TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Märt Erlenheim 112671IAPB

ISIKUTE OTSIMINE NIME ALUSEL POSTGRESQL ANDMEBAASIS ETTEVÕTTE X NÄITEL

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: Märt Erlenheim

21.05.2019

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on leida PostgreSQL andmebaasis olevate tekstiliste andmete otsinguvõimaluste hulgast isikute täisnimede alusel otsimiseks sobivaim meetod ettevõtte X jaoks. Lisaks on vaja analüüsida otsingumeetodite poolt kasutatavate andmete indekseerimise viiside mõju andmebaasis toimuvatele operatsioonidele.

Töös viiakse läbi eksperiment, mille käigus luuakse isikunimede generaator ning mõõdetakse päringute ja andmete muutmise operatsioonide täitmisaegu, tehes selleks testandmebaasis päringuid ja andmemuudatusi. Samuti analüüsitakse tabelitele loodud indekseid, mida otsingumeetodid kasutavad.

Töö tulemuseks on analüütiliste hierarhiate meetodit e Saaty meetodit kasutades leitud otsingumeetod, mis vastab kõige paremini ettevõtte X poolt kehtestatud kriteeriumitele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, 5 peatükki, 27 joonist, 19 tabelit.

Abstract

Searching for Persons by Name in a PostgreSQL Database Based on the Example of Company X

The goal of this thesis is to find the most suitable text searching method for searching persons by full name in a PostgreSQL database for company X. I compared search methods based on the criteria set by the company X.

To accomplish the goal, I firstly researched about the complexity of names of persons and how various text search methods suit for searching the names. Moreover, I gave a short overview about the analytic hierarchy process and on how to normalize names to improve search results.

Secondly, I conducted an experiment to measure the execution times of different queries and data manipulation statements and the creation time and disk space usage of the indexes, which the database management system (DBMS) uses internally to support the search methods. For the experiment, I created a virtual environment with a database server and used a person name generator that I wrote in Java to fill the database with test data. I used the test environment for measuring the performance of all the statements and conducting the analysis. Findings showed that in general *gin* indexes make queries faster, are faster to create, and use less disk space than their *gist* counterparts.

Lastly, I used the results of the experiment as the input to the analytic hierarchy process. I used the process to compare the search methods based on the criteria that the company X provided. The search methods most suitable for the needs of company X are trigram search using a *gin_trgm_ops* index on the name field with similarity limit of 0.7 and full text search using a *gin* index with *ts_rank_cd* method to order the results of the search.

The thesis is in Estonian and contains 34 pages of text, 5 chapters, 27 figures, 19 tables.

Lühendite ja mõistete sõnastik

AHM	*Analytical Hierarchy Process, AHP* Analüütilise hierarhiate meetod ehk Saaty meetod on struktureeritud viis keeruliste subjektiivsete otsuste tegemiseks. Antud meetodiga võrreldakse võimalikke valikuid omavahel paarides kindlaks määratud kriteeriumite alusel [16]. Kriteeriumid tulenevad meetodi kasutamise eesmärgist ja kontekstist ning nende olulisuse määramiseks kasutatakse samuti nende paariviisilist võrdlemist.
ASCII	*American Standard Code for Information Interchange* Tähemärkide kodeerimise standard, mis määrab millised kahendsüsteemi arvud vastavad ladina tähemärkidele.
CM	*Consistency Measure* Saaty meetodis kasutatav kooskõla määr, mis näitab kui omavahel kooskõlas on võrdluses määratud hinnangud.
Lekseem	*Lexeme* Lekseemid on tekstis leiduvatest sõnadest võetud sõnatüved.
Levenshteini kaugus	Tekstiline väljenduse kahe jada vahelistest erinevustest, mille muutmisel muutuks üks jada teiseks. PostgreSQL'i puhul on see tähtede arv, mis eristavad kahte sõna. Näiteks, nimede „Mart" ja „Mark" Levenshteini kaugus on 1.
Ligikaudne otsing	*Fuzzy search* Ähmane otsing, mis kasutab sõnade, antud töös nimede, võrdluseks ligikaudset moodust. Sellise võrdluse alusel oleksid näiteks nimed „Mart" ja „Mark" võrdsed.
PEP	*Politically exposed person* „Riikliku taustaga isik on füüsiline isik, kes täidab või on täitnud avaliku võimu olulisi ülesandeid, sealhulgas riigipea, valitsusjuht, minister ning ase- või abiminister, parlamendiliige või parlamendiga sarnase seadusandliku organi liige, erakonna juhtorgani liige, ülemkohtu ja riigikohtu liige, riigikontrolli ja keskpanga nõukogu liige, suursaadik, asjur ja kaitsejõudude kõrgem ohvitser, riigiäriühingu juhatuse ja haldus- või järelevalveorgani liige, rahvusvahelise organisatsiooni juht, juhi asetäitja ja juhtorgani liige või samaväärseid ülesandeid täitev isik, kes ei ole kesk- või alamastme ametniku staatuses." [1]
Päring	*Query*

	Lause andmete otsimiseks andmebaasist. SQLis on selleks SELECT lause.
RahaPTS	Rahapesu ja terrorismi rahastamise tõkestamise seadus [1]
Rakendusliides	*API, Application Programming Interface* Rutiinide hulk (liides), läbi mille saab näiteks üks rakendus kasutada teise rakenduse teenuseid.
SphinxQL	Sphinxi otsingumootori poolt kasutatav andmekäitluskeel, mis on sarnane SQL keelele. [11]
Trigraaf/Trigramm	Ühe hääliku tähistus kolme tähega, nt saksa sch. PostgreSQL kontekstis on tegu sõnast võetud kolme järjestikuse tähe grupiga. [10]
tsvektor	*Tsvector* PostgreSQL'i andmetüüp, mida kasutab täistekstiotsing. Iga tsvektor tüüpi kuuluv väärtus sisaldab sorteeritud nimekirja unikaalsetest lekseemidest. [10]
Unikood	*Unicode* Standard, mis sätestab reeglid teksti tähemärkide kodeerimiseks, kuvamiseks ja töötlemiseks.
UTF-8	Enamlevinud tähemärkide kodeerimise standard, mis vastab unikoodi standardile ja ühildub ASCII kodeeringuga.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Ettevõte X pakub krediidiasutustele isikute taustakontrolli teenuseid. Üheks teenuseks on ka isikute otsimine rahvusvahelistest sanktsioonide ja riiklike taustaga isikute nimekirjadest. Krediidiasutustel on kohustuslik läbi viia antud kontroll lähtuvalt ***„Rahapesu ja terrorismi rahastamise tõkestamise seadusest“*** [1]. Ettevõttes X on kasutusel PostgreSQL andmebaas ka muude teenuste osutamiseks ning seepärast oli isikute otsimine vaja realiseerida selles andmebaasis.

Lõputöö eesmärgiks on leida PostgreSQL andmebaasis olevate tekstiliste andmete otsinguvõimaluste hulgast isikute täisnimede alusel otsimiseks sobivaim meetod. Lisaks on vaja analüüsida otsingumeetodite poolt kasutatavaid andmete indekseerimise viiside mõju andmebaasis toimuvatele operatsioonidele. Veel on vaja määratleda ka nimede normaliseerimise ja transliteratsiooni reeglid, mis muudaksid isikute otsingu täpsemaks, kuid samas ei põhjustaks vääraid tulemusi. Lõputöös väljatoodud tulemusi oleks võimalik kasutada ka muudes PostgreSQL andmebaasi kasutavates süsteemides, kus on vajalik teostada isikute otsingut. Otsingumeetodite ja indeksite võrdlemiseks kasutatakse lõputöös analüütiliste hierarhiate meetodit ehk Saaty meetodit, et leida ettevõtte X tingimustele kõige paremini vastav lahendus.

Lõputöö on jaotatud kolme põhipeatükki. Esimeses peatükis antakse ülevaade isikunimede keerukusest ja normaliseerimise vajalikkusest ning kirjeldatakse PostgreSQL’i tekstiotsingu meetodeid ja nende sobivust isikute otsimiseks nimede alusel. Lisaks on ära toodud ettevõtte X kriteeriumid, mida otsingumeetodite võrdlemisel peaks arvestama. Teises peatükis kirjeldatakse otsingumeetodite võrdlemiseks läbi viidud eksperimenti ja selle eksperimendi tulemusi. Samuti kirjeldatakse isikunimede generaatorit, mis oli vaja luua, et oleks võimalik otsingumeetodeid omavahel võrrelda. Viimases peatükis analüüsitakse otsingumeetodite vastavust ettevõtte X poolt seatud tingimustele kasutades Saaty meetodit, mille võrdluste aluseks on teises peatükis kirjeldatud eksperimendi läbi viimisest saadud tulemused.

2 Teoreetiline taust

Lõputöö sissejuhatuses tõstatud probleemi lahendamiseks peab esiteks andma ülevaate miks isikute otsimine nime alusel on keerulisem kui esialgu võib tunduda. Kui sellesse probleemi süveneda, siis oleks sellest võimalik kirjutada pikki raamatuid, kuid käesolevas lõputöös antakse ainult lühike ülevaade.

2.1 Isikunimed

Rahvusvahelistes sanktsioonide ja riiklike taustaga isikute nimekirjades on mitmetest rahvustest isikute nimesid ning nimed ei ole alati algsel ja õigel nimekujul. Näiteks peab Eestis kontrollima isikute leidumist Euroopa Liidu majandussanktsioonide nimekirjast [2]. Seega ei saa isikute otsimisel nimede alusel eeldada, et nimed järgivad alati Eestis kehtivaid isikunimede reegleid [3][4], vaid peab arvestama isikunimede eripäradega. Isikunimede mitmekesisusest ja keerukusest on põhjalikumalt kirjutatud Mairi Jõgi bakalaureusetöös ***„Mõned disainilahendused isikunimede hoidmiseks SQL-andmebaasides“*** [5]. Näiteks ei ole alati võimalik isikute puhul eristada eesnimesid ja perekonnanimesid või isikul ongi ainult üks nimi [6]. Lisaks võib ka nimede erinevate osade järjestus olla nimekirjades erinev. Välja toodud puudused võivad esineda ka otsingutingimustes kasutatavatel nimedel kuna ka otsijatel võib nimest olla vale kirjapilt. Nendel ja muudel põhjustel peaks isikute otsing toimuma isikute täisnimede alusel ning suutma leida isikuid sõltumata sellest, mis järjekorras on nende täisnime osad kirja pandud.

Isikute otsimisel nime alusel peab lisaks veel arvestama, et vähesed nimed on unikaalsed. Sõltuvalt nime populaarsusest kipuvad nimed rohkem või vähem korduma. Näiteks, Ameerika Ühendriikides, rahvaarvuga 328 896 867 (17.05.2019. seisuga), on perekonnanimi Smith 2,44 miljonil ja Johnson peaaegu 2 miljonil ameeriklasel [7]. Eesnimedest levinuim, John, on eelneva 70 aasta jooksul esinenud umbes 3,5 miljonit korda [8]. Otsingumeetodid peavad suutma tõhusalt toimida nii populaarsete kui ka haruldaste nimedega.

Käesolev lõputöö keskendub ainult isikute otsimisele nimede alusel, kuid reaalselt toimiv süsteem peaks suutma otsingutulemusi filtreerida veel ka muude isikut iseloomustavate omaduste alusel, nagu näiteks sünniaasta või elukoht. Üldistest otsingute strateegiatest on põhjalikumalt kirjutatud Mike Dunkerley ja Geoff Holloway raamatus „***The Math, Myth and Magic of Name Search and Matching***“ [9].

2.2 Tekstiotsingu võimalused PostgreSQL’is

PostgreSQL’is on olemas mitmeid tekstide otsimise meetodeid, mis on loodud erinevate otsinguvajaduste täitmiseks: on meetodeid, mis suudavad otsida tekste täpse võrdluse alusel ning on ka meetodeid, mis võimaldavad teostada ligikaudset tekstide võrdlust. Järgmistes jaotistes kirjeldatud otsingumeetodite täispikkuses päringute näidised on jaotises 3.2.3.

2.2.1 Võrdsusotsing

Kõige tavalisem võrdlusmeetod, mida kasutatakse ka muud tüüpi väljade puhul täpseks otsinguks. Suudab leida ainult nimesid, mis täielikult kattuvad. See tähendab, et otsitava nime osad ja andmebaasis oleva nime osad peavad olema täpselt samas järjekorras ja täpselt samasuguse kujuga. Samuti on võrdlus tõstutundlik. Joonis 1 on võrdsusotsingu aluseks olev tekstide võrdlusmeetod PostgreSQL’is.

```
täisnimi = 'otsitav nimi'
```

Joonis 1. Võrdsusotsingu aluseks olev tekstide võrdlus PostgreSQL'is.

2.2.2 Lihtne mustriga otsing

Isikute otsimiseks saab LIKE predikaadis kasutada protsendimärki %, mis mustriga otsingus vastab ükskõik millisele märgile null või enam korda [10]. Sellega on võimalik mustriga otsing muuta sõltumatuks otsitavate nimede järjekorrast. Mustriga otsingut on võimalik teostada tõstutundetult kasutades ILIKE predikaati. Joonis 2 on mustriga otsingu aluseks olev tekstide tõstutundlik võrdlusmeetod PostgreSQL’is, mis ei sõltu otsitava nime osade järjekorrast.

```
täisnimi LIKE '%nimi%’ AND täisnimi LIKE '%otsitav%'
```

Joonis 2. Mustriga otsingu aluseks olev tekstide võrdlus PostgreSQL'is.

Kuna % märk võib vastata ükskõik mis arvule märkidele, siis sellise päringu tulemuste arvu peaks piirama täisnimede pikkuse kontrolliga. Näiteks võib seada piirangu, et otsitava täisnime pikkus võib olla kuni kaks tähemärki pikem kui otsitav täisnimi. Piiramata tulemuste hulgas on muidu ka nimed, mis sisaldavad otsitud nime osasid ainult ühe osana pikemast nimest. Suurtest nimekirjadest otsides tähendab see, et tulemuses on palju üleliigseid ridu. Pikkusega piiramine muudab otsingu ka ligikaudsemaks, sest leitaks ka isikuid, kelle nimi erinev otsitavast nimest ainult lisatähtede poolest.

2.2.3 Trigrammi otsing

Trigrammi otsingu võimekuse saab andmebaasile lisada kui installeerida laiendusmoodul *pg_trgm*. Trigrammi otsing kasutab sõnade võrdlemiseks trigramme. Joonis 3 on näiteks minu perekonnanime trigrammid.

```
SELECT show_trgm('Erlenheim');
                     show_trgm
-----------------------------------------------
 {"  e","  er",eim,enh,erl,hei,"im ",len,nhe,rle}
```

Joonis 3. PostgreSQL'is trigrammide loomise näide.

Tühikutega trigrammid tulenevad sellest, et trigrammide moodustamiseks lisatakse sõnade ette kaks tühikut ja järele üks tühik [10]. Trigrammi võrdluses võrreldaksegi omavahel nimedest moodustatud trigramme ja tulemuseks on arv (0 ehk täiesti erinevad kuni 1 ehk täielikult kattuvad), mis tähistab, kui sarnased on nimede trigrammid. Trigrammi võrdluses pole oluline sõnade (antud juhul nime osade) järjekord. Trigrammi otsingule on võimalik määrata sarnasuse arvule piirang, mis määrab tõeseks ainult võrdlused, mille tulemuseks olev arv on üle selle piirarvu. Vaikimisi on piirarvuks 0.3. Seda piirangut on võimalik määrata ka ainult PostgreSQL’i sessiooni põhiselt. Selle piirarvu peaks määrama vastavalt nimekirjale, et mõjutada otsingu tulemuste täpsust. Lisaks on kasulik seda muuta nimede pikkuse põhiselt kuna tähtede erinevused mõjutavad lühikesi ja pikki nimesid erinevalt. Näiteks, ühe tähe erinevus neljatäheliste sõnade puhul tähendab võrreldavate sõnade sarnasuse hinnangut 0.25, samas ühe tähe erinevus kümnetäheliste sõnade korral tähendab võrreldavate sõnade sarnasuse hinnangut 0.6. Joonis 4 on mustriga otsingu aluseks olev tekstide võrdlusmeetod PostgreSQL’is.

```
täisnimi % 'otsitav nimi'
```

Joonis 4. Trigrammi otsingu aluseks olev tekstide võrdlus PostgreSQL'is.

Otsingu tulemusi on võimalik järjestada sarnasuse arvu alusel kui tulemuste järjestamise reeglites kasutada Joonis 5 olevat sarnasuse leidmise funktsiooni.

```
similarity(täisnimi, 'otsitav nimi')
```

Joonis 5. PostgreSQL'i funktsioon leidmaks sõnade trigrammide sarnasuse hinnangut.

Trigrammi moodulis on andmete indekseerimise moodused *gist_trgm_ops* ja *gin_trgm_ops*, mis kiirendavad trigrammi võrdlusega päringuid. Lisaks toetavad need indeksid ka mustriga otsinguid, milles kasutatakse % märki.

2.2.4 Täistekstiotsing

Täistekstiotsingu aluseks on dokumentidest moodustatavad tekstiotsingu vektorid e tsvektorid (*tsvector*), mis sisaldavad järjestatud unikaalseid lekseeme. Lekseemid on tekstis leiduvatest sõnadest võetud sõnatüved ja tsvektori moodustamisel on võimalik määrata keel, mille reeglite alusel eraldatakse sõnatüved sõnadest. Nimede puhul peab kasutama tsvektori moodustamisel keele asemel liiki *simple*, et nimesid ei muudetaks vektori loomisel. Lekseemidele on võimalik lisada ka indeksid, mis näitavad mitmendast lause sõnast on lekseem moodustatud. Lisaks saab lekseemidele lisada kaalu, mis näitab, kui oluline on tulemuste leidmisel antud lekseem. Kaaludeks on tähtsuse järjekorras A, B, C ja D ning vaikimisi on nende väärtused 1.0, 0.4, 0.2 ja 0.1, kuid nendele on võimalik määrata ka oma väärtused. Kui tsvektori sisendiks kasutada erinevaid välju, siis on võimalik igale väljale anda erinev kaal. Näiteks oleks võimalik muuta isiku peamine täisnimi tähtsamaks kui isikule kuuluvad teised nimed ja sellega järjestada kõrgemalt tulemusi, kus otsitav nimi leidub peamises isiku täisnimes. Joonis 6 on näidisena loodud tsvektor PostgreSQL’is.

```
SELECT to_tsvector('simple', 'esimene sõna, korduv sõna, järjekord abc');
                        to_tsvector
-----------------------------------------------------------
 'abc':6 'esimene':1 'järjekord':5 'korduv':3 'sõna':2,4
```

Joonis 6. PostgreSQL'is täistekstiotsinguks kasutatava tsvektori näidis.

Otsinguks on vajalik luua otsitavatest sõnadest tekstiotsingu päring e tspäring (*tsquery*), millega moodustatakse sõnadele otsimiseks kasutatavad lekseemid. Nimede otsimisel peab kasutama uuesti *simple* liiki. Ka tspäringu lekseemidele on võimalik määrata

erinevad kaalud, et mõjutada otsingutulemuste järjestust. Joonis 7 on näidisena loodud tspäring PostgreSQL’is.

```
SELECT plainto_tsquery('simple', 'esimene sõna, korduv sõna, järjekord abc');
                       plainto_tsquery
-------------------------------------------------------------
 'esimene' & 'sõna' & 'korduv' & 'sõna' & 'järjekord' & 'abc'
```

Joonis 7. PostgreSQL'is täistekstiotsinguks kasutatava tspäringu näidis.

Kui otsingutute toimumise kiirus on olulisem kui kasutatav kettamaht, siis saab tsvektorite jaoks lisada tabelisse veeru, kuhu salvestatakse tekstiotsingus kasutatavate veergude väärtustest moodustatud tsvektorid. Võib teha nii, et igas otsingus kasutatavas tabelis on selle jaoks eraldi veerg, kus on just selles tabelis olevatest andmetest moodustatud tsvektorid. Võib ka teha nii, et tsvektor on eraldi tabelis (Joonis 11) ja selles on üle kõikide otsitavate tabelite moodustatud tsvektor väärtused Samuti on võimalik kas tsvektor veerule või nime veerule tsvektori funktsioonil põhinevalt luua *btree_gist* ja *btree_gin* indeksid. Joonis 8 on eraldatud väljaga täistekstiotsingu võrdluse näide.

```
(tsvektor @@ plainto_tsquery('simple', 'otsitav nimi')
```

Joonis 8. Täistekstiotsingu aluseks olev tekstide võrdlus PostgreSQL'is.

Otsingu tulemuste järjestamiseks on kaks funktsiooni: *ts_rank* ja *ts_rank_cd*. Mõlemad kasutavad järjestamiseks otsitavate lekseemide tsvektoris leidumise sagedust ja arvestavad määratud kaaludega. Järjestades tulemused *ts_rank_cd* alusel arvestatakse ka otsitavate lekseemide omavahelist kaugust. Selleks peavad tsvektorid olema loodud koos lekseemide järjestuse indeksitega. See meetod ei arvesta järjestamise loomisel ilma indeksita lekseeme.

Täistekstiotsinguga on võimalik luua ka veelgi keerukamaid otsinguid, kuid selles lõputöös neid ei käsitleta [10].

2.2.5 Levenshteini kaugusel põhinev otsing

Levenshteini kaugusel põhineva otsingu võimaluse saab andmebaasi lisada kui installeerida laiendusmoodul *fuzzystrmatch*. Moodul arvutab Levenshteini kauguse arvestades milliste tegevustega oleks võimalik otsitavast sõnast moodustada võrdluses

osaleva teise sõna (Joonis 9). Nendeks tegevusteks on tähemärgi lisamine, kustutamine ja asendamine. Nendele tegevustele saab määrata hinna, kuid vaikimisi on hinnaks 1 ning kaugus näitab mitme tähe poolest kaks võrreldavat sõna üksteisest erinevad. Nimede otsimisel on võimalik kasutada kohandatud Levenshteini kauguse arvutamise moodust, millele saab ette anda tähtede erinevuse piiri ja selle piiri ületamisel lõpetakse kauguse edasine arvutamine.

```
levenshtein_less_equal(täisnimi, 'otsitav nimi', 2) <= 2
```

Joonis 9. Levenshteini kaugust kasutava otsingu aluseks olev tekstide võrdlus PostgreSQL'is.

Levenshteini kauguse arvutamine sõltub nimede järjekorrast ning lisaks on võrreldavate nimede pikkuse piirang 255 tähemärki [10].

2.2.6 Otsingumootor Sphinx

Sphinx on avatud lähtekoodiga otsingumootor, mis suudab indekseerida PostgreSQL'i tabelites asuvaid andmeid kasutades konfiguratsioonis määratud SQL päringuid, et pärida tabelitest andmete unikaalseid identifikaatoreid ja indekseeritavaid andmeid. Sphinx hoiustab moodustatud indeksit eraldi PostgreSQL'ist konfiguratsioonis määratud failisüsteemi asukohas. Lõputöö kirjutamise hetkel (2019. aasta kevad) pole kolmanda versiooni lähtekood avalik. Sphinxil pole PostgreSQL'i jaoks loodud muud liidestust peale andmete indekseerimise võimekuse. Pärast indeksite loomist peab otsinguid teostama läbi Sphinxi rakendusliidese. Sphinxi päringud väljastavad tulemusena ainult unikaalse identifikaatori, mida kasutades peab andmeid eraldi PostgreSQL'i andmebaasist küsima [11]. Joonis 10 on näidis Sphinxi andmekäitluskeeles ehk SphinxQL'is kasutatav täpsel võrdlusel põhinev otsingu tingimus.

```
MATCH('otsitav nimi')
```

Joonis 10. Sphinxi otsingu aluseks olev tekstide võrdlus SphinxQL keeles.

2.3 Nimede normaliseerimine

Nimede normaliseerimine on vajalik kuna nimede algallikas või otsijate poolt sisestatud nimedes võib esineda variatsioone ning normaliseerimine aitab ühtlustada nimede kirjapilti, et tagada täpsemad otsingutulemused. Järgnevalt loetletakse ettevõttes X

toimuva nimede normaliseerimise osad, mis on paika pandud sanktsioonide nimekirjade alusel. Need sammud on väljatöötatud ettevõtte X põhiselt.

1. Nimed translitereeritakse. Transliteratsioon on nimede teisendamine ühest tähestikust teise. Selleks asendatakse nimes esinevad lähtekeele tähemärgid nendele sarnaste tähemärkidega sihtkeelest. Asendusreegliteks on olemas mitmeid erinevaid standardeid ja viise ning nende hulgast tuleb valida enda tingimustele sobilik meetod. Näiteks, vene nimede üleviimiseks ladina tähestikule on võimalik kasutada Rahvusvahelise Tsiviillennunduse Organisatsiooni [12] või ISO 9:1995 standardi [13] poolt määratud reegleid. Lisaks saaks kasutada masinkeelseid teisendusreegleid. Näiteks võib kasutada, Sean M. Burke'i loodud reegleid unikoodi tähemärkide teisendamiseks *ASCII* kujule [14].

2. Nimedes esinevad kirjavahemärgid asendatakse kas tühikutega või eemaldatakse nimest. Näiteks võib probleeme põhjustada kahe nimega eesnimi, mille kirjapilt võib olla kas sidekriipsuga ühendatud või lahku kirjutatult. Sidekriips asendatakse tühikuga kuna see on enamasti kasutusel nimede vahelise eraldajana. Koma, ülakoma, punkt jm kirjavahemärgid eemaldatakse nimest kuna need on kas nime lõpus või eraldavad ainult paari tähte ülejäänud nimest.

3. Nimede kõik tähed muudetakse kas väike- või suurtähtedeks, et otsing oleks tõstutundetu. Oluline on, et enne seda oleks nimed translitereeritud, sest esineb tähemärke, millel puudub kas väiketäht või suurtäht. Tähe kirjapildi muutmine võib kaasa tuua ettearvamatuid muutusi. Näiteks, alles mõned aastad tagasi puudus saksa tähel „ß" suur kirjapilt ja viis, selle teisendamiseks suureks täheks, oli selle asendamine tähtedega „SS" [15].

Lisaks, kui isiku täisnimes on ka keskmine nimi või teine perekonnanimi, siis on võimalik otsinguks tekitada uus täisnime kuju, milles kasutatakse ainult eesnime ja/või ühte perekonnanime. See muudab otsingu täpsemaks kuna isikud ei kasuta alati oma täisnime kirja pannes kõiki nime osasid.

2.4 Kriteeriumid, mille alusel tekstiotsingu võimalusi võrrelda

Ettevõte X pakub reaalaja toimuvat isikute taustakontrolli teenust läbi rakendusliidese. Seetõttu peaks isikute otsimine nime alusel toimuma kiiremini kui paar sekundit. Otsingu

meetod peaks toetama isikute leidmist sõltumata sellest, mis järjekorras on isiku täisnime osad kirjutatud. Võimaluse korral peaks otsingu meetod toetama ka ligikaudset otsingut, et välistada isikute mitteleidmine väiksemate kirjavigade korral. Kuna pakutav teenus on tasuline, siis ei saa oodata, et teenuse kasutajad kordaksid ise päringuid nime erinevate variatsioonidega. Seega peavad otsingu tulemused olema piisava täpsusega, et vältida ebaoluliste tulemuste väljastamist.

Rahvusvahelised sanktsioonide nimekirjad uuenevad maksimaalselt kord päevas ja riikliku taustaga isikute nimekiri ainult kord kuus. Enamus nendest nimekirjadest ei paku uuenduste toimumisel nimekirja toimunud muudatustest ja seetõttu luuakse uute andmete saabumisel vastavad tabelid uuesti. Indeksid moodustatakse otsinguks kasutatavatele väljadele pärast kõikide andmete sisestamist. Andmete laadimiseks kustutatakse indeksid, kustutatakse andmed tabelist (DELETE operatsioon), lisatakse andmed uuesti tabelitesse (INSERT operatsioon) ja luuakse uuesti indeksid. Need tegevused on koondatud kokku ühte tehingusse. Kuigi nimekirju uuendatakse öösel, mil teenusele tavaliselt päringuid ei saadeta, ei tohi uute nimekirjade loomine võtta kauem kui üks tund. Andmete laadimise käigus ei ole teenus kättesaadav. See tähendab, et indeksite loomiseks peaks kuluma vähem kui 20 minutit. Indeksite mõju andmete uuendamisele (UPDATE operatsioon) on väheoluline, kuna nimekirjade andmed muutuvad ainult läbi kõikide andmete uuesti sisestamise.

Samuti on väheoluline indeksite poolt kasutatav salvestusruum, sest suurimas nimekirjas on ainult 1,3 miljonit rida ning see on aja möödudes aeglaselt kasvav. See tähendab, et nimekirjade salvestamiseks kuluv kettamaht on niigi suhteliselt madal.

2.5 Analüütiliste hierarhiate meetod

Analüütiliste hierarhiate meetod on loodud Thomas L. Saaty poolt 1970ndatel. Analüütiliste hierarhiate meetodit kasutatakse selleks, et muuta subjektiivseid hinnanguid („mulle tundub“, „ma arvan“, et X on parem kui Y) objektiivsemaks (arvuliste väärtustega põhjendatuks). Meetodi kasutamisel tuleb kõigepealt paika panna eesmärk. Eesmärgiks võib olla alternatiivide hulgast parima valimine mingis kontekstis e olukorras. Seejärel määratakse kindlaks kriteeriumid, mille alusel võimalikke alternatiive omavahel võrrelda. Alternatiivide omadused jaotatakse kriteeriumite alusel gruppidesse, et oleks võimalik neid omadusi võrrelda. Järgmiselt toimub kõikide alternatiivide

omavaheline paariviisiline võrdlemine iga kriteeriumi alusel. Võrreldavatele paaridele määrarakse omaduste võrdlemise alusel skaala suhe, mis näitab kumb alternatiiv on vaadeldavas kriteeriumis eelistatud. Enne või pärast alternatiivide võrdlemist võrreldakse omavahel paaris ka kõiki kriteeriume selle alusel kui oluline on kriteerium eesmärgi saavutamiseks ning määratakse ka neile skaala suhted. Skaala suhete alusel arvutatakse alternatiividele ja kriteeriumitele tähtsuse indeks. Kasutades neid indekseid saab leida igale alternatiivile lõpliku tähtsuse indeksi ning kõrgeima tähtsusega alternatiiv on ka kõige sobilikum valik [16].

3 Eksperiment

Selleks, et hinnata PostgreSQL'i ja Sphinxi vastavust ettevõtte X kriteeriumitele, oli vaja lõputöös korraldada andmete kogumiseks eksperiment.

3.1 Eesmärgid

Eksperimendi peamine eesmärk on võrrelda PostgreSQL'i ja Sphinxi otsingumeetodeid täitmisaja alusel. Lisaks võrreldakse omavahel ka meetodite poolt kasutatavaid erinevaid andmete indekseerimise mooduseid, sh indeksite andmemahte ning nende ajalist mõju andmete sisestamisele, uuendamisele ja kustutamisele.

3.2 Eksperimendi kirjeldus

Võrdluses on täitmisajad üksikute päringute (nii leiduvate kui ka mitteleiduvate isikute kohta) ja ühe failina järjest toimuvate mitmete päringute kohta. Isikuid otsitakse kasvava suurusega nimekirjadest, et võrrelda otsingumeetodite tõhusust erineva arvu isikute korral. Enne päringuid käivitatakse PostgreSQL'i statistika värskendamise protsess ja iga päringut korratakse kolm korda, et vältida tulemuses süsteemi jõudluse erakorralistest muutustest tulenevaid anomaaliaid. Võrdluseks kasutatakse sama päringu erinevate tulemuste geomeetrilist keskmist. Otsingumeetodite poolt kasutatavatele veergudele luuakse järjest erinevaid meetodite poolt toetatud indekseid ning korratakse eelnevaid päringute teostamise samme. Korraga on nendel veergudel ainult üks otsinguid toetav indeks ja indeksite vahetamisel käivitatakse PostgreSQL'i *VACUUM* protsess, millega eemaldatakse andmebaasi sisemiselt tasemelt vanad andmed.

Pärast päringuid käivitatakse sarnaste reeglite alusel ka andmete muutmise lauseid, et jälgida indeksite mõju nende lausete täitmisele.

Eksperimendis on lisaks PostgreSQL'i sisestele otsingumeetoditele võrdluses ka Sphinxi otsingumootor. See aitab võrrelda PostgreSQL'i sisseehitatud võimalustel põhinevate otsingute tulemusi teiste spetsialiseeritud otsimisviisidega.

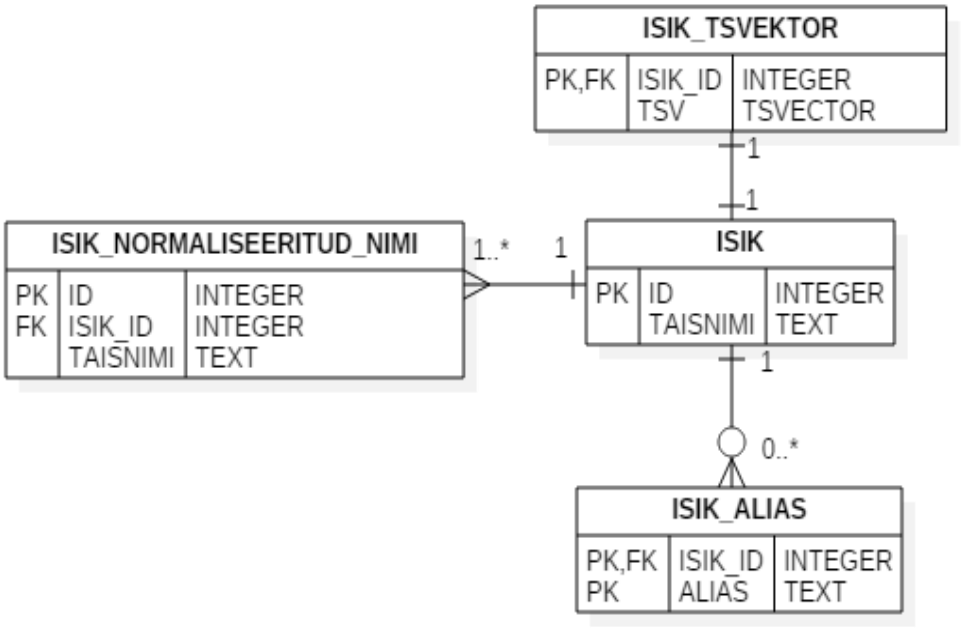
3.2.1 Testkeskkond

Eksperimendiga seotud tegevused toimuvad Linuxi baasil virtuaalmasinas, mis on käivitatud lauaarvutil. Lauaarvuti peamised näitajad on Microsoft Windows, Ryzen 5 1600 protsessor, 16 GB muutmälu ning operatsioonisüsteem paikneb eraldi SSD kettal. Virtuaalmasin töötab VirtualBoxis versiooniga 5.2.22. Sellele on määratud kaks protsessori tuuma, 4 GB muutmälu ning andmete hoiustamiseks kõvaketas, mille kiiruseks on 7200 pööret minutis. Virtuaalmasina operatsioonisüsteemiks on CentOS 7.6.1810 minimaalne paigaldus. PostgreSQL serveri versioon on 11.3. Sphinxi versiooniks valiti 2.3.2-id64-beta kuna see on lõputöö kirjutamise ajal (2019. kevad) viimane avatud lähtekoodiga versioon, mis muutis selle tööle saamise lihtsamaks. PostgreSQL ja Sphinx kasutavad mõlemad vaikimisi konfiguratsioone.

Iga isikute hulga jaoks luuakse eraldi andmebaas. Andmebaasides on tähemärkide kodeeringuks määratud UTF-8. PostgreSQL'i kõik päringud käivitatakse läbi *psql* käsurea terminali. Sphinxi päringud teostatakse Pythoni skripti vahendusel läbi hüperteksti edastusprotokolli rakendusliigese. Pythoni kood on välja toodud Lisas 1.

3.2.2 Andmebaas

Andmebaasis asuvad ainult isikute täisnime alusel otsimiseks vajalikud tabelid. Joonis 11 esitatakse andmebaasi struktuuri kirjeldav diagramm. Kommentaariks olgu öeldud, et kõik mitte-võtme veerud on kohustuslikud (NOT NULL kitsendusega). Lisas 2 on kirjas andmebaasi loomiseks kasutatud SQL laused (laienduste installeerimine ja tabelite loomine). Lisa 2 sisaldab ka välja kommenteerituna eksperimendi käigus kasutatavaid indeksite loomise SQL lauseid.



Joonis 11. Andmebaasi tabelite struktuur.

Tabelis *isik* hoitakse isikute täisnimesid originaalkujul.

Tabelis *isik_alias* hoitakse isikule kuuluvate muude täisnimede originaalkujusid. Aliasteks on näiteks isiku varasemad nimed.

Tabelis *isik_normaliseeritud_nimi* hoitakse jaotises 2.3 välja toodud reeglite alusel kõikide isikule kuuluvate nimede normaliseeritud kuju. Tabelis olevate nimede koguarv võib erineda isiku täisnimede ja aliaste arvu summast kuna normaliseerimise tulemusel võivad aliased muutuda duplikaatideks, mida ei sisestata tabelisse.

Tabelis *isik_tsvektor* hoitakse isiku kõikide nimede normaliseeritud kuju põhjal moodustatud PostgreSQL'i tsvektor tüüpi väärtuseid. Isiku nimedest luuakse ainult üks tsvektor, kuna täistekstiotsing järjestab tulemusi sõnade esinemissageduste alusel.

Kuna isikutele luuakse juhuslikkuse alusel sissekandeid tabelitesse *isik_alias* ja *isik_normaliseeritud_nimi*, siis nende tabelite ridade arv sõltub testandmete generaatori väljundist. Tabel 1 on kirjas tegelikud eksperimendis kasutatud tabelite ridade arvud. Isikute arv algab tuhandest, sest sellise pikkusega on lühemad sanktsioonide nimekirjad, 10 tuhande isikuga on pikemad sanktsioonide nimekirjad. Riikliku taustaga isikute nimekirjas on umbes miljon isikut ning võrdlusesse on kaasatud 10 miljoni isikuga

tabelid, et näha kuidas *PEP* nimekirja kasv võiks tulevikus mõjuda otsingute täitmisaegadele.

Tabel 1. Eksperimendis kasutatud tabelite ridade arvud.

Tabel / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
isik	1000	10000	100000	1000000	10000000
isik_alias	305	2913	29987	299335	2996858
isik_tsvektor	1000	10000	100000	1000000	10000000
isik_normaliseeritud_nimi	1304	12903	129820	1297801	12980812

3.2.3 Päringud

Kõik PostgreSQL'i otsimise päringud annavad tulemuseks tabeli *isik* veergude *id* ja *taisnimi* väärtused nendelt ridadelt, mille *id* väärtus leiti otsinguks kasutatud tabelitest, kus otsitav nimi võrdus mõne tabelis oleva nimega. Peamiseks erinevuseks päringute vahel on võrdlusmeetod ja see kuidas tulemusi järjestatakse. Otsitavad nimed genereeriti jaotises 3.3 kirjeldatud programmi abil kasutades selleks juhuslikult valitud ridu eelnevalt genereeritud andmete sisestamise lausetest.

Võrdsusotsing kasutab nimede võrdlemiseks ranget tõstutundliku ja järjekorrast sõltuvat võrdlust (Joonis 12).

```
SELECT i.id, i.taisnimi FROM isik_normaliseeritud_nimi q JOIN isik i ON
q.isik_id = i.id WHERE q.taisnimi = 'otsitav nimi';
```

Joonis 12. Täpset otsingut kasutav päring.

Mustriga otsing kasutab nimede võrdlemiseks tõstutundliku *LIKE* mustrit (Joonis 13). Nime osad on eraldi *LIKE* võrdluses selleks, et tagada tulemuste sõltumatus järjekorrast. Lisaks on võrreldava nime pikkus piiratud otsitava nime pikkuse alusel, et välistada tunduvalt pikemate nimede väljastamine otsingutulemustes.

```
SELECT i.id, i.taisnimi FROM isik_normaliseeritud_nimi q JOIN isik i ON
q.isik_id = i.id WHERE char_length(q.taisnimi) <= otsitava nime osade pikkus
+ 2 AND q.taisnimi LIKE '%otsitav nime osa 1%' AND q.taisnimi LIKE '%otsitav
nime osa 1%';
```

Joonis 13. Mustril põhinevat otsingut kasutav päring.

Trigrammi otsing kasutab nimede võrdlemiseks trigrammi operaatorit %, millega loetakse võrdseks ainult nimed, mille sarnasuse arv ületab eelnevalt seatud piiri (Joonis 14). Pikkuse piirang on nimedele seatud ainult, et väga pikkade nimede puhul välistada rohkem ebatäpseid tulemusi. See poleks vajalik kui sarnasuse piiri muuta dünaamiliselt nime pikkuse alusel. Tulemuste järjestamiseks kasutatakse leitud sarnasuse hinnangut.

```
SELECT i.id, i.taisnimi, similarity(q.taisnimi, 'otsitav nimi') AS similarity
FROM isik_normaliseeritud_nimi q JOIN isik i ON q.isik_id = i.id WHERE
CHAR_LENGTH(q.taisnimi) BETWEEN otsitava nime pikkus - 2 AND otsitava nime
pikkus + 2 AND q.taisnimi % 'otsitav nimi' ORDER BY similarity DESC;
```

Joonis 14. Mustril põhinevat otsingut kasutav päring.

Täistekstiotsing muudab esiteks otsitava nime tspäringuks ja seejärel võrdleb seda tsvektoriga (Joonis 15). Tulemused järjestatakse algoritmi alusel, mille arvutuses arvestatakse nimede esinemissagedust tsvektoris. Järjestamise meetod *ts_rank_cd* arvestab arvutamisel veel ka nimeosade lähedust esialgses täisnimes.

```
SELECT i.id, i.taisnimi FROM isik_tsvektor q JOIN isik i ON q.isik_id = i.id
WHERE (q.tsv @@ plainto_tsquery('simple', 'otsitav nimi')) ORDER BY
ts_rank(q.tsv, plainto_tsquery('simple', 'otsitav nimi')) DESC LIMIT 100;
```

Joonis 15. Täistekstotsingut kasutav päring.

Levenshteini kaugusega otsing arvutab nimede arvutamisel nimede vahelise kauguse (Joonis 16). Kasutatud *levenshtein_less_equal* meetod lõpetab kauguse arvutamise kui ületatakse ettemääratud piir. Lisaks järjestatakse tulemused kauguse alusel.

```
SELECT i.id, i.taisnimi FROM isik_normaliseeritud_nimi q JOIN isik i ON
q.isik_id = i.id WHERE levenshtein_less_equal(q.taisnimi, 'otsitav nimi', 2)
<= 2 ORDER BY levenshtein_less_equal(q.taisnimi, 'otsitav nimi, 2');
```

Joonis 16. Levenshteini kaugust kasutav päring.

Sphinxi otsingu päring tagastab ainult tabeli *isik* veeru *id* väärtuse (Joonis 17). Võrdluseks kasutatakse Sphinxi sisemist *MATCH* meetodit. Sphinxi otsing SphinxQL keeles:

```
SELECT id FROM indeksi_nimi WHERE MATCH('otsitav nimi') LIMIT 0, 100;
```

Joonis 17. Sphinxi otsingut kasutav päring.

3.2.4 Andmete muutmise laused

Indeksite mõju analüüsimiseks andmebaasi operatsioonidele käivitati andmebaasis andmete sisestamise, uuendamise ja kustutamise lauseid. Sisestamise lauseid korrati kolme erineva uue isikuga ning muutmise ja kustutamise lauseid kolme juhuslikult valitud reaga. Lausete täitmiseks kulunud aega mõõdeti iga lause puhul eraldi.

Esiteks sisestati tabelisse *isik* uue isiku andmed (Joonis 18).

```
INSERT INTO isik VALUES (117285596, 'Uus Isik');
```

Joonis 18. Uue isiku registreerimine.

Seejärel sisestati selle isiku kohta andmed tabelisse *isik_normaliseeritud_nimi* (Joonis 19).

```
INSERT INTO isik_normaliseeritud_nimi VALUES (117285596, 117285596, 'uus
isik');
```

Joonis 19. Isiku normaliseeritud nime registreerimine.

Sama isiku kohta sisestati andmed ka tabelisse *isik_tsvektor*. Nimest tsvektori moodustamiseks kasutati PostgreSQL'i sissehitatud funktsionaalsust (Joonis 20).

```
INSERT INTO isik_tsvektor VALUES (117285570, to_tsvector('simple', 'uus
isik'));
```

Joonis 20. Isiku nime tsvektori registreerimine.

Järgmiseks uuendati tabelis *isik_normaliseeritud_nimi* juhuslikult valitud reas veeru *taisnimi* väärtust juhuslikult genereeritud nimega (Joonis 21). Juhusliku rea valimiseks kasutati veeru *isik_id* väärtust, millele andmebaasis on loodud B-puu indeks.

```
UPDATE isik_normaliseeritud_nimi SET taisnimi = 'uus nimi' WHERE isik_id =
117284733;
```

Joonis 21. Isiku normaliseeritud nime uuendamine.

Tabelis *isik_tsvektor* uuendati samuti juhuslikult valitud rea *tsv* veerus olevat väärtust ning kasutati uuesti PostgreSQL'i sisemist funktsiooni tsvektori moodustamiseks (Joonis 22). Juhusliku rea valmiseks kasutati veeru *isik_id* väärtust. (isik_id) on selles tabelis primaarvõti.

```
UPDATE isik_tsvektor SET tsv = to_tsvector('simple', 'patricio carrasco
lerma') WHERE isik_id = 117285308;
```

Joonis 22. Isiku nime tsvektori muutmine.

Viimasena käivitati kustutamise laused tabelis *isik_normaliseeritud_nimi* kasutades selleks juhuslikku *isik_id* väärtust (Joonis 23).

```
DELETE FROM isik_normaliseeritud_nimi WHERE isik_id = 117285436;
```

Joonis 23. Isiku normaliseeritud nime kustutamine.

Kustutamise lause viidi viimasena läbi ka tabelis *isik_tsvektor* kasutades juhuslikku *isik_id* väärtust (Joonis 24).

```
DELETE FROM isik_tsvektor WHERE isik_id = 117285439;
```

Joonis 24. Isiku tsvektori kustutamine.

3.2.5 Andmekäitluse lausete käivitamine

PostgreSQL'i *psql* terminalis loetakse SQL laused failidest ja nende tulemused suunatakse eraldi failidesse. Lausete täitmisaegade saamiseks on kasutatud *psql'i* täitmisaja leidmise funktsioon (\timing). Mitu lauset korraga käivitades summeeritakse kõikide lausete täitmisajad, et saada kogu kulunud aeg.

Päringute tulemuste arvu ei ole piiratud, et võrrelda leitavate isikute ridade arvu. Tagastavate ridade arvu peaks tegelikult piirama sõltuvalt otsingunimekirja omapäradest, et tagada otsingu tulemuste täpsus ja asjakohasus. Levenshteini otsingut on korratud kaugustega 0, 1 ja 2 ning trigrammi otsingut sõnade sarnasuse läve limiitidega 1, 0.8 ja 0.7. See aitab võrrelda kui palju on täitmisaeg mõjutatud ligikaudsest nimede võrdlusest. Lisaks on täistekstiotsingut korratud mõlema võimaliku tulemuste järjestamise meetoditega, *ts_rank* ja *ts_rank_cd*.

Indekseerimise mooduste uurimiseks sisestati uued read tabelile *isik* ja see järel uued read nendele isikute kohta tabelitesse *isik_normaliseeritud_nimi* ja *isik_tsvektor*. Neid tegevusi korrati kõikide võrreldavate indeksitega. Andmete sisestamise operatsioonid olid genereeritud juhuslikkuse alusel igale indeksile eraldi ning operatsioone ühe ja sama isiku kohta ei korratud.

Muutmise ja kustutamise laused kasutavad juhuslikke *isik_id* väärtuseid, mis valiti isikute otsimiseks genereeritud andmete sisestamise failidest. Tabelis *isik_normaliseeritud_nimi* on veerg *isik_id* välisvõti ning seepärast on lause kiirendamiseks sellele veerule loodud B-puu indeks (PostgreSQL ei loo välisvõtmetele automaatselt indeksit).

3.3 Isikunimede generaator

Eksperimendis otsitakse isikuid nimekirjadest, mille on genereerinud minu enda Javas (versioon 1.8.0_181) kirjutatud kood. Antud programm kasutab erinevatest rahvustest isikute genereerimiseks statistilisi nimekirju, mis sisaldavad ees- ja perekonnanimesid koos isikute arvuga. Isiku genereerimisel valib programm juhuslikkuse alusel ühe etteantud riikidest ning täisnime moodustamiseks valitakse juhuslikult selle riigi eesnimede ja perekonnanimede loenditest nimed. Nimekirjad on pärit erinevate riikide statistiliste andmete hulgast. Nendeks riikideks on Eesti [17], Soome [18], Prantsusmaa [19], [20], Venemaa [21], [22], [23], Hispaania [24] ja Ameerika Ühendriigid [7] [8]. Programmi väljundi suuremaks mitmekesisuseks oleks vajalik rohkemate riikide statistika, kuid paljude riikide puhul pole neid andmeid avalikustatud. Programmis on igale riigile määratud veel ka naiste, keskmise nime ja teise perekonnanime esinemissagedused. Näiteks hispaania nimed genereeritakse alati kahe perekonnanimega. Programm genereerib isikutele lisaks peamisele täisnimele veel ka juhuslikkuse alusel muid täisnimesid. Teiste täisnimede ehk aliaste loomisel on lisatud võimalus, et isikule jääb sama ees- või perekonnanimi. See kattub loomulike nimede muutumistega ning on oluliseks otsingumeetoditele, mis reastavad tulemused nimede esinemissageduste järgi, nagu teeb seda täistekstiotsing.

Generaatoris on lihtsamal kujul realiseeritud jaotises 2.3 paika pandud reeglite alusel nimede normaliseerimine, et genereerida *isik_normaliseeritud_nimi* ja *isik_tsvektor* tabelitesse sissekandeid. Normaliseerimise realisatsioon kasutab nimede transliteratsiooniks vabavaralist Java teeki JUnidecode. Selle teegi autor Giuseppe Cardone kohandas Javale eelnevalt mainitud Sean M. Burke'i poolt loodud Perli mooduli.

Programmi väljundiks on PostgreSQL'i andmete importimiseks kasutatavad *copy* laused. Selleks, et ka tabeli *isik_tsvektor* ridu saaks *copy* lausetega importida, on programmile lisatud lihtsustatud moodus isiku normaliseeritud nimede teisendamiseks PostgreSQL'i tsvektori kujule. Programmi väljundi näidis on ära toodud Lisas 3.

Kuna programm toetub nimede genereerimisel ainult statistikale, siis programmi väljundis on rohkem segarahvusest nimesid kui neid reaalsuses esineb. Näiteks Eesti nimede genereerimisel on umbes veerand nimedest segarahvusest. Generaatoril puudub ka võimekus perekonnanimesid sooliselt eristada. Seetõttu võib vene nimede puhul meestel esineda naiste kirjapildis perekonnanimi ja ka vastupidi. Soolise eristamise funktsionaalsust ei loodud kuna sooliselt eristatud perekonnanimesid leidus ainult eesti populaarsete nimede nimekirjas.

Generaatorile on lisatud ka võimekus genereerida jaotises 3.2.3 kirjeldatud lauseid eksperimendi läbiviimiseks. Lisaks saab selles veel genereerida ka indeksite analüüsimiseks vajalikke andmete sisestamise, muutmise ja kustutamise lauseid. Mõlemad alamprogrammid kasutavad sisendiks nimede generaatori loodud nimekirju, et tagada isikute leidumine nimekirjas. Mitteleiduvate isikute genereerimine kasutab Hispaania ja Ameerika Ühendriikide statistilistest nimekirjadest pärit haruldasi nimesid. Mitteleiduva isiku loomisel ühendatakse Hispaania eesnimed ja Ameerika Ühendriikide perekonnanimed. Antud nimede mitteleidumist kontrolliti koodiga, mis otsis igat haruldast nime muudest nimekirjadest, võrreldes omavahel nimede normaliseeritud kuju, ning eemaldas need nimed, mis leidusid ka mujal.

3.4 Eksperimendi tulemused

Tabelid on sorteeritud täitmisaja alusel, mis kulus otsimiseks 10 miljoni isiku seast. Tabelites on iga otsingumeetodi puhul *alla joonitud* see variant, mille korral 10 miljoni isiku hulgast otsimine oli kõige kiirem. Otsingumeetodid on need, mida kirjeldas jaotis 2.2 ning variandid nende kasutus erinevate juhtparameetrite väärtustega või erinevate indeksitega. PostgreSQL ei kasutanud indeksit tüübiga *gin_trgm_ops* kui otsing toimus 1000 isiku seast, sest päringu täitmiseks koostatud täitmisplaan eelistas indeksi kasutamise asemel filtreerida andmebaasi ridasid nimede pikkuse alusel.

3.4.1 Nimekirjas mitteleiduva isiku otsimise tulemused

Tabel 2 on üksiku päringuna nimekirjas mitteleiduva isiku otsimise täitmisajad. Riikliku taustaga isikute nimekirjast otsimisel on madal tõenäosus, et otsing leiab otsitava isiku. Seetõttu on mitteleiduva isiku otsimiseks kulunud aeg ettevõtte X jaoks olulisem kui nimekirjas leiduvate isikute päring.

Tabel 2. Nimekirjas mitteleiduva isiku otsimise täitmisajad millisekundites.

Meetodi variant / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Võrdsusotsing B-puu	0.40	0.45	0.40	0.40	0.39
Täistekstiotsing *ts_rank_cd gin*	0.52	0.52	0.51	0.54	0.56
Täistekstiotsing *ts_rank gin*	0.51	0.51	0.54	0.54	0.57
Võrdsusotsing B-puu *gist*	0.39	0.44	0.49	0.46	0.63
Sphinx	2.08	2.05	1.92	1.99	2.13
Mustriga otsing *gin_trgm_ops*	0.62	0.51	0.54	0.71	10.10
Trigramm limiit 1 *gin_trgm_ops*	4.77	0.54	0.68	1.77	18.50
Trigramm limiit 0.8 *gin_trgm_ops*	4.68	0.54	0.76	2.11	39.17
Trigramm limiit 0.7 *gin_trgm_ops*	4.67	0.57	0.81	2.71	67.91
Täistekstiotsing *ts_rank gist*	0.54	0.55	1.18	2.51	180.81
Täistekstiotsing *ts_rank_cd gist*	0.52	0.53	1.16	2.54	192.89
Mustriga otsing *gist_trgm_ops*	0.47	0.70	2.88	13.50	203.60
Trigramm limiit 1 *gist_trgm_ops*	0.63	1.43	3.63	20.13	424.98
Võrdsusotsing ilma indeksita	0.47	1.33	9.68	59.75	536.12
Mustriga otsing ilma indeksita	0.58	2.01	17.78	83.17	815.25
Täistekstiotsing *ts_rank_cd* ilma indeksita	0.66	2.14	15.64	87.53	870.79
Täistekstiotsing *ts_rank* ilma indeksita	0.67	2.07	15.47	88.45	875.97
Levenshteini kaugus 0	0.68	2.90	24.09	133.41	1232.23
Levenshteini kaugus 1	0.69	2.76	23.53	129.56	1241.09
Levenshteini kaugus 2	0.71	2.98	28.76	149.96	1369.56
Trigramm limiit 0.8 *gist_trgm_ops*	0.83	3.93	25.12	180.12	2924.11
Trigramm limiit 0.7 *gist_trgm_ops*	0.82	4.10	27.47	263.49	3368.37
Trigramm limiit 1	4.71	43.38	372.01	1957.34	19694.85
Trigramm limiit 0.7	4.74	43.53	368.17	1963.09	19884.01
Trigramm limiit 0.8	4.78	43.19	371.34	1976.97	19903.37

3.4.2 Nimekirjas leiduva isiku otsimise tulemused

Tabel 3 on üksiku päringuna nimekirjas leiduva isiku otsimise täitmisajad. Enamus ettevõtte X teenuse kasutajatest saadavad iga otsitava isiku kohta eraldi päringuid.

Tabel 3. Nimekirjas leiduva isiku otsimise täitmisajad millisekundites.

Meetodi variant / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Võrdsusotsing B-puu	0.38	0.42	0.41	0.45	0.45
Täistekstiotsing *ts_rank gin*	0.56	0.59	0.57	0.84	0.59
Täistekstiotsing *ts_rank_cd gin*	0.61	0.56	0.59	0.90	0.60
Võrdsusotsing B-puu *gist*	0.43	0.46	0.49	0.54	0.84
Sphinx	2.04	1.96	2.08	2.17	2.16
Mustriga otsing *gin_trgm_ops*	0.63	0.53	0.73	1.37	12.08
Täistekstiotsing *ts_rank gist*	0.56	0.85	1.15	14.75	17.46
Täistekstiotsing *ts_rank_cd gist*	0.57	0.86	1.18	14.72	17.70
Trigramm limiit 1 *gin_trgm_ops*	4.07	0.58	1.23	4.17	53.81
Trigramm limiit 0.8 *gin_trgm_ops*	3.99	0.66	2.07	5.80	96.49
Trigramm limiit 0.7 *gin_trgm_ops*	4.01	0.73	2.53	5.92	156.52
Mustriga otsing *gist_trgm_ops*	0.57	0.90	1.77	26.90	163.96
Trigramm limiit 1 *gist_trgm_ops*	0.72	1.78	4.41	40.41	274.51
Võrdsusotsing ilma indeksita	0.48	1.35	9.62	59.84	534.78
Mustriga otsing ilma indeksita	0.60	2.35	14.64	88.95	845.86
Täistekstiotsing *ts_rank_cd* ilma indeksita	0.72	2.20	16.67	95.89	871.46
Täistekstiotsing *ts_rank* ilma indeksita	0.65	2.19	17.16	94.41	882.77
Levenshteini kaugus 1	0.85	4.21	34.65	257.65	2908.85
Levenshteini kaugus 0	0.83	4.21	35.12	249.63	2989.53
Trigramm limiit 0.8 *gist_trgm_ops*	0.79	3.84	37.79	256.99	3022.79
Levenshteini kaugus 2	0.87	4.25	35.79	269.17	3058.85
Trigramm limiit 0.7 *gist_trgm_ops*	0.73	3.88	39.26	277.87	3263.50
Trigramm limiit 0.7	4.12	45.45	493.45	1832.28	20658.84
Trigramm limiit 0.8	3.96	45.05	496.63	1868.31	20849.80
Trigramm limiit 1	3.91	45.05	499.11	1854.09	20877.96

3.4.3 Ühe failina järjest mitmete päringute tulemused

Tabel 4 tulemused on saadud andes *psql* sisendiks faili, milles on järjest juhuslikkuse alusel vaheldumisi 50 leiduvat ja 50 mitteleiduvat isikut. Igas failis on 100 päringut. Iga päringu kohta annab *psql* täitmiseks kulunud aja eraldi (Time: 100 ms, Time: 50 ms jne). Töös leiti nende päringute täitmisaegade summa 100 + 50 + ... = 150+ ms. Seda faili

käivitati kolm korda, mis andis kolm kogu summat, millest omakorda leiti geomeetriline keskmine ehk keskmine aeg kõikide failis olevate päringute tegemiseks. Edasi jagati leitud geomeetriline keskmine päringute arvuga (100), et saada keskmine aeg ühe päringu kohta. Seda tehti, et saada keskmine täitmisaeg, mis on vähem mõjutatud PostgreSQL'i tegevustest enne päringu käivitamist nagu näiteks täitmisplaani koostamine. Miljoni ja 10 miljoni isiku otsingute puhul oli failis ainult 10 leiduvat ja 10 mitteleiduvat isikut kuna vastasel juhul oleks mõnede päringute täitmiseks kulunud ebamõistlikult palju aega. Antud mõõtmises jagunesid leiduvad ja mitteleiduvad isikud võrdselt, kuid ettevõttes X toimuvate päringute korral enamasti ei leita isikut nimekirjast.

Tabel 4. Ühe failina järjest toimunud päringute keskmine ühe päringu täitmisaeg millisekundites.

Meetodi variant / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Võrdsusotsing B-puu	0.29	0.27	0.30	0.23	0.26
Võrdsusotsing B-puu *gist*	0.31	0.30	0.33	0.38	0.59
Täistekstiotsing *ts_rank_cd gin*	0.43	0.38	0.41	0.48	0.78
Täistekstiotsing *ts_rank gin*	0.45	0.39	0.38	0.41	0.92
Sphinx	0.77	0.79	0.98	0.98	1.59
Mustriga otsing *gin_trgm_ops*	0.46	0.37	0.44	1.43	10.73
Trigramm limiit 1 *gin_trgm_ops*	4.40	0.39	0.71	4.03	33.12
Trigramm limiit 0.8 *gin_trgm_ops*	4.41	0.45	0.99	6.33	65.96
Täistekstiotsing *ts_rank gist*	0.37	0.50	1.14	8.77	73.09
Täistekstiotsing *ts_rank_cd gist*	0.37	0.51	1.18	8.74	73.19
Trigramm limiit 0.7 *gin_trgm_ops*	4.39	0.46	1.20	8.60	86.70
Mustriga otsing *gist_trgm_ops*	0.34	0.74	3.27	25.62	373.20
Võrdsusotsing ilma indeksita	0.32	1.21	10.35	60.53	549.43
Trigramm limiit 1 *gist_trgm_ops*	0.53	1.35	6.01	51.88	584.88
Täistekstiotsing *ts_rank_cd* ilma indeksita	0.52	1.99	17.10	92.52	857.90
Täistekstiotsing *ts_rank* ilma indeksita	0.50	2.00	16.60	94.00	862.62
Mustriga otsing ilma indeksita	0.42	1.97	17.39	91.06	877.03
Levenshteini kaugus 0	0.61	3.31	28.45	186.44	2123.48
Levenshteini kaugus 1	0.66	3.46	30.36	195.38	2137.38
Levenshteini kaugus 2	0.65	3.75	32.66	208.39	2259.84
Trigramm limiit 0.8 *gist_trgm_ops*	0.61	3.28	27.15	242.77	2876.03
Trigramm limiit 0.7 *gist_trgm_ops*	0.65	3.39	30.10	282.22	3224.92
Trigramm limiit 0.7	4.48	41.44	379.58	1929.07	19158.59
Trigramm limiit 0.8	4.42	40.30	384.75	1934.53	19185.74
Trigramm limiit 1	4.43	40.90	374.14	1935.26	19234.30

3.4.4 Mitme isiku otsimisega leitud isikute arv

Tabel 5 on loodud eelmises jaotises kirjeldatud otsingute tulemuste alusel. See näitab kui oluline on nime alusel otsimine ühendada muude isikutunnuste alusel otsimisega kuna vastasel juhul suurtest nimekirjadest otsides on otsingutulemustes üleliigne arv ridasid sarnaste nimedega isikute tõttu. Riikliku taustaga isikute nimekirjast otsimisel esineb see

sama probleem, kuid väiksemas mahus, sest nimekirjas olevatel isikute nimedel on suurem variatsioon. Nende tulemuste alusel selgub, et Levenshteini põhimõttel otsimine muutub liialt ebatäpseks kui isikute arv kasvab. Lisaks see näitab kui oluline on trigrammi otsingule määrata nimekirjale sobilik limiit. Tabeli tulemusi vaadates peab arvestama, et miljoni ja 10 miljoni isikuga nimekirjadest otsiti **viis korda vähem** isikuid.

Tabel 5. Mitme isiku otsimisega leitud isikute arv.

Meetodi variant / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Võrdsusotsing	50	50	51	102	171
Mustriga otsing	50	50	51	107	179
Levenshteini kaugus 0	50	50	51	102	171
Levenshteini kaugus 1	50	50	56	115	431
Levenshteini kaugus 2	51	52	103	219	1270
Trigramm limiit 1	50	50	51	102	171
Trigramm limiit 0.8	50	50	60	106	176
Trigramm limiit 0.7	50	50	66	119	276
Täistekstiotsing	50	50	54	119	238

3.5 Indeksite võrdlus

Indeksid aitavad kiirendada päringuid, kuid kasutavad seejuures kettamahtu. Indeksid mõjutavad andmete muutmise lausete täitmisaegasid kuna selliste operatsioonide toimumisel peab andmebaasisüsteem indeksite sisu uuendama.

3.5.1 Indeksite kettamaht

Tabel 6 sisaldab tabeli *isik_normaliseeritud_nimi* veeru *taisnimi* põhjal loodud indeksite poolt kasutatud kettamahte kilobaitides. Nende tulemuste põhjal kasutab väline Sphinxi otsingumootor indeksite haldamiseks kõige vähem kettamahtu. Tabel 6 on alla joonitud tabelis *isik_normaliseeritud_nimi* olevate andmete poolt kasutatud kettamaht.

Tabel 6. Tabeli *isik_normaliseeritud_nimi* võimalike indeksite kettamahud kilobaitides.

Tabel või indeks / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
isik_normaliseeritud_nimi	80	720	7112	70656	710656
Sphinx	20	196	1969	19708	197065
gin_trgm_ops	224	864	5856	50176	372736
B-puu	64	464	4552	45056	452608
B-puu *gist*	72	752	7760	78848	801792
gist_trgm_ops	144	1328	13312	134144	1339392

Tabel 7 on tabeli *isik_tsvektor* veeru *tsv* alusel loodud indeksite kettamahud. Mõlema tabeli tulemustest on näha, et *gist* tüüpi indekseid kasutavad rohkem kettamahtu kui *gin* tüüpi indeksid. Tabel 7 on alla joonitud tabelis *isik_tsvektor* olevate andmete poolt kasutatud kettamaht.

Tabel 7. Tabeli *isik_tsvektor* võimalike indeksite kettamahud kilobaitides.

Tabel või indeks / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
isik_tsvektor	88	800	7888	78848	788480
gin	120	640	3200	15360	99328
gist	48	360	3672	36864	370688

3.5.2 Indeksite loomiseks kulunud aeg

Tabel 8 olevate tulemuste alusel on näha, et kõige kiiremini suudab indeksi moodustada Sphinxi otsingumootor. PostgreSQL'i sisemiste indeksite loomise puhul oli väikseima täitmisajaga täistekstiotsingu tsvektori tüüpi veerule *gin* indeksi loomine. Üldiselt oli *gist* indeksite loomine alati aeglasem kui *gin* indeksite loomine ning B-puu *gist* indeksi loomine oli märgatavalt aeglasem kõigist teistest.

Tabel 8. Indeksite loomise täitmisajad millisekundites.

Indeks / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Sphinx	20.63	53.59	258.86	1919.44	19703.36
gin	7.79	39.41	284.45	2143.03	23057.33
B-puu	19.47	65.26	629.57	5533.82	64169.14
gin_trgm_ops	16.96	85.29	706.21	6886.32	83159.92
gist	15.68	171.56	1993.10	22547.23	257965.35
gist_trgm_ops	10.46	139.46	1715.47	21166.57	632879.51
B-puu gist	21.12	334.64	5547.36	86818.84	3719192.42

3.5.3 Tabeli isik_normaliseeritud_nimi indeksite mõju andmete muutmisele

Andmete muutmise ehk andmete sisestamiste, uuendamise ja kustutamiste lausete täitmisaegade mõõtmistulemused Tabel 9-Tabel 11 näitavad, et üle miljoni rea korral muudavad *gist* tüüpi indeksid andmete muutmise märgatavalt aeglasemaks võrreldes *gin* tüüpi indeksitega.

Tabel 9. Tabeli *isik_normaliseeritud_nimi* andmete sisestamise lausete täitmisajad millisekundites.

Indeks / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Ilma indeksita	1.45	1.36	1.29	1.28	1.29
B-puu	1.54	2.29	1.47	1.92	1.77
B-puu *gist*	1.68	1.43	1.69	1.37	1.62
gin_trgm_ops	1.89	1.28	1.30	2.16	2.04
gist_trgm_ops	2.83	1.35	1.33	1.86	4.98

Tabel 10. Tabeli *isik_normaliseeritud_nimi* andmete muutmise lausete täitmisajad millisekundites.

Indeks / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Ilma indeksita	1.49	1.48	1.69	1.55	13.18
B-puu	1.32	1.64	1.57	1.57	10.51
B-puu *gist*	1.80	1.65	1.66	1.43	33.73
gin_trgm_ops	1.52	1.37	1.76	1.33	16.55
gist_trgm_ops	2.68	1.38	1.35	2.30	43.79

Tabel 11. Tabeli *isik_normaliseeritud_nimi* andmete kustutamise lausete täitmisajad millisekundites.

Indeks / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Ilma indeksita	1.28	1.58	1.37	1.48	1.43
B-puu	1.25	1.48	1.46	1.51	1.47
B-puu *gist*	1.27	1.39	1.60	1.43	18.04
gin_trgm_ops	1.50	1.39	2.29	1.25	1.05
gist_trgm_ops	2.57	1.34	1.27	1.78	30.39

3.5.4 Tabeli isik_tsvektor indeksite mõju andmete muutmisele

Andmete muutmise lausete täitmisaegade mõõtmistulemused Tabel 12-Tabel 14 näitavad, et üle miljoni rea korral muutuvad andmete muutmised ilma indeksita ja *gin* tüüpi indeksiga andmebaasides aeglaseks. Seda erinevust võib selgitada täitmisaegade suur variatsioon. Antud tulemuste kontrollimiseks tehtud kustutamise ja muutmiste lausete täitmiseks kulunud aeg oli vahemikus 5 kuni 30 ms, nii et tulemusi mõjutab milliste täitmisaegade geomeetriline keskmine võtta.

Tabel 12. Tabeli *isik_tsvektor* andmete sisestamise lausete täitmisajad millisekundites.

Indeks / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Ilma indeksita	1.37	1.30	1.48	1.47	1.48
gin	1.43	1.53	1.73	1.26	1.41
gist	2.47	1.34	1.44	1.37	1.52

Tabel 13. Tabeli *isik_tsvektor* andmete muutmise lausete täitmisajad millisekundites.

Indeks / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Ilma indeksita	1.45	1.32	1.36	1.44	11.71
gin	1.87	1.62	1.78	1.47	7.54
gist	2.38	1.41	1.60	1.43	2.12

Tabel 14. Tabeli *isik_tsvektor* andmete kustutamise lausete täitmisajad millisekundites.

Indeks / Isikute arv	**1000**	**10000**	**100000**	**1000000**	**10000000**
Ilma indeksita	1.42	1.31	1.22	1.45	7.04
gin	1.51	1.41	1.37	1.34	8.03
gist	2.25	1.31	1.39	1.40	6.27

4 Parima otsingumeetodi valimine

Ettevõtte X eesmärk on pakkuda reaalajas toimivat isikute nime alusel otsimise teenust rahvusvahelistest sanktsioonide ja riikliku taustaga isikute nimekirjadest. Ettevõtte sihiks on kasutada selle eesmärgi täitmiseks PostgreSQL'i kuna see on juba ettevõtte poolt kasutuses ning sellega ei kaasneks lisakulutusi.

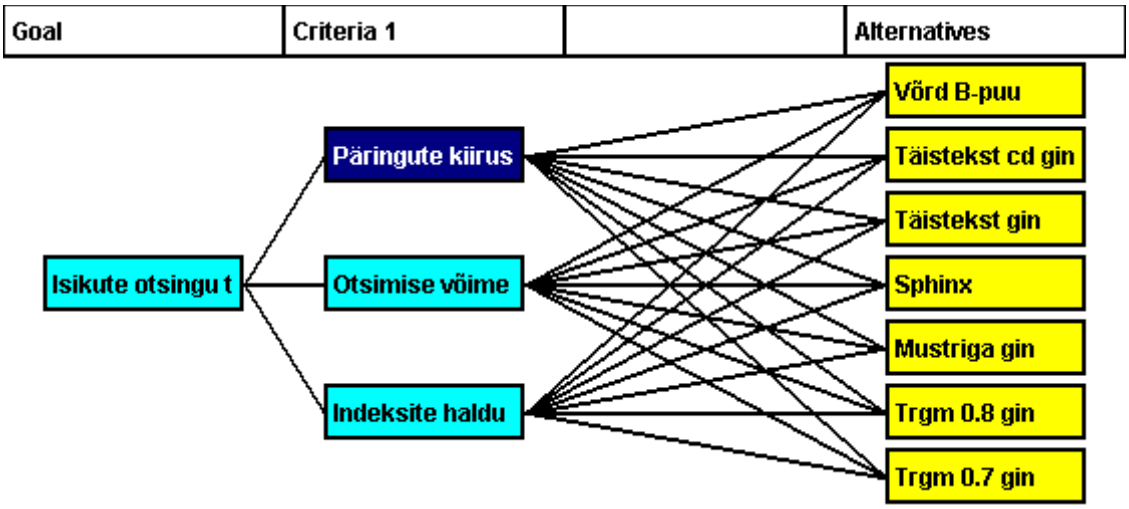
Käesolevas lõputöös on Saaty meetodi rakendamiseks kasutatud Web-HIPRE tarkvara [25]. Paaride vaheliseks võrdluseks kasutatakse Saaty fundamentaalskaalat [16].

Valitud sihi kriteeriumiteks ehk mõjuriteks on päringute kiirus, otsingumeetodi sobivus nimede otsimiseks ja indeksite haldus. Päringute kiirusi võrreldakse täitmisaegade alusel, mis on pärit Tabel 4. Otsingumeetodite sobivuste võrdlemisel on sisendiks jaotises 2.2 toodud kirjeldused ja vaadeldakse kolme omadust: meetodi sõltuvust nimede järjekorrast, ligikaudse otsingu võimalust ja võimekust järjestada tulemusi tähtsuse ehk sarnasuse alusel. Indeksite halduse aluseks on Tabel 8 olevad indeksite loomise täitmisajad. Täitmisaegade teisendamiseks Saaty fundamentaalskaalale jagati võrdluses olevate aegade suuremast arvust väiksem arv ning saadud jagatist võrreldi kahe astmelistele arvudele määratud skaalaga. See skaala vastab Saaty skaalale järgmiselt: 1 on 1, 2 on 1.5, 4 on 2, 8 on 3, 16 on 4, 32 on 5, 64 on 6, 128 on 7 ja 256 on 8. Näiteks, B-puuga võrdsusotsingu, täitmisaeg 0,25, ja Sphinxi, täitmisaeg 1,59, jagatis on 6,12. See arv on kahe astmeliste arvude 4 ja 8 keskel, mis tähendab, et selle Saaty skaala väärtus on 3,5 võrdsusotsingu kasuks.

Indeksite mahte ja andmete muutmise operatsioonide täitmiskiirust ei kasutata analüüsis, kuna eelanalüüs näitab, et need kriteeriumid ei ole ettevõttele olulised.

Võrdluses olevateks alternatiivideks on otsingumeetodid, mis kasutavad indekseid ja mille täitmisaeg 10 miljoni isikuga oli alla poole sekundi. Samuti on võrdlusest eemaldatud *gist* tüüpi indeksiga otsingumeetodite variandid kuna jaotise 3.4 tulemused näitasid, et 10 miljoni isiku korral olid need üldiselt aeglasemad kui samad otsingumeetodid *gin* indeksiga. Ka on võrdlusest eemaldatud trigrammi otsing sarnasuse

piiriga 1, sest see on aeglasem kui täistekstiotsing, ega oma muid eeliseid. Antud piirangud on valitud selliselt, et ei peaks võrdlema otsingumeetodeid, mis kindlalt ei osutuks parimaks. Eelnevalt esitatud kriteeriumite ja alternatiivide põhjal moodustatud analüütiliste hierarhiate meetodi mudel on Joonis 25.



Joonis 25. Parima otsingumeetodi valimise analüütiliste hierarhiate meetodi mudel.

Selle mudeli alusel alternatiividele määratud tähised on järgmised.

A. Võrdsusotsing B-puu indeksiga
B. Täistekstiotsing *gin* indeksiga *ts_rank_cd* järjestusega
C. Täistekstiotsing *gin* indeksiga *ts_rank* järjestusega
D. Sphinxi otsingumootor
E. Mustriga otsing *gin* indeksiga
F. Trigrammi otsing *gin* indeksiga sarnasuse piiriga 0.8
G. Trigrammi otsing *gin* indeksiga sarnasuse piiriga 0.7

Järgnevates tabelites (Tabel 15-Tabel 19) on suurima kogukaaluga tulemus **rasvases kirjas**.

Selleks, et võimaldada hinnata subjektiivsete hinnangute järjepidevust, arvutab Web-HIPRE igale võrdlusmaatriksile kooskõla määra CM (*consistency measure*). Veskioja [26] kohaselt, kui kooskõla määra väärtus on alla 0.1, siis on maatriksis tegemist kooskõlaliste hinnangutega. Web-HIPRE näitas, et kõigi võrdlusmaatriksite puhul on CM alla 0.09.

4.1 Kriteeriumite olulisus

Esitatud tingimuste kohaselt on kõige olulisem otsingu tulemuste täpsus ning seetõttu ka otsingumeetodite üldine sobivus isikute nime alusel otsimiseks. Jaotises 2.4 kirjeldatud põhjustel on kõige vähem olulisem tabeliga seotud indeksite haldus. Kriteeriumite omavahelise võrdluse tulemus on Tabel 15. Võrdluse CM oli 0.018.

Tabel 15. Kriteeriumite omavaheline võrdlus.

	Päringute kiirus	**Otsingumeetodite sobivus**	**Indeksite haldus**	**Kaal**
Päringu kiirus	1	0.5	4	0.301
Otsingumeetodite sobivus	2	1	9	**0.626**
Indeksite haldus	0.25	0.14	1	0.072

4.2 Alternatiivide omavaheline võrdlus

Tabel 16 on näha, et võrdsusotsinguga isikute otsimine on märgatavalt kiirem teistest otsingumeetoditest. Võrdluse CM oli 0.088.

Tabel 16. Alternatiivide omavaheline võrdlus päringute kiiruse alusel.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**Kaal**
A	1	1.7	1.8	2.5	5.3	8.0	8.3	**0.318**
B	0.59	1	1.1	1.5	3.7	6.4	6.8	0.209
C	0.56	0.91	1	1.4	3.5	6.1	6.5	0.196
D	0.4	0.67	0.71	1	2.8	5.3	5.7	0.152
E	0.19	0.27	0.29	0.36	1	2.5	3.0	0.065
F	0.13	0.16	0.16	0.19	0.4	1	1.2	0.032
G	0.12	0.15	0.15	0.18	0.33	0.83	1	0.029

Otsingumeetodite sobivuse võrdluseks vaadeldi Tabel 5 tulemusi, mille alusel moodustatud Tabel 17 põhjal suudavad parimat ligikaudset otsingut pakkuda trigrammi otsing koos piiranguga 0.7 ja täistekstiotsing. Võrdluse CM oli 0.071.

Tabel 17. Alternatiivide omavaheline võrdlus otsingumeetodite sobivuse alusel.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**Kaal**
A	1	0.17	0.25	0.25	0.33	0.25	0.13	0.31
B	6	1	2	2	2.5	2	0.5	0.207
C	4	0.5	1	1	1.5	1	0.33	0.115
D	4	0.5	1	1	1.5	1	0.33	0.115
E	3	0.4	0.67	0.67	1	0.67	0.25	0.081
F	4	0.5	1	1	1.5	1	0.33	0.115
G	8	2	3	3	4	3	1	**0.336**

Tabel 18 näitab, et Sphinxi ja täistekstiotsingu *gin* indeksi loomised on tunduvalt kiiremad teiste indeksite loomisest. Võrdluse CM oli 0.005.

Tabel 18. Alternatiivide omavaheline võrdlus indeksite halduse alusel.

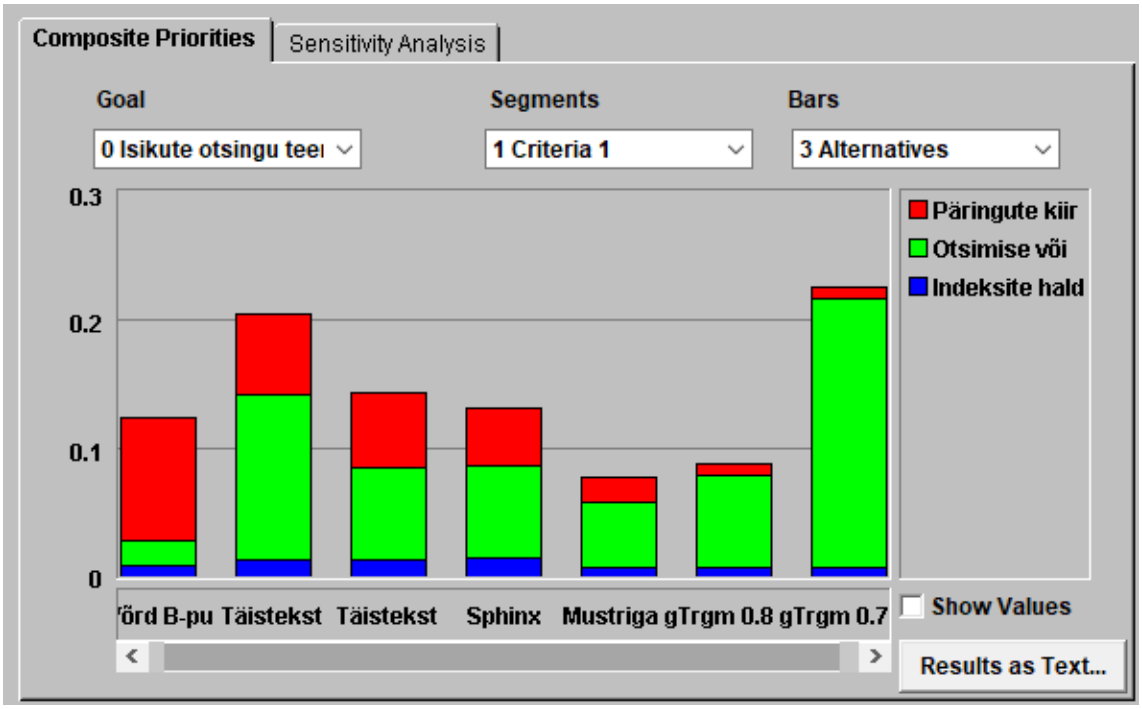
	A	**B**	**C**	**D**	**E**	**F**	**G**	**Kaal**
A	1	0.63	0.63	0.59	1.2	1.2	1.2	0.119
B	1.6	1	1	0.91	1.9	1.9	1.9	0.189
C	1.6	1	1	0.91	1.9	1.9	1.9	0.189
D	1.7	1.1	1.1	1	2.1	2.1	2.1	**0.207**
E	0.83	0.53	0.53	0.48	1	1	1	0.99
F	0.83	0.53	0.53	0.48	1	1	1	0.99
G	0.83	0.53	0.53	0.48	1	1	1	0.99

4.3 Tulemus

Tabel 19 on välja toodud alternatiivide osakaalud kriteeriumite alusel ning lisaks kokku arvutatuna nende lõplik kogukaal, mis väljendab valikute suhtelist sobivust lõpliku otsuse langetamisel sihi saavutamiseks. Joonis 26 esitab Saaty meetodi võrdluste tulemuse graafiliselt.

Tabel 19. Analüütiliste hierarhiate mooduse rakendamise lõplik tulemus.

	Päringute kiirus	**Otsingumeetodite sobivus**	**Indeksite haldus**	**Kokku**
A	0.096	0.020	0.009	0.124
B	0.063	0.130	0.014	0.206
C	0.059	0.072	0.014	0.145
D	0.046	0.072	0.015	0.133
E	0.020	0.051	0.007	0.078
F	0.010	0.072	0.007	0.089
G	0.009	0.210	0.007	**0.226**



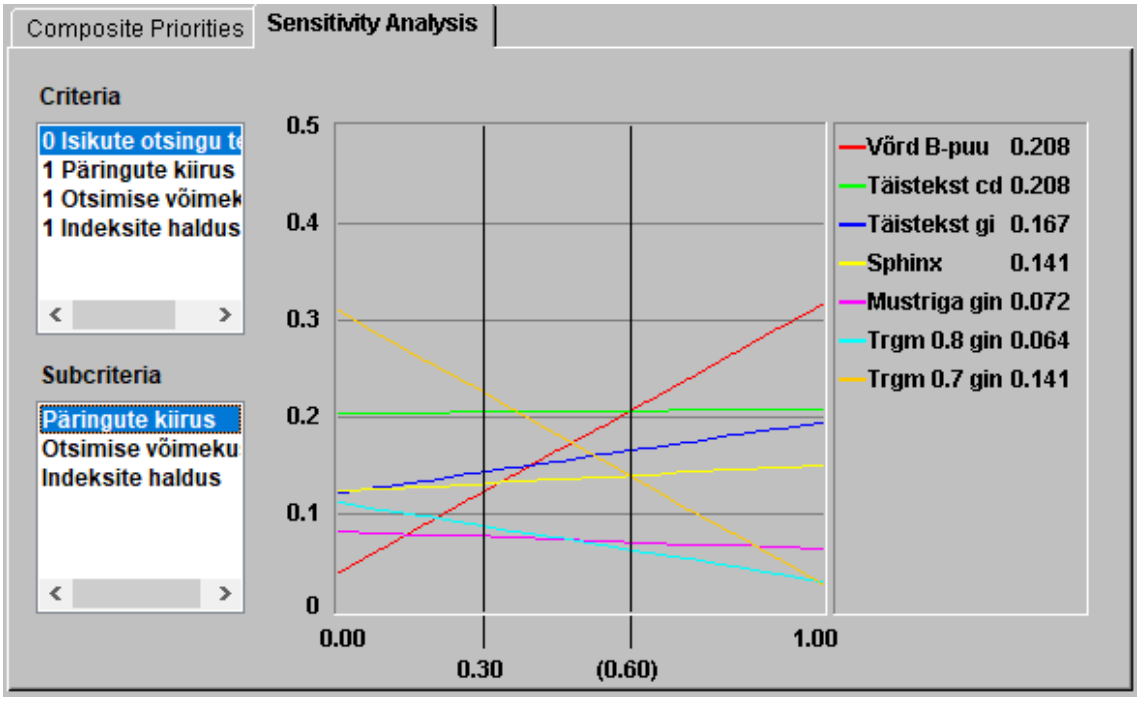
Joonis 26. Saaty meetodi tulemus graafiliselt

Suurim osakaal on trigrammi otsingul *gin* indeksiga koos määratud sarnasuse piiriga 0.7. See otsingumeetod oli küll aeglasem kui enamik teisi valikus olnud meetodeid, kuid selle eeliseks on mõõdukas otsingu tulemuste ligikaudsus, mis võimaldab leida ka otsitud isikuid kui nimedes esineb kirjavigu. Kui see omadus peaks muutuma vähem tähtsamaks, siis kogukaalude järjestuses on sarnase kogukaaluga veel täistekstiotsing *gin* indeksiga, mille tulemused on järjestatud *ts_rank_cd* meetodi alusel. See meetod on parem kui tavaline *ts_rank*, sest tulemuste järjestamisel arvestatakse lisaks ka sõnade lähedust tekstides.

Kui ettevõttes X otsustatakse kasutada soovitatud trigrammi otsingut, siis parimate tulemuste saavutamiseks oleks tegelikult vajalik kohendada trigrammi võrdluse sarnasuse piiri sõltuvalt otsitava nime pikkusest nagu soovitati jaotises 2.2.3.

4.4 Tundlikkuse analüüs

Tundlikkuse analüüs (Joonis 27) näitab, et kui vähesel määral muuta kriteeriumite osakaalu, siis suurima kogukaalu arvestuses ainult vahetuvad eelnevalt mainitud trigrammi otsingu ja täistekstiotsingu paiknevus. Näiteks, kui tõsta päringute kiiruse osakaal 0.30 pealt 0.37 või tõsta indeksite halduse osakaal 0.07 pealt 0.24 peale, siis need kaks otsingumeetodit oleksid võrdsed ja edasisel osakaalude tõstmisel mööduks täistekstiotsing trigrammi otsingust. Joonis 27 on näha, et muudest otsingumeetoditest võib ainult B-puuga võrdsusotsing saada suurima kogukaalu kui piisavalt tõsta päringute kiiruste osakaalu või alandada otsingumeetodite nimede alusel otsimise sobilikkuse osakaalu.



Joonis 27. Analüütiliste hierarhiate meetodi tulemuste tundlikkuse analüüs.

5 Kokkuvõte

Lõputöö eesmärgiks oli leida PostgreSQL andmebaasis olevate tekstiliste andmete otsinguvõimaluste hulgast ettevõttele X isikute täisnimede alusel otsimiseks sobivaim meetod ning analüüsida nende poolt kasutatavaid andmete indekseerimise viise.

Töö käigus viidi läbi eksperiment, mille jaoks seati ülesse testkeskkond ning loodi mõõtmiseks vajalike testandmete saamiseks isikunimede generaator. Eksperimendiga mõõdeti otsingumeetodite päringute täitmisaegu ning meetodite poolt kasutatavate indeksite loomise aegu ja kettamahtu. Üldiselt leiti, et *gin* indeksiga on päringud kiiremad, *gin* indeksid kasutavad vähem kettamahtu ja selliseid indeksid luuakse kiiremini võrreldes samadel veergudel kasutatud *gist* indeksitega. Eksperimendi tulemusi kasutati sisendina analüütiliste hierarhiate meetodile e Saaty meetodile. Tulemusi analüüsides leiti, et ettevõttele X sobiksid etteantud kriteeriumite alusel valikus olnud otsingumeetodite hulgast *gin* indeksiga trigrammide otsing koos sarnasuse piiriga 0.7 ning valikutest teisel kohal oli *gin* indeksiga täistekstiotsing, mis kasutab tulemuste järjestamiseks *ts_rank_cd* meetodit.

Töö eesmärk seega ka saavutati. Antud töös tehti kõik eksperimendi mõõtmised käsitsi, mis annab küll täpsemaid tulemusi, kuid ei võimalda muudatuste korral võrdlusi uuesti mõistliku ajaga läbi viia. Mõõtmiste tegemiseks oleks pidanud looma programmi, mis suudaks automaatselt läbi viia järjest mõõdetavaid päringuid ja tagastada tulemused kergesti töödeldaval kujul. Lisaks jäid otsingumeetodite puhul proovimata haruldasemad isikunimed nagu ühesõnalised täisnimed ja nimed, mis on üle paarisaja tähemärgi pikad. Kindlasti oleks ka neid erandolukordi vaja uurida, et näha, kas nende puhul otsingumeetodite tõhusus muutub.

Kasutatud kirjandus

[1] Rahapesu ja terrorismi rahastamise tõkestamise seadus. (2019). Riigi Teataja I, 13.03.2019, 126 [WWW] https://www.riigiteataja.ee/akt/113032019126 (05.05.2019)

[2] Consolidated list of sanctions. Foreign Policy Instruments, 11.12.2018 [WWW] https://data.europa.eu/euodp/en/data/dataset/consolidated-list-of-persons-groups-and-entities-subject-to-eu-financial-sanctions (19.05.2019)

[3] Nimeseadus (2019) Riigi Teataja I 22.12.2018, 15 [WWW] https://www.riigiteataja.ee/akt/122122018015 (19.05.2019)

[4] Isikunime andmisel ja kohaldamisel kasutatavate eesti-ladina tähtede ja sümbolite loetelu ning võõrkeelsete isikunimede ümberkirjutusreeglid. (2016). Riigi Teataja I, 31.05.2016, 6 [WWW] https://www.riigiteataja.ee/akt/131052016006 (19.05.2019)

[5] Jõgi, M. Mõned disainilahendused isikunimede hoidmiseks SQL-andmebaasides : bakalaureusetöö. Tallinna Tehnikaülikool, Tallinn 2016.

[6] Alice. The Uniquely Indonesian Pains of Having Only One Name. Vice, 06.06.2017 [WWW] https://www.vice.com/en_asia/article/j5xmgp/the-uniquely-indonesian-pains-of-having-only-one-name (19.05.2019)

[7] Frequently Occurring Surnames from the 2010 Census. The U.S. Census Bureau, 27.12.2016 [WWW] https://www.census.gov/topics/population/genealogy/data/2010_surnames.html (15.05.2019)

[8] Popular baby names by decade. Social Security [WWW] https://www.ssa.gov/OACT/babynames/decades/index.html (15.05.2019)

[9] Dunkerley, M, Holloway, G. The Math, Myth and Magic of Name Search and Matching. 5th ed. : SearchSoftwareAmerica, 2004.

[10] PostgreSQL 11.3 Documentation : The PostgreSQL Global Development Group, 2019 [WWW] https://www.postgresql.org/docs/11/index.html (15.05.2019)

[11] Sphinx 2.3.2-beta reference manual : Sphinx Technologies Inc, 2016 [WWW] http://sphinxsearch.com/docs/manual-2.3.2.html (16.05.2019)

[12] Doc 9303, Machine Readable Travel Documents. Seventh Edition. Part 3 — Specifications Common to all MRTDs : International Civil Aviation Organization, 2015. [WWW] https://www.icao.int/publications/Documents/9303_p3_cons_en.pdf (12.05.2019)

[13] Information and documentation -- Transliteration of Cyrillic characters into Latin characters -- Slavic and non-Slavic languages : ISO 9:1995 : International Organization for Standardization, 1995.

[14] Burke, S. Unidecode! Interglacial, 2001 [WWW] http://interglacial.com/~sburke/tpj/as_html/tpj22.html (12.05.2019)

[15] Luyken, S. After century of dispute, the German alphabet just got a new character. The Local de, 11.07.2017 [WWW] https://www.thelocal.de/20170711/after-a-century-of-dispute-the-german-alphabet-just-got-a-new-character (12.05.2019)

[16] Võhandu, L. Subjektiivsetest hinnangutest objektiivsete tulemusteni : loengukonspekt. Tallinn : Tallinna Tehnikaülikooli trükikoda, 1998.

[17] Nimede statistika. Statistikaamet, 2009 [WWW] https://www.stat.ee/public/apps/nimed/ (15.05.2019)

[18] Kirjalainen, E. Finnish names. Väestörekisterikeskus, 13.03.2019 [WWW] https://www.avoindata.fi/data/en_GB/dataset/none (15.05.2019)

[19] Fichier des prénoms. The National Institute of Statistics and Economic Studies, 02.07.2018 [WWW] https://www.insee.fr/fr/statistiques/2540004 (15.05.2019)

[20] Le classement des noms de famille. Filae [WWW] https://www.filae.com/nom-de-famille/classement-general-0-1 (15.05.2019)

[21] 10 000 Самых частых русских фамилий. Лаборатория популяционной генетики. [WWW] http://genofond.ru/genofond.ru/default2e50b.html?s=0&p=56 (15.05.2019)

[22] Information about the most popular male names among newborns. General Register Office, 11.04.2019 [WWW] https://data.mos.ru/opendata/7704111479-information-about-the-most-popular-male-names-among-newborns (15.05.2019)

[23] Information about the most popular female names among newborns. General Register Office, 11.04.2019 [WWW] https://data.mos.ru/opendata/7704111479-information-about-the-most-popular-male-names-among-newborns (15.05.2019)

[24] Most popular names and surnames. The National Statistics Institute, 29.05.2019 [WWW] http://www.ine.es/dyngs/INEbase/en/operacion.htm?c=Estadistica_C&cid=1254736177009&menu=resultados&secc=1254736195454&idp=1254734710990 (15.05.2019)

[25] Web-HIPRE. Helsinki University of Technology, 1998. [WWW] http://hipre.aalto.fi/ (17.05.2019)

[26] Veskioja, T. Lühike Web-Hipre kasutusjuhend eesti keeles. Tallinna Tehnikaülikool, Tallinn [WWW] https://maurus.ttu.ee/ained/IDN5120/doc/20/Web_Hipre_juhend.html (20.05.2019)

Lisa 1 – Sphinxist otsimiseks kasutatud Pythoni kood

```
from timeit import default_timer as timer
from urllib import urlencode
import sys, urllib2, json

# Uses Sphinx http api and SphinxQL queries to get matches
# Accepts a file with a list of names and Sphinx index name
if not sys.argv[2:]:
	print 'Needed arguments are file and index name'
	sys.exit(0)

def http_post(url, data):
	post = urlencode(data)
	req = urllib2.Request(url, post)
	response = urllib2.urlopen(req)
	return response.read()

file = sys.argv[1]
index = sys.argv[2]
api_url = 'http://localhost:9306/sql/'
names = [line.rstrip('\n') for line in open(file)]
query = "SELECT id FROM " + index + " WHERE MATCH('%s') LIMIT 0, 100"
result = {}
data = {}
start = timer()

i = 0
for name in names:
	data['query'] = query % name
	response = json.loads(http_post(api_url, data))
	result[i] = response["matches"]
	i += 1

end = timer()
print 'Time taken for queries: %f' % (end - start)

count = 0
with open('sphinx_result.csv', 'w') as write_file:
	for i, matches in result.iteritems():
		for match in matches:
			write_file.write('%d,%s\n' % (i, match[0]))
			count += 1

print 'Matches found: %d' % count
```

Lisa 2 – Andmebaasi struktuuri loomise SQL laused

```
CREATE EXTENSION pg_trgm;
CREATE EXTENSION btree_gist;
CREATE EXTENSION btree_gin;
CREATE EXTENSION fuzzystrmatch;
CREATE TABLE isik (
	id SERIAL,
	taisnimi TEXT NOT NULL,
	CONSTRAINT pk_isik_id PRIMARY KEY (id)
);
CREATE TABLE isik_alias (
	isik_id INTEGER NOT NULL,
	alias TEXT NOT NULL,
	CONSTRAINT pk_isik_alias PRIMARY KEY (isik_id, alias),
	FOREIGN KEY (isik_id) REFERENCES isik (id)
);
CREATE TABLE isik_tsvektor (
	isik_id INTEGER,
	tsv TSVECTOR NOT NULL,
	CONSTRAINT pk_isik_tsvektor PRIMARY KEY (isik_id),
	FOREIGN KEY (isik_id) REFERENCES isik (id)
);
CREATE TABLE isik_normaliseeritud_nimi (
	id INTEGER,
	isik_id INTEGER NOT NULL,
	taisnimi TEXT NOT NULL,
	CONSTRAINT pk_isik_normaliseeritud_nimi PRIMARY KEY (id),
	FOREIGN KEY (isik_id) REFERENCES isik (id)
);
CREATE INDEX ak_isik_normaliseeritud_nimi_isik_id ON
isik_normaliseeritud_nimi USING btree(isik_id);
/* INDEXES */
/*CREATE INDEX ak_isik_tsvektor_tsv_gist ON isik_tsvektor USING gist(tsv);*/
/*CREATE INDEX ak_isik_tsvektor_tsv_gin ON isik_tsvektor USING gin(tsv);*/
/*CREATE INDEX ak_isik_normaliseeritud_nimi_taisnimi_btree ON
isik_normaliseeritud_nimi USING btree (taisnimi);*/
/*CREATE INDEX ak_isik_normaliseeritud_nimi_taisnimi_btree_gist ON
isik_normaliseeritud_nimi USING gist(taisnimi);*/
/*CREATE INDEX ak_isik_normaliseeritud_nimi_taisnimi_gin ON
isik_normaliseeritud_nimi USING GIN (taisnimi gin_trgm_ops);*/
/*CREATE INDEX ak_isik_normaliseeritud_nimi_taisnimi_gist ON
isik_normaliseeritud_nimi USING GIST (taisnimi gist_trgm_ops);*/
```

Lisa 3 – Isikunimede generaatori näidisväljund

```
COPY public.isik (id, taisnimi) FROM stdin;
117284569    Jacques Capdeville
117284570    Paige Basil
117284571    Мира Шубников
117284572    Laurie Bouillet
117284574    Brandon Zerbe
117284575    Малика Мизеркин
117284576    Всеволод Марк Эберлейн
117284577    Georges Lafont
117284579    Paul Casey
117284580    Léon Vachon
\.

COPY public.isik_alias (isik_id, alias) FROM stdin;
117284570    Kathleen Basil
117284576    Ибрагим Пилюгин
117284580    Lori Sabrina Baldwin
117284580    Vickie Jennifer Valentine
\.

COPY public.isik_normaliseeritud_nimi (id, isik_id, taisnimi) FROM stdin;
1      117284569    jacques capdeville
2      117284570    paige basil
3      117284570    paige bridges
4      117284570    kathleen basil
5      117284571    mira shubnikov
\.

COPY public.isik_tsvektor (isik_id, tsv) FROM stdin;
117284569    'capdeville':2 'jacques':1
117284570    'basil':2,6 'bridges':4 'kathleen':5 'paige':1,3
117284571    'mira':1 'shubnikov':2
117284572    'bouillet':2 'laurie':1
\.
```