TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Elisabeth Abel 185932IADB

Tagarakenduse loomine visualiseerimiseks sobivate andmete saamiseks PostgreSQL'iga

Bakalaureusetöö

Juhendaja: Toomas Lepikult
PhD
Kaasjuhendaja: Martti Savolainen

Tallinn 2021

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: Elisabeth Abel

17.05.2021

Annotatsioon

Andmete visualiseerimist kasutatakse andmete analüüsimiseks ning andmetest hea ülevaate saamiseks. Visualiseerimisel võib tekkida aga situatsioon, kus visualiseeritud andmehulk on nii suur, et visualiseerides joonisel tekib liiga palju andmepunkte, mida ei ole võimalik üksteisest eristada. Andmete pärimine visualisatsiooniks on aeglane, juhul kui andmebaasi tabelis on palju andmeid.

Lõputöö raames loodi Avatud Lahendused OÜ-le visualiseerimissüsteemi tagarakendus, kus neid probleeme lahendati. Tagarakendusest on võimalik saada ükskõik missuguse struktuuriga tabelist kirjeid. On võimalik määrata, kui palju andmeid andmebaasitabelist soovitakse saada. Samuti määrata mis välju saada, mis kalkulatsioone üle väljade teha ning kuidas neid andmeid järjestada. Tagarakenduses on andmebaasist kirjete pärimine optimiseeritud materialiseeritud vaadetega, mis teevad andmete tagastamise kiiremaks.

Tagarakenduses andmebaasist kindla arvu kirjete valimiseks oli vaja kasutada meetodit, mida kasutades oleks võimalik täita tagarakendusele esitatud nõudeid. Samuti valida andmete talletamiseks meetod, selleks, et andmete tagastus oleks kiire. Kõige raskem oli mõelda valmis funktsionaalsuse loogika, mis kasutab meetodit andmebaasist kindla arvu kirjete valimiseks, andmete talletamise meetodit ning täidab kõiki määratud nõudeid.

Tagarakendus realiseeriti ning seda testiti lokaalselt. Avatud Lahendused OÜ peaarendaja, kes soovis, et ettevõte hakkaks arendama visualiseerimissüsteemi ning kellega arutades määrati nõuded, kinnitas, et tagarakendus täidab ette määratud funktsionaalseid ja mittefunktsionaalseid nõudeid. Loodud tagarakendusega on võimalik saada andmebaasitabelist visualiseerimiseks sobivaid andmeid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 6 peatükki, 12 joonist, 1 tabelit.

Abstract

Creating a backend fo getting data suitable for visualization with PostgreSQL

Data visualization is beneficial for analyzing and getting a good overview of specific data. Visualizing data can help in spotting different patterns, trends and correlations in datapoints, that can give a lot of information to the viewer. Information, which is hard to get from just looking at raw and unorganized data. However it can be hard to analyze graphs and charts that have a lot of datapoints on it, because the points become hard to distinguish from eachother. Therefore a lot of information is lost. In addition, querying large datasets from a database can be really slow.

In this thesis a backend for solving these problems was created. A company named Avatud Lahendused OÜ requested a backend for a visualization system. With created backend, it is possible to get data from database tables with any table structure. When requesting data, it is possible to specify how much data to get from database table. Also which columns to get, what calculations to perform across table rows and how to order queried data.

To get a specific amount of records from a database table, it was necessary to anayze two different methods for querying data from database. The method had to meet the requirements set for the backend. In addition to make querying data as fast as possible two different data storing options were analyzed and compared. As an outcome, a method for querying data and a method for storing that queried data were chosen. Both methods were used in backend logic.

As a result a backend for data visualization system was created. The backend was tested locally. Avatud Lahendused OÜ head developer, who requested the backend, confirmed, that the backend met all functional and not functional requirements set. Created backend made it possible to get data suitable for visualization. The thesis is in Estonian and contains 29 pages of text, 6 chapters, 12 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application Program Interface.* Rakendusliides, mis määrab, kuidas rakenduste omavahelil suhtlevad.
JSON	*Javascript Object Notation.* Andmevahetusformaat, mis jaotab andmed võti väärtus paaridesse
HTTP	*Hypertext Transfer Protocol.* Protokoll arvutivõrgus teabe esitamiseks.
UUID	*Universally Unique Identifier.* Universaalne unikaalne identifikaator.
SQL	*Structured Query Language.* Andmebaasi päringukeel.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval kogutakse erinevatest valdkondadest aina rohkem andmeid. Kogutud andmete analüüsimiseks kasutatakse andmete visualisatsiooni. Andmete visualiseerimine erinevate jooniste ja graafidega aitab nendest saada kiiremini vajalikku informatsiooni ning ülevaadet. Andmeid visualiseerides on võimalik leida nendest mustreid, korrelatsioone ja palju muud, mille abil saab teha andmete kohta järeldusi ning informeeritud otsuseid.

Lõputöö raames arendatakse tagarakendust ettevõttele Avatud Lahendused OÜ, kus töötab ka lõputöö autor. Ettevõte soovib luua andmete visualiseerimissüsteemi, mida oleks võimalik tulevastele ning praegustele klientidele pakkuda. Seda süsteemi ei disainita ühele kindlale kliendile. Andmebaasitabelite andmete maht võib erineda. Suurte andmehulkade korral võib tekkida olukord, kus andmeid on liiga palju ning nende visualiseerimisel ei ole võimalik punkte eristada. Lisaks sellele on suurte andmehulkade pärimine andmebaasist aeglane.

Lõputöö eesmärgiks on luua visualiseerimissüsteemile tagarakendus. Tagarakendus peab olema võimeline saama andmeid visualiseerimiseks ükskõik missuguse andmebaasitabeli struktuuriga PostgreSQL andmebaasisüsteemist. Peab olema võimalik tagastada soovitud arv andmebaasi tabeli kirjeid, soovitud väljadega, kalkulatsioonidega, mis on võetud üle andmebaasi tabeli väljade ning järjestusega. Tagarakenduses peab olema võimalik andmeid tagastada nii kiiresti kui võimalik.

Selleks, et oleks võimalik võtta andmebaasitabelitest ainult teatud arv kirjeid, analüüsitakse kõigepealt kahte meetodit andmete valimiseks andmebaasitabelist. Analüüsi tulemusena valitakse üks, mida kasutatakse tagarakenduses. Seejärel analüüsitakse kahte viisi, kuidas talletada päringust saadud andmeid, et tagarakendusest andmete tagastamine oleks optimeeritud. Valitakse üks andmete talletamise viis, mida kasutatakse tagarakenduses. Lõputöö neljandas osas kirjeldatakse tagarakenduse realiseerimist. Lõpus antakse ülevaade lõputöö tulemusest, testimisest, tagarakenduse kasutamisest ning edasiarendamise võimalustest.

2 Meetodite analüüs kindla arvu andmete valimiseks andmebaasitabelist

Antud peatükis analüüsitakse ja võrreldakse kahte meetodit kindla arvu kirjete valimiseks andmebaasitabelist ning valitakse nendest üks, mida kasutatakse tagarakenduse loomise loogikas.

2.1 Meetodite nõuded andmete valimiseks

Nõuded koguti arutades ettevõtte peaarendajaga, kes püstitas ülesande ettevõttele visualiseerimissüsteemi loomise jaoks.

2.1.1 Funktsionaalsed nõuded

Järgnevas loetelus on välja toodud funktsionaalsed nõuded andmebaasitabelist kirjete valimise meetodile.

- Meetodiga saadud uues väiksemas andmehulgas olevad andmed peavad kõik eksisteerima ka andmebaasitabelis.
- Meetodit peab saama kasutada ükskõik missuguse struktuuriga andmebaasitabelist kirjete saamiseks.
- Meetodit kasutades peab olema võimalik teha üle andmebaasitabeli väljade kalkulatsioone. Näiteks peab olema võimalik võtta summat, keskmist, miinimumi ja maksimumi.
- Meetodis peab olema võimalik määrata, mitu kirjet tagastatakse.
- Meetodit kasutades peab olema võimalik määrata, missuguseid andmebaasitabeli välju soovitakse saada.
- Meetodiga peab olema võimalik andmebaasitabeli välja järgi järjestada tagastatavaid kirjeid.

- Andmete valimine andmebaasitabelist peab käima kasutades PostgreSQL-i andmebaasisüsteemi.

2.1.2 Mittefunktsionaalsed nõuded

Järgnevas loetelus on välja toodud mittefunktsionaalsed nõuded andmebaasitabelist kirjete valimise meetodile.

- Meetodi juures tagastavate kirjete arvu määramine peab olema kerge, et seda oleks võimalik muuta vastavalt sisendile.

- Meetodiga tagastatud andmekogum peab esindama võimalikult hästi kõiki andmebaasitabeli kirjeid. Võrreldes tagastatud andmehulka kujutavat joondiagrammi ja tabeli kõiki kirjeid kujutavat joondiagrammi, peaksid joonte tõusud ja langused olema sarnased.

- Meetodi ei tohiks olla liiga keeruline. Meetodit peaks olema võimalik täiendada tulevikus ilma suure vaevata.

2.2 Kahe meetodi valimine analüüsiks

Meetodid kindla arvu andmete valimiseks andmebaasitabelist järgivad tavaliselt ühte kahest ideest. Esimene idee on valida mingi kindla algoritmiga välja kirjetest need kirjed, mida tagastada. Teiseks ideeks on teha kalkulatsioon üle teatud arv kirjete grupi ning saadud kalkulatsiooni tulemus esindab seda kirjete gruppi [1]. Näiteks võtta igast sajast andmebaasi kirjest kirje keskmine väärtus ning need keskmised väärtused tagastada.

Esimest ideed järgivad algortimid annavad tulemuseks andmehulga, kus kõik kirjed eksisteerivad ka andmebaasitabelis. Teist ideed järgivad algortimid annavad tulemuse, kus tagastavad andmed ei ole enam originaalsed. Kuna üheks funktsionaalseks nõudeks on tagastada andmehulk, kus kõik kirjed eksisteerivad ka andmebaasitabelis, siis lõputöös analüüsitakse kahte meetodit, mis järgivad esimest ideed.

Lõputöös analüüsitakse kahte meetodit andmete valimiseks: Suurim-Kolmnurk-Kolm-Gruppi algoritm (*Largest-Triangle-Three-Buckets*) ja PostgreSQL *row_function() window* funktsiooniga andmete valimine andmebaasist.

2.3 Analüüsi metootika

Selleks, et oleks võimalik analüüsida mõlemat meetodit, luuakse näidisandmebaasitabel. Analüüsiks tehakse SQL (*Structured query language)* päring näidisandmebaasi ning päringu tagastatud kirjed visualiseeritakse joondiagrammina Microsoft Excelis. Joondiagrammi abil on võimalik näha küsitud väljade järgi tehtud joone tõuse ja languseid, mis aitab meetodeid analüüsida. Kõigepealt analüüsitakse lühidalt näidisandmebaasist kirjete saamist ning kirjete visualisatsiooni ilma meetodeid kasutamata, selleks, et oleks võimalik meetodite tulemust kõikide andmebaasitabeli kirjetega võrrelda. Seejärel analüüsitakse mõlema meetodi päringulauseid, visualisatsioone ning nõuete täitmist. Siis võrreldakse mõlemat meetodit ning valitakse üks, mida kasutatakse tagarakenduse loomisel.

2.3.1 Näidisandmebaas

Näidisandmebaasis on väljad temperatuur ja kuupäev. Kuupäev esindab seda hetke, millal temperatuur mõõdeti. Näidisandmebaasi lisati 546 kirjet. Kirjed esindasid iga päev samal ajal mõõdetud temperatuuri kuupäevadel 01.06.2020 - 30.11.2021. Andmebaasi sisestatud temperatuurid on suvalised.

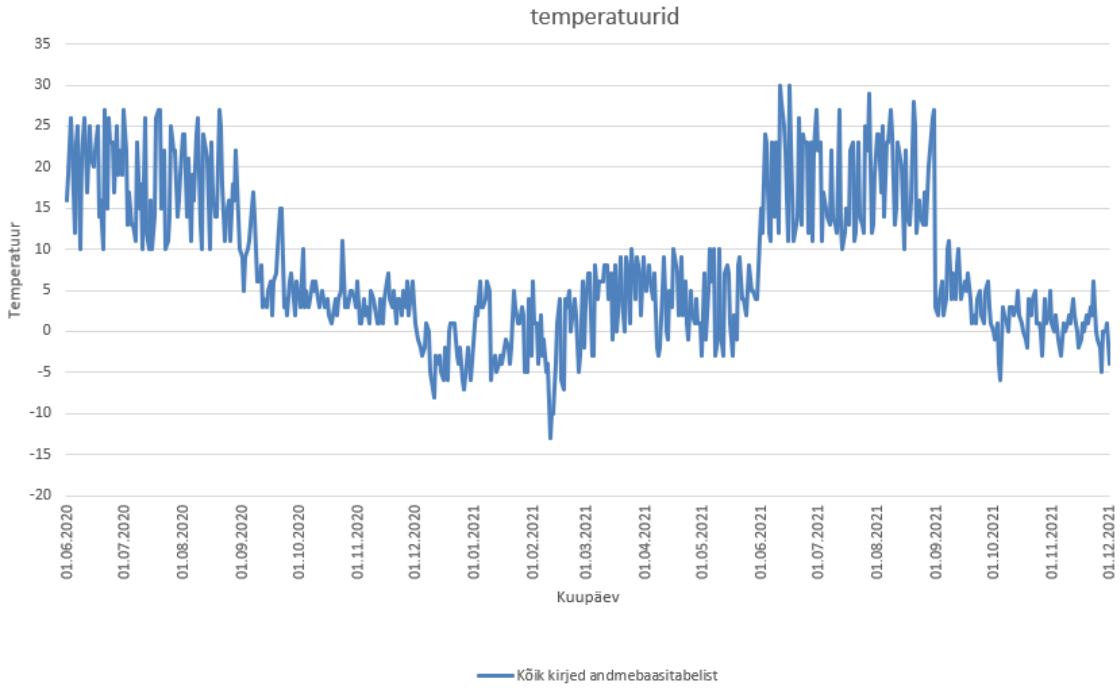
2.4 Näidisandmebaasist ilma kindla arvu kirjete saamise meetodita saadud kirjete analüüs

Näidisandmebaasist kõikide kirjete saamiseks tehakse *select* päring (Joonis 1).

```
SELECT
   "temperatuur",
   "kuupaev"
FROM
   "temperatuurid";
```

Joonis 1. Päringulause kõikide kirjete saamiseks andmebaasitabelist.

Päring tagastab 546 kirjet. Tagastatud kirjete joondiagrammist on näha, et suurt andmehulka on raske analüüsida, sest diagrammil ei ole andmepunkte üksteisest võimalik eristada (Joonis 2).



Joonis 2. Joondiagramm, millel on kujutatud kõiki andmebaasitabeli kirjeid.

2.5 Suurim-Kolmnurk-Kolm-Gruppi algoritm

Esimeseks algoritmiks on Suurim-Kolmnurk-Kolm-Gruppi algoritm. See on võetud Sveinn Steinarssoni magistritööst *„Downsampling time series for visual representation“* [2]. Selles töös võrreldi erinevaid andmebaasitabelist andmete valimise algoritme andmete visualiseerimiseks. Suurim-Kolmnurk-Kolm-Gruppi algoritm töötas Sveinn Steinarssoni töös kõige efektiivsemalt.

2.5.1 Suurim-Kolmnurk-Kolm-Gruppi algoritmi kirjeldus

Suurim-Kolmnurk-Kolm-Gruppi algoritm käib järgnevalt. Andmebaasitabeli kirjed jagatakse ligikaudu sama suurtesse gruppidesse. Esimene ja viimane tabeli kirje on mõlemad omaette grupis ning kõik ülejäänud kirjed jagatakse võrdselt gruppidesse mingi kindla arvu järgi. Igast grupist valitakse üks kirje, mis läheb tagastavasse andmehulka [2]. Vaadates kõiki kirjeid kujutavat joondiagrammi (Joonis 2), hakkab algoritm liikuma x teljel vasakult paremale

Kuna esimeses grupis on ainult üks kirje, siis kõigepealt valitakse see. Seejärel valitakse järgnevatest gruppidest kirjed selliselt:

1. Olles praeguses grupis, võetakse eelmises grupis juba valitud kirje väärtus.
2. Arvutatakse järgmises grupis olevate kirjete väärtuste keskmine väärtus.
3. Praegusest grupist valitakse selline kirje, mille väärtus looks joondiagrammil koos eelmisest grupist valitud kirje väärtuse ja järgmise grupi kirjete väärtuste keskmisega kõige suurema pindalaga kolmnurga.
4. Algoritm läheb järgmise grupi juurde ja teeb neid samme uuesti.

Kirje väärtuseks on see väärtus, mida joondiagrammil tahetakse visualiseerida või mille järgi tahetakse valida kirjeid. Näiteks Joonisel 2 oleks selleks väärtuseks temperatuur. Kui jõutakse eelviimase grupi juurde, siis järgmise grupi keskmist kirje väärtust ei tule arvutada, sest seal on ainult üks kirje. Põhjus, miks järgmisest grupist ei valita ette eksisteerivat kirjet on see, et muidu oleks vaja arvutada kolmnurga pindala läbi iga praeguse grupi kirje ja iga järgmise grupi potensiaalse kirjega ning see läheks liiga keeruliseks [2].

Algoritmi illustreerimiseks lisati joonis (Joonis 3). Joonisel kõik sinised täpid on esialgsed kirjed. Hall ala tähistab gruppi. Sinise ringiga punktid on need kirjed, mis valiti tagastatavasse andmehulka. Punased täpid on gruppidest arvutatud kirjete väärtuste keskmised väärtused [1].

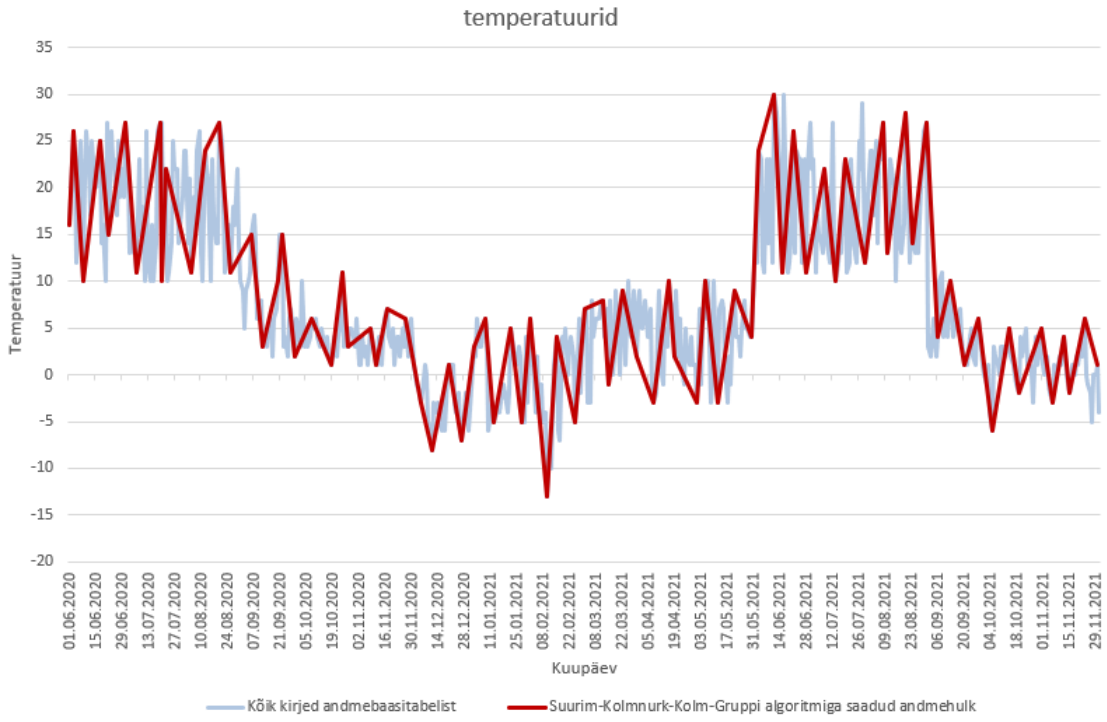
Joonis 3. Suurim-Kolmnurk-Kolm-Gruppi algoritmi illustratsioon [1].

2.5.2 Suurim-Kolmnurk-Kolm-Gruppi algoritmi kasutamine

SQL implementatsioon algoritmist leiti internetist [1]. Selleks, et määrata, kui suured tulevad grupid, tuleb määrata selles SQL koodis *bucketSize*. *BucketSize'ks* määrati 86400, mis tähendab, et üle iga seitsme päeva valitakse üks kirje. See tagastab 81 kirjet.

2.5.3 Suurim-Kolmnurk-Kolm-Gruppi algoritmi tulemuse joondiagrammi analüüs

Kui võrrelda kõiki kirjeid kujutavat joondiagrammi tagastatud andmehulka kujutava joondiagrammiga, siis on näha, et need on sarnased (Joonis 4). Väiksema andmehulga joondiagrammi joone tõusud ja langused on peaaegu samad kõikide kirjete omaga. Eemaldatud on informatsioon, mis segas selle joone punktide eristamist. Sellist joondiagrammi vaadates on võimalik teha järeldusi andmebaasitabeli kõikide andmete kohta. Samuti on näha, et esimene ja viimane kirje on samad kõiki kirjeid kujutava joondiagrammiga. See on tänu sellele, et esimene ja viimane kirje pandi algoritmis omaette gruppidesse, mis kindlustas, et need oleksid olemas ka tagastatavas andmehulgas. Esimesest ja viimasest kirjest võib saada palju kasulikku informatsiooni, sest neid on võimalik kasutajal võrrelda ja teha nendest järeldusi.



Joonis 4. Joondiagramm, millel on kujutatud Suurim-Kolmnurk-Kolm-Gruppi algoritmiga saadud andmed (punane joon) ja kõiki kirjeid andmebaasitabelist (sinine joon).

2.5.4 Suurim-Kolmnurk-Kolm-Gruppi algortimi analüüs

Algoritmi tugevus on, et tagastatud andmehulga joondiagramm on sarnane kõiki andmebaasitabeli kirjeid kujutava andmehulga joondiagrammiga. Algoritmi nõrkuseks on, et kõik ülejäänud kirjed peale esimese ja viimase jaotatakse ühtlaselt gruppide vahel ära. Juhul, kui tekib situatsioon, kus ühes grupis on kirjed, mille väärtuste vahe on suur, siis seda ei ole võimalik ühe punktiga joondiagrammil kirjeldada nii, et see esindaks piisavalt täpselt seda gruppi [2]. Näiteks, kui ühes grupis on kogu andmebaasitabeli kõige minimaalsem ja maksimaalsem kirje väärtus koos, siis seda ühe kirjega esindades kaotab joondiagramm palju kasulikku informatsiooni. Lisaks sellele on algoritm sõltuv kindlast andmebaasitabeli struktuurist. See on mõeldud kasutamiseks andmetabelite peal, kus väljadeks on kuupäev ja sellele kuupäevale vastav väärtus. Tagarakenduses peab aga võimalik olema võtta andmeid ükskõik missuguse struktuuriga andmebaasitabelist. Samuti on algoritm keeruline ning ei ole piisavalt paindlik, et lisada juurde võimalus saada ükskõik mitu tabeli välja ning väljade peal teha kalkulatsioone.

2.6 PostgreSQL *window* funktsiooniga andmete valimine andmebaasitabelist

Teiseks meetodiks on PostgreSQL *row_number()* funktsiooni kasutamine. Funktsioon *row_number()* on PostgreSQL *window* funktsioon. PostgreSQL *window* funktsioonid teostavad erinevaid kalkulatsioone üle tabeli kirjete, mis on seotud praeguse kirjega. *Window* funktsioonide kalkulatsioonide tulemusena jätavad kõik kirjed omaette identiteedi ning tagastuses ei grupeerita kirjeid kokku [3]. *Row_number()* funktsioon määrab järjestikuseid numbreid igale andmereale [4]. Selle numbri abil on võimalik andmeid andmebaasist valida võttes ühe kirje üle iga kindla arvu kirjete.

Window funktsioonid on sarnased agregaatfunktsioonidega. Agregaatfunktsioonid teevad kalkulatsioone üle kirjete, kuid tagastavad ainult ühe väärtuse [5]. Agregaatfunktsioonideks on näiteks keskmise arvutamine, mis tagastab tulemusena ühe arvu. Keskmise arvutamine *window* funktsiooniga on sarnane tavalise keskmise arvutamisega, kuid selle puhul lisatakse *over()* tingimus. *Over()* määrab, üle mille tehakse kalkulatsioon, ehk siis üle missuguse välja võetakse keskmine. See keskmine määratakse igale tagastatud kirjele [3].

2.6.1 PostgreSQL *window* funktsiooni kasutamise näide

Window funktsiooni kasutamiseks tehakse näidisandmebaasi tabel, kus on väljad õpilase hinde ja soo kohta. Päringuga tahetakse saada õpilaste sugu, hinne ning keskmine hinne vastavalt soole (Joonis 5). Päringus on *over()* tingimuses *partition by()*, mis määrab mille järgi keskmisi hindeid grupeerida [3].

```
SELECT
   "sugu",
   "hinne",
   AVG("hinne") over (partition by "sugu"),
   row_number() over()
FROM
   "hinded";
```

Joonis 5. Näide päringulausest PostgreSQL *window* funktsioonide kasutamise kohta.

Tagastatud kirjetest on näha, et kõikidel naissoost õpilastel on arvutatud keskmine üle naissoost õpilaste hinnete ning meessoost õpilastel on arvutatud keskmine üle meessoost

õpilaste hinnete (Joonis 6). Samuti on iga kirje juures rea number. Päring tagastab kõik read.

sugu	hinne	avg	row_number
Female	5	4.3333333333333333	1
Female	4	4.3333333333333333	2
Female	4	4.3333333333333333	3
Male	3	3.3333333333333333	4
Male	4	3.3333333333333333	5
Male	3	3.3333333333333333	6

Joonis 6. PostgreSQL *window* funktsioone kasutava päringu tagastatud kirjed.

2.6.2 *Window* funktsiooni kasutamine andmete valimiseks andmebaasitabelist

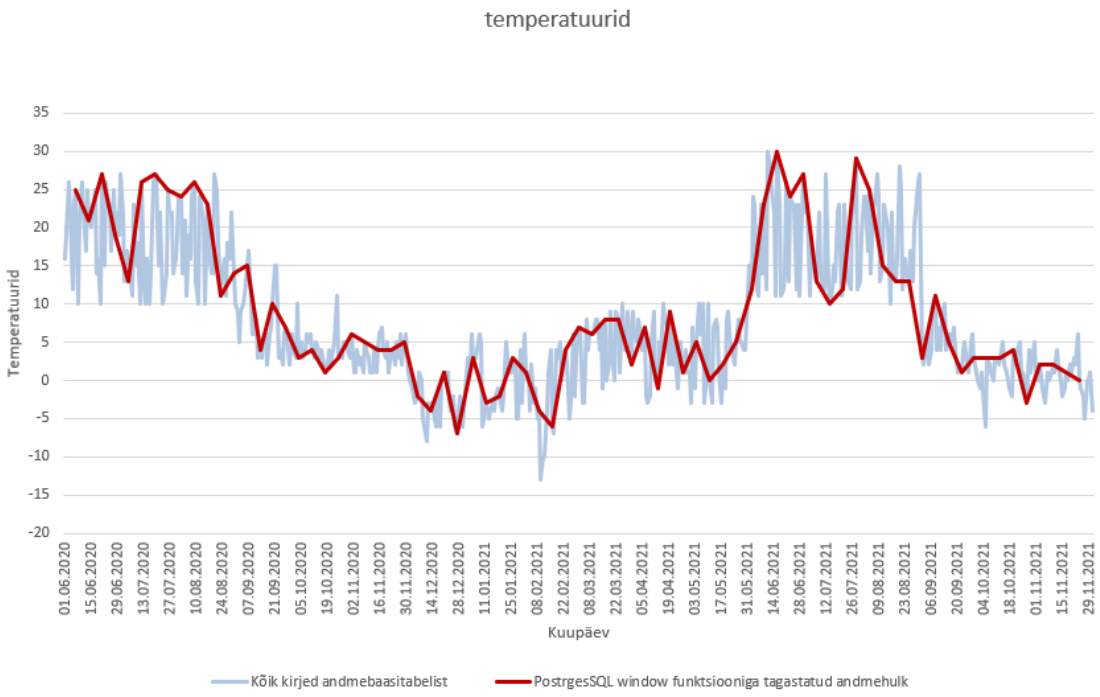
Andmete valimiseks luuakse päringulause, milles määratakse, et iga 7 kirje tagant valitakse 1 kirje (Joonis 7). Lausega tagastati andmebaasist 78 kirjet.

```
SELECT
   x."kuupaev",
   x."temperatuur",
   x.bucket,
   row_number
FROM
   (
      SELECT
         "kuupaev",
         "temperatuur",
         ntile( ( FLOOR( (
         SELECT
            COUNT(*)
         FROM
            "temperatuurid") * 20 / 100 ) )::INTEGER )
over () as bucket,
            row_number() over ()
         FROM
            "temperatuurid"
   )
   x
WHERE
   mod(row_number, 7) = 0;
```

Joonis 7. Päringulause, mis kasutab PostgreSQL *window* funktsioone andmebaasitabelist kirjete valimiseks

2.6.3 *Window* funktsiooniga valitud andmete joondiagrammi analüüs

Joondiagrammilt on näha, et meetod ei järgi kõiki kirjeid kujutava joondiagrammi joone maksimaalseid ja minimaalseid punkte (Joonis 8). Joonisel on sinine joon kõiki kirjeid kujutav joon ja punane on tagastatud kirjeid kujutav joon. Sisse on tekkinud ka valed langused ja tõusud kohtades, kus kõiki tabeli kirjeid kujutaval joondiagrammil neid ei ole. Selline vale joone liikumine võib kasutajat eksitada andmete analüüsimisel. Põhjuseks on, et meetod võtab iga seitsme kirje tagant ühe kirje ilma midagi arvutamata. Selle seitsme kirje seas võis olla nii maksimaalseid kui ka minimaalseid kirje väärtusi. Samuti ei ole kindlustatud, et esimene ja viimane kirje on sama andmebaasitabeli esimese ja viimase kirjega. Ka sealt tekib palju informatsiooni kadu.



Joonis 8. Joondiagramm, millel on kujutatud PostgreSQL *row_number()* funktsiooniga valitud andmed (punane joon) ja kõiki kirjeid andmebaasist (sinine joon).

2.6.4 *Window* funktsioonidega andmebaasitabelist andmete valimise analüüs

Meetodi tugevuseks on, et päringulauses on kerge määrata, kui palju tagastatakse kirjeid. Nii ei ole tahetud kirjete arvu lisamine päringusse keeruline. Lisaks sellele ei sõltu meetod kindla struktuuriga andmebaasitabelist. Samuti on võimalik saada lisa informatsiooni, kasutades *window* funktsioone kalkulatsioonide tegemiseks üle väljade. Meetod on

piisavalt paindlik, sest päringulausesse on võimalik lisada ükskõik kui palju välju ning väljade lisamine päringusse vastavalt kasutaja soovile ei ole keeruline.

Meetodi nõrkuseks on, et tagastatud andmekogumi joondiagrammi maksimaalsed ja minimaalsed punktid ei ole piisavalt sarnased kõiki andmebaasitabeli kirjeid kujutava joondiagrammi maksimaalsete ja minimaalsete punktidega. Lisaks ei ole joondiagrammil esimene ja viimane kirje samad kõikide kirjete esimese ja viimase kirjega. See on nõrkuseks ainult juhul, kui andmed on seotud ajaga.

2.7 Meetodite võrdlus

Suurim-Kolmnurk-Kolm-Gruppi algoritm on silmapaistvamalt parem kõiki andmebaasitabeli kirjeid kujutava joondiagrammi joone järgimisel. Samas ei täitnud see palju nõudeid (Tabel 1). *Window* funktsioonil põhinev meetod ei esinda piisavalt hästi kõiki andmebaasitabeli kirjeid. See meetod ei ole disainitud selleks, et tagastatavas andmehulgas oleks garanteeritud, et andmebaasitabeli maksimaalsed ja minimaalsed kirje väärtused on olemas. PostgreSQL'i *row_number()* funktsiooniga andmete valimine täitis aga kõiki teisi nõudeid (Tabel 1).

Tabel 1. PostgreSQL *row_number() window* funktsiooniga andmete valimise nõuete täitmise ja Suurim-Kolmnurk-Kolm-Gruppi algoritmiga andmete valimise nõuete täitmise võrdlus.

Nõue	**PostgreSQL *row_number() window* funktsioon**	**Suurim-Kolmnurk-Kolm-Gruppi algoritm**
Uues andmehulgas olevad andmed eksisteerivad ka andmebaasitabelis.	JAH	JAH
Saab kasutada ükskõik missuguse struktuuriga andmebaasitabelist kirjete saamiseks.	JAH	EI
On võimalik teha üle andmebaasitabeli väljade kalkulatsioone.	JAH	EI
On võimalik määrata, mitu kirjet tagastatakse.	JAH	JAH
On võimalik määrata, missuguseid andmebaasitabeli välju soovitakse saada.	JAH	EI

Kasutab andmete valimiseks PostgreSQL andmebaasisüsteemi.	JAH	JAH
Tagastavate kirjete arvu määramine on kerge.	JAH	EI
Tagastatud andmekogum esindab võimalikult hästi kõiki andmebaasitabeli kirjeid.	EI	JAH
Meetod ei ole liiga keeruline.	JAH	EI

2.8 Ühe meetodi valimine

Analüüsi tulemusena valiti *window* funktsioonidega andmebaasitabelist andmete valimise meetod, sest see täidab enamus nõudeid.

3 Päringu tulemuse talletamise meetodite analüüs

Lõputöö üheks eesmärgiks on optimeerida andmebaasitabelist andmete tagastamist nii, et andmete tagastamine toimuks võimalikult kiiresti. Andmete saamiseks tuleb teha iga kord läbi kõik andmete filtreerimised, kalkulatsioonid ja järjestamised ning see on aeglane. Selle probleemi lahendamiseks on vaja kindlate sisendite puhul päringute poolt tagastatud andmehulki hoida kuskil, kust oleks võimalik neid iga kord kiirelt kätte saada.

Selles peatükis analüüsitakse kahte võimalust, kuidas hoida päringute tulemusi. Tuuakse välja nõuded tulemuste talletamisele ning tuuakse välja, mis nõudeid vastav lahendus täidab. Seejärel valitakse analüüsi tulemusena välja ühe andmehulga talletamise viis.

3.1 Päringu tulemuse talletamise nõuded

Järgnevas loetelus on toodud välja nõuded päringu tulemuse talletamise viisile.

- Päringu tulemust on kerge ja arusaadav talletada.
- Talletatud päringu tulemust on võimalik uuendada.
- Talletatud päringu tulemus on alati kättesaadav.
- Peab olema võimalus andmeid edasi töödelda ja filtreerida.
- Andmed on tagastamiseks õigel kujul.

3.2 Päringu tulemuste talletamine PostgreSQL materialiseeritud vaadetega

Selles alapeatükis kirjeldatakse ning analüüsitakse päringu tulemuse talletamist PostgreSQL materialiseeritud vaadetega.

3.2.1 PostgreSQL materialiseeritud vaate kirjeldus

Materialiseeritud vaated on sarnased andmebaasi tabelitega. Materialiseeritud vaatesse on võimalik salvestada päringu tulemus ja päringulause ning sellele panna nimi. Seejärel saab materialiseeritud vaate nimega pärida andmeid samamoodi nagu tavalisest andmebaasitabelist. Materialiseeritud vaatesse ei ole võimalik sisestada andmeid. Selleks,

et uuendada materialiseeritud vaadet, tuleb teha SQL päring *refresh materialized view* koos materialiseeritud vaate nimega. See päring käivitab uuesti vaate loomisel kasutatud SQL päringut ning talletab värsked andmed samasse materialiseeritud vaatesse [6]. Andmete värskendamise ajal ei ole võimalik materialiseeritud vaate poole pöörduda. PostgreSQL 9.4 versioon aga võimaldab uuendamise ajal materialiseeritud vaate poole pöörduda, kui uuendamise päringule lisada enne nime ka tingimus *concurrently* [7].

3.2.2 Päringu tulemuse talletamine materialiseeritud vaadetega analüüs

Materialiseeritud vaadete tugevuseks on, et materialiseeritud vaadet hoitakse andmebaasis, kus sellele on kerge ligi pääseda. Materialiseeritud vaateid on kerge uuendada, vaja on ainult materialiseeritud vaate nime. Selle eeliseks on veel, et teades materialiseeritud vaate nime, ei ole vaja uuendatud tulemuse saamiseks genereerida uut SQL päringut, vaid see on salvestatud materialiseeritud vaatesse. See on kasulik juhul, kui SQL päringute genereerimine on keeruline. Tugevuseks on veel see, et materialiseeritud vaatest on võimalik pärida andmeid, neid filtreerida, järjestada ning teha nende üle kalkulatsioone. Kui muutub andmebaasi tabeli struktuur, mille kohta on tehtud materialiseeritud vaade, siis kustutatakse see materialiseeritud vaade automaatselt [8].

Materialiseeritud vaadete üheks nõrkuseks on, et selles vaates olevad andmed ei pruugi olla värsked. Juhul, kui andmebaasi tabelis olevaid kirjeid kustutatakse, muudetakse või lisatakse juurde, kuid materialiseeritud vaadet ei uuendata, siis materialiseeritud vaates on veel vanad andmed. Lisaks sellele on probleemiks see, et uuendamise ajal ei ole võimalik materialiseeritud vaate poole pöörduda, juhul kui ei uuendata tingimusega *concurrently* [7]. See võib tekitada probleeme, kui soovitakse saada tagarakendusega andmeid mille puhul on vaja materialiseeritud vaate poole pöörduda.

3.3 Päringu tulemuse talletamine jsonb andmetüübina

Andmebaasitabelist kirjete saamise päringu tulemus on list, milles on kirjed JSON’i (*Javascript Object Notation*) kujul. Ideeks on salvestada need kirjed JSON’i kujul andmetüübina PostgreSQL andmebaasitabelisse. PostgreSQL toetab kahte erinevat JSON andmetüüpi: json ja jsonb. Json ja jsonb andmetüübid kindlustavad, et seal hoitud väärtus oleks JSON’i reegleid järgiv [9].

Json ja jsonb andmetüübid on peaaegu identsed, kuid erinevus on nende efektiivsuses. Json andmetüüp salvestab täpse koopia sisestatud JSON'i tekstist, mille töötlemisel tuleb funktsioonidel JSONi teksti iga kord parsida. Jsonb andmetüüp salvestab sisendi binaarkujul (*binary format*). Jsonb kujul JSON'i teksti on aeglasem sisestada, sest seda tuleb teisendada binaarkujule, kuid jsonb andmetüüpi on kiirem töödelda, sest parsimist ei ole vaja teha [9]. Selle tõttu antud analüüsis analüüsitakse jsonb andmetüüpi, sest seda on kiirem töödelda. Jsonb andmetüübil on erinevad funksioonid ning operatoorid, mida on võimalik kasutada SQL päringu tegemisel [10].

Ideeks on luua iga päringu poolt tagastatud andmehulga jaoks eraldi andmebaasitabel. Tabeli väljas hoitakse andmehulga kirjet jsonb andmetüübina. Tabeli nimi sõltub sisendist. Sisendi korral kontrollitakse, kas sisendi järgi genereeritud nimega andmebaasitabelit eksisteerib. Kui tabel eksisteerib, päritakse sellest tabelist kõik väljad ning need tagastatakse. Kui tabelit, kus päringu tulemust hoitakse jsonb andmetüübina, ei eksisteeri, tehakse uus päring andmebaasitabelisse. Loodakse genereeritud nimega andmebaasitabel, kuhu lisatakse päringu poolt tagastatud kirjed ning seejärel need tagastatakse. Järgmine kord sama sisendi puhul on päringu tulemusega andmebaasitabel juba olemas ning siis on võimalik sealt otse kõik andmed kätte saada juba õigel kujul.

3.3.1 Päringu tulemuse talletamine jsonb andmetüübina analüüs

Andmebaasitabelis jsonb andmetüübina päringu poolt tagastatud kirjete hoidmise tugevuseks on, et tulemus on kättesaadav andmebaasist. Positiivseks küljeks on veel, et kirjeid hoitakse JSON'i kujul, mis tähendab, et neid ei pea enam eraldi konverteerima andmehulka tagastades. Jsonb andmetüübil on palju funktsionaalsusi ning seal hoitud JSON'it on võimalik filtreerida [11].

Tabeli uuendamiseks ei ole eraldi käsku vaid tabeli uuendamiseks tuleb tabelist kustutada eelnevad kirjed, genereerida päring uuesti ning päringu tulemused uuesti tabelisse lisada. Samuti kui see andmetabel, kust saadi tulemuse kirjed, kustutatakse, siis tuleb tulemuste tabel kustutada käsitsi.

3.4 Päringu tulemuste talletamise võimaluste võrdlemine

Mõlemad viisid täidavad enamus nõudeid. Nii jsonb andmetüübina tabelis kui ka materialiseeritud vaatena tulemuse talletamine on kättesaadavad andmebaasist. Jsonb

puhul ei ole vaja tulemust enam eraldi JSON’i kujule konverteerida, kuid materialiseeritud vaatest tulemuse saamisel tuleb seda teha. Jsonb andmetüübina tulemuse talletamine tabelis ning selle uuendamine on keerulisem, kui materialiseeritud vaadete loomine ja uuendamine. Materialiseeritud vaadetega ei ole vaja uuendamisel päringulauset genereerida, kuid jsonb kujul tulemuse talletamisel tabelis seda tuleb teha. Materialiseeritud vaadete puhul on võimalik kustutada materialiseeritud vaade automaatselt, kui tabeli, mille kohta vaade käib struktuuri muudetakse või andmetabel kustutatakse [8]. Sellisel juhul tabeli kustutamist tuleks teha jsonb andmetüübina kirjete hoidmisel käsitsi. Lisaks sellele on materialiseeritud vaadete filtreerimine selgem, sest see käib samamoodi nagu tavalisest andmebaasitabelist andmete filtreerimine. Jsonb puhul käiks aga andmete filtreerimine jsonb operaatoritega [11]. Selle tõttu valiti andmete talletamiseks materialiseeritud vaated.

4 Tagarakenduse realiseerimine

Tagarakendusel peab olema kaks põhifunktsionaalsust:

- Funktsionaalsus etteantud andmebaasitabeli nime järgi tabeli struktuuri saamine.
- Funktsionaalsus sisendi järgi päringulause genereerimine, päringu tulemuse talletamine materialiseeritud vaatesse ning seejärel saadud kirjete tagastamine õigel kujul.

4.1 Nõuded tagarakendusele

Nõuded tagarakendusele koguti ettevõtte peaarendajaga arutades.

4.1.1 Funktsionaalsed nõuded

Järgnevas loetelus on välja toodud tagarakenduse funktsionaalsed nõuded.

- Peab olema võimeline tagastama andmebaasitabeli struktuuri tabeli nime andmisel. Tagastatud struktuuris peab olema väljade nimed, väljade andmetüübid, võimalikud funktsioonid mida saab võtta väljast ja välja indeksite list indeksite olemasolu korral. Lisaks sellele protsentide list ja protsendile vastav kirjete arv.
- Peab looma materialiseeritud vaate, kui sisendis protsendi väärtus on kas 100, 25, 10, 5, 1, 0.1, 0.01 või 0.001.
- Peab võimeline olema tagastama andmebaasitabelist sisendis küsitud väljad ja funktsioonide tulemused üle väljade.
- Peab võimeline olema tagastama andmebaasitabelist sisendis määratud järjekorraga kirjed.
- Peab võimeline olema tagastama andmebaasitabelist ainult sisendis määratud protsent kirjeid.
- Peab valideerima, et sisendis oleksid kõik kohustulikud väljad olemas.

- Peab valideerima, et sisendis olev tabel eksisteeriks, tabeli väljad eksisteeriksid ning et väljade funktsioonid oleksid sobivad välja andmetüübiga.

- Kui sisendi valideerimine ei õnnestu, siis peab tagastama vea.

4.1.2 Mittefunktsionaalsed nõuded

Järgnevas loetelus on toodud välja tagarakenduse mittefunktsionaalsed nõuded.

- Andmete saamine andmebaasist peab olema kiire ning tegema päringuid andmebaasi nii vähe kui võimalik.

- Kui valideerimine ei õnnestu, siis tagastatavas veas peab olema selgesti mõistetav veateade.

- Tagarakenduse kood peab taaskasutama koodi kõikjal, kus on võimalik.

4.2 Kasutatavad tehnoloogiad

Tagarakendus on kirjutatud Pythonis. Python on interpreteeritav, objekt-orienteeritud ning üks populaarsemaid programmeerimiskeeli [12]. Python on disainitud olema hästi loetav, mis tähendab, et Pythonis kirjutatud koodist on kerge aru saada, koodi hallata ja kirjutada. See on dünaamiliselt tüübitud ning toetab mitmeid programmeerimisstiile. Lisaks on Pythonit võimalik laiendada erinevate moodulitega, tänu millele on võimalik koodi taaskasutada ning lisada koodile juurde funktsionaalsusi. Python toetab enamus levinud andmebaasisüsteeme, sealhulgas ka PostgreSQL-i [13].

Tagarakendus kasutab Sanic raamistikku. See on loodud spetsiaalselt kiirete HTTP (*Hypertext Transfer Protocol*) päringute tegemiseks tänu asünkroonsele päringute käsitlusele. Sanic töötab Pythoni 3.7+ versiooniga ning sellega saab kasutada Pyhton 3.5+ versioonis lisatud *await/async* süntaksit [14].

Tagarakendus kasutab PostgreSQL-i andmebaasisüsteemi. PostgreSQL on 1986ndal aastal loodud avatud lähtekoodiga relatsioonilise andmebaasi süsteem. PostgreSQL kasutab ja laiendab SQL keelt. Seda on võimalik kasutada kõikides levinumates operatisoonisüsteemides nagu näiteks Windowsis, macOS'is ja Linuxis [15].

4.3 Tabeli struktuuri saamiseks funktsionaalsuse loomine

Esimese funktsionaalsuse eesmärgiks on tagastada tabeli struktuur, kui sisendiks antakse tabeli nimi. Tulevikus kasutatakse seda eesrakenduses selleks, et kasutajale näidata valitud tabeli kogu informatsiooni, mille järgi kasutaja saab valida, mis andmeid ning mis suuruses ta sellest tabelist saada soovib.

Võimalikke funktsioone iga veeru jaoks saadi võrreldes funktsiooni argumendi andmetüüpe välja andmetüübiga. Võimalikke protsente saadi arvutades kogu andmebaasitabeli kirjete arvust vastav protsent ning juhul, kui selle tulemus oli võrdne või suurem ühega, siis see protsent lisati võimalike protsentide listi.Väljade nimede, indeksite ja andmetüübi saamiseks tehakse päring PostgreSQL *pg_class*, *pg_index* ja *pg_attribute* süsteemi kataloogidesse, päring leiti internetist ning tehti ümber nii, et oleks tagarakenduse nõuetele vastav [16]. PostgreSQL süsteemi kataloogides hoitakse informatsiooni andmebaasi metaandmete kohta, nagu näiteks informatsiooni andmebaasi tabelitest ja tabelite väljadest [17].

4.4 Soovitud andmete valimiseks andmebaasitabelist funktsionaalsuse loomine

Selles alapeatükis kirjeldatakse andmebaasitabelist kindel arv kirjete saamise funktsionaalsuse loogikat.

4.4.1 Andmete valimine andmebaasitabelist

Andmete valimist andmebaasist tehakse töös *select* päringulausega, kus kasutatakse funktsiooni *row_number()*, mis määrab igale kirjele rea numbri. Selle numbri järgi on võimalik võtta kirjeid üle teatud arv kirjete. Päringulauses on mitu *select* lauset üksteise sees. Nendes toimub *row_number()* funktsiooniga andmete valimine, andmete filtreerimine, PostgreSQL *window* funktsioonidega kalkulatsioonide tegemine ja kirjete järjestamine.

4.4.2 Funktsiooni optimeerimine materialiseeritud vaadetega

Funktsionaalsuse optimeerimiseks salvestatakse tagastatud kirjed materialiseeritud vaadetesse. Kirjeid salvestatakse materialiseeritud vaatesse ainult protsentide 100, 25, 10, 5, 1, 0.1, 0.01, 0.001 korral. Need protsendid valiti selle järgi, et igast protsendi

suurusastmest oleks vähemalt üks esindatud. Põhjus, miks kõikidest võimalikest protsentidest ei saa teha materialiseeritud vaadet on selles, et sellisel juhul tekiks materialiseeritud vaateid liiga palju, mida tuleb pidevalt uuendada. Kuigi seda on võimalik tulevikus muuta.

Materialiseeritud vaateid tuleb kuidagi eristada. Iga materialiseeritud vaate nimi tuleks genereerida sama loogika järgi. Juhul kui vaade on juba teatud sisendi kohta loodud, siis materialiseeritud vaadet on võimalik sisendi järgi leida. Materialiseeritud vaate nimi võib aga olla maksimaalset 63 bitti pikk [18].

Kuna sisendi järgi genereeritud vaate nimi ei oleks garanteeritud pikkusega, siis vaadete nimesid hoitakse eraldi tabelis. Materialiseeritud nimede tabelis on andmebaasitabeli nimi, kust päriti andmeid, sisendi järgi genereeritud võti ning sellele võtmele vastav materialiseeritud vaate nimi. Materialiseeritud vaate võti genereeritakse järgnevalt: liidetakse "_"-ga kokku protsent, andmebaasitabeli väljade nimed tähestikulises järjekorras, funktsioonide nimed tähestikulises järjekorras, iga funktsiooni juures välja nimi, mille üle funktsioon võetakse ja seejärel järjestuse korral ka välja nimi mille järgi soovitakse tulemust järjestada ning järjestuse suund.

Võtmele vastav suvaline materialiseeritud vaate nimi liidetakse kokku algusega „*materialized_view_*" ning seejärel lisandub UUID (*Universally Unique Identifier*). UUID on 128 bitist koosnev number või id, mis on unikaalne [19]. Seda on vaja materialiseeritud vaate nimesse selleks, et materialiseeritud vaate nimed ei korduks. UUID genereeritakse pythoni uuid 4 versiooni järgi. Uuid 4 versiooni genereerimisel kasutatakse *random* generaatorit, mille korral on samasuguse uuid genereerimise võimalus väga väike [19].

4.4.3 Andmete tagastamine

Juhul kui sisendis olev protsent ei kuulu nende protsentide hulka, mille korral luuakse materialiseeritud vaade, tehakse varem genereeritud päringulausega päring ning tagastatakse selle tulemus.

Juhul, kui sisendis on protsent mille korral tuleb luua materialiseeritud vaade, siis genereeritakse materialiseeritud vaate võti, selle võtme abil saadakse materialiseeritud vaadete nimede tabelist vaate nimi. Materialiseeritud vaate nime saamisel luuakse

andmebaasi uus materialiseeritud vaade, juhul kui see ei ole juba olemas. Materialiseeritud vaate loomiseks kasutatakse varem genereeritud select lauset. Seejärel küsitakse sellest materialiseeritud vaatest kõik tulemused, tulemus konverteeritakse jsoni kujule ning tagastatakse.

5 Tulemused

Lõputöö tulemusena realiseeriti tagarakendus. Avatud Lahendused OÜ pearendajalt, kes esitas ülesande arendada visualiseerimissüsteemi ettevõttele, saadi tagasisidet, et rakendus täidab nii funktsionaalseid kui ka mittefunktsionaalseid nõudeid. Testiti, et tagarakendusega tabeli struktuuri saamine ning andmete saamine töötaks ning nende tagastatud tulemused oleksid õiged. Testimisest on räägitud järgmises alapeatükis. Tagarakendus kasutab lõputöös valitud andmete valimise meetodit, mis kasutab PostgreSQL *row_number()* funktsiooni ning andmete talletamise viisiks materialiseeritud vaateid.

Andmebaasitabelis olevate kirjete arvud võivad tulevast visualiseerimisüsteemi kasutavatel klientidel erineda. Juhul, kui tahetud andmebaasitabeli kirjete arv on suur, siis ei ole võimalik mõnede jooniste puhul andeid üksteisest eristada ning selliseid jooniseid on kasutajatel raske analüüsida. Seda probleemi ei teki tagarakendusest saadud andmete visualiseerimisel. Tagarakendusest andmete pärimisel on võimalik valida, kui palju ning mis andmeid soovitakse andmebaasitabelist kuvada.

Lisaks on suurte andmebaasitabelite korral andmete pärimine aeglane, sest kalkulatsioone, andmete filtreerimist ning järjestamist tehakse läbi iga kord. Tagarakenduses käib see aga kiiresti. Materialiseeritud vaatega on kindel arv filtreeritud ning järjestatud andmed juba vaates olemas koos kalkulatsioonide tulemustega ning neid saab sealt otse pärida.

Samuti ei pruugi andmestiku kirjed omaette anda piisavalt informatsiooni. Tagarakendusega on aga võimalik saada lisa informatsiooni kirjete arvu, keskmise, summa, miinimumi ja maksimumi kohta. Üle kirjete on võimalik võtta funktsioone ning nendele lisada tingimusi, mille suhtes kalkulatsioone tehakse. Näiteks kui on tabel temperatuuri, riigi ja linna väljaga. Siis on võimalik sellest tabelist tagarakendusega saada temperatuuri keskmist nii riigi kui ka linna kohta. Tulemuse järgi võib järeldada, et lõputöös seatud eesmärgid on täidetud.

5.1 Tagarakenduse kasutamine

Selleks, et saada andmebaasitabeli struktuuri, tuleb teha päring API (*Application Program Interface*) *endpointile* ***http://localhost:9400/api/v1/table-structure/get-table-structure***. Sisend on JSON'i kujul, kus peab olema andmebaasitabeli nimi (Joonis 9). Juhul, kui andmebaasitabeli nime ei eksisteeri, tagastatakse veateade.

```
{
  "table_name": "hinded"
}
```

Joonis 9. Sisend tagarakendusest andmebaasitabeli struktuuri saamiseks.

Tagastatavas tabeli struktuuris on andmebaasi tabeli kõikide väljade nimed, väljade andmetüübid, indeksid ning kõik funktsioonid, mis sellest väljast vastava andmetüübiga võtta saab. Lisaks on tabeli struktuuris list võimalikest protsentidest, mitu protsenti kirjeid on andmebaasi kõikidest kirjetest võimalik saada ning iga protsendi juures vastav arv, mitu andmerida tagastatakse selle protsendi korral (Joonis 10). Tagastatavate kirjete arv ümardatakse alati alla.

```
{
  "columns": [
    {
      "column_name": "id",
      "data_type": "integer",
      "index_names": [
        "hinded_pkey"
      ],
      "possible_window_functions": ["avg","sum","min","max","count"]
    },
    {
      "column_name": "sugu",
      "data_type": "text",
      "index_names": null,
      "possible_window_functions": ["min","max","count"]
    },
    {
      "column_name": "hinne",
      "data_type": "integer",
      "index_names": null,
      "possible_window_functions": ["avg","sum","min","max","count"]
    }
  ],
  "possible_downsampling_percents": [
    {
      "percent": "100",
      "result_count": "6"
    },
    {
      "percent": "50",
      "result_count": "3"
    },
    {
      "percent": "25",
      "result_count": "1"
    }
  ]
}
```


Joonis 10. Andmebaasitabeli struktuuri saamise päringu vastus.

Selleks, et saada andmebaasitabelist andmeid, tuleb teha päring API *endpointile* ***http://localhost:9400/api/v1/table-structure/get-data*.** Sisendis on kohustuslikuks anda tabeli nimi, tahetud veergude nimed ja protsent. Võimalikud protsendid, mitu protsenti kirjeid kõikidst kirjetest on võimalik saada on 100, 50, 25, 20, 10, 5, 4, 2, 1, 0.5, 0.25, 0.2, 0.1, 0.05, 0.04, 0.02, 0.01 ja 0.001. 100 jagatud valitud protsendiga määrab, üle mitme rea võetakse kirje. Selle tõttu valiti vastavad protsendid, sest 100 jagatud nende

protsentidega on alati täisarv ning nii saab kõige täpsema protsendile vastava arvu tulemusi. Mitte kohustuslikeks osadeks on iga tahetud välja juures list funktsioonidest, kus on funktsiooni nimi ning välja nimi, millest sõltuvalt võetakse funktsioon. Funktsioonid võivad olla: *min* (minimaalne), *max* (maksimaalne) , *count* (arv), *sum* (summa) või *avg* (keskmine). Samuti järjekord kas väiksemast suuremaks ehk siis „ASC“ või siis suuremast väiksemaks ehk „DESC“ ning välja nimi, mille järgi tahetakse tulemust järjestada (Joonis 11).

```
{
  "table_name": "hinded",
  "columns": [
    {
      "column_name": "id"
    },
    {
      "column_name": "sugu"
    },
    {
      "column_name": "hinne",
      "window_functions": [
        {
          "function_name": "avg",
          "function_over": "sugu"
        },
        {
          "function_name": "max",
          "function_over": "sugu"
        }
      ]
    }
  ],
  "percent_to_downsample_to": "50",
  "order_by": {
    "column_name": "hinne",
    "order": "DESC"
  }
}
```

Joonis 11. Sisend andmebaasitabeli andmete saamiseks.

Selle päringu tulemusena tagastastatakse kirjed, kus on kõik tahetud väljad (Joonis 12). Väli *hinne_avg_over_sugu* on üle kõikide andmebaasitabeli kirjete võetud vastava soo keskmine hinne. Väli *hinne_max_over_sugu* on vastava soo maksimaalne hinne kogu andmebaasitabeli kirjetest. Tagastati kolm kirjet, sest protsendiks määrati 50 ning

andmebaasi tabelis „hinded“ on kuus kirjet. Sisendis määratud järjekorra järgi järjestati kõigepealt kõik andmebaasitabeli kirjed ning seejärel võeti üle iga kahe rea üks kirje. Väljad *bucket* ja *row_number* on vajalikud tagarakendusele ning neid tulevikus eesrakenduses kasutajale ei kuvata.


```
[
  {
    "id": 2,
    "sugu": "Male ",
    "hinne": 4,
    "hinne_avg_over_sugu": "3.33",
    "hinne_max_over_sugu": "4.00",
    "bucket": 1,
    "row_number": 2
  },
  {
    "id": 6,
    "sugu": " Female",
    "hinne": 4,
    "hinne_avg_over_sugu": "4.33",
    "hinne_max_over_sugu": "5.00",
    "bucket": 2,
    "row_number": 4
  },
  {
    "id": 3,
    "sugu": "Male ",
    "hinne": 3,
    "hinne_avg_over_sugu": "3.33",
    "hinne_max_over_sugu": "4.00",
    "bucket": 3,
    "row_number": 6
  }
]
```


Joonis 12. Andmebaasitabelist andmete saamise päringu vastus.

5.2 Testimine

Tagarakendust testiti lokaalselt. Testiti seda, et tagarakendus valideeriks sisendeid, tagastaks ainult küsitud veerud, jõuaks veergude kalkuleerimisel õigele tulemusele ning tagastaks õige arv kirjeid õiges järjekorras. Samuti seda, et rakenduse poolt tagastatud andmebaasitabeli struktuur on sama andmebaasitabeli struktuuriga andmebaasis.

5.3 Edasiarenduse võimalused

Tagarakendust on võimalik veel palju edasi arendada. Tagarakendusele on võimalik lisada juurde võimalus võtta üle adnmebaasitabeli väljade rohkem funktsioone, kui seda hetkel on lubatud. Lisaks sellele on võimalik lisada juurde ka protsente, mitu protsenti kirjetest soovitakse tagastada. Samuti on võimalik lisada funktsionaalsus, mis tabeli nime sisendi korral, uuendaks kõik materialiseeritud vaated, mis on selle tabeli jaoks tehtud. Lisaks oleks võimalik hakata arendama kasutajaliidest tagarakendusele, mille kaudu oleks võimalik kasutajasõbralikult küsida andmeid andmebaasist ning neid ka visualiseerida.

6 Kokkuvõte

Lõputöö eesmärgiks oli luua ettevõttele Avatud Lahendused OÜ visualiseerimisüsteemi tagarakendus. Tagarakendusest pidi olema võimalik saada visualiseerimiseks andmeid ükskõik missuguse struktuuriga andmebaasitabelist. Oli vaja luua võimalus valida, kui palju andmeid soovitakse saada, mis tabeli välju soovitakse visualiseerida ning mille järgi neid järjestada. Samuti pidi olema võimalik pärida lisaks väljadele ka üle väljade tehtud kalkulatsioone, mis annavad juurde informatsiooni andmete analüüsiks. Lisaks sellele pidi andmete pärimine tagarakendusest olema kiire.

Eesmärgi täitmiseks analüüsiti kahte meetodit teatud arv andmete valimiseks andmebaasitabelist. Analüüsiti Suurim-Kolmnurk-Kolm-Gruppi algoritmi ning PostgreSQL *row_number()* funktsiooni kasutamist andmete valimiseks. Kõigepealt pandi koos Avatud Lahendused OÜ peaarendajaga paika funktsionaalsed ja mittefunktsionaalsed nõuded. Seejärel kasutati näidisandmebaasi meetodite analüüsiks. Tehti meetodit realiseeriv päring ning siis visualiseeriti andmebaasist tagastatud andmeid joondiagrammiga. Mõlemat joondiagrammi võrreldi kõikide andmete joondiagrammiga ning toodi välja meetodite tugevused ja nõrkused. Analüüsi tulemusena valiti PostgreSQL *row_number()* funktsioon, mida kasutati tagarakenduses andmete valimiseks. Seejärel analüüsiti andmebaasist päritud andmete kahte talletamise viisi. Jsonb andmetüübina kirjete hoidmist tabelis ning materialiseeritud vaateid. Pandi paika nõuded koos ettevõtte peaarendajaga. Toodi välja mõlema viisi nõrkused ja tugevused ning valiti materialiseeritud vaated tagarakenduses päringute optimeerimiseks. Tagarakendusele määrati koos ettevõtte peaarendajaga funktsionaalsed ja mittefunktsionaalsed nõuded. Tagarakendus realiseeriti ning kirjeldati tagarakenduse realiseerimise loogikat.

Arendatud tagarakenduse funktsionaalsuseid testiti lokaalselt. Samuti Avatud Lahendused OÜ peaarendaja, kelle ideeks oli arendada visualiseerimissüsteemi andis tagasisidet, et tagarakendus täidab seatud funktsionaalseid ja mittefunktsionaalseid nõudeid

Kasutatud kirjandus

[1] P. Coppens, *Sampling time series data sets,* 2019. [Online]. Loetud aadressil: https://medium.com/@hayley.morrison/sampling-time-series-data-sets-fc16caefff1b. Kasutatud 12.05.2021.

[2] S. Steinarsson, *Downsampling Time Series For Visual Representation*, 2013. [Online]. Loetud aadressil: https://skemman.is/bitstream/1946/15343/3/SS_MSthesis.pdf. Kasutatud 12.05. 2021.

[3] PostgreSQL, *3.5. Window Functions*. [Online]. Loetud aadressil: https://www.postgresql.org/docs/9.1/tutorial-window.html. Kasutatud 12.05. 2021.

[4] PostgreSQL tutorial, *PostgreSQL ROW_NUMBER Function*. [Online]. Loetud aadressil: https://www.postgresqltutorial.com/postgresql-row_number/. Kasutatud 12.05.2021.

[5] PostgreSQL, 9.20. *Aggregate Functions*. [Online]. Loetud aadressil: https://www.postgresql.org/docs/9.5/functions-aggregate.html. Kasutatud 13.05 .2021.

[6] PostgreSQL, *CREATE MATERIALIZE VIEW*. [Online]. Loetud aadressil: https://www.postgresql.org/docs/current/sql-creatematerializedview.html. Kasutatud 17.05.2021.

[7] M. Nedrich, *An Introduction to PostrgeSQL Materialized Views*. [Online]. Available: https://spin.atomicobject.com/2018/04/09/postgres-materialized-views/. Kasutatud 12.05.2021.

[8] PostgreSQL, *ALTER TABLE*. [Online]. Loetud aadressil: https://www.postgresql.org/docs/current/sql-altertable.html. Kasutatud 17.05. 2021.

[9] PostgreSQL, *8.14. JSON Types*. [Online]. Loetud aadressil: https://www.postgresql.org/docs/9.4/datatype-json.html. Kasutatud 12.05.2021.

[10] PostgreSQL, *9.15. JSON Functionas and Operators*. [Online]. Loetud aadressil: https://www.postgresql.org/docs/12/functions-json.html. Kasutatud 12.05.2021.

[11] E. Markou, *Storing JSON in PostgreSQL: A must-know feature*, 2018. [Online]. Loetud aadressil: https://www.blendo.co/blog/storing-json-in-postgresql/. Kasutatud 12.05.2021.

[12] Python, *What is Python? Executive summary.* [Online]. Loetud aadressil: https://www.python.org/doc/essays/blurb/. Kasutatud 12.05.2021.

[13] Tutorialspoint, *Python - Overview*. [Online]. Loetud aadressil: https://www.tutorialspoint.com/python/python_overview.htm. Kasutatud 12.05. 2021.

[14] Y. Petlovana, *Top 13 Python eb Frameworks to Learn in 2020.* [Online]. Loetud aadressil: https://steelkiwi.com/blog/top-10-python-web-frameworks-to-learn/. Kasutatud 12.05.2021.

[15] PostgreSQL, *About*. [Online]. Loetud aadressil: https://www.postgresql.org/about/. Kasutatud 12.05.2021.

[16] Stack Overflow, *List columns with indexes in PostgreSQL*, 2010. [Online]. Loetud aadressil: https://stackoverflow.com/questions/2204058/list-columns-with-indexes-in-postgresql. Kasutatud 12.05.2021.

[17] PostgreSQL, *Chaper 45. System Catalogs*. [Online]. Loetud aadressil: https://www.postgresql.org/docs/9.1/catalogs.html. Kasutatud 12.05.2021.

[18] J. Branchaud, *PostgreSQL's Max Identifier Lenght is 63 Bytes*. [Online]. Loetud aadressil: https://til.hashrocket.com/posts/8f87c65a0a-postgresqls-max-identifier-length-is-63-bytes. Kasutatud 13.05.2021.

[19] PYnative, *Python UUID Module to Generate Universally Unique Identifiers*, 2021. [Online]. Loetud aadressil: https://pynative.com/python-uuid-module-to-generate-universally-unique-identifiers/. Kasutatud 10.05.2021.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Elisabeth Abel

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Tagarakenduse loomine visualiseerimiseks sobivate andmete saamiseks PostgreSQL'iga“, mille juhendaja on Toomas Lepikult
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

17.05.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.