TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDU40LT

Mairi Jõgi 134496IAPB

MÕNED DISAINIMUSTRID ISIKUNIMEDE HOIDMISEKS SQL-ANDMEBAASIDES

Bakalaureusetöö

Juhendaja: Erki Eessaar
Doktor
Dotsent

Tallinn 2016

Autorideklaratsioon

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

Autor: Mairi Jõgi

23.05.2016

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on kirjeldada disainimustreid isikunimede hoidmiseks SQL-andmebaasis, disainida näited, realiseerida mõne mustri põhjal päringute tegemiseks näiteandmetega andmebaas ja saadud kogemust analüüsida.

Töös uuritakse maailmas eksisteerivaid isikunimede struktuure. Neid arvesse võttes ning olemasolevatele mustritele ja soovitustele põhinedes koostatakse kataloog, kus kirjeldatakse mustreid isikunimede hoidmiseks SQL-andmebaasis ning tuuakse välja nende tugevaid ja nõrku külgi.

Töö tulemuseks valmisid mustri formaadis kirjeldatud kümme disainilahendust ja näiteandmebaasi tabelite diagrammid. Nelja mustri puhul need realiseeriti, tehti näitepäringuid ning esitati järeldused.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 55 leheküljel, 7 peatükki, 21 joonist, 4 tabelit.

Abstract

Some Design Patterns for Keeping Personal Names in SQL Databases

The goal of this thesis is to describe design patterns for keeping personal names in SQL databases, design examples, create an example database based on selected patterns, and analyze the experience.

To achieve the goal, the author firstly examines the structures of personal names around the world. Taking into account the results of this analysis, and based on existing patterns and suggestions, the author creates a catalogue of possible design patterns. The thesis will then analyze the different strengths and weaknesses of those design solutions.

The most important results of this study are the following.

- Specification of ten design solutions in the pattern format, including physical database design diagrams of example database.
- Implementation of four designs in PostgreSQL and sample queries based on the tables.
- Analysis of strengths and weaknesses of each solution as well as general recommendations.

The author of this thesis found that it is not possible to determine, which of the patterns is the best. It depends on the requirements on the particular information system. One can combine and extend the patterns presented in the work in multiple ways. Keeping in mind that personal names vary widely around the world, the author suggests trying to avoid separating different name types of persons' full names. In case one needs the separation, it is not sufficient to only use two attributes – given name and surname.

Several important aspects were left out of the scope of this work. The use of titles in personal names and performance comparison of queries in each pattern are some of the examples that need further investigation.

The thesis is in Estonian and contains 55 pages of text, 7 chapters, 21 figures, 4 tables.

Lühendite ja mõistete sõnastik

Antimuster	*Antipattern* Muster, mis kirjeldab probleemile ebasobivat ja halba lahendust.
ASCII	*American Standard Code for Information Interchange* Numbrilised koodid vahemikus 0-127, mis esitavad ingliskeelseid tähemärke, numbreid, keelemärke ja klaviatuuri funktsioone.
CASE	*Computer-Aided Software Engineering* Arenduskeskkond, mis võimaldab projekteerida, disainida ja luua infosüsteeme ja tarkvara.
FHIR	*Fast Healthcare Interoperability Resources* Standardi mustand, mis kirjeldab elektrooniliste terviseandmete vahetamise andmetüüpe, elemente ja liidest [11].
Foneetiline nimi	Häälduslik, häälduspärane kuju isikunimest.
Kuues normaalkuju	*Sixth normal form (6NF)* Normaalkujud on andmebaasi andmestruktuuride organiseerimisreeglid, mis võimaldavad ära hoida anomaaliaid andmete seostamisel ja käsitlemisel ning võimaldavad parandada andmestruktuuride arusaadavust. Tabel on kuuendal normaalkujul, kui teda ei ole võimalik kadudeta tükeldada. See olukord tekib, kui tabelis on ainult üks kandidaatvõti ja lisaks maksimaalselt üks veerg, mida kandidaatvõti ei hõlma.
NULL	Marker SQL-is, mis tähistab andmete puudumist.
PostgreSQL	Populaarne, tasuta ning vabatarkvarana pakutav objekt-relatsiooniline andmebaasisüsteem, kus saab kasutada andmebaasikeelt SQL.
Romaniseeritud nimi	*Romanized name* Romaniseeritud nimi on ladina kirjamärkidega ümberkirjutatud nimi. Näiteks vene mehe Никита romaniseeritud nimi on Nikita.
SQL	*Structured Query Language* Andmebaasikeel andmebaasis olevate andmete, andmestruktuuride ning neid ümbritsevate muude andmebaasiobjektide ja andmebaasi pääsuõiguste haldamiseks. SQL-andmebaas on antud töös andmebaas, mille loomisel on kasutatud andmebaasisüsteemi, milles kasutatakse SQL-i.

Tabel	*Table, base table* SQL-andmebaasi põhiline ehitusplokk, mida ei ole defineeritud teiste tabelite põhjal.
Transkriptsioon	*Transcription* Keelte vaheline foneetilise teisendamise meetod, mis annab häälikuid edasi sihtkirja reeglite ja kirjasüsteemi abil. Antud töös mõistetakse transkriptsioonina isikunimede romaniseerimist, nende ümberkirjutamist ASCII-sse ja foneetiliseks kujuks.
UML	*Unified Modeling Language* Üldotstarbeline (paljudes eri valdkondades kasutatav), visuaalne ja standardiseeritud modelleerimiskeel tarkvara- ja infosüsteemide projekteerimiseks.
Unikood	*Unicode* Rahvusvaheline standard arvutites kirjasüsteemide kodeerimiseks.
UTF-8	*Unicode Transformation Format* Kaardistus unikoodi kirjapanekuks. UTF-8 puhul on ühe tähemärgi kirjapanekuks tarvis minimaalselt 8 bitti.
W3C	*The World Wide Web Consortium* Rahvusvaheline organisatsioon, mis arendab veebi pikaajalise ning jätkusuutliku arengu tarvis protokolle, juhtnööre ja standardeid.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Bakalaureusetöö teemaks on uurida erinevaid SQL-andmebaasi disainilahendusi andmebaasis isikute nimede hoidmiseks. Isiku nimi on tema identiteedi tähtis osa ja on oluline, et seda käsitletakse korrektselt. Globaliseeruvas maailmas tekib üha enam vajadus infosüsteemide järele, mis peavad toime tulema erinevate rahvaste isikuandmetega. Endiselt on väljakutseks, kuidas disainida infosüsteeme, mis kohanevad inimeste järgi, selle asemel, et sundida inimesi kohanema infosüsteemidega.

1.1 Taust ja probleem

Väga paljudes andmebaasides ja rakendustes vajatakse isikunimede salvestamist. Nimed aga varieeruvad erinevates riikides, kultuurides ja ajajärkudes palju – on erinev arv eesnimesid, erinev arv perenimesid, ees- ja perenime asemel ainult üks nimi, väga pikad nimed, lisaks ees- ja perenimele on oluline isanimi, nimed muutuvad aja jooksul jne. Ühtset standardit isikunimede puhul maailmas ei ole. Seega on oluline teada, millise struktuuri ja piirangutega peaks olema isikunimede hoidmiseks mõeldud andmebaas.

Antud töö on mõeldud kõikidele, kes tegelevad andmebaasidega, kus hoiustatakse eri rahvaste isikunimesid, sõltumata andmebaasi aluseks olevast andmemudelist (nt SQL, relatsiooniline, XML, dokumendipõhine). SQL-andmebaasidega toimetajatele annab see otseseid tehnilisi juhtnööre. Isikunimede keerukuse mõistmise seisukohast võiks see olla kasulik kõigile infosüsteemide arendamisega tegelejatele ning töö teine peatükk kõigile, kelle töö osaks on eri kultuuridest pärit isikutega suhelda.

Esitatud andmebaasi füüsilise disaini diagrammid on disainitud CASE-vahendis Enterprise Architect ning näiteandmebaasid on realiseeritud PostgreSQL 9.4 andmebaasisüsteemis.

1.2 Ülesande püstitus

Töö põhieesmärkideks on:

- Leida ja panna kirja mõningad disainimustrid isikunimede hoidmiseks SQL-andmebaasis. Eesmärgiks ei ole kirja panna ammendavat nimekirja (näiteks on võimalik erinevaid pakutavaid lahendusi kas kombineerida või universaallahendust laiendada).
- Realiseerida mõne mustri põhjal näiteandmetega andmebaas ja teha päringuid.
- Saadud kogemuste põhjal disaine võrrelda ning analüüsida erinevate lahenduste eeliseid ja puuduseid.

1.3 Metoodika

Töös kasutatakse disainiteaduse (*design science*) metoodikat [21]. Eesmärkideni jõudmiseks analüüsitakse kõigepealt isikunimede struktuure ja uuritakse olemasolevat kirjandust. Töö tulemusena valmivad disainimustrid (artefaktid ehk tehised). Kirjeldatud mustrite põhjal luuakse CASE-vahendis andmebaasi füüsilise disaini diagrammid ning valikuliselt neid ka realiseeritakse ja katsetatakse konkreetsete päringutega.

1.4 Ülevaade tööst

Käesolev töö koosneb viiest suuremast sisupeatükist. Ülevaade erinevate rahvaste nimede varieeruvusest antakse peatükis „Isikunimede mitmekesisus ja keerukus“. Sellele järgneb peatükk „Isikunimede käsitlemine andmebaasides kirjanduse põhjal“, kus antakse lühike ülevaade, kuidas ja mil määral on antud teemat käsitletud olemasolevas kirjanduses. „Mustrid, nende struktuur ja realiseerimine“ kirjeldab mustreid üldisemalt, esitab antud töös kasutatava mustri struktuuri ja käsitleb mõningaid aspekte mustrite realiseerimise tehnilisest poolest. Töö põhitulemused on kõige mahukamas peatükis „Mustrite kataloog“. Lõpetuseks annab autor peatükis „Autori soovitused mustrite kasutamiseks“ omalt poolt nõu, kuidas töö tulemusi kasutada ja millele lisaks tähelepanu pöörata.

2 Isikunimede mitmekesisus ja keerukus

Isikunimede traditsioonid on maailmas väga erinevad. Enne mustrite koostamise alustamist võttis autor eesmärgiks välja selgitada, kas leidub korduvaid elemente või sarnaseid struktuure ning mida saaks isikunimede puhul eeldada ja mida mitte. Antud peatükk annab selles osas kokkuvõtliku ülevaate.

Märkimisväärseks impulsiks antud tööle on Patrick McKenzie' [19] blogipostitus, kus esitatakse nimekiri aspektidest, mida isikunimede puhul eeldada ei saa. Nimekiri on üllatavalt pikk, kus alustatakse väärarusaamadest, et inimesel on kindlasti ees- ja perenimi, et eesnimi on järjekorras alati esimene ja perenimi alati viimane nimi, et need on erinevad[1], et need ei sisalda numbreid[2], et nimel võib eeldada teatud pikkust[3], et nimi antakse koheselt pärast sündi[4], kuni lõpetades sellega, et inimesel üldse eksisteerib nimi.

Järgnevas tabelis (Tabel 1) tuuakse välja erinevates riikides ja kultuurides leiduvad nimede struktuurid [1]. Ruumi kokkuhoiu mõttes on asendatud tabelis sõna *nimi* lühendiga *n*. Antud töö keskendub erinevatele nimetüüpidele nimes nii, et ei loe korduvaid nimetüüpe erinevateks osadeks. Näiteks isiku nimi Karl Martin Sinijärv jaotatakse struktuuriks *eesn* + *peren*, mitte aga *eesn* + *eesn* + *peren*. Lisakeerukust toob teemasse tiitlite kasutamine isikunimede osana. Kuna aga bakalaureusetöö maht on piiratud, siis on tiitlid siit teadlikult välja jäetud. Töö Lisast 1 võib leida veel tabeleid, mis nimede variatsioone illustreerivad [1]. Tabelid pole kõikehõlmavad ja lõplikud. Need on üksikud näited, mis on esitatud eesmärgiga anda sissejuhatuseks ettekujutus isikunimede keerukusest ja variatsioonide ulatusest.

[1] ameeriklane Emery Emery [13]
[2] ameeriklane 3ric Johanson [7]
[3] Pablo Picasso täisnimi oli Pablo Diego José Santiago Francisco de Paula Juan Nepomuceno Crispín Crispiniano de los Remedios Cipriano de la Santísima Trinidad Ruiz Picasso.
Pikimate nimede näited aga ulatuvad veelgi kaugemale: Saksamaalt pärit mehe täisnimi koosnes 26 osast ja üksnes tema perekonnanimi oli 666 tähemärki pikk, inglasest naise täisnimi koosneb 161 osast ja kokku 898 tähemärgist, Rootsis elava noormehe täisnimi koosneb 63 osast ja kokku 372 tähemärgist [28]
[4] olümpiavõitja Picabo Street valis endale ise nime alles 3aastaselt [22]

Tabel 1. Nimede struktuurid erinevates riikides ja kultuurides.

Riik/ kultuur	USA, UK	Island	India	Afganistan	Pakistan (moslem, mees)	India (2), Sri Lanka	Hiina, Korea
Nime struktuur	**eesn + keskmine n + peren + generat-siooni n**	**eesn + isan/eman**	**eesn + abikaasa eesn + peren**	**eesn + (peren)**	**religioosne n + eesn + (peren) *igas järjekorras**	**kohan + isan + eesn**	**peren + generat-siooni n (võib omada) + eesn**
Täisnimi	John P. Bitt Jr.	Ólafur Jónsson	Nita Sanjay Vadgama	Hamid Shah	Muhammad Hafiz Khan	Kuppali Venkatappa Puttappa (K. V. Puttappa)	毛澤東 (Mao Ze Dong)
Eesnimi	John	Ólafur	Nita	Hamid Shah	Hafiz	Puttappa	東
Keskmine nimi	P.						
Isanimi/ emanimi		Jónsson				Venkatappa	
Abikaasa eesnimi			Sanjay				
Religioosne nimi					Muhammad		
Kohanimi						Kuppali	
Perekonna-nimi	Bitt		Vadgama		Khan		毛
Generatsiooni nimi	Jr.						澤
Eesnimi romani-seeritud							Dong
Perekonna-nimi romani-seeritud							Mao
Generatsiooni nimi romani-seeritud							Ze

Esimene suurem erinevus on nimede järjekord – väga paljudes kultuurides on isiku perekonnanime asukoht nimes esikohal ja eesnimi hoopiski viimasel kohal. Afganistan on näide selle kohta, et isikunimes võib olla ainult üks nimetüüp – eesnimi.

Mitmetes kultuurides on nime oluline osa isanimi (või ka emanimi). Selle nimetüübi positsioon nimes võib aga olla erinev – Islandil viimase nimena, vene nimedes pigem keskmise nime rollis, Indias hoopiski esimese nimena (Tabel 4). Oluline on märkida, et näiteks Islandil pole isanimi perekonnanime rollis – isiku poole pöördutakse kas eesnime või täisnime järgi, aga mitte „härra Jónsson“. Ka telefoniraamatud on sorditud

eesnime järgi [14]. Kreekas ja Indias võetakse abielludes nime osaks ka abikaasa eesnimi (Kreekas isanime asemele).

Läänekultuurile võõramad nimetüübid on näiteks religioossed nimed, aunimed ja kohanimed. Eelkõige moslemite seas on meeste puhul kasutusel religioosne nimi, vastav nimetüüp naiste puhul on aunimi (Tabel 3). Meeste religioosne nimi käib tavapäraselt kõige ette, naiste aunimi pärast eesnime, kuid neid võib kohata igas võimalikus järjekorras. Indias ja Sri Lankal võib nime osana olla kohanimi, mis koos isanimega võivad olla märgitud ka initsiaalidena.

Hiinas paigutatakse generatsiooni nimi erinevalt kui lääne traditsioonis. Nimelt generatsiooni nimed Jr. ja Sr. käivad tavapäraselt nime lõppu, aga Hiina vaste pannakse perenime ja eesnime vahele.

Lisakeerukust toovad teemasse erinevate kultuuride tähestikud, mistõttu võib osutuda vajalikuks hoida andmebaasides nimede transkriptsioone. Tabelis 1 on ühe näitena toodud hiina nimi, kuid samamoodi erineb nime originaalkuju romaniseeritud kujust tabeli India, Afganistani ja Pakistani veergudes.

Töö Lisas 1 leidub veel tabeleid nüanssidega, millest tõstetakse siinkohal mõned esile.

Hispaaniakeelsete kultuuride perenimed (Tabel 2) on kombinatsioonid isa ja ema (või ka teiste esivanemate) perenimede osadest. Hispaanias käib isa perenimi ettepoole kui ema oma, Brasiilias ja Portugalis võib järjekord olla pigem vastupidine. Lisanüansid tekivad isiku poole pöördumisel. Isiku Juan Pablo Fernández de Calderón García-Iglesias poole pöördutakse Señor Fernández de Calderón, aga mitte Señor García-Iglesias [14].

Andmebaasides nimede sortimisel tekivad keerukused, kui nimedes on eesliited nagu van, de, le jne. Perenime van der Burgh tuleks sortida b-, mitte v-tähe järgi (Tabel 2).

Indias suhtutakse nimedesse hoopiski paindlikumalt – perenime võib ära jätta ja kasutada keskmist nime perenime rollis (Tabel 4).

Võttes arvesse eri rahvaste isikunimede mitmekesisust, uuritakse töö järgmises peatükis, milliseid lahendusi pakub olemasolev kirjandus selleks, et andmebaasi disainimisel niivõrd laia variatsioonide hulgaga toime tulla.

3 Isikunimede käsitlemine andmebaasides kirjanduse põhjal

Vaadates läbi olemasolevat kirjandust ja tehes päringuid[1], ilmnes suuremalt jaolt sarnane lähenemine. Nimelt mustrites, mida kasutati isikunimede salvestamiseks, oli kaks välja – *ees*- ja *perenimi*. Seega võttes arvesse eelnevat infot isikunimede variatsioonide hulga kohta, võib öelda, et olemasolev kirjandus vaatab teemale pealiskaudselt ja läänekeskselt. Tegelikkuses on isikunimed oluliselt keerulisemad.

Tuleb märkida, et otsingute tegemisel kasutas autor eesti- ja ingliskeelseid päringuid ning uuris kirjandust vastavates keeltes. Võimalik, et ka sellest tulenevalt olid tulemused läänekesksed. Kuid arvestades, et inglise keel on küllaltki rahvusvaheline keel, siis võiks olla isikunimede käsitlemine selles keeles ka eri rahvusi arvesse võttev.

Dünaamilisema lahenduse pakkus Silverston [27], kes kasutas isikunimede hoidmiseks universaalset disainilahendust. Selline lahendus võimaldab määrata isikule mitmeid erinevaid nimetüüpe ja ka salvestada nimede muutumist ajas. Universaalset disainilahendust analüüsitakse pikemalt käesoleva töö „Mustrite kataloogis“.

Autor uuris mitmeid foorumeid ja blogipostitusi, kus arutleti eri rahvaste isikunimede salvestamise teemadel. Palju oli seisukohti, et lahendus sõltub sellest, milleks on vaja nimesid salvestada. Juhul, kui nime kasutatakse ainult ekraanil kuvamiseks (e-kirjas pöördumine või tuvastatud kasutaja kuvamine), selle järgi ei otsita, ei sorteerita, ei tehta statistikat, siis võiks kasutada ainult ühte välja – täisnime. Pidades silmas nimede keerukust, soovitatakse võimalusel igasuguseid nimeosade tükeldusi ja kitsendusi vältida. Vajadusel lisada ka kuvanimi, mille puhul isik saab ise määrata, kuidas tema poole pöördutakse. Juhul, kui aga siiski on vaja nime erinevaid osasid eristada, tuleks väga põhjalikult läbi mõelda, milliseid osasid vaja oleks ja milleks.

[1] Otsingu märksõnu : „database desing for personal names“, „database international names“, „database for storing personal names“ jm.
Andmemudelite kataloogist mudelid, milles hoitakse isikuandmeid (k.a isikunimed) [4].

4 Mustrid, nende struktuur ja realiseerimine

Antud töö peatükis „Mustrite kataloog“ esitatakse mustreid, mis kirjeldavad isikunimede hoidmist SQL-andmebaasides. Enne veel määratletakse aga mustri mõiste ja autori poolt kasutatava mustri struktuur.

Muster üldises mõistes on millegi kordus või seaduspära. Käesoleva töö kontekstis on muster probleemi ja selle lahenduse struktuurne kirjeldamise viis. Mustri abil esitatakse nii probleem, lahendus kui ka lisateave, mis aitab valida probleemile õige lahenduse ja annab eeskuju lahenduse kasutamiseks [10].

Mustri üks olulisimaid omadusi on taaskasutatavus. Selleks, et luua enda jaoks parim lahendus, on mõistlik ära kasutada olemasolevaid teadmisi, kuna need hõlmavad nii parimaid valikuid, kui ka teadmisi ja kogemusi halbadest lahendustest. Disainimuster tarkvaraarenduses kirjeldab üldist taaskasutatavat lahendust tihti korduvale tarkvaraarenduse probleemile [29].

Töös esitatakse kogum struktureeritud disainilahendusi. Mustrite kirjeldamisel kasutatakse ühist struktuuri, mille aluseks on võetud Rõzova [25] ja Vellemaa [29] bakalaureusetööd. Nimetatud töödes kasutati struktuuri, mis on sobiv ka käesoleva töö jaoks. Mustri struktuuri osad:

- Nimi, mis kirjeldab mustrit võimalikult täpselt. Kuna iga mustri nimi on esitatud peatüki „Mustrite kataloogi“ alampeatükkide pealkirjades, siis mustri nimesid enam mustris ei korrata.
- Probleem, mida see muster lahendab. Antud töös on probleem kõikidel disainimustritel sarnane: kuidas hoida isikunimesid SQL-andmebaasides nimekultuuride variatsioonidest ja erinevustest tulenevat keerukust arvesse võttes. Seetõttu probleemi „Mustrite kataloogis“ uuesti ei esitata.
- Taust, millal nimetatud probleem võib tekkida. Antud töös on taust samuti kõikidel disainimustritel sarnane: uue SQL-andmebaasi disainimisel või olemasoleva edasiarendamisel on vajalik otsustada, mil viisil isikunimesid andmebaasis esitada. Seetõttu tausta „Mustrite kataloogis“ uuesti ei esitata.

- Jõud, mis võivad mõjutada probleemi või selle lahendust.
- Lahendus probleemile. Need ignoreerivad teisi võimalikke veerge, mida tabel *Isik* võib sisaldada, peale unikaalse identifikaatori *isiku_id*.
- Allikad ja autorid, mis demonstreerivad selle mustri kasutamist ning kelle soovitusel on käesoleva töö autor mustreid kasutanud. (Võib ka puududa, kuid sellisel juhul oleks vastavat mustrit õige nimetada kandidaatmustriks, sest lahenduse headuse kohta pole veel piisavalt praktilisi tõendeid.)
- Lahenduse mõistlikkus, mustri nõrgad ja tugevad küljed.
- Variatsioonid antud mustrile. (Võib ka puududa.)
- Näide. PostgreSQL-is realiseeritud disaini puhul ka näitepäringud. Juhul, kui muster on ülem-muster, siis esitatakse näited ja näitepäringud alam-mustrites.

4.1 Andmebaaside PostgreSQL-is realiseerimise tehnilisi küsimusi

Peatükis „Mustrite kataloog“ esitatud tabelite füüsilise disaini diagrammid on loodud PostgreSQL andmebaasisüsteemi silmas pidades ning mõned näited on realiseeritud PostgreSQL 9.4 versioonis. Käesolevas peatükis keskendutakse lühidalt selle andmebaasisüsteemi spetsiifikale ning mustrite realiseerimise tehnilisele küljele.

Edaspidistes kirjeldustes tähendab tabel andmebaasi baastabelit. Baastabelite põhjal saab luua vaateid (virtuaalseid tabeleid), mis isikunimede andmeid erinevatele kasutajatele erinevalt esitlevad.

„Mustrite kataloogis“ on näitetabelite diagrammides valitud nimede andmetüübiks PostgreSQL-is pakutav *text*[1], sest isikunimede võimalik pikkus on väga varieeruv, et tabelite loomisel määrata väljale kindel pikkus (näiteks inglasest naise täisnimi koosneb 161 osast ja kokku 898 tähemärgist), ning katsetustele [5] põhinedes ei ole *varchar(n)* ja *text* tüüpide kasutamisel jõudluse poolest märgatavat vahet. Tüübi *text* kasutamisega kaasnevad aga ka ohud. Kuna pikkuse piirangut sisuliselt ei ole, on kasutajal vaba voli sisestada välja mida iganes – nimetüübi välja on võimalik panna täisnimi või hoopiski nime asemel pikemaid kommentaare.

Lisaks kindla pikkuse määramisele on isikunimede puhul üleüldiselt kitsenduste rakendamine komplitseeritud. Näiteks juhul, kui on teada, et andmebaasis hoitakse vaid

[1] Piiranguta varieeruva pikkusega tekstiväli. Maksimaalne välja suurus on umbes 1GB [23].

selliste inimeste andmeid, kelle kultuuris on numbrid nimedes keelatud, saab kirjutada kitsenduse, et nimes ei tohi olla numbreid. Kui aga nimede kultuuriruum ei ole piiratud, siis seda ja ka muid kitsendusi enam eeldada ei saa. Mustrite „Nimetüüpide tabelid“ ja „Universaallahendus“ näidetes on loodud kaks kitsendust – üks kontrollib, et nimi ei koosneks tühikutest ja teine kontrollib nime pikkust. Viimane on esitatud pigem demonstreerimiseks, et kui tõepoolest on vaja määrata nimele maksimumpikkust, saab seda mugavalt teha domeeni kasutades. Domeeni kaudu on hiljem võimalik pikkust ka hõlpsalt (ühest kohast) muuta. Üldiselt antud töö põhjalikumale kitsenduste uurimisele ei keskendu.

Isikunimede puhul tuleks kindlasti kasutada unikoodi, kuna see hõlmab enamikku suuremaid kirjasüsteeme (ladina, kirillitsa, kreeka, araabia, heebrea, hiina, korea, jaapani jm kirjamärgid) [24]. PostgreSQL-is on võimalik serverit seadistada nii, et kõigil loodavatel andmebaasidel on unikoodi UTF-8 kodeering ning seda saab määrata ka individuaalselt iga andmebaasi loomisel.

Kuuenda normaalkuju tuntud probleemiks on andmebaasi struktuuri keerukuse kasv. Mustri „Nimetüüpide tabelid“ realiseerimise märkimisväärne miinus on, et suure hulga nimetüüpide puhul muutub päringute tegemine üle tabelite kiiresti keeruliseks ja mahukaks (palju tabeleid, mida ühendada). Kuuenda normaalkuju tabelite kasutamise muudab efektiivsemaks ankurmodelleerimine [2] ja tabeli elimineerimise teisendus [18], mis aitavad parandada päringute jõudlust ning millest viimast PostgreSQL ka toetab. Päringute jõudlust käesolevas töös ei uurita ning seega siinkohal ei peatu pikemalt nende tehnikate sisul ja lahtiseletamisel. Hea ülevaate teemast saab näiteks Saali magistritööst [26].

Nimede ajalise kehtivuse säilitamiseks on „Mustrite kataloogis“ näidetes kasutatud veerge *algusaeg* ja *loppaeg,* kus mõlema andmetüübiks on *date*. See pole aga ainuke võimalus. PostgreSQL pakub andmetüübiks ka vahemikku (*range types*), sh *DATERANGE (date)* [16]. Vahemiku tüübi sisestamisel võib jätta ka ühe vahemiku otstest määramata – näiteks kehtiva isikunime sisestamisel puudub kehtimise lõpuaeg. Kasutades vahemiku tüüpi ja *EXCLUDE* kitsendust, saab lihtsalt jõustada kitsenduse, et nimetüübi tabelis ei tohi samal isikul olla mitu rida, kus alguse ja lõpu vaheline intervall osaliselt või täielikult kattub.

5 Mustrite kataloog

Järgnevad mustrid on disainilahendused isikunimede hoidmiseks SQL-andmebaasides.

5.1 Nimetüüpide veerud

Antud muster on üldisem nimetus järgnevatele alam-mustritele. Kõik, mis siin välja tuuakse, kehtib ka alam-mustrite puhul.

Jõud:

- Arvatakse, et üks ühine tabel on lihtsam ja universaalsem lahendus.
- Andmebaasis ei ole vaja hoida isikunimede ajalugu.

Lahendus: Luuakse üks tabel, kuhu lisatakse kõik nimetüübid kui tabeli veerud.

Mustri tugevad küljed: Lahenduse realiseerimine andmebaasi loomisel on kiire – vajalik on luua ainult üks tabel.

5.1.1 Kõikvõimalike nimetüüpide veerud

Jõud:

- Andmebaasis on vajalik hoida kõikvõimalikke nimetüüpe.
- Andmebaasis on vajalik lisaks nimele hoida ka nime transkriptsiooni (nt romaniseeritud kuju).
- Suur hulk nimetüüpe, iga jaoks eraldi tabel luua ei tundu otstarbekas.

Lahendus: Luuakse üks tabel, kuhu lisatakse kõikvõimalikud nimetüübid ning nimede transkriptsioonid kui tabeli veerud (Joonis 1).

Mustri tugevad küljed:

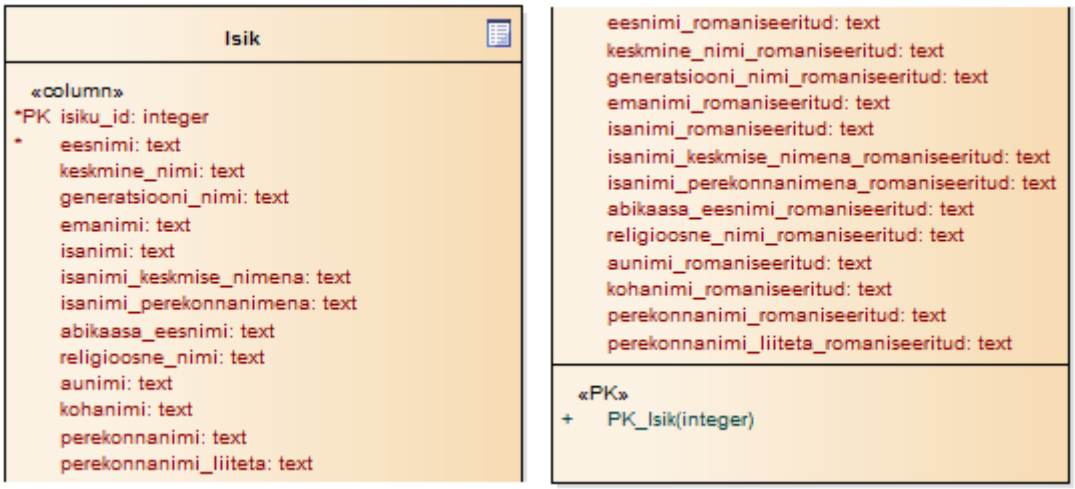
- Suur nimetüüpide hulk ei tekita andmebaasi suurel hulgal nimetüüpide tabeleid.
- Võimaldab detailselt erinevates kultuurides eksisteerivaid nimetüüpe salvestada.
- Nime osasid on lihtne andmebaasist küsida.

- Saab eristada, kas isanimi on keskmise nime või perekonnanime rollis.
- Liidetega perenimesid saab sorteerida ilma liiteta, kuid kuvada liitega.

Mustri nõrgad küljed:

- Keeruline andmete sisestamine. Läbi kasutajaliidese sisestamine vajaks palju erinevaid vormivälju. Juhul, kui sisestatakse otse andmebaasi, peab sisestaja väga hästi kursis olema kultuuride nimetraditsioonidega.
- Kuna erinevad isikunimed vajavad erinevaid nimetüüpe ehk atribuute, siis tabelisse tekib palju veerge, kus enamus välju on tühjad – tekib palju NULL-e.
- Puudub informatsioon, kuidas erinevad nimeosad täisnimes paiknevad – millises järjekorras neid valida, et kokku moodustada täisnimi.

Näide: Ruumi kokkuhoiuks on järgneval joonisel toodud andmebaasi disaini diagramm poolitatud. Diagrammi esitamise eesmärgiks on demonstreerida, kuivõrd palju ja milliseid erinevaid nimega seotud veerge (kindlasti mitte lõplik hulk) võiks tabelis *Isik* salvestada. Sellist mustrit reaalselt kasutusele võtta ei ole mõttekas.



Joonis 1. Andmebaasi disaini diagramm mustrile „Kõikvõimalike nimetüüpide veerud“.

5.1.2 Valik nimetüüpidest veergudes esinemishulga põhjal

Jõud:

- Kõikvõimalikke nimetüüpe ei soovita andmebaasis hoida, soovitakse teha mingi valik nimetüüpidest.
- Soovitakse teha valik nimetüüpide esinemissageduse põhjal.

Lahendus: Luuakse tabel, kus enim esinevad nimetüübid on tabeli veerud (Joonis 2).

Mustri tugevad küljed:

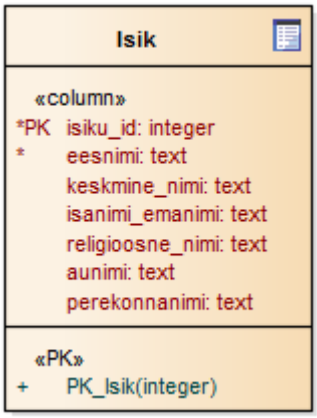
- Tabel võimaldab salvestada valikuliselt erinevaid nimetüüpe. Enim esinevad nimetüübid on esindatud.
- Nime osasid on lihtne andmebaasist küsida.

Mustri nõrgad küljed:

- Juhul, kui andmed sisestatakse otse andmebaasi, siis peab sisestaja väga hästi kursis olema erinevate kultuuride nimetraditsioonidega.
- Kuna erinevad isikunimed vajavad erinevaid nimetüüpe ehk atribuute, siis tabelisse tekib palju veerge, kus enamus välju on tühjad – tekib palju NULL-e.
- Puudub informatsioon, kuidas erinevad nimeosad täisnimes paiknevad – millises järjekorras neid valida, et täisnimi kokku saada.
- Selleks, et teha mõttekas valik, milliseid nimetüüpe tabeli veergudeks määrata, on vajalik väga hästi kursis olla erinevate nimetüüpide olemasolu ja esinemissagedusega maailmas ning huvipakkuvate isikute seas.

Variatsioonid: Teades konkreetselt, milline on piiratud hulk erinevaid nimekultuure, kuhu kuuluvate isikute andmeid andmebaasis hoitakse, võib valiku teha selle põhjal ja saab olla kindlam, et suvalise isiku nime osa jaoks eksisteerib andmebaasis vastav nimetüübi veerg.

Näide:



Joonis 2. Andmebaasi disaini diagramm mustrile „Valik nimetüüpidest veergudes esinemishulga põhjal“.

Kõiki salvestatud nimetüüpe on lihtne andmebaasist küsida, kuna kõik on ühes tabelis. Joonis 3 esitab vastava päringu ja tulemuse:

```
SELECT * FROM Isik ORDER BY perekonnanimi;
```

isiku_id	eesnimi	keskmine_nimi	isanimi_emanimi	religioosne_nimi	aunimi	perekonnanimi
6	3ric	J.	*NULL*	*NULL*	*NULL*	Bitt
5	Hafiz	*NULL*	*NULL*	Muhammad	*NULL*	Khan
3	Zedong	*NULL*	*NULL*	*NULL*	*NULL*	Mao
4	Johannes	*NULL*	*NULL*	*NULL*	*NULL*	van der Gaals
1	Björk	*NULL*	Guðmundsdóttir	*NULL*	*NULL*	*NULL*
2	Rasiah	*NULL*	Alagaratnam	*NULL*	*NULL*	*NULL*
7	Jameela	*NULL*	*NULL*	*NULL*	Khatoon	*NULL*

Joonis 3. Päring kõikide nimetüüpide leidmiseks ja kuvatõmmis päringu tulemusest.

- Tabelis on väga palju tühje välju (NULL-e).
- Probleemid on ka sortimisel – perekonnanime van der Gaals peaks sortima G-tähe järgi.
- Sortimise tulemust mõjutavad ka võrdlusreeglid (*collation*). Need määravad märkide (ja antud juhul nendest moodustatud isikunimede) võrdlemise tulemuse (kas kaks märki on võrdsed või üks on suurem kui teine) [9] ning seega mõjutavad sortimise tulemusel moodustatud nimede järjekorda. Näiteks sortides kasvavalt ja kasutades PostgreSQL-i võrdlusreeglit *et_EE.utf8*, järjestatakse Õ-tähega algavad nimed eesti keele järgi nimekirja lõpupoole, kuid tehes sama päringut, kasutades võrdlusreeglit *en_US.UTF-8*, suhtutakse inglise keele järgi Õ-tähte kui ühte varianti O-tähest ja nimi Õie sorditakse N- ja P-tähe vahele. Võttes arvesse, kuivõrd palju on maailmas eri keelte tähestikke, võib ette kujutada nüansi- ja detailirikkust, mida tuleks võrdlusreeglite valikul silmas pidada, ja seetõttu siinkohal ka pikemalt ei peatu. Tehniliselt on võrdlusreeglite määramisel PostgreSQL-is mitmeid võimalusi – saab määrata kogu andmebaasile, tabelile, veerule, päringus olevale sortimiseeskirjale. Antud näites on sortimine võrdlusreegli järgi *et_EE.utf8*, mis on määratud kogu andmebaasile.

Joonis 4 näitab, et nimeosadest täisnime moodustamine on komplitseeritud:

```
SELECT concat_ws (' ', eesnimi, keskmine_nimi, isanimi_emanimi,
religioosne_nimi, aunimi, perekonnanimi) AS taisnimi
FROM Isik;
```

taisnimi
Björk Guðmundsdóttir
Rasiah Alagaratnam
Zedong Mao
Johannes van der Gaals
Hafiz Muhammad Khan
3ric J. Bitt
Jameela Khatoon

Joonis 4. Päring täisnime saamiseks ja kuvatõmmis päringu tulemusest.

Antud päringu tulemuseks on kokkupandud nimeosad (kusjuures nimetüüpide valimisel NULL välju ignoreeritakse). Täisnime mõistes on aga mitmed read väärad: teisel real oleva isiku täisnimi on tegelikult Alagaratnam Rasiah (isanimi + eesnimi), kolmandal real Mao Zedong (perenimi + eesnimi), viiendal real Muhammad Hafiz Khan (religioosne nimi + eesnimi + perenimi).

Korrektseteks tulemusteks peab päringu tegemisel iga isiku puhul eraldi teadma, millises järjekorras nimetüüpe küsida. Juhul, kui küsitavad nimetüübid antud isikutel kindlasti andmebaasis eksisteerivad, siis järgneva päringu (Joonis 5) tulemusteks on korrektsed täisnimed – Mao Zedong ja Alagaratnam Rasiah.

```
SELECT perekonnanimi || ' ' || eesnimi AS taisnimi
FROM Isik WHERE isiku_id = 3
UNION SELECT isanimi_emanimi || ' ' || eesnimi AS taisnimi
FROM Isik WHERE isiku_id = 2;
```

Joonis 5. Päring nimetüüpidest kokkupandud täisnimede saamiseks juhul, kui on teada, millistest nimetüüpidest täisnimi koosneb.

5.1.3 Valik nimetüüpidest veergudes Google'i kontaktide põhjal

Jõud:

- Kõikvõimalikke nimetüüpe ei soovita andmebaasis hoida, soovitakse teha mingi valik nimetüüpidest.
- Soovitakse hoida andmebaasis kindlasti täisnime ja valikuliselt veel lisaks nimeosasid ja nende erinevaid transkriptsioone.

Lahendus: Luuakse tabel, mille veergudes on täisnimi ja lisaks nimede osad (Joonis 6).

Allikad ja autorid: Antud muster on tehtud selle põhjal, kuidas Google isikunimega seotud infot kasutaja Google'i konto kontaktides küsib. Kuna Google'i konto on paaril miljardil kasutajal üle maailma, on antud töö raames asjakohane näide sellest, kuidas reaalses praktikas eri rahvaste isikunimesid andmebaasides hoitakse. (Siinkohal vajab äramärkimist aga fakt, et Google'i konto loomisel on kasutaja sunnitud määrama kaks nimetüüpi – ees- ja perekonnanimi.) Näitamaks, et mõnel juhul võib vajalikuks osutuda hoida andmebaasis ka nime osade ASCII kuju, lisas autor omalt poolt vastavad veerud.

Mustri tugevad küljed:

- Võimaldab valikuliselt eri kultuuridest pärit isikute nimede osasid salvestada.
- Andmebaasist on võimalik väga lihtsalt küsida isiku täisnime.
- Nime osasid on lihtne andmebaasist küsida.
- On eraldi nimekuju, mis võimaldab korrektselt sortida.
- Kui foneetiline kuju erineb nime originaalkujust, on ka seda võimalik salvestada.

Mustri nõrgad küljed:

- Kuna ainult üks veerg on kohustuslik (*täisnimi*), siis tabelisse võib tekkida palju tühje välju ehk NULL-e.
- Veerud foneetiliseks ja ASCII nimekujuks eeldavad ees- ja perenime olemasolu.
- Raskused andmete halduses. Ühes veerus oleva nimeosa muutumisel (näiteks perekonnanime muutus) tuleks muuta ka teisi veerge (näiteks *perekonnanime_ascii* ja *taisnimi*).

Näide:

Joonis 6. Andmebaasi disaini diagramm mustrile „Valik nimetüüpidest Google'i kontaktide põhjal".

5.1.4 Üks nimetüüp, ülejäänud nimetüübid

Jõud: Vaja on defineerida kindlad väljad enimkasutatud nimetüübi jaoks ja samas talletada ka erinevat tüüpi lisanimesid. Näiteks päringute tegemiseks, sorteerimiseks, nime osa kuvamiseks või muul põhjusel on vajalik täisnimest eraldada eesnimi.

Lahendus: Luuakse üks tabel, kus nime hoidmiseks on kaks veergu – esimeses üks nimetüüp ja teises ülejäänud nimetüübid (Joonis 7).

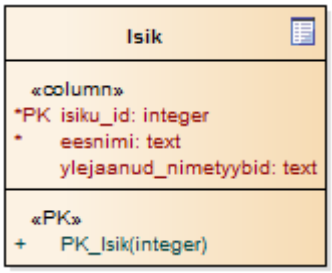
Allikad ja autorid: Muster on loodud W3C veebilehel [14] esitatud soovituste põhjal. Ka Docbooki (dokumenditüübi määrang (DTD) struktureeritud dokumentide loomiseks) mudelis on kasutusel mõiste *other name*, mis võib tähistada ükskõik missugust nime osa [6]. Täpselt missuguste nimetüüpide salvestamiseks atribuuti kasutatakse, jääb mudeli kasutaja enda otsustada.

Mustri tugevad küljed:

- Ühte vajalikku nime osa (eesnimi) on lihtne andmebaasist küsida.
- Võimaldab kõikvõimalikest erinevatest kultuuridest pärit isikute nimesid salvestada. Eesnimi on ainuke nimetüüp, mille olemasolu võiks eeldada erinevates nimekultuurides. Veergu *ylejaanud_nimetyybid* saaks salvestada isiku nime kõikvõimalikud muud osad (nende olemasolul).
- Enamasti eksisteerib lisaks eesnimele erinevates nimekultuurides veel nimeosasid, seega tabelis on vähe välju, mille väärtuseks on NULL-id.

Mustri nõrgad küljed: Puudub informatsioon, kuidas eesnimi täisnimes paikneb, et tabelis olevatest nimedest moodustada täisnimi. Juhul, kui eesnimi ei paikne nime alguses või lõpus, siis oleks täisnime ka tehniliselt raske moodustada.

Näide:



Joonis 7. Andmebaasi disaini diagramm mustrile „Üks nimetüüp, ülejäänud nimetüübid“.

5.1.5 Täisnime ja transkriptsiooni veerud

Jõud:

- Täisnimest pole vaja eraldada erinevaid osasid.
- Vaja on salvestada ka täisnime transkriptsiooni (näiteks romaniseeritud kuju, juhul kui see peaks nime originaalkujust erinema).

Lahendus: Luuakse tabel kahe veeruga – täisnimi ja selle transkriptsioon (Joonis 8).

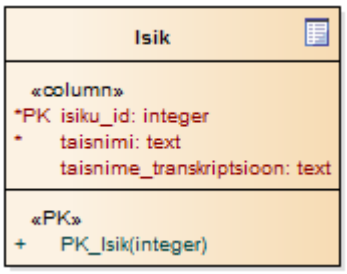
Allikad ja autorid: Muster on loodud W3C veebilehel [14] esitatud soovituste põhjal.

Mustri tugevad küljed:

- Isiku täisnime on lihtne andmebaasist küsida.
- Võimaldab salvestada eri kultuuridest pärit isikute nimesid. Isiku nime mingi osa ei jää andmebaasi sisestamata põhjusel, et vastava nimetüübi veerg puudub.
- Täisnime kasutades pole ohtu, et isiku poole pöördumisel (näiteks e-kirjas) kuvatakse nime ebakorrektselt või isegi solvavalt.

Mustri nõrgad küljed: Juhul, kui isiku täisnimi on väga pikk, jääb see e-kirjas või veebilehel kuvamisel kohmakas ja võib rikkuda lehe kujunduse. Sellele lisaks, mõned isikud ei soovi igapäevaselt kasutada enda väga pikki täisnimesid kõikide selle osadega.

Näide:



Joonis 8. Andmebaasi disaini diagramm mustrile „Täisnime ja transkriptsiooni veerud“.

5.1.6 Täisnime ja kuvanime veerud

Jõud:

- Täisnimest pole vaja eraldada erinevaid osasid.
- Isiku poole tahetakse pöörduda (nt e-kirjas) isiku poolt eelistatud nime kujuga.

- Soov oleks isiku poole pöördumisel kasutada ka tiitleid.

Lahendus: Luuakse tabel kahe veeruga täisnime ja kuvanime hoidmiseks (Joonis 9).

Allikad ja autorid: Muster on loodud W3C veebilehel [14] esitatud soovituste põhjal. Sarnaselt veerule *kuvanimi* on Arlow' ja Neustadti nimemudelis atribuut *preferredName*, mis tähistab nime, mida isik ise igapäevaselt kasutada eelistab [3].

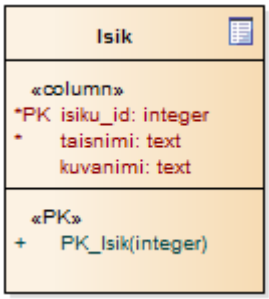
Mustri tugevad küljed:

- Isiku täisnime on lihtne andmebaasist küsida.
- Võimaldab salvestada nimesid eri kultuuridest. Pole ohtu, et nimes esinevat nimetüüpi ei saa andmebaasi sisestada põhjusel, et vastav veerg puudub.
- Võimaldab pöörduda isiku poole korrektselt ja talle meeldivalt. Isikult võiks küsida, millist nimekuju kasutada tema poole pöördumisel, ja vastus salvestada kuvanime veergu. Kuvanimes võib hoida ka tiitleid (härra, proua, dr jm).

Mustri nõrgad küljed: Antud mustril pole märkimisväärseid puudusi.

Variatsioonid: Formaalse ja mitteformaalse pöördumise eristamiseks lisada täisnimele kaks nime kuju – formaalne kuvanimi ja mitteformaalne kuvanimi.

Näide:



Joonis 9. Andmebaasi disaini diagramm mustrile „Täisnime ja kuvanime veerud".

5.1.7 Täisnime veerg

Jõud: Täisnimest pole vaja eraldada erinevaid osasid.

Lahendus: Luuakse üks tabel, kus on üks veerg – täisnime hoidmiseks (Joonis 10).

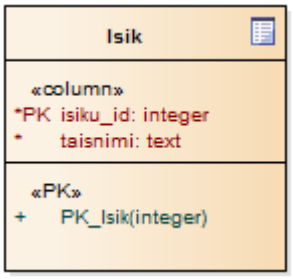
Allikad ja autorid: Muster on loodud W3C veebilehel [14] esitatud soovituste põhjal.

Mustri tugevad küljed:

- Isiku täisnime on lihtne andmebaasist küsida.
- Võimaldab salvestada nimesid eri kultuuridest. Pole ohtu, et nimes esinevat nimetüüpi ei saa andmebaasi sisestada põhjusel, et vastav veerg puudub.
- Juhud, kus isikul puuduvad igasugused nimelised identifikaatorid, on pigem väga harva esinevad. Üldjuhul eksisteerib isikul vähemalt eesnimi. Seega ei teki andmebaasi liigseid NULL-e, kuna täisnime veerud on täidetud.

Mustri nõrgad küljed: Juhul, kui isiku täisnimi on väga pikk, jääb see e-kirjas või veebilehel kuvamisel kohmakas ja võib rikkuda lehe kujunduse. Lisaks sellele, mõned isikud ei soovi igapäevaselt kasutada enda väga pikki täisnimesid kõikide selle osadega.

Näide:



Joonis 10. Andmebaasi disaini diraamm mustrile „Täisnime veerg“.

Antud mustri puhul on toodud jõuks „Täisnimest pole vaja eraldada erinevaid osasid“. Siinkohal siiski uuritakse põgusalt võimalusi nimetüüpide eraldamiseks (Joonis 11).

```
SELECT regexp_split_to_array(taisnimi, E'\\s+') AS nime_osad
FROM Isik;
```

nime_osad
{Conjeepuram,Mahendrum,Krishnamurthy}
{Suzuki,Ichiro}
{Adela,Landová-Štychková}
{Muhammad,ibn,Salman,ibn,Ameen,Ahl-Farsi}
{Juan,Pablo,Fernández,de,Calderón,García-Iglesias}
{Ravinder,Singh,Sahota}
{Tom,O'Neil}

Joonis 11. Päring täisnimest osade eraldamiseks tühiku järgi ja kuvatõmmis päringu tulemusest.

Antud päring tükeldas täisnimed tühikute järgi osadeks. Tulemuseks on massiivid, millest saab küsida erinevaid osasid (Joonis 12).

```
SELECT nimeosad[1] FROM (SELECT regexp_split_to_array(taisnimi,
E'\\s+') AS nimeosad FROM veerg_taisnimi.Isik) AS esimene_osa;
```

nime_osad
Conjeepuram
Suzuki
Adela
Muhammad
Juan
Ravinder
Tom

Joonis 12. Päring nime osade massiivist esimese elemendi saamiseks ja kuvatõmmis päringu tulemusest.

Tulemuses puudub aga info, milliseid nimetüüpe leitud nimeosad esindavad. Paljudel juhtudel koosnevad eesnimed (või ka näiteks perekonnanimed) mitmest tühikuga eraldatud nimest ning sellisel viisil leitakse nendest ainult üks. Tühiku järgi täisnimede osadeks tükeldamise tulemusel saadud massiivide puhul (Joonis 11) ei ole võimalik kindlalt eristada, kus üks nimetüüp lõpeb ja järgmine algab.

5.2 Nimetüüpide tabelid

Jõud:

- Andmebaasis on vaja hoida isikunimede ajalugu.
- Andmebaasis on vaja hoida nimetüüpe, mis erinevates nimekultuurides varieeruvad.
- Soovitakse vältida liigset NULL-ide kasutust andmebaasis, sest see võib viia loogiliselt ebakorrektsete päringutulemusteni.

Lahendus: Luuakse mitu tabelit, mis välisvõtme abil viitavad tabelile *Isik*. Iga nimetüübi jaoks luuakse üks tabel. Nimetüüpide tabelid on kuuendal normaalkujul (Joonis 13).

Allikad ja autorid: Selleks, et ajalooliste andmetega toime tulla, on välja pakutud ankurmodelleerimine [8], mille kasutamise tulemusena tekivad SQL-andmebaasis enamasti kuuendal normaalkujul olevad tabelid. Ankurmodelleerimise sisul ja tehnoloogial antud töö ei peatu, küll aga uurib kuuenda normaalkuju tabeleid lahendusena olukorrale, kui andmebaasis on vajalik hoida eri rahvaste isikunimede ajalugu.

Mustri tugevad küljed:

- Võimaldab eri kultuuridest pärit isikute nimesid salvestada.
- Puudub oht, et isiku nime mingit osa ei saa andmebaasi sisestada – vajaliku nimetüübi puudumisel saab selle lisada, luues vastava nimetüübi tabeli. See muudatus ei ole invasiivne, st ei ole vaja muuta olemasolevate tabelite struktuuri.
- Andmebaasi ei teki liigseid NULL-e.
- Võimaldab ajaloo säilitamist – igale nimetüübile saab määrata kehtimise algus- ja lõpuaja.

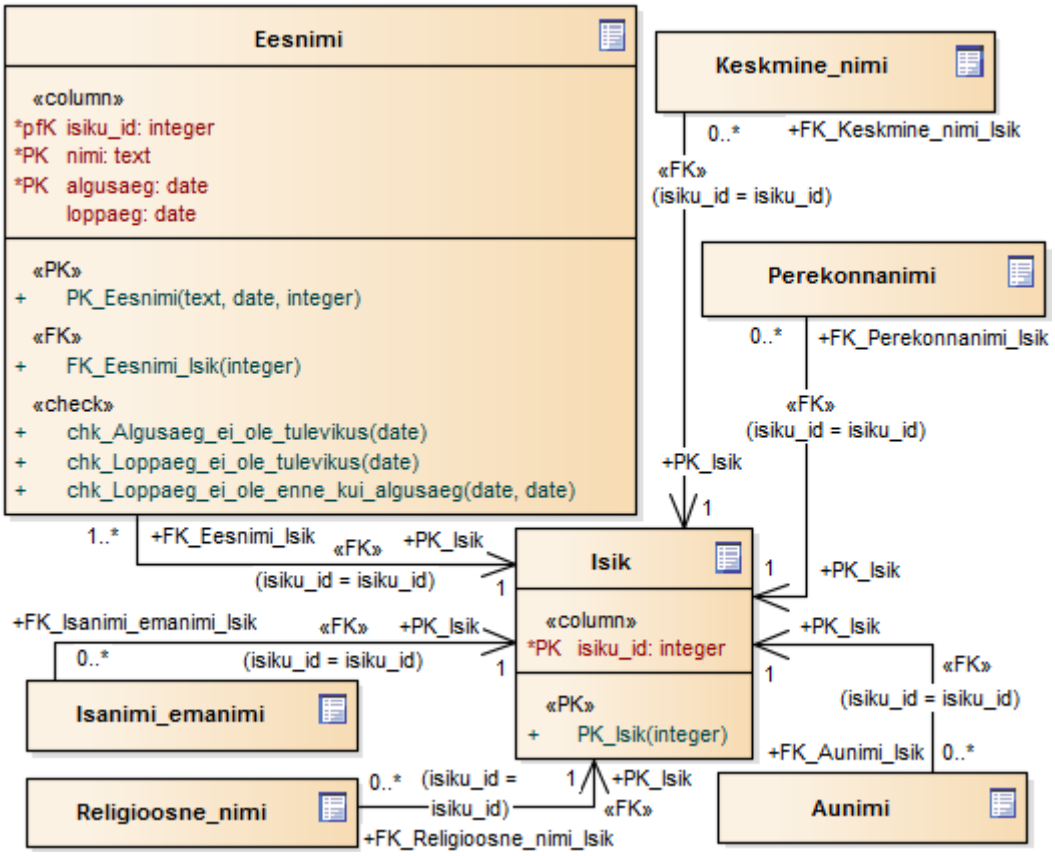
Mustri nõrgad küljed:

- Lahenduse realiseerimine andmebaasi loomisel võtab kaua aega – vaja on luua palju tabeleid.
- Keeruline andmete sisestamine. Juhul, kui andmeid sisestatakse läbi kasutajaliidese, vajaks see palju erinevaid vormivälju. Juhul, kui sisestatakse otse andmebaasi, peab sisestaja väga hästi kursis olema eri kultuuride nimetraditsioonidega.
- Puudub informatsioon, kuidas erinevad nimetüübid täisnimes paiknevad – millises järjekorras neid valida, et moodustada kokku täisnimi.
- Suure hulga nimetüüpide puhul tekib andmebaasi suurel hulgal nimetüüpide tabeleid.
- Andmebaasi tasemel on keeruline jõustada reeglit, et vähemalt mõni nimetüüp (nt eesnimi) peab olema määratud. See nõuab ilmselt trigerite loomist, sest olemasolevate SQL-andmebaasisüsteemide võimalused selliste mitut tabelit hõlmavate kitsenduste deklareerimiseks on praktikas piiratud.

Variatsioonid: Selleks, et isiku täisnime lihtsamalt andmebaasist küsida, võib lisada tabelisse *Isik* veeru *täisnimi*. Näiteks FHIR mudelis on lisaks nimetüüpide veergudele ka veerg *text*, mis on mõeldud täisnime hoidmiseks [11]. Sellega kaasnevad aga komplikatsioonid nagu andmete liiasus (isiku nimi on andmebaasi registreeritud kaks korda) ja raskused andmete halduses (kui ühes tabelis nime osa muuta, peab seda tegema ka teises tabelis).

Kuuenda normaalkuju lahendust saaks kombineerida mustris „Universaalne disain kultuuriga“ pakutud võimalusega määrata iga isiku kohta kultuur, mis omakorda määrab tema jaoks vajalikud nimetüüpide tabelid.

Näide: Andmebaasi disaini diagrammil (Joonis 13) on kõikidel nimetüübi tabelitel ühesugused veerud ja kitsendused, mida parema loetavuse huvides on joonisel näidatud vaid nimetüübi tabelil *Eesnimi* ja ülejäänutel on need ära peidetud. Ainuke mittekohustuslik veerg on *loppaeg*, mille puhul peaks olema täidetud nõue, et igal isikul on iga nimetüübi kohta maksimaalselt üks rida, kus nime kehtivuse lõppaeg on NULL. See tähendab, et antud hetkel saab isikul olla kõige rohkem üks[1] kehtiv eesnimi ja/või keskmine nimi jne.



Joonis 13. Andmebaasi disaini diagramm mustrile „Nimetüüpide tabelid“.

Joonis 14 esitab päringu, mis leiab andmebaasist kõik nimetüübid. Joonisel on kujutatud ka päringu tulemust:

[1] Nagu eespool mainitud, siis antud töö raames loetakse nimes korduvad nimetüübid üheks nimetüübiks – nimes Eva Liisa Tamm on eesnimeks Eva Liisa ja perekonna nimeks Tamm (mitte esimeseks eesnimeks Eva ja teiseks eesnimeks Liisa).

```
SELECT Isik.isiku_id, Eesnimi.nimi AS eesnimi, Keskmine_nimi.nimi AS
keskmine_nimi, Isanimi_emanimi.nimi AS isanimi_emanimi,
Religioosne_nimi.nimi AS religioosne_nimi, Aunimi.nimi AS aunimi,
Perekonnanimi.nimi AS perekonnanimi
FROM Isik
LEFT JOIN Eesnimi ON Isik.isiku_id=Eesnimi.isiku_id
LEFT JOIN Keskmine_nimi ON Isik.isiku_id=Keskmine_nimi.isiku_id
LEFT JOIN Isanimi_emanimi ON Isik.isiku_id=Isanimi_emanimi.isiku_id
LEFT JOIN Religioosne_nimi ON Isik.isiku_id=Religioosne_nimi.isiku_id
LEFT JOIN Aunimi ON Isik.isiku_id=Aunimi.isiku_id
LEFT JOIN Perekonnanimi ON Isik.isiku_id=Perekonnanimi.isiku_id
ORDER BY Isik.isiku_id;
```

isiku_id	eesnimi	keskmine_nimi	isanimi_emanimi	religioosne_nimi	aunimi	perekonnanimi
1	Mari	Veroonika	*NULL*	*NULL*	*NULL*	Kuusepuu
2	Georg	M.	*NULL*	*NULL*	*NULL*	Smith
3	Björk	*NULL*	Guðmundsdóttir	*NULL*	*NULL*	*NULL*
4	Hafiz	*NULL*	*NULL*	Muhammad	*NULL*	Khan
5	Konstantin	*NULL*	Nikolajevitš	*NULL*	*NULL*	Uljanov
5	Ilja	*NULL*	Nikolajevitš	*NULL*	*NULL*	Uljanov
6	Jameela	*NULL*	*NULL*	*NULL*	Khatoon	*NULL*

Joonis 14. Päring kõikide nimetüüpide leidmiseks ja kuvatõmmis päringu tulemusest.

Isiku puhul, kelle *isiku_id* on 5, kuvatakse kaks rida, kuna isik on vahetanud enda eesnime ja seega on andmebaasis koos hetkel kehtiva ja eelneva täisnimega kokku kaks täisnime.

Joonis 15 kujutab päringut isikute täisnimede leidmiseks (ainult antud hetkel kehtivate nimede leidmiseks on lisatud piirang, et nime lõppaeg peab olema NULL):

```
SELECT concat_ws (' ', eesnimi, keskmine_nimi, isanimi_emanimi,
religioosne_nimi, aunimi, perekonnanimi) AS taisnimi
FROM (SELECT Isik.isiku_id, Eesnimi.nimi AS eesnimi,
Keskmine_nimi.nimi AS keskmine_nimi, Isanimi_emanimi.nimi AS
isanimi_emanimi, Religioosne_nimi.nimi AS religioosne_nimi,
Aunimi.nimi AS aunimi, Perekonnanimi.nimi AS perekonnanimi
FROM Isik
LEFT JOIN Eesnimi ON Isik.isiku_id=Eesnimi.isiku_id
LEFT JOIN Keskmine_nimi ON Isik.isiku_id=Keskmine_nimi.isiku_id
LEFT JOIN Isanimi_emanimi ON Isik.isiku_id=Isanimi_emanimi.isiku_id
LEFT JOIN Religioosne_nimi ON Isik.isiku_id=Religioosne_nimi.isiku_id
LEFT JOIN Aunimi ON Isik.isiku_id=Aunimi.isiku_id
LEFT JOIN Perekonnanimi ON Isik.isiku_id=Perekonnanimi.isiku_id
WHERE (Eesnimi.loppaeg IS NULL AND Keskmine_nimi.loppaeg IS NULL AND
isanimi_emanimi.loppaeg IS NULL AND Religioosne_nimi.loppaeg IS NULL
AND Aunimi.loppaeg IS NULL AND Perekonnanimi.loppaeg IS NULL)
ORDER BY Isik.isiku_id) AS Tulemus;
```

Joonis 15. Päring hetkel kehtivate täisnimede saamiseks, kus täisnime moodustamine sõltub nimetüüpide küsimise järjekorrast.

Joonisel 16 näidatakse eelneva päringu tulemust:

taisnimi
Mari Veroonika Kuusepuu
Georg M. Smith
Björk Guðmundsdóttir
Hafiz Muhammad Khan
Konstantin Nikolajevitš Uljanov
Jameela Khatoon

Joonis 16. Kuvatõmmis Joonisel 15 esitatud päringu tulemusest.

Päringu tulemus sõltub sellest, millises järjekorras välises SELECT-lauses veergusid küsitakse. Seetõttu võib tulemuseks saadud täisnimedes esineda ebakorrektsusi. Näiteks neljandal real on isiku täisnimeks Hafiz Muhammad Khan, aga korrektne täisnimi on Muhammad Hafiz Khan. Juhul, kui andmebaasis oleks hiina nimesid, annaks antud päring (eesnime küsitakse enne kui perekonnanime) täisnimed, kus ees- ja perekonnanimi oleksid vales järjekorras.

5.3 Universaallahendus

Antud muster on üldisem nimetus järgnevatele alam-mustritele. Kõik, mis siin välja tuuakse, kehtib ka alam-mustrite puhul.

Jõud:

- Andmebaasis on vaja hoida erinevaid nime osasid, mis erinevates nimekultuurides varieeruvad.
- Andmebaasis on vaja hoida isikunimede ajalugu.
- Soovitakse vältida liigset NULL-ide kasutust andmebaasis.

Lahendus: Luuakse mitu tabelit. Tabelis *Isik* on isikute üldandmed, sh isiku unikaalne identifikaator. Erinevates tabelites hoitakse: nimede osasid; määratlusi, mis nimetüübid antud osad on; kultuuri, kuhu isik kuulub; infot, millises kultuuris milliseid nimetüüpe kasutatakse. Isikutele, kelle kohta kultuur pole teada, saab välja töötada vaikimisi „kultuuri“.

Allikad ja autorid: Universaalse disaini lahendus on Karwini [15] poolt iseenesest nimetatud antimustriks ja seda soovitab vältida ka Larson [17]. Siinkohal vajab täpsustust, et antimustri moodustaks universaalne disain sellisel juhul, kui kogu andmebaas oleks ehitatud sellest disainist lähtuvalt (st kõik andmed on tabelites kui *Objekti_tüüp, Objekt, Atribuut, Atribuudi_väärtus*). Antud töö raames kirjeldab muster „Universaallahendus“ ainult sellist osa kogu andmebaasist, mis on seotud isikunimede andmete salvestamisega ja seetõttu antimustriks ei loeta.

Mustri tugevad küljed:

- Võimaldab eri kultuuridest pärit isikute nimesid salvestada. Puudub oht, et isiku nime mingit osa ei saa andmebaasi sisestada – vajaliku nimetüübi puudumisel saab selle hõlpsalt juurde tekitada. Tabelis *Nimetyyp* võib nimetüübiks määrata ka nime osade transkriptsioone.
- Universaalmustrit kasutades on võimalik minna detailidesse – saab määrata järjekorranumbreid, transkriptsioone, nime osade erinevaid kujusid (näiteks liitega perekonnanime puhul saaks lisada liiteta perekonnanime, et selle järgi sortida).
- Andmebaasi ei teki liigseid NULL-e.
- Võimaldab ajaloo salvestamist – igale nimetüübile saab määrata kehtimise algus- ja lõpuaja.

Mustri nõrgad küljed:

- Keeruline andmete sisestamine. Läbi kasutajaliidese sisestamine vajaks palju erinevaid vormivälju. Juhul, kui sisestatakse otse andmebaasi, peab sisestaja väga hästi kursis olema kultuuride nimetraditsioonidega.
- Andmebaasi tasemel on keeruline jõustada reeglit, et vähemalt mõni nime osa (nt eesnimi) peab olema määratud. See nõuab ilmselt trigerite loomist, sest olemasolevate SQL-andmebaasisüsteemide võimalused selliste mitut tabelit hõlmavate kitsenduste deklareerimiseks on praktikas piiratud.

5.3.1 Universaalne disain kultuuriga

Jõud: Soovitakse säilitada infot, millisesse nimekultuuri isik kuulub, et määrata, millistest osadest tema täisnimi koosneb.

Lahendus: Luuakse viis tabelit. Tabelis *Isik* on isikute üldandmed, sh isiku unikaalne identifikaator. Ülejäänud erinevates tabelites hoitakse: erinevaid nime osasid; määratlusi, mis nimetüübid antud osad on; kultuuri, kuhu isik kuulub; infot, millises kultuuris milliseid nimetüüpe kasutatakse (Joonis 17).

Allikad ja autorid: Universaalse mustri pakkus välja Silverston [27]. Autor lisas tabelid *Kultuur*, *Nimetyyp* ja *Nimetyyp_kultuuris*, et säilitada infot, millises kultuuris millistest nimetüüpidest isikunimede struktuur koosneb.

Mustri tugevad küljed: Järk-järgult laiendatav – lisades ridu tabelitesse *Kultuur*, *Nimetyyp* ja *Nimetyyp_kultuuris,* saab moodustada palju võimalikke kombinatsioone.

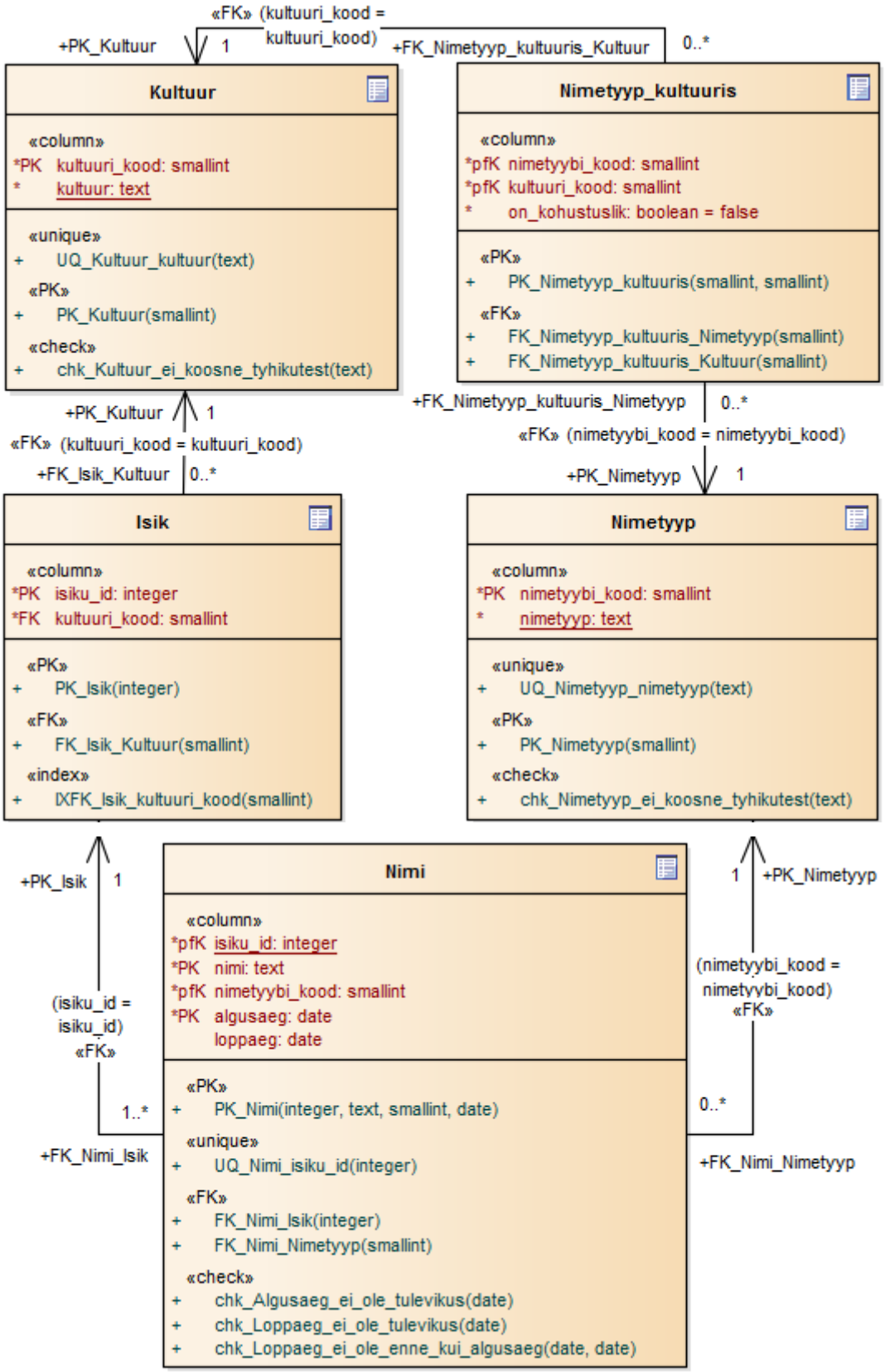
Mustri nõrgad küljed:

- Kõikvõimalike maailmas esinevate nimetüüpide kaardistamine on äärmiselt mahukas[1] töö.
- Kultuuri raames võib nimestruktuuris esineda varieeruvusi. Isegi kui on teada, millisesse kultuuri isik kuulub, pole võimalik lõpuni kindel olla, millistest nimetüüpidest (ja millises järjekorras) tema täisnimi koosneb.
- Lahenduse realiseerimine andmebaasi loomisel võtab kaua aega – vaja on luua palju tabeleid.

Variatsioonid: Tabelisse *Kultuur* või sellega seotud täiendavatesse tabelitesse saaks lisada nimemustrid nagu seda kirjeldab Mõtshärg [20]. Tabelisse *Nimetyyp_kultuuris* saaks lisada järjekorranumbri, mille põhjal oleks võimalik registreerida nimetüüpide järjekorda.

Näide: Ainuke mittekohustuslik veerg on *loppaeg* tabelis *Nimi*, mille puhul peaks olema täidetud nõue, et igal isikul on iga nimetüübi kohta maksimaalselt üks rida, kus nime kehtivuse lõppaeg on NULL.

[1] Näiteks IBM-i isikunimede tuvastamise tehnoloogia on programm eri rahvaste isikunimede sõelumiseks, mis põhineb nimekultuuride analüüsil. Seda on arendatud 15 aastat ja kokku on kogutud igast riigist maailmas 750 miljonit nime, mille põhjal lingvistilist ja statistilist analüüsi tehakse [12].



Joonis 17. Andmebaasi disaini diagramm mustrile „Universaalne disain kultuuriga“.

5.3.2 Universaalne disain järjekorranumbriga

Jõud: Andmebaasis on vaja hoida infot, mille järgi oleks võimalik erinevad nimeosad kokku panna, et moodustada isiku korrektne täisnimi.

Lahendus: Luuakse kolm tabelit. Tabelis *Isik* on isikute üldandmed, sh isiku unikaalne identifikaator. Ülejäänud tabelites hoitakse erinevaid nime osasid, määratlusi, missugused nimetüübid antud osad on. Lisaks hoitakse järjekorranumbreid, mis näitavad, millises järjekorras nimetüübid täisnimes paiknevad (Joonis 18).

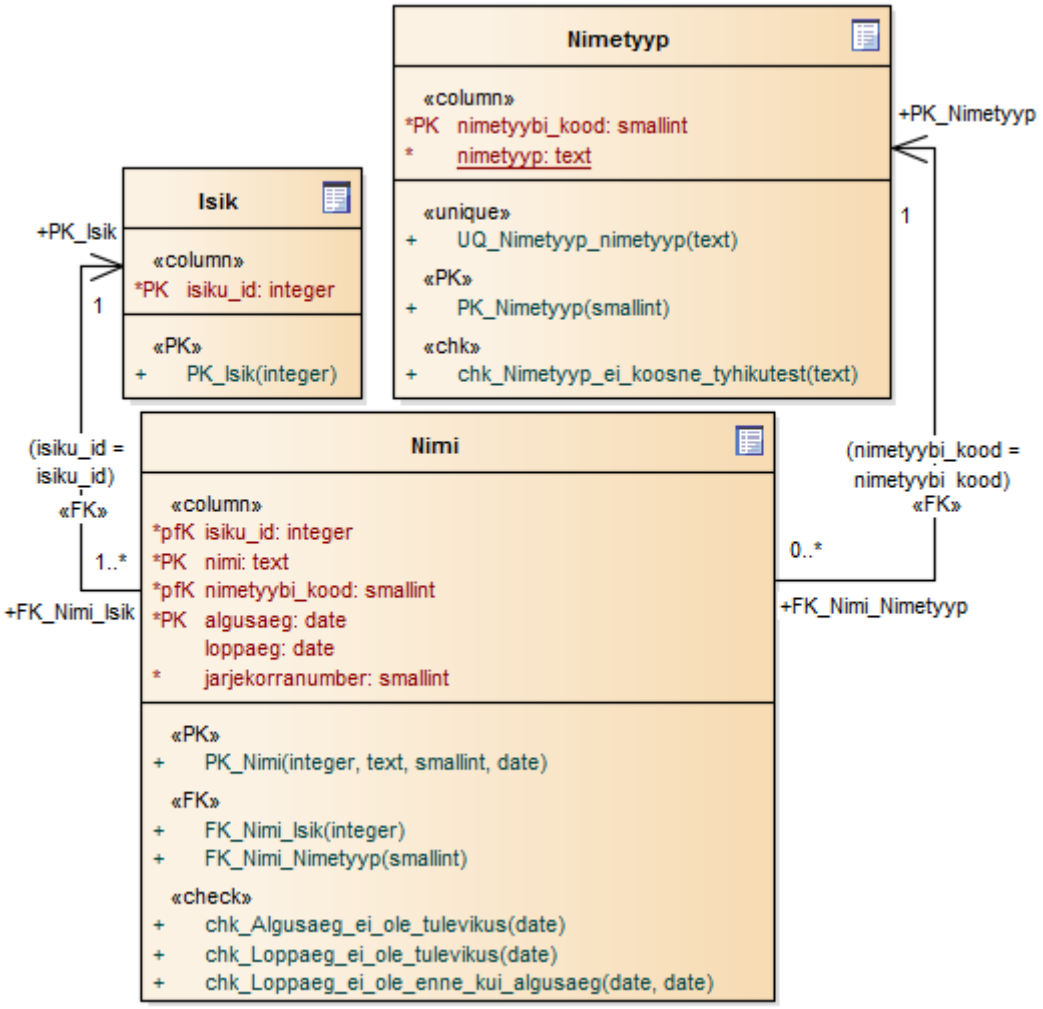
Allikad ja autorid: Universaalse mustri pakkus välja Silverston [27].

Mustri tugevad küljed:

- Järk-järgult laiendatav – kui vajalik nimetüüp puudub, saab seda lihtsasti tekitada ühe rea lisamisega tabelisse *Nimetyyp*.
- Kuna iga nimetüüp salvestatakse koos järjekorranumbriga, mis näitab nimetüübi positsiooni täisnimes, siis on võimalik kokku moodustada korrektne täisnimi.

Mustri nõrgad küljed: Antud mustri ülem-mustris „Universaallahendus“ välja toodud nõrkadele külgedele pole märkimisväärseid puudusi lisada.

Näide: Ainuke mittekohustuslik veerg on *loppaeg* tabelis *Nimi*, mille puhul peaks olema täidetud nõue, et igal isikul on iga nimetüübi kohta maksimaalselt üks rida, kus nime kehtivuse lõppaeg on NULL.



Joonis 18. Andmebaasi disaini diagramm mustrile „Universaalne disain järjekorranumbriga“.

Suhteliselt lihtsa päringuga (Joonis 19) on võimalik küsida andmebaasist kõiki nimetüüpe (Joonis 20). Päringus kasutatakse vasakpoolset välisühendamist (*LEFT JOIN*), kuna eeldatakse, et mõnel inimesel ei pruugi nime olla või nimi võib olla registreerimata:

```
SELECT Isik.isiku_id, Nimi.nimi, Nimetyyp.nimetyyp, Nimi.algusaeg,
Nimi.loppaeg, Nimi.jarjekorranumber AS jrk_nr
FROM Isik
LEFT JOIN Nimi ON Isik.isiku_id=Nimi.isiku_id
LEFT JOIN Nimetyyp ON Nimi.nimetyybi_kood = Nimetyyp.nimetyybi_kood
ORDER BY Isik.isiku_id, Nimi.jarjekorranumber;
```

Joonis 19. Päring kõikide isikute, nendega seotud nimede ja nimede kohta käiva informatsiooni leidmiseks.

isiku_id	nimi	nimetyyp	algusaeg	loppaeg	jrk_nr
1	Иля	eesnimi	1980-01-01	*NULL*	1
1	Николайевитш	isanimi_keskmise_nimena	1980-01-01	*NULL*	2
1	Улянов	perekonnanimi	1980-01-01	*NULL*	3
2	Mao	perekonnanimi_romaniseeritud	1904-01-01	*NULL*	1
2	Ze	generatsiooni_nimi_romaniseeritud	1904-01-01	*NULL*	2
2	Dong	eesnimi_romaniseeritud	1904-01-01	*NULL*	3
3	John Nick	eesnimi	1990-01-01	*NULL*	1
3	John	eesnimi	1955-01-01	1990-01-01	1
3	Patrick	keskmine_nimi	1955-01-01	*NULL*	2
3	Bitt	perekonnanimi	1955-01-01	*NULL*	3
3	Jr.	generatsiooni_nimi	1955-01-01	*NULL*	4
4	Ekaterini	eesnimi	1970-01-01	*NULL*	1
4	Spyrou	abikaasa_eesnimi	2000-01-01	*NULL*	2
4	Nikolaou	isanimi_keskmise_nimena	1970-01-01	2000-01-01	2
4	Kyprianou	perekonnanimi	1970-01-01	*NULL*	3
5	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Joonis 20. Kuvatõmmis Joonisel 19 esitatud päringu tulemusest.

- Nimede osad on sorditud nende korrektse järjekorra järgi täisnimes.
- Andmebaasis on vähe välju, mille väärtusteks NULL – ainult veeru *loppaeg* ridades, kus nime osa käesoleval hetkel kehtib. (Viimane rida näitab, et andmebaasis on isik, kellel nimi puudub).

Andmebaasist kõikide nime omavate isikute antud hetkel kehtivate täisnimede küsimiseks on võimalik kasutada funktsiooni *string_agg*, mis annab tulemuse, kus nimetüübid asetsevad täisnimedes korrektses järjekorras. Joonis 21 esitab sellekohase näite:

```
SELECT isiku_id, string_agg (nimi, ' ' ORDER BY jarjekorranumber) AS
taisnimi
FROM universaalne.Nimi WHERE loppaeg IS NULL GROUP BY isiku_id;
```

isiku_id	taisnimi
1	Иля Николайевитш Улянов
2	Mao Ze Dong
3	John Nick Patrick Bitt Jr.
4	Ekaterini Spyrou Kyprianou

Joonis 21. Päring hetkel kehtivate korrektsete täisnimede saamiseks ja kuvatõmmis päringu tulemusest.

6 Autori soovitused mustrite kasutamiseks

Uurinud isikunimede variatsioone erinevates kultuurides ja mõistnud nende mitmekesisuse hulka, on autori soovitus – kui võimalik, siis vältida nimetüüpide eristamist. Sel viisil saab kindel olla, et isikul on võimalik (näiteks veebilehel oleva ankeedivormi kaudu) enda nime sisestada korrektselt ja nii, et ei pea seda kuidagi moonutama ankeedis olevate väljade järgi. Nimetüüpe eristama hakates tekib kohe väga oluline küsimus – kuivõrd detailseks minna. Kõikvõimalikke maailmas eksisteerivaid nimetüüpe eristada ei tundu mõttekas, samas on raske otsustada, millised nimetüübid valida ja milliseid kõrvale jätta.

Siinkohal mõned näited mustrite kasutamiseks.

- Juhul, kui on vajalik salvestada isikunimede ajalugu, võiks lähtuda mustritest „Nimetüüpide tabelid“ ja „Universaallahendus“.
- Juhul, kui puudub vajadus isikunimede ajalugu säilitada, kuid on vaja eristada nimede osasid, võiks lähtuda mustritest „Valik nimetüüpidest veergudes esinemishulga põhjal“ ja „Valik nimetüüpidest veergudes Google’i kontaktide põhjal“. Töö autor ei pea neid kõige parematekslahendusteks, kuid neid on võimalik modifitseerida konkreetse infosüsteemi vajaduste järgi.
- Juhul, kui puudub vajadus isikunimede ajalugu säilitada ja nimede osasid eristada, võiks lähtuda mustrist „Täisnime veerg“. Sobilikud on ka „Täisnime ja transkriptsiooni veerud“ ning „Täisnime ja kuvanime veerud“. Viimane on autori arvates neist kolmest parim lahendus, kuna selle puhul saab isik ise määrata, millist nimekuju kasutada tema poole pöördumisel ja on tõenäoline, et pöördumine saab olema korrektne ja kasutajat mitte solvav. Lisaks lahendab see ka tiitlite keerukuse probleemi, kuna kuvanimes võib hoida ka tiitleid.
- Juhul, kui on vajalik nimetüüpe eristada, aga ühtlasi kasutada ka isiku täisnime, võiks kasutada midagi, mille järgi saaks nimetüüpidest kokku moodustada korrektse täisnime. Näiteks mustri „Universaalne disain järjekorranumbriga“ puhul on selleks järjekorranumber.
- Juhul, kui tahetakse eraldada vaid ühte nimetüüpi täisnimest, võiks kasutada mustrit „Üks nimetüüp, ülejäänud nimetüübid“. Rahvusvahelises mõõtmes soovitaksin väljavalitud nimetüübiks eesnime, kuna see on ainuke nimetüüp,

mille olemasolu võiks eeldada. Lisaks võiks andmete salvestamisel märkida kuidagi ka eesnime asetsemine täisnimes.

- Juhul, kui pole piiratud, millistesse nimekultuuridesse isikud, kelle andmeid andmebaasi salvestatakse, kuuluvad, võiks loobuda kitsendustest nimele.
- Juhul, kui on soov kasutada andmebaasitabelitel ingliskeelseid veerunimetusi, peaks *eesnime* ja *perekonnanime* puhul vältima paljukasutatud *first name* ja *last name.* See tekitab segadust, kui isikunime struktuuris on esimesel kohal perekonnanimi ja viimasel eesnimi. Autor soovitab kasutada eesnimeks *given name* ja perekonnanimeks *surname*. Ka eestikeelne *eesnimi* võib selles mõttes eksitav olla, kuid meie kultuuriruumis tajutakse eesnime tähendust siiski pigem lapsele sündides antava mittepärandatava individuaalnimena kui nime struktuuri *esimese* osana. Ülejäänud nimetüüpide ingliskeelsete vastete väljatoomise vajadus siinkohal puudub.

Antud töös parimat mustrit välja valida ei saa, suuresti sõltub kasutatava mustri valik sellest, miks konkreetses andmebaasis isikunimesid hoitakse ja milleks neid kasutatakse. Autor toob siinkohal välja hoopiski eri rahvaste nimede hoidmiseks ühe halvimatest mustritest, mida „Mustrite kataloogis“ küll ei esitatud, aga mida reaalsuses palju kasutatakse – selleks on kaheatribuudiline „Eesnimi ja perekonnanimi“, kus lisaks on mõlemad väljad kohustuslikud. Andmebaasi disainimisfaasis pikemalt kaalutlemata lisada veerud *eesnimi* ja *perekonnanimi* põhjusel, et tavaliselt nii tehakse, pole hea lahendus. Tuleks põhjalikumalt läbi mõelda, kas üldse on vajalik nimetüüpe eristada ja kui, siis milliseid.

7 Kokkuvõte

Töö eesmärgiks oli kirjeldada mõningaid disainimustreid isikunimede hoidmiseks SQL-andmebaasis, realiseerida mõne mustri põhjal näiteandmetega andmebaas, mille põhjal teha päringuid, saadud kogemuste põhjal disaine võrrelda ja analüüsida lahenduste eeliseid ning puuduseid.

Töö tulemuseks on valminud kümme disainimustrit, millest nelja põhjal on realiseeritud näited. Erinevate näidete puhul on tehtud erinevaid päringuid ja kõigi mustrite puhul on esitatud nende tugevad ja nõrgad küljed.

Seega võib öelda, et töö eesmärk saavutati. Teemasse süvenedes ilmnesid nüansid ja detailid, mida alguses ei osanud eeldada. Ühelt poolt isikunimede varieeruvuse maht, teiselt poolt arusaam, et polegi lihtne leida olemasolevatest materjalidest selgeid ja sobivaid lahendusi. Ilmnes, et paljukasutatud muster „Eesnimi ja perekonnanimi“ pole hoopiski hea lahendus eri rahvaste isikunimede andmebaasi salvestamiseks.

Antud töös esitatud disainimustrite puhul on mitmeid võimalusi neid varieerida ja ka kombineerida. Mustrite tugevate ja nõrkade külgede uurimisel jäeti välja päringute jõudluste analüüs, mis oleks kindlasti üks töö edasiarendamise võimalustest. Andmebaaside realiseerimise tehnilistest küsimustest toodi esile vaid mõned aspektid – ka siin on ruumi põhjalikumaks käsitluseks. Loodud mustrid on tehtud lihtsustatud nime struktuuri arvestades, kust on välja jäetud tiitlid ning nimes esinevaid mitut ühesugust nimetüüpi vaadatakse kui ühte nimetüüpi. Seega on võimalus uurida disaine, mis rahuldaksid tiitlite kasutamise ja näiteks kahe eesnime puhul nende eraldi väljadesse salvestamise vajadusi.

Kasutatud kirjandus

[1] A Guide To Names And Naming Practices. [WWW] https://www.fbiic.gov/public/2008/nov/Naming_practice_guide_UK_2006.pdf (20.03.2016)

[2] Anchor [WWW] http://www.anchormodeling.com/ (22.05.2016)

[3] Arlow, J., Neustadt, I. Enterprise patterns and MDA: Building Better Software with Archetype Patterns and UML. 1st ed. Boston : Addison-Wesley, 2004

[4] Database Answers [WWW] http://www.databaseanswers.org/data_models/index.htm (10.05.2016)

[5] Depesz blog. CHAR(x) vs. VARCHAR(x) vs. VARCHAR vs. TEXT [WWW] https://www.depesz.com/2010/03/02/charx-vs-varcharx-vs-varchar-vs-text/ (18.05.2016)

[6] DocBook: The Definitive Guide [WWW] http://www.docbook.org/tdg/en/html/personname.html (15.05.2016)

[7] Dubinsky, Z. CBC News. [WWW] http://www.cbc.ca/news/technology/new-credit-cards-pose-security-problem-1.904220 (22.04.2016)

[8] Eessaar, E. Ankurmodelleerimine. Andmebaasid II õppematerjalid. TTÜ, 2015. [ONLINE]

[9] Eessaar, E. Baastabelid, süsteemikataloog. Andmebaasid II õppematerjalid. TTÜ, 2016. [ONLINE]

[10] Eessaar, E. Mustrid ja nende kasutamist abistavad tarkvarasüsteemid. [WWW] http://deepthought.ttu.ee/aa/modules.php?name=News&file=article&sid=353 (03.05.2016)

[11] FHIR [WWW] http://www.hl7.org/FHIR/datatypes.html#humanname (03.05.2016)

[12] Hermansen, J. Advanced Global Name Recognition Technology. IBM Corporation, 2006 [ONLINE]

[13] IMDb [WWW] http://www.imdb.com/name/nm1804460/ (02.05.2015)

[14] Ishida, R. Personal names around the world. [WWW] http://www.w3.org/International/questions/qa-personal-names (16.05.2015)

[15] Karwin, B. SQL Antipatterns: Avoiding the Pitfalls of Database Programming. Pragmatic Bookshelf, 2010

[16] Katz, J. Range Types: Your Life Will Never Be The Same [ONLINE] https://wiki.postgresql.org/images/7/73/Range-types-pgopen-2012.pdf

[17] Larson, A. Five Simple Database Design Errors You Should Avoid [WWW] https://www.simple-talk.com/sql/database-administration/five-simple--database-design-errors-you-should-avoid/ (02.04.2016)

[18] MariaDB. What Is Table Elimination? [WWW] https://mariadb.com/kb/en/mariadb/what-is-table-elimination/ (21.05.2016)

[19] McKenzie, P. Falsehoods Programmers Believe About Names. [WWW] https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/ (16.05.2016)

[20] Mõtshärg. L. Isikunimede analüüs, parsimine ja vormindamine Eesti, Saksamaa ja Suurbritannia nimede näitel : bakalaureusetöö. TTÜ Informaatikainstituut, 2015. [ONLINE]

[21] Peffers, K. The Design Science Research Process: A Model For Producing And Presenting Information Systems Research. Claremont, CA : CGU, 2006. [ONLINE]

[22] Picabo Street. [WWW] https://en.wikipedia.org/wiki/Picabo_Street (02.05.2016)

[23] PostgreSQL. 8.3. Character Types [WWW] http://www.postgresql.org/docs/9.4/static/datatype-character.html (15.05.2016)

[24] PostgreSQL. 22.3. Character Set Support [WWW] http://www.postgresql.org/docs/9.4/static/multibyte.html (18.05.2016)

[25] Rõzova, N. Disainimustrid üldistusseoste realiseerimiseks SQL-andmebaasides : bakalaureusetöö. TTÜ Informaatikainstituut, 2011. [ONLINE]

[26] Saal, E. Ankurmodelleerimise mudelite realisatsiooni generaator PostgreSQL jaoks : magistritöö. TTÜ Informaatikainstituut, 2015. [ONLINE]

[27] Silverston, L. The Data Model Resource Book. Revised Edition, Volume 1. New York: John Wiley & Sons, Inc., 2001

[28] Top 5 People With the Longest Names. [WWW] http://www.historyrundown.com/top-5-people-with-the-longest-names/ (05.04.2016)

[29] Vellemaa, A. Mõned disainimustrid klassifikaatorite esitamiseks SQL andmebaasides : bakalaureusetöö. TTÜ Informaatikainstituut, 2015. [ONLINE]

Lisa 1 – Nimede struktuur erinevates riikides ja kultuurides

Tabel 2. Variatsioonid nimetüüpidega *eesnimi, perekonnanimi*.

Riik/ kultuur	Eesti, Poola, Tšehhi, Slovakkia, Albaania	Holland	Ungari, Horvaatia, Serbia	Jaapan	Hispaania	Brasiilia, Portugal
Nime struktuur	**eesn + peren**	**eesn + peren**	**peren + eesn**	**peren + eesn**	**eesn + isa esimene peren + ema esimene peren**	**eesn + peren**
Täisnimi	Adela Landová-Štychková	Johannes van der Burgh	Segesdy Kovács János	Suzuki Ichiro	Juan Pablo Fernández de Calderón García-Iglesias	José Eduardo Santos Tavares Melo Silva
Eesnimi	Adela	Johannes	János	Ichiro	Juan Pablo	José Eduardo
Perekonna-nimi	Landová-Štychková (*abielus)	van der Burgh	Segesdy Kovács	Suzuki	Fernández de Calderón García-Iglesias	Santos Tavares Melo Silva

Tabel 3. Variatsioonid nimetüüpidega *religioosne nimi, aunimi*.

Riik/ kultuur	India (3)	Pakistan (moslem, naine)	Pakistan (moslem, mees) (2)
Nime struktuur	**eesn + religioosne n+ (peren)**	**eesn + aun + (peren) (jgas järjekorras)**	**aun + eesn + isanimi + (peren)**
Täisnimi	Ravinder Singh Sahota	Jameela Khatoon	Abu Muhammad Ahmad Husain
Eesnimi	Ravinder	Jameela	Ahmad
Religioosne nimi	Singh		
Aunimi		Khatoon	Abu Muhammad
Isanimi			Husain
Perekonnanimi	Sahota		

Tabel 4. Variatsioonid nimetüüpidega *isanimi/emanimi, keskmine nimi*.

Riik/ kultuur	Araabia (1)	Araabia (2)	Kreeka	Vene	India (4)	India (5)	Vietnam
Nime struktuur	**eesn + isan**	**eesn + isan + isan + peren**	**eesn + isan/ abikaasa eesn + peren**	**eesn + isan + peren**	**isa eesn + eesn**	**eesn + keskmine n (peren rollis)**	**peren + (keskmine n) + eesn**
Täisnimi	Ibrahim ibn Khaldun	Muhammad ibn Salman ibn Ameen Ahl-Farsi	Ekaterini Spyrou Kyprianou	Pjotr Iljitš Uljanov	Rasiah Laxmi	Raj Lal	Nguyen Van Nam
Eesnimi	Ibrahim	Muhammad	Ekaterini	Pjotr	Laxmi	Raj	Nam
Keskmine nimi						*Lal	Van
Isanimi/ emanimi	ibn Khaldun	ibn Salman ibn Ameen	...	Iljitš	Rasiah		
Abikaasa eesnimi			Spyrou (*abielus)				
Perekonna-nimi		Al-Farsi	Kyprianou	Uljanov		*Lal	Nguyen

Lisa 2 – Skriptid

Muster „Valik nimetüüpidest veergudes esinemishulga põhjal“

Skeemi loomise lause:

```
CREATE SCHEMA veerg_valik;
```

Tabeli loomise laused:

```
CREATE DOMAIN veerg_valik.nime_veerg AS text
  CONSTRAINT chk_nimi_ei_koosne_tyhikutest
  CHECK(VALUE!~'^[[:space:]]*$')
  CONSTRAINT chk_nime_pikkus_pole_yle_300
  CHECK(char_length(VALUE)<=300);

CREATE TABLE veerg_valik.Isik (
  isiku_id integer NOT NULL,
  eesnimi veerg_valik.nime_veerg NOT NULL,
  keskmine_nimi veerg_valik.nime_veerg,
  isanimi_emanimi veerg_valik.nime_veerg,
  religioosne_nimi veerg_valik.nime_veerg,
  aunimi veerg_valik.nime_veerg,
  perekonnanimi veerg_valik.nime_veerg,
  CONSTRAINT PK_Isik PRIMARY KEY (isiku_id)
);
```

Andmete lisamise laused:

```
INSERT INTO veerg_valik.Isik (isiku_id, eesnimi, isanimi_emanimi)
VALUES
  (1, 'Björk', 'Guðmundsdóttir'),
  (2, 'Rasiah', 'Alagaratnam');

INSERT INTO veerg_valik.Isik (isiku_id, eesnimi, perekonnanimi) VALUES
  (3, 'Zedong', 'Mao'),
  (4, 'Johannes', 'van der Gaals');

INSERT INTO veerg_valik.Isik (isiku_id, eesnimi, religioosne_nimi,
perekonnanimi) VALUES
  (5, 'Hafiz', 'Muhammad', 'Khan');
```

```
INSERT INTO veerg_valik.Isik (isiku_id, eesnimi, keskmine_nimi,
perekonnanimi) VALUES
  (6, '3ric', 'J.', 'Bitt');

INSERT INTO veerg_valik.Isik (isiku_id, eesnimi, aunimi) VALUES
  (7, 'Jameela', 'Khatoon');
```

Muster „Täisnime veerg“

Skeemi loomise lause:

```
CREATE SCHEMA veerg_taisnimi;
```

Tabeli loomise lause:

```
CREATE TABLE veerg_taisnimi.Isik (
  isiku_id integer NOT NULL,
  taisnimi text NOT NULL,
  CONSTRAINT PK_Isik PRIMARY KEY (isiku_id),
  CONSTRAINT chk_nimi_ei_koosne_tyhikutest
  CHECK(taisnimi!~'^[[:space:]]*$'),
  CONSTRAINT chk_nime_pikkus_pole_yle_300
  CHECK(char_length(taisnimi)<=300)
);
```

Andmete lisamise lause:

```
INSERT INTO veerg_taisnimi.Isik (isiku_id, taisnimi) VALUES
  (1, 'Conjeepuram Mahendrum Krishnamurthy'),
  (2, 'Suzuki Ichiro'),
  (3, 'Adela Landová-Štychková'),
  (4, 'Muhammad ibn Salman ibn Ameen Ahl-Farsi'),
  (5, 'Juan Pablo Fernández de Calderón García-Iglesias'),
  (6, 'Ravinder Singh Sahota'),
  (7, 'Tom O''Neil');
```

Muster „Nimetüüpide tabelid“

Skeemi loomise lause:

```
CREATE SCHEMA nimetyybi_tabelid;
```

Tabelite loomise laused:

```
CREATE DOMAIN nimetyybi_tabelid.nime_veerg AS text NOT NULL
  CONSTRAINT chk_nimi_ei_koosne_tyhikutest
  CHECK(VALUE!~'^[[:space:]]*$')
  CONSTRAINT chk_nime_pikkus_pole_yle_300
  CHECK(char_length(VALUE)<=300);

CREATE TABLE nimetyybi_tabelid.Isik (
  isiku_id integer NOT NULL,
  CONSTRAINT PK_Isik PRIMARY KEY (isiku_id)
);

CREATE TABLE nimetyybi_tabelid.Aunimi (
  nimi nimetyybi_tabelid.nime_veerg,
  algusaeg date NOT NULL,
  loppaeg date,
  isiku_id integer NOT NULL,
  CONSTRAINT PK_Aunimi PRIMARY KEY (nimi, algusaeg, isiku_id),
  CONSTRAINT chk_Algusaeg_ei_ole_tulevikus
  CHECK(algusaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_tulevikus
  CHECK(loppaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_enne_kui_algusaeg
  CHECK(loppaeg>=algusaeg),
  CONSTRAINT FK_Aunimi_Isik FOREIGN KEY (isiku_id) REFERENCES
  nimetyybi_tabelid.Isik (isiku_id) ON DELETE CASCADE ON UPDATE NO
  ACTION
);

CREATE TABLE nimetyybi_tabelid.Eesnimi (
  nimi nimetyybi_tabelid.nime_veerg,
  algusaeg date NOT NULL,
  loppaeg date,
  isiku_id integer NOT NULL,
  CONSTRAINT PK_Eesnimi PRIMARY KEY (nimi, algusaeg, isiku_id),
  CONSTRAINT chk_Algusaeg_ei_ole_tulevikus
  CHECK(algusaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_tulevikus
  CHECK(loppaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_enne_kui_algusaeg
  CHECK(loppaeg>=algusaeg),
  CONSTRAINT FK_Eesnimi_Isik FOREIGN KEY (isiku_id) REFERENCES
  nimetyybi_tabelid.Isik (isiku_id) ON DELETE CASCADE ON UPDATE NO
  ACTION
);
```

```
CREATE TABLE nimetyybi_tabelid.Isanimi_emanimi (
  nimi nimetyybi_tabelid.nime_veerg,
  algusaeg date NOT NULL,
  loppaeg date,
  isiku_id integer NOT NULL,
  CONSTRAINT PK_Isanimi_emanimi PRIMARY KEY (nimi, algusaeg,
  isiku_id),
  CONSTRAINT chk_Algusaeg_ei_ole_tulevikus
  CHECK(algusaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_tulevikus
  CHECK(loppaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_enne_kui_algusaeg
  CHECK(loppaeg>=algusaeg),
  CONSTRAINT FK_Isanimi_emanimi_Isik FOREIGN KEY (isiku_id)
  REFERENCES nimetyybi_tabelid.Isik (isiku_id) ON DELETE CASCADE ON
  UPDATE NO ACTION
);

CREATE TABLE nimetyybi_tabelid.Keskmine_nimi (
  nimi nimetyybi_tabelid.nime_veerg,
  algusaeg date NOT NULL,
  loppaeg date,
  isiku_id integer NOT NULL,
  CONSTRAINT PK_Keskmine_nimi PRIMARY KEY (nimi, algusaeg, isiku_id),
  CONSTRAINT chk_Algusaeg_ei_ole_tulevikus
  CHECK(algusaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_tulevikus
  CHECK(loppaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_enne_kui_algusaeg
  CHECK(loppaeg>=algusaeg),
  CONSTRAINT FK_Keskmine_nimi_Isik FOREIGN KEY (isiku_id) REFERENCES
  nimetyybi_tabelid.Isik (isiku_id) ON DELETE CASCADE ON UPDATE NO
  ACTION
);

CREATE TABLE nimetyybi_tabelid.Perekonnanimi (
  nimi nimetyybi_tabelid.nime_veerg,
  algusaeg date NOT NULL,
  loppaeg date,
  isiku_id integer NOT NULL,
  CONSTRAINT PK_Perekonnanimi PRIMARY KEY (nimi, algusaeg, isiku_id),
  CONSTRAINT chk_Algusaeg_ei_ole_tulevikus
  CHECK(algusaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_tulevikus
  CHECK(loppaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_enne_kui_algusaeg
  CHECK(loppaeg>=algusaeg),
  CONSTRAINT FK_Perekonnanimi_Isik FOREIGN KEY (isiku_id) REFERENCES
  nimetyybi_tabelid.Isik (isiku_id) ON DELETE CASCADE ON UPDATE NO
  ACTION
);
```

```
CREATE TABLE nimetyybi_tabelid.Religioosne_nimi (
  nimi nimetyybi_tabelid.nime_veerg,
  algusaeg date NOT NULL,
  loppaeg date,
  isiku_id integer NOT NULL,
  CONSTRAINT PK_Religioosne_nimi PRIMARY KEY (nimi, algusaeg,
  isiku_id),
  CONSTRAINT chk_Algusaeg_ei_ole_tulevikus
  CHECK(algusaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_tulevikus
  CHECK(loppaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_enne_kui_algusaeg
  CHECK(loppaeg>=algusaeg),
  CONSTRAINT FK_Religioosne_nimi_Isik FOREIGN KEY (isiku_id)
  REFERENCES nimetyybi_tabelid.Isik (isiku_id) ON DELETE CASCADE ON
  UPDATE NO ACTION
);
```

Andmete lisamise laused:

```
INSERT INTO nimetyybi_tabelid.Isik (isiku_id) VALUES
  (1),
  (2),
  (3),
  (4),
  (5),
  (6);

INSERT INTO nimetyybi_tabelid.Eesnimi(nimi, algusaeg, isiku_id) VALUES
  ('Mari', '1980-01-01', 1),
  ('Georg', '1960-01-01', 2),
  ('Björk', '1986-01-01', 3),
  ('Hafiz', '1930-01-01', 4),
  ('Ilja', '1999-01-01', 5),
  ('Jameela', '1900-01-01', 6);

INSERT INTO nimetyybi_tabelid.Keskmine_nimi (nimi, algusaeg, isiku_id)
VALUES
  ('Veroonika', '1980-01-01', 1),
  ('M.', '1960-01-01', 2);

INSERT INTO nimetyybi_tabelid.Isanimi_emanimi (nimi, algusaeg,
isiku_id) VALUES
  ('Guðmundsdóttir', '1986-01-01', 3),
  ('Nikolajevitš', '1999-01-01', 5);
```

```
INSERT INTO nimetyybi_tabelid.Perekonnanimi (nimi, algusaeg, isiku_id)
VALUES
  ('Kuusepuu', '1980-01-01', 1),
  ('Smith', '1960-01-01', 2),
  ('Khan', '1930-01-01', 4),
  ('Uljanov', '1999-01-01', 5);

INSERT INTO nimetyybi_tabelid.Aunimi (nimi, algusaeg, isiku_id) VALUES
  ('Khatoon', '1900-01-01', 6);

INSERT INTO nimetyybi_tabelid.Religioosne_nimi (nimi, algusaeg,
isiku_id) VALUES
  ('Muhammad', '1930-01-01', 4);

UPDATE nimetyybi_tabelid.Eesnimi SET loppaeg = '2000-01-01' WHERE
isiku_id = 5;

INSERT INTO nimetyybi_tabelid.Eesnimi(nimi, algusaeg, isiku_id) VALUES
  ('Konstantin', '2000-01-01', 5);
```

Muster „Universaalne disain järjekorraga“

Skeemi loomise lause:

```
CREATE SCHEMA universaalne;
```

Tabelite loomise laused:

```
CREATE DOMAIN universaalne.nime_veerg AS text NOT NULL
  CONSTRAINT chk_nimi_ei_koosne_tyhikutest
  CHECK(VALUE!~'^[[:space:]]*$')
  CONSTRAINT chk_nime_pikkus_pole_yle_300
  CHECK(char_length(VALUE)<=300);
CREATE TABLE universaalne.Isik (
  isiku_id integer NOT NULL,
  CONSTRAINT PK_Isik PRIMARY KEY (isiku_id)
);

CREATE TABLE universaalne.Nimetyyp (
  nimetyybi_kood smallint NOT NULL,
  nimetyyp text NOT NULL,
  CONSTRAINT PK_Nimetyyp PRIMARY KEY (nimetyybi_kood),
  CONSTRAINT UQ_Nimetyyp_nimetyyp UNIQUE (nimetyyp),
  CONSTRAINT chk_nimetyyp_ei_koosne_tyhikutest
  CHECK(nimetyyp!~'^[[:space:]]*$')
);
```

```
CREATE TABLE universaalne.Nimi (
  isiku_id integer NOT NULL,
  nimi universaalne.nime_veerg,
  nimetyybi_kood integer NOT NULL,
  algusaeg date NOT NULL,
  loppaeg date,
  jarjekorranumber smallint NOT NULL,
  CONSTRAINT PK_Nimi PRIMARY KEY (isiku_id, nimi, nimetyybi_kood,
  algusaeg),
  CONSTRAINT chk_Loppaeg_ei_ole_enne_kui_algusaeg
  CHECK(loppaeg>=algusaeg),
  CONSTRAINT chk_Algusaeg_ei_ole_tulevikus
  CHECK(algusaeg<=CURRENT_DATE),
  CONSTRAINT chk_Loppaeg_ei_ole_tulevikus
  CHECK(loppaeg<=CURRENT_DATE),
  CONSTRAINT FK_Nimi_Isik FOREIGN KEY (isiku_id) REFERENCES
  universaalne.Isik (isiku_id) ON DELETE CASCADE ON UPDATE NO ACTION,
  CONSTRAINT FK_Nimi_Nimetyyp FOREIGN KEY (nimetyybi_kood) REFERENCES
  universaalne.Nimetyyp (nimetyybi_kood) ON DELETE NO ACTION ON
  UPDATE CASCADE
);
```

Andmete lisamise laused:

```
INSERT INTO universaalne.Isik(isiku_id) VALUES
  (1),
  (2),
  (3),
  (4);

INSERT INTO universaalne.Nimetyyp(nimetyybi_kood, nimetyyp) VALUES
  (1, 'eesnimi'),
  (2, 'isanimi_keskmise_nimena'),
  (3, 'perekonnanimi'),
  (4, 'eesnimi_romaniseeritud'),
  (5, 'generatsiooni_nimi_romaniseeritud'),
  (6, 'perekonnanimi_romaniseeritud'),
  (7, 'generatsiooni_nimi'),
  (8, 'keskmine_nimi'),
  (9, 'abikaasa_eesnimi');

INSERT INTO universaalne.Nimi(isiku_id, nimi, nimetyybi_kood,
algusaeg, jarjekorranumber) VALUES
  (1, 'Иля', 1, '1980-01-01', 1),
  (1, 'Николайевитш', 2, '1980-01-01', 2),
  (1, 'Уланов', 3, '1980-01-01', 3);
```

```
INSERT INTO universaalne.Nimi(isiku_id, nimi, nimetyybi_kood,
algusaeg, jarjekorranumber) VALUES
  (2, 'Mao', 6, '1904-01-01', 1),
  (2, 'Ze', 5, '1904-01-01', 2),
  (2, 'Dong', 4, '1904-01-01', 3);

INSERT INTO universaalne.Nimi(isiku_id, nimi, nimetyybi_kood,
algusaeg, loppaeg, jarjekorranumber) VALUES
  (3, 'John', 1, '1955-01-01', '1990-01-01', 1);

INSERT INTO universaalne.Nimi(isiku_id, nimi, nimetyybi_kood,
algusaeg, jarjekorranumber) VALUES
  (3, 'John Nick', 1, '1990-01-01', 1),
  (3, 'Patrick', 8, '1955-01-01', 2),
  (3, 'Bitt', 3, '1955-01-01', 3),
  (3, 'Jr.', 7, '1955-01-01', 4);

INSERT INTO universaalne.Nimi(isiku_id, nimi, nimetyybi_kood,
algusaeg, jarjekorranumber) VALUES
  (4, 'Ekaterini', 1, '1970-01-01', 1),
  (4, 'Nikolaou', 2, '1970-01-01', 2),
  (4, 'Kyprianou', 3, '1970-01-01', 3);

UPDATE universaalne.Nimi SET loppaeg = '2000-01-01' WHERE isiku_id = 4
AND nimetyybi_kood = 2;

INSERT INTO universaalne.Nimi(isiku_id, nimi, nimetyybi_kood,
algusaeg, jarjekorranumber) VALUES
  (4, 'Spyrou', 9, '2000-01-01', 2);

INSERT INTO universaalne.Isik(isiku_id) VALUES
  (5);
```