TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kajar Karuauk 211604IAPM

Ankurmudeli põhjal PostgreSQL jaoks SQL koodi koostava generaatori edasiarendamine

Magistritöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2023

Autorideklaratsioon

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

Autor: Kajar Karuauk

10.05.2023

Annotatsioon

Ankurmodelleerimine on andmebaaside arendamise metoodika, mis on mõeldud eeskätt andmeaitade loomiseks. Selle alusel loodud SQL-andmebaasides on tabelid kõrgel (kuuendal) normaalkujul ning tulemuseks saadud andmebaas pakub nii head operatsioonide töökiirust kui ka muid eeliseid nagu skeemimuudatuste halduse lihtsus ning hea toimetulek puuduvate andmetega ja vajadusega säilitada ajaloolisi andmeid. Ankurmodelleerimise jaoks on loodud avatud lähtekoodiga veebipõhine modelleerimisvahend ning koodigeneraator, millega on võimalik ankurmudelitest erinevatele andmebaasisüsteemidele sobivat SQL-koodi genereerida. Töö kirjutamise hetkel (2023. aasta kevad) toetas see keskkond viite erinevat andmebaasisüsteemi: Microsoft SQL Server, PostgreSQL, Oracle, Snowflake ja Vertica. Käesolev magistritöö keskendub PostgreSQL SQL-koodigeneraatori paremaks muutmisele. Magistritöö tulemusena selgitatakse välja parendamist ja parandamist vajavad kohad. Muuhulgas uuritakse ankurmudeli andmebaasi põhjal SELECT lausete täitmise kiirust, mille tulemused võivad tingida PostgreSQL SQL-koodigeneraatori muudatusi. Töö tulemusena tehakse muudatusi PostgreSQL SQL-koodigeneraatoris.

Magistritöö materjalid on kättesaadavad GitHubis-s aadressil https://github.com/kajargit/magister

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 48 leheküljel, 8 peatükki, 22 joonist, 10 tabelit.

Abstract

Further Development of a Anchor Model-Based SQL Code Generator for PostgreSQL

Anchor modelling is a database development methodology that is primarily designed to create data warehouses. The SQL databases created on this basis have highly normalized tables (in sixth normal form) and the resulting database offers good speed of operations. Moreover, it has other benefits such as the ease of schema change management, good handling of missing data, and the ease to preserve historical data. An open-source web-based modeling tool and a set of code generators have been created for anchor modeling. The latter are used to generate SQL code for different database management systems (DBMSs) based on anchor models. At the time of writing the thesis (spring 2023) this environment supports five different DBMSs: Microsoft SQL Server, PostgreSQL, Oracle, Snowflake, and Vertica. This master's thesis focuses on the PostgreSQL SQL code generator. Firstly, the things that need improvement in the generator will be identified. Among other things, the speed of execution of SELECT statements is investigated, the results of which may be an input for further changes to the PostgreSQL SQL code generator. As a result of this work changes will be made in the code generator.

The materials of the master thesis are available in GitHub: https://github.com/kajargit/magister

The thesis is in Estonian and contains 48 pages of text, 8 chapters, 22 figures, 10 tables.

Lühendite ja mõistete sõnastik

6NK	Kuues normaalkuju
API	*Application Programming Interface,* reeglite ja protokollide kogum, mis võimaldab erinevatel rakendustel omavahel suhelda.
CASE	*Computer Aided Software Engineering*, infosüsteemide ja tarkvara kavandamist abistav töövahend, mille abil saab koostada süsteeme kirjeldavaid mudeleid ja nendest uusi tulemeid (sh uusi mudeleid ja lähtekoodi) genereerida.
HTTP	*Hypertext Transfer Protocol*, protokoll andmete edastamiseks arvutivõrkudes.
JSON	*JavaScript Object Notation*, andmevahetuse formaat.
Päring	Andmete otsimine. SQL puhul on selleks vaja kirjutada SELECT lause.
REST	*Representational State Transfer,* tarkvaraarhitektuuri stiil rakenduste loomiseks.
SQL	*Structured Query Language*, päringukeel andmete haldamiseks ja manipuleerimiseks andmebaasis.
UML	*Unified Modeling Language.* Visuaalne modelleerimiskeel infosüsteemide ja tarkvara kavandamiseks.
XML	*The Extensible Markup Language*, märgistuskeel struktureeritud info jagamiseks infosüsteemide vahel.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäevane kiiresti muutuv maailm, kus aina rohkem kogutakse erinevad andmeid, seab täiendavaid nõudeid andmebaaside loomisele ja muutmisele. Juba loodud andmebaasid peavad toime tulema skeemi muutmiste, andmete osalise puudumise ja ka ajalooliste andmete säilitamise probleemidega. Selliste probleemidega toime tulemiseks on välja töötatud ankurmodelleerimise metoodika [1], mille üks, kuid mitte ainus, osa on andmetele esitavate nõuete modelleerimine ankurmudelitena. Kuna sellise metoodika tulemusel tekib andmebaasi rohkem tabeleid, sest tabelid on kuuendal normaalkujul, siis andmebaasi loomise lihtsustamiseks on loodud veebipõhine vahend [2], kus on võimalik mudeleid koostada ja nende põhjal erinevate andmebaasisüsteemide loomiseks vajalikku SQL-koodi genereerida. Keskkond toetab 2023. aasta kevade seisuga viite erinevat andmebaasisüsteemi: Microsoft SQL Server, PostgreSQL, Oracle, Vertica ja Snowflake. Käesolev magistritöö keskendub antud vahendi PostgreSQL SQL-koodi generaatorile.

1.1 Taust ja probleem

PostgreSQL-i koodigeneraator lisati modelleerimisvahendile 2015. aastal Elari Saali magistritöö [3] raames, mille juhendaja oli sama kes käesoleva töö juhendaja. Generaatori kood saadi Microsoft SQL Serveri jaoks mõeldud koodi tõlkimise tulemusena, kuid mõnikord oli see tõlkimine mehaaniline ja tõlkimisel ei süvenetud piisavalt sellesse, mida see kood teeb, miks see kood midagi niimoodi teeb ning kuidas saaks seda koodi optimaalsemaks muuta. Tollal oli eesmärk saada lihtsalt generaator tööle. Sellest tulenevalt on kahjuks PostgreSQLi ankurmudelite realisatsiooni generaatori korral ankurmudelite realiseerimise põhimõtted halvasti või üldse mitte dokumenteeritud ja põhjendatud.

1.2 Töö kirjeldus

Töö võib suures plaanis jaotada kolmeks osaks: testimise eesmärgil konkreetse ankurmudeli ja andmebaaside loomine, PostgreSQL SQL-koodigeneraatori parendamine ja täiendamine ning andmebaasi põhjal toimuvate SELECT lausete töökiiruse uurimine.

1.2.1 Ankurmudeli ja andmebaaside loomine

Töö käigus luuakse kolm andmebaasi. Esimene baas on nõndanimetatud tavaline andmebaas, mis andmete koosluselt on sama, mis ankurmudeli põhjal koostatud baas, kuid erineb struktuurilt. Selles andmebaasis ei ole tabelid kuuenda normaalkuju tasemeni normaliseeritud. Teine ja kolmas andmebaas koostatakse ankurmudeli põhjal, kuid erinevus seisneb selles, et teise baasi genereerimiseks kasutatakse SQL-i koodi generaatorit enne parendamist ja kolmanda juhul pärast parendamist. Kõikides andmebaasides kasutatakse võrreldavuse huvides genereeritud andmeid ühtede ja samade olemite ja seoste kohta.

1.2.2 PostgreSQL SQL-koodigeneraatori olemasoleva olukorra analüüs

Selles etapis analüüsitakse ja testitakse olemasolevat generaatorit, et saada ülevaade olemasolevast olukorrast.

1.2.3 SELECT lausete töökiiruse uuring

Töökiiruse uuringus sõnastatakse andmete otsimise ülesanded, koostatakse nende lahendamiseks päringud ja käivitatakse neid tavalises andmebaasis ja enne generaatori parendamist genereeritud koodi abil loodud ankurmudeli andmebaasis. Tulemusi võrreldakse, et selgitada välja, kas hetkel genereeritava PostgreSQL andmebaasi realisatsiooni puhul on töökiirust üldse vaja parandada. Juhul kui selline vajadus on, siis analüüsitakse mida oleks selleks võimalik teha ja milliseid muudatusi vajab selleks PostgreSQL SQL-koodigeneraator, et genereerida selleks vajalik SQL-i kood.

1.2.4 PostgreSQL SQL-koodigeneraatori parendamine ja täiendamine

Selles etapis täiendatakse generaatorit ja testitakse tulemusi.

1.3 Töö struktuur

Töö koosneb kaheksast peatükist: sissejuhatus, metoodika, teoreetiline taust, PostgreSQL koodigeneraatori seis enne arendust, töökiiruse uuring, koodigeneraatori parandused, arendusvaade ja kokkuvõte. Sissejuhatuses antakse ülevaade probleemist, mida lahendama hakatakse ja kirjeldatakse tehtavat tööd. Metoodika peatükis kirjeldatakse töö eesmärkideni jõudmiseks kasutatavaid tegevusi ja töövahendeid. Teoreetiline taust teeb lühiülevaate mõistetest, mida tuleks tunda käesoleva töö paremaks mõistmiseks. PostgreSQL koodigeneraatori seis enne arendust kirjeldab olukorda generaatoriga enne arenduste alustamist. Töökiiruse uuring annab ülevaate eksperimendist, kus võrreldakse päringute kiiruseid erinevate andmebaaside põhjal. Koodgeneraatori paranduste peatükis tuuakse välja muudatused, mis tehakse SQL-koodigeneraatori mallides. Selles peatükis kirjutatakse ka tulemuste valideerimisest. Arendusvaade annab ülevaate ettepanekutest, mida võiks veel tulevikus PostgreSQL SQL-koodigeneraatori paremaks muutmiseks teha. Kokkuvõte annab lühiülevaate tehtud tööst.

2 Metoodika

Antud peatükis antakse ülevaade töö objektist, protsessist ning tulemuste saavutamiseks kasutavatest tööriistadest.

2.1 Ülevaade objektist

Objektiks on PostgreSQL SQL-koodigeneraator. Kui veebipõhine modelleerimisvahend loodi, siis algselt oli selle koodigeneraatoril vaid Microsoft SQL-i tugi, kuid 2015. aastal lisati ka PostgreSQL-i tugi. Peale 2015. aastat pole sinna suuremahulisi täiendusi tehtud.

2.2 Ülevaade töö protsessist

Töö näol on tegemist disainiteadusliku [4] tööga, mille käigus luuakse täiustatud tehis (PostgreSQL jaoks mõeldud ankurmodelleerimise mudelitest SQL lausete generaator). Disainteadus on metoodika, mis keskendub praktiliste probleemide lahenduste loomisele ja hindamisele infotehnoloogia valdkonnas. See ei tähenda alati uute lahenduste loomist, vaid on mõeldud ka juba eksisteerivate süsteemide parandamiseks. Disainiteadus koosneb järgmistest sammudest:

- Probleemi tuvastamine
- Tehise kavandamine
- Tehise loomine ja testimine
- Tehise hindamine

Tulemuste valideerimiseks katsetatakse andmemuudatuste operatsioonide läbiviimist andmebaasis, mille loomiseks on kasutatud vana generaatorit *vs.* andmebaasis, mille loomiseks on kasutatud uut generaatorit, veendumaks, et uues andmebaasis on kõrvaldatud kõik need probleemid, mis esinesid vanas andmebaasis.

2.3 Tööriistade kirjeldus

Peamisteks tööriistadeks, mida kasutatakse magistritöö eesmärkide saavutamiseks on andmebaasisüsteem PostgreSQL (versioon 14.3), ankurmudelite veebipõhine modelleerimisvahend [2] ning UML keelt toetav CASE vahend.

2.3.1 PostgreSQL

PostgreSQL on vabavaraline objekt-relatsiooniline andmebaasisüsteem ja 2023. aasta aprilli seisuga on tegemist populaarsuselt neljanda andmebaasisüsteemiga [5] PostgreSQL on populaarne ka Eestis, leides palju kasutamist nii avalikus kui ka erasektoris.

2.3.2 Ankurmudelite modelleerimisvahend

Andurmudelite veebipõhise modelleerimisvahendi abil saab luua mudeleid, mis on sisendiks SQL-koodigeneraatorile PostgreSQL andmebaasisüsteemile sobiva koodi genereerimiseks.

2.3.3 Modelleerimine UML keeles

Olemi-suhte diagrammide ja tabelite disain esitavate diagrammide koostamiseks UML klassidiagrammide baasil kasutati CASE vahendit Enterprise Architect (12).

3 Teoreetiline taust

Selles peatükis antakse lühiülevaade mõistetest, mida tuleks tunda käesolevast tööst paremini arusaamiseks.

3.1 Kuues normaalkuju

Andmestruktuuride täiendav normaliseerimine (sageli öeldakse lihtsalt "normaliseerimine") on andmestruktuuride korrastamine, mille eesmärgiks on vähendada andmete liiasust ja sellest tulenevaid andmete muutmise anomaaliaid. Andmete liiasus tähendab, et üks ja sama fakt on võimalik andmebaasist leida või tuletada rohkem kui ühest kohast. Täiendavalt normaliseerida saab erinevat tüüpi andmestruktuure, sh SQL-andmebaasi tabeleid. Kui mainitakse tabelite täiendavat normaliseerimist, siis üldiselt peetakse silmas nende viimist esimeselt normaalkujult kuni viienda normaalkujuni. See protsess aitab vähendada, kuid mitte täielikult kaotada, andmete liiasust ja sellest tulenevaid andmete muutmise anomaaliaid. Tegelikkuses eksisteerib ka kuues normaalkuju (*6NK*), millele tabeli viimine tähendab, et seda tabelit ei ole enam võimalik dekomponeerides väiksemateks osadeks jagada. Sellises tabelis on üks kandidaatvõti ja lisaks sellesse kuuluvatele veergudele maksimaalselt veel üks veerg.

Praktikas on kuuendal normaalkujul tabeleid üha rohkem kasutama hakatud tänu ankurmodelleerimisele. Kuuendale normaalkujule viidud tabelite kasutamise eelisteks on paremad võimalused tulla toime ajalooliste andmete säilitamisega, puuduvate andmetega ning andmebaasi evolutsiooniga. Parem toimetuleks ajalooliste andmete säilitamisega tähendab, et andmebaasi arendajal on kontroll selle üle, milliste olemitüübi atribuutide või seosetüüpide puhul säilitada ajaloolisi andmeid (vastavad muudatused tehakse atribuudile või seosetüünile vastavas 6NK tabeli struktuuris) ja milliste puhul mitte [6]. Parem toimetulek puuduvate andmetega tähendab, et kui olemi atribuudi väärtus pole teada või pole arvutatav, siis atribuudile vastavasse tabelisse sellele väärtusele vastavat rida ei teki [7]. See vabastab andmebaasi kasutajad vajadusest esitada puuduvaid andmeid NULLide abil. Parem toimetulek andmebaasi evolutsiooniga tähendab, et kui on vaja hakata koguma mingeid täiendavaid andmeid, siis ei ole vaja olemasolevate tabelite struktuuri muuta, vaid tekitatakse juurde uus tabel.

Tööks, kus kuuenda normaalkuju tabelite disaini on võrreldud teiste SQL-andmebaasi disainidega, on [8]. Töös võrreldakse traditsioonilist, universaalset, olem-atribuut-väärtus ja kuuenda normaalkuju disaini kahe erineva konteksti jaoks, leides nende disainide suhtelise headuse nendes kontekstides. Hinnatavate kriteeriumite hulgas oli nii päringute keerukus kui töökiirus. Mõlemal juhul jäi kuuenda normaalkuju disain suhteliselt headuselt teisele kohale traditsioonilise disaini järel.

3.2 Ankurmodelleerimine

Ankurmodelleerimine on andmebaasi arendamise metoodika, mille üks, kuid mitte ainus, osa on andmetele esitatavate nõuete modelleerimine ankurmudelitena. Rahvusvaheliselt esitles seda meetodikat esmakordselt 2007. aastal rootslane Lars Rönnbäck Amsterdamis toimunud konverentsil *Transforming Data with Intelligence*. Metoodika näeb ette mudelitest koodi genereerimise ja pakub nii modelleerimise kui ka koodi genereerimise jaoks töövahendi. Metoodika arvestab sellega, et nõuded tarkvarale ja andmebaasile on pidevas muutumises (need evolutsioneeruvad) ning seega peab muutuma ka andmebaasi ülesehitus ja andmeid kasutav rakendus. Ankurmudelite järkjärguline loomine, loodud mudelitest koodi genereerimine ning SQL-andmebaasi täienduste lihtne kasutuselevõtt (olemasolevate tabelite struktuuri muutmise asemel on vaja lisada uusi tabeleid) on kooskõlas agiilse arenduse põhimõtetega. Sellel põhimõttel loodud andmebaas aitab edukalt toime tulla andmebaasi skeemi muutmiste, andmete osalise puudumise ja ajalooliste andmete säilitamise probleemidega. Ankurmodelleerimise alusel loodud andmebaas kasutab ära tabelite kõrge tasemeni normaliseerimise eeliseid, sest sellel põhimõttel loodud SQL-andmebaasis on tabelid üldiselt kuuendal normaalkujul (6NK).

Peamiselt kasutatakse ankurmodelleerimist andmeaitade ja andmevakkade loomiseks. Samas peaks see sobima kasutamiseks ka operatiivandmeid haldavates onlain-tehingutöötluse süsteemides, sest ka seal tuleb toime tulla skeemimuudatuste ja puuduvate andmetega, kuid selle kohta pole piisavalt uuringuid. Esimene andmeait, mille loomisel kasutati ankurmodelleerimise põhimõtteid, valmis Saal [3] andmetel Rootsis 2004. aastal ning suurimas antud viisil loodud temale teadaolevas andmebaasis hoiti üle 40TB andmeid. Golov ja Rönnbäck [9] kirjutavad ankurmodelleerimise rakendamisest suurandmete hoidmisel ja töötlemisel Venemaa e-kommertsi ettevõte

Avito näitel. Päevas registreeriti selle infosüsteemis üle ühe miljardi kasutaja tegevuse. Ettevõte võttis edukalt kasutusele ankurmodelleerimisel põhineva andmeaida. Litsenseerimisest tulenevate piirangute tõttu seati selle andmemahu ülapiiriks 51TB.

Ankurmudelite modelleerimiseks on loodud avatud lähtekoodiga veebipõhine keskkond (https://www.anchormodeling.com/modeler/latest/). Tabel 1 annab ülevaate elementidest, mida seal keskkonnas kasutada saab.

Tabel 1. Ankurmodelleerimise põhielemendid.

Nimetus	**Kirjeldus**
Ankur (*anchor*)	Olemitüüp – üldistus reaalses maailmas esinevatest olemitest (füüsilistest või abstraktsetest asjadest)
Sõlm (*knot*)	Olemitüüp, mis esitab klassifikaatorit.
Staatiline atribuut (*static attribute*)	Olemitüüpi kuuluvate olemite omadus, mille väärtuste ajalugu ei säilitata.
Ajalooline atribuut (*historized attribute*)	Olemitüüpi kuuluvate olemite omadus, mille väärtuste ajalugu säilitatakse.
Sõlmitud staatiline atribuut (*knotted static attribute*)	Olemi ja sõlme vaheline side, mis kirjeldab omadusi, mis ajas ei muutu.
Sõlmitud ajalooline atribuut (*knotted historized attribute*)	Olemi ja sõlme vaheline side, mis kirjeldab omadusi, mis võivad ajas muutuda.
Staatiline side (*static tie*)	Seos kahe ankru vahel.
Sõlmitud staatiline side (*knotted static tie*)	Seos kahe ankru ja ühe sõlme vahel
Ajalooline side (*historized tie*)	Ajalooline seos kahe ankru vahel.
Sõlmitud ajalooline side (*knotted historized tie*)	Kahe ankru ja ühe sõlme vaheline ajalooline side.

3.3 Tabeli elimineerimise teisendus

Tabeli elimineerimise teisenduse eesmärgiks on eemaldada e elimineerida lauses tehtavad pöördumised tabelite poole, mis antud lause täitmiseks vajalikud ei ole, kuid siiski esinevad algses lauses [10]. Tegemist on andmebaasisüsteemi poolse lause lihtsustamisega. Andmebaasisüsteemi tasemel toimub lause lihtsustamine nõnda, et see oleks kiiremini täidetav, kuid samas loogiliselt samaväärne algse lausega. Kuna ankurmodelleerimise alusel tehtud andmebaas sisaldab rohkem tabeleid, kui tavaline

andmebaas, siis on tabeli elimineerimise teisendusest kasu selle andmebaasi põhjal tehtavate päringute kiirendamisel. Bakalaureusetöö [11] demonstreeris, et PostgreSQL oskab samuti tabeli elimineerimise teisendust läbi viia.

3.4 Sisula

Sisula (*simple substitution language*) on tarkvara, mis on loodud lihtsustamaks SQL-koodigeneraatori arendamist. See on kirjutatud Javascript keeles ja tegemist on vabatarkvaraga. Sisula tõlgib ankurmudeli, mis on XML (*The Extensible Markup Language*)-formaadis, JSON (*Javascript Object Notation*)-formaati.

3.5 PostgreSQL andmebaasis SELECT lausete töökiiruse parandamise võimalusi

PostgreSQL pakub erinevaid võimalusi, kuidas SELECT lausete töökiirust parandada. Antud jaotises antakse lühiülevaate mõningatest võimalustest, mida on plaanis ankurmudeli SQL-koodigeneraatori juures katsetada ja ka kasutusele võtta, kui katsetused edukaks kujunevad.

3.5.1 Indeks

Indeks on andmebaasi siseskeemi kuuluv (ja seega andmebaasi kasutajate nagu rakendused poolt otse mitte kasutatav) andmestruktuur, mille abil on võimalik kiirendada andmete andmebaasist ülesleidmist. Indeksit võiks võrrelda raamatu indeksiga, kus erinevalt indeksist, milles on kirjas märksõna leheküljenumbritest, tekitab indeks viida asukohaga, kus andmed tabelis asuvad. Raamatu indeksit kasutavad raamatu lõppkasutaja e lugejad samas kui andmebaasi indeksi kasutamise üle otsustab andmebaasisüsteem. Indeksid aitavad küll kiirendada teatud olukordades andmete ülesleidmist andmebaasist, aga alati tuleb hoolikalt kaaluda milliseid indekseid luua, sest liiga palju indekseid võib viia olukorrani, kus päringute kiirused hoopis langevad. Samuti võib kannatada andmete muutmise ja lisamise operatsioonide jõudlus, sest muutes või lisades andmeid tuleb teha muudatus ka indeksis. Vaikimisi indeksi tüübiks on B-puu e tasakaalustatud puu indeks, kus indeksipuu lehtedes on indekseeritud väärtused koos viidetega ridadele, milles need väärtused esinevad. Tasakaalustatud puu tähendab, et igast lehest on puu juureni ühesuguse pikkusega tee.

3.5.2 Tabeli klasterdamine

Andmete klasterdamine tähendab sarnaste või kooskasutatavate andmete kokkukoondamist ja üksteise lähedale paigutamist. Tabeli klasterdamine tähendab PostgreSQL-is andmete järjestamist kettal füüsiliselt mingi indeksi põhjal [12]. Selle kasutamise miinuseks on see, et andmete lisandumisel tabelisse ei säilitata andmebaasisüsteemi poolt automaatselt ridade järjestatust ja seega tuleb järjestamist perioodiliselt korrata. Soovitatav on selleks kirjutada skript, mis mingi perioodi möödudes tabeli uuesti klasterdab.

3.5.3 Vaade

Vaade (*view*) on virtuaalne tabel, kus reaalselt andmeid ei hoita ja mille andmed võivad pärineda ühest või mitmest tabelist või vaatest. Teiste sõnadega vaade on tuletatud tabel, mis on defineeritud teiste tabelite põhjal ja mille väärtus arvutatakse välja hetkel, kui keegi sellest andmeid küsib. Ankurmodelleerimise vaatenurgast on vaadete kasutamine väga oluline, kuna ankurmudeli põhjal koostatud baasis on tabelid e baastabelid kuuendal normaalkujul ja neid on üsna palju. Vaadete kasutamine lihtsustab andmete pärimist, sest on võimalik keerulised päringud ühte vaatesse panna ja edaspidi ei pea seda keerulist päringut uuesti kirjutama, vaid on võimalik selle poolt kokku pandud andmeid läbi vaate kasutada. Selles mõttes on vaade nagu makro, mis lihtsustab keeruka operatsiooni läbiviimist – antud juhul päringu kirjutamist. Joonis 1 näitab kuidas tehakse vaadet SQL-is. Antud näites koondatakse kolm tabelit ühte päringusse ja edaspidi, kui soovitakse küsida kino nime ja aadressi, saab selle lihtsalt kätte otse vaate „lci_cinema“ poole pöördumisega. Vaadete põhjal päringute tegemine ei kiirenda päringute täitmist, kuid üldjuhul ka olulisel määral ei halvenda [13]. Töökiiruse halvenemise põhjuseks võib olla see, kui andmebaasisüsteem koostab vaate põhjal päringut tehes lausele halvema (aeglasemalt täidetava) täitmisplaani kui andmeid otse baastabelitest küsides.

```
CREATE OR REPLACE VIEW public.lci_cinema
 AS
 SELECT ci.ci_id,
    nam.ci_nam_cinema_name,
    adr.ci_adr_cinema_address
   FROM ci_cinema ci
     LEFT JOIN ci_nam_cinema_name nam ON nam.ci_id = ci.ci_id
     LEFT JOIN ci_adr_cinema_address adr ON adr.ci_id = ci.ci_id;
```

Joonis 1. Vaate loomise näide SQL-is.

3.5.4 Materialiseeritud vaade

Materialiseeritud vaade (*materialized view*) e hetktõmmis (*snapshot*) on tuletatud tabel (tabel, mis on defineeritud teiste tabelite põhjal), kus vaate väärtus e selle aluseks oleva päringu täitmise tulemus on enne vaate poole pöördumist välja arvutatud ja eraldi koopiana kettale salvestatud [14]. Materialiseeritud vaadete kasutamise miinuseks PostgreSQLis (14) on see, et materialiseeritud vaates olevad andmed automaatselt ei uuene ja andmete värskendamist tuleks ise teha. PostegreSQL-is on selleks lause REFRESH MATERIALIZED VIEW. PostgreSQL-is pole sisseehitatud võimalust vahepealsete andmemuudatuste logimiseks ja nende alusel materialiseeritud vaate osaliseks värskendamiseks. Värskendamine tähendab kogu vaate väärtuse uuesti arvutamist, mis võib olla aeganõudvam kui osaline värskendamine.

3.5.5 Tabeli osadeks jaotamine

Tabeli osadeks e sektsioonideks jaotamine (*table partitioning*) on ühe suure tabeli jagamine väiksemateks alamosadeks viisil, et kontseptuaalsel tasemel paistab tabel kui üks tervik, kuid sisemiselt saab andmebaasisüsteem erinevaid operatsioone täita üksikute alamosade põhjal. Joonis 2 näitab, kuidas toimub tabeli „Movie_Showed“ osadeks jaotamine välja „ChangedAt“ järgi PostgreSQL-is [15]. Näitest on näha, et moodustatakse kaks täiendavat tabelit „Showed_part_1“ ja „Showed_part_2“. Esimesse tabelisse lisatakse read, millel välja „ChangeAt“ väärtus jääb 2018. aastasse ja teise read, millel välja väärtus jääb 2019. aastasse.

```
CREATE TABLE IF NOT EXISTS public.Movie_Showed (
    CI_ID integer not null,
    MV_ID integer not null,
    SHS_ID integer not null,
    ChangedAt timestamp not null,
    constraint fkCI_Cinema foreign key (
        CI_ID
    ) references public.Cinema(CI_ID),
    constraint fkMV_Movie foreign key (
        MV_ID
    ) references public.Movie(MV_ID),
    constraint pkCI_Movie_Showed primary key (
        CI_ID ,
        MV_ID ,
        ChangedAt
    )
) PARTITION BY RANGE (ChangedAt);

CREATE TABLE Movie_Showed_part_1 PARTITION OF public.Movie_Showed
    FOR VALUES FROM ('2018-01-01 00:00:00') TO ('2019-01-01
00:00:00');

CREATE TABLE Movie_Showed_part_2 PARTITION OF public.Movie_Showed
    FOR VALUES FROM ('2019-01-01 00:00:00') TO ('2020-01-01
00:00:00');
```

Joonis 2. Tabeli osadeks jaotamise näide PostgreSQL-is.

4 PostgreSQL koodigeneraatori seis enne arendust

Töö autor alustas modelleerimisvahendiga tutvumist viimasest versioonist [2] ja üsna ruttu selgus, et PostgreSQL SQL-koodigeneraator on katki. Tuli välja, et kui lisada mudelil ankrutele ajaloostatud (*historized*) atribuute, siis generaator enam ei tööta. Kuna modelleerimisvahendil on olemas ka testversioon, mis on kättesaadav aadressil https://roenbaeck.github.io/anchor/ ja milles seda viga ei esinenud, siis jätkas autor mudeli koostamist seal.

Ankurmudelite veebipõhise modelleerimisvahendi lähtekood asub Github-is aadressil https://github.com/Roenbaeck/anchor. Töö autor võttis aluseks „master“ haru (*branch*), mis on kättesaadav aadressilt https://github.com/Roenbaeck/anchor/tree/master. Analüüsides PostgreSQL SQL-koodigeneraatori lähtekoodi, selgus, et uusi arendusi pärast selle loomist seal tehtud ei ole. Arendatud on peamiselt ankurmudeli modelleerimisvahendit ja Microsoft SQL Serveri jaoks mõeldud koodi generaatorit. Arenduste puudumine PostgreSQL koodigeneraatori harus ei tähenda seda, et selle lähtekood on seal olnud muutmata alates selle loomisest. Ligemale kolm aastat tagasi (2020. aastal) on mingi arenduse käigus PostgreSQL generaatori failid kirjutatud üle Microsoft SQL generaatori failidega. Hiljem (2-3 kuud) on üritatud olukorda taastada, aga selle tulemuseks on see, et töö tegemise ajal testversioonis olev PostgreSQL SQL-koodigeneraator on väiksema funktsionaalsusega, kui algselt loodud generaator. Näitena võib tuua tabelitele kommentaaride lisamise, mille kood algses versioonis genereeriti, aga enam seda ei tehta. Samuti ei genereerita trigereid, mis on vajalikud andmeoperatsioonide teostamiseks.

4.1 Genereeritav kood

Selles osas antakse ankru *Movie* näitel väike ülevaade genereeritavast koodist ja objektidest, mida PostgreSQL SQL-koodigeneraatori poolt koostatud koodi käivitamisel andmebaasi lisatakse.

4.1.1 Tabelid

Tabel 2 esitab loetelu tabelitest, mis genereeritakse ankru *Movie* põhjal PostgreSQL SQL-koodigeneraatoriga.

Tabel 2. Ankru *Movie* põhjal genereeritud tabelid.

Tabeli nimetus	**Kirjeldus**
Mv_movie	Ankru *movie* tabel
Mv_nam_movie_name	Atribuudi *name* tabel
Mv_ath_movie_author	Atribuudi *author* tabel
Mv_rat_movie_rating	Atribuudi *rating* tabel
Mv_get_movie_gerne	Atribuudi *gerne* tabel
Typ_genretype	Klassifkaatori *genre type* tabel
Rtt_ratingtype	Klassigikaatori *rating type* tabel

4.1.2 Vaated

Vaadetel on väga oluline koht ankurmodelleerimise metoodikaga loodud andmebaasides. Esiteks on vaated abiks andmete pärimisel, sest ühte vaatesse saab koostada päringu, mis pärib andmed nii ankrust kui ka sellega seotud atribuutidest ja ei teki vajadus pidevalt seda päringut uuesti koostada (vt joonis 3 näidet vaate loomisest). Teiseks toimub läbi vaadete andme lisamine, muutmine ja kustutamine. Vaadetele lisatavad INSTEAD OF trigereid peaks hoolitsema selle eest, et andmed saaks atribuutide tabelites korrektselt värskendatud.

PostgreSQLis (14) saab vaikimisi andmeid muuta ainult läbi lihtsate vaadete, mille alampäringus viidatakse vaid ühele tabelile [16]. Samas saab vaikimisi mittemuudetavaid vaateid muudetavaks programmeerida INSTEAD OF trigerite abil. Sellist tüüpi triger käivitub andmete muutmisel vaates ja selle asemel, et teha muudatus vaate põhjal, täidetakse trigeri protseduur, mis muudab andmeid vaate aluseks olevates tabelites.

```
-- Latest perspective ------------------------------------------------
-------------------------------------------------
-- lMV_Movie viewed by the latest available information (may include
future versions)
-----------------------------------------------------------------------
-------------------------------------------------
CREATE OR REPLACE VIEW public.lMV_Movie AS
SELECT MV.MV_ID
     , NAM.MV_NAM_Movie_Name
     , kGER.TYP_GenreType AS TYP_GenreType
     , GER.TYP_ID
     , ATH.MV_ATH_Movie_Author
     , RAT.MV_RAT_ChangedAt
     , kRAT.RTT_RatingType AS RTT_RatingType
     , RAT.RTT_ID
  FROM public.MV_Movie MV
  LEFT
  JOIN public.MV_NAM_Movie_Name NAM
    ON NAM.MV_ID = MV.MV_ID
  LEFT
  JOIN public.MV_GER_Movie_Gerne GER
    ON GER.MV_ID = MV.MV_ID
  LEFT
  JOIN public.TYP_GenreType kGER
    ON kGER.TYP_ID = GER.TYP_ID
  LEFT
  JOIN public.MV_ATH_Movie_Author ATH
    ON ATH.MV_ID = MV.MV_ID
  LEFT
  JOIN public.lMV_RAT_Movie_Rating RAT
    ON RAT.MV_ID = MV.MV_ID
  LEFT
  JOIN public.RTT_RatingType kRAT
    ON kRAT.RTT_ID = RAT.RTT_ID;
```

Joonis 3. Ankru *Movie* põhjal genereeritud vaade.

Tabel 3 esitab loetelu vaadetest, mis genereeritakse ankru *Movie* põhjal PostgreSQL SQL-koodigeneraatoriga.

Tabel 3. Ankru *Movie* põhjal genereeritud vaated.

Vaate nimetus	**Kirjeldus**
Lmv_movie	Vaade tagastab filmide loetelu, kus iga filmi kohta on selle atribuudi väärtused. Kui filmi mõnel atribuudil väärtus puudub, siis tänu välisühendamisele vaate aluseks olevas päringus on vaates selle atribuudi väärtuse asemel NULL. Kui ankur sisaldab ajaloostatud atribuute, siis nendest tagastatakse kõige suurema *changetAt* väärtusega atribuudi rida. Läbi selle vaate toimub ka ridade lisamine atribuutide tabelitesse.

Vaate nimetus	Kirjeldus
Lmv_rat_movie_rating	Kuna *rating* on ajaloostatud atribuut, siis tegemist on abistava vaatega, mida vaade *Lmv_movie* kasutab leidmaks maksimaalse *changedAt* väärtusega *rating* rida.
Nmv_movie	Vaade tagastab filmide loetelu. Erinevus *Lmv_movie* vaatega seisneb selles, et ajaloostatud atribuudi muutmise kuupäeva väljale *changetAt* rakendatakse täiendav filter, et see peab olema väiksem või võrdne tänase kuupäevaga.
Nmv_rat_movie_rating	Kuna *rating* on ajaloostatud atribuut, siis tegemist on abistava vaatega, mida vaade *Nmv_movie* kasutab leidmaks *raiting* väärtust, mille muutmise kuupäevale *changedAt* on rakendatud filter, et see peab olema väiksem või võrdne tänase kuupäevaga.

4.1.3 Trigerid

Kood, mis sai võetud *master* harust kahjuks ühtegi trigerit ei genereerinud. Korras versioon peaks genereerima vaadetele INSTEAD OF trigerid, mille kaudu toimub atribuut tabelite värskendamine.

4.1.4 Funktsioonid

Tabel 4 esitab loetelu funktsioonidest, mis genereeritakse ankru *Movie* põhjal PostgreSQL SQL-koodigeneraatoriga.

Tabel 4. Lisatavad funktsioonid.

Funktsiooni nimetus	Kirjeldus
Dmv_movie	Funktsioonis leitakse vaadeldava ajaloolise atribuudi kohta kõik etteantud vahemikku jäänud andmete muutmise ajahetked ja seejärel, leitakse neile hetkedele vastavad andmed. Põhimõtteliselt oleks tegemist nagu logiga, kus on näha atribuutide väärtusi erinevatel ajahetkedel.
Pmv_movie	Tagastab andmebaasist filmide andmed koos kõikide atribuutidega ja ajaloolistest atribuutidest tagastab versiooni, mis leitakse funktsiooniga *pmv_rat_movie_rating*
Pmv_rat_movie_rating	Tagastab ajaloolise atribuudi tabelist need read, mille muutmisaeg on väiksem või võrdne funktsioonile ette antud ajahetkega.
Rfmv_rat_movie_rating	Funktsioon kontrollib, kas sisendiks antud atribuudi väärtus on lisandunud väärtus või duplikaat.

5 Töökiiruse uuring

Töökiiruse uuringu eesmärgiks on välja selgitada vajadus ja võimalused PostgreSQL SQL-i koodigeneraatori täiendamiseks.

5.1 Eksperimendi kirjeldus

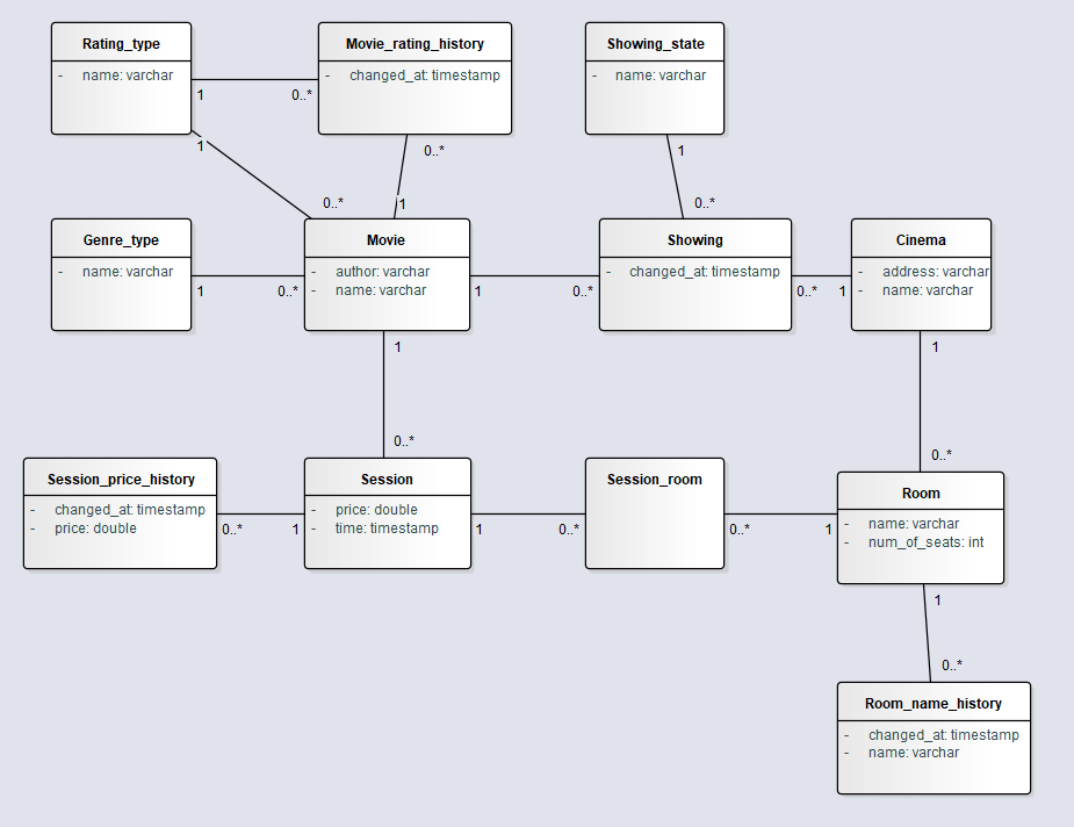
Eksperimendi jaoks luuakse kolm andmebaasi – tavaline andmebaas ja kaks andmebaasi, mille loomisel on järgitud ankurmodelleerimise põhimõtteid. Sõnastatakse andmete otsimise ülesanded, koostatakse neid lahendavad päringud ja käivitatakse neid kõikide loodud andmebaaside põhjal. Testimise jooksul katsetatakse päringute töökiiruse parendamiseks erinevaid strateegiaid (nt indeksite loomine). Tabel 5 esitab loodavate andmebaaside kontseptuaalses andmemudelis esitatud olemitüübid ja nende sõnalised kirjeldused. Tegemist on filmide andmebaasiga, kus hoitakse infot filmide kohta ja millistes kinodes neid näha saab.

Tabel 5. Olemitüübid ja nende kirjeldused.

Nimetus	**Kirjeldus**
Cinema	Kino. Asutus, kus kohas näidatakse filme.
Room	Ruum. Kinos asuvad saalid, kus filme näidatakse.
Room_name_history	Ruumide nimede ajalugu. Kui ruumi nime muudetakse, siis tuleb säilitada varasemad nimed.
Movie	Film. Film mida kinos näidatakse.
Movie_rating_history	Hinnangute ajalugu. Filmidele antavate hinnangute ajalugu.
Session	Seanss. Näitab aega, millal filmi kinos näidatakse.
Session_price_history	Seansi hindade ajalugu.
Session_room	Seansi ruum. Kus kinos ja mis ruumis seanss toimub.
Showing	Näitamine. Info selle kohta, kas film on veel kinos kavas.
Genre_type	Žanri klassifikaator, mis näitab, mis tüüpi filmiga on tegu. Näited on komöödia, draama, dokumentaal.
Rating_type	Reitingu klassifikaator, mis näitab, mis hinnanguid on filmile antud. Näited on hea ja halb
Showing_state	Näitamise staatuse klassifikaator. Näitab seda, kas film on veel kinos kavas. Näited jah ja ei.

5.1.1 Kontseptuaalne andmemudel

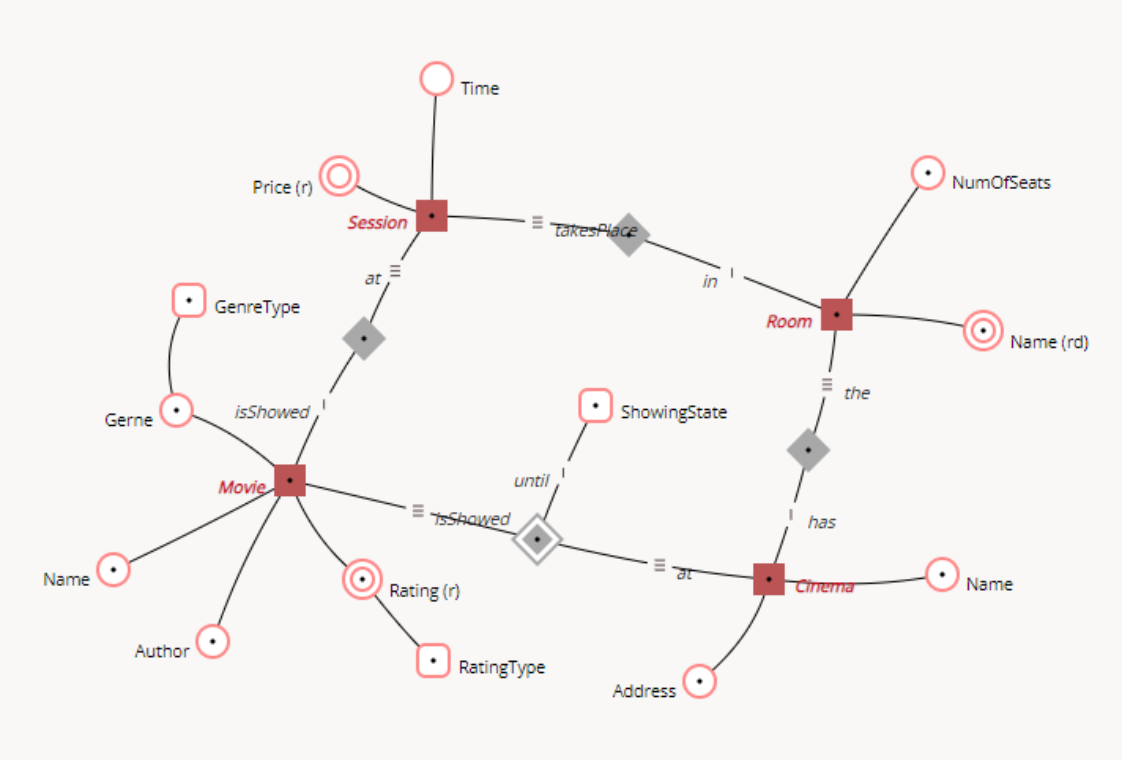
Joonisel 4 on kujutatud loodavate andmebaaside kontseptuaalse andmemudeli osaks olev olemi-suhte diagramm.



Joonis 4. Andmebaasi kontseptuaalse andmemudeli osaks olev olemi-suhte diagramm.

5.1.2 Ankurmudel

Joonis 5 esitab ankurmudeli, mis on koostatud veebipõhise modelleerimisvahendiga ja see vastab joonis 4 olevale andmebaasi kontseptuaalsele andmemudelile.



Joonis 5. Ankurmudel.

Tabel 6 esitab loodud ankurmudeli elemendid ja tüübid.

Tabel 6. Loodud ankurmudeli elemendid.

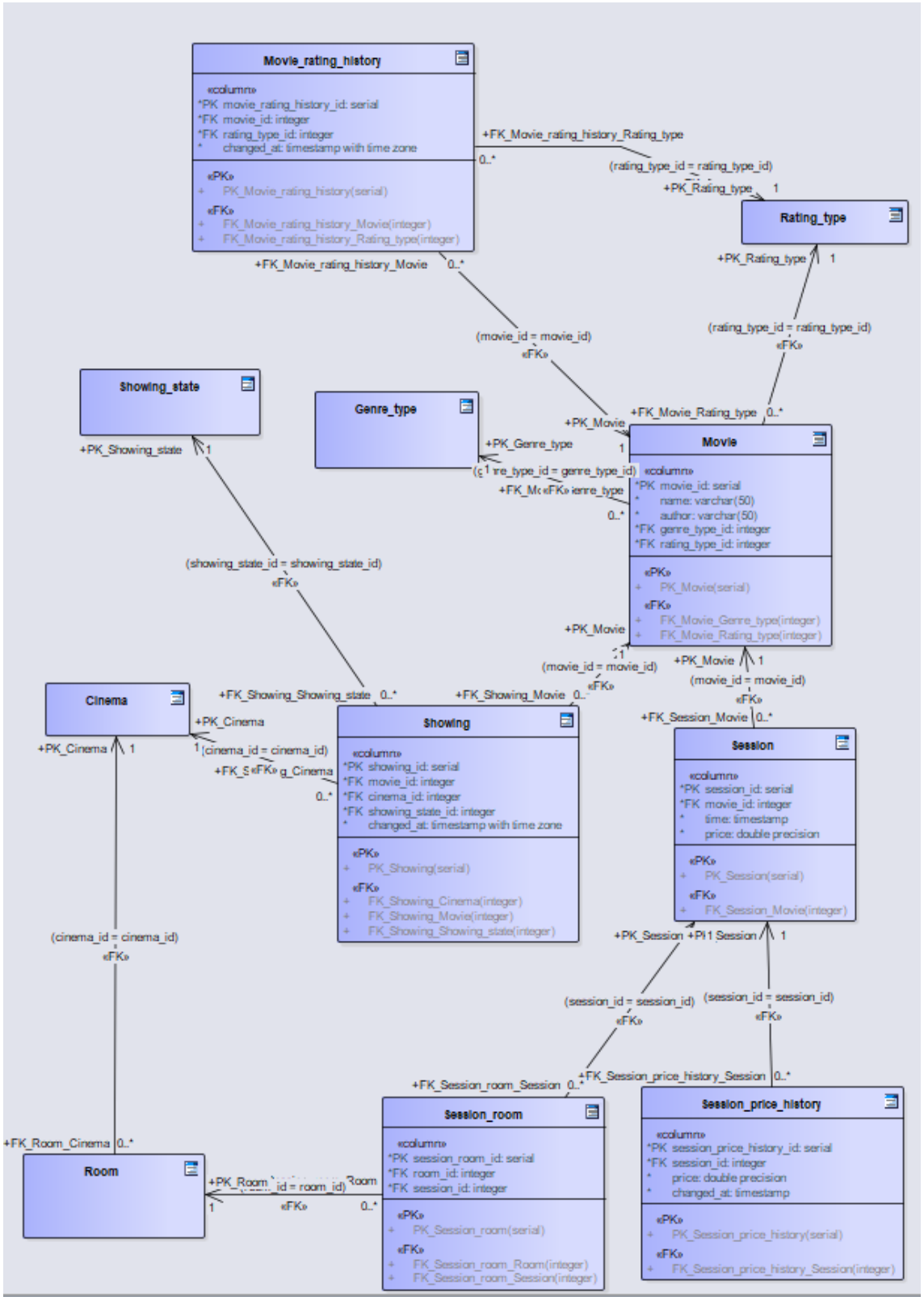
Nimetus	**Tüüp**
Cinema	Ankur Atribuudid: 1. Name – staatiline atribuut 2. Address - staatiline atribuut
Movie	Ankur Atribuudid: 1. Name – staatiline atribuut 2. Author – staatiline atribuut 3. Gerne – sõlmitud staatiline 4. Rating – sõlmitud ajalooline atribuut
Room	Ankur Atribuudid: 1. Name – ajalooline atribuut 2. NumOfSeats – staatiline atribuut
Session	Ankur Atribuudid: 1. Price – ajalooline atribuut

Nimetus	Tüüp
	2. Time – staatiline atribuut
Movie_isShowed_at_Cinema	Sõlmitud ajalooline side
Cinema_has_the_Room Session_takesplace_in_Room Movie_isShowed_at_Session	Staatiline side
ShowinState GenreType Ratingtype	Sõlmed

5.1.3 Tavalise andmebaasi disain

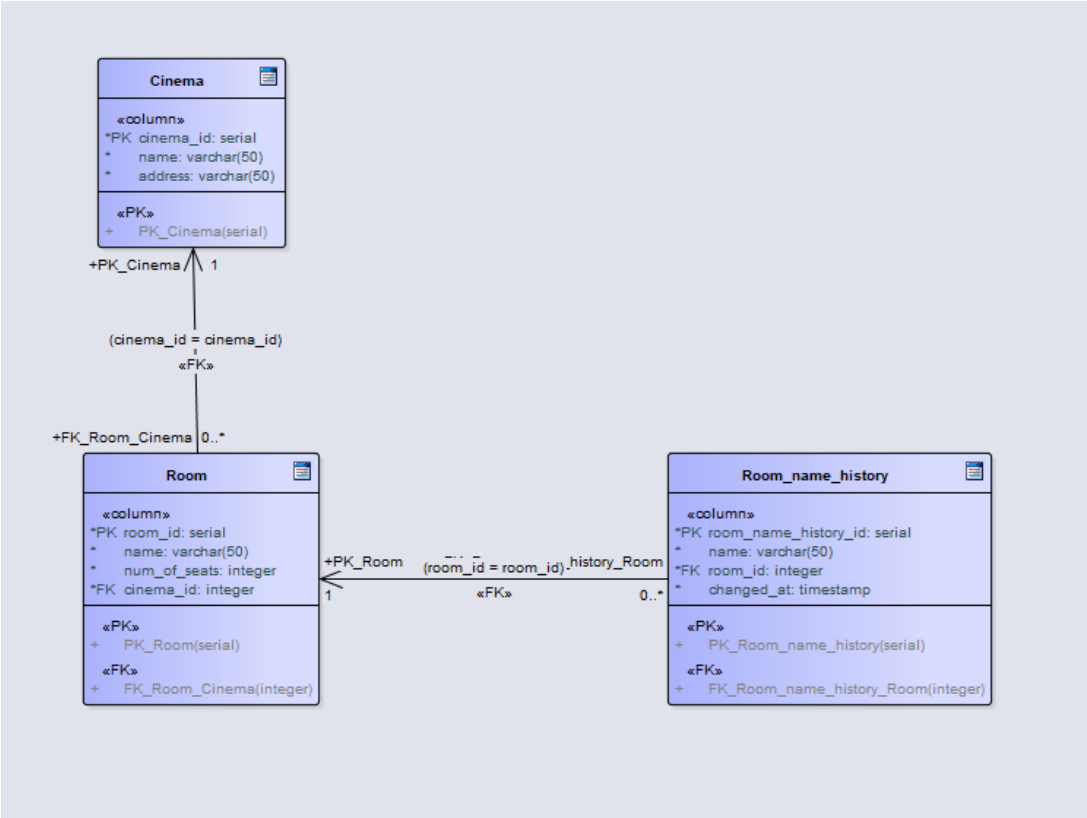
Tavalise andmebaasi loomiseks genereeritud kontseptuaalsest andmemudelist andmebaasi füüsilise disaini mudel, mida täiendati, genereeriti sellest SQL kood ja see kood käivitati andmebaasisüsteemis. Andmebaasis loodi kokku 14 tabelit, 0 funktsiooni, 0 trigerit ja 0 indeksit.

Joonis 6 esitab kinode registri füüsilise disaini tavalise andmebaasi korral.



Joonis 6. Kinode registri füüsiline disain.

Joonis 7 esitab filmide registri füüsilise disaini tavalise andmebaasi korral.



Joonis 7. Filmide registri füüsiline disain.

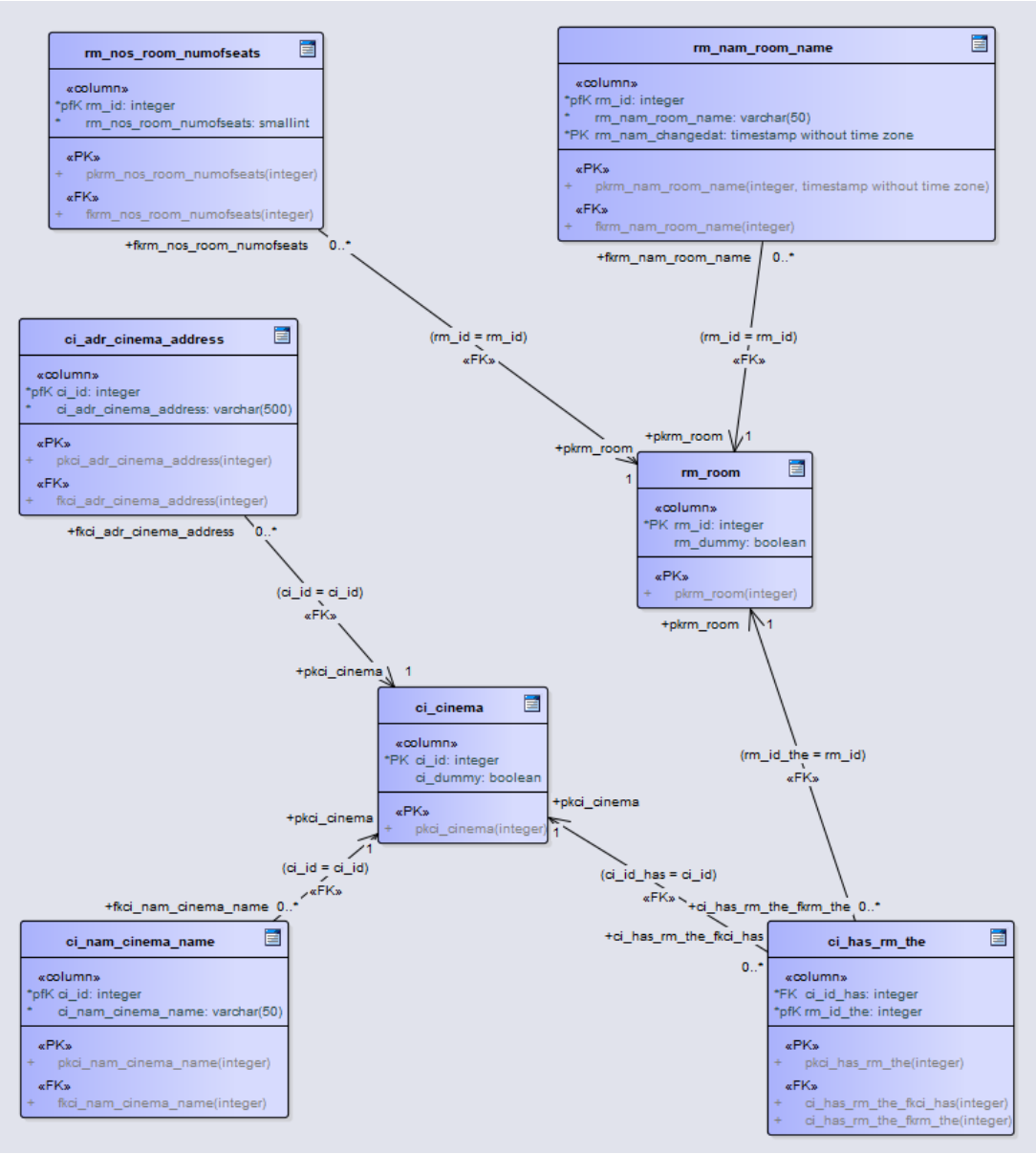
Joonis 8 esitab klassifikaatorite registri füüsilise disaini tavalise andmebaasi korral.

Joonis 8. Klassifikaatorite registri füüsiline disain.

5.1.4 Ankurmudeli andmebaasi disain

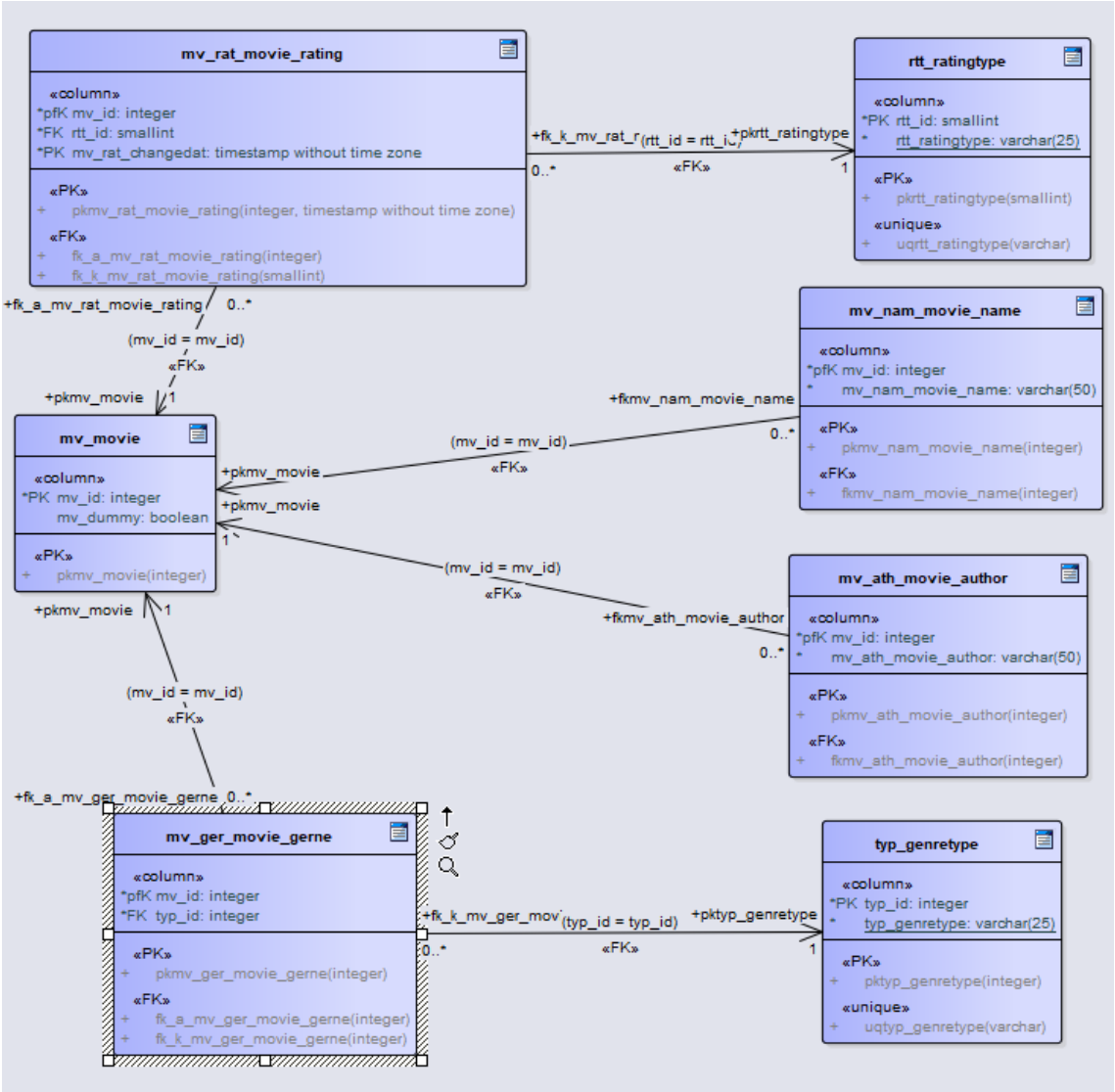
Andmebaasis loodi kokku 22 tabelit, 19 vaadet, 17 mitte-trigerifunktsiooni ja 0 trigerit (iga trigeriga käib paaris trigerifunktsioon). Ankurmodelleerimise andmebaas loodi ankurmudelist genereeritud SQL koodi käivitamise tulemusena. Seejärel kasutati Enterprise Architect CASE vahendit loodud andmebaasi struktuuri pöördprojekteerimiseks andmebaasi disaini esitavateks mudeliteks.

Joonis 9 esitab kinode registri füüsilise disaini ankurmudeli põhjal koostatud andmebaasis. Autor juhib tähelepanu, et andmetüübi nimi *timestamp* on samaväärne andmetüübi nimega *timestamp without time zone*.



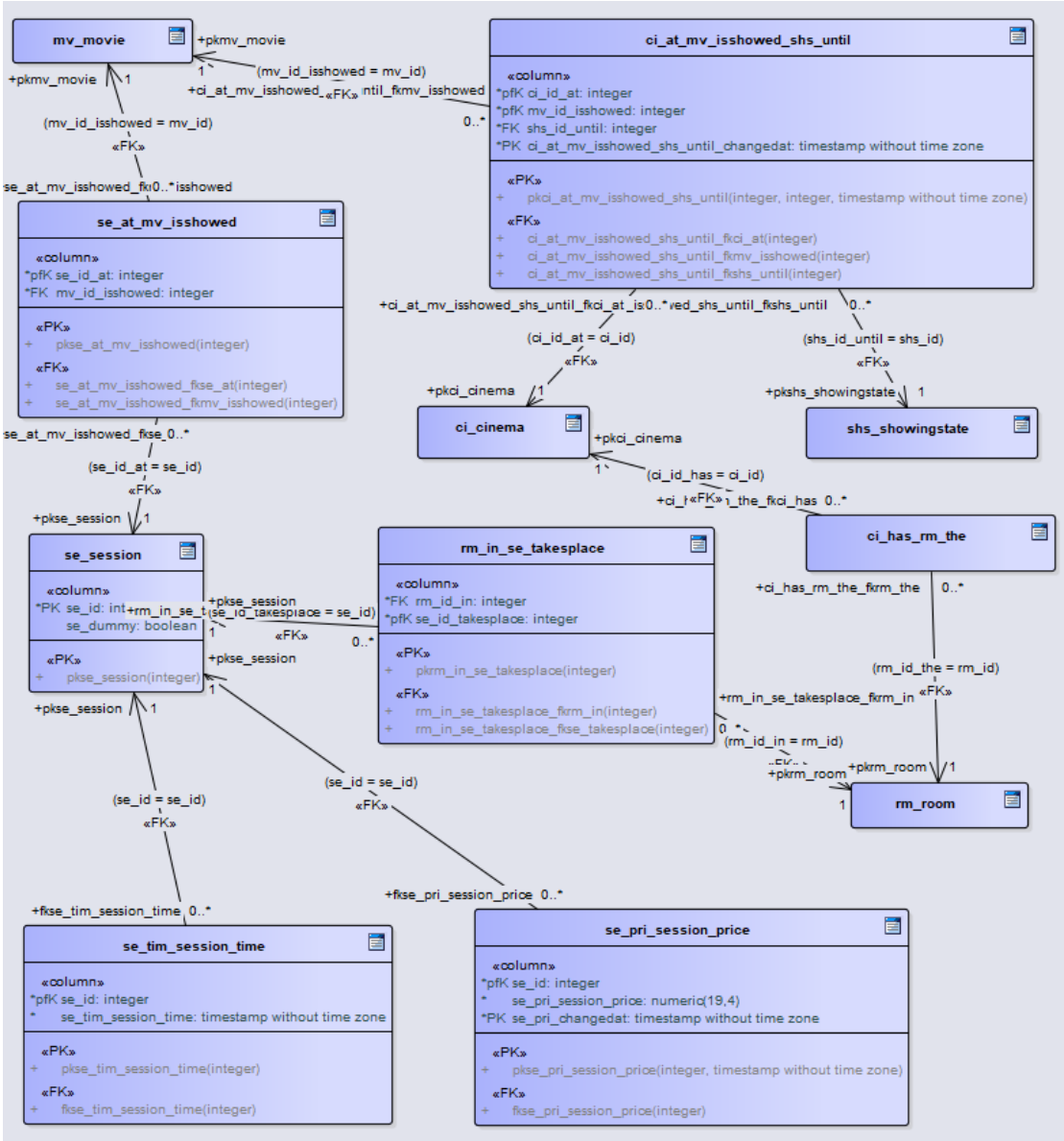
Joonis 9. Kinode registri füüsiline disain ankurmudeli põhjal koostatud andmebaasis.

Joonis 10 esitab filmide registri füüsilise disaini ankurmudeli põhjal koostatud andmebaasis.



Joonis 10. Filmide registri füüsiline disain ankurmudeli põhjal koostatud andmebaasis.

Joonis 11 esitab ankurmudeli põhjal koostatud andmebaasi füüsilise disaini kinode ja filmide vaheliste seoste kohta.



Joonis 11. Filmide ja kinode seoste füüsiline disain ankurmudeli põhjal koostatud andmebaasis.

Joonis 12 esitab ankurmudeli põhjal koostatud andmebaasi klassifikaatorite registri füüsilise disaini.

Joonis 12. Klassifikaatorite registri füüsiline disain ankurmudeli põhjal koostatud andmebaasis.

5.1.5 Ankurmudeli andmebaasi töökiiruse parandamise strateegiad

Käesoleva magistritöö autor otsustas, et katsetab päringute töökiiruse parendamiseks järgnevaid võimalusi:

- indeksi loomine,
- tabeli klasterdamine,
- tabeli osadeks jaotamine,
- materialiseeritud vaade.

Valitud strateegiatest on pikemalt kirjutatud jaotise 3.5.

5.1.6 Testülesanded

Magistritöö autor otsustas, et eksperimendis võetakse vaatluse alla ajaloolisi andmeid säilitavad tabelid ja nende peale ehitatud vaated. Tavalisest andmebaasist langesid valikusse tabelid *sessioon* ja *sessioon_price_history* ning ankurmudeliga genereeritud

andmebaasist neile vastavad tabelid *se_session* ja *se_pri_session_price*. Need tabelid valiti põhjusel, et nendesse on genereeritud kõige rohkem andmeid (vt tabel 7).

Tabel 7. Testimisse valitud tabelid.

Tabeli nimetus	**Ridade arv**
session	4630283
session_price_history	32411961
se_session	4630283
se_pri_session_price	32411961

Ülesandeks on leida informatsiooni konkreetse filmi kohta ja see koosneb kahest katsest. Esimeses katses küsitakse infot filmi seansi hinna ja aja kohta ning teises täiendatakse päringut ja küsitakse lisaks infot kino ja saali kohta.

5.1.7 Testandmete genereerimine

Testandmete genereerimiseks koostas töö autor PostgreSQL andmebaasi protseduurid. Andmed lisatakse kõigepealt tavalisse andmebaasi ja seejärel andmelinki (*dblink*) kasutades lisatakse andmed ankurmudeli põhjal genereeritud andmebaasidesse (vt näidet joonis 13). Andmelink on PostgreSQL laiendus, mille abil saab võtta ühendust teiste PostgreSQL andmebaasidega [17]. Andmete genereerimise protsess võttis aega ligemale 7.5 tundi.

Autor katsetas testandmete genereerimiseks ka *generate_series* andmebaasifunktsiooni, mis tagastab ridade hulga [18]. Lõpuks loobus ta selle kasutamisest, sest tema hinnangul võttis genereerimine niimoodi kauem aega. Näide testandmete genereerimisest PostgreSQL andmebaasi on esitatud lisas Lisa 2.

```
-- Andmete lisamine cinema
-- tabelisse nn tavalisse andmebaasi

      CREATE OR REPLACE PROCEDURE public.generate_cinema(
            IN num_of_cinemas integer)
      LANGUAGE 'plpgsql'
      AS $BODY$
      begin

            for cnt in 1..num_of_cinemas loop

                  insert into cinema(name, address)
                  values('Cinema Name ' ||md5(random()::text),
                        'Cinema Address ' ||md5(random()::text));

            end loop;

      end;
      $BODY$;

-- Andmete lisamine ankurmudeli põhjal genereeritud andmebaasi
-- kasutaded dblink-i.

      insert into public.ci_cinema(CI_ID) SELECT cinema_id FROM
      dblink('dbname=magister port=5432 host=localhost user=postgres
password=postgres',
      'SELECT * FROM public.cinema') AS tb2(cinema_id int, name text,
address text);
```

Joonis 13. Andmete genereerimise näide.

5.1.8 Andmebaasi suurus

Peale testandmete andmebaasidesse lisamist mõõdeti PostgreSQL süsteemi informatsiooni funktsioonide abil erinevate testandmebaaside suurust [19]. Tulemustest võib näha, et ankurmudeli andmebaasi maht oli suurem. Samas tuleb tõdeda, et tavalise andmebaasi korral loobuti indeksite kasutamisest ja see vähendab andmebaasi suurust.

- Tavaline andmebaas: 5633 MB. Loodud tabelite suurus ilma indeksiteta oli kokku 3928MB ja loodud indeksite suurus oli kokku 1696MB.

- Ankurmudeli andmebaas: 6163 MB. Loodud tabelite suurus ilma indeksiteta oli kokku 2934MB ja loodud indeksite suurus oli kokku 3220MB.

5.2 Eksperimendi tulemused

Töö järgnevas osa antakse ülevaade SELECT lausete töökiiruse uuringu katsetest ja tulemustest. Katsete läbiviimiseks koostas töö autor päringute käivitamiseks Python skriptid ja igat päringut käivitatakse järjest 25 korda.

5.2.1 Katse 1

Joonis 14 esitab esimese katse algse päringu, mida käivitatakse tavalise andmebaasi vastu ja millega soovitakse teada saada konkreetse filmi toimunud seansid, koos aja, hinna (viimane kehtiv hind) ja filmi nimega.

```
SELECT
       se.session_id,
       se.time AS session_time,
       se.price AS session_price,
       mv.name AS movie_name
FROM session se
LEFT JOIN movie mv ON se.movie_id = mv.movie_id
WHERE mv.name = 'Movie Name f86ffeeb6f3de9e1fc8d34395946d022'
AND se.time between '2023-01-01' and '2023-01-31'
ORDER BY se.price ASC
```

Joonis 14. Katse 1 päring tavalise andmebaasi jaoks.

Joonis 15 esitab esimese katse algse päringu, mida käivitatakse ankurmudeli põhjal genereeritud andmebaasi vastu.

```
SELECT
       ses.se_id AS session_id,
       ses.se_tim_session_time AS session_time,
       ses.se_pri_session_price AS session_price,
       mv.mv_nam_movie_name AS movie_name
FROM lse_session ses
LEFT JOIN se_at_mv_isshowed ishowed ON ses.se_id = ishowed.se_id_at
LEFT JOIN lmv_movie mv ON ishowed.mv_id_isshowed = mv.mv_id
WHERE mv.mv_nam_movie_name = 'Movie Name
f86ffeeb6f3de9e1fc8d34395946d022'
AND ses.se_tim_session_time between '2023-01-01' and '2023-01-31'
ORDER BY ses.se_pri_session_price ASC
```

Joonis 15. Katse 1 päring ankurmudeli põhjal koostatud andmebaasi jaoks.

Käivitades esitatud päringud vastu andmebaasi saadi tulemuseks 14 rida ja päringute käivitamise keskmisteks aegadeks 25 käivitamise korral olid vastavalt 0.7 ja 39 sekundit. Kuna päring vastu tavalist andmebaasi oli väga kiire, siis prooviti parandada päringut, mis käivitati vastu ankurmudeli põhjal koostatud andmebaasi. Tabel 8 esitab

esimese katse tulemused muudetud päringutega. Päringus nr. 1 eemaldati vaate *lmv_movie* kasutamine ja asendati see tabelite *mv_movie* ja *mv_nam_movie_name* tabelitega. Põhjus oli selles, et vaade *lmv_movie* tagastab meile kõik tabeliga *mv_movie* seotud atribuutide tabelite andmed, aga antud päringu jaoks ei olnud peale nime midagi muud vaja. Päringus nr. 2 pandi tagasi vaate *lmv_movie* kasutamine, kuid muudeti päringu struktuuri. Päringus nr. 3 võeti kasutusele B-puu indeks, mis loodi tabeli *mv_nam_movie_name* veerule *mv_nam_movie_name.*

Tabel 8. Katse 1 päringud ja käivitamise tulemused.

Nr.	**Päring**	**Käivitamise tulemus**
1.	SELECT ses.se_id AS session_id, ses.se_tim_session_time AS session_time, ses.se_pri_session_price AS session_price, mvname.mv_nam_movie_name AS movie_name FROM lse_session ses LEFT JOIN se_at_mv_isshowed ishowed ON ses.se_id = ishowed.se_id_at LEFT JOIN mv_movie mv ON ishowed.mv_id_isshowed = mv.mv_id LEFT JOIN mv_nam_movie_name mvname ON mv.mv_id = mvname.mv_id WHERE mvname.mv_nam_movie_name = 'Movie Name f86ffeeb6f3de9e1fc8d34395946d022' AND ses.se_tim_session_time between '2023-01-01' and '2023-01-31' ORDER BY ses.se_pri_session_price ASC	Käivitamiste arv 25 Kirjeid 14 Keskmine päringu käivitamise aeg (sekundites) 2.0145
2.	SELECT ses.se_id AS session_id, ses.se_tim_session_time AS session_time, ses.se_pri_session_price AS session_price, mv.mv_nam_movie_name AS movie_name FROM lmv_movie mv, se_at_mv_isshowed ishowed, lse_session ses WHERE mv.mv_nam_movie_name = 'Movie Name f86ffeeb6f3de9e1fc8d34395946d022' AND mv.mv_id = ishowed.mv_id_isshowed AND ses.se_id = ishowed.se_id_at AND ses.se_tim_session_time between '2023-01-01' and '2023-01-31' ORDER BY ses.se_pri_session_price ASC	Käivitamiste arv 25 Kirjeid 14 Keskmine päringu käivitamise aeg (sekundites) 1.59452

Nr.	Päring	Käivitamise tulemus
3.	SELECT ses.se_id AS session_id, ses.se_tim_session_time AS session_time, ses.se_pri_session_price AS session_price, mvname.mv_nam_movie_name AS movie_name FROM mv_movie mv, mv_nam_movie_name_index mvname, se_at_mv_isshowed ishowed, lse_session ses WHERE mv.mv_id = mvname.mv_id AND mvname.mv_nam_movie_name = 'Movie Name f86ffeeb6f3de9e1fc8d34395946d022' AND mv.mv_id = ishowed.mv_id_isshowed AND ses.se_id = ishowed.se_id_at AND ses.se_tim_session_time between '2023-01-01' and '2023-01-31' ORDER BY ses.se_pri_session_price ASC	Käivitamiste arv 25 Kirjeid 14 Keskmine päringu käivitamise aeg (sekundites) 1.257

5.2.2 Katse 2

Esimese katse päringuid täiendati, et saada teada kus kinos ja saalis seansid toimusid. Joonis 16 esitab teise katse algse päringu, mida käivitatakse tavalise andmebaasi vastu ja millega soovitakse teada saada konkreetse filmi toimunud seansid, koos aja, hinna (viimane kehtiv hind), filmi nimega, kino ja saaliga.

```
SELECT
       se.session_id,
       se.time AS session_time,
       se.price AS session_price,
       mv.name AS movie_name,
       cm.name,
       rm.name
FROM session se
LEFT JOIN movie mv USING (movie_id)
LEFT JOIN session_room srm USING (session_id)
LEFT JOIN room rm USING (room_id)
LEFT JOIN cinema cm USING (cinema_id)
WHERE  se.time between '2023-01-01' and '2023-01-31'
   and mv.name = 'Movie Name f86ffeeb6f3de9e1fc8d34395946d022'
ORDER BY se.price ASC
```

Joonis 16. Katse 2 päring tavalise andmebaasi jaoks.

Joonis 17 esitab teise katse algse päringu, mida käivitatakse ankurmudeli põhjal genereeritud andmebaasi vastu.

```
SELECT
       ses.se_id AS session_id,
       ses.se_tim_session_time AS session_time,
       ses.se_pri_session_price AS session_price,
       mv.mv_nam_movie_name AS movie_name,
       lrm.rm_nam_room_name AS room_name,
       lci.ci_nam_cinema_name AS cinema_name
FROM lmv_movie mv,
        se_at_mv_isshowed ishowed,
        lse_session ses,
        rm_in_se_takesplace rmst,
        lrm_room lrm,
        ci_has_rm_the chrm,
        lci_cinema lci
WHERE
       mv.mv_nam_movie_name = 'Movie Name
f86ffeeb6f3de9e1fc8d34395946d022'
  AND ses.se_tim_session_time between '2023-01-01' and '2023-01-31'
  AND mv.mv_id = ishowed.mv_id_isshowed
  AND ishowed.se_id_at = ses.se_id
  AND ses.se_id = rmst.se_id_takesplace
  AND rmst.rm_id_in = lrm.rm_id
  AND lrm.rm_id = chrm.rm_id_the
  AND chrm.ci_id_has = lci.ci_id
ORDER BY ses.se_pri_session_price ASC
```

Joonis 17. Katse 2 päring ankurmudeli põhjal koostatud andmebaasi jaoks.

Päringute käivitamisel tagastati 14 rida ja 25 käivitamise keskmised ajad olid vastavalt 1.4 ja 23 sekundit. Alljärgnev tabel 9 esitab teise katse tegevused ja tulemused.

Tabel 9. Katse 2 tegevused ja tulemused.

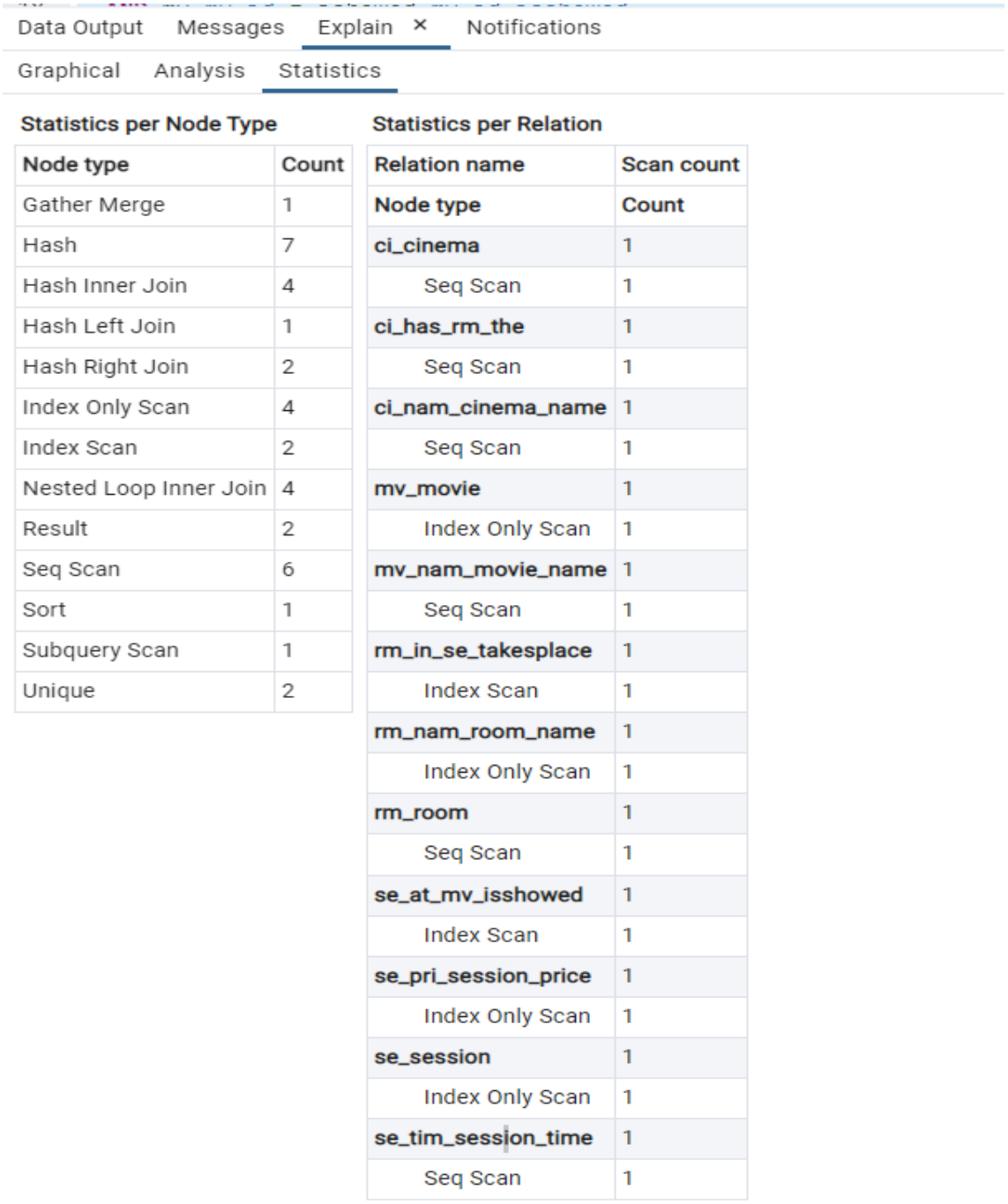
Tegevus	**Tulemus**
Tabeli *se_pri_session_time* osadeks jaotamine. Tekitati neli osa: 2020, 2021, 2022 ja 2023.	Käivitamiste arv 25 Kirjeid 14 Keskmine päringu käivitamise aeg (sekundites) 13.7822
Vaadete *lm_movie*, *lse_session* ja *lrm_room* asendamine materialiseeritud vaadetega.	Käivitamiste arv 25 Kirjeid 14 Keskmine päringu käivitamise aeg (sekundites) 0.985
Rakendati tabeli klasterdamist tabelile *se_tim_session_time* ja väljale	Käivitamiste arv 25 Kirjeid 14

Tegevus	Tulemus
se_tim_session_time.	Keskmine päringu käivitamise aeg (sekundites) 22.1627

5.2.3 Kokkuvõte

Eksperiment näitas, et tavalise andmebaasi vastu tehtud päringud olid üsna kiired ja ei vajanud täiendavat refaktoreerimist või muude päringu töökiiruse parandamise meetmete kasutuselevõttu, samas kui ankurmudeli andmebaasi päringud vajasid parendamist. Üks põhjuseid, miks päringud vastu ankurmudeli põhjal koostatud andmebaasi on aeglasemad, seisneb selles, et seoseid erinevate tabelite vahel on tunduvalt rohkem. Teises katses oli tavalise andmebaasi päringus kokku viis tabelit, samas kui ankurmudeli andmebaasi jaoks koostatud päringus oli kokku 19 tabelit (osad tabelid olid kasutusel läbi vaadete). Päringute kiirust aitas kõige paremini tõsta materialiseeritud vaadete kasutusele võtmine ja märgatav kasu oli ka tabeli osadeks jaotamisest.

Joonis 18 näitab päringu (vt joonis 17) täitmisel kasutatud tabeleid ja sealt selgub, et kuigi päring hõlmas kokku 19 tabelit (osad vaadete sees), siis kasutatud tabeleid on seal 12. Näiteks kuna me ei küsinud päringus infot filmi hinnangu kohta, siis sellepärast elimineeriti hinnangu ja selle klassifikaatori tabel (*Mv_rat_movie_rating* ja *Rtt_ratingtype*). See näitab, et antud päringu käivitamisel töötas edukalt ka tabelite elimineerimise teisendus. Samas päringu töökiirus ei olnud vaatamata sellele kuigi hea.



Joonis 18. Päringu seoste näide.

Tabelite klasterdamisest e nende andmete kettal sorteerimisest abi ei olnud. Samas MS SQL Serveri andmebaasisüsteemi korral, mille jaoks loodud koodi tõlkimise teel loodi algselt ka PostgreSQL koodigeneraator, on klasterdatud indeks põhiline töökiiruse parandamise meede.

Tavalise andmebaasi tabelid loodi ilma täiendavate indeksiteta, sh ilma indeksiteta välisvõtme veergudele. Algne plaan oli, et andmebaasis tehakse ka täiendusi tavalise andmebaasi päringute töökiiruse parandamiseks, lisades näiteks indekseid. Sellepärast kohe täiendavaid indekseid ei loonud. Kuna katsetustel tavalisel andmebaasil tehtavate päringute töökiirusega probleeme ei olnud, siis loobuti indeksite lisamisest. Töö autor ei taha väita, et indeksitest poleks tavalise andmebaasi puhul kasu ning et neid ei peaks töökiiruse paremaks timmimisel kasutama. Töökiiruse paremus tuli välja võrreldes ankurmudeli andmebaasiga ja seetõttu keskenduti sellele. Indeksid luuakse andmebaasisüsteemi poolt automaatselt primaarvõtme ja unikaalsuse kitsenduste alusel.

6 Koodigeneraatori parandused

Selles peatükis antakse ülevaade PostgreSQL SQL-koodgeneraatori lähtekoodis tehtud muudatustest.

6.1 Refaktoreerimine

Koodigeneraatori lähtekoodi sai tehtud järgmised parandused:

1. Taastatud tabelitele kommentaaride lisamise SQL-koodi genereerimine.
2. Taastatud *Tie* vaadete lisamise SQL-koodi genereerimine.
3. Taastatud *Anchor* vaadetele INSERT trigerite lisamise SQL-koodi genereerimine.
4. Eemaldatud *Anchor* vaadetele BEFORE INSERT ja AFTER INSERT trigerite genereerimine (vt eelmine punk). Kogu kood on viidud INSTEAD OF INSERT trigeri alla. Ka MS SQL Serveri genereeritud koodis oli tehtud nii, et kogu loogika on INSTEAD OF INSERT trigeris.

6.2 Täiendused

Koodigeneraatori lähtekoodi viidi sisse järgmised täiendused:

1. INSTEAD OF UPDATE ja INSTEAD OF DELETE trigerite lisamise SQL-koodi genereerimine *Anchor* vaadetele. Tänu loodud trigeritele saab nende vaadete kaudu muuta ankrule vastavate olemite atribuutide väärtuseid ja kustutada andmebaasist olemite andmeid.
2. Ajalooliste atribuutidega tabelitele indeksi lisamise SQL-koodi genereerimine. Töö autor otsustas sinna juure genereerida ka tabeli klasterdamise koodi, aga välja kommenteerituna. Kuna tabeli klasterdamine on selline tegevus, mis andmete lisamise või muutmise korral vajab selle kordamist, siis selle kasutamise otsuse peab tegema andmebaasi looja ja automaatselt seda mõistlik teha ei ole.

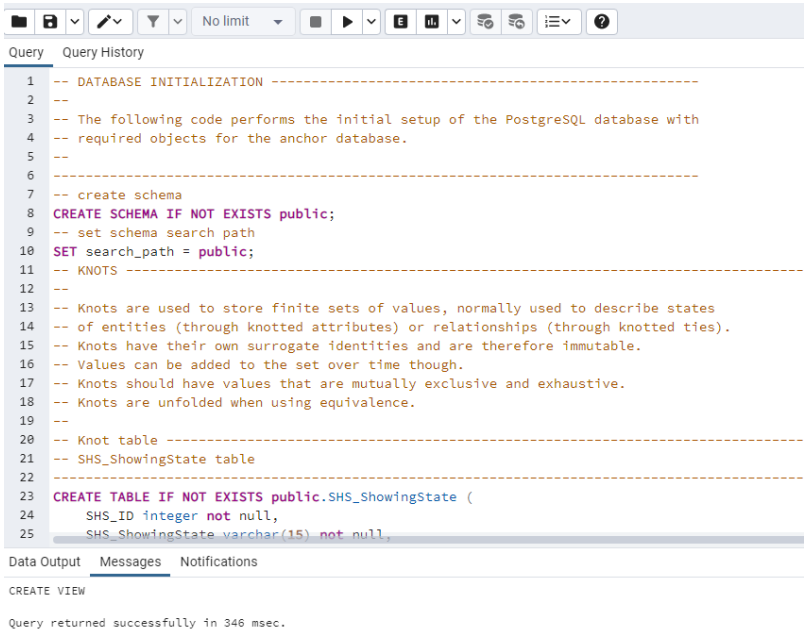
3. Tabeli osadeks jaotamise SQL-koodi genereerimine. Magistritöö autor otsustas, et lisab selle osa mallidesse välja kommenteerituna, et kui andmebaasi skript käima pannakse, siis automaatselt osadeks jaotamist ei toimu. Põhjus on selles, et osadeks jaotamist on väga keeruline automaatselt teha, selleks peab omama infot loodava andmebaasi kohta, et saaks defineerida tingimused, mille alusel toimuks tabeli osadeks jaotamine. Välja kommenteeritud kood lihtsalt näitab, et selline võimalus on olemas ja võib-olla oleks seda mõistlik teha.

4. Microsoft SQL Server SQL-koodigeneraatori malli *CreateTieTriggers.js* tõlkimine PostgreSQL SQL-koodigeneraatori jaoks sobivaks. Antud mall genereerib SQL-koodi, mis lisab INSTEAD OF trigerid *Tie* vaadetele. Sisult on need sarnased *Anchor* vaadetele lisatavate INSTEAD OF trigeritega ja võimaldavad nende vaadete kaudu sidemete andmeid muuta.

Kuigi töökiiruse katsetusel andis parima tulemuse materialiseeritud vaadete kasutamine, siis otsustati nende genereerimisest loobuda, sest vaateid on andmebaasis juba niigi palju ja seda tüüpi vaade nõuab regulaarset värskendamist. Liiga suur andmebaasiobjektide hulk muudab andmebaasist arusaamise ja selle haldamise keerulisemaks.

6.3 Testimine

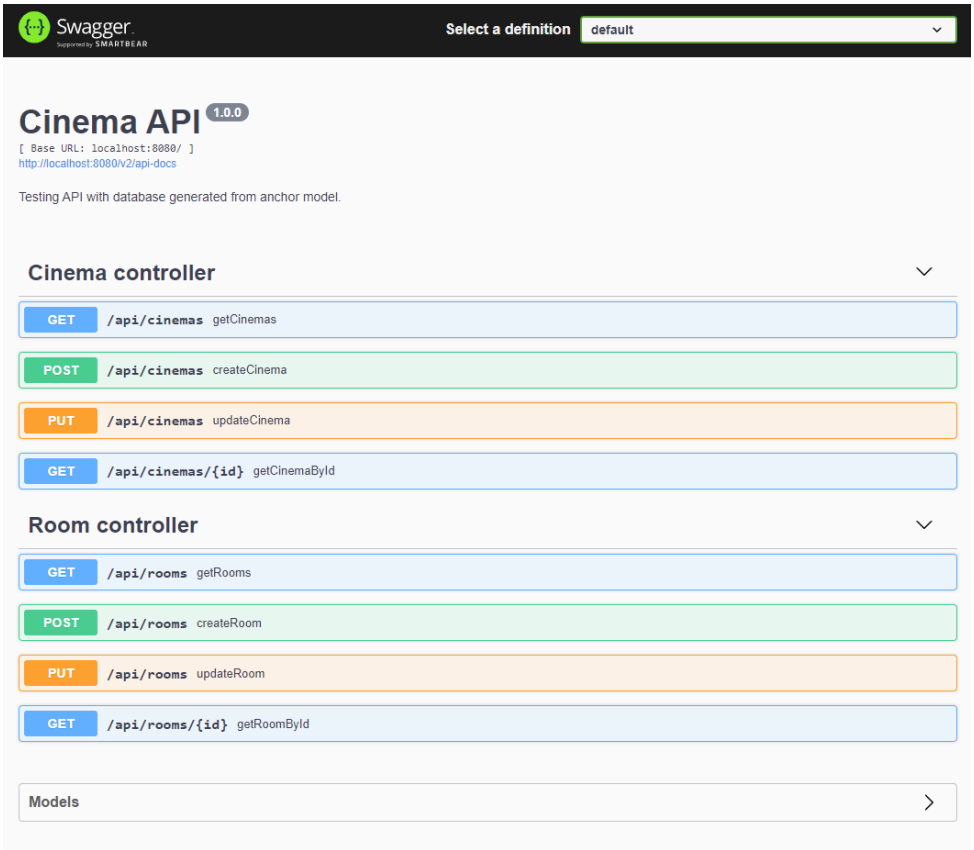
Pärast koodigeneraatori koodi parandamist ja täiendamist genereeris töö autor uue PostgreSQL SQL-koodi ja käivitas selle kasutades tarkavara pgAdmin. Koodi genereerimine kui ka skripti käivitamine õnnestusid (vt joonis 19).



Joonis 19. pgAdmin kuvatõmmis õnnestunud andmebaasi loomisest.

Andmebaasioperatsioonide testimiseks koostas töö autor erinevad INSERT, UPDATE ja DELETE lauseid andmete muutmiseks vaadetes *lci_cinema* ja *lrm_room*. Nende muudatuste tulemusena muutusid andmed tabelites *ci_cinema* ja *rm_room* ja nende atribuutide tabelites. Need kaks vaadet valiti sellepärast, et testitud saaks tabelid, kus hoitakse nii ajaloolisi andmeid kui ka need, kus selliseid andmeid ei ole. Koostatud SQL-laused käivitati kasutades rakendust pqAdmin.

Täiendavaks testimiseks programmeeris magistritöö autor REST (*Representational State Transfer*) API (*Application Programming Interface*), kasutades selleks Spring Boot [20] raamistikku. Lisaks andmemuudatuste operatsioonide testimisele oli API loomise eesmärgiks katsetada ankurmudeli põhjal koostatud andmebaasi sobivust operatiivandmeid haldavate onlain-tehingutöötluse süsteemide jaoks. Joonis 20 näitab, millised HTTP (*Hypertext Transfer Protocol*) meetodid andmete lisamiseks (POST [21] meetod), muutmiseks (PUT [22] meetod) ja pärimiseks (GET [23] meetod) realiseeriti. REST API kaudu tehti muudatus andmebaasis loodud vaates.



Joonis 20. API kasutajaliides.

6.4 Koodigeneraatori täienduste näide

Joonis 21 näitab, kuidas toimub PostgreSQL SQL-koodi lisamine generaatori mallidesse. Antud näites on realiseeritud INSTEAD OF UPDATE trigeri realiseerimine ankru vaatele, mis on vajalik andmete muutmiseks.

```
--INSTEAD OF UPDATE trigger ----------------------------------------
------------------------------------------------------------
--DROP TRIGGER IF EXISTS itu_l$anchor.name ON
$anchor.capsule\.l$anchor.name;
--DROP FUNCTION IF EXISTS $anchor.capsule\.itu_l$anchor.name();
CREATE OR REPLACE FUNCTION $anchor.capsule\.itu_l$anchor.name()
RETURNS trigger AS '

  BEGIN
   ~*/
    while (attribute = anchor.nextAttribute()) {
        knot = attribute.knot;
/*~
        IF NEW.$attribute.valueColumnName IS NULL THEN
            CREATE TABLE IF NOT EXISTS
$anchor.capsule\.$attribute.deletionName AS SELECT *, null::timestamp
as $attribute.deletionTimeColumnName FROM
$attribute.capsule\.$attribute.name WHERE FALSE;
            WITH tmp AS
            (DELETE FROM $attribute.capsule\.$attribute.name WHERE
$attribute.anchorReferenceName = OLD.$anchor.identityColumnName
RETURNING *)
               INSERT INTO $anchor.capsule\.$attribute.deletionName
SELECT *,  localtimestamp(0) FROM tmp where
$attribute.anchorReferenceName = OLD.$anchor.identityColumnName;
        ELSIF OLD.$attribute.valueColumnName <>
NEW.$attribute.valueColumnName THEN
~*/
        if(attribute.timeRange){
/*~
            INSERT INTO $attribute.capsule\.$attribute.name(
                $attribute.anchorReferenceName,
                $(schema.METADATA)?$attribute.metadataColumnName,
                $attribute.valueColumnName,
                $attribute.changingColumnName
            )VALUES(
                OLD.$anchor.identityColumnName,
                $(schema.METADATA)?NEW.$attribute.metadataColumnName,
                NEW.$attribute.valueColumnName,
                CASE
                    WHEN OLD.$attribute.changingColumnName <>
NEW.$attribute.changingColumnName THEN
                        NEW.$attribute.changingColumnName
                    ELSE
                        CAST($schema.metadata.now AS
$attribute.timeRange)
                END
            );
~*/
        }else {
/*~
            UPDATE $attribute.capsule\.$attribute.name
```

```
            SET $attribute.valueColumnName =
NEW.$attribute.valueColumnName
            WHERE $attribute.anchorReferenceName =
OLD.$anchor.identityColumnName;
~*/
       }
/*~
       END IF;
~*/
    }
/*~
    RETURN NEW;
  END;
' LANGUAGE plpgsql;

CREATE TRIGGER itu_l$anchor.name
INSTEAD OF UPDATE ON $anchor.capsule\.l$anchor.name
FOR EACH ROW
EXECUTE PROCEDURE $anchor.capsule\.itu_l$anchor.name();
~*/
```

Joonis 21. Näide INSTEAD OF UPDATE trigeri koodi genereerimisest ankru vaatele.

Joonis 22 näitab joonis 21 põhjal genereeritud SQL-koodi. Tegemist on INSTEAD OF UPDATE trigeriga.

Seal tehakse kolme asja kahe atribuudiga (RM_NOS_ROOM_NUMOFSEATS ja RM_NAM_ROOM_NAME) :

1. Kui uue atribuudi uus väärtus puudub (on NULL), siis tekitatakse ..Deleted tabel, kuhu kantakse üle seni atribuudi tabelis olnud read. Seda tehakse nii ajalooliste kui mitteajalooliste atribuutide korral. Siinkohal lähtutakse ankurmodelleerimise autori nägemusest, mille kohaselt UPDATE lause korral ei tohiks andmeid kustutada [24].

2. Kui tegemist on ajaloolise atribuudiga, siis lisatakse rida atribuudi tabelisse, sest atribuudi tabelis säilib ka ajalooline väärtus.

3. Kui tegemist ei ole ajaloolise atribuudiga, siis muudetakse rida atribuudi tabelis.

```
--INSTEAD OF UPDATE trigger ----------------------------------------
------------------------------------------------------------
--DROP TRIGGER IF EXISTS itu_lRM_Room ON public.lRM_Room;
--DROP FUNCTION IF EXISTS public.itu_lRM_Room();
CREATE OR REPLACE FUNCTION public.itu_lRM_Room() RETURNS trigger AS '
  BEGIN
        IF NEW.RM_NOS_Room_NumOfSeats IS NULL THEN

                    CREATE TABLE IF NOT EXISTS
public.RM_NOS_Room_NumOfSeats_Deleted AS SELECT *, null::timestamp as
RM_NOS_Deleted FROM public.RM_NOS_Room_NumOfSeats WHERE FALSE;
            WITH tmp AS
            (DELETE FROM public.RM_NOS_Room_NumOfSeats WHERE RM_ID =
OLD.RM_ID RETURNING *)
               INSERT INTO public.RM_NOS_Room_NumOfSeats_Deleted SELECT
*, localtimestamp(0) FROM tmp where RM_ID = OLD.RM_ID;

        ELSIF OLD.RM_NOS_Room_NumOfSeats <> NEW.RM_NOS_Room_NumOfSeats
THEN

                    UPDATE public.RM_NOS_Room_NumOfSeats
            SET RM_NOS_Room_NumOfSeats = NEW.RM_NOS_Room_NumOfSeats
            WHERE RM_ID = OLD.RM_ID;

        END IF;

           IF NEW.RM_NAM_Room_Name IS NULL THEN

              CREATE TABLE IF NOT EXISTS
public.RM_NAM_Room_Name_Deleted AS SELECT *, null::timestamp as
RM_NAM_Deleted FROM public.RM_NAM_Room_Name WHERE FALSE;
            WITH tmp AS
            (DELETE FROM public.RM_NAM_Room_Name WHERE RM_ID =
OLD.RM_ID RETURNING *)
               INSERT INTO public.RM_NAM_Room_Name_Deleted SELECT *,
localtimestamp(0) FROM tmp where RM_ID = OLD.RM_ID;

             ELSIF OLD.RM_NAM_Room_Name <> NEW.RM_NAM_Room_Name THEN
            INSERT INTO public.RM_NAM_Room_Name(
                RM_ID,
                RM_NAM_Room_Name,
                RM_NAM_ChangedAt
            )VALUES(
                OLD.RM_ID,
                NEW.RM_NAM_Room_Name,
                CASE
                    WHEN OLD.RM_NAM_ChangedAt <> NEW.RM_NAM_ChangedAt
THEN
                        NEW.RM_NAM_ChangedAt
                    ELSE
                        CAST(current_timestamp AS timestamp)
                END
```

```
            );
        END IF;
    RETURN NEW;
  END;
' LANGUAGE plpgsql;
CREATE TRIGGER itu_lRM_Room
INSTEAD OF UPDATE ON public.lRM_Room
```

Joonis 22. Ankru *Room* vaatele *lrm_room* genereeritud triger.

6.5 Tulemuste valideerimine

Uue generaatori versiooni abil SQL koodi genereerimisel oli tulemuseks 22 tabelit, 19 vaadet, 17 mitte-trigerifunktsiooni ja 12 trigerit (iga trigeriga käib paaris trigeri funktsioon).

Valideerimiseks katsetatakse andmemuudatuste operatsioonide läbiviimist andmebaasis, mille loomiseks on kasutatud vana generaatorit *vs.* andmebaasis, mille loomiseks on kasutatud uut generaatorit, veendumaks, et uues andmebaasis on kõrvaldatud kõik need probleemid, mis esinesid vanas andmebaasis. Tabel 10 annab ülevaate katsetamise tulemustest.

Tabel 10. Tulemuste valideerimine.

Tegevus	**Kommentaar**	**Vana generaator**	**Uus generaator**
Andmebaasi loomine kasutades genereeritud skripti	Andmebaasi loomine oli katki ja töö autor tegi seal paranduse ka vana generaatori osas, et saaks andmete muutmise operatsioone edasi testida.	ERROR: syntax error at or near ":" LINE 896: :bigint AS Metadata_TYP ^ SQL state: 42601 Character: 36570	Query returned successfully in 304 msec.

Tegevus	**Kommentaar**	**Vana generaator**	**Uus generaator**
`insert into lci_cinema(ci_nam_cinema_name, ci_adr_cinema_address) values('Kino Koit', 'Tallinn Lepa tee 3');`	Uue kino andmete registreerimine lisades andmed vaatesse, kuhu on koondatud iga kino kohta selle atribuutide kõige värskemad väärtused	ERROR: cannot insert into view "lci_cinema" DETAIL: Views that do not select from a single table or view are not automatically updatable. HINT: To enable inserting into the view, provide an INSTEAD OF INSERT trigger or an unconditional ON INSERT DO INSTEAD rule. SQL state: 55000	INSERT 0 1 Query returned successfully in 236 msec.
`update lci_cinema set ci_adr_cinema_address = 'Tallinn Koidu tee 3' where ci_id = 1`	Kino andmete muutmine, muutes andmeid vaates, kuhu on koondatud iga kino kohta selle atribuutide kõige värskemad väärtused	ERROR: cannot update view "lci_cinema" DETAIL: Views that do not select from a single table or view are not automatically updatable. HINT: To enable updating the view, provide an INSTEAD OF UPDATE trigger or an unconditional ON UPDATE DO INSTEAD rule. SQL state: 55000	UPDATE 1 Query returned successfully in 96 msec.
`update lrm_room set rm_nam_room_name = 'Room President suite'`	Katsetatakse ajaloostatud andmete muutmist	ERROR: cannot update view "lrm_room"	UPDATE 1

Tegevus	Kommentaar	Vana generaator	Uus generaator
`where rm_id = 1`	andmebaasis loodud vaate põhjal, kuhu on koondatud iga saali kohta selle atribuutide kõige värskemad väärtused	DETAIL: Views that do not select from a single table or view are not automatically updatable. HINT: To enable updating the view, provide an INSTEAD OF UPDATE trigger or an unconditional ON UPDATE DO INSTEAD rule. SQL state: 55000	Query returned successfully in 76 msec.

6.6 Tulemuste avaldamine

Magistritöö materjalid on kättesaadavad Githubis-s aadressil https://github.com/kajargit/magister. Seal on loodud mudelid (nii ankurmudel kui UMLis loodud mudelid), mudelitest genereeritud SQL kood, testandmete genereerimise kood, andmebaaside katsetamiseks mõeldud SQL laused, REST API ja ka täiendatud generaator.

Töö autoril on loomisel kontakt ankurmodelleerimise veebipõhise modelleerimisvahendi looja Lars Rönnbäck-iga, et töös tehtud parendused ja täiendused jõuaksid projekti versioonihaldusesse.

7 Arendusvaade

Järgnevalt antakse ülevaade sellest, mida võiks töö autori arvates tulevikus modelleerimisvahendi arendamisel edasi teha.

7.1 Arendustööde seisu jälgimine

Magistritööga alustades oli töö autorile suureks probleemiks aru saada, mis seisus on PostgreSQL SQL-koodigeneraatori arendusööd ning mis seisus on need võrreldes Microsoft SQL Server SQL-koodigeneraatoriga. Projektil peaks olema detailsem muudatuste või arendustööde logi, mis annab ülevaate hetkeseisust. Magistritöö autori arvates on selline ülevaade projekti jätkusuutlikkuse tagamiseks hädavajalik.

7.2 Temporaalsus

Temporaalsus modelleerimisvahendi kontekstis tähendab ajalooliste andmete käsitluspõhimõtteid. Praegu saab rakenduses valida kahe valiku vahel: monotemporaalne ja bitemporaalne generatsioon. Monotemporaalsuse korral jälgitakse väärtuste muutmise aega, aga bitemporaalsuse korral lisamise ja muutmise aega. PostgreSQL SQL-koodigeneraatori korral on praegu arendatud, vaid monotemporaalsuse tuge, kuid tuleks lisada ka bitemporaalsuse tugi.

7.3 Töökiiruse parandamine

Luua võimalus genereerida valikuliselt vaadete asemel materialiseeritud vaateid koos nende värskendamise koodiga. Seda võiks kasutada näiteks suure andmemahuga andmebaaside korral kui parema töökiiruse huvides ollakse valmis aktsepteerima materialiseeritud vaates olevate andmete vananemise ja sellest tuleneva vaate jätkuva värskendamise vajadust.

7.4 Testimine

Vajalik oleks projektile juurde lisada testid. Need oleksid hädavajalikud vältimaks olukorda, kuhu sattus töö autor, kui selgus, et PostgreSQL SQL-koodigeneraatori kood on osaliselt üle kirjutatud Microsoft SQL Server-i jaoks mõeldud koodiga ning mille

tõttu varasemalt töötanud funktsionaalsus enam ei töötanud. Töö autoril on ka endal üks mõte, kuidas võiks kaitsta töötavat koodi juhuslike muudatuste vastu. Näiteks võiks koostada ühe ankurmudeli, kus on olemas enamus elemente ja selle põhjal genereerida kõikide modelleerimisvahendi poolt toetavate andmebaasisüsteemide loomise SQL-skriptid ning salvestada need failidesse. Loomulikult tuleks need üle testida, et nende käivitamisel vigu ei esine. Nimetame neid etalonideks, mille vastu edaspidi saaks generaatori tööd kontrollida. Kontrollimiseks tuleks luua eraldi skript, mis genereerib hetkeseisust SQL-koodi ja kontrollib genereeritud lauseid vastu varasemalt loodud etalone. Selline kaitse aitab juhuslike muudatuste vastu, aga kui tehakse arendust (SQL kood muutub), siis peab loomulikult uued etalonid genereerima.

7.5 Kasutus operatiivandmete andmebaaside loomiseks

Peale seda, kui koodigeneraator väljastab ankurmudeli korral täieliku ja töötava koodi, saaks hakata eksperimenteerima, et teha kindlaks see, kuivõrd efektiivne oleks ankurmodelleerimise andmebaasi kasutamine operatiivandmete andmebaasina ja kuivõrd lihtne oleks sellele luua andmebaasirakendust ning seda koos skeemimuudatustega muuta. Jaotises 6.3 kirjeldatud REST API liides näitab, et see oleks põhimõtteliselt võimalik. Andmebaasirakendus peaks põhinema andmebaasis loodud vaadetel. Tuleb uurida, kuivõrd lihtne oleks jõustada erinevat tüüpi kitsendusi (mis kontrollivad andmete vastavust valideerimisreeglitele) selles andmebaasis ja kas selliste kitsenduste koodi genereerimist saaks automatiseerida.

8 Kokkuvõte

Käesoleva magistritöö eesmärgiks oli ankrumodelleerimise veebipõhise keskkonna PostgreSQL SQL-koodigeneraatori edasiarendamine. Töö jaoks loodi filmide näitamise andmebaasi kontseptuaalne andmemudel ning ankurmudel. Kontseptuaalse andmemudeli alusel loodi tavalise disainiga andmebaas, kus tabelid ei ole üldjuhul kuuendal normaalkujul. Ankurmudeli põhjal genereeriti andmebaas, mis järgib ankurmodelleerimise põhimõtteid ja kus tabelid on kuuendal normaalkujul. Loodud kahe andmebaasi põhjal tehti SELECT lausete töökiiruse uuring ning testiti andmemuudatuste operatsioone vastu ankurmudeli põhjal koostatud andmebaasi. Eksperimendi ja testimise tulemuste põhjal parandati ja täiendati PostgreSQL SQL-koodigeneraatorit.

Töö tulemused:

- Kõrvaldati segadus, mis oli põhjustatud PostgreSQL SQL-koodigeneraatori lähtekoodi failide üle kirjutamisest Microsoft SQL Server SQL-koodigeneraatori failidega.
- Täiendati PostgreSQL SQL-koodigeneraatori malle viisil, mille vajadus selgus andmemuudatuste operatsioone testides.
- Analüüsiti erinevaid SELECT lausete töökiiruse parandamise strateegiaid.
- Täiendati PostgreSQL SQL-koodigeneraatori malle vastavalt SELECT lausete töökiiruse uuringu tulemustele.
- Kirjeldati süsteemi erinevaid arendusvõimalusi.

Magistritöö autor ei olnud varasemalt ankurmodelleerimisega kokku puutunud ja seega võib öelda, et antud metoodikaga esmakordselt nii süviti tutvumine oli väga huvitav ja silmaringi laiendav. Autor loodab, et tal avaneb kunagi ka tööalaselt reaalne võimalus rakendada andmebaasi loomisel ankurmodelleerimist.

Magistritöö materjalid on kättesaadavad GitHubis-s aadressil https://github.com/kajargit/magister

Autor tänab käesoleva magistritöö juhendajat Erki Eessaart osutatud abi eest.

Kasutatud kirjandus

[1] L. Rönnbäck, O. Regardt, M. Bergholtz, P. Johannesson ja P. Wohed, „Anchor modeling — Agile information modeling in evolving data environments,“ *Data & Knowledge Engineering,* 2010.

[2] L. Rönnbäck ja V. Krumlinde, „The Anchor Modeler,“ 22 01 2021. [Võrgumaterjal]. Available: http://www.anchormodeling.com/modeler/latest.

[3] E. Saal, „Ankurmodelleerimise mudelite realisatsiooni generaator PostgreSQL jaoks,“ [Magistritöö], Infotehnoloogia teaduskond, Tallinna Tehnikaülikool, Tallinn, 2015. [Võrgumaterjal]. Available: https://digikogu.taltech.ee/et/item/7227f765-0d27-49d8-825d-1b3f0373f250.

[4] A. R. Hevner, S. T. March, J. Park ja S. Ram, „Design Science in Information Systems Research,“ *MIS Quarterly,* kd. 28, nr 1, p. 31, 2004.

[5] „DB-Engines,“ [Võrgumaterjal]. Available: https://db-engines.com/en/ranking.

[6] E. Potter, „Ankurmodelleerimise ja Oracle Workspace Manageri võrdlus temporaalsete andmete haldamisel SQL-andmebaasides,“ [Bakalaureusetöö],Infotehnoloogia teaduskond, Tallinna Tehnikaülikool, Tallinn, 2013. [Võrgumaterjal]. Available: https://maurus.ttu.ee/download.php?aine=346&document=29299&tyyp=do.

[7] E. Saal, „Puuduvate andmete esitamine SQL-andmebaasides,“ [Bakalaureusetöö],Infotehnoloogia teaduskond, Tallinna Tehnikaülikool, Tallinn, 2010. [Võrgumaterjal]. Available: https://maurus.ttu.ee/download.php?aine=346&document=29297&tyyp=do.

[8] M. Soobik, „Andmebaasi disainide hindamine Saaty meetodi abil,“ [Magistritöö],Infotehnoloogia teaduskond,Tallinna Tehnikaülikool, Tallinn, 2009. [Võrgumaterjal]. Available: https://maurus.ttu.ee/download.php?aine=346&document=29302&tyyp=do.

[9] L. Rönnbäck ja N. Golov, „Big data normalization for massively parallel processing databases.,“ *Computer Standards & Interfaces,* nr 54, pp. 86-93, 2017.

[10] „MariaDB,“ [Võrgumaterjal]. Available: https://mariadb.com/kb/en/what-is-table-elimination/.

[11] K. Pukk, „Tabelite elimineerimise teisenduse rakendamine erinevate SQL-andmebaasisüsteemide poolt,“ [Bakalaureusetöö],Infotehnoloogia teaduskond, Tallinna Tehnikaülikool, Tallinn, Eesti, 2014. [Võrgumaterjal]. Available: https://maurus.ttu.ee/download.php?aine=346&document=29319&tyyp=do.

[12] „PostgreSQL: The World's Most Advanced Open Source Relational Database,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/sql-cluster.html.

[13] D. Kašnikova, „Vaadete mõju päringute täitmisplaanide koostamisele kahe andmebaasisüsteemi näitel,“ [Magistritöö],Infotehnoloogia teaduskond, Tallinna Tehnikaülikool, Tallinn, 2015. [Võrgumaterjal]. Available:

https://digikogu.taltech.ee/et/Item/ead6c641-4e76-4a25-b974-d14f265d0cb3.

[14] „PostgreSQL: The World's Most Advanced Open Source Relational Database,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/rules-materializedviews.html.

[15] „PostgreSQL: The World's Most Advanced Open Source Relational Database,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/ddl-partitioning.html.

[16] „PostgreSQL Tutorial,“ [Võrgumaterjal]. Available: https://www.postgresqltutorial.com/postgresql-views/postgresql-updatable-views/.

[17] „PostgreSQL: The World's Most Advanced Open Source Relational Database,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/contrib-dblink-function.html.

[18] „PostgreSQL: The World's Most Advanced Open Source Relational Database,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/functions-srf.html.

[19] „PostgreSQL: The World's Most Advanced Open Source Relational Database,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/functions-info.html.

[20] „Spring Boot Reference Guide,,“ [Võrgumaterjal]. Available: http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/.

[21] „Resources for Developer, by Developers,“ Mozilla Corporation, [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST.

[22] „Resources for Developers, by Developers,“ Mozilla Corporation, [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT.

[23] „Resources for Developers, by Developers,“ Mozilla Corporation, [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET.

[24] L. Rönnbäck, „Anchormodeling,“ [Võrgumaterjal]. Available: https://www.anchormodeling.com/poor-mans-auditability/.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Kajar Karuauk

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Ankurmudeli põhjal PostgreSQL jaoks SQL koodi koostava generaatori edasiarendamine“, mille juhendaja on Erki Eessaar
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

10.05.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Näide andmete genereerimise protseduurist, kasutades generate_series funktsiooni.

```
CREATE OR REPLACE PROCEDURE generate_cinema(num_of_cinemas INTEGER)
LANGUAGE SQL AS
$$
INSERT INTO Cinema(name, address)
SELECT
'Cinema Name ' ||md5(random()::text),
'Cinema Addressr ' ||md5(random()::text)
FROM generate_series(1,num_of_cinemas, 1);
$$;
```