TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Karl-Martin Miidu 162728IABM

SQL- JA GRAAFANDMEBAASI VÕRDLUS FINANTSPETTUSTE TÕKESTAMISE NÄITEL

Magistritöö

Juhendaja: Innar Liiv
PhD

Tallinn 2019

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: Karl-Martin Miidu

30.12.2019

Annotatsioon

Käesoleva töö teemaks on SQL- ning graafandmebaasisüsteemide võrdlus finantspettuste tõkestamise süsteemi praktilise kaasuse näitel. Käesoleva töö eesmärgiks on selgitada välja, milline andmebaasisüsteem on efektiivsem kavandatud finantspettuste tuvastamise süsteemis.

Teema uurimiseks defineeris autor loodavas süsteemis 12 andmeagregatsiooni, mis võimaldavad potentsiaalselt pahatahtlike osapoolte tuvastamist. Järgnevalt loodi identsel andmestikul mõlemas andmebaasisüsteemis vajalik struktuur, mis võimaldaks eeldefineeritud päringuid sooritada. Seejärel analüüsiti mõlema süsteemi päringute tulemusi, nende kiirust ning käivitusplaane, mis selgitavad mõlema andmebaasisüsteemi puhul päringu jooksutamise detaile. Kumbki andmebaasisüsteem ei olnud selgelt sobivaim - mõlemal andmebaasil oli eeliseid olenevalt päringust. Tulemuste analüüsist selgus, et sobivamaks andmebaasisüsteemiks agregatsioonides, mis eeldavad andmestikus läbi mitme objekti seose liikumist, osutus kiiremaks graafandmebaas, mis edestas SQL-andmebaasisüsteemi mõningate agregatsioonide puhul ligikaudu 70-kordse kiiruse vahega. Agregatsioonide käivitsuplaanidest selgus, et kuna graafandmebaas salvestab iga graafis asuva sõlme ning serva puhul temaga seotud vastavad servad ning sõlmed, siis suudab graafandmebaas agregatsioonide käivitamisel ära kasutada salvestatud seoseid ilma, et oleks tarvis kasutada indeksit või muud optimeerimismeetodit nagu seda peab kasutama SQL-andmebaasisüsteem. Samas oli mõningate agregatsioonide puhul tulemus ka vastupidine, kus SQL-andmebaas oli märgatavalt kiirem.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 70 leheküljel, 6 peatükki, 22 joonist, 2 tabelit.

Abstract

A Comparison between a SQL Database and a Graph Database in the context of a Financial Fraud Detection

The research topic for this thesis is creating and comparing fraud detection systems in graph and SQL database systems. The aim of the research is to determine which database system is more suitable given scenario of fraud detection.

To solve the research question, author defined 12 data aggregations that would be used in detecting suspicious actors in the financial system. Required structure was created in both databases on identical dataset to validate which database would perform better. Then needed aggregations were executed on both databases. All executions were supplemented by the query plans databases used for given queries. From 12 defined aggregations both databases outperformed each other an equal amount. However analysis of the results revealed, that graph database outperforms SQL up to 70 times in execution speed in aggregations where the database needs to traverse relationships on nodes in depth. The results revealed that graph database handles nodes and relationships between nodes as equally important storing pointers of relations and nodes on both objects. Using given structure, graph database has advantages over relational database when queries need to traverse through the graph as no additional optimisation tools are needed for efficient graph traversal. However the results of the experiment also revealed that there were multiple aggregations where SQL database was significantly faster compared to graph database.

The thesis is written in Estonian and contains 70 pages of text, 6 chapters, 23 figures, 2 tables.

Lühendite ja mõistete sõnastik

ACID	Atomic, Consistent, Isolated, Durable
AWS	Amazon Web Services
CNP	Card-Not-Present
CSV	Comma Separated Value
LUBM	Lehigh University Benchmark
TCP/IP	Transmission Control Protocol/Internet Protocol
SQL	Structured Query Language
SSN	*Social Security Number*, USA valitsuse poolt väljastatav sotsiaalkindlustusnumber

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

1.1 Taust ja probleem

Tänapäeva finantsmaailm on tehnoloogia hoogsa arengu tõttu kiiresti muutumas. Ajalooliselt oli finantstehingute tegemiseks inimesel vajalik füüsiliselt suhelda panga telleriga või müüjaga, kelle abil vajalikke tehinguid sooritada. Nüüdseks on pangad ning poed kolinud igaühe nutiseadmesse – pangakontost ülevaate saamine ning raha ülekandmine on kaasaegses eraisiku panganduses standardiks [1]. Reaalajas toimiv ning kasutajasõbralik finantsteenus on lisaks tavakliendile ahvatlevaks sihtmärgiks pahatahtlike kavatsustega osapooltele, kes üritavad moodsate süsteemide nõrkuseid ära kasutada nii rahapesuks kui ka finantspettusteks [2]. Üha karmimad regulatsioonid [3] ja suurenev kuritegude hulk [2] kohustab finantsettevõtteid konkurentsieelise saavutamiseks nutikamaid ning efektiivsemaid vastumeetmeid rakendama, et tõkestada kriminaalset tegevust. Keerukad tehisintellekti süsteemid on muutunud tavapäraseks tööriistaks illegaalsete tegevuste tuvastamiseks ning ennetamiseks. Lisaks on tihtipeale eduka kaitsemehhanismide eelduseks oma kliendi tundmine ning temaga võimalike seotud osapoolte võrgustiku efektiivne analüüs. Levinud võrgustike esindamise vahend on graaf. Üldiselt mõistetakse graafina süsteemi, mis koosneb sõlmedest ning sõlmi siduvatest servadest. Näiteks võib sotsiaalvõrgustikus esindada sõlmed isikuid ning sõlmi siduvad servad isikute vahelisi suhteid.

Graafe kirjeldavaid andmestruktuure on tehniliselt võimalik koostada erinevates andmebaasisüsteemides. Enamlevinud andmebaasisüsteemid põhinevad relatsioonilisel andmemudelil. Samas on viimase 15 aasta jooksul hoogsalt populaarsust kogunud just graafandmebaasid – need on andmebaasisüsteemid, mis põhinevad graafi andmemudelil ning on disainitud andmestike jaoks, mida on just graafidena mõistlik esitada. Samas on relatsioonilised andmebaasid populaarsuselt siiski kaugelt enam levinud kui graafandmebaasid – andmebaaside populaarsuse edetabelis 10 kõige enam kasutatava andmebaasis seast 6 on relatsioonilised andmebaasid. Populaarseim graafandmebaas Neo4j on aga 22. kohal [4]. Seetõttu tekib küsimus, kuivõrd mõistlik on uue tehnoloogia

kasutuselevõtt – millised on täpsemalt relatsioonilise andmebaasi nõrkused graafe kirjeldavate andmestruktuuride esitamisel ning millisel puhul tasuks kaaluda graafi andmemudelil põhinevat andmebaasisüsteemi kasutamist.

1.2 Ülesande püstitus

Magistritöö eesmärgiks on võrrelda SQL- ja graafandmebaasi suutlikkust finantspettuste tuvastamiseks vajalike andmepäringute sooritamisel ning lahti mõtestada, millised on reaalse süsteemi näitel mõlema andmebaasi fundamentaalsed erinevused, tugevused ja nõrkused konkreetses probleemivaldkonnas. Magistritöö eesmärgi täitmiseks on vajalik implementeerida lihtsustatud disainiga finantsettevõttes rakendatav illegaalse tegevuse tuvastamise süsteemi andmebaas, kasutades selleks nii graafandmebaasi kui ka SQL-andmebaasi. Töös rakendatakse PostgreSQL-i kui relatsioonilise andmebaasi lahendusena ning Neo4j graafandmebaasi lahendusena.

1.3 Metoodika

Probleemi lahendamiseks luuakse järgnevad andmebaasid: relatsiooniline andmebaas PostgreSQL abil ning graafandmebaas kasutades Neo4j platvormi. Mõlemas andmebaasis kasutatavad andmed on identsed. Töös kasutatavad andmed esindavad lihtsustatud kujul finantsettevõttes kasutatavat pettuste tuvastamise süsteemi, mis hõlmab endas andmeid klientide, maksete, maksete saajate, klientide IP-aadresside ning nende omavaheliste seoste kohta. Töös defineeritakse fikseeritud 12 andmepunkti, mida potentsiaalselt kasutatakse illegaalsete maksete tuvastamise süsteemis. Seejärel valitakse konkreetsed kliendid, kelle puhul sooritatakse vastavad andmebaasipäringud eelnevalt defineeritud andmepunktide väärtuste väljastamiseks ning võrreldakse mõlema andmebaasi jõudlust vastavate päringute jooksutamisel. Lisaks tuuakse välja mõlemas andmebaasisüsteemis käivitusplaanid ning põhjendatakse päringute kiiruste erinevusi.

1.4 Ülevaade tööst

Magistritöö esimeses osas antakse ülevaade finantspettuste olemusest ning trendidest. Lisaks tuuakse välja varasemad uurimused finantspettuste tõkestamisest – millised süsteemid on varasemalt tõestanud efektiivse finantspettuste tuvastusmeetmena ning millised on varasemates uurimustes välja toodud andmepunktid, mis on osutunud

tõhusateks kahtlaste isikute ning maksete indikaatoriteks. Nende andmepunktid on sisendiks hilisemas peatükis eksperimendi koostamisel (vt peatükk 4.3).

Töö kolmandas peatükis selgitatakse SQL ning graafi andmemudelil põhinevate andmebaaside teoreetilist erinevust ning tööpõhimõtteid. Antud teoreetilistele tööpõhimõtetele tuginedes on võimalik töö teises pooles täpsemalt selgitada, millised on mõlema andmebaasisüsteemi tugevused ning nõrkused. Lisaks tuuakse välja varasemad uurimused graafipõhisel andmemudelil realiseeritud andmebaaside ning SQL andmebaasisüsteemide varasemates võrdlustest.

Töö teises pooles viiakse läbi eksperiment, milles realiseeritakse mõlemas andmebaasisüsteemis lihtsustatud andmemudel, mis võimaldaks illegaalsete finantstehingute tuvastamiseks vajalikke andmepunkte tagastada. Eksperimendi käigus võrreldakse konkreetsete andmeagregatsioonide kiiruseid mõlemas andmebaasisüsteemis. Eksperimendi tulemustes selgitatakse saadud tulemuste põhjuseid tuginedes mõlema andmebaasi teoreetilisele tööpõhimõttele ning kasutades eksperimendi käigus tuvastatud päringute käivitusplaane.

Magistritöö lõpetuseks tuuakse kokkuvõtlikult välja kõik töö olulisemad tulemused.

2 Finantspettused ja varasemad uurimused nende tõkestamise süsteemidest

2.1 Finantspettuste ülevaade

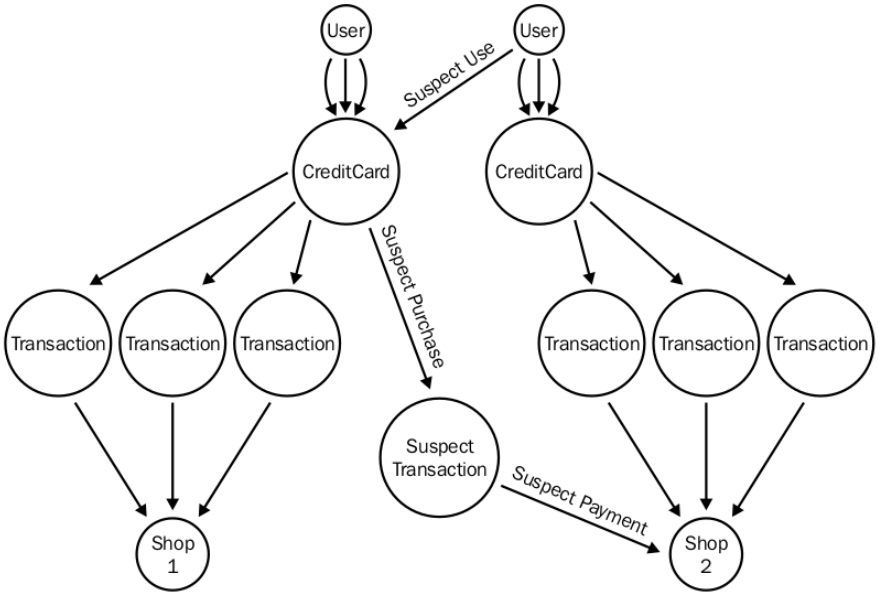
Finantspettuse all peetakse antud töös silmas rahalisi tehinguid, mis on iseloomult pahatahtlikud või on sooritatud ebaeetiliste teel omandatud vahendite abil. Selleks võib olla näiteks varastatud krediitkaardiga e-poes ostude sooritamine või varastatud pangakontolt raha ülekandmine. Artikli [2] andmetel, oli Ühendkuningriigis 2016. aastal finantspettustest tulenev kahju 768.8 miljonit naela, mis on 2% rohkem võrreldes varasema aastaga. Samas suudeti finantspettuste tõkestamise süsteemide abil ennetada kahju, mille suurus ulatunuks 1.38 miljardi naelani.

Seoses tehnoloogia arenguga on erinevad finantspettuste tüübid märgatavalt arenenud ning laienenud. Illegaalseid tehinguid on tihtipeale keeruline tuvastada legaalsetest maksetest, samas illegaalsed tehingud võivad ettevõttele tekitada otsest rahalist kahju, mistõttu on mõistlik tõkestada petturlikke tehinguid. Ettevõtete ja finantsinstitutsioonide jaoks on kriitilise tähtsusega arendada erinevaid tehnoloogilisi lahendusi pettuste ennetamiseks ja toimunud pettuste efektiivseks käsitlemiseks [5].

Traditsioonilised reeglipõhised pettuste tuvastamise süsteemid on olulised vahendid minimeerimaks pettustest saadavat kahju. Samas üha enam pettureid omandavad võtteid, kuidas pääseda mööda traditsioonilistest pettuste tuvastamise reeglitest, näiteks moodustades erinevate meetoditega mitmeid vale identiteediga sünteetilisi profiile. Ükski pettuste tuvastamise süsteem ei ole ideaalne – samas on võimalik märgatavalt parendada finantspettuste tõkestamise süsteeme vaadeldes mitte ainult üksikuid andmepunkte profiilil, vaid analüüsida profiili seoseid läbi nende andmepunkide. Tihtipeale on just seosed erinevate profiilide andmepunktide vahel oluliseks indikaatoriks pettuste avastamisel. See ei tähenda alati vajadust hakata koguma uusi andmepunkte – tihtipeale on võimalik juba olemasolevast andmestikust leida olulisi seoseid vaadeldes andmestikku kui graafi [6].

Näiteks võib vaadelda stsenaariumi tavalises e-poe süsteemis. E-poe kliendid on loonud endale kasutajad, millega seotakse kasutaja IP-aadress, krediitkaardi andmed ning elukoha aadress. Üldiselt eeldatakse, et iga andmepunkt on ainult seotud vaid mõne kasutajaga. Näiteks võivad ühe pere liikmete kasutajakontod olla seotud IP-aadressi ja elukoha aadressi järgi. Samas seoste kasvamisel on tihedalt seotud kasutajate võrgustikes tihti tegu petturlike motiividega isikutega [6].

Joonis 1 illustreerib stsenaariumi võimalikust kahtlase krediitkaardi tehingust.



Joonis 1. Kahtlase krediitkaardi tehingu näide [6].

Joonisel 1 on näha, et konkreetne kasutaja sooritab oste ühes konkreetses poes alati sama krediitkaardiga. Sarnaselt teine kasutaja aga sooritab oste oma krediitkaardiga teises poes. Äkitselt ostab teine kasutaja esimese kasutaja krediitkaardiga ostu poes, kus ainult teine kasutaja on eelnevalt oste sooritanud. Antud muster võib osutuda kahtlaseks ning võib järgneda võimalik järelkontroll tagamaks, et tuvastatud anomaalia ei ole petturlik [6].

2.2 Varasemad uurimused finantspettuste tõkestamise süsteemidest

Finantspettuste tõkestamise süsteemidest on varasemalt avaldatud mitmeid artikleid. Järgnevalt käsitleb autor artikleid, milles on käsitletud andmetes võrgustike mustrite tuvastamist, andmestikus seoste leidmist ning masinõppe kasutamist. Masinõppega seotud artikleid vaadeldakse eeskätt seetõttu, et välja selgitada, millised andmepunktid on osutunud efektiivseteks indikaatoriteks pettuste tuvastamisel – finantspettuste

tõkestamise süsteemi kavandajale on just need sisendiks, milliseid agregatsioone peab andmebaas võimaldama.

Van Vlasselaer [7] uurimisrühm pakkus välja uudse krediitkaardi maksete pettuse tuvastamise süsteemi APATE. APATE on disainitud suutma anda otsuse reaalajas, kas konkreetne krediitkaardi makse on petturlik või mitte. APATE kasutab masinõppe mudelit otsustamaks, kas makse peatada või mitte. Uurimisrühm koostas eksperimendi, mille käigus kasutati konkreetse Belgia panga andmestikku ühe kuu jooksul tehtud krediitkaardi maksetest. Andmestik sisaldas infot maksete tegijast (näiteks etnilised andmed, vanus), maksetest (summa, aeg) ning kaupmeestest (asukoht, tüüp), kellega maksja makse sooritas. Eksperiment keskendus e-poe tehingutele, mille puhul ei ole võimalik tuvastada, kas tegelikkuses kaardi omanik on ostu sooritaja (CNP). Van Vlasselaer otsustas masinõppe mudelites kasutatavad andmepunktid jaga kahte klassi. Esimene klass sisaldas tehingu andmetest otseselt tuletatavad andmepunkte, mis iseloomustavad makse sooritaja ostukäitumist – näiteks viimase makse soorituse aeg, viimase makse soorituse aeg sama kaupmehe juures, kogu maksete arv ning summa, kogu maksete arv konkreetse kaupmehe juures. Teine andmepunktide klass tuletati kaardiomanike, maksete ning kaupmeeste andmestikust koostatud võrgustiku analüüsist. Van Vlasselaer koostas sisendandmestikust graafi, mille sõlmedeks olid makse sooritajad, maksed ning kaupmehed. Koostatud graafil märgiti ära petturlikud maksed ning kasutades võrgustikus propageerimise meetodit (ingl *Influence propagation*) suudeti määrata igale graafi sõlmele skoor, mis näitab kui kahtlane konkreetne sõlm on. Masinõppe mudelis kasutati andmepunktidena graafis kalkuleeritud makse tegija, makse ning kaupmehe kahtlustuse skoore. Mõlema andmepunktide tüübi puhul kasutati kolme eri ajaakent andmepunktide arvutamiseks (viimane tund, viimane päev, viimane nädal). Eksperimendi tulemused näitasid, et kasutades APATE meetodit antud andmestikus suudeti tuvastada 90% maksetest korrektselt petturlikeks samal ajal klassifitseerides ainult 1% maksetest ekslikult petturlikeks. APATE üheks tugevuseks toodi välja just võrgustikest tulenevate andmepunktide tugevus, mis toetasid esimese klassi andmepunkte ning võimendasid kokkuvõttes mudeli täpsemaks.

Artiklis [6] kirjeldatakse kahe finantspettuse stsenaariumeid. Esmalt vaadeldakse "esimese" isiku finantspettuseid – need on pettused, kus väärteo sooritaja võtab laene või muid krediidikohustusi eesmärgiga omandatud võlakohustust mitte tagastada. Artikli

autorid leiavad, et vaadeldes kasutajaid ning nendega seotud andmeid nagu aadress, telefoninumber, sotsiaalkindlustusnumber (SSN) ning otsides vastavaid mustreid seostes, on finantsasutustel võimalik pettused avastada enne kui pettused on toime pandud. Tihtipeale kasutavad esimese isiku finantspettuste kurjategijad jagatud dokumente, telefoninumbreid jne. Teise näite puhul toovad autorid välja stsenaariumi e-poe süsteemis, kus kuriteo toimepanijad võivad luua mitmeid kontosid ning sooritada oste kasutades varastatud krediitkaarte. Antud keskkonnas leitakse, et vaadeldes kasutajate IP-aadresse, jagatud krediitkaarte ning kasutaja e-poe sessiooni identifikaatoreid on võimalik tuvastada loodud pahatahtlikud kasutajad. Eeskätt just vaadeldes esmapilgul täiesti erinevaid kasutajaid on neid omavahel siduvad ühised andmepunktid tugevaks indikaatoriks kahtlaste osapoolte tuvastamiseks. Samas nenditakse, et samalt IP-aadressilt võib ka legitiimsetel põhjustel olla tehtud suurel hulgal makseid erinevatelt kasutajatelt - parema arusaamise kliendi käitumisest tagab kliendi andmetest moodustatud võrgustiku mustri analüüs.

Kasutajate IP-aadresside jälgimist ning nendega seotud kasutajate ning maksete seoste analüüsi peetakse e-kaubanduses oluliseks vahendiks finantspettuste ennetamiseks ka pankade ning krediitkaardimaksete vahendajate seas [8-10]. Näiteks toob Ühendriikide üks suurim pank Barclays välja, et üheks oluliseks finantspettuste ennetusmeetmeks on kasutajate IP-aadresside jälgimine – petturid kasutavad hulgi varastatud krediitkaarte tihtipeale samalt IP-aadressilt, mistõttu võib vajadusel seada piirangud või blokeerida konkreetsed IP-aadressid, mille puhul on näha märgatavalt suurenenud petturlike maksete arvu [8]. IP-aadresside põhist kasutajate analüüsi ning võimalikke blokeeringuid pakub ka näiteks rahvusvaheliste maksete töötleja Paypal [9] ning krediitkaardimaksete vahendaja WorldPay [10].

Artiklis [11] keskendusid autorid krediitkaardi pettuste tuvastamise süsteemile masinõppe perspektiivist. Artiklis kirjeldatud uurimuse eesmärk oli välja selgitada millistel masinõppe algoritmidega treenitud ennustusmudelid on konkreetse testandmestiku peal täpsemad tuvastamaks petturlikke tehinguid. Lisaks algoritmidele toodi välja ka millised masinõppe mudelis kasutatavad andmepunktid olid efektiivsed pettuste indikaatorid ning aitasid parendada mudelite täpsust. Tulemustes selgus, et mudelite täpsus oli suurim, kui treenimisel kasutati lisaks tehinguandmetele ka agregeeritud andmepunkte kasutaja

maksekäitumise kohta, näiteks agregeerides viimaste tundide ja päevade makseid nende tüübi (internetimakse, terminalimakse, raha väljavõtmine, jne) järgi.

Artiklis [12] uuriti täpsemalt, millised agregatsioonid on masinõppe mudelis krediitkaardi maksete pettuste tuvastamiseks kõige efektiivsemad. Uurimise tulemustes selgus, et varasemate maksete agregatsioonid 1, 3 ja 7 päeva lõikes olid efektiivsed olenevalt konkreetsest mudeli algoritmist ning testandmestikust. Lisaks toodi artiklis välja, et olenemata agregatsioonis kasutatava ajavahemiku pikkusest on tehniliselt ideaalne, kui süsteem suudaks sooritada vajalikud agregatsioonid reaalajas, mitte fikseeritud ajahetkel korra päevas.

Molloy [13] toob uurimuses välja meetodid, kuidas kasutades graafipõhiseid agregatsioone on võimalik parendada finantspettuste tõkestamise süsteemi täpsust. Molly uuris, kuidas on võimalik kasutada erinevaid graafi algoritme nagu lühim tee, tugevalt seotud komponendid (SCC - ingl *Strongly Connected Components*) ning PageRank. PageRank on Google poolt arendatud algoritm, mille põhiline eesmärk oli kalkuleerida, milline veebileht on tähtsam/olulisem vaadeldes veebilehtede omavahelisi viitasid [14]. Rakendades PageRank algoritmi krediitkaardimaksete võrgustikus, oli võimalik kalkuleerida maksja konto valiidsust temale tehtud maksete hulka vaadeldes. Hüpoteesiks toodi välja, et kõrge PageRank tulemusega kontod on vähem petturlikud. Uurimuse käigus vaadeldi ühe Euroopa panga klientide poole aasta maksete andmeid ning kasutades eelnimetatud algoritme koostati andmestik, mida kasutati pettuse tuvastamise süsteemis kui lisa andmepunktidena. Molloy peamine eesmärk oli uurida milline on graafipõhiste algoritmide andmepunkte kasutamise efekt vale positiivsete tulemuste vähendamiseks finantspettusi hindavates masinõppe mudelites. Teisisõnu, vaadeldes makseid, mida olemasolev pettuste tuvastamise süsteem on märkinud kahtlaseks, kuidas on võimalik vähendada valesti kahtlasteks märgitud maksete arvu kasutades graafanalüüsi tulemusi. Molloy leidis, et kasutades lühima tee algoritmi, on võimalik vale positiivsete otsuste hulka vähendada 63% võrra. PageRanki puhul oli sama andmestiku puhul vale positiivseid otsuseid vähendada 16% võrra. Lisaks uuris Molly, kui efektiivne on kasutada graafipõhiste algoritmide andmepunkte eraldiseisvas mudelis, mis seekord ei püüaks vähendada ainult olemasoleva süsteemi vale positiivseid tulemusi vaid suudaks eraldiseisvalt tuvastada ka petturlikke makseid. Eksperimendi tulemus näitas, et graafi

algoritmide mudelid oli efektiivsed ka iseseisvalt klassifitseerimisel – täpseimaks mudeliks osutus PageRank algoritmi andmepunkte kasutav süsteem.

3 Teoreetilised aspektid

3.1 SQL andmemudelil põhinev andmebaasisüsteem

Antud peatükis kirjeldab autor lähemalt SQL andmemudelil põhinevat andmebaasisüsteemi.

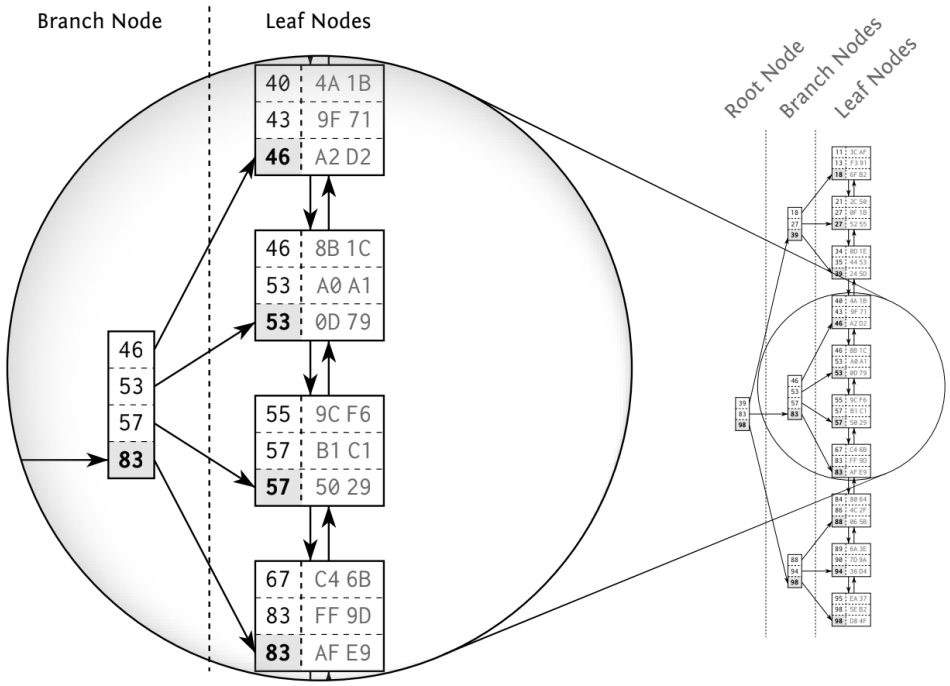
Andmemudeli all peetakse eeskätt silmas relatsioonilise andmemudeli väljamõtleja Edgar F. (Ted) Codd (1981) definitsiooni. Codd käsitleb andmemudelit järgneva kolme komponendi kogumina [15]:

- andmestruktuuride tüüpide kogum, mis on antud mudeli ehitusplokkideks. Relatsioonilises mudelis on andmestruktuurideks relatsioonilised baasmuutujad (ingl *relational variable*). SQL andmemudelis, mis põhineb relatsioonilisel andmemudelil, on andmestruktuuriks baastabelid. Eristatakse relatsioonilist mudelit ja SQL andmemudelit, sest kuigi SQL põhineb relatsioonilisel mudelil ei järgi see kõiki relatsioonilise mudeli põhimõtteid.
- operaatorite ning tuletusreeglite kogum, mida on võimalik rakendada eelnevalt defineeritud andmestruktuuridele. Näiteks relatsioonilises andmemudelis on üheks selliseks operatsiooniks tabelite ühendamine (*join*).
- üldised terviklikkuse reegleid, mis kirjeldavad hulga terviklikke andmebaasi seisundeid, seisundi muutuseid või mõlemaid. Näiteks relatsiooniline mudel nõuab, et igas relatsioonilises muutujas peab olema vähemalt üks kandidaatvõti, SQL aluseks olev andmemudel aga lubab võtmeta tabeleid.

Ühe andmemudeli alusel on võimalik luua üks või mitu andmebaasikeelt. Iga andmebaasisüsteem toetab ühte või mitut andmebaasikeelt ning sellest tulenevalt ka nende andmebaasikeelte aluseks olevaid andmemudeleid [15]. Lisaks relatsioonilisele andmemudelile on tänaseks välja mõeldud palju teisi andmemudeleid näiteks hierarhiline ja graafidel põhinev andmemudel. Igal andmemudelil on oma tugevused ning nõrkused, mis tulenevad eelnevalt defineeritud andmemudeli omadustest. Andmebaasisüsteeme, kus saab kasutada SQL keelt nimetan SQL-andmebaasisüsteemideks.

3.2 Indeks

Indeks on andmebaasi andmestruktuur, mis võimaldab viidatud tabelite kirjete kiiret otsingut [16]. Indeks on tabelist eraldiseisev ning valikuline lisand – indeks vajab lisa salvestusruumi ning hoiab koopiat põhitabeli andmetest. Indeksi loomine ei muuda baastabeli andmeid – indeksi loomisel vaid luuakse uus andmestruktuur, mis viitab tema baastabelile. Indeksi loomiseks kasutatakse andmebaasikäsku *CREATE INDEX* kus täpsustatakse baastabel millele indeks luuakse. Indeksi põhiidee on hoida sorteeritud järjestust loodud baastabeli väärtustest andmestruktuuris, mis võimaldab kiiret baastabeli ridade otsimist defineeritud tabeli väärtuste otsimisel. Selle saavutamiseks kasutatakse indeksis kombineeritult kahte andmestruktuuri – kahepoolne lingitud list (ingl *doubly linked list*) ning tasakaalustatud puu (ingl *balanced tree*). Kahepoolne lingitud list sisaldab fikseeritud arv kirjeid, milles on indekseeritud andmetulba või tulpade väärtus ning viide antud tabeli kirjele. List on sisemiselt sorteeritud indekseeritud andmetulba väärtuste järgi. Listid on omavahel kahepoolselt seotud samuti indekseeritud andmetulba väärtuste järjestuse järgi. Kahepoolsed lingitud listid on grupeeritud tasakaalustatud puu struktuuri, mis võimaldab `Olog(n)` keerukusega kirje otsimist [16]. Joonis 2 illustreerib indeksis kasutatavat kahepoolset lingitud listi ning tasakaalustatud puu struktuuri.



Joonis 2. Indeksi andmestruktuur [16].

Indekseid kasutatakse nii SQL-, graaf-, kui ka muudes andmebaasisüsteemides.

3.3 Graafipõhisel andmemudelil põhinev andmebaasisüsteem

Järgnevas peatükis annan ülevaate graafi definitsioonist ning graafipõhise andmemudeli andmebaasisüsteemidest.

3.3.1 Graaf

Graafi mõistete kirjeldus põhineb raamatul [17].

Graaf koosneb sõlmedest (tippudest) ning sõlmi omavahel siduvate servade (seoste) hulgast. Iga serv seob omavahel kuni kahte sõlme – serv mis seob ühte sõlme iseendaga nimetatakse silmuseks.

Graafis olevad servad võivad olla suunatud või suunamata. Suunatud graafi korral on iga serv ühesuunaline. Suunatud graafi serval on võimalik defineerida lähtesõlm ning sihtsõlm.

Ahelaks (ehk teeks) sõlmest u sõlme v nimetatakse sellist servade järjendit, kus iga eelneva serva lõppsõlm on järgneva serva algussõlmeks ning kus esimese serva algussõlmeks on u ning viimase serva lõppsõlmeks on v.

Tsükkel on kinnine ahel, kus alustades liikumist sõlmest u, on võimalik samasse sõlme u tagasi jõuda. Tsükliline graaf sisaldab vähemalt üht tsüklit. Tsükliteta (ehk atsükliline) graaf ei sisalda ühtegi tsüklit.

Sidus graaf on graaf, kus asendades kõik suunatud seosed suunamata seostega, leidub iga sõlmepaari {u,v} korral ahel sõlmest u sõlme v.

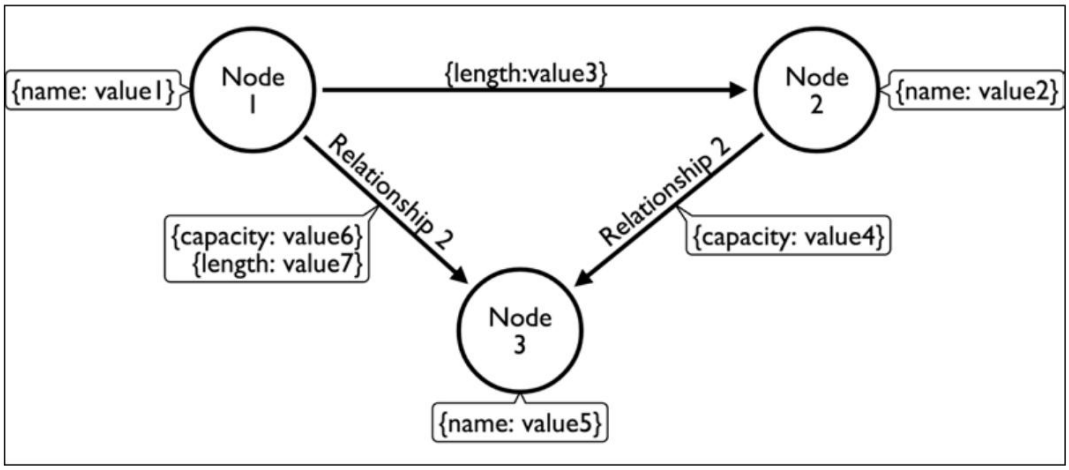
3.3.2 Graafandmebaas

Graafi andmemudeli puhul on andmed esitatud graafidena või andmestruktuuridega, mis üldistavad graafi mõistet.

Graafipõhistes andmebaasides (mis on loodud mõne graafi andmemudelit toetava andmebaasisüsteemi abil) on andmed sõlmedes, millel on omadused. Sõlmede vahel on

seosed, millel on ka omadused [18]. Käesolevas töös käsitletakse omadustega graafi (ingl *Property Graph*) andmemudelit.

Omadustega graafil põhinevas andmebaasisüsteemis puudub fikseeritud olemite struktuur. See tähendab, et servade ning sõlmede omadused ning andmetüübid ei järgi fikseeritud struktuuri. Sellest tulenevalt võib sama tüübiga sõlmedel olla erinev arv eri omadusi (atribuute), millel puudub kindel andmetüüp. Sama lähenemine rakendub ka servade (seoste) omadustele. Joonisel 3 on kujutatud omadustega graafi näide.



Joonis 3. Omadustega graafi näide [18].

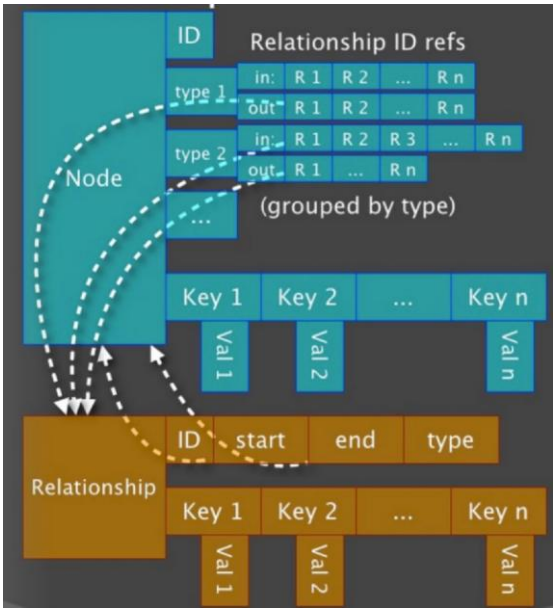
Graafi mudelil põhinevad andmebaasisüsteemid on olnud populaarsed juba 1990. esimesest poolest, kuna paljud eri süsteemides kasutatavad andmestikud on oma olemuselt graafina hõlpsasti jäljendatavad. Populaarseim graafi andmemudelit pakkuv andmebaasisüsteem – Neo4j – on 2019. aasta novembri seisuga kõikide andmebaasisüsteemide seas populaarsuselt 22. kohal [4].

3.3.3 Neo4j

Neo4j on 2007. aastal loodud omadustega graafi andmebaasisüsteem. Neo4j toetab sarnaselt tuntud relatsioonilistele andmebaasidele *ACID* transaktsioone. Neo4j andmebaasi põhielementideks on sõlmed (ingl *Nodes*) ning sõlmi siduvad servad (ingl *Relationships*) [18]. Sõlmedele omistatakse sõlme tüüpi määrav silt (ingl *Label*). Sama tüüpi sõlmed grupeeritakse sama sildiga. Näiteks on kasutajat iseloomustavad sõlmedel silt *User*. Sõlmi siduvatele servadele omistatakse serva tüüp (ingl *Relationship type*).

Servadele ja sõlmedele on võimalik omistada erinevaid atribuute. Atribuut on võti-väärtus andmepunkt, mis on igal relatsioonil ning sõlmed defineeritud dünaamiliselt - see

tähendab, et nii sõlmedel kui ka servadel ei ole fikseeritud atribuutide struktuure. Näiteks võivad kaks samat tüüpi sõlme omada erinevaid atribuute. Joonis 4 iseloomustab kuidas Neo4j süsteemis sõlmed, relatsioonid ning atribuudid on mälus hoitud.



Joonis 4. Neo4j andmete mälus salvestamise skeem [19].

Joonisel 4 on näha, et iga sõlmega salvestatakse viited antud sõlmega seotud relatsioonidele. Relatsioonide viited on grupeeritud relatsiooni tüübi ning suuna kaupa. Lisaks on igal sõlmel salvestatud temaga seotud atribuutide kogum lingitud listis (ingl *Linked List*). Igal serval on sarnaselt sõlmedele viide antud serva algus- ja lõppservale ning lisaks lingitud list servaga seotud atribuutidest. Päringute sooritamisel on andmebaasimootoril võimalik igast sõlmest liikuda edasi läbi sõlmega seotud servade jälgides vaid sõlme ning servade juures viiteid ilma, et oleks tarvis kasutada muid optimeeritud andmestruktuure. Võrreldes relatsioonilise andmemudeliga tähendab see optimaalsemat andmebaasiridade seoste leidmist, kuna puudub vajadus otsida viiteid läbi relatsiooniliste tabelite ning tabelite ridadele viitavaid indekseid (ingl *index free adjacency*). Indekseid kasutatakse graafandmebaasis seoste otsimise asemel tihti hoopis graafis päringu sisendsõlmede leidmiseks.

3.3.4 Cypher

Omadustega graafi andmemudeli üheks tuntuimaks andmebaasikeeleks on Cypher.

Cypher on SQL päringukeelest inspireeritud Neo4j andmebaasi päringukeel mis võimaldab andmeid pärida ning muuta graafandmebaasis. Cypher päringukeel põhineb mustrite otsingul – päringutes on võimalik defineerida sõlmede ning servade atribuutide

väärtuseid ning sõlmede vahelisi seoseid, mille kaudu andmebaasimootor graafis liigub [18].

3.4 Varasemad uurimused graafipõhise andmemudeli ning SQL andmebaasisüsteemide võrdlustes

Varasemalt on tehtud mitmeid uuringuid graafipõhiste andmemudelite ning SQL andmebaasisüsteemide võrdlustest.

2010. aastal publitseeritud artiklis [20] vaadeldakse kaht andmebaasisüsteemi: MySQL Community Server 5.1.42 (SQL) ja Neo4j versioon 1.0-b11 (graafipõhine). Kasutatavas andmestikuks loodi juhuslik suunatud atsükliline graaf esitamaks andmete päritoluga illustreerivat andmestikku. Uurimuse käigus loodi kokku 12 andmestikku, mille põhjal mõõdeti andmebaasi salvestusmahtu ning päringute kiiruseid. Süsteemide skaleeruvuse hindamiseks loodi andmed kasutades erinevaid andmetüüpe ning andmemahte - erinevate andmestike graafide suurused varieerusid 1000 sõlmest kuni 100 000 sõlmeni kasutades nii numbrilisi väärtuseid kui ka tekstilisi väärtuseid. Uuringu tulemustest selgus, et struktuuripäringute puhul oli Neo4j märgatavalt kiirem – suurtemate andmemahtude puhul oli vahe lausa 10 kordne. Mitte relatsiooniliste ehk andmehulga päringute puhul oli tulemus vastupidine – MySQL oli kõigis kogu andmehulka puudutavates päringutes kiirem. Lisaks anti subjektiive hinnang andmebaaside paindlikkusele, turvalisusele ning kasutajasõbralikkusele. Samas polnud välja toodud töös kasutatavad andmebaasipäringud ning nende käivitusplaanid ning tulemuste selgitus.

2013. aastal publitseeritud artiklis [21] koostatakse eksperiment võrdlemaks erinevaid andmebaasisüsteeme tehislikult loodud sotsiaalvõrgustiku iseloomustava andmestiku põhjal. Töös kasutatakse graafi andmemudelil põhinevaid andmebaasisüsteeme (Neo4j 1.8.2 ja Dex 4.7), RDF mudelil põhinevat RDF-3X ning SQL mudelil põhinevat PostgreSQL 9.1 ning Virtuoso 7.0. Andmestik luuakse luues objektid “isik” ja “veebileht”. Isikuid seob omavahel seos “sõber” ning isikud ja veebilehte seos “meeldib”. Testandmestiku võimalikult reaalse süsteemiga sarnaseks genereerimisel kasutati sotsiaalvõrgustiku Facebook 2012 aasta raportit sotsiaalvõrgustiku andmetest. Selles tulenevalt oli andmestikus objektidest 80% isikud ning 20% veebilehed. Eksperimendi koostamisel loodi 5 eri andmestiku hulka, varieerudes 1000 sõlmest kuni 10 miljoni

sõlmeni graafis. Uuringu tulemustes selgub, et graafandmebaasid on võimekamad iga andmehulga puhul ning suuremate andmemahtude puhul skaleeruvad paremini.

2016. aastal publitseeritud artiklis [22] vaadeldakse MySQL (v5.5.6) ning Neo4j (v1.8.1) andmebaaside võrdlust läbi personaliseeritud vähiravi rakenduse. Eksperimendiks koostatud andmestik sisaldas 22 objektitüüpi ning nendevahelisi seoseid, mis representeerivad vähiravi andmeid, näiteks patsient ja konkreetse vähi tüüp, ravim. Loodud andmestik genereeriti kolmes eri suuruses vastavalt 1000, 10 000 ning 100 000 erinevat objekti iga objektitüübi kohta. Lisaks defineeriti päringud, mida mõlemas andmebaasis iga andmestiku suuruse kohta genereeriti. Uurimuse tulemustes selgus, et MySQL oli valdavalt edukam enamikes päringutes. Neo4j eelis oli vaid kahes päringus, mille mõlemad päringud olid teiste vaadeldavate päringutega võrreldes keerukamas eeskätt tabelite liitmise hulga poolest. Neo4j oli selgelt kiirem nendes päringutes, kus tuli relatsioonilise andmebaasi puhul tuli rohkem tabeleid liita.

Gubichev [23] koostas oma eksperimendi uurimuse Lehigh ülikooli andmetöötlusvõrdluse raamistikul LUBM [24] võrreldes omavahel graaf ning relatsioonilisi andmebaase. Gubichev kasutas oma töös järgnevaid andmebaase: Virtuoso 6.1.8 (relatsiooniline), Virtuoso 7.1.0 (relatsiooniline), Neo4j 2.0.1 (graaf), Sparksee 5.0.0 (graaf) and TripleRush 2014 (eksperimentaalne RDF andmebaas). Uurimuses loodi kaks eri suurusega andmestikku, mille puhul võrreldi kõigi andmebaaside suutlikkust tagastada LUBM raamistiku pool defineeritud päringutele. Väiksema andmestiku puhul osutus enamus päringute puhul kiireimaks Virtuoso 7.1. Lisaks tuuakse välja, et TripleRush on väiksema andmestiku puhul võrdlemisi kiire ning edastab enamus teisi andmebaase. Samas suurema andmestiku puhul ei olnud TripleRushi kasutamine võimalik, kuna antud andmestik vajas TripleRushi poolelt liialt suurt mälukasutust. Suurema andmestiku puhul oli Virtuoso süsteemide puhul näha märgatavat aeglustumist – 2 testpäringu puhul 8-st Virtuoso ei suutnud tagastada päringu vastust. Neo4j puhul oli aga tulemus suuremal andmestikul veelgi halvem – 6 päringu puhul 8-st ei suutnud andmebaas tagastada vastust. Samas polnud artiklis välja toodud konkreetsed päringud ning põhjendused, miks andmebaas nende päringute puhul ei suutnud vastust tagastada.

Pacaci [25] koostas uurimisrühmaga eksperimendi võrdlemaks graaf ning relatsioonilist andmebaasi sotsiaalvõrgustikku imiteeriva andmestiku näitel. Eksperimendis kasutati

LDBC (ingl *Linked Data Benchmark Council*) sotsiaalvõrgustiku võrdlusraamistikku [26], mille puhul disainiti süsteem, mis võimaldas vaadeldavale andmebaasile tekitada järjestikkuses voos sõnumeid, mille puhul vaadeldav andmebaas teostas andmete lisamise ning muutmise operatsioone. Uurimuses oli graafandmebaasidest kasutusel Neo4j versioon 2.3.6 ning TitanDB versioon 1.1. Relatsioonilistest andmebaasidest oli kasutusel Virtuoso versioon 7.2.4 ning PostgreSQL 9.5. Uurimuse käigus loodi kaks eri suurusega andmestikku, mille puhul sooritati kõigis andmebaasides eeldefineeritud päringud. Eksperimendis vaadeldi 4 eri tüüpi lugemispäringut: konkreetse objekti otsing, objekti esimese astme seoste leidmine, objekti teise astme seoste leidmine, lühima tee leidmine kahe objekti vahel. Virtuoso osutus enamus päringutes kiiremaiks mõlemal andmestikul. PostgreSQL tulemused olid Virtuosoga võrredles enamus päringutes 10-40% aeglasemad, kuid siiski kiiremad kui Neo4j ja TitanDB. Erandiks oli lühima tee leidmine, mille puhul Neoj4 edestas PostgreSQLi ligikaudu 100 kordse vahega. Siiski oli Virtuoso kiireim ka lühima tee leidmisel. Pacaci märgib, et Neo4j puhul oli ainukesena märgata päringute kiiruse sõltumatust andmestikku suurusest – ehk päringute kiirused ei olnud oluliselt muutunud andmestikku kasvades. Samas vaadeldes väljatoodud eksperimendi tulemusi võib sama väita ka Virtuoso puhul.

Wecel-i [27] uurimisrühm koostas eksperimendi, mille käigus võrreldi relatsioonilisi ning graafandmebaaside suutlikkust sooritada lugemispäringuid ning andmete esmase sisselaadmise kiirust. Wecel kasutas eksperimendis telekommunikatsiooniettevõtte arvete koostamise andmestikku. Uurimuse käigus defineeriti neli eri tüüpi päringuid: andmete sisselugemise päringud, lihtsamad lugemispäringud ning keerukamad agregatsioonidest koosnevad lugemispäringud ning võrgustiku lugemispäringud. Kõigi kolme päringutüübi puhul kasutati kolme eri suurusega andmestikku: väike (1 miljon objekti), keskmine (5 miljonit objekti) ning suur (25 miljonit objekti). Eksperimendis kasutati järgnevaid andmebaase: MySQL, Neo4j, TitanDB ning RDF Virtuoso. Andmete sisselaadimisel osutusid kiiremaks MySQL ning TitanDB, mille puhul andmemahu kasvades andmete sisselugemise kiirust oluliselt ei mõjutanud. Lihtsamate lugemispäringute puhul olid MySQL ning Virtuoso selgelt kiiremad graafandmebaasidest, edestades mõningatel juhtudel Neo4j-d isegi sajakordselt. Keerukamate agregatsioonipäringute puhul olid MySQL ning Virtuoso veelgi edukamad võrreldes Neo4j ning TitanDB-ga. Virtuoso oli agregatsioonipäringutes kiireim. MySQL tulemus oli sarnane Virtuosole välja arvatud ühes päringus, kus Neo4j edestas ka MySQLi. Võrgustike lugemipäringus kordus

eelnevate lugemispäringute tulemus – MySQL ning Virtuoso olid selgelt kiiremad Neo4j-st. TitanDB ei suutnud suurema osa keerukamatest ning võrgustiku päringutest jooksutada. Paraku polnud uurimuse kirjelduses välja toodud milliseid andmebaasiversioone kasutati ning millised olid konkreetsed päringud.

Graafandmebaaside võrdlusuuringute puhul on mitmeid aspekte, mis võivad oluliselt mõjutada andmebaasi suutlikkust jooksutada võrreldavaid agregatsioonipäringuid, näiteks andmete olemus ning olemite omavaheline seotus – samuti ka milline on nominaalne koormus andmebaasile [28]. Lisaks on erinevatel päringukeeltel erinevad võimalused ning funktsioonid, mis ei mõjuta ainult päringu sooritamiseks kulunud aega vaid ka aega, mis kulub päringu käivitusplaani koostamiseks [29]. Nagu eelnevatest uurimustest võib järeldada, sõltub andmebaaside võrdluste tulemused suuresti sellest, millise domeeni andmestikul neid võrreldakse. Seetõttu võib järeldada, et laiapõhjaliste järelduste saamiseks on mõistlik võrdlusi läbi viia erinevates ärivaldkondades.
Senimaani ei ole autor veel leidnud uurimust, mis finantspettuste valdkonnas kahe konkreetse süsteemi võrdluses tooks välja tulemused koos andmebaasipäringute selgitustega, mis põhjendaksid võrreldavate päringute tulemusi. Lisaks on varasemates uurimustes tihtipeale kasutatud andmestik mitte avalik või piiratud kirjeldusega ning puudulike päringu tulemustega, mis aitaks lugejal mõista võrdluse tagamaid.

4 Eksperiment

Käesolevas peatükis on detailsemalt kirjeldatud, mis oli eksperimendi eesmärk, kuidas eksperiment läbi viidi, millised olid andmed ja millistele järeldustele jõuti.

4.1 Eksperimendi eesmärk

Eksperimendi eesmärk oli välja selgitada, milline on graafi mudelil põhineva andmebaasisüsteemi võimekus sooritada konkreetses ärivaldkonnas vajalikke andmeagregatsioone ning võrrelda seda identsel andmestikul relatsioonilise andmebaasisüsteemi võimekusega. Käesolevas töös vaadeldakse võimekusena konkreetselt andmeagregatsioonide päringute kiirust. Eksperiment ei käsitle endas andmete lisamise, muutmise ja kustutamise operatsioonide kiiruseid.

4.2 Eksperimendi disain

Eksperimendi koostamisel peeti silmas, et võrreldavad andmebaasisüsteemid oleks võimalikult identses keskkonnas. See tähendab, et mõlema süsteemi puhul kasutati sama riistvara ning sama andmestikku.

Eksperiment viidi läbi Amazon Web Service (AWS) keskkonnas. Mõlema süsteemi puhul kasutati AWS-i poolset toodet virtuaalserveri tüüpi *t3.2xlarge*. Riistvaraliselt on teenusepakkuja andmetel antud toote puhul kasutusel Intel Xeon Platinum 8000 seeria protsessorid parameetritega: 8 vCPU ja 32GB vahemälu.

Relatsioonilise andmebaasina kasutati PostgreSQL versiooni 12.0 vaikimisi seadistusega. PostgreSQL on 2019. aasta oktoobri seisuga andmebaaside populaarsuse edetabelis 4. kohal [4]. Vaadates viimase 6 aasta andmebaasisüsteemide populaarsuse trende on selgelt näha, et kui populaarseimad andmebaasid (Oracle, MSSQL ning MySQL) on langustrendis, siis PostgreSQL on viimase 6 aasta jooksul ligi 3 korda populaarsust kasvatanud [4]. Lisaks väidavad mitmed allikad, et PostgreSQL on populaarsete SQL andmebaasisüsteemide seas efektiivsem keerukamate analüütiliste päringute

jooksutamises [30, 31]. Graafimudelil põhineva andmebaasina kasutati Neo4j versioon 3.5.12 Enterprise vaikimisi seadistusega. Eksperimendi läbiviimise ajal olid need kõige hilisemalt avaldatud versioonid mõlemast andmebaasist.

Eksperiment koosnes 3 suuremast etapist, mis on järgnevalt kirjeldatud.

4.2.1 Andmebaaside ülesseadmine ning testandmestiku loomine

Eksperimendi ettevalmistamiseks oli vajalik luua testandmestik. Testandmestikku loomisel peeti silmas, et andmete hulk oleks piisavalt mahukas, reaalses ärivaldkonnas kasutatavaid andmestikke, kuid samas ka sobiva keerukusega, et lugejal oleks võimalik hõlpsasti aru saada andmestiku sisust ning omavahelistest seostest. Testandmete korrektsel loomisel on suur tähtsus, sest andmestiku sisu mõjutab suurel määral milline on andmebaaside võimekus päringute sooritamiseks. Käesolevas töös rõhuti, et andmestikus olevad objektid oleks omavahel piisavalt seostatud, et tekiks võrgustike efekt. Testandmestik genereeriti PostgreSQL andmebaasis ning samad andmed imporditi CSV failide abil Neo4j andmebaasi. PostgreSQLis oli andmetebaasi kogumaht 432MB, Neo4j andmebaasis 1.4GB

4.2.2 Andmebaasisüsteemides päringute koostamine

Käesolevas töös defineeriti 12 agregatsiooni, mida mõlema andmebaasi päringud peavad tagastama. Päringute defineerimisel vaadeldi varasemaid uurimusi petturlike maksete tuvastamise süsteemidest ning uurimustes väljatoodud andmeagregatsioone. Päringute koostamisel peeti silmas eeskätt päringute kiirust, kuid samas ka lihtsust ning loetavust. Kuna kasutatud andmestik on mõlemas andmebaasis sama, siis peavad mõlema andmebaasi päringud tagastama samad tulemused. Koostatud andmebaasipäringud on välja toodud töö lisades (Lisa 2 – Lisa 25).

4.2.3 Andmebaasisüsteemides päringute käivitamine ning tulemuste märkimine

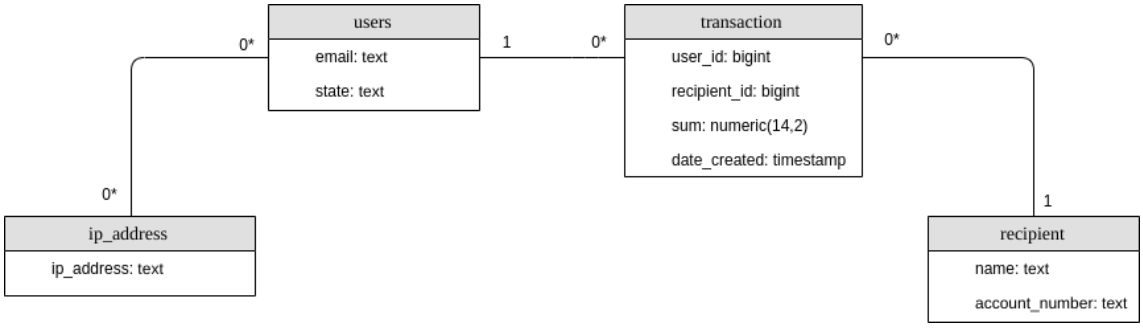
Kõigi agregatsioonide puhul märgiti ära mõlemas andmebaasis kogu käivitusaeg ning päringu tulemus. Lisaks sellele, kasutati päringute profileerimiseks mõlemas andmebaasis ka päringu selgitusplaane, mis on lisatud viitena iga tulemuse juurde. Päringu selgitusplaan (ingl *execution plan*) annab lugejale täpse ülevaate, milline oli konkreetse päringu puhul andmebaasis käivitunud tabelite ning indeksite lugemise järjekord ning

võimaldab valideerida, kas andmebaas sooritas päringu nii nagu päringu koostaja optimaalseks pidas.

Viimases alapeatükis tehakse kokkuvõtlik ülevaade eksperimendi tulemustest ning mõtestatakse lahti agregatsiooni tulemustest nähtuvad kummagi andmebaasi tugevused

4.2.4 Kontseptuaalne andmemudel

Eksperimendis kasutatav andmemudel oli lihtsustatud jäljendus finantsettevõtte illegaalsete maksete tuvastamise süsteemist. Andmemudel koosnes neljast põhiobjektist - kasutajad (ingl *users*), IP-aadressid, maksed (ingl *transactions*) ja maksete saajad (ingl *recipients*). Lisaks on defineeritud objektide omavahelised seosed. Joonis 5 illustreerib koostatud testandmestiku loogilist andmemudelit.



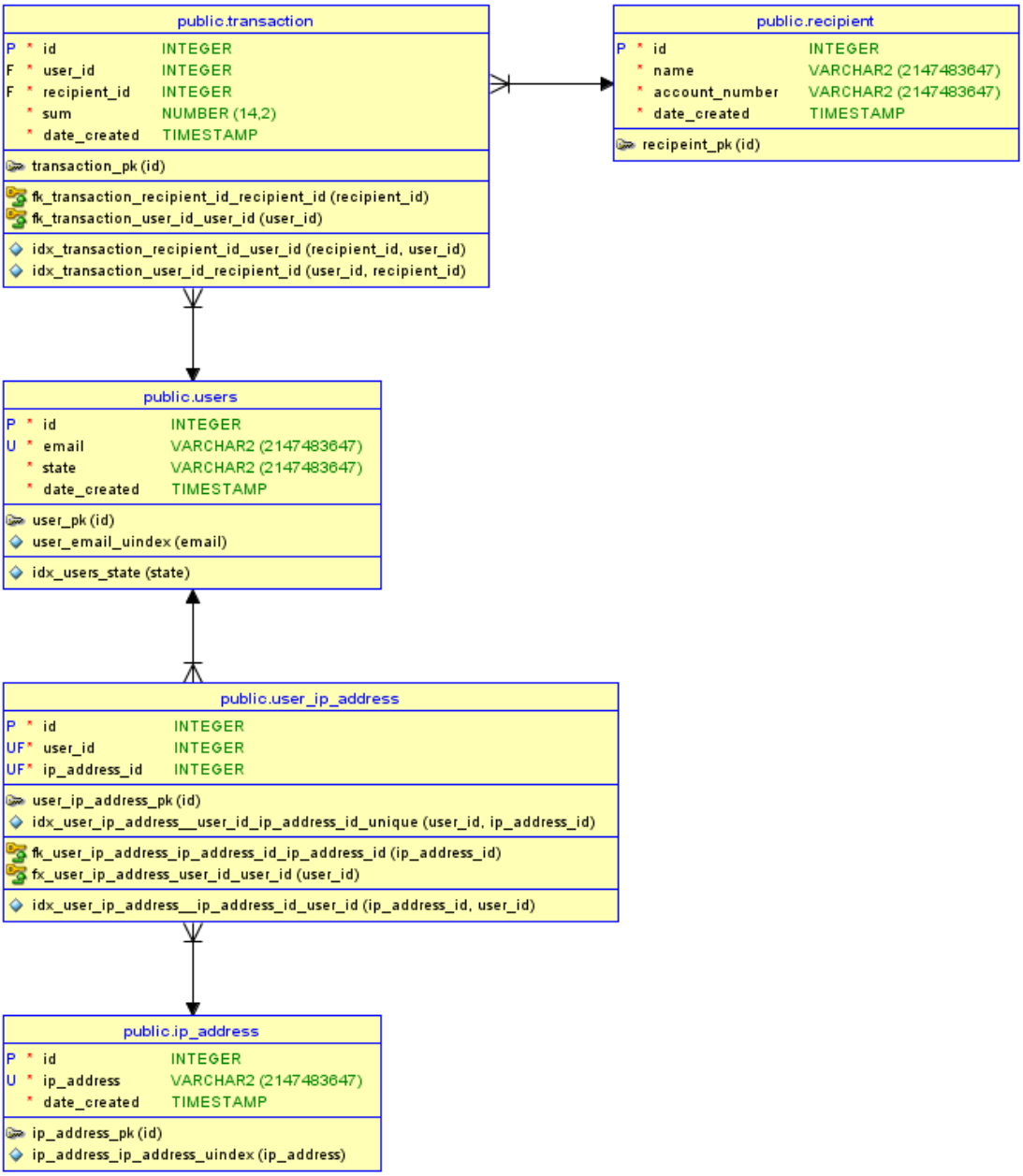
Joonis 5. Testandmestiku loogiline andmemudel.

Kasutaja on süsteemis keskne uurimisobjekt, mida süsteem üritab tuvastada kui potentsiaalselt pahatahtliku iseloomuga. Igal kasutajal on kohustuslik atribuut e-posti aadress ning staatus (ingl *state*). Igal kasutajal peab olema unikaalne e-posti aadress ehk kahel erineval kasutajal ei saa olla sama e-posti aadress. Staatus väljendab pettuste ennetamise süsteemis konkreetse kasutaja kahtlustuse olekut. Ehk kui süsteemis tuvastatakse petturlik kasutaja, siis muudetakse tema staatus vastavalt. Petturlikke kasutajad märgitakse staatusega "*BAD*". Iga kasutajaga võib olla seotud mitu erinevat IP-aadressi. Näiteks võib sama kasutaja siseneda internetipanka oma koduvõrgust või töövõrgust. Samuti võib olla ühe IP-aadressiga seotud mitu kasutajat, näiteks erinevad pereliikmed kasutavad sama internetipanka oma kodu internetivõrgust. Kasutajatega on lisaks seotud maksed (ingl *transaction*) ning maksetega omakorda makse saajad (ingl *recipient*). Ühel kasutajal võib olla mitu makset, kuid ühe maksega võib olla seotud vaid üks kasutaja. Sarnaselt võib ühe maksega olla seotud vaid üks makse saaja - samas ühele makse saajale võib olla mitu erinevat makset tehtud. Makse olemit iseloomustab lisaks saajale ning makse teinud kasutajale ka summa (ingl *sum*) ning makse tegemise aeg (ingl

date_created). Makse saajat iseloomustab nimi (ingl *name*) ja saaja pangakonto number (ingl *account_number*).

4.2.5 Andmebaaside realiseerimine

Lähtuvalt eelpool kirjeldatud andmemudelist realiseeriti mõlemas andmebaasis identne andmestik. PostgreSQL puhul luuakse tabelid kasutades *CREATE TABLE* käsku. Lisaks põhiolemite tabelite loomisele luuakse ka vajalikud külgnevusnimistu tabelid, mis viitavad välisvõtmega põhitabelitele. Näiteks luuakse tabel *user_ip_account* kasutajate ning IP-aadresside vaheliste seoste hoidmiseks. Joonis 6 kirjeldab relatsioonilise andmebaasi diagrammi.

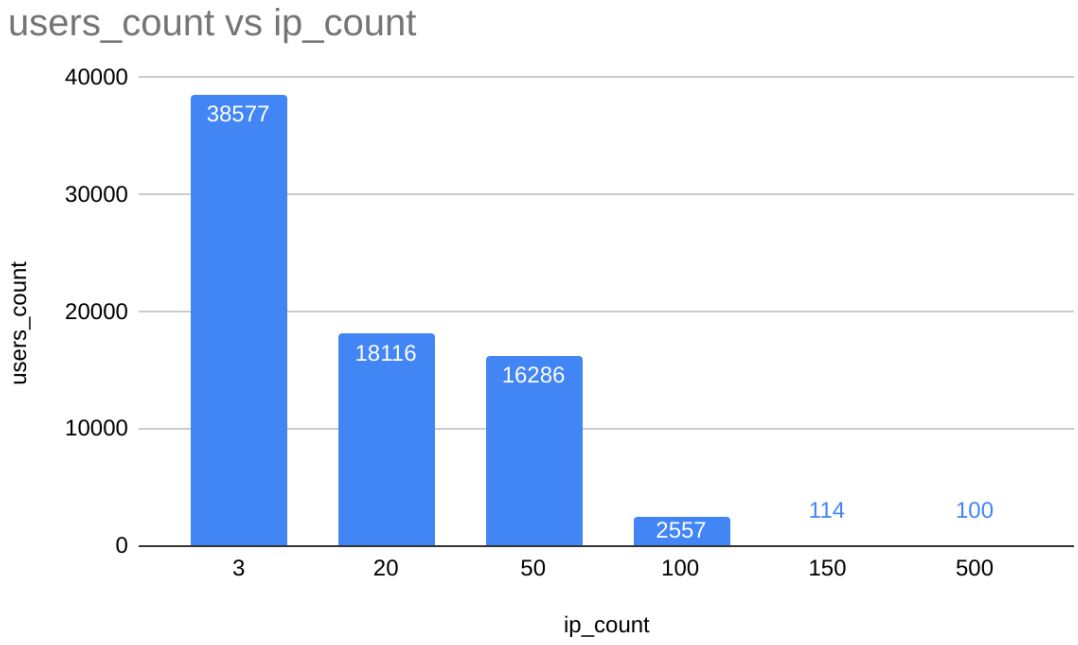


Joonis 6. PostgreSQL testandmestiku andmebaasi diagramm.

4.2.6 Testandmete iseloomustus

Testandmed genereeriti PostgreSQL andmebaasis autori poolt. Testandmete genereerimisel lähtuti järgnevatest nõuetest – testandmete hulk peab olema piisavalt suur, et iseloomustada reaalse süsteemi omadusi ning objektide seostatus testandmetes ei tohiks olla ühtlane – näiteks on kasutajaid, kellel on 5 makset kui ka kasutajaid, kellel on 500 makset.

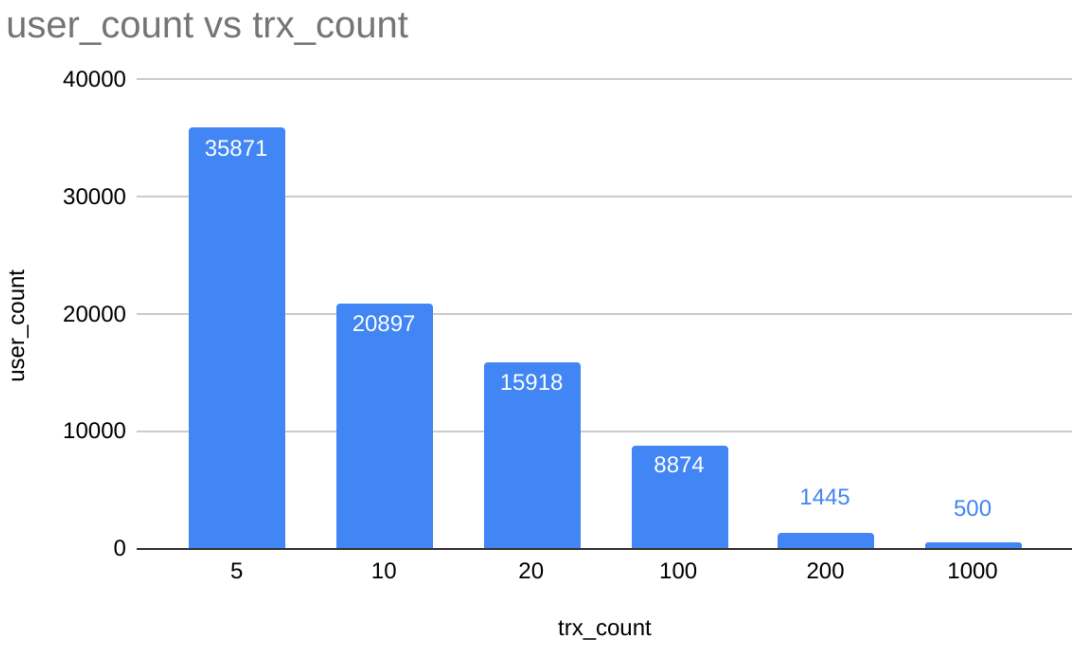
Autor genereeris 100 000 kasutaja objekti vastavalt andmemudeli reeglitele. Lisaks genereeriti 200 000 IP aadressi ning 1 799 416 kasutaja ning IP-aadressi vahelist seost. Keskmiselt teeb see iga kasutaja kohta ligi 18 IP-d. Joonis 7 näitab testandmetes kasutaja ning IP aadresside vaheliste seoste hulka.



Joonis 7. Kasutajate ning IP aadresside seostatus testandmestikus.

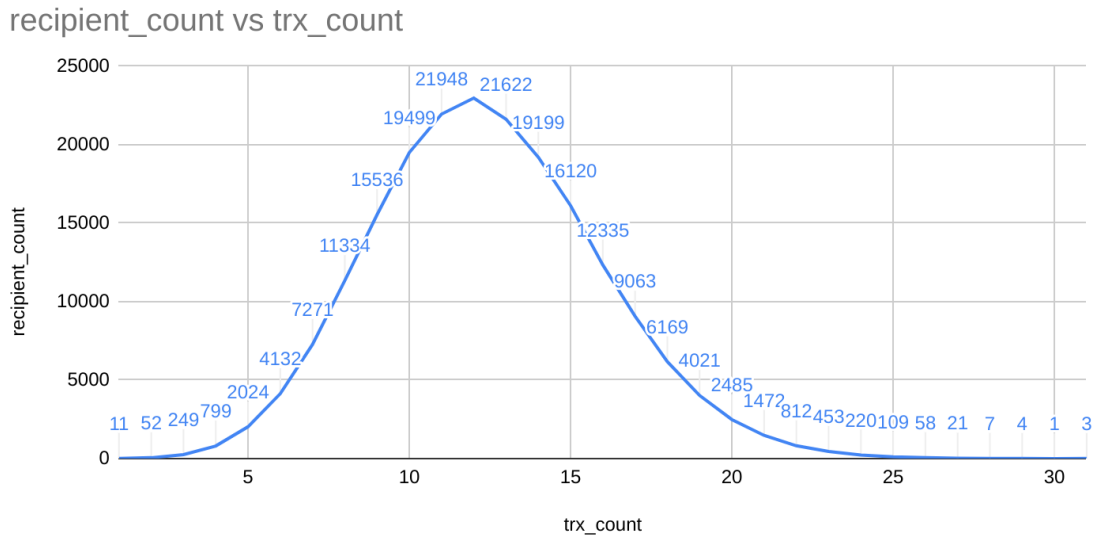
Joonis 7 näitab, et on kasutajaid, kellega on seotud 500 erinevat IP-d kui ka kasutajaid samas suurem osa kasutajatel on kuni 20 IP-aadressi seotud.

Sarnaselt kasutajate IP aadressidega, loodi ka kasutajate maksete arv ebavõrdselt. Süsteemis genereeriti 2 482 375 makset, mille jaotatust kasutajate vahel näitab Joonis 8.



Joonis 8. Kasutajate maksete arvu distributsioon testandmetes.

Maksete loomiseks oli tarvis luua ka maksete saajate andmehulk. Kokku loodi süsteemis 200 000 makse saajat. Ühel makse saajaga võib olla seotud mitu makset. Makse saajate distributsioon seotud maksete hulga suhtes kujutab Joonis 9.



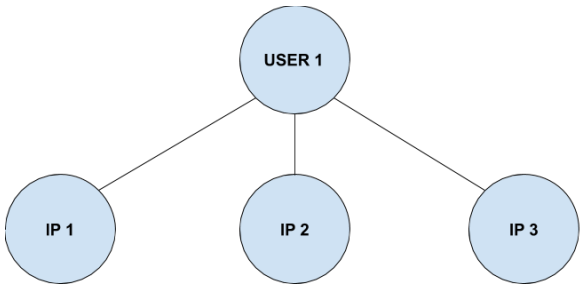
Joonis 9. Makse saajate ning seotud maksete distributsioon testandmestikus.

4.3 Uuritavad andmeagregatsioonid

Toetudes varasematele uurimustele illegaalsete maksete tuvastamise süsteemides on antud eksperimenti valitud 12 agregatsiooni, mis iseloomustavad andmepunkte, mille puhul võrreldakse mõlema süsteemi kiirust. Järgnevalt kirjeldatakse detailsemalt iga agregatsiooni sisu. Iga agregatsiooni puhul on välja toodud kirjeldus, viited artiklitele ning joonis näitega selgitamaks agregatsiooni sisu ning agregatsiooni väärtus näitel. Lisaks on toodud viited mõlema andmebaasi käivitusplaanidele töö lisas. Igale agregatsioonile antakse unikaalne identifikaator, et eksperimendi tulemusi hõlpsamalt viidata. Joonistel on otsitavaks sisendiks *User 1*. Joonistel on kahtlased objektid märgitud punase tooniga illustreerimaks petturlikku staatust.

4.3.1 Kasutajaga seotud IP-aadresside arv

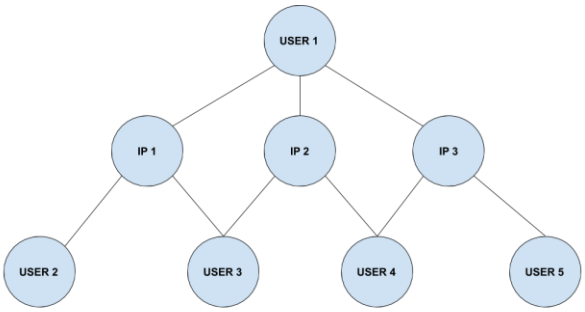
Identifikaator: AGR1
Kirjeldus: Kasutajaga seotud erinevate IP-aadresside arv.
Viited: 3, 8, 9, 10
Käivitusplaanid: Lisa 2, Lisa 3
Väärtus näitel: 3



Joonis 10. Kasutajaga seotud IP-aadresside arv näide.

4.3.2 Kasutajaga seotud teiste kasutajate arv läbi IP-aadresside

Identifikaator: AGR2
Kirjeldus: Kasutajaga seotud erinevate kasutajate arv läbi seotud IP-aadresside.
Viited: 3, 8, 9, 10
Käivitusplaanid: Lisa 4, Lisa 5
Väärtus näitel: 4



Joonis 11. Kasutajaga seotud teiste kasutajate arv läbi IP-aadresside.

4.3.3 Kasutajaga seotud erinevate kahtlaste isikute arv läbi IP-aadresside

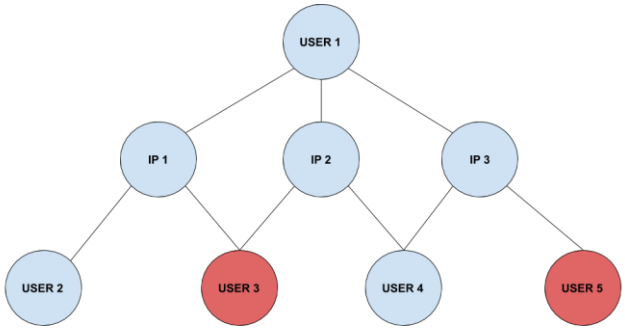
Identifikaator: AGR3

Kirjeldus: Kasutajaga seotud erinevate kahtlaste kasutajate arv läbi seotud IP-aadresside.

Viited: 3, 8, 9, 10

Käivitusplaanid: Lisa 6, Lisa 7

Väärtus näitel: 2



Joonis 12. Kasutajaga seotud teiste kahtlaste kasutajate arv läbi IP-aadresside.

4.3.4 Kasutajaga seotud IP-aadresside arv, millega on seotud vähemalt üks kahtlane kasutaja

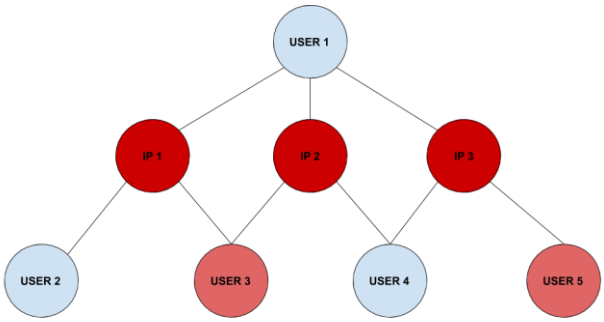
Identifikaator: AGR4

Kirjeldus: Kasutajaga seotud IP-aadresside arv, millega on seotud vähemalt üks kahtlane kasutaja

Viited: 3, 8, 9, 10

Käivitusplaanid: Lisa 8, Lisa 9

Väärtus näitel: 3



Joonis 13. Kasutajaga seotud IP-aadresside arv, millega on seotud vähemalt üks kahtlane kasutaja.

4.3.5 Kasutajaga seotud IP-aadresside suurim kahtlaste kasutajate hulk läbi ühe seotud IP-aadressi

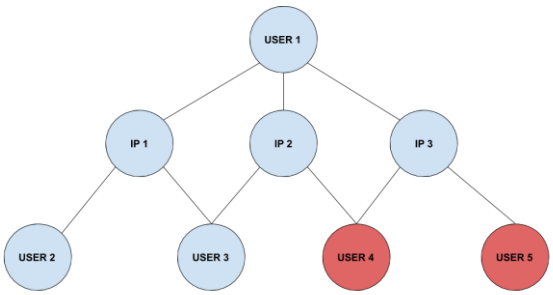
Identifikaator: AGR5

Kirjeldus: Vaadeldes kasutajaga seotud IP-aadresse, milline on suurim kahtlaste isikute arv läbi ühe IP aadressi

Viited: 3, 8, 9, 10

Käivitusplaanid: Lisa 10, Lisa 11

Väärtus näitel: 2



Joonis 14. Kasutajaga seotud IP-aadresside suurim kahtlaste kasutajate hulk.

4.3.6 Kasutajaga läbi teise astme seotud kahtlaste isikute arv läbi IP-aadresside

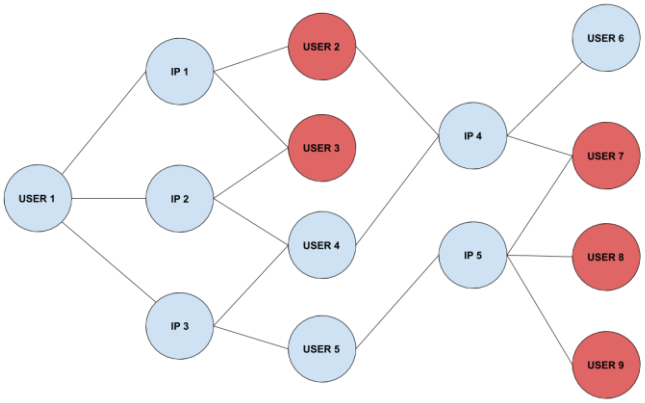
Identifikaator: AGR6

Kirjeldus: Vaadeldes kasutajaga seotud IP-aadresse ja nendega seotud kasutajate teisi seotud IP-aadresse ja omakorda nendega seotud kasutajaid, kui mitu kahtlast isikut nende hulgas leidub

Viited: 3, 32, 33

Käivitusplaanid: Lisa 12, Lisa 13

Väärtus näitel: 3



Joonis 15. Kasutajaga IP-aadressiga läbi teise astme seotud kasutajate arv.

4.3.7 Kasutajaga seotud teiste kasutajate arv läbi ühiste makse saajate

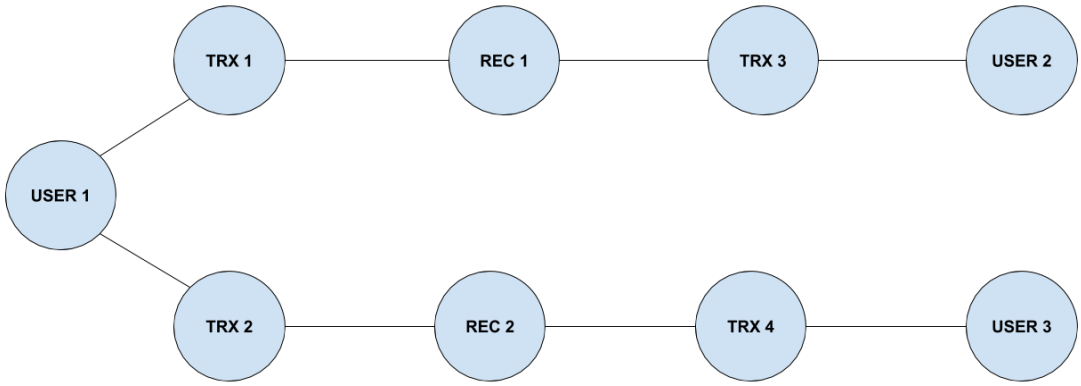
Identifikaator: AGR7

Kirjeldus: Vaadeldes kasutajaga makseid ning makse saajad, kui mitu teist isikut saadavad raha samadele maksete saajatele

Viited artiklitele: 5, 7

Käivitusplaanid: Lisa 14, Lisa 15

Väärtus näitel: 2



Joonis 16. Kasutajaga läbi ühise makse saaja seotud kasutajad.

4.3.8 Kasutajaga seotud teiste kahtlaste kasutajate arv läbi ühise makse saajate

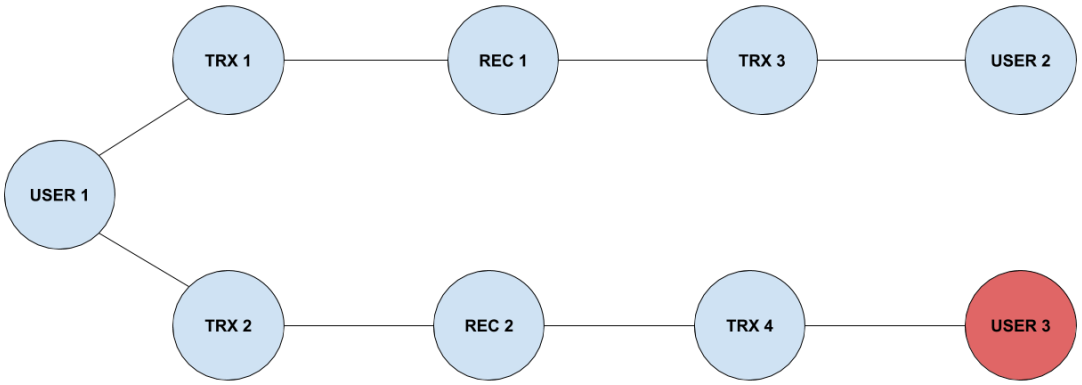
Identifikaator: AGR8

Kirjeldus: Vaadeldes kasutajaga makseid ning makse saajad, kui mitu teist kahtlast kasutajat saadavad raha samadele maksete saajatele

Viited artiklitele: 5, 7

Käivitusplaanid: Lisa 16, Lisa 17

Väärtus näitel: 1



Joonis 17. Kasutajaga seotud teised kahtlased isikud läbi ühiste makse saajate.

4.3.9 Kasutajaga seotud teiste kahtlaste kasutajate arv läbi teise ringi ühise makse saajate

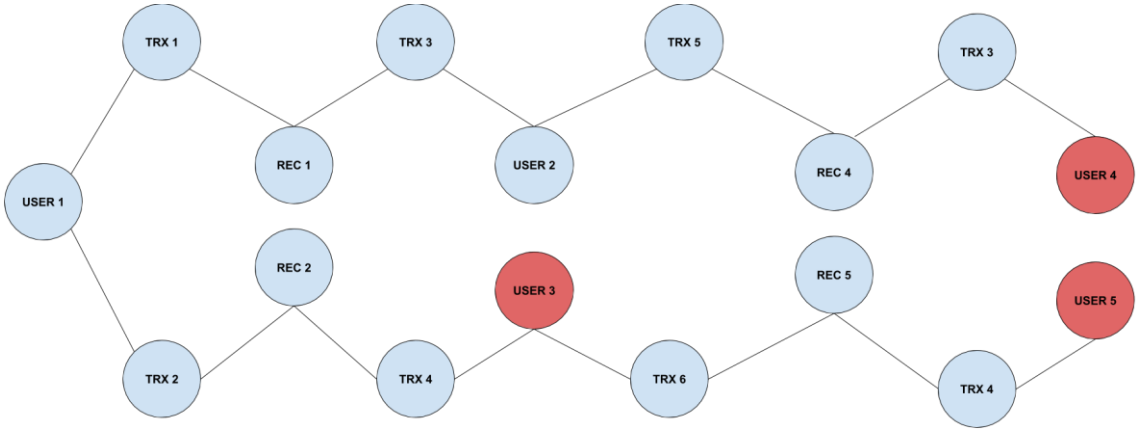
Identifikaator: AGR9

Kirjeldus: Vaadeldes kasutajaga seotud makseid ning makse saajad, kui mitu teist kahtlast kasutajat saadavad raha samadele maksete saajatele, kellele saadavad raha vaadeldava kasutajaga ühised makse saajad

Viited artiklitele: 5, 7

Käivitusplaanid: Lisa 18, Lisa 19

Väärtus näitel: 2 (*User 4, User 5*)



Joonis 18. Kasutajaga läbi ühise makse saaja teise ringi seotud kahtlased kasutajad.

4.3.10 Kasutajaga varasema 24 tunni tehingute summa

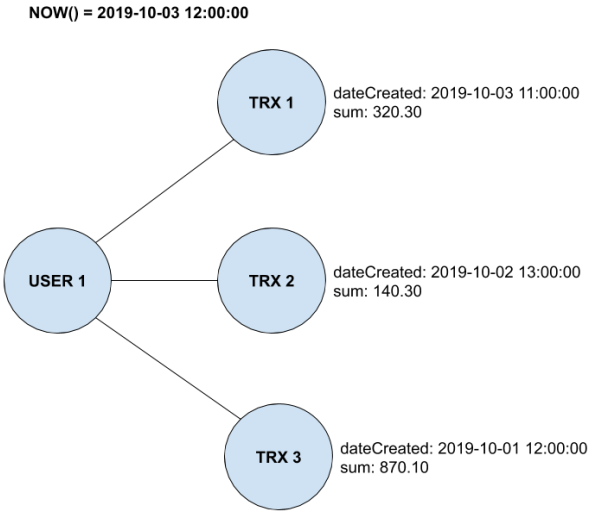
Identifikaator: AGR10

Kirjeldus: Vaadeldes kasutajaga makseid, mis on viimase 24 tunni jooksul tehtud tehingute summa

Viited artiklitele: 7, 12

Käivitusplaanid: Lisa 20, Lisa 21

Väärtus näitel: 460.60



Joonis 19. Kasutaja varasema 24 tunni tehingute summa.

4.3.11 Kasutajaga varasema 7 päeva tehingute summa

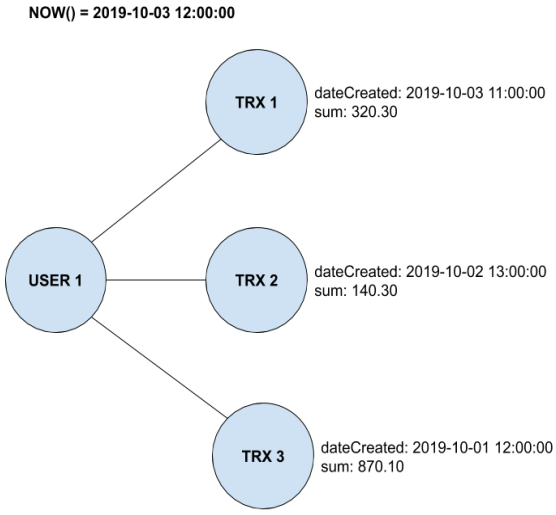
Identifikaator: AGR11

Kirjeldus: Vaadeldes kasutajaga makseid, mis on viimase 7 päeva jooksul tehtud tehingute summa

Viited artiklitele: 7, 12

Käivitusplaanid: Lisa 22, Lisa 23

Väärtus näitel: 1330.70



Joonis 20. Kasutaja varasema 7 päeva tehingute summa.

4.3.12 Kasutaja seotud teiste kasutajate arv, kes kasutavad ühiseid IP-aadresse ning makse saajaid

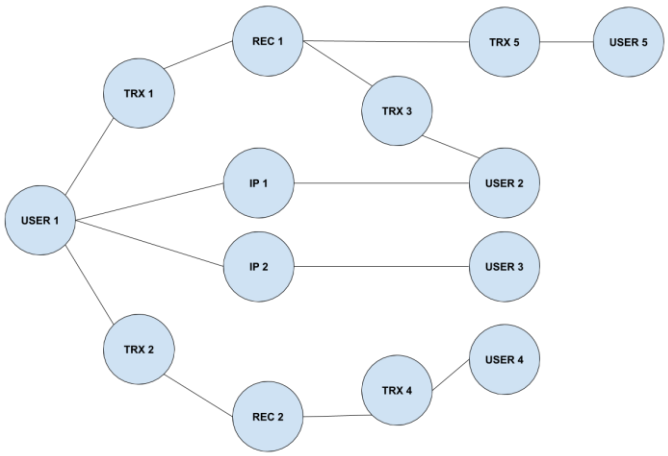
Identifikaator: AGR12

Kirjeldus: Vaadeldes kasutajaga seotud IP- aadresse ning makse saajaid, kui palju on teisi kasutajaid, kes kasutavad samu IP-aadressi ning makse saajat

Viited artiklitele: 3

Käivitusplaanid: Lisa 24, Lisa 25

Väärtus näitel: 1 (*User 2*)



Joonis 21. Kasutajaga seotud teised kasutajad läbi ühiste IP-aadressi ning makse saaja.

4.4 Eksperimendi sooritamine

Eksperimendi läbiviimiseks sooritati mõlemas andmebaasis vajalikud päringulaused. PostgreSQL andmebaasipäringuid jooksutati DataGrip andmebaasihaldus töövahendiga. Neo4j päringuid käivitati Neo4j platvormi käsurea tööriistaga *cypher-shell*. Mõlemas andmebaasisüsteemis käivitatud päringulaused on välja toodud tabelis 1. Iga päringuga on vastav päringu käivitusplaan leitav töö lisades (Lisa 2 - Lisa 25).

Tabel 1. PostgreSQL ja Neo4j andmebaasipäringud.

Agregatsiooni identifikaator	PostgreSQL	Neo4j

AGR1	`SELECT`
	`  user_id,`
	`  count(ip_address_id)`
	`as ip_total_count`
	`FROM user_ip_address`
	`WHERE`
		`MATCH (user:User)-`
		`[r:USED]->(ips)`
		`WHERE user.id IN ['53429',`
		`'77649', '98627', '2486',`
		`'86675', '20274']`
		`RETURN user.id,`

Agregatsiooni identifikaator	PostgreSQL	Neo4j
	user_ip_address.user_id IN (53429, 77649, 98627, 2486, 86675, 20274) GROUP BY user_id;	count(DISTINCT ips);
AGR2	SELECT uipa.user_id, count(distinct uipa2.user_id) FILTER (WHERE uipa2.user_id != uipa.user_id) as ip_total_other_users FROM user_ip_address uipa JOIN user_ip_address uipa2 ON (uipa2.ip_address_id = uipa.ip_address_id) WHERE uipa.user_id IN (53429, 77649, 98627, 2486, 86675, 20274) GROUP BY uipa.user_id;	MATCH (u:User)-[r1:USED]->(ip:IpAddress)<-[r2:USED]-(u2:User) WHERE u.id IN ['53429', '77649', '98627', '2486', '86675', '20274'] RETURN u.id,count(distinct u2) AS ip_total_other_users;
AGR3	SELECT uipa.user_id, count(distinct uipa2.user_id) FILTER (WHERE uipa2.user_id != uipa.user_id) as ip_total_other_users_bad FROM user_ip_address uipa JOIN user_ip_address uipa2 ON (uipa2.ip_address_id = uipa.ip_address_id) JOIN users u ON (uipa2.user_id = u.id AND state = 'BAD') WHERE uipa.user_id IN (53429, 77649, 98627, 2486, 86675, 20274) GROUP BY uipa.user_id;	MATCH (u:User)-[r1:USED]->(ip:IpAddress)<-[r2:USED]-(u2:User) WHERE u.id IN ['53429', '77649', '98627', '2486', '86675', '20274'] AND u2.state = 'BAD' RETURN u.id,count(distinct u2) AS ip_total_other_users_bad;
AGR4	SELECT uipa.user_id, count(distinct uipa.ip_address_id) as total_suspicious_ip_count FROM user_ip_address uipa JOIN user_ip_address uipa2 ON (uipa2.ip_address_id = uipa.ip_address_id)	MATCH (user:User)-[r:USED]->(ips)<-[:USED]-(u3:User) WHERE user.id IN ['53429', '77649', '98627', '2486', '86675', '20274'] AND u3.state = 'BAD' WITH user.id AS userid,

Agregatsiooni identifikaator	PostgreSQL	Neo4j
	JOIN users u ON (uipa2.user_id = u.id AND state = 'BAD') WHERE uipa.user_id IN (53429, 77649, 98627, 2486, 86675, 20274) GROUP BY uipa.user_id;	ips.id AS ipid, count(u3) as bad_count RETURN userid, count(ipid) as total_suspicious_ip_count;
AGR5	SELECT users_ips_stats.d_user_id, max(users_ips_stats.fraud_count) as max_fraud_count_on_ip FROM (SELECT uipa.user_id as d_user_id, count(distinct u.id) FILTER (WHERE u.state = 'BAD') as fraud_count FROM user_ip_address uipa JOIN user_ip_address uipa2 ON (uipa2.ip_address_id = uipa.ip_address_id) JOIN users u ON (uipa2.user_id = u.id) WHERE uipa.user_id IN (53429, 77649, 98627, 2486, 86675, 20274) GROUP BY uipa.user_id, uipa.ip_address_id) as users_ips_stats GROUP BY d_user_id;	MATCH (user:User)-[r:USED]->(ips)<-[:USED]-(u2:User) WHERE user.id IN ['53429', '77649', '98627', '2486', '86675', '20274'] AND u2.state = 'BAD' WITH user.id as user_id, ips, count(u2) as fraud_count RETURN user_id, max(fraud_count) AS max_fraud_count_on_ip;
AGR6	SELECT uipa.user_id, *count*(distinct uipa4.user_id) FROM user_ip_address uipa JOIN user_ip_address uipa2 ON (uipa2.ip_address_id = uipa.ip_address_id AND uipa2.user_id != uipa.user_id) JOIN user_ip_address uipa3 ON (uipa3.user_id = uipa2.user_id AND uipa3.ip_address_id !=	MATCH (user:User)-[r:USED]->(ips)<-[:USED]-(u2:User)-[r2:USED]->(ips2)<-[:USED]-(u3:User) WHERE user.id IN ['53429', '77649', '98627', '2486', '86675', '20274'] AND u3.state = 'BAD' RETURN user.id, count(distinct u3) as bad_count;

Agregatsiooni identifikaator	PostgreSQL	Neo4j
	uipa2.ip_address_id) JOIN user_ip_address uipa4 ON (uipa4.ip_address_id = uipa3.ip_address_id AND uipa4.user_id != uipa3.user_id AND uipa4.user_id != uipa2.user_id) JOIN users u ON (uipa4.user_id = u.id AND u.state = 'BAD') WHERE uipa.user_id IN (53429, 77649, 98627, 2486, 86675, 20274) GROUP BY uipa.user_id;	
AGR7	SELECT trx.user_id, count(distinct trx2.user_id) as total_user_count_via_recip ients FROM transaction trx JOIN transaction trx2 ON (trx2.recipient_id = trx.recipient_id AND trx2.user_id != trx.user_id) WHERE trx.user_id IN (53429, 77649, 98627, 2486, 86675, 20274) GROUP BY trx.user_id;	MATCH (user:User)- [r:SUBMITTED]-(trx)- [po:PAYED_OUT]- (rec:Recipient)<- [po2:PAYED_OUT]-(trx2)- [r2:SUBMITTED]-(u2:User) WHERE user.id IN ['53429', '77649', '98627', '2486', '86675', '20274'] RETURN user.id, count(distinct u2) as users_count;
AGR8	SELECT trx.user_id, count(distinct trx2.user_id) FROM transaction trx JOIN transaction trx2 ON (trx2.recipient_id = trx.recipient_id AND trx2.user_id != trx.user_id) JOIN users u ON (u.id = trx2.user_id AND u.state = 'BAD') WHERE trx.user_id IN (53429, 77649, 98627, 2486, 86675, 20274) GROUP BY trx.user_id;	MATCH (user:User)- [r:SUBMITTED]-(trx)- [po:PAYED_OUT]- (rec:Recipient)<- [po2:PAYED_OUT]-(trx2)- [r2:SUBMITTED]-(u2:User) WHERE user.id IN ['53429', '77649', '98627', '2486', '86675', '20274'] AND u2.state = 'BAD' RETURN user.id, count(distinct u2) as bad_users_count;

Agregatsiooni identifikaator	PostgreSQL	Neo4j
AGR9	SELECT trx.user_id, count(distinct trx4.user_id) as user_count FROM transaction trx JOIN transaction trx2 ON (trx2.recipient_id = trx.recipient_id AND trx2.user_id != trx.user_id) JOIN transaction trx3 ON (trx3.user_id = trx2.user_id AND trx3.recipient_id != trx2.recipient_id) JOIN transaction trx4 ON (trx4.recipient_id = trx3.recipient_id AND trx4.user_id != trx3.user_id) JOIN users u ON (u.id = trx4.user_id AND u.state = 'BAD') WHERE trx.user_id IN (53429, 77649, 98627, 2486, 86675, 20274) GROUP BY trx.user_id;	MATCH (user:User)-[r:SUBMITTED]-(trx)-[po:PAYED_OUT]-(rec:Recipient)<-[po2:PAYED_OUT]-(trx2)-[r2:SUBMITTED]-(u2:User)-[r3:SUBMITTED]-(trx3)-[po3:PAYED_OUT]-(rec2:Recipient)<-[po4:PAYED_OUT]-(trx4)-[r4:SUBMITTED]-(u3:User) WHERE user.id IN ['53429', '77649', '98627', '2486', '86675', '20274'] AND u3.state = 'BAD' RETURN user.id, count(distinct u3) as bad_user_count;

AGR10	SELECT
	  user_id,
	  sum(sum)
	FROM transaction
	WHERE user_id IN (53429, 77649, 98627, 2486, 86675, 20274)
	 AND date_created between to_timestamp('2019-11-01', 'YYYY-MM-DD') and to_timestamp('2019-11-02', 'YYYY-MM-DD')
	GROUP BY user_id;
		MATCH (user:User)-[r:SUBMITTED]-(trx)
		WHERE user.id IN ['53429', '77649', '98627', '2486', '86675', '20274']
		AND trx.dateCreated < datetime({epochmillis:apoc.date.parse('2019-11-02 00:00:00')})
		AND trx.dateCreated >= datetime({epochmillis:apoc.date.parse('2019-11-01 00:00:00')})
		RETURN user.id, sum(trx.sum) as last_1_d_sum;

AGR11	SELECT
	  user_id,
	  sum(sum)
	FROM transaction
	WHERE user_id IN (53429, 77649, 98627, 2486, 86675,
		MATCH (user:User)-[r:SUBMITTED]-(trx)
		WHERE user.id IN ['53429', '77649', '98627', '2486', '86675', '20274']
		AND trx.dateCreated <

Agregatsiooni identifikaator	PostgreSQL	Neo4j
	20274) AND date_created between to_timestamp('2019-10-26', 'YYYY-MM-DD') and to_timestamp('2019-11-02', 'YYYY-MM-DD') GROUP BY user_id;	datetime({epochmillis:apoc.date.parse('2019-11-02 00:00:00')}) AND trx.dateCreated >= datetime({epochmillis:apoc.date.parse('2019-10-26 00:00:00')}) RETURN user.id, sum(trx.sum) as last_7_d_sum;

AGR12	SELECT
	  trx.user_id,
	  count(distinct trx2.user_id) as total_user_count_via_recipients_and_ip
	FROM transaction trx
	JOIN transaction trx2 ON (trx2.recipient_id = trx.recipient_id AND trx2.user_id != trx.user_id)
	JOIN (
	   -- läbi ip seotud kasutajad
	   SELECT
	       uipa.user_id as source_user_id,
	       uipa2.user_id as connected_user_id
	   FROM user_ip_address uipa
	 JOIN user_ip_address uipa2 ON (uipa2.ip_address_id = uipa.ip_address_id AND uipa2.user_id != uipa.user_id)
	   WHERE uipa.user_id IN (53429, 77649, 98627, 2486, 86675, 20274)
	   GROUP BY uipa.user_id, uipa2.user_id
	
	) as user_relations_via_ip ON (user_relations_via_ip.source_user_id = trx.user_id)
	WHERE trx.user_id IN
		MATCH (user:User)-[r:SUBMITTED]-(trx)-[po:PAYED_OUT]-(rec:Recipient)<-[po2:PAYED_OUT]-(trx2)-[r2:SUBMITTED]-(u2:User)-[ipr:USED]->(ip:IpAddress)<-[ipr2:USED]-(user:User)
		WHERE user.id IN ['53429', '77649', '98627', '2486', '86675', '20274']
		RETURN user.id, count(distinct u2) as users_count;

Agregatsiooni identifikaator	PostgreSQL	Neo4j
	(53429, 77649, 98627, 2486, 86675, 20274) AND user_relations_via_ip.connected_user_id = trx2.user_id GROUP BY trx.user_id;	

Agregatsioonipäringutes kasutati sisendina eraldi väljavalitud 6 kasutaja ID atribuuti. Valitud sisendkasutajad varieerusid nendega seotud maksete ning IP-aadresside hulga poolest – näiteks oli mõne kasutajaga seotud vaid 3 IP-aadressi ning teise kasutajaga 500 IP-aadressi (vt. Lisa 26).

4.5 Eksperimendi tulemused

Järgnevalt tuuakse välja kõikide agregatsioonipäringute kiirused ning päringute tulemused mõlemas andmebaasisüsteemis. Kiiruse mõõtmiseks kasutati PostgreSQL-i puhul *EXPLAIN ANALYZE* käsku, mis tagastab päringu sooritamiseks kulunud aega andmebaasiserveris. Käesolevas eksperimendis ei mõõdeta kliendipoolset päringu vastuse saamise kiirust, kuna see võib olla oluliselt mõjutatud konkreetsest andmebaasikliendi liidestusest ning samuti kliendi ning serveri vahelisest võrgukiirusest. Neo4j puhul kasutati Neo4j baasinstallatsioonis sisalduvat käsurea tööriista *cypher-shell*. Tabel 2 esitab kõikide eeldefineeritud agregatsioonipäringute kiiruste tulemused.

Tabel 2. Agregatsioonipäringute kiiruste tulemused PostgreSQL ja Neo4j andmebaasisüsteemis.

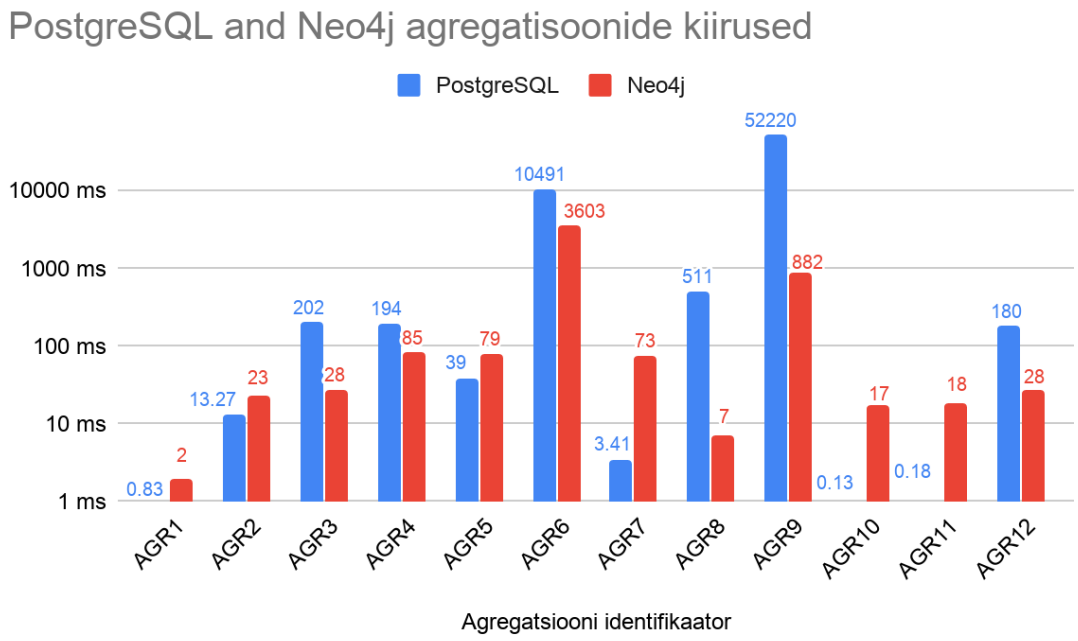
Agregatsiooni identifikaator	PostgreSQL	Neo4j
AGR1	0.83 ms	2 ms
AGR2	13.27 ms	23 ms
AGR3	202 ms	28 ms
AGR4	194 ms	85 ms
AGR5	39 ms	79 ms
AGR6	10 s 491 ms	3 s 603 ms
AGR7	3.41 ms	73 ms

Agregatsiooni identifikaator	PostgreSQL	Neo4j
AGR8	511 ms	7 ms
AGR9	52 s 220 ms	882 ms
AGR10	0.13 ms	17 ms
AGR11	0.18 ms	18 ms
AGR12	180 ms	28 ms

Lisa 26 illustreerib mõlema andmebaasisüsteemi tagastatud tulemused iga eeldefineeritud agregatsiooni kohta.

5 Tulemuste analüüs ja järeldused

Eksperiment koosnes 12 erinevas andmeagregatsioonist. Joonis 22 illustreerib tulemusi võrreldes Neo4j ja PostgreSQL päringute kiiruseid agregatsioonide lõikes.



Joonis 22. PostgreSQL ja Neo4j agregatsioonide kiirused.

Eksperimendi tulemused näitavad, et 6 agregatsiooni puhul 12-st oli kiirem PostgreSQL. Olenevalt konkreetsest vaatlusest võis kahe süsteemi kiiruse vahe olla 2 - 100 kordne. Absoluutväärtustel oli AGR1 puhul on kiiruste vahe kõige minimaalsem – ligikaudu 1 millisekund. Suurim kiiruste vahe oli AGR9 puhul, kus Neo4j oli PostgresSQL-ist ligi 70 korda kiirem.

Järgnevalt vaadeldakse agregatsioonide käivitsuplaane, mille põhjal võib selgitada mõlema andmebaasisüsteemi kiirust konkreetsel agregatsioonil.

AGR1 puhul oli mõlemas süsteemis tegu kiire päringuga. Lisa 2 ja Lisa 3 illustreerib mõlema süsteemi käivitusplaane. Lisa 1 viitab, et PostgreSQL puhul oli päringu täitmisel kasutusel indeks *idx_user_ip_address__user_id_ip_address_id_unique*, mille puhul otsustas PostgreSQL kasutada *Bitmap Index Scan* lähenemist optimeerimaks

tulemusridade püüdmise tihedust. Pärast vajalike tabeli ridade püüdmist tulemused sorteeritakse ning grupeeritakse kasutades *user_id* väärtust võtmena. Neo4j käivitusplaan (Lisa 3) näitab, et päringumootor kasutas samuti indeksit (*NodeUniqueIndexScan*) graafis sisendi *User* sõlmede leidmiseks. Järgmisena loetakse mälust iga sõlmega seotud servad, mille tüüp on *USED*. Neo4j puhul hoitakse sõlmega seotud servade viited mälus konkreetse sõlme viitega seotult. Järgnevalt agregeeritakse mälus saadud unikaalsed servade seotud sõlmed.

AGR2 puhul PostgreSQL päringus liidetakse *user_ip_address* iseendaga kasutades liitmiseks *ip_address_id* väärtust. Käivitsuplaanis kasutatakse seetõttu tabelite liitmiseks indeksit *idx_user_ip_address__ip_address_id_user_id*. Neo4j puhul vaadeldakse seotud *IpAddress* sõlmedega seotud servi, mille tüüp on *USED* ning mis ei ole sama serv, mille kaudu algselt sõlm *IpAddress* leiti.

AGR3 puhul tekib esimest korda suurem kiiruste vahe kahe süsteemi võrdluses. Antud juhul kasutab PostgreSQL erinevalt eelmisest päringust tabeli *users* indeksit *idx_users_state* leidmaks kõik kahtlaste isikutega seotud IP-d. Tulem seotakse läbi *user_ip_address* tabeli ridadega, millele vastab etteantud isikute IP aadressid. Sisendina etteantud IP-d leitakse sarnaselt AGR1-le kasutades *user_ip_address* tabeli indeksit *idx_user_ip_address__user_id_ip_address_id_unique*. Neo4j puhul on käivitusplaan identne AGR2-le. Seda on näha ka võrdlemisi sarnases kiirustes - AGR2 puhul oli Neo4j kiirus 23ms ning AGR3 puhul 28ms. Antud juhul saab tõlgendada Neo4j eelist otse sõlmede viitade järgi seotud sõlmede otsimist vastandina PostgreSQL-le, kus andmebaas peab lisatööd tegema tabeli *user_ip_address*-i liitmiseks.

AGR4 ja AGR5 on oma olemuselt edasiarendused AGR3-le. Mõlema puhul tuleb agregatsioonis arvestada seotud kasutaja staatust. Jällegi on näha, et Neo4j puhul on mõlema agregatsiooni käivitusplaan sarnane, mistõttu on ka päringute kiirused sarnased. Neo4j puhul toimub lisaks eelnevale agregatsioonile ka tulemuste filtreerimine sõlme *User* atribuut *state* põhjal. PostgreSQL käsitleb AGR4 võrdväärselt AGR3-le, mis kajastub ka loogiliselt samas kävitusplaanis, mistõttu on ka agregatsioonide kiirused sarnased. AGR5 puhul on PostgreSQL puhul kasutatud *SELECT* klauslis *FILTER WHERE* filtreerimise käsku, mille puhul antud päringus PostgreSQL ei kasuta indeksit.

Antud näite puhul teeb see PostgreSQL päringu võrreldes AGR4-ga ligi kahe korra kiiremaks.

AGR6 puhul on esimest korda tegemist päringuga, mis vaatab kaugemale kui ühe seose. Seetõttu on ka päringud keerukamad, eeskätt PostgreSQL puhul, mille puhul *user_ip_address-i* liidetakse iseendaga 3 korda. Sellest tulenevalt on ka PostgreSQL käivitusplaan oluliselt keerukam. PostgreSQL puhul AGR6 käivitusplaan näitab, et andmebaasimootor otsib esmalt *idx_users_state* abil kõik IP aadressid, millega on seotud kahtlased isikud. Tulemus liidetakse läbi *user_ip_address* tabeli kõigi sisendparameetritega seotud IP-aadressidega, otsides eelnevalt need kasutades indeksit *idx_user_ip_address__user_id_ip_address_id_unique.* AGR6 puhul ei erine käivitusplaan oluliselt eelnevatest agregatsioonidest. Antud juhul lisandub n-ö teise ringi sõlmedes liikumine (*Expand All*). AGR6 on Neo4j puhul antud eksperimendi oluliselt aeglaseim agregatsioon edestates siiski PostgreSQL ligi 3 korda. Neo4j puhul võib aegluse põhjuseks pidada asjaolu, et testandmestikus leidub mõningaid ülimalt tihedaid sõlmi (ingl *supernodes*), mis seovad omavahel suure osa graafist. Neo4j puhul tähendab see antud päringu ning sisendite puhul sisuliselt kogu graafi *User* ja *IpAddress* sõlmede läbi käimist. Agregatsiooni sisendiks on 6 kasutaja ID atribuudi väärtus. AGR1 päringu vastusest on näha, et 3 kasutaja puhul on juba iseendaga seotud ligi 100 või rohkem erinevat IP aadressi, mis hõlpsasti võivad antud sõlmed graafis tihedalt siduda teiste *User* ja *IpAddress* sõlmedega. Samas kui vähendada sisendparameetreid 3 kõige vähima IP aadressiga kasutajale (53429, 77649, 98627), siis Neo4j puhul sooritab andmebaas päringu ligi 10 korda kiiremini (291 ms). PostgreSQL puhul jääb aga kävitusplaan ka vähendatud sisendite arvuga samaks, mistõttu ka päringu kiirus PostgreSQL puhul ei muutu.

AGR7 puhul on PostgreSQL puhul kasutuses tabelid *transaction* ja *recipient.* PostgreSQL puhul otsitakse üles kasutajaga seotud maksed kasutades indeksit *idx_transaction_user_id_recipient_id.* Leitud kirjetega seotakse uuesti tabel *transaction* läbi *recipient_id* väärtuse kasutades seekord indeksit *idx_transaction_recipient_ip_user_id.* Neo4j puhul leitakse sarnaselt eelnevatele agregatsioonidele *NodeUniqueIndexScan* indeksi abil graafis sisend *User* sõlmed. Järgnevalt liigutakse graafis kasutades sisendkasutajatega seotud servi tüübiga *SUBMITTED* kasutajaga seotud maksed ning sealt edasi makse saajateni kasutades serva

tüüpi *PAYED_OUT*. Leitud makse saajate sõlmedest kasutab Neo4j eelnenud päringule sarnast mustrit, liikudes edasi järgmiste maksete ning makse sooritanud kasutajateni.

AGR8 puhul kasutab PostgreSQL AGR7-ga sarnast käivitusplaani. Lisaks kasutab PostgreSQL seotud makse saajate puhul indeksit esmalt indeksit *idx_users_state*. Neo4j puhul toimub kahtlaste isikute filtreerimine alles lõppsõlmede juurde jõudmisel ning indeksit ei kasutata.

AGR9 on antud eksperimendis suurim tulemuste vahe kahe andmebaasisüsteemi vahel. PostgreSQL puhul on käivitusplaanis kasutusel esmalt indeks *idx_users_state*. Järgnevalt seotakse läbi *transactions* tabeli 2 korda lisaks sama tabel kasutades esmalt indeksit idx_transaction_recipient_id_user_id ning teisekordsel tabelit liitmisel indeksit *idx_transaction_user_id_recipient_id*. Seejärel otsib PostgreSQL üles tabeli *transaction* need kirjed, mis on seotud sisendparameetrite kasutajatega kasutades indeksit *idx_transaction_user_id_recipient_id* ning seejärel seotakse eelnevalt saadud tulem leitud kasutajate maksetega. Neo4j kasutab AGR9 puhul kahe alampäringu liitmist kasutades *NodeHashJoin*-i. Esimeses alampäringus on indeks kasutuses graafi sisendsõlmede leimiseks *User* sõlmedes. Edasi liigub Neo4j mööda vastavaid *Transaction, User* ja *Recipient* sõlmi kuni teise astme seotud kasutajateni. Teine alampäring tehakse kasutades *NodeIndexSeek* indeksi otsingut sõlmetüübi *User* atribuudil *state,* mille abil leitakse kõik kahtlaste isikutega seotud maksed. Kahe alampäringu liitmisel tulemused agregeeritakse ning saadakse otsitud agregatsioon iga kasutajaga kohta.

AGR10 ja AGR11 on iseloomult sarnased - mõlema puhul on nii käivitusplaanid kui ka kiirused sarnased. PostgreSQL kasutab indeksit *idx_transaction_user_id_recipient_id* leidmakse sisendkasutajatega seotud maksed ning filtreerib need etteantud kuupäeva vahemike lõikes iga saadud rea puhul. Neo4j puhul on loogika sarnane.
AGR12 puhul vaadeldi, milline on andmebaaside kiirus agregeerides kahte erinevad relatsioonide tüüpi koos. PostgreSQL puhul leitakse sarnaselt AGR2-le esmalt kasutaja seotud isikud läbi IP aadresside kasutades *user_ip_address* tabelit ning indeksit *idx_user_ip_address_user_id_ip_address_id*. Saadud tulemus sorteeritakse. Teine alampäring leiab kasutajaga seotud maksed kasutades indeksit *idx_transaction_user_id_recipient_id*. Saadud tulemusele liidetakse veelkord tabel

transactions kasutades uuesti *idx_user_ip_address_user_id_ip_address_id* indeksit viidated indeksis nii *user_id* kui ka *recipient_id* väärtusele. Neo4j puhul leitakse eraldi andmestikud kasutajaga seotud IP-aadresside kasutajatest ning kasutajaga seotud maksete saajatest. Mõlema alampäringu tulemused liidetakse ning agregeeritakse.

5.1 Eksperimendi tulemuste kokkuvõte

Eksperimendi tulemused näitavad, et mõlemal andmebaasisüsteemil on omad eeliseid konkreetsete agregatsioonide jooksutamisel. Küll tasub märkida, et agregatsioonides, kus PostgresSQL oli kiirem, oli mõlemas andmebaasisüsteemis kiirused alla 100 ms. Samas aga agregatsioonides, kus Neo4j oli kiirem, võis PostgreSQL kiirus ulatuda mitmekümne sekundini. **Tulemustes selgus, et Neo4j edestas PostgreSQL-i just nende agregatsioonide puhul, mis eeldasid andmestikus läbi mitme seose astme liikumist.** Eriti selgelt on seda näha AGR9 puhul, kus agregatsiooni kiiruste vahe oli ligi 70 kordne Neo4j kasuks ning mille puhul oli tarvis vaadelda andmestikus teise astme seoseid kasutajate ning makse saajate vahel. Samas kui vaadelda agregatsioone, mis vaatlevad vaid ühe serva läbimist, siis kahe süsteemi vahel erilist erinevust ei ole. Seetõttu tasuks enne kummagi süsteemi valimist välja selgitada, millised on need konkreetsed päringud, mida andmebaas peab tagastama.

6 Kokkuvõte

Tehnoloogia kiire arengu tõttu on tänapäeval internetis ostlemine ning raha ülekandmine iseenesest mõistetav. Seetõttu ootab kaasaegne klient kiiret ning mugavat finantsteenust. Reaalajas toimiv ning kasutajasõbralik finantsteenus on lisaks tavakliendile ahvatlevaks sihtmärgiks ka pahatahtlike kavatsustega osapooltele, kes üritavad moodsate süsteemide nõrkuseid ära kasutada nii rahapesuks kui ka finantspettusteks. Tihe konkurents, üha karmimaks muutuvad regulatsioonid ning suurenev pettuste arv sunnib finantsinstitutsioone leidma uusi ning efektiivseimaid lahendusi, kuidas ennetada ning tuvastada võimalikke petturlikke tehinguid.

Käesoleva töö peamiseks eesmärgiks oli võrrelda SQL- ning graafandmebaasi suutlikkust finantspettuste tõkestamiseks vajalike andmepäringute sooritamisel. Lisaks oli töö eesmärgiks koostatud võrdluse põhjal selgitada välja, millised on mõlema andmebaasi tugevused ja nõrkused konkreetses probleemivaldkonnas.

Töö käigus loodi eksperiment pettuste tuvastamise andmeagregatsioonide võrdlemiseks kahe andmebaasisüsteemi vahel. Eksperimendi käigus loodi andmestik, mis jäljendab lihtsustatud kujul finantstehingu süsteemi. Järgnevalt defineeriti konkreetsed andmeagregatsioonid, mida antud andmestiku peal mõlemad andmebaasisüsteemid pidid täitma.

Käesoleva töö peamiseks tulemuseks on, et mõlemal andmebaasil on eelis olenevalt sellest, milline on agregatsiooni sisu ning kasutatav andmestik. SQL andmebaasisüsteemina kasutatud PostgreSQL oli kiirem päringutes, mis ei vajanud andmestikus läbi mitme seose astme liikumist. Samas agregatsioonid, kus päringu käigus läbitud andmepunktide seoste arv kasvas eksponentsiaalselt, oli selgelt kiirem graafandmebaase esindanud Neo4j. Agregatsioonid, mille sooritamine eeldas andmestikus läbis mitme seose astme liikumist oli PostgreSQL puhul aegluse põhjuseks vajadus liita päringutes tabeleid mitmeid kordi, et jõuda vajalike defineeritud seosteni, mis tähendas andmebaasile hüppelist kasvu vajalike tabelite ridade liitmiseks tehtavate operatsioonide täitmiseks. Samas siiski poolte agregatsioonide puhul oli kiirem PostgreSQL, mistõttu on mõistlik järeldada, et andmebaasisüsteemi valikul tuleks

eelkõige aru saada millistele päringutele konkreetses andmestikus on vaja andmebaasil vastus anda ning millised on andmestiku läbimiseks võimalikud meetodid.

Käesolevas töös ei käsitletud andmete lisamise, muutmise, kustutamise operatsioone. Samuti ei uuritud, milline on andmebaaside jõudlus käsitledes graafis lühima tee leidmist või võrdlust teiste mitte relatsiooniliste andmebaasidega näiteks dokumendipõhine andmebaasisüsteem.

Kasutatud kirjandus

[1] “What is Open Banking?,” [Võrgumaterjal] https://www.openbanking.org.uk/customers/what-is-open-banking. (09.10.2019).

[2] Financial Fraud Action UK, “Fraud the Facts 2017,“ [Võrgumaterjal] https://www.financialfraudaction.org.uk/fraudfacts17. (11.10.2019).

[3] European Commission, “Payment services (PSD 2) - Directive (EU) 2015/2366,” [Võrgumaterjal] https://ec.europa.eu/info/law/payment-services-psd-2-directive-eu-2015-2366_en. (20.11.2019).

[4] “DB-Engines Ranking,“ [Võrgumaterjal] https://db-engines.com/en/ranking. (03.10.2019).

[5] Anderson, R. The Credit Scoring Toolkit: theory and practice for retail credit risk management and decision automation. New York : Oxford University Press, 2007.

[6] G. Sadowksi, P. Rathle. “Fraud Detection: Discovering Connections Using Graph Databases,” [Võrgumaterjal] https://neo4j.com/whitepapers/fraud-detection-graph-databases. (10.10.2019).

[7] V. Van Vlasselaer, “APATE: A Novel Approach for Automated Credit Card Transaction Fraud Detection using Network-Based Extensions,” – *Decision Support Systems,* kd. 75, lk. 38-48, 2015.

[8] Barclays. “Fraud Detection Module Advanced: Checklist,” [Võrgumaterjal] https://www.barclaycard.co.uk/business/files/ePDQ_Fraud_detection_module_checking_FDMC.pdf. (14.12.2019).

[9] PayPal. “Fraud Management Filters Summary,” [Võrgumaterjal] https://developer.paypal.com/docs/classic/fmf/integration-guide/FMFSummary. (16.10.2019).

[10] WorldPay. “Avoid 3 common types of online payment fraud,” [Võrgumaterjal] https://www.worldpay.com/en-us/insights-hub/article/avoid-3-common-types-of-online-payment-fraud. (14.12.2019).

[11] A.C. Bahnsen, D. Aouada, A. Stojanovic, “Feature engineering strategies for credit card fraud detection,” – *Expert Systems With Applications,* kd. 51, lk. 134-142, 2016.

[12] D. Hand, N. Adams, “Transaction aggregation as a strategy for credit card fraud detection,” – *Data Mining and Knowledge Discovery*, kd. 18, lk 30-55, 2009.

[13] I. Molloy, “Graph Analytics for Real-time Scoring of Cross-channel Transactional Fraud,” – *International Conference on Financial Cryptography and Data Security*, pp 22-40, 2016.

[14] I. Rogers, “The Google Pagerank Algorithm and How It Works,” [Võrgumaterjal] https://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm. (21.12.2019).

[15] E. Eessaar. Andmebaaside projekteerimine. Tallinn : Tallinna Tehnikaülikooli kirjastus, 2008.

[16] M.Winand. SQL Performance Explained. Viin : Graphikservice GmbH, 2012.

[17] A. Buldas, P. Laud ja J. Villemson, Graafid. Tartu : Tartu Ülikooli Kirjastus, 2008.

[18] Rik Van Bruggen. Learning Neo4j. London : Packt Publishing, 2014

[19] Max De Marzi, "The Secret Sauce of Neo4j," [Võrgumaterjal] https://neo4j.com/graphconnect-2018/session/secret-sauce-neo4j. (05.10.2019).

[20] C. Vicknair, M. Macias, Z. Zhao, "A comparison of a graph database and a relational database: a data provenance perspective," – *ACM SE '10 Proceedings of the 48th Annual Southeast Regional Conference,* 2010.

[21] R. Angles, A. Prat-Pérez, D. Dominguez-Sal, "Benchmarking database systems for social network applications," – *GRADES '13 First International Workshop on Graph Data Management Experiences and Systems,* pp. 1-7, 2013.

[22] Alexandra Martinez , Rodrigo Mora , Daniel Alvarado, "A Comparison between a Relational Database and a Graph Database in the context of a Personalized Cancer Treatment Application," – *Proceedings of the Alberto Mendelzon International Workshop on Foundations of Data Management*. 2016

[23] A. Gubichev, "Graph Pattern Matching: Do We Have to Reinvent the Wheel?," – *GRADES'14 Proceedings of Workshop on Graph Data management Experiences and Systems*. 2014.

[24] Lehigh University, "LUBM: A Benchmark for OWL Knowledge Base Systems," [Võrgumaterjal] http://swat.cse.lehigh.edu/projects/lubm. (10.12.2019).

[25] A. Pacaci, "Do We Need Specialized Graph Databases? Benchmarking Real-Time Social Networking Applications," – *GRADES'17 Proceedings of the Fifth International Workshop on Graph Data-management Experiences & Systems*. 2017.

[26] Data Benchmark Council, "Social Network Benchmark," [Võrgumaterjal] http://ldbcouncil.org/developer/snb. (10.12.2019).

[27] K. Wecel, "Benchmarking of databases for Big Data Exploration in the Social Graph Analysis," – *Studia Oeconomica Posnaniensia,* kd. 5, lk. 56-72. 2017

[28] M. Paradies, "Challenges in the Design of a Graph Database Benchmark," [Võrgumaterjal] https://www.slideshare.net/graphdevroom/challenges-in-the-design-of-a-graph-database-benchmark. (11.12.2019).

[29] F. Holzschuher, "Performance of Graph Query Languages: comparison of cypher, gremlin and native access in Neo4j," – *Proceedings of the Joint EDBT/ ICDT 2013 Workshops,* pp. 195-204, 2013.

[30] M. Drake, "SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems," [Võrgumaterjal] https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. (19.11.2019).

[31] C. Antognini, "Query Optimizer – MySQL vs. PostgreSQL," [Võrgumaterjal] https://www.percona.com/live/18/sessions/query-optimizer-mysql-vs-postgresql. (19.11.2019).

[32] B. Liu, “Real-Time Fraud Detection at Scale - Integrating Real-Time Deep-Link Graph Analytics with Spark AI,” [Võrgumaterjal] https://databricks.com/session_eu19/real-time-fraud-detection-at-scale-integrating-real-time-deep-link-graph-analytics-with-spark-ai. (10.12.2019).

[33] A. Disney, “Link Analysis for Fraud Detection,” [Võrgumaterjal] https://cambridge-intelligence.com/link-analysis-fraud-detection. (14.12.2019).

Lisa 1 – Tabelite loomise laused PostgreSQLis

```
create table users
(
      id serial not null
            constraint user_pk
                  primary key,
      email text not null,
      state text default 'NEW'::text not null,
      date_created timestamp default now() not null
);

create unique index user_email_uindex
      on users (email);

create index idx_users_state
      on users (state);

create table ip_address
(
      id serial not null
            constraint ip_address_pk
                  primary key,
      ip_address text not null,
      date_created timestamp default now() not null
);

create unique index ip_address_ip_address_uindex
      on ip_address (ip_address);

create table recipient
(
      id serial not null
            constraint recipeint_pk
                  primary key,
      name text not null,
      account_number text not null,
      date_created timestamp default now() not null
);

create table transaction
(
      id serial not null
            constraint transaction_pk
                  primary key,
      user_id bigint not null
            constraint fk_transaction_user_id_user_id
                  references users,
```

```
	recipient_id bigint not null
		constraint fk_transaction_recipient_id_recipient_id
			references recipient,
	sum numeric(14,2) not null,
	date_created timestamp default now() not null
);

create index idx_transaction_recipient_id_user_id
	on transaction (recipient_id, user_id);

create index idx_transaction_user_id_recipient_id
	on transaction (user_id, recipient_id);

create table user_ip_address
(
	id serial not null
		constraint user_ip_address_pk
			primary key,
	user_id bigint not null
		constraint fx_user_ip_address_user_id_user_id
			references users,
	ip_address_id bigint not null
		constraint fk_user_ip_address_ip_address_id_ip_address_id
			references ip_address
);

create unique index idx_user_ip_address__user_id_ip_address_id_unique
	on user_ip_address (user_id, ip_address_id);

create index idx_user_ip_address__ip_address_id_user_id
	on user_ip_address (ip_address_id, user_id);
```

Lisa 2 - AGR1 PostgreSQL käivitusplaan

```
HashAggregate  (cost=1050.39..1053.29 rows=290 width=16)
  Group Key: user_id
  ->  Bitmap Heap Scan on user_ip_address  (cost=28.80..1048.93 rows=291 width=16)
"        Recheck Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
        ->  Bitmap Index Scan on idx_user_ip_address__user_id_ip_address_id_unique  (cost=0.00..28.73 rows=291 width=0)
"              Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
```

Lisa 3 - AGR1 Neo4j käivitusplaan

```
+----------------------+----------------+------+---------+-----------+-----------+-----------------------------+-------------------------------+
| Operator             | Estimated Rows | Rows | DB Hits | Cache H/M | Time (ms) | Identifiers                 | Other                         |
+----------------------+----------------+------+---------+-----------+-----------+-----------------------------+-------------------------------+
| +ProduceResults      |             21 |    6 |       0 |    2844/0 |     0.017 | count(DISTINCT ips), user.id | 17386                         |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------+-------------------------------+
| +EagerAggregation    |             21 |    6 |       0 |  204552/0 |     0.779 | count(DISTINCT ips), user.id | user.id; 779471               |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------+-------------------------------+
| +Expand(All)         |            450 |  822 |     828 |    1444/0 |     1.217 | cached[user.id], ips, r, user | (user)-[r:USED]->(ips); 1216728 |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------+-------------------------------+
| +NodeUniqueIndexSeek |             25 |   12 |      18 |    2888/0 |     2.915 | cached[user.id], user        | 2915060; :User(id)            |
+----------------------+----------------+------+---------+-----------+-----------+-----------------------------+-------------------------------+
```

Lisa 4 - AGR2 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=0.85..11951.50 rows=290 width=16)
  Group Key: uipa.user_id
  ->  Nested Loop  (cost=0.85..11928.05 rows=2741 width=16)
        ->  Index Only Scan using idx_user_ip_address__user_id_ip_address_id_unique on user_ip_address uipa  (cost=0.43..1183.63 rows=291 width=16)
"              Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
        ->  Index Only Scan using idx_user_ip_address__ip_address_id_user_id on user_ip_address uipa2  (cost=0.43..36.83 rows=9 width=16)
              Index Cond: (ip_address_id = uipa.ip_address_id)
```

Lisa 5 - AGR2 Neo4j käivitusplaan

```
+----------------------+----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| Operator             | Estimated Rows | Rows | DB Hits | Cache H/M   | Time (ms) | Identifiers                 | Other                       |
+----------------------+----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +ProduceResults      |             64 |    6 |       0 |   107472/0 |     0.069 | ip_total_other_users, u.id  | 68962                       |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +EagerAggregation    |             64 |    6 |       0 | 67318622/0 |     7.513 | ip_total_other_users, u.id  | u.id; 7512948               |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +Filter              |           4046 | 1583 |    8330 | 74696117/0 |    14.398 | ip, u, cached[u.id], r1, r2, u2 | not r1 = r2; 14398245       |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +Expand(All)         |           4046 |  833 |    9152 |  7398201/0 |    21.541 | ip, u, cached[u.id], r1, r2, u2 | (ip)<-[r2:USED]-(u2); 21541009 |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +Filter              |            450 |   82 |     822 |  7398201/0 |    22.335 | cached[u.id], ip, r1, u     | ip:IpAddress; 22334562      |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +Expand(All)         |            450 |   82 |     828 |    55195/0 |    23.035 | cached[u.id], ip, r1, u     | (u)-[r1:USED]->(ip); 23035206  |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +NodeUniqueIndexSeek |             25 |   12 |      18 |   110390/0 |    47.284 | cached[u.id], u             | 47284106; :User(id)         |
+----------------------+----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
```

Lisa 6 - AGR3 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=213.01..213.04 rows=1 width=16)
  Group Key: uipa.user_id
  ->  Sort  (cost=213.01..213.02 rows=1 width=16)
        Sort Key: uipa.user_id
        ->  Nested Loop  (cost=5.54..213.00 rows=1 width=16)
              ->  Nested Loop  (cost=5.11..200.34 rows=18 width=16)
                    ->  Index Scan using idx_users_state on users u  (cost=0.29..4.31 rows=1 width=4)
                          Index Cond: (state = 'BAD'::text)
                    ->  Bitmap Heap Scan on user_ip_address uipa2  (cost=4.82..195.53 rows=50 width=16)
                          Recheck Cond: (user_id = u.id)
                          ->  Bitmap Index Scan on idx_user_ip_address__user_id_ip_address_id_unique  (cost=0.00..4.80 rows=50 width=0)
                                Index Cond: (user_id = u.id)
              ->  Index Only Scan using idx_user_ip_address__ip_address_id_user_id on user_ip_address uipa  (cost=0.43..0.69 rows=1 width=16)
                    Index Cond: (ip_address_id = uipa2.ip_address_id)
"                   Filter: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
```

Lisa 7 - AGR3 Neo4j käivitusplaan

```
+----------------------+----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| Operator             | Estimated Rows | Rows | DB Hits | Cache H/M   | Time (ms) | Identifiers                 | Other                       |
+----------------------+----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +ProduceResults      |             45 |    6 |       0 |   157578/0 |     0.018 | ip_total_other_users_bad, u.id | 17580                       |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +EagerAggregation    |             45 |    6 |       0 |  1114507/0 |     0.212 | ip_total_other_users_bad, u.id | u.id; 212005                |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +Filter              |           2023 | 8508 |   16660 | 109483789/0 |    13.593 | ip, u, cached[u.id], r1, r2, u2 | not r1 = r2; 13592700       |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +Expand(All)         |           4046 | 8330 |    9152 | 10844028/0 |    20.843 | ip, u, cached[u.id], r1, r2, u2 | (ip)<-[r2:USED]-(u2); 20842603 |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +Filter              |            450 |  822 |     822 | 10844028/0 |    21.740 | cached[u.id], ip, r1, u     | ip:IpAddress; 21739773      |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +Expand(All)         |            450 |  822 |     828 |    80936/0 |    22.234 | cached[u.id], ip, r1, u     | (u)-[r1:USED]->(ip); 22233578  |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
| +NodeUniqueIndexSeek |             25 |   12 |      18 |   161872/0 |    45.279 | cached[u.id], u             | 45278923; :User(id)         |
+----------------------+----------------+------+---------+-------------+-----------+-----------------------------+-----------------------------+
```

Lisa 8 - AGR4 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=213.01..213.03 rows=1 width=16)
  Group Key: uipa.user_id
  ->  Sort  (cost=213.01..213.02 rows=1 width=16)
        Sort Key: uipa.user_id
        ->  Nested Loop  (cost=5.54..213.00 rows=1 width=16)
              ->  Nested Loop  (cost=5.11..200.34 rows=18 width=8)
                    ->  Index Scan using idx_users_state on users u  (cost=0.29..4.31 rows=1 width=4)
                          Index Cond: (state = 'BAD'::text)
                    ->  Bitmap Heap Scan on user_ip_address uipa2  (cost=4.82..195.53 rows=50 width=16)
                          Recheck Cond: (user_id = u.id)
                          ->  Bitmap Index Scan on idx_user_ip_address__user_id_ip_address_id_unique  (cost=0.00..4.80 rows=50 width=0)
                                Index Cond: (user_id = u.id)
              ->  Index Only Scan using idx_user_ip_address__ip_address_id_user_id on user_ip_address uipa  (cost=0.43..0.69 rows=1 width=16)
                    Index Cond: (ip_address_id = uipa2.ip_address_id)
"                    Filter: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
```

Lisa 9 - AGR4 Neo4j käivitusplaan

```
+----------------------+----------------+------+---------+-------------+-----------+-----------------------------------------+-----------------------------------+
| Operator             | Estimated Rows | Rows | DB Hits | Cache H/M   | Time (ms) | Identifiers                             | Other                             |
+----------------------+----------------+------+---------+-------------+-----------+-----------------------------------------+-----------------------------------+
| +ProduceResults      |              3 |    6 |       0 |   153720/0 |     0.022 | total_suspicious_ip_count, userid       | 22220                             |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------------------+-----------------------------------+
| +EagerAggregation    |              3 |    6 |       0 |  2228940/0 |     0.104 | total_suspicious_ip_count, userid       | userid; 104364                    |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------------------+-----------------------------------+
| +EagerAggregation    |             12 |   86 |      89 |  1088039/0 |     0.463 | bad_count, ipid, userid                 | userid, ipid; 462532              |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------------------+-----------------------------------+
| +Filter              |            136 | 1592 |   15838 | 106872698/0 |    13.924 | anon[34], cached[user.id], u3, ips, r, user | u3.state = $`  AUTOSTRING1`; 13924431 |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------------------+-----------------------------------+
| +Expand(All)         |            271 | 8330 |    9152 |  10585103/0 |    21.157 | anon[34], cached[user.id], u3, ips, r, user | (ips)<-[anon[34]:USED]-(u3); 21156554 |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------------------+-----------------------------------+
| +Expand(All)         |            450 |  822 |     828 |     78983/0 |    21.692 | cached[user.id], ips, r, user           | (user)-[r:USED]->(ips); 21692112  |
| |                    +----------------+------+---------+-------------+-----------+-----------------------------------------+-----------------------------------+
| +NodeUniqueIndexSeek |             25 |   12 |      18 |   157966/0 |    44.167 | cached[user.id], user                   | 44166941; :User(id)               |
+----------------------+----------------+------+---------+-------------+-----------+-----------------------------------------+-----------------------------------+
```

Lisa 10 - AGR5 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=1.15..12821.88 rows=200 width=16)
  Group Key: uipa.user_id
  ->  GroupAggregate  (cost=1.15..12815.51 rows=291 width=24)
"        Group Key: uipa.user_id, uipa.ip_address_id"
        ->  Nested Loop  (cost=1.15..12785.19 rows=2741 width=24)
              ->  Nested Loop  (cost=0.85..11928.05 rows=2741 width=24)
                    ->  Index Only Scan using idx_user_ip_address__user_id_ip_address_id_unique on user_ip_address uipa  (cost=0.43..1183.63 rows=291
width=16)
"                          Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
                    ->  Index Only Scan using idx_user_ip_address__ip_address_id_user_id on user_ip_address uipa2  (cost=0.43..36.83 rows=9 width=16)
                          Index Cond: (ip_address_id = uipa.ip_address_id)
              ->  Index Scan using user_pk on users u  (cost=0.29..0.31 rows=1 width=8)
                    Index Cond: (id = uipa2.user_id)
```

Lisa 11 - AGR5 Neo4j käivitusplaan

```
+----------------------+----------------+------+---------+-----------+-----------------------------------------+-----------------------------+
| Operator             | Estimated Rows | Rows | DB Hits | Cache H/M | Identifiers                             | Other                       |
+----------------------+----------------+------+---------+-----------+-----------------------------------------+-----------------------------+
| +ProduceResults      |              3 |    6 |       0 |       0/0 | max_fraud_count_on_ip, user_id          |                             |
| |                    +----------------+------+---------+-----------+-----------------------------------------+-----------------------------+
| +EagerAggregation    |              3 |    6 |       0 |       0/0 | max_fraud_count_on_ip, user_id          | user_id                     |
| |                    +----------------+------+---------+-----------+-----------------------------------------+-----------------------------+
| +EagerAggregation    |             12 |   86 |       0 |   26258/0 | fraud_count, ips, user_id               | user_id, ips                |
| |                    +----------------+------+---------+-----------+-----------------------------------------+-----------------------------+
| +Filter              |            136 |   89 |   16660 |   26258/0 | anon[34], cached[user.id], ips, r, u2, user | not `anon[34]` = r          |
| |                    +----------------+------+---------+-----------+-----------------------------------------+-----------------------------+
| +Expand(All)         |            271 | 8330 |    9152 |   26258/0 | anon[34], cached[user.id], ips, r, u2, user | (ips)<-[anon[34]:USED]-(u2) |
| |                    +----------------+------+---------+-----------+-----------------------------------------+-----------------------------+
| +Expand(All)         |            450 |  822 |     828 |   26258/0 | cached[user.id], ips, r, user           | (user)-[r:USED]->(ips)      |
| |                    +----------------+------+---------+-----------+-----------------------------------------+-----------------------------+
| +NodeUniqueIndexSeek |             25 |    6 |      12 |   26258/0 | cached[user.id], user                   | :User(id)                   |
+----------------------+----------------+------+---------+-----------+-----------------------------------------+-----------------------------+
```

Lisa 12 - AGR6 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=1748.08..1748.36 rows=16 width=16)
  Group Key: uipa.user_id
  ->  Sort  (cost=1748.08..1748.12 rows=16 width=16)
        Sort Key: uipa.user_id
        ->  Hash Join  (cost=726.33..1747.76 rows=16 width=16)
              Hash Cond: (uipa.ip_address_id = uipa2.ip_address_id)
              Join Filter: (uipa2.user_id <> uipa.user_id)
              ->  Bitmap Heap Scan on user_ip_address uipa  (cost=28.80..1048.93 rows=291 width=16)
"                     Recheck Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
                    ->  Bitmap Index Scan on idx_user_ip_address__user_id_ip_address_id_unique  (cost=0.00..28.73 rows=291 width=0)
"                          Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
              ->  Hash  (cost=565.32..565.32 rows=10577 width=24)
                    ->  Nested Loop  (cost=5.96..565.32 rows=10577 width=24)
                          Join Filter: (uipa4.user_id <> uipa2.user_id)
                          ->  Nested Loop  (cost=5.54..213.63 rows=168 width=24)
                                ->  Nested Loop  (cost=5.11..200.34 rows=18 width=16)
                                      ->  Index Scan using idx_users_state on users u  (cost=0.29..4.31 rows=1 width=4)
                                            Index Cond: (state = 'BAD'::text)
                                      ->  Bitmap Heap Scan on user_ip_address uipa4  (cost=4.82..195.53 rows=50 width=16)
                                            Recheck Cond: (user_id = u.id)
                                            ->  Bitmap Index Scan on idx_user_ip_address__user_id_ip_address_id_unique  (cost=0.00..4.80 rows=50
width=0)
                                                  Index Cond: (user_id = u.id)
                                ->  Index Only Scan using idx_user_ip_address__ip_address_id_user_id on user_ip_address uipa3  (cost=0.43..0.65 rows=9
width=16)
                                      Index Cond: (ip_address_id = uipa4.ip_address_id)
                                      Filter: (uipa4.user_id <> user_id)
                          ->  Index Only Scan using idx_user_ip_address__user_id_ip_address_id_unique on user_ip_address uipa2  (cost=0.43..1.47
rows=50 width=16)
                                Index Cond: (user_id = uipa3.user_id)
                                Filter: (uipa3.ip_address_id <> ip_address_id)
```

Lisa 13 - AGR6 Neo4j käivitusplaan

Operator	Estimated Rows	Rows	DB Hits	Cache H/M	Time (ms)	Identifiers	Other
+ProduceResults	38	6	0	81584292/0	0.096	bad_count, user.id	95973
+EagerAggregation	38	6	0	271697265099/0	8.653	bad_count, user.id	user.id; 8653150
+Filter	1472	8634591	8514654	30463447546480/0	2657.687	anon[34], cached[user.id], ips2, anon[71], u3, ips, r2, r, u2, user	u3:User; 2657686865
+Expand(All)	2945	4481648	4930290	3050817770476/0	4160.906	anon[34], cached[user.id], ips2, anon[71], u3, ips, r2, r, u2, user	(ips2)<-[anon[71]:USED]-(u3); 4160905917
+Filter	4881	897284	0	3101220057026/0	4209.016	anon[34], cached[user.id], ips2, ips, r2, r, u2, user	not r = r2; 4209015678
+Expand(All)	4881	456150	463658	50409142003/0	4307.945	anon[34], cached[user.id], ips2, ips, r2, r, u2, user	(u2)-[r2:USED]->(ips2); 4307945153
+Filter	271	15838	8330	55949399901/0	4310.090	anon[34], cached[user.id], ips, r, u2, user	not `anon[34]` = r; 4310089724
+Expand(All)	271	8330	9152	5544854630/0	4313.343	anon[34], cached[user.id], ips, r, u2, user	(ips)<-[anon[34]:USED]-(u2); 4313342677
+Expand(All)	450	822	828	41676433/0	4313.923	cached[user.id], ips, r, user	(user)-[r:USED]->(ips); 4313923446
+NodeUniqueIndexSeek	25	12	18	83352866/0	8629.087	cached[user.id], user	8629087111; :User(id)

Lisa 14 - AGR7 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=0.86..22466.62 rows=508 width=16)
  Group Key: trx.user_id
  ->  Nested Loop  (cost=0.86..22427.73 rows=6761 width=16)
        ->  Index Only Scan using idx_transaction_user_id_recipient_id on transaction trx  (cost=0.43..1655.51 rows=513 width=16)
"             Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
        ->  Index Only Scan using idx_transaction_recipient_id_user_id on transaction trx2  (cost=0.43..40.36 rows=13 width=16)
              Index Cond: (recipient_id = trx.recipient_id)
              Filter: (user_id <> trx.user_id)
```

Lisa 15 - AGR7 Neo4j käivitusplaan

Operator	Estimated Rows	Rows	DB Hits	Cache H/M	Time (ms)	Identifiers	Other
+ProduceResults	73	5	0	15420/0	0.015	user.id, users_count	15053
+EagerAggregation	73	5	0	1091337/0	0.725	user.id, users_count	user.id; 724723
+Filter	5336	1410	705	1088253/0	1.217	trx, cached[user.id], po, rec, r2, r, u2, po2, user, trx2	not r = r2; 1217399
+Expand(All)	10672	705	1410	1088253/0	2.370	trx, cached[user.id], po, rec, r2, r, u2, po2, user, trx2	(trx2)-[r2:SUBMITTED]-(u2); 2370182
+Filter	6411	705	0	1172326/0	2.550	trx, cached[user.id], po, rec, r, po2, user, trx2	not po = po2; 2549915
+Expand(All)	6411	760	815	86405/0	3.212	trx, cached[user.id], po, rec, r, po2, user, trx2	(rec)<-[po2:PAYED_OUT]-(trx2); 3211890
+Filter	516	55	55	86405/0	3.261	trx, cached[user.id], po, rec, r, user	rec:Recipient; 3261264
+Expand(All)	1033	55	110	86405/0	3.340	trx, cached[user.id], po, rec, r, user	(trx)-[po:PAYED_OUT]-(rec); 3340203
+Expand(All)	621	55	61	13159/0	3.402	cached[user.id], r, trx, user	(user)-[r:SUBMITTED]-(trx); 3402441
+NodeUniqueIndexSeek	25	12	18	26318/0	7.214	cached[user.id], user	7214285; :User(id)

Lisa 16 - AGR8 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=320.21..320.23 rows=1 width=16)
  Group Key: trx.user_id
  ->  Sort  (cost=320.21..320.21 rows=1 width=16)
        Sort Key: trx.user_id
        ->  Nested Loop  (cost=1.15..320.20 rows=1 width=16)
              ->  Nested Loop  (cost=0.72..299.21 rows=25 width=16)
                    ->  Index Scan using idx_users_state on users u  (cost=0.29..4.31 rows=1 width=4)
                          Index Cond: (state = 'BAD'::text)
                    ->  Index Only Scan using idx_transaction_user_id_recipient_id on transaction trx2  (cost=0.43..294.00 rows=90 width=16)
                          Index Cond: (user_id = u.id)
              ->  Index Only Scan using idx_transaction_recipient_id_user_id on transaction trx  (cost=0.43..0.83 rows=1 width=16)
                    Index Cond: (recipient_id = trx2.recipient_id)
"                   Filter: ((trx2.user_id <> user_id) AND (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[])))"
|
```

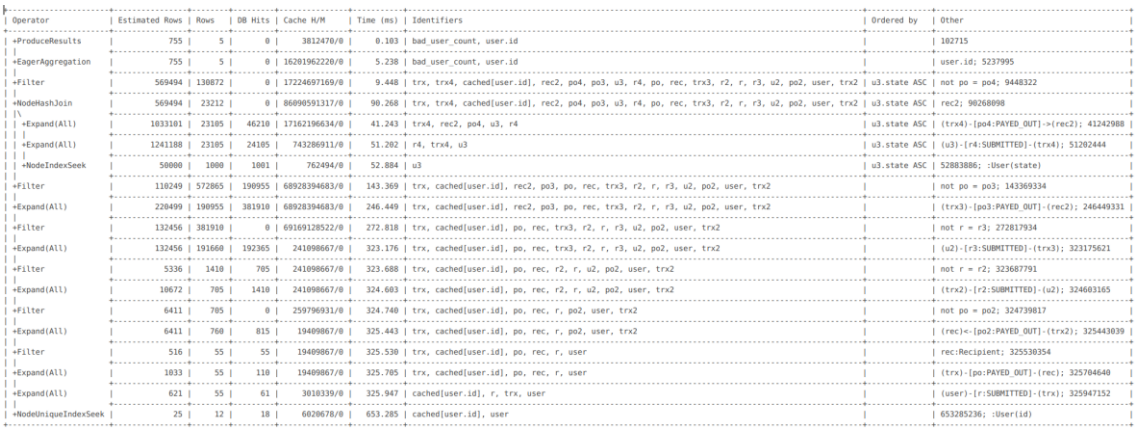
Lisa 17 - AGR8 Neo4j käivitusplaan

```
+----------------------+----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| Operator             | Estimated Rows | Rows | DB Hits | Cache H/M | Time (ms) | Identifiers                                                           | Other                                               |
+----------------------+----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +ProduceResults      |             52 |    4 |       0 |   15180/0 |     0.013 | bad_users_count, user.id                                              | 12698                                               |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +EagerAggregation    |             52 |    4 |       0 |   14825/0 |     0.048 | bad_users_count, user.id                                              | user.id; 47937                                      |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +Filter              |           2668 | 1415 |    1410 | 1339698/0 |     1.303 | trx, cached[user.id], po, rec, r2, r, u2, po2, user, trx2             | u2.state = $`  AUTOSTRING1`; 1302854                |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +Expand(All)         |          10672 |  705 |    1410 | 1339698/0 |     2.468 | trx, cached[user.id], po, rec, r2, r, u2, po2, user, trx2             | (trx2)-[r2:SUBMITTED]-(u2); 2467905                 |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +Filter              |           6411 |  705 |       0 | 1443173/0 |     2.652 | trx, cached[user.id], po, rec, r, po2, user, trx2                     | not po = po2; 2652487                               |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +Expand(All)         |           6411 |  760 |     815 |  106395/0 |     3.331 | trx, cached[user.id], po, rec, r, po2, user, trx2                     | (rec)<-[po2:PAYED_OUT]-(trx2); 3331486              |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +Filter              |            516 |   55 |      55 |  106395/0 |     3.381 | trx, cached[user.id], po, rec, r, user                                | rec:Recipient; 3380949                              |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +Expand(All)         |           1033 |   55 |     110 |  106395/0 |     3.454 | trx, cached[user.id], po, rec, r, user                                | (trx)-[po:PAYED_OUT]-(rec); 3453806                 |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +Expand(All)         |            621 |   55 |      61 |   16205/0 |     3.559 | cached[user.id], r, trx, user                                         | (user)-[r:SUBMITTED]-(trx); 3558549                 |
| |                    +----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
| +NodeUniqueIndexSeek |             25 |   12 |      18 |   32410/0 |     7.550 | cached[user.id], user                                                 | 7549940; :User(id)                                  |
+----------------------+----------------+------+---------+-----------+-----------+-----------------------------------------------------------------------+-----------------------------------------------------+
.
```

Lisa 18 - AGR9 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=3666.71..3669.20 rows=142 width=16)
  Group Key: trx.user_id
  ->  Sort  (cost=3666.71..3667.07 rows=142 width=16)
        Sort Key: trx.user_id
        ->  Hash Join  (cost=2002.85..3661.64 rows=142 width=16)
              Hash Cond: (trx.recipient_id = trx2.recipient_id)
              Join Filter: (trx2.user_id <> trx.user_id)
              ->  Index Only Scan using idx_transaction_user_id_recipient_id on transaction trx  (cost=0.43..1655.51 rows=513 width=16)
"                   Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
              ->  Hash  (cost=1348.50..1348.50 rows=52314 width=24)
                    ->  Nested Loop  (cost=1.58..1348.50 rows=52314 width=24)
                          ->  Nested Loop  (cost=1.15..320.76 rows=324 width=24)
                                ->  Nested Loop  (cost=0.72..299.21 rows=25 width=16)
                                      ->  Index Scan using idx_users_state on users u  (cost=0.29..4.31 rows=1 width=4)
                                            Index Cond: (state = 'BAD'::text)
                                      ->  Index Only Scan using idx_transaction_user_id_recipient_id on transaction trx4  (cost=0.43..294.00 rows=90 width=16)
                                            Index Cond: (user_id = u.id)
                                ->  Index Only Scan using idx_transaction_recipient_id_user_id on transaction trx3  (cost=0.43..0.73 rows=13 width=16)
                                      Index Cond: (recipient_id = trx4.recipient_id)
                                      Filter: (trx4.user_id <> user_id)
                          ->  Index Only Scan using idx_transaction_user_id_recipient_id on transaction trx2  (cost=0.43..2.27 rows=90 width=16)
                                Index Cond: (user_id = trx3.user_id)
                                Filter: (trx3.recipient_id <> recipient_id)
```

Lisa 19 - AGR9 Neo4j käivitusplaan

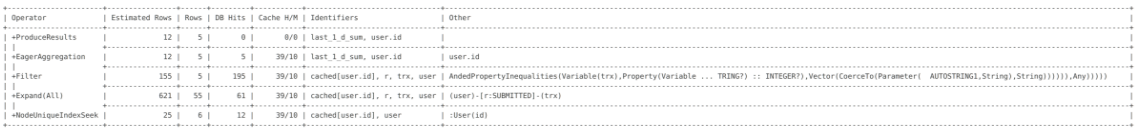


Operator	Estimated Rows	Rows	DB Hits	Cache H/M	Time (ms)	Identifiers	Ordered by	Other
+ProduceResults	755	5	0	3812470/0	0.103	bad_user_count, user.id		102715
+EagerAggregation	755	5	0	16201962220/0	5.238	bad_user_count, user.id		user.id; 5237995
+Filter	569494	130872	0	17224697169/0	9.448	trx, trx4, cached[user.id], rec2, po4, po3, u3, r4, po, rec, trx3, r2, r, r3, u2, po2, user, trx2	u3.state ASC	not po = po4; 9448322
+NodeHashJoin	569494	23212	0	86090591317/0	90.268	trx, trx4, cached[user.id], rec2, po4, po3, u3, r4, po, rec, trx3, r2, r, r3, u2, po2, user, trx2	u3.state ASC	rec2; 90268098
+Expand(All)	1033101	23105	46210	17162196634/0	41.243	trx4, rec2, po4, u3, r4	u3.state ASC	(trx4)-[po4:PAYED_OUT]->(rec2); 41242988
+Expand(All)	1241188	23105	24105	74328691 1/0	51.202	r4, trx4, u3	u3.state ASC	(u3)-[r4:SUBMITTED]-(trx4); 51202444
+NodeIndexSeek	50000	1000	1001	762494/0	52.884	u3	u3.state ASC	52883886; :User(state)
+Filter	110249	572865	190955	68928394683/0	143.369	trx, cached[user.id], rec2, po3, po, rec, trx3, r2, r, r3, u2, po2, user, trx2		not po = po3; 143369334
+Expand(All)	220499	190955	381910	68928394683/0	246.449	trx, cached[user.id], rec2, po3, po, rec, trx3, r2, r, r3, u2, po2, user, trx2		(trx3)-[po3:PAYED_OUT]-(rec2); 246449331
+Filter	132456	381910	0	69169128522/0	272.818	trx, cached[user.id], po, rec, trx3, r2, r, r3, u2, po2, user, trx2		not r = r3; 272817934
+Expand(All)	132456	191660	192365	241098667/0	323.176	trx, cached[user.id], po, rec, trx3, r2, r, r3, u2, po2, user, trx2		(u2)-[r3:SUBMITTED]-(trx3); 323175621
+Filter	5336	1410	705	241098667/0	323.688	trx, cached[user.id], po, rec, r2, r, u2, po2, user, trx2		not r = r2; 323687791
+Expand(All)	10672	705	1410	241098667/0	324.603	trx, cached[user.id], po, rec, r2, r, u2, po2, user, trx2		(trx2)-[r2:SUBMITTED]-(u2); 324603165
+Filter	6411	705	0	259796931/0	324.740	trx, cached[user.id], po, rec, r, po2, user, trx2		not po = po2; 324739817
+Expand(All)	6411	760	815	19409867/0	325.443	trx, cached[user.id], po, rec, r, po2, user, trx2		(rec)<-[po2:PAYED_OUT]-(trx2); 325443039
+Filter	516	55	55	19409867/0	325.530	trx, cached[user.id], po, rec, r, user		rec:Recipient; 325530354
+Expand(All)	1033	55	110	19409867/0	325.705	trx, cached[user.id], po, rec, r, user		(trx)-[po:PAYED_OUT]-(rec); 325704640
+Expand(All)	621	55	61	3010339/0	325.947	cached[user.id], r, trx, user		(user)-[r:SUBMITTED]-(trx); 325947152
+NodeUniqueIndexSeek	25	12	18	6020678/0	653.285	cached[user.id], user		653285236; :User(id)

Lisa 20 - AGR10 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=1834.41..1834.43 rows=1 width=40)
  Group Key: user_id
  ->  Sort  (cost=1834.41..1834.41 rows=1 width=14)
        Sort Key: user_id
        ->  Bitmap Heap Scan on transaction  (cost=30.41..1834.40 rows=1 width=14)
"              Recheck Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
"              Filter: ((date_created >= to_timestamp('2019-11-01'::text, 'YYYY-MM-DD'::text)) AND (date_created <= to_timestamp('2019-11-02'::text, 'YYYY-MM-DD'::text)))"
              ->  Bitmap Index Scan on idx_transaction_user_id_recipient_id  (cost=0.00..30.40 rows=513 width=0)
"                    Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
```

Lisa 21 - AGR10 Neo4j käivitusplaan

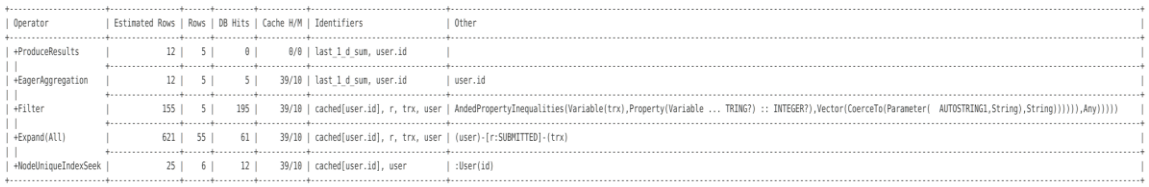


Operator	Estimated Rows	Rows	DB Hits	Cache H/M	Identifiers	Other
+ProduceResults	12	5	0	0/0	last_1_d_sum, user.id	
+EagerAggregation	12	5	5	39/10	last_1_d_sum, user.id	user.id
+Filter	155	5	195	39/10	cached[user.id], r, trx, user	AndedPropertyInequalities(Variable(trx),Property(Variable ... TRING?) :: INTEGER?),Vector(CoerceTo(Parameter(AUTOSTRING1,String),String))))))),Any)))))
+Expand(All)	621	55	61	39/10	cached[user.id], r, trx, user	(user)-[r:SUBMITTED]-(trx)
+NodeUniqueIndexSeek	25	6	12	39/10	cached[user.id], user	:User(id)

Lisa 22 - AGR11 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=1834.41..1834.43 rows=1 width=40)
  Group Key: user_id
  ->  Sort  (cost=1834.41..1834.41 rows=1 width=14)
        Sort Key: user_id
        ->  Bitmap Heap Scan on transaction  (cost=30.41..1834.40 rows=1 width=14)
"              Recheck Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
"              Filter: ((date_created >= to_timestamp('2019-10-26'::text, 'YYYY-MM-DD'::text)) AND (date_created <= to_timestamp('2019-11-02'::text, 'YYYY-MM-DD'::text)))"
              ->  Bitmap Index Scan on idx_transaction_user_id_recipient_id  (cost=0.00..30.40 rows=513 width=0)
"                    Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
```

Lisa 23 - AGR11 PostgreSQL käivitusplaan



Operator	Estimated Rows	Rows	DB Hits	Cache H/M	Identifiers	Other
+ProduceResults	12	5	0	0/0	last_1_d_sum, user.id	
+EagerAggregation	12	5	5	39/10	last_1_d_sum, user.id	user.id
+Filter	155	5	195	39/10	cached[user.id], r, trx, user	AndedPropertyInequalities(Variable(trx),Property(Variable ... TRING?) :: INTEGER?),Vector(CoerceTo(Parameter(AUTOSTRING1,String),String))))))),Any)))))
+Expand(All)	621	55	61	39/10	cached[user.id], r, trx, user	(user)-[r:SUBMITTED]-(trx)
+NodeUniqueIndexSeek	25	6	12	39/10	cached[user.id], user	:User(id)

Lisa 24 - AGR12 PostgreSQL käivitusplaan

```
GroupAggregate  (cost=11957.21..14051.94 rows=1 width=16)
  Group Key: trx.user_id
  ->  Nested Loop  (cost=11957.21..14051.93 rows=1 width=16)
        ->  Merge Join  (cost=11956.78..13668.46 rows=51 width=24)
              Merge Cond: (uipa.user_id = trx.user_id)
              ->  Group  (cost=11956.35..11976.90 rows=2740 width=16)
"                   Group Key: uipa.user_id, uipa2.user_id"
                    ->  Sort  (cost=11956.35..11963.20 rows=2740 width=16)
"                         Sort Key: uipa.user_id, uipa2.user_id"
                          ->  Nested Loop  (cost=29.23..11799.90 rows=2740 width=16)
                                ->  Bitmap Heap Scan on user_ip_address uipa  (cost=28.80..1048.93 rows=291 width=16)
"                                     Recheck Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
                                      ->  Bitmap Index Scan on idx_user_ip_address__user_id_ip_address_id_unique  (cost=0.00..28.73 rows=291 width=0)
"                                           Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
                                ->  Index Only Scan using idx_user_ip_address__ip_address_id_user_id on user_ip_address uipa2  (cost=0.43..36.85 rows=9 width=16)
                                      Index Cond: (ip_address_id = uipa.ip_address_id)
                                      Filter: (user_id <> uipa.user_id)
              ->  Index Only Scan using idx_transaction_user_id_recipient_id on transaction trx  (cost=0.43..1655.51 rows=513 width=16)
"                   Index Cond: (user_id = ANY ('{53429,77649,98627,2486,86675,20274}'::bigint[]))"
        ->  Index Only Scan using idx_transaction_user_id_recipient_id on transaction trx2  (cost=0.43..7.51 rows=1 width=16)
              Index Cond: ((user_id = uipa2.user_id) AND (recipient_id = trx.recipient_id))
              Filter: (user_id <> trx.user_id)
```

Lisa 25 - AGR12 Neo4j käivitusplaan

```
+----------------------+----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| Operator             | Estimated Rows | Rows | DB Hits | Cache H/M       | Time (ms)   | Identifiers                                                           | Other                                   |
+----------------------+----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| +ProduceResults      |              3 |    4 |       0 |      79604/4356 |       0.014 | user.id, users_count                                                  | 14020                                   |
| |                    +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| +EagerAggregation    |              3 |    4 |       0 |     116049/4342 |       0.064 | user.id, users_count                                                  | user.id; 63701                          |
| |                    +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| +NodeHashJoin        |              9 |    5 |       0 | 80672149/342855 |       8.986 | trx, ip, cached[user.id], ipr, po, rec, r2, r, u2, po2, user, ipr2, trx2 | user, u2; 8985972                       |
| |\                   +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| | +Filter            |          10672 | 1410 |     705 | 13371439/342855 |       0.973 | trx, cached[user.id], po, rec, r2, r, u2, po2, user, trx2              | not r = r2; 973103                      |
| | |                  +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| | +Expand(All)       |          10672 |  705 |    1410 | 13371439/342855 |       4.151 | trx, cached[user.id], po, rec, r2, r, u2, po2, user, trx2              | (trx2)-[r2:SUBMITTED]-(u2); 4150766     |
| | |                  +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| | +Filter            |           6411 |  705 |       0 | 14414078/369299 |       4.357 | trx, cached[user.id], po, rec, r, po2, user, trx2                      | not po = po2; 4356991                   |
| | |                  +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| | +Expand(All)       |           6411 |  760 |     815 |   1044084/27331 |       7.252 | trx, cached[user.id], po, rec, r, po2, user, trx2                      | (rec)<-[po2:PAYED_OUT]-(trx2); 7252030  |
| | |                  +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| | +Filter            |            516 |   55 |      55 |   1044084/27331 |       7.483 | trx, cached[user.id], po, rec, r, user                                 | rec:Recipient; 7482662                  |
| | |                  +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| | +Expand(All)       |           1033 |   55 |     110 |   1044084/27331 |       7.578 | trx, cached[user.id], po, rec, r, user                                 | (trx)-[po:PAYED_OUT]-(rec); 7577559     |
| | |                  +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| | +Expand(All)       |            621 |   55 |      61 |    116190/4420 |       7.645 | cached[user.id], r, trx, user                                          | (user)-[r:SUBMITTED]-(trx); 7644697     |
| | |                  +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| | +NodeUniqueIndexSeek |             25 |   12 |      12 |    232380/8840 |      15.515 | cached[user.id], user                                                  | 15515242; :User(id)                     |
| |                    +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| +Filter              |           4046 | 15838 |    8330 |     74696117/0 |      15.821 | ip, cached[user.id], ipr, u2, user, ipr2                               | not ipr = ipr2; 15820901                |
| |                    +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| +Expand(All)         |           4046 |  8330 |    9152 |      7398201/0 |      24.477 | ip, cached[user.id], ipr, u2, user, ipr2                               | (ip)<-[ipr:USED]-(u2); 24476988         |
| |                    +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| +Filter              |            450 |  822 |     822 |      7398201/0 |      25.399 | cached[user.id], ip, ipr2, user                                        | ip:IpAddress; 25398601                  |
| |                    +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| +Expand(All)         |            450 |  822 |     828 |        55195/0 |      26.129 | cached[user.id], ip, ipr2, user                                        | (user)-[ipr2:USED]->(ip); 26129370      |
| |                    +----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
| +NodeUniqueIndexSeek |             25 |   12 |      18 |       110390/0 |      52.675 | cached[user.id], user                                                  | 52675346; :User(id)                     |
+----------------------+----------------+------+---------+-----------------+-------------+-----------------------------------------------------------------------+-----------------------------------------+
```

Lisa 26 – Agregatsioonide tulemused

<table>
<tr><th>Agregatsiooni identifikaator</th><th>PostgreSQL</th><th>Neo4j</th></tr>
<tr><td>AGR1</td><td>
<table>
<tr><td>user_id</td><td>total_ip_count</td></tr>
<tr><td>2486</td><td>99</td></tr>
<tr><td>20274</td><td>500</td></tr>
<tr><td>53429</td><td>3</td></tr>
<tr><td>77649</td><td>20</td></tr>
</table>
</td><td>
<table>
<tr><td>user_id</td><td>total_ip_count</td></tr>
<tr><td>2486</td><td>99</td></tr>
<tr><td>20274</td><td>500</td></tr>
<tr><td>53429</td><td>3</td></tr>
<tr><td>77649</td><td>20</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Agregatsiooni identifikaator</th><th>PostgreSQL</th><th>Neo4j</th></tr>
<tr><td></td><td><table><tr><td>86675</td><td>150</td></tr><tr><td>98627</td><td>50</td></tr></table></td><td><table><tr><td>86675</td><td>150</td></tr><tr><td>98627</td><td>50</td></tr></table></td></tr>
<tr><td>AGR2</td><td><table><tr><td>user_id</td><td>ip_total_other_users</td></tr><tr><td>2486</td><td>906</td></tr><tr><td>20274</td><td>4242</td></tr><tr><td>53429</td><td>32</td></tr><tr><td>77649</td><td>179</td></tr><tr><td>86675</td><td>1314</td></tr><tr><td>98627</td><td>443</td></tr></table></td><td><table><tr><td>user_id</td><td>ip_total_other_users</td></tr><tr><td>2486</td><td>906</td></tr><tr><td>20274</td><td>4242</td></tr><tr><td>53429</td><td>32</td></tr><tr><td>77649</td><td>179</td></tr><tr><td>86675</td><td>1314</td></tr><tr><td>98627</td><td>443</td></tr></table></td></tr>
<tr><td>AGR3</td><td><table><tr><td>user_id</td><td>ip_total_other_users_bad</td></tr><tr><td>2486</td><td>10</td></tr><tr><td>20274</td><td>53</td></tr><tr><td>53429</td><td>1</td></tr><tr><td>77649</td><td>2</td></tr><tr><td>86675</td><td>10</td></tr><tr><td>98627</td><td>7</td></tr></table></td><td><table><tr><td>user_id</td><td>ip_total_other_users_bad</td></tr><tr><td>2486</td><td>10</td></tr><tr><td>20274</td><td>53</td></tr><tr><td>53429</td><td>1</td></tr><tr><td>77649</td><td>2</td></tr><tr><td>86675</td><td>10</td></tr><tr><td>98627</td><td>7</td></tr></table></td></tr>
<tr><td>AGR4</td><td><table><tr><td>user_id</td><td>total_suspicious_ip_count</td></tr><tr><td>2486</td><td>10</td></tr><tr><td>20274</td><td>57</td></tr><tr><td>53429</td><td>1</td></tr><tr><td>77649</td><td>2</td></tr><tr><td>86675</td><td>9</td></tr></table></td><td><table><tr><td>user_id</td><td>total_suspicious_ip_count</td></tr><tr><td>2486</td><td>10</td></tr><tr><td>20274</td><td>57</td></tr><tr><td>53429</td><td>1</td></tr><tr><td>77649</td><td>2</td></tr><tr><td>86675</td><td>9</td></tr></table></td></tr>
</table>

<table>
<tr><th>Agregatsiooni identifikaator</th><th>PostgreSQL</th><th>Neo4j</th></tr>
<tr><td></td><td><table><tr><td>98627</td><td>7</td></tr></table></td><td><table><tr><td>98627</td><td>7</td></tr></table></td></tr>
<tr><td>AGR5</td><td><table><tr><td>user_id</td><td>max_fraud_count_on_ip</td></tr><tr><td>2486</td><td>1</td></tr><tr><td>20274</td><td>2</td></tr><tr><td>53429</td><td>1</td></tr><tr><td>77649</td><td>1</td></tr><tr><td>86675</td><td>2</td></tr><tr><td>98627</td><td>1</td></tr></table></td><td><table><tr><td>user_id</td><td>max_fraud_count_on_ip</td></tr><tr><td>2486</td><td>1</td></tr><tr><td>20274</td><td>2</td></tr><tr><td>53429</td><td>1</td></tr><tr><td>77649</td><td>1</td></tr><tr><td>86675</td><td>2</td></tr><tr><td>98627</td><td>1</td></tr></table></td></tr>
<tr><td>AGR6</td><td><table><tr><td>user_id</td><td>count</td></tr><tr><td>2486</td><td>697</td></tr><tr><td>20274</td><td>857</td></tr><tr><td>53429</td><td>131</td></tr><tr><td>77649</td><td>377</td></tr><tr><td>86675</td><td>747</td></tr><tr><td>98627</td><td>554</td></tr></table></td><td><table><tr><td>user_id</td><td>bad_count</td></tr><tr><td>2486</td><td>697</td></tr><tr><td>20274</td><td>857</td></tr><tr><td>53429</td><td>131</td></tr><tr><td>77649</td><td>377</td></tr><tr><td>86675</td><td>747</td></tr><tr><td>98627</td><td>554</td></tr></table></td></tr>
<tr><td>AGR7</td><td><table><tr><td>user_id</td><td>total_user_count_via_recipients</td></tr><tr><td>2486</td><td>127</td></tr><tr><td>20274</td><td>129</td></tr><tr><td>53429</td><td>63</td></tr><tr><td>86675</td><td>125</td></tr><tr><td>98627</td><td>253</td></tr></table></td><td><table><tr><td>user_id</td><td>users_count</td></tr><tr><td>2486</td><td>127</td></tr><tr><td>20274</td><td>129</td></tr><tr><td>53429</td><td>63</td></tr><tr><td>86675</td><td>125</td></tr><tr><td>98627</td><td>253</td></tr></table></td></tr>
</table>

<table>
<tr><th>Agregatsiooni identifikaator</th><th>PostgreSQL</th><th>Neo4j</th></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>AGR8</td><td>
<table>
<tr><td>user_id</td><td>count</td></tr>
<tr><td>2486</td><td>1</td></tr>
<tr><td>53429</td><td>1</td></tr>
<tr><td>86675</td><td>2</td></tr>
<tr><td>98627</td><td>1</td></tr>
</table>
</td><td>
<table>
<tr><td>user_id</td><td>bad_users_count</td></tr>
<tr><td>2486</td><td>1</td></tr>
<tr><td>53429</td><td>1</td></tr>
<tr><td>86675</td><td>2</td></tr>
<tr><td>98627</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>AGR9</td><td>
<table>
<tr><td>user_id</td><td>user_count</td></tr>
<tr><td>2486</td><td>683</td></tr>
<tr><td>20274</td><td>674</td></tr>
<tr><td>53429</td><td>559</td></tr>
<tr><td>86675</td><td>682</td></tr>
<tr><td>98627</td><td>782</td></tr>
</table>
</td><td>
<table>
<tr><td>user_id</td><td>bad_user_count</td></tr>
<tr><td>2486</td><td>683</td></tr>
<tr><td>20274</td><td>674</td></tr>
<tr><td>53429</td><td>559</td></tr>
<tr><td>86675</td><td>682</td></tr>
<tr><td>98627</td><td>782</td></tr>
</table>
</td></tr>
<tr><td>AGR10</td><td>
<table>
<tr><td>user_id</td><td>sum</td></tr>
<tr><td>2486</td><td>1114.68</td></tr>
<tr><td>20274</td><td>701.34</td></tr>
<tr><td>53429</td><td>1186.27</td></tr>
<tr><td>86675</td><td>1226.87</td></tr>
<tr><td>98627</td><td>1310.32</td></tr>
</table>
</td><td>
<table>
<tr><td>user_id</td><td>last_1_d_sum</td></tr>
<tr><td>2486</td><td>1114.68</td></tr>
<tr><td>20274</td><td>701.34</td></tr>
<tr><td>53429</td><td>1186.27</td></tr>
<tr><td>86675</td><td>1226.87</td></tr>
<tr><td>98627</td><td>1310.32</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Agregatsiooni identifikaator</th><th>PostgreSQL</th><th>Neo4j</th></tr>
<tr><td>AGR11</td><td>
<table>
<tr><td>user_id</td><td>sum</td></tr>
<tr><td>2486</td><td>4304.25</td></tr>
<tr><td>20274</td><td>6640.74</td></tr>
<tr><td>53429</td><td>3081.25</td></tr>
<tr><td>86675</td><td>6136.12</td></tr>
<tr><td>98627</td><td>10137.22</td></tr>
</table>
</td><td>
<table>
<tr><td>user_id</td><td>last_7_d_sum</td></tr>
<tr><td>2486</td><td>4304.25</td></tr>
<tr><td>20274</td><td>6640.74</td></tr>
<tr><td>53429</td><td>3081.25</td></tr>
<tr><td>86675</td><td>6136.12</td></tr>
<tr><td>98627</td><td>10137.22</td></tr>
</table>
</td></tr>
<tr><td>AGR12</td><td>
<table>
<tr><td>user_id</td><td>total_user_count_via_recipients_and_ip</td></tr>
<tr><td>2486</td><td>1</td></tr>
<tr><td>20274</td><td>2</td></tr>
<tr><td>86675</td><td>1</td></tr>
<tr><td>98627</td><td>1</td></tr>
</table>
</td><td>
<table>
<tr><td>user_id</td><td>users_count</td></tr>
<tr><td>2486</td><td>1</td></tr>
<tr><td>20274</td><td>2</td></tr>
<tr><td>86675</td><td>1</td></tr>
<tr><td>98627</td><td>1</td></tr>
</table>
</td></tr>
</table>