TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Apache Cassandra andmebaasisüsteem ja sellele ühe rakenduse migreerimine

bakalaureusetöö

Üliõpilane: Marten Tall
Üliõpilaskood: 123338
Juhendaja: dotsent 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

Apache Cassandra andmebaasisüsteem ja sellele ühe rakenduse migreerimine

Uurimus tutvustab NoSQL andmebaasisüsteemide hulka kuuluvat Apache Cassandra andmebaasisüsteemi. Lugejat valmistab ette töö teoreetiline osa, kus selgitatakse üldiseid NoSQL põhimõtteid ning käsitletakse detailsemalt Cassandra andmebaasisüsteemi.

Töö esimene eesmärk on migreerida Java veebirakendus PostgreSQL andmebaasilt Cassandra andmebaasile.

Teine eesmärk on anda ülevaatlik ning praktiline lähenemine Cassandra andmebaaside loomisele, mis aitab andmebaasiarendajatel luua kvaliteetset andmebaasi. Cassandra andmebaasi disainimine on keerukas protsess, mis vajab ettevalmistust ning ettenägelikkust. Antud uurimistöö võiks aidata seda protsessi kiirendada ning lihtsustada.

Töö tulemusena valmib Cassandra andmebaas ning kirjeldused ja selgitused selle protsessi kohta.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 56 leheküljel, 9 peatükki, 12 joonist, 1 tabelit.

Abstract

Apache Cassandra Database Management System and Migrating an Application to it

This thesis presents a NoSQL database management system called Apache Cassandra. The reader is prepared with a theoretical chapter where main principles of NoSQL and detailed description of Cassandra database management system are covered.

The first goal of this thesis is to migrate a Java web application from a PostgreSQL database to a Cassandra database.

The second goal is to create a hands-on way overview of designing Cassandra databases. The overview should help developers to create quality databases. Designing a Cassandra database is a complicated process that needs preparations and foresight. This thesis could help developers to speed up and simplify this process.

As a result of this thesis a Cassandra database is created with descriptions and explanations of this process.

The thesis is in Estonian and contains 56 pages of text, 9 chapters, 12 figures, 1 table., etc.

Lühendite ja mõistete sõnastik

SQL	***Structured Query Langue*** struktuuripäringukeel (1)
CQL	***Cassandra Query Language*** (2) Cassandra päringukeel
Avatud lähtekoodiga	***Open source*** Avalik ja vaba juurdepääs toote lähtekoodile (3)
Partnervõrguline	***Peer-to-peer*** Sidevõrk, „kus kõigil osalistel on ühesugused õigused ja võimalused ning iga osaline võib algatada sideseanssi“ (1)
Veerule-orienteeritud	***Column-oriented*** Veerule-orienteeritud andmebaasis salvestatakse andmeid pigem veergude sektsioonidena, mitte ridadena.
Ülem-alluv süsteem	***Master/slave system*** „Andmesidesüsteem, milles algatusvõime on ainult ühel komponendil e. ülemal, mis algatab ja juhib sideseansse. Alluv reageerib ülema käskudele.“ (1)
Veergude perekond	***Column-Family*** Veergude perekonna(mitte-SQL) andmebaasisüsteemide vaste relatsioonilisele tabelile
krahhitaaste	***Crash recovery*** Võime jätkata operatsiooni pärast suurt riket (4)
uhtuma	***flush*** Andmeid ajutisest mälupiirkonnast püsiandmekandjale kopeerimine (1)

Relatsioon-baasihaldur	***(RDBMS)Relational Database Management System*** Tarkvara, mis toetab kõiki relatsioonilisi operaatoreid(nagu projektsioon, ühend jt) ja võimaldab kasutajatel töötada relatsiooniliste andmebaasidega (5)
SQL-andmebaasi-süsteem	Andmebaasisüsteem, kus andmete halduseks kasutatakse SQL andmebaasikeelt
võtmeruum	***keyspace*** Veergude perekonna vaste relatsioonilise andmebaasisüsteemi andmebaasile
NoSQL	***Not only SQL*** koondnimetus andmebaasisüsteemidele, mis ei ole SQL-andmebaasisüsteemid
UUID	**Universally Unique IDentifier** (6) sõltumatult unikaalne identifikaator
IRC	***Internet Relay Chat*** Rühmadiskussioon internetis (1)
DAO	***Data Access Object*** andmepääsu objekt (1)
UML	***Unified Modeling Language*** unifitseeritud modelleerimiskeel (1)

Jooniste nimekiri

Joonis 1. Relatsioonilise ja Cassandra andmemudeli võrdlus 16
Joonis 2. Cassandra lihtsustatud andmemudel 17
Joonis 3. Kasutusjuhtude diagramm 21
Joonis 4. Seisundidiagramm 24
Joonis 5. Tellimuste andmebaasi diagramm 25
Joonis 6. Tellimuste andmebaasi teenuste alamdiagramm 26
Joonis 7. Tellimuste andmebaasi isikute alamdiagramm 27
Joonis 8. Tellimuste register 28
Joonis 9. Isikute register 29
Joonis 10. Klassifikaatorite register 30
Joonis 11. Kõik tellimused 38
Joonis 12. Tellimuse detailvaade 39

Tabelite nimekiri

Tabel 1. Süsteemi mittefunktsionaalsed nõuded 23

Sisukord

1. Sissejuhatus 11
1.1 Taust ja probleem 11
1.2 Ülesande püstitus 11
1.3 Metoodika 12
1.4 Ülevaade tööst 12
2. Teoreetiline taust 13
2.1 NoSQL süsteemid 14
2.2 Veergude perekondade andmemudel 15
2.3 Cassandra 16
3. Analüüs 19
3.1 Tervik süsteemi ülevaade 19
3.1.1 Organisatsiooni eesmärgid 19
3.1.2 Infosüsteemi eesmärgid 19
3.1.3 Funktsionaalsed allsüsteemid 20
3.2 Tellimuse funktsionaalse allsüsteem 20
3.2.1 Allsüsteemi eesmärgid 20
3.2.2 Allsüsteemi kasutusjuhtude eskiismudel 21
3.2.3 Ärireeglid 23
3.2.4 Mittefunktsionaalsed nõuded 23
3.2.5 Tellimuse seisundidiagramm 24
4. PostgreSQL andmebaasi disain 25
4.1 Andmebaasi diagramm 25
5. Cassandra andmebaasi disain 28
5.1 Andmebaasi diagramm 28
6. NoSQL andmete modelleerimine 31
6.1 Teenustele orienteeritud andmebaasi mudel 31
7. Migreerimise protsess 34
7.1 Realiseeritav töökoht 34
7.2 Reaalsed kasutusjuhud 34
7.3 Teenustele orienteeritud disain kasutuslugude näitel 38

7.4 Rakenduse lähtekoodi muutmine 41
7.5 Tähelepanekuid disainist kasutuslugude väliselt 41
8. Arutelu ja järeldused 44
9. Kokkuvõte 45
Summary 46
Kasutatud kirjandus 47
Lisa 1 49
Lisa 2 54
Lisa 3 56

1. Sissejuhatus

Üha suuremate andmehulkade tõttu, mida tuleb igapäevaselt süsteemidel töödelda ning andmebaasi salvestada, on tekkinud olukord, kus SQL-andmebaasisüsteemid ei tule enam toime. Selle probleemi taustal on tekkinud uued andmebaasisüsteemid, mis hülgavad relatsioonilise idee.

Käesolev bakalaureusetöö uurib NoSQL andmebaasisüsteeme, eelkõige veergude perekonna mudelit rakendavat Cassandrat.

1.1 Taust ja probleem

Mitterelatsioonilised andmemudelid erinevad tugevalt relatsioonilisest. NoSQL andmebaaside disainietapp on olulisem, kui SQL-andmebaasi puhul, sest halvad(või head) disainiotsused mõjutavad rakenduse kvaliteeti oluliselt rohkem, kui SQL-andmebaasi puhul.

Uurimus on suunatud arendajatele, kellel on vaja otsustada, kas kasutada uues projektis Cassandrat või mõnda muud NoSQL andmebaasisüsteemi.

Uurimus aitab arendajatel mõista NoSQL olemust ning annab näpunäiteid ja soovitusi andmebaasi disainimisel.

1.2 Ülesande püstitus

Töö tulemusena antakse põgus teoreetiline ülevaade erinevatest NoSQL süsteemidest ning detailne kirjeldus Apache Cassandra andmebaasisüsteemi tehnilistest omadustest ja peamistest erinevustest SQL-andmebaasisüsteemidega.

Olemasolevale Java rakendusele, mis kasutab PostgreSQL andmebaasi luuakse Cassandra andmebaas ja migreeritakse rakendus sellele.

Analüüsitakse Cassandra andmebaasi disaini- ja migratsiooniprotsessi, põhjendatakse otsuseid ning tuuakse alternatiivseid võimalusi. Töös ei tegeleta rakenduste jõudluse võrdlemisega erinevate andmebaaside korral, kuid see oleks üks võimalik töö edasiarendus.

1.3 Metoodika

Töö valmimiseks tutvutakse kirjandusega, mis toetab kõiki eesmärke.

Kasutatakse Andmebaasid I (IDU0220) ja Andmebaasid II (IDU0230) ainetes loodud veebirakendusi pakkuva ettevõtte infosüsteemi tellimuste allsüsteemi detailanalüüsi ning selle põhjal loodud Java rakendust ning PostgreSQL andmebaasi. Tellimuste infosüsteemi detailanalüüs valmis dotsent Erki Eessaare juhendamisel ning koostöös üliõpilasega Silver Vaas. (Viide: (7))

Cassandra andmebaas luuakse toetudes olemasolevale detailanalüüsile, mis kirjeldab funktsionaalseid ja mittefunktsionaalseid nõudeid ning reaalseid kasutuslugusid.

1.4 Ülevaade tööst

Peatükis 2 selgitan NoSQL teket ning erinevaid mitterelatsioonilisi andmemudeleid, sh veergude perekonda kuuluva Cassandra põhiomadusi.

Peatükis 3 on tellimuste allsüsteemi detailanalüüs ärireeglite, kasutuslugude, funktsionaalsete ning mittefunktsionaalsete nõuetega.

Peatükis 4 esitan PostgreSQL ning peatükis 5 Cassandra andmebaasi disaini kirjeldavad diagrammid.

Peatükis 6 selgitatakse NoSQL andmete modelleerimise viisi – teenustele orienteeritud disain, mida on uurimuse käigus kasutatud.

Peatükis 7 analüüsitakse PostgreSQL andmebaasilt Cassandra andmebaasile migreerimise protsessi.

2. Teoreetiline taust

NoSQL – väljend mis on tarkvaraarenduses tänaseks laialt levinud, ulatub tagasi Londonist pärit tarkvaraarendaja Johan Oskarssoni 2009. aastal korraldatud kokkusaamiseni. BigTable ja Dynamo näited innustasid mitmeid projekte katsetama alternatiivsete andmete hoiustamise meetoditega ja arutelu nende üle oli selle aja tarkvara-konverentside osa. Johan soovis San Franciscos olles uutest andmebaasisüsteemidest rohkem teada saada, kuid kuna tal oli väga vähe aega, siis ta ei uskunud, et jõuab kõiki konverentse külastada. Ta otsustas korraldada kokkusaamise, kus kõik kokku saaksid ja esitaksid oma tööd korraga kõigile kellel huvi. (8)

Johanil oli vaja sellele üritusele nime – midagi mis oleks hea Twitteri *hashtag*[eesti k. „teemaviide“]: lühike, meeldejääv ja väheste Google vastustega, et see oleks kiiresti leitav. Ta küsis soovitusi #cassandra IRC kanalil ja sai paar vastust. Ta valis Eric Evansi soovituse „NoSQL“, kuigi see nimi ei iseloomusta süsteemi ja on negatiivne, sobis see *hashtagiks*. Sel ajal mõtlesid nad ainult kokkusaamise nime peale ega arvanud, et see tehnoloogiatrend selle nime kasutusele võtab. (8)

NoSQLil puudub kindel definitsioon. „No“[eesti k. „ei“] viitab sellele nagu soovitaks öelda „ei“ SQL-le, kuid see pole tõsi. Paljud NoSQL süsteemid kasutavad andmebaasikeeli, mis on väga sarnased SQL-le, nagu näiteks CQL (Cassandra Query Language). Seetõttu on tekkinud uus arusaam, et NoSQL tähendab „Not only SQL“[eesti k. „Mitte ainult SQL“], mis tähendab, et NoSQL andmebaasisüsteemid tekivad SQL-andmebaasisüsteemide kõrvale, mitte ei asenda neid. (8)

Üha kasvava andmehulgaga, mida on vaja pidevalt töödelda ja hoiustada on tekkinud olukord, kus SQL-andmebaasid ei tule koormustega toime. Neid on keeruline klasterdada ja pidevalt muutuva andmemudeli tõttu on vajalik pidev andmeskeemi muutmine. Nende ja paljude muude probleemide tõttu hakkasid tekkima NoSQL andmebaasisüsteemid, mis on olemuselt skeemivabad, kergesti klasterdatavad ja avatud lähtekoodiga.

2.1 NoSQL süsteemid

Arendades või disainides tarkvara on väga oluline valida sobivaim andmebaasisüsteem. NoSQL andmemudelid erinevad üksteisest väga palju. Selleks, et teha hea otsus, peab tundma süsteemide tugevusi ja nõrkusi.

Võti-väärtus andmemudelis hoitakse andmeid võti-väärtus paarina. Võtme alusel on võimalik küsida väärtus (agregaat). Selline implementatsioon on väga sarnane räsitabelile või sõnastikule. Andmed on sellises andmemudelis skeemita, kuid andmete sisu on nähtamatu, st agregaate saab eristada ainult võtme järgi aga mitte sisu järgi. Võti-väärtus andmemudel on üsna efektiivne ja lihtne. (8)

Võti-väärtus andmebaasis on soovitatav hoida kasutaja profiili, sessiooni, e-poe ostukäru jms andmeid. Sellisest andmebaasist tuleks hoiduda, kui andmeid tuleb pärida andme sisu järgi või on vajalik luua seoseid andmete vahel. (9)

Võti-väärtus andmebaasisüsteemide näited: Redis, Memcached, Amazon DynamoDB, Riak. (10)

Dokumendi andmebaasisüsteemis saab hoida ja pärida dokumente, mis võivad olla XML, JSON, BSON või muus formaadis. Need dokumendid on iseennast kirjeldavad, hierarhilised puustruktuurid mis võivad koosneda sõnastikest, kollektsioonidest ja skalaarsetest väärtustest (11). (9) Kasutades võtmeväärtust dokumentide pärimiseks, on põhimõte väga sarnane võti-väärtus tüüpi süsteemile. (8) Dokumendi andmebaasisüsteemid on kasulikud blogimiseks, veebianalüütikas, e-kommertsis jt (9).

Dokumendi andmebaasisüsteemide näited: MongoDB, CouchDB, Couchbase, Amazon DynamoDB. (12)

Graafi andmemudelis erinevad teistest olulisel määral (8). Need võimaldavad salvestada olemeid ja seoseid nende vahel. Olemeid nimetatakse ka sõlmedeks. Seoseid nende vahel nimetatakse kaarteks. (9)

SQL ja enamik NoSQL süsteeme on orienteeritud suurtele agregaatidele ja vähestele suhetele nende vahel. Graafisüsteemid lahendavad aga probleemi, kus agregaadid on väikesed, kuid seotud väga paljude teistega. (8) Näiteks sotsiaalmeedias on üks kasutaja seotud sadade teiste

kasutajatega, huvidega, üritustega jne. Lisaks sotsiaalmeediale sobib graafi andmebaas veel geoandmebaasiks, kaupade ning raha teekonna info hoidmiseks jms (9).

Graafi andmebaasisüsteemide näited: Neo4j, OrientDB, Titan, ArandoDB. (13)

Veergude perekondade andmemudelis on võimalik veeruperekonna ridu esitada veergude kogumina (14). See tähendab, et andmeid esitatakse uues veerus, mitte uues reas ja iga rida esitab erinevaid andmeid. Teise võimalusena on võimalik kasutada relatsioonilisele sarnast mudelit, kus iga rida esitab uusi andmeid ja veergude tüübid on määratletud. (8)

2.2 Veergude perekondade andmemudel

Veergude perekonna andmebaasisüsteemid hoiavad andmeid veeru perekondades ridadena, millel on mitmeid reavõtmega seotud veerge. Veeruperekonnad on seotud andmete grupid, mida tihti päritakse koos. Näiteks kliendi andmete puhul päritakse samaaegselt tihti tema profiili andmeid, aga mitte tellimusi. (9)

Iga veeruperekonda saab võrrelda SQL-andmebaasisüsteemi tabeliga, mis hoiab endas võtmega identifitseeritud ridu, mis koosnevad veergudele vastavatest väljadest. Erinevus on selles, et erinevatel ridadel ei pea veeruperekonna mudelis olema samu veerge ja neid saab lisada suvalisele reale teistele ridadele lisamata. (9)

Kui veerg koosneb veergude sõnastikust/räsitabelist, siis see on superveerg. Superveerg koosneb nimest ja väärtusest, mis on veergude sõnastik. Mõtle superveerust kui veergude konteinerist. (9) Superveerud on head, et hoida seotud andmeid koos, kuid kui mõnda veergu pole enamus ajast vaja, laetakse ja serialiseeritakse need ikkagi (8).

Veergude perekonna andmebaasisüsteem sobib blogimise platvormiks, sisu(andmete) haldamise süsteemi, rohketeks kirjutamisoperatsioonideks nagu näiteks logide salvestamine. Veergude perekonna andmebaasisüsteemi tuleks vältida, kui rakendus on varajases arenguetapis ja puuduvad kindlad andmete lugemise mustrid. (9)

Veergude perekondade andmebaasisüsteemide näited: **Cassandra**, HBase, Accumulo, Hypertable. (15)

2.3 Cassandra

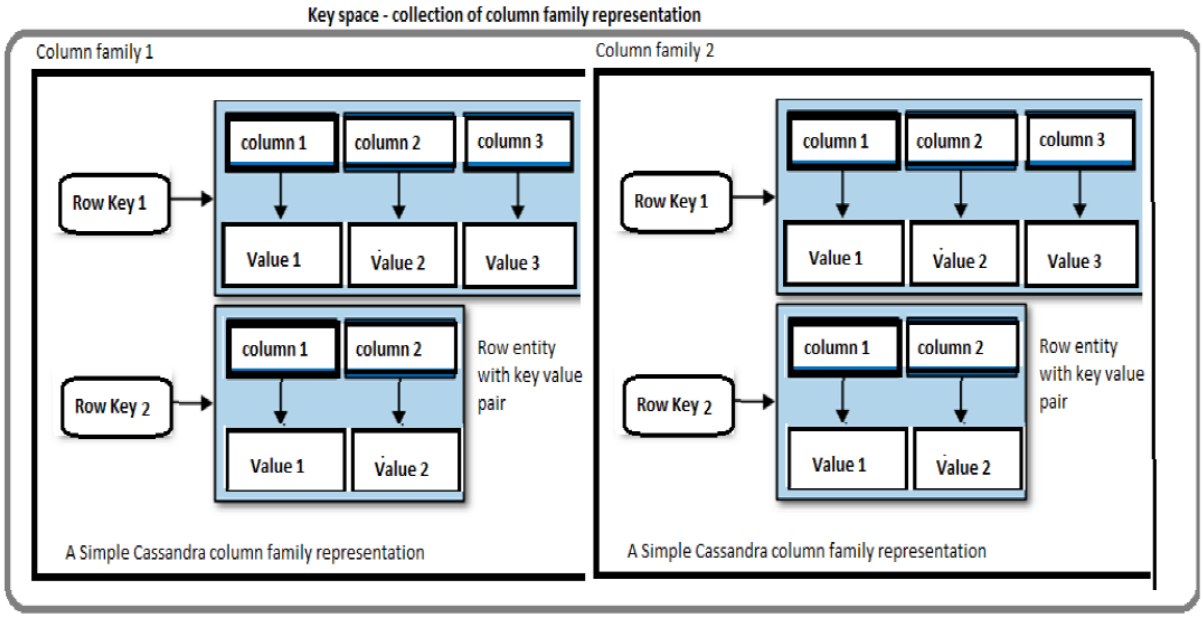
Kokkuvõtlikult on Apache Cassandra avatud lähtekoodiga, jagatud partnervõrguline andmebaasi arhitektuur, detsentraliseeritud, kergesti skaleeritav, veakindel, käideldav, terviklik, skeemivaba, veerule-orienteeritud andmebaasisüsteem. Cassandra andmebaas võib olla jaotatud mitme arvutivõrgu sõlme vahel. Tavaliselt ülem-alluv süsteemides seiskub ülemsõlme töö lõppedes kogu süsteem. Cassandra kasutab aga partnervõrgulist jaotuse mudelit, sellist, kus iga sõlm on struktuuriliselt võrdväärne iga teise sõlmega – ehk puudub ülemsõlm, mis käitub erinevalt, kui alluvsõlmed. Cassandra disaini eesmärk on üldine süsteemi käideldavus ja lihtne skaleeritavus. Cassandra andmemudel sisaldab võtmeruumi (nagu andmebaas SQL-andmebaasisüsteemis) ja veergude perekondi(tabelid). Cassandra defineerib veergude perekonna kui loogilise jaotuse, mis seostab sarnaseid andmeid. (16)

Relational Model	Cassandra Model
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

Joonis 1. Relatsioonilise ja Cassandra andmemudeli võrdlus
(Viide: (17))

Põhilised Cassandra andmestruktuurid on: veerg, mis on nime/väärtuse paar ning kliendipoolne uuendamise ajatempel, ja veergude perekond, mis on ridade konteiner, millel on sarnased, kuid mitte identsed veerud. (16)

Klaster on võtmeruumide konteiner(tavaliselt üks võtmeruum). Võtmeruum on Cassandras kõige välisem konteiner andmetele. Veergude perekond hoiab endas sorteeritud ridade hulka, millest igaüks on sorteeritud hulk veergusid. (16)



Joonis 2. Cassandra lihtsustatud andmemudel

Viide: (16)

Cassandra tagab püsivuse kasutades *commit* logisid, mis on krahhitaaste mehhanism. Kirjutusi ei loeta püsivuse tagamise eesmärgil õnnestunuks enne, kui andmed on kirjutatud *commit* logisse. Pärast *commit* logisse kirjutamist kirjutatakse väärtus mälus olevasse andmestruktuuri nimega *memtable*. *Memtable'i* täitudes objektidega, uhutakse selle sisu kõvakettal olevasse faili nimega *SSTable* ja luuakse uus *memtable*. Pärast *memtable'i* uhtumist kõvakettale *SSTable'ks* on see täielikult muutumatu. (16)

Kirjutamisoperatsioonid on Cassandras väga kiired, sest selle disain ei vaja kettalt lugemisi ega otsimisi. Tänu *memtable'ile* ja *SSTable'ile* ei pea Cassandra kirjutamisel neid operatsioone tegema, mis väga paljusid andmebaasisüsteeme aeglasemaks teevad. Kõik kirjutamisoperatsioonid on *append-only* ehk andmeid kirjutatakse alati vanade andmete järele (andmetele lisaks, mitte ei kirjutata andmeid üle). (16)

Cassandra parim omadus on muudetav terviklikkuse aste, mis laseb kasutajal seda vastavalt nõutele määrata. Kõrgem terviklikkuse aste tähendab, et rohkem sõlmi peavad päringule vastama, andes rohkem kindlust, et väärtused igal koopial (sõlmel) ühtivad. Kui kaks sõlme vastavad erineva ajatempliga, võidab kõige uuem väärtus ja see tagastatakse kliendile. Taustal toimub aga *read-repair* operatsioon, mis saab teate aegunud andmete kohta ja vastavalt uuendab terviklikkuse tagamiseks koopiaid (sõlmesid) kõige uuema väärtusega. (16)

Cassandra terviklikkuse astmeid on mitmeid, kuid enim tähelepanu väärivad järgmised kolm:

- ONE – Lugemisel tagastatakse lähima sõlme väärtus, kirjutamisel veendutakse, et vähemalt üks sõlm on salvestanud andmed *commit* logisse.

- QUORUM – Lugemisel tagastatakse kõige uuemad andmed 51% sõlmedest, kirjutamisel veendutakse, et vähemalt 51% sõlmedest on salvestanud andmed *commit* logisse. *Quorum* ehk kvoorum on (N/2 + 1), kus N on serverite arv.

- ALL – Lugemisel tagastatakse kõige uuemad andmed kõigilt sõlmedelt, kirjutamise veendutakse, et kõik sõlmed on salvestanud andmed *commit* logisse.

Kirjutamisel saadetakse andmed kõigile sõlmedele, kuid kirjutamine loetakse õnnestunuks siis, kui terviklikkuse aste on rahuldatud, isegi kui mõnele sõlmele pole jõutud veel kirjutada või see ebaõnnestub. (18)

Cassandras on võimalik andmebaasist lugemiseks ja andmebaasi kirjutamiseks kasutada andmebaasikeelt CQL. Cassandras, erinevalt relatsioonbaasihaldurist, pole võimalik jõustada välisvõtme kitsendusi ja ühendada veeru perekondi (*join* operatsioon). Cassandra töötab kõige tõhusamalt, kui andmemudel on denormaliseeritud (suure andmete liiasusega). Cassandral puudub otsene andmete uuendamise operatsioon, mis tähendab, et puudub *update* operatsioon. *Insert* operatsioon võtmele, mis juba eksisteerib, tähendab samanimeliste veergude ülekirjutamist; kui päringus on veerge, mis selles reas puuduvad, siis lisatakse need, seega topelt võtmed pole võimalikud. (16)

3. Analüüs

Selles peatükis esitatakse valik veebirakendusi pakkuva ettevõtte infosüstemi tellimuste allsüsteemi detailanalüüsi mudelitest.

3.1 Tervik süsteemi ülevaade

Terviksüsteemiks on veebilehtede arendamise ja hooldamise allüksuse infosüsteem. Tarkvara võimaldab erinevatel töötajatel täita tööülesandeid vastavalt rollile.

3.1.1 Organisatsiooni eesmärgid

Pakkuda klientidele võimalikult professionaalseid ning dünaamilisi lahendusi. Teenida kasumit.

3.1.2 Infosüsteemi eesmärgid

- Saada ülevaade ettevõttega seotud isikutest(töötajad, kliendid)
- Saada ülevaade pakutavatest teenustest
- Saada ülevaade pakutavate teenuste hinnakirjast
- Saada ülevaade tellimustest igas seisundis
- Saada ülevaade tellimuse seisundist
- Saada ülevaade teenustega seotud töötajatest
- Saada ülevaade tellimustega seotud teenustest
- Saada ülevaade tellimustega seotud arvetest

3.1.3 Funktsionaalsed allsüsteemid

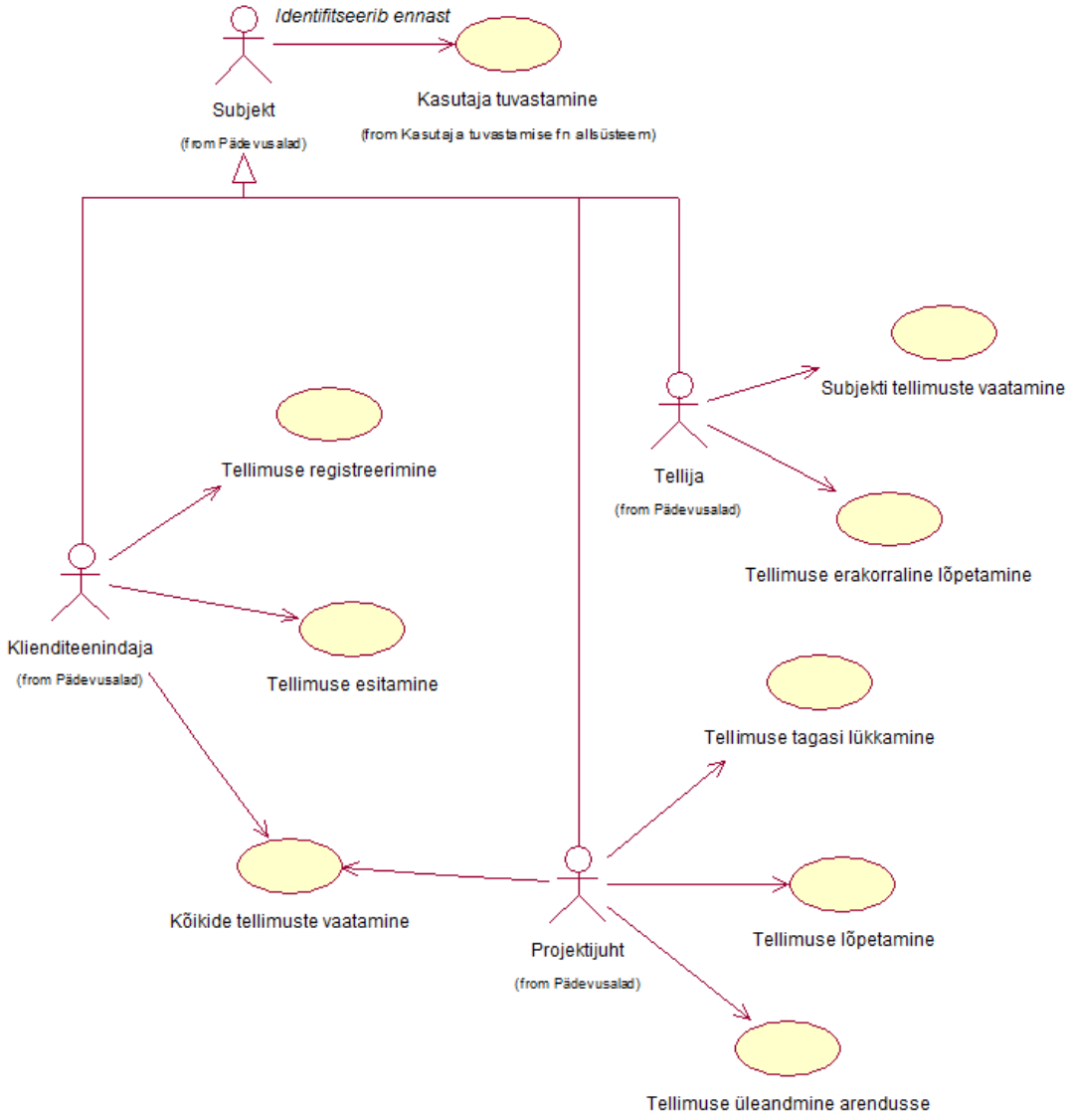
- Klientide funktsionaalne allsüsteem
- Töötajate funktsionaalne allsüsteem
- Isikute funktsionaalne allsüsteem
- Teenuste funktsionaalne allsüsteem
- **Tellimuste funktsionaalne allsüsteem**
- Arvete funktsionaalne allsüsteem
- Vara funktsionaalne allsüsteem
- Ülesannete funktsionaalne allsüsteem
- Klassifikaatorite funktsionaalne allsüsteem

3.2 Tellimuse funktsionaalse allsüsteem

3.2.1 Allsüsteemi eesmärgid

- Võimaldab luua uut tellimust
- Võimaldab tellimust tühistada(enne kinnitamist)
- Võimaldab tellimust muuta(enne ja pärast kinnitamist)
- Võimaldab tellimuste seisundimuutuste andmete registreerimist
- Võimaldab tellimuste registreerimist
- Võimaldab tellimuste kinnitamist
- Ühegi andebaasi päringu tegemine ei tohi võtta kauem kui 3 sekundit

3.2.2 Allsüsteemi kasutusjuhtude eskiismudel



Joonis 3. Kasutusjuhtude diagramm

Kasutusjuht: Tellimuse registreerimine
Tegutsejad: klienditeenindaja
Kirjeldus: Klienditeenindaja sisestab andmed

Kasutusjuht: Tellimuse esitamine
Tegutsejad: Klienditeenindaja
Kirjeldus: Klienditeenindaja esitab tellimuse projektijuhile

Kasutusjuht: Tellimuse lõpetamine
Tegutsejad: Projektijuht
Kirjeldus: Projektijuht vormistab kõik tarkvara õiguslikud suhted kliendile

Kasutusjuht: Tellimuse tagasi lükkamine
Tegutsejad: Projektijuht
Kirjeldus: Projektijuht lükkab tellimuse tagasi, ei võta töösse

Kasutusjuht: Tellimuse üleandmine arendusse
Tegutsejad: Projektijuht
Kirjeldus: Projektijuht võtab tellimuse vastu, määrab arendajad ning annab tellimuse töösse.

Kasutusjuht: Tellimuse erakorraline lõpetamine
Tegutsejad: Projektijuht
Kirjeldus: Projektijuht lõpetab arenduses tellimuse erakorraliselt.

Kasutusjuht: Kasutaja tuvastamine
Tegutsejad: Klienditeenindaja, Projektijuht, Tellija
Kirjeldus: Subjekt identifitseerib ennast

Kasutusjuht: Kõikide tellimuste vaatamine
Tegutsejad: Klienditeenindaja, Projektijuht
Kirjeldus: Subjekt vaatab kõiki tellimusi

Kasutusjuht: Subjekti tellimuste vaatamine
Tegutsejad: Tellija
Kirjeldus: Subjekt vaatab kõiki enda tellimusi

3.2.3 Ärireeglid

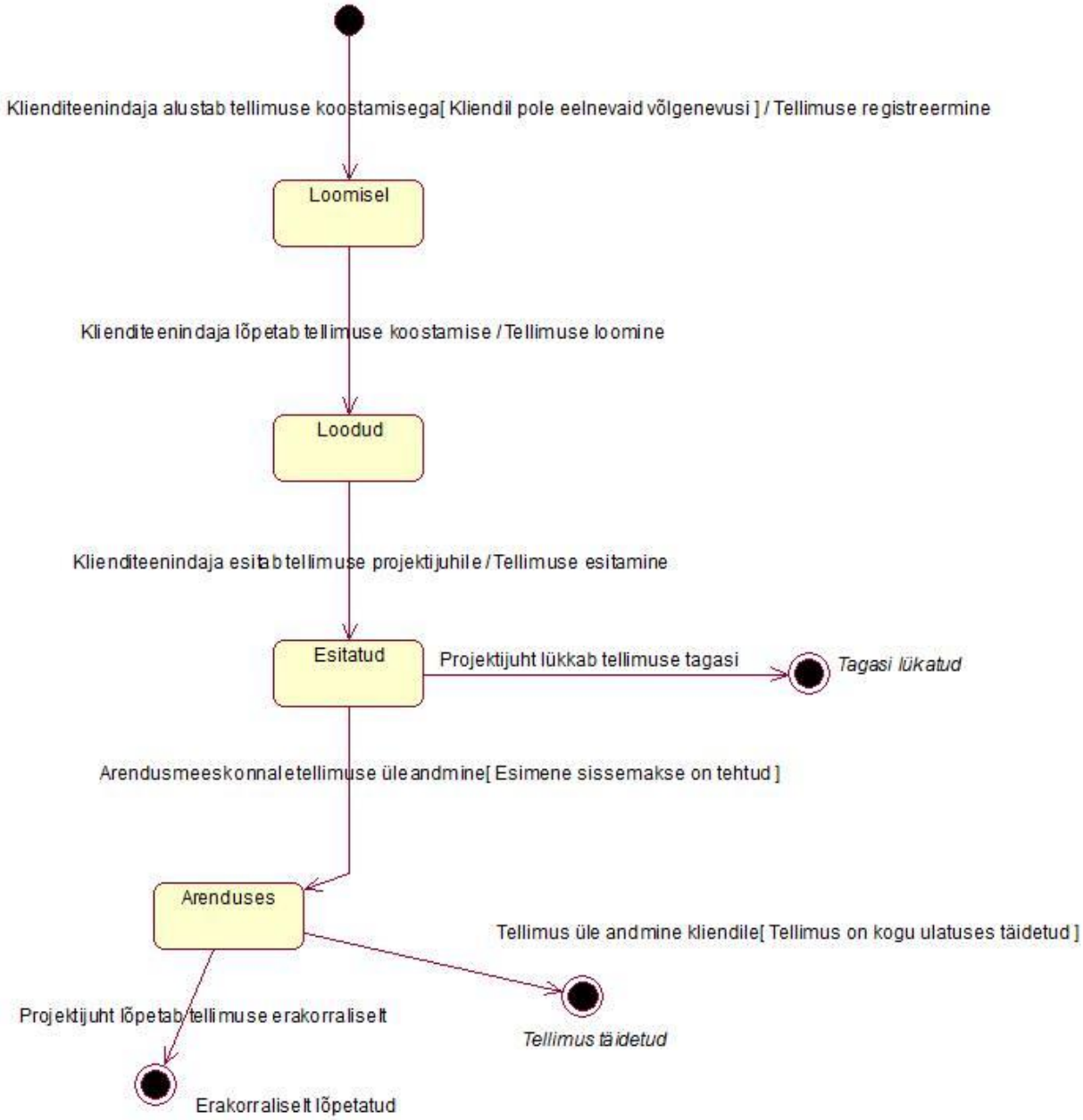
- Uue tellimuse vormistase alguseks ei tohi esineda võlgnevusi eelnevate tellimuste eest, kui ei ole määratud teisiti(Kliendi kasutaja on aktiveeritud olekus).
- Tellimuse tühistamisel kliendi poolt ettemaksu ei tagastata.
- Kui tellimust ei suudeta ettevõttest mitteolenevatel põhjustel tähtajaks täita, siis hilinemist kliendile ei hüvitata, vastasel juhul lepitakse hüvitamine kokku kliendiga individuaalselt.
- Projekt antakse kliendile üle, kui tellimus on täies ulatuses täidetud.
- Projekti tähtaeg saab olla maksimaalselt 10 aastat pärast tellimuse koostamise aega.

3.2.4 Mittefunktsionaalsed nõuded

Tabel 1. Süsteemi mittefunktsionaalsed nõuded

Tüüp	***Nõude kirjeldus***
andmebaasi-süsteem	Süsteem peab andmete hoidmiseks kasutama SQL-andmebaasisüsteemi.
arendusvahendid	Vabad kui ei ole teenusena tellitud teisiti.
keel	Kasutajaliides ja dokumentatsioon peavad olema eesti keeles.
kasutajaliides	Kasutajaliides peab olema veebi põhine.
töökiirus	Päringu tegemisel ei tohi vastuse kuvamine võtta rohkem aega kui 5 sekundit. Andmete muudatuste salvestamine ei tohi võtta rohkem aega kui 5 sekundit.
töökindlus	Taasteaja siht (*recovery time objective*)("maksimaalne talutav süsteemi käideldamatuse kestus pärast intsidenti" (19)): Kui tekib veaolukord ja andmebaas või rakendus kahjustub siis tuleb need taastada viimase tehtud varukoopia põhjal. Taasteseisu siht (*recovery point objective*)("intsidendijärgsele taastele seatud eesmärk ajahetkena, millele eelnevad andmed peavad olema täielikult taastatud (näiteks eelmine tund, eelmine tööpäev, eelmine nädal)" (19)):Maksimaalselt võivad kaotsi minna viimase 24 tunni andmed.
varukoopiad	Varukoopiaid tuleb teha iga nädal.
turvalisus	Kui parooli hoitakse andmebaasis, siis ei tohi see olla avatekst, vaid peab olema parooli räsiväärtus, mis on leitud selle parooli jaoks genereeritud soola kasutades.

3.2.5 Tellimuse seisundidiagramm

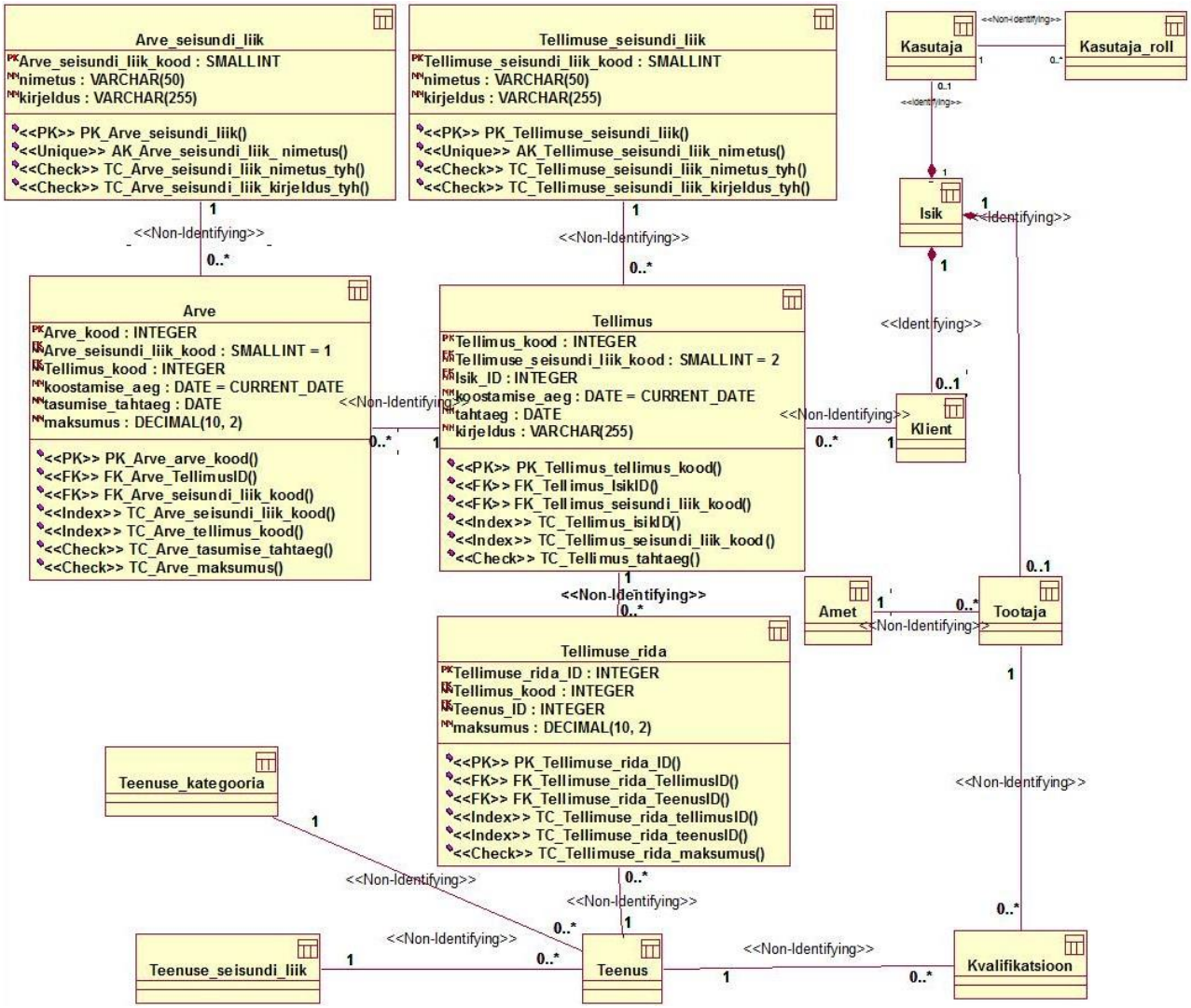


Joonis 4. Seisundidiagramm

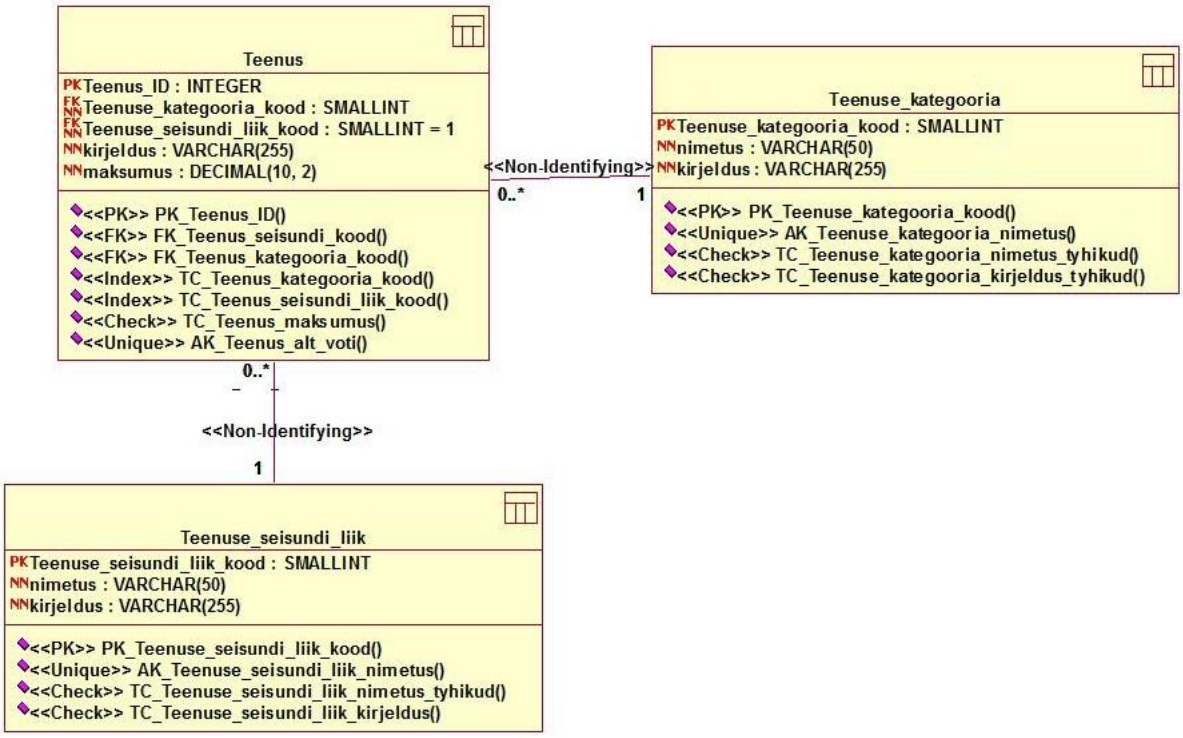
4. PostgreSQL andmebaasi disain

Selles peatükis esitatakse veebirakendusi pakkuva ettevõtte infosüsteemi tellimuste allsüsteemi kasutatava PostgreSQL andebaasi disaini kirjeldav mudel. See on loodud Rational Rose andmete modelleerimise moodulit kasutades.

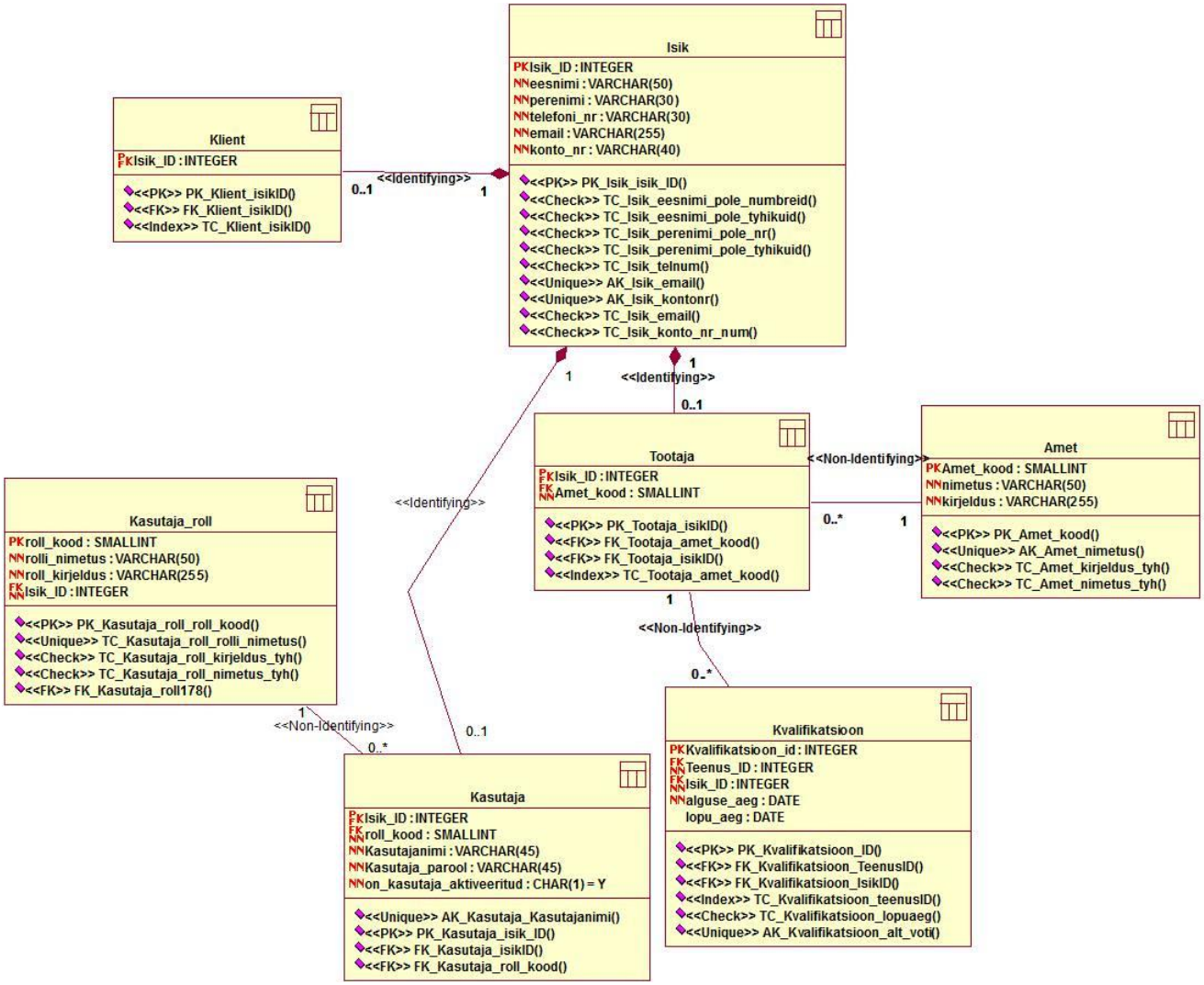
4.1 Andmebaasi diagramm



Joonis 5. Tellimuste andmebaasi diagramm



Joonis 6. Tellimuste andmebaasi teenuste alamdiagramm

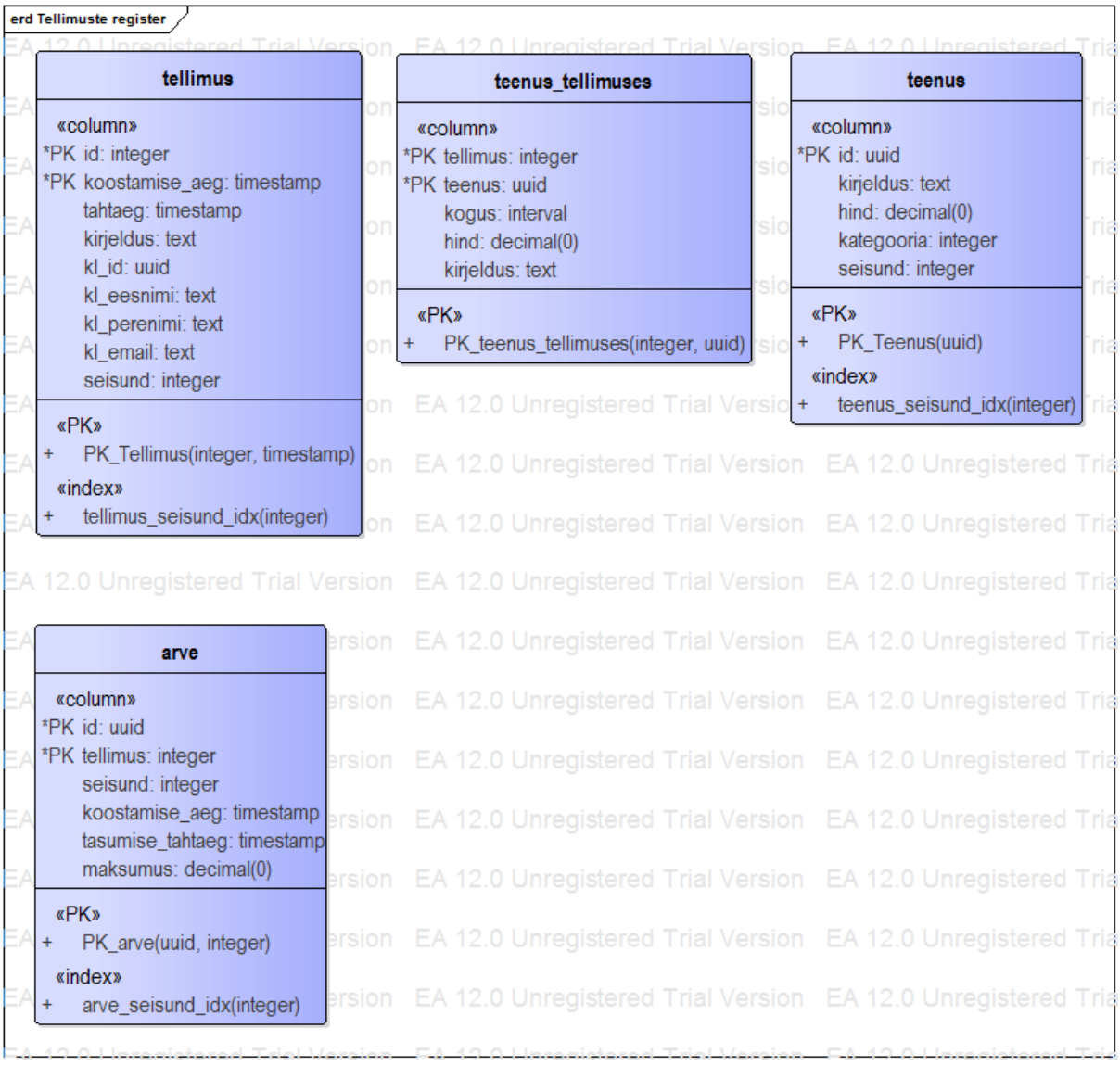


Joonis 7. Tellimuste andmebaasi isikute alamdiagramm

5. Cassandra andmebaasi disain

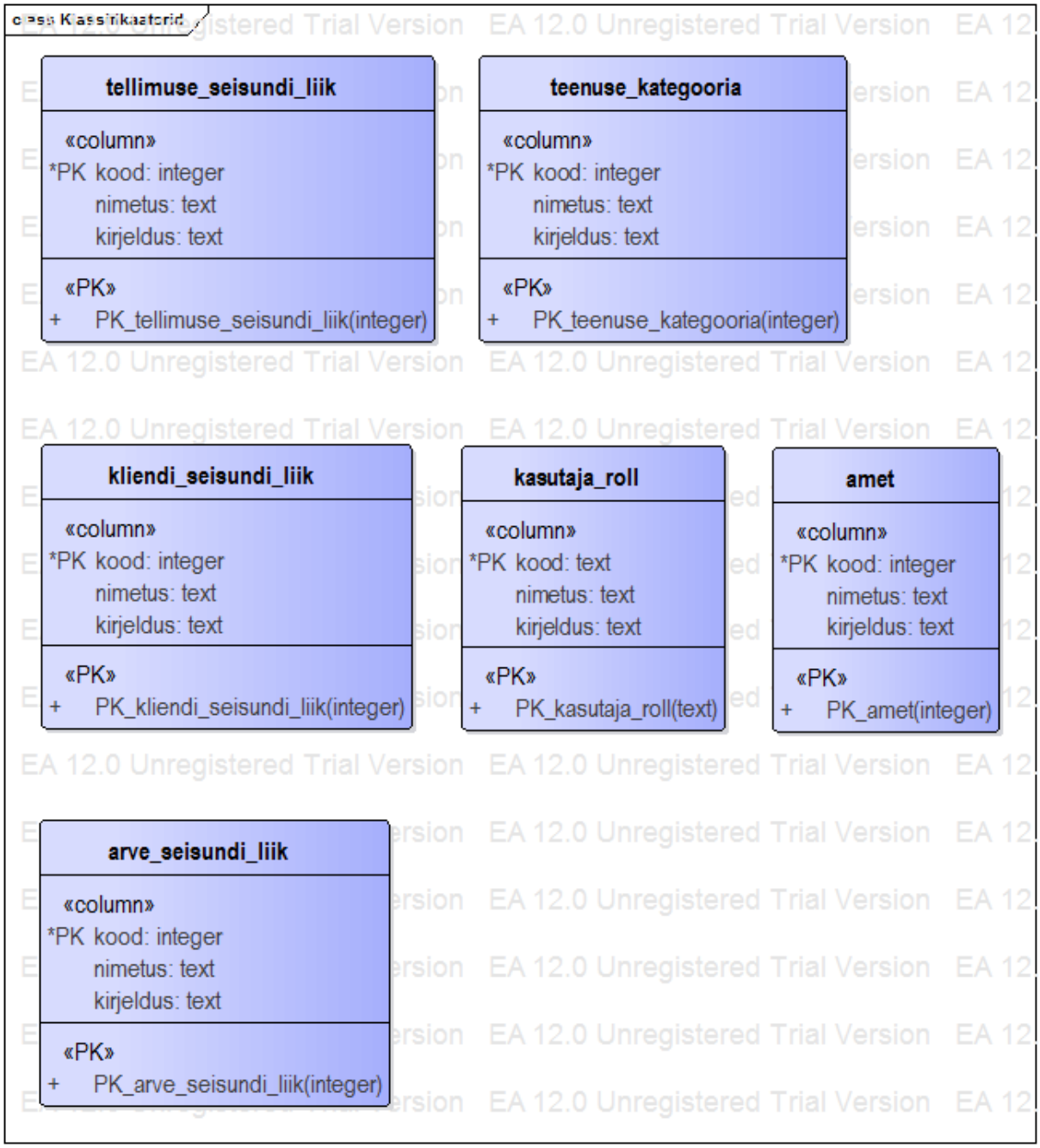
Selles peatükis esitatakse Cassandra andmebaasi disaini kirjeldav diagramm. See on koostatud Enterprise Architect vahendiga kasutades selle poolt pakutavat SQL-andmebaaside modelleerimise profiili. Seda on võimalik teha tänu sarnasustele SQLi aluseks oleva andemudeli ning veergude perekondade andmemudelite vahel. Iga kast tähistab veergude perekonda, PK tähistab primaarvõtit.

5.1 Andmebaasi diagramm



Joonis 8. Tellimuste register

Joonis 9. Isikute register



Joonis 10. Klassifikaatorite register

6. NoSQL andmete modelleerimine

NoSQL andmebaasid erinevad SQL-andmebaasidest: need on loodud töötama hajusalt, on skeemivabad (skeemi ei kirjeldata andmebaasi tasemel ilmutatud kujul) ja puuduvad ühendoperatsioonid (*join*). Need erinevused on põhjus, miks NoSQL andmebaaside andmemudel hülgab tabelistruktuuri, millega enamik arendajatest on harjunud ning mille põhjal on välja kujunenud relatsiooniline mõtlemine.

Rakendusele SQL-andmebaasi disainides peab arendaja analüüsima ainult olemeid ning seoseid nende vahel. Andmeid pärides saavad arendajad kasutada ühendoperatsiooni, et luua soovitud andmetest agregaate. NoSQL andmebaasi andmemudeli disainimisel, aga seda kasutada ei saa ja see teeb andmete andmebaasist küsimise keerukamaks. (20)

6.1 Teenustele orienteeritud andmebaasi mudel

Andmed NoSQL andmebaasis pole tagantjärgi kapseldatavad (ümberorganiseeritavad). See tähendab, et kui andmebaasi disain ei näe ette teatud andmete koos pärimist, siis on seda hiljem keeruline või võimatu saavutada. Tagajärjeks on aeglased päringud, mida pole võimalik optimeerida. Seetõttu on disaini etapp NoSQL andmebaaside loomisel isegi olulisem kui SQL baaside puhul.

SQL-andmebaasi modellerimist tuleks alustada kontseptuaalsest andmemudelist ja selle põhjal leida teisendusreeglite abil tabelite kirjeldused, sh olemite identifitseerimiseks ja suhete esitamiseks vajalikud primaar-, alternatiiv- ja välisvõtmed. Päringute tegemisele mõtlemine pole nii oluline. Eeldatakse, et hästi disainitud tabelitest on alati võimalik info kätte saada. (21)

Ka Cassandra puhul on andmete kontseptuaalne modelleerimine omal kohal, sest see aitab aru saada, milliseid andmeid on põhimõtteliselt süsteemis vaja säilitada. Andmebaasi disainimist ei alustata andmemudeliga, vaid päringute mudeliga (21). Selleks selgitatakse välja, milliseid päringuid rakendus teeb. Päringute järgi otsustatakse, milliseid andmeid tuleb koos hoida ning millised võib üksteisest eraldada – tekivad agregaadid. Cassandras pole vaateid, mis võimaldaksid samade lähteandmete põhjal erinevat tüüpi kasutajatele andmeid erineval viisil serveerida. Cassandra veeruperekondadest võib mõelda, kui salvestatud päringutulemustest

(mingil määral sarnased SQL süsteemide hetketõmmiste ehk materialiseeritud vaadetega). Päringute leidmiseks saab edukalt kasutada laiendatud formaadis kasutusjuhtude tekstikirjeldusi, millest tuleb leida andmete lugemisoperatsioonid.

Agregaat on „tükike“ seotud andmeid, millel on keerukas väärtus ning unikaalne identifikaator, mis on rakendusele andmete poole pöördumise ja muutmise üksus. Samuti on agregaadid tähtsad skaleeritavuse ja terviklikkuse seisukohalt, sest pakuvad loomulikku andmete jaotust andmete killustamiseks ja atomaarseks manipulatsiooniks hajusates keskkondades. Agregaate tuleks disainida kui üksust, kus atomaarsus peab olema garanteeritud. (22)

Kui atomaarsus on tagatud, siis see tähendab, et kas kõik järjestatud andmebaasioperatsioonid tehakse edukalt või ebaõnnestuvad kõik (23).

Kuna enamikes NoSQL süsteemides(k.a. Cassandras) puuduvad mitmeoperatsioonilised transaktsioonid (24), on atomaarsed agregaadid väga olulised atomaarsuse tagamiseks.

Kui agregaadid on loodud vastavalt rakenduse poolt teostatavatele päringutele, võime olla kindlad, et kulukate lugemisoperatsioonide arv on viidud miinimumini ja päringud on optimaalsed. Selle asemel, et teha mitu päringut ja rakenduse tasemel andmeid siduda, piisab ühest päringust, et saada kätte kõik olulised andmed.

Selline lähenemine toob endaga kaasa andmete denormaliseerimise ja liiasuse. Kui SQL-andmebaaside puhul üritatakse neid vältida, siis NoSQL süsteemide puhul on andmete denormaliseerimine üks disaini lahutamatu osa. Kuna kirjutamisoperatsioonid on Cassandras väga odavad ja lugemiseoperatsioonid kallid, siis tuleb esimesi eelistada teisele. See tähendab, et samu andmeid kirjutatakse erinevatesse agregaatidesse kuhu vaja, et hiljem oleksid need andmed kõigis ühe päringuga koheselt kättesaadavad.

Selline lähenemine on autori hinnangul loomulikum, kui relatsiooniline. SQL-andmebaasis tekivad mitu-mitmele suhete puhul tabelid, kus hoitakse ainult võtmeid. See on päris maailmast kauge ning ebaloomulik kõrvalnäht. (21)

Mitterelatsiooniline andmemudel pakub aga autori hinnangul arendajatele lihtsa mooduse töödelda kogu olulist informatsiooni ühe tervikuna, nii nagu see on ka päris maailmas. Muidugi, kui tuleb teha uusi programme, mis peavad lugema samu andmeid, kuid tegema seda teistsuguste päringute mustritega, muutub olukord keerulisemaks, sest andmebaas on orienteeritud ühe konkreetse rakenduse vajadustele. Siis tuleks mõelda nende uute programmide jaoks uute andmebaaside loomisele ja sinna vajalike andmete dubleerimisele

7. Migreerimise protsess

Antud rakendus on PostgreSQL andmebaasiga juba realiseeritud. Sellisel juhul on võimalik ka rakenduse andmepääsukihi päringute järgi Cassandra andmebaasi disainida. Uurimistöö raames lähtume siiski detailanalüüsis kirjeldatud kasutusjuhtudest.

7.1 Realiseeritav töökoht

Kõik järgnevalt kirja pandud kasutusjuhud implementeeritakse programmikoodis kasutades Cassandra andmebaasi.

Projektijuhi töökoht

7.2 Reaalsed kasutusjuhud

Punases OP – lugemisoperatsioon

Sinises OP – kirjutamisoperatsioon

Kasutusjuht: Kõikide tellimuste vaatamine

Primaarne tegutseja: Klienditeenindaja, Projektijuht

Osapooled ja nende huvid:

- Subjekt: Soovib vaadata kõiki tellimusi

Käivitav sündmus: Subjektil tekib infovajadus tellimus(t)e kohta

Eeltingimused: Töötaja on registreerunud kasutajaks, kasutajal on õigused kõiki tellimusi vaadata.

Järeltingimused: Subjektile kuvatakse kogu info tellimuste kohta.

Stsenaarium(tüüpiline sündmuste järjestus):

1. Töötaja soovib infot kõigi tellimuste kohta
2. Süsteem kuvab: tellimuse kood, kliendi eesnimi, perenimi, email, tellimuse kogusumma, kirjeldus, koostamise aeg, tähtaeg, tellimuse_seisundi(**OP1.1**)

Kasutusjuht: Tellimuse üleandmine arendusse

Primaarne tegutseja: Projektijuht

Osapooled ja nende huvid:

- Projektijuht: Soovib, et tellimust hakataks arendama

Käivitav sündmus: Tellimus on esitatud Projektijuhile

Eeltingimused: Tellimus on seisundis „esitatud“, tellimus on korrektne

Järeltingimused: Tellimus on seisundis „arenduses“

Stsenaarium(tüüpiline sündmuste järjestus):

1. Süsteem kuvab „esitatud“ seisundis tellimused (**OP2.1**)
2. Projektijuht valib tellimuse mille seisundit soovib muuta
3. Süsteem kuvab tellimuse detailvaate (**OP2.2**)
4. Projektijuht ei leia tellimuses puudusi
5. Projektijuht valib rippmenüüst tellimuse uue seisundi „arenduses“
6. Vajutab nupule „Salvesta“
7. Süsteem uuendab tellimuse seisundit (**OP2.3**)

Laiendused (või alternatiivne sündmuste käik):

4a. Projektijuht leiab tellimuses puudusi

5a. Projektijuht valib rippmenüüst tellimuse uue seisundi "tagasi lükatud“

Kasutusjuht: Tellimuse lõpetamine

Primaarne tegutseja: Projektijuht

Osapooled ja nende huvid:

- Projektijuht: Soovib tellimust edastada kliendile
- Klient: Soovib oma tellimust kätte saada

Käivitav sündmus: Tellimus on kogu ulatuses täidetud

Eeltingimused: Arved on tasutud, Tellimus on seisundis „arenduses“

Järeltingimused: Tellimus on seisundis „täidetud“, tellimus on kliendile edastatud

Stsenaarium(tüüpiline sündmuste järjestus):

1. Süsteem kuvab „arenduses“ seisundis tellimused (**OP2.1**)
2. Projekti juht valib tellimuse mille seisundit soovib muuta
3. Süsteem kuvab tellimuse detailvaate (**OP2.2**)
4. Projektijuht valib rippmenüüst tellimuse uue seisundi „täidetud“
5. Vajutab nupule „Salvesta“
6. Süsteem uuendab tellimuse seisundit (**OP2.3**)

Kasutusjuht: Tellimuse erakorraline lõpetamine

Primaarne tegutseja: Projektijuht

Osapooled ja nende huvid:

- Projektijuht: Soovib tellimust erakorraliselt lõpetada

Käivitav sündmus: Tellimuse täitmine ei ole võimalik

Eeltingimused: Tellimus on seisundis „arenduses“

Järeltingimused: Tellimus on seisundis „erakorraliselt lõpetatud“

Stsenaarium(tüüpiline sündmuste järjestus):

1. Süsteem kuvab „arenduses“ seisundis tellimused (**OP2.1**)
2. Projekti juht valib tellimuse mille seisundit ta soovib muuta
3. Süsteem kuvab tellimuse detailvaate (**OP2.2**)
4. Projektijuht valib rippmenüüst tellimuse uue seisundi „erakorraliselt täidetud“
5. Vajutab nupule „Salvesta“
6. Süsteem uuendab tellimuse seisundit (**OP2.3**)

7.3 Teenustele orienteeritud disain kasutuslugude näitel

1) Operatsioon **OP1.1** *Kõikide tellimuste kuvamine*

Tellimused

KOOD	KIRJELDUS	KLIENT	KOOSTATUD	TÄHTAEG	MAKSUMUS	SEISUND
53	tellimus 1 kirjeldus	Peeter Termomeeter(peeter@termomeeter.ee)	31/05/2014	31/10/2014	26000.00€	loodud
54	tellimus 2 kirjeldus	Karl Kaalikas(karl@kaalikas.ee)	31/05/2014	30/11/2014	24500.00€	loodud
56	tellimus 4 kirjeldus	Karl Kaalikas(karl@kaalikas.ee)	31/05/2014	01/05/2015	16500.00€	esitatud
55	tellimus 3 kirjeldus	Deodora Vahtel(deodora.vahtel@gmail.com)	01/06/2014	28/12/2014	16000.00€	loodud

Joonis 11. Kõik tellimused

Veeruperekond: **tellimus**

Kuvatakse kõik tellimused.

Kasutusjuhus on kirjeldatud vajalikud väärtused iga tellimuse kuvamiseks. Nende väärtuste hulka kuuluvad andmed tellimuse ja selle tellinud kliendi kohta. Selleks, et mitte teha kahte päringut, ühte tellimuse ning teist kliendi andmete saamiseks, otsustasin luua tellimuse agregaadi, mis on denormaliseeritud: tellimuse agregaat hoiab endas ka kliendi identifikaatorit, ees- ja perenime ning emaili, need andmed on dublikaat veeruperekonnast *klient*.

```
SELECT * FROM tellimus;
```

2) Operatsioon **OP2.1** *Tellimuse üle andmine arendusse*

Veeruperekond: **tellimus**

Kuvatakse kõik tellimused seisundis „**esitatud**“(või mõnes muus seisundis).

Kuvatakse samu andmeid, mis eelmise kasutusloo korral, kuid päringu tulemuseks peavad olema ainult esitatud tellimused. Cassandra CQL päringukeel lubab andmeid otsida ainult primaatvõtme(te) järgi (25). Selleks, et andmeid filtreerida mõne teise veeru järgi, tuleb see veerg indekseerida (26). Loon indeksi nimega *tellimus_seisund_idx*.

```
CREATE INDEX tellimus_seisund_idx ON tellimus(seisund);
```

Indekseerida ei soovitata selliseid välju, mis on erinevatel kirjetel väga erinevad: see muudab päringud väga aeglaseks ja tuleks kaaluda andmete denormaliseerimist või muud meetodit indekseerimise asemel (27). Väga hästi sobib aga indekseerimiseks klassifikaatorväärtus, mis ei muutu palju ja mida pole palju erinevaid, nagu näiteks tellimuse seisund.

```
SELECT * FROM tellimus WHERE seisund = ?;
```

3) Operatsioon **OP2.2** *Tellimuse üleandmine arendusse*

Joonis 12. Tellimuse detailvaade

Veeruperekond: **tellimus, teenus_tellimuses**

Kuvatakse tellimuse detailvaade, kus on näha teenused tellimuses, kliendiinfo ning võimalus muuta tellimuse seisundit.

Tellimuse andmete kuvamiseks kasutatakse olemasolevat agregaati *tellimus*. See aga ei sisalda tellimuses sisalduvate teenuste infot. Loon selle jaoks uue agregaadi nimega *teenus_tellimuses*, mis kasutab liitvõtmena tellimuse ja teenuse primaarvõtmeid.

```
PRIMARY KEY(partitsioonivõti, klastrivõti)
```

```
PRIMARY KEY(tellimus, teenus)
```

Kui agregaadi primaarvõti on liitvõti, siis selle esimene veerg(tellimus) on partitsioonivõti. Sellel on omadus, et kõik read, millel on sama võti, hoitakse samal füüsilisel sõlmel. Samuti toimuvad samade partitsioonivõtmetega ridade kõik sisestus-, uuendamis- ja kustutamisoperatsioonid atomaarselt ja isoleeritult. Teine veerg on klastrivõti, mida kasutatakse veergude grupeerimiseks reas. (28)

Klastrivõtme alusel sorteeritakse andmeid ühe partitsiooni piires (29). Partitsiooni- ja klastrivõti võivad koosned ka mitmest veerust.

Et ära hoida lisapäringut iga teenuse andmete küsimiseks, denormaliseerime jällegi andmeid: Lisame *teenus_tellimuses* veeruperekonnale veerud, mis dubleerivad veeruperekonna *teenus* andmeid: identifikaator, hind, kirjeldus.

```
SELECT * FROM teenus_tellimuses WHERE tellimus=? AND teenus=?;
```

Alternatiivne võimalus selle probleemi lahendamiseks on *teenus_tellimuses* veeruperekonda mitte luua vaid selle asemel hoida tellimuses olevaid teenuseid veeruperekonnas *tellimus*. Alates Cassandra 2.1 versioonist on võimalik luua kasutaja defineeritud andmetüüpe. Luues teenuste kohta andmetüübi *teenus_type*, saaksime kõiki tellimusega seotud teenuseid hoida tellimuse agregaadis sõnastikuna.

```
CREATE TYPE teenus_type (
     id uuid,
     kirjeldus text,
     kogus int,
     hind decimal
);
CREATE TABLE tellimus (
     id int,
     ...,
     teenused_tellimuses set<teenus_type>,
     ...
);
```

Sellise lahenduse korral ei pea tegema lisapäringut teenuste lugemiseks tellimuses, kuid samas laetakse ja serialiseeritakse kasutaja defineeritud andmetüüpide korral kõik selle veerud, mida alati aga ei kasutata. Kuna antud rakenduses kuvatakse teenuseid tellimuses ainult detailvaates, on mõistlikum siiski luua eraldi veeruperekond ja teha vastavalt vajadusele lisapäring.

3) Operatsioon OP2.3 *Tellimuse üleandmine arendusse*

Veeruperekond: **tellimus**

Uuendatakse tellimus kasutaja valitud seisundisse.

Tegemist on triviaalse uuendamisoperatsioniga, kus tellimuse kirje seisund uuendatakse.

```
UPDATE tellimus SET seisund = ? WHERE id = ?;
```

7.4 Rakenduse lähtekoodi muutmine

Java rakenduse lähtekoodis tuli ümber kirjutada andmepääsukihi klassid. See oli autori arvates suhteliselt vaevatu, kuna CQL on väga sarnane SQLile. Samuti on koodis kasutatud Datastaxi poolt arendatud Cassandra andmebaasisüsteemi draiver väga lihtne ja mugav. JDBC draiverit aga autor ei soovita kasutada, sest sellega tekkis probleeme: draiver ei tunnistanud osasid veeruperekondi võtmeruumi kuuluvaks, kuigi need olid loodud.

Samuti tuli muuta mõningate andmemudeli elementidele vastavate klasside primaarvõtmeid esitavate muutujate andmetüüpe, kuna need muutusid täisarvulisest uuid andmetüübiks.

Programmi koodis tuleb realiseerida andmete valideerimine, mis eelnevalt tehti andmebaasi tasemel. Kui kogu andmete valideerimine tehti enne andmebaasis, on selle realiseerimine programmi koodis ajamahukas, sest tuleb veenduda kõigi kitsenduste olemasolus. Antud ülesande raames polnud see aga väga keeruline, sest osa valideerimisest oli lisaks andmebaasi andmekitsendustele ka koodis juba eelnevalt realiseeritud.

7.5 Tähelepanekuid disainist kasutuslugude väliselt

1) Veeruperekonnad isik, klient ja töötaja

Veeruperekonnad *isik*, *klient* ja *tootaja* on kõik väga sarnased: need kõik sisaldavad informatsiooni seotud isiku kohta nagu nimi, email ja pangakonto number. Cassandra korral võib näiteks dubleerida üldised isikuandmed kõigis kolmes veeruperekonnas, sest see võimaldab andmeid klientide ja töötajate kohta teha ühe päringuga. See tähendab aga, et rakenduse koodis tuleb hoolitseda andmete terviklikkuse eest: kui üldiseid isikuandmeid

muudetakse, tuleb neid muuta ka *kliendi* ja *töötaja* veeruperekonnas. Hästi disainitud relatsioonilise andebaasi korral üritatakse sellist dubleerimist vältida.

Kui isikuandmed pole ühegi kliendi või töötaja päringu korral olulised, võiksime kaaluda ka ainult võtmetega tabelit, nagu relatsioonilises mudelis, kuid antud juhul pole selline lähenemine optimaalne.

Teise alternatiivina võiks hoida veeruperekonnas *isik* sõnastikku, kus hoitaks kõiki rolle, mis isikule kuuluvad – see hoiaks ära vajaduse veeruperekondade *klient* ja *tootaja* ning sellest tuleneva andmete liiasuse järele. Küll aga tekiks uus klassifikaatorite veeruperekond, mis neid rolle defineeriks.

```
CREATE TABLE isik (
     id uuid,
     eesnimi text,
     ...,
     rollid set<text>
);
```

Antud rakenduse kontekstis pole selline lähenemine aga optimaalne, sest erinevate rollide andmetega tehakse erinevaid tegevusi. Selleks, et välja selgitada, kas isik kuulub mingisse rolli, tuleb päringuga laadida kogu informatsioon isiku kohta ja programmi koodis selgitada välja, kas isikul on roll või mitte. Kuna tellimuse koostamisel on vajadus kuvada kõiki kliente, tähendaks see kõigi kirjete läbi vaatamist, mis on väga ebaefektiivne.

2) Süsteemikitsenduste puudumine

Cassandra andmebaasisüsteemil puuduvad nii välisvõtme- kui ka muud andmekitsendused (v.a primaarvõtmed). Pole võimalik määrata teenuse kirjelduse teksti maksimum pikkust ega kontrollida tellimuse seisundi vastavust klassifikaatoritele veeruperekonnas *tellimuse_seisundi_liik*. Samuti puudub võimalus määrata vaikimisi väärtusi, näiteks tellimuse koostamise ajale.

See ei tähenda, et Cassandra oleks ebaturvalisem, kuid andmeid peab valideerima täies ulatuses programmi koodis. Selline lähenemine on populaarsem uuemates arendustes, sest paljud on seisukohal, et andmebaas on vaid andmete säilitamiseks, mitte nende töötlemiseks.

Lisaks annab kitsenduste puudumine andmebaasile dünaamilisust, mis on kiiresti arenevate tarkvara juures väga oluline.

3) UUID unikaalse võtmena

Cassandra andmemudelis on süsteemse unikaalse identifikaatorina soovitatav kasutada UUID andmetüüpi kuuluvaid väärtusi. Enamus veeruperekondade unikaalsed identifikaatorid on käesoleva töö näites muudetud täisarvuliselt andmetüübilt UUID andmetüübile.

UUID on 128-bitine arv, mida saab kasutada olemi unikaalseks identifitseerimiseks. Sõltuvalt spetsiifilisest mehhanismist on UUID kas täiesti unikaalne või väga suure tõenäosusega unikaalne väärtus. (30)

8. Arutelu ja järeldused

NoSQL andmebaasisüsteemid on siin, et jääda – juba praegu on näha väga tormilist arengut ja populaarsuse kasvu – kuid mitte siin, et SQL süsteeme välja suretada. Nagu pole olemas universaalselt head tehnoloogiat, ei sobi ka Cassandra igasse projekti.

Cassandra sobib väga hästi projektidele, mis on läbi mõeldud ning jõudnud teatud stabiilsuseni. See tähendab, kui kasutuslood on teada ja need ei muutu eriti palju. See võimaldab andmeid päringute järgi modelleerida ning jõuda väga hea tulemuseni. Vastasel juhul tuleks kaaluda mõnda muud liiki NoSQL või SQL-andmebaasisüsteemi.

Cassandra veergude perekonna andmemudeli loomine ei ole lihtne. See nõuab rakenduse poolt käivitavate päringute mõistmist ning ka andmebaasisüsteemi ja selle CQL päringukeele tehniliste omaduste tundmist. Iga otsus on tarkvara edasise kvaliteedi ja eluea suhtes kriitilise kaaluga.

Migreerimisel ei saa kasutada ettekirjutatud norme, sest lähtuda tuleb konkreetsest rakendusest ja selle arhitektuurist. Kui arendaja tunneb süsteemi ja see on hästi dokumenteeritud, lihtsustab see oluliselt migreerimise protsessi. Siiski peab olema kursis Cassandra nn pudelikaeltega – võimalike jõudluse probleemidega, selleks, et neid vältida.

Kui andmemudel on loodud, peab kindlasti arvestama ka sellega, et kõik andmebaasi kitsendused tuleb realiseerida programmi koodis.

Antud uurimuses migreerimise protsessi oli edukas – rakendus töötab Cassandra andmebaasiga ning disainis pole ilmnenud vigu. Sellest järeldan, et SQL-andmebaasisüsteemilt on võimalik edukalt migreeruda Cassandra andmebaasisüsteemile nii, et tarkvara on terviklik ja käideldav.

Kuna Cassandra andmebaasi diagrammide loomiseks puudub standardne viis, siis töö käigus koostasin võimaliku UML kontseptsiooni Cassandra andmebaaside disaini esitamiseks, kasutades Enterprise Arhitect tarkvara PostgreSQL andmebaaside modelleerimise funktsionaalsust (selle andmetüübid sarnanevad enim Cassandra andmetüüpidele) (vt Joonis 8-Joonis 10).

9. Kokkuvõte

Uurimistöö annab põgusa ülevaate NoSQL süsteemidest üleüldiselt ja detailse kirjelduse veergude perekonda kuuluva Apache Cassandra tehnilistest omadustest. Seejärel migreeritakse tellimuste allsüsteemi tarkvara olemasolevalt PostgreSQL andmebaasilt Cassandra andmebaasile, analüüsides tehtud valikuid ning tuues alternatiive.

Uurimuse tulemusena koostasin Cassandra andmebaasi disaini kirjeldava diagrammi ning realiseerisin selle põhjal Java veebirakendusele DAO kihi. Diagrammi koostamise käigus kujundasin Cassandra andmemudeli esitamiseks sobiva UML lahenduse, kuna ametlik standard sellele puudub. Migreerimise tulemusena saavutasin töötava tarkvara hajutatava Cassandra andmebaasiga. Õppisin palju NoSQL andmebaasisüsteemide ning eriti Cassandra kohta.

Selleks, et kasutada Cassandra andmebaasisüsteemi, peab olema selgelt defineeritud ülesande püstitus ja detailanalüüs. Ilma selleta on võimalik Cassandra andmebaasi luua, kuid tõenäoliselt pole see jätkusuutlik ning tuleks kaaluda SQL süsteemi.

Antud dokument sobib hästi arendajatele, kes soovivad Cassanda andmebaasisüsteemi kasutada, kuid puudub kogemus NoSQL süsteemidega üleüldiselt või Cassandraga spetsiifiliselt. Teoreetilised ja praktilised näited aitavad mõista sellise andmemudeli omadusi ning erinevusi relatsioonilisest mudelist.

Eesmärgid saavutati täies ulatuses, lisaks saavutasin loomuliku protsessi tulemusena eelmainitud UML abil Cassandra andmebaasi modelleerimise kontseptsiooni.

Uurimistöö edasiarendusena on võimalik rakendada koormuse-, stressi- jms teste, et veenduda või ümber lükata konkreetse andmebaasi või Cassandra töövõimekust ja efektiivsust üleüldiselt ning kõrvutada seda SQL-andmebaasisüsteemidega ja teiste NoSQL andmebaasisüsteemidega. Cassandra tõeline tugevus on andmebaasi hajutamise võimalus mida on võimalik riistvara olemasolul rakendada, kuid mida käesolevas töös ei katsetatud.

Summary

This thesis gives a brief overview of NoSQL systems and a detailed technical description of a column-family store Apache Cassandra. As a result, a migration from an existing PostgreSQL database to Cassandra database is conducted while analyzing the choices and pointing out alternative approaches.

As a result of this study, I created a Cassandra database diagram and implemented a DAO layer for a Java web application on the basis of the diagram. Since there is no official standard for creating Cassandra database design diagrams, I constructed a custom UML solution to Cassandra data model. As a result of migration I created a working software with a distributed database. I learned a lot about NoSQL database management systems, especially about Cassandra.

In order to use a Cassandra database management system, one has to have a clear definition of the task and the results of detailed analysis. It is possible to create a Cassandra database without it, but most probably it will not be sustainable and a SQL system should be considered instead.

This document is most suitable for developers who wish to use Cassandra, but lack experience in NoSQL in general or in Cassandra particular. Theoretical and practical examples help us to understand the characteristics of this data model and differences from the relational model.

The goals of this thesis were fully achieved. In addition, a concept of how to model the design of Cassandra databases in UML was proposed as a result of the process.

Further research on this topic could include load, stress, and other tests to prove or overturn the effectiveness of this database in particular or Cassandra in general. One should compare the results with the results in case of SQL and other NoSQL database management systems. The real strength of Cassandra is a fully distributed database, which can be applied when you have the needed hardware. The present work did not consider the distribution aspect.

Kasutatud kirjandus

1. [Võrgumaterjal] [Tsiteeritud: 18. aprill 2015. a.] http://www.vallaste.ee/.
2. [Võrgumaterjal] [Tsiteeritud: 28. aprill 2015. a.] https://cassandra.apache.org/doc/cql3/CQL.html#CQLSyntax.
3. [Võrgumaterjal] 9. märts 2013. a. [Tsiteeritud: 28. aprill 2015. a.] http://et.wikipedia.org/wiki/Avatud_l%C3%A4htekood.
4. [Võrgumaterjal] [Tsiteeritud: 28. aprill 2015. a.] https://et.glosbe.com/en/et/crash%20recovery.
5. **Gulutzan, Peter ja Pelzer, Trudy.** [Võrgumaterjal] 1999. a. http://www.ocelot.ca/glossary.htm.
6. [Võrgumaterjal] [Tsiteeritud: 15. mai 2015. a.] http://en.wikipedia.org/wiki/Universally_unique_identifier.
7. **Eessaar, Erki.** [Võrgumaterjal] detsember 2014. a. http://maurus.ttu.ee/aine_index.php?aine=328.
8. **P. J. Sadalage, M. Fowler.** *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* 2012.
9. **Sadalage, Pramod.** [Võrgumaterjal] 3. oktoober 2014. a. http://www.thoughtworks.com/insights/blog/nosql-databases-overview.
10. [Online] [Cited: aprill 4, 2015.] http://db-engines.com/en/ranking/key-value+store.
11. **Ekstrand, Michael.** [Võrgumaterjal] 8. juuli 2011. a. http://stackoverflow.com/questions/6623130/scalar-vs-primitive-data-type-are-they-the-same-thing.
12. [Võrgumaterjal] [Tsiteeritud: 4. april 2015. a.] http://db-engines.com/en/ranking/document+store.
13. [Võrgumaterjal] [Tsiteeritud: 4. aprill 2015. a.] http://db-engines.com/en/ranking/graph+dbms.
14. [Võrgumaterjal] [Tsiteeritud: 3. mai 2015. a.] http://curah.microsoft.com/58515/column-family-databases.
15. [Võrgumaterjal] [Tsiteeritud:] http://db-engines.com/en/ranking/wide+column+store.
16. *COMPARATIVE STUDY OF NOSQL DOCUMENT, COLUMN STORE DATABASES AND EVALUATION OF CASSANDRA.* **Manoj, J.** 4, s.l. : International Journal of Database Management Systems, 2014. a., Kd. 6.
17. [Võrgumaterjal] [Tsiteeritud: 30. aprill 2015. a.] http://www.ebaytechblog.com/wp-content/uploads/2012/07/analogy.png.
18. [Võrgumaterjal] [Tsiteeritud: 1. mai 2015. a.] http://planetcassandra.org/blog/consistency-cassandra/.
19. **Veldre, Anto, et al., et al.** [Võrgumaterjal] 2011-2012. a. [Tsiteeritud: 23. 05 2015. a.] http://akit.cyber.ee/.
20. *QODM: A query-oriented data modeling approach for NoSQL databases.* **Li, Xiang, Ma, Zhiyi ja Chen, Hongjie.** Beijing, China : IEEE, 2014. 14808579.
21. *The NoSQL Principles and Basic Application of Cassandra Model.* **Wang, Guoxi ja Tang, Jianfeng.** Shanghai, China : IEEE, 2012. 13227039.
22. **Bugiotti, Francesca, et al., et al.** [Võrgumaterjal] http://www.dia.uniroma3.it/~torlone/pubs/er22014.pdf.
23. [Võrgumaterjal] [Tsiteeritud: 13. mai 2015. a.] http://en.wikipedia.org/wiki/Atomicity_(database_systems)#cite_note-1.

24. **Scherer, Dave.** [Võrgumaterjal] 06. september 2013. a.
http://blog.foundationdb.com/those-are-not-transactions-cassandra-2-0.
25. [Võrgumaterjal] 19. detsember 2013. a. [Tsiteeritud: 4. mai 2015. a.]
http://planetcassandra.org/blog/flite-breaking-down-the-cql-where-clause/.
26. [Võrgumaterjal] 23. august 2014. a. [Tsiteeritud: 12. mai 2015. a.]
http://blog.websudos.com/2014/08/23/a-series-on-cassandra-part-2-indexes-and-keys/.
27. [Võrgumaterjal] [Tsiteeritud: 13. aprill 2015. a.]
http://docs.datastax.com/en/cql/3.1/cql/ddl/ddl_when_use_index_c.html.
28. **Morton, Aaron.** [Võrgumaterjal] 11. jaanuar 2013. a.
http://thelastpickle.com/blog/2013/01/11/primary-keys-in-cql.html.
29. **Bertuccini, Carlo.** [Võrgumaterjal] 25. juuli 2014. a.
http://stackoverflow.com/questions/24949676/difference-between-partition-key-composite-key-and-clustering-key-in-cassandra.
30. [Võrgumaterjal] [Tsiteeritud: 15. mai 2015. a.]
http://searchsoa.techtarget.com/definition/UUID.
31. **Sterling, Leon S.** *The Art of Agent-Oriented Modeling.* London : The MIT Press, 2009.
32. [Võrgumaterjal] [Tsiteeritud: 15. mai 2015. a.]
http://docs.datastax.com/en/cql/3.0/cql/cql_reference/uuid_type_r.html.

Lisa 1

Andmebaasi loomise skriptid

```
CREATE TABLE tellimus (
      id int,
      koostamise_aeg timestamp,
      tahtaeg timestamp,
      kirjeldus text,
      seisund int,
      maksumus decimal,
      kl_id uuid,
      kl_eesnimi text,
      kl_perenimi text,
      kl_email text,
      PRIMARY KEY(id)
);
CREATE INDEX tellimus_seisund_idx
ON tellimus(seisund);


CREATE TABLE teenus_tellimuses (
      tellimus int,
      teenus uuid,
      kogus int,
      hind decimal,
      kirjeldus text,
      PRIMARY KEY(tellimus, teenus)
);
```

```
CREATE TABLE teenus (
      id uuid,
      kirjeldus text,
      hind decimal,
      kategooria int,
      seisund int,
      PRIMARY KEY(id)
);
CREATE INDEX teenus_seisund_idx
ON teenus(seisund);


TABLE arve (
      id uuid,
      tellimus int,
      seisund integer,
      koostamise_aeg timestamp,
      tasumise_tahtaeg timestamp,
      maksumus decimal,
      PRIMARY KEY(id, tellimus)
);
CREATE INDEX arve_seisund_idx
ON arve(seisund);
```

```
CREATE TABLE isik (
      id uuid,

      eesnimi text,

      perenimi text,

      telefoni_nr text,

      email text,

      konto_nr text,

      kasutaja text,

      PRIMARY KEY(id)

);

CREATE INDEX isik_kasutaja_idx ON isik(kasutaja);



CREATE TABLE kasutaja (

      kasutajanimi text,

      parool text,

      aktiveeritud boolean,

      roll text,

      PRIMARY KEY(kasutajanimi),



);

CREATE INDEX kasutaja_aktiveeritud_idx

ON kasutaja (aktiveeritud);
```

```
CREATE TABLE klient(
      isik_id uuid,
      eesnimi text,
      perenimi text,
      telefoni_nr text,
      konto_nr text,
      seisund int,
      PRIMARY KEY(isik_id)
);
CREATE INDEX klient_seisund_idx ON klient (seisund);


CREATE TABLE tootaja (
      isik_id uuid,
      amet_id int,
      amet_nimetus text,
      amet_kirjeldus text,
      eesnimi text,
      perenimi text,
      email text,
      telefoni_nr text,
      konto_nr text,
      PRIMARY KEY(isik_id)
);
```

Kõik klassifikaator veeruperekonnad kasutavad sama skeemi:

```
CREATE TABLE %nimetus% (
      kood int,
      nimetus text,
      kirjeldus text,
      PRIMARY KEY(kood)
);
```

Kus **%nimetus%** on:

1) tellimuse_seisundi_liik
2) teenuse_kategooria
3) kliendi_seisundi_liik
4) kasutaja_roll
5) amet
6) arve_seisundi_liik

Lisa 2

Testandmete skriptid projektijuhi rollile

```
INSERT INTO tellimus(id, koostamise_aeg, tahtaeg, kirjeldus, seisund,
maksumus, kl_id, kl_eesnimi, kl_perenimi, kl_email)

VALUES (1, '2015-04-23', '2015-06-23', 'kirjeldus bla bla 1', 3, 6999.99,
74d49c4e-eaaa-11e4-b02c-1681e6b88ec1,          'Maali',          'Paali',
'maali.paali@gmail');

INSERT INTO tellimus(id, koostamise_aeg, tahtaeg, kirjeldus, seisund,
maksumus, kl_id, kl_eesnimi, kl_perenimi, kl_email)

VALUES (2, '2015-03-21', '2015-07-01', 'kirjeldus bla bla 2', 4, 10000.0,
ccf6d14c-eb33-11e4-b02c-1681e6b88ec1,          'Klient',          'Klient',
'klient.klient@gmail');



INSERT INTO teenus_tellimuses(tellimus, teenus, kogus, hind, kirjeldus)

VALUES (1, 707dbac0-eaac-11e4-9edd-456fc188f56e, 2, 6500.50, 'teenus 1
kirjeldus');

INSERT INTO teenus_tellimuses(tellimus, teenus, kogus, hind, kirjeldus)

VALUES (1, ed9bf3e0-eaad-11e4-9edd-456fc188f56e, 2, 4200, 'teenus 2
kirjeldus');

INSERT INTO teenus_tellimuses(tellimus, teenus, kogus, hind, kirjeldus)

VALUES (2, ed9bf3e0-eaad-11e4-9edd-456fc188f56e, 1, 4200, 'teenus 2
kirjeldus');

INSERT INTO teenus(id, kirjeldus, hind, kategooria, seisund)

VALUES (707dbac0-eaac-11e4-9edd-456fc188f56e, 'teenus 1 kirjeldus',
6500.50, -1, 1);



INSERT INTO teenus(id, kirjeldus, hind, kategooria, seisund)

VALUES (ed9bf3e0-eaad-11e4-9edd-456fc188f56e, 'teenus 2 kirjeldus', 4200, -
1, 1);



INSERT INTO isik(id, eesnimi, perenimi, telefoni_nr, email, konto_nr,
kasutaja)

VALUES (74d498c0-eaaa-11e4-b02c-1681e6b88ec1, 'Marten', 'Tall', '58502277',
'tall.marten@gmail.com', 'EE001100110011', 'admin');
```

```
INSERT INTO isik(id, eesnimi, perenimi, telefoni_nr, email, konto_nr,
kasutaja)

VALUES (74d49b0e-eaaa-11e4-b02c-1681e6b88ec1, 'Silver', 'Vaas', '58502277',
'silver.vaas@gmail.com', 'EE001100110011', 'service');

INSERT INTO isik(id, eesnimi, perenimi, telefoni_nr, email, konto_nr,
kasutaja)

VALUES (74d49c4e-eaaa-11e4-b02c-1681e6b88ec1, 'Maali', 'Paali', '58502277',
'maali.paali@gmail.com', 'EE001100110011', 'client');

INSERT INTO isik(id, eesnimi, perenimi, telefoni_nr, email, konto_nr,
kasutaja)

VALUES    (ccf6d14c-eb33-11e4-b02c-1681e6b88ec1,    'Klient',    'Klient',
'58502277', 'klient.paali@gmail.com', 'EE001100110011', 'client2');



INSERT INTO kasutaja (kasutajanimi, parool, aktiveeritud, roll)

VALUES                                                    ('admin',
'$2a$11$KHCZxuSKeg9DDq/FT2TQ9emO18w7L1w3dU5ahawvgf3CW8ZFw7OlS',       true,
'ROLE_ADMIN');

INSERT INTO kasutaja (kasutajanimi, parool, aktiveeritud, roll)

VALUES                                                  ('service',
'$2a$11$KHCZxuSKeg9DDq/FT2TQ9emO18w7L1w3dU5ahawvgf3CW8ZFw7OlS',       true,
'ROLE_SERVICE');

INSERT INTO kasutaja (kasutajanimi, parool, aktiveeritud, roll)

VALUES                                                   ('client',
'$2a$11$KHCZxuSKeg9DDq/FT2TQ9emO18w7L1w3dU5ahawvgf3CW8ZFw7OlS',       true,
'ROLE_CLIENT');

INSERT INTO kasutaja (kasutajanimi, parool, aktiveeritud, roll)

VALUES                                                  ('client2',
'$2a$11$KHCZxuSKeg9DDq/FT2TQ9emO18w7L1w3dU5ahawvgf3CW8ZFw7OlS',       true,
'ROLE_CLIENT');



INSERT  INTO  klient(isik_id,  eesnimi,  perenimi,  telefoni_nr,  konto_nr,
seisund)

VALUES(ccf6d14c-eb33-11e4-b02c-1681e6b88ec1,      'Klient',      'Klient',
'58502277', 'EE1111111111111', 1);
```

Lisa 3

Veebirakenduse lähtekood asub avalikus git repositooriumis.

https://bitbucket.org/martentall/loput.git