TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Andreas Aadli 164679IAPB

POSTGRESQL ANDMEBAASIDE DENORMALISEERIMIST ABISTAVA TARKVARA LOOMINE

bakalaureusetöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Andreas Aadli

24.05.2019

Annotatsioon

Antud lõputöö eesmärgiks on luua tarkvara, mis abistaks PostgreSQL andmebaaside arendajaid andmebaaside denormaliseerimisel. Tegemist on SQL-andmebaaside refaktoreerimist abistava tarkvaraga. Kõigepealt selgitan töös denormaliseerimise olemust ning teen lühikokkuvõte mõningatest sellega seonduvatest teadustöödest, kus analüüsitakse selle rakendusi reaalses elus ja kus seda oleks võimalik kasutada. Seejärel esitan tarkvara analüüsi ja disaini tulemused. Analüüsi eesmärgiks on selgitada välja nõuded rakendusele ning disaini eesmärgiks on koostada nõuete, võimaluste ja piirangutega arvestav tehniline kavand.

Töö lõplikuks tulemuseks on Pythonis kirjutatud kasutaja arvutisse installeeritav rakendus, mille abil on andmebaasi disaineril võimalik luua olemasoleva andmebaasi põhjal uus, denormaliseeritud baastabelitega (edaspidi tabelitega), andmebaasi skeem. Denormaliseerimise programm realiseerib Steve Hobermani poolt välja pakutud SQL-andmebaasi tabelite denormaliseerimise algoritmi. Ma ei ole teadlik, et seda oleks varem programmina realiseeritud. Rakenduse kasutaja peab vastama küsimustele tabelite vaheliste seoste kohta ning lõpptulemuseks on hulk SQL lauseid, millega saab luua uue struktuuriga tabelid.

Töö valideerimiseks rakendan tarkvara andmebaasile, mille denormaliseerimise näidet kasutati varasemas lõputöös. Seejärel võrdlen eelneva lõputöö tulemusel valminud SQL koodi ja käesoleva lõputöö programmi poolt genereeritud SQL koodi ning analüüsin erinevusi. Töö viimases osas hindan enda poolt tehtud töö vastavust püstitatud eesmärkidele ning analüüsin ebaedu põhjuseid.

Tarkvara on avatud lähtekoodiga ning publitseeritud MIT litsentsiga. Tarkvara lähtekood on avaldatud aadressil: https://github.com/andreasaadli/hoberman-algorithm

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 43 leheküljel, 9 peatükki, 15 joonist, 3 tabelit.

Abstract

Creating a Software for Aiding the Denormalization of PostgreSQL Databases

The main goal of this thesis is to make a software that can aid PostgreSQL database designers in denormalizing databases. The software can be used for refactoring PostgreSQL databases in order to potentially improve the performance of certain queries. For this purpose the system implements the David Hoberman's denormalization algorithm. I am not aware of any other software that impelements the algorithm. Firstly, I give a short overview of the purpose of denormalization and where it can be used in real-life. Next, I present the results of analysis and design of the software. The purpose of analysis is to find requirements to the software and design has to figure out how to implement the requirements in the context of possibilities and restrictions of the technical environment. I will give a shot overview of design principles that I will take into account when creating the software.

The result of the work is a desktop software written in Python, which one has to install to a computer. The software can be used to help database designers to create a new schema with denormalized base tables (tables in short). User has to choose a set of tables and in case of each relationship between the tables has to answer to a set of questions. The questionnaire is based on Hoberman's algorithm where each answer gives a certain amount of points. In case of each considered relationship the points are summarized. The total number of points suggests as to whether to denormalize database based on this relationship or not. Finally, the software produces a set of SQL data definition language statements in the PostgresSQL SQL dialect. Database developer should look the code, make adjustments in it if needed, and finally has to execute it by himself/herself.

I validate the software by comparing its work result (SQL code) with the result of a denormalization process of a particular database that was done in a previous bachelor thesis (also SQL code). In the end of the work I also evaluate as to whether I achieved all the goals and that are the reasons of unsuccess.

The software is open source and is published with the MIT Licence. The source code is available at: https://github.com/andreasaadli/hoberman-algorithm

The thesis is in Estonian and contains 43 pages of text, 9 chapters, 15 figures, 3 tables.

Lühendite ja mõistete sõnastik

MIT	*Massachusetts Institute of Technology*
MVC	*Model-View-Controller*
MVP	*Model-View-Presenter*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*, ühtne modelleerimiskeel
XML	*Extensible Markup Language*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Kaasaegses maailmas, kus infotehnoloogilisi lahendusi pidevalt igapäevaelus kasutatakse, kogutakse ja salvestatakse aina rohkem andmeid. Kogutud andmed on sellises maailmas oluline vara ja tegevusi käimapanev jõud. Andmete salvestamiseks kasutakse küllaltki sageli andmebaasisüsteemide abil loodud andmebaase, milles võimalikult kiire ja efektiivne operatsioonide läbiviimine on aluseks infotehnoloogia toega infosüsteemide edukale tööle. Sellest tulenevalt otsitakse andmebaasi kasutusvaldkonna mõttes võimalikult otstarbekaid andmebaaside ülesehitamise võimalusi. Üheks andmebaasis toimuvate andmeotsingute kiiremaks tegemise võimaluseks on andmebaasi denormaliseerimine. Andmebaasi denormaliseerimine tähendab andmebaasi andmestruktuuride ümberkorraldamist nii, et ühes andmestruktuuris on koos andmed erinevat tüüpi olemite e reaalse maailma objektide kohta. Selle tulemusena suureneb andmebaasis andmete liiasus, kuid võib paraneda mõningate andmeotsingute e päringute kiirus. Denormaliseerimist abistava tarkvara loomine PostgreSQL andmebaaside jaoks on käesoleva töö sisuks.

Käsitlen töös SQL keele abil loodud andmebaase (edaspidi SQL-andmebaas), sest need on töö kirjutamise ajal endiselt laialt kasutatavad ning 2019. aasta alguse seisuga on andmebaasisüsteemide populaarsuse indeksi [1] esikümnes enamuses andmebaasisüsteemid, mis toetavad SQL keelt (edaspidi SQL-andmebaasisüsteem). Loon tarkvara PostgreSQL jaoks, sest see on populaarne [1], avatud lähtekoodiga (ka minu tarkvara saab olema avatud lähtekoodiga) ning olen seda varem tundma õppinud.

Lähtekoodi refaktoreerimine on protsess olemasoleva lähtekoodi ümberstruktureerimiseks või muul viisil esituse muutmiseks, muutmata koodi poolt saavutatavat tulemust e kasutajale nähtavat käitumist. [2] Lisaks lähtekoodile saab ja peab refaktoreerima [3] ka teisi süsteemiarenduse tulemusena valminud tehiseid nagu näiteks andmebaas ja mudelid. Andmebaasi refaktoreerimine tähendab andmebaasi struktuuri ümberkorraldamist nii, et kogutavate andmete sisu ja andmetega seotud vaadeldav käitumine (kitsendused) ei muutu. Antud töö keskendub sellele, et kasutajal (andmebaasi

arendajal) oleks teatud aspektis võimalik mugavalt andmebaasi struktuuri refaktoreerida. Täpsemalt on töö tulemusena loodava tarkvara abil tehtava refaktoreerimise sisuks SQL-andmebaasi baastabelite (edasipidi tabelite) valikuline denormaliseerimine. Seda, kas ja millal denormaliseerimist läbi viia, aitab otsustada Steve Hobermani koostatud algoritm, mille esituse leiab allikast [4] jaotisest „Denormalization Survival Guide“ (edaspidi Hobermani algoritm). Tegemist oleks mulle teadaolevalt selle algoritmi esimese tarkvaralise realisatsiooniga. Loodav tarkvara peab olema sarnaselt PostgreSQLile avatud lähtekoodiga. Tarkvara peab olema konstrueeritud nii, et sellele oleks hiljem võimalikult lihtne lisada teiste andmebaasisüsteemide (nt Oracle) tuge. Tarkvara kasutajaliidest peab olema võimalik tõlkida erinevatesse keeltesse. Selle kasutamiseks on iga keele kohta eraldi fail, kus kasutajaliidese elemendid on ära tõlgitud [5]. Lisaks peab olema installeerimine kasutajale võimalikult lihtne.

Töö metoodikana kasutatan disainiteadust [6]. Minu eesmärgiks on luua uus tehis – andmebaasi arendajat abistav tarkvara. Selle nõuded ja tehniline lahenduse panen kirja, kasutades kobinatsiooni UML diagrammidest (nt tegevusdiagrammid, klassidiagrammid, kasutusjuhtude diagrammid) ja tekstilistest (nt mittefunktsionaalsed nõuded, kasutusjuhtude kõrgtaseme formaadis kirjeldused) mudelitest. Kasutan töö tegemiseks sisendina teadmisi valdkonnas tehtud tööde kohta [4, 7, 8, 9, 10]. Töö tulemuse valideerimiseks kasutan Susanna Peeki bakalaureusetöö [9] tulemusena ülikooli õppeserveris (apex.ttu.ee) loodud andmebaasi *denormaliseerimise_uurimine*. Tarkvara jaoks kasutan sisendina skeemis *normalized* asuvaid tabeleid ning tarkvara kasutamise lõpptulemusena peaksid tulema sarnased tabelid skeemile *denormalized_by_algorithm*.

Töös annan kõigepealt ülevaate denormaliseerimisest. Seejärel esitan nõuded loodavale tarkvarale. Seejärel kirjeldan ma loodava tarkvara disainimisel järgitavatest põhimõtetest ja tutvustan valminud tarkvara. Töö lõpetab tarkvara produtseeritud väljundi hindamine ning tagasivaade lõputöö projekti õnnestumistele, ebaõnnestumistele ning nende põhjustele.

2 SQL-andmebaasi denormaliseerimine

SQL-andmebaasi normaliseerimine on protsess, kus SQL-andmebaasi baastabeleid e tabeleid restruktureeritakse vastavalt normaalkujudega seotud reeglitele, et vähendada andmete liiasust ja sellest tingitud seotud andmete muutmise anomaaliaid. Igal sammul viiakse tabelid mingile normaalkujule ja hoolitsetakse selle eest, et tabelid vastaksid selle normaalkuju reeglitele. Kui need ei vasta, siis on vaja tabeleid tükeldada. Protsessi tulemusena tabelite hulk kasvab ja igal tabelil on täpsem fookus. Näiteks pole enam nii, et ühes tabelis on nii töötajate iskuandmed kui ka nende müüdavate kaupade andmed, vaid need andmed jagunevad kahe erineva, välisvõtme kaudu seotud tabeli vahel – ühes töötajate ja teises kaupade andmed. Protsessi käigus tükeldatakse tabeleid nii, et nendest saaks vajadusel taastada algsed tabelid, et taastamiseks läheks vaja kõiki tükke ja et nende tükkide põhjal saaks jõustada kõiki kitsendusi, mis algsete tabelite põhjal. SQL-andmebaaside normaliseerimisest annavad näiteks ülevaate allikad [8, 11]. Öeldakse, et SQL-andmebaas on täielikult normaliseeritud kui iga selles olev tabel on vähemalt viiendal normaalkujul. SQL-andmebaasi tabelite viimine kuuendale normaalkujule [12] ei aita küll enam võrreldes viiendal normaalkujul tabelitega vähendada andmete liiasust, kuid on kasulik muudes aspektides nagu andmebaasi skeemi evolutsiooniga toimetulek, andmemuudatuste ajaloo efektiivne säilitmine ja hakkamasaamine puuduvate andmetega.

Kuigi operatiivandmete andmebaasides on soovitav viia kõik tabelid viiendale normaalkujule, leidub ka olukordi, kus kõikide tabelite nii kõrgel normaalkujul olek pole tüüpiliste päringute töökiiruse mõttes pragmaatiliselt parim valik. Seda juhtub sagedamini andmeaitades ja andmevakkades [13] kui operatiivandmete andmebaasides, sest esimestes on valdavad keerukad ja palju andmeid hõlmavad koondandmete leidmise päringud ning pole vajalikud üksikute olemite andmete muutmise operatsioonid. Põhjus on selles, et sageli on andmebaasisüsteemides paraku kontseptuaalsel tasemel elementide (SQL-andmebaasi korral tabelid) ja sisemise taseme elementide (nt failid) vahel üks-ühele vastavus. Seega, kui soovida kiirendada päringu tulemustes andmete kokkuühendamist ja selleks andmebaasi sisemisel tasemel andmeid ühendada, siis ei jäta andmebaasisüsteemid paraku arendajatele muud valikut kui panna kokku ka andmestruktuurid andmebaasi kontseptuaalsel tasemel.

2.1 SQL-andmebaaside denormaliseerimisest üldiselt

Denormaliseerimine on normaliseerimise vastandprotsess, millega SQL-andmebaaside korral vähendatakse tabelite normaliseerituse astet, et parandada andmebaasis tehtavate andmeotsingute e päringute täitmise kiirust. Üheks peamiseks protsessi eesmärgiks on vähendada baastabelite arvu, mida vaja soovitud andmete saamiseks päringusse kaasata, sest tabelite ühendamist sisaldavad päringud on olemasolevates andmebaasisüsteemides mõnikord liiga aeganõudvad. [14]

Andmebaasi denormaliseerimisega võivad kaasneda ka probleemid. Peamised probleemid tekivad andmete liiasusest – andmemahu suurenemine, andmete sisestamise ja muutmise ajakulu, andmete liiasusest tingitud vastuolud andmetes, kitsenduste jõustamise keerukus ning andmebaasi konseptuaalse skeemi halb arusaadavus. [9]

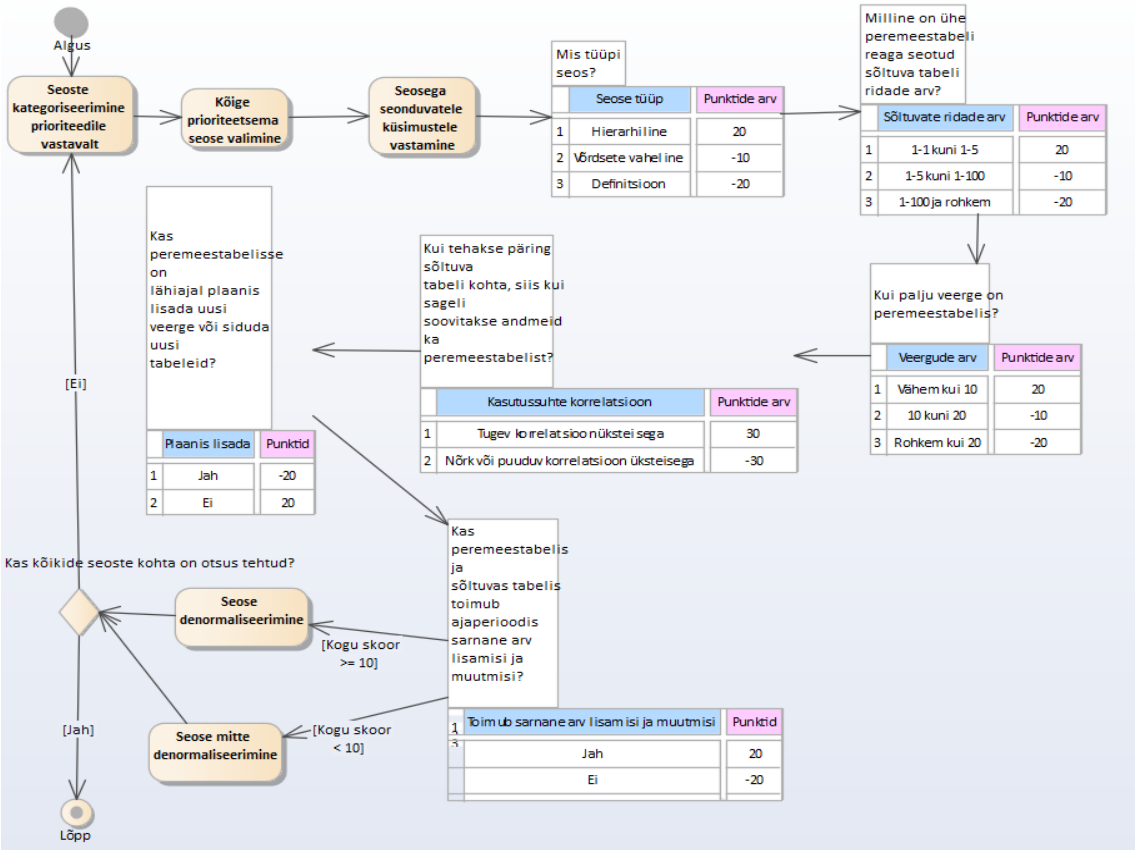
Alternatiiviks baastabelite denormaliseerimisele on denormaliseerimine vaadete tasemel. See tähendab, et baastabelite normaliseerituse astet ei muudeta, kuid päringute lihtsustamise huvides luuakse nendele baastabelitele vaated, kus on rohkem andmete liiasust. Kuna need andmed loetakse automaatselt baastabelitest, siis ei ole selline liiasus probleemiks, sest andmete muutmisel on igal faktil baastabelites oma üks kindel koht kuhu see kirjutada. Samas ei aita vaadete kasutamine parandada päringute töökiirust, mis on sageli denormaliseerimise motivaatoriks [15].

2.2 Hobermani denormaliseerimise algoritm

Tabelite denormaliseerimise üheks võimalikuks algoritmiks on Hobermani algoritm, millest ta kirjutab oma raamatus [4] lehekülgedel 342-361. Algoritmi rakendamise eelduseks on kasutatavate tabelite eelnev viimine viiendale normaalkujule, sest muidu võib algoritmist tulenevate skeemimuudatustega kaasneda halvad disainilahendused. Selle algoritmi põhjal konstrueeritud küsimustiku täitmise tulemusena on võimalik otsustada, millised andmebaasi tabelid tuleks jätta algsele kujule ning millised denormaliseerida. Kõikidel küsimustikus sisalduvatel küsimustel on kindla punktide arvuga vastusevariandid. Joonis 1 kujutataval tegevusdiagrammil on välja toodud algoritmi üldine idee koos küsimustikus olevate küsimustega ning vastusevariantidele vastavate punktide arvuga. Tegevusdiagrammi ning tarkvara realisatsiooni valideerimise

aluseks on võetud Peeki lõputöö [9], lisades sisaldunud algoritmi küsimuste vastused otsustustabelitena.

Algoritmi töö algab tabelite vaheliste seoste e suhete kategoriseerimisega, mille järgi määratakse seoste analüüsi järjekord. Iga uue seose analüüsil tuleb arvestada sellele eelnenud tabelite analüüsi ja denormaliseerimisega kaasnenud tulemustega. [4]



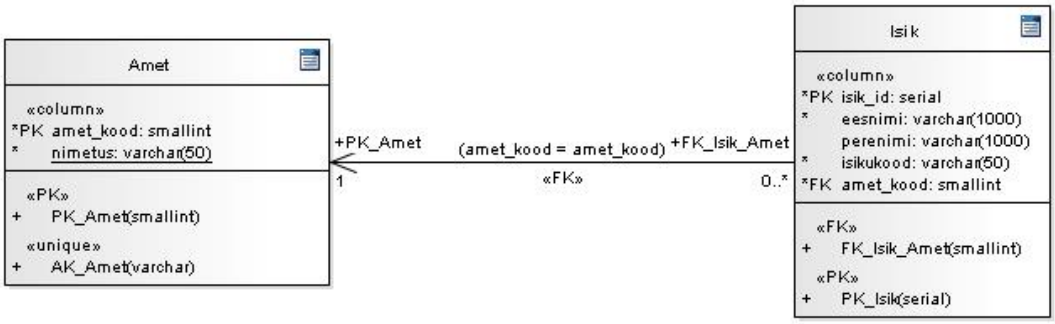
Joonis 1. Denormaliseerimise (Hobermani) algoritmi tegevusdiagramm.

2.3 Võimalikud skeemimuudatused

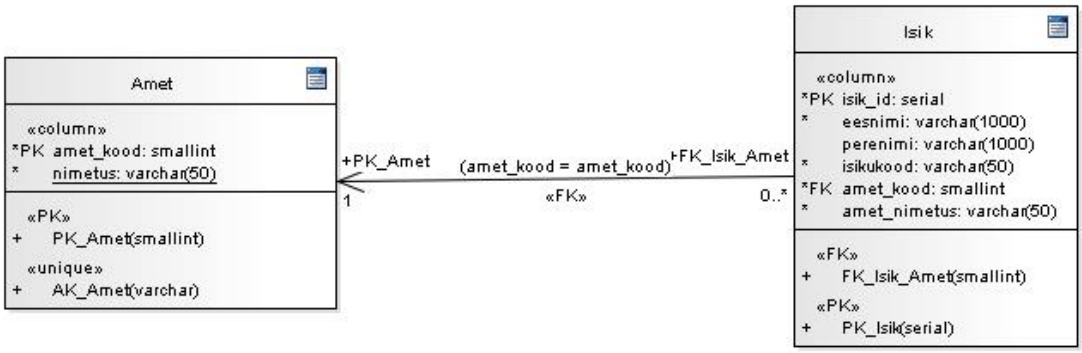
Enne denormaliseerimist tuleks tabelid viia viiendale normaalkujule (Joonis 2, Joonis 4), tagamaks võimalikult hea arusaamise andmete struktuurist ning nägemaks ideaali andmete liiasuse vähendamise mõttes. Denormaliseerimisel muudetakse algoritmi küsimustiku täitmisest lähtuvalt andmebaasi skeemi. Skeemi arusaadavus seejuures halveneb. [8] Näiteks Joonis 3 vaadates tekib küsimus, kas isikule on ette nähtud üks või kaks ametit. Skeemi võimalikeks muutusteks on:

- veergude dubleerimine (Joonis 3),

- tabelite ühendamine (Joonis 5).

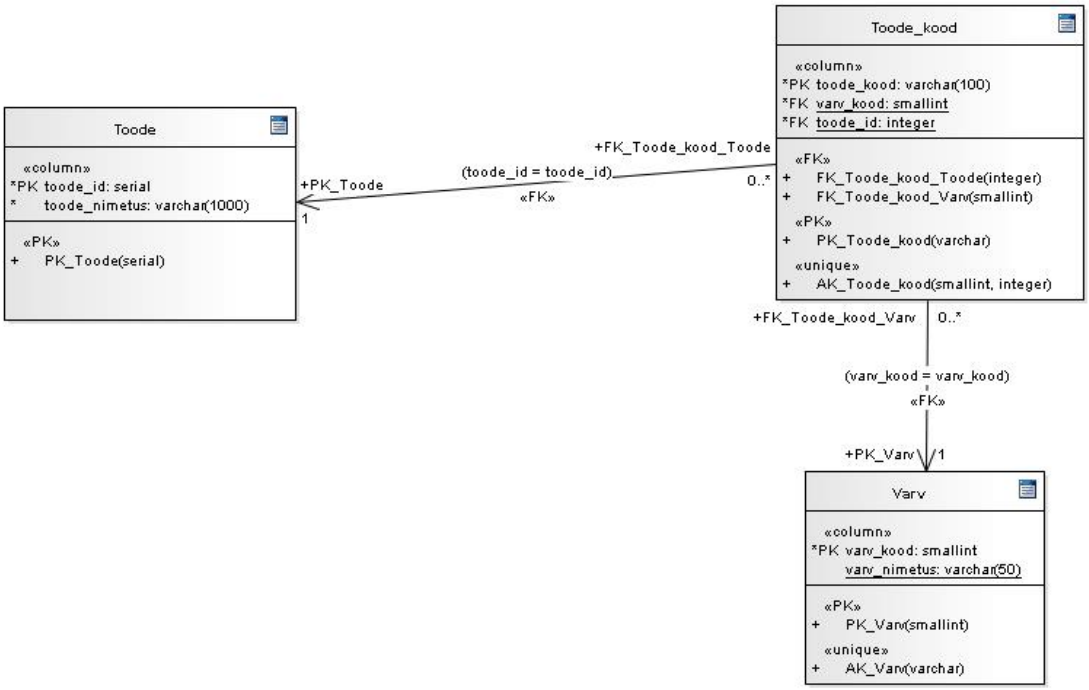


Joonis 2. Kõrge tasemeni normaliseeritud tabelid.

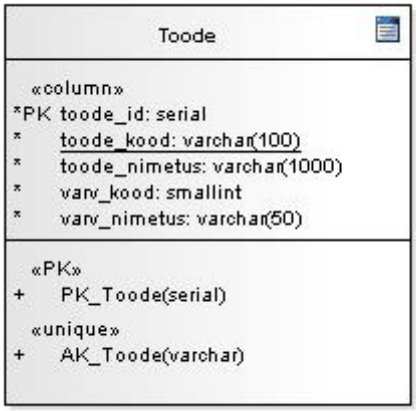


Joonis 3. Denormaliseeritud tabelid (Veergude dubleerimine).

Denormaliseeritud skeemis (Joonis 3) on tabelisse *Isik* dubleeritud veerg *nimetus* tabelist *Amet*. Antud juhul võib denormaliseerida, sest ametinimetused tihti ei muutu ning päringus isiku ametinimetuse küsimiseks pole vajalik tabelite ühendamise (*join*) operatsiooni läbi viia. Denormaliseerimist tasub antud juhul kaaluda, kui päringus küsitakse sageli isikuandmeid koos ametinimetusega.



Joonis 4. Kõrge tasemeni normaliseeritud tabelid.



Joonis 5. Denormaliseeritud tabelid (Tabelite ühendamine).

Denormaliseeritud skeemis (Joonis 5) on kolm algset tabelit – *Toode*, *Toode_kood*, *Varv* – ühendatud kokku tabeliks *Toode*. Antud juhul võib denormaliseerida, sest toodete ja värvide nimetused tihti ei muutu ning nimetuste lugemiseks pole vajalik tabelite ühendamise (*join*) operatsiooni läbi viia. Denormaliseerimist tasub antud juhul kaaluda, kui päringus küsitakse sageli toote andmeid, mis on nendes erinevates tabelites.

3 Seotud tööd

Normaliseerimise ja denormaliseerimise protsess pole iseloomulik ainult relatsioonilistele/SQL-andmebaasidele. Olemas on normaalkujud ja normaliseerimise protsess ka objektorienteeritud [16], XML [17], JSON [18] dokumentide põhistele andmebaasidele. Kuna normaliseerimise pöördprotsess on denormaliseerimine, siis saab ka sellistes andmebaasides läbi viia denormaliseerimist. Antud töös käsitletakse SQL-andmebaaside denormaliseerimist, sest sellised andmebaasid on endiselt väga laialt levinud ning 2019. aasta maikuu seisuga on vaatamata NoSQL süsteemide turuletuleku lainele populaarseimate andmebaasisüsteemide [1] hulgas endiselt enamuses SQL-andmebaasisüsteemid.

Selles peatükis antakse lühiülevaade eelnevatest SQL-andmebaaside denormaliseerumise kohta käivatest töödest, millel antud töö baseerub ning millest võetakse tähelepanekuid andmebaasi refaktoreerimise tarkvara koostamiseks. Eelnevatest töödest saadud tulemusi kasutatakse käesoleva lõputöö käigus valmiva tarkvara valideerimiseks.

3.1 Alternatiivse denormaliseerimise algoritmi kasutus biomeditsiini rakenduses

Suurtes rahvusvahelistes projektides kasutatakse sageli SQL-andmebaase biomeditsiiniliste andmete hoiustamiseks. Nendes andmebaasides on keerulise struktuuriga andmed. Muuhulgas visualiseerimise või masinõppe jaoks on vaja sellistele andmetele paindlikku ja kiiret ligipääsu. Selliste ülesannete paremaks lahendamiseks on tihti vajalik tabeleid denormaliseerida. Sellest tulenevalt kavandati ja programmeeriti Pythoni teek *sqlAutoDenorm.py* PostgreSQL andmebaaside analüüsi ja denormaliseerimise jaoks [7]. Selles realiseeritud algoritm põhineb ideel, et relatsioonilistes/SQL-andmebaasides olevaid andmeid saab esitada graafina [19]. Antud tarkvara on piisavalt kiire, et olla praktiline, kuid on mainitud, et leidub võimalusi optimiseerimiseks kasutades andmebaasi indekseerimist ning muutmälu. Töö [7] annab hea ülevaate denormaliseerimise võimalustest ning reaalsetest kasutusvõimalustest.

3.2 Hobermani algoritmi kasutanud töö

Peeki [9] bakalaureusetöös võrreldi PostgreSQL andmebaasis viienda normaalkujuni normaliseeritud tabelite disaini ning Hobermani algoritmiga denormaliseeritud tabelite disaini erinevate kriteeriumite alusel. Andmebaasid olid täidetud samasuguseid fakte esitavate testandmetega. Töö tulemusena valminud andmebaasi skeeme analüüsiti ning võrreldi. Järeldustest selgus, et denormaliseerimise tulemusena koondandmete päringute keerukus vähenes ja täitmiskiirus paranes. Samas selgus negatiivne mõju andmebaasi andmemahtudele, konkreetse olemi otsimise päringute täitmiskiirusele, andmemuudatuse operatsioonidele ning keerulisemate kitsenduste jõustamisele. Lõpptulemusena väitis autor, et andmebaasi kõrge tasemeni normaliseerimine on operatiivandmete andmebaasides parem lähenemine kui tabelite normaliseerituse astme vähendamine e denormaliseerimine. Siiski järeldus tulemusest ka positiivseid külgi denormaliseerimise kasuks. Käesoleva lõputöö käigus valmiva tarkvara valideerimiseks (vt peatükk 7) kasutatakse selle lõputöö käigus valminud näiteandmebaase.

3.3 Denormaliseerimise strateegiad andmeaitadest andmete otsingu optimeerimiseks

Seoses andmete katkematu kogumisega, andmehulkade kasvuga, erinevate IT süsteemide rohkusega, milles on üksteis dubleerivad või üksteisele vastukäivad faktid ja andmeanalüüsi pideva läbiviimise vajadusega, tekib järjest kasvav vajadus andmeaitade ja andmevakkade järele. Nendes andmebaasides integreeritakse kokku erinevatest allikatest pärit andmed eesmärgiga neid pikemat aega säilitada ning teha nende põhjal keerukaid koondandmete leidmise päringuid. Töö [13] käsitleb nelja denormaliseerimise strateegiat, hindab strateegiate mõjusid andmeotsingute kiirusele ning illustreerib olukordi, kus need strateegiad on kõige efektiivsemad. Järelduseks on, et denormaliseerimise strateegiad võivad parandada andmebaasis toimuvate operatsioonide jõudlust ning analüütilised meetmed pakuvad tõhusaid denormaliseerimisest tingitud muutuste hindamise viise. Kas denormaliseeida või mitte sõltub andmebaasi kasutusotstarbest ja selles põhiliselt läbiviidavatest operatsioonidest.

3.4 Denormaliseerimise positiivsed mõjud analüütilistes ärirakendustes

Hahnke [20] uuring kirjeldab denormaliseerimise positiivseid mõjusid analüütilistes ärirakendustes. Analüütiliste süsteemide disainimisel pakub denormaliseeritud andmemudel tema hinnagul andmebaasi kasutajate jaoks paremat ning intuitiivsemat andmete struktuuri. Denormaliseeritud andmemudeli kaks peamist komponenti, faktid (päringu tulemuses sialduvad andmeelemendid) ning dimensioonid (defineerivad päringute kitsendusi), on samuti denormaliseerimise protsessis kasutusel, et lahendada hierarhilistest suhetest tekkivaid probleeme, mis on olulised ka mitmedimensioonilises analüüsis.

Schkolnick ja Sorenson [21] olid ühed esimesed denormaliseerimise idee väljakäijatest, tehes seda juba aastal 1980. Oma töös [21] toovad nad välja, et parandamaks andmebaasi disaini tulemuslikkust peab disainerile olema nähtav andmemudel, mis kajastab andmetele valdkonnast tulevaid piiranguid e kitsendusi. Nende töö eesmärgiks oli illustreerida, kuidas andmetele kehtivaid kitsendusi võib kasutada denormaliseerimise algoritmi täitmisprotsessis. Käesolevas töös valmiva tarkvara arendamisel arvestatakse samuti, et andmebaasi disaineril on eelnevalt teada vastava andmebaasi valdkonnast tulenevad piirangud ning sellest tulenevalt kasutatakse disaineri tehtud valikuid denormaliseerimise algoritmi rakendamisel.

Hanus [14] arendas välja mitmeid normaliseerimise ja denormaliseerimise liike, mida on võimalik analüütilisetes ärirakendustes kasutada, aga sealjuures soovitas, et denormaliseerimist kasutataks ettevaatlikult ning olenevalt andmete kasutamise viisist.

4 Süsteemianalüüs

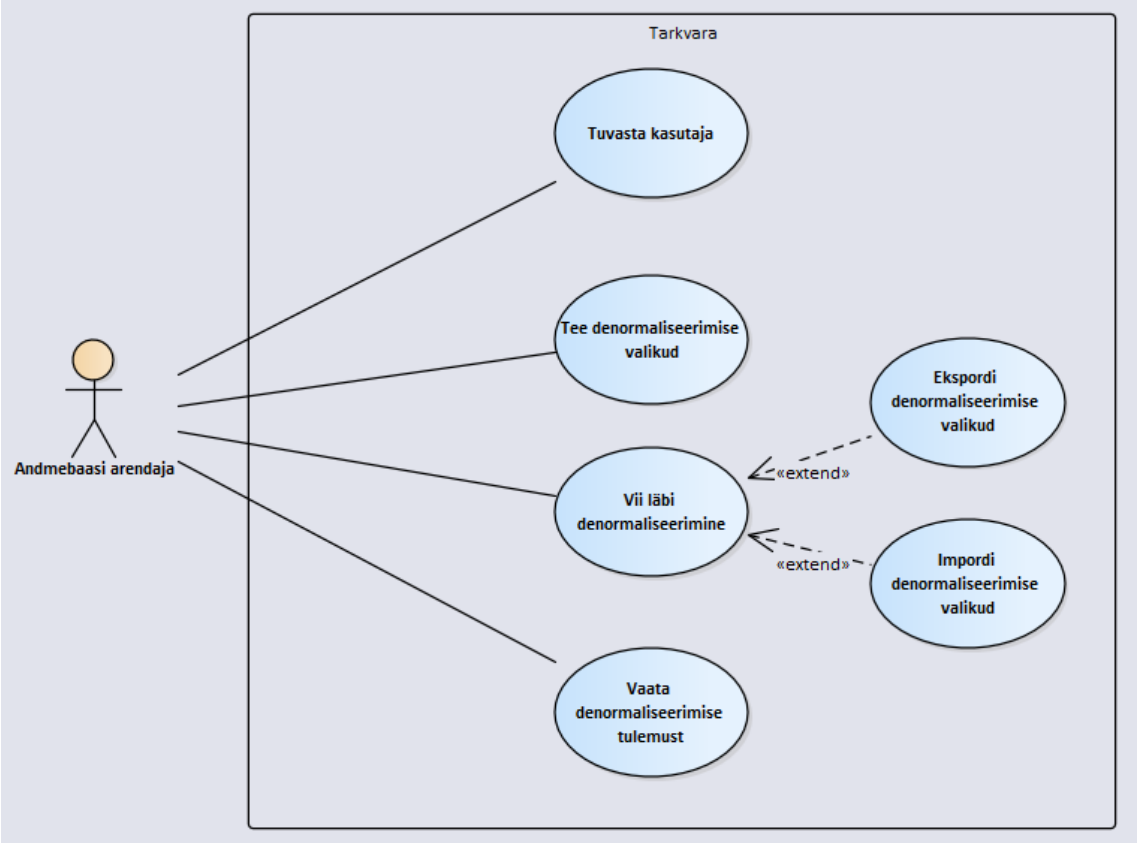
Selles peatükis kirjeldatan nõudeid loodavale tarkvarale.

4.1 Tarkvara eesmärgid

- Automatiseerida PostgreSQL tabelite denormaliseerimist kasutades Hobermani algoritmi.
- Säilitada denormaliseerimise tulemusena algse struktuuriga tabelid ning luua ühtlasi eraldi skeemi uue struktuuriga tabelid.
- Kanda uutesse tabelitesse üle eelnevalt veergudel eksisteerinud kitsendused.
- Toetada kasutajat indeksite loomisel uue struktuuriga tabelitele.
- Kanda andmed automaatselt üle uue struktuuriga tabelitesse.

Nende eesmärkide saavutamiseks vajalik funktsionaalsus on lahti kirjutatud kasutusjuhtude mudelis, mille „sisukorraks“ on kasutusjuhtude diagramm (Joonis 6).

4.2 Kasutusjuhtude mudel



Joonis 6. Kasutusjuhtude diagramm.

Kasutusjuht: Tuvasta kasutaja

Tegutsejad: Andmebaasi arendaja (edaspidi Subjekt)

Kirjeldus: Subjekt identifitseerib ennast. Selleks sisestab ta hosti aadressi, andmebaasi, andmebaasis loodud kasutaja kasutajanime ja parooli. Programm autendib subjekti, st kontrollib subjekti väidetavat identiteeti ning kas tal on õigused vastavat andmebaasi kasutada. Autentimiseks kasutatakse andmebaasi tasemel loodud kasutajaid.

Kasutusjuht: Tee denormaliseerimise valikud

Tegutsejad: Andmebaasi arendaja (edaspidi Subjekt)

Kirjeldus: Subjekt valib andmebaasi ning tabelid, mida soovib denormaliseerida. Seejärel vastab ta iga valikusse kuuluva tabelite vahelise seose korral küsimustikule. Seejuures leiab süsteem osadele küsimustele automaatselt vastused tehes päringuid

andmebaasist – nii denormaliseeritavatest tabelitest kui ka andmebaasi süsteemikataloogist.

Kasutusjuht: Ekspordi denormaliseerimise valikud

Tegutsejad: Andmebaasi disainer (edaspidi Subjekt)

Kirjeldus: Subjektil on võimalik süsteemist eksportida fail, mis sisaldab endas juba denormaliseerimise kohta tehtud valikuid. Eksporditud fail salvestatakse subjekti arvuti kettale.

Kasutusjuht: Impordi denormaliseerimise valikud

Tegutsejad: Andmebaasi arendaja (edaspidi Subjekt)

Kirjeldus: Subjekt saab importida faili eelnevalt salvestatud denormaliseerimise valikutega ning jätkata tööd poolelijäänud kohast.

Kasutusjuht: Vii läbi denormaliseerimine

Tegutsejad: Andmebaasi arendaja (edaspidi Subjekt)

Kirjeldus: Subjekt valib skeemi milles luua uued tabelid. Subjektil on võimalus ka luua uus skeem ning määrata selle nimi. Tulemusena tekitab süsteem uue skeemi koos uue struktuuriga tabelitega ning kannab kasutaja soovi korral vastavad andmed uue struktuuriga tabelitesse üle. Tulemusena valmivas skeemis jõustatakse kõik deklaratiivsed kitsendused, mida saab otse uutele tabelitele üle tõsta. Süsteem pakub subjektile nõuandeid ning tuge indeksite loomisel uutes tabelites. Süsteem pakub lähtetabelite indeksite põhjal välja esialgse indeksite valiku. Subjektil on võimalus luua esialgse indeksite valiku põhjal hiljem oma soovide ja valdkonnast tulenevate nõuetega arvestades ise indeksid.

Kasutusjuht: Vaata denormaliseerimise tulemust

Tegutsejad: Andmebaasi arendaja (edaspidi Subjekt)

Kirjeldus: Subjekt saab peale denormaliseerimise protsessi vaadata üle tekkinud skeemi ning seal sisalduvad tabelid. Skeemist ülevaate saamiseks on programmi poolt loodud fail, kuhu on denormaliseerimise tulemusena valminud kood salvestatud. Subjektil on võimalik saada ülevaade tekkinud skeemist PostgreSQL koodi kujul.

4.3 Mittefunktsionaalsed nõuded

Tabel 1 esitab tarkvara mittefunktsionaalsed nõuded.

Tabel 1 Mittefunktsionaalsed nõuded tarkvarale.

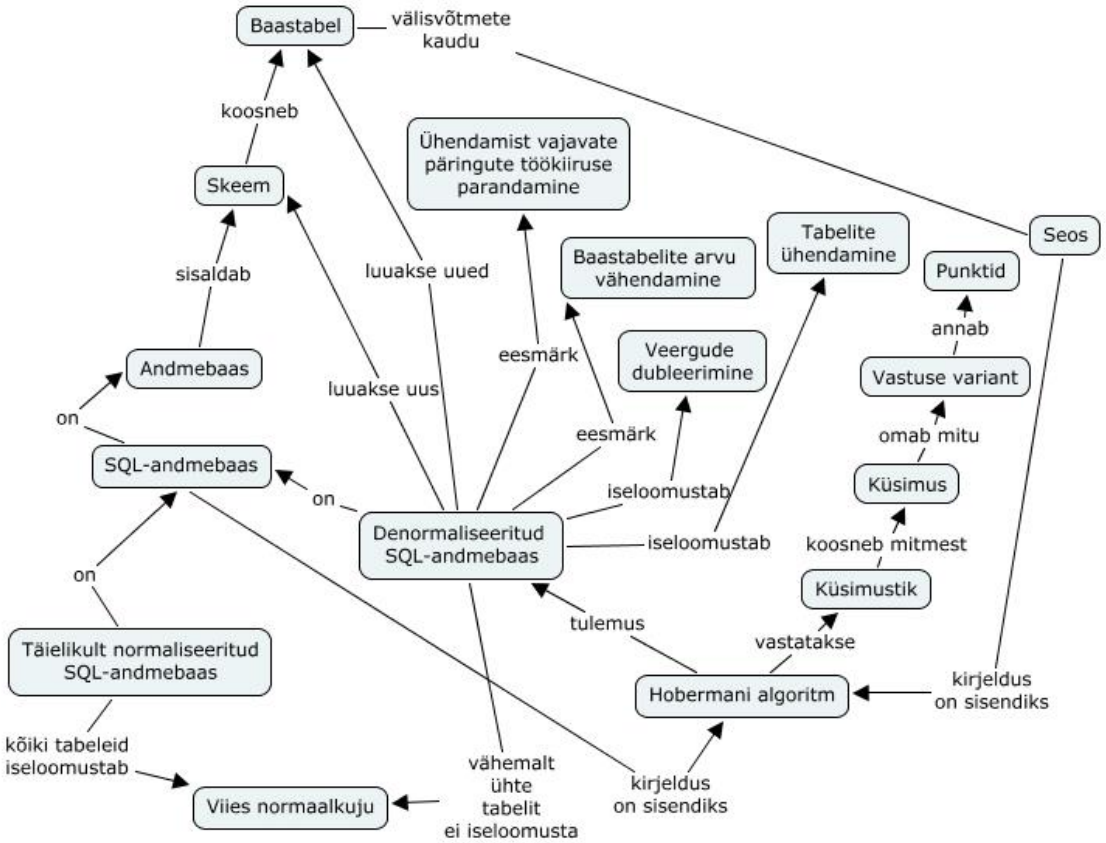
Tüüp	***Nõude kirjeldus***
Arendusvahendid	Arendusvahendina tuleks kasutada objektorienteeritud programmeerimiskeelt, tulenevalt sellise programmeerimise paradigma laiast kasutatavuset ning võimalustest, mida see pakub koodi taaskasutamiseks
Keel	Tarkvara kasutajaliides peab olema inglise keeles. Tarkvara dokumentatsioon võib olla eestikeeles, kuid tarkvara avaldamisel tuleb avaldada ka selle ingliskeelne lühiülevaade.
Laiendatavus	Tarkvara peab olema konstrueeritud nii, et sellele oleks hiljem võimalikult lihtne lisada teiste andmebaasisüsteemide (nt Oracle) tuge. Tarkvara tuleks üles ehitada nii, et oleks lihtne lisada kasutajaliidesesse uusi keeli (nt eesti keel). Selle kasutamiseks saavutamiseks onolema iga keele kohta eraldi fail, kus kasutajaliidese elemendid on ära tõlgitud [5].
Kasutajaliides	Rakendus on kahekihiline, kus kasutaja arvutis on rakendus ning see suhtleb üle arvutivõrgu serveril paikneva andmebaasisüsteemiga. Nõuded kasutajaliidese ülesehitusele.

Tüüp	*Nõude kirjeldus*
	• Ülesehituse põhimõtteid tuleb järjekindlalt järgida. • Rakenduses kasutada kõikide vaadete jaoks kindla stiiliga vormi • Failide salvestamiseks tuleb kasutada hüpikaknaid. • Parooli sisestusväli ei tohi kuvada parooli • Andmete lugemiseks ning andmete muutmiseks mõeldud väljad peavad erinevalt välja nägema (nt olema erineva taustavärviga). • Tegevused, mida süsteem saab ise teha, peab tegema süsteem ilma kasutajalt tagasiside küsimisega tülitamata. • Kõikides olemite nimekirjades tuleb esitada selline hulk andmeid, et nende andmete alusel oleks võimalik olemeid üksteisest üheselt eristada ning et need andmed oleksid konkreetse kasutaja jaoks mõistetavad ja sisukad.
Paigaldamine	Installeerimine peab olema kasutajale võimalikult lihtne.
Turvalisus	Andmebaasi kasutamine toimub andmebaasis loodud kasutajakonto alt. Andmebaasi denormaliseerijal peavad olema kõik vajalikud õigused, viimaks läbi denormaliseerimise protsessi jaoks ettenähtud tegevusi. Programmi töö käigus koostatakse dünaamiliselt SQL lauseid. See protsess ei tohi olla rünnatav SQL süstimise tehnika abil (pahatahtlikul osapoolel ei tohi olla võimalik programmi käivitatavate SQL lausete hulka soovimatul viisil muuta selle kaudu, et ta annab mingil kavalal viisil väärtuseid programmi juhtparameetritele või kasutab andmebaasis nimesid, mis sunnivad programmi täitma mingeid täiendavaid SQL lauseid).

Tüüp	*Nõude kirjeldus*
Avaldamine	Loodav tarkvara peab olema sarnaselt PostgreSQLile avatud lähtekoodiga.
Töökindlus	Rakendus ei tohi tekitada andmebaasisüsteemile liigset koormust ega tõrkeid.
Varukoopiad	Rakenduses peab olema võimalus salvestada rakenduse kasutamisest tulenevad andmed faili.

4.4 Valdkonnamudel

Joonis 7 kujutatakse Hobermani algoritmi põhjal realiseeritava denormaliseerimise protsessi kohta käivaid mõisteid ja nende mõistete seoseid mõistekaartina.



Joonis 7 Denormaliseerimise valdkonnamudel

5 Disain

Tarkvara disainimisel ning programmeerimisel arvestasin puhta koodi [3] [22] põhimõtetega ning loomise vahendite valikul lähtusin sellest, et arendamiseks kasutatav tarkvara oleks ajakohane ning vaba tarkvara.

5.1 Kasutatavad vahendid ja valiku põhjendus

Kõigepealt uurisin, kas ülesande lahendamiseks võiks täiendada mõnda olemasolevat andmebaaside refaktoreerimise tarkvara. Selleks tegin taustauuringu Googles. Kasutasin otsingu tegemiseks märksõnu: *database denormalization software, database refactoring software, database refactoring tools.* Ma ei leidnud ühtegi sobivat tarkvara.

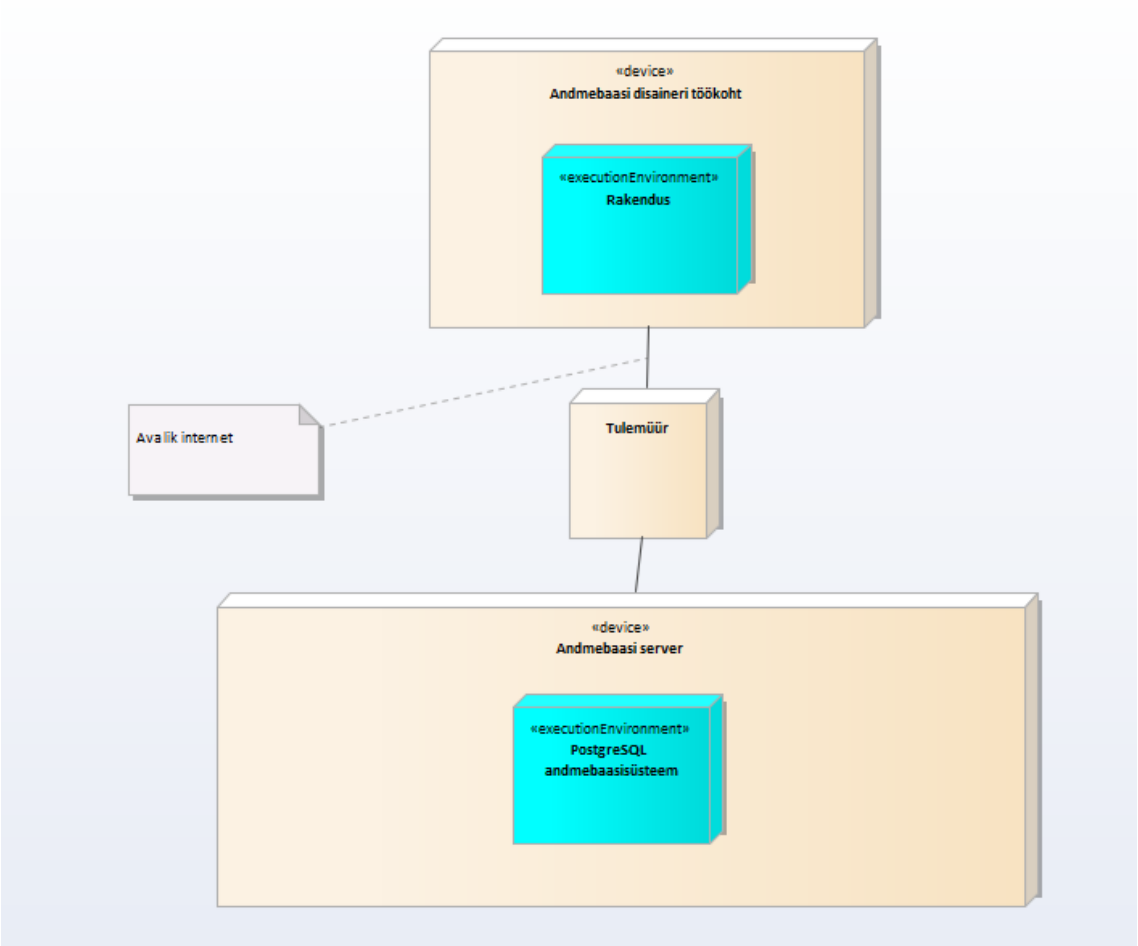
Üheks mittefunktsionaalseks nõudeks on kasutada objektorienteeritud programmeerimiskeelt. Valisin programmeerimiskeeleks Pythoni [23] tulenevalt selle populaarsusest [24], aga ka oma varasemast kokkupuutest selle keelega.

Pythoni vahendite taustauuringut tehes ei leidnud ma laiemas kasutuses olevaid andmebaaside refaktoreerimise vahendeid.

Sobiva Pythoni kasutajaliidese raamistiku leidmise jaoks kasutasin artiklit [25], kus on mitmete raamistike kohta kirjeldus. Valisin PyQt, mis on C++-s kirjutatud mitmeplatvormiline raamistik. PyQt [26] valisin pika ajaloo pärast ning sellepärast, et seda on võimalik kasutada tasuta, kui realiseeritav rakendus on vabavara.

5.2 Tehniline arhitektuur

Realiseerisin programmi andmebaasi disaineri töökohal oleva rakendusena, mis andmebaasisüsteemi sisse logimiseks võtab üle avaliku interneti ühendust andmebaasisüsteemiga (Joonis 8).



Joonis 8. Rakenduse tehniline arhitektuur

5.3 Tarkvara disain

Tarkvara disainimisel lähtusin MVC mudelist, mida kasutatakse enamike objektorienteeritud rakenduste realiseerimisel.

5.3.1 Kasutatud tarkvara disainimustrid

MVC ehk Model-View-Controller [27] on rakenduse disainimuster, kus kõikidel komponentidel on oma eesmärk ning nad on omavahel ühendatud vastavate eesmärkide täitmiseks. Model–View–Presenter (MVP) [28] on MVC üks alamliike, mida kasutatakse peamiselt kasutajaliideste ehitamisel, kus kasutajaliidese poolt tulenevad operatsioonid edastatakse *View* komponendilt *Presenter* komponendile, aga *View* komponent saab ka osad andmed *Model* komponendilt kui seal peaks andmetes muudatus toimuma.

5.3.2 Tarkvara olulise protsessi kirjeldus

Tarkvara kasutates kulub enamus ajast küsimustikule vastamiseks ning tarkvara salvestab tulemusi ning teeb ka päringuid automaatselt küsimustele vastuste leidmiseks andmebaasist (vt mittefunktsionaalne nõue „Tegevused, mida süsteem saab ise teha, peab tegema süsteem ilma kasutajalt tagasiside küsimisega tülitamata.“). Selle jaoks salvestab programm peale järgmisele lehele liikumise nupu vajutamist kasutajaliidesest andmebaasi disaineri poolt tehtud küsimuse vastuse valiku ning muud vajalikud andmed. Edasi salvestab programm need vastavasse objekti või siis teeb veel andmebaasi suunas vajalikud päringud. Edasi salvestatakse tulemus *Model* komponendi attribuudina ning tehakse vajalikud arvutused skoori uuendamiseks. Skoori uuendus kuvatakse kasutajale järgneval lehel. Ning protsess kordub kuni kõikidele küsimustele on vastava suhte kohta vastatud.

5.3.3 Kasutajaliidese disaini põhimõtted

Kasutajaliidese disainil lähtun sellest, et kasutajale oleks kõigi funktsioonide kasutamine arusaadav ning intuitiivne. Andmete mõistmine ja järgneva sammu tegemine tuleb kasutajale teha võimalikult lihtsaks ja vältida frustratsiooni tekitamist programmi kasutajale.

Üritan kasutaja sisestatud andmete kogumisel ning nende edasikandmisel teistesse programmi osadesse järgida kasutajaliidese disainimustreid (nagu näiteks *kindlad kasutajaliidese vormid, andmete õigsuse kontroll*) [29] .

Minu eesmärgiks on järgida puhta koodi põhimõtteid [3], näiteks muutujate eesmärgist tuleneva nimetamise ja korduva koodi vältimise kaudu. Nende järgmisel on ka hilisem koodi muutmine ning koodist arusaamine vähem aeganõudev ja lihtsam. Puhta koodi [22] vastand on halbade lõhnadega kood, mis tuleneb liigsest kiirustamisest ja vähesest koodi refaktoreerimisest.

5.4 Testimine

Testi rakendust andmebaasi disaineri kasutusviisi järgi erinevaid stsenaariume korduvalt läbi mängides ning lõpptulemusena genereeritava SQL-koodi hindamise läbi.

6 Tarkvara

Programmi installimiseks ning edasiseks kasutamiseks on vajalik, et kasutajal oleks eelnevalt installitud arvutisse Python. Kasutajal on võimalik kasutada programmi nii inglise kui eesti keeles ning tal on võimalus lisada omale sobivaid keeli, tõlkides kasutajaliidese elemendid JSON formaadis olevasse faili ning lisades selle tarkvara ’/resources/languages’ kausta (Joonis 9).

```
{
  "EST": {
    "toolbar": {
      "file": "Fail",
      "options": "Seaded",
      "info": "Info",
      "import": "Impordi",
      "export": "Ekspordi",
      "language": "Keel",
      "about": "Teave"
    },
```

Joonis 9. Fragment kasutajaliidese tõlget sisaldavast failist.

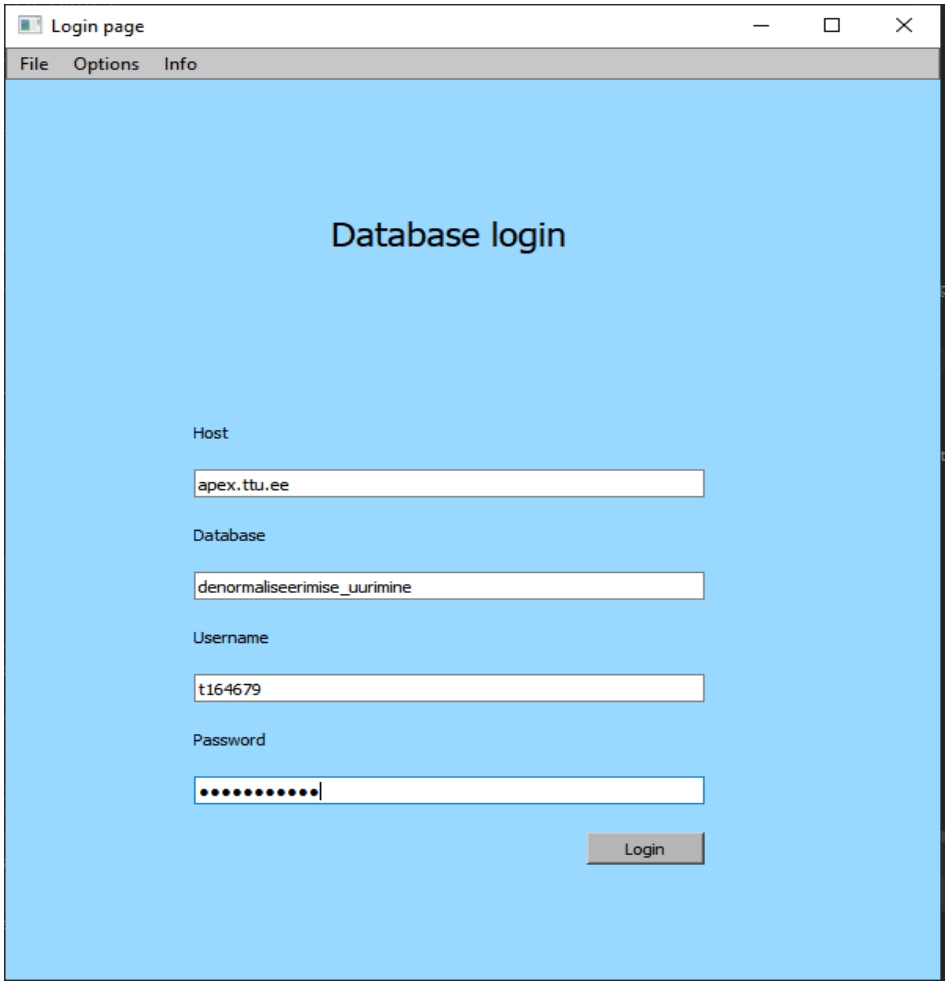
Tarkvara on litsentseeritud MIT litsentsiga [30]. Valisin selle litsentsi, sest see on üks avatud lähtekoodiga projektides kasutatavatest litsentsidest, mis annab edasisele arendajale võimalikult palju vabadust otsuste tegemisel. Nende vabaduste alla kuulub näiteks hilisem juurdearenduse mitteavalikustamine või see, et juuredearendust võib hiljem väljastada muu litsentsi alt. MIT litsentsi kasutavad populaarsed projektid nagu näiteks React.js, Bootstrap ja Angular.js.

Tarkvara on kättesaadav GitHubis: https://github.com/andreasaadli/hoberman-algorithm

6.1 Tarkvara kasutamine

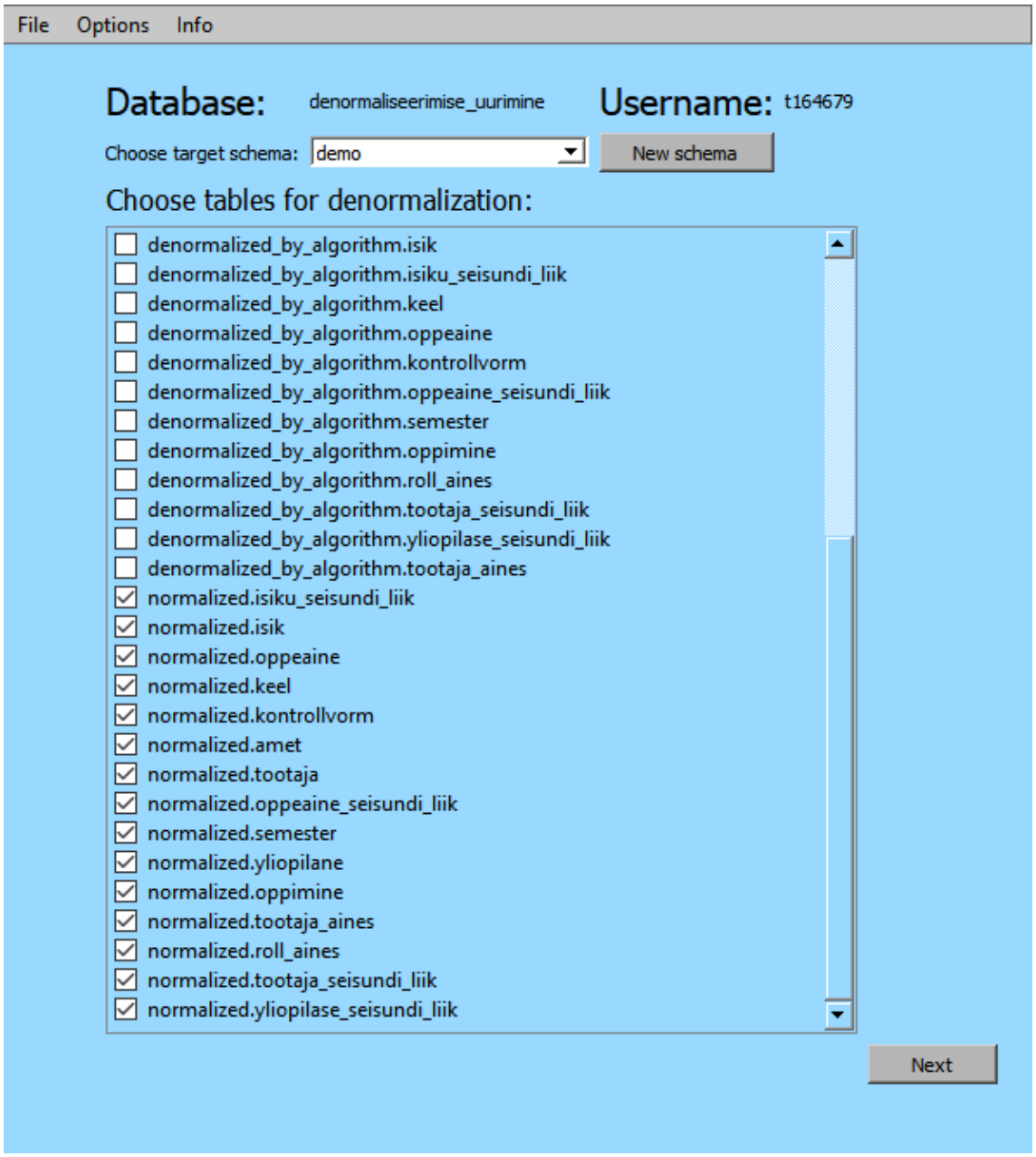
Tarkvara käivitamisel kuvatakse kasutajale sisselogimise leht (Joonis 10), kuhu peab programmi edasiseks kasutamiseks sisestama PostgreSQL andmebaasi serveri (hosti) aadressi, kasutatava andmebaasi nime ning kasutajanime ja parooli. Tarkvarasse logitakse sisse andmebaasi tasemel (PostgreSQLis CREATE USER lausega) loodud kasutajana

ning sellel peavad olema kõik vajalikud õigused andmete nägemiseks ning funktsioonide käivitamiseks.



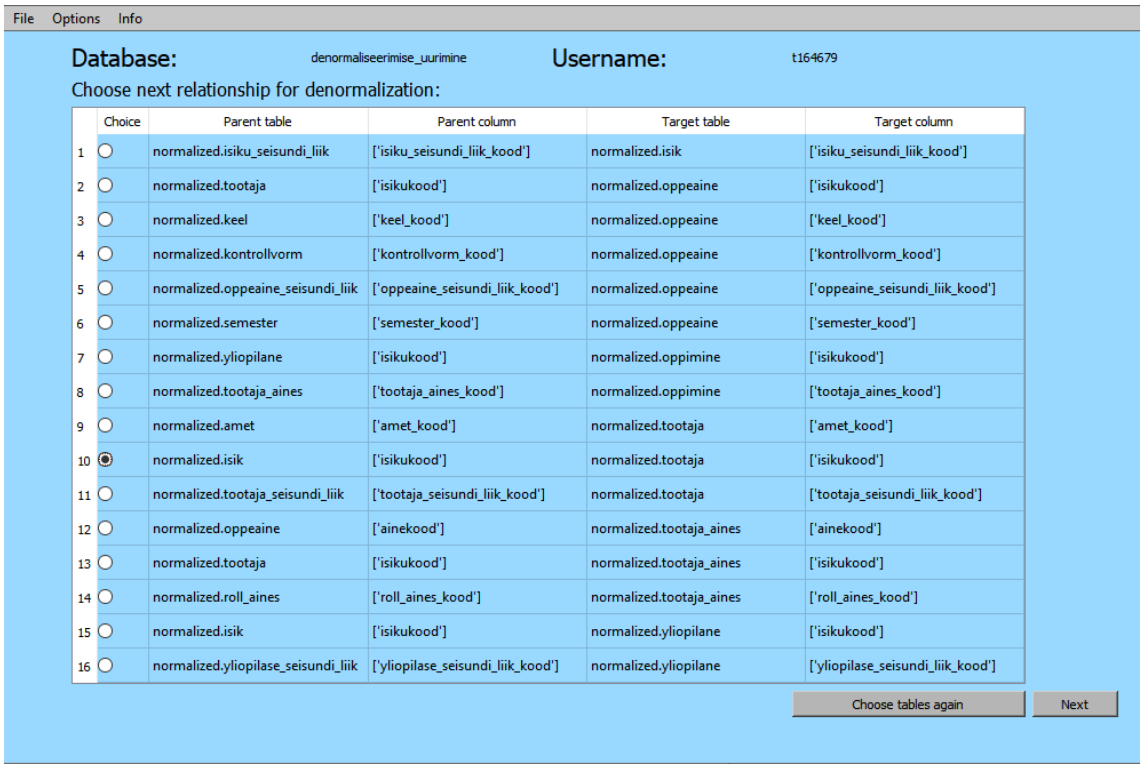
Joonis 10. Sisselogimise kasutajaliides.

Esmalt valitakse tarkvara kasutajaliidese kaudu skeem, kuhu denormaliseeritavad tabelid lisatakse (andmebaaside kuuluvate skeemide hulgast või sisestatakse ise uue nimega skeem) ning denormaliseerimise kandidaatideks olevad tabelid (Joonis 11). Kui sisestan olemasoleva skeemi nime, siis korduvat skeemi nime ei teki. Tabelite valikul näidatakse nii skeemi nime kui tabeli nime, sest erinevates skeemides võib olla sama nimega tabeleid (vt mittefunktsionaalne nõue „Kõikides olemite nimekirjades tuleb esitada selline hulk andmeid, et nende andmete alusel oleks võimalik olemeid üksteisest üheselt eristada.“).



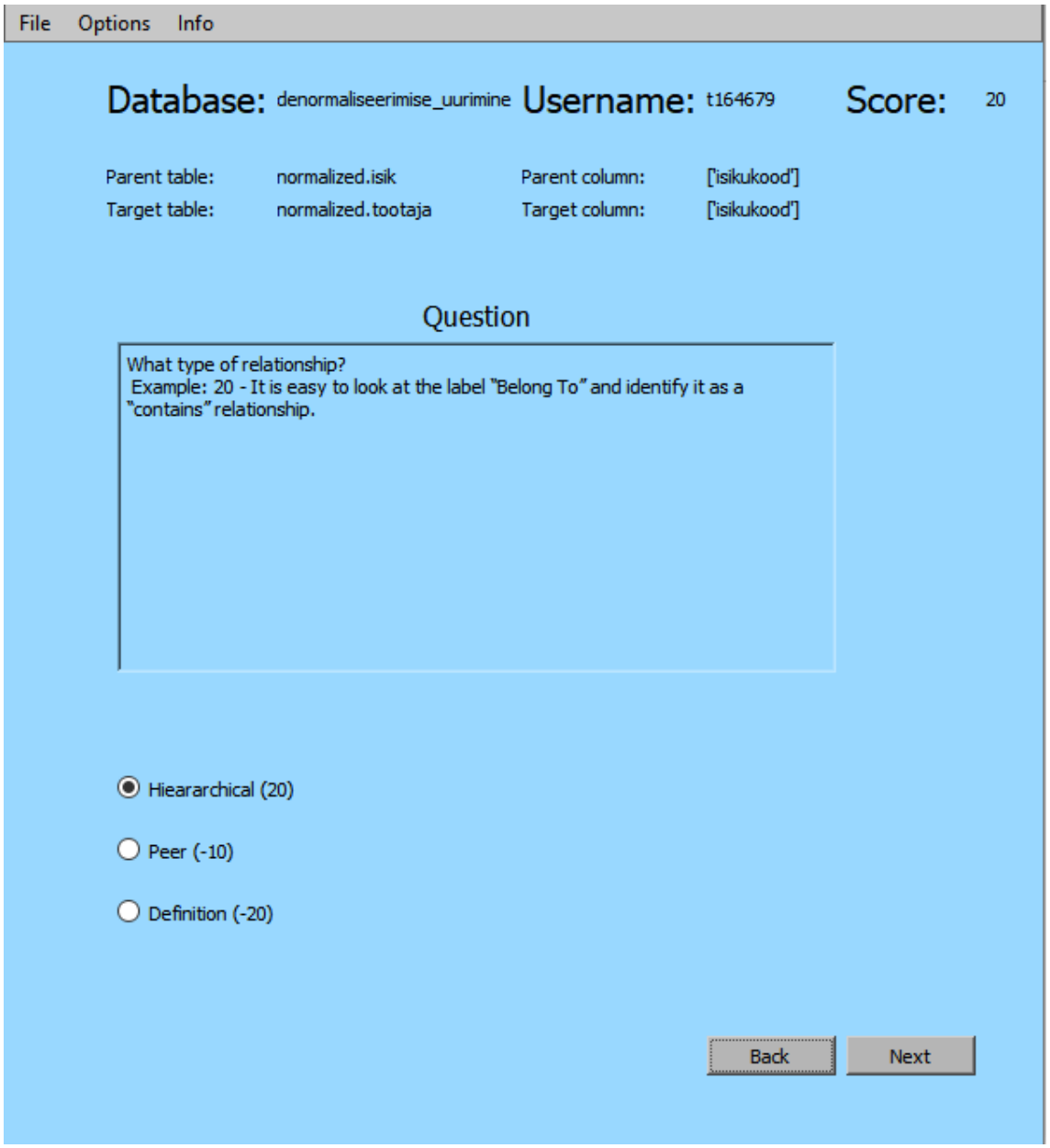
Joonis 11. Tabelite valiku kasutajaliides.

Seejärel valitakse tabelite vaheliste seoste hulgast järgmine denormaliseeritav suhe (Joonis 12). Seoste prioriteetsus pole tarkvara poolt määratud, kuid Hobermani algoritm näeb ette, et kõigepealt denormaliseeritakse üks-üks seosed, seejärel klassifikaatorite ja põhiandmete tabelite omavahelised seosed, siis transaktsiooniliste andmetega tabelite omavahelised seosed ning lõpuks klassifikaatorite või põhiandmete tabelite ning transaktsiooniliste andmetega tabelite vahelised seosed.



Joonis 12. Järgneva suhte valiku kasutajaliides.

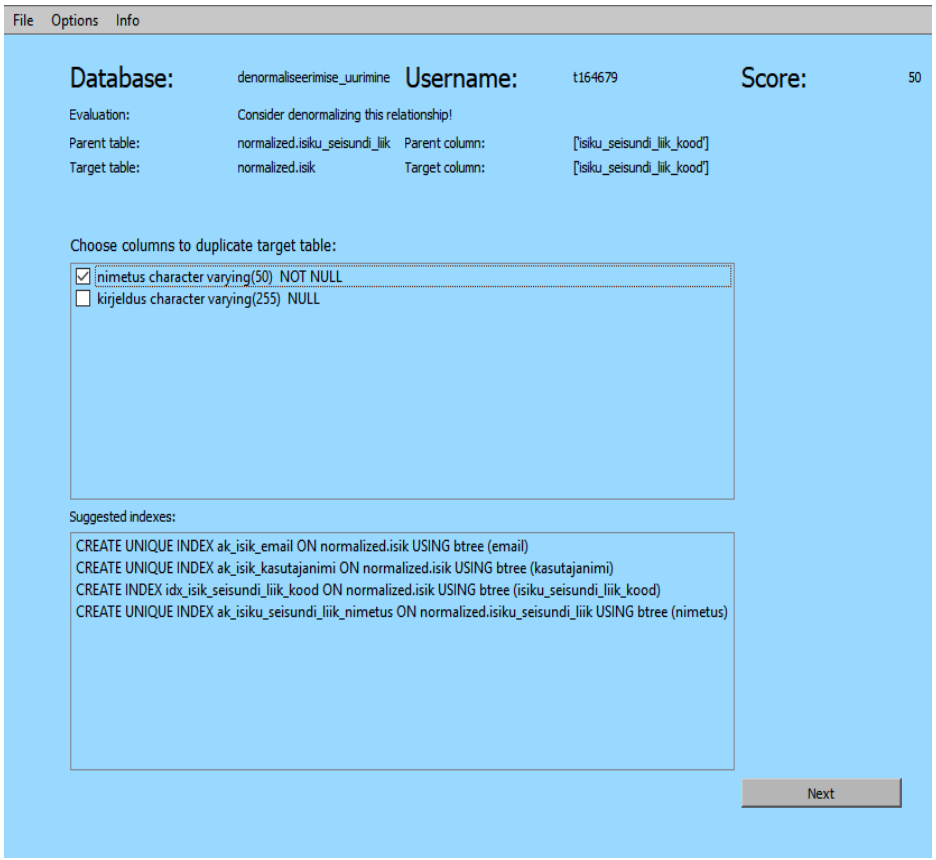
Järgnevalt vastatakse iga valikusse kuuluva tabelite vahelise seose korral küsimustikule (Joonis 13). Seejuures saab osadele küsimustele süsteem leida vastuse automaatselt tehes päringuid andmebaasist – nii denormaliseeritavatest tabelitest kui ka andmebaasi süsteemikataloogist e andmebaasisüsteemi poolt andmebaasi kohta peetavast andmebaasist. Tarkvara kuvab kasutajale hetkel denormaliseeritavat suhe, hetke skoori suhte kohta ning küsimust vastusevariantidega. Kui seose kohta küsimustikule vastamise lõpptulemus on madalam kui 10 punkti, siis seoses osalevad tabelid jäetakse algsele kujule.



Joonis 13. Küsimustiku kasutajaliides.

Kui kõigi valitud seoste kohta on küsimustikule vastatud, siis lõpptulemusena tekitab tarkvara SQL koodi (vt Lisa 2), mis on mõeldud uue struktuuriga tabelite loomiseks valitud skeemis. Töö tulemusena loodud tabelite loomise lausetes kehtestatakse kõik deklaratiivsed kitsendused, mida saab otse uutele tabelitele üle tõsta. Tarkvara pakub kasutajale soovitusi indeksite valikul ning laseb kasutajal teha veergude dubleerimise valikuid. Kasutajal on endal hilisem valikuvõimalus indeksite loomisel uutes tabelites, sest denormaliseeritavas andmebaasis on protsessi keskel raske hinnata, mis indekseid hiljem reaalselt tarvis on, kuna lõpptulemusena genereeritav skeem võib veel mitu korda

muutuda. Samuti luuakse indeksid tavaliselt töökiiruse huvides alles peale andmete tabelitesse ülekandmist.



Joonis 14. Hinnangu ning valikute kuvamise kasutajaliides.

6.2 Denormaliseeritud koodi genereerimine

Tarkvara sulgemine tekitab kasutaja arvutisse denormaliseeritud tabelite loomise koodi nende tabelite kohta, mille puhul kasutaja vastas küsimustikule. Koodi genereerimisel kasutatakse *TableResult* klassi, mille objektidesse on salvestatud iga tabeli kohta veergude, kitsenduste ja seoste andmed. Igal objektil on olemas atribuut, mille väärtus sisaldab selle objektiga seonduvaid küsimustiku tulemusi, mille põhjal on antud hinnang tabeli denormaliseerimise kohta ning määratud kindlaks denormaliseerimise liik. Lõpptulemusena genereeritakse iga tabeli kohta eraldi CREATE TABLE lause, mis siis kokku ühendatakse, faili salvestatakse ning sellega kasutajale kättesaadavaks muudetakse. Kuna PostgreSQL võimaldab koondada transaktsiooni andmekirjelduskeele lauseid, siis on genereeritud kood ühes transaktsiooni plokis (START TRANSACTION; ja COMMIT; lausete vahel). Koodi skriptina käivitamisel tagab see, et kõik andmebaasi struktuuri muudatused tehakse kas täielikult või jäetakse täielikult tegemata.

6.3 Tarkvara edasised arendused

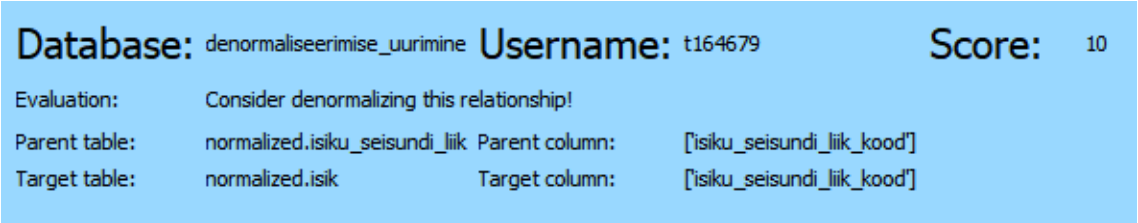
Üks edasiarendus oleks luua tarkvarale seoste prioriteedi kuvamine vastavalt seoste tüübile ning sorteerida enne denormaliseerimise aluseks oleva seose valimist need prioriteedi järgi.

Samuti oleks võimalik arendada peale skeemi loomist ning andmete ülekandmist veel indeksite automaatne loomine.

7 Eksperiment

Eksperiment baseerub Peeki lõputöö [9] käigus valminud denormaliseeritud skeemi loomise SQL-koodi ning käesoleva lõputöö käigus valminud tarkvara poolt genereeritud SQL-koodi võrdlusel. Eksperimendiks ühendun programmi abil andmebaasiga *denormaliseerimise_uurimine*, mis asub serveris apex.ttu.ee ning valin küsimustikule vastamiseks kõik *normalized* skeemi tabelid. Iga seose kohta vastan küsimustele samamoodi nagu on küsimustikule vastatud eelmainitud lõputöö lisades.

Joonis 15 esitab küsimustikule vastamise tulemuse klassifikaatorseose (isiku_seisundi_liik – isik) kohta. Tulemus on sama nagu Peeki lõputöö lisas arvutatud tulemus. Sellest võib järeldada, et vähemalt selle seose kohta on tarkvara poolt tehtud skoori lõpptulemuse arvutus õige.



Joonis 15. Tulemus seose kohta.

Eksperimendi lõpptulemusena võrdlen minu programmi poolt genereeritavat koodi (Lisa 2) Peeki lõputöö [9] lisade hulgas (Lisa 1) oleva denormaliseeritud andmebaasi loomise skriptiga.

Esimene erinevus on domeenide loomine, mida antud programm ei võimalda ning need tuleks andmebaasi disaineril ise koodi üle vaadates luua. Samuti on näha ka erinevus tabelite loomise lausete järjestuses. Programmi poolt genereeritud kood ei arvesta välisvõtme deklaratsioonidega ning seega tuleb praegu teha arendajal lausete järjekorras käsitsi muudatusi.

Positiivse aspektina võib välja tuua, et kõik vastavate tabelite veerud on dubleeritud nii nagu programmis said valikud tehtud ja nendele deklareeritud kitsendused on samuti dubleeritud veergudega uues tabelis kaasas. Sama kehtib ka tabelite ühendamise alla kuuluvate juhtumite korral.

Minu programm genereerib välisvõtme kitsenduste deklaratsioone CREATE TABLE lausete osana. Võrdluseks kasutatud lõputöös lisati välisvõtme kitsendused tabelitesse ALTER TABLE lausetega. Minu programmi väljundi eelis on, et info tabeli kohta on lähtekoodis ühes kohas koos. Samas on puuduseks, et tabelite loomise laused peavad käivitamisel olema välisvõtme kitsendustega määratud järjekorras. Kui lauseid koostav programm ei oska ise seda järjekorda määrata, siis on seda vaja teha programmi genereeritud koodi kasutajal.

Minu programmi väljundi puhul on negatiivne ka see, et andmebaasi disaineril pole kohe võimalik koodi andmebaasis käivitada. Tal on vaja teha parandusi, et kood oleks reaalselt käivitatav. Paranduse sisuks on näiteks tabelite loomise järjekorra muutmine.

8 Tagasivaade tehtud tööle

Iga projekti juures on oluline vaadata tagasi sellele, mis sai tehtud ning õppida vigadest, et teha järgmine kord paremini. Tabel 2 esitab hinnangu funtsionaalsete nõuete (vt jaotis 4.2) täidetusele loodud tarkvaras. Tabel 3 esitab hinnangu mittefuntsionaalsete nõuete (vt jaotis 4.3) täidetusele loodud tarkvaras. Kasutan mõlemas tabelis realiseerimise hinnangute andmiseks järgmist tähistust:

- + : täielikult realiseeritud,
- o : osaliselt realiseeritud,
- - : realiseerimata

Tabel 2 Hinnang funktsionaalsete nõuete täidetusele.

Kasutusjuht	Realiseerimise hinnang	Kommentaar
Tuvasta kasutaja	+	
Tee denormaliseerimise valikud	+	
Vii läbi denormaliseerimine	o	Programmi töö tulemusena tekib hulk SQL lauseid, mida peab ülevaatama ja võimalik, et parandama. Ei ole võimalik anda korraldust teha muudatus andmebaasis. Samuti ei genereerita lauseid andmete ülekandmiseks uue struktuuriga tabelitesse.
Ekspordi denormaliseerimise valikud	-	Käesoleval hetkel pole võimalik kasutaja poolt tehtud valikuid JSON-kujul eksportida ega importida. See on oluline funktsionaalsus, sest

Kasutusjuht	Realiseerimise hinnang	Kommentaar
Impordi denormaliseerimise valikud	-	paljude reaalses elus kasutatavate andmebaaside tabelite arv on nii suur, et selle denormaliseerimise protsess võtab kaua aega ja seda ei tehta ühe korraga. Lisaks oleks siis protsessi võimalik peatada ning jätkata endale sobival ajal ja kohas.
Vaata denormaliseerimise tulemust	+	

Tabel 3 Hinnang mittefunktsionaalsete nõuete täidetusele.

Mittefunktsionaalne nõue	Realiseerimise hinnang	Kommentaar
Arendusvahendid	+	
Keel	+	
Laiendatavus	+	
Kasutajaliides	+	
Paigaldamine	+	Installeerimiseks on loodud eraldi programm (setup.exe). Samuti on kasutajal võimalus programmi deinstalleerimiseks.
Turvalisus	o	Kasutaja antud sisend on peaasjalikult märkeruutude või liitbokside kaudu. Vabatekstina saab kasutaja sisestada skeemi nime. Loodud koodi hetkel andmebaasis automaatselt ei käivitata.

Mittefunktsionaalne nõue	Realiseerimise hinnang	Kommentaar
Avaldamine	+	
Töökindlus	+	
Varukoopiad	-	Tehtud tööd ei saa paraku pooleli jätta seda salvestades, selleks et kasutada salvestatud faili poolelijäänud kohast jätkamiseks.

Üheks põhjuseks, miks ma soovitud süsteemi täies mahus valmis ei saanud, on hilisest alustamisest tulenenud ajapuudus.

Töö läbiviimisel ei kasutanud ma iteratiivset arendust ning arendusprotsess meenutas pigem koskstiilis arendust. Kõigepealt kogusin kokku kõik nõuded ja siis asusin nende alusel programmi realiseerima. Selline projekti korraldus ei võimaldanud ka toime tulla uute nõuetega, mis tekkisid peale töötava süsteemi esimese versiooni koostamist.

Tagantjärgi hinnates tuleks sellise projekti juures kasutada iteratiivset arendamist ja väljalaske planeerimist nagu seda näiteks soovitatakse selles artiklis [31]. Samuti on tehtud töö puudujäägiks koodi tasemel testimise puudumine. Selleks saaks näiteks kasutada unittest library teeki [32]. Tegin sellega mõned testimismallid (*test case*), kuid mitte mahus, mis kataks kogu koodi.

9 Kokkuvõte

Käesoleva töö eesmärgiks oli luua tarkvara, mis abistaks PostgreSQL andmebaaside arendajaid andmebaaside denormaliseerimisel. Loodud tarkvara aitab PostgreSQL andmebaase refaktoreerida. Tarkvara realiseerib Steve Hobermani välja pakutud SQL-andmebaaside baastabelite denormaliseerimise algoritmi. Kasutasin programmeerimiseks Python 3-e.

Tarkvara on avatud lähtekoodiga ning publitseeritud MIT litsentsiga. Tarkvara lähtekood on avaldatud aadressil: https://github.com/andreasaadli/hoberman-algorithm

Kirjeldasin töös denormaliseerimise protsessi reaalseid kasutuskohti ning selle positiivseid ja negatiivseid külgi. Kogusin nõudeid tarkvarale, koostasin nendest lähtuvalt tarkvara tehnilise kavandi ja realiseerisin selle alusel tarkvara esimese versiooni.

Tarkvaraks on kahekihiline rakendus, mis suhtleb denormaliseeritavate tabelite kohta infot hankimiseks PostgreSQL andmebaasisüsteemiga. Tarkvara suudab andmebaasi süsteemikataloogi põhjal päringuid tehes ning rakenduse kasutajalt sisendit hankides luua tabelite loomise laused uue (denormaliseeritud) struktuuriga andmebaasi loomiseks. Rakendus ei käivita ise neid lauseid andmebaasis. Kasutaja saab need laused kätte, veendub nende sobivuses, teeb vajadusel parandusi ning täiendusi ja seejärel võib ise need laused käivitada. Sellega täidab tarkvara eesmärki olla andmebaasi arendaja abivahend ja aidata tal andmebaasi teatud päringute töökiiruse tõstmise huvides refaktoreerida.

Muuhulgas on tarkvaral mitmekeelse kasutajaliidese tugi ning praegu saab valida eestikeelse ja ingliskeelse kasutajaliidese vahel.

Programmeerimise käigus selgus, et mul jäi ajast ning oskustest puudu ja tarkvara ei realiseeri seetõttu kõiki funktsionaalsuseid, mida see võiks pakkuda. Realiseerimata jäi denormaliseerimise protsessi tulemuste salvestamine kujul, mida denormaliseerimise protsessi keskel oleks võimalik eksportida ning hiljem importida ja jätkata denormaliseerimist pooleli jäänud kohast. Indeksite uutele tabelitele ülekandmise asemel kuvatakse soovitatavad indeksid. Süsteem ei genereeri loodud tabelitesse andmete ülekandmise lauseid. Küsides kasutaja arvamust, saaks luua võimaluse, kus arvamuse baasil luuakse ka protsessi lõppedes uued indeksid. Tööjärje salvestamise ja indeksite

loomise funktsionaalsus oleksid esimesed järgnevad täiendused, mida programmis tuleks realiseerida.

Kasutatud kirjandus

[1] „http://db-engines.com/en/ranking," [Võrgumaterjal]. Available: http://dbengines.com/en/ranking. [Kasutatud 11 Veebruar 2019].

[2] „https://et.wikipedia.org/wiki/Kasutaja:Aleemet/Koodi_refaktoreerimine," [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Kasutaja:Aleemet/Koodi_refaktoreerimine. [Kasutatud 11 Veebruar 2019].

[3] Refactoring.Guru, „Refactoring.Guru," Refactoring.Guru, 2014. [Võrgumaterjal]. Available: https://refactoring.guru/refactoring. [Kasutatud 5 Mai 2019].

[4] S. Hoberman, Data Modeler's workbench: Tools and Techniques for Analysis and Design, New York: John Wiley & Sons, Inc, 2002.

[5] M. Vanaveski, „Andmebaasi disaineri töökoha realiseerimine PostgreSQL andmebaasi disaini kontrollimise süsteemi jaoks: bakalaureusetöö," Tallinna Tehnikaülikool, Tallinn, 2015.

[6] S. M. J. P. A. Hevner, „Design Science in Information Systems Research. MIS Quarterly Vol. 28 No. 1," 2004, pp. 75-105.

[7] S. Štefanič ja M. Lexa, „A Flexible Denormalization Technique for Data Analysis above a Deeply Structured Relational Database: Biomedical Applications," %1 *International Conference on Bioinformatics and Biomedical Engineering*, Springer, Cham, 2015.

[8] E. Eessaar, „Teema 14. Andmebaaside projekteerimine," Tallinna Tehnikaülikool, 2018.

[9] S. Peek, „Denormaliseerimise praktika uurimine ühe SQL-andmebaasi näitel: bakalaureusetöö," Tallinna Tehnikaülikool, Tallinn, 2016.

[10] The PostgreSQL Global Development Group, The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/10/index.html. [Kasutatud 20 5 2019].

[11] E. Eessaar, „Teema 9. Andmebaasi loogilise disaini tulemuse parandamine ja headuse kontrollimine," Tallinna Tehnikaülikool, 2018.

[12] E. Saal, „Ankurmodelleerimise mudelite realisatsiooni generaator PostgreSQL jaoks," Tallinna Tehnikaülikool, Tallinn, 2015.

[13] S. Shin ja G. Sanders, „Denormalization strategies for data retrieval from data warehouses Vol 42, No 1," Decision Support Systems, 2006, pp. 267-282.

[14] M. Hanus, To normalize or denormalize, that is the question, no. 1, Chicago: CMG Proceedings, 1994.

[15] D. Kašnikova, „Vaadete mõju päringute täitmisplaanide koostamisele kahe andmebaasisüsteemi näitel," Tallinna Tehnikaülikool, Tallinn, 2015.

[16] V. Merunka, J. Brožek, M. Šebek ja M. Molhanec, „Normalization rules of the object-oriented data model," %1 *Proceedings of EOMAS, International Workshop on Enterprises & Organizational Modeling and Simulation (8-9 June 2009, Amsterdam, The Netherlands)*, New York, 2009.

[17] T. Lv, N. Gua ja P. Yanb, „Normal forms for XML documents," Inform. Software Tech, 2004, p. 839–846.

[18] A. Kanade, A. Gopal ja S. Kanade, „A study of normalization and embedding in MongoDB," %1 *Proceedings of IACC, IEEE International Advance Computing Conference*, Gurgaon, 2014.

[19] S. M. Mikk, „Graafide esitamine SQL-andmebaasides," Tallinna Tehnikaülikool, Tallinn, 2017.

[20] J. Hahnke, „Data model design for business analysis Vol. 14, No. 10," Unix Review, 1996.

[21] M. Schkolnick ja P. Sorenson, „Denormalization: A Performance Oriented Database Design Technique," %1 *In Proceedings of the AICA 1980 Congress (Bologna, Italy)*, Brussels, 1980.

[22] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2008.

[23] Python Software Foundation, „Python," Python Software Foundation, [Võrgumaterjal]. Available: https://www.python.org/. [Kasutatud 19 May 2019].

[24] TIOBE Software BV, „TIOBE Index," TIOBE Software BV, [Võrgumaterjal]. Available: https://www.tiobe.com/tiobe-index/. [Kasutatud 20 Mai 2019].

[25] Dice, „7 Top Python GUI Frameworks," Dice, 7 August 2017. [Võrgumaterjal]. Available: https://insights.dice.com/2017/08/07/7-top-python-gui-frameworks/. [Kasutatud 19 Mai 2019].

[26] Qt, „Qt for Python," Qt, [Võrgumaterjal]. Available: https://www.qt.io/qt-for-python. [Kasutatud 19 Mai 2019].

[27] T. Reenskaug ja J. O. Coplien, „The DCI Architecture: A New Vision of Object-Oriented Programming," 2009.

[28] Microsoft, [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649571(v=pandp.10). [Kasutatud 19 Mai 20].

[29] A. Toxboe, „UI-Patterns.com," UI-Patterns.com, 2007. [Võrgumaterjal]. Available: http://ui-patterns.com/. [Kasutatud 19 Mai 2019].

[30] FOSSA, Inc., „tl;dr Legal," [Võrgumaterjal]. Available: https://tldrlegal.com/license/mit-license. [Kasutatud 20 Mai 2019].

[31] T. Norman, „Agile Release Planning 101," [Võrgumaterjal]. Available: http://tommynorman.blogspot.com/2012/09/agile-release-planning-101.html. [Kasutatud 23 Mai 2019].

[32] Python Software Foundation, „unittest - Unit testing framework - Python 3.7.3 documentation," Python Software Foundation, [Võrgumaterjal]. Available: https://docs.python.org/3/library/unittest.html. [Kasutatud 22 Mai 2019].

Lisa 1 – "Võrdlemiseks kasutatav denormaliseeritud skript"

Järnev skript pärineb allikast [9]:

```
CREATE DOMAIN d_eesnimi VARCHAR(60) NOT NULL CONSTRAINT
chk_eesnimi_tyhistring_ja_ei_koosne_numbritest CHECK (VALUE!~'^[[:space:]]*$'
AND VALUE!~'^.*[[:digit:]].*$');
CREATE DOMAIN d_perekonnanimi VARCHAR(100) NOT NULL CONSTRAINT
chk_perekonnanimi_tyhistring_ja_ei_koosne_numbritest CHECK
(VALUE!~'^[[:space:]]*$' AND VALUE!~'^.*[[:digit:]].*$');
CREATE DOMAIN d_nimetus VARCHAR(50) NOT NULL CONSTRAINT
chk_nimetus_tyhistring CHECK (VALUE!~'^[[:space:]]*$');
CREATE DOMAIN d_sisu VARCHAR(1000) NOT NULL CONSTRAINT chk_sisu_tyhistring
CHECK (VALUE!~'^[[:space:]]*$');
CREATE DOMAIN d_kirjeldus VARCHAR(255) CONSTRAINT chk_kirjeldus_tyhistring
CHECK (VALUE!~'^[[:space:]]*$');

CREATE TABLE Tootaja_aines (
 Tootaja_aines_kood SERIAL NOT NULL,
 Isikukood CHAR ( 11 ) NOT NULL,
 Ainekood CHAR ( 7 ) NOT NULL,
 Roll_Aines_kood SMALLINT NOT NULL,
 Roll_Aines_nimetus d_nimetus,
 Alguse_aeg DATE NOT NULL,
 Lopu_aeg DATE,
 Eesnimi d_eesnimi,
 Perekonnanimi d_perekonnanimi,
 Email VARCHAR ( 320 ) NOT NULL,
 Nimetus d_nimetus,
 Punktid SMALLINT NOT NULL,
 Sisu d_sisu,
 CONSTRAINT PK_Tootaja_aines PRIMARY KEY (Tootaja_aines_kood),
 CONSTRAINT AK_Tootaja_aines_kombinatsioon UNIQUE (Ainekood, Alguse_aeg,
Roll_Aines_kood, Isikukood),
 CONSTRAINT chk_Tootaja_aines_email_vastab_oigele_kujule CHECK (Email~*
'^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'),
 CONSTRAINT chk_Tootaja_aines_lopu_aeg_ylempiir CHECK (Lopu_aeg<='2118-09-
17'),
 CONSTRAINT chk_Tootaja_aines_oppeaine_punktid_suurem_0 CHECK
(Punktid>=0),
 CONSTRAINT chk_Tootaja_aines_oppeaine_punktid_vaiksem_200 CHECK
(Punktid<=200),
 CONSTRAINT chk_Tootaja_aines_alguse_aeg_ei_saa_olla_varem_kui_ttu_loomine
CHECK (Alguse_aeg>='1918-09-17'),
 CONSTRAINT chk_Tootaja_aines_alguse_aeg_ylempiir CHECK
```

```
(Alguse_aeg<='2118-09-17')
 );
CREATE INDEX idx_tootaja_aines_roll_aines_kood ON Tootaja_aines
(Roll_Aines_kood );
CREATE INDEX idx_tootaja_aines_isikukood ON Tootaja_aines (Isikukood );

CREATE TABLE Amet (
 Amet_kood SMALLINT NOT NULL,
 Nimetus d_nimetus,
 Kirjeldus d_kirjeldus,
 CONSTRAINT AK_Amet_nimetus UNIQUE (Nimetus),
 CONSTRAINT PK_Amet PRIMARY KEY (Amet_kood)
 );

CREATE TABLE Yliopilase_seisundi_liik (
 Yliopilase_seisundi_liik_kood SMALLINT NOT NULL,
 Nimetus d_nimetus,
 Kirjeldus d_kirjeldus,
 CONSTRAINT PK_Yliopilase_seisundi_liik PRIMARY KEY
(Yliopilase_seisundi_liik_kood),
 CONSTRAINT AK_Yliopilase_seisundi_liik_nimetus UNIQUE (Nimetus)
 );

CREATE TABLE Isik (
 Isikukood CHAR ( 11 ) NOT NULL,
 Isiku_seisundi_liik_kood SMALLINT DEFAULT 1 NOT NULL,
 Isiku_seisundi_liik_nimetus d_nimetus,
 Tootaja_seisundi_liik_kood SMALLINT DEFAULT 1,
 Tootaja_seisundi_liik_nimetus VARCHAR ( 50 ),
 Yliopilase_seisundi_liik_kood SMALLINT DEFAULT 1,
 Yliopilase_seisundi_liik_nimetus VARCHAR ( 50 ),
 Amet_kood SMALLINT,
 Amet_nimetus VARCHAR ( 50 ),
 Eesnimi d_eesnimi,
 Perekonnanimi d_perekonnanimi,
 Email VARCHAR ( 320 ) NOT NULL,
 Telefon VARCHAR ( 31 ) NOT NULL,
 Kasutajanimi VARCHAR ( 20 ) NOT NULL,
 Parool VARCHAR ( 60 ) NOT NULL,
 Matrikli_nr CHAR ( 6 ),
 CONSTRAINT PK_Isik PRIMARY KEY (Isikukood),
 CONSTRAINT AK_Isik_kasutajanimi UNIQUE (Kasutajanimi),
 CONSTRAINT AK_Isik_email UNIQUE (Email),
 CONSTRAINT AK_Isik_matrikli_nr UNIQUE (Matrikli_nr),
 CONSTRAINT chk_Isik_telefoni_nr_ei_sisalda_tahti CHECK
(Telefon!~'^.*[[:alpha:]].*$'),
 CONSTRAINT chk_Isik_isikukood CHECK (Isikukood~'^([3-
6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-3]{1}[[:digit:]]{5})$'),
 CONSTRAINT chk_Isik_kasutajanimi_5_kuni_20_marki CHECK
(Kasutajanimi~'^([[:alpha:]]{5,20})$'),
 CONSTRAINT chk_Isik_email_vastab_oigele_kujule CHECK (Email~* '^[A-Za-z0-
```

```
9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'),
 CONSTRAINT chk_Isik_telefoni_nr_ei_koosne_tyhikutest CHECK
(Telefon!~'^[[:space:]]*$'),
 CONSTRAINT chk_Isik_telefoni_nr_sisaldab_nr_pluss_thk CHECK
(Telefon~'^[+]{1}[[:digit:]]{3}[[:space:]]{1}[[:digit:]]{6,8}$'),
 CONSTRAINT chk_Isik_yliopilane_matrikli_nr_kuus_numbrit CHECK
(Matrikli_nr~'^([[:digit:]]{6})$'),
 CONSTRAINT chk_Isik_tootaja_seisundi_liik_nimetus_ei_koosne_tyhikutest
CHECK (Tootaja_seisundi_liik_nimetus!~'^[[:space:]]*$'),
 CONSTRAINT chk_Isik_yliopilase_seisundi_liik_nimetus_ei_koosne_tyhikutest
CHECK (Yliopilase_seisundi_liik_nimetus!~'^[[:space:]]*$'),
 CONSTRAINT chk_Isik_amet_nimetus_ei_koosne_tyhikutest CHECK
(Amet_nimetus!~'^[[:space:]]*$')
 );
CREATE INDEX idx_isik_yliopilase_seisundi_liik_kood ON Isik
(Yliopilase_seisundi_liik_kood );
CREATE INDEX idx_isik_tootaja_seisundi_liik_kood ON Isik
(Tootaja_seisundi_liik_kood );
CREATE INDEX idx_isiku_seisudi_liik_kood ON Isik (Isiku_seisundi_liik_kood );
CREATE INDEX idx_isik_amet_kood ON Isik (Amet_kood );

CREATE TABLE Keel (
 Keel_kood CHAR ( 3 ) NOT NULL,
 Nimetus d_nimetus,
 Kirjeldus d_kirjeldus,
 CONSTRAINT AK_Keel_nimetus UNIQUE (Nimetus),
 CONSTRAINT PK_Keel PRIMARY KEY (Keel_kood),
 CONSTRAINT chk_Keel_kood_vaikesed_tahed CHECK (Keel_kood =
lower(Keel_kood)),
 CONSTRAINT chk_Keel_kood_koosneb_kolmest_tahest CHECK
(Keel_kood~'^[[:alpha:]]{3}$')
 );

CREATE TABLE Tootaja_seisundi_liik (
 Tootaja_seisundi_liik_kood SMALLINT NOT NULL,
 Nimetus d_nimetus,
 Kirjeldus d_kirjeldus,
 CONSTRAINT PK_Tootaja_seisundi_liik PRIMARY KEY
(Tootaja_seisundi_liik_kood),
 CONSTRAINT AK_Tootaja_seisundi_liik_nimetus UNIQUE (Nimetus)
 );

CREATE TABLE Semester (
 Semester_kood VARCHAR ( 2 ) NOT NULL,
 Nimetus d_nimetus,
 Kirjeldus d_kirjeldus,
 CONSTRAINT PK_Semester PRIMARY KEY (Semester_kood),
 CONSTRAINT AK_Semester_nimetus UNIQUE (Nimetus),
 CONSTRAINT chk_Semester_kood_koosneb_kuni_kahest_tahest CHECK
(Semester_kood~'^[[:alpha:]]{1,2}$')
 );
```

```
CREATE TABLE Kontrollvorm (
 Kontrollvorm_kood CHAR ( 1 ) NOT NULL,
 Nimetus d_nimetus,
 Kirjeldus d_kirjeldus,
 CONSTRAINT PK_Kontrollvorm PRIMARY KEY (Kontrollvorm_kood),
 CONSTRAINT AK_Kontollvorm_nimetus UNIQUE (Nimetus),
 CONSTRAINT chk_Kontrollvorm_kood_koosneb_yhest_tahest CHECK
(Kontrollvorm_kood~'^[[:alpha:]]{1}$')
 );

CREATE TABLE Oppeaine_seisundi_liik (
 Oppeaine_seisundi_liik_kood SMALLINT NOT NULL,
 Nimetus d_nimetus,
 Kirjeldus d_kirjeldus,
 CONSTRAINT AK_Oppeaine_seisundi_liik_nimetus UNIQUE (Nimetus),
 CONSTRAINT PK_Oppeaine_seisundi_liik PRIMARY KEY
(Oppeaine_seisundi_liik_kood)
 );

CREATE TABLE Roll_aines (
 Roll_Aines_kood SMALLINT NOT NULL,
 Nimetus d_nimetus,
 Kirjeldus d_kirjeldus,
 CONSTRAINT AK_Roll_Aines_nimetus UNIQUE (Nimetus),
 CONSTRAINT PK_Roll_Aines PRIMARY KEY (Roll_Aines_kood)
 );

CREATE TABLE Isiku_seisundi_liik (
 Isiku_seisundi_liik_kood SMALLINT NOT NULL,
 Nimetus d_nimetus,
 Kirjeldus d_kirjeldus,
 CONSTRAINT PK_Isiku_seisundi_liik PRIMARY KEY (Isiku_seisundi_liik_kood),
 CONSTRAINT AK_Isiku_seisundi_liik_nimetus UNIQUE (Nimetus)
 );

CREATE TABLE Oppeaine (
 Ainekood CHAR ( 7 ) NOT NULL,
 Isikukood CHAR ( 11 ) NOT NULL,
 Kontrollvorm_kood CHAR ( 1 ) NOT NULL,
 Kontrollvorm_nimetus d_nimetus,
 Semester_kood VARCHAR ( 2 ) NOT NULL,
 Semester_nimetus d_nimetus,
 Keel_kood CHAR ( 3 ) NOT NULL,
 Keel_nimetus d_nimetus,
 Oppeaine_seisundi_liik_kood SMALLINT DEFAULT 1 NOT NULL,
 Oppeaine_seisundi_liik_nimetus d_nimetus,
 Nimetus d_nimetus,
 Punktid SMALLINT NOT NULL,
 Sisu d_sisu,
```

```
 Loomise_aeg DATE DEFAULT CURRENT_DATE NOT NULL,
 Koduleht VARCHAR ( 700 ),
 Kinnitamise_aeg DATE,
 Eesnimi d_eesnimi,
 Perekonnanimi d_perekonnanimi,
 Email VARCHAR ( 320 ) NOT NULL,
 CONSTRAINT PK_Oppeaine PRIMARY KEY (Ainekood),
 CONSTRAINT chk_Oppeaine_punktid_vaiksem_200 CHECK (Punktid<=200),
 CONSTRAINT chk_Oppeaine_loomise_aeg_alampiir CHECK (Loomise_aeg>='1918-
09-17'),
 CONSTRAINT chk_Oppeaine_email_vastab_oigele_kujule CHECK (Email~* '^[AZa-z0-
9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'),
 CONSTRAINT chk_Oppeaine_kinnitamise_aeg_ylempiir CHECK
(Kinnitamise_aeg<='2118-09-17'),
 CONSTRAINT chk_Oppeaine_loomise_aeg_ylempiir CHECK (Loomise_aeg<='2118-
09-17'),
 CONSTRAINT chk_Oppeaine_ainekood_kolm_tahti_neli_numbrit CHECK
(Ainekood~'^([[:alpha:]]{3}[[:digit:]]{4})$'),
 CONSTRAINT chk_Oppeaine_punktid_suurem_0 CHECK (Punktid>=0)
 );
CREATE INDEX idx_oppeaine_semester_kood ON Oppeaine (Semester_kood );
CREATE INDEX idx_oppeaine_seisundi_liik_kood ON Oppeaine
(Oppeaine_seisundi_liik_kood );
CREATE INDEX idx_oppeaine_isikukood ON Oppeaine (Isikukood );
CREATE INDEX idx_oppeaine_kontrollvorm_kood ON Oppeaine (Kontrollvorm_kood );
82
CREATE INDEX idx_oppeaine_keel_kood ON Oppeaine (Keel_kood );

CREATE TABLE Oppimine (
 Oppimine_kood SERIAL NOT NULL,
 Isikukood CHAR ( 11 ) NOT NULL,
 Tootaja_aines_kood SMALLINT NOT NULL,
 Deklareerimise_aeg DATE NOT NULL,
 Ainekood CHAR ( 7 ) NOT NULL,
 Roll_aines_kood SMALLINT NOT NULL,
 Alguse_aeg DATE NOT NULL,
 CONSTRAINT AK_Oppimine_kombinatsioon UNIQUE (Deklareerimise_aeg,
Isikukood, Tootaja_aines_kood, Ainekood),
 CONSTRAINT PK_Oppimine PRIMARY KEY (Oppimine_kood),
 CONSTRAINT chk_Oppimine_alguse_aeg_ei_saa_olla_varem_kui_ttu_loomine
CHECK (Alguse_aeg>='1918-09-17'),
 CONSTRAINT
chk_Oppimine_deklareerimise_aeg_ei_saa_olla_varem_kui_ttu_loomine CHECK
(Deklareerimise_aeg>='1918-09-17'),
 CONSTRAINT chk_Oppimine_alguse_aeg_ylempiir CHECK (Alguse_aeg<='2118-09-
17'),
 CONSTRAINT chk_Oppimine_ainekood_kolm_tahti_neli_numbrit CHECK
(Ainekood~'^([[:alpha:]]{3}[[:digit:]]{4})$')
 );
CREATE INDEX idx_oppimine_tootaja_aines_kood ON Oppimine (Tootaja_aines_kood
);
```

```
CREATE INDEX idx_oppimine_isikukood ON Oppimine (Isikukood );
ALTER TABLE Isik ADD CONSTRAINT FK_Isik_yliopilase_seisundi_liik_kood FOREIGN
KEY (Yliopilase_seisundi_liik_kood) REFERENCES Yliopilase_seisundi_liik
(Yliopilase_seisundi_liik_kood) ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE Isik ADD CONSTRAINT FK_Isik_tootaja_seisundi_liik_kood FOREIGN
KEY (Tootaja_seisundi_liik_kood) REFERENCES Tootaja_seisundi_liik
(Tootaja_seisundi_liik_kood) ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE Isik ADD CONSTRAINT FK_Isik_amet_kood FOREIGN KEY (Amet_kood)
REFERENCES Amet (Amet_kood) ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE Isik ADD CONSTRAINT FK_Isiku_seisundi_liik_kood FOREIGN KEY
(Isiku_seisundi_liik_kood) REFERENCES Isiku_seisundi_liik
(Isiku_seisundi_liik_kood) ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE Tootaja_aines ADD CONSTRAINT FK_Tootaja_aines_ainekood FOREIGN
KEY (Ainekood) REFERENCES Oppeaine (Ainekood) ON DELETE NO ACTION ON UPDATE
CASCADE;
ALTER TABLE Tootaja_aines ADD CONSTRAINT FK_Tootaja_aines_isikukood FOREIGN
KEY (Isikukood) REFERENCES Isik (Isikukood) ON DELETE NO ACTION ON UPDATE
CASCADE;
ALTER TABLE Tootaja_aines ADD CONSTRAINT FK_Tootaja_aines_roll_aines_kood
FOREIGN KEY (Roll_Aines_kood) REFERENCES Roll_aines (Roll_Aines_kood) ON
DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE Oppimine ADD CONSTRAINT FK_Oppimine_tootaja_aines_kood FOREIGN
KEY (Tootaja_aines_kood) REFERENCES Tootaja_aines (Tootaja_aines_kood) ON
DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE Oppimine ADD CONSTRAINT FK_Oppimine_isikukood FOREIGN KEY
(Isikukood) REFERENCES Isik (Isikukood) ON DELETE NO ACTION ON UPDATE
CASCADE;
ALTER TABLE Oppeaine ADD CONSTRAINT FK_Oppeaine_kontrollvorm_kood FOREIGN KEY
(Kontrollvorm_kood) REFERENCES Kontrollvorm (Kontrollvorm_kood) ON DELETE NO
ACTION ON UPDATE CASCADE;
83
ALTER TABLE Oppeaine ADD CONSTRAINT FK_Oppeaine_seisundi_liik_kood FOREIGN
KEY (Oppeaine_seisundi_liik_kood) REFERENCES Oppeaine_seisundi_liik
(Oppeaine_seisundi_liik_kood) ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE Oppeaine ADD CONSTRAINT FK_Oppeaine_isikukood FOREIGN KEY
(Isikukood) REFERENCES Isik (Isikukood) ON DELETE NO ACTION ON UPDATE
CASCADE;
ALTER TABLE Oppeaine ADD CONSTRAINT FK_Oppeaine_semester_kood FOREIGN KEY
(Semester_kood) REFERENCES Semester (Semester_kood) ON DELETE NO ACTION ON
UPDATE CASCADE;
ALTER TABLE Oppeaine ADD CONSTRAINT FK_Oppeaine_keel_kood FOREIGN KEY
(Keel_kood) REFERENCES Keel (Keel_kood) ON DELETE NO ACTION ON UPDATE
CASCADE;
```

Lisa 2 – "Tarkvara loodud denormaliseeritud skript"

```
START TRANSACTION;
CREATE TABLE demo.isik (isikukood character(11)  NOT NULL,
 isiku_seisundi_liik_kood smallint DEFAULT 1 NOT NULL,
 eesnimi character varying(60)  NOT NULL,
 perekonnanimi character varying(100)  NOT NULL,
 email character varying(320)  NOT NULL,
 telefon character varying(31)  NOT NULL,
 kasutajanimi character varying(20)  NOT NULL,
 parool character varying(60)  NOT NULL,
 tootaja_seisundi_liik_kood smallint DEFAULT 1 NULL,
 amet_kood smallint  NULL,
 yliopilase_seisundi_liik_kood smallint DEFAULT 1 NULL,
 matrikli_nr character(6)  NULL,
 isiku_seisundi_liik_nimetus character varying(50)  NULL,
 isiku_seisundi_liik_kirjeldus character varying(255)  NULL,
 tootaja_seisundi_liik_nimetus character varying(50)  NULL,
 tootaja_seisundi_liik_kirjeldus character varying(255)  NULL,
 yliopilase_seisundi_liik_nimetus character varying(50)  NULL,
 yliopilase_seisundi_liik_kirjeldus character varying(255)  NULL,
 amet_nimetus character varying(50)  NULL,
 amet_kirjeldus character varying(255)  NULL,
 isikukood character(11)  NULL,
 roll_aines_kood smallint  NULL,
 ainekood character(7)  NULL,
 alguse_aeg date  NULL,
 lopu_aeg date  NULL,
 CONSTRAINT chk_isik_eesnimi_ei_koosne_tyhikutest    CHECK (((eesnimi)::text
!~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_isik_eesnimi_ei_sisalda_numbreid    CHECK (((eesnimi)::text
!~ '^.*[[:digit:]].*$'::text)),
 CONSTRAINT chk_isik_email_vastab_oigele_kujule    CHECK (((email)::text ~*
'^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'::text)),
 CONSTRAINT chk_isik_isikukood    CHECK ((isikukood ~ '^([3-
6]{1}[[:digit:]]{2}[0-1]{1}[[:digit:]]{1}[0-3]{1}[[:digit:]]{5})$'::text)),
 CONSTRAINT chk_isik_kasutajanimi_5_kuni_20_marki    CHECK
(((kasutajanimi)::text ~ '^([[:alpha:]]{5,20})$'::text)),
 CONSTRAINT chk_isik_perekonnanimi_ei_koosne_tyhikutest    CHECK
(((perekonnanimi)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_isik_perekonnanimi_ei_sisalda_numbreid    CHECK
(((perekonnanimi)::text !~ '^.*[[:digit:]].*$'::text)),
 CONSTRAINT chk_isik_telefoni_nr_ei_koosne_tyhikutest    CHECK
(((telefon)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_isik_telefoni_nr_ei_sisalda_tahti    CHECK (((telefon)::text
!~ '^.*[[:alpha:]].*$'::text)),
 CONSTRAINT chk_isik_telefoni_nr_sisaldab_nr_pluss_thk    CHECK
(((telefon)::text ~
'^[+]{1}[[:digit:]]{3}[[:space:]]{1}[[:digit:]]{6,8}$'::text)),
 CONSTRAINT chk_yliopilane_matrikli_nr_kuus_numbrit    CHECK ((matrikli_nr ~
'^([[:digit:]]{6})$'::text)),
```

```
 CONSTRAINT
isiku_seisundi_liik_chk_isiku_seisundi_liik_nimetus_ei_koosne_tyhikutest
CHECK (((isiku_seisundi_liik_nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT
isiku_seisundi_liik_chk_isiku_seisundi_liik_kirjeldus_ei_koosne_tyhikutest
CHECK (((isiku_seisundi_liik_kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT
tootaja_seisundi_liik_chk_tootaja_seisundi_liik_nimetus_ei_koosne_tyhikutest
CHECK (((tootaja_seisundi_liik_nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT
tootaja_seisundi_liik_chk_tootaja_seisundi_liik_kirjeldus_ei_koosne_tyhikutes
t    CHECK (((tootaja_seisundi_liik_kirjeldus)::text !~
'^[[:space:]]*$'::text)),
 CONSTRAINT
yliopilase_seisundi_liik_chk_yliopilase_seisundi_liik_nimetus_ei_koosne_tyhik
utest    CHECK (((yliopilase_seisundi_liik_nimetus)::text !~
'^[[:space:]]*$'::text)),
 CONSTRAINT
yliopilase_seisundi_liik_chk_yliopilase_seisundii_liik_kirjeldus_ei_koosne_ty
hikutest    CHECK (((yliopilase_seisundi_liik_kirjeldus)::text !~
'^[[:space:]]*$'::text)),
 CONSTRAINT amet_chk_amet_nimetus_ei_koosne_tyhikutest    CHECK
(((amet_nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT amet_chk_amet_kirjeldus_ei_koosne_tyhikutest    CHECK
(((amet_kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_tootaja_aines_lopu_aeg_ylempiir    CHECK ((lopu_aeg <= '2118-
09-17'::date)),
 CONSTRAINT chk_tootaja_aines_ainekood_kolm_tahti_neli_numbrit    CHECK
((ainekood ~ '^([[:alpha:]]{3}[[:digit:]]{4})$'::text)),
 CONSTRAINT chk_tootaja_aines_alguse_aeg_ei_saa_olla_varem_kui_ttu_loomine
CHECK ((alguse_aeg >= '1918-09-17'::date)),
 CONSTRAINT chk_tootaja_aines_alguse_aeg_ylempiir    CHECK ((alguse_aeg <=
'2118-09-17'::date)),
 CONSTRAINT ak_isik_email    UNIQUE (email),
 CONSTRAINT ak_isik_kasutajanimi    UNIQUE (kasutajanimi),
 CONSTRAINT ak_yliopilane_matrikli_nr    UNIQUE (matrikli_nr),
 CONSTRAINT pk_tootaja_aines    UNIQUE (tootaja_aines_kood),
 CONSTRAINT pk_isik    PRIMARY KEY (isikukood),
 CONSTRAINT fk_isik_seisundi_liik_kood    FOREIGN KEY
(isiku_seisundi_liik_kood) REFERENCES
demo.isiku_seisundi_liik(isiku_seisundi_liik_kood) ON UPDATE CASCADE,
 CONSTRAINT fk_tootaja_amet_kood    FOREIGN KEY (amet_kood) REFERENCES
demo.amet(amet_kood) ON UPDATE CASCADE,
 CONSTRAINT fk_tootaja_seisundi_liik_kood    FOREIGN KEY
(tootaja_seisundi_liik_kood) REFERENCES
demo.tootaja_seisundi_liik(tootaja_seisundi_liik_kood) ON UPDATE CASCADE,
 CONSTRAINT fk_yliopilane_seisundi_liik_kood    FOREIGN KEY
(yliopilase_seisundi_liik_kood) REFERENCES
demo.yliopilase_seisundi_liik(yliopilase_seisundi_liik_kood) ON UPDATE
CASCADE,
 CONSTRAINT fk_tootaja_aines_ainekood    FOREIGN KEY (ainekood) REFERENCES
demo.oppeaine(ainekood),
 CONSTRAINT fk_tootaja_aines_roll_aines_kood    FOREIGN KEY (roll_aines_kood)
REFERENCES demo.roll_aines(roll_aines_kood) ON UPDATE CASCADE);
```

```
CREATE TABLE demo.isiku_seisundi_liik (isiku_seisundi_liik_kood smallint  NOT
NULL,
 nimetus character varying(50)  NOT NULL,
 kirjeldus character varying(255)  NULL,
 CONSTRAINT chk_isiku_seisundi_liik_kirjeldus_ei_koosne_tyhikutest    CHECK
(((kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_isiku_seisundi_liik_nimetus_ei_koosne_tyhikutest    CHECK
(((nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT ak_isiku_seisundi_liik_nimetus    UNIQUE (nimetus),
 CONSTRAINT pk_isiku_seisundi_liik    PRIMARY KEY
(isiku_seisundi_liik_kood));

CREATE TABLE demo.tootaja_seisundi_liik (tootaja_seisundi_liik_kood smallint
NOT NULL,
 nimetus character varying(50)  NOT NULL,
 kirjeldus character varying(255)  NULL,
 CONSTRAINT chk_tootaja_seisundi_liik_kirjeldus_ei_koosne_tyhikutest    CHECK
(((kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_tootaja_seisundi_liik_nimetus_ei_koosne_tyhikutest    CHECK
(((nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT ak_tootaja_seisundi_liik_nimetus    UNIQUE (nimetus),
 CONSTRAINT pk_tootaja_seisundi_liik    PRIMARY KEY
(tootaja_seisundi_liik_kood));

CREATE TABLE demo.yliopilase_seisundi_liik (yliopilase_seisundi_liik_kood
smallint  NOT NULL,
 nimetus character varying(50)  NOT NULL,
 kirjeldus character varying(255)  NULL,
 CONSTRAINT chk_yliopilase_seisundi_liik_nimetus_ei_koosne_tyhikutest
CHECK (((nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_yliopilase_seisundii_liik_kirjeldus_ei_koosne_tyhikutest
CHECK (((kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT ak_yliopilase_seisundi_liik_nimetus    UNIQUE (nimetus),
 CONSTRAINT pk_yliopilase_seisundi_liik    PRIMARY KEY
(yliopilase_seisundi_liik_kood));

CREATE TABLE demo.amet (amet_kood smallint  NOT NULL,
 nimetus character varying(50)  NOT NULL,
 kirjeldus character varying(255)  NULL,
 CONSTRAINT chk_amet_kirjeldus_ei_koosne_tyhikutest    CHECK
(((kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_amet_nimetus_ei_koosne_tyhikutest    CHECK (((nimetus)::text
!~ '^[[:space:]]*$'::text)),
 CONSTRAINT ak_amet_nimetus    UNIQUE (nimetus),
 CONSTRAINT pk_amet    PRIMARY KEY (amet_kood));

CREATE TABLE demo.keel (keel_kood character(3)  NOT NULL,
 nimetus character varying(50)  NOT NULL,
 kirjeldus character varying(255)  NULL,
 CONSTRAINT chk_keel_kirjeldus_ei_koosne_tyhikutest    CHECK
(((kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_keel_kood_koosneb_kolmest_tahest    CHECK ((keel_kood ~
'^[[:alpha:]]{3}$'::text)),
```

```
 CONSTRAINT chk_keel_kood_vaikesed_tahed    CHECK (((keel_kood)::text =
lower((keel_kood)::text))),
 CONSTRAINT chk_keel_nimetus_ei_koosne_tyhikutest    CHECK (((nimetus)::text
!~ '^[[:space:]]*$'::text)),
 CONSTRAINT ak_keel_nimetus    UNIQUE (nimetus),
 CONSTRAINT pk_keel    PRIMARY KEY (keel_kood));

CREATE TABLE demo.oppeaine (ainekood character(7)  NOT NULL,
 kontrollvorm_kood character(1)  NOT NULL,
 semester_kood character varying(2)  NOT NULL,
 keel_kood character(3)  NOT NULL,
 isikukood character(11)  NOT NULL,
 oppeaine_seisundi_liik_kood smallint DEFAULT 1 NOT NULL,
 nimetus character varying(50)  NOT NULL,
 punktid smallint  NOT NULL,
 sisu character varying(1000)  NOT NULL,
 loomise_aeg date DEFAULT ('now'::text)::date NOT NULL,
 koduleht character varying(700)  NULL,
 kinnitamise_aeg date  NULL,
 keel_nimetus character varying(50)  NULL,
 keel_kirjeldus character varying(255)  NULL,
 kontrollvorm_nimetus character varying(50)  NULL,
 kontrollvorm_kirjeldus character varying(255)  NULL,
 semester_nimetus character varying(50)  NULL,
 semester_kirjeldus character varying(255)  NULL,
 oppeaine_seisundi_liik_nimetus character varying(50)  NULL,
 oppeaine_seisundi_liik_kirjeldus character varying(255)  NULL,
 tootaja_tootaja_seisundi_liik_kood smallint DEFAULT 1 NULL,
 tootaja_amet_kood smallint  NULL,
 CONSTRAINT chk_oppeaine_ainekood_kolm_tahti_neli_numbrit    CHECK ((ainekood
~ '^([[:alpha:]]{3}[[:digit:]]{4})$'::text)),
 CONSTRAINT chk_oppeaine_kinnitamise_aeg_ylempiir    CHECK ((kinnitamise_aeg
<= '2118-09-17'::date)),
 CONSTRAINT chk_oppeaine_loomise_aeg_alampiir    CHECK ((loomise_aeg >=
'1918-09-17'::date)),
 CONSTRAINT chk_oppeaine_loomise_aeg_ylempiir    CHECK ((loomise_aeg <=
'2118-09-17'::date)),
 CONSTRAINT chk_oppeaine_nimetus_ei_koosne_tyhikutest    CHECK
(((nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_oppeaine_punktid_suurem_0    CHECK ((punktid >= 0)),
 CONSTRAINT chk_oppeaine_punktid_vaiksem_200_    CHECK ((punktid <= 200)),
 CONSTRAINT chk_oppeaine_sisu_ei_koosne_tyhikutest    CHECK (((sisu)::text !~
'^[[:space:]]*$'::text)),
 CONSTRAINT keel_chk_keel_nimetus_ei_koosne_tyhikutest    CHECK
(((keel_nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT keel_chk_keel_kirjeldus_ei_koosne_tyhikutest    CHECK
(((keel_kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT kontrollvorm_chk_kontrollvorm_nimetus_ei_koosne_tyhikutest
CHECK (((kontrollvorm_nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT kontrollvorm_chk_kontrollvorm_kirjeldus_ei_koosne_tyhikutest
CHECK (((kontrollvorm_kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT semester_chk_semester_nimetus_ei_koosne_tyhikutest    CHECK
(((semester_nimetus)::text !~ '^[[:space:]]*$'::text)),
```

```
 CONSTRAINT semester_chk_semester_kirjeldus_ei_koosne_tyhikutest    CHECK
(((semester_kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT
oppeaine_seisundi_liik_chk_oppeaine_seisundii_liik_nimetus_ei_koosne_tyhikute
st    CHECK (((oppeaine_seisundi_liik_nimetus)::text !~
'^[[:space:]]*$'::text)),
 CONSTRAINT
oppeaine_seisundi_liik_chk_oppeaine_seisundi_liik_kirjeldus_ei_koosne_tyhikut
est    CHECK (((oppeaine_seisundi_liik_kirjeldus)::text !~
'^[[:space:]]*$'::text)),
 CONSTRAINT pk_oppeaine    PRIMARY KEY (ainekood),
 CONSTRAINT fk_oppeaine_isikukood    FOREIGN KEY (isikukood) REFERENCES
demo.isik(isikukood) ON UPDATE CASCADE,
 CONSTRAINT fk_oppeaine_keel_kood    FOREIGN KEY (keel_kood) REFERENCES
demo.keel(keel_kood) ON UPDATE CASCADE,
 CONSTRAINT fk_oppeaine_kontrollvorm_kood    FOREIGN KEY (kontrollvorm_kood)
REFERENCES demo.kontrollvorm(kontrollvorm_kood) ON UPDATE CASCADE,
 CONSTRAINT fk_oppeaine_seisundi_liik_kood    FOREIGN KEY
(oppeaine_seisundi_liik_kood) REFERENCES
demo.oppeaine_seisundi_liik(oppeaine_seisundi_liik_kood),
 CONSTRAINT fk_oppeaine_semester_kood    FOREIGN KEY (semester_kood)
REFERENCES demo.semester(semester_kood) ON UPDATE CASCADE);

CREATE TABLE demo.kontrollvorm (kontrollvorm_kood character(1)  NOT NULL,
 nimetus character varying(50)  NOT NULL,
 kirjeldus character varying(255)  NULL,
 CONSTRAINT chk_kontrollvorm_kirjeldus_ei_koosne_tyhikutest    CHECK
(((kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_kontrollvorm_kood_koosneb_yhest_tahest    CHECK
((kontrollvorm_kood ~ '^[[:alpha:]]{1}$'::text)),
 CONSTRAINT chk_kontrollvorm_nimetus_ei_koosne_tyhikutest    CHECK
(((nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT ak_kontrollvorm_nimetus    UNIQUE (nimetus),
 CONSTRAINT pk_kontrollvorm    PRIMARY KEY (kontrollvorm_kood));

CREATE TABLE demo.semester (semester_kood character varying(2)  NOT NULL,
 nimetus character varying(50)  NOT NULL,
 kirjeldus character varying(255)  NULL,
 CONSTRAINT chk_semester_kirjeldus_ei_koosne_tyhikutest    CHECK
(((kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_semester_kood_koosneb_kuni_kahest_tahest    CHECK
(((semester_kood)::text ~ '^[[:alpha:]]{1,2}$'::text)),
 CONSTRAINT chk_semester_nimetus_ei_koosne_tyhikutest    CHECK
(((nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT ak_semester_nimetus    UNIQUE (nimetus),
 CONSTRAINT pk_semester    PRIMARY KEY (semester_kood));

CREATE TABLE demo.oppeaine_seisundi_liik (oppeaine_seisundi_liik_kood
smallint  NOT NULL,
 nimetus character varying(50)  NOT NULL,
 kirjeldus character varying(255)  NULL,
 CONSTRAINT chk_oppeaine_seisundi_liik_kirjeldus_ei_koosne_tyhikutest
CHECK (((kirjeldus)::text !~ '^[[:space:]]*$'::text)),
```

```
 CONSTRAINT chk_oppeaine_seisundii_liik_nimetus_ei_koosne_tyhikutest    CHECK
(((nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT ak_oppeaine_seisundi_liik_nimetus    UNIQUE (nimetus),
 CONSTRAINT pk_oppeaine_seisundi_liik    PRIMARY KEY
(oppeaine_seisundi_liik_kood));

CREATE TABLE demo.tootaja_aines (tootaja_aines_kood SERIAL NOT NULL,
 isikukood character(11)  NOT NULL,
 roll_aines_kood smallint  NOT NULL,
 ainekood character(7)  NOT NULL,
 alguse_aeg date  NOT NULL,
 lopu_aeg date  NULL,
 roll_aines_nimetus character varying(50)  NULL,
 roll_aines_kirjeldus character varying(255)  NULL,
 oppeaine_kontrollvorm_kood character(1)  NULL,
 oppeaine_semester_kood character varying(2)  NULL,
 oppeaine_keel_kood character(3)  NULL,
 oppeaine_oppeaine_seisundi_liik_kood smallint DEFAULT 1 NULL,
 oppeaine_nimetus character varying(50)  NULL,
 oppeaine_punktid smallint  NULL,
 oppeaine_sisu character varying(1000)  NULL,
 oppeaine_loomise_aeg date DEFAULT ('now'::text)::date NULL,
 oppeaine_koduleht character varying(700)  NULL,
 oppeaine_kinnitamise_aeg date  NULL,
 CONSTRAINT chk_tootaja_aines_lopu_aeg_ylempiir    CHECK ((lopu_aeg <= '2118-
09-17'::date)),
 CONSTRAINT chk_tootaja_aines_ainekood_kolm_tahti_neli_numbrit    CHECK
((ainekood ~ '^([[:alpha:]]{3}[[:digit:]]{4})$'::text)),
 CONSTRAINT chk_tootaja_aines_alguse_aeg_ei_saa_olla_varem_kui_ttu_loomine
CHECK ((alguse_aeg >= '1918-09-17'::date)),
 CONSTRAINT chk_tootaja_aines_alguse_aeg_ylempiir    CHECK ((alguse_aeg <=
'2118-09-17'::date)),
 CONSTRAINT roll_aines_chk_roll_aines_nimetus_ei_koosne_tyhikutest    CHECK
(((roll_aines_nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT roll_aines_chk_roll_aines_kirjeldus_ei_koosne_tyhikutest    CHECK
(((roll_aines_kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT oppeaine_chk_oppeaine_punktid_suurem_0    CHECK ((punktid >= 0)),
 CONSTRAINT oppeaine_chk_oppeaine_punktid_vaiksem_200_    CHECK ((punktid <=
200)),
 CONSTRAINT oppeaine_chk_oppeaine_sisu_ei_koosne_tyhikutest    CHECK
(((oppeaine_sisu)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT oppeaine_chk_oppeaine_loomise_aeg_alampiir    CHECK ((loomise_aeg
>= '1918-09-17'::date)),
 CONSTRAINT oppeaine_chk_oppeaine_loomise_aeg_ylempiir    CHECK ((loomise_aeg
<= '2118-09-17'::date)),
 CONSTRAINT oppeaine_chk_oppeaine_kinnitamise_aeg_ylempiir    CHECK
((kinnitamise_aeg <= '2118-09-17'::date)),
 CONSTRAINT ak_tootaja_aines_kombinatsioon    UNIQUE (ainekood, alguse_aeg,
roll_aines_kood, isikukood),
 CONSTRAINT pk_tootaja_aines    PRIMARY KEY (tootaja_aines_kood),
 CONSTRAINT fk_tootaja_aines_ainekood    FOREIGN KEY (ainekood) REFERENCES
demo.oppeaine(ainekood),
```

```
 CONSTRAINT fk_tootaja_aines_isikukood    FOREIGN KEY (isikukood) REFERENCES
demo.isik(isikukood) ON UPDATE CASCADE,
 CONSTRAINT fk_tootaja_aines_roll_aines_kood    FOREIGN KEY (roll_aines_kood)
REFERENCES demo.roll_aines(roll_aines_kood) ON UPDATE CASCADE);

CREATE TABLE demo.oppimine (oppimine_kood SERIAL NOT NULL,
 isikukood character(11)  NOT NULL,
 tootaja_aines_kood smallint  NOT NULL,
 deklareerimise_aeg date  NOT NULL,
 tootaja_aines_roll_aines_kood smallint  NULL,
 tootaja_aines_ainekood character(7)  NULL,
 tootaja_aines_alguse_aeg date  NULL,
 tootaja_aines_lopu_aeg date  NULL,
 CONSTRAINT chk_oppimine_deklareerimise_aeg_ei_saa_olla_varem_kui_ttu_loomi
CHECK ((deklareerimise_aeg >= '1918-09-17'::date)),
 CONSTRAINT tootaja_aines_chk_tootaja_aines_ainekood_kolm_tahti_neli_numbrit
CHECK ((tootaja_aines_ainekood ~ '^([[:alpha:]]{3}[[:digit:]]{4})$'::text)),
 CONSTRAINT
tootaja_aines_chk_tootaja_aines_alguse_aeg_ei_saa_olla_varem_kui_ttu_loomine
CHECK ((tootaja_aines_alguse_aeg >= '1918-09-17'::date)),
 CONSTRAINT tootaja_aines_chk_tootaja_aines_alguse_aeg_ylempiir    CHECK
((tootaja_aines_alguse_aeg <= '2118-09-17'::date)),
 CONSTRAINT tootaja_aines_chk_tootaja_aines_lopu_aeg_ylempiir    CHECK
((tootaja_aines_lopu_aeg <= '2118-09-17'::date)),
 CONSTRAINT ak_oppimine_kombinatsioon    UNIQUE (deklareerimise_aeg,
isikukood, tootaja_aines_kood),
 CONSTRAINT pk_oppimine    PRIMARY KEY (oppimine_kood),
 CONSTRAINT fk_oppimine_isikukood    FOREIGN KEY (isikukood) REFERENCES
demo.isik(isikukood) ON UPDATE CASCADE,
 CONSTRAINT fk_oppimine_tootaja_aines_kood    FOREIGN KEY
(tootaja_aines_kood) REFERENCES demo.tootaja_aines(tootaja_aines_kood) ON
UPDATE CASCADE);

CREATE TABLE demo.roll_aines (roll_aines_kood smallint  NOT NULL,
 nimetus character varying(50)  NOT NULL,
 kirjeldus character varying(255)  NULL,
 CONSTRAINT chk_roll_aines_kirjeldus_ei_koosne_tyhikutest    CHECK
(((kirjeldus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT chk_roll_aines_nimetus_ei_koosne_tyhikutest    CHECK
(((nimetus)::text !~ '^[[:space:]]*$'::text)),
 CONSTRAINT ak_roll_aines_nimetus    UNIQUE (nimetus),
 CONSTRAINT pk_roll_aines    PRIMARY KEY (roll_aines_kood));


COMMIT;
```