TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Katrin Aibast 135166IAPB

BAASTABELITE KITSENDUSTE NIMEDE ÜHTLUSTAMISEKS MÕELDUD POSTGRESQL LAIENDUSE LOOMINE

Bakalaureusetöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2018

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: Katrin Aibast

16.05.2018

Annotatsioon

Käesoleva töö põhieesmärgiks on luua PostgreSQL andmebaasisüsteemi laiendus andmebaasis olevate baastabelite kitsenduste ühtlustamiseks. Esmalt uuriti kolme avatud lähtekoodiga tarkvara (FusionForge (versioon 6.1beta1) [1], LedgerSMB (versioon 1.5.20) [2] ja OTRS (versioon 6.0.6) [3]) poolt kasutatavate PostgreSQL andmebaaside põhjal, milliseid nimetamise tavasid kitsenduste nimetamisel kasutatakse ja milline on üldine nimetamise olukord nendes andmebaasides. Tehtud analüüsist selgus, et mitte üheski vaadeldud andmebaasis pole kitsenduste nimetamine järjekindel. Kasutusel oli palju erinevaid kitsenduste nimede mustreid. Selline järjekindlusetus teeb andmebaasi haldamise keerulisemaks. Hästi valitud kitsenduste nimed muudavad lihtsamaks kitsenduste kustutamise, andmemuudatustel tekkinud veateadetest arusaamise ja andmekäitluskeele lausete täitmisplaanide mõistmise. Seega oleks taoline laiendus kindlasti kasulik töövahend igale andmebaasi arendajale, sest see aitab viia andmebaasi paremini kooskõlla puhta koodi põhimõtetega.

Erinevates kirjandusallikates põhinevatel kitsenduste nimetamise soovitustel tuginedes valmis PostgreSQL andmebaasisüsteemi laiendus, mida saab andmebaasi `CREATE EXTENSION` lausega lisada. Laiendus arvestab üldisi häid tavasid kitsenduste nimetamisel ning võimaldab ühe käsuga ja ühe mustri alusel ümber nimetada kas kõik andmebaasi kitsendused või ainult ühte tüüpi kitsendused. Tulemuseks on ühte stiili järgivate nimedega kitsendused.

Laiendus on avatud lähtekoodiga ning publitseeritud MIT litsentsiga. Tarkvara lähtekood on kättesaadav:

- GitHubi koduleht: https://github.com/katrinaibast/constr_name_unif
- PostgreSQL Extension Network: https://pgxn.org/dist/constr_name_unif

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 52 leheküljel, 6 peatükki, 19 joonist, 8 tabelit.

Abstract

Creating a PostgreSQL Extension for Uniforming the Names of Base Table Constraints

The main aim of this work is to create a PostgreSQL extension, which can be installed to a database by `CREATE EXTENSION` statement, for uniforming the names of base table constraints. First, I conducted a research, based on the PostgreSQL databases of three open source systems (FusionForge (version 6.1beta1) [1], LedgerSMB (version 1.5.20) [2] and OTRS (version 6.0.6) [3]), to find out used conventions for naming constraints and what is the state of the art of naming database objects in general.

Good intention-revealing names and following certain naming conventions are a part of clean code [4] practices. Naming things is one of the hardest things in Computer Science [5]. Bad naming habits make administrating and further development of any system, including databases, significantly more difficult and it also makes it harder to read and understand the code. Specifically, bad naming of database constraints makes the administration of the database schema more complex and also makes it tricky to understand error messages as well as execution plans of data manipulation statements.

From the analysis, it turned out that none of the observed databases had complete consistency in naming constraints. There were many different patterns by which the constraints were named. That kind of inconsistency makes database administration more difficult. Therefore, an extension for uniforming constraints names would certainly be a useful tool for any database developer.

The extension is open source and published with the MIT licence. The manual and the source code are available at:

- GitHub: https://github.com/katrinaibast/constr_name_unif
- PostgreSQL Extension Network: https://pgxn.org/dist/constr_name_unif

The thesis is in Estonian and contains 52 pages of text, 6 chapters, 19 figures, 8 tables.

Lühendite ja mõistete sõnastik

Andmebaasiobjekti identifikaator e nimi	Andmebaasis loodud andmebaasiobjektile antud nimi. Sõltuvalt realisatsioonist võib selle nime anda andmebaasiobjekti looja või süsteem ise. Nime kasutades on võimalik andmebaasiobjektile viidata. Näiteks kui tegemist on andmebaasis loodud andmestruktuuri või selle osa nimega, siis selle alusel saab andmestruktuurist andmeid küsida. See nimi on osa nende andmete loogilisest aadressist (vastandina füüsilisele aadressile, mis kirjeldab andmete füüsilist asukohta salvestusseadmetel).
Andmekirjelduskeel	***Data Definition Language, DDL*** Andmebaasikeele alamkeel. Keel, mida kasutatakse andmebaasi struktuuri ja käitumise kirjeldamiseks andmebaasi tasemel. SQL andmebaasikeeleles kuuluvad sellesse alamkeelde `CREATE`, `ALTER` ja `DROP` laused. Selle keele lausetes määratakse ära erinevate andmebaasiobjektide identifikaatorid e nimed.
Andmekäitluskeel	***Data Manipulation Language, DML*** Andmebaasikeele alamkeel. Keel, mida kasutatakse andmebaasis registreeritud faktide hulga muutmiseks, st andmete lisamiseks, muutmiseks või kustutamiseks. SQL andmebaasikeeleles kuuluvad sellesse alamkeelde `SELECT`, `INSERT`, `UPDATE`, `DELETE` ja `MERGE` laused. Selle keele lausetes kasutatakse erinevate andmebaasiobjektide identifikaatoreid e nimesid.
FusionForge	FusionForge on tarkvaraarenduse planeerimiseks mõeldud rakendus, millega saab projektiga seotud olevatele asjaosalistele ülesandeid koostada, arendust dokumenteerida, vigu jälgida ning kasutada versioonihaldust [1].
LedgerSMB	***Ledger Small Medium Business*** Väikeste-keskmiste ettevõtete pearaamat [2]
OTRS	***Open Ticket Request System (Open Technology – Real Service)*** Vabavaraline veahaldussüsteem organisatsioonide päringute vastuvõtmiseks ja haldamiseks [3]

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Hea ja ühtse stiiliga järjekindel asjade nimetamine on osa puhta koodi praktikatest [4]. Heade nimede andmine on raske ülesanne (seda nii rakenduste kirjutamisel kui ka andmebaasiobjekte luues) [5]. Halb nimetamine teeb süsteemi, sealhulgas ka andmebaasi, haldamise ja edasiarendamise oluliselt keerulisemaks ning koodi raskesti loetavaks. Eriti suureks probleemiks muutuvad halvad nimed juhul kui süsteemi arendamisega tegelevad erinevad inimesed (mis on tavapärane). Andmebaasis defineeritud kitsenduste halb nimetamine raskendab andmebaasi skeemi haldamist, andmekäitluskeele lausete täitmisplaanidest arusaamist ja andmesisestusel tekkivate veateadete mõistmist.

Lõputöö peamiseks eesmärgiks on luua PostgreSQL andmebaasisüsteemile laiendus (nii nagu tehti töös [6]) andmebaasis olevate kitsenduste nimede ühtlustamiseks. Laienduse vajalikkust aitab põhjendada eelnevalt mõne avatud lähtekoodiga rakenduse andmebaasis olevate andmebaasiobjektide nimede uurimine ja analüüsimine. Olemasolevate andmebaaside uurimise tulemus aitab aru saada, kui suur vajadus võib loodavale laiendusele olla ning millises ulatuses võiks laiendus rakendust leida. Soov on korrata allikates [7], [8] esitatud uuringut andmebaasi objektide identifikaatorite põhjal.

Laiendus luuakse PostgreSQL andmebaasisüsteemis, sest see on üsna laialdasel kasutusel ning seetõttu ka hea kasutaja- ja kogukonnatoega. Andmebaasisüsteemide populaarsuse indeksi kohaselt [9] on see 2018. aasta mais populaarsuselt neljas andmebaasisüsteem. Lisaks kasutatakse seda avatud lähtekoodiga süsteemides, mis võimaldab vajalikku uurimust teha ning sellesse on sisse ehitatud võimalus andmebaasisüsteemile laienduste abil uut funktsionaalsust lisada. PostgreSQL andmebaasisüsteemis ei ole eraldi süsteemi-defineeritud funktsioone, mis nimetamise stiili ja ühtlust kontrollivad, ega ka seda parandavaid funktsioone. Töös tegemiseks kasutatakse PostgreSQL andmebaasisüsteemi versiooni 10.

Kasutatavaks meetodiks on disainiteadus [10], mille tulemuseks on tehniline artefakt. Antud lõputöö puhul on selleks PostgreSQL laiendus, mida katsetatakse valideerimiseks konkreetse andmebaasi kitsenduste nimede ühtlustamiseks.

Töö on jaotatud nelja ossa. Esimeses osas on teoreetiline ülevaade andmebaasi kitsenduste nimetamise kohta. Sellele järgneb kolme avatud lähtekoodiga süsteemi andmebaaside analüüs kitsenduste nimetamise seisukohalt. Analüüsi tulemusena tuuakse välja tähelepanekud uuritud andmebaasides esinenud probleemidest ning esitatakse mõtteid kuidas oleks olnud parem toimida. Töö kolmandas osas kavandatakse ja realiseeritakse PostgreSQL laiendus. Neljandas osas katsetatakse loodud laiendust ja teavitatakse lugejaid tulemustest.

2 Nimede olulisusest

Tarkvaraarenduse käigus on oluline saavutada võimalikult puhas lähtekood (edaspidi kood) (*clean code*). See tagab rühmas töötades üksteise koodi mõistmise ja hea hallatavuse, lisaks on sel juhul ka koodist lihtsam vigu üles leida. Kui koodi loomisel on järgitud puhta koodi põhimõtteid, siis teised arendajad peaksid üldjuhul sellest, kuigi nende jaoks võõrast koodist, küllaltki lihtsalt aru saama [11]. Suure tõenäosusega peab koodi kirjutaja kunagi selle juurde ka ise tagasi pöörduma ning koodi hea struktuur ning esitus muudavad talle sellesse uuesti sisseelamise lihtsamaks.

Koodist kõige suurema osa, ligikaudu 70%, moodustavad erinevate objektide nimed e identifikaatorid [7]. Eelnev hinnang põhineb rakenduse koodi uurimisel. Joonis 1 esitab `CREATE TABLE` lause PostgreSQL andmebaasisüsteemi jaoks. Selles lauses sisaldub tabeli nimi, veergude nimed, andmetüüpide nimed, kitsenduse nimi, funktsiooni nimi. Nimedele on lauses joon alla tõmmatud. Kokku on selles lauses 14 nime. Wordi märkide lugemise funktsionaalsus loeb tühikuid arvestamata Joonis 1 olevast lausest kokku 225 märki ning ilma nimedeta lausest 87 märki (vt Joonis 2). Seega moodustavad nimed Joonis 1 tekstimahust 138 märki ehk 61%. Lisaks nimedele on SQL lauses ka võtmesõnad (nt `CREATE`), literaalid (nt 0) ja SQL spetsiaalmärgid (nt semikoolon).

```
CREATE TABLE user_bookmarks (
bookmark_id SERIAL NOT NULL,
user_id INTEGER DEFAULT 0 NOT NULL,
reg_time TIMESTAMP NOT NULL DEFAULT LOCALTIMESTAMP(0),
bookmark_url TEXT NOT NULL,
bookmark_title TEXT,
CONSTRAINT pk_user_bookmarks PRIMARY KEY (bookmark_ id));
```

Joonis 1. Tabeli loomise SQL lause [12].

```
CREATE TABLE (
NOT NULL,
DEFAULT 0 NOT NULL,
NOT NULL DEFAULT (0),
NOT NULL,
,
CONSTRAINT PRIMARY KEY ());
```

Joonis 2. Tabeli loomise SQL lause, kust nimed on eemaldatud.

Osad SQL lauses viidatud nimedest on andmebaasi arendaja välja mõeldud (tabeli nimi, veergude nimed, kitsenduse nimi), osad süsteemi-defineeritud (süsteemi-defineeritud tüüpide ja funktsioonide nimed), st neile andsid nime andmebaasisüsteemi arendajad.

Tuleb nõustuda, et nimetamine üks tähtsamaid puhta koodi aspekte. Tarkvara elementidele nime andmine võib esmapilgul tunduda üsna triviaalne, aga tegelikult on see üks keerulisemaid ülesandeid tarkvaararenduse valdkonnas [5]. Erinevate funktsioonide, muutujate ja muude tarkvara elementide nimetamine peaks puhta koodi põhimõtetest lähtuvalt olema läbi projekti järjekindel ning ühtset stiili jälgiv. Elemente nimetades peaks sellele andma võimalikult ilmeka nime, mis kajastab täpselt ja arusaadavalt elemendi tähendust kasutaja jaoks. Näiteks funktsioonide puhul peaks nimi kirjeldama selle tegevust ja muutujate puhul selle oodatavaid väärtuseid. Samas tuleks vältida sünonüüme. Näiteks Joonis 1 olevas SQL lauses oleks tüübi nime INTEGER asemel saanud kasutada nimesid INT ja INT4. Nende nimede läbisegi kasutamine ühes projektis ei mõju arusaadavusele hästi. Lisaks mängib olulist rolli nimetamistava (*naming convention*) valimine ja järjekindel järgimine. Nimetamistava võib näiteks määratleda milliseid alamkomponente nimes kasutada ja kuidas erinevaid alamkomponente üksteisega siduda (nt alakriipsud *vs.* suur- ja väiketähtede vaheldumine) [8]. Võib öelda, et iga nimetamistava kirjeldab võimaliku mustri, mida nimed võiksid järgida. Ühtse stiili määramiseks tuleks kasutatavad nimetamistavad valida enne arenduse algust ja kõik arendajad peaksid nendest kogu arenduse käigus kinni pidama. Kõige sobivam nimetamistava sõltub programmeerimiskeele iseärasustest, st erinevates keeltes ja erinevat tüüpi keeltes võib see olla erinev.

Minul isiklik kogemus ära määratud nimetavadega puudub. Olen liitunud kahe juba arendusjärgus arendustiimiga, kus kasutatavate nimede muster polnud kuskil dokumentatsioonis ära kirjeldatud ning pidin tarkvara elementide nimetamisel eeskuju võtma enne defineeritud elementides (seda nii andmebaasis kui ka programmikoodis). Seetõttu polnud kõik minu loodud nimed sama mustriga, mis eelnevalt loodud nimed ning pidin päris mitmeid kordi tegelema enda loodud elementide ümbernimetamisega.

Koodi puhastamine tähendab koodi refaktoreerimist. See seisneb olemasoleva koodi ümberkirjutamises ja paremaks muutmises, muutmata sealjuures programmi väljapoole paistvat käitumist [13]. Koodi halbadeks lõhnadeks (*code smell*) nimetatakse tundemärke probleemidest, mis ei ole programmivead, kuid raskendavad hiljem koodi haldamist,

edasiarendamist ja taaskasutamist. Kood refaktoreerimise ülesanne on koodis halvasti lõhnavad kohad üles leida, veenduda, et halb lõhn osutab probleemile ja asendada halvasti lõhnav koht uuega. Selle käigus tuleb tihti ette juhuseid, kus asju tuleb ümber nimetada, et kood oleks loetavam ning ühtlasi puhtam. Refaktoreerimise eesmärgiks on vähendada tehnilist võlga, mille tekkimine kiirustamisega üsna paratamatult kaasas käib, kuid mis hiljem muudab koodi haldamise, edasiarendamise ja taaskasutamise raskemaks [14]. Võlal on omadus ajas kasvada ning lõpuks kontrolli alt väljuda kui selle kontrolli all hoidmise ja vähendamisega igapäevaselt ei tegeleta. Nagu igasugune võlg tuleb ka tehniline võlg koos intressidega tagasi maksta. Intressideks on kulud, mis seostuvad halvasti kirjutatud koodist arusaamise ning koodi korrastamisega. Inimeste maailmas lõpeb kontrolli alt väljunud võlg pankrotiga, tarkvaraarenduse maailmas tehtud töö prügikasti viskamise ja puhtalt lehelt alustamisega.

Näiteks ühe SQL lause refaktoreerimine (vt Joonis 4).

```
create table staff (
primary key (id), id int NOT NULL,
firstName varchar(1000) not null,
pensindrawer smallint not null,
constraint pensindrawerrange
check(pensindrawer >= 1 and pensindrawer < 100)
);
```

Joonis 3. Halvasti loetav tabeli loomise lause.

```
CREATE TABLE staff (
staff_num        INT           NOT NULL,
given_name       VARCHAR(1000) NOT NULL,
pens_in_drawer SMALLINT        NOT NULL,
CONSTRAINT pl_staff PRIMARY KEY (staff_num),
CONSTRAINT pens_in_drawer_range
CHECK(pens_in_drawer >= 1 AND pens_in_drawer < 100)
);
```

Joonis 4. Refaktoreeritud tabeli loomise lause [15].

Koodi puhtuse tagamine ei piirdu koodis heade nimede kasutamisega (nt muudeti ka lause osade järjekorda, lisati reavahetusi ning võtmesõnad kirjutati suurtähtedega), kuid see on selle tegevuse oluline osa.

Andmebaasi realiseerimine on programmeerimine. Andmebaasis andmekirjelduskeele koodi käivitamise tulemuseks on andmebaasiobjektide hulga või omaduste muutumine. Kui valdavalt on puhta koodi käsitluste puhul juttu just rakenduse koodist [4], [7], siis tegelikult kehtib see ka andmebaaside kohta. Arendades andmebaase peaks muuhulgas

samamoodi kokku leppima andmebaasiobjektide nimetamise põhimõtted ja neid kogu arenduse käigus järjekindlalt järgima. Puhas kood andmebaaside korral tagab andmebaasi parema hallatavuse ja andmebaasiobjektidest erinevate arendajate poolt ühtemoodi arusaamise.

Sarnaselt kontseptuaalsetele mudelitele saab ka nimesid iseloomustada kolmes aspektis – süntaks, semantika ja pragmaatika [16]. Andmebaasiobjektide nimed peavad olema süntaktiliselt korrektsed, st järgima andmebaasisüsteemide kehtestatud reegleid. Näiteks peab PostgreSQL-is nimi algama tähe (a-z, kuid sobivad ka diakriitilise märgiga tähed või mitte ladina tähtedega) või alakriipsuga (_), kuid järgnevad tähemärgid võivad ka sisaldada araabia numbreid (0-9) või dollarimärki ($) [17]. Seda kontrolli teostavad andmebaasisüsteemid ja süntaktiliselt ebakorrektseid nimesid ei saa andmebaasis kasutada. Andmebaasiobjektide nimed peavad olema semantiliselt korrektsed, st peegeldama nimetatava objekti tähendust ja eesmärke. Näiteks eksitakse selle vastu, kui isikuandmete tabeli nimi on *Kaup*. Andmebaasiobjektide nimed peavad olema pragmaatiliselt hästi kasutatavad. Siia hulka kuulub nimede hea stiil ja selle järjekindel kasutamine. Näiteks eksitakse selle vastu, kui osade tabelite nimed on ainsuses ja osade nimed on mitmuses või kui osade tabelite nimed on eesti keeles ja osade nimed on inglise keeles. Refaktoreerimine tähendab nimede parandamist semantika ja pragmaatika aspektides.

Kuna antud töö raames valmib just kitsenduste nimesid ühtlustav PostgreSQL andmebaasisüsteemi laiendus, siis käsitlen eraldi jaotistes kitsenduste nimetamist SQL-andmebaasides.

2.1 Kitsenduste nimetamine SQL-andmebaasides

Andmebaasis olevad kitsendused piiravad andmeid, mida saab andmebaasis registreerida. Kitsenduste jõustamine aitab paranda andmete kvaliteeti ning jõustada ärireegleid. Tegemist on ennetava meetmega, mis üritab ilmselgelt valedest andmetest tekkivaid probleeme vältida, mitte probleeme tagantjärgi lahendada. Kitsendused ei taga andmete õigsust, kuid väldivad ilmselgelt ebaõigete andmete andmebaasi jõudmist. Kitsendused on andmebaasi mõttes nimega objektid, st igal kitsendusel on nimi. Kitsenduste nimetamiseks on välja pakutud mitmeid erinevaid tavasid ja mustreid. Hea praktika on kitsenduse nimes ära märkida, mis tabeli(te) ja veer(g)u(de)ga see on seotud ning lisada

juurde kas ees- või järelliide vastavalt kitsenduse tüübile [18], [19]. Sellist stiili läbi terve andmebaasi järgides saab kitsenduste haldamise lauseid väga lihtsalt kirjutada kitsenduse nime enne järgi vaatamata, sest selle saab kergesti tuletada vastavalt tabelile.

SQL tabelite loomise ja muutmise lauseid (`CREATE TABLE` ja `ALTER TABLE`) saab kirjutada nii, et seal kitsenduste nimesid ei mainita, vt Joonis 5. Sellisel juhul määrab need nimed andmebaasisüsteem, kusjuures erinevates andmebaasisüsteemides on kasutusel erinevad nimetamise põhimõtted. Alternatiiv on anda kitsendustele nimi juba tabeli loomise või muutmise lauses, vt Joonis 6. `NOT NULL` kitsendustele tavatsetakse tabelite loomise ja muutmise lausetes nime mitte anda, st selle nime annab süsteem.

```
CREATE TABLE lsmb_module (
id int not null unique,
label text primary key
);
```

Joonis 5. Kitsenduste defineerimine ilma neid nimetamata [20].

```
CREATE TABLE country_code (
 country_name character varying(80),
 ccode character(2) NOT NULL
);
ALTER TABLE ONLY country_code
ADD CONSTRAINT country_code_pkey PRIMARY KEY (ccode);
```

Joonis 6. Kitsenduse loomine koos selle kasutajapoolse nimetamisega [12].

Kui inimkasutaja ei anna loodud kitsendusele ise nime, siis nimetab PostgreSQL andmebaasisüsteem neid hästi ning loetavalt. Proovisin ise tabelit luues lisada sellele erinevaid kitsendusi ning süsteemi antud nimede mustrid olid järgmised:

- check kitsendus ühe veeru korral: `{table_name}_{column_name}_check`
- check kitsendus mitme veeru korral: `{table_name}_check`
- välisvõti: `{child_table_name}_{first_child_column_name}_fkey`
- primaarvõti: `{table_name}_pkey`
- unikaalsuse kitsendus: `{table_name}_{column_names}_key`
- exclude kitsendus: `{table_name}_{column_names}_excl`

Isiklikust kogemusest saan võrdluseks tuua näiteks Microsoft SQL Server andmebaasisüsteemi [21], kus kasutaja poolt nimetamata jäänud kitsendus saab nimeks `UQ_t2_40BBEA3A7F60ED59` [22]. Antud näitest loeb välja, et see on UNIQUE kitsendus, aga selle täpsem tähendus (mis veerud on unikaalsed) on inimesele mõistetamatu.

2.2 Kitsenduste nimetamise olulisus SQL-andmebaasides

Kitsenduse kustutamiseks SQL lausega on vaja teada selle nime. Kui arendaja pole nime ise andnud, siis peab ta hakkama seda andmebaasi süsteemikataloogist taga otsima, mis kulutab aega.

Teine põhjus kitsendusele hea nime andmiseks on veateadetest parem aru saamine. Kui mingi andmemuudatus rikub kitsendusega kehtestatud reeglit, siis kuvatakse kasutajale veateade rikutud kitsenduse kohta. Sageli on selles veateates kitsenduse nimi. Näiteks PostgreSQL-is kuvatakse kitsenduse nime `CHECK, UNIQUE, PRIMARY KEY, FOREIGN KEY` tüüpi kitsenduste vastu eksimisel. Kui kitsendusel on arusaamatu nimi, siis pole andmete muutjal (arendaja, testija) võimalik aru saada, mis reegli vastu ta just eksis [23], [24]. Lõppkasutajale ei peaks ilmselt kitsenduse nimega veateadet näitama. Prototüübi või esimese arenduse versioonide korral on see siiski aktsepteeritav ja siis on ka neile sisukas nimi väga oluline. Lõppkasutaja eelistaks täislausega vea kirjeldust, koos selgitusega, kuidas seda parandada. Näiteks PostgreSQL-is *check* kitsenduse vastu eksides esitatakse järgnev veateade: `ERROR: check constraint "{constraint_name}" is violated by some row`. Samuti annab viide kitsenduse realisatsioonile liigset infot ründajale, kes süsteemi vastu ebatervet huvi tunneb.

Kolmas põhjus kitsendusele hea nime andmiseks on SQL lausete täitmisplaanidest parem arusaamine. Andmebaasisüsteemid loovad primaarvõtme ja unikaalsuse kitsenduse toetuseks automaatselt indekseid, kusjuures nende nimi on samasugune kui kitsenduse nimi. Kui indeksi nimi on arusaamatu, siis on ka täitmisplaanist halb aru saada. . Näiteks kui Oracle andmebaasis käivitada Joonis 7 toodud laused on tulemuseks Joonis 8 esitatud täitmisplaan, kus viidatakse indeksil `SYS_C00115477`.

```
CREATE TABLE lsmb_module (
id number(10) not null unique,
label varchar(4000) primary key
);
SET AUTOTRACE ON;
SELECT * FROM lsmb_module WHERE id=1;
```

Joonis 7. Kitsenduste ja nende jõustamiseks vajalike indeksite süsteemipoolse nimetamise uurimine.

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     1 |  2015 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| LSMB_MODULE   |     1 |  2015 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | SYS_C00115477 |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Joonis 8. Täitmisplaani näide Oracle andmebaasisüsteemis.

2.3 Kirjanduses pakutud soovitused kitsenduste nimetamiseks SQL-andmebaasides

Kitsenduste nimetamine andmebaasis sõltub väga paljult selle arendajatest. Ideaalis lepitakse arenduse alguses ühtne nimetamistava kokku, mida siis arenduses järgima hakatakse.

Ühtset rahvusvahelist standardid kitsenduste nimetamiseks SQL-andmebaasides ei ole. Erinevaid soovitusi nimetavade kirjutamiseks leiab veebist väga palju. PostgreSQL-i dokumentatsioon soovitas ainult kitsendustele anda selgitava ja sisuka nime, et veateate kuvamisel saaks kasutaja tehtud veast aru, aga otseselt ei andnud mustrit, kuidas nimi kasutaja poolt genereerida [25]. Tarkvaraarendajate foorumis *Stack Overflow* on ära märgitud, kuidas PostgreSQL ise kitsendusi nimetab [18]. PostgreSQL nimetab kitsendusi inimestele loetavalt ning see nimetamise tava sobiks ka hästi võtta andmebaasis läbivalt kasutatavaks nimetavaks. Selle järgi tuleb kitsendus nimetada järgmise mustri järgi: `{table_name}_{column_name(s)}_{suffix}`, kus *suffix* kohale märgitakse lühendina ära, mis tüüpi kitsendusega on tegemist. Tüüpide lühendid on järgnevad:

- `pkey` – primaarvõti,
- `key` – unikaalsuse kitsendus,
- `fkey` – välisvõti,
- `check` – *check* kitsendus.

Selle soovituse kohaselt tuleks nimetamisel kasutada ka *snake case*'i, st erinevad nime komponendid on üksteisest eraldatud alakriipsudega.

Sarnaselt eelmisele nimetamise stiilile, on OpenACS [19] dokumentatsioonis kitsenduste nimetamiseks välja toodud sama muster koos *snake case*-iga. Erinevuseks on ainult kitsenduste tüüpide lühendid.

- `pk – primaarvõti,`
- `un – unikaalsuse kitsendus,`
- `fk – välisvõti,`
- `ck – check kitsendus.`

OpenACS raamistik on mõeldud veebirakenduste loomiseks Oracle või PostgreSQL andmebaasidele.

Leian, et OpenACS soovitus nimetamiseks on igati asjakohane, kui seda stiili järgitakse kõikide kitsenduste nimetamisel. Näiteks antakse seal ka juhiseid nimede lühendamiseks. Oracle'is võib kitsenduse nimi olla maksimaalselt kuni 30 baiti ja PostgreSQL-is ilma süsteemi juhtparameetri väärtust muutmata kuni 63 baiti. Koodi ülekantavuse huvides oleks seega soovitav piirduda ka PostgreSQL-is kitsenduste nimetamisel 63 baidi jagu märkidega.

Kui PostgreSQL-i süsteemis kasutatakse kitsenduste nimetamisel segamini kasutaja- ja süsteemipoolset nimetamist, tuleks ikkagi nimesid ühtlustada, et nimed oleksid antud järjekindlalt.

Andmebaasiobjektide nimetamiseks leidub palju soovitusi ja mustreid lisaks *snake case*'i ja järelliidete kasutamisele. Mõnedes nendes kasutatakse hoopis *camel case*'i [26] ning eesliiteid [27]-[29]. *Camel case*'i puhul kirjutatakse nimes sõnad (nime osad) ilma neid eraldava tähemärgita kokku ning iga sõna kirjutatakse suure algustähega. Mõnede süsteemide korral eelistavad arendajad *camel case*'i kasutada esisuurtähega, teised jälle mitte (*CamelCase vs camelCase*). Esisuurtähega *camel case*'i nimetatakse vahel ka *Pascal case*'iks. Samas ei ole SQL-andmebaasides nimedes suur- ja väiketähtede eristamine hea praktika, sest nime salvestamisel ilma seda jutumärkidesse panemata, salvestatakse kogu nimi üldjuhul, sõltuvalt süsteemist, kas väiketähtedes või suurtähtedes. Sellist nime kutsutakse regulaarseks identifikaatoriks. Juhul kui soovitakse nimes säilitada nii suur- kui väiketähti, tuleb nimi panna jutumärkidesse ehk võtta kasutusele piiritletud identifikaator. Siis saab nimes kasutada ka tühikuid ning nimi võib olla võtmesõna. Piiritletud identifikaatorite kasutamise korral aga ei loe andmebaasisüsteem näiteks nimesid *Name* ja *NAME* samaväärseteks, st nimed on tõstutundlikud. See muudab andmebaasi haldamise ning selles andmekäitluse operatsioonide tegemise jällegi keerukamaks.

Erinevatest allikatest on [27]-[29] näha, et kasutatakse rohkem lühemaid kitsenduste tüüpide lühendeid (*PK*, *FK* jne), kui PostgreSQL-i poolt automaatselt antud nimedes kasutatavad kitsenduste lühendid.

3 SQL-andmebaaside andmebaasiobjektide nimetamise uuring

Selles peatükis esitatakse töö analüütiline osa, mis seisneb vabavaraliste rakenduste SQL-andmebaaside erinevate andmebaasiobjektide nimetamise uurimises.

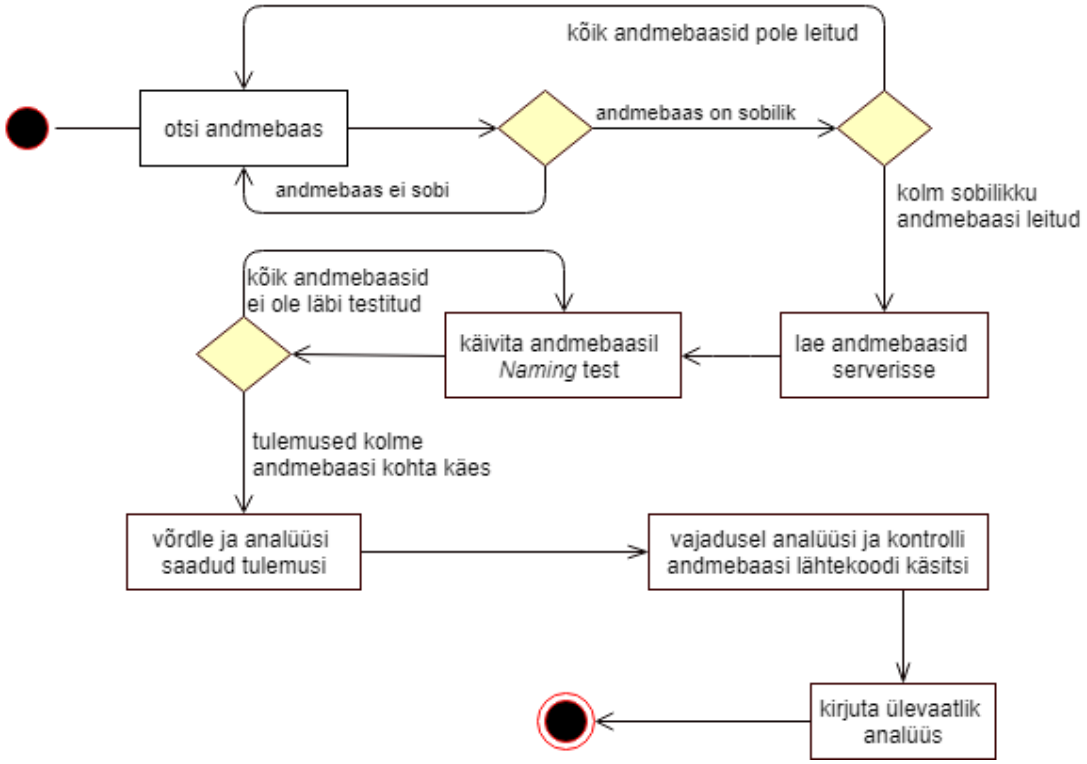
Töö analüütilise osa eesmärgiks on uurida, kuidas on nimetatud erinevate avatud lähtekoodiga tarkvarade SQL-andmebaaside andmebaasiobjekte, milliseid mustreid on erinevate andmebaasiobjektide nimetamisel kasutatud ning kui järjekindlalt on seda tehtud. Nimetamistavasid on uuritud eelnevalt vaid rakenduste programmikoodis [7], [8] ning seepärast analüüsin, milliseid nimetamise probleeme esineb SQL-andmebaasides ja mida võiks seal paremini teha. Valisin teemaks SQL-andmebaasid, sest andmebaasisüsteemide populaarsuse indeksi [9] kohaselt on 2018. aasta alguse seisuga SQL-andmebaasisüsteemid kõige populaarsemad ja laialt levinumad andmebaasisüsteemid. Erinevalt paljudest NoSQL andmebaasisüsteemidest, kus andmebaasi tasemel pole võimalik skeemi (andmebaasi struktuuri ja reegleid) defineerida või on need võimalused piiratud, tuleb SQL-andmebaasi korral luua andmebaasi tasemel ilmutatud kujul skeem ja anda selles olevatele objektidele nimesid.

SQL-andmebaasides on andmebaasiobjektideks nii baastabelid (edaspidi tabelid) kui ka nende ümber olev ökosüsteem teistest objektidest nagu näiteks trigerid, vaated, indeksid ja arvujada generaatorid. Vaatlen nimetamise uuringus ka veergude ja kitsenduste nimesid, mis on osa tabelite kirjeldusest. Nimesid kutsutakse kirjanduses ka identifikaatoriteks, kuid kasutan töös terminit „nimi“.

3.1 Protsessi kirjeldus

Uuritavateks andmebaasideks peavad olema andmebaasid, mis toetavad PostgreSQL andmebaasisüsteemi versiooni 10, sest see on töö realiseerimise hetkel kasutatavas

serveris installeeritud. Lisaks leitud andmebaasi loomise skript ei tohiks vajada modifikatsioone ning peaks ilma nendeta töötama.



Joonis 9. Analüüsi protsess.

Uuritavateks andmebaasideks valisin kolm avatud lähtekoodiga rakenduse andmebaasi, mis toetavad PostgreSQL andmebaasisüsteemi, sest selle andmebaasisüsteemi jaoks on juba loodud ulatuslik hulk kontrollpäringuid. Valitud rakendused on avatud lähtekoodiga, sest nende koodile pääseb ligi ning uurimuse tulemusi saab avalikustada, erinevalt kommertstarkvarast, kus nii koodi kui tulemusi hoitakse salajas. Lisaks on sarnased uuringud [7], [8] tehtud just nimelt avatud lähtekoodiga tarkvaradel. Valisin kolm rakendust, sest kahe või ühe andmebaasi uurimisel ei saaks rääkida sageli esinevatest vigadest. Kuna antud lõputöö eesmärk on luua kitsenduste nimede ühtlustamiseks mõeldud laiendus, siis ei saanud uuritavate andmebaaside arvu ka suurendada, sest vastasel korral oleks analüüsi osa muutunud liiga suureks.

Otsisin rakendusi allikatest [30], [31]. Ainukeseks rangeks kriteeriumiks oli PostgreSQL 10 versiooni toetamine. Valisin analüüsiks esimesed kolm leitud rakendust, mille andmebaasis oli üle 100 tabeli ning mille andmebaasi loomise laused läksid ilma neid modifitseerimata tööle. Teiste potentsiaalsete rakenduste andmebaaside loomise skript

sisaldas aga näiteks suurel osal erinevate kasutajarollide loomise ja õiguste jagamise lauseid, mis takistasid skripti kohest käivitamist.

Esimene valitud rakendus on tarkvaraarenduse planeerimiseks mõeldud FusionForge (versioon 6.1beta1) [1], millega saab projektiga seotud olevatele asjaosalistele ülesandeid koostada, arendust dokumenteerida, vigu jälgida ning olemas on ka versioonihaldus. Teiseks rakenduseks valisin väikestele ja keskmistele ettevõtetele suunatud ressursside planeerimise tarkvara LedgerSMB (versioon 1.5.20) [2]. Viimaseks uuritavaks andmebaasiks valisin veahaldus- ja tugisüsteemi OTRS (versioon 6.0.6) [3], mida kasutatakse ettevõtete päringute haldamiseks. Kõigi kolme andmebaasi genereerimiseks vajalik lähtekood on GitHubis saadaval [12], [20], [33].

Lähtekoodid kõikidel andmebaasidel leidsin rakenduste GitHubi lehtedelt ning andmebaasi loomiseks mõeldud SQL lausetega tekitasin vajalikud andmebaasid TTÜ üliõpilastele mõeldud *apex.ttu.ee* serveris, kus töö tegemise ajal on PostgreSQL 10. Andmebaaside nimed on: *135166_FusionForge*, *135166_LedgerSMB* ja *135166_OTRS*. Ma ei käivitanud õiguste jagamise ega tabelitesse andmete lisamise lauseid.

Serveris on tööle pandud peamiselt üliõpilaste arendatavate andmebaaside kontrollimiseks rakendus [34], [35], kus saab serveris loodud PostgreSQL andmebaaside põhjal käivitada erinevaid süsteemikataloogil põhinevaid SQL päringuid. Iga sellise päringu ülesandeks on kas

- anda ülevaade mingist andmebaasi disaini puudutavast küsimusest (nt tuues välja sorteeritud nimekirja kõikidest tabelitega seotud kitsenduste nimedest),
- leida mingi mõõdiku väärtus (nt leides erinevate kasutatud nime alamkomponentide arvu üle kõikide andmebaasiobjektide nimede),
- leida disaini vigu või probleeme (nt leides nimed, mis ei järgi mõnda laialt levinud tava).

Idee on selles, et informatsioon andmebaasi kohta on andmebaasi süsteemikataloogi tabelites ning nende tabelite ja neile loodud vaadete põhjal SQL päringuid (`SELECT` lauseid) käivitades on võimalik andmebaasi ülesehituse kohta palju teada saada. Kasutatavad päringud ei puuduta süsteemikataloogi moodustavaid andmebaasiobjekte.

Kasutan andmebaaside uurimiseks nimetatud vahendis loodud testi *Naming*, milles on töö tegemise ajal 59 päringut andmebaasiobjektide nimede statistika ja probleemide välja toomiseks ning nimekasutusest parema ülevaate saamiseks. Käivitades samu päringuid erinevates andmebaasides, saab selle tulemusena neid andmebaase võrrelda ja leida nimetamise probleeme. Kuna antud töö raames valmib just kitsenduste nimesid ühtlustav PostgreSQL andmebaasisüsteemi laiendus, siis uurin muudest objektidest eraldi ka just kitsenduste nimetamisega kaasnevaid iseärasusi. *Naming* testpäringute kasutajaliidese pildid on toodud Lisas 1 Joonis 20 ja Joonis 21.

Uurimise tulemusena peaks välja tulema SQL-andmebaasides nimekasutuse iseärasused ja esinevate probleemide olemasolu. Analüüs peaks kitsenduste puhul andma vastuseid järgmistele küsimustele.

- Kas ja milliseid mustreid on järgitud erinevate kitsenduste nimetamisel?
- Kas kitsenduste nimetamise stiil on järjekindel?
- Kas kitsenduste nimed viitavad objektidele, millega need on seotud?
- Kas kitsenduste nimed on automaatselt genereeritud?
- Kas kitsenduste nimed on selgelt mõistetavad ja kui palju kasutatakse lühendeid?
- Kui pikki kitsenduste nimesid kasutatakse?

Uuring ei ole piisavalt esinduslik, et teha järeldusi kõikide SQL-andmebaaside kohta.

3.2 Uuringu tulemused

Peale *Naming* testi käivitamist kõigil kolmel andmebaasil on kohe näha nende suuruste ja disainide erinevused, mis on välja toodud Tabel 1. Tabelis olevad arvud näitavad erinevat tüüpi andmebaasiobjektide hulka. Igal andmebaasiobjektil on nimi. Seega näitab andmebaasiobjektide arv ka uuritavate nimede arvu. Kui *erinevate* nimede arv on väiksem kui andmebaasiobjektide koguarv, siis esitan selle sulgudes. Näiteks erinevates tabelites võib olla sama nimega veerge või kitsendusi. OTRS andmebaas erineb ülejäänud kahest kõige rohkem: see on väiksem tabelite arvu poolest, kuid samas on selles kõige rohkem välisvõtme kitsendusi. Veel on üheks disaini erinevuseks selles andmebaasis trigerite, vaadete ja protseduuride puudumine. *Check* kitsendused olid olemas ainult LedgerSMB andmebaasis. Üheski uuritud andmebaasis ei olnud *exclude* kitsendusi, domeene ega hetktõmmiseid e materialiseeritud vaateid.

Tabel 1. Ülevaade uuritavatest andmebaasidest.

Objekt	FusionForge	LedgerSMB	OTRS
Skeemid	1	1	1
Tabelid (e baastabelid)	161	164	116
Vaated	36	4	0
Veerud (nii tabelites, vaadetes kui hetktõmmistes)	1265 (erinevaid nimesid 419)	989 (erinevaid nimesid 390)	963 (erinevaid nimesid 293)
Võtme kitsendused (PRIMARY KEY ja UNIQUE kitsendused)	127	217	138
Välisvõtme kitsendused	82 (erinevaid nimesid 74)	251	264
Tabelitega seotud *check* kitsendused	0	54	0
Arvujada generaatorid	97	80	80
Trigerid	12	12	0
Funktsioonid	8	16	0
Kokku andmebaasi-objekte	2035	2064	1827
Kokku erinevaid nimesid	1049	1221	1015
Erinevate nimede protsent nimede koguarvust	51.5%	59.2%	55.6%

3.2.1 Kitsenduste nimetamine

FusionForge rakenduse andmebaasis on loodud kõige vähem kitsendusi. Ma vaatasin koodist käsitsi järgi ning osad on nimetatud süsteemi poolt, teised kasutaja poolt sama mustri järgi. Testpäringu kohaselt on 63% vaadeldud andmebaasis olevates kitsendustest samasuguse nime mustriga nagu andmebaasisüsteem ise nimetamisel kasutab. Teises kahes on kitsendusi loodud rohkem: LedgerSMB andmebaasis on kitsendusi

kokku 522, millest 99% kitsendustest on süsteemi nimetatud ning OTRS baasis leidus 402 kitsendust, millest ainult 20% kitsendustest on süsteemi nimetatud. Täpsemad arvud on toodud välja Tabel 2. OTRS andmebaasis olid kõik süsteemi poolt nimetatud kitsendused ainult primaarvõtmed, teiste kitsenduste nimed olid kõik kasutaja defineeritud, mis on üldpõhimõttena hea viis, kuidas andmebaasi disainida. Kitsendusi ei tohiks nimetada segamini kasutaja ja süsteemi poolt, sest see pole järjekindel ja tekitab segadust. FusionForge'i andmebaasil on just see probleemiks – kõiki erinevaid kitsendusi on nii kasutaja poolt nimetatud, kui ka süsteemi poolt nimetatuid. LedgerSMB andmebaasil olid peaaegu kõik kitsendused süsteemi poolt nimetatud välja arvatud viis kitsendust, millele mingil põhjusel nimi kasutaja poolt anti. Nimed anti kasutaja poolt kahele *check*, ühele unikaalsuse kitsendusele, kahele välis- ja ühele primaarvõtme kitsendusele. Mingit seaduspärasust nende puhul tuua ei oska, sest nimesid anti vahel kohe tabeli loomisel kui ka hiljem tabeli muutmisel. Kasutaja loodud nimed on näiteks `invoice_allocation_constraint` (*check* kitsendus), `menu_node_parent_key` (unikaalsuse kitsendus).

Tabel 2. Kitsenduste nimetamine süsteemi poolt.

	FusionForge	LedgerSMB	OTRS
Kitsendusi kokku	209	522	402
Süsteemi poolt nimetatuid kokku ja nende osakaal	133 – 63%	517 – 99%	82 – 20%

Naming testis on päring, mis üritab tuvastada andmebaasist erinevaid kitsenduste nimetamise mustreid. Antud päringu tulemusi ei saa aga 100% tõeselt võtta, sest see vaatab ainult kitsendusi, millega on seotud üks veerg ning proovib sobitada erinevaid kitsendusega seotud olevaid tabelite ja veergude nimesid kitsenduse nimesse. Seetõttu võib antud päring mõningad mustrid natuke valesti kokku panna. Päringu järgi on kõige halvemini nimetatud FusionForge'i andmebaasi kitsendusi, sest leiti 62 erinevat mustrit. Käsitsi üle vaadates tuleb välja, et mustrite arv on seetõttu nii suur, sest kitsenduste nimedes on paljude tabelite nimedes sõnade eraldaja ära võetud. Näiteks tabeli nimi on `doc_data` ja seal oleva välisvõtme kitsenduse nimi on `docdata_stateid`. Sellest ka nii suur mustrite arv. Teises kahes andmebaasis on tunduvalt vähem mustreid: LedgerSMB-s 24 ja OTRS-s 8 kitsenduse nime mustrit. Populaarseimad mustrid on välja toodud Tabel 3, kus mustri taga sulgudes on arv, mitu korda seda esines. Kõikide andmebaaside kitsenduste nimed olid kirjutatud kasutades *snake case*'i.

Tabel 3. Kitsenduste nimede mustrid.

Andmebaasi-objekti tüüp	FusionForge
Välisvõtme kitsendus	{child_table}_{child_column}_fk (13) {child_table}_{child_column}_fkey (11) "$1" (6) "$2" (4)
Primaarvõtme ja unikaalsuse kitsendus	{table}_pkey (82) {table}_{column}_key (2)
	LedgerSMB
Check kitsendus	{table}_{column}_check (39) entity_{column}_{column}_check (2)
Välisvõtme kitsendus	{child_table}_{child_column}_fkey (239) payroll_pa{primary_column}_timeoff_{child_column}_fkey(3)
Primaarvõtme ja unikaalsuse kitsendus	{table}_pkey (82) {column}_pkey (3)
	OTRS
Välisvõtme kitsendus	fk_{child_table}_{child_column}_{primary_column} (259) fk_val{primary_column}_{child_column_{primary_column}(2)
Primaarvõtme ja unikaalsuse kitsendus	{table}_pkey (82) {table}_{column} (141)

Primaarvõtme kitsenduste nimetamine oli kõige järjekindlam OTRS ja FusionForge'i andmebaasides, sest seal olid kõik nimetatud mustriga `{tabeli_nimi}_pkey`. 102 primaarvõtme kitsendust on FusionForge'i andmebaasis kasutaja nimetatud. OTRS andmebaasis on kõik primaarvõtme kitsendused süsteemi nimetatud. LedgerSMB andmebaasis oli 95-st primaarvõtme kitsendusest automaatselt nimetatud 94. Kuna kõik kolm andmebaasi on loodud PostgreSQL andmebaasisüsteemis ning suur osa primaarvõtmetest on süsteemi nimetatud, siis läbiv primaarvõtme kitsenduste muster on neil sama.

Kuna testpäringus vaadatakse ainult ühe veeruga seonduvate kitsendustega, siis tuleb rohkem kui ühe veeruga seotud kitsenduste nimede mustreid vaadata andmebaasist käsitsi. FusionForge'i andmebaasis on rohkem kui ühe veeruga seotud üks välisvõtme kitsendus, üks unikaalsuse kitsendus ning 28 primaarvõtme kitsendus. Süsteemi poolt on lähtekoodist järgi vaadates nimetatud ainult osa primaarvõtme kitsendusi. Kasutaja poolt

nimetatud kitsendused on peale vaadates kõik siiski sama mustri järgi koostatud, nagu andmebaasisüsteem seda teeb. OTRS andmebaasis on tabeli mitme veeruga seotud ainult üheksa unikaalsuse kitsendust. Kõik nendest on kasutaja nimetatud jättes kitsenduse tüübi nimesse märkimata ning kõigest üheksa kitsenduse kohta on kasutusel kuus erinevat nime mustrit:

1. `{table_name}_{column_names}`
2. `dynamic_field_{column_name}` (osa tabeli nimest ja teine veerg puudu)
3. `{table_name}_view`
4. `{table_name}_list`
5. `{table_name}_{table_name}`
6. `{table_name}_per_user`

LedgerSMB andmebaasis on kokku 80 kitsendust, mis hõlmavad rohkem kui ühe veeru. Neist seitse on unikaalsuse kitsendused, kaheksa *check* kitsendused, viis välisvõtme kitsendused ning ülejäänud on primaarvõtme kitsendused. Ühele *check* kitsendusele andis nime kasutaja mustriga `{table_name}_allocation_constraint`. Lisaks andis kasutaja ühe nime unikaalsuse kitsendusele mustriga `{table_name}_{first_column_name}_key`. Kõikidele ülejäänud kitsendustele andis nimed andmebaasisüsteem.

Primaarvõtme kitsenduste kõrval peaksid ka teiste kitsenduste (välisvõtmed, unikaalsuse, *exclude* ja *check* kitsendused) nimedes olema nende tabelite ja veergude nimed, millega need seotud on. Sellele viitavad mitmed erinevad allikad, kus defineeritakse kitsenduste nimetamise häid praktikaid [18], [19], [27]. Ainult LedgerSMB tabelites olid kõikide teiste kitsenduste nimedes tabelite nimed olemas. Nii OTRS kui ka FusionForge'i andmebaasides esinesid kitsendused, kus seda polnud, vastavalt 10-s ja lausa 138-s kitsenduse nimes. Veergude nimed olid väga paljudel kitsendustel puudu kõigis kolmes andmebaasis.

Kitsenduste nimede pikkused on valdavalt sarnased. Enamus osa kõiki tüüpi kitsenduste nimedest läbi kolme andmebaasi oli pikkusega 7-10 tähemärki. Eranditeks oli OTRS andmebaasi välisvõtmed, kus keskmine nime pikkus oli 18-20 tähemärki, nt `fk_article_change_by_id`, `fk_queue_change_by_id`, ning FusionForge'i välivõtmed, kus ühed populaarsemad nimed olid kahe tähemärgi pikkused, nt `$1`, `$2`.

Kitsenduste nimetamist uurides tuleb tõdeda, et nende nimetamine ei ole läbi vaadeldud andmebaaside järjekindel. Kui oleks olemas laiendus, mis neid ühtlustaks, siis tõstaks see kindlasti andmebaasi skeemi kvaliteeti ja hallatavust. Uurimise käigus tuli idee lisada loodavasse laiendusse funktsionaalsus, mis laseb erinevat tüüpi kitsendusi eraldi ümber nimetada. See oleks vajalik juhul, kui mingil kindlal kitsenduse tüübil on stiilid ebaühtlased. Näiteks esineb selline olukord kui välisvõtme kitsendustest on osa nimetatud süsteemi ja osa kasutaja poolt, kusjuures kasutaja antud nimed erinevad märgatavalt süsteemi antud nimedest. Samuti läheks seda vaja juhul, kui kasutaja soovib teatud tüüpi kitsenduste (nt *check*) puhul säilitada enda antud nimed, et näiteks veateated oleksid sisukamad.

3.2.2 Üldised tähelepanekud nimekasutuse kohta

Peale kitsenduste muid andmebaasiobjekte (tabelid, veerud, vaated, funktsioonid) uurides, tulevad erinevates aspektides välja enam-vähem sarnased tähelepanekud kui kitsenduste nimede korral. Nimed on kõigis kolmes andmebaasis umbes ühesuguse pikkusega: suurimad nime pikkused on kõikides andmebaasides 30-50 tähemärki ning nii pikad nimed on ka kõige ebapopulaarsemad. Kuna PostgreSQL lühendab automaatselt kõik nimed, mis on pikemad kui 63 baiti, 63 baidi pikkuseks, siis sellest võib järeldada, et vaadeldud andmebaasides ühtegi andmebaasisüsteemi reeglite kohaselt liiga pikka nime ei kasutatud. Kõige rohkem esineb nimesid kõigis kolmes andmebaasis pikkusega 7-11 tähemärki, täpsemalt vt Tabel 4.

Tabel 4. Nimede pikkus tähemärkides ning nende esinemissagedus.

	FusionForge		LedgerSMB		OTRS	
	pikkus	sagedus	pikkus	sagedus	pikkus	sagedus
1.	8	178	11	146	11	215
2.	9	133	9	143	9	190
3.	7	123	8	143	8	118
4.	11	117	10	128	2	83
5.	10	108	7	126	10	70

Kõigis kolmes andmebaasis oli järjekindlalt andmebaasi-objekte nimetatud *Snake case* mustrit [37] kasutades, millest tulenevalt nimede alamosad on kõik kirjutatud väiketähtedega ning eraldajana kasutatakse alakriipsu (nt *lower_case_with_underscores*).

FusionForge rakenduse andmebaasis leidus teistsuguse mustriga nimesid 28 ning teises kahes andmebaasis oli selliseid nimesid mõlemas ainult neli. Kõik nimed, mis ei vastanud *snake case* mustrile, olid mingis osas ilma eraldajata kokku kirjutatud, nt `attachmentid`, `tbtypeid`, `artifactcannedresponses_groupid`.

Nimesid, mis koosnesid rohkem kui viiest sõnast, tuleks minu arvates võimalusel vältida, sest nii pikki nimesid on üsna raske jälgida ning selliselt nimetatud elemendi tagamõte võib jääda arusaamatuks. Lisaks on *Naming* testis vastav kontrollpäring ka olemas. Nimes sisalduvate sõnade arvult kõige pikemad nimed anti OTRS andmebaasis, kus 389 nime olid soovitatust pikemad ning kõige pikem nimi sisaldas kümme sõna. LedgerSMB oli üsna sarnane: 328 pika nimega, millest pikim oli üheksa sõna pikk. FusionForge'is oli üle poole vähem, 146 pikka nime ning kõige pikem oli ka ainult kaheksa sõna pikk (nt `artifact_extra_field_list_extra_field_id_seq`).

Numbreid kitsenduste nimedes esineb kõikide andmebaasides. Kõige vähem on neid LedgerSMB andmebaasis, kus numbrid on lisatud ainult neljale välisvõtme nimele, et vältida nimekonflikti. OTRS rakenduse andmebaasis sisaldavad 12 nime endas numbrit. Enamasti on samuti lisatud järjekorranumber sama nimega veerule, kuid kolmel kitsendusel on lõpus järelliide md5. Ilmselt viitab see, räsifunktsioonile. FusionForge'i andmebaasis sisaldavad 25 nime mingit numbrit, neist kümnel on kitsenduse nimeks eelnevalt välja toodud `$1` või `$2`. Kahes tabeli nimes on number, ühel nimekonflikti vältimiseks (`user_metric0`), teisel põhjust ei suutnud ma tuvastada (`project_metric_tmp1`). Seetõttu on nende tabelitega seotud kitsenduste nimedes samuti antud number. Lisaks esineb numbreid ka veerunimedes.

Hakkas silma sõna id rohkus nimede hulgas. See oli kõikides andmebaasides üks kõige rohkem kasutatud nimesid ning ühtlasi ka liiga lühike, et eraldi objekti nimi olla. Samuti oli see ka `pk, fk, pkey` kõrval kõige populaarsem nimekomponent võtmete ja teiste kitsenduste ees- või järelliidete. Kui kitsenduse või arvujada generaatori nimi on tuletatud veeru nimest, siis see on mõistetav.

Lisaks oli kõikides andmebaasides `id` liiga üldine primaarvõtmesse kuuluva veeru nimi: FusionForge'i puhul esines sellist primaarvõtme veeru nime 12 tabelis, OTRS andmebaasis aga koguni 79 tabelis. Täpsemalt on populaarseemad primaarvõtme veergude nimed välja toodud Tabel 5.

Tabel 5. Primaarvõtme veeru nime populaarseimad mustrid ja nende esinemissagedused.

	FusionForge	LedgerSMB	OTRS
1.	`{table}_id (14)`	`id (40)`	`id (79)`
2.	`id (12)`	`label (10)`	`object_id (1)`
3.	`month (4)`	`entry_id (7)`	`data_key (1)`
4.	`week (4)`	`class (5)`	`customer_id (1)`
5.	`ranking (4)`	`{table} (3)`	-

Primaarvõtmete veergude nimede muster oli kõige järjekindlam OTRS andmebaasis, kus 79 tabelil 82-st oli veeru nimeks lihtsalt `id`. See ei ole eriti hea praktika [38]. LedgerSMB andmebaasis oli samuti ülekaalus nimi `id`, mis on nimeks umbes pooltel primaarvõtme veergudel. Lisaks olid esindatud nimed `label, entry_id, class` ning tabeli nimi, kokku oli primaarvõtme veergude seas 28 erinevat nime mustrit. Kõige rohkem erinevaid mustreid ja kõige vähem järjekindlam oli FusionForge'i andmebaas 50 erineva mustriga. Kõige populaarsem oli üllatavalt vähe, 14 korda esinenud `{tabeli_nimi}_id`. Selles andmebaasis esines ka väga huvitavaid primaarvõtmete veergude nimesid nagu `week`, `ranking`, mis asusid tabelites, mille üks nime komponentidest oli vastavalt `weekly` või `metrics`. Enamustel veergudel oli nime lõpus siiski liide `_id`. Tabelite ühendamine toimub päringutes enamasti üle primaarvõtme ja välisvõtme veergude. Selleks, et primaarvõtme veeru nime saaks ilma pikema mõtlemata kasutada ja tänu sellele tabelite ühendamise päringuid kiiremini kirjutada, peaks see minu hinnangul sisaldama tabeli nime. Paraku aga kõigis kolmes vaadeldud andmebaasis on suur hulk tabeleid, 85-100%, kus primaarvõtme veeru nimi tabeli nime ei sisalda.

LedgerSMB andmebaasis kasutati andmebaasiobjektide nimedes kõige rohkem lühendeid, nii pikemates kitsenduste nimedes nime osadena kui ka veeru ja tabeli nimedes. Kuue tabeli ja 12 veeru nimed ei selgitanud objekti sisu ning nende puhul ei olnud nimedest võimalik aru saada, mis andmeid seal täpselt hoitakse. Mõned näited: veerud `ap, ar, gl, mfg_lot, oe, itu` ja tabelid `ap, ar, gl, oe`. OTRS andmebaasis paistis silma kaks lühendit, mis esinesid eraldi tabelite nimedena ning vastavalt ka nende tabelitega seotud kitsenduste nimedes: `acl` ja `sla`. FusionForge'i andmebaasis oli selle nurga alt vaadatuna objekte kõige paremini nimetatud, sest lühendeid kasutati suhteliselt vähe (paar üksikut lühendit, mis esinevad funktsioonide ja trigerite nimedes).

3.3 Järeldused ja kokkuvõtted kitsenduste nimetamise kohta

Kolme täiesti erinevat andmebaasi analüüsides selgus, et olles samaaegselt väga erinevad, tulevad nendes välja mitmed samad probleemid. Ootuspäraselt ei näinud ma nimede andmisel täielikku järjekindlust ning perfektsust.

Jaotises 3.1 esitati kitsenduste nimetamise kohta küsimused. Esitan nendele lühikesed ja kokkuvõtlikud vastused.

1. Kas ja milliseid mustreid on järgitud erinevate kitsenduste nimetamisel?
   Vastus: Oleneb andmebaasist. Ühel kolmest oli väga vähe erinevaid mustreid, kuid ühel leidus umbes 60 erinevat. Nimetamisviiside rohkus vähendab andmebaasi hallatavust.
2. Kas kitsenduste nimetamise stiil on järjekindel?
   Vastus: Ei ole. Järjekindlust on keeruline saavutada, kui kõik andmebaasi arendajad ühtsest stiilist kinni ei pea. Seda aitaks tagada dokumentatsioonis stiili ära määramine ning üksteise koodi üle vaatamine ehk *code review* [39]. Lisaks aitaks stiili ühtlustada just mingisugune ühtlustamise funktsiooniga rakendus/laiendus.
3. Kas kitsenduste nimed viitavad objektidele, millega need seotud on?
   Vastus: Enamasti jah – nimes oli märgitud tabel ning tihti ka veerg, millega antud kitsendus seotud oli. Kuid FusionForge'i andmebaasis esines mitmeid kordi kummaline kitsenduse nimi "$1".
4. Kas kitsenduste nimed on automaatselt genereeritud?
   Vastus: Jah ja ei. Kõikides andmebaasides esines automaatselt genereeritud nimesid. Olenevalt andmebaasist oli nende osakaal väga erinev, OTRS andmebaasis oli 20% kitsenduse nimedest süsteemi genereeritud, LedgerSMB-s aga lausa 99%.
5. Kas kitsenduste nimed on selgelt mõistetavad ja kui palju kasutatakse lühendeid?
   Vastus: Suures osas on nimed mõistetavad ning lühendeid kasutatakse enamasti kitsenduse tüübi ära märkimiseks kitsenduse nimes. Samas leidus ka erandeid, kus nii tabeli kui sellega seotud kitsenduste nimed on lühendatud ning rakenduse arendusega mitte tuttavana, ei ole võimalik tuvastada objekti tähendust.
6. Kui pikki kitsenduste nimesid kasutatakse?
   Vastus: Kõigis vaadeldud andmebaasides olid kõige populaarsemad nime pikkused 7-10 tähemärki. Väga pikki nimesid kasutati kõigis andmebaasides väga vähe. OTRS andmebaasis oli suurel hulgal kasutatud ka kahe tähemärgi pikkust nime *id*.

4 PostgreSQL laiendus andmebaasis kitsenduste nimede ühtlustamiseks

Selles peatükis projekteeritakse ja realiseeritakse PostgreSQL nimede ühtlustamise laiendust. Töö kirjutamise ajal (2018. aasta kevad) ei ole kättesaadav ükski PostgreSQL laiendus või mõni muu programm, mis andmebaasi objektide nimesid automaatselt ühtlustab. Otsisin nii Google'ist kui ka PostgreSQL Extension Networkist [40] kasutades järgnevaid otsingu fraase: *„postgresql constraint refactor tool", „constraint refactor tool", „postgresql refactor extension", „refactor extension", „postgresql naming extension", „naming extension", „postgresql naming convention extension"*. Saadaval on ainult erinevaid SQL lausete ning andmebaasi refaktoreerimise tarkvarasid nagu näiteks Instant SQL Formatter [41] ja Liquibase [42], kus kasutaja peab ise muutused defineerima, aga mitte ühtegi rakendust/laiendust pole juba olemasoleva andmebaasi automaatseks refaktoreerimiseks. Seega leian, et reaalne vajadus selle järgi on täiesti olemas.

4.1 PostgreSQL laienduste mehhanism

PostgreSQL andmebaasisüsteemis on kasulik võimalus seotud funktsioone, tabeleid ja muid andmebaasiobjekte kokku grupeerida üheks kogumiks, mida nimetatakse laienduseks (*extension*). Selline funktsionaalsus aitab hoida laienduse jaoks loodud andmebaasiobjektide loomise koodi eraldi teiste andmebaasiobjektide loomise lausetest ning see teeb andmebaasi haldamise lihtsamaks ja selgemaks [43]. Laienduse saab andmebaasi lisada ühe lausega (`CREATE EXTENSION`).

Laienduse realisatsioon peab sisaldama vähemalt kahte faili [43].

1. Laienduse kontrollfaili, mis kirjeldab laienduse konfiguratsiooni. Selle faili nimi peab olema `extension_name.control` ning see peab asuma kaustas `SHAREDIR/extension`. Kontrollfailis on defineeritud erinevad parameetrid koos vastavate väärtustega, üks parameeter rea kohta, samal põhimõttel nagu `postgresql.conf` failiski. Mõned kontrollfailis olevad parameetrid:

- `directory` (*string* tüüpi) – SQL skriptifaili asukoht,
- `comment` (*string* tüüpi) – vabatekstiline kommentaar laienduse kohta. Seda saab määrata ainult laiendust luues, sest hilisema uuendamise käigus võidakse see skriptiga üle kirjutada,
- `requires` (*string* tüüpi) – sõltuvate laienduste nimekiri, mis tuleb enne installeerida.
- `relocatable` (*boolean* tüüpi) – tõeväärtus, kas laienduse objekte saab ümber tõsta teistesse skeemidesse.

2. Laienduse moodustavate objektide andmebaasis loomiseks mõeldud SQL lauseid sisaldav skriptifail. See fail peab asuma samas kaustas, kus kontrollfail ning peab olema nimetatud `extension_name--version.sql`.

Luues andmebaasis laienduse lausega `CREATE EXTENSION extension_name`, käivitatakse skriptifailis olevad käsud ning andmebaasi tekivad kõik seal defineeritud andmebaasiobjektid. Loodud objektid pannakse kasutaja vaikimisi skeemi. `CREATE EXTENSION` lauses saab ka määrata skeemi, millesse laienduse objektid paigutada. Kõik laiendusega seotud objektid moodustavad andmebaasisüsteemi jaoks terviku ning need saab andmebaasist eemaldada lausega `DROP EXTENSION extension_name`. Laienduse kasutamise üks eelis lihtsalt üksikute skriptifailis olevaid käskude käima panemise ees ongi see, neid saab andmebaasist ühe käsuga eemaldada. Samuti piisab andmebaasist *pg_dump* programmi abil tõmmise tegemisel sellesse `CREATE EXTENSION` lause lisamisest, selle asemel, et lisada kõikide laienduse objektide loomise laused. Tänu sellele on lihtsam andmebaasi ühelt andmebaasisüsteemi versioonilt teisele migreerida [43].

4.2 PostgreSQL piirangud nimedele

Järgnev ülevaade põhineb PostgreSQL 10 dokumentatsioonil [17]. Andmebaasikeele järgi on objektide identifikaatoritel ja kasutatavatel võtmesõnadel (nt `SELECT`, `UPDATE`, `VALUES`) sama leksikaalne struktuur. Seega peab lugeja lause osadest arusaamiseks teadma, millised on keele võtmesõnad ja millist tüüpi lause osad (võtmesõnad, identifikaatorid) võivad mingis lause osas esineda.

PostgreSQL andmebaasisüsteem seab andmebaasiobjektide nimedele ja võtmesõnadele mitmeid piiranguid. Andmebaasikeele võtmesõnad ja objektide nimed peavad algama tähe või alakriipsuga. Tähed ei pea olema ladina tähed ning võivad olla ka diakriitilise

ehk rõhu märgiga. Järgnevateks märkideks võivad lisaks tähtedele olla numbrid ja ka dollarimärk. Samas SQL standardi järgi pole dollarimärk identifikaatori nimes lubatud. Seega oleks parem seda nimedes mitte kasutada, sest see võib andmebaasi ülekantavust raskendada. Meenutan, et peatükis 3. esitatud uuring leidis kolmes andmebaasis kokku 2 nime, mis sisaldasid dollarimärki (FusionForge'i andmebaasis esines $1 kuus korda ja $2 neli korda).

Maksimaalseks identifikaatori pikkuseks on PostgreSQL-is määratud 63 baiti. SQL lausetes on küll võimalik pikemaid nimesid sisestada, aga andmebaasisüsteem lühendab need automaatselt 63 baidini. Kasutaja määratud liiga pikk nimi lühendatakse lõpust jupi äralõikamise teel (st mitte eriti arukalt ja paindlikult). Seega tuleks lubatud pikkust silmas pidada ning koostada nimed, mis seda ei ületa. Andmebaasisüsteemi poolt määratud nimed (nt kitsendustele), mis vaikimisi mustri järgi genereerides tuleksid liiga pikad, lühendatakse natuke intelligentsemalt. Näiteks liiga pika välisvõtme korral jäetakse kitsenduse tüübi lühend *_fkey* alles ning nime lühendatakse muude komponentide arvelt (vt Joonis 10).

```
CREATE TABLE Tuba(
tuba_kood SERIAL NOT NULL PRIMARY KEY
);
CREATE TABLE
t123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890(
tuba_kood INTEGER REFERENCES Tuba(tuba_kood),
CONSTRAINT pk_t PRIMARY KEY (tuba_kood)
);
/*Välisvõtme kitsenduse nimi:
"t123456789012345678901234567890123456789012345678901234567_tuba_kood_fkey*/
```

Joonis 10. Automaatselt lühendatud välisvõtme kitsenduse nimi.

Erinevatel kitsenduste tüüpidel on ka erinevad skoobid, kus sama kitsenduse nimi esineda tohib. Ühes tabelis kahte sama tüüpi kitsendust sama nimega olla ei saa, kuid mitmes erinevas tabelis tohib olla sama nimega ainult `FOREIGN KEY` ja `CHECK` kitsendusi (vt Joonis 11).

```
CREATE TABLE Riik1(riigi_kood CHAR(3),
CONSTRAINT uq_riik UNIQUE(riigi_kood));

CREATE TABLE Riik2(riigi_kood CHAR(3),
CONSTRAINT uq_riik UNIQUE(riigi_kood));
/*ERROR: relation "uq_riik" already exists*/
```

Joonis 11. Samanimelisi unikaalsuse kitsendusi samas skeemis olla ei saa.

Primaarvõtme, unikaalsuse kitsenduse ja `EXCLUDE` kitsenduse toetuseks loob süsteem automaatselt indeksi, mille nimi on samasugune nagu kitsenduse nimi. Indeksi nimi peab olema skeemi piires unikaalne. Seega moodustavad primaarvõtme, unikaalsuse kitsenduste ja `EXCLUDE` kitsenduste nimed ühise nimeruumi ja peavad olema kogu skeemi piires unikaalsed. Samas andmebaasis, kuid erinevates skeemides, tohivad kõik kitsendust nimed korduda.

4.3 Kitsenduste nimetamise mustrid

Loodav laiendus toetab mitmeid erinevaid nimetamise mustreid e skeeme. Peale installeerimist on automaatselt olemas mõned mustrid, kuid kasutajal on võimalus ise sobivaid juurde defineerida. Kasutaja saab omale sobiva mustri ära kirjeldada, andes väärtuseid erinevatele parameetritele:

- millist sõnade eraldajat kasutatakse (valida saab ükskõik, mis ASCII tähemärgi [44], numbrimärgi või dollarimärgi või jätta see märkimata, mille tulemusena on sõnad kokku kirjutatud). Kui eraldajaks määratakse tühik, siis paneb laiendus loodavad nimed jutumärkidesse, st tegemist on tõstutundlike piiritletud identifikaatoritega,
- kas kitsenduse tüüpi esitav lühend on kasutusel ees- või järelliitena,
- kas kitsenduse lühend on pikem, nagu PostgreSQL kasutab (nt `pkey`, `fkey`) või lühem kahetäheline, nagu on väga populaarne arendajate seas (nt `pk`, `fk`).

Uut mustrit defineerides tuleb sellele anda unikaalne nimi, mille järgi neid pärast kasutamisel tuvastatakse. Uued mustrid saavad olla variatsioonid sellest, et kitsenduse nimes on tabeli nimi, millele järgnevad kitsenduse poolt samas tabelis hõlmatud veergude nimed.

Installeerimisjärgselt on kasutusvalmis kaks kõige rohkem silma jäänud mustrit. Nimekirjas esitatakse mustri nimi ja selle ülesehituse kirjeldus.

- *postgresql_default*:
  `{table_name}_{column_names}_{long_constraint_abbreviation}`,
- *snake_case_with_short_prefix*:
  `{short_constraint_abbreviation}_{table_name}_{column_names}`

Laiendus ei paku võimalust kasutada nimedes suurtähti. Põhjus on selles, et suurtähtede kasutamiseks peaks nimi olema piiritletud identifikaator. Piiritletud identifikaatorid on SQL andmebaasides jutumärkides ja need on tõstutundlikud. Piiritletud identifikaatori näide on "*PK_Isik_Isikukood*". Tõstutundlikud identifikaatorid ei ole kasutusmugavuse seisukohalt head, sest identifikaatori kasutaja peab oskama selle suur- ja väiketähtede osas täpselt kirja panna (vt Joonis 12). Teiste sõnadega, kõik laienduse loodavad identifikaatorid on tõstutundetud regulaarsed identifikaatorid. Sellise identifikaatori näide on *pk_isik_isikukood*. Kui näiteks soovitakse sellise identifikaatoriga kitsendust kustutada, siis võib näiteks kasutada nii nimekuju *pk_isik_isikukood* kui ka *PK_Isik_Isikukood*.

```
CREATE TABLE Isik(isikukood CHAR(11) NOT NULL,
CONSTRAINT "PK_Isik_Isikukood" PRIMARY KEY (isikukood));

ALTER TABLE Isik DROP CONSTRAINT "Pk_isik_isikukood";
/*ERROR: constraint "Pk_isik_isikukood" of relation "isik" does not exist*/

ALTER TABLE Isik DROP CONSTRAINT "PK_Isik_Isikukood";
--Õnnestus
```

Joonis 12. Kitsenduse nime tõstutundlikkus.

PostgreSQL-is on nime maksimaalne pikkus vaikimisi 63 baiti. Kuigi andmebaasisüsteemi ühe juhtparameetri väärtuse suurendamisega saab maksimaalset lubatud nime pikkust suurendada, genereerib laiendus koodi ülekantavuse huvides nimesid, mille maksimaalne pikkus on kuni 63 baiti. Kuna kitsenduste nimed koostatakse tabelite ja veergude nimesid arvesse võttes, siis vajadusel peab laiendus nimesid lühendama järgnevaid põhimõtteid silmas pidades [19].

1. Esmalt lühendatakse tabeli nimi selle akronüümiks. Näiteks tabeli nimega **t**12345678901234567890123456789_**0**12345678901234567890123456789012 akronüüm on **t_0**.
2. Kui mingi tabeli nime mingis kitsenduses lühendati akronüümiks, siis lühendatakse kõikide teiste sama tabeli kitsenduste nimedes tabeli nime samuti ja samal viisil.
3. Kui nimi on peale tabeli nime lühendamist ikka liiga pikk, siis lühendatakse kitsenduse nimes veeru nimesid nii palju kui vaja. Kõiki komponente lühendatakse sealjuures võrdselt ning komponente, mis on lühemad kui neli tähemärki selguse ja loetavuse pärast ei lühendata. Kui lühendatud komponendi lõppu peaks jääma enne sõnu eraldanud alakriips, siis see eemaldatakse.

4.4 Eesmärgid

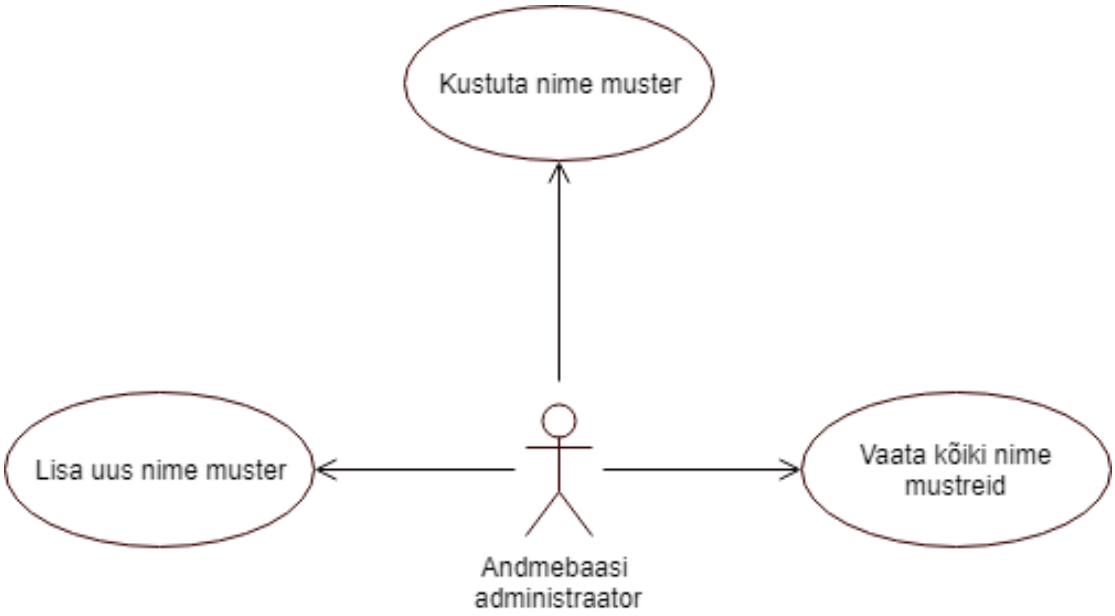
Võimaldada ühtlustada PostgreSQL andmebaasis loodud otse baastabelitega seotud `PRIMARY KEY`, `UNIQUE`, `FOREIGN KEY`, `CHECK` ja `EXCLUDE` kitsenduste nimesid ja suurendada selle kaudu andmebaasi realisatsiooni vastavust puhta koodi põhimõtetele.

Võimaldada viia ühtlustamist läbi väikese käskude hulgaga, st mitte iga kitsenduse nime ükshaaval muutes ja muuta sellega andmebaasi refaktoreerimist antud aspektis efektiivsemaks.

4.5 Tegutsejad

Mistahes arendaja, kellel on andmebaasis ülikasutaja õigused või kes on kõigi ümberimetatavate objektide omanik. Kasutusjuhtude mudelis nimetatakse sellist tegutsejat tinglikult andmebaasi administraatoriks.

4.6 Kasutusjuhtude mudel



Joonis 13. Laienduse kasutusjuhtude mudel, 1. osa.

Joonis 14. Laienduse kasutusjuhtude mudel, 2. osa.

Kõikide kasutusjuhtude kõrgtaseme formaadis kirjeldused on esitatud allpool.

Kasutusjuht: Lisa uus nime muster

Tegutseja: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator saab registreerida uue kitsenduste nime mustri, mida saab hiljem kitsendusi ümber nimetades kasutada.

Kasutusjuht: Kustuta nime muster

Tegutseja: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator saab mustri selle nime järgi kustutada.

Kasutusjuht: Vaata kõiki nime mustreid

Tegutseja: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator saab vaadata kõiki registreeritud nime mustreid, mille alusel on võimalik massilist kitsenduste ümbernimetamist läbi viia.

Kasutusjuht: Vaata kõiki kitsendusi

Tegutseja: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator saab vaadata kõiki andmebaasi mittesüsteemsetes skeemides olevate baastabelite `PRIMARY KEY`, `UNIQUE`, `FOREIGN KEY`, `CHECK ja EXCLUDE` kitsendusi ühes koondtabelis. Süsteemsed skeemid (*pg_catalog* ja *INFORMATION_SCHEMA*) sisaldavad süsteemikataloogi. Administraator saab valida, kas vaadata korraga kõiki nimesid või vaadata kindlat tüüpi kitsenduste nimesid. Süsteem esitab iga kitsenduse kohta skeemi nime, tabeli nime, hõlmatud veergude nimed ja kitsenduse nime. Kui administraator soovib näha kõiki erinevat tüüpi kitsendusi, siis esitatakse iga kitsenduse kohta lisaks selle tüüp. Süsteem esitab ainult baastabelitega otse seotud kitsenduste nimed, st ei too välja domeenide `CHECK` kitsenduste nimesid. Süsteem esitab ainult selliste kitsenduste nimed, mida saab ümber nimetada, st ei esita pärimise kaudu loodud tabelitesse ülemtabeli alusel defineeritud `CHECK` kitsendusi.

Kasutusjuht: Vaata kitsendusi valitud skeemis

Tegutseja: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator saab vaadata valitud mittesüsteemses skeemis olevate baastabelite `PRIMARY KEY`, `UNIQUE`, `FOREIGN KEY`, `CHECK ja EXCLUDE` kitsendusi ühes koondtabelis. Süsteemsed skeemid (*pg_catalog* ja *INFORMATION_SCHEMA*) sisaldavad süsteemikataloogi. Administraator saab valida, kas vaadata korraga kõiki nimesid või vaadata kindlat tüüpi kitsenduste nimesid. Süsteem esitab iga kitsenduse kohta skeemi nime, tabeli nime, hõlmatud veergude nimed ja kitsenduse nime. Kui administraator soovib näha kõiki erinevat tüüpi kitsendusi, siis esitatakse iga kitsenduse kohta lisaks selle tüüp. Süsteem esitab ainult baastabelitega otse seotud kitsenduste nimed, st ei too välja domeenide `CHECK` kitsenduste nimesid. Süsteem esitab ainult selliste kitsenduste nimed, mida saab ümber nimetada, st ei esita pärimise kaudu loodud tabelitesse ülemtabeli alusel defineeritud `CHECK` kitsendusi.

Kasutusjuht: Nimeta ümber kõik võimalikud kitsendused valitud skeemis

Tegutseja: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada soovitud skeemis olevad kõik baastabelite `PRIMARY KEY`, `UNIQUE`, `FOREIGN KEY`, `CHECK ja EXCLUDE` kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi ja pärimise kaudu loodud `CHECK` kitsendusi (vt kasutusjuht *Vaata kõiki kitsendusi*). Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik primaarvõtme kitsendused valitud skeemis
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada soovitud skeemis olevad kõik primaarvõtme kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik unikaalsuse kitsendused valitud skeemis
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada soovitud skeemis olevad kõik unikaalsuse kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik välisvõtme kitsendused valitud skeemis
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada soovitud skeemis olevad kõik välisvõtme kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik *exclude* kitsendused valitud skeemis
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada valitud skeemis olevad kõik *exclude* kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik võimalikud *check* kitsendused valitud skeemis
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada soovitud skeemis olevad kõik baastabelitega otse seotud *check* kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Muuta ei ole võimalik domeenidega seotud kitsendusi, sest sama domeeni võidakse kasutada erinevates tabelites ja veergudes või ka mitte üheski tabelis. Kuna uued nimed arvestavad tabeli ja veergude nimedega, siis see ei sobi domeenide kitsenduste nimetamiseks. Muuta ei ole võimalik pärimise tõttu alamtabelis loodud kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik võimalikud kitsendused kõigis skeemides
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga korraga ümber nimetada kõigis skeemides olevad kõik baastabelite `PRIMARY KEY`, `UNIQUE`, `FOREIGN KEY`, `CHECK ja EXCLUDE` kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi ja pärimise kaudu loodud `CHECK` kitsendusi (vt kasutusjuht *Vaata kõiki kitsendusi*). Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik primaarvõtme kitsendused kõigis skeemides
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada kõigis skeemides olevad kõik primaarvõtme kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik unikaalsuse kitsendused kõigis skeemides
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada kõigis skeemides olevad kõik unikaalsuse kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik välisvõtme kitsendused kõigis skeemides
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada kõigis skeemides olevad kõik välisvõtme kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik *exclude* kitsendused kõigis skeemides
Tegutseja: Andmebaasi administraator
Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada kõigis skeemides olevad kõik *exclude* kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

Kasutusjuht: Nimeta ümber kõik võimalikud *check* kitsendused kõigis skeemides
Tegutseja: Andmebaasi administraator

Kirjeldus: Andmebaasi administraator saab ühe käsuga ümber nimetada kõigis skeemides olevad kõik *check* kitsendused. Muuta ei ole võimalik süsteemikataloogi skeemide kitsendusi. Muuta ei ole võimalik domeenidega seotud kitsendusi, sest sama domeeni võidakse kasutada erinevates tabelites ja veergudes või ka mitte üheski tabelis. Kuna uued nimed arvestavad tabeli ja veergude nimedega, siis see ei sobi domeenide kitsenduste nimetamiseks. Muuta ei ole võimalik pärimise tõttu alamtabelis loodud kitsendusi. Käsus antakse ette muster, mille alusel muutmine läbi viiakse.

4.7 Mittefunktsionaalsed nõuded

Järgnevalt on välja toodud loodava laienduse mittefunktsionaalsed nõuded.

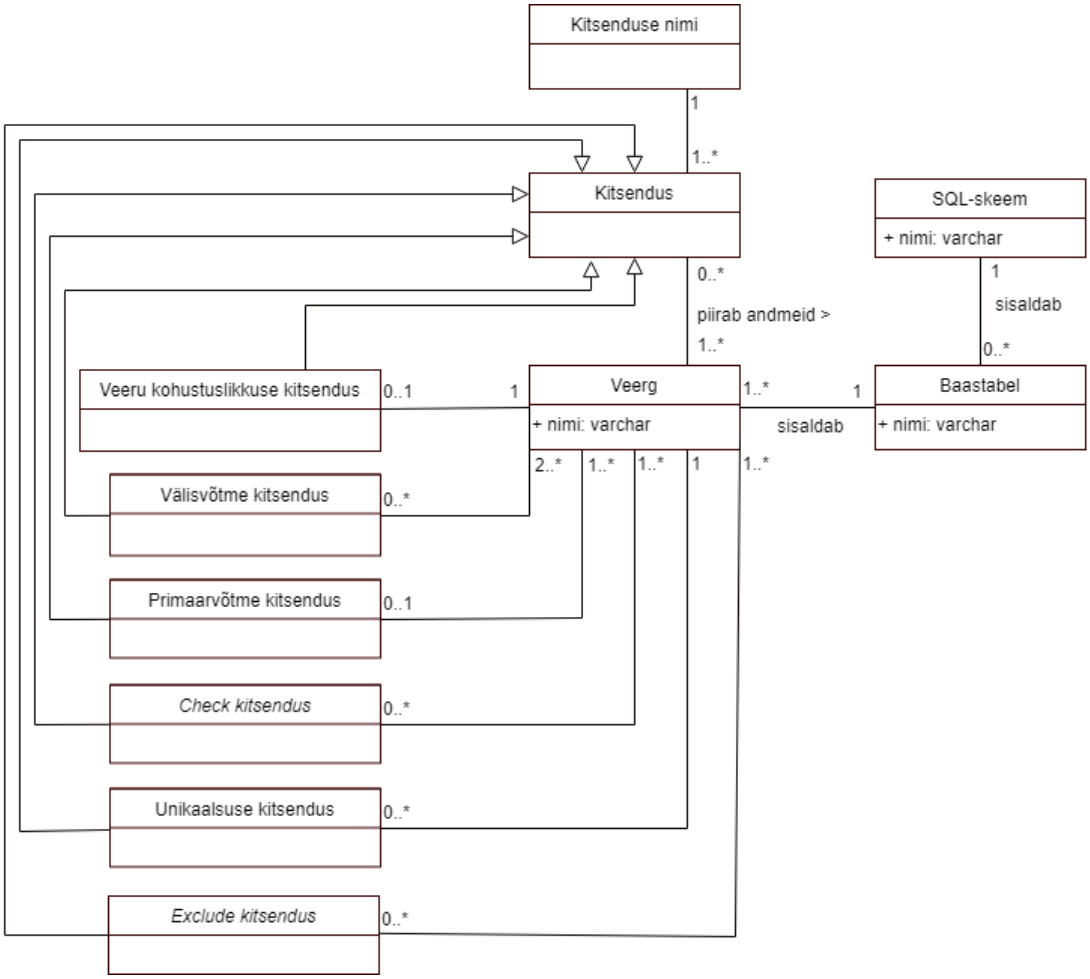
- **Avalikult kättesaadav.** Loodav PostgreSQL laiendus on peale valmimist laetud üles *PostgreSQL Extension Network* [40] keskkonda ning seetõttu peab see olema avatud lähtekoodiga ning kõigile vabalt kättesaadav.
- **Publitseeritud MIT Litsentsiga.** Laiendus on loodud MIT litsentsiga, sest see on kõige lihtsam kasutusel olev litsents teiste vaba tarkvara litsentside kõrval [45]. MIT litsents lubab sellest teada andmata loodavat tarkvara kasutada kõigil koos kõikide enda tehtud muudatuste ning kasutustega. Lisaks ei eelda antud litsents tarkvara loojalt mingit vastutust ega garantiid [46]. Kokkuvõttes annab see laienduse kasutajatele võimaluse laiendust oma soovide järgi muuta ning enne kasutamist koodi ja kogu funktsionaalsusega tutvuda.
- **Sõltumatus.** Loodav laiendus ei vaja eelnevalt mingi välise sõltuva komponendi installeerimist. Muuhulgas tähendab see, et andmebaasi skeemis muudatusi tegevad funktsioonid realiseeritakse PL/pgSQL protseduurses keeles, sest alates PostgreSQL 9.0 on see andmebaasis vaikimisi installeeritud [47].
- **Konfiguratsiooni tabelite sisu otsene muutmine andmekäitluskeele lauseid kasutamata.** Laiendusega seotud abitabelite sisu saab administraator muuta kasutades laienduse erinevaid funktsioone, andmekäitluskeele lauseid otse käivitamata.
- **SQL-süstimise vältimine.** Laienduses olevad funktsioonid koostavad dünaamiliselt SQL lauseid ja käivitavad neid. Laiendus peab olema realiseeritud viisil, et funktsioonide kasutaja ei saaks argumente kavalalt valides või pahatahtlikult

tabeleid, veerge või kitsendusi nimetades sundida andmebaasisüsteemi täitma mõnda muud SQL lauset, kui laienduse looja on ette näinud.

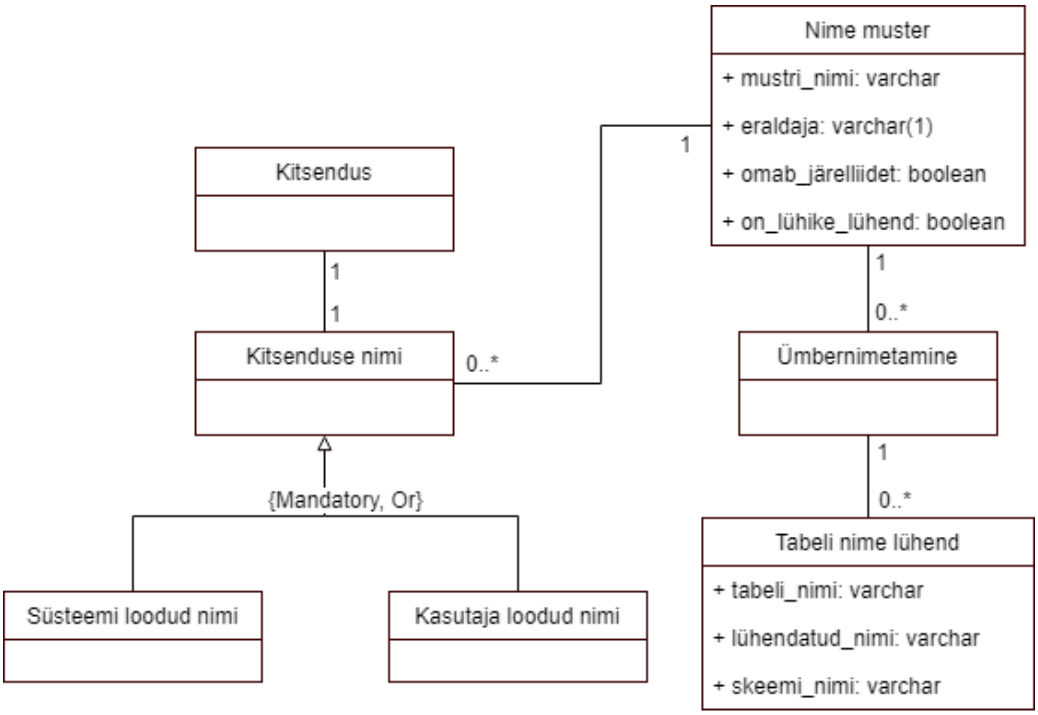
- **Toetab PostgreSQL 9.2+.** Laiendus toetab PostgreSQL andmebaasisüsteemi alates versioonist 9.2. Varasematel versioonidel see ei tööta, sest kitsenduste ümbernimetamine lausega `ALTER TABLE ... RENAME CONSTRAINT ...` on kasutusel alates 9.2 versioonist.

4.8 Valdkonnamudel

Valdkonnamudel on esitatud Joonis 15 ja Joonis 16 ning nendel kujutatud klasside kirjeldused on välja toodud Tabel 6. Klasside atribuutide kirjeldused on välja toodud Tabel 7.



Joonis 15. Laienduse valdkonnamudel, 1. osa.



Joonis 16. Laienduse valdkonnamudel, 2. osa.

Tabel 6. Valdkonnamudeli objektide kirjeldused.

Objekt	Kirjeldus
Baastabel	Baastabel e tabel on andmebaasis `CREATE TABLE` SQL lausega loodud nimega tabel. Tegemist on tabeliga, mida ei ole defineeritud teiste tabelite põhjal.
Check kitsendus	*Check* kitsendus võimaldab määrata piirangu (loogikaavaldise) ühe ja sama tabeli ühte või mitmesse veergu lisatavatele andmetele. Lisatavad andmed peavad rahuldama kitsendusega ära määratud tingimust või peab nende korral olema tingimuse kehtivus määramata, sest vähemalt osa nendest andmetest puudub (on NULL).
Exclude kitsendus	Jõustab reegli, et kui sama tabeli ridade paari võrrelda mingi avaldise alusel, siis juhul kui kontrolli tulemus on tõene, ei saa ühte nendest tabelis olla. Tegemist on unikaalsuse kitsenduse üldistusega, sest unikaalsuse kitsenduse korral toimub ridade teatud väljades olevate väärtuste võrdlemine võrduse kontrolli operaatori alusel. Exclude kitsenduse korral võib see operaator olla teistsugune. Exclude kitsendus on PostgreSQL omapära, SQL standard (SQL:2016) seda ei toeta.

Objekt	Kirjeldus
Kasutaja loodud nimi	Kasutaja poolt SQL andmekirjelduskeele lauses ära defineeritud kitsenduse nimi.
Kitsendus	Kitsendused andmebaasides on reeglid, mis kehtivad nendes olevatele andmetele. Iga SQL-andmebaasi kitsendus kehtib ühe või mitme tabeli ühe või mitme veeru kohta. Iga kitsendus määrab, millised andmed võivad sellega seotud veergudes paikneda. Erinevateks PostgreSQL-is toetatud kitsendusetüüpideks on `PRIMARY KEY`, `FOREIGN KEY`, `NOT NULL`, `UNIQUE`, `CHECK` ja `EXCLUDE` kitsendused. Kui mingi andmebaasis tehtud andmemuudatus rikub kitsenduses seatud reeglit, siis muudatus katkestatakse ning andmebaasisüsteemi poolt tagastatakse veateade.
Kitsenduse nimi	Igal andmebaasis loodud kitsendusel on nimi, mille järgi saab kitsendust kustutada ning sõltuvalt kitsenduse tüübist ka muuta. Andmebaasisüsteem võib selle nime esitada kitsenduse reegli rikkumise kuvatavas veateates. Nimi saab olla kasutaja poolt defineeritud, selle tegemata jätmisel nimetab kitsenduse andmebaasisüsteem.
Nime muster (andmebaasis *pattern*)	Nime muster on muster, mille järgi on nimi kokku pandud. Antud kontekstis määrab see kitsenduste nimede üldkuju. Iga muster paneb paika nimede püsiva osa ja muutuva osa. Püsivaks on, et igas mustri alusel loodud nimes viidatakse tabelile milles kitsendus asub, kitsenduse hõlmatud veergudele ja kitsenduse tüübile. Muutuv osa võimaldab määrata nimes kasutatavad eraldajad ja kitsenduse tüübile viitamise viisi.
Primaarvõtme kitsendus	Primaatvõtme väärtust kasutatakse rea unikaalseks identifitseerimiseks. Primaarvõti on veergude hulk, kuhu kuulub üks või rohkem veergu. Primaarvõtme kitsendus kohustab andmebaasisüsteemi tagama, et hõlmatud veergudes on iga rea kohta unikaalne väärtuste kombinatsioon ning igas reas on nendele veergudele vastavates väljades väärtus. Iga tabeli kohta on lubatud ainult üks primaarvõti. Tabelis võib olla mitu

Objekt	Kirjeldus
	kandidaatvõtit, millest üks valitakse tavalise disaini praktika kohaselt primaarvõtmeks ja ülejäänuid hakatakse kutsuma alternatiivvõtmeteks.
SQL-skeem	Andmebaasi skeem on andmebaasi skeemiobjektide kogum, mis on ühtlasi nende objektide suhtes nimeruum.
Süsteemi loodud kitsenduse nimi	Kitsenduse identifikaator. Kui kasutaja jätab kitsenduse loomisel sellele nime andmata, siis andmebaasisüsteem annab sellele ise nime. Erinevad süsteemid nimetavad kitsendusi erinevate mustrite järgi.
Tabeli nime lühend (andmebaasis *abbreviated_table*)	Ümbernimetamise käigus koostatud tabeli nime lühend, mida süsteem peab meeles, et kasutada seda ka teistes sama tabeli kitsendustes. See aitab tagada nimede järjekindlust.
Unikaalsuse kitsendus	Iga unikaalsuse kitsendus hõlmab tabeli ühte või mitut veergu. Unikaalsuse kitsendus kohustab andmebaasisüsteemi tagama, et hõlmatud veergudes on iga rea kohta unikaalne väärtus (või väärtuste kombinatsioon mitut veergu hõlmava kitsenduse korral). Iga tabeli kohta on lubatud null või rohkem unikaalsuse kitsendust. Koos kohustuslikkuse kitsendusega saab unikaalsuse kitsendust kasutada tabelis alternatiivvõtmete deklareerimiseks.
Veerg	Veerg on osa tabeli struktuurist. Iga veerg vastab mingile parameetrile, mis on osa tabeli päise poolt esitatud üldistatud väitest reaalse maailma kohta e predikaadist. Tabeli väärtuses on veergudes selle tüübile ja muudele tabeliga seotud kitsendustele vastavad väärtused, mis aitavad moodustada päisele vastavaid tõeseid väiteid reaalse maailma olemite kohta. SQL-andmebaasis on igas tabelis üks või rohkem veergu.
Veeru kohustuslikkuse kitsendus	SQL tabelites on vaikimisi kõik veerud, v.a veerud mis kuuluvad primaarvõtmesse, mittekohustuslikud, st nendele veergudele vastavates ridade väljades võib väärtus puududa (seal on selle tähistuseks NULL marker). Veeru kohustuslikkuse kitsendus jõustab reegli, et igas tabeli reas peab veerule vastavas väljas

Objekt	Kirjeldus
	olema väärtus. Veeru kohustuslikkuse kitsendusele annab üldjuhul nime andmebaasisüsteem ning ka käesolev laiendus ei võimalda nende kitsenduste nimesid muuta.
Välisvõtme kitsendus	Välisvõti on veergude hulk, milles olevatele väärtustele peab leiduma vaste mõne tabeli kandidaatvõtme väärtuste hulgas (võib olla sama tabel, mis välisvõtit sisaldab). Seda reeglit nimetatakse viidete terviklikkuse reegliks. Välisvõtme kitsendus kohustab andmebaasisüsteemi välisvõtmesse kuuluvate veergude korral jõustama viidete terviklikkuse reegli.
Ümbernimetamine	Ühte või kõiki tüüpi kitsenduste ümbernimetamine ühes või kõikides skeemides. Kasutajal pole piirangut ümbernimetamiste hulgale.

Tabel 7. Valdkonnamudeli objektide atribuutide kirjeldused.

Klass	Atribuut	Kirjeldus
Baastabel	nimi	Baastabeli nimi.
SQL-skeem	nimi	SQL-skeemi nimi.
Veerg	nimi	Veeru nimi.
Nime muster	mustri_nimi (andmebaasis *pattern_name*)	Nimi, mille kasutaja uut mustrit lisades sisestab. Hiljem mustrit kasutades identifitseeritakse muster tabelist just nime järgi. Seetõttu peab see tabelis olema unikaalne.
Nime muster	eraldaja (andmebaasis *delimiter*)	Märk, mille kasutaja määrab nimes sõnade (nime alamosade) eraldajaks.
Nime muster	on_järelliide (andmebaasis *has_suffix*)	Tõeväärtus selle kohta, kas mustri järgi loodavas nimes kitsenduse tüübi lühend on ees- või järelliitena. Eesliite korral on väärtuseks *false*.
Nime muster	on_lühike_lühend (andmebaasis *has_short_abbrevation*)	Tõeväärtus, kas mustri järgi loodavas nimes on kitsenduse tüübi lühend pikk (`pkey, fkey, key, check, exclude`) (kui väärtus on FALSE) või lühike (`pk,`

Klass	Atribuut	Kirjeldus
		`fk, uq, ck, ex`) (kui väärtus on TRUE).
Tabeli nime lühend	skeemi_nimi (andmebaasis *schema_name*)	Skeemi nimi, milles tabel asub.
Tabeli nime lühend	tabeli_nimi (andmebaasis *table_name*)	Tabeli algne nimi.
Tabeli nime lühend	Lühendatud_nimi (andmebaasis *abbreviated_name*)	Lühendatud tabeli nimi.

4.9 Kitsenduste ümbernimetamise võimalused PostgreSQL-is

Kitsenduste ümbernimetamiseks pakub PostgreSQL mitmeid võimalusi. Alates PostgreSQL versioonist 9.2 saab kitsendusi muuta andmekirjelduskeele lausega `ALTER TABLE {table_name} RENAME CONSTRAINT {old_name} TO {new_name}`. Eelmiste versioonidega tuli kitsenduse nime muutmiseks olemasolev kitsendus kustutada ning uuesti defineerida kasutades uut nime. Sellist moodust saab muidugi kasutada ka uuemate versioonidega. See ei ole aga eriti otstarbekas, sest nt `ALTER TABLE` lausega *check* kitsenduse ümbernimetamiseks pole tarvis kirja panna kitsenduse loogikaavaldist, vaid ainult selle nime. Kitsenduse loomisel kontrollib andmebaasisüsteem, kas kitsendusega kaetud andmebaasis juba olemasolevad andmed rahuldavad kitsendust või mitte. Kui ei rahulda, siis kitsendust luua ei õnnestu. Kitsenduse kustutamisel ja uuesti loomisel peab andmebaasisüsteem selle kontrolli uuesti läbi tegema (mis suure andmehulga korral on kindlasti kulukas), samas kui ümbernimetamise korral ei ole seda vaja teha. Kitsenduste, mille toetamiseks loob andmebaasisüsteem automaatselt indeksi (`PRIMARY KEY`, `UNIQUE` ja `EXCLUDE`), kustutamine ja uuesti loomine tähendab ka indeksi kustutamist ja uuesti loomist. See on kulukas tegevus. `PRIMARY KEY`, `UNIQUE` või `EXCLUDE` kitsenduse nime muutmisel muudab andmebaasisüsteem automaatselt ka seda toetava indeksi nime.

Primaarvõtme, unikaalsuse ja ka *exclude* kitsendusel on olemas neid toetav kitsendusega samanimeline indeks. Seetõttu on nendel kahel kitsenduse tüüpidel lisaks veel kolmas viis, kuidas nime muuta. Seda saab teha käsuga `ALTER INDEX {old_name} RENAME TO {new_name}`. Kuna indeksid kuuluvad andmebaasi siseskeemi, kuid tabelite kitsendused

kontseptuaalsesse skeemi, siis oleks koodi ülekantavuse ja arusaadavuse mõttes parem, kui kontseptuaalse skeemi arendaja töötaks kitsendustega, mitte indeksitega.

Kitsenduste ja neid toetavate indeksite nimede muutmine on teiste andmebaasiobjektide nimede muutmisest selles mõttes lihtsam, et andmekäitluskeele lauseid ja rakendusi ei tule seetõttu ümber kirjutada. Andmekäitluskeele lausetes ja rakendustes ei viidata otse kitsendustele ja indeksitele. See on oluline erinevus võrreldes näiteks tabelite või veergude nimede muutmisega, mis nõuab mõjuanalüüsi ja tingib vajaduse muuta andmekäitluskeele lauseid, rakendusi, teste.

Antud töö raames laienduse funktsionaalsuses kasutan kitsenduste nime muutmiseks kõige esimest viisi, sest muuta peab olema võimalik kõiki kitsendusi (v.a. `NOT NULL`) ning iga tüübi jaoks eraldi funktsioon eri viisiga luua ei ole otstarbekas, kui seda saab ühte moodi teha.

4.10 Realisatsiooni detailid

Baastabelite kitsenduste nimede ühtlustamiseks loodud PostgreSQL laiendus sisaldab endas kahte tabelit, ühte kohandatud andmetüüpi ning 31 funktsiooni, millest 11 on kasutajale mõeldud (moodustavad laienduse avaliku liidese) (täpsemaid funktsioonide kirjeldusi vt Lisa 2. Tabel 9) ning ülejäänud on laienduse sisesed funktsioonid, mida kasutavad teised funktsioonid (täpsemaid funktsioonide kirjeldusi vt Lisa 2. Tabel 10). Kahjuks ei ole PostgreSQL laiendust luues võimalik sisemisi funktsioone privaatseteks märkida, et keelata kasutajal neid ise välja kutsumast, seega saab tegelikkuses neid ka välja kutsuda. Parema hallatavuse huvides võib kasutaja luua laiendusega määratud objektid selleks otstarbeks loodud eraldi skeemis (vt Joonis 17).

```
CREATE SCHEMA uniform;
CREATE EXTENSION constr_name_unif WITH SCHEMA uniform;
SELECT * FROM uniform.get_all_constraints();
```

Joonis 17. Laienduse loomine eraldi skeemis.

Locmetric [48] programmi kohaselt on laienduses füüsiliste koodiridade arv ilma tühjasid ridu arvestamata kokku 629.

22 funktsiooni on kirjutatud PL/pgSQL keeles ja üheksa funktsiooni SQL keeles. SQL keeles on kirjutatud ainult `SELECT` lausega tabelit tagastavad funktsioonid, nimest

numbrite eemaldamise ja nime pikkuse küsimise funktsioonid. Kõik ülejäänud keerukamad funktsioonid on PL/pgSQL keeles. Kõik kasutajatele andmete vaatamiseks mõeldud funktsioonid on tabelifunktsioonid. Iga tabelifunktsiooni väljakutse tulemuseks on hulk ühesuguse struktuuriga ridu, mis moodustavad tabeli.

Iga funktsiooni loomise SQL lause salvestasin eraldi faili, mille nimes on funktsiooni nimi ja nime alguses lisaks järjekorranumber (nt *01_add_pattern.sql*). Kuna laienduse andmebaasi lisamiseks peab kogu vajaminev SQL kood olema ühes failis, siis kasutan ühe SQL faili genereerimiseks Makefile'i [49]. Kasutasin sellist lähenemist koodi parema hallatavuse pärast: ühte pikka faili on keerulisem edasi arendada ja parandada, kui mitut väiksemat faili. . Paljude failide seast saab vajaliku funktsiooni leida kiiresti üles faili nime järgi. Kuna osad funktsioonid sõltuvad üksteisest, siis aitab järjekorranumber faili nimes laienduse installeerimiseks vajaliku SQL skripti kokkupanemisel tagada, et laienduse loomisel luuakse funktsioonid õiges järjekorras.

Kuna laienduse võimalik kasutajaskond on kõikjal maailmas, siis kasutan koodi elementide nimedes inglise keelt. Funktsioonidele andsin nimed vastavalt nende tegevusele: kõik funktsioonid, mis midagi tagasi annavad, algavad sõnaga *get* (ühe sellise funktsiooni lähtekoodi vaata Lisa 3. Joonis 22) ja kõik, mis näiteks nimesid ümber nimetavad algavad sõnaga *rename* (ühe sellise funktsiooni lähtekoodi vaata Lisa 3. Joonis 23). Andsin funktsioonides kõikidele parameetritele nimed, et parameetritele oleks mugavam funktsioonide kehandites viidata ja soovi korral saaks parameetritele ka funktsiooni väljakutses nimeliselt viidata. Kõik laienduses olevad funktsioonide, parameetrite ja muutujate nimed on kirjutatud *snake case*'is. Enamasti vältisin nimede lühendamist, et objekti sisu oleks tulevasele kasutajale selge. Kahjuks siiski ilma lühendamata läbi ei saanud. Kuna enamus funktsioonidest tegeleb tabelitest andmete otsimisega, siis tihti pidin funktsiooni parameetrite nimesid vigade vältimiseks ja selguse tagamiseks lühendama. Näiteks kui algsed mõeldud parameetrite nimed olid `table_name` või `constraint_name`, siis lühendasin need vastavalt `table_n` ja `constraint_n`. Probleem tekkis sellest, et kasutatav tabel sisaldas sellise nimega veerge ning funktsiooni parameetrit ei saanud kasutada päringu tingimises (tekkis näiteks lause `SELECT … WHERE table_name = table_name`).

Saamaks andmeid baastabelite ja kitsenduste kohta teeb laiendus päringuid nii andmebaasi süsteemikataloogile loodud vaadete põhjal (vaated skeemis

INFORMATION_SCHEMA) kui ka süsteemikataloogi baastabelite põhjal (tabelid skeemis pg_catalog)

PostgreSQL-is saaks kõik kitsenduste ümbernimetamise laused koondada kokku üheks transaktsiooniks, mis tähendab, et kas tehakse kõik ümbernimetamised või jäetakse kõik tegemata. Antud laienduse realisatsioonis seda siiski ei tehta, mis tähendab, et kui mõne kitsenduse ümbernimetamine mingil põhjusel ebaõnnestub, siis tehakse ümbernimetamine osaliselt.

Koodi puhtust silmas pidades vältisin koodi kordamist erinevates funktsioonides, defineerides mitmel juhul korratavad koodiread eraldi funktsioonina, mida teised funktsioonid saavad välja kutsuda. Näiteks abifunktsiooni `get_pattern(name TEXT)` kustub välja teine sisemine abifunktsioon `get_new_name(type TEXT, pattern_name TEXT, old_name TEXT, schema_n TEXT, table_n TEXT, column_names TEXT, foreign_table_n TEXT DEFAULT NULL, is_table_name_abbreviated BOOLEAN DEFAULT FALSE)`. Funktsioon `get_pattern` tagastab seda kutsuvale funktsioonile küsitud mustri. Lisaks kasutasin kasutajamugavuse tagamiseks mitme erineva nimega funktsiooni defineerimise asemel funktsioonide ülelaadimist e *overloading*'ut. Funktsiooni ülelaadimine on sama nime, kuid erinevate sisendparameetritega funktsiooni loomine. Näiteks on laienduses kasutatavad kaks funktsiooni:

- `rename_all_constraints(pattern_name TEXT)`
- `rename_all_constraints(pattern_name TEXT, type TEXT)`

Esimene nimetab ümber kõik kitsendused etteantud mustri järgi ning teine ainult etteantud tüüpi kitsendused.

Pidasin PostgreSQL-is kehtivat nimede skoopi silmas ka laienduse arenduses. Kergelt võib tekkida olukord, kus laiendus genereerib kahele kitsendusele samas tabelis sama nime. Seega pidin laiendusse lisama iga nime genereerimise juurde selle nime skoobis esinemise kontrolli. Vajadusel leiab see kitsendusele alternatiivse nime. Selline olukord võib näiteks tekkida *check* kitsenduse korral, kui ühele tabeli veeru väärtusele seatakse kaks kontrolli (nt üks *check* kitsendus kontrollib `kohtade_arv>0` ja teine `kohtade_arv<100`). Nendele kahele kitsendusele looks laiendus vaikimisi sama nime, sest nad on samas tabelis ja sama veeru kohta. Konflikti vältimiseks lisab laiendus nimele juurde täisarvu, mis näitab, kui mitmes sama nimega kitsendus see nimetamise järjekorras

on. Kui peaks tekkima olukord, kus mitmel sama tabeli kitsendusel tekiks sama nimi, siis suudab laiendus olukorraga ilma nimekonfliktita hakkama saada.

Loodud laiendus on kaitstud ka SQL-süstimise (*SQL injection*) rünnaku eest. Enamus loodud funktsioone sisaldavad endas staatilisi SQL lauseid. Staatilise lause puhul on käivitatav lause andmebaasisüsteemile teada juba kompileerimise ajal ning on SQL-süstimine ei ole võimalik. Ühes funktsioonis on kasutuses aga dünaamiline SQL lause, mille puhul käivitatav lause selgub alles seda käivitades ning seetõttu teeb SQL-süstimise võimalikuks. Võimaliku rünnaku vältimiseks kasutasin antud lauses PostgreSQL-i funktsiooni `quote_ident(string)`, mis vajadusel (nt sõnes tühikute olemasolul) lisab kasutaja poolt ette antud sõnele jutumärgid ümber.

Laiendust arendasin vahendis JetBrains IntelliJ IDEA Ultimate 2017 [50] ning kasutasin vahendi poolt pakutavat koodi automaatset ümber kujundamist e *reformatting*, millega näiteks lisatakse koodi loetavuse tagamiseks automaatselt vajalik arv taandeid.

4.11 Testimisest

Loodud laienduse iseärasusi arvestades oli kõige asjakohasem seda testida kasutades funktsionaalset testimist. Ühikteste ei olnud antud olukorras otstarbekas kirjutama hakata, sest kõige olulisemad funktsioonid ei tagasta kasutajale mitte midagi. Seetõttu otsustasin kasutada terve arenduse jooksul ainult funktsionaalset testimist. Funktsionaalse testimise käigus käivitasin peale igat muutust seda puudutavad funktsioonid, et kontrollida, kas kõik töötab või on kuskil tekkinud viimati tehtud muudatuse pärast mingi viga. Lisaks testisin arenduse hilisemas järgus läbi erinevaid võimalikke eriolukordi. Mõned testitud veaohtlikud kohad olid järgmised.

1. Teiselt tabelilt pärimise kaudu loodud *check* kitsenduste ümbernimetamine. . PostgreSQL-is saab luua tabeli teise tabeli põhjal kasutades pärimist (`INHERITS` klausel). Sellisel juhul kehtib päritava tabeli (nt *Isik*) *check* kitsendus (nt perenimi ei tohi olla tühi string) ka pärijaks oleva tabeli (nt *Klient*) kohta, kuid kitsendust ise pärijaks olevas tabelis füüsiliselt ei ole ning seega ei saa ka päritud kitsenduse nime läbi pärijaks oleva tabeli muuta. Alguses jäi see asjaolu funktsioone koostades arvestamata ning testimisel tekkis viga. Edaspidi jätab laiendus päritud kitsendused muutmisele kuuluvatest kitsendustest välja.

2. Väga paljude seotud veergudega kitsenduse nime muutmine. Kui kitsendusega on seotud suur hulk veergusid, siis tõenäoliselt tuleb genereeritav nimi lubatust pikem ning seda tuleb lühendada. Testimisel tekkis olukord, kus nime lühendamisel tekkis järjestiku mitu alakriipsu. Selline tulemus tekkis, sest veergude nimed sisaldasid sõnade eraldamiseks alakriipse ning veeru nime lühendades jäi lühendatud nime lõppu viimaseks märgiks just alakriips. Selle probleemi lahendamiseks võetakse peale nime lühendamist lühendatud nime lõppu jäänud alakriipsud ära.
3. Mitme samade parameetritega (sama tüüp või seotud veerud) kitsenduse ümbernimetamine. Suurt tähelepanu vajas kitsenduste nimede genereerimisel erinevat tüüpi kitsenduse nimede skoobiga arvestamine ja nimes sobiva järjekorranumbri valimine. Selleks tuli palju erinevaid võimalikke olukordi läbi katsetada, et avastada, kus võiks nimekonflikt tekkida.

Testimiseks proovisin ka laiendust rünnata SQL-süstimisega. Ilma seda vältimata võimaldaks seda ainult üks loodud funktsioon `rename_constraint(table_n REGCLASS, old_name TEXT, new_name TEXT)`, kus dünaamiliselt luuakse kitsenduse nime muutmiseks `ALTER TABLE` lause. Kuigi olen selle vastu rakendanud süsteemipoolset funktsiooni quote_ident(), käivitasin siiski kindluse tagamiseks testandmebaasis lause `SELECT rename_constraint('calendar', 'calendar_pkey', 'calendar_pk; drop table acl;')`. Peale seda andmebaasi uurides on tabel `acl` alles ning `calendar` tabeli primaarvõtme nimi „`calendar_pk; drop table acl;`“. Lisaks proovisin ka sellist võimalust, kus ühe välisvõtme kitsendusega seonduva veeru nimi on „create user x;“. Käivitasin lause `SELECT rename_all_constraints_in_schema('postgresql_default', 'public', 'foreign key')`. Kui laiendus ei oleks kaitstud, siis tekkiks serverisse kasutaja nimega x. Peale lause käivitamist aga antud nimega kasutajat ei loodud. Seega on laiendus SQL-süstimise vastu kaitstud.

4.12 Installeerimine

Laienduse installeerimiseks tuleb teha järgmised sammud.

1. Tõmba alla laienduse failid ja paiguta need serverid PostgreSQL lähtekoodi *contrib* kataloogi alamkataloogi.
2. Koodi kaustas olles käivita käsurealt installeerimiseks
   `make`

```
make install
```

3. Andmebaasis, kuhu laiendus lisatakse, käivita lause

```
CREATE EXTENSION constr_name_unif;
```

Peale seda on laiendus andmebaasis kasutamiseks valmis. Olemasolevate kitsenduste nimedega tutvumiseks saab käivitada funktsiooni signatuuriga `get_all_constraints()` ning nende muutmiseks `rename_all_constraints(pattern_name TEXT)`, kus tuleb täpsustada kasutatava mustri nimi. Lisaks on olemas variatsioone nendest kahest funktsioonist, mis võimaldavad vaadata ja muuta kitsenduste nimesid vastavalt tüübi ja/või andmebaasiskeemi järgi. Peale installeerimist on olemas kaks mustrit, aga neid saab vajadusel juurde defineerida käivitades funktsiooni signatuuriga `add_pattern(pattern_name TEXT, delimiter TEXT, has_suffix BOOLEAN, has_short_abbreviation BOOLEAN)`. Täpsema kasutusjuhendiga saab tutvuda laienduse lehel kas GitHubis [51] või PostgreSQL laienduste lehel [52]. Funktsioonide väljakutsumiseks tuleb PostgreSQL-is kasutada `SELECT` lauset (vt Joonis 18). `get_all_constraints` on tabelifunktsioon ja selle poole pöördumise saab panna SELECT lause FROM klauslisse. Väljakutses saab parameetritele viidata nii nime järgi kui ka positsiooniliselt.

```
SELECT * FROM get_all_constraints();
SELECT rename_all_constraints(pattern_name:='postgresql_default');
SELECT rename_all_constraints('postgresql_default');
```

Joonis 18. Laienduse funktsioonide väljakutsumise näide.

4.13 Piirangud ja edasise arendamise võimalused

See punkt kirjeldab loodud laienduse funktsionaalsuse piiranguid ja seega ka võimalusi laienduse edasiarendamiseks.

- Kõige suurem piirang on seatud PostgreSQL andmebaasisüsteemi versioonile. Esialgne laienduse versioon töötab ainult alates 9.2 versioonist, sest alates antud versioonist on võimalik kitsendusi ümber nimetada `ALTER TABLE` lausetega. Tulevikus võiks mõelda funktsionaalsusele, mis tuleks toime ka vanematel versioonidel põhinevates andmebaasides kitsenduste ümbernimetamisega.
- Laienduses on hetkel võimalik valida ainult kahe kitsenduse lühendi tüübi vahel: näiteks `pk` või `pkey` primaarvõtme kitsenduste korral. See seab olulise piirangu

kasutatavatele mustrite koostamisele. Tulevikus võiks kasutaja saada ise defineerida, milliseid lühendeid kasutada soovib ja need vastavalt kitsenduse tüübile defineerida.

- Praegu ei ole võimalik kasutaja poolt määrata, mis järjekorras erinevad komponendid nimesse panna. Kasutaja saab ainult valida, kas kitsenduse tüübi märkimiseks kasutatakse ees- või järelliidet. Samas tabelite ja veergude nimede järjekorda kasutaja määrata ei saa. Tulevikus võiks olla kasutajal võimalus määrata, mis järjekorras ja kas üldse mingeid komponente nimesse lisatakse.
- Kui sama tabeli samal veergude hulgale kehtib mitu samatüübilist kitsendust, siis tulemuseks olevad kitsenduse nimed erinevad üksteisest nimes kasutatavate järjekorranumbrite poolest. Eriti *check* kitsenduste korral, mille vastu eksimisel näidatakse seda nime veateates, pole sellised nimed kasutajatele arusaadavuse mõttes parimad. Tulevikus võiks kaaluda võimalust kasutada *check* kitsenduse nime genereerimisel ühe sisendina ka selle loogikaavaldist.
- Hetkel ei logita andmebaasis ümbernimetamisi. Tulevikus võiks kaaluda laiendusse logitabeli lisamist, kus registreeritakse kõik ümbernimetamise katsetused. Lisaks käesolev realisatsioon ei logi ümbernimetamisel tekkinud vigu. See tähendab, et kui kitsenduste ümbernimetamisel tekkis viga ja see toimus osaliselt, siis tuleb arendajal ümbernimetamata jäänud kitsenduste nimed käsitsi üles leida.
- Loodud koodi nõrkuseks on, et kui funktsioonis kutsutakse välja abifunktsioon, siis parameetrite väärtused (argumendid) antakse ette positsiooniliselt, mitte parameetrite nimesid kasutades. Kui väljakutsutava funktsiooni definitsioonis parameetrite järjekord muutub, siis läheb väljakutsumise kood katki. Parem oleks kutsuda funktsioone välja nii, et väljakutses viidatakse nimeliselt parameetritele (vt Joonis 19).

```
PERFORM generate_new_name_and_rename_constraint(constraint_row, pattern_name,
FALSE);

PERFORM
generate_new_name_and_rename_constraint(constraint_row:=constraint_row,
pattern_name:=pattern_name, table_abbreviation:=FALSE);
```

Joonis 19. Positsiooniline vs. parameetrite nimedel põhinev argumentide etteandmine.

5 Laienduse katsetus

Pärast laienduse valmimist tegin kõigist kolmest jaotises 3 välja toodud andmebaasist koopiad. Kopeeritud andmebaaside nimedeks panin sama nime, mis algsel andmebaasil ning lisasin lõppu liite `_test`. Testimiseks mõeldud andmebaasidesse installeerisin loodud laienduse, et katsetada laienduse töötamist ning võrrelda kitsenduste nimesid enne ja pärast laienduse kasutamist.

Kõigis kolmes andmebaasis käivitasin laienduse funktsiooni `rename_all_constraints({mustri_nimi})`, mis muudab vastavalt ette antud mustrile kõikides mittesüsteemsetes skeemides igat tüüpi (v.a. `NOT NULL`) kitsenduste nimed. Kasutasin katsetamiseks mõlemat installeerimisega kaasa tulevat nimede mustrit.

Esimesena ühtlustasin kitsenduse nimed OTRS rakenduse testandmebaasis mustriga `postgresql_default`, sest enamuses kitsenduste nimedes seal andmebaasis oli kasutusel kitsenduse tüübi eesliide ning valitud muster lisab järelliite. Selleks käivitasin lause `SELECT rename_all_constraints('postgresql_default')`. Funktsiooni tegevus võttis aega üks minut ning selle ajaga muudeti ära 406 kitsenduse nimed. Ülejäänud kahes andmebaasis kitsenduste ümbernimetamiseks kasutasin mustrit `snake_case_with_short_prefix`, sest seal olid paljud kitsendused andmebaasisüsteemi poolt nimetatud ning seega sisaldasid nimed kitsenduse tüüpi iseloomustavat järelliidet. Käivitasin lause `SELECT rename_all_constraints('snake_case_with_short_prefix')`. LedgerSMB testandmebaasis võttis kõigi 526 kitsenduse nimetamine aega kaks minutit. FusionForge'i testandmebaasis nimetati ümber 213 kitsendust ning see võttis aega 23 sekundit. Lisas 4 on esitatud ekraanipildid LedgerSMB andmebaasis tehtud `SELECT * FROM get_all_constraints()` päringu tulemusest nii enne (vt Joonis 24) kui pärast (vt Joonis 25) nimede ühtlustamist.

Seejärel käivitan kõigis kolmes testandmebaasis *Naming* kontrollpäringud, et leida uued kõige populaarsemad nimetamise mustrid ning võrrelda erinevate mustrite arvu enne ja pärast laienduse kasutamist. Võrdluseks, enne laienduse rakendamist oli erinevaid mustreid päringu järgi: FusionForge – 62, OTRS – 8, LedgerSMB – 24. Peale nimede

ühtlustamist on erinevaid mustreid järgmiselt: FusionForge – 20, OTRS – 22, LedgerSMB – 37. Siit tuleb selgelt välja, et testpäring ei anna täielikult õiget tulemust sobitades erinevad nime komponendid mingil määral valesti. Viga tuleb testpäringus sisse siis, kui näiteks tabeli nime mingi osa sisaldab endas mingi seotud veeru nime, sel juhul sobitatakse kitsenduse nimest mustrisse vahel enne veeru nimi ja hilje9m alles tabeli nimi, jättes ülejäänud tabeli nime mustrist välja (nt `fk_payroll_pa{primary_column}_timeoff_{child_column}`, kus tabeli nimi on `payroll_paid_timeoff`). Lisaks ei tuvasta testpäring ära, kui liiga pika nime puhul tabeli nimest akronüüm tehakse. Sellest ka suuremad erinevate mustrite arvud. Peale laienduse rakendamist käsitsi andmebaasist üle vaadates on selgelt näha, et tegelikult on kõik nimed siiski ühe mustri järgi koostatud. Näiteks kui OTRS mustrite arvus akronüümidega nimed üheks mustriks lugeda ja nime komponendid õigesti nimesse sobitada, on erinevaid mustreid tegelikult kokku seitse. Seda saan väita ka teades kirjutatud programmikoodi. Populaarsemad leitud kitsenduste nimede mustrid iga testandmebaasi kohta peale nimede ühtlustamist on toodud Tabel 8.

Tabel 8. Kitsenduste nimede mustrid peale ühtlustamist.

Andmebaasi-objekti tüüp	FusionForge
Välisvõtme kitsendus	`fk_{child_table}_{primary_table}_{child_column}` (67) `fk_{child_table}_{child_column}` (3) `fk_{child_table}_{primary_table}_{child_column}2` (2)
Primaarvõtme ja unikaalsuse kitsendus	`pk_{table}` (87) `uq_{table}_{column}` (4)
	LedgerSMB
Check kitsendus	`ck_{table}` (28) `ck_{table}2` (9)
Välisvõtme kitsendus	`fk_{child_table}_{primary_table}_{child_column}` (205) `fk_{child_table}_{child_column}_{child_column}` (13) `fk_{child_table}_{child_table}_{child_column}` (5) `fk_payroll_pa{primary_column}_timeoff_{child_column}`(3)
Primaarvõtme ja unikaalsuse kitsendus	`pk_{table}` (87) `uq_{table}_{column}` (59)

	OTRS
Välisvõtme kitsendus	{child_table}_{primary_table}_{child_column}_fkey (210) {child_table}_val{primary_column}_{child_column}_fkey(29) {child_table}_{child_table}_type_{child_column}_fkey (4)
Primaarvõtme ja unikaalsuse kitsendus	{table}_pkey (84) {table}_{column}_key (49)

6 Kokkuvõte

Antud lõputöö põhieesmärk oli luua PostgreSQL laiendus (*extension*), mis ühtlustaks PostgreSQL andmebaasisüsteemis baastabelitega otse seotud kitsenduste nimesid. Eelnevalt oli tarvis veenduda laienduse tarvilikkuses ja selles, et nii nagu programmikoodis, on ka andmebaasides nimede andmine keeruline ülesanne, mistõttu jääb tihti nimede andmisel täielik järjekindlus saavutamata. Selle tõestamiseks uuriti kolme vabavaralise rakenduse andmebaase. Uuritavateks andmebaasideks olid rakenduste FusionForge (versioon 6.1beta1), LedgerSMB (versioon 1.5.20) ja OTRS (versioon 6.0.6) andmebaasid.

Valitud andmebaaside uurimisel selgus, et üheski andmebaasis ei olnud kasutusel ühtset kitsenduste nimetamise mustrit. Näiteks FusionForge'i andmebaasis leidus 60 erinevat kitsenduste nimetamise mustrit. Seega on probleem aktuaalne ning laienduse arendus õigustatud.

Antud lõputöö koosneb neljast osast: nimetamise teoreetilisest taustast, nimede uurimisest vabavaraliste rakenduste andmebaasides, loodud laienduse nõuetest, funktsionaalsusest ja realisatsioonist ning viimaks laienduse katsetusest. Teine peatükk on teoreetiline taust nimetamise olulisusest tarkvaraarenduses üldiselt ja ka SQL-andmebaasides. Sealjuures käsitletakse eraldi nimetamise üldpõhimõtteid SQL-andmebaasides ning eraldi ka kitsenduste nimetamist. Eraldi alapeatükis esitatakse kirjandusest pärit kitsenduste nimetamise soovitusi. Need on ühtlasi ka aluseks loodava laienduse funktsionaalsusele.

Järgmine peatükk sisaldab valitud andmebaaside põhjal andmebaasiobjektide nimede üldist analüüsi ja eraldi kitsenduste nimede analüüsi. Seal on välja toodud erinevates andmebaasides kasutusel olevad nimede mustrid, nimede pikkused ja muud näitajad. Kõik kolm uuritud andmebaasi olid disainilt erinevad, kuid neid kõiki ühendas üks tunnusjoon – nimetamise järjekindlusetus.

Järgmine peatükk on laienduse kavandist ja realisatsioonist. Alguses selgitatakse PostgreSQL-i laienduste mehhanismi tausta ning kuidas selles andmebaasisüsteemis on

võimalik kitsendusi ümber nimetada. Peale seda esitan laienduse funktsionaalsed ja mittefunktsionaalsed nõuded, valdkonnamudeli ning ka tähelepanekud ja detailid seoses süsteemi realiseerimisega ning testimisega. Samuti kirjeldab laienduse installeerimist ja kasutamist. Loodud laiendus on kättesaadav:

- GitHubi koduleht: https://github.com/katrinaibast/constr_name_unif
- PostgreSQL Extension Network: https://pgxn.org/dist/constr_name_unif

Viimane peatükk selgitab laienduse katsetamist eelnevalt uuritud andmebaasides. Kõigis kolmes andmebaasis käivitati funktsioon kõikide kitsenduste ümbernimetamiseks. Peale ühtlustamist kitsenduste nimede mustrite arvu vaadates olid erinevate mustrite arvud tunduvalt vähenenud ehk järjekindlus andmebaasides kitsenduste nimetamisel suurenes.

Käesoleva lõputöö raames loodud PostgreSQL andmebaasisüsteemi laiendus kitsenduste nimede ühtlustamiseks on kasulik töövahend igale antud süsteemis andmebaasiga töötavale arendajale, et juba olemasolevas andmebaasis, kus ühtne kitsenduste nimetamise stiil puudub, ilma suurema vaeva ja ajakuluta kitsendusi järjekindlamalt nimetada.

Kasutatud kirjandus

[1] FusionForge [WWW] https://fusionforge.org/ (18.04.2018)

[2] LedgerSMB [WWW] https://ledgersmb.org/ (18.04.2018)

[3] OTRS [WWW] https://otrs.com/ (18.04.2018)

[4] Martin, R.C. Clean Code. A Handbook of Agile Software Craftsmanship. Pearson Education, 2009

[5] Fowler, M. TwoHardThings, 2009 [WWW] https://martinfowler.com/bliki/TwoHardThings.html (19.04.2018)

[6] Volkov, I. (2017). Postgresql andmebaasisüsteemi laiendamine andmete maskimise võimalustega: magistritöö. Tallinn : Tallinna Tehnikaülikool. – TTÜR Digikogu. https://digi.lib.ttu.ee/i/?8031 (10.05.2018)

[7] Deissenboeck, F. and Pizka, M. – Concise and consistent naming. Software Quality Journal, 2006, 14(3), pp. 261-282.

[8] Wang, Y., Wang, C., Li, X., Yun, S. and Song, M. How are Identifiers Named in Open Source Software? On Popularity and Consistency. – International Journal of Computer and Information Technology, 2014, 3(3), pp. 616-625.

[9] DB-Engines Ranking – DB-Engines [WWW] https://db-engines.com/en/ranking (10.05.2018)

[10] Hevner, A. R, March, S. T., Park, J, Ram, S. Design Science in Information Systems Research – MIS Quarterly, 2004, 28(1), pp. 75-105. The ACM Digital Library

[11] Importance of Writing Clean Code – Dev.to [WWW] https://dev.to/mohitrajput987/importance-of-writing-clean-code (19.04.2018)

[12] FusionForge source code – GitHub [WWW] https://github.com/fusionforge/fusionforge/blob/4fdc095972d45b03a5a1576bdfda1db299fc38aa/src/db/1-fusionforge-init.sql (18.04.2018)

[13] Koodi refaktoreerimine – Vikipeedia, vaba entsüklopeedia [WWW] https://et.wikipedia.org/wiki/Kasutaja:Aleemet/Koodi_refaktoreerimine (20.04.2018)

[14] Paide, R. (2017). Juhtumiuuring tehnilisest võlast Eesti IKT sektori ettevõtetes: magistritöö. Tallinn : Tallinna Tehnikaülikool. – TTÜR Digikogu. https://digi.lib.ttu.ee/i/?8865 (10.05.2018)

[15] SQL Style Guide [WWW] http://www.sqlstyle.guide/ (26.04.2018)

[16] Lindland, O.I., Sindre, G., Solvberg, A. Understanding quality in conceptual modeling. IEEE Software 11, 42–49 (1994). doi: 10.1109/52.268955

[17] Lexical Structure – PostgreSQL Documentation [WWW] https://www.postgresql.org/docs/10/static/sql-syntax-lexical.html (10.05.2018)

[18] PostgreSQL: default constraint names – Stack Overflow forum [WWW] https://stackoverflow.com/questions/4107915/postgresql-default-constraint-names (22.04.2018)

[19] Constraint naming standard – Open ACS [WWW] https://openacs.org/doc/eng-standards-constraint-naming (22.04.2018)

[20] LedgerSMB source code – GitHub [WWW] https://github.com/ledgersmb/LedgerSMB/blob/master/sql/Pg-database.sql (18.04.2018)

[21] Microsoft SQL Server – Wikipedia, The Free Encyclopedia [WWW] https://en.wikipedia.org/wiki/Microsoft_SQL_Server (27.04.2018)

[22] Defining Constraints – MSSQLTips [WWW] https://www.mssqltips.com/sqlservertutorial/2905/defining-constraints/ (27.04.2018)

[23] What is the purpose of constraint naming– Stack Overflow forum [WWW] https://stackoverflow.com/questions 1397440/what-is-the-purpose-of-constraint-naming/ (27.04.2018)

[24] Eessaar, E. SQL-andmebaaside ja nende projekteerimise põhimõisteid. [WWW] http://maurus.ttu.ee/346 (30.04.2018)

[25] CREATE TABLE – PostgreSQL Documentation [WWW] https://www.postgresql.org/docs/10/static/sql-createtable.html (10.05.2018)

[26] Camel case – Wikipedia, The Free Encyclopedia [WWW] https://en.wikipedia.org/wiki/Camel_case (02.05.2018)

[27] Constraint Naming Standard – Database Design Resource [WWW] http://www.databasedesign-resource.com/constraint-naming-standard.html (26.04.2018)

[28] SQL Server Naming Conventions and Standards – DotNetTricks [WWW] https://www.dotnettricks.com/learn/sqlserver/sql-server-naming-conventions-and-standards (02.05.2018)

[29] J. Edison.Naming Conventions in Database Modeling – Vertabelo [WWW] http://www.vertabelo.com/blog/technical-articles/naming-conventions-in-database-modeling (02.05.2018)

[30] PostgreSQL. Ecosystem:Customer relationship management (CRM) [WWW] https://wiki.postgresql.org/wiki/Ecosystem:Customer_relationship_management_(CRM) (10.05.2018)

[31] PostgreSQL. Ecosystem:Enterprise resource planning (ERP) [WWW] https://wiki.postgresql.org/wiki/Ecosystem:Enterprise_resource_planning_(ERP) (10.05.2018)

[32] FusionForge source code – GitHub [WWW] https://github.com/fusionforge/fusionforge/blob/4fdc095972d45b03a5a1576bdfda1db299fc38aa/src/db/1-fusionforge-init.sql (18.04.2018)

[33] OTRS source code – GitHub [WWW] https://github.com/OTRS/otrs/blob/891a56430b6e355e758e9a05a15142bfb32605cc/scripts/database/otrs-schema.postgresql.sql (18.04.2018)

[34] apex.ttu.ee serveri rakendus [WWW] http://apex.ttu.ee/queries2 (18.04.2018)

[35] Vanaveski, M. (2015). Andmebaasi disaineri töökoha realiseerimine PostgreSQL andmebaaside disaini kontrollimise süsteemi jaoks: bakalaureusetöö. Tallinn: Tallinna Tehnikaülikool. – TTÜR Digikogu https://digi.lib.ttu.ee/i/?3466 (10.05.2018)

[36] Eessaar, E. (2015). On Query-based Search of Possible Design Flaws of SQL Databases . In: Innovations and Advances in Computer, Information, Systems Sciences, and Engineering, Lecture Notes in Electrical Engineering, Vol. 313: International Conference on Systems, Computing Sciences and Software Engineering (SCSS 12). pp. 53-60.

[37] Snake case – Wikipedia, The Free Encyclopedia [WWW] https://en.wikipedia.org/wiki/Snake_case (22.04.2018)

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

[39] Code Review - Wikipedia, The Free Encyclopedia [WWW] https://en.wikipedia.org/wiki/Code_review (26.04.2018)

[40] PostgreSQL Extension Network [WWW] https://pgxn.org/ (27.04.2018)

[41] Instant SQL Formatter [WWW] http://www.dpriver.com/pp/sqlformat.htm (19.05.2018)

[42] Liquibase [WWW] https://www.liquibase.org (19.05.2018)

[43] Packaging Related Objects into an Extension [WWW] https://www.postgresql.org/docs/current/static/extend-extensions.html (25.04.2018)

[44] ASCII Table and Description [WWW] https://www.asciitable.com/ (04.05.2018)

[45] Choose an open source licence [WWW] https://choosealicense.com/ (30.04.2018)

[46] MIT Licence – Choose an open source licence [WWW] https://choosealicense.com licenses/mit/ (30.04.2018)

[47] Overview – PostgreSQL Documentation [WWW] https://www.postgresql.org/docs/10/static/plpgsql-overview.html (10.05.2018)

[48] Locmetrics [WWW] http://www.locmetrics.com/ (19.05.2018)

[49] Extension Building Infrastructure – PostgreSQL Documentation [WWW] https://www.postgresql.org/docs/current/static/extend-pgxs.html (15.05.2018)

[50] IntelliJ IDEA – JetBrains [WWW] https://www.jetbrains.com/idea/ (15.05.2018)

[51] constr_name_unif – GitHub [WWW] https://github.com/katrinaibast/constr_name_unif (15.05.2018)

[52] constr_name_unif – PostgreSQL Extension Network [WWW] https://pgxn.org/dist/constr_name_unif (19.05.2018)

Lisa 1. Ekraanipildid testkeskkonnast

Please select a test: Naming ▼ Select

The test contains 61 queries.

Press **one of the following buttons** to execute all the queries in the test.

Execute | Execute and show only queries that return rows | After clicking the button you can click **here** to jump to the results or just scroll down.

Number of database objects	Names should be expressive
Median of the length of database object names	Names should be expressive.
Median of the length of database object names by their type	Names should be expressive.
Number of different names	Names should be expressive.
Number of of objects and different names by object type	Names should be expressive.
Frequent names	Names should be expressive.
Too short names	The names should be meaningful and searchable
Frequency of name lengths	Names should be expressive.
The longest names by object type	Names should be expressive.
The shortest names by object type	Names should be expressive.
Popularity of name components	Names should be expressive.
Names with perhaps too many subcomponents	The number of subcomponents should not be bigger than four.
Number of names by the number of subcomponents in names	Names should be expressive.
Median number of subcomponents in names	Names should be expressive.
Too generic names (tables and their columns)	In SQL databases data/information is represented as values in columns. It is not a good style to use generic names like table, data, information, column etc in the names of tables of columns. Also avoid too generic column names like: id, tyyp, kood, aeg, kp,type, code, time, date, fk, pk. The corresponding code smell in case of cleaning code is "Add Meaningful Context".
The name of the table that implements a relationship does not explain the nature of the relationship	The names should be derived from the domain. For instance, instead of name Course_Lecturer it is better to have name Teaching.
Too generic names (routines)	Do not use the same word with multiple purposes. Do not use the words 'lisa', 'muuda', 'kustuta', 'add', 'delete', 'update' as the names of routines because it would make the code much less understndable. It corresponds to the clean code suggestion "Don't Pun".
Too generic names (primary key columns)	You shouldn't use generic names like ID, code, kood etc. In different tables as the primary key columns. The corresponding code smell in case of cleaning code is "Add Meaningful Context".
Too generic names (view columns)	The names of view columns that contain names should refer to the role that the corresponding entity type has in the view. For instance, if view active_product has a column with the name surname, then the column name does not give information what is the role of the person in the context of the view. Better name would be registrator_surname.
Names that do not take into account cultural context	It would be better to replace first_name => given_name and last_name => surname
Case sensitive names	Identifiers of database objects should be case insensitive in order to simplify their management
Using SQL identifiers as names causes confusion	Names shouldn't cause confusion
Overview of prefixes	One should be consistent in naming, including in the use of prefixes
Prefixes of base table names	Do not use prefixes in case of base table names. Derive the names from the names of entity types.
Names contain numbers	Names should be informative. Duplicates should be avoided. Numbers in names are a possible sign of duplication or database objects or unclear names.
Name contains two or more consecutive subscripts	Improve the readability of names
Name starts or ends with a subscript	Improve the readability of names
Database public interface elements that names contain the letters õäöüÕÄÖÜ	Although permitted by the DBMS, such letters might make it more difficult to use the interface by other programs.
Names of the columns of views that have been given by the system	The creators of the view should specify the name themselves to avoid ugly names and nasty surprises.
Perhaps too long names that have been automatically shortened	Automatic code modification could break it somewhere
Perhaps is not snake_case	Preffer snake_case over PascalCase and camelCase in names
Repeating constraint names	Different things should have different names. But here different constraints have the same name.
Naming of constraints	To find out how consistent are the constraint names and whether users have given themselves the names.
System-generated domain CHECK constraint names	Names should follow the same style. If there is a mix of system-generated and user-defined names, then the style is most probably

Joonis 20. Ekraanipilt *Naming* testis olevate päringute kirjeldusest.

Query: **Mixing system- and user defined names of constraints (constraints that involve one column)**
There should not be any
The query found 3 rows!

nr	type	cnt_total	cnt_system_generated	user_defined
1	FOREIGN KEY	264	261	3
2	PRIMARY KEY	84	84	0
3	UNIQUE	49	49	0

Query: **Table constraints with the cardinality bigger than one**
Overview

nr	cardinality	type	schema_name	table_name	constraint_name
1	5	UNIQUE	public	link_relation	link_relation_source_objec_source_target_objec_target_typ_key
2	3	UNIQUE	public	calendar_appointment_ticket	calendar_appointment_ticket_calendar_id_ticket_id_rule_id_key
3	3	UNIQUE	public	ticket_flag	ticket_flag_ticket_id_ticket_key_create_by_key
4	2	UNIQUE	public	dynamic_field_obj_id_name	dynamic_field_obj_id_name_object_name_object_type_key
5	2	UNIQUE	public	notification_event_message	notification_event_message_notification_id_language_key
6	2	UNIQUE	public	pm_entity_sync	pm_entity_sync_entity_type_entity_id_key
7	2	UNIQUE	public	scheduler_recurrent_task	scheduler_recurrent_task_name_task_type_key
8	2	UNIQUE	public	service_sla	service_sla_service_id_sla_id_key
9	2	UNIQUE	public	sysconfig_modified	sysconfig_modified_sysconfig_default_id_user_id_key

Query: **Patterns of the names of primary key, unique, check, exclude, and foreign key constraints as well as user-defined non-unique indices that are associated with exactly one column**
Overview. The fewer, the better. The more there are patterns, especially with the cnt=1, the worse.

nr	pattern	types	cnt
1	**childtab_childcol**_fkey	FK	259
2	**childtab_childtab_childcol**_fkey	FK	3
3	val**primarycol_childcol**_fkey	FK	2
4	**table_column**	INDEX	95
5	communication_**column**	INDEX	4
6	article_data_mime_transmission_**column**	INDEX	2
7	**table**_key	INDEX	1
8	**table**_message_id_md5	INDEX	1
9	**table**_pkey	KEY	84
10	**table_column**_key	KEY	49

Query: **Names of constraints or non-unique indices that are directly connected to a base table and that do not contain the table name**
If exists, then suspicious
The query found 8 rows!

nr	type	table_schema	table_name	column_name	constraint_name
1	INDEX	public	article_data_mime_send_error	article_id	article_data_mime_transmission_article_id
2	INDEX	public	article_data_mime_send_error	message_id	article_data_mime_transmission_message_id
3	INDEX	public	communication_log	direction	communication_direction
4	INDEX	public	communication_log	start_time	communication_start_time
5	INDEX	public	communication_log	status	communication_status

Joonis 21. Ekraanipilt *Naming* testis olevate päringute tulemustest.

Lisa 2. Laienduse funktsioonide kirjeldused

Tabel 9. Kasutajale väljakutsumiseks mõeldud funktsioonide kirjeldused.

Funktsiooni signatuur	Kirjeldus
`get_all_constraints()` `RETURNS SETOF constraint_type`	Kõikide mitte-süsteemsete skeemide kitsenduste, mille nime on võimalik muuta, vaatamiseks mõeldud funktsioon. Tagastab kitsenduste andmed tabeli kujul.

`get_all_constraints_in_schema(`
`schema_n TEXT)`
`RETURNS SETOF constraint_type`
	Kõikide kindlas mitte-süsteemses skeemis olevate kitsenduste, mille nime on võimalik muuta, vaatamiseks mõeldud funktsioon. Tagastab kitsenduste andmed tabeli kujul. Parameetri `schema_n` oodatud väärtuseks on soovitud skeemi nimi.

`get_all_constraints(type TEXT)` `RETURNS SETOF constraint_type`	Kõikide mitte-süsteemsete skeemide etteantud tüüpi kitsenduste, mille nime on võimalik muuta, vaatamiseks mõeldud funktsioon. Tagastab kitsenduste andmed tabeli kujul. `type` võimalikud väärtused: 'PRIMARY KEY', 'FOREIGN KEY', 'CHECK', 'UNIQUE', 'EXCLUDE' (väärtused on tõstutundetud).

`get_all_constraints_in_schema(`
`schema_n TEXT,`
`type TEXT)`
`RETURNS SETOF constraint_type`
	Kindlas mitte-süsteemses skeemis olevate etteantud tüüpi kitsenduste, mida on võimalik muuta, vaatamiseks mõeldud funktsioon. Tagastab kitsenduste andmed tabeli kujul. Parameetri `schema_n oodatud` väärtuseks on soovitud skeem. Parameetri `type` võimalikud väärtused: 'PRIMARY KEY', 'FOREIGN KEY', 'CHECK', 'UNIQUE', 'EXCLUDE' (väärtused on tõstutundetud).

`add_pattern(`
`pattern_name TEXT,`
`delimiter TEXT(1),`
`has_suffix BOOLEAN,`
`has_short_abbreviation`
`BOOLEAN)`
`RETURNS VOID`
	Funktsioon lisamaks uut mustrit, mille järgi kitsenduse nimesid hiljem muuta. Parameetri `pattern_name` oodatud väärtus on mustri unikaalne nimi, `delimiter` oodatud väärtuseks on tähemärk, millega sõnu eraldada, `has_suffix` oodatud väärtuseks on tõeväärtus vastavalt järelliite rakendamise soovile, `has_short_abbreviation` oodatud väärtuseks on tõeväärtus vastavalt lühikese kitsenduse tüübi lühendi kasutamise soovile. Funktsioon ei tagasta midagi.

Funktsiooni signatuur	Kirjeldus
`get_all_patterns()` `RETURNS SETOF PATTERN`	Funktsioon vaatamaks kõiki salvestatud mustreid, mida on võimalik nimede muutmisel rakendada. Tagastab mustrite andmed tabeli kujul.
`delete_pattern(name TEXT)` `RETURNS VOID`	Funktsioon kustutamaks kindlat salvestatud mustrit. Parameetri `name` oodatud väärtus on kustutatava mustri nimi. Funktsioon ei tagasta midagi.

`rename_all_constraints(`
`pattern_name TEXT)`
`RETURNS VOID`
	Kõikide võimalike kitsenduste kõikides mitte-süsteemsete skeemides ümbernimetamiseks mõeldud funktsioon. Parameetri `pattern_name` oodatav väärtus on kasutatava mustri nimi. Funktsioon ei tagasta midagi.

`rename_all_constraints`
`_in_schema(`
`pattern_name TEXT,`
`schema_n TEXT)`
`RETURNS VOID`
	Kõikide võimalike kitsenduste kindlas mitte-süsteemses skeemis ümbernimetamiseks mõeldud funktsioon. Parameetri `pattern_name` oodatav väärtus on kasutatava mustri nimi ja `schema_n` oodatav väärtus on soovitud skeemi nimi. Funktsioon ei tagasta midagi.

`rename_all_constraints(`
`pattern_name TEXT,`
`type TEXT)`
`RETURNS VOID`
	Kõikide võimalike ette antud tüüpi kitsenduste kõikides mitte-süsteemsetes skeemides ümbernimetamiseks mõeldud funktsioon. Parameetri `pattern_name` oodatav väärtus on kasutatava mustri nimi. `type` võimalikud väärtused: ’PRIMARY KEY', 'FOREIGN KEY', 'CHECK', 'UNIQUE', 'EXCLUDE' (väärtused on tõstutundetud). Funktsioon ei tagasta midagi.

`rename_all_constraints`
`_in_schema(`
`pattern_name TEXT,`
`schema_n TEXT,`
`type TEXT)`
`RETURNS VOID`
	Kõikide võimalike ette antud tüüpi kitsenduste kindlas mitte-süsteemses skeemis ümbernimetamiseks mõeldud funktsioon. Parameetri `pattern_name` oodatav väärtus on kasutatava mustri nimi ja `schema_n` oodatav väärtus on soovitud skeemi nimi. `type` võimalikud väärtused: ’PRIMARY KEY', 'FOREIGN KEY', 'CHECK', 'UNIQUE', 'EXCLUDE' (väärtused on tõstutundetud). Funktsioon ei tagasta midagi.

Tabel 10. Abifunktsioonide kirjeldused.

Funktsiooni signatuur	Kirjeldus

`add_to_abbreviated_tables(`
`  table_n TEXT,`
`  abbreviated_name TEXT,`
`  schema_n TEXT)`
`RETURNS VOID`
	Funktsioon, mida kustutakse välja juhul, kui mingi tabeli nime on vaja ümbernimetatava kitsenduse nimes kasutamiseks lühendada. Sellega lisatakse andmed lühendatud tabeli nime kohta lühendatud tabelinimede tabelisse. Parameetri `table_n` oodatav väärtus on tabeli lühendamiseelne nimi, `abbreviated_name` oodatav väärtus on lühendatud tabeli nimi, `schema_n` oodatav väärtus on skeemi nimi, milles on lühendatava nimega tabel. Funktsioon ei tagasta midagi.

`empty_abbreviated_tables()` `RETURNS VOID`	Funktsioon tühjendamaks lühendatud tabelinimede tabelit. Funktsioon ei tagasta midagi.

`abbreviate_table_name(`
`  table_n TEXT,`
`  schema_n TEXT)`
`RETURNS TEXT`
	Funktsioon, mis lühendab tabeli nime selle vastavaks akronüümiks, eeldusel, et tabeli nimes on sõnad alakriipsuga eraldatud ning tagastab loodud akronüümi. Vastasel juhul tagastab sama nime. Samuti registreerib funktsioon uue nime lühendatud tabelinimede tabelis. Parameetri `table_n` oodatav väärtus on tabeli nimi, mida soovitakse lühendada ja `schema_n` oodatav väärtus on selle tabeli skeemi nimi.

`get_abbreviated_tables()` `RETURNS SETOF ABBREVIATED_TABLE`	Kõikide lühendatud tabelinimede saamiseks mõeldud funktsioon. Tagastab kogu lühendatud tabelinimede tabeli sisu tabelina.

`truncate_name(`
`  name TEXT,`
`  character_count INT)`
`RETURNS TEXT`
	Funktsioon lühendab nime ette antud tähemärkide arvu võrra. Kasutatakse hetkel ainult välisvõtme kitsenduste puhul primaarse tabeli nime lühendamiseks. Parameetri `name` oodatav väärtus on lühendatav nimi ning `character_count` oodatav väärtus on märkide arv. Tagastab lühendatud nime.

`truncate_names(`
`  names TEXT [],`
`  character_count INT)`
`RETURNS TEXT`
	Funktsioon lühendab kõiki nimesid kokku ette antud tähemärkide arvu võrra. Kasutatakse veerunimede lühendamisel, kus kõiki nimesid proovitakse võimalikult ühtlaselt lühendada. Parameetri `name` oodatav väärtus on lühendatavad nimed massiivis ning `character_count` oodatav väärtus on vastav

Funktsiooni signatuur	Kirjeldus
	arv. Tagastab lühendatud nimed ühe sõnena komadega eraldatuna.

`get_constraint_abbreviation(`
`  type TEXT,`
`  is_short BOOLEAN)`
`RETURNS TEXT`
	Etteantud tüüpi kitsenduse lühendi saamiseks mõeldud funktsioon. Tagastab vastavalt kitsenduse tüübile ja vastavalt kas lühema või pikema kitsenduse tüübi valikule lühendi. Parameetri `type` võimalikud väärtused: 'PRIMARY KEY', 'FOREIGN KEY', 'CHECK', 'UNIQUE', 'EXCLUDE' (väärtused on tõstutundlikud). `is_short` oodatav väärtus on tõeväärtus vastavalt, kas soovitakse lühikest (TRUE) või pikka lühendit (FALSE).

`remove_numbers_from_name(`
`  name TEXT)`
`RETURNS VARCHAR`
	Funktsioon, mis eemaldab nimest kõik esinenud numbrit. Kasutatakse, et kontrollida, kas vastava nimega kitsendus ilma järjekorranumbrita on olemas. Parameetri `name` oodatav väärtus on nimi, millest soovitakse numbreid eemaldada. Tagastab sama nime ilma numbriteta.

`get_constraint_name_count(`
`  constraint_n TEXT,`
`  schema_n TEXT)`
`RETURNS INT`
	Funktsioon, mis tagastab arvu, kui mitu korda antud nimi, millest numbrid kontrolli käigus eemaldatakse, ette antud skeemis esineb. Parameetri `constraint_n` oodatav väärtus on otsitava kitsenduse nimi ja `schema_n` oodatav väärtus on skeemi nimi.

`get_constraint_name_count(`
`  constraint_n TEXT,`
`  schema_n TEXT,`
`  table_n TEXT)`
`RETURNS INT`
	Funktsioon, mis tagastab arvu, mitu korda antud nimi, millest numbrid kontrolli käigus eemaldatakse, ette antud tabelis. Parameetri `constraint_n` oodatav väärtus on otsitava kitsenduse nimi, `schema_n` oodatav väärtus on tabeli skeemi nimi ja `table_n` oodatav väärtus on tabeli nimi.

`get_name_length(name TEXT)` `RETURNS INT`	Nime pikkuse saamiseks mõeldud funktsioon. Parameetri `name` oodatav väärtus on soovitud nimi. Funktsioon tagastab nime pikkust tähistava märkide arvu.

`rename_constraint(`
`  _table_n REGCLASS,`
`  old_name TEXT,`
`  new_name TEXT)`
`RETURNS VOID`
	Kitsenduse nime muutmiseks `ALTER TABLE` lause käivitamiseks mõeldud funktsioon. Parameetri `_table_n` oodatav väärtus on tabeli nimi, `old_name` oodatav väärtus on kitsenduse vana ja `new_name` oodatav väärtus on uus nimi. Funktsioon ei tagasta midagi.

Funktsiooni signatuur	Kirjeldus

`rename_constraint_with_same_name(`
`  constraint_n TEXT,`
`  suffix TEXT,`
`  has_suffix BOOLEAN,`
`  number INT)`
`RETURNS VARCHAR`
	Funktsioon, mis sama nime esinemisel vastavas skoobis muudab genereeritud nime, lisades sellele nimekonflikti vältimiseks järjekorranumbri . Kui kasutusel oli järelliide, siis lisab funktsioon järjekorranumbri järelliite ette. Parameetri `constraint_n` oodatav väärtus on muudetav kitsenduse nimi, `suffix` oodatav väärtus on nimele lisatud kitsenduse tüübi lühend, `has_suffix` oodatav väärtus on tõeväärtus, kas kasutusel oli järelliide (TRUE) või eesliide (FALSE). `number` oodatav väärtus on lisatav järjekorranumber.

`get_new_name(`
`  type TEXT,`
`  pattern_name TEXT,`
`  old_name TEXT,`
`  schema_n TEXT,`
`  table_n TEXT,`
`  column_names TEXT,`
`  foreign_table_n TEXT`
`  DEFAULT NULL,`
`  is_table_name_abbreviated`
`  BOOLEAN DEFAULT FALSE)`
`RETURNS TEXT`
	Funktsioon, kus genereeritakse kitsendusele ette antud mustri järgi uus nimi. Tagastatakse süsteemi jaoks sobiva pikkusega ja nimekonflikti vältiv kitsenduse nimi. Parameetri `type` võimalikud väärtused: 'PRIMARY KEY', 'FOREIGN KEY', 'CHECK', 'UNIQUE', 'EXCLUDE' (väärtused on tõstutundlikud). `pattern_name` oodatav väärtus on nime mustri nimi, `old_name` oodatav väärtus on kitsenduse vana nimi, `schema_n` oodatav väärtus on kitsenduse skeemi nimi, `table_n` oodatav väärtus on tabeli nimi, millega kitsendus on seotud. `column_names` oodatav väärtus on kitsendusega seotud olevate veergude nimed, mis on komadega eraldatud, `foreign_table_n` oodatav väärtus on välisvõtme korral primaarse tabeli nimi ning `is_table_name_abbreviated` oodatav väärtus on tõeväärtus – juhul kui tabeli nimi on lühendatud, siis TRUE, muidu FALSE.

`generate_new_name_and_`
`rename_constraint(`
`  constraint_row`
`  CONSTRAINT_TYPE,`
`  pattern_name TEXT,`
`  is_table_name_abbreviated`
`  BOOLEAN)`
`RETURNS VOID`
	Kitsenduse uue nime genereerimiseks ja kitsenduse ümbernimetamiseks mõeldud funktsioon. Parameetri `constraint_row` oodatav väärtus on liittüüpi väärtus, mis sisaldab infot kitsenduse kohta (kitsenduse nimi, tüüp, tabel, skeem, veergude nimed ning refereeritav tabel). `pattern_name` oodatav väärtus on kasutatava mustri nimi ja `is_table_name_abbreviated` oodatav väärtus on tõeväärtus, kas tabeli nime on vaja lühendada (TRUE) või mitte (FALSE). Funktsioon ei tagasta midagi.

`rename_all_constraints_in` `_abbreviated_tables(`	Funktsioon, mis nimetab uuesti ümber kitsendused, mis on seotud lühendatud

Funktsiooni signatuur	Kirjeldus
`pattern_name TEXT)` `RETURNS VOID`	tabelinimede tabelis olevate tabelitega, lühendades nendes samuti tabeli nime. Sisendiks on kasutada soovitud mustri nimi. Funktsioon ei tagasta midagi.

`rename_all_constraints_in_abbreviated_tables(`
`pattern_name TEXT,`
`type TEXT)`
`RETURNS VOID`
	Funktsioon, mis nimetab uuesti ümber ette antud tüüpi kitsendused, mis on seotud lühendatud tabelite tabelis olevate tabelitega, lühendades nendes samuti tabeli nime. Sisendiks `pattern_name` kasutada soovitud mustri nimi. Parameetri `type` võimalikud väärtused: 'PRIMARY KEY', 'FOREIGN KEY', 'CHECK', 'UNIQUE', 'EXCLUDE' (väärtused on tõstutundlikud). Funktsioon ei tagasta midagi.

`rename_all_constraints_in_abbreviated_tables_in_schema(`
`pattern_name TEXT,`
`schema_n TEXT)`
`RETURNS VOID`
	Funktsioon, mis nimetab uuesti ümber kindlas skeemis kitsendused, mis on seotud lühendatud tabelinimede tabelis olevate tabelitega, lühendades nendes samuti tabeli nime. Parameetri `pattern_name` oodatav väärtus on mustri nimi ja `schema_n` oodatav väärtus on skeemi nimi. Funktsioon ei tagasta midagi.

`rename_all_constraints_in_abbreviated_tables_in_schema(`
`pattern_name TEXT,`
`schema_n TEXT,`
`type TEXT)`
`RETURNS VOID`
	Funktsioon, mis nimetab uuesti ümber ette antud tüüpi kitsendused kindlas skeemis, mis on seotud lühendatud tabelinimede tabelis olevate tabelitega. Ümbernimetamise tulemusena on uues nimes lühendatud tabeli nimi. Parameetri `pattern_name` oodatav väärtus on soovitud mustri nimi ja `schema_n` oodatav väärtus on skeemi nimi. `type` võimalikud väärtused: ’PRIMARY KEY', 'FOREIGN KEY', 'CHECK', 'UNIQUE', 'EXCLUDE' (väärtused on tõstutundlikud). Funktsioon ei tagasta midagi.

Lisa 3. Laienduse lähtekoodi näited

```
CREATE OR REPLACE FUNCTION rename_all_constraints_in_schema(
       pattern_name TEXT, schema_n TEXT, type TEXT)
 RETURNS VOID AS $$
DECLARE
 constraint_row CONSTRAINT_TYPE;
BEGIN
 FOR constraint_row IN (SELECT *
  FROM get_all_constraints_in_schema(schema_n, type))
  LOOP
      PERFORM generate_new_name_and_rename_constraint(constraint_row,
      pattern_name, FALSE);
  END LOOP;
 PERFORM rename_all_constraints_in_abbreviated_tables_in_schema(
 pattern_name, schema_n, type);
END;
$$ LANGUAGE plpgsql;
```

Joonis 22. Funktsioon laienduse lähtekoodist kõikide teatud tüüpi kitsenduste kindlas skeemis ümber nimetamiseks.

```
CREATE OR REPLACE FUNCTION get_all_constraints_in_schema(
       schema_n TEXT, type TEXT)
   RETURNS SETOF CONSTRAINT_TYPE AS $$
SELECT *
FROM get_all_constraints_in_schema(schema_n)
WHERE constraint_type = UPPER(type);
$$ LANGUAGE SQL;
```

Joonis 23. Funktsioon laienduse lähtekoodist kõikide teatud tüüpi kitsenduste vaatamiseks.

Lisa 4. Ekraanipildid enne ja pärast laienduse rakendamist

	constraint_name character varying	constraint_type character varying	table_name character varying	constraint_schema character varying	related_columns text	references_table character varying
46	location_line_one_check	CHECK	location	public	line_one	[null]
47	location_mail_code_check	CHECK	location	public	mail_code	[null]
48	location_state_check	CHECK	location	public	state	[null]
49	menu_acl_acl_type_check	CHECK	menu_acl	public	acl_type	[null]
50	note_class_class_check	CHECK	note_class	public	class	[null]
51	oe_class_id_check	CHECK	oe_class	public	id	[null]
52	person_first_name_check	CHECK	person	public	first_name	[null]
53	person_last_name_check	CHECK	person	public	last_name	[null]
54	session_token_check	CHECK	session	public	token	[null]
55	account_checkpoint_account_id_fkey	FOREIGN KEY	account_checkpoint	public	account_id	account
56	account_heading_fkey	FOREIGN KEY	account	public	heading	account_heading
57	account_heading_parent_id_fkey	FOREIGN KEY	account_heading	public	parent_id	account_heading
58	account_heading_translation_trans_id_fkey	FOREIGN KEY	account_heading_tra...	public	trans_id	account_heading
59	account_link_account_id_fkey	FOREIGN KEY	account_link	public	account_id	account
60	account_link_description_fkey	FOREIGN KEY	account_link	public	description	account_link_description
61	account_translation_trans_id_fkey	FOREIGN KEY	account_translation	public	trans_id	account
62	acc_trans_chart_id_fkey	FOREIGN KEY	acc_trans	public	chart_id	account
63	acc_trans_trans_id_fkey	FOREIGN KEY	acc_trans	public	trans_id	transactions
64	acc_trans_voucher_id_fkey	FOREIGN KEY	acc_trans	public	voucher_id	voucher
65	ac_tax_form_entry_id_fkey	FOREIGN KEY	ac_tax_form	public	entry_id	acc_trans
66	ap_entity_credit_account_fkey	FOREIGN KEY	ap	public	entity_credit_account	entity_credit_account
67	ap_entity_id_fkey	FOREIGN KEY	ap	public	entity_id	entity
68	ap_id_fkey	FOREIGN KEY	ap	public	id	transactions
69	ap_person_id_fkey	FOREIGN KEY	ap	public	person_id	entity_employee
70	ar_entity_credit_account_fkey	FOREIGN KEY	ar	public	entity_credit_account	entity_credit_account
71	ar_entity_id_fkey	FOREIGN KEY	ar	public	entity_id	entity
72	ar_id_fkey	FOREIGN KEY	ar	public	id	transactions

Joonis 24. LedgerSMB andmebaas enne nimede ühtlustamist.

```
3
4  select * from get_all_constraints()
```

Data Output | Explain | Messages | Query History

	constraint_name character varying	constraint_type character varying	table_name character varying	constraint_schema character varying	column_name character varying	references_table character varying
45	ck_location_class_class	CHECK	location_class	public	class	[null]
46	ck_location_line_one	CHECK	location	public	line_one	[null]
47	ck_location_mail_code	CHECK	location	public	mail_code	[null]
48	ck_location_state	CHECK	location	public	state	[null]
49	ck_menu_acl_acl_type	CHECK	menu_acl	public	acl_type	[null]
50	ck_note_class_class	CHECK	note_class	public	class	[null]
51	ck_oe_class_id	CHECK	oe_class	public	id	[null]
52	ck_person_first_name	CHECK	person	public	first_name	[null]
53	ck_person_last_name	CHECK	person	public	last_name	[null]
54	ck_session_token	CHECK	session	public	token	[null]
55	fk_account_account_heading_heading	FOREIGN KEY	account	public	heading	account_heading
56	fk_account_checkpoint_account_account_id	FOREIGN KEY	account_checkpoint	public	account_id	account
57	fk_account_heading_account_heading_parent...	FOREIGN KEY	account_heading	public	parent_id	account_heading
58	fk_account_heading_translation_account_hea...	FOREIGN KEY	account_heading_translation	public	trans_id	account_heading
59	fk_account_link_account_account_id	FOREIGN KEY	account_link	public	account_id	account
60	fk_account_link_account_link_description_des...	FOREIGN KEY	account_link	public	description	account_link_descript...
61	fk_account_translation_account_trans_id	FOREIGN KEY	account_translation	public	trans_id	account
62	fk_acc_trans_account_chart_id	FOREIGN KEY	acc_trans	public	chart_id	account
63	fk_acc_trans_transactions_trans_id	FOREIGN KEY	acc_trans	public	trans_id	transactions
64	fk_acc_trans_voucher_voucher_id	FOREIGN KEY	acc_trans	public	voucher_id	voucher
65	fk_ac_tax_form_acc_trans_entry_id	FOREIGN KEY	ac_tax_form	public	entry_id	acc_trans
66	fk_ap_entity_credit_account_entity_credit_acc...	FOREIGN KEY	ap	public	entity_credit_account	entity_credit_account
67	fk_ap_entity_employee_person_id	FOREIGN KEY	ap	public	person_id	entity_employee
68	fk_ap_entity_entity_id	FOREIGN KEY	ap	public	entity_id	entity
69	fk_ap_transactions_id	FOREIGN KEY	ap	public	id	transactions
70	fk_ar_entity_credit_account_entity_credit_acco...	FOREIGN KEY	ar	public	entity_credit_account	entity_credit_account
71	fk_ar_entity_employee_person_id	FOREIGN KEY	ar	public	person_id	entity_employee

Joonis 25. LedgerSMB andmebaas pärast nimede ühtlustamist.