TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Grete Saar IABB185537

Relatsioonilise ja mitterelatsioonilise andmebaasi jõudluse ristanalüüs relatsioonilise ja mitterelatsioonilise andmemudeliga PostgreSQL ja MongoDB näitel

Bakalaureusetöö

Juhendaja: Jakob Jõgi, MSc
Kaasjuhendaja: Mart Roost, MSc

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: Grete Saar

18.05.2021

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on tekitada relatsioonilise ja mitterelatsioonilise andmebaasi jõudluste võrdlusmoment PostgreSQL-i ja MongoDB näitel, rakendades mõlemas andmebaasisüsteemis nii relatsioonilist kui mitterelatsioonilist andmemudelit.

Võrdlusmomendi tekitamiseks viiakse läbi eksperiment, kus PostgreSQL-i ja MongoDB andmebaasides implementeeritakse nii relatsiooniline kui mitterelatsiooniline andmemudel ning simuleeritakse reaalseid töökoormusi nendes süsteemides.

Tulemuseks on valitud karakteristikute põhjal tekkiv võrdlusmoment, mille alusel hinnatakse andmebaasisüsteemide jõudlust ning analüüsitakse lahenduste paindlikkust ja skaleeritavust üldisemalt.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 26 leheküljel, 8 joonist, 1 tabelit.

Abstract

Cross-Analysis of Relational and Non-Relational Database Performance with Relational and Non-Relational Data Model on the Example of PostgreSQL and MongoDB

The aim of this thesis is to create a comparison between relational and non-relational database performances on the example of PostgreSQL and MongoDB by applying relational and non-relational data models in both systems.

To generate the reference moment, an experiment is conducted in which a relational and non-relational data model is implemented in PostgreSQL and MongoDB and real workloads in these systems are simulated.

The result is a reference moment based on the selected characteristics, from which conclusions are drawn in terms of database performance. Flexibility and scalability of the solutions in general are analyzed.

The thesis is written in Estonian and contains 26 pages of text, 8 figures, 1 table.

Lühendite ja mõistete sõnastik

Analüütiline andmebaas	Andmebaas, kus hoitakse staatilisi andmeid [1]
Andmemudel	Mudel, mis kirjeldab andmete struktureerimise viisi andmebaasis
API	*Application Programming Interface* – rakendusliides, meetodite ja parameetrite kollektsioon, mida üks tarkvara kasutab toimingute taotlemiseks teiselt tarkvaralt [2]
BSON / *Binary* JSON	JSON-formaadis andmete binaarne esitus MongoDB-s
CRUD operatsioonid	Andmesalvestuse põhioperatsioonid – sisestamine (*Create*), lugemine (*Read*), uuendamine (*Update*), kustutamine (*Delete*)
DDL	*Data Definition Language* – andmete määratlemise keel, mille käske kasutatakse andmebaaside ja tabelite haldamiseks [3]
DML	*Data Manipulation Language* – andmete manipuleerimise keel, mille käske kasutatakse andmebaasis andmete sisestamiseks, kustutamiseks, uuendamiseks ja valimiseks [3]
JSON	*JavaScript Object Notation* – standardvorming andmete salvestamiseks ja vahetamiseks
JSONB / JSON *Binary*	JSON-formaadis andmete binaarne esitus PostgreSQL-is
OLAP	*Online Analytical Processing* – veebianalüütiline töötlus, analüütilisi töötlussüsteeme kasutatakse organisatsiooni tulemuslikkuse analüüsi ja strateegilise planeerimise toetamiseks [4]
OLTP	*Online Transacion Processing* – veebitehingute töötlus, tehingute töötlussüsteemid tegelevad organisatsiooni igapäevaste tehingute töötlemisega [4]
OOME	*Out Of Memory Exception* – erand, mis esineb kui programmi täitmiseks pole piisavalt mälu [5]

Operatiivne andmebaas	Andmebaas, kus hoitakse dünaamilisi/pidevalt muutuvaid andmeid [1]
RAM	*Random Access Memory* – arvuti füüsiline mälu ehk muutmälu
SQL	*Structured Query Language* – Struktureeritud päringukeel eelkõige relatsiooniliste andmebaaside haldamiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Relatsioonandmebaasid on olnud andmesalvestuse *de facto* standard 1970. aastate algusest [6]. Andmete hoiustamist relatsioonilises andmebaasis nähti kui ainuõiget lähenemist, kuid 2005. aastast hakkas relatsiooniliste andmebaaside ülemvõimu aeg lõppema [7] ning esile kerkisid alternatiivse andmesalvestusviisiga eksperimenteerivad mitterelatsioonilised andmebaasid. Nende tõus on viinud selleni, et relatsioonandmebaase ei nähta enam kui ainuvõimalikku viisi andmete salvestamiseks ja töötlemiseks. Seega erinevates olukordades võidakse kasutada erinevaid andmehoidlaid. Seda vaatepunkti nimetatakse polügloti püsivuseks (*polyglot persistence*). Selleks, et otsustada millist andmebaasi kasutada, peab esmalt mõistma salvestatavate andmete olemust ning seda, kuidas andmeid on vaja manipuleerida [8].

Andmebaasi ja -mudeli valikul tuleks lähtuda esmalt ärivajadustest. Fikseerida, millised on nõuded andmebaasile, kui skaleeritav see peab olema, kas tegemist on pigem staatilise andmebaasiga, kust on vaja andmeid peaasjalikult lugeda ning vähem manipuleerida või on oluline nii kiire andmete sisestamine, lugemine kui muutmine. Sealjuures tuleks arvestada ka andmete kvaliteedinõuetega.

Vajadustest ei pruugi alati ilmneda, milline andmebaas sobib antud olukorras kõige paremini ning sellisel juhul tuleks potentsiaalseid valikuid põhjalikumalt uurida. Üks võimalik variant on simuleerida olemasolevaid töökoormusi erinevates andmebaasisüsteemides ning mõõta ja võrrelda olulisi karakteristikuid nagu näiteks päringu kiirus, protsessori koormus, *etc*. Eelkõige jõudluse ja mastaapsusega seotud probleemid ilmnevad alles mõne aja pärast, kui andmemahud suurenevad ja transaktsioonide arv kasvab. Võimalike probleemide tuvastamiseks on kõige parem viia läbi pikaajalised ja võimalikult täpsed reaalse tootmise seadistusega simulatsioonid. Reaalse töökoormuse simuleerimine aitab paremini mõista, kuidas antud andmebaas toimib [9].

Töö teema tuleneb autori töökohast, täpsemalt ehitustarkvara loovast ettevõttest, mis pakub strateegilist pilvelahendust BIM-põhise ressursside planeerimise, eelarvestamise, tootmise ja hoonete ehitamise jaoks. Probleem seisneb selles, et hetkel kasutatakse antud ettevõttes andmete hoiustamiseks relatsioonilist andmebaasi, kuid andmemudeli kiirest kasvust tulenevalt on vaja otsustada, kas jätkata olemasoleva lahendusega, luua täiesti uus andmemudel ja vahetada andmebaasi või leida mõni hübriidlahendus. Ettevõtte ärilistest vajadustest tulenevalt peab alternatiivne lahendus olema skaleeruv ning andmebaasisüsteem piisavalt kiire ka suurte andmemahtude juures.

Käesoleva töö eesmärk on tekitada relatsioonilise ja mitterelatsioonilise andmebaasi jõudluste võrdlusmoment PostgreSQL-i ja MongoDB näitel, kasutades mõlemas süsteemis nii relatsioonilist kui mitterelatsioonilist andmemudelit. Taolise ristanalüüsi eesmärk laiemalt on paremini mõista andmebaaside võimekust nii normaalolukorras, kus andmeid hoitakse baasis kujul, mille jaoks see baas loodud on, kui ka olukorras, kus andmeid hoitakse viisil, mis ei ole andmebaasis algselt ette nähtud.

Autori hüpoteesiks on, et andmebaaside jõudlus väheneb, kui rakendada alternatiivseid andmemudeleid, kuid mõju andmebaasi kiirusele on piisavalt väike, et teatud olukordades võib alternatiivsete lahenduste implementeerimine olla õigustatud valik.

Töö käigus viiakse läbi eksperiment, kus Dockeri konterineritesse luuakse kaks PostgreSQL-i ja kaks MongoDB andmebaasi, mõlemas baasis implementeeritakse nii relatsiooniline kui dokumendipõhine andmemudel. Seejärel käivitatakse loogikakihti loodud rakendusliidesega asünkroonsed CRUD operatsioonid, mis simuleerivad reaalse elu kasutust.

Eksperimendi läbiviimiseks otsustati kasutada Dockeri konteinereid, sest andmebaaside konteineriseerimine võimaldab süsteeme jooksutada samas masinas ilma, et need üksteist mõjutaksid.

Eksperimendi tulemuste põhjal tehakse järeldused andmebaaside jõudluste seisukohalt, lähtudes numbrilistest tulemitest, lisaks analüüsitakse lahenduste paindlikkust ja skaleeruvust üldisemalt. Eksperimendi tulemuste ja implementatsioonide laialdasemale analüüsile tuginedes tehakse ettepanek ettevõttele, lähtudes selle ärilistest vajadustest.

Töö struktuur on järgmine. Esmalt antakse ülevaade relatsioonilistest ja mitterelatsioonilistest andmebaasidest ning nende jõudlusega seotud varasematest uuringutest. Seejärjel kirjeldatakse eksperimendi metoodikat, kasutatud tööriistu ja läbiviidud simulatsiooni sisu. Kolmanda peatüki all esitletakse eksperimendi tulemusi, millele tuginedes tehakse järeldused. Seejärel analüüsitakse saadud tulemuste sisu ning tehakse kokkuvõte.

2 Kirjanduse ülevaade

Andmebaasihalduses liigitatakse andmebaasid operatiivseteks ja analüütilisteks. Operatiivses andmebaasis hoitavad andmed on dünaamilised ning muutuvad pidevalt. Sellist tüüpi andmebaase kasutatakse üldiselt veebipõhiste tehingute töötlemise (OLTP – *Online Transaction Processing*) olukordades, kus on vaja igapäevaselt andmeid koguda ja muuta. Analüütilist andmebaasi kasutatakse peamiselt veebipõhise analüütilise töötlemise (OLAP– *Online Analytical Processing*) situatsioonides, kus on vaja säilitada ja jälgida ajaloolisi ja ajast sõltuvaid andmeid [1]. Relatsioonilised andmebaasid töötati välja just veebipõhiste tehingute töötlemiseks [4].

2.1 Relatsiooniline andmebaas ja -mudel

Relatsiooniline andmebaas põhineb relatsioonilisel mudelil ehk baasi loogiline struktuur koosneb relatsioonide kogumist. Relatsioonilise andmemudeli kontseptsiooni esitas 1970. aastal Edgar Frank Codd. Sünonüümselt võib relatsiooni nimetada ka tabeliks, mille struktuur koosneb olemi atribuutidest ja andmekogumitest, teisisõnu veergudest ja ridadest, kus iga veerg saab hoida ainult ühte tüüpi andmeid [4].

Relatsioonandmebaasi kasutamisel on omad eelised, näiteks sisseehitatud mitmetasandiline terviklikkus, andmete järjepidevuse tagatus ning lihtne andmete otsimine. Mudelisse on integreeritud kontrollid tabeli välja, tabeli ja tabelite vahelise seose tasandil. Nimetatud kontrollide eesmärk on tagada andmete täpsus ja tabeliseoste kehtivus ning vältida kirjete dubleerimist [1].

Struktureeritud päringukeel (SQL – *Structured Query Language*) on standardkeel, mida kasutatakse relatsiooniliste andmebaaside loomiseks, muutmiseks, hooldamiseks ning andmepäringute tegemiseks [1]. Andmebaaside ja tabelite haldamiseks kasutatakse SQL-i DDL (*Data Definition Language*) käske ning andmete sisestamiseks, küsimiseks, muutmiseks ja kustutamiseks kasutatakse DML (*Data Manipulation Language*) käske [3].

2.1.1 PostgreSQL

PostgreSQL on laialdaselt kasutatav avatud lähtekoodiga relatsiooniline andmebaas, mis pakub kasutajatele kindlat stabiilsust, mastaapsust ning andmete turvalisust [3].

2.1.2 JSON ja JSONB PostgreSQL-is

Aastal 2012 versiooniga 9.2 lainedas PostgreSQL oma paindlikkust ning lisas võimaluse hoiustada andmeid JSON formaadis [10]. Versiooniga 9.4 aastal 2014, lisati JSON tüüpi andmete salvestamiseks võimekam ja tõhusam andmetüüp JSONB, mis toetab ka indekseerimist [11].

JSON tüüpi veerud PostgreSQL-is on mõeldud hoiustama JSON tüüpi andmeid. JSON tüüpi andmeid on võimalik salvestada kahes erinevas formaadis – JSON või JSONB. JSON kujul salvestamine tähendab, et baasis hoiustatakse sisendi tekstist täpset koopiat. See aga tähendab, et iga operatsiooni korral on vaja salvestatud tekst parsida ehk viia see loetavale kujule. Seevastu JSONB tüüpi andmeid säilitatakse parsitud kujul binaarformaadis (JSON *Binary*). Viimasel juhul võib baasi kirjutamise kiirus küll aeglustuda, seevastu andmete küsimine toimub kiiremini. Lisaks toetab JSONB paindlikumaid andmete manipuleerimise funktsioone. PostgreSQL-i ametlik dokumentatsioon soovitab kasutada pigem JSONB andmeformaati, mistõttu kasutati ka käesolevas eksperimendis PostgreSQL-is JSON tüüpi andmete salvestamiseks JSONB andmetüüpi.

Lähtudes andmebaasi kasutatavusest, JSON ja JSONB andmetüüpide toetamisest ning nendega seotud funktsioonide ja operaatorite laialdasest valikust, otsustati antud eksperimendis kasutada PostgreSQL andmebaasi kui relatsioonilise andmebaasi esindajat.

2.2 Mitterelatsiooniline andmebaas ja -mudel

Mitterelatsioonilised andmebaasid hakkasid populaarsust koguma 2000. aastate alguses [7], kui IT-organisatsioonid seisid vastamisi kasvava andmehulgaga ning suurenes vajadus hoida andmeid struktureerimata kujul [12].

Mitterelatsioonilise andmebaasi kohta kasutatakse ka terminit NoSQL. See termin tuleneb asjaolust, et andmebaas ei kasuta päringukeelena SQL-i [8]. Sisuliselt oleks NoSQL andmebaase korrektsem kutsuda mitterelatsioonilisteks (*NoRelational*), sest need vastanduvad relatsioonmudelile endale, mitte SQL päringukeelele [12].

Mitterelatsioonilistes andmebaasides on kasutusel erinevad mudelid, populaarsemad on võti-väärtus (*key-value*), dokumendi- ja graafimudelid [8]. Dokumendimudelit kasutavates andmebaasides hoitakse andmeid dokumendi kujul, enamasti JSON formaadis. JSON formaadis dokumendid toetavad keerukamaid andmestruktuure, näiteks alamobjekte või objektide kollektsioone, mis tähendab, et dokumendid on palju paindlikumad ning võimaldavad hoida andmeid struktureerimata kujul. Käesolevas töös valiti mitterelatsioonilise mudeli esindajaks dokumendimudel.

2.2.1 MongoDB

MongoDB on dokumendipõhise disainiga mitterelatsiooniline andmebaas, kus saab hoida nii struktureeritud kui struktureerimata andmeid. MongoDB andmebaas koosneb kollektsioonidest ning igas kollektsioonis võib olla üks või mitu dokumenti. Kollektsioonid ja dokumendid on analoogsed tabelitele ja ridadele relatsioonilises andmebaasis [13].

Kuna MongoDB on avatud lähtekoodiga ning hetkel kõige populaarsem dokument-orienteeritud andmebaas [14], otsustati seda kasutada antud eksperimendis kui mitterelatsioonilise andmebaasi esindajat.

2.2.2 JSON ja BSON MongoDB-s

JSON on MongoDB-s dokumendimudeli primaarne andmete esitamise viis, ent kuna JSON on tekstipõhine, siis on andmete parsimine aeglane ning hoiustamine mahukas. Parema jõudluse tagamiseks kasutab MongoDB andmete hoiustamiseks BSON (*Binary* JSON) formaati, mis on JSON formaadis andmete binaarne esitus. PostgreSQL-i JSONB ja MongoDB BSON formaadid on oma olemuselt samaväärsed.

2.3 Varasemad uuringud

2013. aastal toimunud rahvusvahelisel intelligentsete andmete ja veebitehnoloogiate konverentsil esitlesid Zhao *et al.* uuringut [15], milles püstitati küsimus, kas relatsioonandmebaasist on võimalik andmeid MongoDB-sse migreerida. Lähtuti asjaolust, et kõik SQL operatsioonid põhinevad relatsioonalgebral ehk relatsiooniline mudel esindab relatsioonilise andmebaasisüsteemi võimekuskomplekti. Seega kui MongoDB pakub sama võimekuskomplekti, mis relatsiooniline andmebaas, võib järeldada, et andmeid on võimalik migreerida relatsioonandmebaasist MongoDB-sse. Uuringus koostati relatsioonalgebra abil MongoDB matemaatiline mudel ning leiti, et on võimalik luua MongoDB eksemplar relatsioonmudeli põhjal, mis pakub sama andmehalduse ja päringu funktsionaalsust, mis relatsiooniline andmebaas.

Uuringus [16] vaatlesid autorid füüsilise andmebaasi disainivalikute mõju relatsioonilise ja mitterelatsioonilise süsteemi jõudlusele. Fookus seati JSON andmemudelile ning sellele, kuidas selle kasutamine relatsioonilises andmebaasis PostgreSQL näitel mõjutab baasi jõudlusvõimet võrreldes relatsioonilise mudeli kasutamisega PostgreSQL-is ja dokumendipõhise mudeli kasutamisega MongoDB-s. Nimetatud uuringus keskenduti rohkem füüsilise disaini valikutele ning sellega seotud erinevate muutujate mõjule, mitte spetsiifiliselt CRUD operatsioonide kiirustele. Tulemusena toodi välja, et kuigi JSON-i formaadis andmete hoiustamine PostgreSQL-is võtab rohkem ruumi võrreldes MondoDB-ga, on PostgreSQL-i JSON andmete pärimine siiski kiirem MongoDB-st.

2015. aasta rahvusvahelisel juhtimissüsteemide ja teaduse konverentsil esitlesid Boicea *et al.* uuringut [17], kus võrreldi mitterelatsiooniliste dokument-orienteeritud ja relatsiooniliste andmebaaside CRUD operatsioonide kiirust nii hajutatud kui üksikus andmebaasi haldussüsteemis. Antud uuringus rakendati mitterelatsioonilistes süsteemides dokumendipõhist mudelit ning relatsioonilistes andmebaasisüsteemides relatsioonilist mudelit. Tulemuste üldises järelduses leiti, et mitterelatsiooniliste süsteemide jõudlus on parem võrreldes relatsiooniliste süsteemidega.

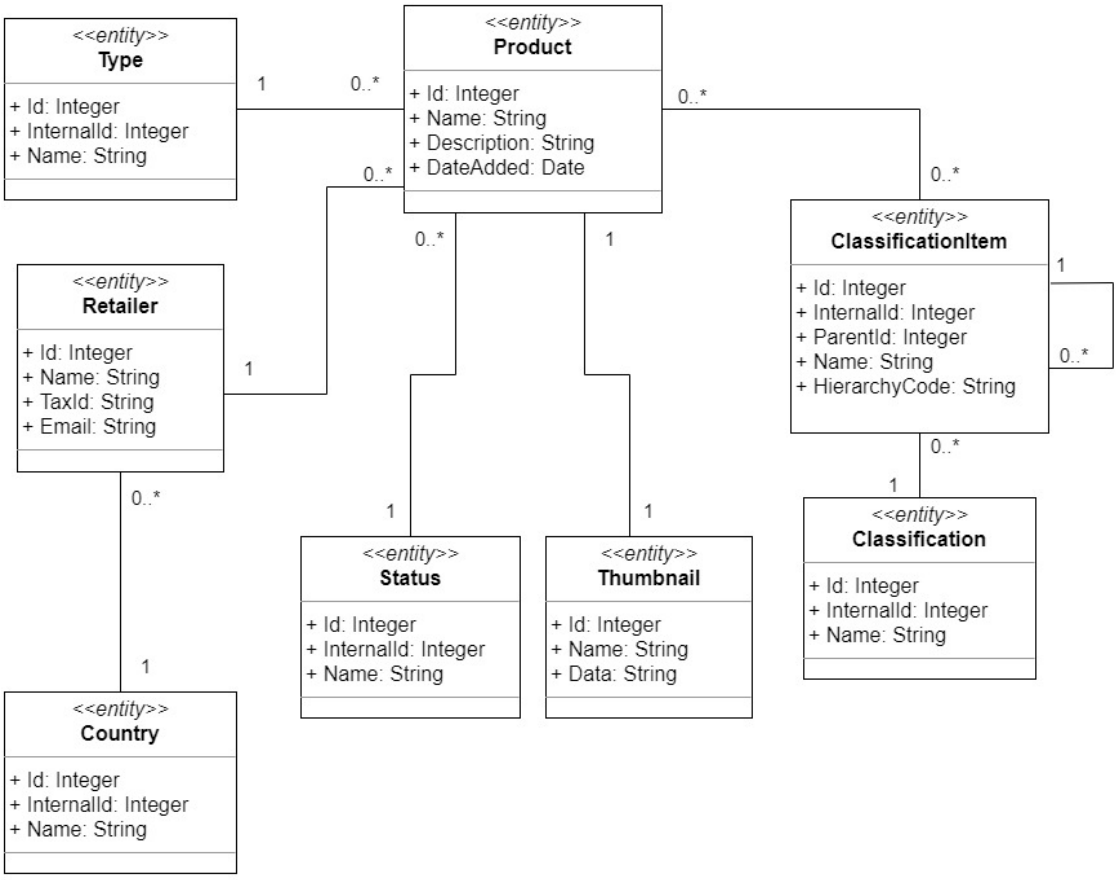
Tallinna Tehnikaülikooli magistritöös [18] mõõdeti PostgreSQL ja MongoDB jõudlusvõimet, kasutades PostgreSQL-is nii JSON kui JSONB tüüpi veerge. Reaalse Java rakenduse näitel läbiviidud jõudluse testide tulemused näitasid, et MongoDB-s on

andmete kirjutamise operatsioonid kordades kiiremad kui PostgreSQL-is (JSON tüüpidega). Täheldati ka seda, et standardse disainiga PostgreSQL-i jõudlus oli madalam võrreldes JSON tüüpe kasutava PostgreSQL-iga.

Kuigi analoogseid andmebaaside jõudluse võrdluseid on läbi viidud rohkem kui antud töös välja toodi, siis pole teadaolevalt ükski varasem uuring kõrvutanud korraga relatsioonilist ning mitterelatsioonilist andmebaasi, kus on rakendatud nii relatsiooniline kui dokumendipõhine andmemudel. Eelnevalt läbiviidud uuringutes on jõudluse võrdlemiseks valitud vaid teatud alamhulk nimetatud implementatsioonidest ehk võrldlusmomendi tekitamisel on üks kahest andmebaasist pandud olukorda, mille jaoks see loodud ei ole, samas teine andmebaas on olukorras, mille jaoks see loodud oli. Lisaks tuleks jõudluse probleemide tuvastamiseks läbi viia võimalikult reaalset olukorda peegeldav simulatsioon ning vältida toiminguid, mida tavalises andmebaasi töösituatsioonis ei tehta (näiteks vahemälu kustutamine iga operatsiooni järel, mida tehti [18] võrdlusuuringus).

3 Eksperiment

Eksperimendi aluseks võeti toote ja sellega seotud andmete hoiustamise struktuur. Tegemist on autori töökohas kasutatava mudeli lihtsustatud versiooniga, kuid taolist tootepõhist struktuuri võib teatud variatsioonides leida ka teistes ettevõtetes, kus kesksel kohal on toode ning kõik muu sellega seonduv. Joonisel 1 on kujutatud antud eksperimendi jaoks loodud olemi-suhte diagramm, millele tuginedes loodi füüsilised andmemudelid.



Joonis 1. Olemi-suhte diagramm

Igal tootel on lisaks nimele, kirjeldusele ja lisamiskuupäevale küljes ka tüüp (joonisel kujutatud olem *Type*), tarnija (*Retailer*), staatus (*Status*), pilt (*Thumbnail*) ja klassifikaatori kirje (*ClassificationItem*). Iga tarnija on omakorda seotud ühe riigiga (*Country*) ning iga klassifikaatori kirje kuulub ühe klassifikatsiooni süsteemi alla (*Classification*).

Klassifikatsiooni süsteemi alla kuuluvad klassifikaatorid on kirjeldatud puu-struktuurina. Igal klassifikaatoril on küljes viide millise klassifikaator alla see kuulub. Olukorras, kus rakenduses on vaja kuvada kõiki valitud klassifikaatori alla kuuluvaid tooteid, on päringute lihtsustamiseks klassifikaatori küljes ka hierarhia koodi atribuut sõne (*string*) formaadis. Nimetatud väli koosneb kõikidest selle klassifikaatori vanemtippude identifikaatoritest (vt Tabel 1).

Tabel 1. Klassifikaatori hierarhiakoodi näide

Id	**Nimi**	**Hierarhiakood**
1	Sise- ja viimistlustooted	
1.1	Põrandakatted	1/
1.1.1	Puitpõrandad	1/1.1
1.1.2	Plaatpõrandad	1/1.1
1.2	Seinakatted	1/

3.1 PostgreSQL-i relatsiooniline mudel

PostgreSQL-i relatsioonilises mudelis (vt Lisa 2) kirjeldati üheksa tabelit: *Product*, *ProductClassificationItem*, *ClassificationItem*, *Classification*, *Type*, *Status*, *Thumbnail*, *Retailer*, *Country*.

Igas tabelis defineeriti olemi atribuudid, unikaalne primaarvõti ning välisvõtmed relatsioonidevaheliste seoste loomiseks. Unikaalseks primaarvõtmeks määrati olemi automaatne inkrementaalne *Id* atribuut.

3.2 PostgreSQL-i mitterelatsiooniline mudel

PostgreSQL-i mitterelatsioonilises ehk denormaliseeritud mudelis (vt Lisa 3) defineeriti sarnaselt relatsioonilise mudeliga staatiliste andmete tabelid – *Classification*, *ClassificationItem*, *Country*, *Status* ja *Type*. Tabeli *Product* atribuutideks olid *Id* ja *Data*, viimase andmetüübiks määrati JSONB. *Data* veerg hoidis kogu tootega seotud informatsiooni JSONB formaadis.

3.3 MongoDB relatsiooniline mudel

MongoDB relatsiooniline mudel (vt Lisa 4) loodi sarnaselt PostgreSQL-i relatsioonmudelile. Defineeriti üheksa kollektsiooni ning vastavad olemi atribuudid.

Dokumendi loomisel genereerib MongoDB vaikimisi *_id* välja, mis on ühtlasi dokumendi primaarvõtmeks [13]. Nimetatud välja väärtuseks on unikaalne ObjectId, mis koosneb kolmest osast – ObjectId loomise ajahetk (suurus 4 baiti), juhuslik väärtus (5 baiti) ning kasvav väärtus, mis lähtub juhuslikust väärtusest (suurus 3 baiti) [19]. Seda *_id* välja kasutati kollektsioonide vaheliseks viitamiseks.

3.4 MongoDB mitterelatsiooniline mudel

MongoDB mitterelatsioonilises mudelis (vt Lisa 5) defineeriti staatiliste andmete kollektsioonid – *Classification*, *ClassificationItem*, *Country*, *Status* ja *Type*. Sarnaselt PostgreSQL-i mitterelatsioonilise mudeliga loodi ka siin *Product* kollektsioon ning dokumendil defineeriti kõik tootega seotud atribuudid.

3.5 Docker

Docker on avatud platvorm rakenduste arendamiseks ja käitamiseks nõrgalt isoleeritud keskkonnas, mida nimetatakse konteineriks. Konteineriseerimine võimaldab korraga jooksutada mitut erinevat rakendust ilma, et need üksteist mõjutaksid [20]. Konteiner põhineb Dockeri tõmmisfailil (*Docker image*), mis sisaldab kogu rakenduse käivitamise jaoks vajalikku informatsiooni. Tõmmist saab alla laadida registritest, levinuim register

on Docker Hub. Tõmmis laetakse kohalikku Dockeri masinasse, kus seejärel saab seda kasutada ühe või mitme konteineri käivitamiseks [21].

Antud eksperimendis loodi iga implementatsiooni jaoks eraldi konteiner. Kahe konteineri loomisel kasutati avalikus registris oleva PostgreSQL tõmmise kõige hilisemat versiooni (13.3) ning järgneva kahe konteineri loomisel avaliku registri MongoDB tõmmise kõige hilisemat versiooni (4.4). Kasutati andmebaaside vaikimisi seadeid, ühtegi tõmmisfaili ei kohandatud.

Vaikimisi pole Dockeri konteineritel mälupiiranguid ehk Docker võib süsteemilt küsida nii palju ressurssi, kui kernel parasjagu ajastada lubab. Kui Linux masinas tuvastatakse, et oluliste süsteemifunktsioonide täitmiseks pole enam piisavalt mälu, sest töötav konteiner kasutab liiga palju masina ressursse, annab süsteem *Out Of Memory Exception*-i (OOME). Selle tagajärjel alustab süsteem protsesside katkestamist, et mälu vabastada [22]. Vältimaks OOME võimalikku esinemist, määrati igale konteinerile mälukasutuse piirang 4GB.

Simulatsioon viidi läbi Surface Book 3 sülearvutiga, mille installeeritud füüsilise mälu maht (RAM – *Random Access Memory*) oli 32GB. Protsessori (Intel Core i7-1065G7, 4 tuuma, 8 loogilist protsessorit) kasutust ei piiratud.

3.6 Vahemälu

PostgreSQL vajab operatsioonisüsteemi protsesside vahelise kommunikatsiooni (IPC – *Inter-Process Communication*) funktsioone, üks nendest on jagatud mälu. Jagatud mälu, ehk *shared_buffers* parameeter määrab mälumahu, mida andmebaasiserver jagatud mälupuhvrite jaoks kasutab. Jagatud mälupuhvri vaikimisi suurus PostgreSQL-is on 128MB [23].

Alates versioonist 3.2 kasutab MongoDB vaikimisi WiredTiger mälumootorit. Koos sellega kasutab MongoDB ka WiredTigeri sisemist ning failisüsteemi vahemälu. Sisemise vahemälu vaikimisi suuruseks on kas 50% RAM - 1GB või 256MB. Juhul kui andmebaas töötab konteineris, millele on seatud mälupiirang mis on väiksem kui kogu

süsteemimälu, kasutatakse maksimaalset saadaolevat RAM-i antud mälupiirangu suhtes [24].

Lähtudes eesmärgist simuleerida reaalse rakenduse olukordi, ei kustutatud iga CRUD operatsiooni järgselt andmebaasi vahemälu. Ka kummagi andmebaasisüsteemi vahemälu suurust ei muudetud, kasutati süsteemide vaikimisi väärtusi.

3.7 Staatiliste andmete import

Staatilisteks andmeteks olid riigid, toote tüübid, staatused, klassifikaatorikirjed ja klassifikatsiooni süsteemid, mis olid kirjeldatud eraldi csv failides. Enne CRUD operatsioonide käivitamist imporditi nimetatud andmed ning klassifikaatoritele lisati hierarhiakood. Staatilisi andmeid simulatsiooni käigus ei muudetud.

3.8 Skeemi valideerimine

Relatsioonandmebaasis tuleb enne andmete salvestamist määratleda, milline on andmete struktuur (sh millised on andmeüksuste atribuudid ja tüübid), seda määratlust nimetatakse skeemiks (*Schema*). Seevastu mitterelatsioonilised andmebaasid võimaldavad andmeid salvestada ka ilma skeemita [13]. Kuna MongoDB võimaldab valikuliselt skeemi valideerimist [25], loodi skeem ka MongoDB relatsioonilistele kollektsioonidele. Skeemi kasutamine tagab võrdse lähteülesande keerukuse mõlemale relatsioonilisele implementatsioonile.

3.9 Rakendusliides

CRUD operatsioonide tegemiseks loodi eraldi Node.js rakendusliides ehk API (*Application Programming Interface*), mille käivitamisel loodi ühendus andmebaasiga, imporditi staatilised andmed ning seejärel käivitati automaatselt asünkroonsed andmebaasi päringud. Lisa 6-s on toodud asünkroonsete CRUD operatsioonide käivitamise koodinäide loodud rakendusliidesest.

3.9.1 Andmete sisestamine (*Create*)

Relatsiooniliste implementatsioonide korral sisestati andmeid *Product*, *ProductClassificationItem*, *Retailer* ja *Thumbnail* tabelitesse/kollektsioonidesse. Dokumendipõhistes implementatsioonides sisestati andmeid ainult *Product* tabelisse/kollektsiooni.

3.9.2 Andmete lugemine (*Read*)

Kokku loodi kuus erinevat lugemisoperatsiooni:

- Juhuslikult valitud riigi põhjal kõikide toodete otsimine läbi nendega seotud tarnijate
- Juhuslikult valitud klassifikaatori hierarhia koodi põhjal toodete otsimine ehk tulemuseks saadi kõik tooted, mis olid seotud kas antud klassifikaatoriga või mõne selle alamklassifikaatoriga
- Kõikide klassifikaatorita toodete lugemine
- Kõikide pildiga toodete lugemine
- Toodete koguarvu lugemine
- Tarnijate koguarvu lugemine

3.9.3 Andmete uuendamine (*Update*)

Andmete uuendamise operatsioone loodi kokku kaks:

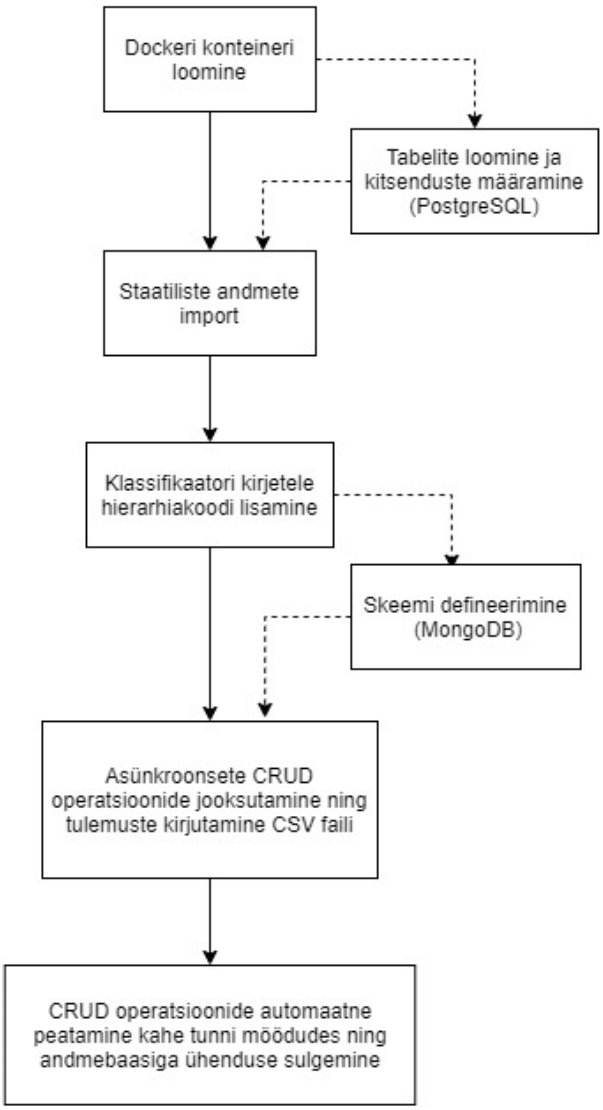
- Juhuslikult valitud toote nime uuendamine
- Toote staatuse uuendamine juhuslikult valitud staatuse ja tüübi põhjal

3.9.4 Andmete kustutamine (*Delete*)

Kustutamisoperatsiooni käigus valiti andmebaasist juhuslikult üks toode ning kustutati see koos kogu seonduva informatsiooniga

3.10 Simulatsiooni protsess

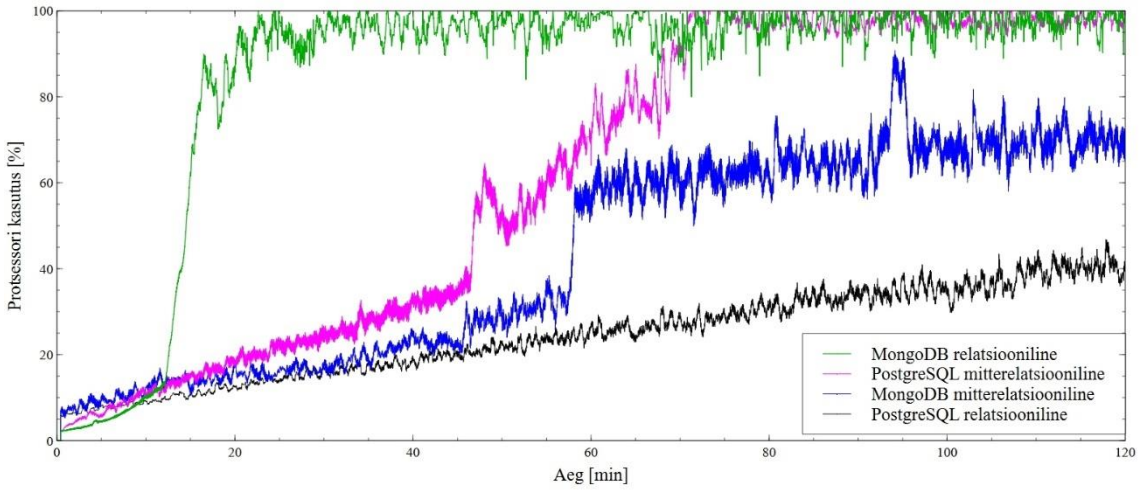
Joonisel 2 on kirjeldatud simulatsiooni protsessi. Iga simulatsioon kestis 120 minutit, mille jooksul käivitati asünkroonseid CRUD operatsioone iga teatud intervalli tagant ehk erinevaid päringuid tehti läbisegi. Iga operatsiooni juures registreeriti ajahetked, millal sooritati päring andmebaasi ning millal tuli vastus. Selle pealt arvutati päringu kestus. Protsessori ning mälukasutuse mõõdikute hankimiseks kasutati Node.js *systeminformation* moodulit (versioon 5.0). Kõik tulemused salvestati csv failidesse. Usaldusväärsete tulemuste tagamiseks jooksutati simulatsioone mitu korda ning veenduti, et tulem oli iga kord kvalitatiivselt samaväärne.



Joonis 2. Simulatsiooni protsess

4 Tulemused

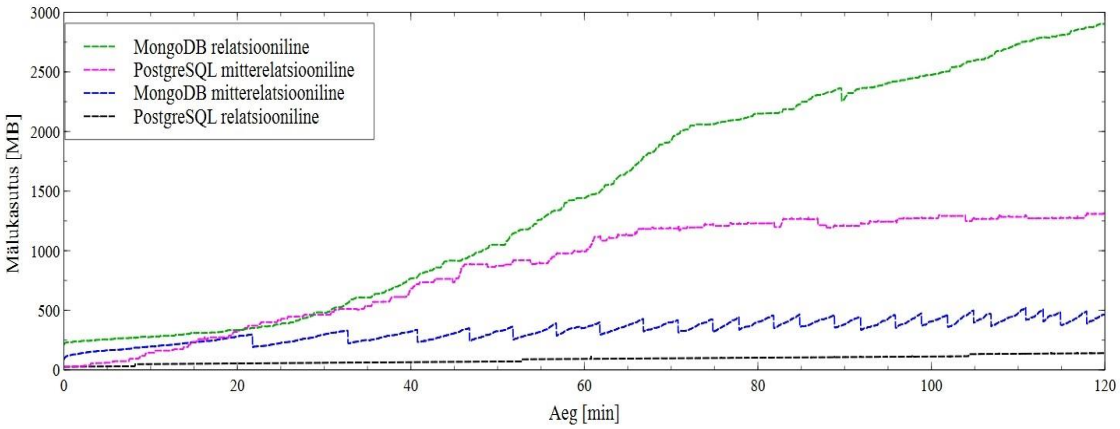
Joonis 3 kujutab protsessori kasutust ajas. Relatsioonilise MongoDB simulatsiooni 15.-ndal minutil kasvas protsessori kasutuse protsent hüppeliselt ning jõudis 100%-ni. Teisena küllastus protsessor ka mitterelatsioonilisel PostgreSQL-il. MongoDB mitterelatsioonilise implementatsiooni protsessori kasutus jäi alguses 20% ringi, hiljem kõikus 60-80% vahel. Kõige väiksem koormusprotsent oli relatsioonilisel PostgreSQL-il, mis jäi terve simulatsiooni vältel alla 50%.



Joonis 3. Protsessori kasutusprotsent ajas

Joonis 4 kujutab andmebaasi mälukasutust ajas. Simulatsiooni esimese 15 minuti sees oli mõlema MongoDB implementatsiooni mälukasutus suhteliselt stabiilne, kuid siiski suurem võrreldes PostgreSQL-iga. Samas oli PostgreSQL-i mitterelatsioonilise versiooni mälukasutuse tõusutrend võrreldes ülejäänutega märgatavalt suurem ning hetkeks ületas ka MongoDB-d.

MongoDB relatsioonilise implementatsiooni mälukasutuse järsk tõus algas 30.-ndal minutil, jõudes simulatsiooni lõpuks pea 3000 megabaidini. Mitterelatsioonilise PostgreSQL-i mälukasutus stabiliseerus 65-ndal minutil jäädes vahemikku 1190MB - 1290MB. MongoDB mitterelatsioonilise implementatsiooni mälukastus kõikus 250 ja 430 megabaidi vahel, kuid püsis võrdlemisi stabiilne ilma suuremate tõusudeta. Relatsioonilise mudeliga PostgreSQL-i mälukasutus oli ülejäänud implementatsioonidega võrreldes märkimisväärselt väiksem, jäädes kogu simulatsiooni ajal alla 150 megabaidi.

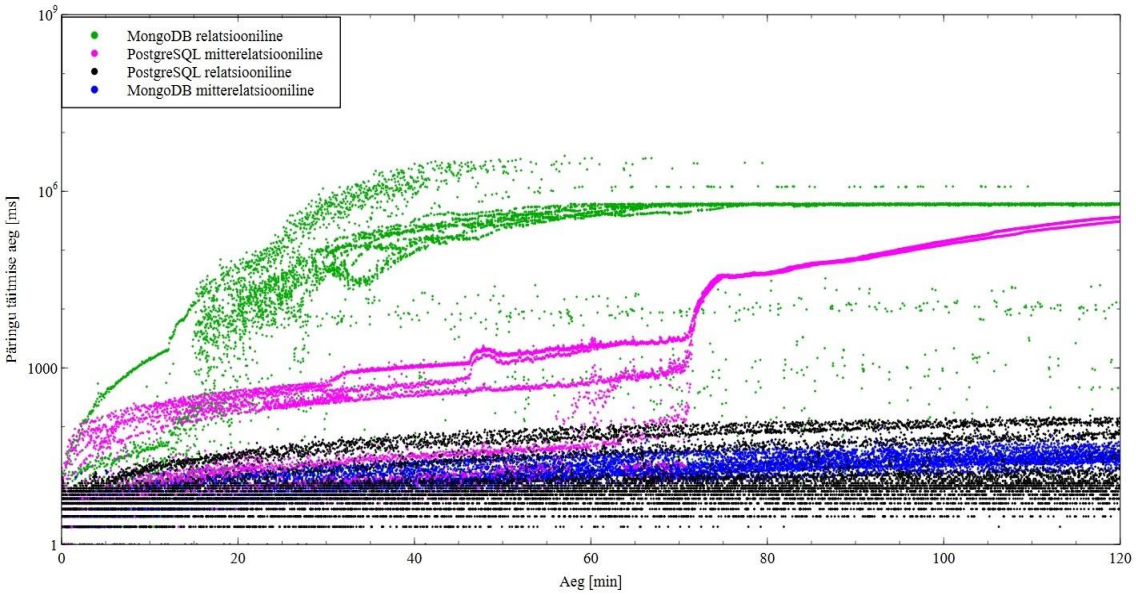


Joonis 4. Mälukasutus ajas

Joonis 5 kujutab kõikide imlplementatsioonide SELECT ehk lugemisoperatsioonide kestust ajas. Päringu andmepunkti asukoht y-teljel näitab, kui kaua päring aega võttis ning punkti asukoht x-teljel näitab, mis hetkel päring tehti.

Joonisel saab eristada andmepunktidest tekkinud selgeid jooni, mis kujutavad ühte kindlat SELECT operatsiooni. Kõige kiiremad päringud eristuvad juba simulatsiooni esimeste minutite juures, kus enamus mitterelatsioonilise MongoDB ja relatsioonilise PostgreSQL-i päringute kestused jäävad mõne millisekundi juurde. Kuigi mõned nimetatud implementatsioonide päringud andmehulga kasvades aeglustuvad (tarnijate ja toodete koguarvu ning klassifikaatorita toodete päringud) on tõus siiski võrdlemisi stabiilne ning päringute kestused jäävad alla 150 millisekundi. Mitterelatsioonilise PostgreSQL-i päringute aeg kümnekordistus 70-ndal minutil kasvades 3500 millisekundilt 35 000 millisekundile ning tõus jätkus kuni simulatsiooni lõpuni.

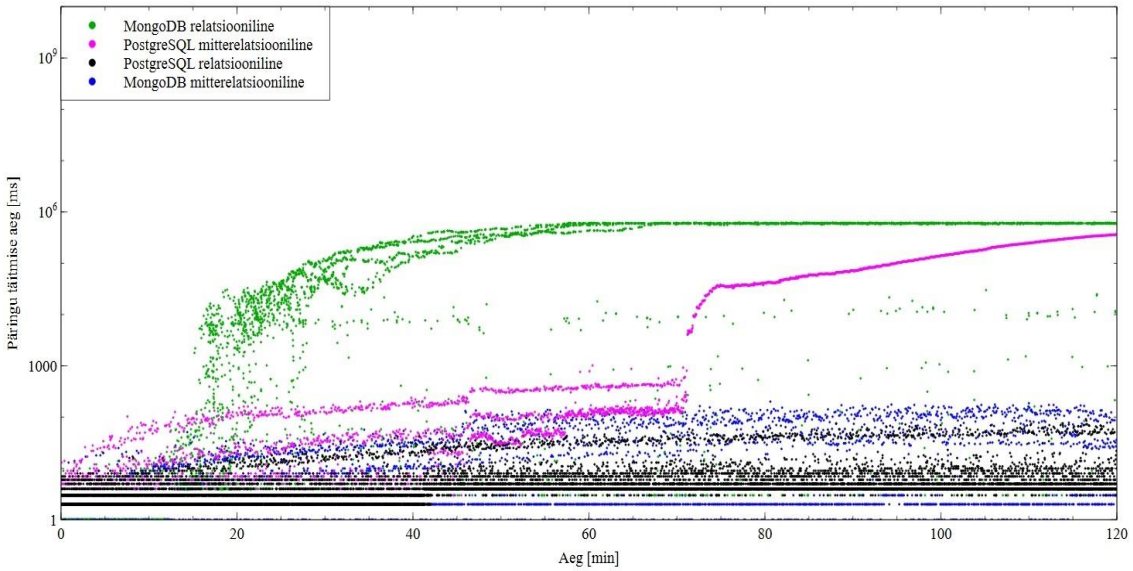
Kõige ajakulukamad olid relatsioonilise MongoDB päringud, nendest kõige kauem võtsid aega pildiga toodete ja klassifikaatorita toodete küsimine. Lisa 7-s on toodud relatsioonilise MongoDB riigi põhjal toodete lugemisoperatsiooni koodinäide.



Joonis 5. SELECT operatsioonide kestus ajas

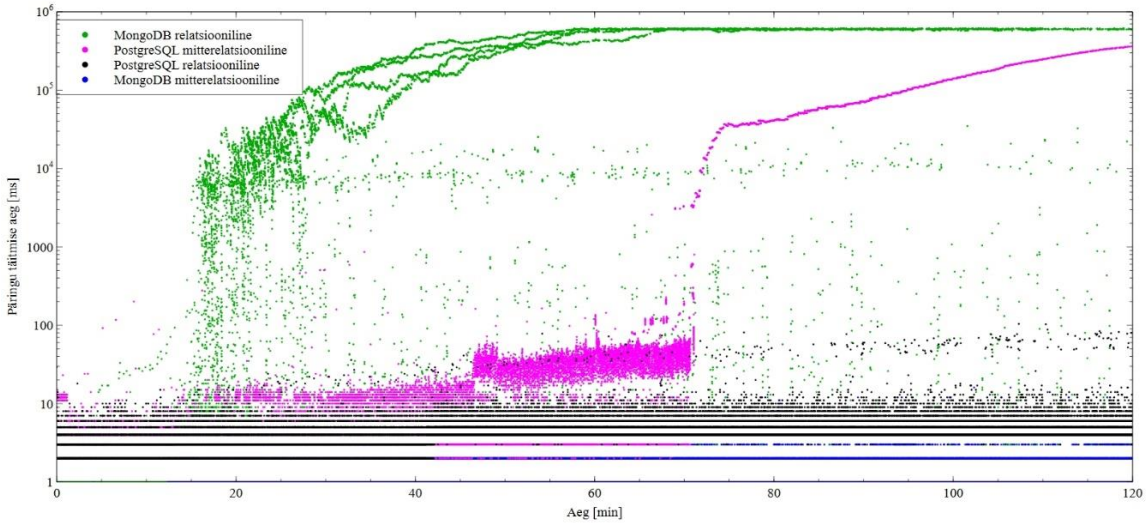
Joonisel 6 on kujutatud UPDATE ehk andmete uudendamisoperatsioonide kestust ajas. Simulatsiooni esimeste minutite juures on näha, et osad mitterelatsioonilise PostgreSQL-i uuendamisoperatsioonid võtsid ligikaudu 50 millisekundit rohkem aega kui ülejäänud implementatsioonide päringud ning kuni 70-nda minutini püsisid nimetatud implementatsiooni päringute kiirused vahemikus 50 - 400 millisekundit. Peale seda toimus hüppeline tõus nagu oli näha ka mitterelatsioonilise PostgreSQL-i SELECT operatsioonide puhul. Lisa 8-s on toodud mitterelatsioonilise PostgreSQL-i toote staatuse uuendamise koodinäide.

Selgelt eristub umbes 15-ndal minutil relatsioonilise MongoDB päringute kiiruse järsk aeglustumine kasvades paarilt millisekundilt 43 sekundile.



Joonis 6. UPDATE operatsioonide kestus ajas

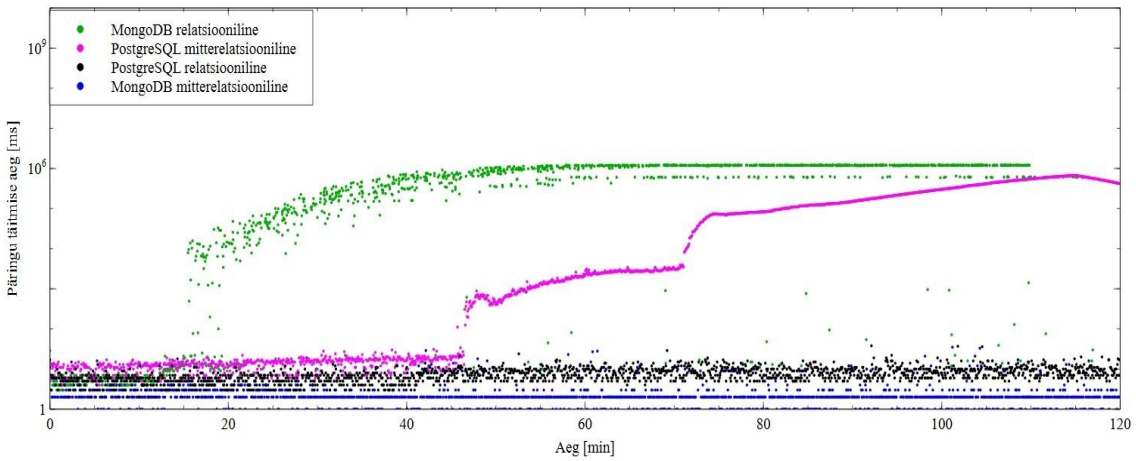
Joonis 7 kujutab implementatsioonide INSERT ehk andmete sisestusoperatsioonide kestust simulatsiooni aja suhtes. Nii mitterelatsioonilise MongoDB kui relatsioonilise PostgreSQL-i sisestusoperatsioonide täitmise aeg oli terve simulatsiooni vältel stabiilne jäädes alla 10 millisekundi. Järk-järguliselt kasvas päringute aeg mitterelatsioonilises PostgreSQL-is, kuid suurem aeglustumine toimus 75-ndal minutil. Kõige märkimisväärsem aeglustumine toimus jällegi relatsioonilises MongoDB-s.



Joonis 7. INSERT operatsioonide kestus ajas

Joonisel 8 on kujutatud iga implementatsiooni DELETE ehk andmete kustutamisoperatsioonide kestust ajas. Operatsioonide kestuse üldine trend sarnaneb UPDATE päringutega. Kuni 15-nda minutini jäid kõikide implementatsioonide kustutamispäringute kestused 1-10 millisekundi vahemikku, kuid seejärel aeglustusid järsult relatsioonilise MongoDB päringud. Mitterelatsioonlise PostgreSQL-i päringud võtsid alguses küll ligikaudu 10 millisekundit kauem aega võrreldes ülejäänutega, kuid jooniselt on näha kahte hüppelist tõusu 45 ja 70-nda minuti juures, misjärel jätkub järk-järguline tõus 115 minutini jõudes lõpuks relatsioonilise MongoDB päringutega samale tasemele.

Terve simulatsiooni vältel olid mitterelatsioonilise MongoDB kustutamispäringud kõige kiiremad, nende täitmise aeg jäi paari millisekundi juurde. Veidi aeglasemad olid relatsioonilise PorstgreSQL-i päringud, mille täitmise kiirus oli üldiselt 5-10 millisekundit. See varieeruvus väljendub jooniselt, kus on näha, et relatsioonilise PorstgreSQL-i päringute andmepunktid ei koondu ühele joonele nagu mitterelatsioonilise MongoDB puhul täheldada võib.



Joonis 8. DELETE operatsioonide kestus ajas

5 Analüüs

Mitte ühegi imlementatsiooni mälukasutus ei jõudnud küll seatud 4 gigabaidi mälupiirini, kuid mitterelatsioonilise PostgreSQL-i ja relatsioonilise MongoDB puhul oli märgata oluliselt suuremat mälukasutust. See võis olla tingitud asjaolust, et mõlemal juhul oli tegemist implementatsiooniga, kus kasutati andmemudelit, mis ei olnud antud andmebaasis primaarne andmete hoiustamise viis.

Mitterelatsioonilise MongoDB mälukasutuse juures võis täheldada hüppelisi languseid. Selle trendi üks võimalik põhjendus on prügikoguja (*garbage collector*), mis kogub kokku objektid, mida enam ei kasutata. Prügikogumise algoritmid võivad olla rakendatavad väikesemahuliste andmehulkade juures, kuid suurandmetega töötavate rakenduste puhul need ei sobi, sest nende läbilaskevõime ei skaleeru [26]. Jooniselt 2 oli märgata, et mitterelatsioonilises MongoDB-s toimus andmemahtude suurenemisel mälu vabastamine järjest sagedamini, mis tagas selle, et simulatsiooni vältel jäi nimetatud andmebaasi maksimaalne mälukasutuse maht võrdlemisi madalaks. Samasugust trendi ei olnud aga näha relatsioonilises implementatsioonis, eristada saab küll hetki, mil toimus mälukasutuse langus, kuid see oli väga väike või praktiliselt olematu. Autori hüpoteesiks on, et mälu vabastamise protsess MongoDB-s on ebaefektiivne kui kasutatakse relatsioonilist andmemudelit. Ka mitterelatsioonilises PostgreSQL-is võib täheldada, et toimus mälu vabastamine, kuid see oli mõnevõrra ebaefektiivsem võrreldes mitterelatsioonilise MongoDB-ga, sest vabastatud mälu suurus oli oluliselt väiksem ja mälukasutuse maht siiski tõusis. Relatsioonilises PostgreSQL-is perioodilist mälu vabastamist näha ei ole, mis võib olla tingitud sellest, et nii väikse mälukasutuse juures polnud nimetatud protsessi rakendamiseks vajadust.

Relatsioonilise mudeliga MongoDB jõudis oma maksimaalse võimekuse piirini peale simulatsiooni esimest 15 minutit. Oluliselt hiljem jõudis ka mitterelatsioonilise PostgreSQL-i protsessori koormusprotsent maksimumini ning see kajastub ka CRUD operatsioonide graafikutel.

Veel üks tähelepanek protsessori koormuse graafikult on see, et nii MongoDB kui PostgreSQL-i mitterelatsiooniste implementatsioonide puhul eristuvad hetked, kus protsessori koormus järsult kasvas, kuid langes taas mõni hetk hiljem. Võimalik, et nendel hetkedel toimus baasis andmete korrastamine, mis vajas ajutiselt rohkem ressurssi. Taolist koormuse kõikumist ei ole aga näha relatsioonilises PostgreSQL-is.

Kuna eksperimendis kasutati andmebaaside loomiseks Dockerit, siis ei ole päringute kiirused kindlasti samad, mis füüsilises andmebaasis, kuid võrdlusmoment säilib sellegipoolest, sest kõik andmebaasid olid võrdses seisus.

Relatsioonilises PostgreSQL-is ja mitterelatsioonilises MongoDB-s olid toote uuendamise ja klassifikaatori järgi toodete küsimise päringud stabiilselt kiired. Tegemist oli suhteliselt lihtsate päringutega, mistõttu võttis nende täitmine vähe aega ka suuremate andmehulkade juures. Keerukamad ja mahukamad olid riigi põhjal toodete leidmise ning toodete ja tarnijate koguarvu küsimise päringud, mistõttu nende täitmise aeg varieerus rohkem ning aeglustus andmemahu suurenedes.

Mitterelatsioonilise PostgreSQL-i ja relatsioonilise MongoDB päringute hüppelised aeglustumised olid tingitud protsessori koormusest. Kui protsessor jõudis maksimaalse võimekuse piirini, aeglustusid ka vastava andmebaasi päringud. Suurte andmehulkade töötlemisega sai kõige kehvemini hakkama relatsiooniline MongoDB. SELECT operatsioonide jooniselt oli näha, kuidas poole simulatsiooni pealt hakkas andmepunktide arv vähemena. See tähendab, et päringud läksid niivõrd aeglaseks, et nende vastus ei jõudnud enne simulatsiooni lõppu tagasi.

Valitud karakteristikute põhjal tehtud võrdlusest võib järeldada, et relatsioonilise mudeli kasutamine MongoDB-s ei ole hea lahendus isegi väiksemate andmemahtude korral. Samas kui relatsioonandmebaasi kasutamine on äriliselt põhjendatud, võib JSONB andmetüübi kasutamine PostgreSQL-is olla üks võimalik valik, kuid seda ainult väiksemate koormuste korral.

Andmebaasi päringute kiiruste poole pealt on relatsiooniline PostgreSQL ja mitterelatsiooniline MongoDB üsna samal tasemel, kuid protsessori koormustest lähtudes võib öelda, et PostgreSQL skaleerub paremini. Seepärast soovitab autor töö alguses

mainitud ettevõttele jätkata sama lahendusega ehk relatsioonilise andmemudeliga relatsioonilises andmebaasis ning uurida hübriidlahenduste võimalusi. Tulemustest lähtudes ei saa ühtegi teist eksperimendis käsitletud implementatsiooni ettevõttele soovitada, sest need ei vasta ärivajadustele

6 Kokkuvõte

Antud töö teema tulenes autori töökohas tekkinud probleemist, kus andmemudeli kiire kasvu tõttu tuli kaaluda alternatiivseid andmete säilitamise viise ning otsustada, kas jätkata relatsioonilise mudeli kasutamisega relatsioonilises andmebaasis või leida mõni muu lahendus.

Eesmärgiks oli tekitada relatsioonilise ja mitterelatsioonilise andmebaasi jõudluste võrdlusmoment PostgreSQL-i ja MongoDB näitel, rakendades mõlemas andmebaasissüsteemis nii relatsioonilist kui mitterelatsioonilist andmemudelit. Ristanalüüsi laiem eesmärk oli paremini mõista andmebaaside võimekust nii normaal- kui alternatiivses olukorras.

Andmebaasi jõudluste võrdlusmomendi tekitamiseks viidi läbi eksperiment, kus PostgreSQL-is ja MongoDB-s implementeeriti nii relatsiooniline kui mitterelatsiooniline andmemudel ning simuleeriti reaalseid töökoormusi nendes süsteemides.

Tekkinud võrdlusmomendist järeldati, et reaalsete töökoormuste juures on kõige paremini skaleeruv lahendus relatsioonilise mudeliga PostgreSQL, kuid palju ei jäänud maha ka mitterelatsioonilise mudeliga MongoDB. Andmemudeli paindlikkuse osas on samuti parem pigem PostgreSQL, mis tuleb madalate töökoormuste juures toime ka JSONB formaadis andmete hoiustamisega.

Täideti töös püstitatud eesmärk. Saadi ülevaade, kuidas toimivad andmebaasid normaal- ja alternatiivolukorras. Andmebaasi jõudluste võrdlusele tuginedes soovitati autori töökohas jätkata relatsioonilise andmemudeli kasutamisega relatsioonilises andmebaasis.

Autori püstitatud hüpotees pidas osaliselt paika. Andmebaasis alternatiivsete mudelite rakendamine vähendas tõesti jõudlust, kuid autor pidi tõdema, et mõju kiirusele oli oodatust märkimisväärselt suurem. Alternatiivsete lahenduste implementeerimine on realistlik vaid madalate töökoormuste juures.

Eksperimendi käigus jõudsid nii relatsiooniline MongoDB kui mitterelatsiooniline PostgreSQL-i oma võimekuse piirini. Töö võimalik edasiarendus oleks lasta simulatsioonil joosta ilma ajapiiranguta ning vaadata, millal tuleks vastu relatsioonilise PostgreSQL-i ja mitterelatsioonilise MongoDB võimekuse piir. Lisaks ei toodud antud töös võrdlusse ühtegi hübriidlahendust, kuid see on üks potentsiaalne implementatsioon, mis võiks olla uurimisobjektiks edasistes töödes. Huvitav oleks ka teada, kuidas mõjutab jõudlust andmete valideerimine. Täpsemalt, kas PostgreSQL-i JSONB formaadis dokumendile skeemi defineerimine mõjutab jõudust ning kas mõju avaldub ka MongoDB-s, kui lisada andmete valideerimine dokumendimudelile või kui eemaldada see relatsiooniliselt mudelilt.

Kasutatud kirjandus

[1] M. J. Hernandez, *Database Design for Mere Mortals®: A Hands-on Guide to Relational Database Design, Third Edition*. Addison-Wesley Professional, 2013, pt. 1, pt. 4.

[2] *Information technology - Home Electronic System - Guidelines for product interoperability - Part 1: Introduction: ISO/IEC 18012-1:2004(en)*.

[3] E. Pirozzi, L. Ferrari, *Learn PostgreSQL*. Packt Publishing, 2020, jt. 1, pt 1, jt. 2, pt 1.

[4] J. L. Harrington, *Relational Database Design and Implementation, 4th Edition*. Morgan Kaufmann, 2016, jt. 1, pt. 1, jt. 2, pt 3-5.

[5] Microsoft, "OutOfMemoryException Class". Kasutatud: 17.05.2021. [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/dotnet/api/system.outofmemoryexception?view=net-5.0.

[6] B. W. Wade, D. D. Chamberlin, "IBM Relational Database Systems: The Early Years", IEEE Annals of the History of Computing, vol. 34, no. 4, okt-dets 2012. Kasutatud: 16.05.2021. [Võrgumaterjal]. Saadaval: https://ieeexplore.ieee.org/document/6297962.

[7] G. Harrison, *Next Generation Databases: NoSQL, NewSQL, and Big Data.* Apress, 2016, jt. 1, pt 1.

[8] P. J. Sadalage, M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 2012, jt, 1, pt, 1-3.

[9] A. Petrov, *Database Internals*. O'Reilly Media, Inc., 2019, jt. 1, pt 1.

[10] PostgreSQL Global Development Group, "PostgreSQL 9.2 released". 10.09.2012. Kasutatud: 01.05.2020. [Võrgumaterjal]. Saadaval: https://www.postgresql.org/about/news/postgresql-92-released-1415/.

[11] PostgreSQL Global Development Group, "PostgreSQL 9.4.26 Documentation". 18.012.2014. Kasutatud: 01.05.2021. [Võrgumaterjal]. Saadaval: https://www.postgresql.org/docs/9.4/release-9-4.html.

[12] G.Tillmann, *Usage-Driven Database Design: From Logical Data Modeling through Physical Schema Definition.* Apress, 2017, pt. 1, pt. 8.

[13] A. Phaltankar; L. Nedov; M. Harrison, J. Ahsan, *MongoDB Fundamentals*. Packt Publishing, 2020, pt. 1-2.

[14] "DB-Engines Ranking". Kasutatud: 17.05.2021 [Võrgumaterjal]. Saadaval: http://db-engines.com/en/ranking.

[15] G. Zhao, W. Huang, S. Liang, Y. Tang, “Modeling MongoDB with Relational Model”. IEEE: Fourth International Conference on Emerging Intelligent Data and Web Technologies, 2013. Kasutatud: 06.05.2021. [Võrgumaterjal]. Saadaval: https://ieeexplore.ieee.org/abstract/document/6631603/authors#authors.

[16] M. Hewasinghage, S. Nadal, A. Abello, “On the Performance Impact of Using JSON, Beyond Impedance Mismatch”. ADBIS 2020: *New Trends in Databases and Information Systems*, pp 73-83. Kasutatud: 25.04.2021. [Võrgumaterjal]. Saadaval: https://link.springer.com/chapter/10.1007/978-3-030-54623-6_7.

[17] C.-O. Truica, F. Radulescu, A. Boicea, I. Bucur, “Performance Evaluation for CRUD Operations in Asynchronously Replicated Document Oriented Database”, 20th International Conference on Control Systems and Science, 2015. Kasutatud 29.04.2021. [Võrgumaterjal]. Saadaval: https://www.researchgate.net/publication/280832003_Performance_Evaluation_for_CRUD_Operations_in_Asynchronously_Replicated_Document_Oriented_Database.

[18] D. Maksimov, “Performance Comparison of MongoDB and PostgreSQL with JSON types”, Magistritöö, Infotehnoloogia teaduskond, Tallinna Tehnikaülikool, 2015. Kasutatud: 25.04.2021. Saadaval: https://digikogu.taltech.ee/et/Item/8494a95f-ecf6-4566-bc25-09a26f142861.

[19] MongoDB Inc., “ObjectId”. Kasutatud: 16.05.2021. [Võrgumaterjal]. https://docs.mongodb.com/manual/reference/method/ObjectId/.

[20] Docker Inc., “Docker overview”. Kasutatud: 15.05.2021. [Võrgumaterjal]. Saadaval: https://docs.docker.com/get-started/overview/.

[21] N. Poulton, “Docker Deep Dive”. Packt Publishing, 2020, pt. 7.

[22] Docker Inc., “Runtime options with Memory, CPUs, and GPUs”. Kasutatud: 09.05.2021. [Võrgumaterjal]. Saadaval: https://docs.docker.com/config/containers/resource_constraints/.

[23] PostgreSQL Global Development Group, “PostgreSQL 12.7 Documentation”. pt 18.4.1, pt 19.4.1. Kasutatud: 13.05.2021. [Võrgumaterjal]. Saadaval: https://www.postgresql.org/docs/12/kernel-resources.html#id-1.6.5.6.5.

[24] MongoDB Inc., “WiredTiger Storage Engine”. Kasutatud: 01.05.2021. [Võrgumaterjal]. Saadaval: https://docs.mongodb.com/manual/core/wiredtiger/.

[25] MongoDB Inc., “Schema Validation”. Kasutatud: 10.05.2021. [Võrgumaterjal]. Saadaval: https://docs.mongodb.com/manual/core/schema-validation/.

[26] R. Bruno, P. Ferreira, “A Study on Garbage Collection Algorithms for Big Data Environments”. ACM Computing Surveys, no. 20, jan 2018. Kasutatud: 15.05.2021. [Võrgumaterjal]. Saadaval: https://dl.acm.org/doi/abs/10.1145/3156818.

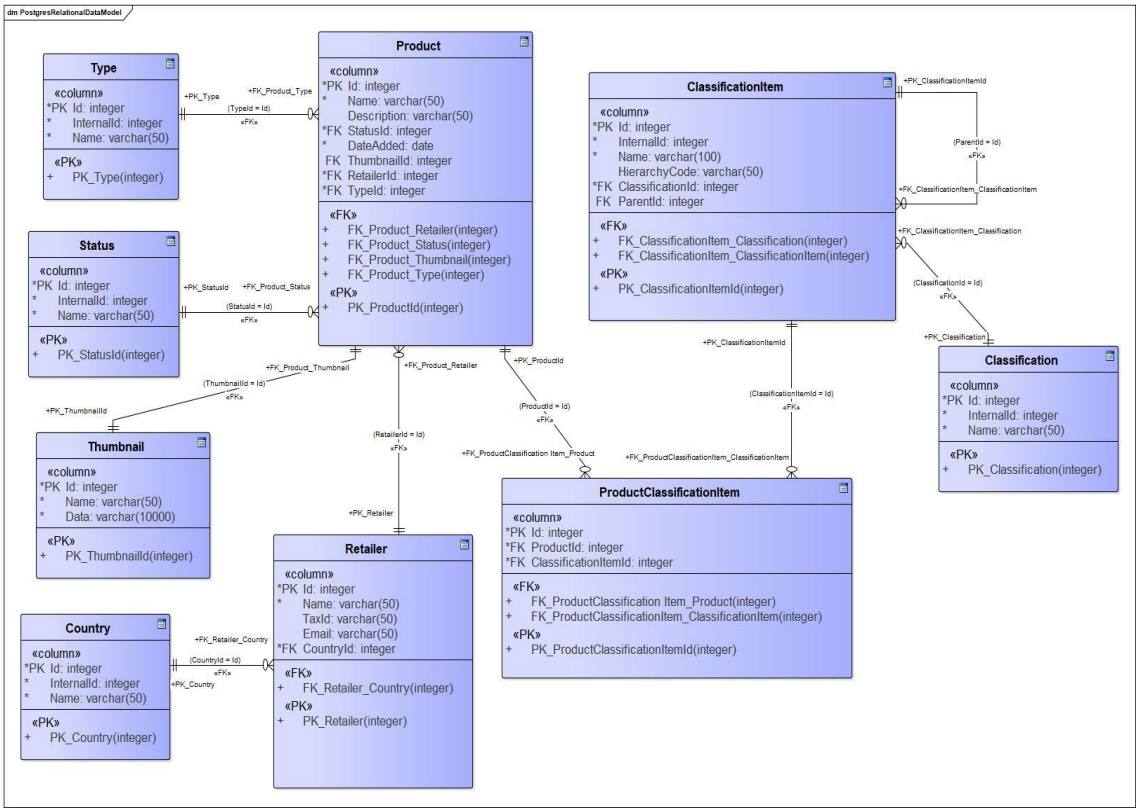
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Grete Saar

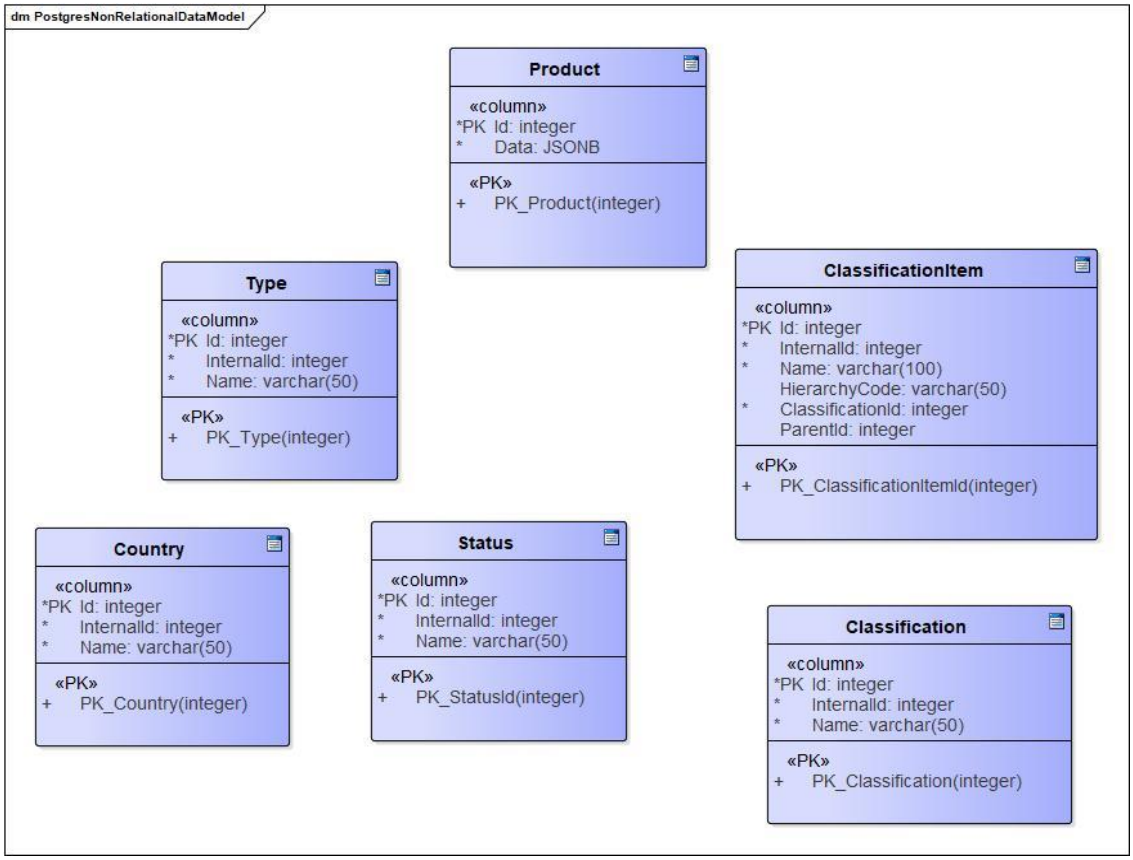
1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Relatsioonilise ja mitterelatsioonilise andmebaasi jõudluse ristanalüüs relatsioonilise ja mitterelatsioonilise andmemudeliga PostgreSQL ja MongoDB näitel" , mille juhendajad on Jakob Jõgi ja Mart Roost
    1.1. reprodutseerimiseks 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

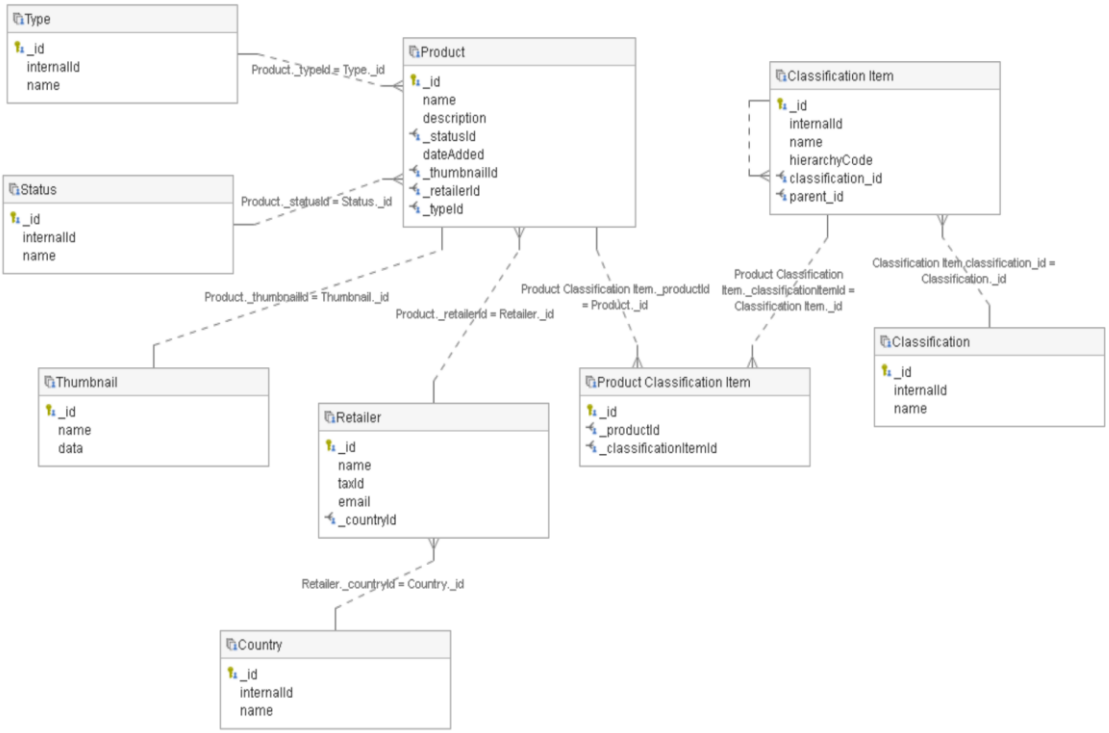
Lisa 2 – PostgreSQL-i relatsiooniline andmemudel



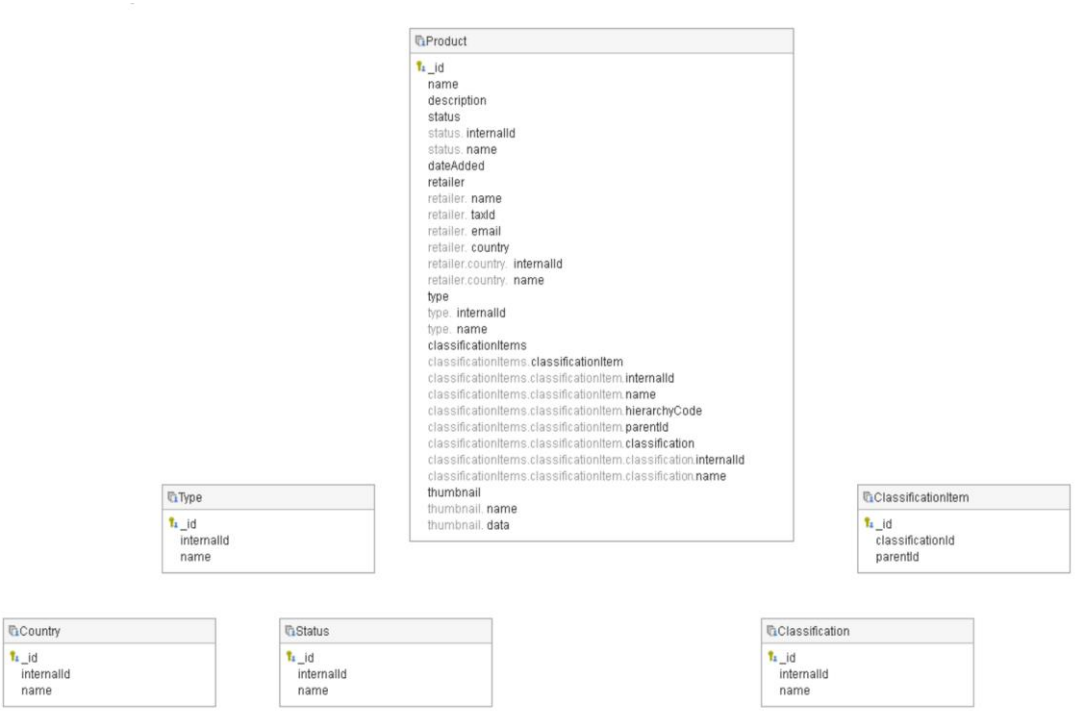
Lisa 3 – PostgreSQL-i mitterelatsiooniline andmemudel



Lisa 4 – MongoDB relatsiooniline andmemudel



Lisa 5 – MongoDB mitterelatsiooniline andmemudel



Lisa 6 – Asünkroonsete CRUD operatsioonide käivitamine

```
async function executeQueries (db, sStartTime){
  const sQueryResultsFileName = '../QueryResults.csv';
  const sContainerStatsFileName = '../ContainerStats.csv';

    runQuery(db, addProduct, 200, sQueryResultsFileName, sStartTime);
    setTimeout(runQuery, 2000, db, getCountryProducts, 4000, sQueryResultsFileName, sStartTime);
    setTimeout(runQuery, 2400, db, getProductsWithHierarchyCode, 4000, sQueryResultsFileName, sStartTime);
    setTimeout(runQuery, 2800, db, getUnclassifiedProducts, 4000, sQueryResultsFileName, sStartTime);
    setTimeout(runQuery, 3200, db, getProductsWithThumbnails, 4000, sQueryResultsFileName, sStartTime);
    setTimeout(runQuery, 3600, db, updateProductsStatuses, 4000, sQueryResultsFileName, sStartTime);
    setTimeout(runQuery, 4000, db, updateProductName, 4000, sQueryResultsFileName, sStartTime);
    setTimeout(runQuery, 400, db, deleteRandomProduct, 4000, sQueryResultsFileName, sStartTime);

    setTimeout(runQuery, 400, db, getProductCount, 2000, sQueryResultsFileName, sStartTime);
    setTimeout(runQuery, 400, db, getRetailerCount, 2000, sQueryResultsFileName, sStartTime);

    setTimeout(runQuery, 400, '*', si.dockerContainerStats, 500, sContainerStatsFileName, sStartTime);
  }

async function runQuery (db, fRunFunction, iDelay, sFileName, sStartTime){
  while(Date.now() < sStartTime + 7200000){
    fRunFunction(db).then(oDataToWrite => {
      writeToFile(sFileName, oDataToWrite);
    });
    await delay(iDelay);
  }
}
```

Lisa 7 – Relatsioonilise MongoDB riigi põhjal toodete leidmise koodinäide

```
205     const aResult = await oModels.Country.aggregate([
206         {
207             $match: { "_id": oCountryId }
208         }, {
209             $lookup: {
210                 from: "Retailer",
211                 localField: "_id",
212                 foreignField: "_countryId",
213                 as: "retailers"
214             }
215         }, {
216             $unwind: "$retailers"
217         }, {
218             $lookup: {
219                 from: "Product",
220                 localField: "retailers._id",
221                 foreignField: "_retailerId",
222                 as: "products"
223             }
224         }, {
225             $project: {
226                 _id: 1,
227                 name: 1,
228                 products: "$products"
229             }
230         }
231     ]).catch(console.log);
```

Lisa 8 – Mitterelatsioonilise PostgreSQL-i toote staatuse uuendamise koodinäide

```
165     const oUpdateResult = await db.query(`UPDATE "Product"
166                                           SET "Data" = jsonb_set("Data"::jsonb, '{status}',
167                                           '{"InternalId": ${oUpdateStatus.InternalId}, "Name": "${oUpdateStatus.Name}"}')
168                                           WHERE ("Data" -> 'status' ->> 'InternalId')::int = ${iConditionStatusId}
169                                           AND ("Data" -> 'type' ->> 'InternalId')::int = ${aTypeResult.rows[0].InternalId}`)
170                                           .catch(console.log);
171
```