutatud Tabel 3. Selleks, et leida kasutaja portfelli koguväärtus ning tootlikus (ehk kui palju on väärtus tõusnud või langenud) on vaja leida portfellis olevate aktsiate väärtused. Tabel 6 esitab vaate loomiseks vajaliku päringu näite.

Tabel 6. Kasutaja portfellis olevate väärtpaberite väärtuste päringu tulemus.

Aktsiat tähistav sümbol	**Viimane teadaolev hind**	**Viimase teadaoleva hinna ajatempel**
GOOG	1 200.00 €	29.03.2019 11:30:00
FB	150.00 €	29.03.2019 11:30:00

Sellise päringu tulemuse põhjal on võimalik rakenduses kuvada nii iga individuaalse aktsia kui ka kogu portfelli tootlikust. Tootlikuse arvutamise eelduseks on, et kasutaja on selle informatsiooni varem rakendusse sisestanud.

Autori kogemuse põhjal on nende kahe päringuga võimalik kokku võtta enamus portfelli statistika loomiseks vajalikud andmed ning seega sobivad ka hästi erinevate disainilahenduste testimiseks.

3 Varasemad uuringud

Käesolevas peatükis antakse ülevaade varem aegridade SQL-andmebaasides esitamise kohta tehtud töödest. Tuuakse välja artiklid, raamatud ja teadustööd, kus on toodud välja mõni aegridade SQL-andmebaasides esitamiseks sobilik disainilahendus. Tööde leidmiseks kasutatud Google Scholari ja Google otsingumootori otsingusõnade kombinatsioonid on välja toodud Lisas 1.

Abel Matus Castillejos [30] toob välja erinevaid disainilahendusi varasematest töödest ning pakub välja uue disainilahenduse, et optimeerida kettaruumi kasutust ning päringute tegemiseks kuluvat aega relatsioonilistes andmebaasisüsteemides. Peatükis 4.2.1 kirjeldatakse disainilahendust „iga aegrea kohta üks relatsiooniline tabel“ (*modelling one time series per relational table*), kus iga erineva aegrea esitamiseks luuakse uus tabel. Autor toob välja ka disainilahenduse nõrga koha – tabelite arv võib kasvada liiga suureks ning haldamatuks. Lisaks võib piirajaks olla andmebaasisüsteemi maksimaalne lubatud tabelite arv või siis see, et tulenevalt andmebaasisüsteemi sisemisest ülesehituses võivad teatud tabelite hulgast alates muutuda andmebaasiga tehtavad operatsioonid liiga aeglaseks [31]. Sellist disainilahendust kutsutakse antimustriks (*antipattern*), mis pealtnäha lahendab probleemi, kuid tegelikkuses tekitab neid juurde [32]. Karwin [32] kirjeldab seda antimustrit nime all *Metadata Tribbles* (metaandmete vohamine) [32]. Peatükis 4.2.2 kirjeldatakse disainilahendust, kus iga aegreatüübi kohta luuakse uus tabel. Aegreatüübi all mõeldakse ajaperioodi (uus tabel luuakse päeva, nädala ja kuu jaoks). Peatükis tuuakse ka näide sellise disainilahenduse realiseerimisest finantsandmete näitel. Peatükis 4.2.3 toob autor välja aegridade päiste (*header*) või metaandmete (*metadata*) eraldamise teise tabelisse, et vähendada andmete dubleeritust. Peatükis 4.3 kirjeldatakse autori poolt välja pakutud disainilahendusi nii ühe kui ka mitme muutujaga aegridade esitamiseks. Autori eksperimendi käigus kasutakse välja pakutud disainilahendusi finantsandmete esitamiseks ning autor leidis, et kettaruumikasutus vähenes 55% võrreldes peatükis 4.2.1 kirjeldatud disainilahendusega.

Artiklites [33], [34] ja [35] toob Gregory Trubetskoy välja erinevaid disainilahendusi aegridade esitamiseks SQL- andmebaasis PostgreSQL näitel. Esimeses artiklis [33] toob

artikli autor välja esiteks naiivse meetodi (iga ajatempli kohta luuakse uus rida) ning sellega seotud probleemid ja teiseks järjestikplaanur meetodi (*round robin*), mis on ringikujuline struktuur koos eraldi salvestatud viitega viimasele elemendile ning selle ajale. Järjestikplaanur meetodis toob artikli autor välja kolm erinevat viisi, kuidas seda PostgreSQLis realiseerida. Teises artiklis [34] seletatakse, kuidas järestikplaanur meetodile luua vaade (*view*), mis võimaldaks kirjutada andmete otsimiseks lihtsamaid päringuid. Kolmandas artiklis tuuakse välja eelnevates artiklites pakutud meetodi võimalik nõrk koht – ridade lisamise kiirus. Artikli autor pakub välja uue disainilahenduse, millega paraneb ridade lisamise kiirus, kuid selle arvelt kannatab üksikute ajahetkede (või -perioodide) otsimise kiirus.

Artiklites [36] ja [37] kirjeldab Edmund Horner kahte disainilahendust aegridade esitamiseks SQL-andmebaasis PostgreSQL näitel. Esimeses artiklis [36] viib autor läbi eksperimendid (päringute kiiruste ja tabeli suuruste mõõtmised) enda kogutud riistvara sensorite andmete (4.8 miljonit mõõtmist) peal. Algselt realiseerib artikli autor naiivse lahenduse (iga ajatempli kohta lisatakse uus rida) ning viib läbi mõõtmised sellel disainil. Seejärel pakub artikli autor välja nii horisontaalselt kui ka vertikaalselt vektoriseeritud disainlahenduse. Naiivse lahenduse puhul leiab artikli autor, et tabeli suurus (koos indeksitega) on 508MB, kuid horisontaalselt vektoriseeritud lahenduse puhul vaid 50MB ja vertikaalselt vektoriseeritud lahenduse puhul 90MB, mis aitab andmeid lihtsamini mällu mahutada ning seega tõstab andmete otsimise kiirust. Teises artiklis [37], sarnaselt artiklile [34], pakutakse välja luua vaade (*view*) eelnevalt mainitud lahendustele, viiakse läbi töökiiruse eksperimendid ning jagatakse vaate loomiseks vajalike käske.

Artiklis [38] tuuakse valja BRIN (plokivahemike indeks, *Block-Range Index*) indeksite kasutamise eeliseid PostgreSQL andmebaasis, kiirendamaks aegridade põhjal tehtavaid otsinguid ja hoidmaks kokku salvestusruumi.

BRIN indeksi plokid sisaldavad kokkuvõtteinfot plokkide vahemike kohta, milles on indekseeritavas veerus olevad väärtused. Näiteks võib indeksi plokis olla informatsioon, et tabeli plokkides 50–90 on veeru *date* väärtused 01.01.2019 00:00:00 kuni 01.01.2019 00:15:00. BRIN indeks võimaldab andmebaasisüsteemil kiiresti leida plokid, mis võivad sisaldada huvipakkuvaid andmeid ja välistada otsingust plokid, mis neid andmeid kindlasti ei sisalda. BRIN indeks töötab hästi, kui andmed indekseeritud veerus on kettal

sorteeritud, mitte ei paikne plokkides läbisegi. Kuna aegridade korral saabuvad andmed loomulikul viisil ajalises järjestuses, siis on see nõue täidetud.

BRIN indeks on 2019. aasta märtsi seisuga saadaval ainult PostgreSQLis (alates versioonist 9.5), kuid plaanis on see lisada ka teistel andmebaasisüsteemidel (näiteks Oracle) [39]. Artikkel [38] väidab, et 31.5 miljoni ajatempli (*timestamp*) indekseerimisel B-puu indeksiga kasutatakse kettaruumi 676MB, kuid BRIN indeksi puhul vaid 72KB. Lisaks kettaruumi säästmisele väidab autor, et paranesid ka andmete otsingu (ühe päeva keskmise temperatuuri leidmine) tegemise kiirused.

Väljatoodud teadustöödes ja artiklites ei ole võrreldud kõiki disainilahendusi omavahel, vaid on pakutud uus disainilahendus ning seda võrreldud naiivse lahendusega (iga ajatempli jaoks uus rida). Väljatoodud töödes puudub ka disainilahenduste esitamine ühtset mustri formaati kasutades. Lisaks ei kasutata disainilahenduste realiseerimisel praeguseks saadaolevaid aegridadele sobilikemaid indeksitüüpe (näiteks BRIN indeks).

4 Erinevad disainilahendused aegridade esitamiseks SQL-andmebaasides

Aegridade SQL-andmebaasides esitamist puudutavates töödes on järgnevaid puudujääke, mida soovin käesoleva tööga parandada.

- Aegridade esitamist puudutavates töödes (peatükk 3) ei ole koondatud ühte allikasse kokku kõiki disainlahendusi.

- Allikad ei esita disainilahendusi ühtses mustri formaadis. Muster on struktuurse kirjutamise viis, mis võimaldab esitada häid või halbu disainilahendusi. Mustri formaadi kasutamine parandab disainilahenduste kirjelduse loetavust ning tänu ühesugusele struktuurile võimaldab lugejal erinevaid lahendusi paremini võrrelda. Disainilahenduste nimed esitavad sõnavara, mida kasutades saavad valdkonnast huvitunud disainilahendustele viidata. Selleks, et disainilahendust saaks nimetada mustriks, peaks leiduma vähemalt kolm näidet selle lahenduse kasutamise kohta. Kuna minu pakutavate disainilahenduste puhul see alati nii ei ole, siis kasutan mõistet „disainilahendus“.

Kõigi eelnevalt väljatoodud probleemide lahendamisele aitab kaasa disainilahenduste koondamine ühte kataloogi ja seal nende mustri formaadis esitamine. Mustri formaadi komponentide valimisel on eeskujuks võetud magistritöö [40].

Formaadi kohaselt tuuakse iga disainilahenduse korral välja järgnevad alampunktid.

Nimi eesti keeles: eestikeelne nimi, mis on enamasti tuletatud ingliskeelsest nimest.

Nimi inglise keeles: ingliskeelne nimi, mis on enamasti võetud lähteallikast.

Viited allikatele: viited kõigile allikatele, kus kirjeldatakse disainilahenduse kasutamist aegridade esitamiseks.

Kirjeldus: selgitatakse disainilahenduse põhimõtet ning tuuakse välja aegread, mida selle disainilahendusega esitada saab.

Tugevad küljed: selle disainilahenduse tugevad küljed allikate põhjal

Nõrgad küljed: selle disainilahenduse nõrgad küljed allikate põhjal.

Näide: iga disainilahenduse puhul tuuakse näide andmemudelist, kuidas seda disainilahendust kasutada OHLCV andmete aegridade esitamiseks. Andmete struktuuri on kirjeldatud jaotises 2.3. Lisaks tuuakse välja tabelite veergude kirjeldused, tabelite ning nendega seotud kitsenduste loomise laused ja tabelid näiteväärtustega. Kõikide disainide puhul kasutatakse võrreldavuse huvides näiteväärtustena ühesugust aegrida (Tabel 7). Näide põhineb andmebaasisüsteemil PostgreSQL (11).

Tabel 7. Disainilahenduste tabelite näiteväärtused.

ticker	**date**	**open**	**high**	**low**	**close**	**volume**
GOOGL	01.01.2019 00:00:00	100.10	103.45	100.04	102.30	1200
GOOGL	01.01.2019 00:01:00	102.30	105.33	101.90	104.22	1322
GOOGL	01.01.2019 00:02:00	104.22	104.67	103.14	104.11	997
FB	01.01.2019 00:00:00	130.10	130.38	129.38	130.03	1018
FB	01.01.2019 00:01:00	130.03	131.01	129.44	129.88	921
FB	01.01.2019 00:02:00	129.88	130.44	129.22	129.92	1100
AMZN	01.01.2019 00:00:00	1410.15	1412.55	1409.34	1409.39	200
AMZN	01.01.2019 00:01:00	1409.39	1410.17	1406.22	1406.22	190
AMZN	01.01.2019 00:02:00	1406.22	1409.33	1403.21	1405.38	102

4.1 Naiivne meetod

Nimi eesti keeles: naiivne meetod

Nimi inglise keeles: naive method

Viited allikatele: [33], [36]

Kirjeldus: Disainilahenduses pannakse kõigi aegridade andmed ühte tabelisse. Selleks, et eristada erinevaid aegridu, lisatakse tabelile uusi veerge (näiteks aktsia sümbol või temperatuuri mõõtmisel asukoht). Disainilahendus koosneb ajatempli veerust, erinevate aegridade identifitseerimiseks vajalikest veergudest ning väärtuste veergudest.

Tugevad küljed:

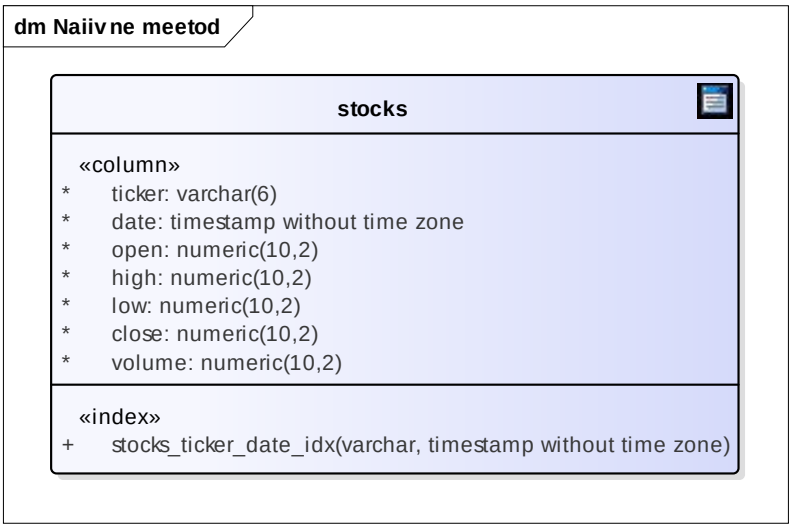
- Selge struktuur.
- Andmete lisamise, kustutamise, uuendamise ja leidmise päringud on loetavad ja lihtsad.

Nõrgad küljed:

- Uue veeru lisamisel tekib palju NULLe. Lisatavad veerud peavad olema mittekohustuslikud (lubama NULLe).
- Ridades on palju korduvaid väärtuseid, mis tähendab lisa kettaruumi ja muutmälu kasutamist. Korduvateks väärtusteks võivad olla aegridasid identifitseerivad või/ja ajatempli väärtused.
- Tabelis olevate ridade arv kasvab väga suureks, mis võib omakorda mõjutada tabelil tehtavate päringute kiirust, sest tabel ei mahu enam muutmällu.

Näide:

Luuakse tabel nimega s*tocks*, kus hoiustatakse börsil kaubeldavate aktsiate hinna muutused. Näide füüsilise disaini andmemudelist OHLCV andmete esitamiseks on Joonis 3, veergude kirjeldused on Tabel 8 ning näide tabelist näite andmetega on Tabel 9. Tabeli loomise lause on Joonis 4.



Joonis 3. Füüsilise disaini andmemudel disainilahendusele „Naiivne meetod“.

Tabel 8. Disainilahenduse „Naiivne meetod“ tabeli *stocks* veergude kirjeldus.

Veerunimi	**Kirjeldus**
ticker	finantsinstrumendi sümbol
date	intervalli lõpu ajatempel
open	intervalli algushind
high	intervalli kõrgeim hind
low	intervalli madalaim hind
close	intervalli lõpphind
volume	intervalli jooksul ostetud/müüdud kogus

Tabel 9. Disainilahenduse „Naiivne meetod“ tabel *stocks* koos näiteväärtustega.

ticker	**date**	**open**	**high**	**low**	**close**	**volume**
GOOGL	01.01.2019 00:00:00	100.10	103.45	100.04	102.30	1200
GOOGL	01.01.2019 00:01:00	102.30	105.33	101.90	104.22	1322
GOOGL	01.01.2019 00:02:00	104.22	104.67	103.14	104.11	997
FB	01.01.2019 00:00:00	130.10	130.38	129.38	130.03	1018
FB	01.01.2019 00:01:00	130.03	131.01	129.44	129.88	921
FB	01.01.2019 00:02:00	129.88	130.44	129.22	129.92	1100

ticker	date	open	high	low	close	volume
AMZN	01.01.2019 00:00:00	1410.15	1412.55	1409.34	1409.39	200
AMZN	01.01.2019 00:01:00	1409.39	1410.17	1406.22	1406.22	190
AMZN	01.01.2019 00:02:00	1406.22	1409.33	1403.21	1405.38	102

Märkus: Tabelis s*tocks* ei tohiks olla samal finantsinstumendil korduvaid (ehk sama ajatempliga) ridu. Andmebaasi kavandamisel lähtusin sellest, et kui lisada tabelisse PRIMARY KEY või UNIQUE kitsendus (ticker, date) (millega kaasneb süsteemi poolt automaatselt loodav B-puu indeks nendele veergudele), siis kaotab nendele veergudele võimalikult loodav BRIN indeks oma väärtuse täielikult – tõenäoliselt päringu tegemisel kasutatakse hoopis B-puu indeksit ning tabeli ning indeksite maht läheb ka väga suureks, mille vältimine oli BRIN indeksi põhiline eelis. Allikas [33] ei deklareerita sellises tabelis primaarvõtit, allikas [36] deklareeritakse. Tabelisse veeru stocks_id BIGSERIAL lisamine ning primaarvõtme (stocks_id) deklareerimine ei tagaks seda, et (ticker, date) kombinatsioon oleks unikaalne. Küll aga suureneks tabeli andmemaht märgatavalt, sest iga bigint tüüpi väärtuse salvestamiseks kulub kaheksa baiti ning sellele lisandub veel indeksist tulenev andmemaht. Eelnevast lähtuvalt otsustasin minna antud juhul vastuollu üldise hea disaini praktikaga ning tabelisse mitte ühtegi võtit deklareerida. Tabeli *stocks* indeksi loomise laused erinevate indeksitüüpide korral on esitatud Lisa 6.

```
CREATE TABLE stocks
(
	ticker varchar(6) NOT NULL,
	date timestamp without time zone NOT NULL,
	open numeric(10,2) NOT NULL,
	high numeric(10,2) NOT NULL,
	low numeric(10,2) NOT NULL,
	close numeric(10,2) NOT NULL,
	volume numeric(10,2) NOT NULL
);
```

Joonis 4. Naiivse meetodi tabeli loomise lause.

4.2 Iga aegrea kohta üks tabel

Nimi eesti keeles: iga aegrea kohta üks tabel

Nimi inglise keeles: one relational table per time series

Viited allikatele: [30] peatükk 4.2.1

Kirjeldus: Disainilahenduses luuakse iga aegrea kohta üks tabel. Näiteks, kui eesmärgiks on hoiustada kümne erineva aegrea väärtuseid, siis luuakse kümme erinevat tabelit. Juhul kui andmebaasisüsteem toetab tabelite loomise kaudu, siis on võimalik luua ka ülatabel, mille kaudu saab otsida andmeid kõigist alamtabelitest korraga.

Tugevad küljed:

- Vähem andmeid ühes tabelis.
- Selge struktuur.
- Andmete lisamise, kustutamise, uuendamise ja leidmise laused on loetavad ja lihtsad.

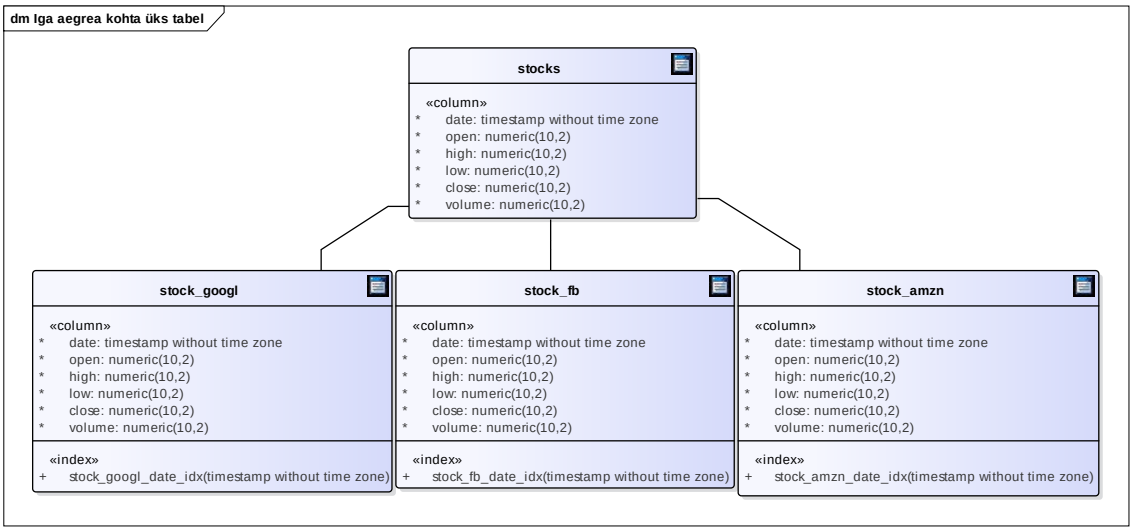
Nõrgad küljed:

- Aegridade lisamiseks või kustutamiseks tuleb muuta andmebaasi skeemi, mis ei ole hea andmebaasi kavandamise praktika.
- Maksimaalne tabelite arv andmebaasisüsteemis võib olla limiteeritud, mis omakorda piirab erinevate võimalike aegridade arvu.
- Tulenevalt andmebaasisüsteemi sisemisest ülesehituses võivad teatud tabelite hulgast alates muutuda andmebaasiga tehtavad operatsioonid liiga aeglaseks [31].

Näide:

Luuakse ülatabel s*tocks* (loomise lause on Joonis 6) ning alamtabelid s*tock_googl*, s*tock_fb*, *stock_amzn* (loomise laused on Joonis 7), mida kasutatakse aktsiate hinna muutuste salvestamiseks. Alamtabelite loomiseks kasutatakse PostgreSQLi pakutavat pärimise kaudu tabelite loomise võimalust. Aegridade andmed lisatakse alamtabelitesse.

Tabeli *stocks* kaudu on võimalik küsida andmeid kõigist alamtabelitest korraga. Näide füüsilise disaini andmemudelist on toodud Joonis 5 ja veergude kirjeldus on toodud Tabel 10. Alamtabelid *stock_fb*, *stock_amzn* ja *stock_googl* on näiteandmetega esitatatud vastavalt Tabel 11, Tabel 12 ja Tabel 13.



Joonis 5. Füüsilise disaini andmemudel disainilahendusele „Iga aegrea kohta üks tabel“.

Kaalutlused, millest tulenevalt ei deklareerita tabelites PRIMARY KEY või UNIQUE kitsendust on samasugused nagu kirjeldati disainilahenduses „naiivne meetod“ (jaotis 4.1). Indeksi loomise laused alamtabelitele erinevate indeksitüüpide korral on esitatud Lisa 6.

Tabel 10. Disainilahenduse „Iga aegrea kohta üks tabel“ tabelite veergude kirjeldus.

Veerunimi	**Kirjeldus**
date	intervalli lõpu ajatempel
open	intervalli algushind
high	intervalli kõrgeim hind
low	intervalli madalaim hind
close	intervalli lõpphind
volume	intervalli jooksul ostetud/müüdud kogus

Tabel 11. Disainilahenduse „Iga aegrea kohta üks tabel“ tabel *stock_googl* koos näiteväärtustega.

date	**open**	**high**	**low**	**close**	**volume**
01.01.2019 00:00:00	100.10	103.45	100.04	102.30	1200

date	open	high	low	close	volume
01.01.2019 00:01:00	102.30	105.33	101.90	104.22	1322
01.01.2019 00:02:00	104.22	104.67	103.14	104.11	997

Tabel 12. Disainilahenduse „Iga aegrea kohta üks tabel“ tabel *stock_fb* koos näiteväärtustega.

date	open	high	low	close	volume
01.01.2019 00:00:00	130.10	130.38	129.38	130.03	1018
01.01.2019 00:01:00	130.03	131.01	129.44	129.88	921
01.01.2019 00:02:00	129.88	130.44	129.22	129.92	1100

Tabel 13. Disainilahenduse „Iga aegrea kohta üks tabel“ tabel *stock_amzn* koos näiteväärtustega.

date	open	high	low	close	volume
01.01.2019 00:00:00	1410.15	1412.55	1409.34	1409.39	200
01.01.2019 00:01:00	1409.39	1410.17	1406.22	1406.22	190
01.01.2019 00:02:00	1406.22	1409.33	1403.21	1405.38	102

```
CREATE TABLE stocks
(
	date timestamp without time zone NOT NULL,
	open numeric(10,2) NOT NULL,
	high numeric(10,2) NOT NULL,
	low numeric(10,2) NOT NULL,
	close numeric(10,2) NOT NULL,
	volume numeric(10,2) NOT NULL
);
```

Joonis 6. Disainilahenduse „Iga aegrea kohta üks tabel“ ülatabeli loomise lause.

```
CREATE TABLE stock_googl() INHERITS (stocks);
CREATE TABLE stock_fb() INHERITS (stocks);
CREATE TABLE stock_amzn() INHERITS (stocks);
```

Joonis 7. Disainilahenduse „Iga aegrea kohta üks tabel“ alamtabelite loomise laused.

4.3 Metaandmete eraldamine aegridadest

Nimi eesti keeles: metaandmete eraldamine aegridadest

Nimi inglise keeles: separating metadata from time series

Viited allikatele: [30] jaotis 4.2.3

Kirjeldus: Disainilahenduse eesmärgiks on vähendada ridades andmete dubleeritust viies metaandmed eraldi tabelitesse. Disainilahenduse aluseks võib pidada naiivset meetodit, kust eraldatakse korduvad andmed. Teiste sõnadega suurendakse tabelite normaliseerituse astet ja sellega seoses vähendatakse andmete liiasust.

Tugevad küljed:

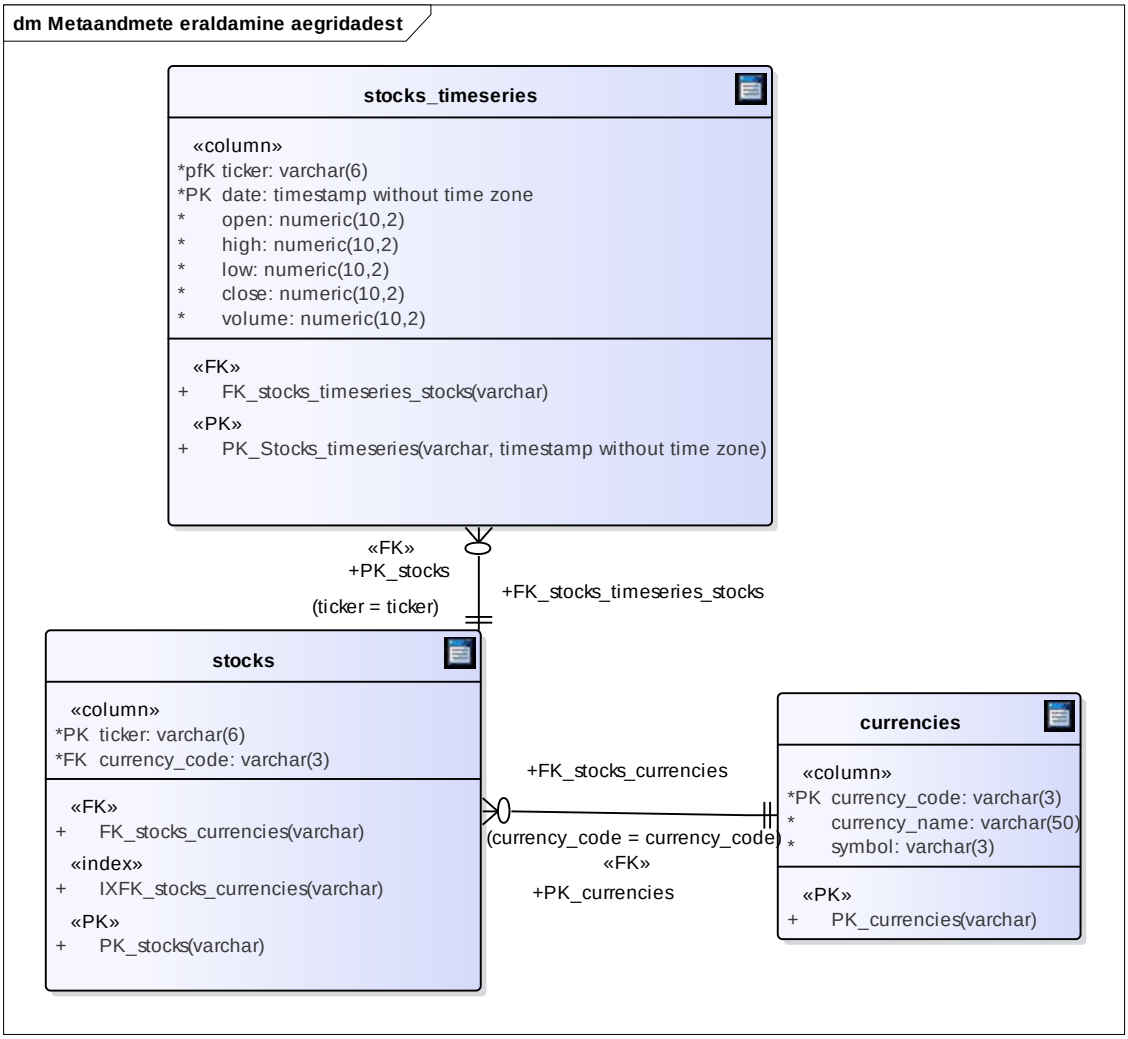
- Selge struktuur.
- Andmete lisamise, kustutamise, uuendamise ja leidmise laused on loetavad ja lihtsad.
- Seda saab kasutada kombinatsioonis teiste (ka antud kataloogis väljatoodud) disainidega.

Nõrgad küljed:

- Tabelis olevate ridade arv kasvab väga suureks, mis võib omakorda mõjutada tabelil tehtavate päringute kiirust.

Näide:

Luuakse tabelid nimega ja *currencies* (loomise laused Joonis 9), *stocks* (loomise laused Joonis 10) ja *stocks_timeseries* (loomise laused Joonis 11). s*tocks* sisaldab informatsiooni jälgitavate aktsiate kohta, *currencies* on klassifikaatori tabel ning *stock_timeseries* sisaldab informatsiooni aktsiate hinnaliikumiste kohta. Disainilahenduse füüsilise disaini andmemudel on välja toodud Joonis 8, tabelite veergude kirjeldused on välja toodud Tabel 14, Tabel 15 ja Tabel 16, tabelite näiteväärtustega on välja toodud Tabel 17, Tabel 18 ja Tabel 19.



Joonis 8. Füüsilise disaini andmemudel disainilahendusele „Metaandmete eraldamine aegridadest“.

Tabel 14. Disainilahenduse „Metaandmete eraldamine aegridadest“ tabeli *currencies* veergude kirjeldus.

Veerunimi	**Kirjeldus**
currency_code	valuuta ISO 4217 kood, unikaalne identifikaator
currency_name	valuuta nimi
symbol	valuuta sümbol

Tabel 15. Disainilahenduse „Metaandmete eraldamine aegridadest“ tabeli s*tocks* veergude kirjeldus.

Veerunimi	**Kirjeldus**
ticker	unikaalne identifikaator, finantsinstrumendi sümbol
currency_code	rahaühiku lühend, tabeli *currencies* unikaalne identifikaator

Tabel 16. Disainilahenduse „Metaandmete eraldamine aegridadest“ tabeli *stocks_timeseries* veergude kirjeldus.

Veerunimi	**Kirjeldus**
ticker	tabeli *stocks* unikaalne identifikaator
date	intervalli lõpu ajatempel
open	intervalli algushind
high	intervalli kõrgeim hind
low	intervalli madalaim hind
close	intervalli lõpphind
volume	intervalli jooksul ostetud/müüdud kogus

Tabel 17. Disainilahenduse „Metaandmete eraldamine aegridadest“ tabel *currencies* koos näideväärtustega.

currency_code	**currency_name**	**symbol**
USD	United States dollar	$

Tabel 18. Disainilahenduse „Metaandmete eraldamine aegridadest“ tabel *stocks* koos näideväärtustega.

ticker	**currency_code**
GOOGL	USD
FB	USD
AMZN	USD

Tabel 19. Disainilahenduse „Metaandmete eraldamine aegridadest“ tabel *stocks_timeseries* koos näideväärtustega.

ticker	**date**	**open**	**high**	**low**	**close**	**volume**
GOOGL	01.01.2019 00:00:00	100.10	103.45	100.04	102.30	1200
GOOGL	01.01.2019 00:01:00	102.30	105.33	101.90	104.22	1322
GOOGL	01.01.2019 00:02:00	104.22	104.67	103.14	104.11	997
FB	01.01.2019 00:00:00	130.10	130.38	129.38	130.03	1018
FB	01.01.2019 00:01:00	130.03	131.01	129.44	129.88	921
FB	01.01.2019 00:02:00	129.88	130.44	129.22	129.92	1100

ticker	date	open	high	low	close	volume
AMZN	01.01.2019 00:00:00	1410.15	1412.55	1409.34	1409.39	200
AMZN	01.01.2019 00:01:00	1409.39	1410.17	1406.22	1406.22	190
AMZN	01.01.2019 00:02:00	1406.22	1409.33	1403.21	1405.38	102

Tabeli *currencies* loomise laused:

```
CREATE TABLE currencies
(
	currency_code varchar(3) NOT NULL,
	currency_name varchar(50) NOT NULL,
	symbol varchar(3) NOT NULL
);

ALTER TABLE currencies ADD CONSTRAINT PK_currencies
	PRIMARY KEY (currency_code);
```

Joonis 9. Disainilahenduse „Metaandmete eraldamine aegridadest“ tabeli *currencies* realiseerimise laused.

Tabeli s*tocks* loomise laused:

```
CREATE TABLE stocks
(
	ticker varchar(6) NOT NULL,
	currency varchar(3) NOT NULL
);

ALTER TABLE stocks ADD CONSTRAINT PK_stocks
	PRIMARY KEY (ticker);

CREATE INDEX IXFK_stocks_currencies ON stocks (currency_code ASC);

ALTER TABLE stocks ADD CONSTRAINT FK_stocks_currencies
	FOREIGN KEY (currency_code) REFERENCES currencies
(currency_code) ON DELETE No Action ON UPDATE CASCADE;
```

Joonis 10. Disainilahenduse „Metaandmete eraldamine aegridadest“ tabeli s*tocks* realiseerimise laused.

Tabeli s*tocks_timeseries* loomise laused:

```
CREATE TABLE stocks_timeseries
(
	ticker varchar(6) NOT NULL,
	date timestamp without time zone NOT NULL,
	open numeric(10,2) NOT NULL,
	high numeric(10,2) NOT NULL,
	low numeric(10,2) NOT NULL,
	close numeric(10,2) NOT NULL,
	volume numeric(10,2) NOT NULL
);

ALTER TABLE stocks_timeseries ADD CONSTRAINT PK_Stocks_timeseries
	PRIMARY KEY (ticker, date);

ALTER TABLE stocks_timeseries ADD CONSTRAINT FK_stocks_timeseries_stocks
	FOREIGN KEY (ticker) REFERENCES stocks (ticker) ON DELETE CASCADE ON
UPDATE CASCADE;
```

Joonis 11. Disainilahenduse „Metaandmete eraldamine aegridadest“ tabeli s*tocks_timeseries* realiseerimise laused.

4.4 Aegridade segmenteerimine

Nimi eesti keeles: aegridade segmenteerimine

Nimi inglise keeles: segmenting time-series

Viited allikatele: [30] peatükk 4.3

Kirjeldus: Disainilahenduse eesmärgiks on vähendada ridade arvu tabelis, segmenteerides muidu ridades hoitavad aegrea elementide väärtused veergudeks.

Tugevad küljed:

- Võimalik hoiustada erineva intervalliga andmeid (näiteks minut, päev, tund).
- Lihtne lisada uusi aegridasid.
- Tabelid kasutavad vähe kettaruumi, sest ajatempli salvestamise asemel see tuletatakse aegrea metaandmetest.

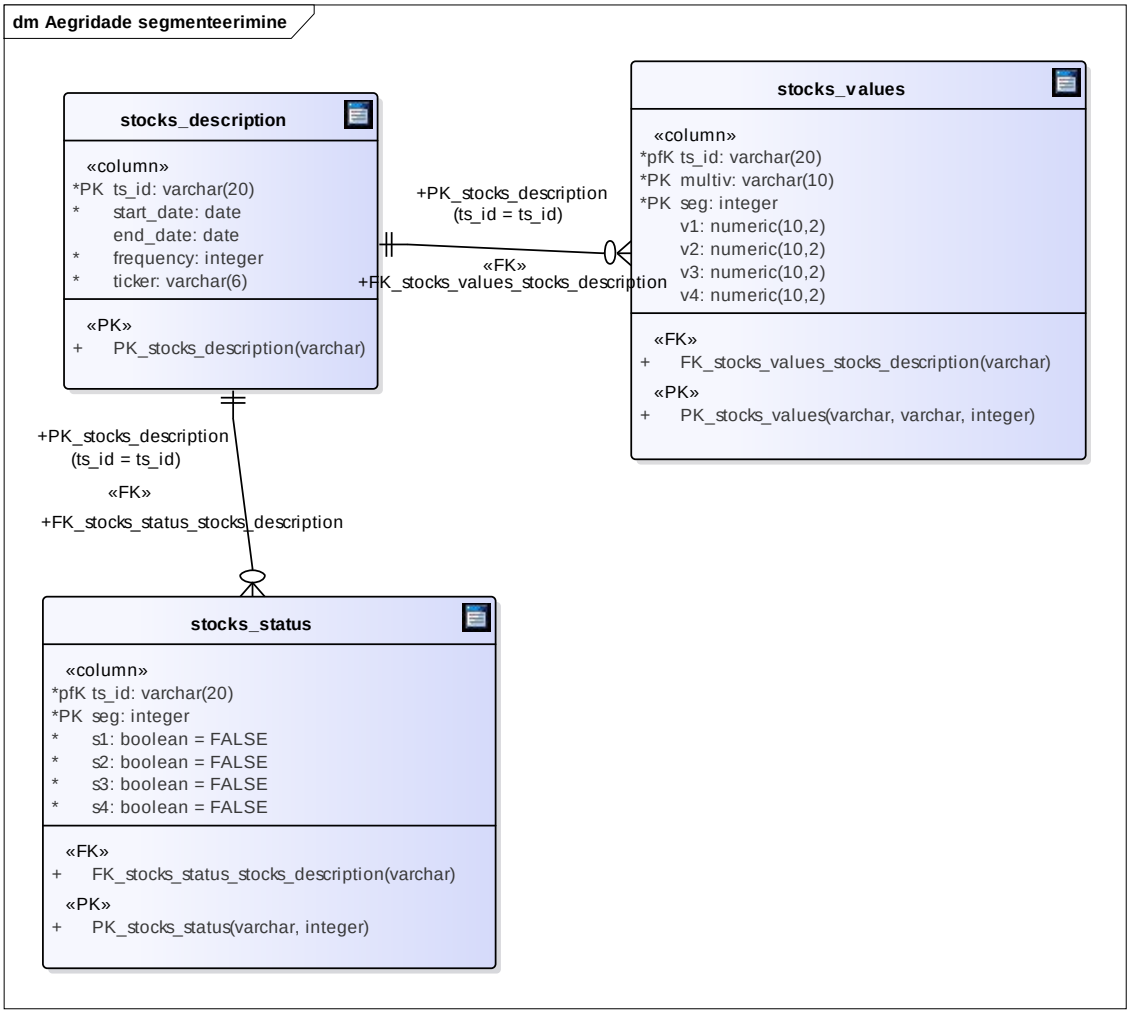
Nõrgad küljed:

- Keeruline struktuur.
- Andmete lisamise, kustutamise, uuendamise ja leidmise laused on rasketi loetavad.
- Tabeli maksimaalne veergude arv seab piirangu sellele, kui mitu aegrea elementi saab ühes segmendis registreerida.

Näide:

Luuakse tabelid *stocks_description* (loomise laused Joonis 13), s*tocks_values* (loomise laused Joonis 14) ja *stocks_status* (loomise laused Joonis 15). s*tocks_description* tabelis salvestatakse aegridade metaandmed, näiteks aktsiate puhul sümbol. Iga rida tabelis *stocks_description* tähendab ühte aegrida. Tabelis s*tocks_values* salvestatakse aegrea elementide omaduste väärtused. Tabelis s*tocks_status* salvestatakse informatsioon omaduste väärtuste olemasolu kohta. Disainilahenduse füüsilise disaini andmemudel on

Joonis 12, tabeli veergude kirjeldused on tabelites Tabel 20, Tabel 21 ja Tabel 22, tabelid näiteväärtustega on tabelites Tabel 23, Tabel 24 ja Tabel 25.



Joonis 12. Füüsilise disaini andmemudel disainilahendusele „Aegridade segmenteerimine“.

Tabel 20. Disainilahenduse „Aegridade segmenteerimine“ tabeli *stocks_description* veergude kirjeldus.

Veerunimi	**Kirjeldus**
ts_id	aegrea unikaalne identifikaator
start_date	aegrea esimese elemendi ajatempel
end_date	aegrea viimase elemendi ajatempel
frequency	intervall aegrea elementide vahel
ticker	finantsinstrumendi sümbol

Tabel 21. Disainilahenduse „Aegridade segmenteerimine“ tabeli s*tocks_values* veergude kirjeldus.

Veerunimi	**Kirjeldus**
ts_id	aegrea unikaalne identifikaator
multiv	aegrea elemendi omaduse e atribuudi nimetus
seg	segmendi number
v1, ..., v4	aegrea elemendi omaduste väärtused

Tabel 22. Disainilahenduse „Aegridade segmenteerimine“ tabeli s*tocks_status* veergude kirjeldus.

Veerunimi	**Kirjeldus**
ts_id	aegrea unikaalne identifikaator
seg	segmendi number
s1, ..., s4	tõeväärtused aegrea elementide omaduste väärtuste olemasolu kohta

Tabel 23. Disainilahenduse „Aegridade segmenteerimine“ tabel s*tocks_description* koos näiteväärtustega.

ts_id	**start_date**	**end_date**	**frequency**	**ticker**
1	01.01.2019 00:00:00	NULL	60	GOOGL
2	01.01.2019 00:00:00	NULL	60	FB
3	01.01.2019 00:00:00	NULL	60	AMZN

Tabel 24. Disainilahenduse „Aegridade segmenteerimine“ tabel s*tocks_values* koos näiteväärtustega.

ts_id	**multiv**	**seg**	**v1**	**v2**	**v3**	**v4**
1	open	1	100.10	102.30	104.22	NULL
1	high	1	103.45	105.33	104.67	NULL
1	low	1	100.04	101.90	103.14	NULL
1	close	1	102.30	104.22	104.11	NULL
1	volume	1	1200	1322	997	NULL
2	open	1	130.10	130.03	129.88	NULL
2	high	1	130.38	131.01	130.44	NULL
2	low	1	129.38	129.44	129.22	NULL
2	close	1	130.03	129.88	129.92	NULL
2	volume	1	1018	921	1100	NULL
3	open	1	1410.15	1409.39	1406.22	NULL

ts_id	multiv	seg	v1	v2	v3	v4
3	high	1	1412.55	1410.17	1409.33	NULL
3	low	1	1409.34	1406.22	1403.21	NULL
3	close	1	1409.39	1406.22	1405.38	NULL
3	volume	1	200	190	102	NULL

Tabel 25. Disainilahenduse „Aegridade segmenteerimine“ tabel s*tocks_status* koos näiteväärtustega.

ts_id	seg	s1	s2	s3	s4
1	1	TRUE	TRUE	TRUE	FALSE
2	1	TRUE	TRUE	TRUE	FALSE
3	1	TRUE	TRUE	TRUE	FALSE

Aegrea segmentide arv sõltub selle elementide arvust. Erinevatel aegridadel võib olla erinev hulk elemente ja seega ka erinev segmentide hulk. Antud näites on aegridades nii vähe elemente, et iga aegrea andmed mahuvad ühte segmenti. Igale aegrea segmendile vastab üks rida tabelis *stocks_status* ning üks või rohkem rida tabelis *stocks_values*. Ühele segmendile vastavate ridade arv tabelis *stocks_values* sõltub aegrea elemendi omaduste arvust – veerg *multiv*. Näites on selleks viis (omadused: *open*, *high*, *low*, *close*, *volume*).

Iga segment hoiab andmeid ühe või rohkema aegrea elemendi kohta. Näites on igas segmendis võimalik salvestada kuni nelja aegrea elemendi omaduste väärtused (v1, ..., v4 ja s1, ..., s4). Päriselt võiks segmendis hoida rohkemate või ka vähemate aegrea elementide omaduste väärtuseid – vastavalt sellele muutub ka tabeli veergude hulk.

Hetkel võib näitest näha, et tabelis *stocks_status* on kõigi aegridade segmentide (ts_id. seg) puhul s4 väärtused FALSE (lisaks puuduvad vastavad v4 väärtused), mis tähendab, et tegu on täielikult täitmata (pooliku, lõpetamata) segmendiga.

Kui salvestatakse olemasolevate aegridade uued elemendid, siis täidetakse kõigepealt veerud s4 (tabelis *stocks_status*) ja v4 (tabelis *stocks_values*) ning nüüd on segment täidetud. Segmendi täitumise järel luuakse uus segment (seg väärtus 2) ning kogu protsess algab uuesti.

Tabeli s*tocks_description* loomise laused:

```
CREATE TABLE stocks_description
(
	ts_id varchar(20) NOT NULL,
	start_date date NOT NULL,
	end_date date NULL,
	frequency integer NOT NULL,
	ticker varchar(6) NOT NULL
);

ALTER TABLE stocks_description ADD CONSTRAINT
PK_stocks_description PRIMARY KEY (ts_id);
```

Joonis 13. Disainilahenduse „Aegridade segmenteerimine“ tabeli *stocks_description* loomise laused.

Tabeli s*tocks_values* loomise laused:

```
CREATE TABLE stocks_values
(
	ts_id varchar(20) NOT NULL,
	multiv varchar(10) NOT NULL,
	seg integer NOT NULL,
	v1 numeric(10,2) NULL,
	v2 numeric(10,2) NULL,
	v3 numeric(10,2) NULL,
	v4 numeric(10,2) NULL,
	v5 numeric(10,2) NULL
);

ALTER TABLE stocks_values ADD CONSTRAINT
PK_stocks_values PRIMARY KEY (ts_id, multiv, seg);

ALTER TABLE stocks_values ADD CONSTRAINT
FK_stocks_values_stocks_description FOREIGN KEY (ts_id)
REFERENCES stocks_description (ts_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Joonis 14. Disainilahenduse „Aegridade segmenteerimine“ tabeli *stocks_values* loomise laused.

Tabeli *stocks_status* loomise laused:

```
CREATE TABLE stocks_status
(
	ts_id varchar(20) NOT NULL,
	seg integer NOT NULL,
	s1 boolean NOT NULL   DEFAULT FALSE,
	s2 boolean NOT NULL   DEFAULT FALSE,
	s3 boolean NOT NULL   DEFAULT FALSE,
	s4 boolean NOT NULL   DEFAULT FALSE,
	s5 boolean NOT NULL   DEFAULT FALSE
);
ALTER TABLE stocks_status ADD CONSTRAINT
PK_stocks_status PRIMARY KEY (ts_id, seg);

ALTER TABLE stocks_status ADD CONSTRAINT
FK_stocks_status_stocks_description FOREIGN KEY (ts_id)
REFERENCES stocks_description (ts_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Joonis 15. Disainilahenduse „Aegridade segmenteerimine“ tabeli *stocks_status* loomise laused.

4.5 Vertikaalselt vektoriseerimine

Nimi eesti keeles: vertikaalselt vektoriseerimine

Nimi inglise keeles: vertical vectorisation

Viited allikatele: [36], [37]

Kirjeldus: Disainilahenduses grupeeritakse aegrea elementide omaduste väärtused fikseeritud (andmebaasi disainija poolt määratud) suurusega massiividesse. Sellise lahenduse eesmärk on vähendada tabeli poolt kasutatavat kettaruumi, vähendades ajatemplite arvu. Iga rida tabelis koosneb aegrea identifikaatorist, ajatemplist ning aegreaga seotud väärtuste massiivist. Väärtuste massiivi iga element on seotud erineva aegrea ajatempliga, kus massiivi esimene väärtus on seotud rea ajatempliga ning iga järgmine väärtus on sellest ajatemplist mingi kindla intervalli võrra hilisema ajatempliga. Intervalli ei salvestata tabelisse, vaid see on sissekodeeritud tabeli põhjal loodud vaatesse.

Tugevad küljed:

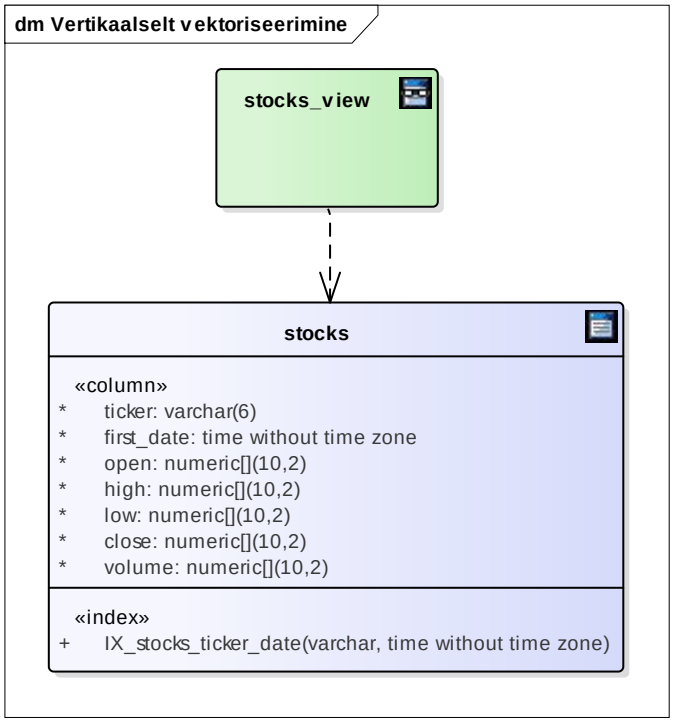
- Salvestatakse vähem ajatempleid – kasutatakse vähem kettaruumi.

Nõrgad küljed:

- Ebaselge struktuur.
- Keeruline sisetada ning uuendada andmeid.
- Kuigi SQL standard kirjeldab massiivitüübi konstruktorit juba alates SQLi versioonist SQL:1999 [41] ei pruugi kõik andmebaasisüsteemid seda toetada (nagu näiteks MySQL vähemalt kuni versioonini 8 [42]).

Näide:

Luuakse tabel s*tocks* (loomise lause Joonis 17), mis sisaldab informatsiooni erinevate aktsiatega seotud aegridade kohta. Tabelile *stocks* luuakse vaade s*tocks_view* (loomise lause Joonis 18), et muuta päringute tegemist mugavamaks. Disainilahenduse füüsilise disaini andmemudel on toodud Joonis 16, tabeli veergude kirjeldused Tabel 26 ning tabel näiteväärtustega Tabel 27.



Joonis 16. Füüsilise disaini andmemudel disainilahendusele „Vertikaalselt vektoriseerimine“.

Kaalutlused, millest tulenevalt ei deklareerita tabelis *stocks* PRIMARY KEY või UNIQUE kitsendust on samasugused nagu kirjeldati disainilahenduses „naiivne meetod“ (jaotis 4.1). Tabeli *stocks* indeksi *IX_stocks_ticker_date* loomise laused erinevate indeksitüüpide korral on esitatud Lisa 6.

Tabel 26. Disainilahenduse „Vertikaalselt vektoriseerimine“ tabeli *stocks* veergude kirjeldus.

Veerunimi	**Kirjeldus**
ticker	finantsinstrumendi sümbol
first_date	massiivi esimese aegrea elemendi ajatempel
open	massiiv algushindadest
high	massiiv kõrgeimatest hindadest
low	massiiv madalamatest hindadest
close	massiiv lõpphindadest
volume	massiiv ostetud/müüdud kogustest

Tabel 27. Disainilahenduse „Vertikaalselt vektoriseerimine“ tabel *stocks* koos näiteväärtustega.

ticker	**first_date**	**open**	**high**	**low**	**close**	**volume**
GOOGL	01.01.2019 00:00:00	{100.10, 102.30, 104.22}	{103.45, 105.33, 104.67}	{100.04, 101.90, 103.14}	{102.30, 104.22, 104.11}	{1200, 1322, 997}
FB	01.01.2019 00:00:00	{130.10, 130.03, 129.88}	{130.38, 131.01, 130.44}	{129.38, 129.44, 129.22}	{130.03, 129.88, 129.92}	{1018, 921, 1100}
AMZN	01.01.2019 00:00:00	{1410.15, 1409.39, 1406.22}	{1412.55, 1410.17, 1409.33}	{1409.34, 1406.22, 1403.21}	{1409.39, 1406.22, 1405.38}	{200, 190, 102}

Tabeli *stocks* loomise lause:

```
CREATE TABLE stocks
(
    ticker character varying(6) NOT NULL,
    first_date timestamp without time zone NOT NULL,
    open numeric(10,2)[] NOT NULL,
    high numeric(10,2)[] NOT NULL,
    low numeric(10,2)[] NOT NULL,
    close numeric(10,2)[] NOT NULL,
    volume numeric(10,2)[] NOT NULL
);
```

Joonis 17. Disainilahenduse „Vertikaalselt vektoriseerimine“ tabeli *stocks* loomise lause.

Vaate s*tocks_view* loomise lause:

```
CREATE VIEW stocks_view AS
SELECT first_date,
    first_date + '00:01:00'::interval * i AS date,
    ticker,
    open[i] AS open,
    high[i] AS high,
    low[i] AS low,
    close[i] AS close,
    volume[i] AS volume
FROM stocks,
     generate_series(0, 59) gs(i)
WHERE open[i] IS NOT NULL;
```

Joonis 18. Disainilahenduse „Vertikaalselt vektoriseerimine“ vaate *stocks_view* loomise lause.

Vaate *stocks_view* loomise lauses tehakse otsekorrutis (ehk Cartesiuse korrutis) tabeli *stocks* ja funktsiooni *generate_series* poolt tagastatud virtuaalse tabeli (aliasega *gs* ning tagastav veerg on nimetatud *i*) vahel. Otsekorrutise tulemuseks on tulemuste komplekt, kus on olemas ühendus iga *stocks* tabeli rea ja virtuaalse tabeli *gs* rea vahel – kogu

tagastatavaks ridade arvuks on tabeli *stocks* ridade arvu ja virtuaalse tabeli gs ridade arvu (näites 60) korrutis. Näites on töö autor otsustanud ühes tabeli *stocks* reas hoida 60ne aegrea elemendi väärtused. Kuna aegrea elementide vaheliseks intervalliks on näites üks minut siis hoitakse ühes tabeli *stocks* reas ühe tunni jagu aegrea elemente, kusjuures virtuaalse tabeli *gs* poolt tagastatud veerg *i* tähistab ajatemplist *first_date i*'ndat minutit. Näiteks, kui tulemuste komplekti ühes reas on *first_date* väärtuseks 01.01.2019 12:00:00 ja *i* väärtuseks on 3 siis päringu tulemusena on välja *date* väärtuseks 01.01.2019 12:03:00 ning *open*, *high*, *low*, *close* ja *volume* saavad *stocks* tabeli vastavatest massiividest kolmanda (ehk *i*'nda) väärtuse, mis vastavad ajatemplil *date* salvestatud väärtustele.

4.6 Horisontaalselt vektoriseerimine

Nimi eesti keeles: horisontaalselt vektoriseerimine

Nimi inglise keeles: horizontal vectorisation

Viited allikatele: [36], [37]

Kirjeldus: Disainilahenduses grupeeritakse erinevate, kuid samade omadustega, aegridade väärtused. Erinevalt vertikaalselt vektoriseerimisest, kus grupeeritakse andmed aegrea põhiselt, grupeeritakse horisontaalselt vektoriseerimisel andmed ajatempli põhiselt ehk iga rida tabelis hoiab mitme erineva aegrea omaduste väärtuseid, kuid jagab nende ühist ajatemplit.

Tugevad küljed:

- Salvestatakse vähem ajatempleid – kasutatakse vähem kettaruumi.

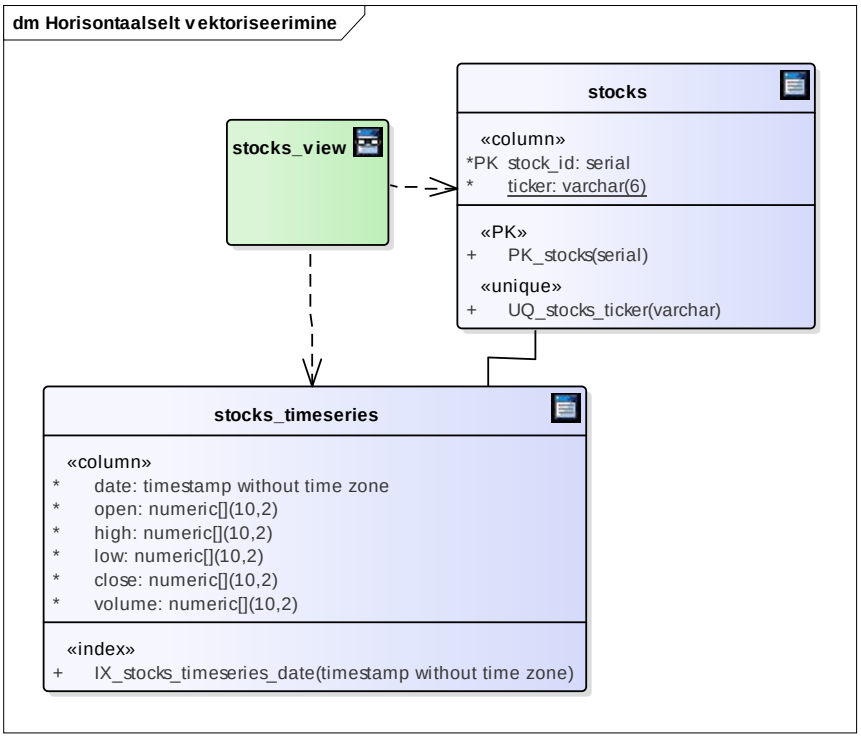
Nõrgad küljed:

- Ebaselge struktuur.
- Keeruline sisetada ning uuendada andmeid.
- Kuigi SQL standard kirjeldab massiivitüübi konstruktorit juba alates SQLi versioonist SQL:1999 [41] ei pruugi kõik andmebaasisüsteemid seda toetada (nagu näiteks MySQL vähemalt kuni versioonini 8 [42]).
- Tuleb arvestada võimalike piiridega massiivide võimalikule suurusele. Näiteks PostgreSQLis ei ole sellist piiri ilmutatult seatud, kuid väljas olev väärtus võib seal olla maksimaalselt kuni 1 GB [43], mis seab ka (tõsi küll väga suure) ülapiiri massiivi elementide arvule.
- Võimalik halb töökiirus väga suurte massiividega töötamisel.

Näide:

Luuakse tabelid s*tocks* (loomise laused Joonis 20), mis sisaldab aegridadega seotud metaandmeid (näiteks aktsia sümbol, valuuta) ja s*tocks_timeseries* (loomise lause Joonis

21), mis sisaldab aegridu ning vaade *stocks_view* (loomise lause Joonis 22), mis lihtsustab päringute tegemist. Füüsilise disaini andmemudeli näide on toodud Joonis 19, tabelite veergude kirjeldused on toodud tabelites Tabel 28 ja Tabel 29 ning tabelid näiteväärtustega on toodud tabelites Tabel 30 ja Tabel 31.



Joonis 19. Füüsilise disaini andmemudel disainilahendusele „Horisontaalselt vektoriseerimine“.

Kaalutlused, millest tulenevalt ei deklareerita tabelis *stocks_timeseries* PRIMARY KEY või UNIQUE kitsendust on samasugused nagu kirjeldati disainilahenduses „naiivne meetod“ (jaotis 4.1). Tabeli *stocks_timeseries* indeksi *IX_stocks_timeseries_date* loomise laused erinevate indeksitüüpide korral on esitatud Lisa 6.

Tabel 28. Disainilahenduse „Horisontaalselt vektoriseerimine“ tabeli s*tocks_timeseries* veergude kirjeldus.

Veerunimi	**Kirjeldus**
date	massiivi esimese aegrea elemendi ajatempel
open	massiiv algushindadest
high	massiiv kõrgeimatest hindadest
low	massiiv madalamatest hindadest
close	massiiv lõpphindadest
volume	massiiv ostetud/müüdud kogustest

Tabel 29. Disainilahenduse „Horisontaalselt vektoriseerimine“ tabeli s*tocks* veergude kirjeldus.

Veerunimi	**Kirjeldus**
stock_id	aegrea unikaalne identifikaator, mis on ühtlasi vääruste indeks massiivides (tabeli *Stocks_timeseries* veergudes)
ticker	finantsinstrumendi unikaalne sümbol

Tabel 30. Disainilahenduse „Horisontaalselt vektoriseerimine“ tabel s*tocks* koos näiteväärtustega.

stock_id	**ticker**
1	GOOGL
2	FB
3	AMZN

Tabel 31. Disainilahenduse „Horisontaalselt vektoriseerimine“ tabel s*tocks_timeseries* koos näiteväärtustega.

date	**open**	**high**	**low**	**close**	**volume**
01.01.2019 00:00:00	{100.10, 130.10, 1410.15}	{103.45, 130.38, 1412.55}	{100.04, 129.38, 1409.34}	{102.30, 130.03, 1409.39}	{1200, 1018, 200}
01.01.2019 00:01:00	{102.30, 130.03, 1409.39}	{105.33, 131.01, 1410.17}	{101.90, 129.44, 1406.22}	{104.22, 129.88, 1406.22}	{1322, 921, 190}
01.01.2019 00:02:00	{104.22, 129.88, 1406.22}	{104.67, 130.44, 1409.33}	{103.14, 129.22, 1403.21}	{104.11, 129.92, 1405.38}	{997, 1100, 102}

Tabeli *stocks* loomise laused:

```
CREATE TABLE stocks
(
    stock_id integer NOT NULL,
    ticker character varying(6) NOT NULL
);

ALTER TABLE stocks ADD CONSTRAINT PK_stocks
PRIMARY KEY (stock_id);

ALTER TABLE stocks ADD CONSTRAINT UQ_stocks_ticker
UNIQUE (ticker);
```

Joonis 20. Disainilahenduse „Horisontaalselt vektoriseerimine“ tabeli *stocks* loomise laused.

Tabeli s*tocks_timeseries* loomise lause:

```
CREATE TABLE stocks_timeseries
(
    date timestamp without time zone NOT NULL,
    open numeric(10,2)[] NOT NULL,
    high numeric(10,2)[] NOT NULL,
    low numeric(10,2)[] NOT NULL,
    close numeric(10,2)[] NOT NULL,
    volume numeric(10,2)[] NOT NULL
);
```

Joonis 21. Disainilahenduse „Horisontaalselt vektoriseerimine“ tabeli *stocks_timeseries* loomise lause.

Vaate s*tocks_view* loomise lause:

```
CREATE VIEW stocks_view AS
SELECT
      ts.date,
      s.ticker,
      ts.open[s.stock_id] AS open,
      ts.high[s.stock_id] AS high,
      ts.low[s.stock_id] AS low,
      ts.close[s.stock_id] AS close,
      ts.volume[s.stock_id] AS volume
FROM
      stocks_timeseries ts,
      stocks s;
```

Joonis 22. Disainilahenduse „Horisontaalselt vektoriseerimine“ vaate *stocks_view* loomise lause.

Tabelis *stocks_timeseries* ei ole välisvõtit (stock_id), sest seos kahes tabelis olevate andmete vahel on läbi väärtuste järjekorra väärtuste massiivis. Näiteks veeru *open* väljas on väärtus (arvude massiiv) {100.10, 130.10, 1410.15}. Massiivis teisel positsioonil olev väärtus 130.10 tähendab, et see on aktsia kohta, mille identifikaator on 2 (antud juhul FB).

4.7 Vertikaalne järjestikplaanur

Nimi eesti keeles: vertikaalne järestikplaanur

Nimi inglise keeles: vertical round-robin

Viited allikatele: [34]

Kirjeldus: Järestikplaanur disainilahendust kasutades luuakse ringikujuline struktuur aegrea salvestamiseks. Ringikujulise struktuuri eesmärgiks on vanade andmete kustutamise asemel need ülekirjutada. Disainilahendus eeldab, et on sätestatud järestikplaanuri maksimaalne indeks ehk maksimaalselt salvestavate aegrea elementide arv. Disainilahenduse kasutamisel näeb aegrea elementide sisestamine välja järgmine: esimene aegrea element saab omale indeksi 0, järgmine 1 jne kuni jõutakse maksimaalse indeksini, misjärel alustatakse uuesti indeksist 0 ning algab aegrea elementide üle kirjutamine. Disainilahenduse nimes „vertikaalne“ viitab aegrea elementide kombineerimisele mööda ajatelge – lähestikku olevaid elemente hoiustatakse massiivis ühes tabeli reas.

Tugevad küljed:

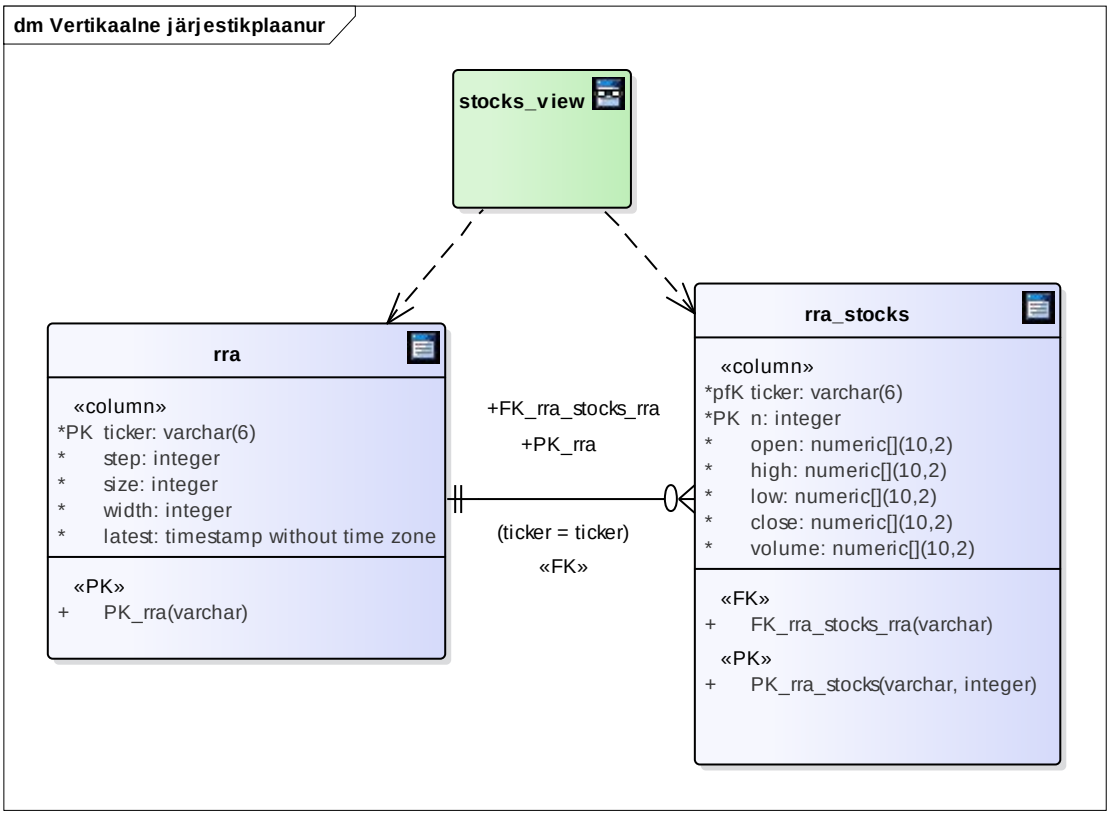
- Säilitatakse ainult viimase sisestatud aegrea elementi ajatempel – väiksem kettaruumi kasutus.
- Automaatne andmete säilituspoliitika – vanad andmed kirjutatakse üle.

Nõrgad küljed:

- Ebaselge andmebaasi struktuur.
- Andmete lisamine ja muutmise laused on keerulised.
- Kuigi SQL standard kirjeldab massiivitüübi konstruktorit juba alates SQLi versioonist SQL:1999 [41] ei pruugi kõik andmebaasisüsteemid seda toetada (nagu näiteks MySQL vähemalt kuni versioonini 8 [42]).

Näide:

Luuakse tabelid *rra* (loomise laused Joonis 24) ja *rra_stocks* (loomise laused Joonis 25) ning vaade s*tocks_view* (loomise lause Joonis 26). RRA ehk *Round Robin Archive* (järjestikplaanur arhiiv) on kogum mingi süsteemi järgi valitud elementidest. Tabel *rra* sisaldab informatsiooni salvestatava perioodrea kohta – aegridade intervall, mitu viimast aegrida säilitada ning viimase aegrea ajatempel. Tabel *rra_stocks* sisaldab finantsandmetega seotud aegridu. Vaade s*tocks_view* lihtustab päringute tegemist. Näide füüsilise disaini andmemudelist on Joonis 23, tabelite veergude kirjeldused on tabelites Tabel 32 ja Tabel 33 ning tabelid koos näiteväärtustega on Tabel 34 ja Tabel 35.



Joonis 23. Füüsilise disaini andmemudel disainilahendusele „Vertikaalne järestikplaanur“.

Tabel 32. Disainilahenduse „Vertikaalne järestikplaanur“ tabeli *rra* veergude kirjeldus.

Veerunimi	**Kirjeldus**
ticker	finantsinstrumendi sümbol
step	aegrea intervalli pikkus sekundites
size	säilitatavate aegrea elementide kogus
width	säilitatavate aegrea elementide kogus ühes reas
latest	viimase sisestuse ajatempel

Tabel 33. Disainilahenduse „Vertikaalne järestikplaanur“ tabeli rra*_stocks* veergude kirjeldus.

Veerunimi	**Kirjeldus**
ticker	finantsinstrumendi sümbol
n	aegrea elementide järjekorranumber, mida kasutatakse ajatemplite arvutamisel vaates
open	massiiv intervalli algushindadest
high	massiiv kõrgeimatest hindadest intervalli jooksul
low	massiiv madalamatest hindadest intervalli jooksul
close	massiiv intervalli lõpphindadest
volume	massiiv intervalli jooksul ostetud/müüdud kogustest

width väärtus tabelis *rra* määrab massiivi elementide maksimaalse arvu tabelis *rra_stocks*. Seega kui width=60, siis saab igas massiivis olla kuni 60 väärtust.

size väärtus tabelis *rra* määrab ära, millal algab ülekirjutamine. Seega kui size=1440 ja width=60 ning 1440/60=24, siis tabelis *rra_stocks* on iga aktsia kohta maksimaalselt 24 rida (n=0...23). Kui need read saavad täis, siis algab ülekirjutamine.

Tabel 34. Disainilahenduse „Vertikaalne järestikplaanur“ tabel *rra* koos näiteväärtustega.

ticker	**step**	**size**	**width**	**latest**
GOOGL	60	1440	60	01.01.2019 00:02:00
FB	60	1440	60	01.01.2019 00:02:00
AMZN	60	1440	60	01.01.2019 00:02:00

Tabel 35. Disainilahenduse „Vertikaalne järestikplaanur“ tabel *rra_stocks* koos näiteväärtustega.

ticker	**n**	**open**	**high**	**low**	**close**	**volume**
GOOGL	0	{100.10, 102.30, 104.22}	{103.45, 105.33, 104.67}	{100.04, 101.90, 103.14}	{102.30, 104.22, 104.11}	{1200, 1322, 997}
FB	0	{130.10, 130.03, 129.88}	{130.38, 131.01, 130.44}	{129.38, 129.44, 129.22}	{130.03, 129.88, 129.92}	{1018, 921, 1100}
AMZN	0	{1410.15, 1409.39, 1406.22}	{1412.55, 1410.17, 1409.33}	{1409.34, 1406.22, 1403.21}	{1409.39, 1406.22, 1405.38}	{200, 190, 102}

Tabeli *rra* loomise laused:

```
CREATE TABLE rra
(
	ticker varchar(6) NOT NULL,
	step integer NOT NULL,
	size integer NOT NULL,
	width integer NOT NULL,
	latest timestamp without time zone NOT NULL
);

ALTER TABLE rra ADD CONSTRAINT PK_rra PRIMARY KEY (ticker);
```

Joonis 24. Disainilahenduse „Vertikaalne järestikplaanur“ tabeli *rra* loomise laused.

Tabeli *rra_stocks* loomise laused:

```
CREATE TABLE rra_stocks
(
	ticker varchar(6) NOT NULL,
	n integer NOT NULL,
	open numeric(10,2)[] NOT NULL,
	high numeric(10,2)[] NOT NULL,
	low numeric(10,2)[] NOT NULL,
	close numeric(10,2)[] NOT NULL,
	volume numeric(10,2)[] NOT NULL
);

ALTER TABLE rra_stocks ADD CONSTRAINT PK_rra_stocks
	PRIMARY KEY (ticker,n);

ALTER TABLE rra_stocks ADD CONSTRAINT FK_rra_stocks_rra
	FOREIGN KEY (ticker) REFERENCES rra (ticker) ON DELETE
CASCADE ON UPDATE CASCADE;
```

Joonis 25. Disainilahenduse „Vertikaalne järestikplaanur“ tabeli *rra_stocks* loomise laused.

Vaate *stocks_view* loomise lause:

```
CREATE VIEW stocks_view AS
SELECT
   rra.ticker AS ticker,
   latest - INTERVAL '1 SECOND' * rra.step * MOD(rra.size +
MOD(EXTRACT(epoch FROM rra.latest)::BIGINT / (rra.step), size) + 1 -
(generate_subscripts(open, 1) + n * width), rra.size) AS date,
   UNNEST(open) AS open,
   UNNEST(high) AS high,
   UNNEST(low) AS low,
   UNNEST(close) AS close,
   UNNEST(volume) AS volume
FROM
   rra
INNER JOIN
   rra_stocks AS ts
   ON ts.ticker = rra.ticker;
```

Joonis 26. Disainilahenduse „Vertikaalne järestikplaanur“ vaate *stocks_view* loomise lause.

Vaate loomise lause (sh väärtuse *date* leidmiseks kasutatud arvutuskäigu) selgitustega kirjelduse võib leida viitest [34].

4.8 Horisontaalne järjestikplaanur

Nimi eesti keeles: horisontaalne järjestikplaanur

Nimi inglise keeles: horizontal round-robin

Viited allikatele: [33], [34]

Kirjeldus: Sarnaselt vertikaalsele järjestikplaanurile luuakse ka horisontaalse järestikplaanuri puhul ringikujuline struktuur aegridade salvestamiseks. Erinevalt vertikaalsest järjestikplaanurist, kus hoiustatakse ajaliselt lähestikku olevaid aegrea elemente massiivis, kombineeritakse horisontaalses järjestikplaanuris erinevate aegridade, kuid olemuselt samade väärtustega aegrea elemendid. Horisontaalses järjestikplaanuris luuake kõigepealt kogum aegridadest, mille tüüp on sama. Tüübi all mõeldakse aegrea elementide vahelist intervalli. Kõik ühte kogumisse kuuluvad aegread saavad omale unikaalse numbrilise tähistuse, mis viitab massiivi indeksile, kus vastava aegrea väärtus salvestatakse.

Tugevad küljed:

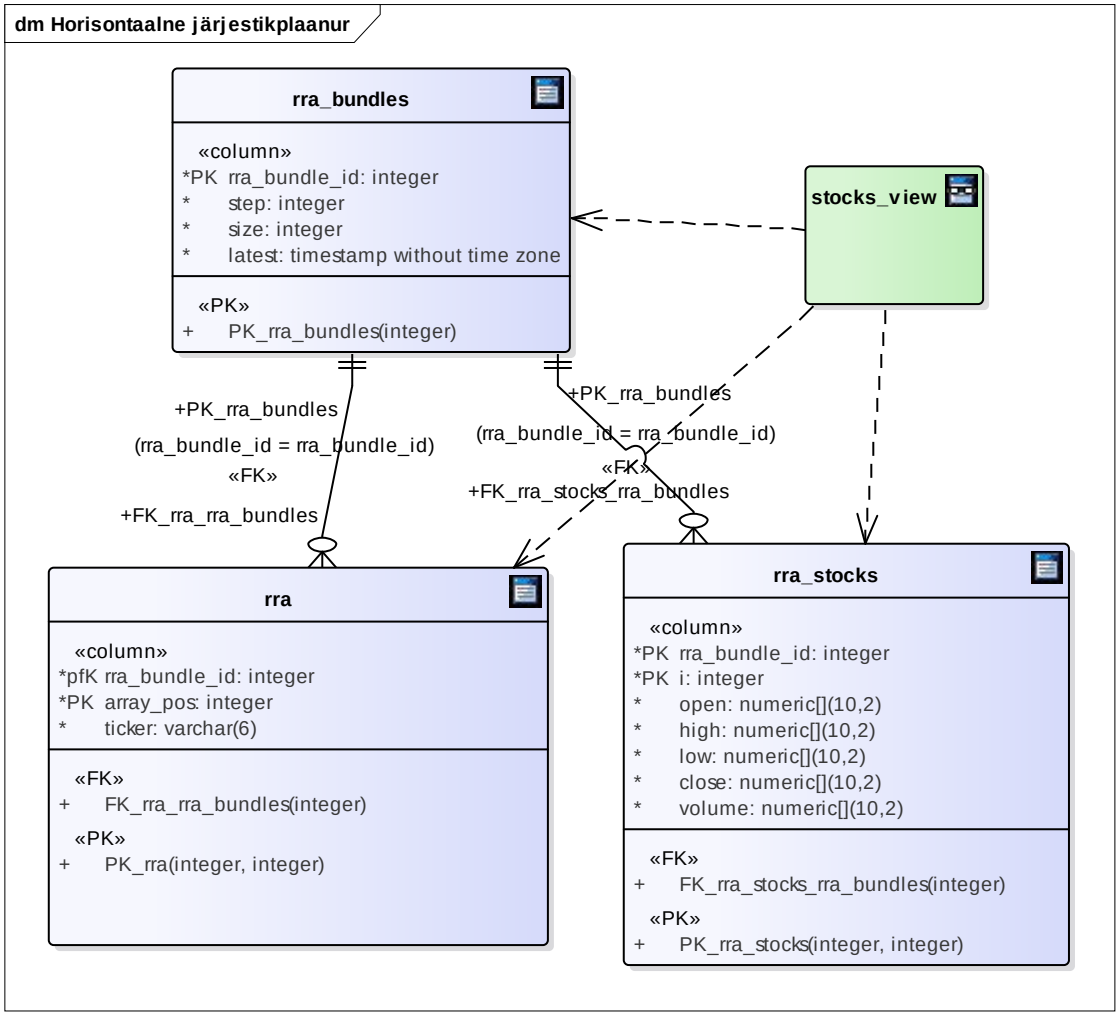
- Ridade lisamine on kiirem võrreldes vertikaalse lahendusega.
- Säilitatakse ainult viimase sisestatud aegrea elemendi ajatempel – väiksem kettaruumi kasutus.
- Automaatne andmete säilituspoliitika – vanad andmed kirjutatakse üle.

Nõrgad küljed:

- Ühe aegrea kohta päringu tegemisel loetakse ka teised aegread – andmete lugemise kiirus kannatab.
- Ebaselge struktuur.
- Kuigi SQL standard kirjeldab massiivitüübi konstruktorit juba alates SQLi versioonist SQL:1999 [41] ei pruugi kõik andmebaasisüsteemid seda toetada (nagu näiteks MySQL vähemalt kuni versioonini 8 [42]).

Näide:

Luuakse tabelid *rra_bundles* (loomise laused Joonis 28), *rra* (loomise laused Joonis 29) ja *rra_stocks* (loomise laused Joonis 30) ning vaade *stocks_view* (loomise lause Joonis 31). Tabel *rra_bundles* sisaldab informatsiooni salvestavate järestikplaanur kimpude (*bundle*) kohta – aegridade intervalli, mitu viimast aegrea elementi säilitada ning viimati sisestatud aegrea elemendi ajatemplit. Tabel *rra* sisaldab suhteid aegridade ja aegrea kimpude vahel ning muid aegreaga seotud metaandmeid. Tabel *rra_stocks* sisaldab aegridade elemente. Vaade *stocks_view* lihtustab päringute tegemist. Andemudeli näide on toodud Joonis 27, tabelite veergude kirjeldused on toodud tabelites Tabel 36, Tabel 37 ja Tabel 38 ning tabelid näiteväärtustega on toodud tabelites Tabel 39, Tabel 40 ja Tabel 41.



Joonis 27. Füüsilise disaini andmemudel disainilahendusele „Horisontaalne järestikplaanur“.

Tabel 36. Disainilahenduse „Horisontaalne järestikplaanur“ tabeli *rra_bundles* veergude kirjeldus.

Veerunimi	**Kirjeldus**
rra_bundle_id	unikaalne aegridade kogumiku indentifikaator
step	aegrea elementide vahelise intervalli pikkus sekundites
size	säilitavate aegrea elementide kogus
latest	viimase aegrea elemendi sisestuse ajatempel

Tabel 37. Disainilahenduse „Horisontaalne järestikplaanur“ tabeli *rra* veergude kirjeldus.

Veerunimi	**Kirjeldus**
rra_bundle_id	unikaalne aegridade kogumiku indentifikaator
array_pos	aegreaga seotud väärtuste indeks massiivis (tabeli *RRA_Stocks* veergudes)
ticker	finantsinstrumendi sümbol

Tabel 38. Disainilahenduse „Horisontaalne järestikplaanur“ tabeli *rra_stocks* veergude kirjeldus.

Veerunimi	**Kirjeldus**
rra_bundle_id	unikaalne aegridade kogumiku indentifikaator
i	aegrea elementide järjekorranumber
open	massiiv intervalli algushindadest
high	massiiv kõrgeimatest hindadest intervalli jooksul
low	massiiv madalamatest hindadest intervalli jooksul
close	massiiv intervalli lõpphindadest
volume	massiiv intervalli jooksul ostetud/müüdud kogustest

Tabel 39. Disainilahenduse „Horisontaalne järestikplaanur“ tabel *rra_bundles* koos näiteväärtustega.

rra_bundle_id	**step**	**size**	**latest**
1	60	1440	01.01.2019 00:02:00

Tabel 40. Disainilahenduse „Horisontaalne järestikplaanur“ tabel *rra* koos näiteväärtustega.

rra_bundle_id	**array_pos**	**ticker**
1	1	GOOGL
1	2	FB
1	3	AMZN

Tabel 41. Disainilahenduse „Horisontaalne järestikplaanur“ tabel *rra_stocks* koos näiteväärtustega.

rra_bundle_id	**i**	**open**	**high**	**low**	**close**	**volume**
1	0	{100.10, 130.10, 1410.15}	{103.45, 130.38, 1412.55}	{100.04, 129.38, 1409.34}	{102.30, 130.03, 1409.39}	{1200, 1018, 200}
1	1	{102.30, 130.03, 1409.39}	{105.33, 131.01, 1410.17}	{101.90, 129.44, 1406.22}	{104.22, 129.88, 1406.22}	{1322, 921, 190}
1	2	{104.22, 129.88, 1406.22}	{104.67, 130.44, 1409.33}	{103.14, 129.22, 1403.21}	{104.11, 129.92, 1405.38}	{997, 1100, 102}

Näiteks tabeli *rra* näiteandmete viimane rida ütleb seda, et tabelis *rra_stocks* on iga rea puhul kus rra_bundle_id=1 Amazoni aktsia kohta käivate omaduste väärtused vastavates massiivides kolmandas positsioonis (PostgreSQL andmebaasisüsteemis algavad massiivide indeksid ühest).

size väärtus tabelis *rra_bundles* viitab maksimaalsele i väärtusele tabelis *rra_stocks* vastava *rra_bundle_id* puhul. Näiteks kui tabelis *rra_bundles* on meil rida, kus rra_bundle_id = 1 ja size = 1440 siis tabelis *rra_stocks* saab olla maksimaalselt 1440 rida selle kimbu kohta (i=0,…,1439).

step väärtus tabelis *rra_bundles* viitab kahe i vahelisele ajale sekundites tabelis *rra_stocks* ehk kahe samasse kimpu kuuluva rea ajatemplite vahe on (esimese rea i - teise rea i) * step.

Tabeli *rra_bundles* loomise laused:

```
CREATE TABLE rra_bundles
(
	rra_bundle_id serial NOT NULL,
	step integer NOT NULL,
	size integer NOT NULL,
	latest timestamp without time zone NOT NULL
);

ALTER TABLE rra_bundles ADD CONSTRAINT PK_rra_bundles PRIMARY
KEY (rra_bundle_id);
```

Joonis 28. Disainilahenduse „Horisontaalne järestikplaanur“ tabeli *rra_bundles* loomise laused.

Tabeli *rra* loomise laused:

```
CREATE TABLE rra
(
	rra_bundle_id integer NOT NULL,
	array_pos integer NOT NULL,
	ticker varchar(6) NOT NULL
);

ALTER TABLE rra ADD CONSTRAINT PK_rra
	PRIMARY KEY (rra_bundle_id,array_pos);

ALTER TABLE rra ADD CONSTRAINT FK_rra_rra_bundles
	FOREIGN KEY (rra_bundle_id) REFERENCES rra_bundles
(rra_bundle_id) ON DELETE CASCADE ON UPDATE CASCADE;
```

Joonis 29. Disainilahenduse „Horisontaalne järestikplaanur“ tabeli *rra* loomise laused.

Tabeli *rra_stocks* loomise laused:

```
CREATE TABLE rra_stocks
(
	rra_bundle_id integer NOT NULL,
	i integer NOT NULL,
	open numeric(10,2)[] NOT NULL,
	high numeric(10,2)[] NOT NULL,
	low numeric(10,2)[] NOT NULL,
	close numeric(10,2)[] NOT NULL,
	volume numeric(10,2)[] NOT NULL
);

ALTER TABLE rra_stocks ADD CONSTRAINT PK_rra_stocks
	PRIMARY KEY (rra_bundle_id,i);

ALTER TABLE rra_stocks ADD CONSTRAINT FK_rra_stocks_rra_bundles
	FOREIGN KEY (rra_bundle_id) REFERENCES rra_bundles
(rra_bundle_id) ON DELETE CASCADE ON UPDATE CASCADE;
```

Joonis 30. Disainilahenduse „Horisontaalne järestikplaanur“ tabeli *rra_stocks* loomise laused.

Vaate s*tocks_view* loomise lause:

```
CREATE VIEW stocks_view AS
SELECT
   ticker,
   rra_bundle.latest - INTERVAL '1 SECOND' * rra_bundle.step *
MOD(rra_bundle.size + MOD(EXTRACT(epoch FROM
rra_bundle.latest)::BIGINT / (rra_bundle.step), rra_bundle.size) - i,
rra_bundle.size) AS date,
   open[array_pos] AS open,
   high[array_pos] AS high,
   low[array_pos] AS low,
   close[array_pos] AS close,
   volume[array_pos] AS volume
FROM
   rra
   JOIN
      rra_bundles
      ON rra_bundles.rra_bundle_id = rra.rra_bundle_id
   JOIN
      rra_stocks AS ts
      ON ts.rra_bundle_id = rra_bundles.rra_bundle_id;
```

Joonis 31. Disainilahenduse „Horisontaalne järestikplaanur“ vaate *stocks_view* loomise lause.

Väärtuse *date* leidmiseks kasutatud arvutuskäigu koos selgitustega võib leida viitest [34].

5 Eksperiment

Käesolevas peatükis kirjeldatakse töö käigus läbiviidavat eksperimenti.

5.1 Eksperimendi eesmärk

Eksperimendi eesmärgiks on võrrelda disainilahendusi ning lahendada järgmised puudujäägid varasemates uuringutes.

- Ei ole võrreldud omavahel erinevate disainilahenduste alusel loodud PostgreSQL andmebaaside puhul päringute (SELECT lausete) täitmise kiiruseid.
- Ei ole võrreldud omavahel erinevate disainilahenduste alusel loodud PostgreSQL andmebaaside puhul ridade lisamise kiiruseid.
- Ei ole analüüsitud aegridade statistika loomiseks vajalike lausete keerukust.
- Ei ole analüüsitud erinevate disainilahenduste alusel loodud andmebaaside andmemahtu.

Eksperimendi käigus otsitakse vastuseid järgmistele küsimustele.

- Kuidas erinevad päringute kiirused PostgreSQL andmebaasisüsteemis erinevate disainilahenduste korral sama andmehulgaga?
- Kuidas erinevad päringute kiirused PostgreSQL andmebaasisüsteemis sama disainilahenduse, kuid erineva andmehulgaga?
- Kuidas erinevad päringute tegemiseks vajalike lausete keerukused PostgreSQL andmebaasisüsteemis erinevate disainilahenduste korral?
- Milline on erinevate disainilahenduste põhjal loodud tabelite andmemaht PostgreSQL andmebaasisüsteemis?

5.2 Uuritavate disainilahenduste valiku põhjendus

Eksperimendis uuritavateks disainilahendusteks on naiivne meetod (jaotis 4.1), iga aegrea kohta üks tabel (jaotis 4.2), vertikaalselt vektoriseerimine (jaotis 4.5) ja horisontaalselt vektoriseerimine (jaotis 4.6). Valiti ainult neli disainilahendust, sest eksperment viiakse läbi suurte andmemahtudega ning kõigi disainilahendustega eksperimenteerimise korral läheks töö maht liiga suureks. Lisaks on välja jäetud disainilahendused, mis on oma omadustelt sarnased teiste, ekspermendis uuritavate, disainilahendustega. Vertikaalne järestikplaanur sarnaneb vertikaalselt vektoriseerimisega. Erinevus seisneb järestikplaanuri omaduses „aegunud“ aegrea elemendid üle kirjutada. Sama kehtib ka horisontaalse järestikplaanuri ning horisontaalselt vektoriseerimise kohta. Aegridade segmenteerimise disainilahendus jäeti välja, sest PostgreSQL andmebaasisüsteemis oleks lihtsam kasutada massiive. Seega võiks eelnevalt mainitud disainilahenduse asemel kasutada, kas vertikaalset järestikplaanurit või vertikaalselt vektoriseeriseerimist, mille puhul on andmebaasiskeem ja päringud paremini arusaadavad.

Vertikaalsetes disainides (vt jaotised 4.5, 4.7) salvestatakse sama aegrea ajaliselt lähestikku olevad andmed ühes massiivis. Näiteks kui aegrea elementide vaheliseks intervalliks on üks minut siis võiks salvestada massiivis ühe tunni ehk 60 aegrea elemendi andmed. Iga massiivi element vastab sama aegrea erineval ajal tehtud mõõtmisele.

Horisontaalsetes disainides (vt jaotised 4.6, 4.8) salvestatakse erinevate aegridade elementide ühesuguste omaduste väärtused ühes massiivis. Näiteks kui mõõdetakse kümnes erinevas asukohas samal ajal temperatuuri siis salvestatakse need mõõtmise tulemused massiivis, kus iga massiivi element vastab ühele asukohale.

Nagu eelnevalt öeldud, siis käesolevas eksperimendis käsitletakse ühte vertikaalset ja ühte horisontaalset disaini.

5.3 Eksperimendi kirjeldus

Eksperimentide läbiviimiseks realiseeritakse disainilahendused PostgreSQL andmebaasisüsteemis. Eksperimendi käigus kasutatakse Tallinna Tehnikaülikooli serverit, kus on kasutusel PostgreSQLi versioon 11.2. Eksperimendis luuakse PostgreSQL andmebaasisüsteemis iga disainilahenduse kohta samas andmebaasis

(*experiment_time_series*) eraldi skeem. Disainilahendused, nendele vastavad lühendid tulemuste tabelis ja skeemide nimed on välja toodud Tabel 42.

Tabel 42. Disainilahendused ning nendele vastavad skeemid.

Disainilahendus	**Lühend tulemuste tabelis**	**Skeemi nimi**
Naiivne meetod (jaotis 4.1)	N	*naive_method*
Iga aegrea kohta üks tabel (jaotis 4.2),	1TPTS	*one_table_per_time_series*
Vertikaalselt vektoriseerimine (jaotis 4.5)	VV	*vertical_vectorisation*
Horisontaalselt vektoriseerimine (jaotis 4.6)	HV	*horizontal_vectorisation*

Kasutatavat serverit kirjeldavad järgmised tehnilised tunnused: virtuaalmasin QEMU Virtual CPU version, kettamaht 811 GB, 40 GB muutmälu, 16 virtuaalset keskprotsessorit ja operatsioonisüsteem CentOS 6.10.

Disainilahenduste võrdlemiseks on valitud järgnevad kriteeriumid.

- Päringute täitmise kiirus.
- Ridade lisamise kiirus.
- Tabelite ning nendega seotud indeksite kettaruumi kasutus.
- Päringute ja operatsioonide keerukus (koodiridade arvu metoodikal).

Päringute ja andmemuudatuste kiiruse mõõtmiseks kasutatakse lauset *EXPLAIN ANALYZE*, mis toob välja nii lause planeerimiseks kui ka täitmiseks kulunud aja [44]. Iga disainilahenduse realisatsioonil teostatakse mõõtmised ilma täiendavate indeksiteta, B-puu indeksitega ning BRIN indeksitega. Enne uue mõõtmise teostamist eelmine indeks eemaldati. Igale disainilahendusele eksperimendi käigus lisatud indeksite loomise laused on toodud Lisa 6, ülevaate loodud indeksitest on Tabel 43.

Tabel 43. Ülevaade eksperimendi käigus lisatud indeksitest.

Disainilahendus	**BRIN indeksiga kaetud veerud**	**B-puu indeksiga kaetud veerud**
Naiivne meetod (jaotis 4.1)	*ticker, date*	*ticker, date*
Iga aegrea kohta üks tabel (jaotis 4.2),	*date*	*date*
Vertikaalselt vektoriseerimine (jaotis 4.5)	*ticker, first_date*	*ticker, first_date*
Horisontaalselt vektoriseerimine (jaotis 4.6)	*date*	*date*

Indeksite loomise laused iga disainilahenduse kohta on esitatud Lisas 6. Kõiki mõõtmisi teostatakse viis korda ning saadud tulemuste põhjal arvutatakse geomeetriline keskmine väärtus. Kasutan geomeetrilist keskmist, aritmeetilise keskmise asemel, sest sama päringu korduval käivitamisel jätab süsteem osad andmed ja täitmisplaani mällu [45] [46]. Päringute ja andmemuudatuste laused erinevate disainilahenduste kohta tuuakse välja jaotises 5.5.

Päringute keerukuse hindamiseks kasutatakse koodiridade arvu metoodikat [47]. Koodiridade arvu metoodikas leitakse füüsilise koodiridade arv, mis tähendab, et koodiridade arvu hulka ei loeta kommentaare ning tühje ridu. Koodiridade arvu on võimalik kasutada tarkvara mahu ja keerukuse hindamiseks. Järgnevalt tuuakse välja koodi vormistamise reeglid, et tagada ühtne päringu koodiridade lugemine erinevate disainilahenduste korral. Vormistamise reeglitele on aluseks võetud [48].

- Iga *SELECT, INSERT, DELETE, WHERE, JOIN, UNION, ORDER BY ja GROUP BY* klausel algab uuelt realt.
- Esimene *WHERE* ja *ON* alamtingimus jääb klausliga samale reale, järgnevad alamtingimused lähevad uutele ridadele.
- *SELECT* klausli iga veerg on uuel real.
- *FROM* klausli iga tabeli nimi on uuel real.

Sellisel viisil vormistatud lausete näited on Joonis 33-Joonis 37. Tabelite ning nendega seotud indeksite kettaruumi kasutuse leidmiseks kasutatakse PostgreSQL andmebaasisüsteemi funktsiooni *pg_total_relation_size*. Funktsioon tagastab tabeli andmemahu baitides (tulemustes esitatakse andmemahud megabaitides). [49] Juhul, kui

disainilahendus koosneb rohkem kui ühest tabelist, siis disainilahenduse andmemahu leidmiseks tabelite andmemahud liidetakse.

5.4 Testandmete genereerimine

Kõik eksperimendi käigus loodud andmebaasid täidetakse testandmetega. Testandmed on pärit *FirstRate Data* veebilehelt [50]. Testandmeteks valiti 16 erineva finantsinstrumendi hinna ajalugu. *FirstRate Data* veebilehel esitatud andmetest on eksperimendi jaoks kasutusele võetud alamhulk andmetest. Andmed, mida kasutatakse eksperimendi käigus on ühe minutilise intervalliga perioodread. Iga perioodrea väärtusteks on avamishind, kõrgeim hind, madalaim hind, sulgemishind ning ostetud/müüdud väärtpaberite kogus. Programmikood, mida on kasutatud andmete kokku liitmiseks, lünkade täitmiseks ja sobivateks andmemahtuteks jagamisel on saadaval Lisa 2. Näide töödeldud andmete formaadist on välja toodud Joonis 32.

```
time,open,high,low,close,volume,ticker
2018-12-21 12:00:00,64.95,64.96,64.87,64.85,12,A
2018-12-21 12:00:00,55.33,55.35,55.32,55.28,143,VZ
2018-12-21 12:00:00,28.62,28.63,28.61,28.58,532,T
2018-12-21 12:00:00,86.85,86.91,86.88,86.78,171,ABBV
2018-12-21 12:00:00,68.16,68.16,68.08,68.05,70,ABT
2018-12-21 12:00:00,153.19,153.26,153.16,153.16,8,AAP
2018-12-21 12:00:00,42.49,42.5,42.46,42.44,356,PFE
2018-12-21 12:00:00,140.95,140.95,140.78,140.73,22,ACN
2018-12-21 12:00:00,126.3,126.36,126.26,126.11,1179,FB
```

Joonis 32. Näide eksperimendis kasutavatest andmetest.

Selleks, et välja selgitada andmemahtude mõju päringute ja andmemuudatuste operatsioonide kiirusele, loodi kolm erineva suurusega andmestikku (100 000 rida, 1 000 000 rida ja 10 000 000 rida). Lisaks loodi veel üks andmestik (100 000 rida), et mõõta disainilahenduste kiirust uute andmete sisestamisel.

Eksperimenti alustatakse kõige väiksemast andmehulgast, milleks on 100 000 rida. Kui eksperiment on läbi viidud, siis kustutatakse tabelitest kõik andmed. Seejärel lisatakse suuruselt järgmine andmestik ning uuendatakse andmebaasi statistikat, kasutades PostgreSQLi lauset *ANALYZE*. See võimaldab andmebaasisüsteemil valida päringu jaoks sobivaima täitmisplaani. [51]

5.5 Eksperimendis testitavad operatsioonid ja päringud

Eksperimendi käigus testitakse disainilahendusi erinevate operatsioonidega. Lugemisoperatsioonide eesmärgiks on testida disainilahenduse suutlikust andmeid kiiresti koondada ning filtreerida. Päringud põhinevad jaotises 2.3 kirjeldatud näiterakendusel.

Andmete lugemine (P1)

Päringus P1 leitakse kahe nädala pikkuse ajavahemiku küünlagraafiku andmed, kus andmepunktide intervalliks on 15 minutit. Näide küünlagraafiku andmetest ning ühtlasi ka oodatavast päringu tulemusest on Tabel 4. Erinevate disainilahendustega kasutatud laused on leitavad Lisas 3.

Andmete lugemine (P2)

Päringus P2 leitakse iga finantsinstrumendi viimane teadaolev hind. Näide oodatavast päringu tulemusest on Tabel 6. Erinevate disainilahendustega kasutatud laused on leitavad Lisas 4.

Andmete lisamine (I)

Operatsiooniga I lisatakse uued andmed disainilahenduse tabelitesse. Erinevate disainilahendustega kasutatud laused on leitavad Lisas 5.

5.6 Eksperimendis testitavad andmekäitluse laused

Käesolevas jaotises võrreldakse võimalikke lauseid sama ülesande lahendamiseks ning valitakse nendest parim põhinedes loetavusele ja päringu kiirusele. Lausete testimiseks realiseeriti disainilahendus ning täideti tabelit kasutades kõige väiksemat andmehulka (100 000 rida). Seejärel käivitati lause kasutades *EXPLAIN ANALYZE* funktsionaalsust, et mõõta päringu tulemuse saamiseks kulunud aega. Ruumi kokkuhoiu mõttes on selles jaotises väljatoodud ainult naiivse meetodi päringute P1 ja P2 lause „kandidaadid“. Kõik konkreetse disainilahenduse peal proovitud laused on välja toodud GitHubi hoidlas disainilahenduse nimelises kaustas [52]. Eksperimendis kasutatud laused on failides *P1.sql* ja *P2.sql*, lausete kandidaadid on failides *P1_Cx.sql* ja *P2_Cx.sql*, kus *x* on lause number.

Naiivne meetod – P1 lause kandidaadid

Kandidaat 1 – Ühine tabeli avaldis (*common table expression*, CTE) + aknafunktsioon

Päringu täitmisaeg oli 194.222 ms ja planeerimisaeg 0.310 ms. Joonis 33 on väljatoodud kasutatud lause. Lauses luuakse ühise tabeli avaldisega virtuaalne tabel nimega *intervals*, mis sisaldab kõiki 15 minutilise intervalliga ajavahemikke vahemikus 2018-12-21 00:00:00 kuni 2018-12-28 00:00:00. Tabelis *intervals* on veergudeks *start* ja *end*. Veerg *start* sisaldab intervalli algus ajatemplit ja *end* lõpu ajatemplit. Kasutades tabelite ühendamise (JOIN) operatsiooni leian kõik ajatemplite *start* ja *end* vahele jäävad ja *ticker* veeru väärtusega AMZN väärtused tabelist *naive_method* (alias *nm*). Kasutades aknafunktsiooni on võimalik teha arvutusi ridadel, mis on kuidagi omavahel seotud (selles lauses on tingimuseks intervalli lõpu ajatempel). Omavahel seotud read sorteeritakse aja järgi kasvavalt ning seejärel kasutatakse vastavaid funktsioone aknast esimese väärtuse (*FIRST_VALUE*), viimase väärtuse (*LAST_VALUE*), minimaalse väärtuse (*MIN*), maksimaalse väärtuse (*MAX*) ja väärtuste summa (*SUM*) leidmiseks. Ainult unikaalsete väärtuste leidmiseks (aknafunktsioon kordab arvutusi iga aknasse jääva rea puhul) kasutatakse *SELECT DISTINCT* avaldist.

```
EXPLAIN ANALYZE
WITH intervals AS
(
   SELECT
      start::TIMESTAMP,
      start::TIMESTAMP + INTERVAL '15min' AS end
   FROM
      generate_series('2018-12-21 00:00:00', '2018-12-28
00:00:00', INTERVAL '15min') AS start
)
SELECT DISTINCT
   ticker,
   intervals.end AS date,
   FIRST_VALUE(open) OVER w AS open,
   MAX(high) OVER w AS high,
   MIN(low) OVER w AS low,
   LAST_VALUE(close) OVER w AS close,
   SUM(volume) OVER w AS volume
FROM
   intervals
JOIN
   naive_method AS nm
   ON nm.ticker = 'AMZN'
   AND nm.date > intervals.start
   AND nm.date <= intervals.end
   WINDOW w AS
   (
      PARTITION BY
             intervals.end
      ORDER BY
             nm.date ASC
      ROWS BETWEEN unbounded preceding AND unbounded following
   )
ORDER BY
   intervals.end;
```

Joonis 33. Disainilahenduse „Naiivne meetod“ P1 lause kandidaat kasutades ühist tabeli avaldist ja aknafunktsiooni.

Kandidaat 2 – Ühine tabeli avaldis + grupeerimine

Päringu täitmisaeg oli 245.773 ms ja planeerimisaeg 0.385 ms. Joonis 34 on väljatoodud kasutatud lause. Sarnaselt eelmise kandidaadiga, luuakse ka selles lauses ajutine tabel nimega *intervals* ja kasutatakse tabelite ühendamise operatsiooni (JOIN), et leida kõik ajatemplite *start* ja *end* vahele jäävad ja *ticker* veeru väärtusega AMZN väärtused tabelist *naive_method* (alias *nm*). Erinevalt eelmisest kandidaadist, kus kasutati aknafunktsiooni, kasutatakse selles lauses ridade grupeerimist (GROUP BY klauslit). Read grupeeritakse veerus *ticker* olevate väärtuste ning intervalli lõpu ajatempli järgi. Selleks, et leida *open*

ja *close* väärtus mingis ajavahemiks, kasutan funktsiooni *array_agg*, mis koondab igas ridade grupis olevad väärtused kokku massiiviks. *Open* väärtuse leidmiseks sorteerin avamishindade massiivis olevad väärtused aja järgi kasvavalt ning võtan esimese väärtuse massiivist, *close* väärtuse leidmiseks sorteerin sulgemishindade massiivis olevad väärtused aja järgi kahanevalt ning võtan esimese väärtuse massiivist. *High*, *low* ja *volume* väärtuste leidmiseks kasutan vastavalt *MAX*, *MIN* ja *SUM* kokkuvõttefunktsioone.

```
EXPLAIN ANALYZE
WITH intervals AS
(
   SELECT
      start::TIMESTAMP,
      start::TIMESTAMP + INTERVAL '15min' AS end
   FROM
      generate_series('2018-12-21 00:00:00', '2018-12-28
00:00:00', INTERVAL '15min') AS start
)
SELECT DISTINCT
   ticker,
   intervals.end AS date,
   (array_agg(open ORDER BY date ASC))[1] AS open,
   MAX(high) AS high,
   MIN(low) AS low,
   (array_agg(close ORDER BY date DESC))[1] AS close,
   SUM(volume) AS volume
FROM
   intervals
JOIN
   naive_method AS nm
   ON nm.ticker = 'AMZN'
   AND nm.date > intervals.start
   AND nm.date <= intervals.end
GROUP BY ticker, intervals.end
ORDER BY
   intervals.end;
```

Joonis 34. Disainilahenduse „Naiivne meetod“ P1 lause kandidaat kasutades CTE ja GROUP BY.

Naiivne meetod – P2 lause kandidaadid

Kandidaat 1 – alampäring FROM klauslis + grupeerimine + tabelite ühendamine INNER JOIN operatsiooniga

Päringu täitmisaeg oli 366.097 ms ja planeerimisaeg 0.332 ms. Joonis 35 on väljatoodud kasutatud lause.

```
EXPLAIN ANALYZE
SELECT
  p1.ticker,
  p1.date,
  p1.close
FROM
  naive_method p1
INNER JOIN (
  SELECT
   pi.ticker,
   MAX(pi.date) AS max_date
  FROM
    naive_method pi
  GROUP BY
    pi.ticker
  ) p2
ON (p1.date = p2.max_date
  AND p1.ticker = p2.ticker);
```

Joonis 35. Disainilahenduse „Naiivne meetod“ P2 lause kandidaat kasutades alampäringut FROM klauslis ja ühendamiseks INNER JOIN operatsiooni.

Kandidaat 2 – tabeli ühendamine iseendaga kasutades LEFT JOIN operatsiooni

Päringu täitmisaeg oli 887.089 ms ja planeerimisaeg 0.295 ms. Joonis 36 on väljatoodud kasutatud lause.

```
EXPLAIN ANALYZE
SELECT
  p1.ticker,
  p1.date,
  p1.close
FROM
  naive_method p1
LEFT JOIN
  naive_method p2
  ON (p1.ticker = p2.ticker
    AND p1.date < p2.date)
WHERE
  p2.date IS NULL;
```

Joonis 36. Disainilahenduse „Naiivne meetod“ P2 lause kandidaat kasutades tabeli iseendaga ühendamiseks LEFT JOIN operatsiooni.

Kandidaat 3 – ühine tabeli avaldis + grupeerimine + tabelite ühendamine INNER JOIN operatsiooniga

Päringu täitmisaeg oli 362.407 ms ja planeerimisaeg 0.326 ms. Joonis 37 on väljatoodud kasutatud lause.

```
EXPLAIN ANALYZE
WITH max_date AS
(
   SELECT
      ticker,
      MAX(date) AS date
   FROM
      naive_method
   GROUP BY
      ticker
)
SELECT
   nm.ticker,
   nm.date,
   nm.close as price
FROM
   max_date
INNER JOIN
   naive_method AS nm
   ON nm.ticker = max_date.ticker
   AND nm.date = max_date.date;
```

Joonis 37. Disainilahenduse „Naiivne meetod“ P2 lause kandidaat kasutades ühist tabeli avaldist.

Kandidaatidest valiti välja disainilahenduse „naiivne meetod“ päringu P1 testimiseks kandidaat 1 (ühine tabeli avaldis + aknafunktsioon) ja päringu P2 testimiseks kandidaat 3 (ühine tabeli avaldis). Kõigi teiste disainilahenduste peal proovitud laused on välja

toodud GitHubi hoidlas disainilahenduse nimelises kaustas [52]. Kokku koostati 18 kandidaatlauset, millest valiti välja 8.

5.7 Eksperimendi tulemused

Käesolevas jaotises tuuakse välja päringute ja andmemuudatuse operatsioonide kiiruste, andmemahtude ning lausete keerukuse mõõtmise tulemused. **Rasvaselt** on märgitud tulemus, mis oli üleüldiselt kõige parem antud ridade arvu puhul tabelis. Alla joonitud on tulemus, mis oli antud indeksi tüübi+andmehulga kombinatsioonis parim. Tärniga (*) tähistatud tulemused tähendavad, et lause ei tagastanud vastust viie minuti jooksul. Tabelites Tabel 44, Tabel 45 ja Tabel 46 on esitatud päringute ja andmemuudatuse operatsiooni kiirused PostgreSQL andmebaasis millisekundites.

Tabel 44. Päringu P1 täitmisajad (millisekundites).

100 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	12339.66	**163.04**	1137.94
1TPTS	12838.85	186.99	757.39
VV	406.97	417.65	411.93
HV	12625.39	239.39	369.95
1 000 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	134240.01	**191.15**	1296.47
1TPTS	145021.04	198.23	1166.02
VV	461.49	452.54	454.48
HV	119549.67	248.11	403.93
10 000 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	*	198.45	1410.75
1TPTS	*	**192.92**	1007.12
VV	733.57	456.36	481.80
HV	*	260.74	508.44

Tabel 45. Päringu P2 täitmisajad (millisekundites).

100 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	766.65	394.32	661.25
1TPTS	814.11	**226.32**	374.79
VV	1960.14	821.53	1181.81
HV	687.16	627.29	591.53
1 000 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	4506.53	**1300.98**	4479.42
1TPTS	6931.71	2001.91	2374.05
VV	17912.17	4888.89	10853.68
HV	5779.79	5527.19	5662.57
10 000 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	48830.25	**11713.04**	43597.09
1TPTS	68548.45	19015.88	18880.75
VV	91996.09	29538.13	106428.23
HV	61427.62	61506.70	58158.49

Tabel 46. Operatsiooni I täitmisajad (millisekundites).

100 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	1082.27	1999.78	1102.96
1TPTS	1492.13	2012.61	1758.65
VV	62.12	84.24	**41.08**
HV	144.26	201.28	152.66
1 000 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	910.40	1941.14	1081.68
1TPTS	1475.33	1950.99	1637.00
VV	**66.26**	68.66	70.64
HV	136.39	179.32	146.66
10 000 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	916.58	2031.05	1092.40
1TPTS	1529.46	2067.33	1667.80
VV	**65.56**	91.41	79.75
HV	149.32	193.88	159.44

Tabel 47 on esitatud lausete keerukuse mõõtmise tulemused kasutades koodiridade arvu metoodikat.

Tabel 47. Koodiridade arv erinevate disainilahenduste korral.

	Lause P1	**Lause P2**
NM	34	**21**
1TPTS	**32**	24
VV	35	30
HV	35	30

Tabel 48 on esitatud erinevate disainilahenduste poolt kasutatavad andmemahud megabaitides.

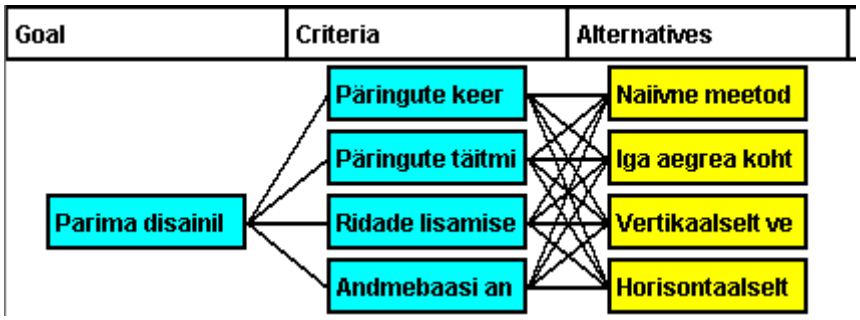
Tabel 48. Andmebaasi maht erinevate disainilahenduste korral (MB).

100 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	7.75	10.95	7.80
1TPTS	7.34	9.86	8.02
VV	**2.83**	2.91	2.88
HV	7.44	7.62	7.50
1 000 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	77.02	108.85	77.07
1TPTS	70.62	93.44	71.30
VV	**28.67**	29.22	28.72
HV	73.82	75.45	73.87
10 000 000 rida	**Ilma indeksiteta**	**B-puu**	**BRIN**
NM	765.54	1081.61	765.62
1TPTS	699.11	924.45	699.16
VV	**286.03**	291.32	286.09
HV	719.83	735.63	719.90

6 Parima disainilahenduse valimine kasutades Saaty meetodit

Saaty meetod ehk analüütiliste hierarhiate meetod (*Analytical Hierarchy Process*) on raamistik, mida kasutatakse valikute tegemisel, et muuta muidu subjektiivsetel hinnangutel põhinevad otsused võimalikult objektiivseks. Meetodi loojaks on Thomas L. Saaty [53]. Meetodi eesmärgiks on leida võimalikest valikutest näiterakenduse jaoks parim ning põhjendada, miks sai selline valik tehtud. Käesolevas töös kasutatakse võrdluste tegemiseks veebipõhist Web-HIPRE tarkvara [54] [55].

Saaty meetodi kasutamiseks tuleb kõigepealt määrata eesmärk, kriteeriumid ja valikud e alternatiivid. Käesolevas töös on eesmärgiks parima andmebaasi disainilahenduse leidmine jaotises 2.3 kirjeldatud näiterakendusele. Kriteeriumiteks on päringute keerukus, päringute täitmise kiirus, ridade lisamise kiirus ja andmebaasi andmemaht. Valikuteks on eksperimendi käigus kasutatud disainilahendused. Probleemi kujutav Saaty hierarhia on välja toodud Joonis 38.



Joonis 38. Saaty hierarhia parima disainilahenduse leidmiseks.

6.1 Kriteeriumite suhteline olulisus

Meetodi järgmisel sammul määratakse kriteeriumite suhteline osatähtsus paarikaupa võrdlemise teel, kus iga kriteeriumit võrreldakse ülejäänud kriteeriumitega ning antakse võrdluse kohta vastav hinnang. Kriteeriumite tähtsusele antud hinnangud põhinevad autori ainuisikulisel arvamusel. Subjektiivsete hinnangute andmisel kasutatakse niinimetatud Saaty fundamentaalskaalat (Tabel 49). Subjektiivsete hinnangute järjepidevuse jälgimiseks on Web-HIPRE rakenduses välja toodud arv kooskõla määr

CM (*consistency measure*) [55]. Veskioja [56] märgib, et CM valem erineb Saaty suhtelise kooskõlaindeksist, aga kasutamine on sama. Seega kui CM väärtus on alla 0.1, siis on tegemist kooskõlaliste hinnangutega. Kõikide järgnevate võrdluste puhul on CM alla 0.1.

Tabel 49. Saaty fundamentaalskaala.

Arvuline väärtus	**Sõnaline hinnang**
1	Võrdtähtis
3	Mõõdukas paremus
5	Oluline paremus
7	Väga tugev paremus
9	Ekstreemne paremus
2, 4, 6, 8 – vahepealsed hinnangud	

Tabel 50 esitab valitud kriteeriumite ning valikute edaspidised tähistused. Tabel 51 esitab kriteeriumite suhtelise olulisuse.

Tabel 50. Kriteeriumite ja valikute tähistused.

Kriteeriumid	**Valikud**
A. Päringute keerukus B. Päringute täitmise kiirus C. Ridade lisamise kiirus D. Andmebaasi andmemaht	E. Naiivne meetod F. Iga aegrea kohta üks tabel G. Vertikaalselt vektoriseerimine H. Horisontaalselt vektoriseerimine

Tabel 51. Kriteeriumite suhteline olulisus.

Mõjur	**A**	**B**	**C**	**D**	**Kaalud**
A	1.0	0.14	0.33	0.17	**0.056**
B	7.0	1.0	4.0	1.0	**0.429**
C	3.0	0.25	1.0	0.33	**0.132**
D	6.0	1.0	3.0	1.0	**0.383**
CM: 0.088					

Autori arvates on kõige olulisemad kriteeriumid parima disainilahenduse leidmisel päringute täitmise kiirus ning andmebaasi maht (Tabel 51). Lausete keerukus pole nii oluline, sest neid ei kirjuta rakenduse lõppkasutaja ning lisamise kiirus pole nii oluline, sest see pole lõppkasutaja poolt läbiviidav operatsioon ning seetõttu teda otseselt selle

töökiirus ei mõjuta (küll aga võib see talle kaudselt probleemiks muutuda, kui ridade lisamine on nii aeglane, et tema käsutuses olevad andmed ei ole enam piisavalt värsked).

6.2 Disainide suhtelise headuse leidmine

Kolmanda sammu käigus võrreldakse iga kasutatava kriteeriumi korral valikuid paariviisiliselt selle kriteeriumi suhtes. Seekord ei ole kaalude väärtusteks autori arvamus, vaid jaotises 5.6 välja toodud eksperimendi tulemused. Kõigi tulemuste puhul on väiksem tulemus parem.

Kriteerium: Päringute keerukus

Disainilahenduste päringute keerukuse hindamiseks liidan päringute keerukused (Tabel 47) kokku. Tabel 52 esitab saadud tulemused.

Tabel 52. Päringute keerukuse tulemused kokku.

Valik	**Lause P1**	**Lause P2**	**Kokku**
E	34	21	**55**
F	32	24	**56**
G	35	30	**65**
H	35	30	**65**

Iga valiku suhtelise headuse leidmiseks kriteeriumi suhtes, leian nende päringute keerukuste suhte, mis omakorda annab päringu keerukuse kriteeriumi hinnangud (Tabel 53).

Tabel 53. Päringute keerukuse kriteeriumi hinnangud.

	E	**F**	**G**	**H**	**Kokku**
E	1.0	1.02	1.18	1.18	**0.272**
F	0.98	1.0	1.16	1.16	**0.267**
G	0.85	0.86	1.0	1.0	**0.230**
H	0.85	0.86	1.0	1.0	**0.230**
CM: 0.001					

Kriteerium: Päringute täitmise kiirus

Disainilahenduste päringute täitmise kiiruse hindamiseks liidan päringute täitmise kiirused (Tabel 44 ja Tabel 45) kokku. Tabel 54 esitab saadud tulemused.

Tabel 54. Päringute täitmise kiirused kokku.

Valik	**Päring P1**	**Päring P2**	**Kokku**
E	198.45 ms	11713.04 ms	**11911.49 ms**
F	192.92 ms	19015.88 ms	**19208.8 ms**
G	456.36 ms	29538.13 ms	**29994.49 ms**
H	260.74 ms	61506.70 ms	**61767.44 ms**

Tabel 55 esitab päringute täitmise kiiruse kriteeriumi hinnanguid ning iga valiku suhtelist headust antud kriteeriumi suhtes.

Tabel 55. Päringute täitmise kiiruse kriteeriumi hinnangud.

	E	**F**	**G**	**H**	**Kokku**
E	1.0	1.61	2.52	5.19	**0.452**
F	0.62	1.0	1.56	3.22	**0.281**
G	0.4	0.64	1.0	2.06	**0.180**
H	0.19	0.31	0.49	1.0	**0.087**
CM: 0.001					

Kriteerium: Ridade lisamise kiirus

Disainilahenduste ridade lisamise kiiruse hindamiseks leian ridade lisamise kiiruste (Tabel 46) keskmise erinevate indeksitüüpide puhul. Tabel 56 esitab saadud tulemused.

Tabel 56. Ridade lisamise kiiruste keskmine.

Valik	**Ilma indeksiteta**	**B-puu**	**BRIN**	**Keskmine**
E	916.58 ms	2031.05 ms	1092.40 ms	**1346.68 ms**
F	1529.46 ms	2067.33 ms	1667.80 ms	**1754.86 ms**
G	65.56 ms	91.41 ms	79.75 ms	**78.91 ms**
H	149.32 ms	193.88 ms	159.44 ms	**167.55 ms**

Tabel 57 esitab ridade lisamise kiiruse kriteeriumi hinnanguid ning iga valiku suhtelist headust antud kriteeriumi suhtes.

Tabel 57. Ridade lisamise kriteeriumi hinnangud.

	E	**F**	**G**	**H**	**Kokku**
E	1.0	1.3	0.11	0.12	**0.054**
F	0.77	1.0	0.11	0.12	**0.047**
G	9.0	9.0	1.0	2.12	**0.544**
H	8.1	8.2	0.47	1.0	**0.354**
CM: 0.087					

Kriteerium: Andmebaasi andmemaht

Disainilahenduste andmebaasi andmemahu hindamiseks leian andmebaasi andmemahtude (Tabel 48) keskmise erinevate indeksitüüpide puhul. Tabel 58 esitab saadud tulemused.

Tabel 58. Andmebaasi andmemahtude keskmine.

Valik	**Ilma indeksiteta**	**B-puu**	**BRIN**	**Keskmine**
E	765.54 MB	1081.61 MB	765.62 MB	**870.92 MB**
F	699.11 MB	924.45 MB	699.16 MB	**774.24 MB**
G	286.03 MB	291.32 MB	286.09 MB	**287.81 MB**
H	719.83 MB	735.63 MB	719.90 MB	**725.12 MB**

Tabel 59 esitab andmebaasi andmemahu kriteeriumi hinnanguid ning iga valiku suhtelist headust antud kriteeriumi suhtes.

Tabel 59. Andmebaasi andmemahtude kriteeriumi hinnangud.

	E	**F**	**G**	**H**	**Kokku**
E	1.0	0.89	0.33	0.83	**0.158**
F	1.12	1.0	0.37	0.93	**0.177**
G	3.0	2.69	1.0	2.52	**0.476**
H	1.2	1.07	0.4	1.0	**0.189**
CM: 0.001					

Mis puudutab seda, et osade paariviisiliste võrdluste juures teostasin tulemuste puhul liitmisoperatsiooni ning osade puhul keskmise leidmise operatsiooni, siis lähtusin sellest, et antud juhul ei ole minu hinnangul vahet, kas liita või leida (artimeetiline) keskmine, sest valiku suhteline headus kriteeriumi suhtes jääb samaks.

Näide ridade lisamise kiiruste põhjal (E ehk naiivne meetod vs F ehk iga aegrea kohta üks tabel) (Tabel 56):

Artimeetilise keskmiste leidmine:

- E: (916.58+2031.05+1092.40) / 3 = 1346.68

- F: (1529.46+2067.33+1667.80) / 3 = 1754.86
- E suhteline headus võrreldes F-ga: 1 / (1346.68/1754.86) = 1.303
- F suhteline headus võrreldes E-ga: 1 / (1754.86/1346.68) = 0.767

Summa leidmine:

- E: 916.58 + 2031.05 + 1092.40 = 4040.03
- F: 1529.46 + 2067.33 + 1667.80 = 5264.59
- E suhteline headus võrreldes F-ga: 1 / (4040.03/5264.59) = 1.303
- F suhteline headus võrreldes E-ga: 1 / (5264.59/4040.03) = 0.767

Seega otseselt mingisugust vahet ei ole liitmisel ja artimeetilise keskmise leidmisel. Töös otsustasin liitmise kasuks päringute täitmise kiiruse juures, sest seal oli arvutuskäik sedasi lühem.

Aritmeetilise keskmise asemel oleks võinud ka leida mediaani. Otsustasin, et antud juhul on aritmeetilise keskmise leidmine piisav, sest katsetatavad operatsioonid ja indeksi valikud on üksteisest sõltumatud, võrdtähtsad ning mõõtmistulemuste hulgas ei ole ekstremaalset erinevaid väärtuseid [57].

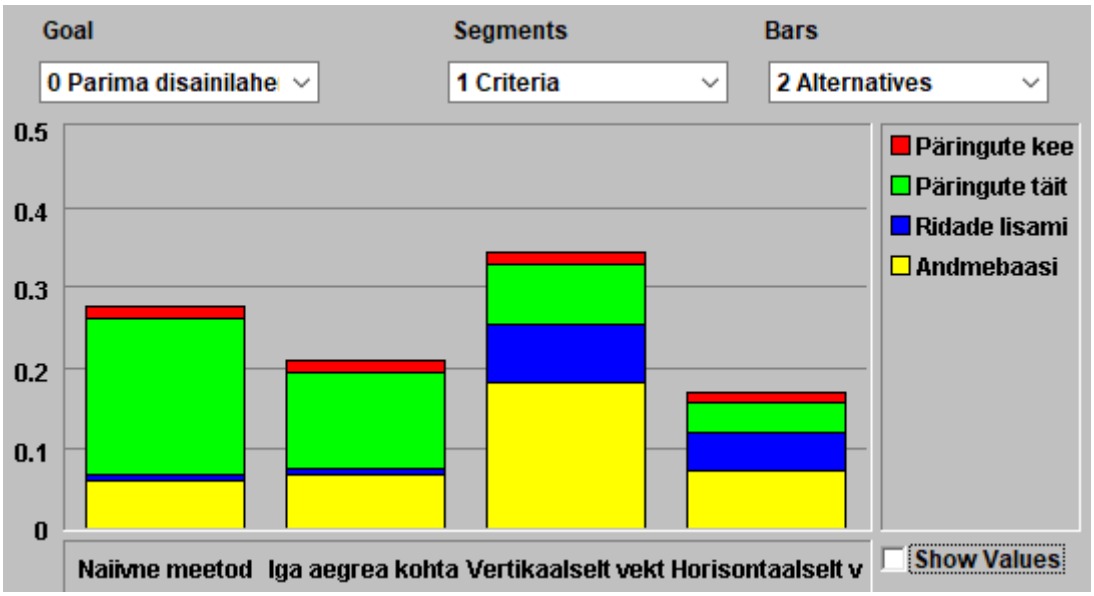
6.3 Tulemuste analüüs

Pärast valikute suhtelisuse headuse leidmist kriteeriumite suhtes, korrutatakse valikute kaalud kriteeriumite kaaludega ning summeeritakse, alustatakse kõige madalamast tasemest järjest ülespoole liikudes ja lõpuks saadakse iga valiku jaoks kogukaal, mis otsustabki paremusjärjestuse. [58] [59]

Saadud lõpptulemused on esitatud Tabel 60 ja Web-HIPRE poolt graafiliselt Joonis 39.

Tabel 60. Saaty meetodi kasutamise lõpptulemus.

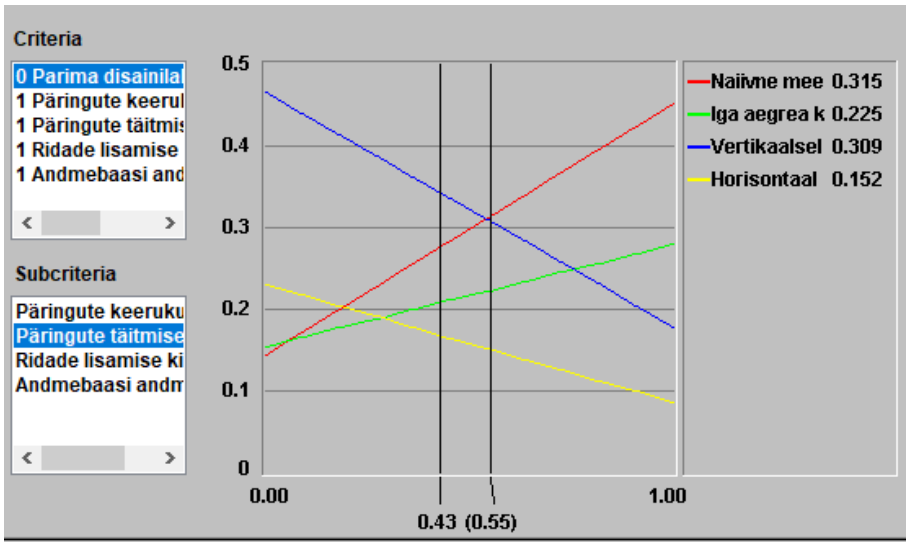
	E	**F**	**G**	**H**
A	0.015	0.015	0.013	0.013
B	0.194	0.120	0.077	0.037
C	0.007	0.006	0.072	0.047
D	0.060	0.068	0.182	0.072
Kokku	**0.277**	**0.209**	**0.344**	**0.170**



Joonis 39. Saaty meetodi kasutamise lõpptulemus graafiliselt.

Tulemustest selgub, et valitud kriteeriumite ning nende olulisuse järgi sobib kõige paremini disainilahendus „vertikaalselt vektoriseerimine“. Edu teiste disainilahenduste ees tuleneb disainilahenduse väikesest andmebaasi andmemahust.

Paremuselt teise koha sai disainilahendus „naiivne meetod“, mis näitas eksperimendis testitud andmemahtude korral häid päringute kiirusi. Tundlikkuse analüüsis näitab, et tõstes päringute täitmise kiiruste kriteeriumi olulisust 0.55-ni tõuseb disainilahendus esikohale (Joonis 40). Kriteeriumi olulisuse vähenedes, aga vertikaalselt vektoriseerimise edumaa teiste ees kasvab, mis tähendab head sobivust süsteemidele, mida iseloomustab pigem suurte andmehulkade salvestamine ja väiksem päringute arv.

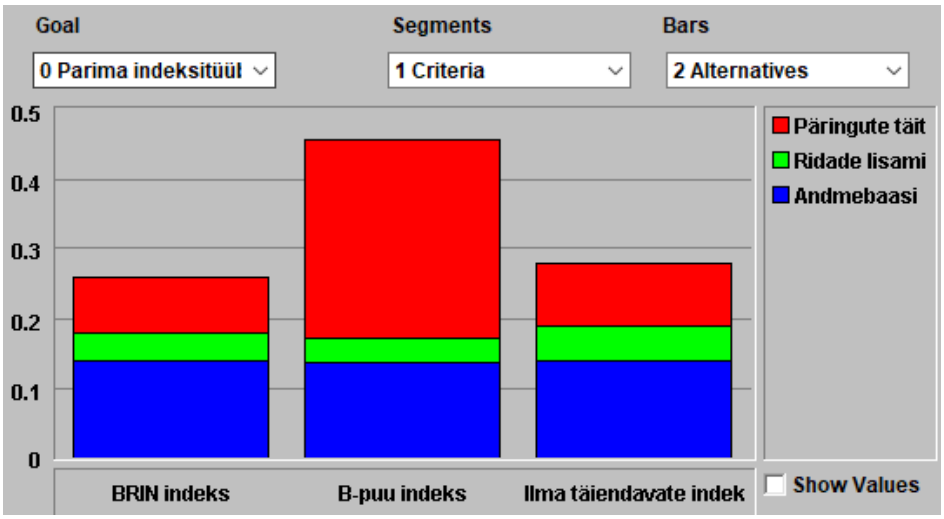


Joonis 40. Päringute täitmise kiiruse tundlikkuse analüüs.

Parima indeksitüübi leidmiseks disainilahendusele „vertikaalselt vektoriseerimine“ viin läbi uue analüüsi kasutades Saaty meetodit. Valikuteks on disainilahenduse kasutamine ilma täiendavate indeksiteta, BRIN indeksiga ning B-puu indeksiga. Meetodi käigus kasutatud kriteeriumitest eemaldatakse päringute keerukuse kriteerium, sest erinevate indeksitüüpide puhul päringute keerukus ei muutu. Analüüsi käigus kasutatud kriteeriumite suhteline olulisus muudel kriteeriumitel jääb samaks. Ruumi kokkuhoiu huvides on analüüsi võrdlusmaatriksid (kriteeriumite olulisuse määramiseks ning valikute hindamiseks kriteeriumite suhtes) esitatud Lisa 7. Analüüsi tulemused on toodud Tabel 61 (graafiliselt Joonis 41), kust on näha, et disainilahenduse „vertikaalselt vektoriseeritud“ puhul oli parimaks indeksitüübiks B-puu indeks, mis oli teistest üle päringute täitmise kiirusega ning ei olnud oluliselt halvem kasutatud andmemahu ega ridade lisamise kiiruse suhtes.

Tabel 61. Saaty meetodi kasutamise lõpptulemus parima indeksitüübi leidmisel.

	BRIN indeks (D)	**B-puu indeks (E)**	**Ilma täiendavate indeksiteta (F)**
Päringute täitmise kiirus (A)	0.080	0.286	0.092
Ridade lisamise kiirus (B)	0.041	0.036	0.049
Andmebaasi andmemaht (C)	0.140	0.137	0.140
Kokku	**0.260**	**0.459**	**0.281**



Joonis 41. Saaty meetodi kasutamise tulemus parima indeksitüübi leidmisel disainilahendusele „vertikaalselt vektoriseeritud“.

Kokkuvõte

Aegridade salvestamiseks SQL-andmebaasis on olemas erinevaid disainilahendusi. Andmebaasi loomisel tuleb hoolikalt läbi mõelda, kuidas andmebaas disainida. Paraku ei leidnud autor ühtegi varasemalt tehtud uuringut, kus oleks võrreldud mitmeid disainilahendusi ning toodud välja nende nõrgad/tugevad küljed ja erinevate ülesannete lahendamiseks mõeldud andmekäitluse operatsioonide täitmise kiirused.

Käesoleva magistritöö esimeseks eesmärgiks oli kirjeldada erinevaid aegridade esitamiseks mõeldud disainilahendusi SQL-andmebaasisüsteemides, kasutades disainide parema loetavuse ja võrreldavuse huvides nende esitamiseks ühtset mustri formaati. Magistritöö teiseks eesmärgiks oli läbi viia eksperiment PostgreSQL (11) andmebaasisüsteemis erinevate disainilahenduste võrdlemiseks. Magistritöö kolmandaks eesmärgiks oli leida eksperimendis käsitletud disainilahenduste seast parim disainilahendus näiterakendusele, mis keskendub väärtpaberite hinna liikumiste statistikale.

Töö disainilahenduste osas anti ülevaade erinevate disainilahendustest. Lugejale parema ülevaate andmiseks kasutati ühtset mustri formaati, mis hõlmas disainilahenduse lühikirjeldust, allikate viiteid, tugevaid ja nõrki külgi ning ning näidet disainilahenduse realiseerimisest OHLCV andmete salvestamiseks. Näites toodi välja kõikide diainide puhul ühesuguse aegrea andmete esitus, füüsilise disaini andmemudel PostgreSQL jaoks, tabelite ja tabelite veergude kirjeldused ning PostgreSQLi SQL mägimurraku laused disainilahenduse realiseerimiseks. Kokku kirjeldati kaheksa erinevat disainilahendust.

Töö teises ehk eksperimendi osas toodi välja eksperimendi eesmärk, metoodika ja kirjeldati testandmete genereerimist. Eksperiment viidi läbi nelja disainiga, erinevate andmehulkadega ning disaine katsetati koos erinevate insekseerimise lahendustega. Pärast eksperimendi läbiviimist tulemused dokumenteeriti. Tulemustes tuuakse välja erinevate disainilahenduste põhjal loodud andmebaasi päringute ja lisamisoperatsiooni täitmise kiirused erinevate andmehulkadega ning indekseerimise lahendustega. Samuti

tuuakse välja tabelite andmemaht erinevate andmehulkadega ning indekseerimise lahendustega. Lisaks esitatakse hinnang päringute lausete keerukusele.

Töö kolmandas osas otsiti parimat andmebaasi disainilahendust näiterakendusele, mis tegeleb väärtpaberite hinna liikumiste statistika loomisega. Parima disainilahenduse valimiseks kasutati töös Saaty ehk analüütiliste hierarhiate meetodit.

Töös korraldatud eksperimendi tulemustest selgus, et õige disainilahenduse ja indeksitüübi valimisel on operatsioonide väikese töökiiruse suurenemise hinnaga võimalik oluliselt vähendada tabelite poolt kasutatavat andmemahtu. Lisaks võib järeldada, et päringute tegemiseks kulunud aeg ja ridade sisestamise kiirus oleneb tabelis olevatest ridade arvust ja kasutatavast indekstitüübist.

Saaty meetodi kasutamisel selgus, et parimaks andmebaasi disainilahenduseks näiterakendusele PostgreSQL (11) kasutamise korral oleks „vertikaalselt vektoriseeritud“ lahendus, mille edu seisnes, võrreldes teiste disainilahendustega, tabelite poolt vähesest kasutatud andmemahust. Disainilahenduse „vertikaalselt vektoriseeritud“ puhul oli parimaks indeksitüübiks B-puu indeks (võrrelduna täiendavate indeksite mitte loomisega ja BRIN indeksite kasutamisega). Selle indeksitüübi kasutamine oli alternatiividest üle päringute täitmise kiirusega ning ei olnud oluliselt halvem kasutatud andmemahu ega ridade lisamise kiiruse suhtes.

Kõik töö käigus kasutatud SQL laused (päringute tegemiseks, ridade sisestamiseks, disainilahenduste realiseerimiseks, indeksite lisamiseks) on salvestatud GitHubi hoidlasse (https://github.com/176660/SQL_aegread) [52].

Üheks võimalikuks töö edasiarenduseks oleks lisaks erinevatele indeksitüüpidele ja andmemahtudele eksperimenteerida veel tabelite sektsioonimisega (*partitioning*) ja andmebaasisüsteemi konfiguratsiooniga. Teiseks võimalikuks töö edasiarenduseks oleks võrrelda käesolevas töös saadud tulemusi mõne spetsialiseerunud andmebaasisüsteemiga. Kolmandaks võimalikuks töö edasiarenduseks oleks võrrelda selles töös leitud tulemusi mõne teise SQL-andmebaasisüsteemiga, näiteks Oracle või MySQL. Veel üheks edasiarenduseks oleks töös mainitud näiterakenduse tegelik realiseerimine.

Kasutatud kirjandus

[1] solid IT, „DB Engines Ranking,“ solid IT, 1 märts 2019. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking. [Kasutatud 20 märts 2019].

[2] Fabio Pardi, „A Timeseries Case Study: InfluxDB VS PostgreSQL to store data,“ [Võrgumaterjal]. Available: https://portavita.github.io/2018-07-31-blog_influxdb_vs_postgresql. [Kasutatud 4 mai 2019].

[3] A. R. Hevner, S. T. March ja J. Park, Design Science in Information Systems Research, MIS Quarterly, 2004.

[4] OECD, „OECD Glossary of Statistical Terms - Time series Definition,“ OECD, 25 september 2001. [Võrgumaterjal]. Available: https://stats.oecd.org/glossary/detail.asp?ID=2708. [Kasutatud 22 märts 2019].

[5] B. Bollverk, „AEGRIDADE ANALÜÜS STATISTIKAPAKETIS SPSS,“ 2001. [Võrgumaterjal]. Available: http://www.cs.tlu.ee/~katrin/wp/wp-content/uploads/2013/11/algread.pdf. [Kasutatud 18 märts 2019].

[6] UBILAB, Union Bank of Switzerland, „An Object-Oriented Data Model for a Time Series Management System,“ [Võrgumaterjal]. Available: https://pdfs.semanticscholar.org/57ec/96db766b9fe4e7329e367cc0f31d87b2b3cb.pdf. [Kasutatud 2 mai 2019].

[7] „database - XML Schema for scientific instrumentation time-series logging - Stack Overflow,“ [Võrgumaterjal]. Available: https://stackoverflow.com/questions/27394878 /xml-schema-for-scientific-instrumentation-time-series-logging. [Kasutatud 2 mai 2019].

[8] „time series - How to store timed data points in sql server - Stack Overflow,“ [Võrgumaterjal]. Available: https://stackoverflow.com/a/27232727. [Kasutatud 2 mai 2019].

[9] M. Fowler, „NosqlDefinition,“ 9 jaanuar 2012. [Võrgumaterjal]. Available: https://martinfowler.com/bliki/NosqlDefinition.html. [Kasutatud 20 märts 2019].

[10] MongoDB, „Schema Validation - MongoDB Manual,“ [Võrgumaterjal]. Available: https://docs.mongodb.com/manual/core/schema-validation/. [Kasutatud 2 mai 2019].

[11] M. Fowler, „Schemaless Data Structures,“ 7 jaanuar 2013. [Võrgumaterjal]. Available: https://martinfowler.com/articles/schemaless. [Kasutatud 2 mai 2019].

[12] Wikipedia, „InfluxDB,“ märts 2019. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/InfluxDB. [Kasutatud 23 märts 2019].

[13] InfluxData, „InfluxDB key concepts,“ InfluxData, [Võrgumaterjal]. Available: https://docs.influxdata.com/influxdb/v1.7/concepts/key_concepts/. [Kasutatud 23 märts 2019].

[14] V. Nigam, „Processing Time Series Data in Real-Time with InfluxDB and Structured Streaming,“ 15 detsember 2018. [Võrgumaterjal]. Available:

https://towardsdatascience.com/processing-time-series-data-in-real-time -with-influxdb-and-structured-streaming-d1864154cf8b. [Kasutatud 2 mai 2019].

[15] InfluxData, „InfluxDB design insights and tradeoffs,“ InfluxData, [Võrgumaterjal]. Available: https://docs.influxdata.com/influxdb/v1.7/concepts/insights_tradeoffs/. [Kasutatud 23 märts 2019].

[16] InfluxData, „InfluxQL functions,“ InfluxData, [Võrgumaterjal]. Available: https://docs.influxdata.com/influxdb/v1.7/query_language/functions/. [Kasutatud 23 märts 2019].

[17] InfluxData, „InfluxDB HTTP API reference,“ InfluxData, [Võrgumaterjal]. Available: https://docs.influxdata.com/influxdb/v1.7/tools/api/. [Kasutatud 23 märts 2019].

[18] InfluxData, „InfluxDB compared to SQL databases,“ [Võrgumaterjal]. Available: https://docs.influxdata.com/influxdb/v1.7/concepts/crosswalk/. [Kasutatud 4 mai 2019].

[19] InfluxData, „InfluxDB Clustering,“ InfluxData, [Võrgumaterjal]. Available: https://www.influxdata.com/blog/influxdb-clustering/. [Kasutatud 25 märts 2019].

[20] E. Eessaar, Andmebaaside projekteerimine, Tallinn: Tallinna Tehnikaülikooli Kirjastus, 2008.

[21] TimescaleDB, „TimescaleDB Docs | Architecture,“ TimescaleDB, [Võrgumaterjal]. Available: https://docs.timescale.com/v1.2/introduction/architecture. [Kasutatud 25 märts 2019].

[22] „timescaledb/CHANGELOG.md at master · timescale/timescaledb · GitHub,“ [Võrgumaterjal]. Available: https://github.com/timescale/timescaledb/blob/master/CHANGELOG.md. [Kasutatud 25 märts 2019].

[23] TimescaleDB, „TimescaleDB Docs | Comparision: PostgreSQL,“ TimescaleDB, [Võrgumaterjal]. Available: https://docs.timescale.com/v1.2/introduction/timescaledb-vs-postgres. [Kasutatud 25 märts 2019].

[24] „TimescaleDB Docs | Comparision: NoSQL,“ [Võrgumaterjal]. Available: https://docs.timescale.com/v1.2/introduction/timescaledb-vs-nosql. [Kasutatud 25 märts 2019].

[25] M. Fowler, „PolyglotPersistence,“ 16 november 2011. [Võrgumaterjal]. Available: https://martinfowler.com/bliki/PolyglotPersistence.html. [Kasutatud 2 mai 2019].

[26] PostgreSQL, „A Brief History of PostgreSQL,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/history.html. [Kasutatud 5 mai 2019].

[27] PostgreSQL, „Window Functions,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/11/tutorial-window.html. [Kasutatud 4 mai 2019].

[28] Wikipedia, „PostgreSQL,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/PostgreSQL. [Kasutatud 4 mai 2019].

[29] Yahoo Finance, „Alphabet Inc. (GOOG) Interactive Stock Chart,“ Yahoo Finance, [Võrgumaterjal]. Available: https://finance.yahoo.com/quote/GOOG/chart?p=GOOG. [Kasutatud 30 märts 2019].

[30] A. M. Castillejos, „Management of time series data,“ 2006. [Võrgumaterjal]. Available: http://www.canberra.edu.au/researchrepository/ file/82315cf7-7446-fcf2-6115-b94fbd7599c6/1/full_text.pdf. [Kasutatud 20 märts 2019].

[31] T. Lane, „PostgreSQL: Re: Maximum Number of Tables in a database,“ [Võrgumaterjal]. Available: https://www.postgresql.org/message-id/ 18728.1027611113@sss.pgh.pa.us. [Kasutatud 8 mai 2019].

[32] B. Karwin, SQL Antipatterns: Avoiding the Pitfalls of Database Programming, Dallas, Texas: Pragmatic Bookshelf, 2010.

[33] G. Trubetskoy, „Storing Time Series in PostgreSQL efficiently,“ september 2015. [Võrgumaterjal]. Available: https://grisha.org/blog/2015/09/23/ storing-time-series-in-postgresql-efficiently/. [Kasutatud 20 märts 2019].

[34] G. Trubetskoy, „Storing Time Series in PostgreSQL (Continued),“ detsember 2016. [Võrgumaterjal]. Available: https://grisha.org/blog/2016/12/16/ storing-time-series-in-postgresql-part-ii/. [Kasutatud 20 märts 2019].

[35] G. Trubetskoy, „Storing Time Series in PostgreSQL - Optimize for Write,“ jaanuar 2017. [Võrgumaterjal]. Available: https://grisha.org/blog/2017/01/21/ storing-time-seris-in-postgresql-optimize-for-write/. [Kasutatud 20 märts 2019].

[36] E. Horner, „Vector denormalisation in PostgreSQL,“ veebruar 2012. [Võrgumaterjal]. Available: https://ejrh.wordpress.com/2011/03/20/ vector-denormalisation-in-postgresql/. [Kasutatud 20 märts 2019].

[37] E. Horner, „Vector re-normalisation with views in PostgreSQL,“ 9 veebruar 2012. [Võrgumaterjal]. Available: https://ejrh.wordpress.com/2012/02/9/ vector-re-normalisation-with-views-in-postgresql/. [Kasutatud 20 märts 2019].

[38] P. Allsopp, „BRIN indexes, what are they and how do you use them?,“ 10 oktoober 2017. [Võrgumaterjal]. Available: https://www.postgresql.fastware.com/blog/ brin-indexes-what-are-they-and-how-do-you-use-them. [Kasutatud 3 märts 2019].

[39] Wikipedia, „Block Range Index,“ Wikipedia, detsember 2018. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Block_Range_Index. [Kasutatud 23 märts 2019].

[40] S. M. Mikk, „Graafide esitamine SQL-andmebaasides,“ 2017. [Võrgumaterjal]. Available: https://digi.lib.ttu.ee/i/?7979. [Kasutatud 27 märts 2019].

[41] A. Eisenberg ja J. Melton, „SQL:1999, formerly known as SQL3,“ [Võrgumaterjal]. Available: https://users.dcc.uchile.cl/~cgutierr/cursos/BD/standards.pdf. [Kasutatud 11 mai 2019].

[42] w3resource, „MySQL Data Types,“ [Võrgumaterjal]. Available: https://www.w3resource.com/mysql/mysql-data-types.php. [Kasutatud 11 mai 2019].

[43] PostgreSQL, „Documentation: 11: 68.2. TOAST,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/storage-toast.html. [Kasutatud 11 mai 2019].

[44] PostgreSQL, „PostgreSQL: Documentation: 11: EXPLAIN,“ PostgreSQL, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/11/sql-explain.html. [Kasutatud 20 aprill 2017].

[45] C. Gallant, „The Difference Between Arithmetic Mean and Geometric Mean,“ [Võrgumaterjal]. Available:

https://www.investopedia.com/ask/answers/06/geometricmean.asp. [Kasutatud 2 mai 2019].

[46] M. B.N, „Understanding caching in Postgres - An in-depth guide,“ [Võrgumaterjal]. Available: https://madusudanan.com/blog/understanding-postgres-caching-in-depth/. [Kasutatud 2 mai 2019].

[47] Wikipedia, „Source lines of code,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Source_lines_of_code. [Kasutatud 2 mai 2019].

[48] Mozilla, „SQL Style Guide,“ [Võrgumaterjal]. Available: https://docs.telemetry.mozilla.org/concepts/sql_style.html. [Kasutatud 2 mai 2019].

[49] PostgresTutorial, „How to Get Table, Database, Indexes, Tablespace, and Value Size in PostgreSQL,“ [Võrgumaterjal]. Available: http://www.postgresqltutorial.com/ postgresql-database-indexes-table-size/. [Kasutatud 2 mai 2019].

[50] FirstRate Data, „FirstRate Data | Historical Intraday Market Price Data,“ FirstRate Data, [Võrgumaterjal]. Available: http://firstratedata.com/. [Kasutatud 16 aprill 2019].

[51] PostgreSQL, „PostgreSQL: Documentation: 11: ANALYZE,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/11/sql-analyze.html. [Kasutatud 2 mai 2019].

[52] K. Jaakson, „Github - 176660/magistritoo,“ [Võrgumaterjal]. Available: https://github.com/176660/magistritoo. [Kasutatud 5 mai 2019].

[53] Wikipedia, „Analytic hierarchy process,“ Wikipedia, veebruar 2019. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Analytic_hierarchy_process. [Kasutatud 20 märts 2019].

[54] „Web-HIPRE,“ [Võrgumaterjal]. Available: http://hipre.aalto.fi/. [Kasutatud 26 märts 2019].

[55] J. Mustajoki ja R. P. Hämäläinen, „WEB-HIPRE - A JAVA APPLET FOR AHP AND VALUE TREE ANALYSIS,“ [Võrgumaterjal]. Available: http://citeseerx.ist.psu.edu/viewdoc/ download?doi=10.1.1.16.5368&rep=rep1&type=pdf. [Kasutatud 11 mai 2019].

[56] T. Veskioja, „Web-HIPRE juhend,“ [Võrgumaterjal]. Available: https://maurus.ttu.ee/ained/IDN5120/doc/20/Web_Hipre_juhend.html. [Kasutatud 11 mai 2019].

[57] Macroption, „Arithmetic Average: When to Use It and When Not,“ [Võrgumaterjal]. Available: https://www.macroption.com/arithmetic-average-when-use/. [Kasutatud 8 mai 2019].

[58] K. Lauk, „Veebipõhiste UML CASE-vahendite võrdlus,“ 2013. [Võrgumaterjal]. [Kasutatud 26 märts 2019].

[59] M. Soobik, „Andmebaasi disainide hindamine Saaty meetodi abil,“ 2009. [Võrgumaterjal]. Available: http://maurus.ttu.ee/ained/IDU0220_IDU0230/ doc/26/Andmebaasi_disainide_hindamine_Saaty_meetodi_abil.pdf. [Kasutatud 26 märts 2019].

[60] International Hellenic University, „Lesson 1 - The Analytic Hierarchy Process (AHP),“ [Võrgumaterjal]. Available: http://rad.ihu.edu.gr/fileadmin/labsfiles/

decision_support_systems/lessons/ahp/AHP_Lesson_1.pdf. [Kasutatud 2 mai 2019].

[61] A. R. Hevner, S. T. March ja J. Park, „Design Science in Information Systems Research,“ *MIS Quarterly*, kd. 28, nr 1, p. 31, 2004.

Lisa 1 – Varasemate uuringute leidmiseks kasutatud otsingusõnad ja -mootorid

Otsingumootor	**Otsingufraas**
Google Scholar	storing time series postgresql
Google Scholar	storing time series relational database
Google Scholar	temporal data relational database -nosql
Google Scholar	temporal data postgresql -nosql
Google Scholar	temporal data sql
Google Scholar	time series database design
Google Scholar	temporal data database design
Google Scholar	time series database design comparison
Google Scholar	temporal data database design comparison
Google Scholar	relational database design “time series”
Google Scholar	time series schema design
Google Scholar	temporal data schema design
Google	storing time series postgresql
Google	storing time series relational database
Google	temporal data relational database -nosql
Google	temporal data postgresql -nosql
Google	temporal data sql
Google	time series database design
Google	temporal data database design
Google	time series database design comparison
Google	temporal data database design comparison
Google	relational database design “time series”
Google	time series schema design
Google	temporal data schema design

Lisa 2 – Testandmete töötlemise programmikood

FirstRate Data veebilehekülje andmed on jaotatud kuuekuuliste intervallidega erinevatesse failidesse, nende failide kokku liitmiseks on kasutatud järgmist Pythoni programmikoodi.

```
import os
import pandas as pd

col_names = ["time", "open", "high", "low", "close", "volume", "trades",
"weighted_avg"]
base_path = "./datasets"

for path, directories, files in list(os.walk(base_path))[1:]:
    frames = []
    for file in files:
        temp_frame = pd.read_csv(
            filepath_or_buffer=f"{path}/{file}",
            header=None,
            names=col_names,
            parse_dates=[0],
            infer_datetime_format=True,
            index_col=[0],
        )
        frames.append(temp_frame)

    frame = pd.concat(frames)
    frame.to_csv(f"{path} combined.csv")
```

Andmetes lünkade täitmiseks ning soovitud andmemahtudega failide genereerimiseks kasutati järgmist Pythoni programmikoodi.

```
import os
import pandas as pd

path = "./combined/"
resample_opts = {
    'open': 'first',
    'high': 'max',
    'low': 'min',
    'close': 'last',
    'volume': 'sum'
}

(_, _, filenames) = next(os.walk(path))
frames = []
for filename in filenames:
    frame = pd.read_csv(
        filepath_or_buffer=f"{path}/{filename}",
        parse_dates=[0],
        infer_datetime_format=True,
        index_col=[0],
    )
    frame.drop(columns=['weighted_avg', 'trades'], inplace=True)
    frame = frame.resample('1Min').agg(resample_opts).ffill()
    frame = frame.loc['2006-01-08 12:00:00': '2018-12-26 11:59:00']
    frame['ticker'] = filename.split(".")[0]

    # optimize data types
    float_cols = ['open', 'high', 'low', 'close']
    int_cols = ['volume']
    frame[float_cols] = frame[float_cols].astype('float32')
    frame[int_cols] = frame[int_cols].astype('int32')
    frames.append(frame)

frame = pd.concat(frames)
frame.sort_index(inplace=True)
frame.to_csv('100mil.csv')
```

Lisa 3 – Andmete lugemise (P1) laused

Naiivne meetod (jaotis 4.1)

```
EXPLAIN ANALYZE
WITH intervals AS
(
   SELECT
      start::TIMESTAMP,
      start::TIMESTAMP + INTERVAL '15min' AS end
   FROM
      generate_series('2018-12-21 00:00:00', '2018-12-28 00:00:00', INTERVAL
'15min') AS start
)
SELECT DISTINCT
   ticker,
   intervals.end AS date,
   FIRST_VALUE(open) OVER w AS open,
   MAX(high) OVER w AS high,
   MIN(low) OVER w AS low,
   LAST_VALUE(close) OVER w AS close,
   SUM(volume) OVER w AS volume
FROM
   intervals
JOIN
   naive_method AS nm
   ON nm.ticker = 'AMZN'
   AND nm.date > intervals.start
   AND nm.date <= intervals.end
   WINDOW w AS
   (
      PARTITION BY
             intervals.end
      ORDER BY
             nm.date ASC
      ROWS BETWEEN unbounded preceding AND unbounded following
   )
ORDER BY
   intervals.end;
```

Iga aegrea kohta üks tabel (jaotis 4.2)

```
EXPLAIN ANALYZE
WITH intervals AS
(
   SELECT
      start::TIMESTAMP,
      start::TIMESTAMP + INTERVAL '15min' AS end
   FROM
      generate_series('2018-12-21 00:00:00', '2018-12-28 00:00:00', INTERVAL
'15min') AS start
)
SELECT DISTINCT
   intervals.end AS date,
   FIRST_VALUE(open) OVER w AS open,
   MAX(high) OVER w AS high,
   MIN(low) OVER w AS low,
   LAST_VALUE(close) OVER w AS close,
   SUM(volume) OVER w AS volume
FROM
   intervals
   JOIN
      stock_amzn AS mb
      ON mb.date > intervals.start
      AND mb.date <= intervals.end
      WINDOW w AS
      (
         PARTITION BY
             intervals.end
          ORDER BY
             mb.date ASC
          ROWS BETWEEN unbounded preceding AND unbounded following
      )
ORDER BY
   intervals.end;
```

Horisontaalselt vektoriseerimine (jaotis 4.6)

```
EXPLAIN ANALYZE
WITH intervals AS
(
   SELECT
      start::TIMESTAMP,
      start::TIMESTAMP + INTERVAL '15min' AS end
   FROM
      generate_series('2018-12-21 00:00:00', '2018-12-28 00:00:00', INTERVAL
'15min') AS start
)
SELECT DISTINCT
   ticker,
   intervals.end AS date
   FIRST_VALUE(open) OVER w AS open,
   MAX(high) OVER w AS high,
   MIN(low) OVER w AS low,
   LAST_VALUE(close) OVER w AS close,
   SUM(volume) OVER w AS volume
FROM
   intervals
   JOIN
      stocks_view AS mb
      ON mb.ticker = 'AMZN'
      AND mb.first_date = date_trunc('hour', intervals.start)
      AND mb.date > intervals.start
      AND mb.date <= intervals.end
      WINDOW w AS
      (
         PARTITION BY
             intervals.end
          ORDER BY
             mb.date ASC
          ROWS BETWEEN unbounded preceding AND unbounded following
      )
ORDER BY
   intervals.end;
```

Vertikaalselt vektoriseerimine (jaotis 4.7)

```
EXPLAIN ANALYZE
WITH intervals AS
(
   SELECT
      start::TIMESTAMP,
      start::TIMESTAMP + INTERVAL '15min' AS end
   FROM
      generate_series('2018-12-21 00:00:00', '2018-12-28 00:00:00', INTERVAL
'15min') AS start
)
SELECT DISTINCT
   ticker,
   intervals.end AS date,
   FIRST_VALUE(open) OVER w AS open,
   MAX(high) OVER w AS high,
   MIN(low) OVER w AS low,
   LAST_VALUE(close) OVER w AS close,
   SUM(volume) OVER w AS volume
FROM
   intervals
   JOIN
      stocks_view AS mb
      ON mb.ticker = 'AMZN'
      AND mb.first_date = date_trunc('hour', intervals.start)
      AND mb.date > intervals.start
      AND mb.date <= intervals.end
      WINDOW w AS
      (
         PARTITION BY
             intervals.end
          ORDER BY
             mb.date ASC
          ROWS BETWEEN unbounded preceding AND unbounded following
      )
ORDER BY
   intervals.end;
```

Lisa 4 – Andmete lugemise (P2) laused

Naiivne meetod (jaotis 4.1)

```
EXPLAIN ANALYZE
WITH max_date AS
(
   SELECT
      ticker,
      MAX(date) AS date
   FROM
      naive_method
   GROUP BY
      ticker
)
SELECT
   nm.ticker,
   nm.date,
   nm.close as price
FROM
   max_date
INNER JOIN
   naive_method AS nm
   ON nm.ticker = max_date.ticker
   AND nm.date = max_date.date;
```

Iga aegrea kohta üks tabel (jaotis 4.2)

```
EXPLAIN ANALYZE
WITH max_date AS
(
   SELECT
      stocks.tableoid AS table_id,
      MAX(date) AS date
   FROM
      stocks
   GROUP BY
      table_id
)
SELECT
   UPPER(SUBSTRING(relname FROM '_(.*?)_')) AS ticker,
   s.date,
   s.close as price
FROM
   max_date
INNER JOIN
   stocks AS s
   ON x.tableoid = max_date.table_id
   AND x.DATE = max_date.date
LEFT JOIN
   pg_class AS pg
   ON pg.oid = max_date.table_id;
```

Horisontaalselt vektoriseerimine (jaotis 4.6)

```
EXPLAIN ANALYZE
WITH max_date AS
(
   SELECT
      ticker,
      MAX(first_date) AS date
   FROM
      stocks_view
   GROUP BY
      ticker
)
SELECT DISTINCT
   sv.ticker,
   FIRST_VALUE(sv.date) OVER w AS date,
   FIRST_VALUE(sv.close) OVER w AS price
FROM
   max_date AS md
LEFT JOIN
   stocksview AS sv
   ON sv.ticker = md.ticker
   AND sv.first_date = md.date
   WINDOW w AS
      (
         PARTITION BY
            sv.ticker,
            sv.first_date
         ORDER BY
            sv.date DESC
         ROWS BETWEEN unbounded preceding AND unbounded following
      );
```

Vertikaalselt vektoriseerimine (jaotis 4.7)

```
EXPLAIN ANALYZE
WITH max_date AS
(
   SELECT
      ticker,
      MAX(first_date) AS date
   FROM
      stocks_view
   GROUP BY
      ticker
)
SELECT DISTINCT
   sv.ticker,
   FIRST_VALUE(sv.date) OVER w AS date,
   FIRST_VALUE(sv.close) OVER w AS close
FROM
   max_date AS md
   LEFT JOIN
      stocks_view AS sv
      ON sv.ticker = md.ticker
      AND sv.first_date = md.date
      WINDOW w AS
         (
            PARTITION BY
                sv.ticker,
                 sv.first_date
            ORDER BY
                sv.date DESC
           ROWS BETWEEN unbounded preceding AND unbounded following
        );
```

Lisa 5 – Andmete lisamise (I) laused

Naiivne meetod (jaotis 4.1)

```
BEGIN;
EXPLAIN ANALYZE
INSERT INTO
	stocks (date, ticker, open, high, low, close, volume)
SELECT
	date,
	ticker,
	open,
	high,
	low,
	close,
	volume
FROM
	data_for_inserts AS df
ORDER BY
	df.date ASC,
	ticker ASC;
ROLLBACK;
```

Iga aegrea kohta üks tabel (jaotis 4.2)

```
BEGIN;
EXPLAIN ANALYZE
WITH insert1 AS (
	INSERT INTO
		stock_a
	SELECT
		date,
		open,
		high,
		low,
		close,
		volume
	FROM data_for_inserts
	WHERE ticker = 'A'
	ORDER BY
		date ASC
), insert2 as (
	INSERT INTO
		stock_aap
	SELECT
		date,
		open,
		high,
		low,
		close,
		volume
	FROM data_for_inserts
	WHERE ticker = 'AAP'
	ORDER BY
		date ASC
)
-- for every table
INSERT INTO
	stock_ads
SELECT
	date,
	open,
	high,
	low,
	close,
	volume
FROM data_for_inserts
WHERE ticker = 'ADS'
ORDER BY
	date ASC;
ROLLBACK;
```

Horisontaalselt vektoriseerimine (jaotis 4.6)

```
BEGIN;
EXPLAIN ANALYZE
INSERT INTO
	stocks_timeseries (date, open, high, low, close, volume)
SELECT
	date,
	array_agg(open) as open,
	array_agg(high) as high,
	array_agg(low) as low,
	array_agg(close) as close,
	array_agg(volume) as volume
FROM
	data_for_inserts
GROUP BY
	date
ORDER BY
	date;
ROLLBACK;
```

Vertikaalselt vektoriseerimine (jaotis 4.7)

```
BEGIN;
EXPLAIN ANALYZE
INSERT INTO
	stocks (first_date, ticker, open, high, low, close, volume)
SELECT
	date,
	ticker,
	open,
	high,
	low,
	close,
	volume
FROM
	data_for_inserts
ORDER BY
	date ASC,
	ticker ASC;
ROLLBACK;
```

Lisa 6 – Eksperimendis kasutatud indeksite loomise laused

Naiivne meetod

B-puu indeksi loomise lause:

```
CREATE INDEX stocks_ticker_date_idx
    ON stocks USING btree
    (ticker ASC NULLS LAST, date DESC NULLS LAST);
```

BRIN indeksi loomise lause:

```
CREATE INDEX stocks_ticker_date_idx
    ON stocks USING brin
    (ticker, date);
```

Iga aegrea kohta üks tabel

B-puu indeksite loomise lause:

```
CREATE INDEX stock_amzn_date_idx
    ON stocks_amzn USING btree
    (date DESC NULLS LAST);

CREATE INDEX stock_googl_date_idx
    ON stocks_googl USING btree
    (date DESC NULLS LAST);

CREATE INDEX stock_fb_date_idx
    ON stocks_fb USING btree
    (date DESC NULLS LAST);
```

BRIN indeksite loomise lause:

```
CREATE INDEX stock_amzn_date_idx
    ON stocks_amzn USING brin
    (date);

CREATE INDEX stock_googl_date_idx
    ON stocks_googl USING brin
    (date);

CREATE INDEX stock_fb_date_idx
    ON stocks_fb USING brin
    (date);
```

Vertikaalselt vektoriseerimine

B-puu indeksi lisamise lause:

```
CREATE INDEX "IX_stocks_ticker_date"
    ON stocks USING btree
    (ticker ASC NULLS LAST, first_date DESC NULLS LAST);
```

BRIN indeksi lisamise lause:

```
CREATE INDEX "IX_stocks_ticker_date"
    ON stocks USING brin
    (ticker, first_date);
```

Horisontaalselt vektoriseerimine

B-puu indeksi lisamise lause:

```
CREATE INDEX "IX_stocks_timeseries_date"
    ON stocks_timeseries USING btree
    (date DESC NULLS LAST);
```

BRIN indeksi lisamise lause:

```
CREATE INDEX "IX_stocks_timeseries_date"
    ON stocks_timeseries USING brin (date);
```

Lisa 7 – Parima indeksitüübi valimine Saaty meetodiga

Kriteeriumite ja valikute tähistused

Kriteeriumid	**Valikud**
A. Päringute täitmise kiirus B. Ridade lisamise kiirus C. Andmebaasi andmemaht	D. BRIN indeks E. B-puu indeks F. Ilma täiendavate indeksiteta

Kriteeriumite suhteline olulisus

Mõjur	**A**	**B**	**C**	**Kaalud**
A	1.0	4.0	1.0	**0.458**
B	0.25	1.0	0.33	**0.126**
C	1.0	3.0	1.0	**0.416**
CM: 0.057				

Päringute täitmise kiiruse kriteeriumi hinnangud

	D	**E**	**F**	**Kokku**
D	1.0	0.28	0.86	**0.174**
E	3.6	1.0	3.1	**0.625**
F	1.16	0.32	1.0	**0.201**
CM: 0.000				

Ridade lisamise kriteeriumi hinnangud

	D	**E**	**F**	**Kokku**
D	1.0	1.15	0.83	**0.326**
E	0.87	1.0	0.72	**0.283**
F	1.2	1.38	1.0	**0.391**
CM: 0.000				

Andmebaasi andmemahtude kriteeriumi hinnangud

	D	**E**	**F**	**Kokku**
D	1.0	1.02	1.0	**0.336**
E	0.98	1.0	0.98	**0.329**
F	1.0	1.02	1.0	**0.336**
CM: 0.000				