TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Erle Maido 164430IABB

Mõned failide salvestamise disainimustrid SQL-andmebaasi kasutavate andmebaasirakenduste jaoks

Bakalaureusetöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2021

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: Erle Maido

18.05.2021

Annotatsioon

Töö eesmärgiks on süstematiseerida infot SQL-andmebaaside disainilahenduste kohta, mis on mõeldud toetama failide salvestamist vajavaid andmebaasirakendusi ning analüüsida ja võrrelda neid disaine nii andmebaasi kui ka andmebaasirakenduse seisukohast.

Töö tulemusena esitati mustri formaadis neli failide salvestamist toetavat SQL-andmebaasi disainimustrit: *bytea*, *suurobjekt*, *failisüsteem* ja *pilvelahendus*. Esimesed kaks nendest põhinevad spetsiifilistel PostgreSQL andmebaasisüsteemi võimekustel ja on alammustrid, mis laiendavad üldist mustrit, mille kohaselt võib hoida faile andmebaasis. Kõikide disainimustrite põhjal projekteeriti ja realiseeriti PostgreSQL-is näiteandmetega andmebaasid. Iga loodud andmebaasi põhjal realiseeriti andmete lisamise, muutmise, kustutamise ja vaatamise funktsionaalsusega veebirakendus C# programmeerimiskeeles kasutades .NET raamistikku, mille autor seadis üles Amazoni pilvekeskkonda.

Disainilahenduste võrdlemiseks planeeriti ning viidi läbi mõõtmised ja selle kaudu võrreldi disainilahendusi andmebaasi andmemahu, päringute täitmise ja rea lisamise töökiiruse, rakenduse füüsiliste koodiridade arvu, rakenduse loomise subjektiivse lihtsuse ja rakenduses andmete kuvamiseks kuluva aja põhjal. Hindamaks andmemahtude mõju päringute täitmisele ja andmete kuvamiseks kuluva aja kiirustele, viis autor mõõtmised läbi kolme erineva andmehulgaga.

Töö käigus realiseeritud rakenduste lähtekood on saadaval https://github.com/erlemaido/File-Storage-Design-Patterns. Kuni kaitsmise perioodi lõpuni hoiab autor ühte näidisrakendust üleval Amazoni pilveserveris. Täpsem info on GitHub'i lehel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 74 leheküljel, 9 peatükki, 39 joonist, 17 tabelit.

Abstract

Some File Storage Design Patterns for Database Applications that Use an SQL Database

The goal of this thesis is to systemize information about design solutions for SQL databases, which are meant to support database applications that need to save files and analyse and compare the designs both from the perspective of database and database application.

The work presents in a pattern format four SQL database designs that support saving files: *bytea*, *large object*, *filesystem*, and *cloud storage*. The first two are specific to PostgreSQL database management system (DBMS) and specialize a general pattern that files can be stored in a database. Based on each design a database and a database application were designed and implemented. The applications were meant for managing product catalogue. The databases were created by using PostgreSQL DBMS. For each database, a web application with the functionality for inserting, editing, deleting, and viewing data was built. The web applications were written in C# programming language and used .NET framework. The applications were deployed to Amazon cloud environment.

Measurements based on the implemented databases and applications were planned and executed in order to compare the designs based on data size, execution speed of queries and insert operations, physical lines of application code, the subjective difficulty of implementing the application, and the time that it takes to load a page of the application. To evaluate the effect of data size to the query execution speed and speed of loading pages the measurements were made with three different data sizes. There was strong positive linear correlation between the growth of data size and worsening of the performance characteristics.

One cannot point out one definitely the best or the worst design. It all depends on the context and different designs have different strong as well as weak points. For instance, keeping files in the local file system or in the cloud results with a smaller database and

good performance of read operations whereas keeping files in the database ensures better consistency, security, and simplifies application development.

The source code of the applications is available at https://github.com/erlemaido/File-Storage-Design-Patterns. The author will host one of the implemented sample applications on Amazon cloud server until the end of the defence period. More information can be found on the GitHub page.

The thesis is in Estonian and contains 74 pages of text, 9 chapters, 39 figures, 17 tables.

Lühendite ja mõistete sõnastik

Andmete terviklikkus	*Data integrity* Andmed on reeglitega kooskõlas.
API	*Application Programming Interface* Rakendusliides
ASCII	*American Standard Code for Information Interchange* „Numbrilised koodid vahemikus 0-127, mis esitavad ingliskeelseid tähemärke, numbreid, keelemärke ja klaviatuuri funktsioone“ [1].
AWS	*Amazon Web Services* Amazoni tütarettevõte, mis pakub pilveandmetöötluse platvormi.
BLOB	*Binary Large Object* Andmebaasi salvestatud suur bitiplokk, mida andmebaasihaldur ise ei interpreteeri [2].
Bytea	PostgreSQL andmetüüp, mis võimaldab salvestada binaarsõnesid.
CASE	*Computer-Aided Software Engineering* Mudelite koostamist võimaldav programm, mis toetab infosüsteemi või tarkvara arendajat arendus- ja haldusprotsesside tegevuste juures.
CPU	*Central Processing Unit* Arvuti keskseade, mis täidab arvutiprogrammide juhiseid ning on arvuti ülesannete täitmisel üks põhilisi vahendeid [3].
EC2	*Amazon Elastic Compute Cloud* Amazoni poolt pakutav serveriruumi rentimise teenus.
Geomeetriline keskmine	„Positiivsete suuruste korrutis, mis on astendatud nende suuruste arvu pöördväärtusega“ [4].
MVC	*Model-View-Controller* Rakenduse arhitektuuri mudel.
OID	*Object Identifier* Objekti identifikaator.
PostgreSQL	Avatud lähtekoodiga, tasuta, SQL keelt kasutada võimaldav andmebaasihaldur e andmebaasisüsteem.
S3	*Amazon Simple Storage Service* AWS-i poolt pakutav failide talletamise pilveteenus

SSH	*Secure Socket Shell* Krüptograafiline võrguprotokoll masinate vahel turvalise ühenduse loomiseks.
SQL	*Structured Query Language* Andmebaasikeel andmete otsimiseks ja muutmiseks ning andmebaasiobjektide, andmebaasis toimuvate tehingute ja andmebaasiobjektide kasutamiste õiguste haldamiseks.
TOAST	*The Oversized-Attribute Storage Technique* Andmete andmebaasi sisesekeemis salvestamise tehnika PostgreSQL-is.
UML	*Unified Modeling Language* Keel, mille abil saab visuaalsete kujundite abil kirjeldada info- ja tarkvarasüsteemide struktuuri ning käitumist.
URI	*Uniform Resource Identifier* Formaaditud sõne, mis identifitseerib mingi ressursi (nt internetiaadressina) [5].
Voogedastus	*Streaming* Tehnika andmete edastamiseks, kus andmete kasutamiseks ei pea ootama, et terve fail oleks alla laetud. Andmed kogutakse puhvermällu ja edastatakse hulkade kaupa [2].
XFS	Silicon Graphics poolt IRIX operatsioonisüsteemi jaoks loodud suure jõudlusega päevikuga failisüsteem, mis porditi Linuxi tuuma ja on toetatud enamikes Linuxi distributsioonides [6].

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Andmed võivad olla andmebaasisüsteemide abil loodud andmebaasides. Kuid see ei ole ainus keskkond, kus IT-süsteemides on andmed. Palju andmeid on ka näiteks erinevate rakendusprogrammide failides. Andmetest tervikpildi saamiseks ja kasutajate mugavaks teenindamiseks peavad IT süsteemid suutma neid andmeid integreerida ja ühises keskkonnas pakkuda.

Meediat (heli, pilt, video) sisaldavate failide, aga ka muude failide (tekstifailid, CASE vahendite failid jne) salvestamine ja haldamine andmebaasirakendustes on teema, mis oli ja on oluline juba pikemat aega ja on muutunud tänu veebirakenduste populaarsuse kasvule aina olulisemaks. Käesolev lõputöö keskendub erinevate failide salvestamise disainilahenduste uurimisele SQL-andmebaasi kasutavate andmebaasirakenduste jaoks. Järgnevalt annab autor ülevaate töö taustast ja probleemidest, tuues lisaks välja töö peamised eesmärgid ja nende saavutamiseks kasutatava metoodika. Samuti antakse ülevaade lõputöö ülesehitusest.

1.1 Taust ja probleem

Pilte ja muud meediat kasutatakse tänapäeval enamikes rakendustes ning seetõttu on rakendusi, mis haldavad terabaitide mahus pilte. Piltidena talletatakse nii inimeste portreesid, erinevaid paiku, mälestusi puhkusereisidest ja pidudest kui ka mitmesuguste analüüside või katsete tulemusi [7]. Tihtipeale on failid seotud olemitega, mille kohta hoitakse andmebaasis andmeid. Tehes andmebaasipäringuid olemite kohta, on vaja saada kätte ka olemitega seotud failid (nt pildid) [8]. Digitaalmeedia levik tõi kaasa rakenduste andmeobjektide suurenemise ning süsteemid, mis vanasti käitlesid staatilisi faile, tegelevad nüüd dünaamilise ja versioneeritud sisuga [9]. Tänapäeva ühte populaarsemat veebipõhist suhtlusvõrgustikku Instagram'i kasutab igapäevaselt ligi 500 miljonit kasutajat. Ühes sekundis laetakse üles ligikaudu 995 pilti ning tänase hetkeni on platvormile üles laetud üle 50 miljardi pildi ja see arv üha kasvab [10]. Veel suuremaid arve näeb Facebooki puhul, kus iga sekundi kohta laetakse kasutajate poolt üles ligikaudu

136 000 pilti [11]. Samas failide salvestamise disainilahendused on olulised igasuguse suurusega ja väga erinevate valdkondade (nt e-poed, koolituskeskkonnad, teenusepakkujad, kuulutuste avaldajad) süsteemides ning andmebaasi kavandamisel on oluline teada, millised on erinevate disainilahenduste eelised ja puudused, et tagada rakenduse ja andmebaasi maksimaalne võimekus.

1.2 Ülesande püstitus

Töö eesmärgiks on süstematiseerida infot SQL-andmebaaside disainilahenduste kohta, mis on mõeldud toetama failide salvestamist vajavaid andmebaasirakendusi. Selleks analüüsitakse ja võrreldakse erinevaid andmebaasi disaine ning realiseeritakse iga disanilahenduse põhjal andmebaas ja andmebaasirakendus.

Töö põhitulemused.

- Failide salvestamist toetavate SQL-andmebaasi disainilahenduste leidmine ja mustri formaadis kirja panemine.
- Kõigi kirja pandud disainimustrite põhjal näiteandmetega andmebaaside realiseerimine PostgreSQL-is. Näiteandmebaas on toodete ja nende piltide andmete hoidmiseks.
- Iga loodud andmebaasi põhjal andmete lisamise, muutmise, kustutamise ja vaatamise funktsionaalsusega veebirakenduse loomine C# programmeerimiskeeles kasutades .NET raamistikku.
- Andmebaasi disainilahenduste võrdlemine andmemahu, päringute ja ridade lisamise töökiiruse, rakenduse füüsiliste koodiridade arvu, rakenduse loomise subjektiivse lihtsuse ja rakenduses andmete kuvamiseks kuluva aja põhjal.

Analüüsitavad lahendused on mõeldud kasutamiseks organisatsioonide ja üksikisikute infosüsteemides, kus andmehulgad, andmete juurdekasv ja struktuuri varieeruvus on mõõdukad. Töö ei käsitle ning ei esita soovitusi suurandmete halduse kohta.

1.3 Metoodika

Lõputöö tegemine põhineb disainiteaduse (*design science*) meetodil, mille idee on arendada artefakti ehk tehist [12]. Kõige pealt uurib autor olemasolevat kirjandust. Töö

tulemusena valmivad uued disainimustrid. Disainimustrite põhjal luuakse CASE-vahendis esmalt andmebaasi tabelite füüsilist disaini kirjeldavad mudelid. Seejärel realiseeritakse tabelid andmebaasisüsteemis ning võrreldakse disaine andmemahu ja andmete otsimise ning lisamise operatsioonide töökiiruse põhjal. Kõikide andmebaaside põhjal realiseeritakse andmete haldamise näiterakendus, mida võrreldakse omavahel rakenduse loomise subjektiivse lihtsuse, koodiridade arvu ja andmete kuvamise aja põhjal.

Töö käsitleb SQL-andmebaase, sest need on tänapäeval endiselt väga laialdlaselt kasutusel ning kuna SQL on olnud väga hea uuendustega kohaneja, siis pole ka ette näha selle olulise vähenemist lähiajal. 2021. aasta aprillikuu seisuga oli kõige populaarsemate andmebaasisüsteemide esikümnes seitse SQL-andmebaasisüsteemi [13].

Füüsilise andmebaasi disaini mudelite loomiseks kasutatakse Enterprise Architect CASE modelleerimisvahendit, mis on mõeldud erinevate süsteemide (sh info- ja tarkvarasüsteemid) protsesside ja struktuuri visualiseerimiseks, analüüsimiseks, modelleerimiseks, testimiseks ja ülalhoiu toetamiseks [14]. Modelleerimisvahendi valiku põhjuseks on lõputöö autori varasem modelleerimise kogemus antud tarkvaraga. Tegemist on võimeka kommertstarkvaraga, mille tasuta täisversioon on üliõpilastele ülikooli vahendusel kättesaadav.

Lõputöö näiteandmebaasid on realiseeritud PostgreSQL andmebaasisüsteemis. PostgreSQL valiku põhjuseks on autori varasem kokkupuude antud süsteemiga, avatud lähtekood, võimalike disainivalikute rikkalikus ning süsteemi suur kasutajaskond ja populaarsus. 2021. aasta märtsi seisuga oli see andmebaasisüsteemide populaarsuse pingereas neljandal kohal [13].

Näiterakendused realiseeritakse C# programmeerimiskeeles. C# valiti, sest see on populaarne veebirakenduste programmeerimiskeel, mida käsitleti ülikooli õpingute ajal ja millele on üliõpilaste jaoks tasuta arenduskeskkonnad. 2021. aasta märtsi seisuga oli see programmeerimiskeelte populaarsuse pingereas viiendal kohal [15]. Arvestades, et uue programmeerimiskeele õppimine on aeganõudev, siis tundus lõputöö autorile kõige sobilikum valida keel, mida ta juba tunneb ning millega seotud teadmiste ja oskuste täiendamine pole seetõttu keeruline.

Kuna serveripoolse keelena on valitud C# ja lõputöö tulemusena valmivad näiterakendused on just veebirakendused, siis valitakse arvutitarkvara raamistikuks .NET 5. .NET raamistik pakub võimalust ehitada veebirakendust ASP.NET Core raamistiku abil.

1.4 Ülevaade tööst

Käesolev lõputöö koosneb kaheksast peatükist. Töö teises peatükis antakse ülevaade SQL-andmebaasidel põhinevate andmebaasirakenduste võimalustest salvestada andmebaasirakenduste kaudu hallatavaid faile. Töö kolmas peatükk, „Mustrite struktuur ja realiseerimine“, kirjeldab üldisemalt mustreid ja esitab antud töös kasutatava mustri struktuuri. Peatükis „Mustrite kataloog“ esitatakse töö üheks põhitulemuseks olevad disainimustrid. Töö viiendas peatükis, „Näiterakendused“, esitatakse realiseeritavate rakenduste analüüs, disainimustrite põhjal realiseeritud andmebaaside disain ning kirjeldatakse rakenduse ülesehitust, kasutajaliidest, testandmete genereerimist ja rakenduste pilvekeskkonda üles seadmist. Peatükis „Disainilahenduste võrdlemine“ pannakse paika andmebaasi disainilahenduste võrdlemise strateegia ja kirjeldatakse mõõtmisi. Seitsmendas peatükis antakse ülevaade tehtud mõõtmiste tulemustest. Kaheksandas peatükis esitatakse mõõtmistulemuste põhjal tehtud analüüs ja järeldused ning tuuakse välja töö puudused. Töö käigus realiseeritavate rakenduste lähtekood on saadaval https://github.com/erlemaido/File-Storage-Design-Patterns. Kuni kaitsmise perioodi lõpuni hoiab autor ühte näidisrakendust üleval Amazoni pilveserveris. Täpsem info on GitHub’i lehel.

2 Andmebaasirakenduste kaudu hallatavad failid

Ka andmebaasisüsteemides hoitavad andmed on tegelikult kettal salvestatud failides. Andmebaasisüsteemid pakuvad andmete kasutajatele andmetega töötamiseks liidese, mille kaudu andmeid kasutada, ilma, et kasutaja peaks midagi teadma nende andmete füüsilisest salvestamisest (sh failidest). Antud töös ei peeta silmas neid faile, vaid failide kujul esitatud andmeid, mida oleks vaja koos olemite ning nende seoste kohta hoitavate muude andmetega mingil viisil hallata.

Tänapäeval andebaasirakendust arendades võib kindel olla, et ühel hetkel on vaja realiseerida piltide (või muud tüüpi failide) üleslaadimise ja kasutajatele esitamise funktsionaalsused. Selleks peab faile kusagil hoiustama. Kuigi see võib tunduda esialgu lihtsa ülesandena, siis failide üleslaadimise ja esitamise funktsionaalsuste halvasti realiseerimisel muutub failide haldamine kiiresti üheks rakenduse valukohaks [16]. Käesolevas peatükis antakse ülevaade failide kujul esitatud andmete salvestamise võimalustest. Tekstis viidatakse kohati pildifailidele, kuid need lahendused sobivad ka muud tüüpi failide jaoks kui just tekstis pole öeldud teisiti. Kuna disainilahenduste põhjal realiseeritavates näiterakendustes realiseeritakse just pildifailide salvestamine, siis kirjeldatakse antud peatükis ka põgusalt andmebaasis piltide hoidmist käsitlevat standardit ja piltidega seonduvaid andmebaasisüsteemi funktsionaalsuseid.

2.1 Failide salvestamine

Aja möödudes kasvavad rakenduste andmeobjektid aina suuremaks. Rakenduste loojad peavad otsustama, kuhu ja kuidas failide kujule esitatud andmeid salvestada. 2006. aastal Microsofti poolt avaldatud uurimistöös väitsid autorid, et rakenduste loojad toetuvad tihtipeale otsuse tegemisel oma olemasolevatele oskustele, mis ei pruugi aga olla rakenduse jaoks parim lahendus [9].

Leidub hulgalisel arutelusid, kas faile peaks hoiustama failisüsteemis või andmebaasis [17], [18], [19], [20], [21]. Arendajatel on selle teema osas erinevad arvamused. Microsofti uurimistöö autorid otsisid oma töös vastust küsimusele, millal on faili tasuvam

salvestada failisüsteemi ja millal andmebaasi. Nad võrdlesid oma töös NTFS (*New Technology File System*)[1] failisüsteemi ja SQL Server 2005 andmebaasisüsteemi. Uurimus näitas, et suurobjekte (faile), mis on väiksemad kui 256 KB, käsitleb tõhusamalt SQL Server. Samal ajal on NTFS tõhusam suurobjektidega, mis on suuremad kui 1 MB. Nende kahe suuruse vahemikus sõltub tõhusus kirjutamise intensiivsusest ja faili keskmisest säilitamise ajast. Muidugi varieerub tasuvuspunkt ka erinevate andmebaaside, failisüsteemide ja töömahtude sees [9].

B. Karwin ütleb oma raamatus „SQL Antipatterns“, et enne, kui planeerida, kuidas peaks rakendus failidega tegelema, tuleks mõelda, kas erinevate antimustrite poolt esile tõstetud halvad tehnilised lahendused mõjutaksid kavandatavat rakendust. Mitte iga rakendus ei vaja tehingutöötlust või SQL-i juurdepääsu pildifailidele. Võib ka juhtuda, et näiteks andmebaasi võrgust välja võtmine varundamise ajaks on sobiv kompromiss [8]. Antimustrite olulisuse leidmiseks tuleks otsida vastuseid järgmistele küsimustele.

- Mis on andmete varundamise ja taastamise metoodika? Kuidas varundamist kontrollitakse?
- Kas failid kogunevad pikema ajaperioodi jooksul või need eemaldatakse süsteemist, kui neid enam vaja ei lähe? Milline on nende eemaldamise protseduur? Kas eemaldamine on automaatne või viiakse läbi käsitsi?
- Millistel kasutajatel on luba faile vaadata? Kuidas on juurdepääs tagatud? Mida näevad kasutajad, kes pärivad faile, mille vaatamiseks ei ole neil õigusi?
- Kas faili andmete muudatust saab tühistada? Kui saab, siis kas rakendus peaks taastama algse faili?

Käesolevas töös antud küsimustele vastuseid ei otsita, sest eesmärk on realiseerida ja katsetada erinevate disainimustritega. Tänapäeval on kolm kõige populaarsemat lahendust failide salvestamiseks andmebaasisüsteemi abil loodud andmebaas, failisüsteem ja pilve platvorm [16]. Peale selle võimaldavad ka nimetatud lahendused omakorda erinevaid võimalusi failide hoidmiseks. Seetõttu tuleks probleemile õigesti

[1] Microsofti poolt välja töötatud Microsoft Windows NT ja selle järeltulijate Windows 2000, Windows XP ja Windows Server 2003 standartne failisüsteem [2].

lähenemiseks jaotada see mitmeks väiksemaks tükiks. Failide salvestamise puhul jaguneksid need:

- valida, kus faile hoida,
- faili hoidmise realiseerimine.

Erinevatel andmebaasisüsteemidel on erinevad võimalused failide salvestamiseks. Näiteks väga populaarne andmebaasisüsteem Microsoft SQL Server (2021. aasta aprilli seisuga andmebaasisüsteemide populaarsuse pingereas kolmandal kohal [13]) võimaldab faile salvestada *Filestream*-i (failivoo) kaudu. *Filestream* võimaldab SQL Serveri põhistel rakendustel salvestada failisüsteemi näiteks dokumente ja pilte. Rakendused saavad kasutada voogedastuse (*streaming*) rakendusliideseid ja failisüsteemi jõudlust, tagades samal ajal struktureerimata ja neile vastavate struktureeritud andmetega tehtavate tehingute e transaktsioonide terviklikkuse [22].

Ka töölaua andmebaasisüsteemil Microsoft Access on oma andmetüüp failide salvestamiseks. Selle andmetüübi nimi on *attachment* ja antud tüübiga veerus saab hoida erinevas vormingus faile. Maksimaalselt saab Accessi andmebaasi lisada 2 GB andmeid (sh faile) ning üksik manustatud fail ei tohi olla suurem kui 256 MB [23]. *Attachment* tüüpi veeru väljas võib olla salvestatud rohkem kui üks fail.

Oracle pakub failide salvestamist *SecureFile* abil, mis pakub Oracle enda sõnul parimat nii välisfailide kui ka andmebaasi suurobjektide maailmast. *SecureFile* võimaldab krüpteerimist, tihendamist, deduplikatsiooni ja palju muud [24].

Kuna käesoleva lõputöö tulemusena esitatavate disainimustrite põhjal realiseeritakse andmebaasid PostgreSQL-is, siis keskendub autor järgnevalt PostgreSQL-i toetavatele lahendustele.

2.1.1 PostgreSQL andmebaas

PostgreSQL pakub kahte erinevat viisi binaarandmete salvestamiseks. Binaarandmeid saab ühe variandina tabelisse salvestada kasutades *bytea* tüüpi veerge. Teine variant on kasutada suurobjekti (*Large Object*, BLOB) võimalust, mille puhul salvestatakse spetsiaalses vormingus binaarandmed andmebaasisüsteemi poolt eraldi tabelisse ja viidatakse selles tabelis olevatele ridadele kasutades objekti identifikaatoreid (*object*

identifier, OID). Sobiva meetodi kindlakstegemiseks peab mõistma kummagi meetodi piiranguid [25].

2.1.1.1 *Bytea* tüüp

Bytea andmetüüp võimaldab salvestada binaarsõnesid (*binary string*). Binaarsõne on oktettide[1] (või baitide) jada. Iga oktett esitab mingit digitaalset teavet ja see koosneb kaheksast bitist [26]. Tekstilised sõned võimaldavad salvestada ainult andmebaasis kasutatava märgistiku (*character set*) kodeeringule vastavaid baitide jadasid. Neid baite kutsutakse kuvatavateks (*printable*) baitideks ning tavaliselt omavad nad kümnendsüsteemis väärtust vahemikus 32–126. Binaarsõned võimaldavad aga salvestada ka baite väljaspool nimetatud vahemiku. Neid baite kutsutakse mittekuvatavateks (*non-printable*) baitideks.

PostgreSQL *bytea* tüüp toetab kahte binaarandmete esitamise formaati: *hex* ja *escape* [27]. PostgreSQL aktsepteerib andmete salvestamisel *bytea* väärtustena nii *hex* kui *escape* formaadis andmeid. *Escape* on PostgreSQLi ajalooline formaat, *hex* formaadi tugi lisati versiooni 9.0 (2021. aasta seisuga on viimane versioon 13). PostgreSQL-i poolt kasutatav *bytea* väärtuste kasutajale esitamise formaat on määratud juhtparameetri *bytea_output* väärtusega. Selle väärtust konkreetses süsteemis saab vaadata käsuga `SHOW bytea_output;` Vaikimisi kasutatakse *hex* formaati.

Hex formaadi korral esitatakse iga bait kahe kuueteistkümnendsüsteemis numbrina. Binaarstringi alguses on märgijada \x. PostgreSQL dokumentatsiooni kohaselt toetavad sellises formaadis andmete kasutamist paljud välised rakendused, väärtuse sellesse formaati teisendamine on kiirem kui *escape* formaati ja seega on see formaat eelistatud.

Escape formaadi korral esitatakse binaarsõne ASCII märkide jadana. Need baidid, mida ei saa ASCII märkidega esitada (mittekuvatavad baidid), teisendatakse spetsiaalseteks paojadadeks, mis on käsuna toimiv erimärgijada. Selleks teisendatakse bait kolmekohaliseks kaheksandarvuks (oktaalarvuks), mille ette lisatakse längkriips (*backslash*) [27]. Sisuliselt tähendab see, et baitide jada esitatakse märkide jadana. PostgreSQL dokumentatsioon märgib, et see esitus hägustab piiri binaasõnede ja

[1] Oktett on kaheksast bitist koosnev jada, mida nimetatakse ka baidiks. Nimetust "oktett" kasutatakse enamasti võrgunduses ja nimetust "bait" arvutiasjanduses [2].

tekstisõnede vahel, paomärkide mehhanism on keeukas ja uute süsteemide puhul võiks sellist esitust vältida.

Binaarsõnedega tehtavad operatsioonid töötlevad tegelikke baite, samas kui tekstiliste sõnadega töötlemine sõltub lokaalsetest seadetest. Lühidalt öeldes sobivad binaarsõned andmete salvestamiseks, mida programmeerijad peavad „toorbaitideks“ („*raw bytes*“) ja tekstisõned teksti salvestamiseks [28].

PostgreSQL kasutab TOAST (*The Oversized-Attribute Storage Technique*) andmete salvestamise tehnikat. Uue tabelirea lisamisel, mis ületab 2 KB, pakitakse muutuva pikkusega väärtuseid sisaldavatesse tüüpidesse (sh *bytea*) kuuluvad väärtused kokku ja/või jagatakse mitmeks väiksemaks osaks, mis salvestatakse eraldi füüsiliste ridadena eraldiseisvas tabelis (TOAST tabelis). TOAST tabel on süsteemne tabel, mille loomine ei ole andmebaasi arendaja ülesanne. TOAST-imist toetavasse tüüpi kuuluv väärtus võib olla kuni 1 GB suurune [29].

2.1.1.2 Suurobjekt

PostgreSQL-il on suurobjekti salvestamise võimalus, mis võimaldab voogedastuse stiilis juurdepääsu kasutaja andmetele, mis on salvestatud spetsiaalsesse suurobjektide struktuuri, st neile viitavatest tabeli ridadest eraldi. Voogedastus on tehnika andmete edastamiseks, kus andmete kasutamiseks ei pea ootama, et terve fail oleks alla laetud. Andmed kogutakse puhvermällu ja edastatakse jooksvalt hulkade (portsude) kaupa [2]. Mahukate andmeväärtuste puhul on voogedastus oluline, sest aitab vältida võrguviivitusest tekkivaid häireid [30].

Tabelisse suurobjekti salvestamisel määrab süsteem suurobjektile objekti identifikaatori, jagab selle 2 KB suurusteks tükkideks ja salvestab need süsteemsesse tabelisse nimega *pg_largeobject*. Tabelile *pg_largeobject* loodud B-puu indeks tagab õige tüki kiire leidmise [28]. Iga suurobjekti kohta on ka rida süsteemses tabelis *pg_largeobject_metadata*. Suurobjekte saab luua, muuta ja kustutada kasutades lugemise/kirjutamise rakendusliidest (*Application Programming Interface,* API) [31].

Suurobjektide kasutamisel põhinev lähenemine sobib *bytea* tüübi kasutamisest paremini väga suurte binaarsete väärtuste salvestamiseks, aga ka sellel on omad piirangud. Tabelis 1 on kujutatud binaarsõne ja suurobjekti lähenemiste võrdlus [28]. Kui kustutada tabelis rida, mis sisaldab suurobjekti viidet, siis suurobjekti ise ei kustutata. Suurobjekti

kustutamine on eraldi toiming. Suurobjekti meetodil on ka mõningad turbeprobleemid, sest igaüks, kes on andmebaasiga ühendatud, saab vaadata ja/või muuta suvalist suurobjekti, isegi kui tal puuduvad õigused suurobjekti viidet sisaldava rea vaatamiseks või värskendamiseks [25].

Tabel 1. *Bytea* ja suurobjekti võrdlus.

Omadus	***Bytea***	**Suurobjekt**
Objekti (väärtuse) suurim lubatud maht	1 GB	4 TB
Andmepöördus (*data access*)	Tervikuna	Voogedastus
Salvestamine	Süsteemses TOAST tabelis	*pg_largeobject* süsteemses tabelis
Andmekäitlus	Kasutades SQL-i ja paojadasid	Ainult erifunktsioonide abil tehinguplokis

Allikas [32] toob näite, kuidas ühes andmebaasis, kus oli 100 miljonit suurobjekti olid 96% nendest orvud, millele üheski reas ei viidatud. Suurobjektide kustutamise probleemi lahendamiseks saab kasutada laiendust *lo* [33], mis võimaldab lihtsa vaevaga siduda suurobjektidele viitavate kasutajate tabelitega trigereid, mis rea kustutamisel või andmete muutmisel suurobjekti viidet sisaldavas kasutaja tabeli veerus kustutavad ka suurobjekti. Eelduseks on, et igale suurobjektile viidatakse vaid ühest tabeli reast. Alternatiiviks trigerite kasutusele oleks käivitada perioodiliselt orvuks osutuvate suurobjektide otsimine ning kustutamine [32]. See sobiks hästi andmebaasis, kus andmeid muudetakse väga sageli, sest võimaldaks trigerite pidevast käivitumisest tekkiva koormuse asendada puhastusoperatsiooniga andmebaasi väiksema kasutusega ajal.

2.1.2 Failisüsteem

Failisüsteem on operatsioonisüsteemi osa, mille eesmärk on paigutada loogilisi faile füüsilisele salvestusseadmele (enamasti kettale), et neid oleks kerge leida ja kasutada. Failisüsteem kirjeldab, kuidas failid paiknevad füüsilisel kettal ja milline on nende struktuur. Failisüsteem seab ka kitsendusi failide nimedele ja nende suurustele [34]. Failisüsteem võimaldab failide organiseerimist, salvestamist, kustutamist, nimetamist ja pakkimist. Lisaks võimaldab see määrata failidele juurdepääsuõigusi. Failisüsteemi peamised objektid on failid ja kaustad. Kaustad on üldiselt organiseeritud puustruktuuri järgi [35].

Käesolevas töös keskendutakse failisüsteemiga seotud disainilahenduse leidmisel XFS failisüsteemile. XFS on ettevõtte Silicon Graphics poolt IRIX operatsioonisüsteemi jaoks loodud suure jõudlusega päevikuga[1] failisüsteem, mis porditi Linuxi tuuma ja on toetatud enamikes Linuxi distributsioonides [6].

Ühtne ressursiidentifikaator (*Uniform Resource Identifier,* URI) on sõne, mida kasutatakse infoallika üheseks määramiseks veebis [35]. URI võib olla internetiaadress (*Uniform Resource Locator,* URL). Internetiaadress identifitseerib ressurssi nii asukoha kui ka peamise juurdepääsu mehhanismi abil (nt http, https, ftp). PostgreSQL-is saab kasutada *uri* andmetüüpi, mille kasutuselevõtuks tuleb installeerida laiendus pguri [36]. Sellise tüübi kasutamise eelis URI esitamise ees tekstilise väärtusena on URI süntaksi kontroll, alamosade eraldamise võimalus spetsiaalsete funktsioonide abil ja kasutajasõbralikum tulemuse sorteerimine URIde alusel.

2.1.3 Pilveteenus

Pilvandmetöötlus (*cloud computing*) võimaldab salvestada andmeid suures mahus, pakkudes samal ajal skaleeritavust ning paindlikust. Pilvandmetöötluse teenuse pakkujad vastutavad andmete haldamiseks vajaliku taristu loomise ja ülalhoiu eest. Teenuse kasutaja annab oma andmed teenusepakkuja hoole alla ja see võib olla teenuse kasutajale majanduslikult kasulik [37]. Pilvandmetöötluse teenust, mis võimaldab interneti kaudu pilve kaugsalvestusserverisse andmeid salvestada, kätte saada ja redigeerida, nimetatakse pilve ladustamise teenuseks (*cloud storage service*) [38].

Salvestus kui teenus on ligipääsetav interneti kaudu ning omab mitmeid eeliseid võrreldes traditsiooniliste hoidlatega, mis on andmete valdaja enese kontrolli all. Üks neist eelistest on suur skaleeritav salvestusmaht, mis on kättesaadav kõigile vaatamata asukohast [37]. Vajadusel saab pilve serverarvuteid juurde lisada või sealt ära võtta, seega võimaldab pilveandmetöötluse süsteem ressursside efektiivset ärakasutamist [39]. Muidugi saab skaleerida ka andmete valdaja enda hoole all olevaid süsteeme, kuid siis on süsteemi ehitamine ja ülalhoid tema vastutusel, mitte teenusepakkuja vastutusel. Suurel

[1] Failisüsteem, mis salvestab enne tegelike muudatuste tegemist failisüsteemis eelseisvad muudatused spetsiaalsesse logifaili ehk päevikusse [2].

teenusepakkujal on võimalik seda kõike teha kiiremini ja odavamalt kui üksikul väiketarbijal.

Paljud kasutajad üle maailma kasutavad pilvandmetöötluse teenuseid. Pilveteenuse pakkujad nagu Amazon, Microsoft ja Google pakuvad erinevat tüüpi teenuseid. Kasutajate nõudlus pilvepõhiste salvestusteenuste järele on kiiresti kasvanud, mis on põhjustanud salvestusteenuse pakkujate vahel suure konkurentsi [38]. Kaks kõige tuntumat konkurenti selles valdkonnas on Amazon Web Services (AWS) ja Microsoft Azure. Mõlemad pakuvad objektide salvestusteenust, mis on hõlpsasti ja ulatuslikult skaleeritavad. Amazonil on selleks teenuseks Amazon Simple Storage Service (Amazon S3) ja Microsoftil Azure Blob Storage [37].

AWS on olnud algusest peale ja on jätkuvalt populaarseim pilveteenuse pakkuja – 2020. aasta seisuga on ettevõttel 32% suurune turuosa [40]. Sellest tulenevalt kasutatakse käesolevas lõputöös näitesüsteemi loomiseks pilveteenust Amazon S3.

2.1.3.1 Amazon S3

Amazon on töö kirjutamise ajaks võitnud paljude klientide usalduse ja on üks juhtivatest pilvandmetöötluse pakkujatest [37]. Amazoni lihtsa salvestusteenuse – Amazon SSS e Amazon S3 – eesmärk on muuta rakendustele (ja neid loovatele arendajatele) lihtsamaks andmete, sh suure hulga, katkematu voona saabuvate ja erinevates formaatides andmete, salvestamine ning lugemine [41].

Amazon S3 on lihtne veebiteenuste liides, mida saab kasutada andmete hankimiseks ja talletamiseks. See annab juurdepääsu samale skaleeritavale, usaldusväärsele, kiirele ja odavale andmesalvestuse taristule, mida Amazon kasutab ka oma ülemaailmse veebisaitide võrgu käitamiseks. Teenuse eesmärk on maksimeerida skaleeritavusega seotud eeliseid ja pakkuda neid eeliseid ka kasutajatele, kelleks võivad olla kasutajad üle maailma [41].

Amazon S3-s salvestatakse objektid korvidesse. Korv (*bucket*) on salvestatud objektide konteiner. Iga objekt võib sisaldada kuni 5 TB andmeid. Versioonimise abil saab samas konteineris hoida objekti mitut versiooni. Iga objekt salvestatakse ja leitakse kasutades unikaalset võtit. Korvidel on järgmised eesmärgid.

- Need korraldavad kõige kõrgemal tasemel Amazon S3 nimeruumi.

- Need identifitseerivad konto, mis vastutab salvestus- ja andmeedastustasude eest.
- Neil on oma roll juurdepääsukontrollis.
- Neid saab kasutada süsteemi kasutamise aruandluse koostamisel koondandmete leidmiseks [41].

Objektid on Amazon S3-s andmete salvestamise põhiüksused. Objektid koosnevad objekti andmetest ja metaandmetest. Andmete osa on Amazon S3-le läbipaistmatu, st nende sisu põhjal ei saa teha päringuid, kuid objekt võib olla mistahes baitide hulk (sh ka näiteks pildifailid). Metaandmed (andmed andmete kohta) on objekti kirjeldavate võtme-väärtuse paaride kogum. Metaandmete hulk objektis on kuni 2 KB. Metaandmete hulka kuuluvad mõned vaikemetaandmed nagu näiteks viimati muudetud kuupäev ja standardsed HTTP-metaandmed nagu näiteks sisutüüp. Objekti salvestamisel saab määrata ka kohandatud metaandmeid [41].

Igal korvis oleval objektil on täpselt üks võti. Võti on korvis oleva objekti kordumatu identifikaator. Korvi, võtme ja versiooni ID kombinatsioon tuvastab iga objekti. Iga Amazon S3-e objekt on unikaalselt adresseeritav veebiteenuse lõppsõlme (*endpoint*), korvi nime, võtme ja valikuliselt versiooni kombinatsiooni abil [41].

Amazon S3-s on objektide uuendamised võtmepõhised. Amazon S3 pakub objektide uuendamisel ja kustutamisel tugevat *read-after-write* andmete terviklikkust. Uuendused on atomaarsed ehk muudatused viiakse lõpule tervikuna või ei tehta neid üldse. Seega peaksid uuenduse järel andmehoidlas puuduma osalised või rikutud andmed. Siiski ei toeta Amazon S3 samaaegsete kirjutajate jaoks objekti lukustamist. Kui samale võtmele tehakse korraga kaks PUT päringut[1], siis võidab viimase ajatempliga päring, st hilisem objekti kirjutamine kirjutab varem kirjutatud objekti üle. Selle probleemi vältimiseks tuleks S3 teenust kasutavasse rakendusse ehitada objekti lukustamise mehhanism [41]. Käesoleva töö pilvelahenduse disaini näidisrakenduses ei ole objekti lukustamise mehhanismi realiseeritud.

Igal Amazon S3 objektil on sellega seotud salvestusklass. Amazon S3 pakub salvestatud objektide jaoks mitmesuguseid salvestusklasse. Klass valitakse sõltuvalt kasutamisstsenaariumist ja jõudlusnõuetest, näiteks salvestusklassid sageli

[1] HTTP meetod, kus asendatakse olemasolev element või luuakse selle puudumisel uus element.

juurdepääsetavatele objektidele, salvestusklassid muutuvate või tundmatute juurdepääsumustritega andmete kasutamise automaatseks optimiseerimiseks, salvestusklassid objektide arhiveerimiseks [42].

Amazon Web Services toetab Amazon S3 jaoks kliendipoolset krüpteerimist ja serveripoolset krüpteerimist (SSE-S3), et kaitsta andmeid volitamata juurdepääsu eest tagades samas, et andmeid ilma loata ei muudeta, ja et andmed on võimalikult kogu aeg kättesaadavad e käideldavus on hea [37].

Amazon S3 hinnakujundus on loodud sellisena, et rakenduse salvestusnõudeid ei pea pikalt ette planeerima. Makstakse ainult selle eest, mida tegelikkuses kasutatakse [41].

2.2 SQL/MM

Andmebaasis piltide hoidmise võimaluste kirjeldamiseks on loodud eraldi standard. Standardi nimi on SQL/MM. Tegemist on mitmeosalise standardiga, mis koosneb mitmetest üksteisest suhteliselt sõltumatutest osadest. „MM“ tähistab multimeediumit (*multimedia*). Standardi viies osa on ISO/IEC 13249-5 SQL/MM *Part5: Still Image*, mis on rahvusvaheline standard staatiliste piltide hoiustamiseks, hankimiseks ja otsimiseks kasutades SQL-i. Standardi eesmärk ei ole piltide ja nende vormingute standardiseerimine, vaid kirjeldada standardiseeritud liidest piltide haldamiseks andmebaasisüsteemides koos muude andmetega [43]. PostgreSQL-is (ver 13 seisuga) pole SQL/MM standard realiseeritud.

SQL/MM *Still Image* pakub lahendusi piltide hoiustamisega ja töötlemisega seotud probleemidele. SQL/MM *Still Image* käsitleb ainult staatiliste piltide andmeid. Standardi järgi kasutatakse piltide jaoks SQL:1999 struktureeritud tüüpi *SI_StillImage*. [44] Staatiline pilt on kahemõõtmeliste andmemassiivide kogum ja massiivides hoitavad andmed esitavad andmeid piltide pikslite kohta. Üks massiiv esindab pildi ühte komponenti nagu näiteks põhivärv punane RGB värviruumis [43]. *SI_StillImage* tüüpi väärtus sisaldab ka teavet pildi kohta nagu näiteks formaat, mõõtmed (kõrgus ja laius pikslites), värviruum ja palju muud [44].

SI_StillImage tüüpi väärtustele rakendatavad meetodid hõlmavad pildi skaleerimist, kärpimist, pööramist ja pisipildi loomist. Piltide erinevate omaduste kirjeldamiseks kasutatakse veel üht gruppi andmetüüpe. Näiteks üks neist tüüpidest on *SI_AverageColor*,

mille väärtust kasutatakse pildi „keskmise“ värvi tähistamiseks. *SI_PositionalColor* tüüpi väärtus tähistab konkreetsete värvide asukohta pildil toetades päringuid nagu „kuna päikeseloojangu ajal on merel punane ja oranž värv tumesininse värvi kohal, siis leia mulle pildid nende värviomadustega“. Kombineerides pildi mitu tunnust, on võimalik kirjutada päringuid, mis hangivad väga suurest pildibaasist palju väiksema piltide kogu, millest saab kiiresti välja valida vajaliku pildi [44].

Mitmed SQL/MM standardi osad ei ole andmebaasisüsteemides laialdlaselt toetatud ning ka *Still Image* toetus kasvab väga aeglaselt [44].

2.3 SQL/MDA

2019. aastal lisandus SQL andmebaasikeele standardisse osa 15 - *Multidimensional Arrays* (SQL/MDA). See kirjeldab mitmemõõtmeliste massiivide kasutamist SQLis. See standardi täiendus peaks aitama kaasa SQL-i senisest suuremale kasutamisele teadusandmete töötlemise ja inseneneeria valdkonnas. SQL/MDA kirjeldab, kuidas mitmemõõtmelisi massiive kui väärtuseid andmebaasis esitada ja neid andmebaasikeele abil töödelda [45]. Näiteks halltoonides rasterpilte (sh satelliidipilte) saab esitada kahemõõtmeliste massiividena, kus iga element näitab vastava piksli värvuse intensiivsust. Värvilisi rasterpilte saab esitada massiivide massiivina, kus iga piksli kohta on massiiv, mis näitab punase, rohelise ja sinise põhivärvi taset selles pikslis. Mitmemõõtmeliste massiividega saab esitada ka näiteks piltide ajaseeriaid. Satelliidipilte mingist piirkonnast tehakse kindlate ajaintervallide tagant. SQL/MDA standardit järgivat süsteemi võiks näiteks kasutada satelliidipiltide arhiivi loomiseks [46].

2.4 Piltidega seotud andmebaasisüsteemi funktsionaalsused

Nagu eespool mainitud, töödeldakse või esitatakse tänapäeval pilte väga paljudes rakendustes. Andmete sh piltide ja muu meedia hulk aina kasvab ning rakendused peavad muuhulgas toime tulema üha suureneva piltide arvuga. Piltide ja videote massiline üleslaadimine toob omadega kaasa korduste e duplikaatide probleemi. Mitmed pilditöötlusega seotud funktsionaalsused nagu näiteks pildiotsing, video ja pildi deduplikatsioon ning pildi ilustamine seisavad silmitsi ühise väljakutsega, mis seisneb tohutu hulga serverisse salvestatud üksteist kordavate piltide ja videote haldamises [47].

Pildiotsing on teine enam levinud otsinguviis peale tekstiotsingut. Piltide otsingumootorid nagu näiteks Alibaba Cloud Image Search võimaldavad üles laadida konkreetse pildi ja otsida sellega sarnaseid pilte. Piltide ja videote korduste tuvastamine on oluline, et vältida suures mahus korduste salvestamist serverisse. Teisest küljest on oluline ka mittesobiva sisuga piltide ja videote tuvastamine [47].

PostgreSQL-il on sarnaste piltide otsingu funktsionaalsuseks mitmeid lahendusi, näiteks ImgSmlr laiendus [48] või PostgreSQL Image Search Plug-In (pistikprogramm) [47]. Mõlemad neist kasutavad Haar'i lainikute teisendamist (*Haar wavelet transform)*. Lainikuid kasutatakse erinevate signaalide analüüsimiseks. Haar'i lainik on ruudukujuliste signaalide järgnevus, mis kokku pannes moodustavad laine tüüpi võnkumise. Selle abil on lihtsasti võimalik tuvastada väga kiireid muutusi signaalis. Lainikute abil tehtavaid matemaatilisi teisendusi nimetatakse lainikute teisendusteks [49]. Lainikute teisendused pakuvadki piltide analüüsimisel sarnaste piltide tuvastamise võimaluse.

Smlar on veel üks PostgreSQL laiendus, mis pakub võimalust sarnaste piltide ja videote otsinguks. Laiendus pakub funktsionaalsust, mis arvutab kahe massiivi sarnasuse ning tagastab väärtuse ujukomaarvuna vahemikus 0 kuni 1, kus 0 tähendab, et massiivi objektid ei ole üldse sarnased ja 1 tähendab objektide samasust [50].

3 Mustrite struktuur ja realiseerimine

Käesoleva töö ühe tulemusena leitakse failide salvestamist toetavad SQL-andmebaasi disainilahendused ja pannakse need kirja mustrite formaadis. Mustri olemuse paremini mõistmiseks kirjeldatakse kõigepealt mustri mõistet ja kasutatavat mustri struktuuri.

3.1 Mustri mõiste

Muster on üldises mõistes seaduspärasus, mille elemendid korduvad ennustataval viisil. Kunstivaldkonnas või tootearenduses mõistetakse mustri all korduvate joonte või kujundite kombinatsiooni nagu näiteks heegelmuster ja rehvimuster, kuid üleüldiselt on muster Eesti keele seletava sõnaraamatu kohaselt eeskuju, mall, näidis, millegi läbiv ühine joon [51]. Seega põhinevad mustrid kordustel ning mustri üheks kõige olulisemaks omaduseks on taaskasutusvõimalus.

Christopher Alexander, Ameerika Ühendriikide arhitekt, oli üks esimesi, kes pakkus välja idee kasutada ehitiste ja linnade disainimisel mustrikeelt. Tema ideed on juurdunud nüüd ka objektorienteeritud programmeerimise paradigmasse. Tarkvaraarenduses nimetatakse disainimustriks taaskasutatavat lahendust korduvalt esinevale disaini probleemile [52]. Disaini all peetakse silmas tehnilise lahenduse kavandamist vastavalt analüüsi käigus kogutud nõuetele, aga ka võimalustele ning piirangutele, mida muuhulgas määravad kasutatavad platvormid ja laiemalt kogu keskkond, mille jaoks lahendust luuakse. Disainimustrid esitavad tuntud probleemilahendusi ja enamasti on disainimustripõhine lahendus elegantsem ja arusaadavam, kui ise leiutatud lahendus. On üpris tõenäoline, et mõne probleemi otsa sattudes on keegi juba varasemalt antud probleemi lahendanud. Parima lahenduse leidmiseks on mõistlik uurida juba olemasolevaid disainimustreid, sest on üsna ebatõenäoline, et iseseisvalt leitakse uus ja parem, veel avastamata lahendus [53].

3.2 Mustri struktuur

Käesoleva töö kontekstis on muster probleemi ja selle lahenduse struktuurne kirjeldamise viis. Mustrite esitamiseks võtab autor aluseks Jõgi [1] ja Höveli [54] bakalaureusetöödes kasutatud mustrite ülesehituse.

Muster kirjeldatakse käesolevas töös järgnevaid struktuurielemente kasutades.

- *Nimi*, mis toob lühidalt ja võimalikult selgelt esile mustri põhimõtte. Mustri eestikeelne nimi esitatakse peatükis „Mustrite kataloog“ alampeatüki pealkirjana ning seetõttu seda mustris rohkem ei korrata.
- *Ingliskeelsed nimed*, mida kasutatakse ingliskeelses kirjanduses.
- *Probleem*, mille lahendust muster välja pakub. Käesolevas töös on probleem kõikidel disainimustritel sama: kuidas hoida infosüsteemis failide kujul esitatud andmeid. Seetõttu probleemi peatükis „Mustrite kataloog“ uuesti ei esitata.
- *Taust*, mis kirjeldab olukorda, kus lahendamist vajav probleem võib esile kerkida. Käesolevas töös on taust kõikidel disainimustritel sama: SQL-andmebaasi projekteerimisel tuleb otsustada, missugusel viisil süsteemis faile salvestada. Seetõttu tausta peatükis „Mustrite kataloog“ uuesti ei esitata.
- *Jõud*, mis võivad mõjutada mustris pakutava lahenduse kasutamist.
- *Lahendus probleemile*. Lahenduse üldine selgitus.
- *Allikad ja autorid*, kus mustri lahendust kasutatakse ja mille põhjal on autor disainimustri sõnastanud.
- *Mustri tugevad ja nõrgad küljed.* Eeliste ja puuduste väljatoomisel tuginetakse nii eespool nimetatud allikatele kui ka autori arusaamadele antud disainimustri kohta.
- *Variatsioonid antud mustrile.* (Võib ka puududa.)
- *Näide*. Näites esitatakse UML klassidiagrammi põhjal loodud andmebaasi füüsilise disaini diagramm PostgreSQL andmebaasi kohta, milles hoitakse andmeid toodete kohta, sh neid illustreerivaid pilte. Andmebaasis on tabelid *toode* (*product*) ja *tootepilt* (*product picture*). Igal toote kohta on vaja salvestada null või rohkem pilti. Piltidel on toote kirjelduses esitamiseks määratud järjekorranumber ja samal tootel ei tohi olla mitu sama järjekorranumbriga pilti.

Käesoleva töö mustrite kataloog on kahetasemeline. Põhimuster esitab üldlahendust, alammustrid (variatsioonide all) kirjeldavad PostgreSQL-spetsiifilisi lahendusi.

4 Mustrite kataloog

Järgnevad mustrid on disainilahendused failide hoidmiseks SQL-andmebaaside põhiste andmebaasirakenduste jaoks.

4.1 Failid kohalikus andmebaasis

See muster on järgnevate alammustrite üldistus, mis tähendab, et kõik antud mustris nimetatu, kehtib ka selle mustri alammustrite korral.

Ingliskeelsed nimed: *Files in local database*

Jõud:

- Soovitakse olla kindel andmete terviklikkuses, et failid oleks tabelites olevate andmetega sünkroonis.
- Soovitakse hoida faile ühes kohas koos teiste andmetega.

Allikad ja autorid: [17], [18], [20], [21], [29], [55], [56], [57], [58], [59]

Mustri tugevad küljed:

- On tagatud transaktsioonide e tehingute ACID[1] omaduste kehtimine failidega töötamisel, sest failid on koos ülejäänud andmetega ühes kohas ja failid ning muud andmed on omavahel sünkroonis. Ei pea muretsema kadunud või hulkurfailide pärast.

[1] ACID (*Automacity, Consistency, Isolation, Durability*) – transaktsioonide e tehingute põhiomadused:

- Atomaarsus – andmebaasi tehing on üks niinimetatud automaatne üksus, loogiline tervik. Korraga viiakse lõpule kas kõik muudatused või mitte ükski muudatus.
- Terviklikkus – andmete muudatuste tulemusel on andmebaasis terviklikkuse reeglitele vastavad andmed, st tehingu tulemusena ei lähe andmebaas mitteterviklikku seisu.
- Isoleeritus – teised sessioonid (kasutajad) näevad muudetud andmeid alles siis, kui muudatuste tegija on muutmise lõpetanud.
- Püsivus – andmemuudatuste püsiv salvestamine õnnestunud tehingu järgi on tagatud [39].

- Varundamine on lihtne, sest kõik andmed on ühes kohas koos ja eraldi failide varundamise strateegiat ei ole vaja.
- Juhul, kui andmebaasisüsteem toetab multiversioon konkurentsjuhtimist (*multi-version concurrency control*) nagu seda teeb näiteks PostgreSQL, siis faili lugemine ei blokeeri kunagi selle kirjutamist ja kirjutamine ei blokeeri kunagi faili lugemist.
- Pole vaja luua eraldi kataloogide struktuuri failide hoidmiseks, vaid kõik andmed on ühes tabelite struktuuris. Teiste sõnadega, eraldi tabelite struktuuri ja kataloogide struktuuri kavandamise asemel on vaja kavandada ainult tabelid.
- Turvalisust tagada ja juurdepääsukontrolli realiseerida on lihtsam, sest õigusi peab haldama ainult andmebaasi siseselt.
- Versioonihaldus on lihtsam.
- Lihtne arhitektuur.
- Kasutades SQL-i on kerge realiseerida keerulisi päringuid.

Mustri nõrgad küljed:

- Andmebaasi maht tõuseb. Suure andmemahuga andmebaase on keerulisem hallata, kui väiksema mahuga andmebaase. Mahukama andmebaasi hooldus võib tähendada suuremaid kulusid.
- Andmebaasi migreerimine (andmebaasi realiseerimiseks kasutatava andmebaasisüsteemi väljavahetamine) võib saada murekohaks, kui kasutada andmebaasisüsteemi kindlaid omadusi/funktsionaalsuseid, mis teistes andmebaasisüsteemides puuduvad.
- Suurte failide andmebaasis hoidmine ja nende põhjal päringute tegemine põhjustab andmebaasiserveri suurenenud koormuse tõttu olulisi jõudluse probleeme.
- Andmebaasi varundamine võtab rohkem aega.
- Andmebaasi taastamine võtab rohkem aega.
- Andmebaasi mälukasutus on suurem.
- Puudub võimalus failide koheseks muutmiseks (näiteks piltide kärpimine, suuruse muutmine jms).
- Tulevikus on raske üle minna pilvelahenduse peale.

4.1.1 *Bytea* veerg

Ingliskeelsed nimed: *Bytea column*

Jõud:

- Soovitakse kasutada PostgreSQL-i ja võimalikult standardilähedast SQL-i.

Lahendus: Binaarandmed salvestatakse tabelisse baitide jadana kasutades *bytea* tüüpi veerge. Kuigi SQL standard näeb ette tüübi BLOB (*binary large object*), mitte *bytea*, siis funktsioonid ja operaatorid vastavat tüüpi andmetega töötamiseks on enamasti ühesugused.

Mustri tugevad küljed:

- Failidena esitatavate andmetega töötamiseks mõeldud SQL laused võiksid töötada erinevates andmebaasisüsteemides või nõuda nende jaoks vähest ümbertegemist.
- Andmete hoidmine ja kasutamine on sama lihtne kui teiste andmetüüpidega veergude korral.
- Ei ole vaja jälitada objekti identifikaatoreid nagu suurobjektide kasutamise korral.
- Kergesti realiseeritav.

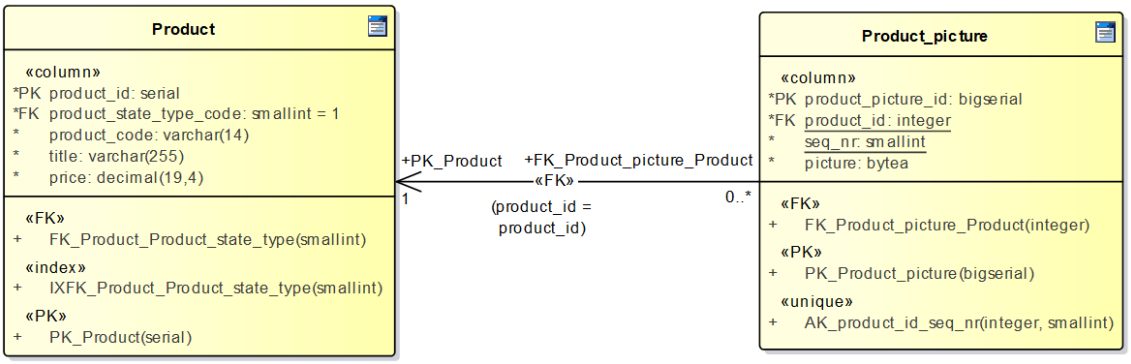
Mustri nõrgad küljed:

- *Bytea* väärtuse lugemisel või kirjutamisel tuleb kogu väärtus korraga ja tervenisti mällu lugeda ning seda seal hoida, sest puudub voogedastuse toetus.
- Serveri mälu nõuded võivad olla väga suured. Bytea väärtus loetakse rakenduses täielikult muutujasse ja siis hakatakse seda töötlema. Ühtlasi suureneb andmebaasisüsteemi ja andmebaasirakenduse muutmälu vajadus.
- Halb jõudlus.
- Maksimaalne väärtuse maht on 1 GB.
- PostgreSQL kasutab TOAST andmete salvestamise tehnikat ja selle puhul ei tohi ühes kasutaja tabelis olla rohkem kui 4 miljard reavälist salvestamist nõudvat (*out-of-line*) väärtust, sest see põhjustaks dubleeritud objekti identifikaatorite (OID) lisamise selle tabeliga seotud süsteemsesse TOAST tabelisse. OID tüüpi kuuluv maksimaalne väärtus on 4 miljardit. Kui kasutaja tabelis on palju väärtuseid, mille puhul peab süsteem rakendama TOAST tehnikat, siis võib

süsteem uue TOAST tabeli rea lisamisel kulutada palju aega järgmise vaba OID otsimisele. See seab piiri kasutaja tabeli ridade arvule.

- Andmeid on vaja kodeerida.

Näide: Joonis 1 esitab *bytea* disaini kasutamise näite.



Joonis 1. *Bytea* disaini kasutamise näide.

Jooniselt on näha, et pildi veerule (*picture*) on andmetüübiks määratud *bytea*. Tehes päringu *bytea* tüüpi veeru põhjal, kodeeritakse baitide jada enne tulemuse väljastamist sobivasse formaati. Joonisel 2 on esitatud näidispäring ja päringu tulemus, kus väljastatakse lisaks muudele tootepildi tabeli andmetele ka *bytea* tüüpi veeru väärtuste 30 kõige vasakpoolsemat sümbolit *hex* formaadis.

```
SELECT product_picture_id, product_id, seq_nr, LEFT(encode(picture, 'hex'),
30) AS picture FROM product_picture LIMIT 5;
```

```
 product_picture_id | product_id | seq_nr |            picture
--------------------+------------+--------+--------------------------------
              99344 |      10001 |      1 | ffd8ffe000104a4649460001010100
              99345 |      10267 |      2 | ffd8ffe000104a4649460001010100
              99346 |      10267 |      1 | ffd8ffe000104a4649460001010100
              99347 |      10266 |      3 | ffd8ffe000104a4649460001010100
              99348 |      10266 |      2 | ffd8ffe000104a4649460001010100
```

Joonis 2. *Bytea* disaini näidispäring ja selle tulemuse näide.

4.1.2 Suurobjekt

Ingliskeelsed nimed: *Large Object*, *Binary Large Object*, LOB, BLOB

Jõud:

- Faile soovitakse hoida koos teiste andmetega andmebaasis, kuid neid päritakse harva ja seetõttu soovitakse hoida muudest andmetest lahus.
- Soovitakse vähendada andmebaasisüsteemi ja andmebaasirakenduse mälukasutust.

Lahendus: Tabelis hoitakse *oid* tüüpi viidet suurobjektile. Binaarandmed salvestatakse andmebaasisüsteemi poolt 2 KB tükkidena eraldi süsteemsesse tabelisse nimega *pg_largeobject*. Suurobjekti salvestamisel määrab süsteem suurobjektile objekti identifikaatori, millega viidatakse selles tabelis olevatele ridadele.

Allikad ja autorid: [60]

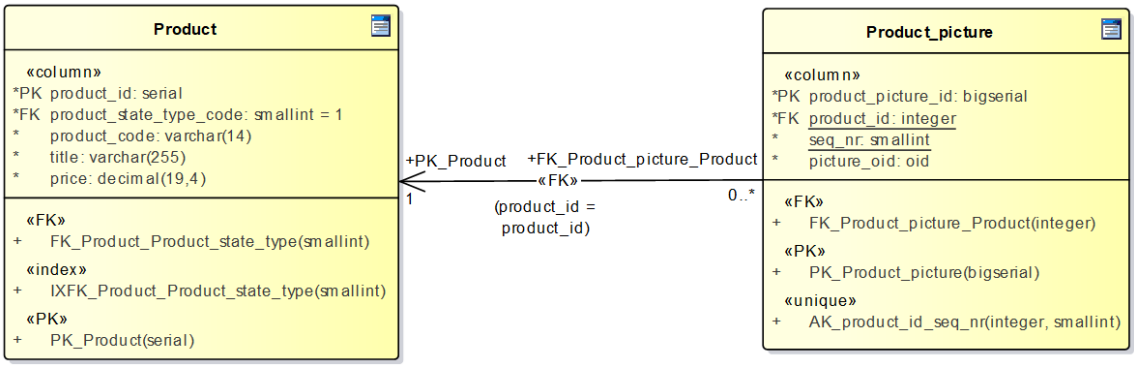
Mustri tugevad küljed:

- Rakendusliides toetab voogedastust, mis vähendab andmebaasisüsteemi ja andmebaasirakenduse muutmälu vajadust.
- Maksimaalne maht on 4 TB väärtuse kohta (alates PostgreSQL 9.3) ja 2 GB enne PostgreSQL 9.3 (mis on ikkagi suurem, kui maksimaalne lubatud *bytea* tüüpi väärtuse suurus).
- Maksimaalselt 4 miljardit suurobjekti andmebaasi kohta.
- Pole vaja kodeerimist ega paojadadeks teisendamisi.

Mustri nõrgad küljed:

- Väga halb jõudlus.
- Suurobjektidele ligipääsemiseks peab kasutama mittestandardset rakendusliidest.
- Andmebaasi migreerimisel (üleminekul ühelt andmebaasisüsteemist teisele) peab andmebaasi ja rakenduse tasemel tegema koodis rohkem muudatusi.
- Vajab eraldi objektide kustutamise haldamist (näiteks trigerite abil), sest tabelist rea kustutamisel, ei kustutata suurobjekti, millele tabeli rida viitab.
- Peab pidama järge objekti identifikaatoritel.

Näide: Joonis 3 esitab suurobjektide disaini kasutamise näite.



Joonis 3. Suurobjekti disaini kasutamise näide.

Jooniselt on näha, et tootepildi tabelisse salvestatakse pildi identifikaator (*picture_oid*), mis on *oid* tüüpi. *Oid* tüüp on ilma märgita 4 baidine täisarv, mille väärtused on vahemikus 0–4 294 967 295. Seetõttu ei ole see piisavalt suur, et pakkuda unikaalsust suurtes andmebaasides [61]. Joonisel 4 on esitatud näidispäring ja päringu tulemus, kus väljastatakse lisaks muudele tootepildi tabeli andmetele ka salvestatud suurobjekti *oid*.

```
SELECT * FROM product_picture LIMIT 5;

 product_picture_id | seq_nr | picture_oid | product_id
--------------------+--------+-------------+------------
             500940 |      1 |     7105491 |      50001
             500941 |      6 |     8170297 |      50288
             500942 |      5 |     8170296 |      50288
             500943 |      4 |     8170295 |      50288
             500944 |      3 |     8170294 |      50288
```

Joonis 4. Suurobjekti disaini näidispäring ja selle tulemuse näide.

Suurobjekti disaini puhul tuleb meeles pidada, et *oid* kustutamine tabelist ei kustuta objekti ennast andmebaasist. Suurobjekti kustutamine on eraldi toiming, mida ei tohi ebaotstarbeka ruumikasutuse vältimiseks jätta tähelepanuta.

4.2 Failid kohalikus arvutis väljaspool andmebaasi

Ingliskeelsed nimed: *Files on a local computer outside the database, files in the filesystem*

Jõud:

- Soovitakse vältida andmebaasi.
- Faile on vaja kiiresti muuta (nt kärpida).

Lahendus: Andmebaasi tabelis hoitakse faili ühtset ressursiidentifikaatorit. Viide salvestatakse tabelisse tekstilise väärtusena. Maksimaalne väljapikkus sõltub ühelt poolt maksimaalsest lubatud väljapikkusest andmebaasisüsteemis ja teisalt sellest, millist aadressi andmebaasis hoitakse ning milline on operatsioonisüsteemist tulenev piirang selle pikkusele. Erinevates operatsioonisüsteemides on maksimaalne võimalik kataloogitee pikkus erinev. Andmebaasis on võimalik hoida:

1. internetiaadressi (URL-i),
2. täielikku kataloogiteed (sh failinime),
3. osalist kataloogiteed (sh failinime),
4. ainult failinime.

Parimaks lahenduseks peetakse valdkonnale vastava osalise kataloogitee salvestamist (nt toodete failid on ühes kataloogis ja isikute failid on teises kataloogis) [62]. Viidet sisaldav tekstiline veerg võiks olla tüüpi, mis lubab vähemalt 4095 märgi pikkuseid väärtuseid (pikim lubatud kataloogitee Linuxis [63]) ning veerul võiks olla CHECK kitsendus, mis sõltuvalt oludest piirab täiendavalt maksimaalset lubatud väärtuse pikkust. Kui operatsioonisüsteem muutub, saab maksimaalset lubatud pikkust muuta CHECK kitsendust kustutades ja uuesti luues. Samuti võib CHECK kitsenduse alusel valideerida aadressi põhipunkte nagu näiteks see, et failinimes peab olema punkt ja laiend, või et kataloogitee puhul peab seal sisalduma kaldkriipse. Parema hallatavuse huvides võiks selleks luua eraldi CHECK kitsenduse, mitte panna mitut tingimust kokku ühte kitsendusse. Kui andmebaasisüsteem pakub spetsiaalset andmetüüpi, kuhu kuuluvad väärtused on ühtsed ressursiidentifikaatorid, siis võib kasutada seda tüüpi (vt jaotis 2.1.2).

Allikad ja autorid: [17]**,** [18]**,** [19]**,** [20]**,** [29]**,** [56]**,** [57]**,** [59]

Mustri tugevad küljed:

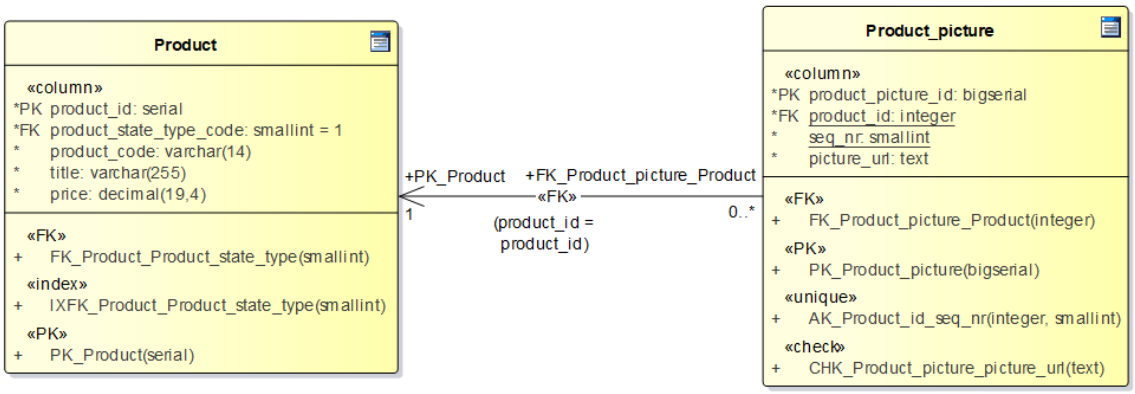
- Andmebaasi andmemaht on väiksem ning seetõttu on ka andmebaasi haldamine (näiteks varundamine) lihtsam.

- Faili lugemisel otse failisüsteemist on jõudlus oluliselt parem, sest andmebaasile juurdepääsuks vajalikud tarkvarakihid jäetakse vahele.
- Failide suurus on limiteeritud failisüsteemi poolt.
- Failide maht on limiteeritud failisüsteemi poolt.
- Lihtne on üle minna pilvelahendusele. Kui hoida andmebaasis viiteid failidele, siis saab viited vahetada pilvelahenduse (näiteks Amazoni S3 korvi) viidete vastu.
- Faile saab kergemini koheselt muuta (näiteks piltide kärpimine).
- Hea vertikaalne skaleeritavus, sest serverisse saab muutmälu lihtsasti juurde lisada.

Mustri nõrgad küljed:

- Andmebaasi kasutaja vajab eraldi õiguseid andmebaasist väljaspool kirjutamiseks (failide salvestamiseks).
- Vaja on arendada liides väliste failide jälgimiseks.
- Raske on tagada andmete terviklikkust, sest välised failid ja andmebaas võivad minna sünkroonist välja.
    - Failisüsteemis võivad rakenduse hallatavate failide hulgas olla failid, millele andmebaasist ei viidata.
    - Andmebaasis võivad olla read viidetega välistele failidele, mis on kustutatud või paigutatud kuhugi mujale.
- Failisüsteemi ja andmebaasi turvalisuse seaded ei ole üksteisest sõltuvad. Võib juhtuda, et kasutaja pääseb ligi failile, mille nägemiseks tal õigus puudub.
- Keerulisem arhitektuur.
- Vaja on kavandada kataloogide struktuur.
- Vaja on realiseerida failide kataloogidesse jaotamise algoritm.
- Vaja on realiseerida eraldi varundamise strateegia.
- Kui viited on pikemad kui 256 tähemärki, siis ei pruugi kõik failisüsteemid viidete haldamisega hakkama saada.
- Puudulik kontroll selle üle, kui mitu kasutajat samaaegselt samu andmeid (sh faile) muudab.

Näide: Joonis 5 esitab väliste failide kasutamise disaini näite PostgreSQL andmebaasisüsteemi jaoks.



Joonis 5. Failisüsteemi ja pilvelahenduse disaini kasutamise näide.

Kuna failid asuvad antud lahenduse puhul väljaspool andmebaasi, siis tuleb tähelepanu pöörata failide kustutamise strateegiale. Failide asukohaviite kustutamine andmebaasi tabelist ei kustuta faili failisüsteemist, mistõttu on hulkurfailid lihtsad tekkima. Veerg *picture_url* on *text* tüüpi (maksimaalne lubatud väärtuse suurus on 1 GB) ja veerule loodud CHECK kitsendus piirab maksimaalset lubatud väärtuse pikkust.

Joonisel 6 on esitatud näidispäring ja päringu tulemus, kus väljastatakse lisaks muudele tootepildi tabeli andmetele ka viide faili asukohale (*picture_url*). Antud juhul on andmebaasis failinimi.

```
SELECT product_id, seq_nr, picture_url FROM product_picture LIMIT 5;

 product_id | seq_nr |                 picture_url
------------+--------+---------------------------------------------
     210001 |      0 | f6f48eeb-e861-4f9e-b7b2-a75aaf805810_1.jpg
     210002 |      1 | 1db23618-cb37-45ae-9f94-36b2ea0d783a_91.jpg
     210034 |     12 | 5f004a45-016f-4b34-99a3-76c2cc691a5a_45.jpg
     210034 |     11 | 5a3ad2ba-cfa0-4ca7-a404-48b1a495eaa4_79.jpg
     210034 |     10 | 6fe6ada4-2645-4d94-b28b-498805c83a0e_64.jpg
```

Joonis 6. Failisüsteemi disaini näidispäring ja selle tulemuse näide.

4.3 Failid pilves

Ingliskeelsed nimed: *Files in the cloud, cloud storage*

Jõud:

- Tänapäeval hoitakse peaaegu et kõike pilves.

- Soovitakse ise andmebaasi hallata nii vähe kui võimalik ja selle kaudu kulusid kokku hoida.

Lahendus: Andmebaasi tabelis hoitakse failile viitavat internetiaadressi, mis viitab faili asukohale pilves või faili nime. Erinevates veebilehitsejates on piirang maksimaalsele lubatud internetiaadressi pikkusele erinev [64]. Selleks, et toetada võimalikult suurt hulka erinevaid lehitsejaid, peaks maksimaalne pikkus piirduma 2000 märgiga [65]. Viidet sisaldaval tekstilisel veerul võiks olla selline maksimaalse väljapikkuse piirang. Viide salvestatakse tabelisse tekstitüüpi väärtusena. Kui salvestatakse veebiaadresse, siis võib veerule loodud CHECK kitsenduse abil valideerida veebiaadressi põhiaspekte nagu see, et algab protokolliga ja sisaldab kaldkriipse. Kui andmebaasisüsteem pakub spetsiaalset andmetüüpi, kuhu kuuluvad väärtused on ühtsed ressursiidentifikaatorid, siis võib kasutada seda tüüpi (vt jaotis 2.1.2).

Allikad ja autorid: [17]**,** [18]**,** [20]**,** [58]

Mustri tugevad küljed:

- Süsteemi väga hea skaleeritavus ja kasutaja ei pea selle pärast muretsema. Koormuse kasvades arvutiressursside lisamise eest hoolitseb teenusepakkuja.
- Ei pea ise haldama suuremahulisi andmebaase.
- Varundamine toimub automaatselt.
- Väga hea versioonihaldus.
- Failidega tehtavad operatsioonid ei mõjuta rakenduse jõudlust.
- Suhteliselt odav. Makstakse selle eest, mida kasutatakse.
- Andmete valdaja ei pea tegelema ülalhoiuga, sest ressursse haldab teenusepakkuja, kes vastutab teenuse toimimise, monitooringu ja turvalisuse küsimuste eest.
- Mitmekülgsem failidele juurdepääsukontroll. Ligipääsetavust on võimalik väga detailselt seadistada.
- Võimaldab säästa raha tarkvara ja seadmete ostmisel, hooldamisel ja täiendamisel.
- Võimaldab andmebaasirakendust kiiresti realiseerida.

Mustri nõrgad küljed:

- Interneti probleemid mõjutavad teenuse töökiirust ja käideldavust.
- Kasutajaandmete turvalisus sõltub teenusepakkujast.
- Kõiki andmeid ei saa teenusepakkujale säilitamiseks ega töötlemiseks usaldada.
- Võib olla oht, et teenusepakkuja ei suuda ühel päeval andmeid varundada ja need lähevad serveri krahhi korral kaduma.
- Kuna teenusepakkuja võib olla globaalne, siis on võimalikud probleemid klienditeenindusega – raske kätte saada, ennast arusaadavaks teha, reageerimine võtab kaua aega, tegevused ja otsused on läbipaistmatud.
- Vaidlusküsimusi võib olla vaja lahendada välisriigi kohtus.

Näide:

Pilvelahenduse andmebaasi disain on samasugune failisüsteemi lahenduse disainiga. Joonis 5 kehtib ka pilvelahenduse puhul.

Joonisel 7 on esitatud näidispäring ja päringu tulemus, kus väljastatakse lisaks toote identifikaatorile ka viide faili asukohale pilves (*picture_url*). Antud juhul on andmebaasis faili veebiaadress.

```
SELECT picture_url FROM product_picture LIMIT 5;

                                picture_url
------------------------------------------------------------------------------
https://cloud-photo-storage.s3.amazonaws.com/assets/ERF1L0/f6f48eeb-e861-4f9e-b7b2-a75aaf805810_23.jpg
https://cloud-photo-storage.s3.amazonaws.com/assets/ERF1L0/1db23618-cb37-45ae-9f94-36b2ea0d783a_91.jpg
https://cloud-photo-storage.s3.amazonaws.com/assets/P1KLMS/5f004a45-016f-4b34-99a3-76c2cc691a5a_45.jpg
https://cloud-photo-storage.s3.amazonaws.com/assets/P1KLMS/5a3ad2ba-cfa0-4ca7-a404-48b1a495eaa4_79.jpg
https://cloud-photo-storage.s3.amazonaws.com/assets/P1KLMS/6fe6ada4-2645-4d94-b28b-498805c83a0e_64.jpg
```

Joonis 7. Pilvelahenduse disaini näidispäring ja selle tulemuse näide.

5 Näiterakendused

Töö ühe tulemusena realiseeritakse kirja pandud disainimustrite põhjal näiteandmetega andmebaasid. Iga andmebaasi põhjal luuakse C# programmeerimiskeeles andmete lisamise, muutmise, kustutamise ja vaatamise funktsionaalsusega veebirakendus[1] kasutades .NET raamistikku. Käesolevas peatükis esitab autor nõuded realiseeritavale rakendusele, annab ülevaate disainimustrite põhjal realiseeritud andmebaaside disainidest ning kirjeldab rakenduste ülesehitust, kasutajaliidest ja rakendusse testandmete genereerimist.

Kavandatav rakendus on funktsionaalsuselt suhteliselt lihtne. Samas on rakenduse vajatav andmebaas piisavalt keeruline, et katsetada selle põhjal erinevaid ja mitmekülgseid päringuid. Selle asemel, et realiseerida üks suhteliselt keeruline rakendus ühes versioonis, realiseeritakse suhteliselt lihtne rakendus mitmes erinevas versioonis ehk üks versioon iga erineva andmebaasi disainilahenduse jaoks. Näiterakendus eeldab, et *reaalses elus* ei korrata ühte ja sama pilti sageli erinevate toodete puhul. Seega puudub süsteemis eraldi pildipanga funktsionaalsus, kuhu pilte üles laadida ning kui tootega on vaja siduda pilt, siis ei valita seda pildipangast. Selle asemel käsitletakse pilte kui toote lisaandmeid, mida tuleb registreerida koos tootega.

Tõsi küll, käesoleva töö mõõtmiste jaoks tehtavas testandmete genereerimisel valitakse tootepilte suhteliselt väikesest piltide hulgast ja seega korduvad samad pildid erinevate toodete juures.

5.1 Analüüs

Järgnevalt esitatakse kavandatava süsteemi alamosa funktsionaalsed ja mittefunktsionaalsed nõuded. Funktsionaalsed nõuded defineerivad süsteemi käitumise

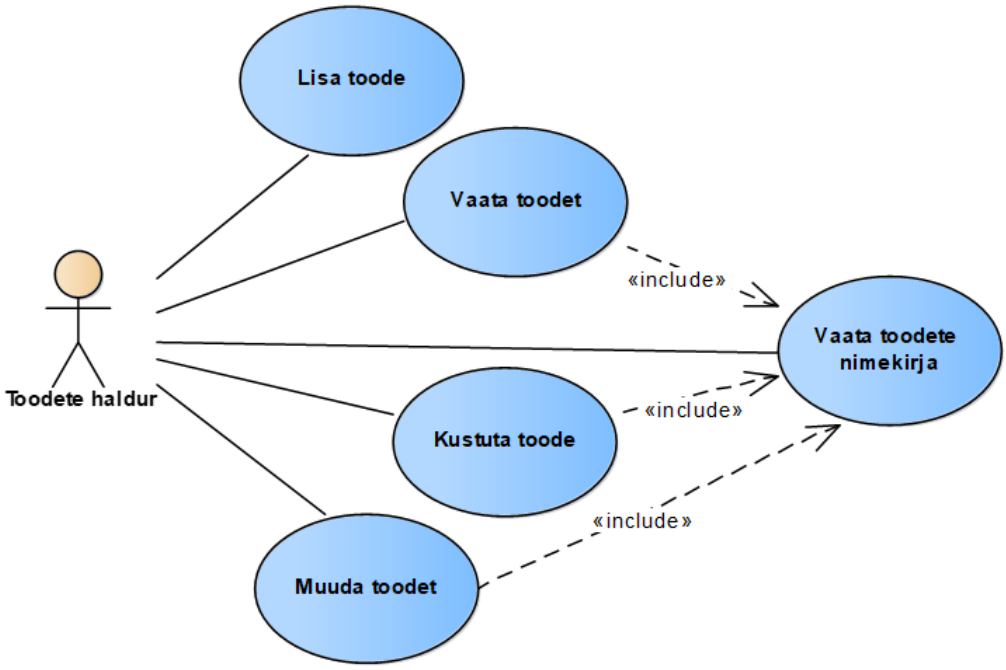
[1] Rakenduste lähtekood: https://github.com/erlemaido/File-Storage-Design-Patterns. Kuni kaitsmise perioodi lõpuni hoiab autor ühte näidisrakendust üleval Amazoni pilveserveris. Täpsem info on GitHub'i lehel.

ehk kirjeldavad, mida süsteem peab tegema. Mittefunktsionaalsed nõuded on kriteeriumid, mis täpsustavad, milline peab süsteem olema.

5.1.1 Kasutusjuhtude mudel

Infosüsteemi funktsionaalsete nõuete kirjeldamiseks kasutatakse kasutusjuhtude mudelit, mis koosneb kasutusjuhtude diagrammist ja kasutusjuhtide tekstikirjeldusest. Joonisel 8 esitatakse ülevaade kavandatava süsteemi alamosast, kus tegutsejaks on toodete haldur. Kavandatava süsteemi allosa peab võimaldama uusi tooteid lisada, olemasolevaid tooteid muuta ja kustutada ning vaadata nii toodete nimekirja kui ka iga toote detailvaadet.

Kavandatavas süsteemis ei realiseerita kasutaja tuvastamist, sest käesoleva lõputöö eesmärgiks ei ole realiseerida valmisrakendus, vaid katsetada mustrite kataloogis kirjeldatud disainimustrite poolt pakutud lahendusi. Tabelites 2–6 esitatakse kasutusjuhtude tekstilised kirjeldused laiendatud formaadis.



Joonis 8. Süsteemi alamosa kasutusjuhtude diagramm.

Tabel 2. Kasutusjuht "Lisa toode".

Nimi	**Lisa toode**
Kirjeldus	Toodete haldur lisab uue toote koos vajalike detailidega andmebaasi. Soovi korral lisatakse tootele tootega seotud pildid.
Eeltingimused	-
Põhiline sündmuste käik	1. Toodete haldur annab toodete nimekirja lehel uue toote lisamise alustamise korralduse. 2. Süsteem kuvab toote sisestamise vormi. 3. Toodete haldur sisestab tootekoodi ja nimetuse, määrab hinna, valib toote seisundi liigi, lisab soovi korral pildi(d) ning annab korralduse salvestada. 4. Süsteem salvestab toote andmed. Toode ilmub toodete nimekirja lehele. Kasutaja suunatakse lisatud toote detailvaate lehele.
Alternatiivne sündmuste käik	Toodete haldur ei soovi toodet lisada: Standardstsenaariumi sammud 1–2. 3. Toodete haldur ei soovi toodet lisada ja annab korralduse tagasi pöörduda. 4. Süsteem ei salvesta toodet ja suunab toodete halduri tagasi toodete nimekirja lehele. Sisestatud andmed ei vasta nõuetele: Standardstsenaariumi sammud 1–3. 4. Toodet ei salvestata, süsteem kuvab veateate.
Lõpptulemused	Edukas stsenaarium: ▪ Toote andmed ja tootega seonduvad pildid salvestati andmebaasi.

Tabel 3. Kasutusjuht "Vaata toodete nimekirja".

Nimi	**Vaata toodete nimekirja**
Kirjeldus	Toodete haldur soovib lisatud toodetest ülevaadet ning vaatab selleks toodete nimekirja.
Eeltingimused	Toodete andmed on registreeritud
Põhiline sündmuste käik	1. Toodete haldur avab rakenduse või valib menüüribalt „Tooted“. 2. Süsteem kuvab toodete nimekirja. 3. Toodete haldur näeb toodete tootekoodi, nimetust, hinda, toote seisundi liiki ja piltide olemasolu korral järjekorras kõige esimest pilti.
Alternatiivne sündmuste käik	Süsteemis ei ole tooteid registreeritud: Standardstsenaariumi samm 1.

Nimi	Vaata toodete nimekirja
	2. Toodete andmeid ei kuvata.
Lõpptulemused	Edukas stsenaarium: ▪ Toodete haldur on saanud toodetest ülevaate.

Tabel 4. Kasutusjuht "Vaata toodet".

Nimi	Vaata toodet
Kirjeldus	Toodete haldur soovib vaadata konkreetse toote andmeid. Toodete haldurile kuvatakse toote andmed eraldi lehel.
Eeltingimused	Toode on andmebaasis olemas.
Põhiline sündmuste käik	1. Käivitub kasutusjuht „Vaata toodete nimekirja“. 2. Toodete haldur valib toodete nimekirja lehelt konkreetse toote ja annab korralduse vaadata selle toote detaile. Kasutaja võib näiteks vajutada toote nimetuse peale või vajutab tootepildi olemasolu korral tootepildi peale. 3. Süsteem kuvab toote detailvaate lehte, kus on esitatud nii toote põhiandmed kui ka selle pildid, mis on järjekorranumbri alusel sorteeritud.
Alternatiivne sündmuste käik	-
Lõpptulemused	Edukas stsenaarium: ▪ Toodete haldur saab näha toote andmeid toote detailvaates.

Tabel 5. Kasutusjuht "Muuda toodet".

Nimi	Muuda toodet
Kirjeldus	Toodete haldur soovib värskendada toote andmeid.
Eeltingimused	Toode on andmebaasis olemas.
Põhiline sündmuste käik	1. Käivitub kasutusjuht „Vaata toodete nimekirja“. 2. Toodete haldur valib toodete nimekirja lehelt konkreetse toote ja annab korralduse seda muuta. 3. Süsteem kuvab toote muutmise vormi, kus kuvatakse tootekood, nimetus, hind, toote seisundi liik ja väljad piltide lisamiseks, muutmiseks või kustutamiseks. 4. Toodete haldur muudab toote andmeid ning annab salvestamise korralduse. 5. Süsteem salvestab toote andmed ja suunab toodete halduri värskendatud toote detailvaate lehele.

Nimi	Muuda toodet
Alternatiivne sündmuste käik	Toodete haldur ei soovi toodet muuta: Standardstsenaariumi sammud 1–3. 4. Toodete haldur ei soovi toodet muuta ja annab korralduse tagasi pöörduda. 5. Süsteem ei salvesta toodet ja suunab toodete halduri tagasi toodete nimekirja lehele Sisestatud andmed ei vasta nõuetele: Standardstsenaariumi sammud 1–4. 5. Toodet ei salvestata, süsteem kuvab veateate.
Lõpptulemused	Edukas stsenaarium: ▪ Toote andmed uuendati andmebaasis.

Tabel 6. Kasutusjuht "Kustuta toode".

Nimi	Kustuta toode
Kirjeldus	Toodete haldur soovib toodet kustutada.
Eeltingimused	Toode on andmebaasis olemas.
Põhiline sündmuste käik	1. Käivitub kasutusjuht „Vaata toodete nimekirja“. 2. Toodete haldur valib toodete nimekirja lehelt konkreetse toote ja annab korralduse see kustutada. 3. Süsteem kuvab toote kustutamise kinnitamise lehe, kus kuvatakse toote andmed. 4. Toodete haldur kordab kustutamise soovi. 5. Süsteem kustutab toote andmed ja tootega seonduvad pildid andmebaasist.
Alternatiivne sündmuste käik	Toodete haldur ei soovi toodet kustutada: Standardstsenaariumi sammud 1–3. 4. Tootehaldur ei soovi toodet kustutada ja annab korralduse tagasi pöörduda. 5. Süsteem ei kustuta toodet ja suunab toodete halduri tagasi toodete nimekirja lehele
Lõpptulemused	Edukas stsenaarium: ▪ Toote andmed ja tootega seonduvad pildid kustutati andmebaasist.

5.1.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsete nõuete kirjeldamiseks kasutab autor FURPS+ mudelit. FURPS on 1987. aastal Robert Grady poolt välja töötatud raamistik nõuete klassifitseerimiseks.

1992. aastal laiendas Grady koostöös IBM-iga nimetatud raamistikku ning loodi FURPS+ [66]. Raamistik jaotab nõuded järgmiste kategooriate vahel.

- Funktsionaalsus (*Functionality*)
- Kasutatavus (*Usability*)
- Usaldusväärsus (*Reliability*)
- Jõudlus (*Performance*)
- Toevõime (*Supportability*)
- Disain (*Design*)
- Teostus (*Implementation*)
- Liidestused (*Interface*)
- Infrastruktuur (*Physical*)

Järgnevalt jäetakse antud mudelist välja funktsionaalsus, sest alamsüsteemi funktsionaalseid nõudeid kirjeldati kasutusjuhtude mudeli alampeatükis. FURPS+ raamistikust lähtuvalt klassifitseeritud mittefunktsionaalsed nõuded on esitatud Tabelis 7.

Tabel 7. Allsüsteemi mittefunktsionaalsed nõuded.

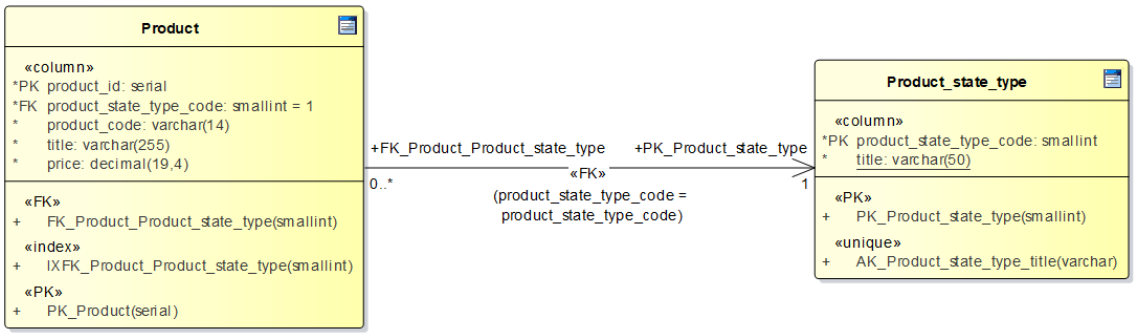
Nõue	**FURPS+ kategooria**
Süsteem peab andmete hoidmiseks kasutama PostgreSQL andmebaasisüsteemi.	*Design*
Süsteemil peab olema veebipõhine kasutajaliides.	*Usability*
Kasutajaliides peab vastama enim levinud tavadele. Kasutusloogika peab olema esimesel kasutuskorral intuitiivselt mõistetav.	*Usability*

5.2 Andmebaasi disain

Käesolevas peatükis kirjeldatakse kirja pandud disainimustrite põhjal realiseeritava andmebaasi disaini. Näiteandmebaas realiseeritakse PostgreSQL-is (ver 12) ning on mõeldud toodete ja nende piltide andmete hoidmiseks.

Andmebaasi projekteerimisel lähtuti funktsionaalsetest nõuetest. Autor lõi PostgreSQL andmebaasisüsteemis iga disaini kohta samas andmebaasis eraldi skeemid vastavalt disainilahenduste nimedega *bytea*, *blob*, *filesystem* ja *cloud*.

Andmebaas koosneb kolmest tabelist – *toode (product)*, *toote seisundi liik* (*product state type)* ja *tootepilt* (*product picture)*. Tabelid *toode* ja *toote seisundi liik* on kõikide disainide puhul ühesugused, kuid tabel *tootepilt* erinev. *Tootepildi* tabeli kirjeldus erinevate disainide jaoks esitati peatükis „Mustrite kataloog“ ja antud peatükis seda enam ei korrata. Joonisel 9 on esitatud tabelite *toode* ja *toote seisundi liik* disaini kujutav klassidiagramm. *Toote seisundi liik* on klassifikaator ning igal *tootel* on alati täpselt üks *toote seisundi liik*, mis näitab selle toote hetkeseisundit. SQL tabelite ja kitsenduste loomise laused on esitatud Lisas 2.



Joonis 9. Tabeleid toode ja toote seisundi liik kujutav klassidiagramm.

Igal tootel on olemas tootekood, nimetus ja hind ning igal toote seisundi liigil on olemas kood ja nimetus. Ühel tootel võib olla mitu pilti. Igal tootepildil on oma järjekorranumber, mis määrab piltide kasutajatele kuvamise järjekorra. Tootepiltide tabelis on pildi veerg, mille tüüp ja võimalikud täiendavad seotud kitsendused on iga disainimustri korral erinevad. Kõikide tabelite kõik veerud on kohustuslikud, mis tähendab, et kõikidele väljadele peab sisestama väärtused ehk NULL-ide kasutus ei ole lubatud. Failide asukoha registreerimisel ei kasutatud mitte *uri* tüüpi vaid tekstitüüpi veergu.

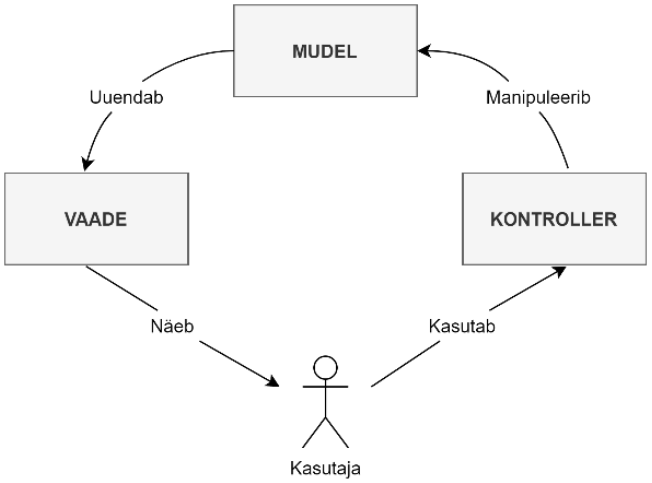
PostgreSQL loob automaatselt indeksid primaarvõtmetele ning unikaalsuse kitsendustega hõlmatud veergudele, kuid mitte välisvõtmetele [67]. Tabelis *toode* sisalduvale välisvõtme veerule toote seisundi liik kood (*product_state_type_code*) lõi autor eraldi indeksi, mille loomise lause on kujutatud Joonisel 10. Lisaks lõi autor unikaalsuse kitsenduse tabelis *toote seisundi liik* veerule *nimetus* (*title*), mis sisaldab nelja erinevat väärtust: *ootel, aktiivne, mitteaktiivne, lõpetatud*. Vaikimisi on klassifikaatorile määratud väärtus „ootel“. Tabelile *tootepilt* on lisatud unikaalsuse kitsendus *toote id* (*product_id*) ja *järjekorranumbri* (*seq_nr*) kombinatsioonile.

```
CREATE INDEX IXFK_Product_Product_state_type
ON Product(product_state_type_code);
```

Joonis 10. Indeksi loomise lause PostgreSQL-is.

5.3 Rakenduse ülesehitus

Rakenduse loomisel kasutati MVC (*Model-View-Controller*) arhitektuurimustrit, mille korral jagatakse rakendus kolmeks osaks. MVC arhitektuur sobib lihtsatele rakendusele ning autor valiski antud arhitektuuri lihtsuse ja paindlikkuse tõttu. Kuna kihid on üksteisega nõrgalt seotud, ehk ühe kihi muutmine ei mõjuta teisi kihte, siis on rakenduse laiendamine mugav. MVC arhitektuuri komponendid on kujutatud Joonisel 11.



Joonis 11. MVC disainimustri diagramm.

Mudeli kiht sisaldab ainult rakenduse mudelite andmeid ning on täiesti sõltumatu kasutajaliidesest. Vaadete kiht vastutab andmete visualiseerimise eest. Vaade pääseb ligi mudeli andmetele, kuid see ei tea, mida need andmed tähendavad. Kontroller on vaate ja mudeli vahel ehk vastab kliendi päringutele ja teostab suhtlemist mudeli objektiga.

Igale vaate veebilehekülje vastab üks GET ja üks POST meetod, välja arvatud *Index* ja *Details* vaatel (ainult GET). Vaated on vastavuses kontrolleritega, need grupeeritakse kontrolleri nimega kataloogi ja lisatakse *Views* kausta.

Järgnevates alampeatükkides kirjeldatakse täpsemalt iga disainimustri korral piltide salvestamise realiseerimist.

5.3.1 *Bytea* disain

Bytea lahenduse realiseerimine on küllaltki lihtne. Sissetulevad pildid teisendatakse baitide jadaks ja salvestatakse andmebaasi. Joonis 12 esitab piltide baitide jadaks teisendamise meetodi. Meetodi tulemus salvestatakse tabelisse *tootepilt* pildi veergu.

```
private static async Task<byte[]> ConvertToBytes(IFormFile image)
{
    await using var memoryStream = new MemoryStream();
    await image.CopyToAsync(memoryStream);
    return memoryStream.ToArray();
}
```

Joonis 12. Pildifailide baitide jadaks teisendamise meetod.

5.3.2 Suurobjekti disain

Pildifaili salvestamise meetodi suurobjektina esitab Joonis 13. Kõige pealt luuakse tühi fail, millele määratakse süsteemi poolt objekti identifikaator. Suurobjektide lugemiseks ja kirjutamiseks on vajalik transaktsiooni e tehingu kasutamine. Transaktsiooni sees avatakse loodud fail lugemiseks ja kirjutamiseks. Seejärel teisendatakse pildifail baitide jadaks, mis salvestatakse tükkidena loodud faili. Tehtud muudatused salvestatakse andmebaasi. Meetod tagastab loodud objekti identifikaatori, mis salvestatakse tabelisse *tootepilt*.

```
private async Task<uint> SaveFile(byte[] fileBytes)
{
    var connection = GetConnection();

    var manager = new NpgsqlLargeObjectManager(connection);
    var oid = await manager.CreateAsync(0);

    await using var transaction = await connection.BeginTransactionAsync();
    await using (var stream = await manager.OpenReadWriteAsync(oid))
    {
        await stream.WriteAsync(fileBytes.AsMemory(0, fileBytes.Length));
        await stream.SeekAsync(0, System.IO.SeekOrigin.Begin);
    }

    await transaction.CommitAsync();
    await connection.CloseAsync();

    return oid;
}
```

Joonis 13. Pildifaili suurobjektiks salvestamise meetod.

5.3.3 Failisüsteemi disain

Pildifailide failisüsteemi salvestamisel lisatakse failid rakenduse siseselt *wwwroot* kataloogi ning andmebaasi salvestatakse andmebaasi suhteline teekond pildini. Pildiaadressi salvestamiseks kasutatakse *Path.Combine()* meetodit, mis võimaldab kombineerida teekonna vastavalt kasutatavale süsteemile. Kuna käesoleva töö mõõtmiste jaoks genereeritavate piltide tabeli ridade hulk on suur, siis jaotatakse pildid veel omakorda tootekoodi järgi alamkataloogidesse. Pildi salvestamisel kombineeritakse pildi nimi unikaalsest tunnusest ja pildi esialgsest nimest. Andmebaasis hoitakse pildi veerus vaid pildi nime. Joonis 14 esitab piltide salvestamise meetodi failisüsteemi lahenduse korral.

```
private string UploadFile(IFormFile file, string productCode)
{
    var uploadsFolder = Path.Combine(_webHostEnvironment.WebRootPath,
        "images/" + productCode);
    if (!Directory.Exists(uploadsFolder))
    {
        Directory.CreateDirectory(uploadsFolder);
    }

    var uniqueFileName = Guid.NewGuid() + "_" + file.FileName;
    var filePath = Path.Combine(uploadsFolder, uniqueFileName);
    using var fileStream = new FileStream(filePath, FileMode.Create);
    file.CopyTo(fileStream);
    return uniqueFileName;
}
```

Joonis 14. Failisüsteemi disaini piltide salvestamise meetod.

5.3.4 Pilvelahenduse disain

Autor kasutas pilvelahenduse puhul piltide salvestamiseks Amazon S3 teenust. Kõigepealt loodi Amazon Web Services keskkonnas uus S3 korv (*S3 bucket*), millele anti unikaalne nimi ja valiti regioon. Autor valis regiooniks US East 1, sest tegemist on kõige laialdasemalt toetatud regiooniga, kus kõik AWS-i teenused on kättesaadavad. Korvi kirjutamise õigused tagati esialgu AWS *credentials* faili abil, kuid hiljem rakenduste pilveserverisse ülesseadmisel anti rakendusele korvi objektide ja korvi kirjutamise õigused. S3 korvi ja korvi objektide nähtavust reguleerivad erinevad eeskirjad (*access control lists, bucket policies, access point policies*) [68]. Käesolevas töös failidele ligipääsetavust eraldi ei reguleeritud. Korvile ja korvi objektidele anti avalik juurdepääs ehk faili URL-i teades on võimalik faili näha.

Pilvelahenduse realiseerimiseks tuli rakendusse lisada AWSSDK.Core ja AWSSDK.S3 paketid. Piltide salvestamise realiseerimiseks tuleb kõige pealt luua uus AmazonS3 klient, millele määratakse korvi regioon. Pildi nimi kombineeritakse unikaalsest tunnusest ja pildi esialgsest nimest ning igale pildile luuakse korvi sees unikaalne objekti võti. Seejärel luuakse uus päring, millele antakse vajalikud andmed nagu korvi nimi, objekti võti, päringu *input stream* ja sisutüüp. S3 klient salvestab *PutObjectAsync()* meetodiga objekti varasemalt loodud korvi. Autor realiseeris S3 disaini kahel viisil. Esimesel juhul tagastab faili salvestamise meetod pildi nime, mis salvestatakse andmebaasi tabelisse *tootepilt*. Antud disaini faili salvestamise meetod on esitatud Joonisel 15.

```
public async Task<string> AddItem(IFormFile file, string subFolderName)
{
    var fileName = Guid.NewGuid() + "_" + file.FileName;
    var objectKey = $"{FolderName}/{subFolderName}/{fileName}";

    await using var ms = new MemoryStream();
    await file.CopyToAsync(ms);

    var putObjectRequest = new PutObjectRequest
    {
        BucketName = BucketName,
        Key = objectKey,
        InputStream = ms,
        ContentType = file.ContentType
    };

    await _s3Client.PutObjectAsync(putObjectRequest);

    return fileName;
}
```

Joonis 15. S3 lahenduse piltide salvestamise meetod.

Teisel juhul on objekti lisamise põhimõte sama, kuid faili salvestamise meetodiga tagastatakse pildi URL, mille genereerimise meetod on esitatud Joonisel 16. Korvi URL-i hoitakse konstandina, millele lisatakse faili aadressi genereerimisel kataloogist, alamkataloogist ja failinimest koosnev objekti võti. URL salvestatakse andmebaasi tabelisse *tootepilt*.

```
private string GenerateUrl(string objectKey)
{
    return $"{BucketUrl}/{objectKey}";
}
```

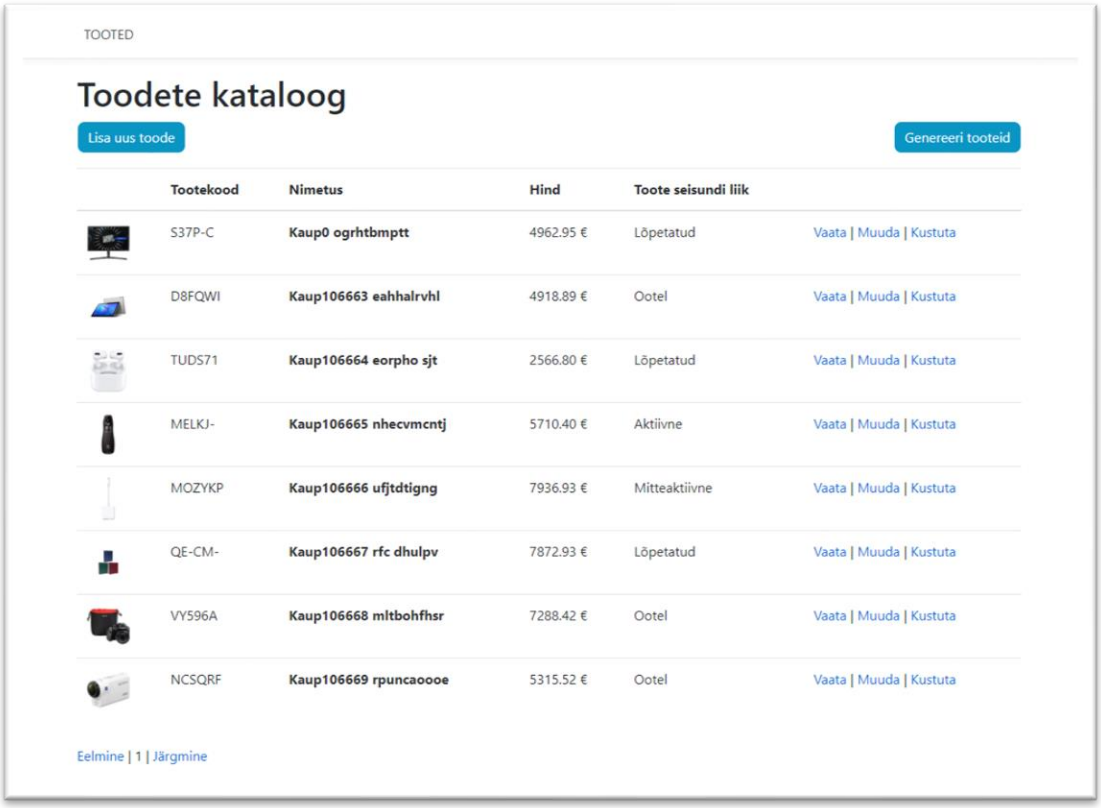
Joonis 16. S3 lahenduse pildi URL-i genereerimise meetod

5.4 Kasutajaliides

ASP.NET Core pakub kliendivaadete arendamiseks Razor Pages raamistikku. Razor leheküljed on *.cshtml* laiendiga failid ning raamistiku süntaksi vaikimisi kasutatavaks keeleks on HTML, kuhu saab vahele kirjutada ka C# koodi. Kasutajaliideses on järgmised vaated:

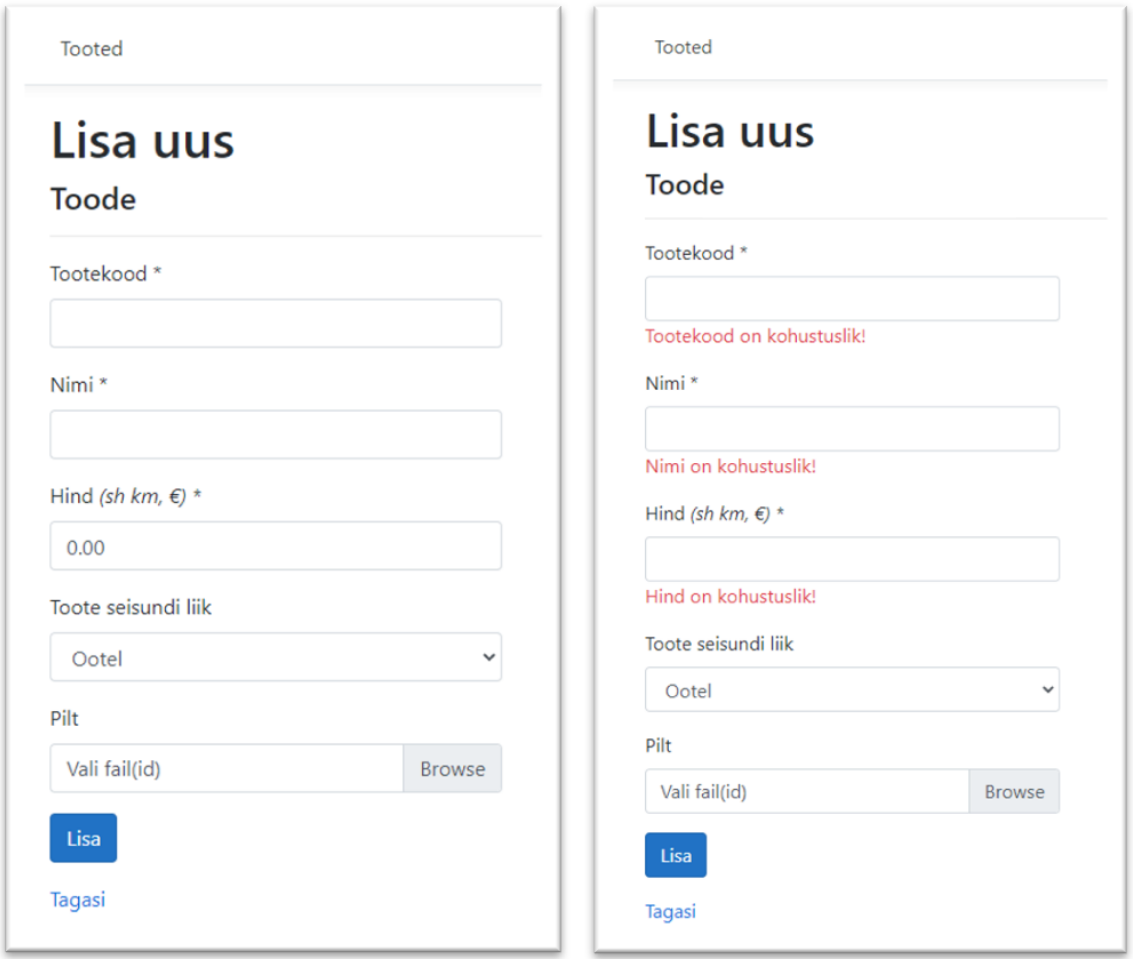
- toodete kataloogi vaade,
- toodete lisamise vormi vaade,
- toodete genereerimise vaade,
- toote detailvaade,
- toote kustutamise vaade,
- toote muutmise vaade.

Toodete kataloogis kuvatakse tabel toodetega (Joonis 17). Toote vaate juures näidatav pilt on toote piltidest kõige väiksema järjekorranumbriga.



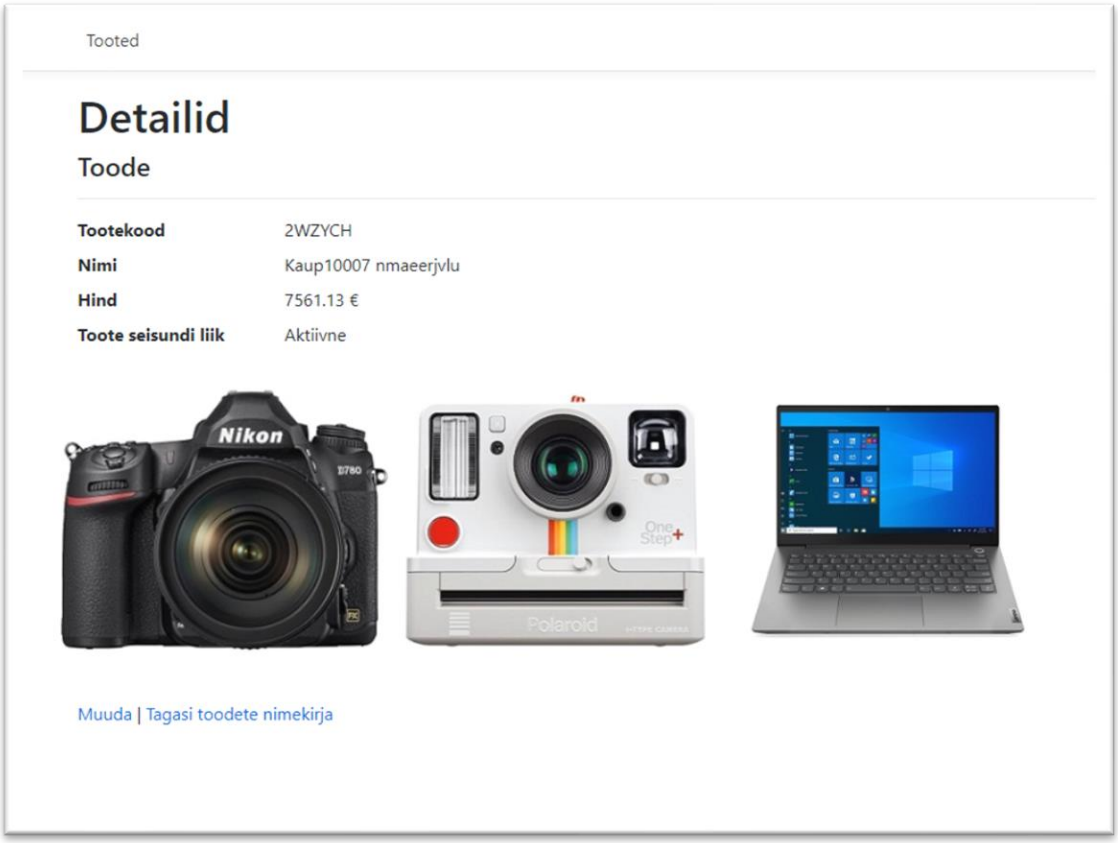
Joonis 17. Toodete nimekirja vaade.

Joonis 18 esitab toote lisamise vormi. Vormil on kõik väljad valideeritud ning ebakorrektse sisendi korral tagastatakse kasutajale vastav hoiatus. Joonise esimene pilt esitab vaate, kui kasutaja on toote lisamise vormi avanud. Joonise teine pilt esitab vaate, kus kasutaja on üritanud salvestada toodet, olles jätnud kohustuslikud väljad täitmata. Rakenduse toodete genereerimise vaade on esitatud Lisas 3.



Joonis 18. Toote lisamise vormi vaated.

Toote detailvaates kuvatakse konkreetse toote andmed. Toote pildid kuvatakse andmete all ridadena, ühes reas maksimaalselt kolm pilti. Pildid on järjestatud järjekorranumbrite alusel alustades väiksemast. Joonis 19 esitab toote detailvaade.



Joonis 19. Toote detailvaade.

5.5 Testandmed

Kõik andmebaasis realiseeritavad tabelid täidetakse mõõtmiste jaoks testandmetega. Selleks, et hinnata andmemahtude mõju andmete otsimise ja rea lisamise operatsioonide täitmise kiirustele, viib autor mõõtmised läbi kolme erineva andmete hulgaga. Genereeritav andmete hulk on esitatud Tabelis 8.

Tabel 8. Mõõtmiseks genereeritavate objektide hulk.

	Tooteid	**Tootepilte**
Andmemaht 1	10 000	0 – 200 000
Andmemaht 2	40 000	0 – 800 000
Andmemaht 3	160 000	0 – 1 600 000

Testandmete genereerimiseks otsustas autor ise kirjutada generaatori. Selleks lõi autor rakenduses eraldi generaatori klassi, millesse kirjutas andmete genereerimiseks vajalikud meetodid ning lisas rakendusele testandmete genereerimise vormi. Vormil peab kasutaja

valima genereeritavate testtoodete hulga ning lisama pildi(d). Testtoodete hulga lisamine on kohustuslik, kuid kui piltide väli jäetakse tühjaks, genereerib süsteem tooted ilma piltideta. Lisas 4 on välja toodud generaatori kood failisüsteemi lahenduse puhul.

Toodete genereerimisel antakse kaasa pildipank, millest programm valib juhuslikult välja tootepildid. Igale tootele genereeritakse 0–20 seotud rida tootepildi tabelisse. Juhul kui kaasa antud pildipanga hulk on väiksem kui 20, siis on maksimaalseks tootele genereeritavate piltide hulgaks kaasa antud pildipanga piltide arv. Testandmete genereerimisel kasutati sajast pildist koosnevat piltide hulka. Rohkete piltidega veebirakendustes nagu näiteks e-poodides, tuleb pildid hoida optimaalse suurusega kuna pilte on väga palju ning eesmärk on leida tasakaal pildi kvaliteedi ja mõistliku suuruse vahel, et tagada nii hea kvaliteet kui ka lehe laadimise kiirus. E-poe rakenduse pildid võiksid jääda alla 70 KB [69]. Käesolevas töös realiseeritud rakenduses kasutati keskmiselt 30 KB suuruseid pilte. Ühte ja sama pilti ei lisata samale tootele mitu korda ehk ühe toote pildid on kõik erinevad. Samas võib testandmetes olla erinevatel toodetel ühesugune seotud pilt. Tootepildi järjekorranumber määratakse automaatselt järjestikuliste numbritena süsteemi poolt. Toote väljade genereerimise formaat ja näidised on esitatud Tabelis 9.

Tabel 9. Toote väljade genereerimise formaat ja näidisväärtused.

Veerg	**Formaat**	**Näidis**
Tootekood	6 tähemärki (tähed, numbrid ja – märk)	A1–HHK
Nimetus	„Kaup“ + genereeritava kauba järjekorranumber + 10 tähemärki (tähed ja tühik)	Kaup23 tha klihst
Hind	vahemikus 0.01 – 9999.99	100.59
Toote seisundi liigi kood	vahemikus 1 – 4	2

Kõikide disainilahenduste puhul oli üldine genereerimise põhimõte sama, erines vaid piltide salvestamise lahendus. Seetõttu otsustas autor võrrelda ka testandmete genereerimise kiirust erinevate disainilahenduste puhul üle erinevate andmehulkade. Tabelis 10 on esitatud juhuslikult genereeritud toodete arv, tootepiltide arv ja genereerimiseks kulunud aeg.

Genereerimiseks kuluvat aega ehk päringu kiirust vaadeldi kasutades veebilehitseja *network* tööriista (Google Chrome veebilehitsejas *inspect element* vaate *network* alammenüü). Kõige kiiremini genereeriti testandmed failisüsteemi lahenduse puhul. Ka *bytea* disaini puhul oli genereerimiseks kuluv aeg küllaltki väike. Tuleb mainida, et kui *bytea* ja failisüsteemi lahendusel genereeriti tootepildi tabeli kirjed hästi kiiresti ja aega võttis pigem andmebaasi salvestamise pool, siis suurobjektide lahenduse puhul võttis aega just tootepiltide ridade lisamine ning kui see oli tehtud, oli andmebaasi salvestamine peaaegu et kohene.

Tabel 10. Testandmete genereerimiseks kulunud aeg erinevate andmemahtude ja disainilahenduste korral.

	Toodete hulk	**Tootepiltide hulk**	**Kulunud aeg**
Bytea	10 000	96 001	1 min 6 s
	40 000	378 857	4 min 24 s
	160 000	1 518 252	18 min 6 s
Suurobjekt	10 000	95 194	6 min 48 s
	40 000	378 268	26 min 30 s
	160 000	1 518 067	1 h 36 min
Failisüsteem	10 000	93 404	25 s
	40 000	382 134	1 min 48 s
	160 000	1 516 646	7 min 48 s
S3 (pilv)	10 000	95 378	2 h
	40 000	381 189	7 h 48 min
	160 000	1 519 739	31 h 12min

Tabelist on näha, et S3 lahenduse puhul osutus andmete genereerimine väga aeganõudvaks. Autor uuris põhjuseid ja üritas probleemile lahendust leida, et genereerimist kiirendada. Amazon pakub S3 korvi andmete kiiremaks ülekandmiseks *Transfer Acceleration* funktsionaalsust, mis lubab 50–500% kiiremat ülekannet [70]. Autor viis funktsionaalsuse kasutamiseks sisse vajalikud muudatused, kuid jättis ilmselt midagi tegemata, sest ei täheldanud mingit muutust andmete genereerimise kiirenemises. Põhjus võib seisneda ka S3 asünkroonses päringu saatmise meetodis. Generaator salvestab andmeid sünkroonselt, kuid .NET raamistik võimaldab kasutada vaid asünkroonset S3 `PutObjectAsync()` meetodit.

Andmebaasisüsteem koostab andmekäitluse lause täitmiseks plaani. Plaani headus ja seega ka lause täitmise kiirus sõltub sellest, kas andmebaasisüsteemil on täpne ettekujutus andmebaasis olevatest andmetest. Selle ettekujutuse saamiseks kasutab andmebaasisüsteem statistikat. Andmebaasi statistika peaks olema värske. PostgreSQL-il on tagaplaanil jooksev programm (deemon) *autovacuum*, mis hoolitseb muuhulgas selle eest, et tabelisse esmakordselt ridade lisamisel ja ka ridade muutmise käigus statistikat automaatselt värskendatakse [71]. Serveris oli see deemon sisselülitatud.

5.6 Pilvekeskkond

Erinevate disainilahenduste võrdlemisel tuleb võrdlemiseks tehtavad mõõtmised viia läbi samal riistvaral. Kuna pilveteenuste populaarsus ajas aina kasvab, siis otsustas autor seada rakendused ja andmebaasi üles pilveserveritesse. Arvestades, et töö autoril on varasem kogemus Amazon Web Services poolt pakutavate EC2 (*Elastic Compute Cloud*) pilveserveritega ja kuna pilvelahenduse disainimuster on realiseeritud kasutades Amazon S3-e, siis kasutatakse ka rakenduste ja andmebaasi ülesseadmisel AWS-i teenuseid.

Amazon Elastic Compute Cloud on 2006. aastal AWS-i poolt turule toodud toode, mis pakub skaleeritavat andmetöötlust pilves [72]. Autor lõi AWS-is viis *t3.large* tüüpi Amazon EC2 pilveserverit – üks iga rakenduse jaoks ja viies andmebaasi jaoks. Serveri tüüp t*3.large* valiti, sest see tundus autorile käesoleva töö jaoks piisavalt suur mõõtmiste edukalt läbiviimiseks. Kõikide serverite regiooniks valiti US East 1.

Rakendused pandi pilveserveritesse Dockeri konteineritena. Docker on arvutiprogramm, mis teostab operatsioonisüsteemi tasemel virtualiseerimist ehk konteineriseerimist. Konteiner on isoleeritud keskkond, mida saab luua, peatada, käitada, kustutada või muuta. Konteineri käitamiseks on vaja Dockeri tõmmist (*image*). Dockeri tõmmiseid hoitakse Docker Hub'is[1] ja need sisaldavad juhiseid konteinerite loomiseks. Docker Hub on register, kust saab konteineri loomiseks vajalikke tõmmiseid alla laadida või oma ehitatud tõmmiseid üles laadida.

PostgreSQL andmebaasisüsteemi jaoks on Docker Hub'is konteineri käitamiseks vajalik tõmmis olemas, kuid käesoleva töö rakenduste jaoks pidi autor Dockeri tõmmised ise

[1] https://hub.docker.com/

valmis ehitama. Tõmmise ehitamiseks tuleb luua Dockeri fail (*Dockerfile*), mis sisaldab juhiseid tõmmise loomiseks. Suurobjektide disaini jaoks mõeldud rakenduse *Dockerfile* on esitatud Lisas 5.

Dockerfile salvestati rakenduse kataloogi. Dockeri tõmmise ehitamiseks avati *Dockerfile*'ga samas kataloogis käsurida ning anti korraldus tõmmise ehitamiseks. Näidis Dockeri tõmmise ehitamise käsk on esitatud Joonisel 20.

```
docker build -t erlemaido/blob-products .
```

Joonis 20. Näidiskäsk Dockeri tõmmise ehitamiseks.

Tõmmise eduka loomise tagajärjel tuleb ehitatud tõmmis saata Docker Hub'i. Selleks anti samas kataloogis `docker push` käsk, mille näidis on esitatud Joonisel 21.

```
docker push erlemaido/blob-products
```

Joonis 21. Näidiskäsk Dockeri tõmmise saatmiseks Docker Hub'i.

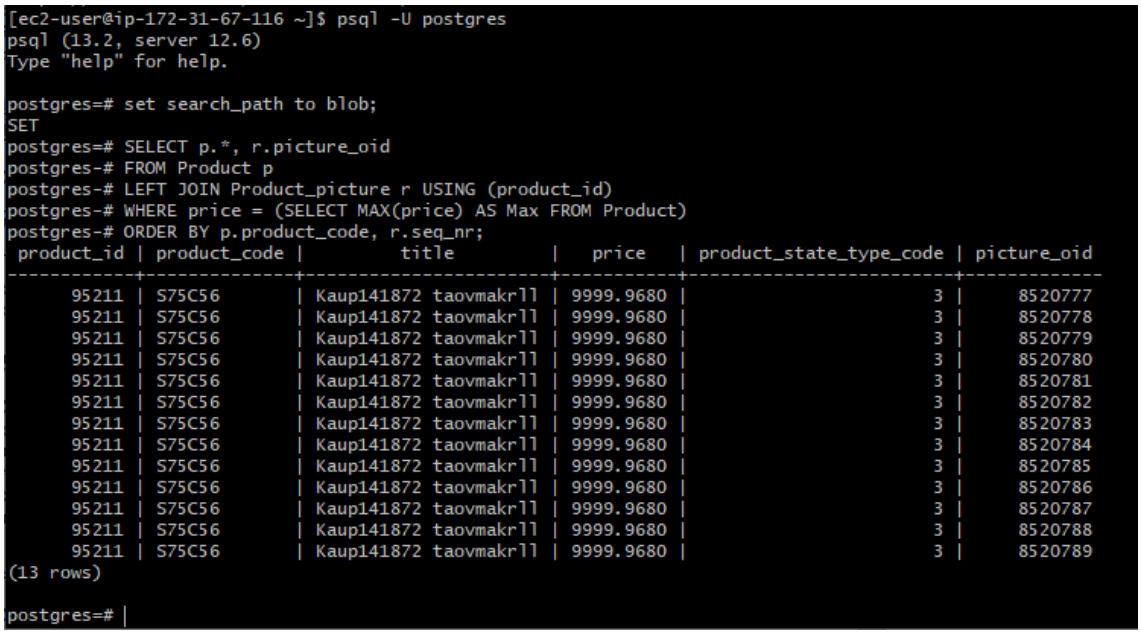
Järgmise sammuna tuleb pilveserveris ehitatud tõmmis käitada. Selleks tuleb siseneda EC2 serverisse. EC2 serverisse sisenemiseks lõi autor privaatse võtme abil läbi käsurea SSH ühenduse. Kõige pealt tuleb serverisse installeerida Docker ning seejärel Docker käima panna. Kui Docker käib, saab esitada käsu `docker run`, millega käitatakse konteiner, kus sees asub rakendus ja määratakse ära port, kuhu rakendus käima pannakse. Serverile ligipääsu aadress on leitav AWS-i lehel vastava serveri andmete juurest. Näidis `docker run` käsk on esitatud Joonisel 22 ning kuvatõmmis pilveserverisse sisenemisest, Dockeri käima panemisest ja konteineri käitamisest on esitatud Lisas 6.

```
docker run -p 5000:5000 erlemaido/blob-products
```

Joonis 22. Näidiskäsk konteineri käitamiseks.

Andmebaasi loomiseks tuleb siseneda eespool kirjeldatud viisil EC2 serverisse ning installeerida PostgreSQL andmebaasisüsteem ning luua andmebaas. Andmebaasile ligipääsemiseks saab kasutada Psql-i, mis on terminalipõhine PostgreSQL kasutajaliides. Andmebaasi sisenemiseks tuleb anda käsk `psql -U postgres`.

Andmebaasi sisenemine ja näidispäring kasutades Psql-i on esitatud Joonisel 23. Päringuga leitakse kõige kallim toode ja kõik selle toote pildid ning sorteeritakse tulemus tootekoodi ja pildi järjekorranumbri järgi kasvavas järjekorras.



```
[ec2-user@ip-172-31-67-116 ~]$ psql -U postgres
psql (13.2, server 12.6)
Type "help" for help.

postgres=# set search_path to blob;
SET
postgres=# SELECT p.*, r.picture_oid
postgres-# FROM Product p
postgres-# LEFT JOIN Product_picture r USING (product_id)
postgres-# WHERE price = (SELECT MAX(price) AS Max FROM Product)
postgres-# ORDER BY p.product_code, r.seq_nr;
 product_id | product_code |         title         |   price   | product_state_type_code | picture_oid
------------+--------------+-----------------------+-----------+-------------------------+-------------
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520777
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520778
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520779
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520780
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520781
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520782
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520783
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520784
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520785
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520786
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520787
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520788
      95211 | S75C56       | Kaup141872 taovmakrll | 9999.9680 |                       3 |     8520789
(13 rows)

postgres=#
```

Joonis 23. PostgreSQL kasutamine Amazon EC2 pilveserveris.

6 Disainilahenduste võrdlemine

Käesolevas peatükis kirjeldatakse disainilahenduste võrdlemiseks tehtavaid mõõtmisi. Põhiküsimused, millele käesolevas töös tehtavate mõõtmistega vastust otsitakse, on järgnevad.

- Milline on tabelite andmemaht erinevate disainimustrite korral?
- Kuidas erinevad andmete otsimise ja ridade lisamiste lausete täitmise kiirused
    - erinevate disainide korral sama andmemahuga?
    - sama disaini korral erinevate andmemahtudega?
- Kuidas erineb rakenduses andmete kuvamiseks kuluv aeg erinevate disainide korral sama andmemahuga?
- Kui suur on rakenduse füüsiliste koodiridade arv erinevate disainide korral?
- Milline rakendus on subjektiivselt hinnates kõige lihtsamini realiseeritav?

6.1 Mõõtmiste kirjeldamine

Disaine võrreldakse järgmiste kriteeriumite alusel.

- Tabelite andmemaht (koos indeksite ja TOAST andmetega).
- Päringute (SELECT lausete) täitmise kiirus.
- Ridade lisamise kiirus.
- Rakenduse füüsiliste koodiridade arv.
- Rakenduse loomise subjektiivne lihtsus.
- Rakenduses andmete kuvamiseks kuluv aeg.

Sama operatsiooni erinevatel käivitamistel võib tekkida täitmiskiiruse erinevusi. Seetõttu käivitatakse kõiki päringuid, andmete lisamise operatsioone ja mõõdetakse rakenduses andmete kuvamiseks kuluvat aega viis korda. Saadud tulemuste põhjal arvutatakse geomeetriline keskmine. Kuna mõõtmiste tulemused on omavahel korrelatsioonis, sest süsteem valmistab korduva operatsiooni täitmise tulemusel ette ja jätab meelde lause täitmisplaani ning loeb muutmällu andmed, mida pole järgnevatel täitmistel enam vaja

teha, siis kasutatakse keskmise tulemuse arvutamiseks just geomeetrilise keskmise arvutamise valemit.

Töökiiruse hindamisel kasutatakse PostgreSQL andmebaasisüsteemis *EXPLAIN ANALYZE* lauset, mille tulemusena väljastatakse SQL lause täitmiseks kasutatud täitmisplaan, plaani koostamiseks kulunud aeg ja lause täitmiseks kulunud aeg. Operatsioonide kiiruste mõõtmisel liidetakse kokku plaani koostamiseks kulunud aeg ja lause täitmiseks kulunud aeg.

Hindamaks andmemahtude mõju päringute ja andmete lisamise operatsioonide kiirustele, viib autor mõõtmised läbi kolme erineva andmehulgaga. Autor alustab mõõtmisi kõige väiksema andmemahuga andmehulgaga, seejärel puhastab andmebaasi ning genereerib järgmise hulgaga testandmed.

6.2 Kasutatava arvutisüsteemi omadused

Autor kasutab PostgreSQL (PostgreSQL 12.6.0) andmebaasisüsteemi. Päringute käivitamiseks ning nende täitmiseks kulunud aja mõõtmiseks kasutab autor Psql terminalipõhist PostgreSQL-i kasutajaliidest.

Eksperimendi käigus kasutab autor *t3.large* tüüpi Amazon EC2 pilveservereid, mille peal on Amazon Linux 2 operatsioonisüsteem, mis toetab XFS failisüsteemi. Andmebaasiserveri ning failisüsteemi lahenduse serveri kõvaketta mahuks määrab autor 100 GiB ($100*1024^3$ baiti) Teiste serverite mahud jätab autor standardse 8 GiB peale. Tabelis 11 on esitatud serverite tehnilised andmed.

Tabel 11. Pilveserverite tehnilised andmed.

	Virtuaalsete keskprotsessorite (CPU-de) arv	**Kettamaht (GiB)**	**Muutmälu (GB)**
Bytea	2	8	8
Suurobjekt	2	8	8
Failisüsteem	2	100	8
Pilvelahendus	2	8	8
PostgreSQL	2	100	8

6.3 Andmemaht

PostgreSQL-is saab tabeli andmemahu leidmiseks kasutada funktsiooni `pg_total_relation_size()`, mis arvutab tabeli andmemahu baitides arvestades ka tabeliga seotud indeksite ja süsteemisiseselt liiga suure andmemahu tõttu tabeli ridadest eraldi salvestatud väärtuste andmetega. Funktsiooni tulemuse loetavuse parendamiseks rakendatakse antud funktsiooni tulemusele veel omakorda `pg_size_pretty()` funktsiooni, mis esitab päringu tulemuse vastavalt sobivusele kas kilo-, mega-, giga-, tera- või lihtsalt baitides [73]. Kogu skeemis olevate tabelite mahu leidmiseks liidab autor tabelite *toode*, *tootepilt* ja *toote seisundi liik* mahud. Suurobjektide skeemi puhul lisatakse juurde ka tabelite *pg_largeobject* ja *pg_largeobject_metadata* mahud. Joonisel 24 on esitatud näidispäring tabeli andmemahu leidmiseks. Ülejäänud andmemahu mõõtmiseks kasutatud päringud on välja toodud Lisas 7.

```
SELECT pg_size_pretty(pg_total_relation_size('product'));
```

Joonis 24. Andmemahu mõõtmise päringu näide.

6.4 Päringute töökiirus

Disainilahenduste võrdlemiseks mõõdab autor viie erineva päringu täitmise kiirust. Päringud on esitatud Tabelis 12. Esimene ja teine päring on erinevate disainilahenduste korral natukene erinevad. Tabelis on välja toodud failisüsteemi ja pilvelahenduse disaini päringud. Kõikide disainide mõõtmiste käigus tehtud päringud on esitatud Lisas 8.

Päringuga 1 (P1) leitakse kõige kallimad tooted ja kõik nende toodete pildid ning tulemus sorteeritakse tootekoodi ja pildi järjekorranumbri järgi kasvavas järjekorras. Kui tootel pilte pole, siis väljastatakse see ikkagi, kuid ilma pildita.

Päringuga 2 (P2) leitakse kõik aktiivsed tooted ehk tooted, mille toote seisundi liik on aktiivne. Päringu tulemusena tuuakse välja toote id, tootekood, toote nimetus ja toote kõige väiksema järjekorranumbriga pilt. Tulemus sorteeritakse tootekoodi järgi kasvavas järjekorras. Kui tootel pilte pole, siis väljastatakse see ikkagi, kuid ilma pildita.

Päringuga 3 (P3) leitakse kõik tooted. Päringu tulemusena esitatakse toote id, tootekood, toote nimetus, toote seisundi liigi nimetus ja lisaks tuuakse välja toote piltide arv. Juhul, kui tootel pilte ei ole, märgitakse toote piltide arvuks 0. Tulemus sorteeritakse tootekoodi järgi kasvavas järjekorras.

Päringuga 4 (P4) leitakse kõige suurema piltide arvuga toodete identifikaatorid.

Päringuga 5 (P5) leitakse tootepiltide koguarv.

Tabel 12. Disainilahenduse võrdlemiseks kasutatavad päringud.

	Päring

P1	`SELECT p.*, r.picture_url`
	`FROM Product p`
	`LEFT JOIN Product_picture r USING (product_id)`
	`WHERE price = (SELECT MAX(price) AS Max FROM Product)`
	`ORDER BY p.product_code, r.seq_nr;`

P2	`SELECT p.product_id, p.product_code, p.title, r.picture_url`
	`FROM Product p`
	`LEFT JOIN Product_picture r USING (product_id)`
	`WHERE p.product_state_type_code = 2`
	`AND (r.seq_nr = (`
	`SELECT Min(seq_nr) AS mi`
	`FROM Product_picture AS min_r`
	`WHERE r.product_id = min_r.product_id)`
	`OR r.seq_nr IS NULL)`
	`ORDER BY p.product_code;`

P3	`SELECT p.product_id, p.product_code, p.title,`
	`s.title AS product_state_type_title,`
	`Count(r.product_id) AS nr_of_pictures`
	`FROM Product_state_type s`
	`INNER JOIN Product p USING(product_state_type_code)`
	`LEFT JOIN Product_picture r USING (product_id)`
	`GROUP BY p.product_id, p.product_code, p.title, s.title`
	`ORDER BY p.product_code;`

P4	`SELECT product_id`
	`FROM Product_picture`
	`GROUP BY product_id`
	`HAVING Count(*)>=ALL (`
	`SELECT Count(*) AS Num FROM Product_picture`
	`GROUP BY product_id);`

P5	`SELECT Count(*) AS num FROM Product_picture;`

6.5 Rea lisamise kiirus

Disainilahenduste võrdlemiseks mõõdab autor tabelisse *tootepilt* ühe rea lisamise operatsiooni töökiirust. *Bytea* lahenduse puhul kasutab autor selleks PostgreSQL-i `pg_read_binary_file()` funktsiooni. Antud funktsioon nõuab, et laaditavad pildid asuvad kataloogis, kus asuvad PostgreSQL-i andmefailid. Suurobjekti lahenduse puhul kasutab autor rea lisamiseks `lo_import()` funktsiooni. Näidisoperatsioonid on esitatud Joonisel

25. Failisüsteemi ja pilvelahenduse puhul ei ole võimalik INSERT lauset kasutades lisada uut rida tabelisse tootepilt, mõõtes samal ajal nii päringu kui ka pildi vastavalt failisüsteemi või pilve üles laadimise kiirust. Samuti ei olnud võimalik mõõta ka eraldi päringu kiirust ja faili üles laadimise kiirust ning neid seejärel summeerida. Realiseeritud rakendus võimaldab tootepilti üles laadida vaid koos tootega. Seetõttu toob tootele pildi lisamine kaasa ka toote objekti loomise või muutmise, mis mõjutab ka faili üles laadimiseks mõõdetavat aega. Kuna autor ei leidnud head alternatiivi rea lisamise töökiiruse mõõtmiseks pilve ja failisüsteemi lahenduse puhul, siis jäi antud operatsioon nende disainide puhul mõõtmata.

```
INSERT INTO Product_picture(product_id, seq_nr, picture) VALUES (
1, 1, pg_read_binary_file('/var/lib/pgsql/12/data/pic.jpg')::bytea);

INSERT INTO Product_picture(product_id, seq_nr, picture_oid) VALUES (
1, 1, lo_import('/var/lib/pgsql/12/data/pic.jpg'));
```

Joonis 25. Disainilahenduste võrdlemiseks kasutatavad ridade lisamise operatsioonid.

6.6 Rakenduse füüsiliste koodiridade arv

Erinevate disainilahenduste realiseerimisel on programmikoodi ridade arv erinev. Koodiridade arv kirjeldab tarkvara lähtekoodi suurust ja keerukust. Käesolevas töös uuritakse ainult piltide salvestamisega seotud füüsiliselt käivitatavaid ridu ehk tühje ridu ja kommentaare ei arvestata. Toote koos tootepiltidega lisamise meetod on kõikidel disainidel samasugune ja pildi salvestamiseks kasutatakse antud meetodi sees disainile vastavat piltide salvestamise meetodit. Seetõttu leitakse ainult pildi salvestamise meetodis sisalduvad füüsiliselt käivitatavad read, mitte kogu toote ja tootepildi salvestamiseks vajaminevad read. Autor loendab iga disainilahenduse korral vastavad koodiread. Selleks, et tulemused oleksid võrreldavad, vormistatakse kood ühesuguseid põhimõtteid arvestades, järgides C# programmeerimiskeele tavasid ja standardset stiili [74], [75].

6.7 Rakenduse loomise subjektiivne lihtsus

Võib üsna kindlalt eeldada, et mõne andmebaasi puhul on rakenduse loomine lihtsam kui teise puhul. Erinevate disainilahenduste kasutamise tulemuseks võib olla nii erinev koodiridade kui ka meetodite või klasside hulk. Lisaks mängib andmebaasi disainilahendustel põhinevate rakenduste realiseerimise lihtsuses rolli ka arendaja

varasem kokkupuude antud disainiga. Tuttava disaini realiseerimine võib võtta vähem aega.

Autor üritab võimalikult objektiivselt anda oma hinnangu erinevate disainilahenduste realiseerimise kohta, võttes arvesse ka võimalikke vajaminevaid edasiarendusi, sest piltide laadimise ja kuvamise funktsionaalsused võib lihtsasti saavutada, kuid mõned disainimustrid nõuavad ka edasisi arendusi nagu näiteks eraldi varundamise ja/või kustutamise strateegiate realiseerimine. Autor analüüsib subjektiivselt rakenduse loomise lihtsust ning esitab disainimustrite paremusjärjestuse.

6.8 Rakenduse andmete kuvamiseks kuluv aeg

Disainilahenduste võrdlemiseks mõõdab autor rakenduse andmete kuvamiseks kuluvat aega. Aega mõõdetakse avalehe kuvamisel ning ühe, kümne ja kahekümne pildiga toote detailvaate kuvamisel erinevate andmemahtude korral. Mõõtmiseks kasutab autor GTmetrix veebirakendust [76], millega saab mõõta veebilehe laadimise kiirust. Nimetatud veebirakendus kasutab mõõtmiseks Google Chrome veebilehitsejat.

7 Mõõtmiste tulemused

Käesolevas peatükis toob autor välja andmemahtude, päringute ja ridade lisamise kiiruste, ning rakenduse andmete kuvamiseks kuluva aja mõõtmiste tulemused erinevate disainide korral ja esitab rakenduse füüsiliste koodiridade arvu. Tulemuste analüüs ja järeldused esitatakse järgmises peatükis (peatükk 8). Pilvelahenduse puhul viidi mõõtmised läbi kahe erineva disaini korral. S3 a esitab mõõtmised disaini kohta, kus pildiaadress on andmebaasi salvestatud täispika URL-ina. S3 b disainis hoitakse andmebaasis vaid pildi nime.

Tabelis 13 esitatakse andmebaasi maht erinevate andmehulkade ja disainide korral. Mahud on gigabaitides ja arvutatud koos tabelitega seotud indeksitega ning TOAST andmetega. Tabeli esimeses veerus on välja toodud ridade arv tabelis *toode*. Kuna igale tootele genereeriti pildid juhuslikult, siis on tootepiltide arv muutuv suurus. Genereeritud tootepiltide arv esitati Tabelis 10.

Tabel 13. Andmebaasi maht erinevate disainide korral (GB).

***Tooted* ridade arv**	***Bytea***	**Suurobjekt**	**Failisüsteem**	**S3 a**	**S3 b**
10 000	1,825	2,131	0,014	0,155	0,014
40 000	7,230	10,076	0,064	0,621	0,060
160 000	28,019	36,274	0,255	2,475	0,248

Tabelis 14 esitatakse päringute täitmise kiiruste mõõtmistulemuste geomeetrilised keskmised. Mõõtmistulemused on sekundites. Päringu koodile vastav SQL laused esitati Tabelis 12 ja Lisas 8. Rasvaselt tähistatakse tulemused, mis on sama andmemahu ja lähteülesande korral parim (väikseim). Mõne päringu korral on parimaks mitu disainimustri tulemust.

Tabelis 15 on esitatud tabelisse *tootepilt* ridade lisamise operatsiooni töökiiruse mõõtmistulemuste geomeetrilised keskmised *bytea* ja suurobjekti disaini korral.

Mõõtmistulemused esitatakse millisekundites. Rasvaselt tähistatakse tulemused, mis on sama andmemahu korral parim (väikseim).

Tabel 14. Päringute täitmise kiiruste geomeetrilised keskmised (sekundites).

Päring	***Tooted* ridade arv**	***Bytea***	**Suurobjekt**	**Faili-süsteem**	**S3 a**	**S3 b**
P1	10 000	0,005	**0,004**	**0,004**	**0,004**	**0,004**
	40 000	0,018	0,017	**0,016**	**0,016**	**0,016**
	160 000	0,074	**0,062**	0,066	0,065	0,072
P2	10 000	0,560	**0,112**	0,117	0,176	0,121
	40 000	2,177	0,463	0,506	0,704	**0,453**
	160 000	7,569	1,778	1,786	2,455	**1,746**
P3	10 000	0,163	0,155	**0,148**	0,170	0,150
	40 000	0,640	**0,561**	0,622	0,622	0,625
	160 000	2,292	**2,186**	2,265	2,204	2,301
P4	10 000	0,725	0,697	0,622	0,762	**0,603**
	40 000	9,312	8,094	8,349	7,783	**7,746**
	160 000	182,214	164,377	163,147	164,306	**162,023**
P5	10 000	0,015	**0,014**	**0,014**	0,017	**0,014**
	40 000	0,059	**0,058**	0,063	0,061	0,061
	160 000	0,212	**0,208**	0,225	0,231	0,216

Tabel 15. Ridade lisamise kiiruste geomeetrilised keskmised (millisekundites).

***Tooted* ridade arv**	***Bytea***	**Suurobjekt**
10 000	0,825	**0,541**
40 000	0,781	**0,614**
160 000	0,836	**0,622**

Tabel 16 esitab lehekülgede laadimisaja geomeetrilised keskmised kiirused sekundites. Mõõdetava lehekülje kirjeldus on esitatud esimeses veerus. Rasvaselt tähistatakse tulemused, mis on sama andmemahu ja lähteülesande korral parim (väikseim).

Tabel 16. Lehekülgede laadimisaja geomeetrilised keskmised kiirused (sekundites).

Mõõdetav operatsioon	***Tooted* ridade arv**	***Bytea***	**Suurobjekt**	**Faili-süsteem**	**S3 a**	**S3 b**
Avalehe laadimine	10 000	0,954	1,126	0,794	0,691	**0,678**
	40 000	0,984	1,102	0,756	0,678	**0,671**
	160 000	0,991	1,015	0,760	0,695	**0,667**
1 pildiga toote laadimine	10 000	0,750	0,762	0,777	**0,747**	0,750
	40 000	0,747	0,769	0,784	0,744	**0,740**
	160 000	0,748	0,768	0,745	0,782	**0,735**
10 pildiga toote laadimine	10 000	1,060	1,056	0,866	0,778	**0,755**
	40 000	1,042	1,094	0,810	**0,737**	0,742
	160 000	1,056	1,100	0,819	0,794	**0,727**
20 pildiga toote laadimine	10 000	1,183	1,275	1,018	**0,867**	**0,867**
	40 000	1,180	1,246	0,995	0,909	**0,868**
	160 000	1,182	1,256	1,038	0,876	**0,804**

Järgnevalt esitatakse Tabelis 17 tarkvara lähtekoodi füüsiliselt käivitavate koodiridade arv piltide salvestamiseks.

Tabel 17. Füüsiliste koodiridade arv erinevate disainide korral.

	Bytea	**Suurobjekt**	**Failisüsteem**	**S3**
Füüsiliste koodiridade arv	10	41	13	23

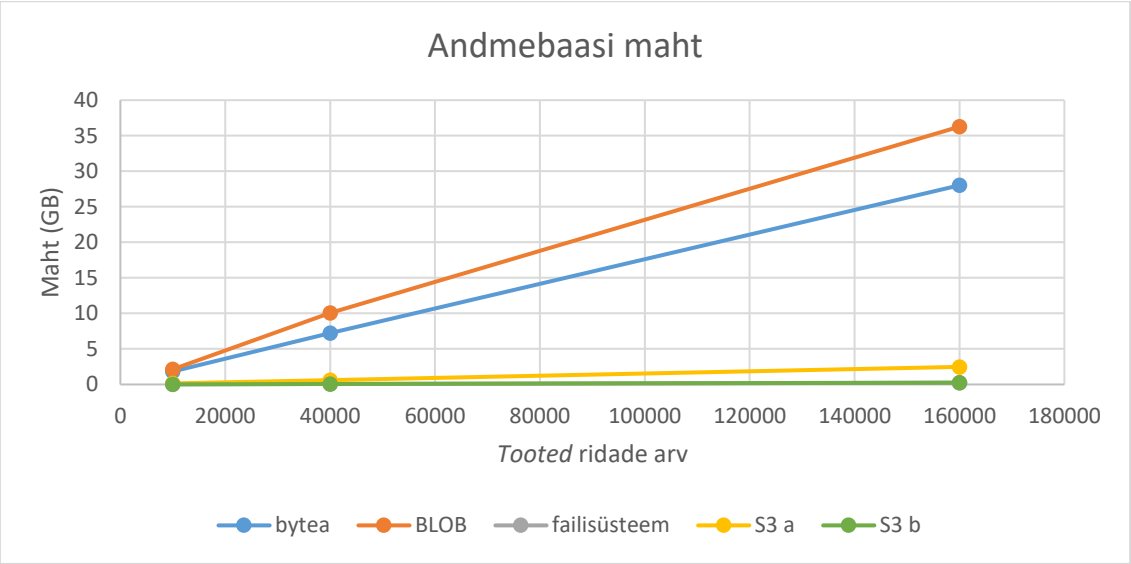
8 Tulemuste analüüs ja järeldused

Käesoleva bakalaureusetöö eesmärgiks on süstematiseerida infot SQL-andmebaaside disainilahenduste kohta, mis on mõeldud toetama failide salvestamist vajavaid andmebaasirakendusi. Disainilahenduste võrdlemiseks viis autor läbi erinevaid mõõtmisi. Antud peatükis analüüsitakse disainide võrdlemiseks tehtud mõõtmiste tulemusi ja tehakse järeldusi. Lisaks tuuakse välja töö võimalikud nõrkused.

Toodete ridade arvu lineaarset sõltuvust erinevate tunnuste vahel mõõdetakse Pearsoni korrelatsioonikordajaga. Kui kahe tunnuse vahel esineb tugev seos, siis on mõõtmistulemused sirgjoonega hästi kokkuvõetavad. Sellisel juhul on korrelatsioonikordaja absoluutväärtus kõrge, lähenemas 1-le. Korrelatsioonikordaja väärtused asuvad vahemikus -1 ja 1. Kordaja positiivne väärtus tähendab kasvavat seost tunnuste vahel ehk kui ühe tunnuse väärtus on suur, on ka tavaliselt teise tunnuse väärtus suur. Korrelatsioonikordaja negatiivne väärtus tähendab tunnuste vahel kahanevat seost ehk ühe tunnuse suure väärtusega käib enamasti kaasas teise tunnuse väike väärtus. Kui korrelatsioonikordaja on 0, siis puudub tunnuste vahel lineaarne seos [77]. Pearsoni korrelatsioonikordaja leidmiseks kasutab autor vastavat kalkulaatorit [78].

8.1 Andmebaasi salvestamiseks kulunud ruum

Andmebaasi mahu mõõtmistulemuste uurimisel näeb selgeid erinevusi faile andmebaasis hoidvate disainide ja andmebaasis faili viidet hoidvate disainide vahel. Faile andmebaasis hoidvad disainid, *bytea* ja *suurobjekti* disain, kasutavad kordades rohkem andmebaasi salvestusruumi, kui *failisüsteemi* ja *pilvelahenduse* disainid ning muudavad andmebaasi mahukaks. *Pilvelahenduse* disain (*S3 b*), kus hoitakse andmebaasis vaid failinime, võtab 160 000 toote korral ligikaudu 146 korda vähem ruumi kui *suurobjekti* disain, vastavalt 0.248 GB ja 36.274 GB. Joonisel 26 on esitatud andmebaasi mahu sõltuvus toodete ridade arvust erinevate disainide puhul. Ka pilvelahenduse disainide puhul on kasutatav andmemaht erinev. *S3 a* disain, kus hoitakse andmebaasis tervet failile viitavat aadressi, võtab ligikaudu 10 korda rohkem ruumi, kui *S3 b* disain. 160 000 toote rea korral on need arvud vastavalt 2.475 GB ja 0.248 GB.



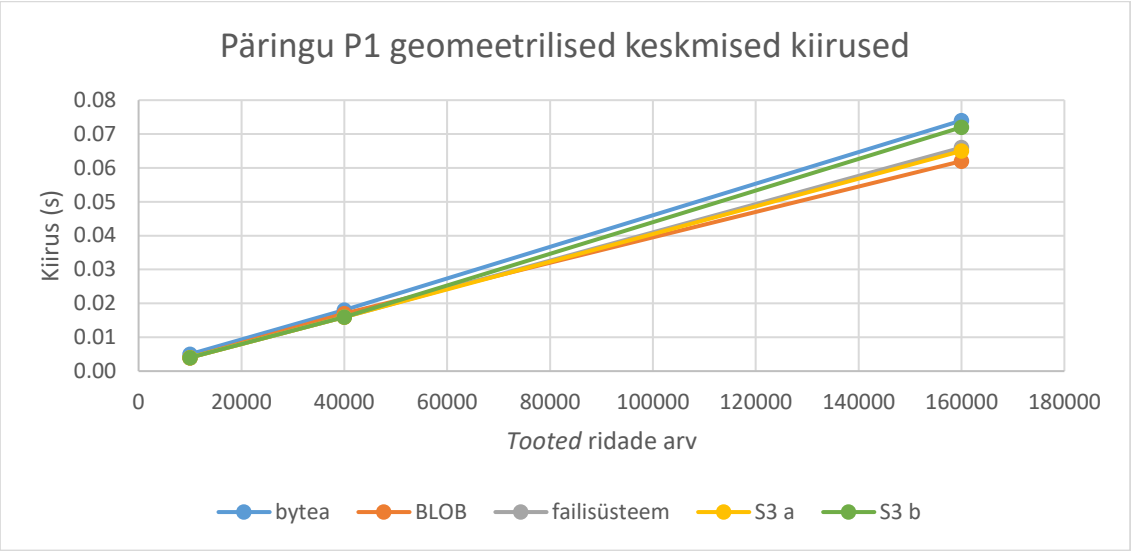
Joonis 26. Andmebaasi salvestamiseks kulunud ruum.

Jooniselt on näha, et kõikide disainide puhul on sõltuvus andmebaasi mahu ja toodete ridade arvu vahel lineaarne. Selle kinnitamiseks leidis autor Pearsoni korrelatsiooni koefitsendid. Arvutustulemused kinnitasid kõikide disainide puhul väga tugevat positiivset lineaarset korrelatsiooni andmebaasi mahu ja toodete ridade arvu vahel. *Bytea*, *failisüsteemi* ja *pilvelahenduse* disainide puhul oli korrelatsioonikordaja 1 ning *suurobjekti* disaini puhul 0.9994.

8.2 Päringute kiirused

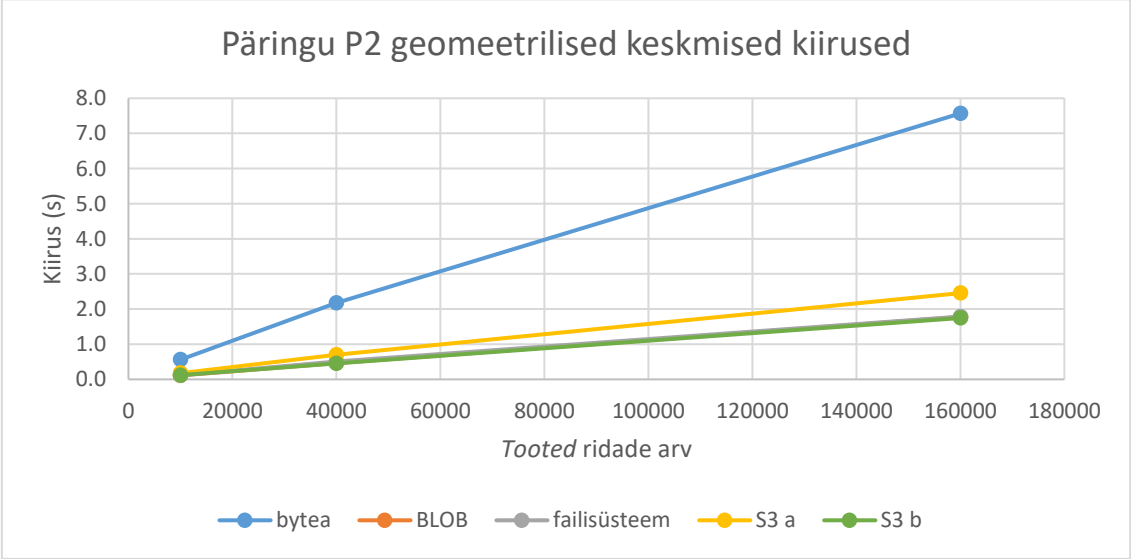
Joonis 27 kirjeldab esimese päringu (P1) kiiruse sõltuvust toodete ridade arvust. Päringu P1 korral on päringu kiirused erinevate disainilahenduste ja samade andmehulkade korral ligikaudu võrdsed. Kõige suurem erinevus (12 ms) on *bytea* ja *suurobjekti* lahenduse puhul suurima toodete ridade arvu korral, vastavalt 0,074 s ja 0,062 s. Kõikide disainilahenduse ja andmehulkade korral jäävad P1 päringu täitmise kiirused alla 0,1 sekundi.

Jooniselt on näha lineaarne sõltuvus päringu kiiruse ja toodete ridade arvu vahel. Seda kinnitavad ka arvutatud Pearsoni korrelatsiooni koefitsendid, mis on *bytea* disaini puhul 0.9999, *suurobjekti* ja *S3 b* disainide puhul 0.9997 ning *failisüsteemi* ja *S3 a* disainide puhul 1. Seega on päringu tegemiseks kuluva aja ja toodete ridade arvu vahel väga tugev positiivne korrelatsioon.



Joonis 27. Päringu P1 geomeetrilised keskmised kiirused.

Joonis 28 kirjeldab teise päringu (P2) kiiruse sõltuvust toodete ridade arvust. Päringu P2 korral on *bytea* disain kõikide andmemahtude korral teistest tunduvalt aeglasem, 160 000 toote rea korral on päringu kiirus 7,569 s. *Suurobjekti*, *failisüsteemi* ja *S3 b* disainide päringu kiirused on omavahel ligikaudu võrdsed kõikide andmehulkade korral ning umbes 5 korda kiiremad *bytea* disainiga tehtud päringutest. *Bytea* ja eelnevalt nimetatud kolme disaini ligikaudne päringu kiiruste vahe 160 000 toote rea korral on 5,8 s. Ka *S3 a* disain on eelnevalt nimetatud kolmest disainist natuke aeglasem, 160 000 toote rea korral on päringu kiiruste vahe ligikaudu 0,7 s.

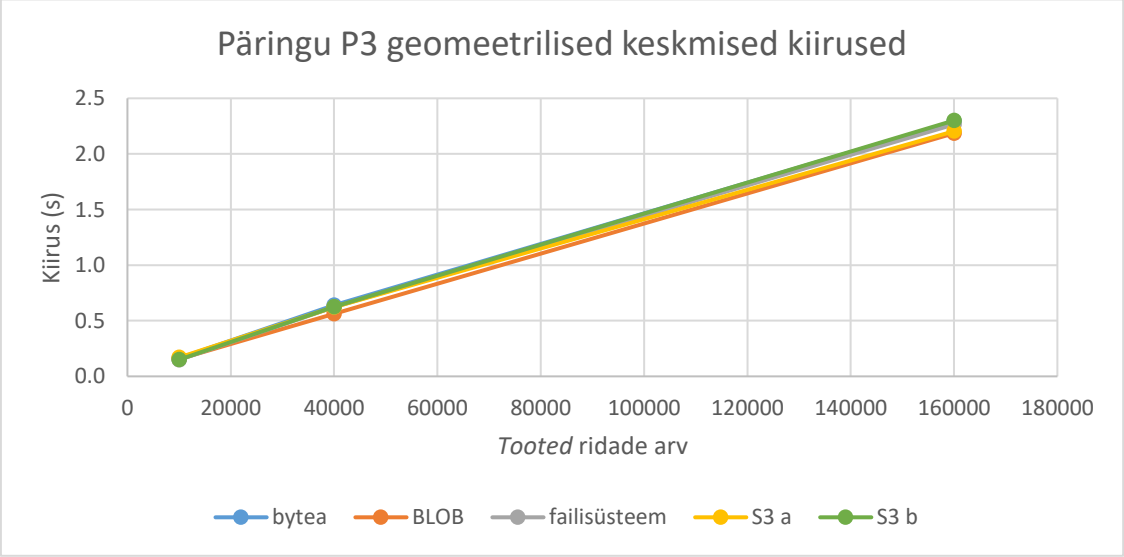


Joonis 28. Päringu P2 geomeetrilised keskmised kiirused.

Jooniselt paistab kõikide disainide puhul välja lineaarne sõltuvus päringu kiiruse ja toodete ridade arvu vahel, mida kinnitavad ka Pearsoni korrelatsioonikordaja arvutustulemused. Korrelatsiooni koefitsent on *bytea* ja *S3 a* disainide puhul 0.9995, *suurobjekti* disaini puhul 0.9999, *failisüsteemi* disaini puhul 0.9994 ja *S3 b* disaini puhul 1 ehk kõikide disainide korral on päringu kiiruse ja toodete ridade arvu vahel väga tugev positiviine lineaarne korrelatsioon.

Joonis 29 esitab kolmanda päringu (P3) kiiruse sõltuvuse toodete ridade arvust. P3 päringu puhul on geomeetrilised keskmised kiirused kõikide disainide korral samade andmemahtude lõikes ligikaudu samad. Kõige suurem erinevus (115 ms) esines *suurobjekti* ja *S3 b* disaini vahel 160 000 toote rea arvu korral, kus kiirused olid vastavalt 2,186 s ja 2,301 s.

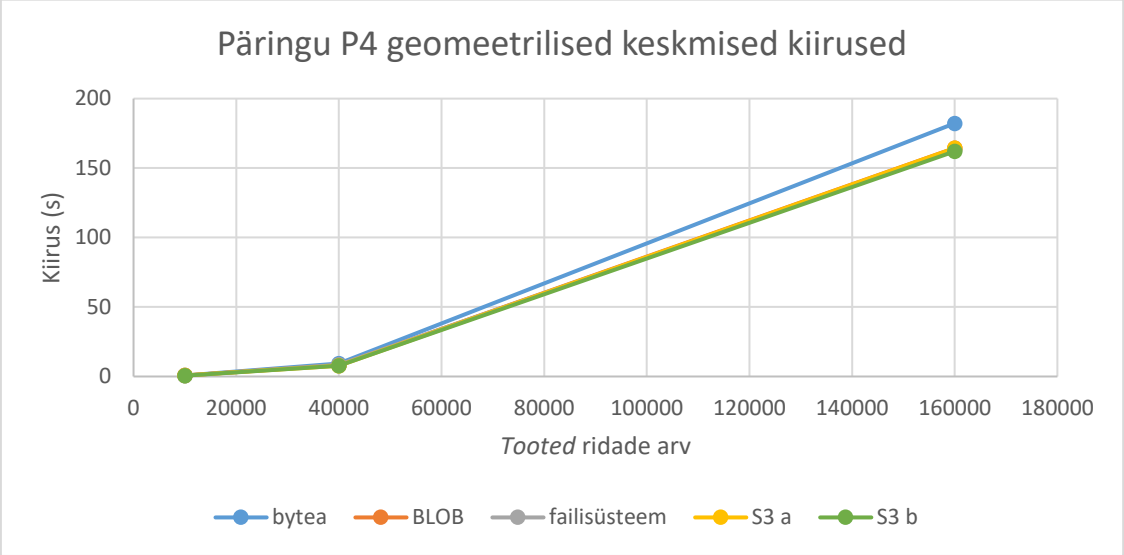
Ka kolmanda päringu korral on kõikide disainide korral päringu kiiruse ja toodete ridade arvu vahel tugev seos. Pearsoni korrelatsiooni koefitsendid näitavad väga tugevat positiivset lineaarset korrelatsiooni, mille väärtused on *bytea*, *failisüsteemi* ja *S3 a* disainide puhul 0,9997, *suurobjekti* disaini puhul 1 ja *S3 b* disaini puhul 0,9998.



Joonis 29. Päringu P3 geomeetrilised keskmised kiirused.

Neljanda päringu (P4) puhul on taaskord päringu kiiruste mõõtmistulemused väga sarnased. *Bytea* disaini päringu kiirus on teiste disainide päringu kiirustest andmemahu suurenemisel natukene väiksem. Kõige suurem erinevus (20,191 s) on *bytea* ja *S3 b* disainide vahel 160 000 toote rea arvu puhul, kus päringu kiirused on vastavalt 182,214 s

ja 162,023 s. Päringu P4 kiiruse sõltuvust toodete ridade arvust esitab Joonis 30. Jooniselt on näha kõikide disainide puhul tõusu järsenemine peale 40 000 toote rida.

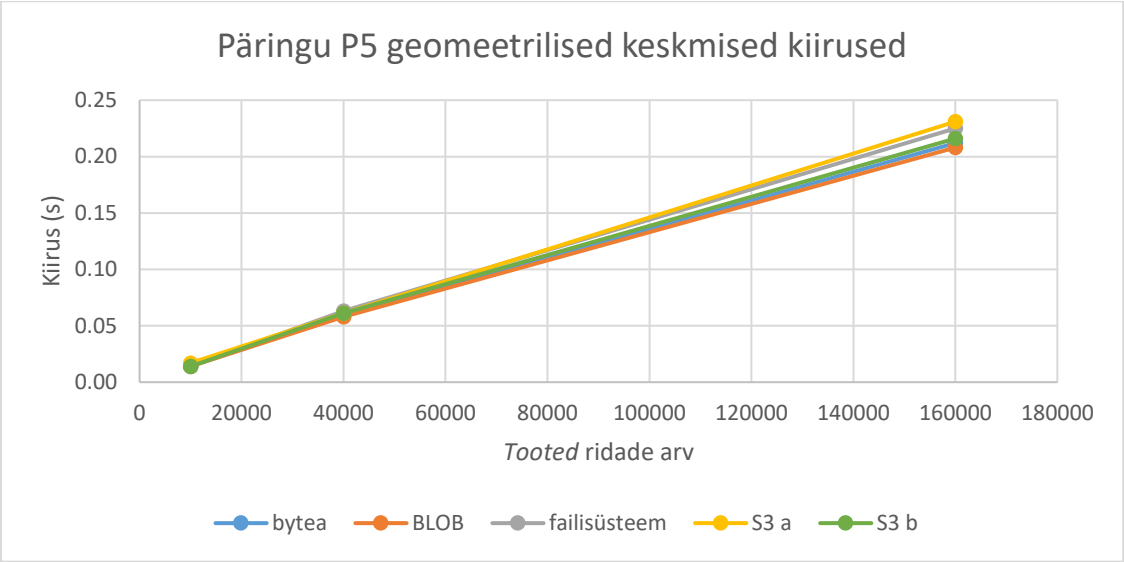


Joonis 30. Päringu P4 geomeetrilised keskmised kiirused.

Ka P4 päringu korral on päringu kiiruse ja toodete ridade arvu vahel positiivne lineaarne korrelatsioon. Arvutatud korrelatsioonikordajad on *bytea* disaini puhul 0,9890, *suurobjekti* disaini puhul 0,9888, *failisüsteemi* disaini puhul 0,9891, *S3 a* disaini puhul 0,9884 ja S3 b disaini puhul 0,9886.

Joonis 31 esitab viienda päringu (P5) kiiruse sõltuvuse toodete ridade arvust. P5 päringu puhul on kiirused kõikide disainide korral samade andmemahtude lõikes ligikaudselt samad. Kõige suurem erinevus (23 ms) on *suurobjekti* ja *S3 a* disaini vahel 160 000 toote rea korral. Kõikide disainilahenduse ja andmehulkade korral jäävad P5 päringu kiirused alla 0,3 sekundi.

Pearsoni korrelatsioonikordaja arvutustulemused kinnitavad jooniselt nähtavat väga tugevat positiivset lineaarset sõltuvust päringu kiiruse ja toote ridade arvu vahel kõikide disainide puhul. Korrelatsioonikordaja on *bytea* disaini puhul 0,9997, *suurobjekti* disaini puhul 0.9996, *failisüsteemi* ja *S3 b* disainide puhul 0.9994 ja *S3 a* disaini puhul 1.



Joonis 31. Päringu P5 geomeetrilised keskmised kiirused.

Kõikide mõõdetud päringute kiiruste ja toote ridade arvu vahel esines tugev positiivne lineaarne korrelatsioon. Päringute kiirused ei omanud erinevate disainilahenduste lõikes samade andmemahtude korral väga suuri erinevusi. Ainukene märkimisväärne erinevus oli P2 päringu puhul *bytea* disaini korral, kus päringu kiirus erines teiste disainide päringu kiirustest ligikaudu 5 korda. Antud päringuga tulemusena väljastati palju tooteid ja lisaks toote andmetele esitati päringu väljade hulgas ka tootepilt. Kuna *bytea* disaini puhul tuleb väljastatavad binaarandmed eelnevalt teisendada sobivasse formaati, siis võib olla just tootepildi väljastamine põhjus *bytea* disaini antud päringu aeglaseks kiiruseks. Seda kinnitab täitmisplaanide võrdlus *bytea* ja *suurobjekti* disaini puhul P2 korral kui andmebaasis oli 160 000 toote andmed (Joonis 32, Joonis 33).

Joonistelt on näha, et täitmisplaanid olid mõlemal juhul samasugused, kuid *bytea* disaini puhul kulus märkimisväärselt rohkem aega lõpptulemuse formeerimiseks (välisühendamise operatsioon). Kuna täitmisplaani puhul on allpool esitatud operatsioonide tulemus sisendiks ülevalpool olevatele operatsioonidele, siis tuleb plaani lugeda alt üles. Kõige viimasena läbiviidavad operatsioonid on kõige üleval. *Exclusive* veerus on näha operatsiooni täitmiseks kulunud aeg ning *inclusive* veerus operatsiooni ja selle alamoperatsioonide täitmiseks kulunud aeg [79]. Plaanide visuaalse esituse parandamiseks kasutati täitmisplaanide visualiseerimise tarkvara [80].

#	exclusive	inclusive	rows x	rows	loops	node
1.	88.067	6,673.308	↓ 20.2	39,943	1	→ Sort (cost=35,252.33..35,257.28 rows=1,979 width=63) (actual time=6,665.433..6,673.308 rows=39,943 loops=1) Sort Key: p.product_code Sort Method: external merge Disk: 2,792kB
2.	5,155.605	6,585.241	↓ 20.2	39,943	1	→ Hash Right Join (cost=3,169.37..35,143.98 rows=1,979 width=63) (actual time=52.194..6,585.241 rows=39,943 loops=1) Hash Cond: (r.product_id = p.product_id) Filter: ((r.seq_nr = (SubPlan 2)) OR (r.seq_nr IS NULL)) Rows Removed by Filter: 359,516
3.	179.348	179.348	↑ 1.0	1,597,931	1	→ Seq Scan on product_picture r (cost=0.00..27,732.31 rows=1,597,931 width=24) (actual time=0.010..179.348 rows=1,597,931 loops=1)
4.	18.125	51.911	↓ 1.0	39,943	1	→ Hash (cost=2,673.97..2,673.97 rows=39,632 width=31) (actual time=51.910..51.911 rows=39,943 loops=1) Buckets: 65,536 Batches: 1 Memory Usage: 2,998kB
5.	31.199	33.786	↓ 1.0	39,943	1	→ Bitmap Heap Scan on product p (cost=723.57..2,673.97 rows=39,632 width=31) (actual time=2.742..33.786 rows=39,943 loops=1) Recheck Cond: (product_state_type_code = 2) Heap Blocks: exact=1,455
6.	2.587	2.587	↓ 1.0	39,943	1	→ Bitmap Index Scan on "IX_product_product_state_type_code" (cost=0.00..713.66 rows=39,632 width=0) (actual time=2.587..2.587 rows=39,943 loops=1) Index Cond: (product_state_type_code = 2)
7.						SubPlan (for Hash Right Join)
8.	399.459	1,198.377	↑ 1.0	1	399,459	→ Result (cost=3.76..3.77 rows=1 width=2) (actual time=0.003..0.003 rows=1 loops=399,459)
9.						Initplan (for Result)
10.	0.000	798.918	↑ 1.0	1	399,459	→ Limit (cost=0.43..3.76 rows=1 width=2) (actual time=0.002..0.002 rows=1 loops=399,459)
11.	798.918	798.918	↑ 14.0	1	399,459	→ Index Only Scan using "IX_product_picture_product_id_seq_nr" on product_picture min_r (cost=0.43..47.08 rows=14 width=2) (actual time=0.002..0.002 rows=1 loops=399,459) Index Cond: ((product_id = r.product_id) AND (seq_nr IS NOT NULL)) Heap Fetches: 397,547

Planning time	:	0.444 ms
Execution time	:	6,675.299 ms

Joonis 32. P2 päringu täitmisplaan 160 000 toote korral *bytea* disaini andmebaasis.

#	exclusive	inclusive	rows x	rows	loops	node
1.	73.314	1,767.881	↓ 19.8	40,092	1	→ Sort (cost=34,085.84..34,090.90 rows=2,025 width=35) (actual time=1,761.979..1,767.881 rows=40,092 loops=1) Sort Key: p.product_code Sort Method: quicksort Memory: 4,078kB
2.	248.140	1,694.567	↓ 19.8	40,092	1	→ Hash Right Join (cost=3,603.60..33,974.63 rows=2,025 width=35) (actual time=43.836..1,694.567 rows=40,092 loops=1) Hash Cond: (r.product_id = p.product_id) Filter: ((r.seq_nr = (SubPlan 2)) OR (r.seq_nr IS NULL)) Rows Removed by Filter: 362,053
3.	196.357	196.357	↑ 1.0	1,597,465	1	→ Seq Scan on product_picture r (cost=0.00..26,150.65 rows=1,597,465 width=10) (actual time=0.046..196.357 rows=1,597,465 loops=1)
4.	21.443	43.635	↑ 1.0	40,092	1	→ Hash (cost=3,096.66..3,096.66 rows=40,555 width=31) (actual time=43.634..43.635 rows=40,092 loops=1) Buckets: 65,536 Batches: 1 Memory Usage: 3,002kB
5.	20.480	22.192	↑ 1.0	40,092	1	→ Bitmap Heap Scan on product p (cost=1,154.72..3,096.66 rows=40,555 width=31) (actual time=2.010..22.192 rows=40,092 loops=1) Recheck Cond: (product_state_type_code = 2) Heap Blocks: exact=1,435
6.	1.712	1.712	↑ 1.0	40,093	1	→ Bitmap Index Scan on "IX_product_product_state_type_code" (cost=0.00..1,144.58 rows=40,555 width=0) (actual time=1.712..1.712 rows=40,093 loops=1) Index Cond: (product_state_type_code = 2)
7.						SubPlan (for Hash Right Join)
8.	402.145	1,206.435	↑ 1.0	1	402,145	→ Result (cost=2.68..2.69 rows=1 width=2) (actual time=0.003..0.003 rows=1 loops=402,145)
9.						Initplan (for Result)
10.	0.000	804.290	↑ 1.0	1	402,145	→ Limit (cost=0.43..2.68 rows=1 width=2) (actual time=0.002..0.002 rows=1 loops=402,145)
11.	804.290	804.290	↑ 13.0	1	402,145	→ Index Only Scan using "IX_product_picture_product_id_seq_nr" on product_picture min_r (cost=0.43..29.71 rows=13 width=2) (actual time=0.002..0.002 rows=1 loops=402,145) Index Cond: ((product_id = r.product_id) AND (seq_nr IS NOT NULL)) Heap Fetches: 400,219

Planning time	:	0.254 ms
Execution time	:	1,769.707 ms

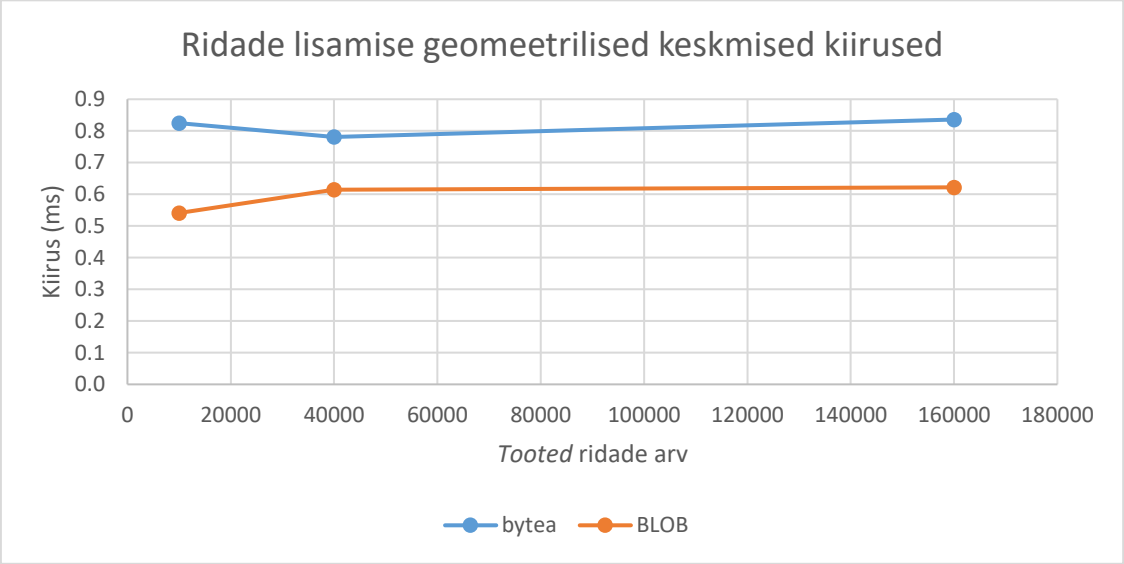
Joonis 33. P2 päringu täitmisplaan 160 000 toote korral suurobjekti disaini andmebaasis.

Ka pilvelahenduse disainide puhul esinesid teise päringu korral kiiruse erinevused. Võib järeldada, et pildi viite pikkus mõjutab antud päringu kiirust. Siiski jäid päringu kiiruste erinevus kahe disaini vahel alla ühe sekundi, ehk erinevus oli pigem väga väike.

Ülejäänud päringute kiiruste mõõtmistel olid tulemused enam-vähem samad. Sellest võib järeldada, et päringu kiiruses ei mängi rolli, kas väljastada numbrilisel kujul (pildi) objekti identifikaator või tekstilisel kujul viide pildile. Ka päringutes, kus ei väljastatud tootepilti, oli *bytea* disaini päringute kiirused ligilähedased teiste disainide päringute kiirustega.

8.3 Rea lisamise kiirus

Joonis 34 esitab ühe rea lisamise kiiruse sõltuvuse toodete ridade arvust. Jooniselt on näha, et toodete ridade arvu suurenemine ei mõjuta ridade lisamise kiirust, vaid kiirus püsib pigem ühtlasena. *Bytea* disaini puhul on rea lisamise geomeetriline keskmine kiirus 40 000 toote rea korral isegi madalam kui 10 000 toote rea puhul. Kahte disaini omavahel võrreldes on näha, et *bytea* disaini puhul on uute ridade lisamine aeglasem kui *suurobjekti* disaini puhul, kuid siiski äärmiselt väike (ligikaudne erinevus 0,2 ms). Autor eeldas enne mõõtmist, et suurobjekti puhul rea lisamise kiiruse aeg andmemahu suurenedes tõuseb, sest süsteem peab leidma uue vaba OID, aga antud mõõtmised seda ei kajastanud.



Joonis 34. Ridade lisamise geomeetrilised keskmised kiirused.

8.4 Rakenduse füüsiliste koodiridade arv

Piltide salvestamisega seotud füüsiliste koodiridade arv oli kõige suurem *suurobjekti* lahenduse puhul. Suurobjektide lugemiseks ja kirjutamiseks oli vajalik transaktsiooni kasutamine, mis nõudis faili salvestamise meetodi sees andmebaasiga eraldi ühenduse loomist. Kogu piltide salvestamisega seotud koodiridade arv *suurobjekti* disaini korral oli 41.

Teisel kohal füüsiliste koodiridade arvu rohkuselt oli S3 lahendus, kus koodiridade arv oli 23. Antud lahenduse puhul tuli kõige pealt luua uus AmazonS3 klient, lisada vajalikud konfiguratsioonisätted ning seejärel koostada objekti lisamise lause.

Failisüsteemi disaini puhul oli füüsiliste koodiridade arv 13. Antud disain vajas ainult ühte meetodit, mis koostaks failinime ja teekonna ning salvestaks faili failisüsteemi.

Kõige vähem koodiridu läks vaja *bytea* lahenduse puhul, kus sissetulevad pildid teisendati baitide jadaks ja salvestati andmebaasi.

8.5 Rakenduse loomine

Töö autoril ei tekkinud raskusi ühegi disaini realiseerimisega. Kuna kõik disainid on laialt levinud, siis leidub nende kohta mitmetes erinevates programmeerimiskeeltes realiseerimiseks rohkelt viiteid ja materjale. Autor oli varasemalt kokku puutunud piltide failisüsteemi salvestamise disainiga.

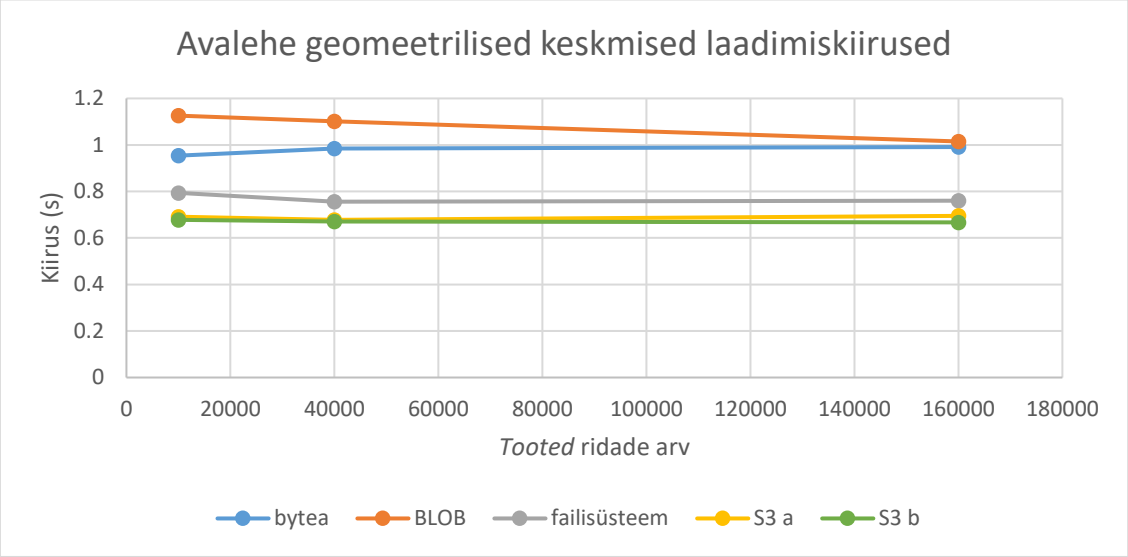
Autori arvates on kõige lihtsam realiseerida *bytea* disaini. *Bytea* disain ei nõua ka märkimisväärseid edasisi arendusi, sest pilte hoitakse tavalises kasutaja poolt loodud tabelis *bytea* tüüpi veerus, mis on nagu tavaline andmebaasi tabeli veerg. Lisaks on *bytea* disaini puhul piltide salvestamiseks vajaminevate koodiridade arv kõige väiksem.

Ülejäänud kolme disaini oli autoril palju raskem paremusjärjestusse seada. Kui hinnata vaid piltide salvestamise funktsionaalsust ja jätta arvestamata edasised arendused nagu varundamise ja kustutamise strateegia, siis paneks autor disainilahendused ritta järgnevalt: *failisüsteem*, *suurobjekt* ja *S3*. Antud kolme disaini realiseerimise keerukus oli enam-vähem võrdne. Siiski tuleb meeles pidada, et autor oli varasemalt *failisüsteemi* disainiga kokku puutunud, kuid *S3-e* disain ja korvile ligipääsetavus olid autori jaoks uudsed.

Kuna rakenduses disaini läbimõtlematu realiseerimine võib hilisemalt kaasa tuua suuri probleeme, peaks autori arvates ikkagi rakenduse loomise keerukuse hindamisel tähelepanu pöörama ka edasistele arendustele. Seetõttu leiab autor, et võttes arvesse disainimustrite kõiki omadusi, on teisel kohal rakenduse loomise lihtsuse kohapealt *suurobjekti* disain. *Suurobjektiga* enam-vähem sama keeruliseks hindab autor *S3* disaini, kuid paigutab selle siiski kolmandale kohale, sest ei ole hästi kursis turvalisuse ja ligipääsetavuse seadete konfigureerimisega. *Failisüsteemi* disaini jätab autor viimaseks, sest leiab, et see disain vajab kõige rohkem tähelepanu ja edasisi arendusi kuna andmete riknemise oht on väga suur ning andmete terviklikkuse tagamine vajab korralikku läbimõtlemist.

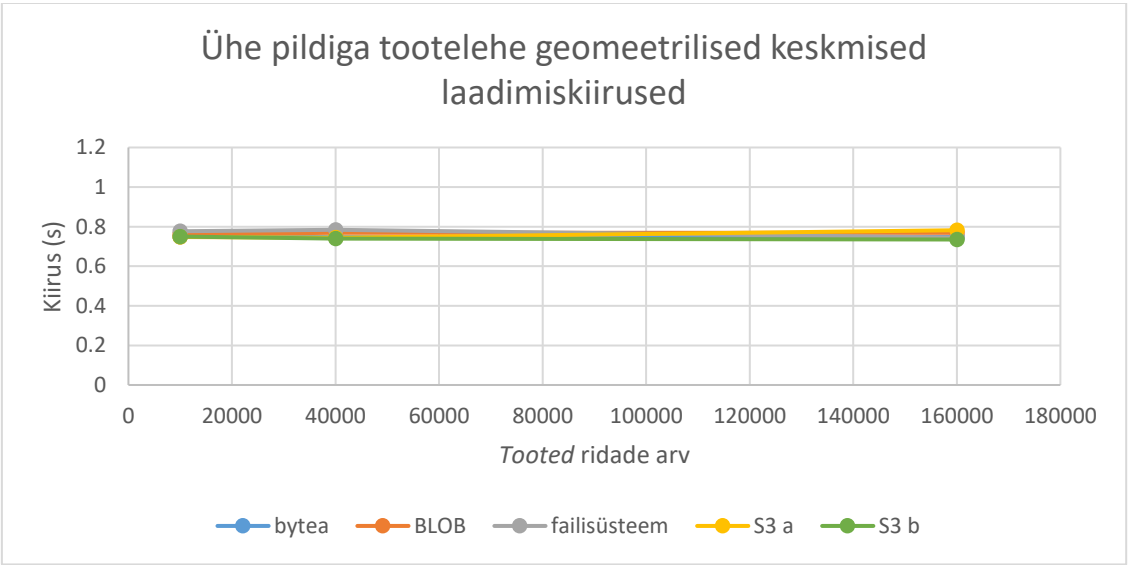
8.6 Rakenduse andmete kuvamiseks kuluv aeg

Joonis 35 esitab avalehe laadimise kiiruse sõltuvuse toodete ridade arvust erinevate disainide korral. Kõige kiirem lehe laadimine on *S3* disainilahenduste puhul ning kõige aeglasem *suurobjekti* disaini puhul, vastavalt ligikaudu 0,6 s ja 1,1 s. Kahe disaini ligikaudne laadimiskiiruse vahe on 0,4 s.



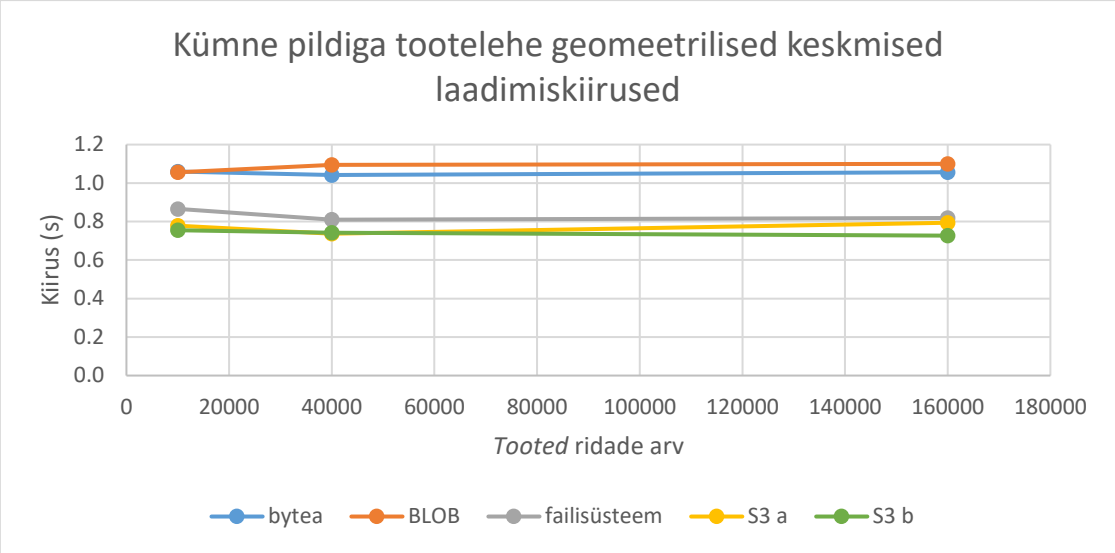
Joonis 35. Avalehe geomeetrilised keskmised laadimiskiirused.

Joonis 36 esitab ühe pildiga tootelehe laadimise kiiruse sõltuvuse toodete ridade arvust erinevate disainide korral. Lehe laadimise kiirus kõikide disainide puhul on peaaegu võrdne, jäädes 0,750 s lähedale.



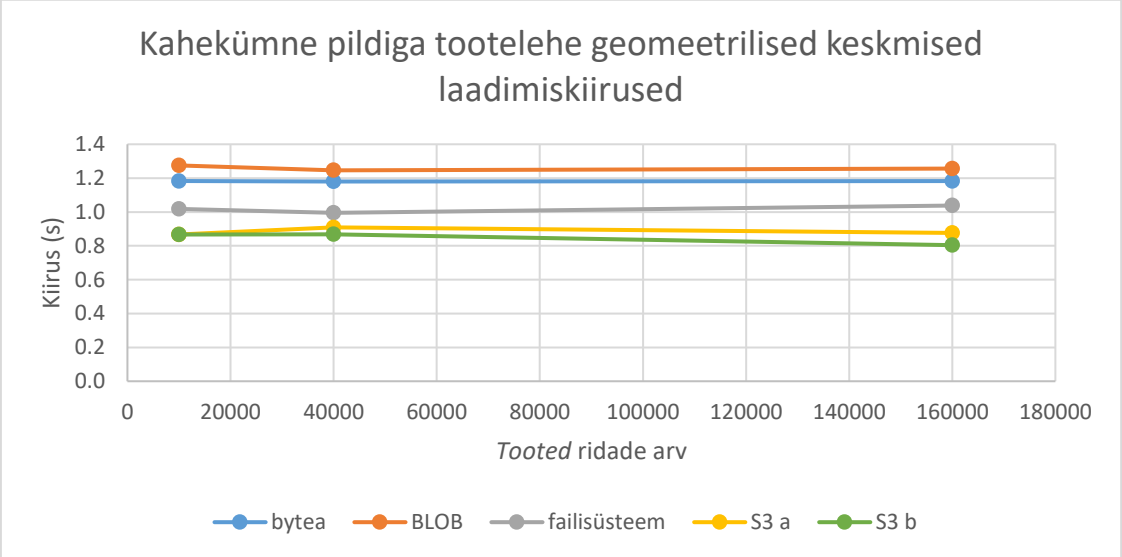
Joonis 36. Ühe pildiga toote lehe geomeetrilised keskmised laadimiskiirused.

Joonis 37 esitab kümne pildiga tootelehe laadimise kiiruse sõltuvuse toodete ridade arvust erinevate disainide korral. Kõige kiirem lehe laadimise kiirus on *S3* disainilahenduste puhul ja kõige aeglasem *suurobjekti* disaini puhul. Kiiruste ligikaudne vahe on 0,4 s.



Joonis 37. Kümne pildiga toote lehe geomeetrilised keskmised laadimiskiirused.

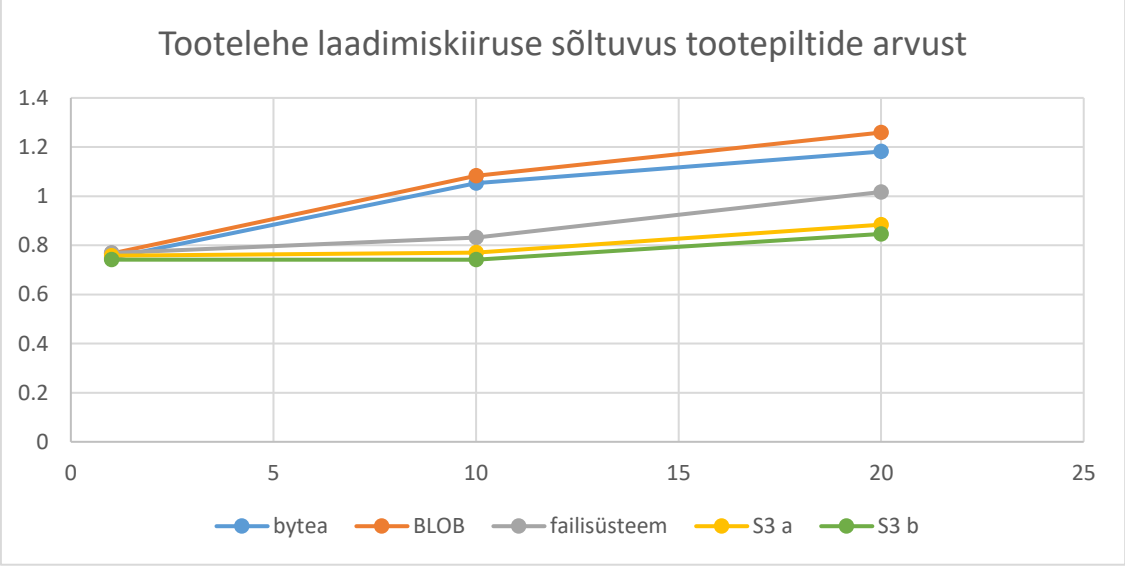
Joonis 38 esitab kahekümne pildiga tootelehe laadimise kiiruse sõltuvuse toodete ridade arvust erinevate disainide korral. Kõige kiirem lehe laadimise kiirus on *S3* disainilahenduste puhul ja kõige aeglasem *suurobjekti* disaini puhul, vastavalt ligikaudu 0,86 s ja 1,26 s. Kiiruste ligikaudne vahe on 0,4 s.



Joonis 38. Kahekümne pildiga toote lehe geomeetrilised keskmised laadimiskiirused.

Kõikidelt joonistelt oli näha, et andmebaasis olevate andmete hulk ei mõjuta lehe laadimise kiirust, vaid kiirus püsib enam-vähem ühesugusena. Kõige kiirem oli lehe laadimine S3 a ja S3 b disainilahenduste puhul, mis olid enam-vähem omavahel võrdsed. Seetõttu saab järeldada, et pildi viite pikkus ei mõjuta lehe laadimise kiirust. Üpris kohe järgnes *S3* disainilahenduste kiirustele *failisüsteemi* disain. Kõige aeglasemad olid *bytea* ja *suurobjekti* disain.

Joonis 39 esitab kõikide disainide puhul laadimiskiiruse sõltuvuse tootepiltide arvust. Jooniselt on näha, et tootepiltide arvu kasvades suureneb kergelt lehe laadimiseks kuluv aeg. Antud tunnuste seose tugevuse leidmiseks leidis autor Pearsoni korrelatsiooni koefitsendi. Koefitsentide väärtused on järgmised: *bytea* disaini puhul 0,9663, *suurobjekti* disaini puhul 0,9812, *failisüsteemi* disaini puhul 0,9698, *S3 a* disaini puhul 0,9184 ja *S3 b* disaini puhul 0,846. Arvutustulemused kinnitavad, et kõikide disainide puhul on lehe laadimise kiiruse ja tootepiltide arvu vahel tugev positiivne lineaarne korrelatsioon.



Joonis 39. Toote lehe laadimise kiiruse sõltuvus tootepiltide arvust.

Autor eeldas, et kõige kiirem lehe laadimise kiirus on *failisüsteemi* disaini puhul, kuid *S3* disain edestas *failisüsteemi* erinevate lehtede laadimise kiiruseid keskmiselt 0,2 sekundiga vaatamata selle, et *S3-e* disaini puhul tuleb lisaks andmebaasile andmeid laadida ka S3-st.

8.7 Töö järeldused

Disainilahenduste võrdlemise põhjal saab öelda, et ükski disain ei edesta teisi märkimisväärselt ning parima disaini valik sõltub konkreetsetest loodava süsteemi vajadustest. Igal disainil on oma tugevusi ja ka mõningaid puudusi.

Bytea disain on väga lihtsasti realiseeritav ja andmed kergesti hallatavad. Lisaks ei vaja *bytea* disain olulisi edasiarendusi. Seega ei ole vaja mõelda eraldi näiteks varundamise strateegiale ega ligipääsetavuse sätetele, sest seda juba tehakse andmebaasi kavandamise käigus. *Bytea* disaini puhul on tagatud andmete terviklikkus, st failide ja nendega seotud olemite/seosetüüpide kohta käivate andmete kooskõla. Siiski toob failide *bytea* kujul andmebaasis hoidmine omadega kaasa suured nõuded serveri mälule ja halva jõudluse. Faile andmebaasis hoides suureneb ka andmebaasi maht. Läbiviidud mõõtmiste tulemused näitasid, et *bytea* ei ole kõige efektiivsem päringutega, millega väljastatakse pildifaile. Ka rakenduse andmete kuvamiseks kuluv aeg on võrreldes teiste disainidega oluliselt suurem. Autor arvab, et *bytea* disaini peale võib mõelda, kui rakenduses kasutatavad pildid on väikese mahuga. Kindlasti tuleks välistada suuremahuliste failide hoidmise *bytea* veerus.

Suurobjekti disaini puhul saab andembaasis hoida väga suuri faile, sest maksimaalne lubatud maht väärtuse kohta on 4 TB. Nagu *bytea* disaini korral, on ka *suurobjekti* puhul tagatud andmete terviklikkus ning ei ole vaja mõelda olulisel määral edasistele arendustele. Siiski tuleb tähelepanu pöörata *suurobjektide* kustutamisele, sest tabelist objekti identifikaatorit sisaldava rea kustutamisel ei kustutata objekti ennast. Kuigi suurobjekti disaini oli võrdlemisi lihtne realiseerida vaatamata sellele, et koodiridade arv oli kõige suurem, siis mõõtmistulemused näitasid, et *suurobjekti* disaini puhul on kasutatud andmebaasi maht ning rakenduse andmete kuvamiseks kuluv aeg kõige suuremad. Sellest tulenevalt arvab autor, et suurobjekti disain sobiks kõige paremini lahenduse puhul, kus on vaja hoida mahukaid objekte, kuid neid tihedalt mitte pärida.

Failisüsteemi kasutamise disaini puhul on peamisteks eelisteks väike andmebaasisüsteemi hallatava andmebaasi maht ja hea jõudlus. Päringute täitmised on kiired ning rakenduses andmete kuvamise aeg on väga väikene. Failisüsteemi kasutamise disain nõuab aga korraliku läbimõtlemist, sest eraldi tuleb realiseerida näiteks nii väliste failide jälgimine kui ka kustutamise ja varundamise strateegia. Lisaks on raske tagada

andmete terviklikkust, sest välised failid ja andmebaas võivad minna sünkroonist välja. Peale selle tuleb failisüsteemi puhul kavandada veel kataloogide struktuur ja ka õiguste haldamine on keerulisem. Failisüsteemi disaini realiseerimine võib tunduda algul lihtne ja hea lahendus, kuid selle halvasti realiseerimisel võib disain endaga kaasa tuua rohkeid probleeme, sest kogu disain on üpris keeruline.

Pilvelahenduse disaini eelisteks on samuti väike andmebaasisüsteemi andmebaasi maht ja hea jõudlus. Mõõtmistulemused näitasid, et pilvelahenduse disaini päringud on kiired ning rakenduses andmete kuvamise aeg oli kõikidest disainidest parim. Pilvelahenduse disaini puhul ei ole vaja andmete valdajal endal hallata suuremahulist andmebaasi ning kasutaja ei pea mõtlema lisategevuste peale nagu näiteks versioonihaldus või varundamise strateegia. Ka disainilahenduse realiseerimine on üpriski lihtne. Käesolevas töös võrreldi kahte pilvelahenduse disaini. Ühe disaini puhul hoiti faili viitena tervet pildiaadressi (URLi), kuid teise disaini puhul vaid faili nime. Kuigi päringute kiiruses ja rakenduses andmete kuvamiseks kuluvas ajas erinevusi ei tekkinud, siis mõjutas pikema faili viite hoidmine kümnekordselt andmebaasi mahtu. Seetõttu soovitab autor hoida andmebaasis viitena vaid faili nime või vajadusel osalist kataloogiteed, kuid mitte tervet URL-i. Pilvelahenduse negatiivse aspektina saab välja tuua interneti probleemide otsese mõju teenuse töökiirusele. Lisaks sõltub kasutajaandmete turvalisus teenusepakkujast ning kõiki andmeid ei pruugi olla võimalik teenusepakkujale säilitamiseks ega töötlemiseks usaldada. Kui viimased kaks ei ole probleemiks, on autori arvates mõistlik kasutada tänapäevaseid pilveandmetöötluse võimalusi ning realiseerida failide salvestamine pilvelahenduse disainimustrina.

8.8 Töö puudused

Käesolevas jaotuses nimetab autor tehtud töö puudused.

Autor leiab, et töö üheks puuduseks on mõõtmiste läbiviimine sarnase suurusega pildifailidega. Käesoleva töö mõõtmiste jaoks kasutatud pildifailide suurused olid keskmiselt 30 KB. Seetõttu ei saa kindlalt öelda, kas käesoleva töö mõõtmistulemuste põhjal tehtud järeldused kehtiksid ka suuremahuliste failide kasutamisel või siis, kui failide suuruses erineksid üksteisest märkimisväärselt.

Veel üks vastuseta küsimus on kas lehtede laadimise kiiruse tulemused oleksid olnud erinevad, kui andmebaas oleks olnud kohalikus serveris (mitte Amazoni pilves nagu praegu) ja sealt oleks viidatud Amazoni S3 pilves olevatele pildiobjektidele.

Töö kolmanda puudusena tegi autor vea ja kasutas suurobjektide disaini puhul *picture_oid* veeru tüübina tüüpi INTEGER, mitte OID. PostgreSQL kasutab OID tüüpi ja selle väärtuseid andmebaasi siseselt näiteks süsteemikataloogis. OID väärtused on märgita täisarvud. Teoreetiliselt ei peaks see eksimus tulemusi mõjutama.

Töö neljandaks puuduseks on, et erinevate disainide sama andmemahu korral võrdlemisel ei olnud andmebaasis samad andmed, vaid igaks kord genereeriti disaini jaoks unikaalsed testandmed.

Töö viienda puudusena toob autor välja kõikide disainilahenduste puhul näiterakenduste mittetäieliku realiseerimise. Mitmed disainid vajavad lisaks failide salvestamisele ka erinevaid edasiarendusi nagu näiteks väliste failide jälgimine, varundamise strateegia või ligipääsu haldamine. Samuti pole S3 näiterakenduses realiseeritud objektide lukustamist. Seetõttu ei pruugi autori poolt esitatud hinnangud realiseerimise lihtsuse osas hõlmata autori kogemuse puudumise tõttu kõiki vajalike aspekte.

9 Kokkuvõte

Pilte ja muud meediat kasutatakse tänapäeval enamikes rakendustes ning seetõttu on rakendusi, mis haldavad terabaitide mahus pilte. Samas on failide salvestamise disainilahendused olulised igasuguse suurusega ja väga erinevate valdkondade süsteemides ning andmebaasi kavandamisel on rakenduse ja andmebaasi maksimaalse võimekuse tagamiseks oluline teada, millised on erinevate disainilahenduste eelised ja puudused.

Töö eesmärgiks on süstematiseerida infot SQL-andmebaaside disainilahenduste kohta, mis on mõeldud toetama failide salvestamist vajavaid andmebaasirakendusi ning analüüsida ja võrrelda neid disaine nii andmebaasi kui ka andmebaasirakednuse seisukohast.

Töö tulemusena esitati neli failide salvestamist toetavat SQL-andmebaasi disainimustrit: *bytea*, *suurobjekt*, *failisüsteem* ja *pilvelahendus*. Esimesed kaks nendest põhinevad spetsiifilistel PostgreSQL andmebaasisüsteemi võimekustel ja on alammustrid, mis laiendavad üldist mustrit, mille kohaselt võib hoida faile andmebaasis Autor analüüsis iga uuritava disainilahenduse eeliseid ja puuduseid ning pani disainilahendused kirja mustri formaadis. Kõikide disainimustrite põhjal projekteeriti ja realiseeriti PostgreSQL-is näiteandmetega andmebaasid. Iga loodud andmebaasi põhjal realiseeriti andmete lisamise, muutmise, kustutamise ja vaatamise funktsionaalsusega veebirakendus C# programmeerimiskeeles kasutades .NET raamistikku. Andmebaasid ja rakendused seati üles Amazoni pilvekeskkonda. Loodud andmebaasid ja rakendused olid mõeldud tootekataloogi haldamiseks.

Disainilahenduste võrdlemiseks planeeriti ning viidi läbi mõõtmised ja selle kaudu võrreldi disainilahendusi andmebaasi andmemahu, päringute täitmise ja ridade lisamise töökiiruse, rakenduse füüsiliste koodiridade arvu, rakenduse loomise subjektiivse lihtsuse ja rakenduses andmete kuvamiseks kuluva aja põhjal. Hindamaks andmemahtude mõju päringute täitmisele ja andmete kuvamiseks kuluva aja kiirustele, viis autor mõõtmised läbi kolme erineva andmehulgaga.

Disainimustreid realiseerides ning mustrite võrdlemiseks läbi viidud mõõtmiste tulemusi analüüsides ilmnes, et parimat disaini ei ole olemas. Parima disaini valik sõltub konkreetse süsteemi nõuetest. Kuigi disainimustri realiseerimine võib tunduda esialgu lihtsa ülesandena, siis failide üleslaadimise ja esitamise funktsionaalsuste halvasti realiseerimisel muutub failide haldamine kiiresti üheks rakenduse valukohaks. Seetõttu tuleb disaini valiku otsus eelnevalt korralikult läbi mõelda. Ridade arvu kasvu ja töökiiruse vähenemise vahel leidis autor alati tugeva lineaarse korrelatsiooni.

Kokkuvõttes võib öelda, et töö eesmärk täideti. Töö käigus realiseeritud rakenduste lähtekood on saadaval https://github.com/erlemaido/File-Storage-Design-Patterns. Kuni kaitsmise perioodi lõpuni hoiab autor ühte näidisrakendust üleval Amazoni pilveserveris. Täpsem info on GitHub'i lehel.

Kasutatud kirjandus

[1] M. Jõgi, „Mõned disainimustrid isikunimede hoidmiseks SQL-andmebaasides,“ Tallinna Tehnikaülikool Informaatikainstituut, Tallinn, 2016.

[2] „e-teatmik,“ [Võrgumaterjal]. Available: http://vallaste.ee/. [Kasutatud 25 04 2021].

[3] „Keskseade,“ 08 04 2020. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Keskseade. [Kasutatud 17 05 2021].

[4] K. Raja, „Päringu filtri predikaadi automaatne lihtsustamine kahe SQL-andmebaasi näitel,“ Tallinna Tehnikaülikool, Tallinn, 2017.

[5] „URI,“ 14 05 2017. [Võrgumaterjal]. Available: https://et.wiktionary.org/wiki/URI. [Kasutatud 17 05 2021].

[6] C.-Y. Lu, „Analyse Journal of XFS Filesystem for Assisting in Event Reconstruction,“ Tallinna Tehnikaülikool, Tallinn, 2020.

[7] K. Kozak, Management of large sets of image data: Capture, Databases, Image Processing, Storage, Visualization, Bookboon, 2014.

[8] B. Karwin, SQL Antipatterns, Pragmatic Bookshelf, 2010.

[9] S. Russell, C. v. Ingen ja J. Gray, „To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?,“ Microsoft Research, Redmond, WA, 2006.

[10] „Photos Uploaded on Instagram,“ [Võrgumaterjal]. Available: https://wouldyouhavethought.com/live-stats/photos-uploaded-on-instagram. [Kasutatud 25 02 2021].

[11] A. Monnappa, „How Facebook is Using Big Data: The Good, the Bad, and the Ugly,“ Simplilearn, 18 01 2021. [Võrgumaterjal]. Available: https://www.simplilearn.com/how-facebook-is-using-big-data-article. [Kasutatud 22 04 2021].

[12] V. K. B. P. S. Vaishnavi, „Design Science Research in Information Systems,“ 2019.

[13] „DB-Engines Ranking,“ 03 2021. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking. [Kasutatud 16 03 2021].

[14] „Enterprise Architect,“ Sparx Systems, [Võrgumaterjal]. Available: https://sparxsystems.com/. [Kasutatud 16 03 2021].

[15] „TIOBE Index for March 2021,“ 03 2021. [Võrgumaterjal]. Available: https://www.tiobe.com/tiobe-index/. [Kasutatud 16 03 2021].

[16] A. Siddique, „Handling thousands of image upload per second with Amazon S3,“ Medium, 23 03 2019. [Võrgumaterjal]. Available: https://medium.com/udroppy/handling-thousands-of-image-upload-per-second-with-amazon-s3-7a1009e8ffc4. [Kasutatud 21 02 2021].

[17] „Is it a bad practice to store large files (10 MB) in a database?,“ StackExchange, [Võrgumaterjal]. Available: https://softwareengineering.stackexchange.com/questions/150669/is-it-a-bad-practice-to-store-large-files-10-mb-in-a-database. [Kasutatud 25 02 2021].

[18] „Storing Images in DB - Yea or Nay?,“ Stack Overflow, [Võrgumaterjal]. Available: https://stackoverflow.com/questions/3748/storing-images-in-db-yea-or-nay#3756. [Kasutatud 25 02 2021].

[19] „Storing a million images in the filesystem,“ StackExchange, [Võrgumaterjal]. Available: https://serverfault.com/questions/95444/storing-a-million-images-in-the-filesystem. [Kasutatud 25 02 2021].

[20] „Should binary files be stored in the database?,“ StackExchange, [Võrgumaterjal]. Available: https://dba.stackexchange.com/questions/2445/should-binary-files-be-stored-in-the-database. [Kasutatud 25 02 2021].

[21] „Storing Images in PostgreSQL,“ Stack Overflow, [Võrgumaterjal]. Available: https://stackoverflow.com/questions/54500/storing-images-in-postgresql. [Kasutatud 25 02 2021].

[22] „Filestream (SQL Server),“ Microsoft, 01 11 2018. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/blob/filestream-sql-server?view=sql-server-ver15. [Kasutatud 22 04 2021].

[23] „Attach files and graphics to the records in your database,“ Microsoft, [Võrgumaterjal]. Available: https://support.microsoft.com/en-us/office/attach-files-and-graphics-to-the-records-in-your-database-d40a09ad-a753-4a14-9161-7f15baad6dbd. [Kasutatud 23 04 2021].

[24] A. Nanda, „SecureFiles: The New LOBs,“ Oracle, [Võrgumaterjal]. Available: https://www.oracle.com/technical-resources/articles/database/sql-11g-securefiles.html. [Kasutatud 22 04 2021].

[25] „Chapter 7. Storing Binary Data,“ [Võrgumaterjal]. Available: https://jdbc.postgresql.org/documentation/80/binary-data.html. [Kasutatud 21 04 2021].

[26] „Oktett (arvutus) - Octet (computing),“ Wikipedia Foundation, [Võrgumaterjal]. Available: https://et.mihalicdictionary.org/wiki/Octet_(computing). [Kasutatud 16 05 2021].

[27] „8.4. Binary Data Types,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/13/datatype-binary.html. [Kasutatud 21 04 2021].

[28] „Bytea vs OID (Large Objects),“ Microolap Technologies LTD, [Võrgumaterjal]. Available: https://www.microolap.com/products/connectivity/postgresdac/help/tipsandtricks_byteavsoid.htm. [Kasutatud 21 04 2021].

[29] L. Albe, „Binary Data Performance in PostgreSQL,“ Cybertec PostgreSQL International GmbH, 14 05 2020. [Võrgumaterjal]. Available: https://www.cybertec-postgresql.com/en/binary-data-performance-in-postgresql/. [Kasutatud 28 04 2021].

[30] „Chapter 34. Large Objects,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/largeobjects.html. [Kasutatud 24 04 2021].

[31] „34.1. Introduction,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/lo-intro.html. [Kasutatud 21 04 2021].

[32] M. Marqués, „Those darn Large Objects,“ 2ndQuadrant, 20 09 2016. [Võrgumaterjal]. Available: https://www.2ndquadrant.com/en/blog/those-darn-large-objects/. [Kasutatud 18 05 2021].

[33] P. Mount, „F.20. lo,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/lo.html. [Kasutatud 16 05 2021].

[34] „Failisüsteem,“ 17 11 2020. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Failis%C3%BCsteem. [Kasutatud 18 05 2021].

[35] I. Suur, „NTFS olemus ja sisu,“ TalTech IT Kolledži wiki, 09 12 2013. [Võrgumaterjal]. Available: https://wiki.itcollege.ee/index.php/NTFS_olemus_ja_sisu. [Kasutatud 24 04 2021].

[36] „Ühtne ressursiidentifikaator,“ 11 11 2020. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/%C3%9Chtne_ressursiidentifikaator. [Kasutatud 16 05 2021].

[37] P. Eisentraut, „uri type for PostgreSQL,“ 25 12 2015. [Võrgumaterjal]. Available: https://github.com/petere/pguri. [Kasutatud 16 05 2021].

[38] S. Baras, I. Saeed ja H. Hajjdiab, „Security and Privacy of AWS S3 and Azure Blob Storage Services,“ Institute of Electrical and Electronics Engineers, 2019.

[39] Z. Daher ja H. Hajjdiab, „Cloud Storage Comparative Analysis Amazon Simple Storage vs. Microsoft Azure Blob Storage,“ International Journal of Machine Learning and Computing, 2018.

[40] E. Eessaar, „Teema 1. Andmebaaside põhimõisteid,“ 2021.

[41] „Global cloud services market Q4 2020,“ Canalys, 02 02 2021. [Võrgumaterjal]. Available: https://www.canalys.com/newsroom/global-cloud-market-q4-2020. [Kasutatud 21 04 2021].

[42] „What is Amazon S3?,“ Amazon Web Services, [Võrgumaterjal]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. [Kasutatud 21 04 2021].

[43] „Using Amazon S3 storage classes,“ Amazon Web Services, [Võrgumaterjal]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/storage-class-intro.html. [Kasutatud 21 04 2021].

[44] K. Stolze, „SQL/MM Part 5: Still Image - The Standard and Implementation Aspects -,“ 2001.

[45] J. Melton ja A. Eisenberg, „SQL Multimedia and Application Packages (SQL/MM),“ ACM SIGMOD Record, 2001.

[46] E. Eessaar, „Teema 10. Objekt-relatsioonilised andmebaasisüsteemid,“ 2020.

[47] P. B. Dimitar Mišev, „SQL Support for Multidimensional Arrays,“ Jacobs University, Bremen, 2017.

[48] „PostgreSQL Application in Image Search and Video and Image Deduplication,“ Alibaba Cloud, 05 02 2020. [Võrgumaterjal]. Available: https://www.alibabacloud.com/blog/postgresql-application-in-image-search-and-video-and-image-deduplication_595794. [Kasutatud 26 03 2021].

[49] A. Korotkov, „ImgSmlr - similar images search for PostgreSQL,“ 2018. [Võrgumaterjal]. Available: https://github.com/postgrespro/imgsmlr. [Kasutatud 26 03 2021].

[50] „Lainik,“ Vikipedia, 16 06 2020. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Lainik. [Kasutatud 05 11 2021].

[51] A. Vasiliev, „Effective similarity search in PostgreSQL,“ Railsware Solutions, 10 05 2012. [Võrgumaterjal]. Available: https://railsware.com/blog/effective-similarity-search-in-postgresql/. [Kasutatud 11 05 2021].

[52] Eesti Keele Instituut, „[EKSS] "Eesti keele seletav sõnaraamat" 2009,“ 2009. [Võrgumaterjal]. Available: http://www.eki.ee/dict/ekss/index.cgi?Q=muster&F=M. [Kasutatud 07 04 2020].

[53] E. Gamma, R. Helm, R. Johnson ja J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Pearson Education, 1994.

[54] „Enimkasutatavad disainimustrid,“ [Võrgumaterjal]. Available: https://ained.ttu.ee/javadoc/oop/oop-design-patterns.html. [Kasutatud 06 04 2021].

[55] E. Hövel, „Mõned mitmekeelsete SQL-andmebaaside disainimustrid,“ Tallinna Tehnikaülikool Tarkvarateaduste instituut, Tallinn, 2017.

[56] „PostgreSQL: BYTEA vs OID+Large Object?,“ Stack Exchange, [Võrgumaterjal]. Available: https://stackoverflow.com/questions/4645007/postgresql-bytea-vs-oidlarge-object. [Kasutatud 28 04 2021].

[57] „File System vs DBMS: Key Differences,“ Guru99, [Võrgumaterjal]. Available: https://www.guru99.com/difference-between-file-system-and-dbms.html. [Kasutatud 28 04 2021].

[58] A. Sulaiman, „File System vs. Database,“ Database Zone, 27 04 2017. [Võrgumaterjal]. Available: https://dzone.com/articles/which-is-better-saving-files-in-database-or-in-fil. [Kasutatud 28 04 2021].

[59] „What is the best place for storing uploaded images, SQL database or disk file system?,“ Stack Exchange, [Võrgumaterjal]. Available: https://stackoverflow.com/questions/348363/what-is-the-best-place-for-storing-uploaded-images-sql-database-or-disk-file-sy. [Kasutatud 28 04 2021].

[60] „BinaryFilesInDB,“ [Võrgumaterjal]. Available: https://wiki.postgresql.org/wiki/BinaryFilesInDB. [Kasutatud 28 04 2021].

[61] „51.30. pg_largeobject,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/13/catalog-pg-largeobject.html. [Kasutatud 17 05 2021].

[62] „8.19. Object Identifier Types,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/13/datatype-oid.html. [Kasutatud 12 05 2021].

[63] „Which is the best practice either to save image name or full URL in database,“ Stack Exchange, [Võrgumaterjal]. Available: https://stackoverflow.com/questions/52096229/which-is-the-best-practice-either-to-save-image-name-or-full-url-in-database/52097082. [Kasutatud 16 05 2021].

[64] „What is the longest file path allowed for Linux?,“ Quora, [Võrgumaterjal]. Available: https://www.quora.com/What-is-the-longest-file-path-allowed-for-Linux. [Kasutatud 16 05 2021].

[65] „How long can a URL be?,“ net-informations.com, [Võrgumaterjal]. Available: http://net-informations.com/q/mis/len.html. [Kasutatud 16 05 2021].

[66] „What is the maximum length of a URL in different browsers?,“ Stack Exchange, [Võrgumaterjal]. Available: https://stackoverflow.com/questions/417142/what-is-the-maximum-length-of-a-url-in-different-browsers. [Kasutatud 18 05 2021].

[67] R. E. Al-Qutaish, „Quality Models in Software Engineering Literature: An Analytical and Comparative Study,“ Journal of American Science, 2010.

[68] „5.4. Constraints,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/12/ddl-constraints.html. [Kasutatud 08 05 2021].

[69] K. Zhao, „IAM Policies and Bucket Policies and ACLs! Oh, My! (Controlling Access to S3 Resources),“ Amazon Web Services, 19 11 2013. [Võrgumaterjal]. Available:

https://aws.amazon.com/blogs/security/iam-policies-and-bucket-policies-and-acls-oh-my-controlling-access-to-s3-resources/. [Kasutatud 17 05 2021].

[70] S. Smith, „How to optimize your e-commerce images to have a website as fast as the wind,“ Doofinder, [Võrgumaterjal]. Available: https://www.doofinder.com/en/blog/optimize-images. [Kasutatud 10 05 2021].

[71] „S3 Transfer Acceleration,“ Amazon Web Services, [Võrgumaterjal]. Available: https://aws.amazon.com/s3/transfer-acceleration/. [Kasutatud 11 05 2021].

[72] „The Autovacuum Daemon,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/routine-vacuuming.html#AUTOVACUUM. [Kasutatud 18 05 2021].

[73] „Announcing Amazon Elastic Compute Cloud (Amazon EC2),“ Amazon Web Services, 24 08 2006. [Võrgumaterjal]. Available: https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/. [Kasutatud 10 05 2021].

[74] „How to Get Table, Database, Indexes, Tablespace, and Value Size in PostgreSQL,“ PostgreSQLTutorial.com, [Võrgumaterjal]. Available: https://www.postgresqltutorial.com/postgresql-database-indexes-table-size/. [Kasutatud 08 05 2021].

[75] „C# Coding Conventions (C# Programming Guide),“ Microsoft, 14 05 2021. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions. [Kasutatud 18 05 2021].

[76] „C# Coding Style,“ Microsoft, 26 03 2021. [Võrgumaterjal]. Available: https://github.com/dotnet/runtime/blob/main/docs/coding-guidelines/coding-style.md. [Kasutatud 18 05 2021].

[77] „How fast does your website load?,“ GTmetrix, [Võrgumaterjal]. Available: https://gtmetrix.com/. [Kasutatud 15 05 2021].

[78] K. Rootalu, „Korrelatsioonikordajad,“ Tartu Ülikool, 2012. [Võrgumaterjal]. Available: http://samm.ut.ee/korrelatsioonikordajad. [Kasutatud 16 05 2021].

[79] A. Szczepanek, „Pearson Correlation Calculator,“ Omni Calculator, 22 10 2020. [Võrgumaterjal]. Available: https://www.omnicalculator.com/statistics/pearson-correlation. [Kasutatud 17 05 2021].

[80] „In pgadmin EXPLAIN ANALYSE , exclusive vs inclusive,“ Stack Exchange, [Võrgumaterjal]. Available: https://stackoverflow.com/questions/61983077/in-pgadmin-explain-analyse-exclusive-vs-inclusive. [Kasutatud 18 05 2021].

[81] „PostgreSQL's explain analyze made readable,“ [Võrgumaterjal]. Available: https://explain.depesz.com/. [Kasutatud 18 05 2021].

[82] „Storing Binary Data,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/7.4/jdbc-binary-data.html. [Kasutatud 25 03 2021].

[83] „NTFS,“ Wikipedia, 16 04 2021. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/NTFS. [Kasutatud 24 04 2021].

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

Mina, Erle Maido

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Mõned failide salvestamise disainimustrid SQL-andmebaasi kasutavate andmebaasirakenduste jaoks“, 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.

18.05.2021

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 – SQL tabelite loomise laused

Suurobjekt

```
CREATE SCHEMA blob;

CREATE TABLE Product_state_type
(
    product_state_type_code  SMALLINT  CONSTRAINT  PK_product_state_type
    PRIMARY KEY,
    title                    VARCHAR(50)    NOT NULL
);

CREATE UNIQUE INDEX IX_product_state_type_title ON Product_state_type
(title);

CREATE TABLE Product
(
    product_id               SERIAL      CONSTRAINT PK_product PRIMARY KEY,
    product_code             VARCHAR(14)              NOT NULL,
    title                    VARCHAR(255)             NOT NULL,
    price                    DECIMAL(19, 4)           NOT NULL,
    product_state_type_code  SMALLINT    DEFAULT 1    NOT NULL,
    CONSTRAINT FK_product_product_state_type_product_state_type_code
        FOREIGN KEY(product_state_type_code)
        REFERENCES Product_state_type(product_state_type_code)
        ON DELETE CASCADE
);

CREATE INDEX IX_product_product_state_type_code ON Product
(product_state_type_code);

CREATE TABLE Product_picture
(
    product_picture_id  BIGSERIAL  CONSTRAINT PK_product_picture PRIMARY KEY,
    product_id          INTEGER    NOT NULL,
    seq_nr              SMALLINT   NOT NULL,
    picture_oid         INTEGER    NOT NULL,
    CONSTRAINT FK_product_picture_product_product_id
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);

CREATE UNIQUE INDEX IX_product_picture_product_id_seq_nr ON Product_picture
(product_id, seq_nr);
```

Bytea

```
CREATE SCHEMA bytea;

CREATE TABLE Product_state_type
(
    product_state_type_code  SMALLINT  CONSTRAINT  PK_product_state_type
    PRIMARY KEY,
    title                    VARCHAR(50)    NOT NULL
);

CREATE UNIQUE INDEX IX_product_state_type_title ON Product_state_type
(title);

CREATE TABLE Product
(
    product_id               SERIAL      CONSTRAINT PK_product PRIMARY KEY,
    product_code             VARCHAR(14)              NOT NULL,
    title                    VARCHAR(255)             NOT NULL,
    price                    DECIMAL(19, 4)           NOT NULL,
    product_state_type_code  SMALLINT    DEFAULT 1    NOT NULL,
    CONSTRAINT FK_product_product_state_type_product_state_type_code
        FOREIGN KEY(product_state_type_code)
        REFERENCES Product_state_type(product_state_type_code)
        ON DELETE CASCADE
);

CREATE INDEX IX_product_product_state_type_code ON Product
(product_state_type_code);

CREATE TABLE Product_picture
(
    product_picture_id  BIGSERIAL  CONSTRAINT PK_product_picture PRIMARY KEY,
    product_id          INTEGER    NOT NULL,
    seq_nr              SMALLINT   NOT NULL,
    picture             BYTEA      NOT NULL,
    CONSTRAINT FK_product_picture_product_product_id
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);

CREATE UNIQUE INDEX IX_product_picture_product_id_seq_nr ON Product_picture
(product_id, seq_nr);
```

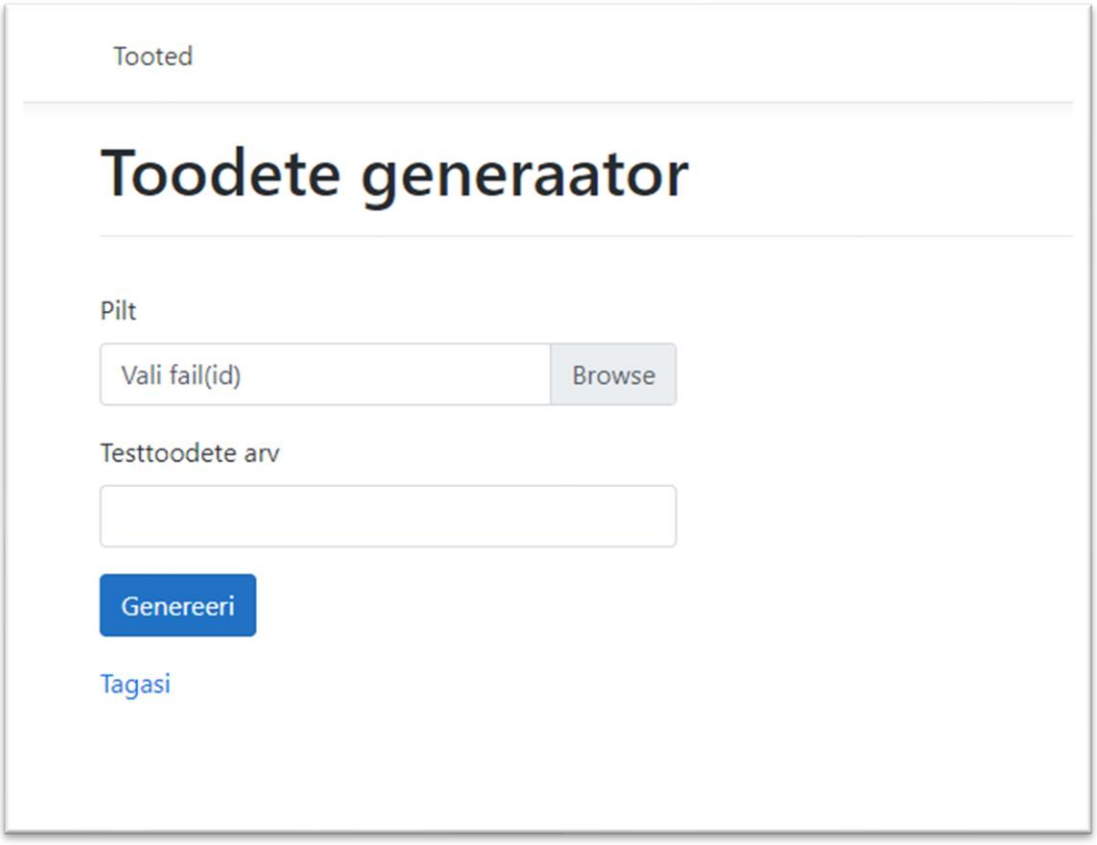
Failisüsteem

```
CREATE SCHEMA filesystem;

CREATE TABLE Product_state_type
(
    product_state_type_code  SMALLINT  CONSTRAINT  PK_product_state_type
    PRIMARY KEY,
    title                    VARCHAR(50)    NOT NULL
);

CREATE UNIQUE INDEX IX_product_state_type_title ON Product_state_type
(title);

CREATE TABLE Product
(
    product_id               SERIAL      CONSTRAINT PK_product PRIMARY KEY,
    product_code             VARCHAR(14)              NOT NULL,
    title                    VARCHAR(255)             NOT NULL,
    price                    DECIMAL(19, 4)           NOT NULL,
    product_state_type_code  SMALLINT    DEFAULT 1    NOT NULL,
    CONSTRAINT FK_product_product_state_type_product_state_type_code
        FOREIGN KEY(product_state_type_code)
        REFERENCES Product_state_type(product_state_type_code)
        ON DELETE CASCADE
);

CREATE INDEX IX_product_product_state_type_code ON Product
(product_state_type_code);

CREATE TABLE Product_picture
(
    product_picture_id  BIGSERIAL  CONSTRAINT PK_product_picture PRIMARY KEY,
    product_id          INTEGER      NOT NULL,
    seq_nr              SMALLINT     NOT NULL,
    picture_url         VARCHAR(255) NOT NULL,
    CONSTRAINT FK_product_picture_product_product_id
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);

CREATE UNIQUE INDEX IX_product_picture_product_id_seq_nr ON Product_picture
(product_id, seq_nr);
```

Pilvelahendus

```
CREATE SCHEMA cloud;

CREATE TABLE Product_state_type
(
    product_state_type_code  SMALLINT  CONSTRAINT  PK_product_state_type
    PRIMARY KEY,
    title                    VARCHAR(50)    NOT NULL
);

CREATE UNIQUE INDEX IX_product_state_type_title ON Product_state_type
(title);

CREATE TABLE Product
(
    product_id               SERIAL      CONSTRAINT PK_product PRIMARY KEY,
    product_code             VARCHAR(14)              NOT NULL,
    title                    VARCHAR(255)             NOT NULL,
    price                    DECIMAL(19, 4)           NOT NULL,
    product_state_type_code  SMALLINT    DEFAULT 1    NOT NULL,
    CONSTRAINT FK_product_product_state_type_product_state_type_code
        FOREIGN KEY(product_state_type_code)
        REFERENCES Product_state_type(product_state_type_code)
        ON DELETE CASCADE
);

CREATE INDEX IX_product_product_state_type_code ON Product
(product_state_type_code);

CREATE TABLE Product_picture
(
    product_picture_id  BIGSERIAL  CONSTRAINT PK_product_picture PRIMARY KEY,
    product_id          INTEGER      NOT NULL,
    seq_nr              SMALLINT     NOT NULL,
    picture_url         VARCHAR(255) NOT NULL,
    CONSTRAINT FK_product_picture_product_product_id
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);

CREATE UNIQUE INDEX IX_product_picture_product_id_seq_nr ON Product_picture
(product_id, seq_nr);
```

Lisa 3 – Rakenduse toodete genereerimise vaade



Lisa 4 – Toodete generaatori programmikood

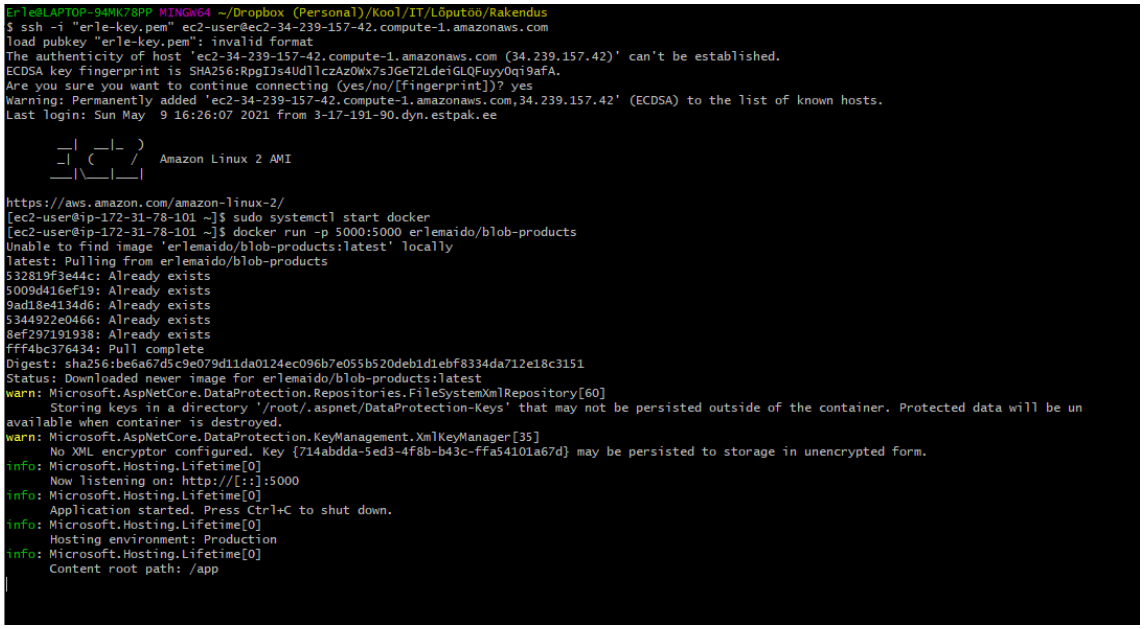
```
public async Task<IActionResult> GenerateProduct(int size,
    List<IFormFile> allPossiblePics)
{
    for (var i = 0; i < size; i++)
    {
        var product = CreateRandomProduct(Convert.ToInt32(i));
        if (allPossiblePics != null && allPossiblePics.Any())
        {
            AddRandomProductPictures(allPossiblePics, product);
        }

        try
        {
            await _context.AddRangeAsync(product);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.StackTrace);}
        }
    }
    await _context.SaveChangesAsync();

    return RedirectToAction(nameof(Index), "Products");
}

private void AddRandomProductPictures(IReadOnlyList<IFormFile>
    allPossiblePics, Product product)
{
    var productPictures = new List<ProductPicture>();
    var allPossiblePicsCount = allPossiblePics.Count;
    var nrOfProductPics = Random.Next(MinPicsPerProduct,
        MaxPicsPerProduct <= allPossiblePicsCount
            ? MaxPicsPerProduct + 1
            : allPossiblePicsCount + 1);

    var unusedIndices = Enumerable.Range(0,allPossiblePics.Count).ToArray();

    for (var i = 0; i < nrOfProductPics; i++)
    {
        var randomPicIndex = GetRandomNrFromArray(allPossiblePicsCount,
            unusedIndices);
            unusedIndices = unusedIndices.Where(x =>!x.Equals(randomPicIndex))
                                         .ToArray();
        var picture = allPossiblePics[randomPicIndex];
```

```
            var pictureUrl = UploadFile(picture, product.ProductCode);

            productPictures.Add(new ProductPicture
            {
                PictureUrl = pictureUrl,
                Product = product,
                SeqNr = Convert.ToInt32(i) + 1
            });
        }

        product.ProductPictures = productPictures;
    }

    private static Product CreateRandomProduct(int i)
    {
        return new()
        {
            Title = "Kaup" + i + " " + GenerateRandomTitle(10),
            Price = GenerateRandomPrice(0.01, 9999.99),
            ProductCode = GenerateRandomProductCode(6),
            ProductStateTypeCode = Random.Next(1, 5)
        };
    }

    private static string GenerateRandomProductCode(int length)
    {
        const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-";
        return new string(Enumerable.Repeat(chars, length)
            .Select(s => s[Random.Next(s.Length)]).ToArray());
    }

    private static string GenerateRandomTitle(int length)
    {
        const string chars = "abcdefghijklmnoprstuv ";
        return new string(Enumerable.Repeat(chars, length)
            .Select(s => s[Random.Next(s.Length)]).ToArray());
    }

    private static decimal GenerateRandomPrice(double min, double max)
    {
        return Convert.ToDecimal(Random.NextDouble() * (max - min) + min);
    }

    private static int GetRandomNumberFromArray(int max, int[] arr)
    {
        var rand = Random.Next(0, max);
        while (!arr.Contains(rand))
        {
            rand = Random.Next(0, max);
        }

        return rand;
    }
```

Lisa 5 – *Dockerfile* Dockeri tõmmise loomiseks

```
# Define base image
FROM mcr.microsoft.com/dotnet/sdk:5.0.201-alpine3.12 AS builder

# Copy project files
WORKDIR /source
COPY ./BlobPhotoStorageApp.sln .
COPY ./WebApp/WebApp.csproj ./WebApp/

# Restore
RUN dotnet restore

# Copy all source code
COPY ./WebApp/ ./WebApp/

# Publish
WORKDIR /source
RUN dotnet publish -c Release -o /publish

# Runtime
FROM mcr.microsoft.com/dotnet/aspnet:5.0.4-alpine3.12
ENV ASPNETCORE_URLS=http://+:5000
EXPOSE 5000
WORKDIR /app
COPY --from=builder /publish .
ENTRYPOINT ["dotnet", "WebApp.dll"]
```

Lisa 6 – Dockeri konteineri käitamine



Lisa 7 – Andmemahu mõõtmise SQL laused

```
SELECT pg_size_pretty(pg_total_relation_size('product'));
SELECT pg_size_pretty(pg_total_relation_size('product_picture'));
SELECT pg_size_pretty(pg_total_relation_size('product_state_type'));
SELECT pg_size_pretty(pg_total_relation_size('pg_largeobject'));
SELECT pg_size_pretty(pg_total_relation_size('pg_largeobject_metadata'));
```

Lisa 8 – SQL päringud

P1: Leida kõige kallim toode ja kõik selle toote pildid ning sorteerida tulemus tootekoodi ja pildi järjekorranumbri järgi kasvavas järjekorras.

Bytea:

```
SELECT p.*, LEFT(ENCODE(r.picture, 'hex'), 30)
FROM Product p
LEFT JOIN Product_picture r
USING (product_id)
WHERE price = (
    SELECT MAX(price) AS Max
    FROM Product)
ORDER BY p.product_code, r.seq_nr;
```

Suurobjekt:

```
SELECT p.*, r.picture_oid
FROM Product p
LEFT JOIN Product_picture r
USING (product_id)
WHERE price = (
    SELECT MAX(price) AS Max
    FROM Product)
ORDER BY p.product_code, r.seq_nr;
```

Failisüsteem ja pilvelahendus:

```
SELECT p.*, r.picture_url
FROM Product p
LEFT JOIN Product_picture r
USING (product_id)
WHERE price = (
    SELECT MAX(price) AS Max
    FROM Product)
ORDER BY p.product_code, r.seq_nr;
```

P2: Leida kõik aktiivsed tooted ehk tooted, mille toote seisundi liik on aktiivne. Päringu tulemusena väljastada toote id, tootekood, toote nimetus ja toote kõige väiksema järjekorranumbriga pilt. Tulemus sorteerida tootekoodi järgi kasvavas järjekorras.

Bytea:

```
SELECT p.product_id, p.product_code, p.title, LEFT(ENCODE(r.picture, 'hex'),
30)
FROM Product p
LEFT JOIN Product_picture r
USING (product_id)
WHERE p.product_state_type_code = 2
    AND (r.seq_nr = (
        SELECT Min(seq_nr) AS mi
        FROM Product_picture AS min_r
        WHERE r.product_id = min_r.product_id)
    OR r.seq_nr IS NULL)
ORDER BY p.product_code;
```

Suurobjekt:

```
SELECT p.product_id, p.product_code, p.title, r.picture_oid
FROM Product p
LEFT JOIN Product_picture r USING (product_id)
WHERE p.product_state_type_code = 2
    AND (r.seq_nr = (
        SELECT Min(seq_nr) AS mi
        FROM Product_picture AS min_r
        WHERE r.product_id = min_r.product_id)
    OR r.seq_nr IS NULL)
ORDER BY p.product_code;
```

Failisüsteem ja pilvelahendus:

```
SELECT p.product_id, p.product_code, p.title, r.picture_url
FROM Product p
LEFT JOIN Product_picture r
USING (product_id)
WHERE p.product_state_type_code = 2
    AND (r.seq_nr = (
        SELECT Min(seq_nr) AS mi
        FROM Product_picture AS min_r
        WHERE r.product_id=min_r.product_id)
    OR r.seq_nr IS NULL)
ORDER BY p.product_code;
```

P3: Leida kõik tooted. Päringu tulemusena esitatada toote id, tootekood, toote nimetus, toote seisundi liigi nimetus ja lisaks tuua välja toote piltide arv. Juhul, kui tootel pilte ei ole, märkida toote piltide arvuks 0. Tulemus sorteerida tootekoodi järgi kasvavas järjekorras.

Kõikide disainide puhul:

```
SELECT p.product_id, p.product_code, p.title, s.title AS
product_state_type_title, Count(r.product_id) AS nr_of_pictures
FROM Product_state_type s
INNER JOIN Product p
USING(product_state_type_code)
LEFT JOIN Product_picture r
USING (product_id)
GROUP BY p.product_id, p.product_code, p.title, s.title
ORDER BY p.product_code;
```

P4: Leida kõige suurema piltide arvuga toodete identifikaatorid.

Kõikide disainide puhul:

```
SELECT product_id
FROM Product_picture
GROUP BY product_id
HAVING Count(*)>=ALL (
    SELECT Count(*) AS Num
    FROM Product_picture
    GROUP BY product_id);
```

P5: Leida tootepiltide koguarv.

Kõikide disainide puhul:

```
SELECT Count(*) AS num FROM Product_picture;
```