TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDU40LT

Kristiina Kristal 135059IAPB

ANDMETÜÜPIDE VALIKU MÕJUST POSTGRESQL ANDMEBAASIDE KASUTAMISELE

Bakalaureusetöö

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: Kristiina Kristal

23.05.2016

Annotatsioon

Töö eesmärgiks on uurida, kuidas mõjutavad tabelite disainimisel erinevate andmetüüpide valikud PostgreSQL andmebaasis andmemahte, päringute kiirust ja lihtsust.

Samade tõeste faktide hoidmiseks mõeldud andmebaasi saab sageli disainida mitmel erineval viisil, kasutades näiteks erinevaid andmetüüpe. Sellest võib sõltuda, kui suureks kujunevad andmemahud, missugune on andmebaasis toimuvate operatsioonide jõudlus ja andmekäitluskeele lausete lihtsus.

Töös luuakse katsetamiseks tabelid, kuhu genereeritakse testandmed, mille peal käivitatakse mõned SQL päringud. Selle protsessi tulemusena on võimalik välja selgitada, kas ja kuidas tehtud disainivalikute puhul erinevad andmetüübid andmebaasi kasutamist mõjutasid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 5 peatükki, 18 joonist, 2 tabelit.

Abstract

On the Influence of the Selection of Data Types to the Use of PostgreSQL Databases

The purpose of this thesis is to find out how the selection of different data types during the design of tables influence the PostgreSQL databases and their use in terms of data size, query performance and query simplicity.

One can design a database that is meant for storing the same true propositions in various ways by using, for instance, different data types. The choices the designer makes determine the data size on disk, goodness of the performance of the database operations and simplicity of the data manipulation statements.

In this research, the author will create tables and generate test data, on which to run some SQL statements. As a result of this process, it will become more obvious if and how the choices of data types affect the use of the database.

The thesis contains four experiments. One compares the use of INTEGER and DECIMAL types. Another compares the use of CHAR and VARCHAR types. There are also experiments with array and enumeration types. In case of the latter, we compare the designs with more "regular" designs that do not use such types.

Based on the results of the experiments, we conclude that the selection of data types has noticeable influence to the properties of database as well as performance and simplicity of queries. The experiments show advantages of INTEGER and VARCHAR over DECIMAL and CHAR, respectively. The experiments also show that one should avoid the use of enumeration types. Array types need more investigation.

The thesis is in Estonian and contains 29 pages of text, 5 chapters, 18 figures, 2 tables.

Lühendite ja mõistete sõnastik

PostgreSQL	Tasuta ja avatud lähtekoodiga objekt-relatsiooniline andmebaasisüsteem, milles saab kasutada andmebaasikeelt SQL [13].
SQL	*Structured Query Language*, relatsioonilise andmemudeli alusel välja töötatud andmebaasikeel, mis hõlmab lauseid andmete otsimiseks või uuendamiseks, struktuuri ning seda ümbritseva andmebaasiobjektide ökosüsteemi loomiseks ja muutmiseks ning võimalust kontrollida juurdepääsu andmetele [11].
SLOC-P	Füüsiliste käivitatavate koodiridade arv. Leitakse nii, et koodiridade koguarvust lahutatakse kommentaari ridade ja tühjade ridade arv.
UML	*Unified Modeling Language*, unifitseeritud modelleerimiskeel tarkvara ja infosüsteemide spetsifitseerimiseks ja visualiseerimiseks [12]. Tegemist on üldotstarbelise keelega, mis tähendab, et seda saab kasutada paljudeks ülesanneteks, sh andmebaasi kavandamiseks.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Andmebaasisüsteemi abil loodud andmebaasis on paljudel juhtudel võimalik andmestruktuurides samade faktide esitamiseks valida mitme andmetüübi vahel. Andmetüüpide valikust sõltuvad andmebaasi omadused nagu andme maht, andmetega tehtavate operatsioonide jõudlus ning andmete reeglitele vastavus. Need omakorda mõjutavad seda, kui palju kasu saavad antud andmebaasi kasutajad, näiteks firmad ja nende kliendid. Kui on tegemist näiteks laialt levinud e-poega, mille andmebaas sisaldab infot kõikide saadaval olevate toodete kohta, siis rakenduse kasutajate jaoks on olulisim kriteerium töökiirus. Sellisel juhul tuleks teha kõik võimalik (sh valida hoolikalt andmetüüpe), et teatud päringud tagastaks soovitud tulemused võimalikult kiiresti ega jääks liiga kauaks tööle ehk jõudlus oleks maksimaalne. Andmetüübi valikust võib sõltuda ka andmekäitluskeele lausete lihtsus ning andmetele kehtivate reeglite kontrolli lihtsus, mis on oluline süsteemi hallatavuse ja edasiarendatavuse seisukohalt.

Kuna andmetüübid on erinevate sisemiste esitusviisidega, siis sellest võib tuleneda operatsioonide kiirus. Operaatorid, mis toimivad nende andmete peal, võivad üht tüüpi andmetega toimida kiiremini, teistega hoopis aeglasemalt. Sama põhimõtte järgi võivad teatud tüüpi andmed enda alla võtta suurema andmemahu.

Andmebaasi disainimise alguses oleks hea, kui leiduks eelinfot disainivalikute tagajärgede kohta. See info peaks põhinema reaalsetel mõõtmistulemustel, mitte lihtsalt suust-suhu kanduvatel kuulujuttudel. Selline eelinfo annaks objektiivse aluse paremate disainiotsuste langetamiseks, et teha kaalutletud valikuid. Paremad otsused disaini käigus tähendavad, et hiljem on vaja andmebaasi disaini vähem parandada ehk refaktoreerida. Hea variant oleks see, kui kusagil on juba loodud mitu konkureerivat disainilahendust ja nende peal on läbi viidud katsed ning toodud välja arvulised tulemused.

Sageli pole arendajatel piisavalt aega andmebaasi kavandamiseks. Disainifoorumitest loetu alusel või kusagilt millegi kuuldu põhjal valitakse esimene disain, mis asja ära

ajab ja võimaldab andmebaasi kasutava tarkvara tööle saada. Samas andmebaas on süsteemi keskne komponent, millest otseselt ja kaudselt sõltuvad peaaegu kõik süsteemi osad ning nende kirjeldused (rakendused, mudelid, testid). Kui tehakse halbu andmebaasi disainivalikuid, siis võib pärast andmebaasi struktuuri parandamine kujuneda väga valuliseks ja töömahukaks tegevuseks. Muidugi on olemas refaktoreerimine, muidugi saab andmebaasi struktuuri muuta, muidugi võib teha vigu, kuid parem informatsioon andmebaaside kohta aitaks tekkivat tüli ja vaeva vähendada.

Hea näide sarnase probleemi kohta on PostgreSQL (9.4) andmebaasis domeeni muutmine käsuga ALTER DOMAIN, millel puudub baastüübi muutmise võimalus. Tuleb hakata vaeva nägema hoopis uue domeeni tekitamise ning vana domeeniga seotud tabelite veergude omaduste muutmisega.

Andmebaasisüsteeme on palju ja kõigi nende andmetüüpide uurimine ületaks kaugelt bakalaureusetöö mahu. Tuleb valida üks andmebaasisüsteem. Käesolevas töös uurin, kuidas mõjutab erinevate andmetüüpide valik populaarse avatud lähtekoodiga SQL-andmebaasisüsteemi PostgreSQL kasutamist. Valikute põhjusteks on nii autori varasem kokkupuude selle süsteemiga, süsteemi populaarsus (2016. aasta mai seisuga on see DB-Engines veebilehekülje andmebaasisüsteemide populaarsuse indeksi viiendal kohal) kui ka selle pakutavate andmetüüpide rohkus. Eesmärgiks on selle näitel heita valgust erinevate andmetüüpide ning disainide valiku headele ja halbadele külgedele. Püstitatud eesmärkide saavutamiseks loon tabelid, millesse genereerin testandmed ja nende peal viin läbi katsed SQL päringutega. Tulemusena saan võrrelda, kas ning kuidas tehtud valikud avaldavad mõju andmemahtudele, päringute kiirusele ja lihtsusele.

2 Teoreetiline taust

Järgnevalt annan lühikese ülevaate sellest, millega on andmetüübi puhul tegemist ja missugused on läbi viidud eksperimentides omavahel võrreldud andmetüübid. Lisaks toon välja nende olulisemad erinevused ning mõningate näidete peal selgitan, kuidas selliste tüüpidega väärtusi andmebaasi salvestatakse. Konkreetsetest andmetüüpidest räägitakse töös kasutatava PostgreSQL andmebaasisüsteemi näitel.

2.1 Mis on andmetüüp?

Andmetüüp on nime omav väärtuste hulk, millega on võimalik ära määrata, missuguseid väärtusi saab põhimõtteliselt andmebaasis hoida. Andmetüübi termini asemel kasutatakse ka mõistet domeen (seda ei tohiks teha SQL-andmebaasidest rääkides, sest seal on tüüp ja domeen erinevad andmebaasiobjektid) ja lühidalt nimetatakse seda lihtsalt tüübiks. Andmetüübi puhul pole tegemist ei muutuja ega väärtusega [3].

Ühel tüübil võib olla mitu alternatiivset nimetust, mida kasutades saab sellele tüübile viidata, sõltuvalt kasutatavast andmebaasist. Näiteks andmetüübi CHARACTER alternatiivne nimi on CHAR ning INTEGER alternatiivne nimi on INT [3]. Arv, mis on sulgudes tüübi võtmesõna järel, näitab maksimaalset väljapikkust, näiteks VARCHAR(100) või CHAR(1). Määratud pikkusest pikemaid väärtuseid sellist tüüpi veergu kirjutada ei saa ning andmebaas tagastab veateate [9].

Relatsiooniline mudel kategoriseerib andmetüüpe järgnevalt:

- Skalaarsed tüübid – sellistesse tüüpidesse kuuluvatel väärtustel ei ole kasutajatele nähtavaid komponente.
- Mitteskalaarsed tüübid – sellistesse tüüpidesse kuuluvatel väärtustel on kasutajatele nähtavad komponendid, mida nimetatakse atribuutideks. Mitteskalaarsed tüübid jaotuvad omakorda kahte klassi: korteeži ja relatsiooni tüüpideks [3].

Lisaks on võimalik jaotada andmetüüpe nende looja järgi:

- Süsteemi-defineeritud tüübid, mis on andmebaasisüsteemi loojate poolt loodud.
- Kasutaja-defineeritud tüübid, mis on andmebaasi disainerite ja ehitajate poolt loodud [3].

Iga andmebaasisüsteemi poolt toetatava tüübi puhul on oluline, et andmebaasisüsteem lubaks kasutada seda tüüpi nii muutujate, operaatorite, parameetrite, atribuutide kui ka avaldiste tüübina. Tüübid on relatsioonilise mudeli suhtes ortogonaalsed ehk ristuvad, mis tähendab, et relatsiooniline mudel ei piira erinevate tüüpide kasutamist [3].

Iga väärtus, muutuja, väärtust tagastav operaator, operaatori parameeter, atribuut ning avaldis on mingit tüüpi. Iga väärtusega peab olema vähemalt kontseptuaalselt seotud tüübi identifikaator, kuhu see väärtus kuulub. Muutuja võimalikud väärtused saavad olla ainult selle muutujaga seotud tüüpi, mis tähendab, et need kuuluvad muutuja tüübiga määratud väärtuste hulka. Operaatoriga tehtud operatsiooni tulemused võivad olla vaid operaatori loomisel määratud tüüpi. Parameetri võimalikud väärtused ehk argumendid saavad olla ainult selle parameetriga seotud tüüpi. Atribuudi võimalikud väärtused saavad olla ainult selle atribuudiga seotud tüüpi. Avaldise tüüp on sama kui selle operaatori tüüp, mille poole avaldise arvutamise käigus kõige viimasena pöördutakse [3].

2.2 DECIMAL ja INTEGER

DECIMAL on fikseeritud komakohaga kümnendarv [9], mille täpsuse saab kasutaja antud numbrilise tüübi kasutamisel määrata [8]. Selle kirjeldamise korral näidatakse ära arvu pikkus ning kümnendkohtade arv. Kui näiteks on vaja andmebaasi panna arv, mille maksimaalne suurus on 999,99 ja minimaalne suurus -999,99, siis tuleb kirjeldada tabelisse veerg, mille tüüp on DECIMAL(5,2) [9]. Antud tüüpi sobib kasutada siis, kui on vaja salvestada suuri komakohaga arve ja teha nendega täpseid arvutusi, näiteks rahaliste väärtuste puhul. Samas on DECIMAL tüüpi väärtustega tehete sooritamine PostgreSQLi näitel aeglasem kui INTEGER tüübiga [8].

INTEGER on andmetüüp täisarvude jaoks, mis mahutab endasse kuni nelja baidi pikkuse täisarvu. Selle tüübi puhul on kindlalt määratud, kui suures vahemikus täisarve on võimalik sisestada. INTEGER on tavaline valik, kui andmebaasi soovitakse

salvestada numbriline väärtus, sest see tasakaalustab kõige paremini suhet salvestatava väärtuse vahemiku, pikkuse ja andmebaasi töökiiruse vahel [8].

2.3 CHAR ja VARCHAR

CHAR on ajalooliselt vanem teksti hoidmiseks mõeldud andmetüüp. Võrreldes VARCHAR tüübiga, eraldatakse andmebaasis CHAR tüübile alati maksimaalselt nii palju mälu, kui kasutaja on eelnevalt defineerinud, ja kasutamata ruum täidetakse tühikutega [6]. Näiteks on tabelis veerg *nimi* tüübiga CHAR(25), siis väärtuste "Mati Maasikas" ja "Alan" sisestamisel reserveeritakse mõlemale 25 baiti [9].

VARCHAR on ajalooliselt uuem teksti salvestamiseks sobilik andmetüüp, mis kasutab mälu kokkuhoidlikumalt kui CHAR tüüp. Nimelt eraldatakse VARCHAR tüüpi veergudes mälu parasjagu nii palju, kui mitu sümbolit konkreetsesse tabeli välja on kirjutatud. Sama näidet jätkates oleks "Mati Maasikas" väärtus VARCHAR tüübi puhul reserveerinud 13 baiti ja "Alan" ainult 4 baiti [9].

2.4 ARRAY ja ENUM

ARRAY on massiivitüübi konstruktor, mille abil saab luua massiivitüüpe igale süsteemi- või kasutaja-defineeritud andmetüübile. Tüübikonstruktor on operaator uute tüüpide loomiseks. Selle poole pöördumiseks tuleb lisada vastava tüübi nimetuse lõppu nurksulud [5]. Näiteks veeru *e_meilid* VARCHAR[] võimalikeks väärtusteks on ühemõõtmelised massiivid, millesse on võimalik salvestada VARCHAR tüüpi väärtusi. Massiivi omaduseks on, et selles sisalduvad väärtused on järjestatud ning nendele väärtustele saab positsiooni teades viidata.

ENUM tüübi puhul on tegemist andmetüübiga, mis moodustab staatilise ning kindla järjekorraga väärtuste hulga. Üks ENUM tüüpi väärtus võtab mälust neli baiti. Tüübis olevate väärtuste järjekord on määratud järjekorraga tüübi loomise lauses (Vt Joonis 1) [7].

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
```

Joonis 1. SQL päringu näide ENUM tüübi loomiseks [7].

3 Metoodika

Siinkohal räägin pikemalt, kuidas valmistasin ette vajalikud tingimused, et sooritada PostgreSQL andmebaasis erinevate disainilahendustega tabelitesse sisestatud testandmete peal katseid. Esmalt toon välja mõned otsingutulemused veebi avarustest, mida oli vaja veendumiseks, et täpselt sarnast uurimistööd pole varem tehtud (või vastupidi käesoleva tööga kõrvutamiseks). Peale selle selgitan ühe konkreetse näite abil vastavate tabelite ja nendesse genereeritud testandmete loomise protsessi.

3.1 Otsingutulemused sarnaste tööde kohta

Internetis päringute tegemiseks kasutasin Google'i otsingumootorit (https://www.google.ee/). Kõik allpool nimetatud märksõnadega päringud vaadati viimati üle 21.05.2016, iga päringu puhul vaid esimesed 5 lehekülge, sest sealt edasi ei olnud otsingutulemused enam soovitud tulemusele piisavalt lähedal või hakkasid teatud tulemused korduma.

Otsingumootorisse sisestati järgmised päringud:

- *on the influence of the selection of data types to the use of postgresql databases*
- *different data types affect postgresql*
- *faster data types postgresql*
- *what data type to use postgresql*
- *compare data types postgresql*
- *research on data types postgresql*
- *decimal vs integer postgresql*
- *char vs varchar postgresql*
- *array vs none postgresql*
- *enum vs none postgresql*

Antud töö pealkirja proovisin seetõttu, kuna see kirjeldab lahendatavat ülesannet kõige täpsemalt. *postgresql* oli alati päringusse lisatud, et tagastataks võimalikult palju tulemusi, mis oleks kindlasti seotud PostgreSQL andmebaasisüsteemiga.

Põhiliselt koosnesid otsingutulemused viidetest PostgreSQL ametlikule dokumentatsioonile, abistavate ja selgitavate juhistega lehekülgedele (näiteks Tutorialspoint: http://www.tutorialspoint.com/postgresql/postgresql_data_types.htm), Stack Overflow foorumites esitatud küsimustele või aruteludele. Nendel veebisaitidel tuuakse pigem välja kõik olemasolevad andmetüübid nimekirjana, üldiselt erinevate tüüpide eelised ja puudused, võrreldakse omavahel andmebaasisüsteeme (PostgreSQL, MySQL, SQLite) või seletatakse lahti, kuidas ühest andmetüübist teisendada väärtust teise tüüpi. Midagi spetsiifilist pole uuritud, vaid on näha korduvaid samu mõtteid, miks peaks üht tüüpi eelistama teisele.

Leidsin kolm huvipakkuvat artiklit, milles on lühidalt räägitud samal teemal ja osaliselt tehtud katseid sarnaste meetoditega, kuid mitte niivõrd põhjalikult:

1. See jaapanikeelne lehekülg tõmbas tähelepanu sellepärast, et seal on üritatud teha midagi sarnast suure hulga testandmete genereerimise ja stringitüüpide võrdlemisega. Ligikaudse tõlke saamiseks kopeerisin teksti Google'i tõlkesse (https://translate.google.ee/) ning selgus, et pigem on uuritud seda, kuidas erineb stringitüüpide korral andmete salvestamine [14].
2. Huvitava näitega kirjutatud artikkel sellest, kuidas SQL päringu ühes reas massiivitüüpi väärtuse asendamine VALUES klauslitega muutis päringu töökiirust 100 korda paremaks. Otsustasin selle esile tõsta, kuna antud töös on oluline, missuguste SQL päringutega katsetusi tehakse [2].
3. Leitud blogi postituses on kirjeldatud, kuidas võrreldakse MySQL andmebaasis kolme erineva disainilahendusega (ENUM, VARCHAR ja INT andmetüüpe sisaldavatele) tabelitele tehtud päringute kiiruseid. Tegemist on kõige lähedasema ja sarnasema põhimõttega uuringuga, mis Google'i otsingutulemustes kajastus, ainult teisel andmebaasisüsteemil põhineva ning väiksema koguse uuritavate aspektidega (ainult päringu kiirus) [1].

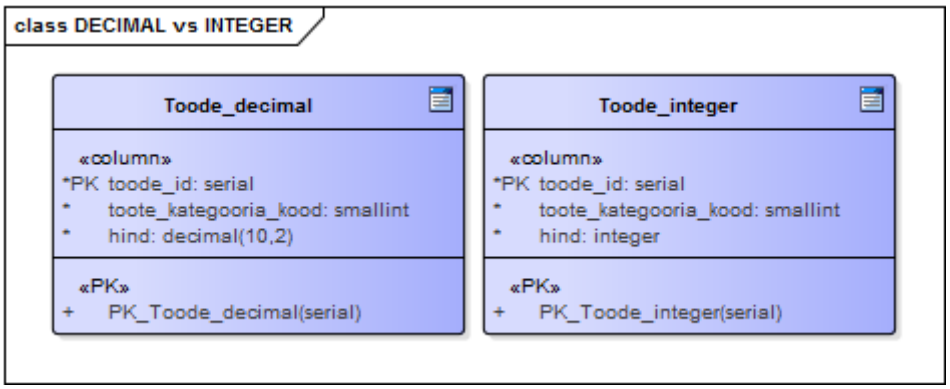
3.2 Tabelite loomine

Andmebaasis loodavate tabelite mudelite disainimiseks kasutasin Enterprise Architecti (edaspidi EA) versiooni 12.0, mille *Generate DDL* käsuga on võimalik genereerida valmis nende tabelite loomiseks vajalikud SQL laused. Saadud SQL lauseid tuli muuta vastavalt vajadusele lihtsamaks või lisada veergude nimetuste taha teatud andmetüübid, mida tabelite disainimisel PostgreSQLi jaoks EA valikute hulgas ei leidunud. Tõsi küll, EA võimaldab ka tabelite disaineritel andmebaasisüsteemi jaoks valikust puuduvaid andmetüüpe juurde defineerida.

Iga võrdluse puhul kavandasin iga disaini kohta 1–2 tabelit. Loodud tabelid on mõeldud antud võrdluses samade andmete hoidmiseks erinevatel viisidel ning esitatud diagrammid on UML notatsioonis. Primaarvõtme (PRIMARY KEY) kitsendused algavad lühendiga PK ja primaarvõtme veeru nimetuse ees esineb vastav tähis, sama kehtib välisvõtme (FOREIGN KEY) kitsendustele, mille lühendiks on FK. Indeksid on sarnaselt tähistatud sõnaga *index*, kuid UNIQUE kitsendused ainult tähega U ja tabeli veerud, mis seda unikaalsust tagavad, on alla joonitud.

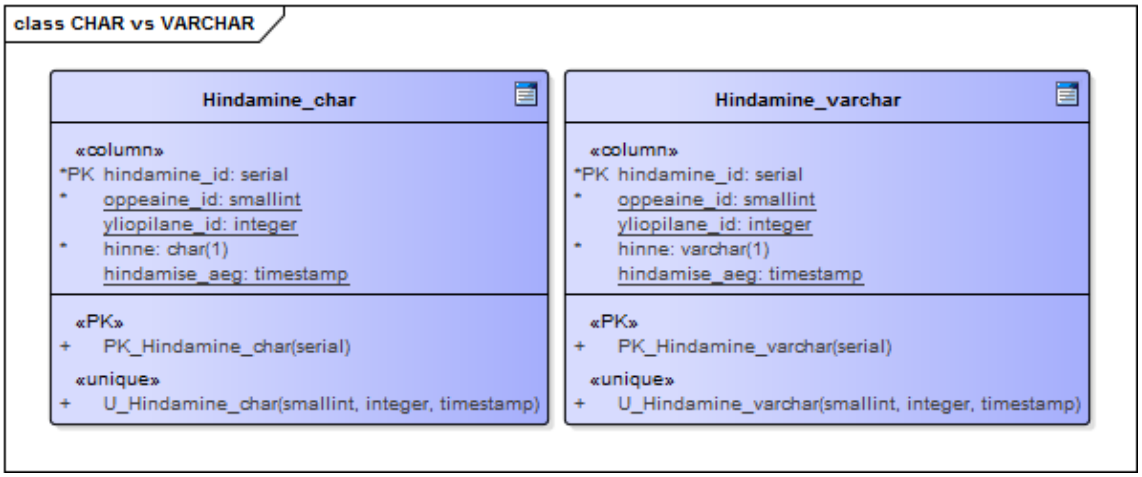
3.3 Tabelite disainid

Järgnevalt kirjeldatakse neljas eksperimendis kasutatud tabeleid.



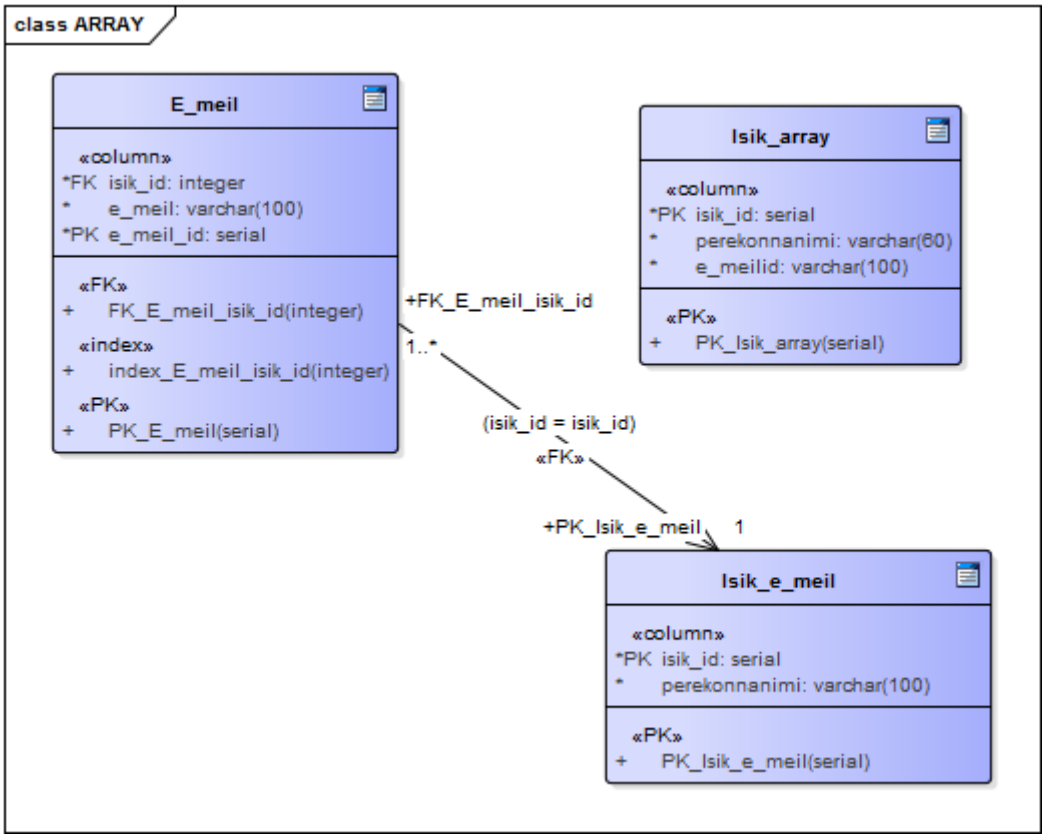
Joonis 2. DECIMAL ja INTEGER tüüpe võrdlevate tabelite füüsilise disaini andmebaasi diagramm.

Esmalt disainisin kaks tabelit arvuliste andmetüüpide võrdlemiseks. *Toode_decimal* tabelis hoitakse veerus *hind* väärtusi DECIMAL(10,2) tüübina ning *Toode_integer* tabelis, nagu nimigi viitab, INTEGER tüübina (Vt Joonis 2). DECIMAL(10,2) tüübiga tabelis näitab kohtade arv peale koma sentide arvu, kuid INTEGER tüübiga tabelis hoitakse hinda ümardatult.



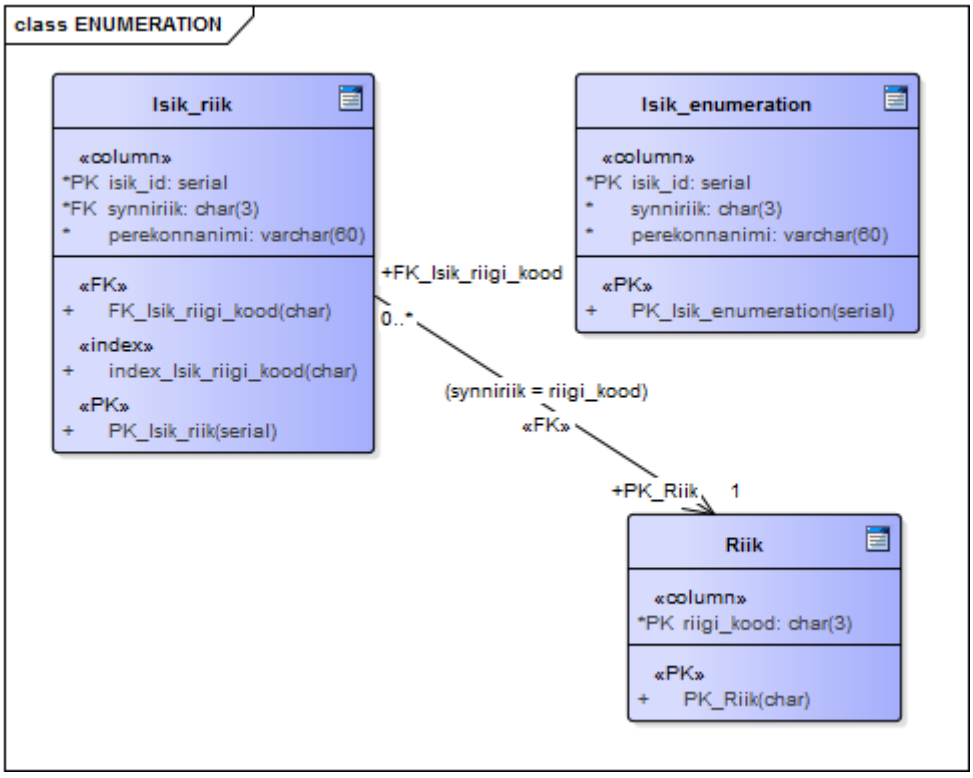
Joonis 3. CHAR ja VARCHAR tüüpe võrdlevate tabelite füüsilise disaini andmebaasi diagramm.

Järgnevalt disainisin kaks tabelit stringitüüpide võrdlemiseks lühikeste väärtuste korral. *Hindamine_char* tabelis salvestatakse *hinne* CHAR(1) tüüpi veerus ja *Hindamine_varchar* tabelis VARCHAR(1) tüüpi veerus (Vt Joonis 3).



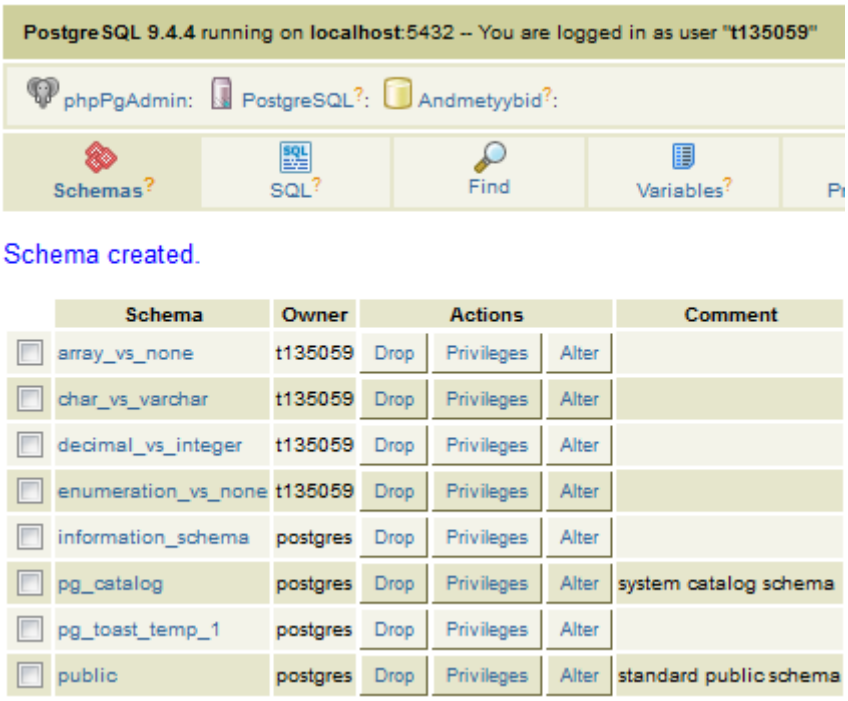
Joonis 4. Massiivitüüpi kasutava ja ilma massiivitüübita tabelite füüsilise disaini andmebaasi diagramm.

Kolmandana disainisin tabelid, millest üks variant kasutab isikute e-posti aadresside hoidmiseks massiivitüüpi veergu. Teise puhul seostatakse sama isiku identifikaator isikute (*Isik_e_meil*) tabelist iga kord uue aadressiga *E_meil* tabelist. (Vt Joonis 4)



Joonis 5. Loendustüüpi kasutava ja ilma loendtüübita tabelite füüsilise disaini andmebaasi diagramm.

Viimased tabelid disainisin sarnaselt eelmises võrdluses loodud tabelitega. Ühes leidus loendtüüpi veerg, aga teises seostati kindlast riikide nimekirjast pärit riigi kood isiku sünniriigiga *Isik_riik* tabelis CHAR(3) tüübina (Vt Joonis 5). See disain illustreerib erinevaid võimalusi SQL-andmebaasis klassifikaatorite esitamiseks. Klassifikaatori näiteks on antud juhul *Riik*. Kuna väga paljud süsteemid kasutavad klassifikaatoreid, siis on selle disaini testimise tulemused huvipakkuvad väga suure süsteemide hulga jaoks.



Joonis 6. Andmebaasis *Andmetyybid* loodud skeemid.

Andmebaasi loomiseks ja eksperimentide teostamiseks valitud keskkond on PostgreSQL (versioon 9.4) andmebaasisüsteem serveris *apex.ttu.ee*. Sinna tekitasin uue andmebaasi nimega *Andmetyybid*. Loodud andmebaasis tegin kokku neli eraldi skeemi (üks iga võrdluse jaoks), mille nimede järgi on kerge leida võrreldavate andmetüüpidega tabelid (Vt Joonis 6Joonis 6). Serveri tehnilised andmed on järgmised: virtuaalmasin QEMU Virtual CPU version, 811 GB HDD, 40 GB RAM, 15 virtuaalset CPU'd, CentOS 6.4.

3.4 Testandmete genereerimine

Kõikidesse testis kasutatud tabelitesse genereeriti üks miljon rida testandmeid. Loodud tabelitesse testandmete genereerimiseks kasutan PostgreSQLi süsteemi-defineeritud *generate_series* funktsiooni. Lisaks sellele õnnestus testandmete generaatori Mockaroo [4] abiga luua 1000 rea kaupa võimalikult tõetruud stringitüüpi testandmed, näiteks isikute perekonnanimed ja e-posti aadressid. Käesolevas peatükis selgitan, kuidas see protsess toimub ühe võrdluse jaoks disainitud tabelitega, milleks valisin välja CHAR(1) ja VARCHAR(1) andmetüüpe sisaldavad *Hindamine_char* ning *Hindamine_varchar* tabelid.

Esiteks loon *Andmetyybid* andmebaasi skeemi *char_vs_varchar* ühe uue tabeli *Hindamine_char* ja sellega seotud kitsendused (Vt Joonis 7, lk 20).

```
CREATE TABLE Hindamine_char (
	hindamine_id SERIAL,
	oppeaine_id smallint NOT NULL,
	yliopilane_id integer NULL,
	hinne char(1) NOT NULL,
	hindamise_aeg timestamp NULL
);

ALTER TABLE Hindamine_char ADD CONSTRAINT PK_Hindamine_char
PRIMARY KEY (hindamine_id);

ALTER TABLE Hindamine_char ADD CONSTRAINT U_Hindamine_char
UNIQUE (oppeaine_id,yliopilane_id,hindamise_aeg);
```

Joonis 7. Laused tabeli *Hindamine_char* ja sellega seotud kitsenduste loomiseks.

Järgmisena on vaja luua kaks abitabelit, mis koosnevad vaid kahest veerust. Esimene veerg on SERIAL tüüpi ja hoiab endas teisele veerule vastava väärtuse identifikaatorit. Teine veerg sisaldab valitud vahemikus genereeritud numbrilist väärtust, mille hiljem tõstan UPDATE lausega baastabelisse. (Vt Joonis 8)

```
CREATE TABLE Oppeaine (
	oppeaine_id SERIAL,
	oppeaine_number smallint NOT NULL
);

CREATE TABLE Yliopilane (
	yliopilane_id SERIAL,
	yliopilase_number integer NOT NULL
);

ALTER TABLE Oppeaine ADD CONSTRAINT PK_Oppeaine
PRIMARY KEY (oppeaine_id);

ALTER TABLE Yliopilane ADD CONSTRAINT PK_Yliopilane
PRIMARY KEY (yliopilane_id);
```

Joonis 8. Laused tabelite *Oppeaine* ja *Yliopilane* ning nendega seotud kitsenduste loomiseks.

Olgu antud juhul nii erinevaid õppeaineid kui ka üliõpilasi kümme tuhat, seega genereerin loodud abitabelitesse miljon rida suvalisi väärtusi vahemikus 1–10000 kasutades RANDOM() funktsiooni (Vt Joonis 9, lk 21).

```
INSERT INTO Oppeaine (oppeaine_number)
SELECT (RANDOM()*(10000-1))::INT + 1
FROM (SELECT generate_series(1,1000000)) AS foo;

INSERT INTO Yliopilane (yliopilase_number)
SELECT (RANDOM()*(10000-1))::INT + 1
FROM (SELECT generate_series(1,1000000)) AS foo;
```

Joonis 9. Laused abitabelitesse miljoni rea väärtuste genereerimiseks.

Seejärel sisestan baastabelisse miljon rida testandmeid. Õppeaine ja üliõpilase identifikaatorite väärtusteks määran esialgu arvu 1, sest need asendan hiljem abitabelites loodud väärtustega. Tänu sellele, et hinde jaoks pole eraldi abitabelit vaja teha, saan RANDOM() funktsiooni abil igale reale määrata suvalise hinde väärtuse, mis jääb vahemikku 0–5. Valitud kuupäeva ja kellaaja vahemik koos piisavalt väikese intervalliga tagab vähemalt miljoni unikaalse rea genereerimise. Kuna määratud ajavahemikku mahub üle miljoni rea, siis tuleb liigsed read tabelist kustutada, et testandmete hulk oleks kõikide katsete puhul sama (Vt Joonis 10).

```
INSERT INTO Hindamine_char (oppeaine_id, yliopilane_id, hinne, hindamise_aeg)
VALUES (1, 1, ((RANDOM()*(5-0))::INT + 0)::char(1), generate_series('2000-
01-01 00:00'::timestamp, '2016-01-01 00:00', '5 minutes'::interval));

DELETE FROM Hindamine_char WHERE hindamine_id BETWEEN 1000001 AND 1683073;
```

Joonis 10. Laused baastabelisse väärtuste genereerimiseks ja liigsete ridade kustutamiseks.

Peale seda on võimalik abitabelitesse genereeritud õppeaine ja üliõpilase identifikaatori numbrilised väärtused ümber paigutada baastabelisse. UPDATE lausega määran veergude *oppeaine_id* ja *yliopilane_id* uued väärtused, mille kontrollimiseks kehtib tingimus, et nii baastabelis kui abitabelis olevad SERIAL tüüpi veeru väärtused peavad olema võrdsed (Vt Joonis 11).

```
UPDATE Hindamine_char SET oppeaine_id=Oppeaine.oppeaine_number
FROM Oppeaine
WHERE Hindamine_char.hindamine_id=Oppeaine.oppeaine_id;

UPDATE Hindamine_char SET yliopilane_id=Yliopilane.yliopilase_number
FROM Yliopilane
WHERE Hindamine_char.hindamine_id=Yliopilane.yliopilane_id;
```

Joonis 11. Laused baastabelis veergude väärtuste uuendamiseks.

Kui esimeses baastabelis on sobilikud testandmed genereeritud, siis jääb ainult luua teine baastabel *Hindamine_varchar* koos vajalike kitsendustega ja INSERT lausega võib esimesest tabelist testandmed sinna üle tõsta (Vt Joonis 12, lk 22). See tagab, et sama võrdluse korral esitavad kõigi disainilahenduste alusel loodud tabelid samu fakte.

Andmete üle tõstmisel tegi süsteem ise ilmutamata tüübiteisenduse CHAR ja VARCHAR tüüpi väärtuste vahel. Loendtüübi puhul oli vaja tabelisse *Isik_riik* andmete üle tõstmisel teha ise veerule *synniriik* tüübiteisendus ENUM ja CHAR(3) tüüpi väärtuste vahel.

```
CREATE TABLE Hindamine_varchar (
	hindamine_id SERIAL,
	oppeaine_id smallint NOT NULL,
	yliopilane_id integer NULL,
	hinne varchar(1) NOT NULL,
	hindamise_aeg timestamp NULL
);

ALTER TABLE Hindamine_varchar ADD CONSTRAINT PK_Hindamine_varchar
PRIMARY KEY (hindamine_id);

ALTER TABLE Hindamine_varchar ADD CONSTRAINT U_Hindamine_varchar
UNIQUE (oppeaine_id,yliopilane_id,hindamise_aeg);

INSERT INTO Hindamine_varchar (oppeaine_id, yliopilane_id, hinne,
hindamise_aeg)
SELECT oppeaine_id, yliopilane_id, hinne, hindamise_aeg
FROM Hindamine_char;
```

Joonis 12. Laused tabeli *Hindamine_varchar* ja sellega seotud kitsenduste loomiseks ning testandmete sisestamiseks.

Peale testandmete genereerimist värskendasin ANALYZE käsuga andmebaasi statistikat.

3.5 Testandmete puudused

Massiivide katse puhul on puuduseks, et igale isikule määrati täpselt üks e-posti aadress. Päriselus on erinevatel isikutel erinev hulk e-meile. Veel üks testandmete puudus seisneb selles, et INTEGER ja DECIMAL tüübi katsetamise korral on ühes andmebaasis hinnad komakohtadega ja teises mitte. Tegelikkuses peaks INTEGER tüübi kasutamise korral hoidma andmebaasis hinda sentides ning vajadusel selle 100-ga läbi jagades leidma hinna eurodes.

3.6 Eksperimentide teostamine

Disainilahenduste võrdluse sisuks on andmemahud, päringute kiirus ja nende jõustamise lihtsus ehk koodiridade arv.

Koodiridade arvu kiireks arvutamiseks saan abi LocMetrics programmilt (http://www.locmetrics.com/). Kasutan koodiridade mõõdikut, mis loeb kokku käivitatavate füüsiliste ridade arvu (SLOC-P). Selle mõõdiku puhul ei arvestata tühikute ja kommentaari ridadega. Selleks, et koodiridade arvud oleksid võrreldavad, tuleb koodi formaatida ühesuguste reeglite abil. Kasutan selleks Instant SQL Formatter vahendit (http://www.dpriver.com/pp/sqlformat.htm) ja valin sealt *database* tüübiks *Generic*.

Andmemahtude mõõtmiseks kasutan igas skeemis päringut, mis tagastab antud skeemis olevate tabelite andmemahu baitides (Vt Joonis 13) [10].

```
SELECT table_name, pg_total_relation_size(table_schema ||'.'|| table_name)
FROM INFORMATION_SCHEMA.tables
WHERE table_type = 'BASE TABLE' AND table_schema = 'char_vs_varchar'
ORDER BY table_name;
```

Joonis 13. SQL päring andmemahtude leidmiseks PostgreSQL andmebaasis.

Katsete käigus teen erinevaid päringuid, mis sisaldavad alati seda veergu, mille andmetüübi valikut kahe tabeli vahel võrreldakse. Jätkates eelmises alapeatükis käsitletud näidet, võiks üheks päringuks olla selline, mis leiab iga hinde kaupa hinnete arvu ehk kui mitu korda on antud hinnet üliõpilased kokku saanud (Vt Joonis 14).

```
SELECT hinne, COUNT(hinne) AS arv
FROM hindamine_char
GROUP BY hinne
ORDER BY hinne;

SELECT hinne, COUNT(hinne) AS arv
FROM hindamine_varchar
GROUP BY hinne
ORDER BY hinne;
```

Joonis 14. Näide eksperimendis kasutatavatest päringutest.

Päringute kiiruse leidmiseks kasutan EXPLAIN ANALYZE lauset. Selle käivitamisel näidatakse lause täitmisplaani, kuid lause täidetakse ka tegelikult ning näidatakse täitmisplaani koostamiseks ja plaani täitmiseks kulunud aega. Käivitasin iga lauset kolm korda ning liitsin iga kord kokku plaani koostamise ning täitmise aja.

4 Tulemused

Andmebaasis loodud ja testandmetega täidetud tabelite andmemahtude tulemused on esitatud alljärgnevas tabelis (Vt Tabel 1). Rohelise taustaga on värvitud ühest võrdlusest parem (väiksema andmemahuga) tulemus, punasega halvem (suurema andmemahuga) tulemus. Andmemahtude korral on märgatavad isegi kahekordsed erinevused sõltuvalt sellest, missugust tüüpi veeruga tabelid on loodud. Siinkohal tuleks ära märkida, et ka indeksid võtavad enda alla teatud hulga andmemahust. Kui oleks vaja langetada otsus põhinedes vaid andmemahtude kriteeriumile, siis oleks soovitatav eelistada:

- numbriliste väärtuste salvestamiseks INTEGER tüüpi DECIMAL tüübile,
- stringitüüpi väärtuste salvestamiseks VARCHAR(1) tüüpi CHAR(1) tüübile,
- massiivitüüpi ja hoiduda tabelitest, mis seda võimalusel ei kasuta,
- loendtüübita tabeleid ja hoiduda tabelitest, mis seda kasutavad.

Tabel 1. Andmemahtude erinevused.

Tabeli nimi	**Andmemaht (baitides)**	**Andmemaht (MB)**
Toode_decimal	133873664	127,67
Toode_integer	66805760	63,71
Hindamine_char	352010240	335,70
Hindamine_varchar	115417088	110,07
Isik_array	113065984	107,83
Isik_e_meil, E_meil	147841024 151207936	140,99 144,20
Isik_enumeration	204300288	194,84
Isik_riik, Riik	96755712 57344	92,27 0,05

Andmebaasis tabelite disainimisel tuleb lisaks arvestada seda, kuidas tehtud valik hakkab mõjutama jõudlust. Tabelites testandmete peal tehtud päringute katsete

tulemused on kirjeldatud järgmises tabelis (Vt Tabel 2). Ühte ja sama päringut käivitati kolm korda ja kõigi kolme katsetuse tulemused pandi tabelisse kirja.

Tabel 2. Päringute kiiruse ja lihtsuse võrdlemine.

Tabeli nimi	**Päringu kiirus (ms)**	**Päringu lihtsus (koodiridade arv)**
Toode_decimal	1,100.815 1,008.825 1,069.345	2
Toode_integer	961.638 968.415 981.590	2
Hindamine_char	5,438.673 5,444.757 5,339.202	2
Hindamine_varchar	5,158.430 5,156.094 5,180.347	2
Isik_array	524.673 562.263 501.548	3
Isik_e_meil, E_meil	1,544.135 2,049.983 1,433.970	5
Isik_enumeration	192.073 309.188 320.388	3
Isik_riik, Riik	70.296 29.858 25.876	3

Kollase taustaga on värvitud need lahtrid, kus päringute lihtsus ehk koodiridade arv on võrdne. Päringute keerukuse erinevus vajaks põhjalikumat uurimist suurema hulga päringute põhjal. Hüpoteesina võiksid erinevused tekkida just disainides, kus kasutatakse loendtüüpi või massiivitüüpi veerge võrrelduna disainidega, kus neid ei

kasutata. Nagu tulemustest näha, siis massiivitüüpi veeru korral on päring tõepoolest koodiridade arvu seisukohalt lihtsam, sest ei ole vaja ühendada kahte tabelit.

Rohelise taustaga on praktiliselt alati kiirema ajaga sooritatud päringud ning punasega vastavalt aeglasemad päringud. Nagu ka andmemahtude puhul, nii on päringute täitmise kiiruste vahel näha olulisi erinevusi, hoolimata sellest, et päringute lihtsus jääb igas võrdluses samaks.

Esimeses võrdluses, kus võrreldakse DECIMAL ja INTEGER andmetüüpe, käivitasin päringu, mis pidi leidma toote keskmise hinna (Vt Joonis 15).

```
SELECT AVG(hind) AS keskmine_hind
FROM toode_decimal;

SELECT AVG(hind) AS keskmine_hind
FROM toode_integer;
```

Joonis 15. Päring keskmise hinna arvutamiseks.

Kuigi DECIMAL andmetüüpi veergu sisaldavast tabelist tuli vastus üks kord suhteliselt kiiresti, siis valdavalt jäid kiiremateks INTEGER tüüpi veeruga tabelis tehtud päringud. Ka ülejäänud eksperimentides kasutatud päringutele leidis süsteem tulemused kiiremini peamiselt tabeli *Toode_integer* põhjal. See kinnitab varem teoorias mainitud väidet, et DECIMAL tüübiga saab tehetele küll täpsema vastuse, kuid arvutamise operatsioon võtab kauem aega.

Teises võrdluses, kus võrreldakse CHAR ja VARCHAR andmetüüpe, käivitasin päringu, mis otsis üles erinevate võimalike hinnete arvu (Vt Joonis 16). Selleks oli alati arv 6, sest hinnete vahemikku kuulusid täisarvud 0–5.

```
SELECT COUNT(DISTINCT hinne) AS arv
FROM hindamine_char;

SELECT COUNT(DISTINCT hinne) AS arv
FROM hindamine_varchar;
```

Joonis 16. Päring erinevate võimalike hinnete arvu otsimiseks.

CHAR tüüpi veeru puhul tagastati vastus ligikaudu 300 ms aeglasemalt kui VARCHAR tüüpi veeru korral. Sarnaselt eelmise võrdlusega selgub, et tabeli *Hindamine_varchar* põhjal leiti teistes katsetes päringule vastus enamjaolt kiiremini.

Kolmandas võrdluses tuli välja, et massiivitüüpi veergu kasutava tabeli peal tehtud päring oli rohkem kui üks sekund kiirem. Antud juhul võis teise disainilahenduse aegluse põhjuseks olla vajadus päringus kaks tabelit ühendada. Soovisin, et süsteem otsiks kõikide S-tähega algava perekonnanimega isikute e-posti aadressid ning eemaldaks kordused (Vt Joonis 17).

```
SELECT DISTINCT e_meilid[1]
FROM isik_array
WHERE perekonnanimi LIKE 'S%';

SELECT DISTINCT e_meil
FROM e_meil INNER JOIN isik_e_meil ON e_meil.isik_id=isik_e_meil.isik_id
WHERE perekonnanimi LIKE 'S%';
```

Joonis 17. Päring e-meilide leidmiseks, mille omaniku perekonnanimi algab S-tähega.

Vastupidiselt juhtus aga teiste läbi viidud katsetega (Vt Lisa 1, lk 31): tabelit *Isik_array* kasutavad päringud olid katsetes pigem aeglasemad kui päringud, mis kasutasid massiivitüüpideta tabeleid. Seega ei saa selle võrdluse puhul olla kindel, millal üks disainivalik on teisest jõudluse poolest parem.

Loendtüübiga *vs* loendtüübita tabelite võrdluses on päringu täitmise kiirustel vahe juba enam kui kolmekordne. Erinevalt eelmisest võrdlusest pole siin päringu täitmisel kaht tabelit ühendatud INNER JOIN operatsiooniga (Vt Joonis 18). Sellest saab järeldada, et põhjus väiksemas jõudluses võibki seisneda ENUM tüübi kasutamises.

```
SELECT COUNT(isik_id) AS rahvaarv
FROM isik_enumeration
WHERE synniriik='LTU';

SELECT COUNT(isik_id) AS rahvaarv
FROM isik_riik
WHERE synniriik='LTU';
```

Joonis 18. Päring Leedust pärit isikute arvu leidmiseks.

Kui oleks vaja langetada otsus põhinedes vaid jõudluse kriteeriumile, siis oleks soovitatav eelistada:

- numbriliste väärtuste salvestamiseks INTEGER tüüpi DECIMAL tüübile,
- stringitüüpi väärtuste salvestamiseks VARCHAR(1) tüüpi CHAR(1) tüübile,
- loendtüübita tabeleid ja hoiduda tabelitest, mis seda kasutavad.

Massiivitüüpi veergudega tabeleid tuleks veel katsetada.

5 Kokkuvõte

Töö põhieesmärgiks oli uurida, kuidas mõjutavad tabelite disainimisel erinevate andmetüüpide valikud PostgreSQL andmebaasis andmemahte, päringute kiirust ja lihtsust. Pärast edukate katsete sooritamist võib väita, et nendest tõepoolest sõltuvad nii andmemahud kui ka andmebaasis toimuvate operatsioonide jõudlus.

Kui arvestada nii andmemahu kui ka jõudluse kriteeriumiga, siis numbriliste väärtuste puhul on soovitatav eelistada kõige tavalisemat INTEGER-i tüüpi, millega saab kiireid operatsioone teostada ja mis hõivab mälus vähem ruumi võrreldes DECIMAL-iga. Kuna CHAR tüüp salvestab alati väärtuse maksimaalse lubatud pikkusega füüsiliselt andmebaasi, siis on stringitüübi jaoks mõistlikum kasutada VARCHAR-i. Loendtüübi ehk ENUM-i asemel tuleks võimalusel midagi muud kasutada, sest tegemist on väga mahuka ning aeglase andmetüübiga. Massiivitüübi kohta oleks vaja põhjalikumalt edasi uurida, millal seda on hea kasutada ja milliste disainivalikute puhul mitte.

Samade tõeste faktide hoidmiseks mõeldud andmebaasi saab sageli disainida mitmel erineval viisil, kasutades erinevaid andmetüüpe. Antud töös käsitleti vaid paari üksikut lahendust, mistõttu ütlen lõpetuseks, et need ei ole kindlasti iga probleemi lahendamiseks sobilikud. Loodetavasti on aga tehtud eksperimendid kasulikud kellelegi, kes disaini käigus hakkab tegema sarnaseid valikuid sama ülesande lahendamiseks ning selleks on eelnevalt läbi proovitud katsete kohta informatsioon kättesaadav just käesolevast tööst.

Autor tänab töö juhendajat, Erki Eessaart, kasulike nõuannete, abi ja mõistva suhtumise eest.

Kasutatud kirjandus

[1] A. Kovyrin, „Enum Fields VS Varchar VS Int + Joined table: What is Faster?,“ [Võrgumaterjal]. Available: https://www.percona.com/blog/2008/01/24/enum-fields-vs-varchar-vs-int-joined-table-. [Kasutatud 18.05.2016].

[2] A. Lê-Quôc, „100x faster Postgres performance by changing 1 line,“ [Võrgumaterjal]. Available: https://www.datadoghq.com/blog/100x-faster-postgres-performance-by-changing-1-line/. [Kasutatud 18.05.2016].

[3] E. Eessaar, „Andmebaaside lisamaterjalid,“ [Võrgumaterjal]. Available: http://maurus.ttu.ee/ained/IDU0220_IDU0230/doc/6/Teema_IDU0220_2_2016.pdf . [Kasutatud 18.05.2016].

[4] „Mockaroo,“ [Võrgumaterjal]. Available: https://www.mockaroo.com/. [Kasutatud 19.05.2016].

[5] „PostgreSQL,“ [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/arrays.html. [Kasutatud 18.05.2016].

[6] „PostgreSQL,“ [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/datatype-character.html. [Kasutatud 18.05.2016].

[7] „PostgreSQL,“ [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/datatype-enum.html. [Kasutatud 18.05.2016].

[8] „PostgreSQL,“ [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/datatype-numeric.html. [Kasutatud 18.05.2016].

[9] P. Raspel. [Võrgumaterjal]. Available: http://enos.itcollege.ee/~priit/E-kursus%20%28I%20245%29%20AB-de%20alused/02/69678595001_2-2.htm. [Kasutatud 18.05.2016].

[10] S. Puustusmaa, „Andmebaaside lisamaterjalid,“ [Võrgumaterjal]. Available: http://maurus.ttu.ee/ained/IDU0220_IDU0230/doc/26/Votmed_SQL_andmebaasides.pdf. [Kasutatud 21.05.2016].

[11] „Vikipeedia,“ [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Struktuurp%C3%A4ringukeel. [Kasutatud 18.05.2016].

[12] „Vikipeedia,“ [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/%C3%9Chtne_modelleerimiskeel. [Kasutatud 18.05.2016].

[13] „Wikipedia,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/PostgreSQL. [Kasutatud 18.05.2016].

[14] [Võrgumaterjal]. Available: http://lets.postgresql.jp/documents/technical/text-processing/1. [Kasutatud 18.05.2016].

Lisa 1 – Eksperimentides kasutatud SQL päringud

1. DECIMAL ja INTEGER andmetüüpide võrdlemiseks loodud tabelite peal käivitatud päringud – esitan vaid tabeli *Toode_decimal* peal tehtud päringud, sest need erinevad ainult tabeli nimes esineva tüübi nimetuse poolest.

```
/* Keskmine hind kategooriate kaupa */
SELECT toote_kategooria_kood AS toote_kategooria, AVG(hind) AS keskmine_hind
FROM toode_decimal
GROUP BY toote_kategooria_kood
ORDER BY toote_kategooria_kood;

/* Keskmine hind kategooriate järgi */
SELECT AVG(hind) AS keskmine_hind_1_10
FROM toode_decimal
WHERE toote_kategooria_kood BETWEEN 1 AND 10;

/* Otsing hinnavahemiku järgi */
SELECT * FROM toode_decimal WHERE hind BETWEEN 50 AND 100 ORDER BY toode_id;

SELECT * FROM toode_decimal WHERE hind >= 900 ORDER BY toode_id;

SELECT * FROM toode_decimal WHERE hind < 10 ORDER BY toode_id;

/* Otsing kategooria järgi */
SELECT toode_id, hind FROM toode_decimal WHERE toote_kategooria_kood=1 ORDER
BY toode_id;

SELECT toode_id, hind FROM toode_decimal WHERE toote_kategooria_kood=50 ORDER
BY toode_id;

/* Läbi jagamise tehe */
SELECT AVG(hind)/2 AS keskmine_hind_pooleks
FROM toode_decimal;

SELECT AVG(hind) AS keskmine_hind, AVG(hind)/5 AS viiendik, AVG(hind)/4 AS
veerand, AVG(hind)/3 AS kolmandik
FROM toode_decimal;
```

2. CHAR ja VARCHAR andmetüüpide võrdlemiseks loodud tabelite peal käivitatud päringud – esitan vaid tabeli *Hindamine_char* peal tehtud päringud, sest need erinevad ainult tabeli nimes esineva tüübi nimetuse poolest.

```
/* Õppeained, kus on saadud antud hinne */
SELECT DISTINCT oppeaine_id
FROM hindamine_char
WHERE hinne='5';

/* Iga hinde kaupa hinnete arv ehk kui palju on seda hinnet saadud */
SELECT hinne, COUNT(hinne) AS arv
FROM hindamine_char
GROUP BY hinne
ORDER BY hinne;

/* Konkreetse üliõpilase hinded */
SELECT oppeaine_id, hinne, hindamise_aeg
FROM hindamine_char
WHERE yliopilane_id=250 ORDER BY hindamise_aeg;

/* Konkreetses õppeaines saadud hinded */
SELECT yliopilane_id, hinne, hindamise_aeg
FROM hindamine_char
WHERE oppeaine_id=1337
ORDER BY yliopilane_id;

/* Teatud ajavahemikus saadud hinded */
SELECT oppeaine_id, yliopilane_id, hinne, hindamise_aeg
FROM hindamine_char
WHERE hindamise_aeg BETWEEN '2001-09-01 00:00:00'::timestamp AND '2002-06-01
00:00:00'::timestamp;

/* Keskmine hinne */
SELECT AVG(hinne::INT) AS keskmine_hinne
FROM hindamine_char;
```

3. Massiivitüübiga ja ilma massiivitüübita loodud tabelite peal käivitatud päringud.

```
/* E-mailid, mis lõppevad .com-ga */
SELECT DISTINCT e_meilid[1]
FROM isik_array
WHERE e_meilid[1] LIKE '%.com';

/* E-mailid, mis kuuluvad isikutele, kelle perekonnanimi on 'Hansen' */
SELECT e_meilid[1] FROM isik_array WHERE perekonnanimi='Hansen';

/* E-mailide arv, mis on seotud Google'iga */
SELECT COUNT(e_meilid[1]) AS arv FROM isik_array WHERE e_meilid[1] LIKE
'%google%';

/* E-mailid, mis on seotud Skype'iga */
SELECT isik_id, perekonnanimi, e_meilid[1] FROM isik_array WHERE e_meilid[1]
LIKE '%skype%' ORDER BY isik_id;
```

```
/* E-mailid, mis lõppevad .com-ga */
SELECT DISTINCT e_meil FROM e_meil WHERE e_meil LIKE '%.com';

/* E-mailid, mis kuuluvad isikutele, kelle perekonnanimi on 'Hansen' */
SELECT e_meil FROM e_meil INNER JOIN isik_e_meil
ON e_meil.isik_id=isik_e_meil.isik_id WHERE perekonnanimi='Hansen';

/* E-mailide arv, mis on seotud Google'iga */
SELECT COUNT(e_meil) AS arv FROM e_meil WHERE e_meil LIKE '%google%';

/* E-mailid, mis on seotud Skype'iga */
SELECT e_meil.isik_id, perekonnanimi, e_meil FROM e_meil INNER JOIN
isik_e_meil
ON e_meil.isik_id=isik_e_meil.isik_id WHERE e_meil LIKE '%skype%' ORDER BY
isik_id;
```

4. Loendtüübiga ja ilma loendtüübita loodud tabelite peal käivitatud päringud.

```
/* Isikud, kes on pärit Eestist */
SELECT *
FROM isik_enumeration
WHERE synniriik='EST';

/* Isikud, kelle perekonnanimi algab 'K'-tähega */
SELECT *
FROM isik_enumeration
WHERE perekonnanimi LIKE 'K%';

/* Erinevad riigid */
SELECT DISTINCT synniriik
FROM isik_enumeration;

/* Isikud, kelle perekonnanimi on Andrews */
SELECT isik_id, synniriik
FROM isik_enumeration
WHERE perekonnanimi='Andrews'
ORDER BY isik_id;

/* Riigi kood, mis sisaldab tähte 'A' */
SELECT synniriik
FROM isik_enumeration
WHERE synniriik::char(3) LIKE '%A%';

/* Riigi kood, mis sisaldab 'AU' */
SELECT synniriik
FROM isik_enumeration
WHERE synniriik::char(3) LIKE '%AU%';
```

```
/* Isikud, kes on pärit Eestist */
SELECT *
FROM isik_riik
WHERE synniriik='EST';

/* Isikud, kelle perekonnanimi algab 'K'-tähega */
SELECT *
FROM isik_riik
WHERE perekonnanimi LIKE 'K%';

/* Erinevad riigid */
SELECT DISTINCT synniriik
FROM isik_riik;

/* Isikud, kelle perekonnanimi on Andrews */
SELECT isik_id, riigi_kood
FROM isik_riik INNER JOIN riik ON riigi_kood=synniriik
WHERE perekonnanimi='Andrews'
ORDER BY isik_id;

/* Riigi kood, mis sisaldab tähte 'A' */
SELECT synniriik
FROM isik_riik
WHERE synniriik LIKE '%A%';

/* Riigi kood, mis sisaldab 'AU' */
SELECT synniriik
FROM isik_riik
WHERE synniriik LIKE '%AU%';
```