TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDU40LT

Maria Ossipova 135215IAPB

MITME VÄITE ÜHE ANDMEVÄÄRTUSENA ESITAMISE EELISED JA PUUDUSED SQL-ANDMEBAASIDES

Bakalaurusetöö

Juhendaja: Erki Eessaar
doktor
dotsent

Tallinn 2016

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: Maria Ossipova

23.05.2016

Annotatsioon

Sageli esineb olukord, et ühe ja sama andmebaasi probleemi lahendamiseks võib kasutada mitut erinevat andmebaasi disaini, mis kasutavad erinevat andmebaasi struktuuri ja jõustavad erinevatel viisidel nõutud piiranguid andmetele.

Mitme väite ühe andmeväärtusena esitamine tähendab, et mitu reaalse maailma kohta käivat väidet on mingi eeskirja kohaselt koondatud kokku üheks väärtuseks (näiteks massiivi).

Käesoleva bakalaurusetöö eesmärgiks on uurida võimalusi SQL-andmebaasis mitme väite ühe andmeväärtusena esitamiseks ning leida erinevate disainilahenduste eeliseid ja puuduseid. Uuringu käigus tutvutakse selle valdkonna kohta ilmunud materjalidega ja tehakse eksperimente. Töö mahtu arvestades tehakse katsetusi ühe andmebaasisüsteemi põhjal, milleks on valitud PostgreSQL (9.4.4). Kuna erinevates andmebaasisüsteemides on kasutatavad erinevad tüübid ja operaatorid, siis andmebaasisüsteemi valik määrab palju selles osas, milliseid andmebaasi disaine on põhimõtteliselt võimalik kasutada.

Bakalaurusetöö eksperimendi osas on kavas realiseerida erinevatel viisidel reiside ja nende toimumise nädalapäevade andmebaas hüpoteetilise linnadevahelise transpordi ettevõtte jaoks ning võrrelda neid disaine otsingu kiiruse, andmemuudatuse kiiruse ning koodi keerukuse seisukohalt. Lisaks „traditsioonilisele“ disainile, kus väiteid ühte andmeväärtusesse ei koondata, käsitletakse mitme väite väärtusesse koondamist ja esitamist VARCHAR tüüpi veerus, kahemõõtmelises massiivis, ning vähem või rohkem hierarhilisena vastaval JSON ja JSONB tüüpi veerus.

Töö lõpuks selgitatakse välja mitme väite ühe andmeväärtusena esitamise eelised ja puudused ning tõstetakse esile autori arvates antud juhul kõige sobivam ja ebasobivam disain.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 48 leheküljel, 6 peatükki, 11 joonist, 7 tabelit.

Abstract

Advantages and Disadvantages of Representing Multiple Propositions as a Data Value in SQL Databases

There is often a possibility of a situation, where we might solve a single database problem with the use of different database designs, which offer a variety of database structures and enforce required data restrictions in different ways.

The purpose of this thesis is to explore the possibilities of representing multiple propositions as a data value in SQL-databases and to find advantages and disadvantages of the different design solutions. We explore existing materials of this field as well as make experiments during the research. Due to the expected size of the thesis and the expected amount of work, we will conduct the experiments based on one database management system (DBMS) that is PostgreSQL (9.4.4). The selection of the DBMS, its provided types and operators, influence a lot the database designs that one can possibly use in the databases created in the system.

Representing multiple propositions as a data value means that several assertions about the real world are pooled together into a single value (string or array, for example) according to some rules.

In the experimental part of the thesis we plan to realize a database where the data about the trips and the days of the week when they occur are stored in a variety of ways. One could use the database in a hypothetical company that organizes intercity transportation. Firstly, we will create a conceptual data model of the database. According to its represented requirements, we offer a variety of database designs, where days of the week, times of occurrence, and actual trips are stored in different ways. Firstly, we use "traditional" database design. In addition, we use designs where multiple propositions are bundled together to a value. These values are in the columns with the VARCHAR, multidimensional array, JSON and JSONB type. We will create a database with multiple schemas and fill tables of different designs with the same propositions. We will

conduct multiple tests to compare the search speed, data modification speed, and the physical lines of the code in case of the designs.

As the result of this thesis, we identify the advantages and disadvantages of representing multiple propositions as a data value. We highlight in our view the most suitable and the most unsuitable database designs for the current context.

The thesis is in Estonian and contains 48 pages of text, 6 chapters, 11 figures, 7 tables.

Lühendite ja mõistete sõnastik

CASE	Computer-Aided Software Engneering
JSON	JavaScript Object Notation
JSONB	Binary JSON
NoSQL	Non SQL, non relational, Not Only SQL
SLOC	Source lines of Code
SQL	Structured Query Language

Sisukord

1 Sissejuhatus 11
1.1 Taust ja probleem 11
1.2 Ülesandepüstitus 12
1.3 Metoodika 12
1.4 Ülevaade tööst 13
2 Teoreetiline taust 14
2.1 Mitu väidet ühe andmeväärtusena 14
2.2 Vektorkodeerimine 15
2.3 Masiivid PostgreSQL andmebaasisüsteemis 16
2.4 JSON PostgreSQL andmebaasisüsteemis 17
3 Eksperiment 18
3.1 Varasemad uuringud 18
3.2 Eksperimendi eesmärgid 19
3.3 Eksperimendi andmebaasi valdkonna kirjeldus 20
3.4 Eksperimendi kirjeldus 21
3.4.1 Kasutatavad andmebaasisüsteemid 22
3.5 Eksperimendi andmebaaside disainimine 23
3.5.1 Andmebaasi disaini mudel „traditsioonilise“ disaini kasutamise korral 24
3.5.2 Andmebaasi disaini mudel vektorkodeerimise kasutamise korral 25
3.5.3 Andmebaasi disaini mudel kahemõõtmelise massiivi kasutamise korral 26
3.5.4 Andmebaasi disaini mudel JSON tüüpi kasutamise korral 28
3.5.5 Andmebaasi disaini mudel hierarhilise JSONB tüüpi kasutamise korral 29
3.6 Testandmed 31
3.7 Eksperimendi käigus katsetavad operatsioonid 33
4 Eksperimendi tulemused 35
5 Tulemuste analüüs 38
5.1 Päringute ja andmemuudatuste operatsioonide kiirused 38
5.2 Koodi keerukus 41
5.3 Järeldused 42

6 Kokkuvõte 45
Kasutatud kirjandus 46
Lisa 1 – “Traditsioonilise” andmebaasi loomise laused 49
Lisa 2 - Vektorkodeerimisega andmebaasi loomise laused 51
Lisa 3 - Kahemõõtmelise massiiviga andmebaasi loomise laused 53
Lisa 4 - JSON tüüpi veeruga andmebaasi loomise laused 55
Lisa 5 - Hierarhilise JSONB tüüpi veeruga andmebaasi loomise laused 57
Lisa 6 – Testandmete sisestamise laused „traditsioonilise“ disaini tabelitesse 59
Lisa 7 – Testandmete sisestamine vektorkodeerimisega disaini tabelitesse 60
Lisa 8 – Testandmete sisestamine kahemõõtmelise massiiviga disaini tabelitesse 62
Lisa 9 – Testandmete sisestamine JSON tüüpi veeruga disaini tabelitesse 64
Lisa 10 – Testandmete sisestamine hierarhilise JSONB tüüpi veeruga disaini tabelitesse
.... 66
Lisa 11 – Päringu S1 laused 68
Lisa 12 – Päringu S2 laused 70
Lisa 13 – Päringu S3 laused 73
Lisa 14 – Päringu S4 laused 75
Lisa 15 - Operatsiooni I1 laused 77
Lisa 16 – Päringute tegemiseks loodud lisafunktsioonid 79

Jooniste loetelu

Joonis 1. Linnadevahelise transpordi olemi-suhte diagramm. 20
Joonis 2. „Traditsiooniline“ andmebaasi disain 24
Joonis 3. Vektorkodeerimisega andmebaasi disain. 25
Joonis 4. Kahemõõtmelise massiiviga andmebaasi disain. 27
Joonis 5. JSON tüüpi veeruga andmebaasi disain. 28
Joonis 6. Hierarhilise JSONB tüüpi veeruga andmebaasi disain. 30
Joonis 7. Hierarhilised andmed JSONB tüüpi veerus. 31
Joonis 8. Päringu S1 T SQL-lause. 38
Joonis 9. Päringu S1 T täitmisplaan PostgreSQL andmebaasisüsteemis. 39
Joonis 10. Päringu S1 H SQL-lause. 39
Joonis 11. Päringu S1 H täitmisplaan PostgreSQL andmebaasisüsteemis. 40

Tabelite loetelu

Tabel 1. Ridade arv „traditsioonilise“ disaini tabelites. 33
Tabel 2. Päringu S1 kiiruse mõõtmise tulemused millisekundites. 35
Tabel 3. Päringu S2 kiiruse mõõtmise tulemused millisekundites. 36
Tabel 4. Päringu S3 kiiruse mõõtmise tulemused millisekundites. 36
Tabel 5. Päringu S4 kiiruse mõõtmise tulemused millisekundites. 36
Tabel 6. Operatsiooni I1 kiiruse mõõtmise tulemused millisekundites.......................... 37
Tabel 7. Füüsiliste koodiridade arv erinevate disainide korral. 37

1 Sissejuhatus

Viimasel ajal on peale SQL-andmebaasisüsteemide populaarseks muutunud ka NoSQL (ingl *Not Only SQL*) andmebaasisüsteemid [1]. Sellised andmebaasisüsteemid hoiavad andmeid näiteks dokumentidena ning andmete pärimine ja muutmine toimub SQL-andmebaasisüsteemidest teistmoodi. NoSQL andmebaasisüsteemid omavad aga piiratud funktsionaalsust [2]. Seda arvestades tuli autorile mõte, et andmeid võib SQL-andmebaasis hoida erineval viisil, näiteks, kasutada SQL-andmebaasisüsteemis NoSQL andmebaasisüsteemi funktsionaalsust. Sellisel juhul peaksid NoSQL andmebaasisüsteemide puudused olema elimineeritud. Pärast väikese uuringu läbiviimist tuli autor selle peale, et PostgreSQL andmebaasisüsteem toetab hulka andmetüüpe, mille abil oleks võimalik hoida andmebaasis andmeid „traditsioonilisest“ viisist erinevalt.

1.1 Taust ja probleem

Kui süsteemide arendamise juures on üldse midagi kindlat siis see, et süsteemides tuleb teha muudatusi. Üldiselt on süsteemide muutmine keeruline. Eriti keeruline on muuta andmebaase, sest need on süsteemi keskseteks osadeks, millest sõltuvad kõik teised osad (lähtekood, mudelid, testid). Sellised muudatused, kus tuleb hakata kohe tegema ringi disainiotsuseid, sest algsed otsused olid halvad, on ressursi raiskamine. Peaks vähemalt üritama süsteeme teha nii, et nende disaini kohene muutmine poleks vajalik. Heade disainiotsuste, mida ei pea kohe kahetsema hakkama, langetamiseks on vaja objektiivseid andmeid. Konkreetsed katsetused koos arvuliste mõõtmistulemustega pakuvad selliseid andmeid. Edasi on juba iga disaineri otsustada, millised kriteeriumid (nt jõudlus, lihtsus, ressursi kasutus) on loodava süsteemi jaoks olulisemad ja millised vähemolulised ning teha vastavaid disainivalikuid. Käesolev töö on kasulik PostgreSQLil põhinevate SQL-andmebaaside disainiotsuste tegijatele. See pakub erinevatele süsteemi kvaliteedi aspektidele vastavaid mõõtmistulemusi, mida disainerid saavad otsuste tegemisel argumentidena kasutada.

Otseste koodinäidete ja mõõtmistulemuste poolest on see töö kasulik PostgreSQLi kasutajatele. Kuid info üldiste lahendusvõimaluste kohta oleks kasulik mistahes SQL-andmebaasisüsteemide kasutajatele ning ka nendele, kes alles otsustavad, millisel andmemudelil põhinevat süsteemi kasutada.

1.2 Ülesandepüstitus

Järgnevalt esitatakse käesoleva bakalaurusetöö eesmärgid.

- Uurida võimalusi SQL-andmebaasis mitme väite ühe andmeväärtusena esitamiseks.
- Leida erinevate disainilahenduste eeliseid ja puuduseid.
- Selgitada välja, milline andmebaasi disain on vaadeldavas kontekstis kõige sobilikum ning milline on kõige ebasobilikum.

Töö mahu arvestades tehakse katsetusi ühe andmebaasisüsteemi põhjal, milleks on valitud PostgreSQL (9.4.4). Kuna erinevates andmebaasisüsteemides on kasutatavad erinevad tüübid ja operaatorid, siis andmebaasisüsteemi valik määrab palju selles osas, milliseid andmebaasi disaine on põhimõtteliselt vaja kasutada.

1.3 Metoodika

Eesmärki saavutamiseks pakutakse välja viis SQL-andmebaasi disaini. Kuna igat disainilahendust tuleb testida ning selle headust hinnata, siis vältimaks töö mahu ületamist, on valitud need viis andmebaasi disaini, mis erinevad üksteisest kõige rohkem. Andmebaasi disaine võrreldakse omavahel ja viiakse läbi testid.

Töö käigus luuakse ühe hüpoteetilise ettevõte vaates linnadevahelise transpordi infosüsteemi kontseptuaalne andmemudel. Seejärel modelleeritakse kontseptuaalses andmemudelis esitatud nõuete alusel viis erinevat andmebaasi disainilahendust, kus reiside toimumise nädalapäevasid ja kellaaegu hoitakse erineval viisil. Esimese andmebaasi disaini puhul kasutatakse „traditsioonilist“ andmebaasi disaini. „Traditsiooniline“ andmebaasidisain tähendab, et iga väidet hoitakse eraldi andmeväärtusena. Teise andmebaasi disaini puhul hoitakse reiside toimumise nädalapäevasid ja kellaaegu VARCHAR tüüpi veerus. Kolmas andmebaasi disain kasutab andmete hoidmiseks kahemõõtmelist massiivi. Neljandas andmebaasi disainis

leidub JSON-tüüpi veerg ning viiendas andmebaasi disainis kasutatakse hierarhilist andmete esitamist JSONB-tüüpi veerus. Andmebaasi disainide modelleerimiseks kasutatakse CASE (*Computer-Aided Software Engineering*) vahendit Rational Rose. Andmebaasid realiseeritakse tasuta pakutavas, avatud lähtekoodiga ja populaarses andmebaasisüsteemis PostgreSQL (9.4.4).

Kõikidele disainidele vastavatessse tabelitesse lisatakse ühesuguseid fakte esitavad testandmed. Testandmete maht peab olema piisavalt suur, et oleks võimalik hinnata andmebaasi käitumist suurte andmehulkade puhul. Eksperimendi käigus viiakse läbi mitmeid teste, mille tulemusena võrreldakse andmebaasi disaine. Testitakse otsingute kiirust, andmemuudatuste kiirust ning koodi keerukust. Viimase mõõtmiseks kasutatakse koodiridade arvu metoodikat, mõõtes füüsilist koodiridade arvu.

1.4 Ülevaade tööst

Käesolev töö sisaldab nelja põhipeatükki. Esimeses peatükis selgitatakse mitme väite ühe andmeväärtusena esitamise põhimõtet ja tutvustatakse töö eksperimendi osas kasutatavaid andmetüüpe. Teises peatükis tutvustatakse autori poolt antud teema kohta leitud varasemaid uuringuid, täpsustatakse eksperimendi eesmärke, kirjeldatakse eksperimendi käiku ning esitatakse eksperimendi käigus katsetavaid päringuid. Kolmandas peatükis pannakse kirja eksperimendi käigus saadud tulemused. Töö neljandas peatükis analüüsitakse saadud tulemusi ning tehakse nende põhjal järeldusi.

2 Teoreetiline taust

Käesolevas peatükis antakse ülevaade mitme väite ühe andmeväärtusena esitamisest, vektorkodeerimisest, massiividest ning JSON tüüpi väärtustest PostgreSQL (9.4.4) andmebaasisüsteemis. Kuna eksperimentides on kasutusele võetud sellisel viisil andmeid esitada võimaldavaid andmetüüpe kasutatavad andmebaasidisainid, siis on oluline selgitada nende andmetüüpide iseärasusi.

2.1 Mitu väidet ühe andmeväärtusena

Tänu inimkeele suure sõnavarale on selles võimalik öelda ühte ja sama mõtet mitmel erineval viisil, kasutades erinevaid sõnu ja lausekonstruktsioone. Selline võimalus teeb keelt palju huvitavamaks ja vaheldusrikkamaks. Nii nagu mõtet saab öelda erinevate lausetega, saab ka andmeid hoida andmebaasis mitmel viisil. Näiteks, saab mitu väidet esitada ühe andmeväärtusena. PostgreSQL andmebaasisüsteem pakub suure hulga erinevaid andmetüüpe, mis annavad võimaluse „mängida“ ja proovida esitada oma andmeid erinevalt. „Traditsioonilise“ andmebaasi disaini puhul on iga erinevat tüüpi väite jaoks eraldi tabel või veerg. PostgreSQL andmebaasisüsteemi andmetüüpide uurimise tulemusena autor leidis andmetüübid, mis oleksid sobivad mitme väite ühe andmeväärtusena esitamiseks. VARCHAR tüüpi veerg on sobilik semantilise kodeerimise kasutamiseks. Kahemõõtmelise massiivi ühes dimensioonis saab näiteks hoida linnatranspordi väljumise nädalapäevasid, teises dimensioonis aga väljumise kellaaegu. JSON ja JSONB andmetüübid võimaldavad esitada võtme-väärtuse paaride abil objekte, kus objekti võtmeks võib olla näiteks väljumise nädalapäev, väärtuseks aga väljumise kellaaeg. Andmeid võib esitada ka mitmetasemelise hierarhiana.

PostgreSQL andmebaasisüsteem toetab ka XML andmetüüpi [3]. See andmetüüp vastab käesoleva töö teema nõuetele, kuid see jäetakse skoobist välja, sest [4] kohaselt on XML sarnane JSON tüübile, aga seda tüüpi väärtuseid on keerulisem kasutada. Lisaks bakalaurusetöö mahupiirangutele oli otsuse langetamisel veel üheks argumendiks vajadus andmebaasisüsteemi XML kasutamiseks spetsiaalselt insalleerida [4].

2.2 Vektorkodeerimine

Vektorkodeerimine ehk semantiline kodeerimine on selline kodeerimise viis, kus hulk informatsiooni koondatakse kokku ühte stringi. Igas stringi positsioonis oleval sümbolil on oma tähendus ning stringi saab dekodeerida teades kodeerimise põhimõtteid. Semantilist kodeerimist kasutatakse igapäevases elus. Näiteks ülikoolis kodeeritakse õppeained ja lõputööd määrates neile ainekoodid. TTÜs 2016.aastal kasutatavad koodid sisaldavad näiteks infot ainet õpetava instituudi kohta. Teise näitena on igal Eesti residendil (sh e-residendil) oma (üldjuhul) unikaalne isikukood, kus esimene number määrab sünni sajandi ja isiku soo, järgmised kaks numbrid määravad sünniaasta. Sünniaastale järgnevad neli numbrit, mis määravad sünni kuu ning sünni päeva. Isikukoodis hoitakse andmeid ka selle kohta, mitmendana on isik sündinud sellel päeval ning viimane number isikukoodis on kontrollnumber [5]. Isik, kes pole kunagi isikukoodiga kokku puutunud, võtab isikukoodi nagu suvalist numbrite komplekti, kuid Eesti residendid võivad teada numbrite tähendust ja päritolu.

Semantilisel kodeerimisel on ka omad negatiivsed küljed. Nimelt, võib tekkida vajadus kodeeringut muuta, sest eelnev enam ei sobi, sest ei arvestata kodeeritavate objektide hulga või muutunud nõuetega. Nii juhtus näiteks Rootsis, kus isikukoodidest tekkis suure põgenike tulva tõttu puudus [6].

Vektorkodeerimise kasutamine andmebaasis ei ole alati mõistlik. Kui näiteks hoida tabelis *Isik* nii Eesti isikukoodi, kui ka sünniaega ja sugu, siis tekib andmete liiasus. See tähendab, et sama fakti on võimalik andmebaasist tuletada rohkem kui ühel viisil. Kontrollitud (st vastuolusid vältiva) liiasuse tagamisel tuleks iga *Isik* tabelisse lisatava või muudetava rea puhul alati kontrollida, et sisestatavad sünnikuupäev ja sugu oleksid isikukoodiga kooskõlas. Teine variant oleks hoida ainult isikukoodi, aga sellisel juhul sünnikuupäeva või soo tunnuse kättesaamiseks tuleks valmis kirjutada isikukoodi dekodeerimise funktsioonid. Kui andmebaasis soovitakse hoida erinevate riikide isikukoode, siis läheb selline kooskõla kontrollimine veel palju keerulisemaks.

Tuleb kindlasti mainida ka teisi semantilise kodeerimise puuduseid.

- Effektiivseks vektorkodeerimise kasutamiseks peab kasutajale kindlasti selgitama dekodeerimise algortimi. Vastasel juhul on need andmed kasutud.

- Andmeid kasutav rakendus peab dekodeerimisalgoritmi kasutades viima kodeeritud andmed kasutajale arusaadavale kujule.
- Kodeeritud andmete alusel otsingu ja sorteerimisoperatsioonide tegemine võib nõuda eraldi funktsioonide kasutamist, mis omakorda nõuab funktsioonidel põhinevate indeksite kasutamist ning välistab sellele veerule loodud „tavalise“ indeksi kasutamise.
- Juhul, kui kodeeringut muudetakse, siis tuleb teha muudatused ka andmebaasis ning seda kasutavates programmides. Näiteks tuleb muuta veeru pikkust, sellega seotud kontrollkitsendusi ja kodeeritud andmeid kasutavaid programme ning kindlasti tuleb muuta andmete kodeerivaid ja dekodeerivaid rutiine.

2.3 Masiivid PostgreSQL andmebaasisüsteemis

Üks huvitavatest PostgreSQL andmebaasisüsteemi iseärasustest on massiivitüübi konstruktori toetamine, mis võimaldab kasutada massiivtüüpi veerge. Seda võimalust paljud SQL-andmebaasisüsteemid ei paku ning isegi need, mis massiive mingil määral toetavad, ei paku nii lihtsat kasutusvõimalust. Massiivid on küll relatsioonilistest struktuuridest kaugel, aga PostgreSQL teeb nende kasutamise päringutes väga lihtsaks.

PostgreSQL võimaldab tabelite massiivtüüpi veergudes hoida erineva pikkusega mitmemõõtmelisi massiive. Massiivtüüpe on võimalik defineerida süsteemi- või kasutaja defineeritud baastüüpide, loendtüüpide või liittüüpide põhjal. Massiivide defineerimine domeenide põhjal ei ole töös kasutatavas PostgreSQL andmebaasisüsteemis (9.4.4) veel toetatud. Massiivüüpi konstruktori poole pöördutakse lisades nurksulud ([]) massiivi elementide andmetüübi nimetusele. Teine võimalus on kasutada võtmesõna ARRAY, aga seda ainult ühemõõtmelise massiivi puhul. Nurksulgude sees olev arv defineerib massiivi suuruse, kuid tegelikkuses käsitleb süsteem sellist massiivi sarnaselt defineerimata suurusega massiivile. Sama olukord kehtib ka dimensioonide korral. Seega on massiivi suuruse ja dimensioonide arvu deklareerimine CREATE TABLE lauses kõigest dokumentatsioon.

Mitmemõõtmelistes massiivides peavad kõik massiivi dimensioonid olema ühe suurusega. Samuti peavad igas dimensioonis olevad väärtused olema ühte tüüpi. Näiteks, kui dimensioon on defineeritud kui numbriline massiiv, ei saa seal massiivi väärtuses olla teksti, vastasel juhul väljastatakse viga.

PostgreSQL andmebaasisüsteem eeldab, et õigesti kirjutatud massiivis on komadega eraldatud ja looksulgudesse paigutatud väärtused. Kui väärtuses sisaldub koma või looksulg, siis peaks väärtuse ümber paigutada jutumärgid.

Masiivide võrdlemisel võrreldakse massiivide sisu elementide kaupa, kasutades tavalist B-puu võrdlusfunktsiooni [7].

2.4 JSON PostgreSQL andmebaasisüsteemis

JSON andmetüüp on loodud JSON (*JavaScript Object Notation*) formaadis andmete hoidmise võimaldamiseks. „JSON on lihtsustatud andmevahetusvorming, mis põhineb JavaScripti programmeerimiskeele alamhulgal. JSON on tekstivormingus ja programmeerimiskeelest sõltumatu.“ [8]. JSON formaadis andmeid on võimalik hoida ka tavalise tekstina, kuid JSON tüübi kasutamisel on omad eelised. Üks nendest on JSON-tüübile spetsiifiliste funktsioonide ja operaatorite kasutamise võimalikuks muutumine.

PostgreSQL toetab kahte JSON andmetüüpi: *json* ja *jsonb*. Seda tüüpi väärtuseks teisendamise funktsioonid aktsepteerivad sisendina peeaaegu identse komplekti väärtuseid, kuid nendel tüüpidel on üks suur praktiline vahe effektiivsuses.

Json tüüpi väärtus sisaldab seda tüüpi väärtuseks teisendamise funktsioonile sisendiks antud teksti täpset koopiat, mida töötlemisfunktsioonid peavad igal väljakutsumisel ümber sõeluma. Selline *json* tüübi omapära põhjustab väärtuses tühja ruumi märkide vahel ning võtmete järjekorra säilitamise. Kui *json* väärtuses e objektis on mitu ühesugust võtit, siis hoitakse alles kõiki võtme-väärtuse paare.

Jsonb tüübi korral salvestatakse andmeid tükeldatuna binaarformaadis, mis küll aeglustab andmete sisestamist, aga hilisem töötlemiskiirus on märgatavalt suurem, sest ümbersõelumist tarvis ei ole. Samuti toetab *jsonb*. *Jsonb* ei säilita tühja ruumi, võtmete järjekorda ning ei hoia alles korduvaid võtmeid. Mitmete ühesuguste võtmete sisestamise puhul säilitatakse neist viimane. Erinevalt *json* tüübist, lükkab *jsonb* tagasi arvud, mis on PostgreSQL *numeric* tüübi määratud arvude hulgast väljaspool [9].

Andmete esitamine JSON formaadis lisab mõnel juhul süsteemi paindlikust.

3 Eksperiment

Selles peatükis kirjeldatakse käesolevas töös läbiviidavat eksperimenti.

3.1 Varasemad uuringud

Autor eeldas, et käesolevas töös kasutatud andmetüüpide „uudsuse“ pärast (näiteks, JSON andmetüüpi toetatakse PostgreSQL andmebaasisüsteemis alates versioonist 9.2) on kindlasti tehtud uuringuid ja kirjutatud artikleid, kus tõstetakse esile nende tüüpide kasutamise nõrgad ja tugevad küljed. Uudishimulikud arendajad on kindlasti teinud eksperimente ja formuleerinud oma arvamust nende kasutamise mugavusest. Artiklite ja uuringute leidmiseks on kasutatud otsingumootorit *Google* (http://www.google.com). Autor otsis infot järgmiste otsingufraaside järgi.

- *JSON in PostgreSQL*
- *JSONB in PostgreSQL*
- *Multidimensional arrays in PostgreSQL*

Järgnevalt esitatakse autori poolt otsingu tulemusena leitud artiklid.

Mei [2] on inimene, kes on tihedalt seotud veebirakenduste arendamisega ja kes teab hästi, milline peaks olema andmebaas, et veebirakendused töötaksid tõrgeteta ja oleksid kergelt evolutsioneeritavad. Antud töös tõstetakse esile põhjused, mille poolest on SQL-andmebaasid NoSQL andmebaasidest paremad ning mis probleemid võivad tekkida arendajal JSON dokumente kasutatava andmebaasi projekteerimisel ja haldamisel.

Robinsoni [10] töös räägitakse sellest, et massiive PostgreSQL andmebaasides ei tasuks kasutada tavaliste massiividena, nagu neid kasutatakse enamuses programmeerimiskeeltes. Töös selgitatakse õiget massiivide kasutamisviisi andmebaasides ja antakse nõu kuidas teha massiivide kasutamist andmebaasis effektiivsemaks.

Ringeri [11] töös soovitatakse keelduda JSONB tüüpi kasutamisest andmebaasides juhul, kui see pole tingimata vajalik. Segitatakse sellise soovituse põhjusi ning tuuakse näiteid. Samuti selgitatakse antud töös mis juhtudel võib kaaluda JSON tüübi kasutamist oma andmebaasis.

Halliday [12] töös räägitakse sellest, kuidas kasutada SQL-andmebaasisüsteemi NoSQL andmebaasisüsteemina ja tuuakse näiteid JSON kasutamisest PostgreSQL andmebaasisüsteemis. Samuti, tuuakse näiteid miks JSON tüüpi üldse toetatakse ja mis juhtudes oleks mõistlik ja vajalik kasutada just JSON tüüpi.

Töös [13] räägitakse sellest, kas tasuks kasutada PostgreSQL andmebaasisüsteemi NoSQL andmebaasisüsteemina ja tuuakse näiteid, miks see on halb mõte. Selgitatakse JSONB tüübi häid külgi ning JSON ja JSONB tüüpide erinevusi.

Maksimov [14] töös võrreldakse andmete lugemise, kirjutamise ja uuendamise jõudlust „traditsioonilise" andmebaasisüsteemi PostgreSQL (koos JSON andmetüüpidega) puhul ja NoSQL andmebaasisüsteemi puhul. Töö tulemusena on tehtud järeldus, et PostgreSQL „traditsiooniline" andmebaasi disain on madalama jõudlusega, kui PostgreSQL andmebaasi disain, kus on kasutatud JSON andmetüüpe. Samuti on leitud, et andmete lisamise ja uuendamise operatsioonid NoSQL andmebaasisüsteemi puhul on kiiremad, kui PostgreSQL'i (koos JSON andmetüüpidega). Autor on teinud järelduse, et JSON andmetüüpide kasutus PostgreSQL andmebaasisüsteemis ei saa asendada NoSQL andmebaasisüsteeme.

3.2 Eksperimendi eesmärgid

Käesoleva eksperimendi eesmärgiks on uurida võimalusi SQL-andmebaasis mitme väite ühe andmeväärtusena esitamiseks ning leida erinevate disainilahenduste eeliseid ja puuduseid. Eksperimendis võrreldakse „traditsioonilist" andmebaasidisaini, kus mitut väidet ühte andmeväärtusesse kokku ei panda, selliste andmebaasi disainidega, kus mitu väidet tuleb esitada ühe andmeväärtusena. Nende disainide kohaselt hoitakse selliseid andmeid VARCHAR, kahemõõtmelise massiivi, JSON ja JSONB tüüpi veergudes.

Uuritavaks andmebaasisüsteemiks on valitud PostgreSQL (9.4.4), sest see on kättesaadav Tallinna Tehnikaülikooli serveris ning autor on valitud

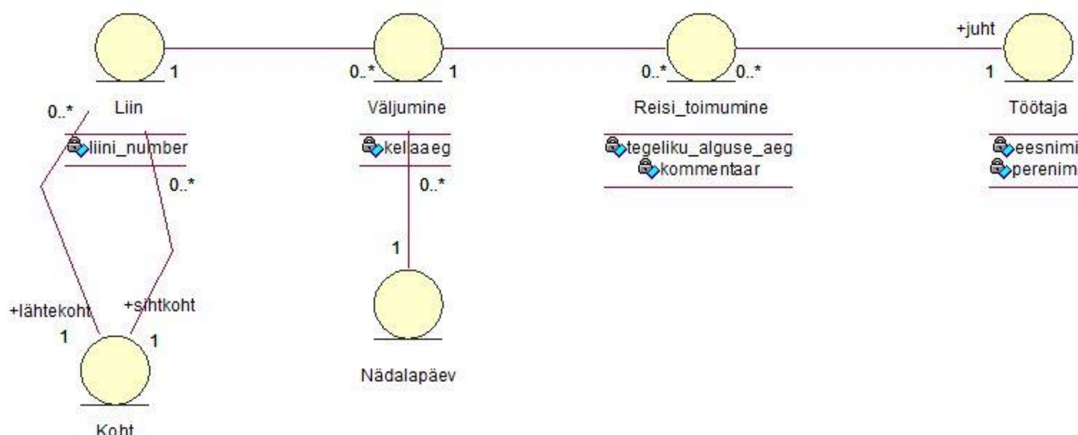
andmebaasisüsteemiga tuttav. PostgreSQL on objekt-relatsiooniline tasuta pakutav avatud lähtekoodiga andmebaasisüsteem. PostgreSQL täidab enamuse SQL:2011 standardi tuuma nõuetest, ning on 2016.aasta mai seisuga andmebaasisüsteemide populaarsuse indeksis viiendal kohal [1].

Eksperimendi käigus püütakse leida vastuseid järgmistele küsimustele.

- Kuidas erinevad päringute ja lisamisoperatsioonide kiirused erinevate disainide ja sama andmete sisu puhul?
- Millised on päringute ja lisamisoperatsioonide keerukused erinevate disainide puhul?
- Milline andmebaasi disain on vaadeldavas kontekstis kõige ebasobilikum?
- Milline andmebaasi disain on vaadeldavas kontekstis kõige sobivam?

3.3 Eksperimendi andmebaasi valdkonna kirjeldus

Esimese andmebaasi projekteerimise sammuna luuakse kontseptuaalne andmemudel linnadevahelisele transpordile ühe hüpoteetilise ettevõte vaates. Selle esitamiseks kasutatakse olemi-suhte diagrammi (vt Joonis 1), mille loomiseks on kasutatud CASE vahendit Rational Rose.



Joonis 1. Linnadevahelise transpordi olemi-suhte diagramm.

Reaalses elus oleks sellise ettevõtte andmebaasi olemi-suhte diagrammis palju rohkem olemitüüpe, kuid käsolevas töös ei oma need tähtsust ning selle pärast on need jäetud vaatluse alt välja. Käesolev kontseptuaalne andmemudel on vaadeldav ühe suure süsteemi fragmendina ning on sobiv erinevatesse infosüsteemidesse.

Joonis 1 kohaselt on iga linn *Koht*. Võttes näitena Eesti riigi, on igal linnal oma unikaalne kood [15]. Seega, hoitakse olemitüübis *Koht* kohta andmebaasis linna *koodi* ja *nimetust*.

Olemitüübis *Liin* kohta hoitakse andmebaasis *liini numbrit* ning *lähtekohta* ja *sihtkohta*. Igal liinil saab olla ainult üks lähtekoht ning ainult üks sihtkoht.

Iga väljumine toimub mingil konkreetsel liinil, kindlal kellaajal ja nädalapäeval. Samal päeval saab liinil olla mitu väljumist. Olemitüübi *Väljumine* kohta hoitakse andmebaasis *väljumise järjekorranumbrit*, *liini*, *nädalapäeva* ja *kellaaega*.

Igal väljumisel saab olla mitu reisi toimumist, mis erinevad tegeliku alguse aja ja/või juhi poolest. Igal reisi toimumisel on täpselt üks juht. Olemitüübi *Reisi_toimumine* kohta hoitakse andmebaasis *väljumise koodi*, *väljumise tegeliku alguse aega*, *töötajat kes transporti juhtis* ning *kommentaari*.

3.4 Eksperimendi kirjeldus

Eksperimendi käigus luuakse andmebaasisüsteemis PostgreSQL (9.4.4) ühe kontseptuaalmudeli alusel (vt Joonis 1) viis erinevat andmebaasi (vt Lisa 1, Lisa 2, Lisa 3, Lisa 4, Lisa 5). Füüsiliselt realiseeritakse see kui üks andmebaas nimega *linnadevaheline_transport*, kus iga andmebaasi disaini jaoks luuakse eraldi skeem. Skeemide nimedeks on *schema_1*, *schema_2*, *schema_3*, *schema_4*, *schema_5*.

Erinevate andmebaasi disainide võrdlemiseks mõõdetakse päringute täitmise ja ridade lisamise kiirust. Päringute ja lisamisoperatsioonide keerukust mõõdetakse koodiridade arvu metoodika alusel. Järgnevalt tutvustatakse lähemalt mõõtmisviise.

Töökiiruse mõõtmiseks kasutatakse PostgreSQL andmebaasisüsteemi käsku *EXPLAIN ANALYZE.* Koguaja leidmiseks liidetakse planeerimis- ja täitmiseaega. Planeerimise aeg on aeg, mida andmebaasisüsteem kulutab lause täitmisplaani koostamiseks. Täitmisaeg on aeg, mida andmebaasisüsteem kulutab täitmisplaani täitmiseks. Päringuid ja lisamisoperatsioone käivitatakse kolm korda ning arvesse võetakse nende täitmise aja aritmeetiline keskmine väärtus.

Päringute ja lisamisoperatsioonide täitmisplaanide detailse info paremaks vaatamiseks kasutatakse veebirakendust *https://explain.depesz.com/*, kuhu impoditakse päringute ja lisamisoperatsioonide täitmisplaanid ja tulemusena saadakse visuaalselt korrastatud täitmisplaani kirjeldus.

Päringute ja operatsioonide keerukuse mõõtmiseks kasutatakse koodiridade arvu metoodikat (ingl *Source lines of Code SLOC*). Koodiridade arvu metoodikat kasutatakse tavaliselt tarkvara loomiseks nõutava pingutuse ennustamiseks ning hooldatavuse hindamiseks kui tarkvara on juba loodud. See metoodika kirjeldab tarkvara mõõdiku, mille peamine kasutusala on tarkvara lähtekoodi suuruse ja keerukuse hindamine. Hindamine toimub valitud tarkvara lähtekoodi koodiridade arvu alusel [16]. Eksperimendi käigus kasutatakse *LocMetrics* tarkvara, mis leiab koodiridade arvu. Uuritakse ainult *füüsiliselt käivitatavaid ridu (füüsiliste koodiridade arv)*, see tähendab, et tühje ridu ja kommentaare ei arvestata. Selleks, et tulemused oleksid usaldusväärsed ja realistlikud tuleb lähtekood vormistada ühesuguseid põhimõtteid arvestades. Antud juhul tuleb kõigi lausete puhul teha järgmist.

- Reserveeritud sõnu (SELECT, WITH) tuleb kirjutada suurtähtedega
- Põhilised võtmesõnad (SELECT, FROM, WITH, ORDER BY, GROUP BY) tuleb kirjutada uuele reale
- Iga veeru nimetus SELECT klauslis tuleb kirjutada uuele reale
- Iga tabeli nimetus FROM klauslis tuleb kirjutada uuele reale
- Iga WHERE alamtingmist tuleb kirjutada uuele reale

3.4.1 Kasutatavad andmebaasisüsteemid

Eksperimendid viiakse läbi andmebaasisüsteemis PostgreSQL (9.4.4). Eksperimendid viiakse läbi Tallinna Tehnikaülikooli serveris *apex.ttu.ee*. Serveri tehnilised andmed: virtuaalmasin QEMU Virtual CPU version, 811 GB HDD, 40 GB RAM, 15 virtuaalset CPU'd, CentOS 6.4. Päringute ja lisamisoperatsioonide käivitamiseks, sellele kulunud aja mõõtmiseks ja täitmisplaanide uurimiseks kasutatakse Tallinna Tehinkaülikooli serveris *apex.ttu.ee* asuvat kasutajaliidest phpPgAdmin.

3.5 Eksperimendi andmebaaside disainimine

Järgnevalt on esitatud PostgreSQL abil realiseerivaid SQL-andmebaasi disaine kirjeldavad füüsilise disaini diagrammid. Autor loob ja kirjeldab ainult neid kitsendusi, mida saab esitada deklaratiivselt kasutades PRIMARY KEY, UNIQUE, FOREIGN KEY, NOT NULL ja CHECK kitsendusi. PostgreSQL lubab kasutada CHECK kitsendustes kasutaja-defineeritud funktsioone. CHECK kitsendus on selline kitsendus, mille abil kontrollitakse andmebaasis olevate andmete vastavust mingile reeglile [17]. PostgreSQL andmebaasisüsteemis ei ole CHECK kitsenduses alampäringute kasutamine lubatud.

Primaarvõtme ja unikaalsuse kitsenduste alusel loob PostgreSQL andmebaasisüsteem automaatselt indeksid. PostgreSQL andmebaasisüsteemis ei indekseerita välisvõtmeid automaatselt. Sellest lähtudes luuakse välisvõtmetele indekseid ühendamisoperatsioonide kiirendamiseks ja andmete lukustamise vähendamiseks. Indeksite loomisel tuleb jälgida, et kasutaja loodavad indeksid ei dubleeriks PRIMARY KEY ja UNIQUE kitsenduste põhjal automaatselt loodavad indeksid.

Järgnevalt on esitatud CHECK kitsendused, mis on loodud iga käesolevas eksperimendis kasutatud andmebaasi disaini puhul.

Tabelis *Koht* luuakse CHECK kitsendus, millega kontrollitakse, et koha kood oleks suurem nullist *(CHECK(koht_kood >0)).*

Tabelis *Liin* luuakse CHECK kitsendus, millega kontrollitakse, et liini number oleks suurem nullist *(CHECK(liin_number >0))* ning kitsendus, millega kontrollitakse, et lähtekoht ja sihtkoht ei oleks samad *(CHECK(sihtkoht<>lahtekoht)).*

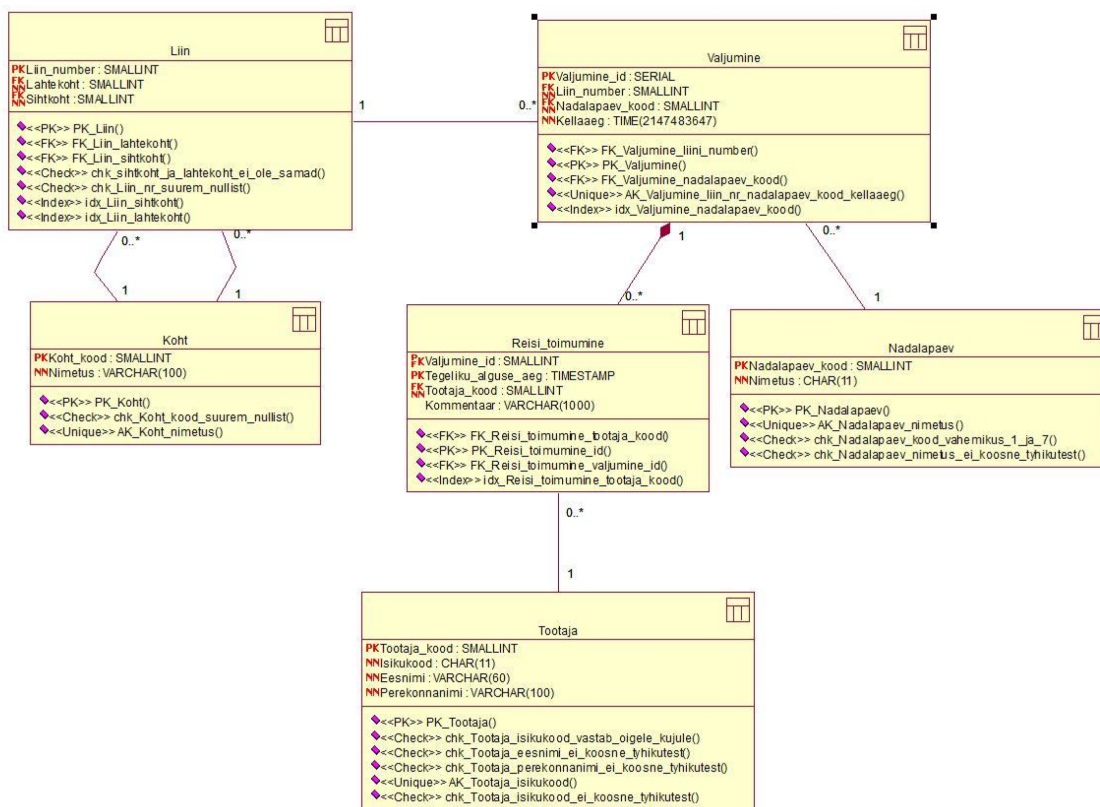
Tabelis *Nadalapaev* luuakse CHECK kitsendus, millega kontrollitakse, et nädalapäeva kood oleks suurem nullist ja väiksem seitsmest *(CHECK(nadalapaev_kood >=1 AND nadalapaev_kood <= 7)).* Samuti, luuakse CHECK kitsendus, millega kontrollitakse, et nädalapäeva nimetus ei koosneks tühikutest *(CHECK(nimetus !~ '^[[:space:]]*$')).*

Tabelis *Tootaja* luuakse mitu CHECK kitsendust, millega kontrollitakse, et väärtused ei koosneks tühikutest *(CHECK(eesnimi !~ '^[[:space:]]*$'))*, *(CHECK(perekonnanimi !~ '^[[:space:]]*$'))*, *(CHECK(isikukood !~ '^[[:space:]]*$')).* Täiendavalt luuakse CHECK kitsendus, millega kontrollitakse, et töötaja isikukood vastaks Eesti isikukoodi

nõuetele *(CHECK(isikukood ~ '^([3-6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-3]{1}[[:digit:]]{5})$')).*

3.5.1 Andmebaasi disaini mudel „traditsioonilise“ disaini kasutamise korral

Joonisel 2 on kujutatud „traditsioonilise“ disaini andmebaasi diagramm. Tabelite arv on 6, veergude arv on 19 ning välisvõtmete arv on 6.



Joonis 2. „Traditsiooniline“ andmebaasi disain

Täiendavad indeksid luuakse *Valjumine* tabeli välisvõtme veergudele *nadalapaev_kood*, *Reisi_toimumine* tabeli välisvõtme veerule *tootaja_kood, Liin* tabeli välisvõtme veergudele *lahtekoht* ja *sihtkoht*. Tabelis *Valjumine* luuakse unikaalne kitsendus *liin_number*, *nadalapaev_kood* ja *kellaaeg* veergude kombinatsioonile.

Välisvõtme määrangut *ON DELETE CASCADE* kasutatakse järgmiste välisvõtmete puhul.

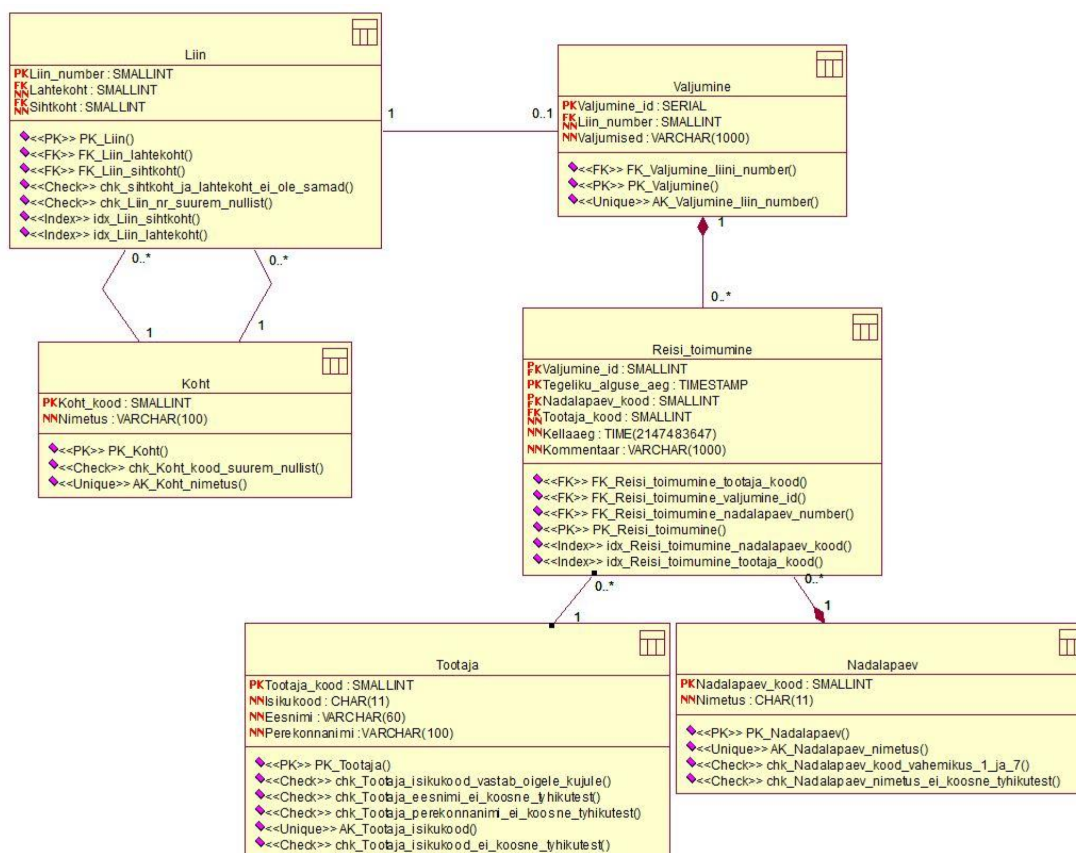
- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*
- Tabel *Valjumine* välisvõti *liin_number*.
- Tabel *Valjumine* välisvõti *nadalapaev_kood.*

Välisvõtme määrangut *ON UPDATE CASCADE* kasutatakse järgmiste välisvõtmete puhul.

- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*
- Tabel *Reisi_toimumine* välisvõti *tootaja_kood.*
- Tabel *Reisi_toimumine* välisvõti *valjumine_id.*
- Tabel *Valjumine* välisvõti *liin_number*.
- Tabel *Valjumine* välisvõti *nadalapaev_kood.*

3.5.2 Andmebaasi disaini mudel vektorkodeerimise kasutamise korral

Joonisel 3 on kujutatud vektorkodeerimisega disaini andmebaasi diagramm. Tabelite arv on 6, veergude arv 20 ja välisvõtmete arv on 6.



Joonis 3. Vektorkodeerimisega andmebaasi disain.

Täiendavad indeksid luuakse *Reisi_toimumine* tabeli välisvõtme veergudele *tootaja_kood* ja *nadalapaev_kood, Liin* tabeli välisvõtme veergudele *lahtekoht* ja *sihtkoht*.

Erinevalt „traditsioonilisest“ andmebaasi disainist hoitakse käesolevas andmebaasi disainis kellaajad ja nädalapäeva koodid ühes VARCHAR tüüpi veerus. Kodeeritakse nädalapäeva koodi järgi: esialgu tuleb nädalapäeva kood, järgnevalt tulevad kriips, kellaaeg, semikoolon (*„1-10:00:00;1-12:00:00;4-16:00:00“*).

Välisvõtme määrangut *ON DELETE CASCADE* kasutatakse järgmiste välisvõtmete puhul.

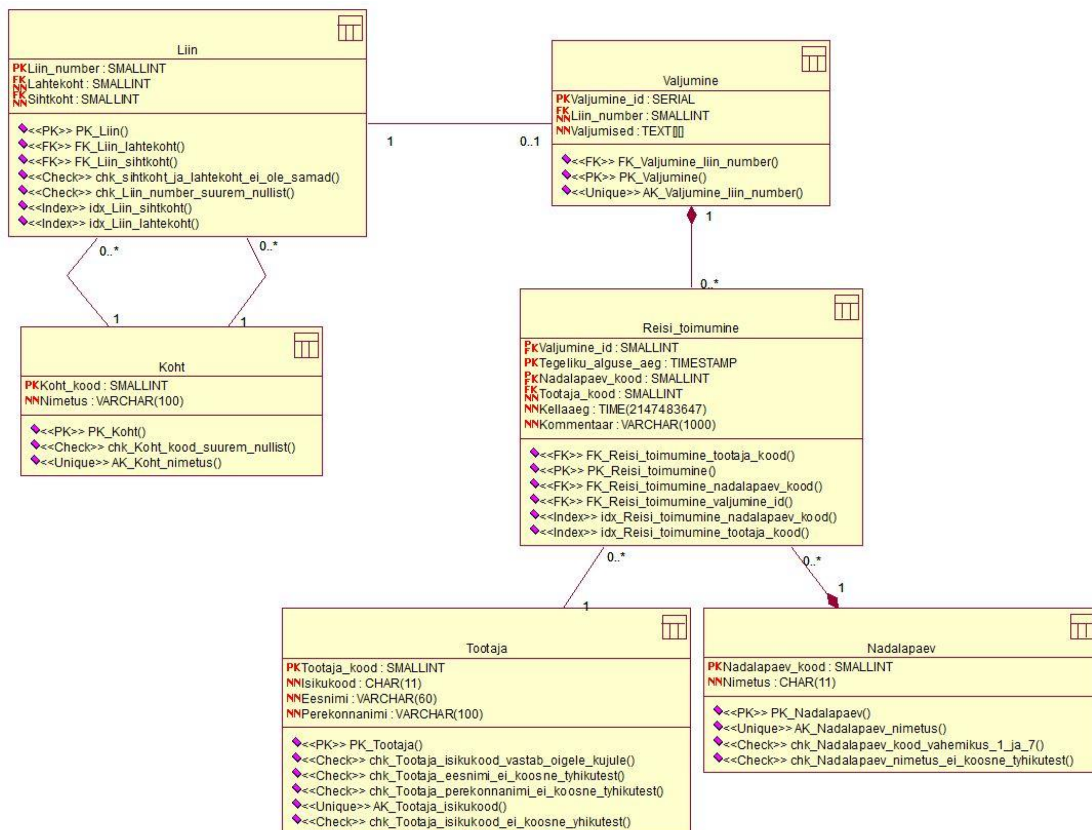
- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*
- Tabel *Valjumine* välisvõti *liin_number*.

Välisvõtme määrangut *ON UPDATE CASCADE* kasutatakse järgmise välisvõtmete puhul.

- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*
- Tabel *Reisi_toimumine* välisvõti *nadalapaev_kood*.
- Tabel *Reisi_toimumine* välisvõti *tootaja_kood.*
- Tabel *Reisi_toimumine* välisvõti *valjumine_id.*
- Tabel *Valjumine* välisvõti *liin_number*.

3.5.3 Andmebaasi disaini mudel kahemõõtmelise massiivi kasutamise korral

Joonisel 4 on kujutatud kahemõõtmelise massiiviga disaini andmebaasi diagramm. Tabelite arv on 6, veergude arv 20 ja välisvõtmete arv on 6.



Joonis 4. Kahemõõtmelise massiiviga andmebaasi disain.

Täiendavad indeksid luuakse *Reisi_toimumine* tabeli välisvõtme veergudele *tootaja_kood* ja *nadalapaev_kood, Liin* tabeli välisvõtme veergudele *lahtekoht* ja *sihtkoht*.

Nädalapäeva koodid ja nendele vastavad väljumiste kellaajad hoitakse kahemõõtmelise massiivi tüüpi veerus. Esimeses massiivi dimensioonis hoitakse nädalapäeva koode, teises dimensioonis vastaval nädalapäeva koodile positsioonis hoitakse kellaaegu (*{{'1','1','4'},{'10:00:00','12:00:00','16:00:00'}}*). Massiivi tüübiks on *text*.

Välisvõtme määrangut *ON DELETE CASCADE* kasutatakse järgmiste välisvõtmete puhul.

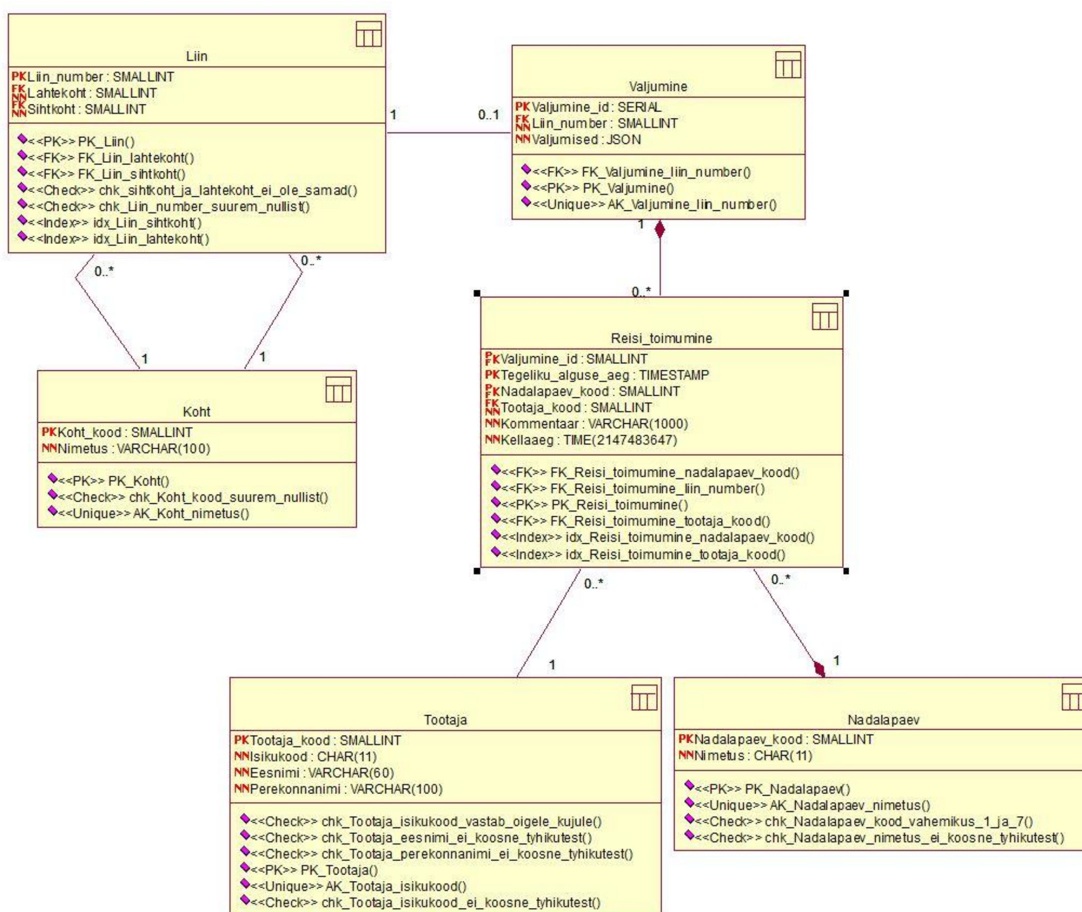
- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*
- Tabel *Valjumine* välisvõti *liin_number*.

Välisvõtme määrangut *ON UPDATE CASCADE* kasutatakse järgmise välisvõtmete puhul.

- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*
- Tabel *Reisi_toimumine* välisvõti *nadalapaev_kood*.
- Tabel *Reisi_toimumine* välisvõti *tootaja_kood.*
- Tabel *Reisi_toimumine* välisvõti *valjumine_id.*
- Tabel *Valjumine* välisvõti *liin_number*.

3.5.4 Andmebaasi disaini mudel JSON tüüpi kasutamise korral

Joonisel 5 on kujutatud JSON tüüpi veeruga disaini andmebaasi diagramm. Tabelite arv on 6, veergude arv 20 ja välisvõtmete arv on 6.



Joonis 5. JSON tüüpi veeruga andmebaasi disain.

Täiendavad indeksid luuakse *Reisi_toimumine* tabeli välisvõtme veergudele *tootaja_kood* ja *nadalapaev_kood, Liin* tabeli välisvõtme veergudele *lahtekoht* ja *sihtkoht*.

Väljumised on esitatud JSON tüüpi veerus, objektina. Objekti võtmeteks on nädalapäeva koodid, nendele vastavateks väärtusteks on kellaajad ({„1“:“10:00:00“, „1“:“12:00:00“, „4“:“16:00:00“})

Välisvõtme määrangut *ON DELETE CASCADE* kasutatakse järgmiste välisvõtmete puhul.

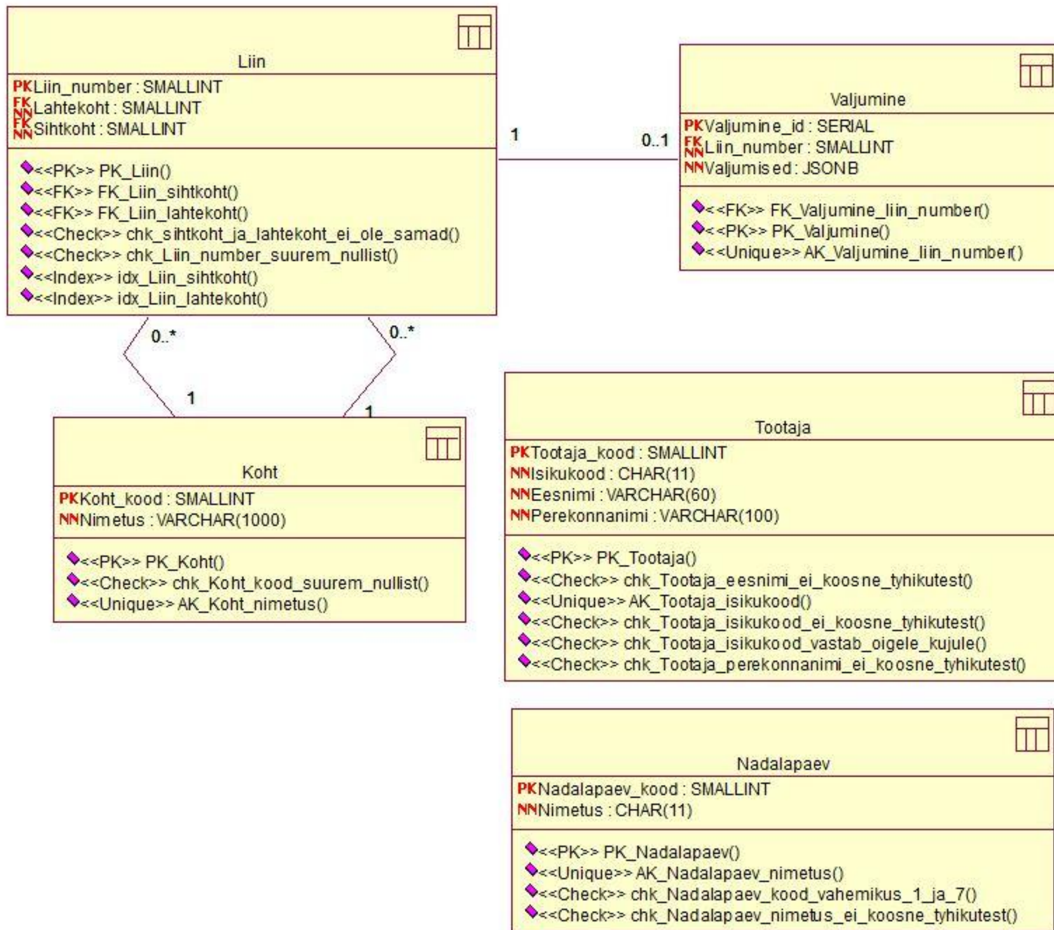
- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*
- Tabel *Valjumine* välisvõti *liin_number*.

Välisvõtme määrangut *ON UPDATE CASCADE* kasutatakse järgmise välisvõtmete puhul.

- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*
- Tabel *Reisi_toimumine* välisvõti *nadalapaev_kood*.
- Tabel *Reisi_toimumine* välisvõti *tootaja_kood.*
- Tabel *Reisi_toimumine* välisvõti *valjumine_id.*
- Tabel *Valjumine* välisvõti *liin_number*.

3.5.5 Andmebaasi disaini mudel hierarhilise JSONB tüüpi kasutamise korral

Joonisel 6 on kujutatud hierarhilise JSONB tüüpi veeruga disaini andmebaasi diagramm. Tabelite arv on 5, veergude arv 14 ja välisvõtmete arv on 3.



Joonis 6. Hierarhilise JSONB tüüpi veeruga andmebaasi disain.

Täiendavad indeksid luuakse *Liin* tabeli välisvõtme veergudele *lahtekoht* ja *sihtkoht*.

Väljumiste veerus hoitakse massiivi objektidest. Igal objektil on võti unikaalse nädalapäeva koodiga. Ühel nädalapäeval saab olla mitu väljumist erineval kellaajal, seega iga objekti *kellaajad* võtme väärtuseks on objektide massiiv, kus iga massiivi *kellaaja* võtme väärtuseks on unikaalne väärtus. Lisaks, hoitakse objekti sees toimunud väljumised. Selliselt tekib JSONB tüüpi veerus hierarhia (vt Joonis 7).

```
[
   {
      "kellaajad":[
         {
            "kellaaeg":"10:00:00",
            "toimunud_valjumised":[
               {
                  "kommentaar":"Kommentaar1",
                  "tootaja_kood":358,
                  "tegeliku_alguse_aeg":"2015-01-01 02:00:00"
               }
            ]
         }
      ],
      "nadalapaeva_kood":"1"
   }
]
```

Joonis 7. Hierarhilised andmed JSONB tüüpi veerus.

Välisvõtme määrangut *ON DELETE CASCADE* kasutatakse järgmiste välisvõtmete puhul.

- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*

Välisvõtme määrangut *ON UPDATE CASCADE* kasutatakse järgmise välisvõtmete puhul.

- Tabel *Liin* välisvõti *lahtekoht*.
- Tabel *Liin* välisvõti *sihtkoht.*
- Tabel *Valjumine* välisvõti *liin_number*.

3.6 Testandmed

Kõik eksperimendi käigus loodud andmebaasid (antud juhul siis tabelite hulgad sama füüsilise andmebaasi eri skeemides) täidetakse ühesuguste testandmetega. Testandmete genereerimiseks kasutatakse ApexSQL ettevõte poolt loodud tarkvara ApexSQL Generate, *trial* versiooni [18]. Tarkvara annab võimaluse luua ühenduse kasutaja määratud serverites asuvate andmebaasidega, mis kergendab tabeli testandmetega täitmist, sest pärast genereerimist täidetakse tabel testandmetega automaatselt. Käesolevas töös kasutatakse selle tarkvara teist võimalust, ehk programmi imporditakse

andmebaasi tabelite loomise lauseid, mille põhjal programm otsustab, mis kujul testandmeid genereerida. Sellel tarkvaral on hulk eeliseid võrreldes teistega veebis olevate testandmete generaatoritega. Programm võimaldab kasutada eeldefineeritud testandmeid nagu ees- ja perekonnanimi, riigi linnad, aadressid ja paljud muud. On võimalik kasutada ise defineeritud regulaaravaldisi, kasutada testandmeid teistest tabelite veergudest ning importida testandmeid CSV-formaadis failina. Ning kõige suuremaks eeliseks on piiramatu andmemahtu loomine, sest teiste testandmete generaatorite puhul muutus testandmete genereerimine tasuliseks pärast teatud ridade hulga genereerimist.

Järgnevalt selgitatakse testandmete genereerimise käiku.

- *Koht*: generaatorisse imporditi csv-failina Eesti linnade nimetused, millele generaator pani vastavusse järjekorra numbrid. Tegelikus elus peaks kasutama Eesti haldus- ja asjajaotuse klassifikaatorit [15], mis määrab linnade koodid. Testandmete genereerimise hõlbustamiseks otsustati neid koode mitte kasutada.
- *Liin*: võeti *koht_kood* väärtused tabelist *Koht* ning kombineeriti neid juhuslikus järjekorras, lisades igale lahtekoht-sihtkoht paarile unikaalse liini numbri.
- *Nadalapaev*: testandmete genereerimine toimus sarnasel tabelile *Koht* viisiga.
- *Tootaja*: kasutati ApexSQL pakutavad eeldefineeritud väljad Eesnimi (mees) ja Perekonnanimi. Lisaks kasutati regulaaravaldist isikukoodi jaoks ning järjekorranumbrite generaatorit töötaja koodi jaoks.
- *Valjumine*: *liin_number* ja *nadalapaev_kood* võeti juhuslikud väärtused vastavate tabelite *Liin* ja *Nadalapaev* veergudest *liin_number* ja *nadalapaev_kood*, *kellaaeg* loodi regulaaravaldisega nii, et kõik väljumised toimuksid täistundidel.
- *Reisi_toimumine*: *kommentaari* jaoks kasutati ApexSQL tekstgeneraatorit, *tootaja_kood* väärtusi võeti juhuslikus järjekorras tabelist *Tootaja* veerust *tootaja_kood* ning *tegeliku_alguse_aeg* loodi regulaaravaldisega.

Testandmeid on võimalik eksportida mitmel viisil – SQL, XML, CSV, JSON ja Excel formaadis. Käsolevas töös eksporditi andmeid CSV formaadis.

Andmete andmebaasi importimiseks lisati „/tmp“ kataloogi testandmete failid ning kopeeriti testandmed „traditsioonilise“ disaini tabelitesse kasutades COPY lauseid (vt Lisa 6).

Kuna teiste andmebaasi disainide puhul teisendati testandmed õigele kujule võttes andmed „traditsioonilise“ disaini tabelitest (vt Lisa 7, Lisa 8, Lisa 9, Lisa 10) ning erinevate disainide puhul on tabelites erinev arv ridu, siis esialgne genereeritav andmete hulk on esitatud Tabelis 1.

Tabel 1. Ridade arv „traditsioonilise“ disaini tabelites.

	Koht	**Liin**	**Nadalapaev**	**Reisi_toimumine**	**Tootaja**	**Valjumine**
Ridade arv	47	2162	7	1155641	1000	6356

Genereeritud andmed on juhusliku pikkuse ja sisuga:

- Koha kood – täisarvud vahemikus 1 kuni 32767 (otspunktid kaasa arvatud)
- Koha nimetus – kuni 100 tähemärki
- Liini number – täisarvud vahemikus 1 kuni 32767 (otspunktid kaasa arvatud)
- Nädalapaeva kood – 1 tähemärk, ainult numbrid, vahemikus 1 kuni 7
- Nädalapäeva nimetus – kuni 11 tähemärki
- Töötaja kood – täisarvud vahemikus 1 kuni 32767 (otspunktid kaasa arvatud)
- Eesnimi inglise keeles – kuni 60 tähemärki, ainult tähed
- Perekonnanimi inglise keeles – kuni 100 tähemärki, ainult tähed
- Kellaaeg – 6 tähemärki, ainult numbrid ja : märk

3.7 Eksperimendi käigus katsetavad operatsioonid

Eksperimendi jaoks on välja valitud neli andmete lugemise päringut ja üks andmete lisamise operatsioon.

Esimese päringu (S1) abil soovitakse leida kõik esmaspäevased väljumised kellaaja vahemikus ’10:00’ – ’12:00’. Päringu tulemusena väljastatakse kõik leitud andmed väljumiste kohta, mis on tabelis *Valjumine*. Tulemused järjestatakse liini numbri järgi kasvavalt. Päringu keerulisus seisneb selles, et tuleb dekodeerida veeru *valjumised* väärtused tabelis *Valjumine* nendes andmebaasi disainides, kus mitu väidet on esitatud ühe andmeväärtusena.

Teise päringu (S2) abil soovitakse leida kõik esmaspäevased väljumised Põlva-Pärnu suunal kellaaja vahemikus ’10:00’-’12:00’. Päringu tulemusena väljastatakse kõik leitud

andmed väljumiste kohta, mis on tabelis *Valjumine*. Tulemused järjestatakse liini numbri järgi. Seda päringut teeb keeruliseks see, et lisaks esimeses päringus tehtud dekodeerimisele tuleb filtreerida andmeid kasutades tabeleid *Liin* ja *Koht*.

Kolmanda päringu (S3) abil soovitakse leida liini number 720 väljumiste arv, mis on toimunud 2015.aasta märtsis. Päringu tulemusena väljastatakse väljumiste arv. Hierarhilise JSONB tüüpi veeru puhul tuleb tegeliku alguse aega teisendada mitu korda: esialgu tekstiks, ja siis TIMESTAMP väärtuseks.

Neljanda päringu (S4) abil soovitakse leida kõik liini 720 reisi toimumised. Päringu tulemusena väljastatakse töötaja number, töötaja perekonnanimi, nädalapäeva nimi, tegeliku alguse aeg ning planeeritud kellaaeg. Päringu keerulisus seisneb selles, et hierarhilise JSONB tüüpi veeru puhul tuleb teha sügavat päringut mitmest tabelist korraga.

Viienda operatsiooniga (I1) lisatakse Tallinn-Pärnu suunale uus väljumine teisipäeval kell 10.00. Hierarhilise JSONB veergu puhul lisatakse uue reisi toimumine hierarhiasse sügavusse, ehk tuleb esialgu leida õige liin number, järnevalt leida kas sellel liinil on teisipäeval väljumine. Kui väljumine olemas, siis lisada uus objekt kellaaegade massiivi. Kui sellisel nädalapäeval väljumist ei ole, tuleb lisada uus objekt selle nädalapäevaga, kuhu kellaaegade massiivi peab lisama uut objekti soovitud kellajaga.

4 Eksperimendi tulemused

Käesolevas peatükis esitatakse päringute ja andmemuudatuse operatsioonide kiiruse mõõtmise tulemused ning lausete keerukuse mõõtmise tulemused, mis on leitud koodiridade arvu metoodika alusel.

Mugavuse mõttes on iga päringu kiiruse mõõtmise tulemuste jaoks tehtud eraldi tabel. Päringute ja andmemuudatuse operatsiooni kiiruse mõõtmise tulemused on esitatud tabelikujul. Veergudeks on operatsioonide identifikaatorid (nt S1 T tähendab operatsiooni S1 „traditsioonilise“ andmebaasi disaini puhul). Ridadeks on keskmine planeerimise aeg, keskmine täitmise aeg ning keskmine aeg kokku.

Tabeli päistes olevad ühetähelised tähised päringu identifikaatori järel on järgmised.

T – „Traditsiooniline“ andmebaasi disain
V – Vektorkodeerimisega andmebaasi disain
M – Kahemõõtmelise massiivi tüüpi veeruga andmebaasi disain
J – JSON tüüpi veeruga andmebaasi disain
H – JSONB tüüpi veeruga andmebaasi disain, kus väärtuses on sügav andmete hierarhia

Järgnevalt (vt Tabel 2) on esitatud päringu S1 kiiruse mõõtmise tulemused millisekundites iga katsetatava andmebaasi disaini puhul. Rasvasega on märgitud tabeli väljad, kus summaarne päringu aeg on kõige suurem ning kõige väiksem.

Tabel 2. Päringu S1 kiiruse mõõtmise tulemused millisekundites.

	S1 T	**S1 V**	**S1 M**	**S1 J**	**S1 H**
Planning time	0,849	1,192	1,486	1,020	0,935
Execution time	1,309	3,861	52,182	31,345	1043,700
Total	**2,158**	5,053	53,668	32,365	**1044,635**

Järgnevalt (vt Tabel 3) on esitatud päringu S2 kiiruse mõõtmise tulemused millisekundites iga katsetatava andmebaasi disaini puhul. Rasvasega on märgitud tabeli väljad, kus summaarne päringu aeg on kõige suurem ning kõige väiksem.

Tabel 3. Päringu S2 kiiruse mõõtmise tulemused millisekundites.

	S2 T	**S2 V**	**S2 M**	**S2 J**	**S2 H**
Planning time	2,228	2,661	3,249	2,514	2,922
Execution time	0,625	1,271	3,743	0,914	3,717
Total	**2,853**	3,932	**6,992**	3,428	6,639

Järgnevalt (vt Tabel 4) on esitatud päringu S3 kiiruse mõõtmise tulemused millisekundites iga katsetava andmebaasi disaini puhul. Rasvasega on märgitud tabeli väljad, kus summaarne päringu aeg on kõige suurem ning kõige väiksem.

Tabel 4. Päringu S3 kiiruse mõõtmise tulemused millisekundites.

	S3 T	**S3 V**	**S3 M**	**S3 J**	**S3 H**
Planning time	2,093	2,310	2,020	2,099	1,139
Execution time	4,644	2,335	2,124	2,387	5,442
Total	**6,737**	4,645	**4,144**	4,486	6,581

Järgnevalt (vt Tabel 5) on esitatud päringu S4 kiiruse mõõtmise tulemused millisekundites iga katsetava andmebaasi disaini puhul. Rasvasega on märgitud tabeli väljad, kus summaarne päringu aeg on kõige suurem ning kõige väiksem.

Tabel 5. Päringu S4 kiiruse mõõtmise tulemused millisekundites.

	S4 T	**S4 V**	**S4 M**	**S4 J**	**S4 H**
Planning time	3,701	3,416	3,437	3,412	2,342
Execution time	7,490	4,9723	5,032	4,734	13,851
Total	11,191	8,388	8,469	**8,146**	**16,193**

Järgnevalt (vt Tabel 6) on esitatud andmemuudatuse operatsiooni I1 kiiruse mõõtmise tulemused millisekundites iga katsetava andmebaasi disaini puhul. Lisaks planeerimis- ja täitmisajale on tabelis esitatud kitsenduste kontrollimiseks kulunud aeg. Tingitud sellest, et vektorkodeerimisega, kahemõõtmelise massiiviga ja JSON tüüpi andmebaasi disainide puhul ei realiseeritud kitsendusi funktsioonide või trigerite abil, ei ole nendel kitsendustele kulunud aega. Rasvasega on märgitud tabeli väljad, kus summaarne päringu aeg on kõige suurem ning kõige väiksem. Siin tuleb mainida, et seoses sellega, et serveris *apex.ttu.ee* on käesoleva töö kirjutamise hetkel (2016.aasta kevadel) PostgreSQL andmebaasisüsteemi mitte kõige viimane versioon (9.4.4), siis ei toeta see

versioon funktsiooni *jsonb_set(target jsonb, path text[], new_value jsonb, [create missing boolean])*. See funktsioon võimaldab uuendada jsonb tüüpi väärtused, edastades funktsioonile algse väärtuse, teed sihtobjektini ja sisestamiseks mõeldud väärtuse. Teist viisi lisamaks JSONB sügavasse hierarhiasse uusi kirjeid ei ole autor leidnud.

Tabel 6. Operatsiooni I1 kiiruse mõõtmise tulemused millisekundites.

	I1 T	**I1 V**	**I1 M**	**I1 J**	**I1 H**
Planning time	1,755	1,498	2,121	2,788	-
Constraints	1,849	-	-	-	-
Execution time	3,544	0,938	3,628	2,349	-
Total	**7,148**	**2,436**	5,749	5,137	-

Järgnevalt (vt Tabel 7) on esitatud lausete keerukuse mõõtmise tulemused. Esitatakse füüsiliselt käivitavate ridade arv iga operatsiooni jaoks. Veergudena on esitatud päringute ja operatsioonide nimetused, ridadena andmebaasi disainid.

Tabel 7. Füüsiliste koodiridade arv erinevate disainide korral.

	S1	**S2**	**S3**	**S4**	**I1**
T	**5**	**15**	**11**	**14**	**13**
V	13	23	**11**	**14**	**13**
M	16	26	**11**	**14**	17
J	11	21	**11**	**14**	**13**
H	15	25	13	21	-

Tabelis 7 on rasvaselt tähistatud kõige paremad ehk kõige väiksemad tulemused. See tõstab esile disainide paremuse.

5 Tulemuste analüüs

Käesolevas peatükis esitatakse saadud tulemuste analüüs ja tehakse tulemuste põhjal järeldused. Järgnevad jaotised kujutavad endast ülesande püstituses (vt Jaotis 3.2) esitatud küsimusi ning nendele analüüsi käigus saadud vastuseid.

5.1 Päringute ja andmemuudatuste operatsioonide kiirused

Kuidas erinevad päringute ja lisamisoperatsioonide kiirused erinevate disainide ja sama andmete sisu puhul? (vt Tabel 2, Tabel 3, Tabel 4, Tabel 5, Tabel 6)

Esimese päringu (S1) puhul on kiiruste erinevus kõige suurem. Nendest arusaamiseks tuleb uurida päringute täitmisplaane. Täitmisplaan kirjeldab, mis algoritmi kasutab andmebaasisüsteem päringu täitmiseks. Kasutaja deklareerib oma soovi SQL-lause näol ning selle soovi alusel koostab andmebaasisüsteem soovi täitmiseks parima võimaliku algoritmi arvestades samal ajal andmebaasis olevate andmetega.

Täitmisplaanide esitamiseks valis autor kõige kiirema ning kõige aeglasema SQL-lause ülesande S1 puhul. Need laused on vastavalt S1 T ja S1 H.

Järgnevalt (vt Joonis 8) esitatakse S1 T lause, mis täideti kõige kiiremini.

```
SELECT *
FROM schema_1.valjumine
WHERE nadalapaev_kood = 1
  AND kellaaeg BETWEEN '10:00:00' AND '12:00:00'
ORDER BY liin_number;
```

Joonis 8. Päringu S1 T SQL-lause.

Järgnevalt (vt Joonis 9) esitatakse S1 T lause täitmisplaan PostgreSQL andmebaasisüsteemis.

#	exclusive	inclusive	rows x	rows	loops	node
1.	0.095	1.000	↑ 1.1	14	1	→ Sort (cost=74.15..74.19 rows=15 width=16) (actual time=0.996..1.000 rows=14 loops=1) Sort Key: liin_number Sort Method: quicksort Memory: 25kB
2.	0.757	0.905	↑ 1.1	14	1	→ Bitmap Heap Scan on valjumine (cost=23.06..73.86 rows=15 width=16) (actual time=0.268..0.905 rows=14 loops=1) Recheck Cond: (nadalapaev_kood = 1) Filter: ((kellaaeg >= '10:00:00'::time without time zone) AND (kellaaeg <= '12:00:00'::time without time zone)) Rows Removed by Filter: 889 Heap Blocks: exact=35
3.	0.148	0.148	↑ 1.0	903	1	→ Bitmap Index Scan on idx_valjumine_nadalapaev_kood (cost=0.00..23.05 rows=903 width=0) (actual time=0.148..0.148 rows=903 loops=1) Index Cond: (nadalapaev_kood = 1)

Joonis 9. Päringu S1 T täitmisplaan PostgreSQL andmebaasisüsteemis.

Päringu täitmisplaani vaadates on näha, et kõige rohkem aega on kulunud andmete sorteerimiseks, kõik ülejäänud funktsionaalsus on mõistliku kiirusega ning ei ole nii aeganõudev.

Järgnevalt (vt Joonis 10) on esitatud S1 H lause, mis täideti kõige aeglasemalt. Selle lause koostamine võttis kõige rohkem aega võrreldes teiste eksperimendi käigus loodud SQL-lausetega.

```
SELECT *
FROM
  (SELECT query_2.valjumine_id,
          query_2.liin_number,
          jsonb_array_elements(query_2.valjumised->'kellaajad') AS kellaajad
   FROM
     (SELECT *
      FROM
        (SELECT valjumine_id,
                liin_number,
                jsonb_array_elements(valjumised) AS valjumised
         FROM schema_5.valjumine) AS query_1
      WHERE query_1.valjumised->>'nadalapaev_kood'='1') AS query_2) AS
query_3
WHERE (query_3.kellaajad->>'kellaaeg')::time BETWEEN '10:00:00' AND
'12:00:00'
ORDER BY query_3.liin_number;
```

Joonis 10. Päringu S1 H SQL-lause.

Järgnevalt (vt Joonis 11) esitatakse S1 H lause täitmisplaan PostgreSQL andmebaasisüsteemis.

#	exclusive	inclusive	rows x	rows	loops	node
1.	0.204	1,092.197	↑ 36.9	14	1	→ Sort (cost=7,792.73..7,794.02 rows=516 width=38) (actual time=1,092.197..1,092.197 rows=14 loops=1) Sort Key: query_3.liin_number Sort Method: quicksort Memory: 698kB
2.	6.092	1,091.993	↑ 36.9	14	1	→ Subquery Scan on query_3 (cost=0.00..7,769.48 rows=516 width=38) (actual time=207.279..1,091.993 rows=14 loops=1) Filter: ((((query_3.kellaajad ->> 'kellaaeg'::text))::time without time zone >= '10:00:00'::time without time zone) AND (((query_3.kellaajad ->> 'kellaaeg'::text))::time without time zone <= '12:00:00'::time without time zone)) Rows Removed by Filter: 889
3.	62.467	1,085.901	↑ 114.3	903	1	→ Subquery Scan on query_1 (cost=0.00..4,673.48 rows=103,200 width=38) (actual time=1.691..1,085.901 rows=903 loops=1) Filter: ((query_1.valjumised ->> 'nadalapaev_kood'::text) = '1'::text) Rows Removed by Filter: 4546
4.	1,023.434	1,023.434	↑ 38.9	5,312	1	→ Seq Scan on valjumine (cost=0.00..1,061.48 rows=206,400 width=24) (actual time=0.695..1,023.434 rows=5,312 loops=1)

Joonis 11. Päringu S1 H täitmisplaan PostgreSQL andmebaasisüsteemis.

S1 H täitmisplaani vaadates on näha, et ei kasutata indekseid ning alampäringuid ja põhipäringuid ei sulatata kokku, vaid täidetakse eraldi ja pannakse siis tulemused kokku. Lisaks sellele on info väljumiste kohta salvestatud suurtes väärtustes, mille läbitöötamine võtab aega. Kokkuvõttes võtab algoritmi kõigi sammude täitmine palju aega.

Teise päringu (S2) puhul on kõige kiiremaks päringuks osutunud S2 T ehk päring „traditsioonilise“ andmebaasi disaini tabeli põhjal. Uurides selle päringu täitmisplaani on ilmne, et otsingul on kasutatud indekseid, mis oluliselt kiirendab otsinguprotsessi. Teiste andmebaasi disainide puhul on päringute kiirused suuremad, kusjuures kõige aeglasem päring oli kahemõõtmelise massiivi tüüpi andmebaasi disaini puhul. See lause oli ka füüsiliste koodiridade arvu mõttes kõige mahukam.

Kolmanda päringu (S3) puhul on osutunud kiireimaks S3 M ehk päring kahemõõtmelise massiivi tüüpi veeruga tabeli põhjal. Kõige aeglasemaks on aga osutunud S3 T päring ehk „traditsioonilise“ disaini tabeli põhjal. S3 T, S3 V, S3 J laused on olnud peaaegu samad (vt Lisa 13). Ainuke erinevus on skeemides. Seega, vaadates päringu S3 kiiruste tulemuste tabelit (vt Tabel 4), siis võrdsete tingimuste juures võib teha järelduse, et päringute täitmise kiirus sõltub päringu käivitamise hetkel andmebaasisüsteemi poolt valitud täitmisplaanist.

Neljanda päringu (S4) puhul on kõige kiiremaks osutunud S4 J ehk päring JSON tüüpi veeruga tabeli põhjal. Sarnaselt päringu S3 lausetele on päringu S4 laused S4 T, S4 V, S4 M ja S4 J peaaegu samad. Erinevus on skeemi nimedes ning S4 T puhul loetakse nädalapäeva koodi tabelist *Valjumine*, kusjuures teiste lausete puhul loetakse seda tabelist *Reisi_toimumine*. Kõige aeglasem lause S4 puhul on S4 H, mis oli ka füüsiliste koodiridade arvu mõttes kõige mahukam.

Andmete lisamise operatsiooni puhul selgus, et kõige kiiremini täideti I1 V. Selle disaini puhul lisati uus väljumine olemasoleva väljumise lõppu ning kontrolli, mis ei lubaks lisada olemasolevat väljumist, läbi ei viidud. Sellisel viisil tekib andmete liiasus. Antud töös ei realiseeritud andmete unikaalsuse kontrolli andmete lisamisel disainidel I1 V, I1 M, I1 J, kuid võib eeldada, et sellise kontrolli jõustamine suurendaks lisamise aega. Kõige aeglasemalt täideti I1 T. Selle disaini puhul on tabelitel kõige täpsem oma „teema“ ning andmete liiasust on võimalikult palju vähendatud. Andmete lisamisel kulus samuti aega kitsenduste kontrolliks.

Hierarhilise JSON tüüpi veeruga andmebaasi disaini kohta ei saa järeldusi teha, kuna autor ei leidnud võimalust sisestada andmeid sügavasse JSONB hierarhiasse töö tegemise ajal *apex.ttu.ee* serveris olnud PostgreSQL versiooni (9.4.4) korral.

5.2 Koodi keerukus

Käesolevas eksperimendis hinnatakse koodi keerukust katsetavate päringute ja operatsioonide lausete põhjal. Kuna ühte ja sama ülesannet lahendavat päringut saab SQLis kirjutada mitut moodi, siis antud järeldused kehtivad ainult käesolevas eksperimendis katsetavate päringute ja operatsioonide lausete jaoks.

Järgnevalt leitakse vastus küsimusele: *millised on päringute ja operatsioonide keerukused erinevate disainide puhul?*(vt Jaotis 3.2).

Koodiridade arvu metoodika järgi on kõige lihtsamad päringud S1 ja S3, kusjuures päringu S1 puhul oli kõige lihtsam „traditsiooniline“ andmebaasi disaini SQL-lause, S3 puhul aga kõik andmebaasi disainide SQL-laused peale hierarhilise JSONB tüüpi veeruga andmebaasi disaini. Autor peab mainima, et päringu S3 vektorkodeerimisega, kahemõõtmelise massiiviga ja JSON tüüpi veeruga andmebaasi disainide SQL-laused olid sisuliselt täpselt samad (vt Lisa 13) erinedes ainult tabelite skeemi nime poolest.

Kõige keerulisem päring on üldiselt S2, kusjuures kõige mahukam oli S2 M SQL-lause ehk kahemõõtmelise massiiviga andmebaasi disaini lause. Lisaks suurele mahule, on selles lauses kasutatud ka kasutaja-defineeritud funktsiooni (vt Lisa 12, Lisa 16). Ilma selle funktsioonita ei ole selle päringu lause koostamine võimalik, mis lisab koodile keerukust juurde.

Andmete lisamise operatsiooni puhul kõige keerulisem lause oli I1 M. Selle lause koostamiseks autor pidi kasutama kasutaja-defineeritud funktsiooni (vt Lisa 16), mis jagab kahemõõtmelist massiivi kaheks tabeli reaks, kus iga rida on üks tabeli dimensioon. Edasi dimensioonidele liideti sisetatavad väärtused ja tekitati kahemõõtmeline massiiv uuesti. Kasutaja-defineeritud funktsioonide kasutamine lisab sellele lausele keerukust juurde. Teised andmete lisamise operatsiooni laused on ühesuguse keerukusega. Kusjuures lausete I1 V, I1 M ja I1 J puhul ei kontrollita, et lisatav väljumine ei eksisteeriks juba väljumiste seas. Selliselt tekib andmete liiasus, sest *valjumised* veerus võib olla mitu samasugust väljumist. Andmete liiasust on võimalik kontrollida funktsioonide ja trigerite abil, kuid käesolevas tööd neid ei realiseeritud.

5.3 Järeldused

Vastavalt ülaltoodud analüüsile teeb autor järelduse, et ei eksisteeri ühe ja ainsa parimat disaini. Igal andmebaasi disainil on oma head ja halvad küljed.

Päringu kiiruse kontekstis on ennast hästi näidanud JSONB ja JSON tüüpi veeruga andmebaasi disainid eeldusel, et SQL-lause on õigesti koostatud. Pärast täitmisplaanide uurimist on autor saanud selge pildi sellest, kuidas täidetakse SQL-laused JSON tüüpide puhul ja mis järjekorras andmebaasisüsteem täidab operatsioone. Järgnevalt vaadates S1 H päringut, mis on olnud kõige aeglasem kõikide katsetavate päringute seast, on autor jõudnud järeldusele, et päringu S1 H aeglane täitmine on tingitud sügavast lause struktuurist. Lisaks on autor teinud järelduse, et päringut on võimalik kiirendada elimineerides sügava alampäringute struktuuri. JSON ja JSONB andmetüüpide puhul on rakendatav filtreerimine, mis kiirendab päringute täitmist. Sellest võib teha järeldust, et need andmetüübid on hästi rakendatavad stsenaariumides, kus otsingu kiirus on oluline. Koodi keerukuse poolest on JSONB päringud kõige mahukamad, mis on tingitud ka andmete hierarhilisest struktuurist, kus ühe objekti sees võib olla omakorda massiiv objektidega. JSON ja JSONB tüüpide suureks puuduseks on raskelt läbi viidav andmemuudatuse operatsioon. JSONB tüüpi puhul ei leidnud autor võimalust andmete uuendamiseks hierarhia sügavuses. JSON tüüpi puhul andmete muutmine on võimalik, aga oma puudusega: JSON tüüp lubab mitu ühesugust võtit ühe objekti sees. Selline omapära ei võimalda teha muudatusi ainult ühes võti-väärtus paaris juhul, kui objektis

on mitu sama võtmega paari. Ise küsimus, miks keegi peaks tahtma teha, sest tulemuseks oleks vastuolud andmetes.

Andmemuudatuse operatsiooni lauset on kõige lihtsam kirjutada „traditsioonilise“ andmebaasi disaini puhul. Lisaks sellele, on andmebaasi disaini eeliseks võimalus kontrollida, kas sisestatav reisi toimumise kellaaeg ja nädalapäev eksisteerivad tabelis *Valjumine* ilma lisafunktsioonide kirjutamiseta. Samuti on „traditsioonilise“ andmebaasi disaini päringute laused kõige lihtsamad koodi keerukuse suhtes sest ei nõua lisafunktsioonide deklareerimist ja tihtipeale ka üldse funktsioonide kasutamist.

Kõikide andmebaasi disainide, peale „traditsioonilise“, miinuseks osutus kitsenduste deklareerimise raskus või võimatus veergudele, kus mitu väidet on esitatud ühe andmeväärtusena. Kitsendusi tuleb sageli realiseerida funktsioonide ja trigeritena, kuid see on autori hinnangul aeganõudev ja tülikas. Kahemõõtmelise massiivi puhul kontrollitakse, et dimensioonides oleksid ühte tüüpi (*text* või *int*) andmed ja dimensioonides olevate elementide arv oleks ühesugune, aga mitte rohkem. Lisaks pidi kahemõõtmelise massiivi andmebaasi disaini päringute kirjutamiseks autor ise defineerima abifunktsioone (vt Lisa 16). Näiteks on päringutes S1 M ja S2 M kasutatud funktsiooni, mis leiab väärtuse indeksi massiivis ning funktsiooni, mis jagab kahemõõtmelise massiivi kaheks tabeli reaks, igal real üks massiivi dimensioon. Huvitav fakt on see, et nende funktsioonide koodid on leitavad veebilehelt *https://wiki.postgresql.org/wiki/*, aga kõige viimane PostgreSQL andmebaasisüsteemi versioon (PostgreSQL 9.5) ei paku neid eeldefineeritud funktsioonidena.

Vektorkodeerimisega andmebaasi disaini näitajad olid kõikjal keskmised. Nii päringute kiirus, kui ka koodi keerukus ei olnud võrreldes teistega tunduvalt suuremad, ega tunduvalt väiksemad. Päringute SQL-lausete kirjutamine tundus autorile selle andmebaasi disaini puhul keeruline ja intuitiivselt tundus, et selline andmebaasi disain ei ole vaadeldava konteksti jaoks hea variant, sest lisaks eelnimetatud puudustele (vt Jaotis 2.2) nõuab vektorkodeerimine ka regulaaravaldiste kirjutamise oskust.

Milline on antud juhul kontekst, mille alusel andmebaasi disainide headuse üle otsustada? (vt Jaotis 3.2)

Iga disain peab arvestama olukorraga (kontekstiga), milles seda kasutatakse. Erinevates olukordades võib disaini valik olla erinev, sest oluliseks peetakse erinevaid

kriteeriumeid (nt andmemahud, lausete keerukus, jõudlus, koormustaluvus, skeemi evolutsioneeritavus, puuduvate andmetega toimetulek). Käesolev töö keskendus ainult osadele nendele kriteeriumitele.

Linnadevahelise transpordi süsteemi andmebaasi puhul on oluline märkida, et see nõuab uute väljumiste toimumisel pidevat andmebaasi tabelitesse andmete sisestamist.

Milline andmebaasi disain on autori arvates kõige ebasobilikum? (vt Jaotis 3.2)

Vaadeldavas kontekstis on ilmselgelt kõige ebasobilikum JSONB tüüpi veeruga andmebaasi disain. JSONB tüüpi veeruga andmebaasi disain ei toeta hierarhia sügavusse lihtsat andmete sisestamist, vähemalt PostgreSQL versioonis 9.4.4.

Milline andmebaasi disain on vaadeldavas kontekstis kõige sobivam? (vt Jaotis 3.2)

Analüüsides kõiki käesoleva eksperimendi tulemusi ning katsetatud andmebaasi disainide eeliseid ja puuduseid, teeb autor järelduse, et vaadeldavas kontekstis on kõige sobivam siiski „traditsiooniline“ andmebaasi disain. See disain võimaldab kergesti kirjutada CHECK kitsendusi, ei tekita andmete liiasust, on kergesti hallatav ja suhteliselt lihtne evolutsioneerimiseda.

6 Kokkuvõte

Antud töö eesmärgiks oli leida võimalusi mitme väite ühe andmeväärtusena esitamiseks SQL-andmebaasides, disainida leitud võimaluste alusel mitu andmebaasi ning uurida iga andmebaasi disaini eeliseid ja puuduseid. Töö mahtu arvestades sooviti seda teha ühes andmebaasisüsteemis, milleks valiti tasuta, avatud lähtekoodiga ja populaarne PostgreSQL (9.4.4).

Käesoleva töö oluliseks tulemuseks on viis andmebaasi disaini, mille põhjal on loodud viis skeemi, andmebaaside põhjal tehtud päringute ja muudatusoperatsioonide näited ning andmebaasi disainide võrdlemise tulemusena tehtud järeldused.

Kõik töö käigus püstitatud eesmärgid on saavutatud ning kõikidele esitatud küsimustele on vastust leitud. Käesoleva töö käigus on tehtud üks oluline järeldus, et mitme väite ühe andmeväärtusena esitamine on mugav ja rakendatav infosüsteemides, kus ei toimu pidevat andmete uuendamist ja kus olulisemaks teguriks on päringute kiirus. Vastasel juhul on soovitatav kasutada „traditsioonilist“ andmebaasi disaini, kus mitut väidet ühte andmeväärtusesse ei koondata.

Selles töös uuritud andmebaasi disainid ei ole ainsad võimalused mitme väite ühe andmeväärtuse esitamiseks. PostgreSQL andmebaasisüsteem pakub veel andmetüüpe, mida on samuti võimalik selle ülesande lahendamiseks kasutada (XML tüüp). Töö skoobist välja on jäänud vektorkoodeerimise, kahemõõtmelise massiivi, JSON ja JSONB tüüpi veergudele kitsenduste loomine. Samuti, ei ole antud töös realiseeritud andmete liiasusega tabelite vahel andmete kooskõla kontrollid. Autor eeldab, et kitsenduste kirjutamine ja trigerite ning funktsioonide loomine veergudele, kus on mitu väidet ühe andmeväärtusena esitatud, on üks suur teema, mida võib eraldi lõputöös käsitleda.

Kasutatud kirjandus

[1] „DB-engines ranking,“ 05.2016. [Võrgumaterjal]. Available: http://db-engines.com/en/ranking. [Kasutatud 10.05.2016].

[2] S. Mei, „Why You Should Never Use MongoDB,“ Sarah Mei, 11.11.2013. [Võrgumaterjal]. Available: http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/ . [Kasutatud 10.05.2016].

[3] „XML Support,“ The PostgreSQL Global Development Group, 25.08.2010. [Võrgumaterjal]. Available: https://wiki.postgresql.org/wiki/XML_Support. [Kasutatud 10.05.2016].

[4] „XML Type,“ The PostgreSQL Global Development Group , [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/datatype-xml.html. [Kasutatud 10.05.2016].

[5] Regionaalminister, „Isikukoodide moodustamise, väljajagamise ja andmise kord,“ Riigi Teataja, 10.06.2015. [Võrgumaterjal]. Available: https://www.riigiteataja.ee/akt/838530?leiaKehtiv. [Kasutatud 22.05.2016].

[6] D. Nyheter, „Rootsis said isikukoodid otsa,“ Postimees, 07.04.2016. [Võrgumaterjal]. Available: http://maailm.postimees.ee/3647037/rootsis-said-isikukoodid-otsa. [Kasutatud 10.05.2016].

[7] „Arrays,“ The PostgreSQL Global Development Group , [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/arrays.html. [Kasutatud 13.05.2016].

[8] „JSON,“ The Wikimedia Foundation, Inc., 24.08.2015. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/JSON. [Kasutatud 20.05.2016].

[9] „JSON Types,“ The PostgreSQL Global Development Group , [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/datatype-json.html. [Kasutatud 13.05.2016].

[10] D. Robinson, „Using PostgreSQL Arrays The Right Way,“ HEAP, 24.02.2014. [Võrgumaterjal]. Available: http://blog.heapanalytics.com/dont-iterate-over-a-postgres-array-with-a-loop/ . [Kasutatud 10.05.2016].

[11] C. Ringer, „PostgreSQL anti-patterns: Unnecessary json/hstore dynamic columns,“ 2ndQuadrant, 05.05.2015. [Võrgumaterjal]. Available: http://blog.2ndquadrant.com/postgresql-anti-patterns-unnecessary-jsonhstore-dynamic-columns/. [Kasutatud 10.05.2016].

[12] L. Halliday, „Unleash the Power of Storing JSON in Postgres,“ Codeship, 09.07.2015. [Võrgumaterjal]. Available: https://blog.codeship.com/unleash-the-power-of-storing-json-in-postgres/ . [Kasutatud 10.05.2016].

[13] „Is PostgreSQL Your Next JSON Database?,“ Compose, 03.05.2015. [Võrgumaterjal]. Available: https://www.compose.io/articles/is-postgresql-your-next-json-database/ . [Kasutatud 10.05.2016].

[14] D. Maksimov, „MongoDB ja JSON tüüpe kasutava PostgreSQLi jõudluse

võrdlemine. Performance Comparison of MongoDB and PostgreSQL with JSON types.,“ 04.06.2016. [Võrgumaterjal]. Available: http://digi.lib.ttu.ee/i/?2983. [Kasutatud 23.05.2016].

[15] „Eesti haldus- ja asustusjaotuse klassifikaator 2015v1,“ Statistikaamet, [Võrgumaterjal]. Available: http://metaweb.stat.ee/view_xml.htm?id=3985559&siteLanguage=ee. [Kasutatud 13.05.2016].

[16] „Source lines of code,“ Wikimedia Foundation, Inc., 01.05.2016. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Source_lines_of_code. [Kasutatud 10.05.2016].

[17] E. Eessaar, „Andmemuudatused SQLis. SQLi andmekirjelduskeel.,“ 2012. [Võrgumaterjal]. Available: http://maurus.ttu.ee/ained/IDU0220_2012/doc/4/Teema_IDU0220_5_2012.pdf. [Kasutatud 20.05.2016].

[18] „SQL test data generator,“ ApexSQL LLC , 2016. [Võrgumaterjal]. Available: http://www.apexsql.com/sql_tools_generate.aspx. [Kasutatud 13.05.2016].

[19] „PostgreSQL,“ Wikimedia Foundation Inc., 06.05.2016. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/PostgreSQL. [Kasutatud 10.05.2016].

[20] „JSON: The Fat-Free Alternative to XML,“ [Võrgumaterjal]. Available: http://www.json.org/xml.html. [Kasutatud 10.05.2016].

[21] Ł. Lewandowski ja H. Lubaczewski, „PostgreSQL's explain analyze made readable,“ 2015. [Võrgumaterjal]. Available: https://explain.depesz.com/. [Kasutatud 13.05.2016].

[22] E. Eessaar, „TTÜ: Loogiline disain. CASE.,“ 2012. [Võrgumaterjal]. Available: http://maurus.ttu.ee/ained/IDU0220_2012/doc/4/Teema_IDU0220_10_2012_ver2.pdf. [Kasutatud 10.05.2016].

[23] „JSON Functions and Operators,“ The PostgreSQL Global Development Group , [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/functions-json.html. [Kasutatud 13.05.2016].

[24] „Array Functions and Operators,“ The PostgreSQL Global Development Group , [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/functions-array.html. [Kasutatud 13.05.2016].

[25] „ Data Types,“ The PostgreSQL Global Development Group , [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/datatype.html. [Kasutatud 13.05.2016].

[26] „String Functions and Operators,“ The PostgreSQL Global Development Group , [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/functions-string.html. [Kasutatud 13.05.2016].

[27] „Unnest multidimensional array,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://wiki.postgresql.org/wiki/Unnest_multidimensional_array. [Kasutatud 13.05.2016].

[28] „Array Index,“ The PostgreSQL Global Development Group , [Võrgumaterjal]. Available: https://wiki.postgresql.org/wiki/Array_Index. [Kasutatud 13.05.2016].

[29] „LocMetrics,“ locmetrics.com , [Võrgumaterjal]. Available: http://www.locmetrics.com/. [Kasutatud 13.05.2016].

[30] „Date/Time Functions and Operators,“ The PostgreSQL Global Development Group , [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/functions-datetime.html. [Kasutatud 13.05.2016].

Lisa 1 – “Traditsioonilise” andmebaasi loomise laused

```
CREATE SCHEMA schema_1;

CREATE TABLE Reisi_toimumine (
	Valjumine_id SMALLINT NOT NULL,
	Tegeliku_alguse_aeg TIMESTAMP NOT NULL,
	Tootaja_kood SMALLINT NOT NULL,
	Kommentaar VARCHAR ( 1000 ),
	CONSTRAINT PK_Reisi_toimumine_id PRIMARY KEY (Valjumine_id,
Tegeliku_alguse_aeg));

CREATE INDEX idx_Reisi_toimumine_tootaja_kood ON Reisi_toimumine
(Tootaja_kood);

CREATE TABLE Valjumine (
	Valjumine_id SERIAL NOT NULL,
	Liin_number SMALLINT NOT NULL,
	Nadalapaev_kood SMALLINT NOT NULL,
	Kellaaeg TIME ( 2147483647 ) NOT NULL,
	CONSTRAINT AK_Valjumine_liin_nr_nadalapaev_kood_kellaaeg UNIQUE
(Liin_number, Nadalapaev_kood, Kellaaeg),
	CONSTRAINT PK_Valjumine PRIMARY KEY (Valjumine_id));

CREATE INDEX idx_Valjumine_nadalapaev_kood ON Valjumine (Nadalapaev_kood);

CREATE TABLE Tootaja (
	Tootaja_kood SMALLINT NOT NULL,
	Isikukood CHAR ( 11 ) NOT NULL,
	Eesnimi VARCHAR ( 60 ) NOT NULL,
	Perekonnanimi VARCHAR ( 100 ) NOT NULL,
	CONSTRAINT AK_Tootaja_isikukood UNIQUE (Isikukood),
	CONSTRAINT PK_Tootaja PRIMARY KEY (Tootaja_kood),
	CONSTRAINT chk_Tootaja_eesnimi_ei_koosne_tyhikutest CHECK
(Eesnimi!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_isikukood_vastab_oigele_kujule CHECK
(isikukood~'^([3-6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-
3]{1}[[:digit:]]{5})$'),
	CONSTRAINT chk_Tootaja_perekonnanimi_ei_koosne_tyhikutest CHECK
(Perekonnanimi!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_isikukood_ei_koosne_tyhikutest CHECK
(isikukood!~'^[[:space:]]*$'));
```

```
CREATE TABLE Liin (
	Liin_number SMALLINT NOT NULL,
	Lahtekoht SMALLINT NOT NULL,
	Sihtkoht SMALLINT NOT NULL,
	CONSTRAINT PK_Liin PRIMARY KEY (Liin_number),
	CONSTRAINT chk_Liin_nr_suurem_nullist CHECK (Liin_number > 0),
	CONSTRAINT chk_sihtkoht_ja_lahtekoht_ei_ole_samad CHECK (sihtkoht !=
lahtekoht));

CREATE INDEX idx_Liin_sihtkoht ON Liin (Sihtkoht);

CREATE INDEX idx_Liin_lahtekoht ON Liin (Lahtekoht);

CREATE TABLE Koht (
	Koht_kood SMALLINT NOT NULL,
	Nimetus VARCHAR ( 100 ) NOT NULL,
	CONSTRAINT PK_Koht PRIMARY KEY (Koht_kood),
	CONSTRAINT AK_Koht_nimetus UNIQUE (Nimetus),
	CONSTRAINT chk_Koht_kood_suurem_nullist CHECK (Koht_kood > 0));

CREATE TABLE Nadalapaev (
	Nadalapaev_kood SMALLINT NOT NULL,
	Nimetus CHAR ( 11 ) NOT NULL,
	CONSTRAINT AK_Nadalapaev_nimetus UNIQUE (Nimetus),
	CONSTRAINT PK_Nadalapaev PRIMARY KEY (Nadalapaev_kood),
	CONSTRAINT chk_Nadalapaev_kood_vahemikus_1_ja_7 CHECK (Nadalapaev_kood
BETWEEN 1 AND 7),
	CONSTRAINT chk_Nadalapaev_nimetus_ei_koosne_tyhikutest CHECK (Nimetus
!~'^[[:space:]]*$'));

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_valjumine_id
FOREIGN KEY (Valjumine_id) REFERENCES Valjumine (Valjumine_id)  ON DELETE NO
ACTION ON UPDATE CASCADE;

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_tootaja_kood
FOREIGN KEY (Tootaja_kood) REFERENCES Tootaja (Tootaja_kood)  ON DELETE NO
ACTION ON UPDATE CASCADE;

ALTER TABLE Valjumine ADD CONSTRAINT FK_Valjumine_nadalapaev_kood FOREIGN KEY
(Nadalapaev_kood) REFERENCES Nadalapaev (Nadalapaev_kood)  ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE Valjumine ADD CONSTRAINT FK_Valjumine_liini_number FOREIGN KEY
(Liin_number) REFERENCES Liin (Liin_number)  ON DELETE CASCADE ON UPDATE
CASCADE;

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_lahtekoht FOREIGN KEY (Lahtekoht)
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_sihtkoht FOREIGN KEY (Sihtkoht)
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;
```

Lisa 2 - Vektorkodeerimisega andmebaasi loomise laused

```
CREATE SCHEMA schema_2;

CREATE TABLE Nadalapaev (
	Nadalapaev_kood SMALLINT NOT NULL,
	Nimetus CHAR ( 11 ) NOT NULL,
	CONSTRAINT AK_Nadalapaev_nimetus UNIQUE (Nimetus),
	CONSTRAINT PK_Nadalapaev PRIMARY KEY (Nadalapaev_kood),
	CONSTRAINT chk_Nadalapaev_kood_vahemikus_1_ja_7 CHECK (Nadalapaev_kood
BETWEEN 1 AND 7),
	CONSTRAINT chk_Nadalapaev_nimetus_ei_koosne_tyhikutest CHECK (Nimetus
!~'^[[:space:]]*$'));

CREATE TABLE Koht (
	Koht_kood SMALLINT NOT NULL,
	Nimetus VARCHAR ( 100 ) NOT NULL,
	CONSTRAINT PK_Koht PRIMARY KEY (Koht_kood),
	CONSTRAINT AK_Koht_nimetus UNIQUE (Nimetus),
	CONSTRAINT chk_Koht_kood_suurem_nullist CHECK (Koht_kood > 0));

CREATE TABLE Liin (
	Liin_number SMALLINT NOT NULL,
	Lahtekoht SMALLINT NOT NULL,
	Sihtkoht SMALLINT NOT NULL,
	CONSTRAINT PK_Liin PRIMARY KEY (Liin_number),
	CONSTRAINT chk_Liin_nr_suurem_nullist CHECK (Liin_number > 0),
	CONSTRAINT chk_sihtkoht_ja_lahtekoht_ei_ole_samad CHECK (Sihtkoht !=
Lahtekoht));

CREATE INDEX idx_Liin_sihtkoht ON Liin (Sihtkoht);

CREATE INDEX idx_Liin_lahtekoht ON Liin (Lahtekoht);

CREATE TABLE Valjumine (
	Valjumine_id SERIAL NOT NULL,
	Liin_number SMALLINT NOT NULL,
	Valjumised VARCHAR ( 1000 ) NOT NULL,
	CONSTRAINT AK_Valjumine_liin_number UNIQUE (Liin_number),
	CONSTRAINT PK_Valjumine PRIMARY KEY (Valjumine_id));

CREATE TABLE Tootaja (
	Tootaja_kood SMALLINT NOT NULL,
	Isikukood CHAR ( 11 ) NOT NULL,
	Eesnimi VARCHAR ( 60 ) NOT NULL,
```

```
	Perekonnanimi VARCHAR ( 100 ) NOT NULL,
	CONSTRAINT AK_Tootaja_isikukood UNIQUE (Isikukood),
	CONSTRAINT PK_Tootaja PRIMARY KEY (Tootaja_kood),
	CONSTRAINT chk_Tootaja_eesnimi_ei_koosne_tyhikutest CHECK
(Eesnimi!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_isikukood_vastab_oigele_kujule CHECK
(isikukood~'^([3-6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-
3]{1}[[:digit:]]{5})$'
),
	CONSTRAINT chk_Tootaja_perekonnanimi_ei_koosne_tyhikutest CHECK
(Perekonnanimi!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_isikukood_ei_koosne_tyhikutest CHECK
(isikukood!~'^[[:space:]]*$'));

CREATE TABLE Reisi_toimumine (
	Valjumine_id SMALLINT NOT NULL,
	Tegeliku_alguse_aeg TIMESTAMP NOT NULL,
	Nadalapaev_kood SMALLINT NOT NULL,
	Tootaja_kood SMALLINT NOT NULL,
	Kellaaeg TIME ( 2147483647 ) NOT NULL,
	Kommentaar VARCHAR ( 1000 ) NOT NULL,
	CONSTRAINT PK_Reisi_toimumine PRIMARY KEY (Valjumine_id,
Nadalapaev_kood, Tegeliku_alguse_aeg));

CREATE INDEX idx_Reisi_toimumine_nadalapaev_kood ON Reisi_toimumine
(Nadalapaev_kood);

CREATE INDEX idx_Reisi_toimumine_tootaja_kood ON Reisi_toimumine
(Tootaja_kood);

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_valjumine_id
FOREIGN KEY (Valjumine_id) REFERENCES Valjumine (Valjumine_id)  ON DELETE NO
ACTION ON UPDATE CASCADE;

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_tootaja_kood
FOREIGN KEY (Tootaja_kood) REFERENCES Tootaja (Tootaja_kood)  ON DELETE NO
ACTION ON UPDATE CASCADE;

ALTER TABLE Reisi_toimumine ADD CONSTRAINT
FK_Reisi_toimumine_nadalapaev_number FOREIGN KEY (Nadalapaev_kood) REFERENCES
Nadalapaev (Nadalapaev_kood)  ON DELETE NO ACTION ON UPDATE CASCADE;

ALTER TABLE Valjumine ADD CONSTRAINT FK_Valjumine_liini_number FOREIGN KEY
(Liin_number) REFERENCES Liin (Liin_number)  ON DELETE CASCADE ON UPDATE
CASCADE;

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_lahtekoht FOREIGN KEY (Lahtekoht)
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_sihtkoht FOREIGN KEY (Sihtkoht)
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;
```

Lisa 3 - Kahemõõtmelise massiiviga andmebaasi loomise laused

```
CREATE SCHEMA schema_3;

CREATE TABLE Nadalapaev (
	Nadalapaev_kood SMALLINT NOT NULL,
	Nimetus CHAR ( 11 ) NOT NULL,
	CONSTRAINT AK_Nadalapaev_nimetus UNIQUE (Nimetus),
	CONSTRAINT PK_Nadalapaev PRIMARY KEY (Nadalapaev_kood),
	CONSTRAINT chk_Nadalapaev_kood_vahemikus_1_ja_7 CHECK (Nadalapaev_kood
BETWEEN 1 AND 7 ),
	CONSTRAINT chk_Nadalapaev_nimetus_ei_koosne_tyhikutest CHECK (Nimetus
!~'^[[:space:]]*$'));

CREATE TABLE Koht (
	Koht_kood SMALLINT NOT NULL,
	Nimetus VARCHAR ( 100 ) NOT NULL,
	CONSTRAINT PK_Koht PRIMARY KEY (Koht_kood),
	CONSTRAINT AK_Koht_nimetus UNIQUE (Nimetus),
	CONSTRAINT chk_Koht_kood_suurem_nullist CHECK (Koht_kood > 0));

CREATE TABLE Valjumine (
	Valjumine_id SERIAL NOT NULL,
	Liin_number SMALLINT NOT NULL,
	Valjumised TEXT[][] NOT NULL,
	CONSTRAINT AK_Valjumine_liin_number UNIQUE (Liin_number),
	CONSTRAINT PK_Valjumine PRIMARY KEY (Valjumine_id));

CREATE TABLE Reisi_toimumine (
	Valjumine_id SMALLINT NOT NULL,
	Tegeliku_alguse_aeg TIMESTAMP NOT NULL,
	Nadalapaev_kood SMALLINT NOT NULL,
	Tootaja_kood SMALLINT NOT NULL,
	Kellaaeg TIME ( 2147483647 ) NOT NULL,
	Kommentaar VARCHAR ( 1000 ) NOT NULL,
	CONSTRAINT PK_Reisi_toimumine PRIMARY KEY (Valjumine_id,
Nadalapaev_kood, Tegeliku_alguse_aeg));

CREATE INDEX idx_Reisi_toimumine_nadalapaev_kood ON Reisi_toimumine
(Nadalapaev_kood);

CREATE INDEX idx_Reisi_toimumine_tootaja_kood ON Reisi_toimumine
(Tootaja_kood);
```

```
CREATE TABLE Liin (
	Liin_number SMALLINT NOT NULL,
	Lahtekoht SMALLINT NOT NULL,
	Sihtkoht SMALLINT NOT NULL,
	CONSTRAINT PK_Liin PRIMARY KEY (Liin_number),
	CONSTRAINT chk_Liin_number_suurem_nullist CHECK (Liin_number > 0),
	CONSTRAINT chk_sihtkoht_ja_lahtekoht_ei_ole_samad CHECK (Sihtkoht != 
Lahtekoht));

CREATE INDEX idx_Liin_sihtkoht ON Liin (Sihtkoht);

CREATE INDEX idx_Liin_lahtekoht ON Liin (Lahtekoht);

CREATE TABLE Tootaja (
	Tootaja_kood SMALLINT NOT NULL,
	Isikukood CHAR ( 11 ) NOT NULL,
	Eesnimi VARCHAR ( 60 ) NOT NULL,
	Perekonnanimi VARCHAR ( 100 ) NOT NULL,
	CONSTRAINT AK_Tootaja_isikukood UNIQUE (Isikukood),
	CONSTRAINT PK_Tootaja PRIMARY KEY (Tootaja_kood),
	CONSTRAINT chk_Tootaja_isikukood_ei_koosne_yhikutest CHECK 
(Isikukood!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_eesnimi_ei_koosne_tyhikutest CHECK (Eesnimi 
!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_isikukood_vastab_oigele_kujule CHECK 
(isikukood~'^([3-6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-
3]{1}[[:digit:]]{5})$'
),
	CONSTRAINT chk_Tootaja_perekonnanimi_ei_koosne_tyhikutest CHECK 
(Perekonnanimi !~'^[[:space:]]*$'));

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_valjumine_id 
FOREIGN KEY (Valjumine_id) REFERENCES Valjumine (Valjumine_id)  ON DELETE NO 
ACTION ON UPDATE CASCADE;

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_tootaja_kood 
FOREIGN KEY (Tootaja_kood) REFERENCES Tootaja (Tootaja_kood)  ON DELETE NO 
ACTION ON UPDATE CASCADE;

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_nadalapaev_kood 
FOREIGN KEY (Nadalapaev_kood) REFERENCES Nadalapaev (Nadalapaev_kood)  ON 
DELETE NO ACTION ON UPDATE CASCADE;

ALTER TABLE Valjumine ADD CONSTRAINT FK_Valjumine_liin_number FOREIGN KEY 
(Liin_number) REFERENCES Liin (Liin_number)  ON DELETE CASCADE ON UPDATE 
CASCADE;

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_lahtekoht FOREIGN KEY (Lahtekoht) 
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_sihtkoht FOREIGN KEY (Sihtkoht) 
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;
```

Lisa 4 - JSON tüüpi veeruga andmebaasi loomise laused

```
CREATE SCHEMA schema_4;

CREATE TABLE Valjumine (
	Valjumine_id SERIAL NOT NULL,
	Liin_number SMALLINT NOT NULL,
	Valjumised JSON NOT NULL,
	CONSTRAINT AK_Valjumine_liin_number UNIQUE (Liin_number),
	CONSTRAINT PK_Valjumine PRIMARY KEY (Valjumine_id));

CREATE TABLE Koht (
	Koht_kood SMALLINT NOT NULL,
	Nimetus VARCHAR ( 100 ) NOT NULL,
	CONSTRAINT PK_Koht PRIMARY KEY (Koht_kood),
	CONSTRAINT AK_Koht_nimetus UNIQUE (Nimetus),
	CONSTRAINT chk_Koht_kood_suurem_nullist CHECK (Koht_kood > 0));

CREATE TABLE Liin (
	Liin_number SMALLINT NOT NULL,
	Lahtekoht SMALLINT NOT NULL,
	Sihtkoht SMALLINT NOT NULL,
	CONSTRAINT PK_Liin PRIMARY KEY (Liin_number),
	CONSTRAINT chk_Liin_number_suurem_nullist CHECK (Liin_number > 0),
	CONSTRAINT chk_sihtkoht_ja_lahtekoht_ei_ole_samad CHECK (Sihtkoht !=
Lahtekoht));

CREATE INDEX idx_Liin_sihtkoht ON Liin (Sihtkoht);

CREATE INDEX idx_Liin_lahtekoht ON Liin (Lahtekoht);

CREATE TABLE Nadalapaev (
	Nadalapaev_kood SMALLINT NOT NULL,
	Nimetus CHAR ( 11 ) NOT NULL,
	CONSTRAINT AK_Nadalapaev_nimetus UNIQUE (Nimetus),
	CONSTRAINT PK_Nadalapaev PRIMARY KEY (Nadalapaev_kood),
	CONSTRAINT chk_Nadalapaev_kood_vahemikus_1_ja_7 CHECK (Nadalapaev_kood
BETWEEN 1 AND 7),
	CONSTRAINT chk_Nadalapaev_nimetus_ei_koosne_tyhikutest CHECK (Nimetus
!~'^[[:space:]]*$'));

CREATE TABLE Reisi_toimumine (
	Valjumine_id SMALLINT NOT NULL,
	Tegeliku_alguse_aeg TIMESTAMP NOT NULL,
	Nadalapaev_kood SMALLINT NOT NULL,
```

```
	Tootaja_kood SMALLINT NOT NULL,
	Kommentaar VARCHAR ( 1000 ) NOT NULL,
	Kellaaeg TIME ( 2147483647 ) NOT NULL,
	CONSTRAINT PK_Reisi_toimumine PRIMARY KEY (Valjumine_id,
Nadalapaev_kood, Tegeliku_alguse_aeg));

CREATE INDEX idx_Reisi_toimumine_nadalapaev_kood ON Reisi_toimumine
(Nadalapaev_kood);

CREATE INDEX idx_Reisi_toimumine_tootaja_kood ON Reisi_toimumine
(Tootaja_kood);

CREATE TABLE Tootaja (
	Tootaja_kood SMALLINT NOT NULL,
	Isikukood CHAR ( 11 ) NOT NULL,
	Eesnimi VARCHAR ( 60 ) NOT NULL,
	Perekonnanimi VARCHAR ( 100 ) NOT NULL,
	CONSTRAINT AK_Tootaja_isikukood UNIQUE (Isikukood),
	CONSTRAINT PK_Tootaja PRIMARY KEY (Tootaja_kood),
	CONSTRAINT chk_Tootaja_eesnimi_ei_koosne_tyhikutest CHECK
(Eesnimi!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_isikukood_vastab_oigele_kujule CHECK
(isikukood~'^([3-6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-
3]{1}[[:digit:]]{5})$'
),
	CONSTRAINT chk_Tootaja_perekonnanimi_ei_koosne_tyhikutest CHECK
(Perekonnanimi!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_isikukood_ei_koosne_tyhikutest CHECK
(isikukood!~'^[[:space:]]*$'));

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_liin_number
FOREIGN KEY (Valjumine_id) REFERENCES Valjumine (Valjumine_id)  ON DELETE NO
ACTION ON UPDATE CASCADE;

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_tootaja_kood
FOREIGN KEY (Tootaja_kood) REFERENCES Tootaja (Tootaja_kood)  ON DELETE NO
ACTION ON UPDATE CASCADE;

ALTER TABLE Reisi_toimumine ADD CONSTRAINT FK_Reisi_toimumine_nadalapaev_kood
FOREIGN KEY (Nadalapaev_kood) REFERENCES Nadalapaev (Nadalapaev_kood)  ON
DELETE NO ACTION ON UPDATE CASCADE;

ALTER TABLE Valjumine ADD CONSTRAINT FK_Valjumine_liin_number FOREIGN KEY
(Liin_number) REFERENCES Liin (Liin_number)  ON DELETE CASCADE ON UPDATE
CASCADE;

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_lahtekoht FOREIGN KEY (Lahtekoht)
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_sihtkoht FOREIGN KEY (Sihtkoht)
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;
```

Lisa 5 - Hierarhilise JSONB tüüpi veeruga andmebaasi loomise laused

```
CREATE SCHEMA schema_5;

CREATE TABLE Nadalapaev (
	Nadalapaev_kood SMALLINT NOT NULL,
	Nimetus CHAR ( 11 ) NOT NULL,
	CONSTRAINT AK_Nadalapaev_nimetus UNIQUE (Nimetus),
	CONSTRAINT PK_Nadalapaev PRIMARY KEY (Nadalapaev_kood),
	CONSTRAINT chk_Nadalapaev_kood_vahemikus_1_ja_7 CHECK (Nadalapaev_kood
BETWEEN 1 AND 7),
	CONSTRAINT chk_Nadalapaev_nimetus_ei_koosne_tyhikutest CHECK (Nimetus
!~'^[[:space:]]*$'));

CREATE TABLE Tootaja (
	Tootaja_kood SMALLINT NOT NULL,
	Isikukood CHAR ( 11 ) NOT NULL,
	Eesnimi VARCHAR ( 60 ) NOT NULL,
	Perekonnanimi VARCHAR ( 100 ) NOT NULL,
	CONSTRAINT AK_Tootaja_isikukood UNIQUE (Isikukood),
	CONSTRAINT PK_Tootaja PRIMARY KEY (Tootaja_kood),
	CONSTRAINT chk_Tootaja_eesnimi_ei_koosne_tyhikutest CHECK
(Eesnimi!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_isikukood_vastab_oigele_kujule CHECK
(isikukood~'^([3-6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-
3]{1}[[:digit:]]{5})$'
),
	CONSTRAINT chk_Tootaja_perekonnanimi_ei_koosne_tyhikutest CHECK
(Perekonnanimi!~'^[[:space:]]*$'),
	CONSTRAINT chk_Tootaja_isikukood_ei_koosne_tyhikutest CHECK
(isikukood!~'^[[:space:]]*$'));

CREATE TABLE Valjumine (
	Valjumine_id SERIAL NOT NULL,
	Liin_number SMALLINT NOT NULL,
	Valjumised JSONB NOT NULL,
	CONSTRAINT AK_Valjumine_liin_number UNIQUE (Liin_number),
	CONSTRAINT PK_Valjumine PRIMARY KEY (Valjumine_id));
```

```
CREATE TABLE Koht (
	Koht_kood SMALLINT NOT NULL,
	Nimetus VARCHAR ( 1000 ) NOT NULL,
	CONSTRAINT PK_Koht PRIMARY KEY (Koht_kood),
	CONSTRAINT AK_Koht_nimetus UNIQUE (Nimetus),
	CONSTRAINT chk_Koht_kood_suurem_nullist CHECK (Koht_kood > 0 ));

CREATE TABLE Liin (
	Liin_number SMALLINT NOT NULL,
	Lahtekoht SMALLINT NOT NULL,
	Sihtkoht SMALLINT NOT NULL,
	CONSTRAINT PK_Liin PRIMARY KEY (Liin_number),
	CONSTRAINT chk_Liin_number_suurem_nullist CHECK (Liin_number > 0),
	CONSTRAINT chk_sihtkoht_ja_lahtekoht_ei_ole_samad CHECK (Sihtkoht !=
Lahtekoht));

CREATE INDEX idx_Liin_sihtkoht ON Liin (Sihtkoht);

CREATE INDEX idx_Liin_lahtekoht ON Liin (Lahtekoht);

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_lahtekoht FOREIGN KEY (Lahtekoht)
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Liin ADD CONSTRAINT FK_Liin_sihtkoht FOREIGN KEY (Sihtkoht)
REFERENCES Koht (Koht_kood)  ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Valjumine ADD CONSTRAINT FK_Valjumine_liin_number FOREIGN KEY
(Liin_number) REFERENCES Liin (Liin_number)  ON DELETE NO ACTION ON UPDATE
CASCADE;
```

Lisa 6 – Testandmete sisestamise laused „traditsioonilise“ disaini tabelitesse

```
COPY schema_1.nadalapaev (nadalapaev_kood, nimetus)
FROM '/tmp/Nadalapaev.csv'
WITH DELIMITER ','
CSV HEADER;

COPY schema_1.koht (koht_kood, nimetus)
FROM '/tmp/Koht.csv'
WITH DELIMITER ','
CSV HEADER;

COPY schema_1.liin (liin_number, lahtekoht, sihtkoht)
FROM '/tmp/Liin.csv'
WITH DELIMITER ','
CSV HEADER;

COPY schema_1.tootaja (tootaja_kood, isikukood, eesnimi, perekonnanimi)
FROM '/tmp/Tootaja.csv'
WITH DELIMITER ','
CSV HEADER;

COPY schema_1.valjumine (liin_number, nadalapaev_kood, kellaaeg)
FROM '/tmp/Valjumine.csv'
WITH DELIMITER ','
CSV HEADER;

COPY schema_1.reisi_toimumine (valjumine_id, tegeliku_alguse_aeg,
tootaja_kood, kommentaar)
FROM '/tmp/Reisi_toimumine.csv'
WITH DELIMITER ','
CSV HEADER;
```

Lisa 7 – Testandmete sisestamine vektorkodeerimisega disaini tabelitesse

```
INSERT INTO schema_2.koht
SELECT *
FROM schema_1.koht;

INSERT INTO schema_2.liin
SELECT *
FROM schema_1.liin;

INSERT INTO schema_2.nadalapaev
SELECT *
FROM schema_1.nadalapaev;

INSERT INTO schema_2.tootaja
SELECT *
FROM schema_1.tootaja;

INSERT INTO schema_2.valjumine ( liin_number, valjumised )

SELECT query.liin_number,
       string_agg(query.lause, ';') AS valjumised
FROM
  ( SELECT nadalapaev_kood || '-' || kellaaeg AS lause,
           liin_number
   FROM schema_1.valjumine
   GROUP BY liin_number,
            nadalapaev_kood,
            kellaaeg
   ORDER BY liin_number,
            nadalapaev_kood,
            kellaaeg ) AS query
GROUP BY query.liin_number;

INSERT INTO schema_2.reisi_toimumine ( valjumine_id, nadalapaev_kood,
kellaaeg, tegeliku_alguse_aeg, tootaja_kood, kommentaar )
SELECT v2.valjumine_id,
       v1.nadalapaev_kood,
       v1.kellaaeg,
       r1.tegeliku_alguse_aeg,
       r1.tootaja_kood,
       r1.kommentaar
```

```
FROM schema_1.valjumine AS v1, schema_1.reisi_toimumine AS r1,
schema_2.valjumine AS v2,
WHERE v1.liin_number = v2.liin_number
  AND v1.valjumine_id = r1.valjumine_id
ORDER BY v2.valjumine_id;
```

Lisa 8 – Testandmete sisestamine kahemõõtmelise massiiviga disaini tabelitesse

```
INSERT INTO schema_3.koht
SELECT *
FROM schema_1.koht;

INSERT INTO schema_3.liin
SELECT *
FROM schema_1.liin;

INSERT INTO schema_3.nadalapaev
SELECT *
FROM schema_1.nadalapaev;

INSERT INTO schema_3.tootaja
SELECT *
FROM schema_1.tootaja;

INSERT INTO schema_3.valjumine ( liin_number, valjumised )
SELECT query.liin_number, ARRAY[
schema_3.array_cat_agg(query.nadalapaev_kood_arr),
                                 schema_3.array_cat_agg(query.kellaaeg_arr) ]
AS valjumised
FROM
  ( SELECT liin_number, ARRAY[nadalapaev_kood::text] AS nadalapaev_kood_arr,
ARRAY[kellaaeg::text] AS kellaaeg_arr
   FROM schema_1.valjumine
   GROUP BY liin_number,
            nadalapaev_kood,
            kellaaeg
   ORDER BY liin_number,
            nadalapaev_kood,
            kellaaeg ) AS query
GROUP BY query.liin_number;

INSERT INTO schema_3.reisi_toimumine ( valjumine_id, nadalapaev_kood,
kellaaeg, tegeliku_alguse_aeg, tootaja_kood, kommentaar )
SELECT v3.valjumine_id,
       v1.nadalapaev_kood,
       v1.kellaaeg,
       r1.tegeliku_alguse_aeg,
       r1.tootaja_kood,
       r1.kommentaar
FROM schema_1.valjumine AS v1,
```

```
     schema_1.reisi_toimumine AS r1,
     schema_3.valjumine AS v3
WHERE v1.liin_number = v3.liin_number
  AND v1.valjumine_id = r1.valjumine_id
ORDER BY v3.valjumine_id;
```

Lisa 9 – Testandmete sisestamine JSON tüüpi veeruga disaini tabelitesse

```
INSERT INTO schema_4.koht
SELECT *
FROM schema_1.koht;

INSERT INTO schema_4.liin
SELECT *
FROM schema_1.liin;

INSERT INTO schema_4.nadalapaev
SELECT *
FROM schema_1.nadalapaev;

INSERT INTO schema_4.tootaja
SELECT *
FROM schema_1.tootaja;

INSERT INTO schema_4.valjumine ( liin_number, valjumised )
SELECT val.liin_number,
       json_object (
                   ( SELECT schema_3.reduce_dim(valjumised) AS reduced_dim
                    FROM schema_3.valjumine
                    WHERE liin_number = val.liin_number LIMIT 1 ),
                   ( SELECT schema_3.reduce_dim(valjumised) AS reduced_dim
                    FROM schema_3.valjumine
                    WHERE liin_number = val.liin_number LIMIT 1
                    OFFSET 1 ) ) AS valjumised
FROM schema_3.valjumine AS val;

INSERT INTO schema_4.reisi_toimumine ( valjumine_id, nadalapaev_kood,
kellaaeg, tegeliku_alguse_aeg, tootaja_kood, kommentaar )
SELECT v4.valjumine_id,
       v1.nadalapaev_kood,
       v1.kellaaeg,
       r1.tegeliku_alguse_aeg,
       r1.tootaja_kood,
       r1.kommentaar
FROM schema_1.valjumine AS v1,
     schema_1.reisi_toimumine AS r1,
     schema_4.valjumine AS v4
WHERE v1.liin_number = v4.liin_number
  AND v1.valjumine_id = r1.valjumine_id
```

```
ORDER BY v4.valjumine_id;
```

Lisa 10 – Testandmete sisestamine hierarhilise JSONB tüüpi veeruga disaini tabelitesse

```
INSERT INTO schema_5.koht
SELECT *
FROM schema_1.koht;

INSERT INTO schema_5.liin
SELECT *
FROM schema_1.liin;

INSERT INTO schema_5.nadalapaev
SELECT *
FROM schema_1.nadalapaev;

INSERT INTO schema_5.tootaja
SELECT *
FROM schema_1.tootaja;

INSERT INTO schema_5.valjumine (liin_number, valjumised)
SELECT v1.liin_number,
  (SELECT array_to_json(array_agg(row_to_json(nadalapaev)))
   FROM
     (SELECT v2.nadalapaev_kood,
        (SELECT array_to_json(array_agg(row_to_json(kell)))
         FROM
           (SELECT v3.kellaaeg,
              (SELECT array_to_json(array_agg(row_to_json(toimunud)))
               FROM
                 (SELECT r1.tegeliku_alguse_aeg::text, r1.tootaja_kood,
                                                      r1.kommentaar
                  FROM schema_1.reisi_toimumine AS r1
                  WHERE r1.valjumine_id = v3.valjumine_id
                  ORDER BY r1.tegeliku_alguse_aeg)toimunud) AS
toimunud_valjumised
            FROM schema_1.valjumine AS v3
            WHERE v3.liin_number = v1.liin_number
              AND v3.nadalapaev_kood = v2.nadalapaev_kood
            ORDER BY v3.kellaaeg) kell) AS kellaajad
      FROM schema_1.valjumine AS v2
      WHERE v2.liin_number = v1.liin_number
      GROUP BY v2.nadalapaev_kood
      ORDER BY v2.nadalapaev_kood)nadalapaev)::jsonb AS valjumised
FROM schema_1.valjumine AS v1
```

```
GROUP BY v1.liin_number
ORDER BY v1.liin_number;
```

Lisa 11 – Päringu S1 laused

```
/* "Traditsiooniline" andmebaasi disain */
SELECT *
FROM schema_1.valjumine
WHERE nadalapaev_kood = 1
  AND kellaaeg BETWEEN '10:00:00' AND '12:00:00'
ORDER BY liin_number;

/* Vektorkodeerimisega andmebaasi disain */
SELECT query_1.valjumine_id,
       query_1.liin_number,
       substring(query_1.valjumised
                 FROM '([1]-[1][0-2]:[0][0]:[0][0])+') AS valjumised
FROM
  (SELECT valjumine_id,
          liin_number,
          valjumised
   FROM schema_2.valjumine
   WHERE valjumised LIKE '%'|| '1-10:00:00' || '%'
     OR valjumised LIKE '%' || '1-11:00:00' || '%'
     OR valjumised LIKE '%' || '1-12:00:00' || '%') AS query_1
ORDER BY query_1.liin_number;


/* Kahemõõtmelise massiiviga andmebaasi disain*/
SELECT query_2.valjumine_id,
       query_2.liin_number,
       query_2.nadalapaev_kood[query_2.idx],
       query_2.kellaaeg[query_2.idx]
FROM
  (SELECT query_1.*,
          schema_3.idx(query_1.nadalapaev_kood, '1')
   FROM
     (SELECT valjumine_id,
             liin_number,
             schema_3.reduce_dim(valjumised[1:1]) AS nadalapaev_kood,
             schema_3.reduce_dim(valjumised[2:array_upper(valjumised,1)]) AS
kellaaeg
      FROM schema_3.valjumine
      WHERE '1' = ANY (valjumised[1:1])) AS query_1) AS query_2
WHERE kellaaeg[idx] BETWEEN '10:00:00' AND '12:00:00'
ORDER BY query_2.liin_number;
```

```
/* JSON tüübiga andmebaasi disain*/
SELECT *
FROM
  (SELECT n.valjumine_id,
          n.liin_number,
          d.key AS nadalapaev_kood,
          d.value AS kellaaeg
   FROM schema_4.valjumine n,
        json_each_text(n.valjumised) d) AS query_1
WHERE query_1.nadalapaev_kood = '1'
  AND (query_1.kellaaeg)::time BETWEEN '10:00:00' AND '12:00:00'
ORDER BY query_1.liin_number;


/* Hierarhilise JSONB tüübiga andmebaasi disain */
SELECT *
FROM
  (SELECT query_2.valjumine_id,
          query_2.liin_number,
          jsonb_array_elements(query_2.valjumised->'kellaajad') AS kellaajad
   FROM
     (SELECT *
      FROM
        (SELECT valjumine_id,
                liin_number,
                jsonb_array_elements(valjumised) AS valjumised
         FROM schema_5.valjumine) AS query_1
      WHERE query_1.valjumised->>'nadalapaev_kood'='1') AS query_2) AS
query_3
WHERE (query_3.kellaajad->>'kellaaeg')::time BETWEEN '10:00:00' AND
'12:00:00'
ORDER BY query_3.liin_number;
```

Lisa 12 – Päringu S2 laused

```
/* "Traditsiooniline" andmebaasi disain */
SELECT v.*
FROM schema_1.valjumine AS v,
     schema_1.liin AS l
WHERE v.nadalapaev_kood = 1
  AND v.kellaaeg BETWEEN '10:00:00' AND '12:00:00'
  AND v.liin_number = l.liin_number
  AND l.lahtekoht =
    (SELECT koht_kood
     FROM schema_1.koht
     WHERE nimetus = 'Põlva')
  AND l.sihtkoht =
    (SELECT koht_kood
     FROM schema_1.koht
     WHERE nimetus = 'Pärnu')
ORDER BY v.liin_number;


/* Vektorkodeerimisega andmebaasi disain */
SELECT query_1.valjumine_id,
       query_1.liin_number,
       Substring(query_1.valjumised
                 FROM '([1]-[1][0-2]:[0][0]:[0][0])+') AS valjumised
FROM
  (SELECT v.valjumine_id,
          v.liin_number,
          v.valjumised
   FROM schema_2.valjumine AS v,
        schema_2.liin AS l
   WHERE (v.valjumised LIKE '%'|| '1-10:00:00'|| '%'
          OR v.valjumised LIKE '%'|| '1-11:00:00'|| '%'
          OR v.valjumised LIKE '%'|| '1-12:00:00'|| '%')
     AND v.liin_number = l.liin_number
     AND l.lahtekoht =
       (SELECT koht_kood
        FROM schema_2.koht
        WHERE nimetus = 'Põlva')
     AND l.sihtkoht =
       (SELECT koht_kood
        FROM schema_2.koht
        WHERE nimetus = 'Pärnu')) AS query_1
ORDER BY query_1.liin_number;
```

```
/* Kahemõõtmelise massiiviga andmebaasi disain*/
SELECT query_2.valjumine_id,
       query_2.liin_number,
       query_2.nadalapaev_kood[query_2.idx],
       query_2.kellaaeg[query_2.idx]
FROM
  (SELECT query_1.*,
          schema_3.idx(query_1.nadalapaev_kood, '1')
   FROM
     (SELECT v.valjumine_id,
             v.liin_number,
             schema_3.reduce_dim(v.valjumised[1:1]) AS nadalapaev_kood,
             schema_3.reduce_dim(v.valjumised[2:array_upper(valjumised,1)])
AS kellaaeg
      FROM schema_3.valjumine AS v,
           schema_3.liin AS l
      WHERE '1' = ANY (valjumised[1:1])
        AND v.liin_number = l.liin_number
        AND l.lahtekoht =
          (SELECT koht_kood
           FROM schema_3.koht
           WHERE nimetus = 'Põlva')
        AND l.sihtkoht =
          (SELECT koht_kood
           FROM schema_3.koht
           WHERE nimetus = 'Pärnu')) AS query_1) AS query_2
WHERE kellaaeg[idx] BETWEEN '10:00:00' AND '12:00:00'
ORDER BY query_2.liin_number;


/* JSON tüübiga andmebaasi disain */
SELECT *
FROM
  (SELECT n.valjumine_id,
          n.liin_number,
          d.key AS nadalapaev_kood,
          d.value AS kellaaeg
   FROM schema_4.valjumine n,
        json_each_text(n.valjumised) d,
        schema_4.liin AS l
   WHERE l.liin_number = n.liin_number
     AND l.lahtekoht =
       (SELECT koht_kood
        FROM schema_1.koht
        WHERE nimetus = 'Põlva')
     AND l.sihtkoht =
       (SELECT koht_kood
        FROM schema_1.koht
        WHERE nimetus = 'Pärnu')) AS query_1
WHERE query_1.nadalapaev_kood = '1'
```

```
  AND (query_1.kellaaeg)::time BETWEEN '10:00:00' AND '12:00:00'
ORDER BY query_1.liin_number;


/* Hierarhilise JSONB tüübiga andmebaasi disain */
SELECT *
FROM
  (SELECT paring.valjumine_id,
          paring.liin_number,
          jsonb_array_elements(paring.valjumised->'kellaajad') AS kellaajad
   FROM
     (SELECT *
      FROM
        (SELECT v.valjumine_id,
                v.liin_number,
                jsonb_array_elements(v.valjumised) AS valjumised
         FROM schema_5.valjumine AS v,
              schema_5.liin AS l
         WHERE v.liin_number = l.liin_number
           AND l.lahtekoht =
             (SELECT koht_kood
              FROM schema_1.koht
              WHERE nimetus = 'Põlva')
           AND l.sihtkoht =
             (SELECT koht_kood
              FROM schema_1.koht
              WHERE nimetus = 'Pärnu')) AS query_1
      WHERE query_1.valjumised->>'nadalapaev_kood'='1') AS paring) AS query_2
WHERE (query_2.kellaajad->>'kellaaeg')::time BETWEEN '10:00:00' AND
'12:00:00'
ORDER BY query_2.liin_number;
```

Lisa 13 – Päringu S3 laused

```
/* "Traditsiooniline" andmebaasi disain */
SELECT count(r.tegeliku_alguse_aeg) as arv
FROM schema_1.reisi_toimumine AS r,
     schema_1.valjumine AS v
WHERE
    (SELECT EXTRACT (MONTH
                     FROM r.tegeliku_alguse_aeg)) = 03
  AND
    (SELECT EXTRACT (YEAR
                     FROM r.tegeliku_alguse_aeg)) = 2015
  AND r.valjumine_id = v.valjumine_id
  AND v.liin_number = 720;


/* Vektorkodeerimisega andmebaasi disain */
SELECT count(r.tegeliku_alguse_aeg) as arv
FROM schema_2.reisi_toimumine AS r,
     schema_2.valjumine AS v
WHERE
    (SELECT EXTRACT (MONTH
                     FROM r.tegeliku_alguse_aeg)) = 03
  AND
    (SELECT EXTRACT (YEAR
                     FROM r.tegeliku_alguse_aeg)) = 2015
  AND r.valjumine_id = v.valjumine_id
  AND v.liin_number = 720;


/* Kahemõõtmelise massiiviga andmebaasi disain */
SELECT count(r.tegeliku_alguse_aeg) as arv
FROM schema_3.reisi_toimumine AS r,
     schema_3.valjumine AS v
WHERE
    (SELECT EXTRACT (MONTH
                     FROM r.tegeliku_alguse_aeg)) = 03
  AND
    (SELECT EXTRACT (YEAR
                     FROM r.tegeliku_alguse_aeg)) = 2015
  AND r.valjumine_id = v.valjumine_id
  AND v.liin_number = 720;
```

```
/* JSON tüübiga andmebaasi disain*/
SELECT count(r.tegeliku_alguse_aeg) as arv
FROM schema_4.reisi_toimumine AS r,
     schema_4.valjumine AS v
WHERE
    (SELECT EXTRACT (MONTH
                     FROM r.tegeliku_alguse_aeg)) = 03
  AND
    (SELECT EXTRACT (YEAR
                     FROM r.tegeliku_alguse_aeg)) = 2015
  AND r.valjumine_id = v.valjumine_id
  AND v.liin_number = 720;


/* Hierarhilise JSONB tüübiga andmebaasi disain */
SELECT count(query_1.toimunud_valjumised) as arv
FROM
  (SELECT valjumine_id,
          liin_number,

jsonb_array_elements(jsonb_array_elements(jsonb_array_elements(valjumised)-
>'kellaajad')->'toimunud_valjumised') AS toimunud_valjumised
   FROM schema_5.valjumine
   WHERE liin_number = 720) AS query_1
WHERE
    (SELECT EXTRACT (MONTH
                     FROM ((toimunud_valjumised-
>'tegeliku_alguse_aeg')::text)::TIMESTAMP)) = 03
  AND
    (SELECT EXTRACT (YEAR
                     FROM ((toimunud_valjumised-
>'tegeliku_alguse_aeg')::text)::TIMESTAMP)) = 2015;
```

Lisa 14 – Päringu S4 laused

```
/* "Traditsiooniline" andmebaasi disain */
SELECT r.tootaja_kood,
       t.perekonnanimi,
       n.nimetus,
       r.tegeliku_alguse_aeg,
       v.kellaaeg
FROM schema_1.reisi_toimumine AS r,
     schema_1.valjumine AS v,
     schema_1.tootaja AS t,
     schema_1.nadalapaev AS n
WHERE v.valjumine_id = r.valjumine_id
  AND v.liin_number = 720
  AND t.tootaja_kood = r.tootaja_kood
  AND n.nadalapaev_kood = v.nadalapaev_kood
ORDER BY r.tegeliku_alguse_aeg;


/* Vektorkodeerimisega andmebaasi disain */
SELECT r.tootaja_kood,
       t.perekonnanimi,
       n.nimetus,
       r.tegeliku_alguse_aeg,
       r.kellaaeg
FROM schema_2.reisi_toimumine AS r,
     schema_2.valjumine AS v,
     schema_2.nadalapaev AS n,
     schema_2.tootaja AS t
WHERE v.valjumine_id = r.valjumine_id
  AND v.liin_number = 720
  AND n.nadalapaev_kood = r.nadalapaev_kood
  AND t.tootaja_kood = r.tootaja_kood
ORDER BY r.tegeliku_alguse_aeg;


/* Kahemõõtmelise massiiviga andmebaasi disain*/
SELECT r.tootaja_kood,
       t.perekonnanimi,
       n.nimetus,
       r.tegeliku_alguse_aeg,
       r.kellaaeg
FROM schema_3.reisi_toimumine AS r,
     schema_3.valjumine AS v,
     schema_3.nadalapaev AS n,
```

```
     schema_3.tootaja AS t
WHERE v.valjumine_id = r.valjumine_id
  AND v.liin_number = 720
  AND n.nadalapaev_kood = r.nadalapaev_kood
  AND t.tootaja_kood = r.tootaja_kood
ORDER BY r.tegeliku_alguse_aeg;

/* JSON tüübiga andmebaasi disain */
SELECT r.tootaja_kood,
       t.perekonnanimi,
       n.nimetus,
       r.tegeliku_alguse_aeg,
       r.kellaaeg
FROM schema_4.reisi_toimumine AS r,
     schema_4.valjumine AS v,
     schema_4.nadalapaev AS n,
     schema_4.tootaja AS t
WHERE v.valjumine_id = r.valjumine_id
  AND v.liin_number = 720
  AND n.nadalapaev_kood = r.nadalapaev_kood
  AND t.tootaja_kood = r.tootaja_kood
ORDER BY r.tegeliku_alguse_aeg;

/* Hierarhilise JSONB tüübiga andmebaasi disain */
SELECT query_2.tootaja_kood,
       t.perekonnanimi,
       n.nimetus,
       query_2.tegeliku_alguse_aeg,
       query_2.kellaaeg
FROM
  (SELECT ((jsonb_array_elements(query_1.toimunud_valjumised)-
>'tootaja_kood')::text)::int AS tootaja_kood,
          query_1.nadalapaev_kood,
          query_1.kellaaeg,
          ((jsonb_array_elements(query_1.toimunud_valjumised)-
>'tegeliku_alguse_aeg')::text)::TIMESTAMP AS tegeliku_alguse_aeg
   FROM
     (SELECT ((jsonb_array_elements(valjumised)-
>'nadalapaev_kood')::text)::int AS nadalapaev_kood,
             ((jsonb_array_elements(jsonb_array_elements(valjumised)-
>'kellaajad')->'kellaaeg')::text)::time AS kellaaeg,
             jsonb_array_elements(jsonb_array_elements(valjumised)-
>'kellaajad')->'toimunud_valjumised' AS toimunud_valjumised
      FROM schema_5.valjumine AS v
      WHERE liin_number = 720) AS query_1) AS query_2,
     schema_5.tootaja AS t,
     schema_5.nadalapaev AS n
WHERE t.tootaja_kood = query_2.tootaja_kood
  AND n.nadalapaev_kood = query_2.nadalapaev_kood
ORDER BY query_2.tegeliku_alguse_aeg;
```

Lisa 15 - Operatsiooni I1 laused

```
/* "Traditsiooniline" andmebaasi disain */
INSERT INTO schema_1.valjumine (liin_number, nadalapaev_kood, kellaaeg)
VALUES (
          (SELECT liin_number
           FROM schema_1.liin
           WHERE lahtekoht=
               (SELECT koht_kood
                FROM schema_1.koht
                WHERE nimetus='Tallinn')
             AND sihtkoht=
               (SELECT koht_kood
                FROM schema_1.koht
                WHERE nimetus='Pärnu')), 2,
                                      '10:00:00');

/* Vektorkodeerimisega andmebaasi disain */
UPDATE schema_2.valjumine
SET valjumised=(valjumised || ';2-10:00:00')
WHERE liin_number =
    (SELECT liin_number
     FROM schema_2.liin
     WHERE lahtekoht=
         (SELECT koht_kood
          FROM schema_2.koht
          WHERE nimetus='Tallinn')
       AND sihtkoht=
         (SELECT koht_kood
          FROM schema_2.koht
          WHERE nimetus='Pärnu'));

/* Kahemõõtmelise massiiviga andmebaasi disain*/
UPDATE schema_3.valjumine
SET valjumised =
  (SELECT array[array_append(
                             (SELECT schema_3.reduce_dim(valjumised[1:1])),
'2'),
                array_append(
                             (SELECT
schema_3.reduce_dim(valjumised[2:array_upper(valjumised,1)])), '10:00:00')])
WHERE liin_number =
    (SELECT liin_number
     FROM schema_2.liin
     WHERE lahtekoht=
```

```
          (SELECT koht_kood
           FROM schema_3.koht
           WHERE nimetus='Tallinn')
        AND sihtkoht=
          (SELECT koht_kood
           FROM schema_3.koht
           WHERE nimetus='Pärnu'));

/* JSON tüübiga andmebaasi disain */
UPDATE schema_4.valjumine
SET valjumised = (schema_4.json_merge(valjumised,json'{"2":"10:00:00"}'))
WHERE liin_number =
    (SELECT liin_number
     FROM schema_4.liin
     WHERE lahtekoht=
          (SELECT koht_kood
           FROM schema_4.koht
           WHERE nimetus='Tallinn')
        AND sihtkoht=
          (SELECT koht_kood
           FROM schema_4.koht
           WHERE nimetus='Pärnu'));
```

Lisa 16 – Päringute tegemiseks loodud lisafunktsioonid

```
CREATE
OR REPLACE FUNCTION schema_3.idx(anyarray, anyelement) RETURNS INT AS $$
SELECT i
FROM
  (SELECT generate_series(array_lower($1, 1), array_upper($1, 1))) g(i)
WHERE $1[i] = $2 LIMIT 1; $$ LANGUAGE SQL IMMUTABLE;
```

```
CREATE OR REPLACE FUNCTION schema_3.reduce_dim(anyarray)
RETURNS SETOF anyarray AS
$function$
DECLARE
    s $1%TYPE;
BEGIN
    FOREACH s SLICE 1  IN ARRAY $1 LOOP
        RETURN NEXT s;
    END LOOP;
    RETURN;
END;
$function$
LANGUAGE plpgsql IMMUTABLE;
```

```
CREATE OR REPLACE FUNCTION schema_4.json_merge(data json, merge_data json)
RETURNS json
IMMUTABLE
LANGUAGE sql
AS $$
    SELECT ('{'||string_agg(to_json(key)||':'||value, ',')||'}')::json
    FROM (
        WITH to_merge AS (
            SELECT * FROM json_each(merge_data)
        )
        SELECT *
        FROM json_each(data)
        WHERE key NOT IN (SELECT key FROM to_merge)
        UNION ALL
        SELECT * FROM to_merge
    ) t;
$$;
```