TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Mõned disainimustrid klassifikaatorite esitamiseks SQL andmebaasides

Bakalaureusetöö

Üliõpilane:	Aleksandra-Salome Vellemaa
Üliõpilaskood:	121060IAPB
Juhendaja:	Erki Eessaar

Tallinn
2015

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.

(*kuupäev*) (*allkiri*)

Annotatsioon

Mõned disainimustrid klassifikaatorite esitamiseks SQL andmebaasides

Töö eesmärgiks on õppida tundma ja kirjeldada mõningaid disainimustreid klassifikaatorite esitamiseks SQL-andmebaasides, disainida ning realiseerida iga mustri põhjal töötav näide ning saadud kogemuste põhjal neid disaine võrrelda.

Klassifikaatorite süsteemi realiseerimiseks on palju erinevaid võimalusi ning ühte ja parimat viisi nende esitamiseks ei ole. Käesolevas töös püütakse analüüsida ning leida, millised on mõnede võimalike disainilahenduste (mida omakorda mustri formaadis kirjeldati) tugevused ja nõrkused.

Tehtud töö tulemuseks valmisid kolme disainilahenduse mustri formaadis kirjeldused, vastava näiteandmebaasi disainid, nende kohta näitepäringud ja andmebaasis kitsenduste jõustamise näited ning järeldused iga disaini kohta.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 54 leheküljel, 8 peatükki, 4 joonist.

Abstract

Some Design Patterns for Representing Reference Data in SQL Databases

The main goal of this work is to study and write some design patterns for representing reference data in SQL databases, create an example database based on each pattern, and compare the designs based on the collected experience.

There are many different solutions to represent reference data in databases. However, there is not one perfect solution. This thesis will try to analyze and find out what are the strengths and weaknesses of some possible design solutions (that we specified in the pattern format).

The result of the work includes specification of three design solutions in the pattern format, designs of corresponding example databases, examples of queries based on the databases as well as examples of enforcing constraints in the database, and conclusions for each design.

The thesis is in Estonian and contains 54 pages of text, 8 chapters, 4. etc.

Lühendite ja mõistete sõnastik

SQL	***Structured Query Language*** Andmebaasikeel, mis põhineb relatsioonilisel andmemudelil ning mida saab kasutada andmete haldamiseks, transaktsioonide juhtimiseks, õiguste haldamiseks ning andmestruktuuride ning neid ümbritsevate muude andmebaasiobjektide haldamiseks. SQL-andmebaasi all nimetatakse antud töös andmebaasi, mis on loodud kasutades andmebaasisüsteemi, milles kasutatakse SQL-i.
UML	***Unified Modeling Language*** Laialdlaselt kasutatav üldotstarbeline visuliseerimiskeel, mida kasutatakse tarkvara ja infosüsteemide struktuuri ning käitumise spetsifitseerimiseks ning visualiseerimiseks. Üldotstarbelisus tähendab, et keelt saab kasutada paljude eritüübiliste süsteemide kirjeldamiseks ning keeles on ka vahendid taolise ülesande lahendamiseks vajalike laienduste lisamiseks.
Alias	***alias*** Alias on pseudonüüm (teine nimi, varjunimi). SQL keeles andmete käitlemise lausete koostamisel kasutatakse aliast tabelitele lause kontekstis uue nime andmiseks, et võimaldada näiteks käsitleda sama tabelit mitme erineva tabelina või lihtsalt vähendada kirjutamist vajava koodi pikkust.
Andmete terviklikkus	***Data integrity*** Andmete terviklikkus tähendab, et andmed on reeglitele vastavad ning andmebaasis pole registreeritud vastuolulisi fakte. Andmete terviklikkuse aste on andmete kvaliteedi näitaja. Andmebaasi mõistes tähendab andmete terviklikkuse kontrolli realiseerimine võimalust ärireeglite jõustamiseks. Äri viib läbi protsesse, protsessidest tekivad jäljed andmete näol, mis registreeritakse andmebaasis. Kui andmete registreerimine lükatakse tagasi, sest andmed eksisid mõne kitsenduse vastu, siis jõuab ka äriprotsessi läbiviijateni teadmine, et nad tegid midagi valesti.

(Techopedia)

CASE ***Computer-Aided Software Engineering***

Arenduskeskkond, mis võimaldab projekteerida, disainida ja luua infosüsteeme ja tarkvara, aitab teha seda kvaliteetselt, vältida vigu. (Википедия)

Meta-andmed ***metadata***

Andmeid kirjeldavad andmed. Käesolevas töös mõistetakse andmebaasis klassifikaatorite metaandemete all muutmise kuupäeva, viimast muutjat, loomise kuupäeva, seisundit jne. (Techtarget)

Jooniste nimekiri

Sisukord

1. Sissejuhatus

Suurem osa tänapäeva infosüsteemidest vajavad andmete hoiustamiseks ja seega ka toimimiseks andmebaasi. Vaatamata sellele, et SQL-andmebaase on loodud 1970-ndate aastate lõpust ning kindlasti on selle ajaga kogunenud palju informatsiooni selle kohta, millised on erinevate disainilahenduste head ning halvad küljed, esineb tänapäeva andmebaasides endiselt halbu, läbimõtlemata ja ebaefektiivseid disainilahendusi. Kuidas hoida andmeid nii, et see oleks kõige ratsionaalsem – andmeid saaks võimalikult lihtsalt sinna sisestada ja ka otsida, samas tagades andmete terviklikkuse ja turvalisuse? Selles töös valiti vaadeldavaks skoobiks klassifikaatorite esitamise SQL-andmebaasides. Selleks on olemas mitmeid võimalusi. Töö sisuks on välja tuua mõned põhilised lahendused ning hinnata nende sobilikkust konkreetse olukorra jaoks. Klassifikaatorid on esindatud peaaegu kõikides andmebaasides, seega on teema aktuaalne senikaua kuni SQL-andmebaasid kasutust leiavad.

1.1 Taust ja probleem

Võimalike lahendusi klassifikaatorite esitamiseks SQL-andmebaasides on palju, kuid struktureeritud võrdlust nende vahel on raske leida. Probleemiks on ka info killustatus, mis pärsib selget arusaama võimalikest lahedustest. Üheks võimaluseks esitada erinevaid disaine selgelt ja struktureeritult on kasutada disainimustreid. See tagab erinevate lahenduste parema võrreldavuse.

Antud töö võib osutuda kasulikuks kõikidele, kes tegelevad andmebaasidega, õpivad andmebaase või tunnevad selle vastu huvi. Isegi kui andmebaas pole SQL-andmebaas, on seal suure tõenäosusega klassifikaatorid kasutusel.

Töös esitatud andmebaasidiagrammid disainitakse CASE-vahendis Rational Rose ning näiteandmebaasid realiseeritakse PostgeSQL 9.3.5 andmebaasisüsteemis.

1.2 Ülesande püstitus

Töö põhieesmärkideks on:

- Tuua välja põhilisemad klassifikaatorite esitamise disainimustrid SQL-andmebaasides

- Realiseerida iga mustri kohta andmebaas näiteandmetega. Kõigi kolme mustri puhul realiseeritakse ühe ja sama kontseptuaalse andmemudeliga andmebaas ja kõikidesse andmebaasidesse lisatakse ühesugused näiteandmed

- Võrrelda disainilahendus andmebaasi loomise ja sealt andmete otsimise keerukuse alusel

1.3 Metoodika

Töö on näide disainiteaduse (*design science*) metoodika rakendamisest (Wikipedia). Selle töö tulemusena valmivad uued artefaktid e tehised (disainimustrid), milles kirjeldatud disainilahendusi katsetatakse konkreetse näiteandmebaasi põhjal. Selleks disainitakse esmalt CASE-vahendis erinevate mustrite põhjal erinevad andmebaasid ning seejärel realiseeritakse need disainid andmebaasisüsteemis.

1.4 Ülevaade tööst

Antud töö koosneb kuuest suuremast sisupeatükist, nendest esimeses peatükis „Klassifikaatorid“ tutvustatakse lühidalt klassifikaatori mõistet. Järgmine peatükk „Mustrid“ räägib mustritest üldisemalt. Minu töös kasutatav mustrite ühtne struktuur on kirjeldatud peatükis „Mustrite struktuur“. Järgnevad peatükid töö põhitulemustega, milleks esmalt on „Mustrite kataloog“, kus esitatakse loodud mustrid. Samas on ka iga mustri kohta näitena andebaasi disaini mudel. Disainitud andmebaaside põhjal päringute tegemist ning andmekontrolli lisamise keerukust võrreldakse peatükis „Mustrite omavaheline võrdlus“. Viimases sisupeatükis „Mustrite võrdlus ja tulemused“ on teostatud mustrite laiem võrdlus ning kirja pandud analüüsi põhjal tehtud järeldused.

2. Klassifikaatorid

„Klassifikaator on täpselt kirjeldatud, üksteist välistavate ning number- või tähtkoodiga tähistatud kategooriate põhjalik ja korrastatud süsteem.“ (Eesti Statistika 2014)

Klassifikaatorit teiste sõnadega selgitades võib öelda, et need on andmed, mis kirjeldavad ja liigitavad infosüsteemides teisi andmeid. Piir klassifikaatorite ja põhiandmete (andmed, mis kirjeldavad transaktsiooniliste andmete konteksti määravaid abstraktseid või füüsilisi "asju") on hägune. Näiteks mõnes süsteemis on *Tootja* klassifikaator ja selle kohta registreeritakse vaid kood ja nimetus, kuid teises kuulub see põhiandmete hulka ja selle kohta talletatakse süsteemis oluliselt rohkem andmeid.

Käesolevas töös tähendab „klassifikaator“ sama mis „klassifikaatori tüüp“ (nt riik, kauba kategooria). Igasse sellisesse tüüpi kuulub üks või mitu klassifikaatori väärtust (nt klassifikaatori tüüpi *Riik* kuulub väärtus „EST“, Eesti).

Rakenduse tavakasutaja ise klassifikaatorite väärtuseid lisada/muuta/kustutada tavaliselt ei saa, selleks on spetsiaalne liides ning vastavate suuremate volitustega kasutaja. Kui igaüks saaks omatahtsi klassifikaatorite väärtuseid välja mõelda ja registreerida oleks tulemuseks anarhia ning hilisemad raskused klassifikaatorite väärtuste alusel andmete otsimisel. Infosüsteemide kasutajad puutuvad klassifikaatoritega näiteks kokku, kui rakenduses tuleb teha valik rippmenüüst, mitme valikuga märkeruutudest või hüpikakendest. Sageli on sellisel viisil valitavad andmed klassifikaatorid (nt kliendi puhul elukoha riik ja emakeel, kaupade puhul nende kategooriad). Sageli saab ühte ja sama klassifikaatorit kasutada andmebaasis erinevates kontekstides. Näiteks kliendi puhul elukoha riik, tarnija puhul äriregistris registreerimise riik, tellimuse kohal riik, kuhu tellimus tuleb kohale viia.

Struktuuri järgi saab klassifikaatorid liigitada hierarhilisteks ja lineaarseteks.

- Lineaarne klassifikaator on ühe tasemega klassifikaator, mis on koodide ja nendele vastavate nimetuste loetelu. Näitena võib tuua riigi koodide ja nende nimede seose: EST – Eesti.

- Hierarhiline klassifikaator on mitme tasemega klassifikaator, mida on võimalik kujutada puustruktuurina. Hierarhiline klassifikaator on näiteks kaupade kategooria, millel omakorda võivad olla alamkategooriad.

Klassifikaatori põhitunnusteks on.

- Kategooriad on korraldatud täpselt, mis võimaldab andmeid koguda ning töödelda.
- Esitatud on kõigi objektide liigitamist võimaldavad kategooriad, see tähendab, et ei leidu objekti, mida ei ole võimalik liigitada. Klassifikaatorisse võib olla lisatud klassifikaatori väärtus, mis ütleb, et kui see objektile määrata, siis pole liigitus teada.
- Klassifikaatori kategooriad on vastastikku välistavad. See tähendab, et iga objekti saab soovi korral liigitada ainult ühte kategooriasse. (Rang 2008)

Klassifikaatoritel on oma elutsükkel. Klassifikaatori väärtus võib hetkel olla näiteks koostamisel, aktiivne või kasutusest väljas. Kui klassifikaatori väärtus on koostamise faasis, siis ei peaks seda veel rakendusest klassifitseerimise eesmärgil valida saama. Kui klassifikaatori väärtus on kasutusest välja läinud (nt riigid „Jugoslaavia“ ja „Nõukogude Liit riikide klassifikaatori korral), siis ei saa seda andmebaasist ära kustutada, sest selle abil on ilmselt teisi andmeid liigitatud.

3. Mustrid

Käesolevas töös esitatakse edaspidi klassifikaatorite SQL-andmebaasides esitamist kirjeldavaid mustreid. Selleks, et neid paremini, mõista tuleks esmalt määratleda mustri olemus ja seda mõistet defineerida.

3.1 Mustri mõiste

Eesti keele seletavas sõnaraamatus on mustri üheks definitsiooniks: „Eeskuju, mall, näidis; millegi läbiv ühine joon, seaduspära vms.“ (EKSS)

Igapäevaelust võib tuua hulgaliselt näiteid ja seoseid mustritega. Rakendades mustri mõistet kunstivaldkonnast saab öelda, et muster on millegi kordus. Seega ka igapäevane tee kodust kooli on kindlas järjekorras tehtavad tegevused ehk mustrid, need põhinevad kordustel. Mustri üks olulisimaid omadusi ongi taaskasutusvõimalus. Luues mingisuguse tegevuse mustri võivad seda järgida paljud inimesed palju kordi.

Milleks kasutada mustreid? Suures osas tegevustest, mida on eelnevalt teiste inimeste poolt mitmeid kordi sooritatud, on kõik võimalused juba läbi proovitud ja parimad kindlaks tehtud. Tundmatut ala iseseisvalt omandades on üsna väike tõenäosus, et avastatakse midagi uut või leitakse uus võimalus probleemide lahendamiseks. Tõenäoliselt jõutakse sama lahenduseni, mis on juba kord välja mõeldud, kogedes enne ka ebaõnnestumisi. Et luua enda jaoks parim lahendus on mõistlik ära kasutada olemasolevaid teadmisi, kuna need hõlmavad mitte ainult parimaid valikuid, vaid ka teadmisi ja kogemusi halbadest lahendustest. Kuna võimalikke valikuid (mustreid) võib olla palju, saab neid analüüsides jõuda parema valikuni, milleni iseseisvalt võib olla ei olekski jõutud.

3.2 Disainimuster

Disainimuster tarkvaraarenduses on eeskiri, mis kirjeldab üldist taaskasutatavat lahendust tihti korduvale tarkvaraarenduse probleemile. Tarkvara arhitektil, arendajal, disaineril või muul probleemi teemasse puutuva rolli esindajal tuleb olemasolevate mustrite seast valida olukorda arvestades parim ning seda konkreetse probleemi lahendamise jaoks kohandada. (Rouse) Seega peab muster olema piisavalt üldine, et seda erinevates olukordades korduvalt

rakendada, kuid samas piisavalt detailne, et selle õpetuste rakendamisest reaalselt kasu oleks. Disainimuster ei ole valmis disaini kirjeldus, see on pigem kirjeldus või õpetus, kuidas probleemi saab lahendada, seejuures erinevate taustadega olukordades. (Wikipedia) Muster pole ühekordne lahendus, vaid korduv lahendus ja selle kasutamise kohta peab olema vähemalt kolm näidet.

Järgnevalt esitatav disainimustri struktuuri on välja toodud raamatust „Design Patterns: Elements of Reusable Object-Oriented Software“ (samuti tuntud ka kui nelikjõugu mustrid). (Gamma et al.1994)

Disainimustri tähtsamad elemendid.

- Nimi
- Probleem, mida see muster lahendab
- Taust, millal nimetatud probleem võib tekkida
- Jõud, mis võivad mõjutada probleemi või selle lahendust
- Lahendus probleemile
- Lahenduse mõistlikkus, näited õnnestunud või ebaõnnestunud projektidest
- Autor ja kuupäev
- Näitekood

Tava kasutada ühiseid mustreid sarnaste probleemide lahendamiseks on pärit arhitekt Christopher Alexanderi hoonete projekteerimisest ja avaldatud teostest. Üks tema raamatutest mis oli kontseptsiooni aluseks on „A Pattern Language: Towns, Buildings Construction“. (Alexander 1977)

SQL-andmebaaside disainiprobleemide kohta on samuti mustreid kirja pandud. Karwin (2010), esitab SQL-andmebaaside disaini antimustreid, mis kirjeldavad halbu lahendusi levinud disainiprobleemidele koos parema lahendusega. Rõzova (2011) kirjeldab disainimustrite formaati kasutades võimalusi üldistuste realiseerimiseks SQL-andmebaasides. Krönström (2015) kirjeldab disainimustrite formaadis võimalusi kuidas esitada SQL-andmebaasides hierarhilisi andmeid.

4. Mustrite struktuur

Edaspidi kirjeldavatele mustrite üle parema ülevaate saamiseks kasutatakse töös ühist struktuuri, mida tutvustatakse käesolevas peatükis.

Mustrite struktuuri aluseks võeti Natalja Rõzova bakalaureusetöö „Disainimustrid üldistusseoste realiseerimiseks SQL-andmebaasides“ (Rõzova, 2011), kus kasutati samasugust struktuuri, mis on igati sobiv ka käesoleva töö jaoks.

- Nimi – peab kirjeldama mustrit võimalikult täpselt.
- * Probleem, mida see muster lahendab on kõikidel disainimustritel sarnane: kuidas hoida klassifikaatoreid andmebaasis nii, et päringud ja andmemuudatused oleksid lihtsad, kuid samas oleks võimalik ka jõustada andmebaasis kitsendusi ning registreerida klassifikaatoritega seotud metaandmeid nagu klassifikaatori väärtuse looja, viimane muutja, viimase muutmise aeg ja hetkeseisund.
- * Taust, millal nimetatud probleem võib tekkida: uue SQL-andmebaasi disainimisel on vajalik leida klassifikaatorite andmebaasis esitamise strateegia.
- Jõud, mis võivad mõjutada probleemi või selle lahendust ehk parima mustri valikut.
- Lahendus probleemile.
- Lahenduse mõistlikkus, näited õnnestunud või ebaõnnestunud projektidest.
- Allikad ja autorid, mis demonstreerivad selle mustri kasutamist ning kelle soovitusel olen mustreid kasutanud.
- Näide. See teostatakse kõikide mustrite korral kasutades ühte põhiobjekti, milleks on *Kaup*. Kaubal on toomise riik, kauba seisundi liik ning kauba kategooria – need on klassifikaatorid. Igal riigil on lisaks nimetusele ja koodile ka pindala, mille registreerimine on kohustuslik. Kauba seisundi liikide korral soovitakse registreerida nende vabatekstilised kirjeldused, kuid osadel seisundi liikidel võivad

need puududa. Kauba kategooria on hierarhiline klassifikaator, iga kategooria võib omada null või rohkem alamkategooriat ning null või üks ülemkategooriat.

* Kuna probleem ja taust on kõikidel mustritel sama, siis seda mustrite struktuuri kirjelduses ei esitata.

5. Mustrite kataloog

Käesolevas peatükis kirjeldatakse kolme disainilahendust mustrite formaadis.

5.1 Iga klassifikaator eraldi tabelis

Mustri nimetus: Iga klassifikaator eraldi tabelis

Inglisekeelne nimi: Separate tables for each reference data type

Jõud:

- Erinevat tüüpi klassifikaatoritel on erinev atribuutide hulk.
- Erinevat tüüpi klassifikaatoritel on erinevat tüüpi ja pikkusega koodid ning erinevad nõudmised koodile vastava nimetuse maksimaalsele pikkusele.
- Soovitakse vältida liigset NULLide (andmete puudumist tähistav marker SQLis) kasutust andmebaasis, sest see võib viia loogiliselt ebakorrektsete päringutulemusteni.
- Soovitakse kontrollida andmete terviklikkust andmebaasi tasemel ning kasutada selleks võimalusel deklaratiivseid kitsendusi.

Lahendus: Igale klassifikaatori tüübile luuakse eraldi baastabel (tabel). Ei looda ühist tabelit, kus oleks koos andmed kõikide klassifikaatorite kohta. Igas klassifikaatorite tabelis on need ja ainult need veerud, mis on vajalikud sellele konkreetsele klassifikaatori tüübile vajalike andmete registreerimiseks. Klassifikaatori tabelitele viitavad välisvõtmed on ON UPDATE CASCADE kompenseeriva tegevusega, et reageerida klassifikaatorite koodide parandamisele. Samas tuleb koodide parandamisega olla väga ettevaatlik, sest koode võib juba olla kasutatud päringutes ja vaikimisi väärtustena.

Positiivsed aspektid:

- Iga klassifikaatori koodi jaoks saab valida kõige sobivama andmetüübi ja hoida seda koodi vastavat tüüpi veerus.

- Tabelile saab lisada CHECK-kitsendusi, mis kehtivad ainult konkreetse klassifikaatori kohta. Mustri kohta realiseeritud näidetes lisas autor täiendava kontrolli tabelile „Riik“, millega kontrollitakse, kas sisestatav riigi kood koosneb täpselt kolmest tähest.

- Klassifikaatori kood (kasutajate jaoks tähendusega väärtus) on selles tabelis primaarvõti ning läheb välisvõtmetena tabelitesse, mis sellele klassifikaatorile viitavad. Järelikult omavad vastavad välisvõtme väärtused kasutajatele tähendust, mis lihtsustab päringute kirjutamist. Klassifikaatori koodid (näiteks riikide kolmetähelised koodid) võivad olla kasutajale juba nii tuttavad, et ei vaja koodidele vastavate nimetuste esitamist. See omakorda tähendab, et peab kirjutama natukene vähem päringuid, kus on vaja tabeleid ühendada (*join*). Olukorras, kus SQL süsteemidele heidetakse ette ühendamisoperatsioonide aeglust, omab see ühtlasi ka mõju süsteemi jõudlusele.

- Kuna klassifikaatorite koodid on fikseeritud ja dokumenteeritud, siis saab olla kindel selles, et need ei muutu ja tänu sellele kirjutada lihtsamaid päringuid, mis ei ühenda klassifitseeritavate andmetega tabelit klassifikaatori tabeliga.

- Kuna ühte tüüpi klassifikaatorite andmed on koos ühes tabelis saab nende leidmiseks kirjutada päringu ühe tabeli põhjal (lihtsam kirjutada ja süsteemil kiirem täita, sest ei pea lugema üleliigseid andmeid).

- Klassifikaatorite koode saab kasutada vastavatel välisvõtme veergudel vaikimisi väärtustena.

- Kuna erinevat tüüpi klassifikaatoritel on erinevad atribuudid, võimaldab see klassifikaatorite tabelistes esitada ainult vajalikku hulka veerge. Selline lahendus välistab olukorra, kus tabelisse tekib hulgaliselt NULL-e.

- Kui klassifikaatori tabelis on lisaks koodile ja nimetusele täiendavaid veerge, millest mõnele võiks lisada vaikimisi väärtuse, siis on vaikimisi väärtust lihtne lisada.

- Igale kasutajale/rollile saab anda õigused kasutada vaid nende tööks vajalikke klassifikaatorite andmeid.

- Kui ühes klassifikaatorite tabelis andmed riknevad või mingil põhjusel lukustatakse, siis ei põhjusta see ilmselt süsteemi töö globaalselt halvamist. Mõju jääb lokaalseks.

- Kuna igas klassifikaatori tabelis eraldi võttes on ridade arv suhteliselt väike, siis kulutaks andmebaasisüsteem selle tervenisti muutmälus hoidmisele vähem ressursse.

Negatiivsed aspektid:

- Uue klassifikaatori tüübi lisamiseks andmebaasi on vajalik uuendada andmebaasi struktuuri, see tähendab uute tabelite ja seoste loomist.

- Suure hulga klassifikaatorite puhul tuleb teha palju tööd andebaasi loomisel, samuti tekib palju erinevaid klassifikaatorite tabeleid. Kui muutuvad üldised põhimõtted klassifikaatorite haldusel (näiteks on vaja hakata registreerima viimast muutmise kuupäeva või muuta kõigi klassifikaatorite puhul kirjelduse veeru andmetüüpi), siis tuleb teha muudatusi paljudes tabelites.

- Erinevad metaandmete veerge tuleb lisada kõikidesse klassifikaatorite tabelitesse. Näites on selliseks veeruks „muutja“, mis tähistab viimast kasutajat, kes on konkreetses reas andmeid muutnud.

- Universaalset rakendust, mis kõiki neid tabeleid haldab, on keeruline luua. Rakendus, kus iga tabeli jaoks on sisseehitatud eraldi ekraanivorm/haldamise leht tuleb klassifikaatorite struktuuri muutumisel pidevalt ümber teha.

Allikad ja autorid, mis demonstreerivad selle mustri kasutamist ning kelle soovitusel on mustreid kasutatud:

- „Five Simple Database Design Errors You Should Avoid“ (Sen 2009)

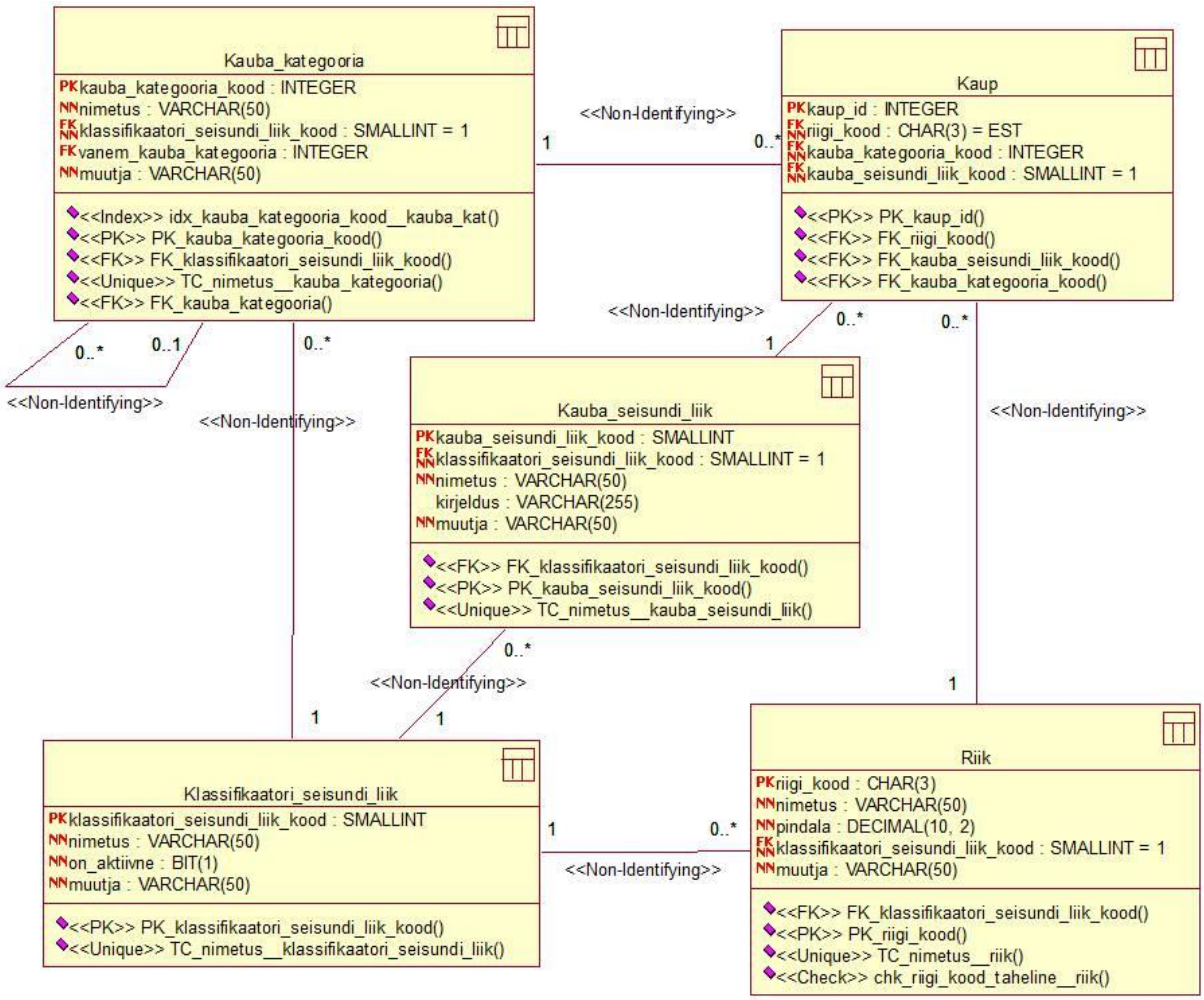
- „Look-up Tables in SQL“ (Celko 2011)

- „OTLT and EAV: the two big design mistakes all beginners make“ (Andrews 2004)

- „Designing for Performance: Lookup Tables“ (Poolet 2006)

Näide

Nagu Joonis 1-lt võib näha, on igale klassifikaatori tüübile loodud eraldi tabel, tabel põhiobjekti (*Kaup*) jaoks ning tabel, kus hoitakse klassifikaatorite seisundite liike. Tabelil *Kauba_kategooria* on välisvõtme kitsendus, mis viitab selles samas tabelis olevale primaarvõtmele. (Mandelstein et al. 2013) Tegemist on külgnevusnimistu (*adjacency list*) disainiga hierarhiliste andmete esitamiseks. (Krönström 2015) Veerud metaandmete registreerimiseks on vaja lisada kõikidesse tabelitesse, ning klassifikaatorite seisundite liikide jaoks on eraldi tabel. Veerg muutja on hetkel vabatekstilise sisestuse veerg ning võib sisaldada näiteks muudatuse teinud andmebaasi kasutaja nimi. Praktikas võib see veerg olla ka välisvõtme veerg, mis viitab näiteks tabelile *Töötaja*. Samas kui soovida andmebaasis sellist muudatust teha, on vaja muuta palju tabeleid.



Joonis 1. Andmebaasi diagramm disainimustrile „Iga klassifikaator eraldi tabelis“.

5.2 Kõik klassifikaatorid ühises tabelis

Mustri nimetus: Kõik klassifikaatorid ühises tabelis

Inglisekeelne nimetus: Common lookup table

Jõud:

- Erinevate klassifikaatorite suurem osa atribuute on sarnased, mis eraldi tabelite puhul tähendaks sarnaseid välju mitmetes tabelites.
- Arvatakse, et üks ühine tabel on lihtsam ja universaalsem lahendus.
- Suur hulk klassifikaatorite tüüpe, iga jaoks luua eraldi tabel ei tundu otstarbekas.
- Vajadus registreerida klassifikaatorite metaandmeid. Selle võimaluse lisamine eraldi igasse klassifikaatori tabelisse tundub keeruline ja tülikas.

Lahendus probleemile: luuakse üks klassifikaatorite tabel, kus on kõikide klassifikaatori tüüpide vajalikud atribuudid ning kuhu lisatakse kõik klassifikaatorid.

Positiivsed aspektid:

- Arvatakse, et lahenduse realiseerimine andmebaasi loomisel on kiirem.
- Uue klassifikaatori tüübi lisandumisel ei ole vaja luua andmebaasi uut tabelit, vaid saab lihtsalt lisada uue rea olemasolevasse tabelisse.
- Suure klassifikaatori tüüpide hulga puhul ei teki andmebaasi suurel hulgal klassifikaatorite tabeleid.
- Klassifikaatorite metaandmed on seotud ühe tabeliga.
- Lihtsam luua klassifikaatorite haldamiseks mõeldud rakendust, sest see peab töötama väikese hulga tabelitega, mille hulk ajas ei muutu.

Negatiivsed aspektid:

- Kuna erinevad klassifikaatorid nõuavad erinevaid atribuute, siis tabelisse tekib palju veerge, kuid enamus välju on tühjad (ehk tekib palju NULL-e). Tabel seisab pooltühjana.

- Klassifikaatorite koodid ei pruugi olla neile kõige sobivamat tüüpi veergudes.

- Raske või võimatu lisada igale klassifikaatorile eraldi deklaratiivseid kitsendusi (sõltub andmebaasisüsteemi võimalustest). Sageli on ainukene viis, kuidas klassifikaatori tüübi põhiselt sisestatavaid andmeid kontrollida, luua andmebaasi trigerid (protseduurne terviklikkuse tagamise vahend), mis on oluliselt aeganõudvam ja tehniliselt raskem lahendus kui deklaratiivse CHECK kitsenduse lisamine.

- Klassifikaatoritele viitavatele välisvõtmetele ei ole mõistlik lisada vaikimisi väärtuseid, kuna neid ei ole võimalik lisada klassifikaatori tüübi põhiselt. Välisvõtme väärtuseks on klassifikaatori väärtuse globaalne unikaalne identifikaator (unikaalne üle kõiki tüüpi klassifikaatorite), mis võib arenduse käigus muutuda ja mis lõppkasutajale midagi ei ütle.

- Klassifikaatoritele viitavatele välisvõtmete väärtused ei ole sisulise tähendusega. Seega peab sellele lõppkasutaja jaoks arusaadavat vastet alati otsima üldisest klassifikaatorite tabelist. Autori koostatud näites vastab Eesti Vabariigile klassifikaatori id „8“, seevastu eelmises näites vastas kood „EST“.

- Rakenduses peab olema väga tähelepanelik, et mitte määrata välisvõtme väärtuseks viidet mingisse teise klassifikaatori tüüpi kuuluvale klassifikaatori väärtusele – näiteks kauba päritoluriigi registreerimisel luuakse rakenduse ebakorrektse tegevuse tulemusena hoopis seos mingisuguse kauba kategooriaga.

- Päringud on keerukad. Suureneb ühendamisoperatsioone nõudvate päringute hulk. Sellel võib olla negatiivne mõju jõudlusele.

- Kui erineva andmetüübiga koodid on erinevates veergudes, siis on keeruline klassifikaatori koodide unikaalsuse tagamine klassifikaatori tüübi piires. Klassifikaatori nimetuse unikaalsuse tagamiseks sama tüübi piires tuleb üldises klassifikaatorite tabelis deklareerida unikaalseks kombinatsioon klassifikaatori tüübi ja nimetuse veerust. Lahenduseks oleks panna koodid kõik ühte kohustuslikku VARCHAR tüüpi veergu ning deklareerida kitsendus, et klassifikaatori tüübi ja koodi kombinatsioon peab olema unikaalne.

- Kui erineva andmetüübiga koodid on erinevates veergudes, siis tuleb luua triger id tagamaks, et iga klassifikaatori väärtuse korral registreeritakse kindlasti kood ja see on just vastavale klassifikaatori tüübile sobivat andmetüüpi. Lahenduseks oleks panna koodid kõik ühte kohustuslikku VARCHAR tüüpi veergu.

- Kui teatud tüüpi klassifikaatorite puhul on ette nähtud teatud atribuutide väärtuste registreerimine ja teatud puhul mitte, siis tuleb nende reeglite täidetust jällegi trigeritega (või rakenduses) kontrollida.

- Kuigi ei pea looma uut tabelit uue klassifikaatori tüübi lisandumisel on siiski tõenäoline et süsteemi arengu käigus tuleb lisada olemasolevasse tabelisse uusi veerge ning seoseid.

- Tabelis on suhteliselt palju ridu.

- Klassifikaatori tabeli riknemine või lukustamine tähendab süsteemi tööle globaalset katkestust.

- Keeruline jagada õiguseid klassifikaatorite tüüpide põhiselt (nõuaks näiteks vaadete loomist).

Allikad ja autorid, mis demonstreerivad selle mustri kasutamist ning kelle soovitusel olen mustreid kasutanud:

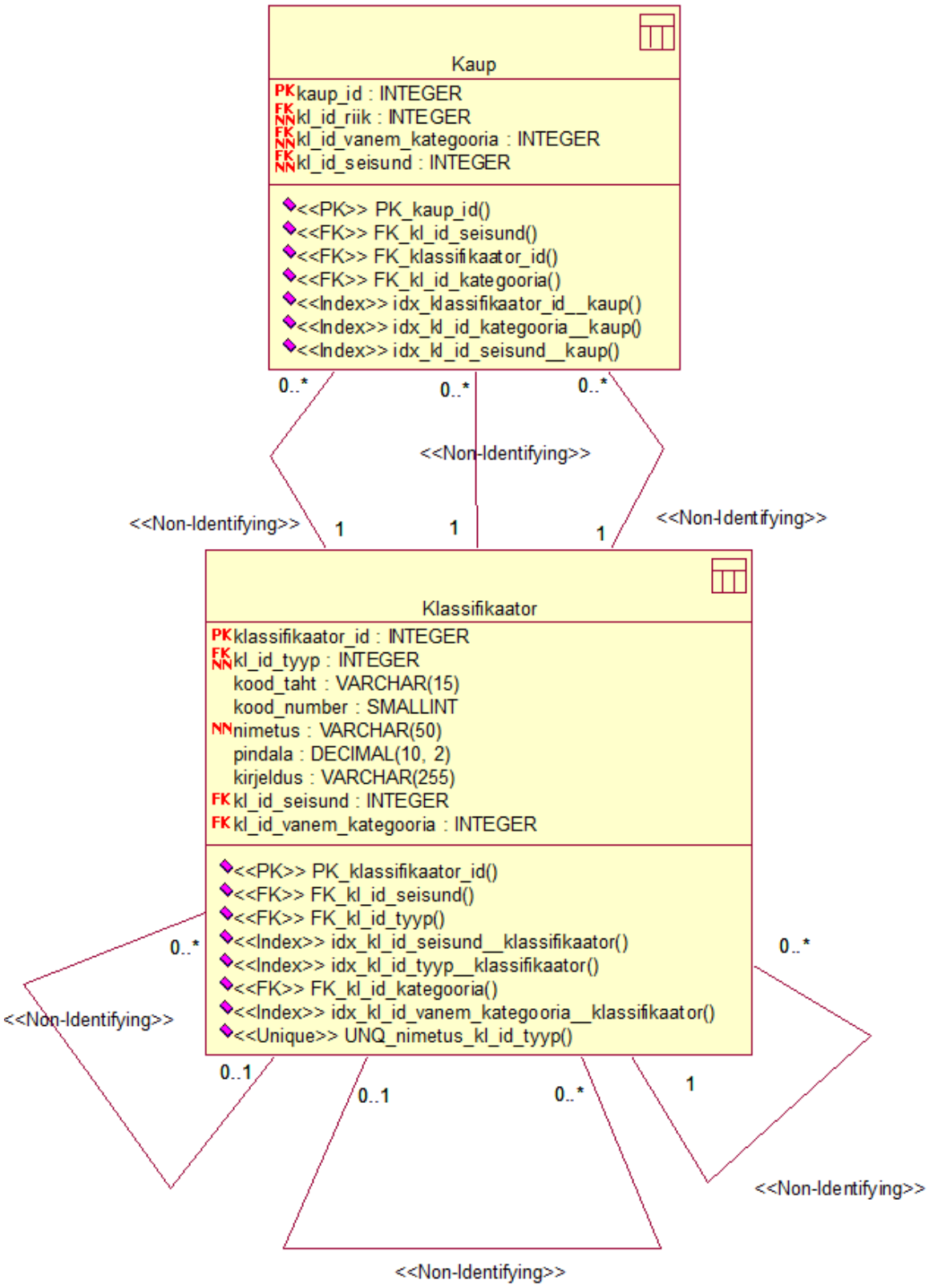
- „OTLT/EAV design - why do people hate it?“ (Mike 2007)

- «Концепция построения системы «Информационный фонд»» (Боженов 2004)

Näide:

Joonis 2 on näha, et tabeleid on vähem, kui eelmises mustris (iga klassifikaator eraldi tabelis). Keerulisi seoseid on seevastu rohkem. Tabelis „Klassifikaator“ on kolm välisvõtit, mis viitavad selles samas tabelis olevatele veergudele – klassifikaatori tüübile, seisundile ning võimalikule ülemkategooriale. Päringute tegemisel kasvatab see oluliselt keerukust. Lähemalt uuritakse seda lihtsamate näidete põhjal järgmises peatükis. Andmebaasi disainimisel abimaterjalidena olid kasutatud Wikipedia lehel avaldatud artikleid: „Hierarchical database model“ ja „Hierarchical and recursive queries in SQL“. (Wikipedia) Antud tabeli puhul

rakendatakse mitmekordselt külgnevusnimistu disaini hierarhiliste andmete esitamiseks. (Krönström 2015)



Joonis 2. Andembaasi diagramm disainimustrile „Kõik klassifikaatorid ühes tabelis“.

Joonis 332 on näha, et sellise tabeli disaini puhul seisab suur osa lahtritest tühjana (NULL), mis on andmebaasi mõistetavuse ja terviklikkuse tagamise seisukohast halb. Veerg *kl_id_tyyp* viitab sellisele id-le, mis tähistaks mõnda klassifikaatori tüüpi (näiteks riik, kauba seisundi liik, kauba kategooria), omakorda nende tüübiks on „klassifikaator“, mis on ülemtüüp. Selle puhul on tehtud rekursiivne lahendus, kus klassifikaatori tüüp on omakorda tema ise. Selline lahendus on vajalik vältimaks klassifikaatori tüübi muutmist mittekohustuslikuks väljaks. *klassifikaator_id* väärtus on süsteemi poolt genereeritud.

Joonis 332 illustreerib klassifikaatori tabelit sisestatud andmetega. Võib näha, et suur osa tabeli lahtritest ei ole täidetud. Kuna erinevatel klassifikaatori tüüpidel on erinevad atribuudid, siis igal tüübil on täidetud vaid osa lahtritest. Eelmise mustri lahenduses on väärtuse puudumine lubatud ainult ühes veerus: „kirjeldus“.

klassifikaator_id	kl_id_tyyp	kood_taht	kood_number	nimetus	pindala	kirjeldus	kl_id_seisund	kl_id_vanem_kategooria
1	7	NULL	NULL	seisundi liik	NULL	NULL	2	NULL
2	1	NULL	1	loomisel	NULL	NULL	3	NULL
3	1	NULL	2	aktiivne	NULL	NULL	3	NULL
4	1	NULL	3	mitteaktiivne	NULL	NULL	3	NULL
5	1	NULL	4	kustutatud	NULL	NULL	3	NULL
6	7	NULL	NULL	riik	NULL	NULL	3	NULL
11	7	NULL	NULL	kauba_kategooria	NULL	NULL	3	NULL
12	11	NULL	100	Toiduaine	NULL	NULL	3	NULL
13	11	NULL	110	pagaritoode	NULL	NULL	3	12
14	11	NULL	120	piimatoode	NULL	NULL	3	12
15	11	NULL	200	riided	NULL	NULL	3	NULL
16	11	NULL	210	spordiriided	NULL	NULL	3	15
18	7	NULL	NULL	kauba seisundi liik	NULL	NULL	3	NULL
19	18	NULL	NULL	laos	NULL	NULL	3	NULL
20	18	NULL	NULL	tarnimisel	NULL	NULL	3	NULL
7	7	NULL	NULL	klassifikaator	NULL	NULL	3	NULL
8	6	EST	233	Eesti Vabariik	45227.00		3	NULL
10	6	USA	840	Ameerika Ühendriigid	9826675.00	NULL	3	NULL
9	6	SUN	810	Nõukogude Sotsialistlike Vabariikide Liit	22402200.00	NULL	4	NULL

Joonis 3. Andmebaasitabel „Klassifikaator“ andmetega.

Erinevat tüüpi klassifikaatori atribuutide esitamiseks on võimalikud variatsioonid. Näiteks võib luua eraldi tabeli *Klasifikaatori_atribuut*, kus on viide olemile (konkreetsele klassifikaatori väärtusele), atribuudi nimi ja atribuudi väärtus. Karwin (2010) kirjeldab sellist lahendust kui antimustrit (halb lahendus probleemile) nimega „Entity-Attribute-Value“.

5.3 Kunstlik ühendaja

Mustri nimetus: Kunstlik ühendaja

Inglisekeelne nimetus: Artificial uniter

Jõud:

- Klassifikaatoritele on vajadus lisada hulgaliselt metaandmeid.
- Klassifikaatorite tüüpidel on erinevad atribuudid.
- On vajadus jõustada palju kitsendusi lähtudes klassifikaatori tüübist. Eelistatakse deklaratiivset kitsenduste jõustamist protseduursele.
- Soovitakse vähendada NULLide kasutamist puuduvate andmete esitamiseks.

Lahendus probleemile: lahenduseks on hübriid mustritest „Iga klassifikaator eraldi tabelis“ ja „Kõik klassifikaatorid ühises tabelis“, mis põhineb üldistuste SQL-andmebaaside esitamise mustril „Lisatabel Kunstlik ülaklass“ (Rõzova 2011). Antud juhul luuakse igale klassifikaatorile eraldi tabeli, mis on seostud välisvõtme kaudu ühise kunstliku klassifikaatorite tabeliga. Klassifikaatorite tüübipõhistes tabelites hoitakse igale tüübile spetsiifilisi andmeid, sh koodid ja nimetused. Metaandmeid (nt viimane muutja, muutmiskuupäev, loomiskuupäev jne) säilitatakse ühises tabelis „Klassifikaator“, mis kaotab vajaduse neid veerge erinevatesse tabelitesse lisada. Kui klassifikaatoritel on harva kasutatavaid ühiseid andmeid (nt kirjeldus), siis võib samuti kaaluda nende sellesse tabelisse paigutamist. Tüübipõhistes tabelites on tabelile *Klassifikaator* viitav veerg unikaalsuse kitsendusega, kuid pole primaarvõti.

Positiivsed aspektid:

- Metaandmete veerge ei pea looma kõikidesse klassifikaatorite tabelitesse. See lihtsustab ka andmebaasi edasiarendamist (evolutsioneerimist) ja refaktoreerimist metaandmete registreerimise osas. Kui peale tabelite loomist on vaja lisada või muuta mingit veergu, siis piisab selle tegemisest ainult ühes tabelis.
- Säilitab mustriga „Iga klassifikaator eraldi tabelis“ tagatud eelised nagu näiteks: Klassifikaatorite tabelitele viitavatele välisvõtme veergudele on võimalik lisada

vaikimisi väärtused. Erinevatele klassifikaatoritele on kerge jõustada erinevaid kitsendusi. Päringute koostamine on suhteliselt lihtne. Klassifikaatori koodid välisvõtmetena on kasutaja jaoks sisulise tähendusega.

- Kui rikneb või lukustatakse tabel *Klassifikaator*, siis muutuvad kättesaamatuks metaandmed, kuid klassifikaatori tüüpide spetsiifilised andmed on endiselt kättesaadavad.

Negatiivsed aspektid:

- Iga kord uue rea lisamisel ükskõik millisesse klassifikaatori tüübi tabelisse on enne vaja lisada rida ka tabelisse *Klassifikaator*, kuna selle primaarvõti on klassifikaatorite tabelites välisvõti.

- Selleks, et saada teada klassifikaatori väärtuse käesoleva hetke seisund (nt selleks, et rakenduses kuvada vaid aktiivseid klassifikaatori väärtuseid), tuleb teha päring kahe tabeli põhjal.

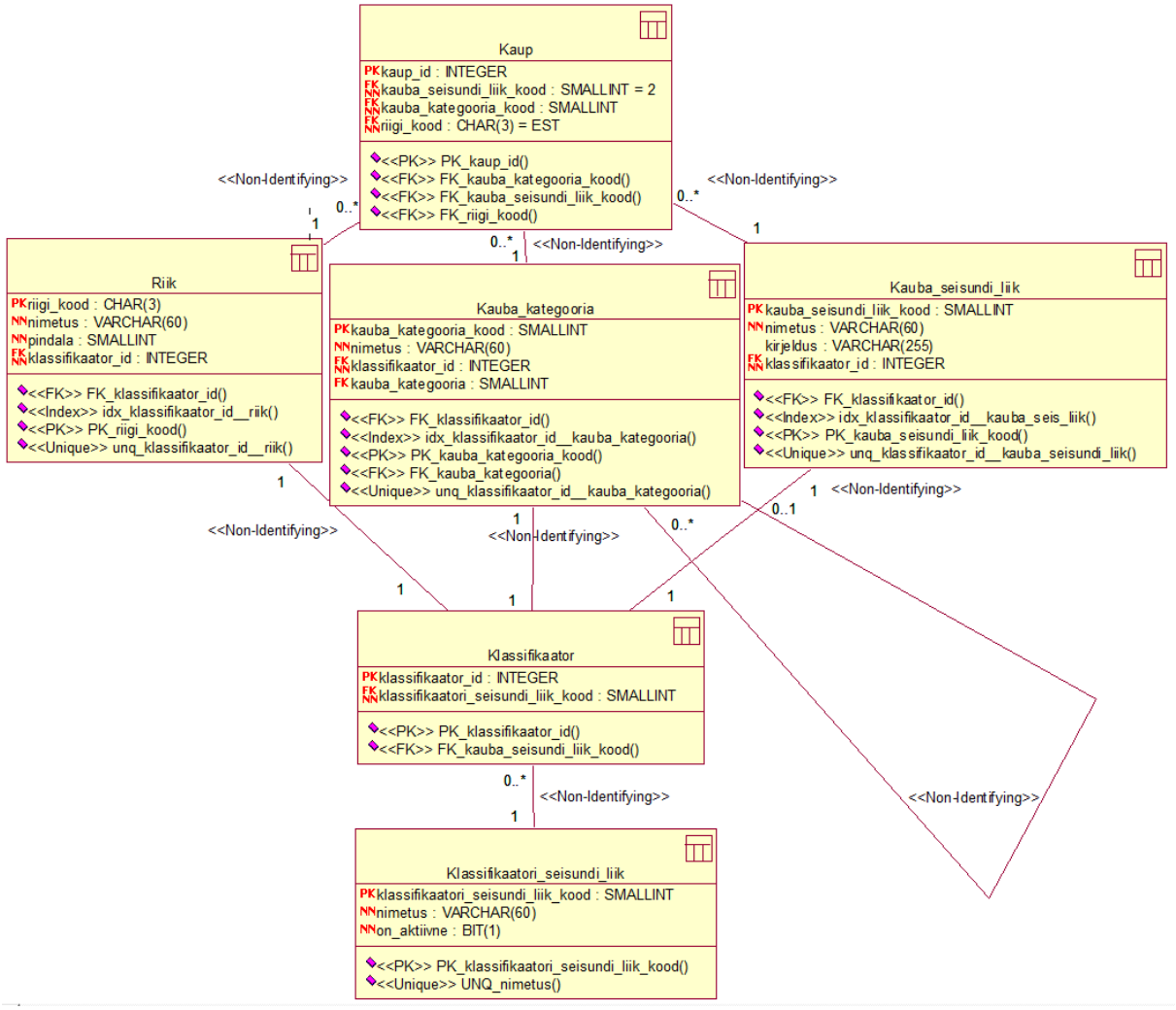
Allikad ja autorid, mis demonstreerivad selle mustri kasutamist ning kelle soovitusel olen mustreid kasutanud:

- *Lisatabel Kunstlik ülaklass* (Rõzova, 2011)

Kuna leitud näiteid on vähe, siis tuleb seda mustrit hetkel pidada mustri kandidaadiks, mille elujõulisust tuleb edasiste praktiliste uuringutega tõestada.

Näide:

Selle lahenduse puhul võib märgata suurt sarnasust „Iga klassifikaator eraldi tabelis“ disainiga. Erinevus seisneb vähemalt ühes lisatabelis. Siiski mängib see suurt rolli. Lisandunud on tabel *Klassifikaator*, mis hetkel määrab ainult klassifikaatori seisundite liigi, kuid lisades üldisesse tabelisse uusi metaandmete veerge, saab hõlpsasti uuendada kõikide klassifikaatorite struktuuri.



Joonis 4. Andmebaasidiagramm disainimustrile „Kunstlik ühendaja“.

5.4 Disainimise võrdlus ja tulemused

Disainimise käigus suuremaid takistusi ühegi mustri realiseerimisel ei esinenud. Siiski peab tõdema, et kõige intuitiivsem ja loomulikum oli „Iga klassifikaator eraldi tabelis“, millele järgnes „Kunstlik ühendaja“, mis oli eelmisele sarnane, kuid olulise täiendusega. Kõige ebaloomulikum, seega ka kasutamiseks ebamugavam oli disainida andmebaasi mustri „ Kõik klassifikaatorid ühises tabelis“ järgi. Selle põhjuseks on, et klassifikaatori tabel sai endale veerud, mis enamuste klassifikaatori tüüpide puhul ei lähe kunagi kasutusse, näiteks veerg „pindala“. Samuti on põhjuseks tabeli välisvõtmed, mille hulk võib kasvada koos klassifikaatori tüüpide arvu suurenemisega.

6. Mustrite omavaheline võrdlus

Selles peatükis võrreldakse omavahel kolme disainilahendust, mis eelnevas peatükis mustri formaadis kirja pandi.

6.1 Päringute keerukuse võrdlus

Selles jaotises võrreldakse erinevate disainimustrite realisatsioonide põhjal päringute keerukust. Selleks püütakse saada erinevate disainide alusel loodud andmebaasidest kätte samad andmed võimalikult sarnasel kujul.

Andmebaasi diagrammide põhjal lõin PostgreSQL andmebaasisüsteemis kolm andmebaasi ning lisasin sinna ühesugused testandmed. Tabelite loomise lauseid selles peatükis eraldi ei käsitleta, need võib leida antud dokumendi lõpust, lisadest (vt Lisa 1). Päringute vastused erinevate disainide puhul on samad.

Põhilised sisestatud testandmed.

- Erinevad riigid, kusjuures vähemalt üks nendest enam ei eksisteeri, et saaks kontrollida võimalust ärireeglite realiseerimiseks. Riikide andmed lisati vastavalt rahvusvahelisele riikide ja territooriumite klassifikaatorile (International Standard Codes for the Representation of the Names of Countries (ISO 3166)), mis on Eesti Statistika lehel eestindatud (Riikide ja territooriumide klassifikaator 2013v1).

- Erinevad kauba kategooriad ning nende alamkategooriad (nt kategooria on toiduaine ning alamkategooriad on piima- ja pagaritooted).

- Kauba seisundi liigid.

Järgnevalt esitatakse neli näidispäringut, mis realiseeritakse iga disaini kohta. Päringu lähtetekst on kirjas iga punkti alguses. Osade päringute kirjutamisel toetuti Codeproject kodulehel avaldatud artiklile: „SQL queries to manage hierarchical or parent-child relational rows in SQL Server“. (*DiponRoy* 2014)

6.1.1 Päringute võrdlus

1. Leidke iga kauba kohta kauba id ja selle riigi kood, kus kaup toodeti.

   Järgnevate päringute tulemuseks on tabel:

kaup_id	riigi_kood
1	EST
2	USA
3	USR

   a. „Iga klassifikaator eraldi tabelis“:

```
SELECT kaup_id, riigi_kood FROM kaup;
```

   b. „Kõik klassifikaatorid ühises tabelis“:

```
SELECT k.kaup_id, kl.kood_taht FROM Kaup k
INNER JOIN Klassifikaator kl ON k.kl_id_riik =
kl.klassifikaator_id;
```

   c. „Kunstlik ühendaja“:

```
SELECT kaup_id, riigi_kood FROM Kaup;
```

   Päringuid vaadates on näha ka visuaalselt, et andmete kätte saamiseks „Kõik klassifikaatorid ühises tabelis“ korral oli vaja rohkem koodi kirjutada ning ühendada omavahel kaks tabelit, seevastu pärides andmeid „Iga klassifikaator eraldi tabelis“ või „Kunstlik ühendaja“ andmebaasidest, saadi andmed kätte ühest tabelist.

2. Leidke iga kauba kohta kauba id ja selle riigi kood ning nimetus, kus kaup toodeti.

   Järgnevate päringute tulemuseks on tabel:

kaup_id	riigi_kood	nimetus
1	EST	Eesti
2	USA	Ameerika Ühendriigid
3	USR	Nõukogude Sotsialistlike Vabariikide Liit

a. „Iga klassifikaator eraldi tabelis“:

```
SELECT k.kaup_id, r.riigi_kood,  r.nimetus FROM kaup k
INNER JOIN Riik r ON k.riigi_kood = r.riigi_kood;
```

b. „Kõik klassifikaatorid ühises tabelis“:

```
SELECT k.kaup_id, kl.kood_taht, kl.nimetus FROM Kaup k
INNER JOIN Klassifikaator kl ON k.kl_id_riik =
kl.klassifikaator_id;
```

c. „Kunstlik ühendaja“:

```
SELECT k.kaup_id, r.riigi_kood, r.nimetus FROM kaup k
INNER JOIN Riik r ON k.riigi_kood = r.riigi_kood;
```

Päringute keerukus on hetkel sarnane. „Iga klassifikaator eraldi tabelis“ ja „Kunstlik ühendaja“ andmebaasidest andmeid küsides tuleb nüüd samuti küsida andmeid kahest tabelist.

3. Leidke aktiivsete riikide koodid ja nimed.

Järgnevate päringute tulemuseks on tabel:

riigi_kood	nimetus
EST	Eesti
USA	Ameerika Ühendriigid

a. „Iga klassifikaator eraldi tabelis“:

```
SELECT r.riigi_kood, r.nimetus FROM Riik r
WHERE r.klassifikaatori_seisundi_liik_kood = 2;
```

b. „Kõik klassifikaatorid ühises tabelis“:

```
SELECT kl.kood_taht, kl.nimetus FROM Klassifikaator kl
INNER JOIN Klassifikaator kl_seisund ON
kl_seisund.klassifikaator_id = kl.kl_id_seisund
INNER JOIN Klassifikaator kl_riik ON kl.kl_id_tyyp =
kl_riik.klassifikaator_id
WHERE kl_seisund.nimetus = 'aktiivne'
AND kl_riik.nimetus = 'riik';
```

c. „Kunstlik ühendaja“:

```
SELECT r.riigi_kood, r.nimetus FROM Riik r
INNER JOIN Klassifikaator k ON r.klassifikaator_id =
k.klassifikaator_id
WHERE k.klassifikaatori_seisundi_liik_kood = 2;
```

Selle näite puhul on selgelt näha, et päring „Kõik klassifikaatorid ühises tabelis“ andmebaasi on keerukam ja pikem kui ülejäänud. Esimeses päringus küsin selliseid riike, mille klassifikaatori seisundi liik on 2, sest fikseerisin, et seisund liigi klassifikaator koodiga „2“ tähistab „aktiivset“ klassifikaatori väärtus (näide sisulise tähendusega koodist). Teises päringus ma seda võimalust kasutada ei saa, kuna id võib olla ükskõik millise väärtusega (*klassifikaator_id* genereerib süsteem). Seega läheb päring pikemaks ja keerukamaks.

4. Leidke iga kauba kohta selle id, kategooria kood, kategooria nimetus ning selle kategooria vahetu ülemkategooria nimetus.

   Järgnevate päringute tulemuseks on tabel:

kaup_id	kauba_kategooria_kood	kauba_kategooria	kauba_ylemkategooria
1	100	Toiduaine	*NULL*
2	110	pagaritoode	Toiduaine
3	210	Spordiriided	riided

a. „Iga klassifikaator eraldi tabelis“:

```
SELECT k.kaup_id, kk_child.kauba_kategooria_kood,
kk_child.nimetus AS kauba_kategooria, kk_parent.nimetus AS
kauba_ylemkategooria FROM Kauba_kategooria kk_child
LEFT JOIN Kauba_kategooria kk_parent ON
kk_child.vanem_kauba_kategooria =
kk_parent.kauba_kategooria_kood
INNER JOIN Kaup k ON kk_child.kauba_kategooria_kood =
k.kauba_kategooria_kood;
```

b. „Kõik klassifikaatorid ühises tabelis“:

```
SELECT k.kaup_id, kl.kood_number, kl.nimetus AS
kauba_kategooria, vanem_kl.nimetus AS kauba_ylemkategooria
FROM Kaup k
INNER JOIN Klassifikaator kl ON k.kl_id_kategooria =
kl.klassifikaator_id
LEFT JOIN Klassifikaator vanem_kl ON
kl.kl_id_vanem_kategooria = vanem_kl.klassifikaator_id
order by k.kaup_id
```

c. „Kunstlik ühendaja“:

```
SELECT k.kaup_id, kk_child.kauba_kategooria_kood,
kk_child.nimetus AS kauba_kategooria, kk_parent.nimetus AS
kauba_ylemkategooria FROM Kauba_kategooria kk_child
LEFT JOIN Kauba_kategooria kk_parent ON
kk_child.kauba_kategooria = kk_parent.kauba_kategooria_kood
INNER JOIN Kaup k ON kk_child.kauba_kategooria_kood =
k.kauba_kategooria_kood;
```

Antud lähteülesande puhul on kõigi kolme päringu struktuur sarnane, kuna kõikides ühendatakse tabelid iseendaga, ehk võrreldakse tabeli veerus olevaid andmeid sama tabeli teises veerus olevate andmetega. Teine ühendamine on tabeliga *Kaup*. Võib öelda, et raskusastmed päringute tegemisel on samad.

6.1.2 Vahetulemused

Autor on võtnud näideteks suhteliselt lihtsad päringud, kuid juba nende põhjal oli näha, et erinev andmebaasi disain tekitab erinevuse ka päringutes ning ka nende keerukuses. Osutus, et ühe kindla klassifikaatorite disainimustri põhjal realiseeritud andmebaasis („Kõik klassifikaatorid ühises tabelis“) on päringud alati kas sama keerukad või keerukamad võrreldes teiste disainidega („Iga klassifikaator eraldi tabelis“ ja „Kunstlik ühendaja“) andmebaasidest info otsimisega. Kordagi ei tulnud ette olukorda, kus „Kõik klassifikaatorid ühises tabelis“ andmebaasist oleks päring hoopis kergem. See on oluline tähelepanek, kuna „Kõik klassifikaatorid ühises tabelis“ tuntakse kui lihtsamat ja vähem tööd nõudvamat lahendust. Kuna andmebaasist päringute tegemine on andmebaasi lahutamatu osa, siis peab seda kindlasti silmas pidama.

Võrreldes omavahel „Iga klassifikaator eraldi tabelis“ ja „Kunstlik ühendaja“ disaine, siis enamasti olid päringud sarnase keerukusega, välja arvatud olukord, kui tuli arvestada klassifikaatori seisundi liigiga (või muude metaandmetega). Sellisel juhul sain „Iga klassifikaator eraldi tabelis“ korral need kätte vahetul sellest klassifikaatori tabelist, mille kohta need käisid. „Kunstlik ühendaja“ puhul pidi aga neid küsima ühisest klassifikaatorite tabelist.

6.2 Kitsenduste jõustamise võrdlus

Ülesandeks on tagada, et tabeli mingisse veergu lisatavad andmed vastaksid mingile reeglile. Näideteks on kontroll, kas mõne tekstivälja sisu ei koosne ainult tühikutest või kas sünnikuupäev on reaalne (st ei ole kaugel tulevikus või minevikus). Kitsendusi saab jõustada deklaratiivselt, luues tabelitega seotud CHECK, NOT NULL, PRIMARY KEY, UNIQUE ja FOREIGN KEY kitsendusi või protseduurselt, kirjutades trigereid, mis käivituvad mingi andmemuudatuse tulemusena. Nii CHECK kitsenduste kui trigerite puhul on stringide kontrolliks võimalik kasutada regulaaravaldisi.

Järgnevate SQL lausete koostamisel kasutas autor Erki Eessaare õppematerjali „Trigerid ja andmekäitluskeele lausete ümberkirjutamise reeglid“ (Eessaar 2014a) ning andmebaasi näiteprojekti „Ülikooli infosüsteemi vastuvõtuaegade allsüsteem“ (Eessaar 2014b). PostreSQLi koduleheküljel avaldatud dokumentatsioone: „38.9. Trigger Procedures“, „CREATE FUNCTION“ (PostgreSQL) ning veebilehel Openbravo olevat artiklit „How to create a Trigger“ ning õpetust „Создание триггеров в PostgreSQL“ (Стасышина 2010) (Openbravo) Samuti kasutati programmeerijatele suunatud veebilehele StackOverflow abi regulaaravaldiste koostamisel. (StackOverflow)

Näitena püüan jõustada enda loodud kolmel mudelil kontrolli, et sisestatav riigi kood tekstivälja koosneks kindlasti kolmest suurtähest.

1. „Iga klassifikaator eraldi tabelis“:

```
ALTER TABLE Riik ADD CONSTRAINT chk_riigi_kood_taheline__riik CHECK
(riigi_kood ~'(^[A-Z]{3}$)');
```

2. „Kõik klassifikaatorid ühises tabelis“:

```
CREATE FUNCTION f_chk_riik() RETURNS trigger AS $chk_riik$
      BEGIN
        -- Kontrollib, kas kl on riik ja kas vastab nõuetele
        IF NEW.kl_id_tyyp IN
               (SELECT KL1.klassifikaator_id
                    FROM Klassifikaator AS KL1
                    INNER JOIN Klassifikaator AS KL2
                    ON KL1.kl_id_tyyp=KL2.klassifikaator_id
                    WHERE KL1.nimetus = 'riik' AND
                      KL2.nimetus='klassifikaator')
                AND (NEW.kood_taht !~'(^[A-Z]{3}$)')
         THEN
            RAISE EXCEPTION 'Riigi kood peab koosneb kolmest
suurtähest!';
        END IF;
     RETURN NEW;
    END;
$chk_riik$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trig_chk_riik BEFORE INSERT OR UPDATE OF kood_taht ON
Klassifikaator
    FOR EACH ROW EXECUTE PROCEDURE f_chk_riik();
```

3. „Kunstlik ühendaja“:

```
ALTER TABLE Riik ADD CONSTRAINT chk_riigi_kood_taheline__riik CHECK
(riigi_kood ~'(^[A-Z]{3}$)');
```

Selleks, et jõustada antud kontrolli, oli „Iga klassifikaator eraldi tabelis“ ja „Kunstlik ühendaja“ korral vaja vaid lisada üks CHECK kitsendus. „Kõik klassifikaatorid ühises tabelis“ korral see aga ei osutunud võimalikuks, kuna ei ole garanteeritud, et veerus *kood_taht* on ainult riikide koodid. Mõne teise klassifikaatori kood võib ka koosneda kolmest märgist. PostgreSQL (9.3) ei luba CHECK kitsendustes alampäringuid. Seega antud kontrolli jõustamiseks on vaja koostada triger ja selle juurde funktsioon, mis käivitub iga kord kui tabelisse proovitakse lisada mõni rida või muuta andmeid veerus *kood_taht*. (PostgeSQL)

6.2.1 Vahetulemused

Kõige keerukam kitsenduse jõustamine oli „Kõik klassifikaatorid ühises tabelis“ andmebaasis. See paistab silma ka vastavaid lauseid visuaalselt hinnates. Kui andmete

kontroll otsustatakse läbi viia andmebaasi tasemel, siis „Iga klassifikaator eraldi tabelis“ ja „Kunstlik ühendaja“ andmebaasides võib see tähendada näiteks igale klassifikaatorite tabelite veerule null või rohkem CHECK kitsendust. Kui teha klassifikaatorite disain „Kõik klassifikaatorid ühises tabelis“ järgi, siis tuleks igale CHECK kitsendusele vastavusse panna üks triger koos funktsiooniga, mis on suur töö.

6.3 Mustrite võrdluse tulemused

Eelnevast kahest alapeatükist, kus esitati päringute loomise ja kitsenduste lisamise võrdlevad laused, nähtub, et ühe disainilahenduse puhul oli päringute ja kontrollide realiseerimine oluliselt raskem. Järgnevalt esitatakse iga disaini kohta võrdluse tulemused ja järeldused, milleni autor jõudis kirjeldatud päringute ja kontrollide loomisel. Välja tuuakse peamised leitud mustrite tugevused ja nõrkused.

„Iga klassifikaator eraldi tabelis“ – selle lahenduse puhul on kerge andmeid otsida. Esiteks seepärast, et klassifikaatorite koodid on sisulise tähendusega. See säästab iga klassifikaatori koodi lisamisel päringusse ühe ühendamisoperatsiooni. Keerukamate päringute puhul on kergem järge hoida, kuna andmed on tabelite vahel loogiliselt jagatud. Puuduseks võib olla, et otsides andmeid mitmetest tabelitest võib langeda päringute kiirus. Seda teemat käesolevas töös ei uuritud ning seega jääb nimetatud puudus teoreetiliseks.

„Kõik klassifikaatorid ühises tabelis“ – päringute tegemine on keerukam. Peab arvestama rohkemate asjaoludega, kuna selle lahenduse puhul on vajalik enamiku andmete kättesaamiseks teha tabeli ühendamist iseendaga (järgnevas koodilõigus tähistatud rohelise taustaga) ning andma tabelile uue nime ehk *alias*-e, et seda saaks kasutada.

```
SELECT kl.kood_taht, kl.nimetus FROM Klassifikaator kl
INNER JOIN Klassifikaator kl_seisund ON
kl_seisund.klassifikaator_id = kl.kl_id_seisund
INNER JOIN Klassifikaator kl_riik ON kl.kl_id_tyyp =
kl_riik.klassifikaator_id
WHERE kl_seisund.nimetus = 'aktiivne' AND kl_riik.nimetus =
'riik';
```

Kõige suurem raskus tekib andmete kontrollimisega, kuna selle realiseerimise raskuse vahe oli võrreldes teise kahe disainiga kõige suurem. Pole vahet, kas see kontroll tehakse andmebaasi tasemel trigeritega või klassifikaatorite haldamise rakenduses – koodi loogika on keeruline. Seega tundus antud disainilahendus autori uuritud disainilahendustest kõige

ebamõistlikum, kuna võrreldes teiste lahendustega võtab päringute ja andmekontrollide realiseerimine kõige rohkem aega.

„Kunstlik ühendaja“ – selle lahenduse puhul on päringute ja kontrollide keerukuse tulemused võrdlemisi sarnased „Iga klassifikaator eraldi tabelis“ lahendusega kuna ka lahendus ise on suures osas sarnane. Mõnel juhul tuli päring veidi keerukam, kuna nõudis klassifikaatori metaandmete kättesaamiseks ühendamist lisatabeliga. Samas kitsenduste jõustamise osas on selline lahendus kompaktsem, sest klassifikaatorite metaandmete kontroll tuleb realiseerida ainult ühes tabelis. Seevastu „Iga klassifikaator eraldi tabelis“ korral peaks seda jõustama täpselt niipalju kordi, kui palju on erinevaid klassifikaatorite tüüpe.

7. Mustrite üldine võrdlus ja tulemused

Selles peatükis võrreldakse veelkord omavahel kirja pandud mustrite pakutavaid disainilahendusi.

7.1 Mustrite omavaheline võrdlus

Igal disainimustril on omad tugevad ja nõrgad küljed ning nad avalduvad erinevalt, olles otseses sõltuvuses, milliseid andmeid ja kui suures mahus soovitakse andmebaasis hoida. Mõnda olukorda on võimalik ette näha, kuid põhivõrdluse mustrite vahel teostan oma töös loodud näidete põhjal, arvestades nende iseärasusi.

Disainimise keerukus: erinevate disainimustrite alusel andmebaaside disainimise keerukus oli suhteliselt sarnane. Erinevus oli ainult ajas. Kõige rohkem läks aega „Iga klassifikaator eraldi tabelis“ mustriga, kuna pidi looma rohkem tabeleid, kontrollima kõikide väljade tüüpe ning andma korrektsed nimed. Kõige kiiremini läks samas „Kõik klassifikaatorid ühises tabelis“ mustriga, kuna tuli luua vaid kaks tabelit, ning deklaratiivseid kitsendusi, mida tuli jõustada, oli palju vähem. Samas võib selline aja jaotus olla tingitud sellest, et autor alustas „Iga klassifikaator eraldi tabelis“ ja „Kunstlik ühendaja“ mustritega ning lõpetas mustriga „Kõik klassifikaatorid ühises tabelis“. Alguses võttis aega harjumine, lõpus olid aga kõik tegevused tuttavad ning töö läks ladusamalt. Siiski suurema hulga klassifikaatorite puhul läheb tõenäoliselt igale eraldi tabeli disainimine ajakulukamaks, kui ühe suure tabeli tegemine.

Päringute keerukus: päringute kirjutamise seisukohalt on kõige lihtsam andmeid otsida „Iga klassifikaator eraldi tabelis“ ning „Kunstlik ühendaja“ andmebaasidest. Mõlemas nendest on klassifikaatoritele viitavad välisvõtmed sisulise tähendusega, mis lubab andmeid otsida klassifikaatori koodide järgi. See on vastupidine disainile „Kõik klassifikaatorid ühises tabelis“, kus selle jaoks peab tegema iga kord ühe ühendamisoperatsiooni. Samuti on päringute kirjutamine lihtsam ka semantika mõttes kuna „Kõik klassifikaatorid ühises tabelis“ korral peab enamasti ühendama tabelit iseendaga, mis tekitab päringute kirjutamisel raskusi tabelitele viitamisega. Täpsem võrdlus päringute kohta on tehtud peatüki nr 6 lõpus.

Andmebaasi skeemi arusaadavus: kõige paremini loetav on „Iga klassifikaator eraldi tabelis“, kuna kogu konkreetse klassifikaatori kohta käiv info on ühes tabelis, ning seal ei ole mitte midagi muud. Kõik seal olevad veerud käivad ühe klassifikaatori kohta. Klassifikaatoritele viitavate välisvõtmete väärtused on arusaadavad, kuna omavad kasutajate jaoks sisulist tähendust. Järgmine paremuselt on „Kunstlik ühendaja“ – seal kehtib sama põhimõte, mis ka esimeses lahenduses, kuid klassifikaatorite metaandmed (nt viimane muutja, seisund, muutmiskuupäev) asuvad eraldi tabelis, kus tuleb teostada eraldi otsing. Kõige ebaloomulikum oli „Kõik klassifikaatorid ühises tabelis“ , kuna andmeid on raske otsida, välisvõtme väärtused on lihtsalt viited süsteemi genereeritud unikaalsele täisarvulisele identifikaatorile, peaaegu kõik klassifikaatori tabeli veerud on mittekohustuslikud. Raske on mõista, millised veerud võiksid käia just konkreetse klassifikaatori tüübi kohta ning millised mitte. Tabelis on palju veerge, mille väljadest on suur osa tühi. Andmete rohkus tekitab segadust – otsides tabelist kaupa kategooriaga „piimatoode“, on selle kohta muuhulgas ka veerg *pindala*. Kujutage nüüd, et keegi on kategooria kohta ka pindala registreerinud.

Andmete terviklikkus: termin tähendab veendumust andmete reeglitele vastavuses ning sidususes. „Iga klassifikaator eraldi tabelis“ ja „Kunstlik ühendaja“ puhul on olukord suhteliselt sarnane. Kindlalt on teada, millised väljad on vaja täita, kuna klassifikaatori tüübist sõltuvalt on lisatud tabelitesse ka vajalikud veerud. Andmete kontorolle on samuti suhteliselt lihtne rakendada, lisades tabelile CHECK kitsendused ning tagades veergude kohustuslikus NOT NULL abil. Ka koodi ning nimetuse unikaalsust on lihtne tagada deklareerides tabelis nendele veergudele vastavalt PRIMARY KEY ja UNIQUE kitsenduse. Keerukam on olukord „Kõik klassifikaatorid ühises tabelis“ mustriga, kuna klassifikaatori tabelit uurides, et ole võimalik aru saada, millised väljad on konkreetse klassifikaatori tüübi jaoks vaja täita. Et seda kontrollida, tuleb teha iga klassifikaatori tüübi jaoks andmebaasi triger. Kui see jätta aga tegemata, siis andmete terviklikus ei saa olla andmebaasi tasemel tagatud. Sellisel juhul tekib võimalus jätta täitmata väljad, mis on selle tüübi jaoks kohustuslikud ning vastupidi lisada väärtused lahtritesse, mis ei ole konkreetse klassifikaatori tüübiga seotud.

Puuduvad andmed (NULL-id): kõige rohkem mittekohustuslikke veerge on klassifikaatori tabelis „Kõik klassifikaatorid ühises tabelis“ mustri korral, mis tekib selle disaini iseärasusest. Kõik veerud, välja arvatud klassifikaatori id ning klassifikaaatori tüüp, on mittekohustuslikud. Suurema osa ridade puhul on NULL-ide arv väga suur (vt Joonis 332) Kahe teise lahendusega seda muret ei ole, kuna mittekohustuslik veerg on ainult üks, milleks on „kirjeldus“.

Paindlikkus: paindlikkus tähendab võimalusi olemasolevat disaini muuta ja lisada sellele uusi elemente. Paindlikkuse poole pealt kiidetakse kirjanduses just mustrit „Kõik klassifikaatorid ühises tabelis“, kuna uue klassifikaatori tüübi lisamine ei tohiks kaasa tuua uute tabelite tekitamist. Vaatamata sellele võib siiski ette tulla vajadus lisada veerge olemasolevasse klassifikaatorite tabelisse, mis teeb selle tabeli ainult veel suuremaks ja raskesti mõistetavamaks. Mustrite „Iga klassifikaator eraldi tabelis“ ja „Kunstlik ühendaja“ puhul tuleks uue klassifikaatori tüübi puhul luua uus tabel ning seoste teiste tabelitega. Mustri „Kunstlik ühendaja“ omaduseks on eraldiseisev klassifikaatorite tabel metaandmete jaoks. Sellisel juhul ei pea lisama soovitud täiendavaid veergusid kõikidele klassifikaatorite tabelitele, nagu mustri „Iga klassifikaator eraldi tabelis“.

7.2 Tulemused

Eelneva töö tulemusena annab autor selles peatükis hinnangu, milline muster sobis valitud näite jaoks kõige paremini ning miks.

Parimaks disainimustriks teostatud näitele valitakse „Kunstlik ühendaja“, kuna see on lihtne – inimene saab sellest pikema süvenemiseta aru. Sellega on lihtne ümber käia – tehtud päringud olid ühed lihtsamatest ning andmete kontrolli enne andmete tabelisse sisestamist oli kerge realiseerida. Lisaks on see paindlik lahendus.

„Iga klassifikaator eraldi tabelis“ ei valitud parimaks just paindlikkuse puudumise pärast. Kuna vajadus registreerida klassifikaatorite kohta metaandmeid on suur, siis parim võimalus selleks on „Kunstlik ühendaja“ korral.

„Kõik klassifikaatorid ühises tabelis“ oli antud näite puhul kõige ebapraktilisem lahendus. See on oma olemuselt keerukas ning päringute kirjutamiseks ebamugav. Andmekontrollide jõustamine andmebaasi tasemel nõuab selle lahenduse puhul hulgaliselt andmebaasi trigerite ja funktsioonide loomist. Samasugune keerukus oleks vaja realiseerida rakenduses, kui andmete kontrolli teha rakenduse tasemel. Klassifikaatori tabel paisub suureks ning erinevat tüüpi klassifikaatorite erinevate atribuutide tõttu jääb see tabel poolenisti tühjaks.

8. Kokkuvõte

Töö eesmärgiks oli õppida tundma ja kirjeldada mõningaid disainimustreid klassifikaatorite esitamiseks SQL-andmebaasides, disainida ning realiseerida iga mustri põhjal töötav näide ning neid disaine saadud kogemuste põhjal võrrelda.

Töö oluliseks tulemuseks on kolm sõnastatud disainimustrit, nende põhjal loodud andmebaaside näited, andmebaaside põhjal tehtud päringute ja andmekontrollide näited ning nende mustrite võrdlemise tulemusena saadud järeldused.

Selles tööd esitatud mustrid on vaid kõige põhilisemad disainivõimalused. Lisaks on mitmeid võimalusi neid kombineerida ja varieerida. Samuti jäid selle töö skoobist välja päringute kiiruste mõõtmised. Huvitav oleks realiseerida klassifikaatorite haldamise tarkvara „Kunstlik ühendaja“ mustri jaoks, mis oleks metaandmetega juhitav ning mille puhul ei peaks uue klassifikaatori tüübi lisamisel hakkama tarkvara ümber kirjutama. Eelnimetatud teemad on näited töö edasiarendamise võimalustest.

Püstitatud eesmärgid saavutati ning kõik näiteandmebaasid on realiseeritud. Disainimustrite võrdlusel tulid välja erinevused ning kitsaskohad, millega algul ei osatud arvestada. Selles töös jäi vaatlemata üks oluline päringute omadus – kiirus. Samas võiks see olla tulevikus osa, mida saaks edasi arendada ning võimalik, et selle tulemused muudavad parima mustri valiku tulemusi.

Summary

The main goal of this work was to study and write some design patterns for representing reference data in SQL databases, create an example database based on each pattern, and compare the designs based on the collected experience.

The important results of the thesis are three specified design patterns, example databases created based on them, examples of queries and data checks based on the databases as well as conclusions based on the comparison of patterns.

The patterns presented in the work are the most basic design solutions. There are multiple ways to combine and modify them. Moreover, the performance comparison of queries was also out of the scope of the work. It would be interesting to implement a classifier management application for “Artificial uniter“ pattern that would be metadata driven and does not need reprogramming if new classifier types are added to the system. These are examples of possible further work with the current topic.

All the goals were achieved and example databases have been implemented. Comparison of the design patterns revealed differences and problems that one did not consider in the beginning. The thesis did not consider an important property of queries – performance. It is an aspect of the work that needs further investigation and it is possible that the results will change the results of selecting the best pattern.

Kasutatud kirjandus

1. Alexander, C. (1977) „A Pattern Language: Towns, Buildings, Construction“ Oxford University Press

2. Andrews, T. (2004) „OTLT and EAV: the two big design mistakes all beginners make“ [WWW] http://tonyandrews.blogspot.com/2004/10/otlt-and-eav-two-big-design-mistakes.html (20.05.2015)

3. Avaliku teabe seadus. (2008). – Riigi Teataja. [WWW] https://www.riigiteataja.ee/akt/12910889 (15.05.2015)

4. Celko, J. (2011) „Look-up Tables in SQL“ [WWW] https://www.simple-talk.com/sql/t-sql-programming/look-up-tables-in-sql-/ (24.05.2015)

5. *DiponRoy*. (2014) „SQL queries to manage hierarchical or parent-child relational rows in SQL Server“ [WWW] http://www.codeproject.com/Articles/818694/SQL-queries-to-manage-hierarchical-or-parent-child (10.05.2015)

6. Eessaar, E. (2014a) „Trigerid ja andmekäitluskeele lausete ümberkirjutamise reeglid“ Andmebaasid II õppematerjalid. [ONLINE]

7. Eessaar, E. (2014b) „Ülikooli infosüsteemi vastuvõtuaegade allsüsteem“ Andmebaasid II õppematerjalid. [ONLINE]

8. Eesti keele seletav sõnaraamat [EKSS] [WWW] http://www.eki.ee/dict/ekss/index.cgi?Q=muster&F=M (15.05.2015)

9. Klassifikaatorid. (2014). – Statistikaamet [WWW] http://www.stat.ee/klassifikaatorid (20.05.2015)

10. Krönström, K. (2015). „Hierarhiliste andmete esitamine SQL-andmebaasides kolme disainilahenduse näitel“ Magistritöö. TTÜ Informaatikainstituut. [ONLINE]

11. Mandelstein, M. et al. (2013) „A Practical Guide to Managing Reference Data with IBM InfoSphere Master Data Management Reference Data Management Hub“

12. *Mike* (2007) „OTLT/EAV design - why do people hate it?“[WWW] http://www.dbforums.com/showthread.php?1619660-OTLT-EAV-design-why-do-people-hate-it (20.05.2015)

13. Openbravo. „How to create a Trigger“ [WWW] http://wiki.openbravo.com/wiki/How_to_create_a_Trigger (15.05.2015)

14. Poolet, M.A. (2006) „Designing for Performance: Lookup Tables“ [WWW] http://sqlmag.com/database-administration/designing-performance-lookup-tables (24.05.2015)

15. PostgreSQL. „38.9. Trigger Procedures“ [WWW] http://www.postgresql.org/docs/8.3/static/plpgsql-trigger.html (15.05.2015)

16. PostgreSQL. „Create function“ [WWW] http://www.postgresql.org/docs/9.1/static/sql-createfunction.html (15.05.2015)

17. Rang, R. (2008) Klassifikaatori struktuur ja kirjeldamine RIHA-s. [WWW] https://www.ria.ee/public/RIHA/RIHA_teabep_ev_14.11.08/riha_reet.doc (27.04.2015)

18. Riikide ja territooriumide klassifikaator 2013v1 [WWW] http://metaweb.stat.ee/view_xml_multi_code.htm?id=3477719&siteLanguage=ee (23.05.2015)

19. Rouse, M. „Pattern (Design Pattern) Definition“ [WWW] http://searchsoftwarequality.techtarget.com/definition/pattern (20.05.2015)

20. Rõzova, N. (2011) „Disainimustrid üldistusseoste realiseerimiseks SQL-andmebaasides“ Bakalaureusetöö. TTÜ Informaatikainstituut. [ONLINE]

21. Sen, A. (2009) „Five Simple Database Design Errors You Should Avoid“ [WWW] https://www.simple-talk.com/sql/database-administration/five-simple--database-design-errors-you-should-avoid/ (15.05.2015)

22. StackOverflow. [WWW] http://stackoverflow.com/questions/6067592/regular-expression-to-match-only-alphabetic-characters (21.05.2015)

23. Techopedia. „Data Integrity“ [WWW] http://www.techopedia.com/definition/811/data-integrity-databases (24.05.2015)

24. Techtarget. „metadata“ [WWW] http://whatis.techtarget.com/definition/metadata (24.05.2015)

25. Wikipedia. „Design science“ [WWW] http://en.wikipedia.org/wiki/Design_science (24.05.2015)

26. Wikipedia. „Hierarchical and recursive queries in SQL“ [WWW] http://en.wikipedia.org/wiki/Hierarchical_and_recursive_queries_in_SQL (15.05.2015)

27. Wikipedia. „Hierarchical database model“ [WWW] http://en.wikipedia.org/wiki/Hierarchical_database_model (15.05.2015)

28. Wikipedia. „Software design pattern“ [WWW] http://en.wikipedia.org/wiki/Software_design_pattern (01.05.2015)

29. Боженов. (2004) „Концепция построения системы «Информационный фонд“ [ONLINE]

30. Википедия. „CASE“ [WWW] https://ru.wikipedia.org/wiki/CASE (24.05.2015)

31. Стасышина,Т,Л. (2010) „Создание триггеров в PostgreSQL“ [WWW] http://postgresql.ru.net/docs/trigger.html (20.05.2015)

Lisa 1

Tabelite loomise skript „Iga klassifikaator eraldi tabelis“ mustri korral

```
CREATE TABLE kauba_kategooria (
    kauba_kategooria_kood integer NOT NULL,
    nimetus character varying(50) NOT NULL,
    klassifikaatori_seisundi_liik_kood smallint DEFAULT 1 NOT NULL,
    kauba_kategooria integer,
    muutja character varying(50) NOT NULL
);
CREATE TABLE kauba_seisundi_liik (
    kauba_seisundi_liik_kood smallint NOT NULL,
    klassifikaatori_seisundi_liik_kood smallint DEFAULT 1 NOT NULL,
    nimetus character varying(50) NOT NULL,
    kirjeldus character varying(255),
    muutja character varying(50) NOT NULL
);
CREATE TABLE kaup (
    kaup_id serial NOT NULL,
    riigi_kood character(3) DEFAULT 'EST'::bpchar NOT NULL,
    kauba_kategooria_kood integer NOT NULL,
    kauba_seisundi_liik_kood smallint DEFAULT 1 NOT NULL
);
CREATE SEQUENCE kaup_kaup_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
CREATE TABLE klassifikaatori_seisundi_liik (
    klassifikaatori_seisundi_liik_kood smallint NOT NULL,
    nimetus character varying(50) NOT NULL,
    on_aktiivne boolean NOT NULL,
    muutja character varying(50) NOT NULL
);
CREATE TABLE riik (
    riigi_kood character(3) NOT NULL,
    nimetus character varying(50) NOT NULL,
    pindala numeric(10,2) NOT NULL,
    klassifikaatori_seisundi_liik_kood smallint DEFAULT 1 NOT NULL,
    muutja character varying(50) NOT NULL
);
ALTER TABLE ONLY kaup ALTER COLUMN kaup_id SET DEFAULT nextval('kaup_kaup_id_seq'::regclass);
ALTER TABLE ONLY kauba_kategooria
    ADD CONSTRAINT pk_kauba_kategooria_kood PRIMARY KEY (kauba_kategooria_kood);
ALTER TABLE ONLY kauba_seisundi_liik
    ADD CONSTRAINT pk_kauba_seisundi_liik_kood PRIMARY KEY (kauba_seisundi_liik_kood);
ALTER TABLE ONLY kaup
    ADD CONSTRAINT pk_kaup_id PRIMARY KEY (kaup_id);
ALTER TABLE ONLY klassifikaatori_seisundi_liik
    ADD CONSTRAINT pk_klassifikaatori_seisundi_liik_kood PRIMARY KEY
(klassifikaatori_seisundi_liik_kood);
```

```
ALTER TABLE ONLY riik
    ADD CONSTRAINT pk_riigi_kood PRIMARY KEY (riigi_kood);
ALTER TABLE ONLY kauba_kategooria
    ADD CONSTRAINT tc_nimetus__kauba_kategooria UNIQUE (nimetus);
ALTER TABLE ONLY kauba_seisundi_liik
    ADD CONSTRAINT tc_nimetus__kauba_seisundi_liik UNIQUE (nimetus);
ALTER TABLE ONLY klassifikaatori_seisundi_liik
    ADD CONSTRAINT tc_nimetus__klassifikaatori_seisundi_liik UNIQUE (nimetus);
ALTER TABLE ONLY riik
    ADD CONSTRAINT tc_nimetus__riik UNIQUE (nimetus);
CREATE INDEX idx_kauba_kategooria_kood__kauba_kat ON kauba_kategooria USING btree
(kauba_kategooria_kood);
ALTER TABLE ONLY kauba_kategooria
    ADD CONSTRAINT fk_kauba_kategooria FOREIGN KEY (kauba_kategooria)
ALTER TABLE ONLY kaup
    ADD CONSTRAINT fk_kauba_kategooria_kood FOREIGN KEY (kauba_kategooria_kood) REFERENCES
kauba_kategooria(kauba_kategooria_kood);
ALTER TABLE ONLY kaup
    ADD CONSTRAINT fk_kauba_seisundi_liik_kood FOREIGN KEY (kauba_seisundi_liik_kood)
REFERENCES kauba_seisundi_liik(kauba_seisundi_liik_kood);
ALTER TABLE ONLY riik
    ADD CONSTRAINT fk_klassifikaatori_seisundi_liik_kood FOREIGN KEY
(klassifikaatori_seisundi_liik_kood) REFERENCES
klassifikaatori_seisundi_liik(klassifikaatori_seisundi_liik_kood);
ALTER TABLE ONLY kauba_seisundi_liik
    ADD CONSTRAINT fk_klassifikaatori_seisundi_liik_kood FOREIGN KEY
(klassifikaatori_seisundi_liik_kood) REFERENCES
klassifikaatori_seisundi_liik(klassifikaatori_seisundi_liik_kood);
ALTER TABLE ONLY kauba_kategooria
    ADD CONSTRAINT fk_klassifikaatori_seisundi_liik_kood FOREIGN KEY
(klassifikaatori_seisundi_liik_kood) REFERENCES
ALTER TABLE ONLY kaup
    ADD CONSTRAINT fk_riigi_kood FOREIGN KEY (riigi_kood) REFERENCES riik(riigi_kood);


Andmete sisestamine:
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus,
klassifikaatori_seisundi_liik_kood, vanem_kauba_kategooria, muutja)  VALUES (100, 'Toiduaine',
2, NULL, 'admin');
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus,
klassifikaatori_seisundi_liik_kood, vanem_kauba_kategooria, muutja)  VALUES (210,
'Spordiriided', 2, 200, 'admin');
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus,
klassifikaatori_seisundi_liik_kood, vanem_kauba_kategooria, muutja)  VALUES (200, 'riided', 2,
NULL, 'admin');
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus,
klassifikaatori_seisundi_liik_kood, vanem_kauba_kategooria, muutja)  VALUES (120,
'piimatoode', 2, 100, 'admin');
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus,
klassifikaatori_seisundi_liik_kood, vanem_kauba_kategooria, muutja)  VALUES (110,
'pagaritoode', 2, 100, 'admin');

INSERT INTO kauba_seisundi_liik (kauba_seisundi_liik_kood, klassifikaatori_seisundi_liik_kood,
nimetus, kirjeldus, muutja) VALUES (1, 2, 'laos', 'Kaup on laos olemas.', 'admin');
INSERT INTO kauba_seisundi_liik (kauba_seisundi_liik_kood, klassifikaatori_seisundi_liik_kood,
nimetus, kirjeldus, muutja) VALUES (2, 2, 'tarnimisel', 'Kaupa laos ei ole, kuid saabub 2
nädala jaaoksul. ', 'admin');
```

```
INSERT INTO kaup (kaup_id, riigi_kood, kauba_kategooria_kood, kauba_seisundi_liik_kood) VALUES
(1, 'EST', 100, 1);
INSERT INTO kaup (kaup_id, riigi_kood, kauba_kategooria_kood, kauba_seisundi_liik_kood) VALUES
(2, 'USA', 110, 2);
INSERT INTO kaup (kaup_id, riigi_kood, kauba_kategooria_kood, kauba_seisundi_liik_kood) VALUES
(3, 'USR', 210, 1);

INSERT INTO klassifikaatori_seisundi_liik (klassifikaatori_seisundi_liik_kood, nimetus,
on_aktiivne, muutja) VALUES (4, 'kustutatud', false, 'admin');
INSERT INTO klassifikaatori_seisundi_liik (klassifikaatori_seisundi_liik_kood, nimetus,
on_aktiivne, muutja) VALUES (1, 'loomisel', true, 'admin');
INSERT INTO klassifikaatori_seisundi_liik (klassifikaatori_seisundi_liik_kood, nimetus,
on_aktiivne, muutja) VALUES (3, 'mitteaktiivne', true, 'admin');
INSERT INTO klassifikaatori_seisundi_liik (klassifikaatori_seisundi_liik_kood, nimetus,
on_aktiivne, muutja) VALUES (2, 'aktiivne', true, 'admin');

INSERT INTO riik (riigi_kood, nimetus, pindala, klassifikaatori_seisundi_liik_kood, muutja)
VALUES ('EST', 'Eesti', 45227.00, 2, 'admin');
INSERT INTO riik (riigi_kood, nimetus, pindala, klassifikaatori_seisundi_liik_kood, muutja)
VALUES ('USA', 'Ameerika Ühendriigid', 9826675.00, 2, 'admin');
INSERT INTO riik (riigi_kood, nimetus, pindala, klassifikaatori_seisundi_liik_kood, muutja)
VALUES ('USR', 'Nõukogude Sotsialistlike Vabariikide Liit', 22402200.00, 3, 'admin');
```

Tabelite loomise skript „Kõik klassifikaatorid ühises tabelis“ mustri korral

```
CREATE TABLE kaup (
    kaup_id serial NOT NULL,
    kl_id_riik integer NOT NULL,
    kl_id_kategooria integer NOT NULL,
    kl_id_seisund integer NOT NULL
CREATE SEQUENCE kaup_kaup_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
CREATE TABLE klassifikaator (
    klassifikaator_id serial NOT NULL,
    kl_id_tyyp integer,
    kood_taht character varying(3),
    kood_number smallint,
    nimetus character varying(50),
    pindala numeric(10,2),
    kirjeldus character varying(255),
    kl_id_seisund integer,
    kl_id_kategooria integer
CREATE SEQUENCE klassifikaator_klassifikaator_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
ALTER TABLE ONLY klassifikaator ALTER COLUMN klassifikaator_id SET DEFAULT
nextval('klassifikaator_klassifikaator_id_seq'::regclass);


ALTER TABLE ONLY kaup
```

```
    ADD CONSTRAINT pk_kaup_id PRIMARY KEY (kaup_id);
ALTER TABLE ONLY klassifikaator
    ADD CONSTRAINT pk_klassifikaator_id PRIMARY KEY (klassifikaator_id);
CREATE INDEX idx_kl_id_kategooria__kaup ON kaup USING btree (kl_id_kategooria);
CREATE INDEX idx_kl_id_seisund__kaup ON kaup USING btree (kl_id_seisund);
CREATE INDEX idx_kl_id_seisund__klassifikaator ON klassifikaator USING btree (kl_id_seisund);
CREATE INDEX idx_klassifikaator__klassifikaator ON klassifikaator USING btree (kl_id_tyyp);
CREATE INDEX idx_klassifikaator_id__kaup ON kaup USING btree (kl_id_riik);
ALTER TABLE ONLY klassifikaator
    ADD CONSTRAINT fk_kl_id_kategooria FOREIGN KEY (kl_id_kategooria) REFERENCES
klassifikaator(klassifikaator_id);
ALTER TABLE ONLY kaup
    ADD CONSTRAINT fk_kl_id_kategooria FOREIGN KEY (kl_id_kategooria) REFERENCES
klassifikaator(klassifikaator_id);
ALTER TABLE ONLY klassifikaator
    ADD CONSTRAINT fk_kl_id_seisund FOREIGN KEY (kl_id_seisund) REFERENCES
klassifikaator(klassifikaator_id);

ALTER TABLE ONLY kaup
    ADD CONSTRAINT fk_kl_id_seisund FOREIGN KEY (kl_id_seisund) REFERENCES
klassifikaator(klassifikaator_id);
ALTER TABLE ONLY klassifikaator
    ADD CONSTRAINT fk_kl_id_tyyp FOREIGN KEY (kl_id_tyyp) REFERENCES
klassifikaator(klassifikaator_id);

ALTER TABLE ONLY kaup
    ADD CONSTRAINT fk_klassifikaator_id FOREIGN KEY (kl_id_riik) REFERENCES
klassifikaator(klassifikaator_id);
```

Andmete lisamise laused:

```
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (1, 7, NULL, NULL, 'seisundi
liik', NULL, NULL, 2, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (2, 1, NULL, 1, 'loomisel',
NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (3, 1, NULL, 2, 'aktiivne',
NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (4, 1, NULL, 3,
'mitteaktiivne', NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (5, 1, NULL, 4,
'kustutatud', NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (6, 7, NULL, NULL, 'riik',
NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (8, 6, 'EST', 123, 'Eesti
Vabariik', 45227.00, '', 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (9, 6, 'USR', 234,
'Nõukogude Sotsialistlike Vabariikide Liit', 22402200.00, NULL, 4, NULL);
INSERT INTO klassifikaator(klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria)  VALUES (10, 6, 'USA', 456,
'Ameerika Ühendriigid', 9826675.00, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (11, 7, NULL, NULL,
'kauba_kategooria', NULL, NULL, 3, NULL);
```

```
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (12, 11, NULL, 100,
'Toiduaine', NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (13, 11, NULL, 110,
'pagaritoode', NULL, NULL, 3, 12);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (14, 11, NULL, 120,
'piimatoode', NULL, NULL, 3, 12);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (15, 11, NULL, 200,
'riided', NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (16, 11, NULL, 210,
'spordiriided', NULL, NULL, 3, 15);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (18, 7, NULL, NULL, 'kauba
seisundi liik', NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (19, 18, NULL, NULL, 'laos',
NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (20, 18, NULL, NULL,
'tarnimisel', NULL, NULL, 3, NULL);
INSERT INTO klassifikaator (klassifikaator_id, kl_id_tyyp, kood_taht, kood_number, nimetus,
pindala, kirjeldus, kl_id_seisund, kl_id_vanem_kategooria) VALUES (7, 7, NULL, NULL,
'klassifikaator', NULL, NULL, 3, NULL);

INSERT INTO kaup (kaup_id, kl_id_riik, kl_id_kategooria, kl_id_seisund)  VALUES (1, 8, 12,
19);
INSERT INTO kaup (kaup_id, kl_id_riik, kl_id_kategooria, kl_id_seisund) VALUES (2, 9, 16, 20);
INSERT INTO kaup (kaup_id, kl_id_riik, kl_id_kategooria, kl_id_seisund) VALUES (3, 10, 15,
19);
```

Tabelite loomise skript „Kunstlik ühendaja“ mustri puhul

```
CREATE TABLE kauba_kategooria (
    kauba_kategooria_kood smallint NOT NULL,
    nimetus character varying(60) NOT NULL,
    klassifikaator_id integer NOT NULL,
    kauba_kategooria smallint
);

CREATE TABLE kauba_seisundi_liik (
    kauba_seisundi_liik_kood smallint NOT NULL,
    nimetus character varying(60) NOT NULL,
    kirjeldus character varying(255),
    klassifikaator_id integer NOT NULL
);

CREATE TABLE kaup (
    kaup_id integer NOT NULL,
    kauba_seisundi_liik_kood smallint NOT NULL,
    kauba_kategooria_kood smallint NOT NULL,
    riigi_kood character varying(3) NOT NULL
);
```

```
CREATE SEQUENCE kaup_kaup_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
CREATE TABLE klassifikaator (
    klassifikaator_id serial NOT NULL,
    klassifikaatori_seisundi_liik_kood smallint NOT NULL
);

CREATE SEQUENCE klassifikaator_klassifikaator_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

CREATE TABLE klassifikaatori_seisundi_liik (
    klassifikaatori_seisundi_liik_kood smallint NOT NULL,
    nimetus character varying(60) NOT NULL,
    on_aktiivne boolean NOT NULL
);

CREATE TABLE riik (
    riigi_kood character varying(3) NOT NULL,
    nimetus character varying(60) NOT NULL,
    pindala numeric(10,2) NOT NULL,
    klassifikaator_id integer NOT NULL,
    CONSTRAINT chk_riigi_kood_taheline__riik CHECK (((riigi_kood)::text ~ '(^[A-Z]+$)'::text))
);
ALTER TABLE ONLY kaup ALTER COLUMN kaup_id SET DEFAULT nextval('kaup_kaup_id_seq'::regclass);

ALTER TABLE ONLY klassifikaator ALTER COLUMN klassifikaator_id SET DEFAULT
nextval('klassifikaator_klassifikaator_id_seq'::regclass);
ALTER TABLE ONLY kauba_kategooria
    ADD CONSTRAINT pk_kauba_kategooria_kood PRIMARY KEY (kauba_kategooria_kood);
ALTER TABLE ONLY kauba_seisundi_liik
    ADD CONSTRAINT pk_kauba_seisundi_liik_kood PRIMARY KEY (kauba_seisundi_liik_kood);
ALTER TABLE ONLY kaup
    ADD CONSTRAINT pk_kaup_id PRIMARY KEY (kaup_id);
ALTER TABLE ONLY klassifikaator
    ADD CONSTRAINT pk_klassifikaator_id PRIMARY KEY (klassifikaator_id);
ALTER TABLE ONLY klassifikaatori_seisundi_liik
    ADD CONSTRAINT pk_klassifikaatori_seisundi_liik_kood PRIMARY KEY
(klassifikaatori_seisundi_liik_kood);
ALTER TABLE ONLY riik
    ADD CONSTRAINT pk_riigi_kood PRIMARY KEY (riigi_kood);
ALTER TABLE ONLY kauba_seisundi_liik
    ADD CONSTRAINT unq_klassifikaator_id UNIQUE (klassifikaator_id);
ALTER TABLE ONLY kauba_kategooria
    ADD CONSTRAINT unq_klassifikaator_id__kauba_kategooria UNIQUE (klassifikaator_id);
ALTER TABLE ONLY riik
    ADD CONSTRAINT unq_klassifikaator_id__riik UNIQUE (klassifikaator_id);
CREATE INDEX idx_klassifikaator_id__kauba_kategooria ON kauba_kategooria USING btree
(klassifikaator_id);
CREATE INDEX idx_klassifikaator_id__kauba_seis_liik ON kauba_seisundi_liik USING btree
(klassifikaator_id);
CREATE INDEX idx_klassifikaator_id__riik ON riik USING btree (klassifikaator_id);
ALTER TABLE ONLY kauba_kategooria
```

```
    ADD CONSTRAINT fk_kauba_kategooria FOREIGN KEY (kauba_kategooria) REFERENCES
kauba_kategooria(kauba_kategooria_kood);
ALTER TABLE ONLY kaup
    ADD CONSTRAINT fk_kauba_kategooria_kood FOREIGN KEY (kauba_kategooria_kood) REFERENCES
kauba_kategooria(kauba_kategooria_kood);
ALTER TABLE ONLY kaup
    ADD CONSTRAINT fk_kauba_seisundi_liik_kood FOREIGN KEY (kauba_seisundi_liik_kood)
REFERENCES kauba_seisundi_liik(kauba_seisundi_liik_kood);
ALTER TABLE ONLY klassifikaator
    ADD CONSTRAINT fk_kauba_seisundi_liik_kood FOREIGN KEY
(klassifikaatori_seisundi_liik_kood) REFERENCES
klassifikaatori_seisundi_liik(klassifikaatori_seisundi_liik_kood);
ALTER TABLE ONLY kauba_seisundi_liik
    ADD CONSTRAINT fk_klassifikaator_id FOREIGN KEY (klassifikaator_id) REFERENCES
klassifikaator(klassifikaator_id);
ALTER TABLE ONLY kauba_kategooria
    ADD CONSTRAINT fk_klassifikaator_id FOREIGN KEY (klassifikaator_id) REFERENCES
klassifikaator(klassifikaator_id);
ALTER TABLE ONLY riik
    ADD CONSTRAINT fk_klassifikaator_id FOREIGN KEY (klassifikaator_id) REFERENCES
klassifikaator(klassifikaator_id);
ALTER TABLE ONLY kaup
    ADD CONSTRAINT fk_riigi_kood FOREIGN KEY (riigi_kood) REFERENCES riik(riigi_kood);
```

Andmete lisamise laused:

```
INSERT INTO klassifikaatori_seisundi_liik (klassifikaatori_seisundi_liik_kood, nimetus,
on_aktiivne) VALUES (1, 'loomisel', true);
INSERT INTO klassifikaatori_seisundi_liik (klassifikaatori_seisundi_liik_kood, nimetus,
on_aktiivne) VALUES (2, 'aktiivne', true);
INSERT INTO klassifikaatori_seisundi_liik (klassifikaatori_seisundi_liik_kood, nimetus,
on_aktiivne) VALUES (3, 'mitteaktiivne', true);
INSERT INTO klassifikaatori_seisundi_liik (klassifikaatori_seisundi_liik_kood, nimetus,
on_aktiivne) VALUES (4, 'kustutatud', false);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (1,
2);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (3,
2);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (4,
2);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (5,
2);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (6,
2);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (7,
2);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (8,
2);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (11,
2);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (12,
2);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (2,
3);
INSERT INTO klassifikaator (klassifikaator_id, klassifikaatori_seisundi_liik_kood) VALUES (13,
2);
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus, klassifikaatir_id,
kauba_kategooria)  VALUES (100, 'Toiduaine', 4, NULL);
```

```
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus, klassifikaatir_id,
kauba_kategooria) VALUES (110, 'pagaritoode', 5, 100);
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus, klassifikaatir_id,
kauba_kategooria) VALUES (120, 'piimatoode', 6, 100);
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus, klassifikaatir_id,
kauba_kategooria) VALUES (200, 'riided', 7, NULL);
INSERT INTO kauba_kategooria (kauba_kategooria_kood, nimetus, klassifikaatir_id,
kauba_kategooria) VALUES (210, 'Spordiriided', 8, 200);
INSERT INTO kauba_seisundi_liik VALUES (1, 'laos', 'Kaup on laos olemas.', 11);
INSERT INTO kauba_seisundi_liik VALUES (2, 'tarnimisel', 'Kaupa laos ei ole, kuid saabub 2
nädala jaaoksul. ', 12);
INSERT INTO riik (kauba_seisundi_liik_kood, nimetus, kirjeldus, klassifikaator_id) VALUES
('EST', 'Eesti', 45227.00, 1);
INSERT INTO riik (kauba_seisundi_liik_kood, nimetus, kirjeldus, klassifikaator_id) VALUES
('USR', 'Nõukogude Sotsialistlike Vabariikide Liit', 22402200.00, 2);
INSERT INTO riik (kauba_seisundi_liik_kood, nimetus, kirjeldus, klassifikaator_id) VALUES
('USA', 'Ameerika Ühendriigid', 9826675.00, 3);
INSERT INTO kaup (kaup_id, kauba_seisundi_liik_kood, kauba_kategooria_kood, riigi_kood) VALUES
(1, 1, 100, 'EST');
INSERT INTO kaup (kaup_id, kauba_seisundi_liik_kood, kauba_kategooria_kood, riigi_kood)
VALUES (2, 1, 110, 'USA');
INSERT INTO kaup (kaup_id, kauba_seisundi_liik_kood, kauba_kategooria_kood, riigi_kood)
VALUES (3, 2, 210, 'USR');
```