LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

POSTGRESQL, NEO4J UN MONGODB VEIKSPĒJAS ANALĪZE

BAKALAURA DARBS

Autors: **Alabama Sole**

Stud. Apl. nr.: AS18258

Darba vadītāja: profesore Dr.sc.comp. Laila Niedrīte

RĪGA 2022

ANOTĀCIJA

Bakalaurā darbā “PostrgeSQL, Neo4j un MongoDB veiktspējas analīze” tiek salīdzinātas divu veidu datubāžu pārvaldības sistēmas – relāciju un NoSQL, kā arī tika veikta veiktspējas analīze.

Mūsdienās relāciju un NoSQL ir vienas no populārākajām datubāzu pārvaldības sistēmām. Darba ietvaros detalizēti tiek apskatīts šo sistēmu jēdziens, ka arī tika noteiktas priekšrocības, trūkumi un īpašības. Salīdzinājumam tika izvelētas trīs dažādu tipu datubāzes - PostgreSQL, MongoDB un Neo4j, kuras tiek izpētītas un savstarpēji salīdzinātas.

Viens no svarīgākajiem aspektiem datubāzes izvēlei ir tas veiktspēja, tāpēc galvenais pētījuma mērķis ir salīdzināt PostgreSQL, MongoDB un Neo4j veiktspēju izmantojot DBeaver testēšanas rīku.

Darba rezultāts ir dati par PostgreSQL, MongoDB un Neo4j DBPS veiktspēju, kas iegūti, izmantojot DBeaver testēšanas sistēmu. Pamatojoties uz iegūtajiem datiem par operāciju izpildes laiku, tika izdarīti secinājumi par pētāmās DBPS veiktspēju.

Atslēgvārdi: datubāzes pārvaldības sistēmas, relāciju datubāzes, NoSQL datubāzes, veiktspēja, PostgreSQL, MongoDB, Neo4j.

ABSTRACT

PERFORMANCE ANALYSIS OF POSTGRESQL, NEO4J AND MONGODB

The bachelor's thesis "Performance analysis of PostgreSQL, Neo4j and MongoDB" compares two types of database management systems - relational and NoSQL, as well contains a performance evaluation.

Nowadays relational and NoSQL are some of the most popular database management systems. The concept of these systems is discussed in more detail in the paper, as well as the advantages, disadvantages and features are identified. Three different types of databases - PostgreSQL, MongoDB and Neo4j have been selected for comparison, which are studied and compared with each other.

One of the most important aspects of choosing a database is its performance, since the speed of the database depends on it. Therefore, the main goal of the study is to compare the performance of PostgreSQL, MongoDB and Neo4j using the DBeaver testing tool.

The papers' result is data regarding the performance of PostgreSQL, MongoDB and Neo4j DBMS obtained using the DBeaver testing system. Based on the obtained data on the execution time of operations, conclusions were made about the performance of the investigated DBMS.

Keywords: Database Management System, Relational Database, NoSQL Database, Performance, PostgreSQL, MongoDB, Neo4j.

SATURS

APZĪMĒJUMU SARAKSTS

DBPS – datubāzes pārvaldības sistēma.

NoSQL – "Not only sql", datubāzes pārvaldības sistēmas veids.

SQL – "Structured Query Language", ir standartizēta programmēšanas valoda, ko izmanto, lai pārvaldītu relāciju datubāzes.

PostgreSQL – relāciju datubāzes pārvaldības sistēma.

MongoDB – NoSQL dokumentorientēta datubāzes pārvaldības sistēma.

Neo4j – NoSQL grafu datubāzes pārvaldības sistēma.

ACID – akronīms, kas apzīmē četras galvenās transakcijas īpašības: atomitāti, konsekvenci, izolāciju un izturību.

Java – objektorientētā programmēšanas valoda.

Cypher - ir Neo4j grafu vaicājumu valoda.

ASCII art - vizuālās mākslas forma, kurā attēlu attēlošanai tiek izmantotas ASCII rakstzīmes uz viena attāluma datora terminālļa vai printera ekrāna.

CSV – "Comma-Separated Values", ir norobežots teksta fails, kurā vērtību atdalīšanai izmanto komatu.

IEVADS

Dažādās darbības jomās ir ļoti daudz datu, tādēļ ir jāpievērš uzmanība to apstrādei un uzglabāšanai, jo īpaši datu pārvaldības sistēmu veiktspējai. Datubāzes veiktspēju var definēt kā ātrumu, ar kādu datubāzes pārvaldības sistēma (DBPS) piegādā informāciju lietotājiem[13].

Divi no izplatītākajiem datubāzu pārvaldības sistēmu (DBPS) veidiem ir relāciju un NoSQL DBPS, kas atšķiras dažādos darba aspektos[15]. Kardinālas atšķirības tādos aspektos kā uzticamība, elastīgums, datu saskaņotība un mērogojamība, prasa rūpīgu dažādu sistēmu funkcionēšanas momentu analīzi, jo īpaši veiktspēju.

Pēdējos gados NoSQL datubāzes kļūst arvien populārākas. Tas, kas aizsākās kā nišas parādība, sāka konkurēt ar "veco" relāciju SQL datubāzu popularitāti. Tomēr relāciju datubāzēm joprojām ir savs pielietojums un tās tuvākajā laikā nepazudīs. Pastāvot plašam piedāvājumam ar dažādām datubāzēm, katrā atsevišķā situācijā ir būtiski izvēlēties piemērotāko, gan atbilstoši datu veidam, gan veiktspējas īpašībām.

Bakalaura darba ietvaros tika veikts relāciju un NoSQL datubāzu pārvaldības sistēmas salīdzinājums izmantojot trīs datubāzes - PostgreSQL, MongoDB un Neo4j.Šīs trīs datubāzes tika izvēlētas pēc šādiem kritērijiem:

- Relāciju datubāze, kas ir diezgan populāra, bet tās izstrādātājs nav Oracle vai Microsoft[16];
- NoSQL datubāzes, kuri ir vispopulārākās savos tipos (dokumentorientētas un grafu DBPS)[17, 18].

Galvenais darba mērķis ir salīdzināt PostgreSQL, MongoDB un Neo4j datubāzes un izvērtēt to atšķirības, priekšrocības un trūkumus, kā arī veikt datubāzes veiktspējas pētījumu.

Darbā gaitā tika izvirzīti sekojoši uzdevumi:

1) izveidot īsu pārskatu par relāciju DBPS un NoSQL datubāzēm, kā arī salīdzināt trīs datubāzes: PostgreSQL, MongoDB un Neo4j;

2) izpētīt veidu, kā veikt datubāzes veiktspējas pārbaudi, izmantojot veiktspējas testēšanas rīku;

3) apkopot un salīdzināt veiktspējas testēšanas rezultātus no literatūras avotiem;

4) sagatavot testēšanas datus un augšupielādēt tos katrā no datubāzēm;

5) veikt veiktspējas pētījumu un apkopot rezultātus.

Bakalaura darbs sastāv no četrām nodaļām:

Datubāzes pārvaldības sistēmas nodaļa satur relāciju un NoSQL datubāzu aprakstu, kā arī tika veikts PostgreSQL, MongoDB un Neo4j teorētisks salīdzinājums;

Datu shēma nodaļā tika aprakstīta datu shēma, kas tika izmantota pētījumā. Šī nodaļa satur datu shēmas diagrammu, tabulu aprakstus un informāciju par testēšanas datu ģenerēšanu;

Datubāzes testēšana nodaļā ir apkopota visa nepieciešamā informācija par darba pētījumu: testēšanas rīka apraksts, eksperimenta vides apraksts un iegūtie dati.

Secinājumi šī nodaļa paredzēta, lai apkopotu bakalaura darbā iegūtos rezultātus.

1. DATUBĀZU PĀRVALDĪBAS SISTĒMAS

Praktiski katrā nozarē, biznesā vai zinātniskajos pētījumos cilvēki saskaras problēmām, kas ir saistītas ar informācijas glabāšanu un apstrādi. Viens no iemesliem, kāpēc tika izstrādātas datubāzes, ir organizēta piekļuve datiem. Datubāzu evolūcija sākās ar datubāzēm, kas balstītas uz saistītajiem sarakstiem, tālāk tika izstrādāts relāciju datu modelis, kopā ar relāciju datubāzēm pēc tam – objektorientētas, un visbeidzot – NoSQL[19].

Mūsdienās ir divi visbiežāk izmantotie datubāzu tipi: relāciju un NoSQL. Neskatoties uz to, ka, salīdzinājumā ar citām, NoSQL datubāzes ir diezgan jaunas, tās ir kļuvušas diezgan populāras, pateicoties spējai ātri apstrādāt ne tikai strukturētus datus. Plašais datubāzu klāsts apgrūtina izstrādātājiem izvelēties vispiemērotāko sistēmu. Datubāzu izvēle ir ļoti svarīgs solis jebkurā izstrādes procesā, jo nepareizai izvēlei var būt nelabojamas sekas nākotnē, tā kā sistēmas darbības laikā ir grūti nomainīt datubāzi, īpaši uz cita veida. Relāciju un NoSQL datubāzu tipu salīdzinājums ļaus identificēt to spēcīgās un vājās puses, kā arī piemērotību konkrēta uzdevuma risināšanai.

1.1. Relāciju datubāzes

1.1.1. Apraksts

Datubāzes tiek definētas kā jebkuru datu jeb informācijas kopums, kas ir strukturēts un organizēts datora ātrai meklēšanai[1]. Lietojot terminu “datubāze”, bieži apzīmē visu sistēmu, bet termins definē tikai kolekcijas un to datus. Datubāzu pārvaldības sistēma (DBPS) ir sistēma jeb programmu kopums, kas pārvalda datus, transakcijas un citus datubāzu lietojumus. Relāciju DBPS ir sistēmas, kurās dati tiek sakārtoti vairākās tabulas jeb relācijās, kas atvieglo datu reorganizēšanu un piekļūšanu tiem[22]. Relāciju datubāzes ir pasaulē populārākās datubāzu pārvaldības sistēmas, un tās atbalsta dažādas piegādātāju implementācijas[19].

No daudzajām relāciju datubāzu pārvaldības sistēmām bakalaura darba pētījumam tika izvēlēta PostgreSQL, kura detalizētāk ir aprakstītā sadaļā 1.2.

1.1.2. Relāciju datu modelis

Relāciju datu modelis izmanto tabulu kolekciju, lai attēlotu gan datus, gan šo datu savstarpējās attiecības (relācijas). Tabulas ir loģiskas struktūras, kuras organizē un uztur datubāzes pārvaldnieks. Relāciju modelis sastāv no trim komponentēm: struktūras, integritātes un manipulēšanas daļām[19]:

- Strukturālā daļa definē datubāzi kā attiecību jeb relāciju kopumu;
- Datubāzes integritāte tiek uzturēta relāciju modelī, izmantojot primārās un ārējās atslēgas;
- Relāciju algebra un relāciju aprēķini ir rīki, ko izmanto, lai manipulētu datus datubāzē. Tādējādi relāciju modelim ir spēcīgs matemātiskais fons.

1.1.3. Transakcijas

Datubāzes transakcija ir sakārtota datu apstrādes operatoru secība, kas var pārveidot datubāzi no viena stāvokļa uz citu[19]. Relāciju datubāzēs tā var nosaukt operatoru vai operatorus, kas tiek izpildīti secīgu operāciju veidā, kas kopīgi pārstāv vienotu uzdevumu.

DBPS, kas ir izmantotā lietojumprogrammā, var pieprasīt vairākas transakcijas. Tīmekļa vidē vairākiem lietotājiem ir iespēja mēģināt piekļūt vienā datubāzē glabātajiem datiem vienlaicīgi. Lai saglabātu datubāzes precizitāti un konsekvenci, tiek izmantoti vairāki plānošanas algoritmi. Lai vecinātu efektīvu DBPS caurlaidspēju ir jāievieš noteiktas vienlaicīgas izpildes[19].

Transakciju pārvaldnieks (transaction manager) ir atbildīgs par transakciju plānošanu un drošāko ceļu nodrošināšanu uzdevuma izpildei. Lai saglabātu datubāzes integritāti vaicājumu apstrādes laikā, DBPS ir jānodrošina četras svarīgas īpašības. Šīs īpašības sauc par ACID īpašībām[19].

1.1.4. ACID

Relāciju datubāzes garantē augstu transakciju uzticamību, kas tiek nodrošināta, pilnībā atbalstot visas četras ACID īpašības[19]:

- atomitāte (atomicity): ja kāda transakcijas daļa netiek izpildīta, tad netiek izpildīts viss vaicājums;
- konsekvence (consistency): ja datubāze bija konsekventā stāvoklī pirms vaicājuma izpildes, tad arī pēc izpildes tā būs konsekventā stāvoklī;
- izolācija (isolation): transakcijas veikšanas brīdī citas paralēli veiktās transkacijas savstarpēji neietekmēs to rezultātus;
- izturība (durability): neatkarīgi no problēmām zemākajos līmeņos, piemēram, sistēmas darbības pārtraukums vai aparatūras kļūmes, veiksmīgi izpildītas transakcijas veiktās izmaiņas būs saglabātas arī pēc sistēmas darba atsākšanas.

1.1.5. Relāciju datubāzes priekšrocības

Relāciju datubāzes pārvaldības sistēmām ir daudz priekšrocību. Dažas no priekšrocībām ir minētas zemāk[19]:

- ***Novērš datu redundanci***

Relāciju datubāzu pārvaldības sistēmās tabulām ar konkrētiem datiem ir savstarpēja saistība, tāpēc nepieciešamie dati tiek ņemti no iepriekšējām tabulām, kas novērš datu redundanci.

- ***Datu drošība***

Relāciju datubāzu pārvaldības sistēmās datu piekļuve ir priviliģēta, kas nozīmē, ka datubāzes administratoram ir tiesības piešķirt piekļuvi datiem dažiem konkrētiem lietotājiem, tādējādi padarot datus drošus.

- ***Ērti un viegli izmantot***

Šī tipa datubāzē tiek izmantotas tabulas, kas viegli veidojamas un lietojamas. Lielākā daļa relāciju datu bāzu izmanto SQL valodu datu pārvaldībai un vaicājumu veikšanai, kas ir vispopulārākā programmēšanas valoda datubāzes vidē.

1.1.6. Relāciju datubāzes trūkumi

Neskatoties uz relāciju datubāzu popularitāti, eksistē tādas situācijas, kad tās nav labākais variants datu glābšanai. Zemāk ir minēti relāciju datubāzu trūkumi[19]:

- ***Veiktspēja***

Relāciju datubāzes veiktspēja ir atkarīga no tabulu skaita. Ja ir vairāk tabulu, tad vaicājumu izpildes ātrums būs lēnāks. Turklāt lielāks datu apjoms ne tikai palēnina sistēmas darbību, bet arī sarežģī informācijas atrašanu.

- ***Fiziskā uzglabāšana***

Relāciju datubāzei ir nepieciešams milzīgs fiziskās atmiņas apjoms, jo tā sastāv no rindām un kolonnām. Katra no operācijām ir atkarīga no atsevišķas fiziskās uzglabāšanas. Tikai ar pienācīgu optimizāciju var izveidot mērķa lietojumprogrammas, lai tām būtu maksimāla fiziskā atmiņa.

- ***Struktūras ierobežojumi***

Lauki, kas atrodas relāciju datubāzē, ir ar ierobežojumiem. Ierobežojumi būtībā nozīmē to, ka konkrētā laukā nevar iekļaut vairāk informācijas, nekā tas bijis paredzēts sākumā. Neraugoties uz to, ka tiek sniegta plašāka informācija, tā var izraisīt datu zaudējumu. Tāpēc ir jāapraksta precīzs datu apjoma apjoms, kas tiks dots šim laukam.

1.1.7. Relāciju datubāzes veiktspēja

Datubāzes veiktspēja ir ātrums, ar kādu datubāze atbild uz datu piekļuves pieprasījumiem. Relāciju datubāzes veiktspēju ietekmē pieci faktori: darba slodze, caurlaidspēja, resursi, optimizācija un konkurence[11].

Darba slodze ir viens no galvenajiem faktoriem, kas ietekmē datubāzes veiktspēju. Darba slodzi nosaka datu piekļuves pieprasījumi, ko sistēma vai galalietotājs sūta datubāzei. Lielāks pieprasījumu skaits nozīmē palielinātu darba slodzi, kas pieprasa vairāk apstrādes operāciju no datubāzes puses.

Caurlaidspējai ir svarīga loma datubāzes veiktspējā, kas ir visa procesa ievades/izvades ātruma mērs. Šis ievades/izvades ātrums parāda, vai uz vaicājumiem tiek atbildēts ātri vai lēni. Tā ir kombinācija no: ievadizvades ātruma, procesora ātruma, sistēmas paralelizācijas iespēju, datubāzes kodola un operētājsistēmas veiktspējas un sistēmas programmatūras. Sistēmas rīcībā esošie aparatūras un programmatūras līdzekļi ir zināmi kā sistēmas resursi. Piemēri: datu bāzes kodols, krātuves vieta, atmiņa, kešatmiņa utt.

Resursi tiešā veidā ietekmē datubāzes caurlaidspēju un citus faktorus. Efektīviem aparatūras un programmatūras resursiem, piemēram, procesoram, atmiņai un ķešatmiņas vadībai ir svarīga loma datu bāzes augstas veiktspējas veicināšanā.

Datubāzes veiktspējas ceturtais elements ir ***optimizācija***. Visi sistēmu tipi var tikt optimizēti, bet daudzas datubāzes sistēmas var veikt vaicājumu optimizāciju, kas galvenokārt notiek pašā DBPS. Tomēr ir arī citi faktori, kurus dažreiz jāoptimizē, piemēram, SQL formulējums, datubāzes iestatījumi, datubāzes organizācija, u.c., lai ļautu datubāzes optimizētājam izveidot efektīvākos piekļuves ceļus datiem.

Ja pieprasījums (darba slodze) konkrētam resursam ir augsts, var rasties ***konkurence***. Konkurence ir stāvoklis, kurā divi vai vairāk darba slodzes komponenti mēģina izmantot vienu resursu konfliktējošajā veidā, piemēram, viena un tā paša datu fragmenta paralēla atjaunināšana.

Tādējādi relāciju datubāzes veiktspēju var noteikt kā resursu lietojuma optimizāciju, lai palielinātu caurlaidspēju un samazinātu konfliktus, kas ļauj apstrādāt maksimāli iespējamo darba slodzi.

1.2. NoSQL datubāzes

Pēdējās desmitgades laikā tīmekļa lietojumprogrammu mijiedarbības metodes ar datiem ir mainījušās. Apkopo un izmanto arvien vairāk datu, un arvien vairāk lietotāju var piekļūt šiem datiem. Tas nozīmē to, ka mērogojamība un veiktspēja kļūst par arvien svarīgākiem faktoriem.

SQL mērogojamības problēma tika atzīta par interneta kompānijām ar plašām pieaugošām datu vajadzībām un infrastruktūru, piemēram, Google, Amazon un Facebook. Tās nāca klajā ar saviem risinājumiem – tādām tehnoloģijām kā BigTable, DynamoDB un Cassandra.

Šī pieaugošā interese apkopot vairāk datu ir radījusi vairākas NoSQL datubāzes pārvaldības sistēmas (DBPS), koncentrējoties uz veiktspēju, uzticamību un konsekvenci. Vairākas esošās indeksu struktūras ir pārskatītas un uzlabotas, lai uzlabotu meklēšanas un lasīšanas veiktspēju.

NoSQL datubāzes nosaukums ietver sevī akronīmu "NoSQL" – kas ir atšifrēts kā "Not Only SQL" un kas precīzāk attēlo pieeju, kas apvieno ne-relāciju datubāzes ar relāciju datubāzes izmantošanu[3].

Pēdējos gados ir izstrādātas daudzas jaunas sistēmas, kas nodrošina labu horizontālo mērogošanu vienkāršām lasīšanas/rakstīšanas darbībām datubāzēs, kas izplatītas pa vairākiem serveriem. Tādu sistēmu piemēri ir Amazon DynamoDB [4], Apache HBase [5], MongoDB [6]. Salīdzinot ar iepriekšminētajiem, relāciju datubāzes piedāvā mazāk iespēju mērogošanai.

Bakalaura darba pētījumam tika izvelēti MongoDB un Neo4j, kuri detalizētāk tiek apskatīti sadaļās 1.4. un 1.5.

1.2.1. NoSQL datubāzes īpašības

Daudzas jaunas sistēmas tiek sauktas par "NoSQL" datu krātuvēm. Parasti šādām sistēmām piemīt sekojošas īpašības[7]:

1. Slodzes izlīdzināšana starp vairākiem serveriem.
2. Datu sadalījuma iespēja vairākos serveros.
3. Vienkāršs protokols operāciju izsaukšanai (salīdzinājumā ar SQL).
4. Zemāks paralēlisma modelis nekā ACID-transakcijām.
5. Efektīva sadalīto indeksu un operatīvās atmiņas izmantošana datu glabāšanai.
6. Fiksētas datu shēmas trūkums.

NoSQL sistēmas parasti neatbilst ACID transakciju īpašībām: galu galā tiek atļauta konsekvence. Tā vietā tiek piedāvāts modelis "BASE" (Basically Available, Soft state, Eventually consistent) pretstatā ACID[7]. Ideja ir tāda, ka daudz labāku veiktspēju un mērogojamību var sasniegt, atceļot ACID ierobežojumus. Lielākā daļa sistēmu atšķiras atkarībā no ACID noņemšanas pakāpes.

1.2.2. CAP teorēma

NoSQL atbalstītāji bieži atsaucas uz CAP teorēmu, kurā teikts, ka sistēma var atbilst tikai divām no šīm trim īpašībām[7]:

- konsekvence (consistency) — dati vienmēr ir vienādi visās kopijās;
- pieejamība (availability) – dati vienmēr ir pieejami lietotājam;
- izturīga pret atdalīšanu (partition tolerance) — datu bāzes sistēma turpina darboties pareizi, neskatoties uz tīkla vai mezgla kļūmēm.

1.2.3. NoSQL datubāzes kategorijas

Bieži vien datu modeļi NoSQL sistēmās ietilpst šādās kategorijās [7]:

1.2.3.1. Tabula NoSQL datubāzes kategorijas

Kategorija	**Apraksts**
Atslēga-vērtība krātuves (key-value stores)	Glabā vērtības un identifikatoru, lai meklētu, pamatojoties uz doto atslēgu.
Dokumentu krātuves (document stores)	Sistēma saglabā datus dokumentu formā, kuri var tikt indeksēti.
Paplašināmas ierakstu krātuves (extensible record stores)	Ierakstus šādās krātuvēs var sadalīt vertikāli un horizontāli pa mezgliem.
Grafu krātuves (graph stores)	Dati ir reprezentēti grafos.

1.2.4. NoSQL datubāzes priekšrocības

Avotā [8] bija veikts relāciju un NoSQL DBPS salīdzinājums, kura rezultātos bija aprakstītas NoSQL datubāzu priekšrocības.

- ***Nestrukturēta datu shēma***

NoSQL nodrošina vienkāršu datu glabāšanas veidu, jo datubāzes struktūra nav ierobežota. To var mainīt pēc vajadzības, bez ierobežojumiem. Lietojumprogrammām, kurās datu struktūra nav

galīga, piemēram, tīkla dati, skaņa vai attēli, NoSQL nodrošina efektīvu datu glabāšanas/iegūšanas apstrādes veidu.

- ***Mērogojamība***

Vajadzība pēc NoSQL bija nepieciešama, jo ģenerēto datu apjoms bija lielāks nekā jebkad agrāk. Turklāt mūsdienās dati ir ne tikai teksta formātos, bet gan arī bināros formātos, piemēram, attēli, video, skaņas, u.c. NoSQL tika ieviests, lai uzlabotu datubāzu mērogojamību. NoSQL ļauj horizontāli mērogot datubāzi, pievienojot vairāk serveru.

- ***Lietošanas ekonomiskums***

NoSQL datubāzes ir salīdzinoši lētākas, lai instalētu un pārvaldītu, salīdzinot ar tradicionāliem SQL serveriem. Tas ir tāpēc, ka NoSQL nav nepieciešamas licences serveru un datu centru darbībai, tādējādi samazinot lietojumprogrammas un uzturēšanas izmaksas.

1.2.5. NoSQL datubāzes trūkumi

Avotā [8] veiktajā relāciju un NoSQL DBPS salīdzinājumā, tika aprakstīti arī šo datubāzu trūkumi.

- ***Mazāka lietotāju sabiedrība***

Izstrādātāji ir izmantojuši NoSQL datubāzes vairāk nekā desmit gadus, un lietotāju kopiena strauji aug, tomēr tā ir mazāka nekā SQL kopiena. Tāpēc potenciāli varētu būt grūtāk atrisināt nedokumentētas problēmas. NoSQL pusē ir arī mazāk konsultantu un ekspertu.

- ***Analīzes trūkums***

NoSQL ir paredzēts, lai apstrādātu lielu datu apjomu, kam ir dažāds raksturs. Tomēr, runājot par analīzi un biznesa ieskatu gūšanu, NoSQL nav tik efektīva kā SQL. Mūsdienās lietojumprogrammas parasti izmanto abus modeļus dažādiem lietošanas gadījumiem.

- ***Konsekvences trūkums***

NoSQL saskarās ar konsekvences problēmām, kad runa ir par lielu datu apjomu glabāšanu. SQL datubāzes nodrošina to, ko sauc par ACID transakcijām (Atomic, Consistent, Isolated,

Durability), kas nozīmē, ka pastāv uzticamu darījumu garantija. NoSQL to standartā neatbalsta, liekot programmētājiem rakstīt savu pielāgoto kodu.

1.2.6. NoSQL datubāzes veiktspēja

Tā kā NoSQL piedāvā dažādus datubāzes veidus, NoSQL veiktspēju ietekmē dažādi faktori. Dažādu datubāzu metodei un pieejai noteikti ir ietekme. Piemēram, dokument-orientēta datubāze darbosies citādi nekā grafu bāzētā datubāze pat tad, ja pamatā esošā aparatūra ir līdzīga.

Aplūkojot NoSQL veiktspēju, jānovērtē tās efektivitāti, izmantojot šādus rādītājus[12]:

- ***caurlaidspēja*** – operāciju skaits sekundē;
- ***ielādēšanas process*** — cik ātri datu bāze var iegūt datus;
- ***vaicājuma veiktspēja*** — cik ātri atbildes uz vaicājumiem atgriež rezultātus, un tas var būt atkarīgs no vairāk nekā no caurlaidspējas.

Tā kā NoSQL datubāzes kļūst sarežģītākas, ir grūtāk rakstīt efektīvi kodētus vaicājumus, tādējādi ietekmējot veiktspēju. Tomēr NoSQL ir arī resursietilpīgs, kas secina, ka tā pamatā esošajai apstrādes jaudai ir liela ietekme uz veiktspēju.

2. SALĪDZINĀMĀS DATUBĀZES

2.1. PostgreSQL

PostgreSQL ir objektu relāciju datubāzes pārvaldības sistēma. Sistēmai ir pārbaudīta arhitektūra, kas ir izpelnījusies labu reputāciju ar uzticamību, datu integritāti un precizitāti. PostgreSQL arī pilnībā atbilst ACID un ANSI-SQL: 2008 standartiem[2].

PostgreSQL sevī ietver netriviālas iespējas, piemēram, atgriešanos noteiktā laikposmā (Point-in-time recovery), asinhronās replicēšanas tabulu starpā, iekšējās transakcijas (saglabāšanas punkti), rezerves kopijas izveidi izpildes laikā, pieprasījumu plānotājs un optimizators un citas[2]. Visas šīs funkcijas ļauj PostgreSQL būt labai alternatīvai NoSQL sistēmām mērogojamības plānā, saglabājot sarežģītus dziļus analītiskos pieprasījumus, izmantojot SQL.

2.1.1. Datu modelis

PostgreSQL ir ne tikai relāciju, bet arī objektu relāciju DBPS. Tas nozīmē, ka līdzīgi kā uz objektorientētām programmēšanas valodām, PostgreSQL atbalsta objektus, klases un mantošanu (inheritance)[2]. Tas tai dod vairākas priekšrocības salīdzinājumā ar citām atvērtā koda SQL datubāzēm, piemēram, MySQL, MariaDB un Firebird.

Objektu relāciju datubāzes pamatīpašība ir atbalsts pielāgotiem objektiem un to uzvedībai, ieskaitot datu tipus, funkcijas, darbības, domēnus un indeksus. Tas padara PostgreSQL elastīgu un uzticamu.

2.1.2. Īpašības

PostgreSQL datubāzes izstrādātāji to ir apveltījuši ar unikālām īpašībām, kas to atšķir no citām relāciju DBPS:

- ***Funkciju deklarēšana***

PostgreSQL funkcijas ir koda bloki, kas darbojas serverī, nevis datubāzes klientā[2]. Lai gan tos var rakstīt tikai SQL, papildu loģikas ieviešana, piemēram, “for-loop” deklarācija, ir ārpus SQL darbības jomas, un ir jāizmanto daži valodu paplašinājumi. Funkcijas var rakstīt izmantojot dažādas programmēšanas valodas. PostgreSQL ļauj izmantot funkcijas, kas atgriež ierakstu kopu, ko pēc tam var izmantot tāpat kā parastā vaicājuma rezultātu. Funkcijas var izpildīt gan ar to veidotāja tiesībām, gan ar pašreizējā lietotāja tiesībām. Dažreiz funkcijas tiek identificētas ar saglabātajām procedūrām, taču starp šiem jēdzieniem pastāv atšķirība.

- ***Jauna datu tipa izveide***

PostgreSQL dod iespēju definēt jaunu datu tipu ar komandu CREATE TYPE[2]. Šī komanda reģistrē jaunu datu tipu izmantošanai pašreizējā datu bāzē. Tipa īpašnieks ir lietotājs, kurš to izveidoja.

- ***Datu izmēri***

PostgreSQL var apstrādāt lielu datu apjomu. Pašreizējie publicētie ierobežojumi ir norādīti zemāk tabulā[2]:

2.1.2.1. Tabula PostgreSQL datu izmēri

Maksimālais datubāzes lielums	Neierobežots
Maksimālais tabulas izmērs	32 TB
Maksimālais rindas izmērs	1,6 TB
Maksimālais lauka izmērs	1 GB
Maksimālais rindu skaits tabulā	Neierobežots
Maksimālais kolonnu skaits tabulā	250–1600 atkarībā no kolonnas veida
Maksimālais indeksu skaits tabulā	Neierobežots

Salīdzinājumam, MySQL un MariaDB rindas lieluma ierobežojums ir 65 535 baiti. Firebird maksimālais rindas izmērs ir tikai 64 Kb. Parasti datu apjomu ierobežo operētājsistēmas maksimālais faila lielums. Tā kā PostgreSQL spēj saglabāt tabulu datus daudzos mazākos failos, tā spēj apiet šo ierobežojumu.

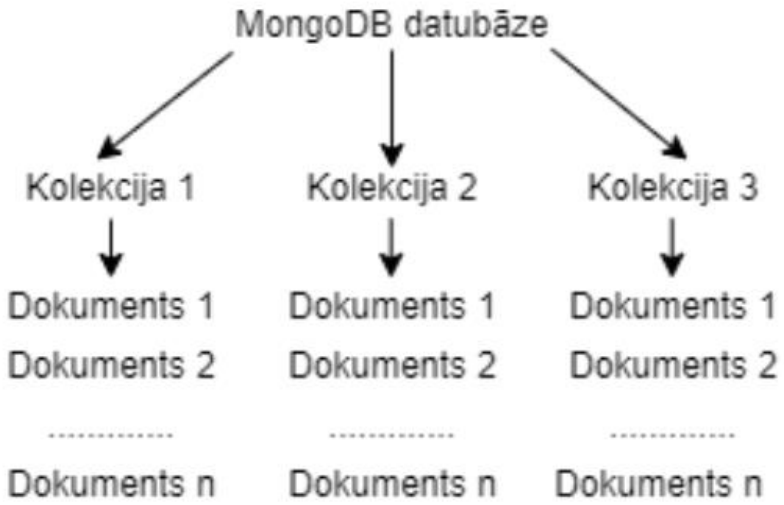
2.2. MongoDB

MongoDB ir dokumentorientēta datubāzu pārvaldības sistēma, kurai nav nepieciešams tabulu shēmas apraksts[9]. Tā tiek uzskatīta par vienu no klasiskajiem NoSQL sistēmu piemēriem, un tā izmanto JSON līdzīgus dokumentus un datubāzes shēmu. Šo datubāzi izmanto tīmekļa izstrādē, piemēram, JavaScript-orientētā MEAN, MEVN, MERN formātā, ko bieži izmanto izstrādātāji kā iecienītākās tehnoloģijas, lai izveidotu lietojumprogrammas.

2.2.1. Datu modelis

Ja relāciju datubāzēs saturu veido tabulas, tad MongoDB datubāze sastāv no kolekcijām[9]. Katrai kolekcijai ir savs unikālais nosaukums – patvaļīgs identifikators, kas sastāv no ne vairāk kā 128 dažādām rakstzīmēm.

Atšķirībā no relāciju datu bāzēm, MongoDB neizmanto tabulas ar fiksētu kolonnu un datu tipu skaitu. MongoDB ir uz dokumentiem orientēta sistēma, kurā galvenā koncepcija ir dokuments. Dokumentu var uzskatīt par objektu, kas glabā kādu informāciju. Savā ziņā tas ir līdzīgs ierakstiem relāciju DBPS, kur ieraksts glabā informāciju par vienu elementu. Zemāk ir MongoDB datu modeļa piemērs.



2.2.1.1. MongoDB datu modelis

2.2.2. Īpašības

Zemāk ir aprakstītas MongoDB īpašības[9]:

- ***Elastīga datu shēma***

Svarīga MongoDB iezīme ir šīs datubāzes tehnoloģijas bezshēmas dizains. Shēma attiecas uz struktūru, kādā dati tiek glabāti datubāzē. MongoDB nepiemēro stingru shēmu, kas noņem shēmas izveides un ievērošanas slogu jau no paša lietojumprogrammas izveides sākuma. Tādējādi tas nodrošina to, ka vienā kolekcijā lietotājs var brīvi saglabāt vairāku veidu dokumentus un veikt izmaiņas pēc vajadzības.

- ***Mērogojamība***

MongoDB ir pazīstama ar savu mērogojamību. Papildus mērogošanai vertikāli, lai pielāgotu vairāk datu, MongoDB var mērogot arī horizontāli. Horizontālā mērogošana attiecas uz datu sadali pa vairākiem serveriem, lai palielinātu vienlaikus apstrādājamo datu bāzes vaicājumu skaitu.

- ***Augsta veiktspēja***

Visas iepriekš apspriestās funkcijas nodrošina vienu augstāko kvalitāti - veiktspēju. Ar elastīgu datu shēmu, kā arī ļoti pieejamu un mērogojamu arhitektūru MongoDB piedāvā nepārspējamu veiktspēju salīdzinājumā ar citām NoSQL datubāzēm.

2.3. Neo4j

Neo4j[10] ir pasaulē vadošā grafu datubāze ar atvērtā avota kodu, kas izstrādāts, izmantojot Java tehnoloģiju. Tā ir labi mērogojama un nesatur shēmu (NoSQL). Grafu datubāze ir datubāze, kas datu modelē grafa formā. Šeit grafa mezgli attēlo entītijas, bet attiecības attēlo šo mezglu asociāciju. Neo4j sniedz elastīgu, vienkāršu, bet vienlaicīgi spēcīgu datu modeli, ko var viegli mainīt atkarībā no programmām un nozarēm. Darbam ar Neo4j tiek izmantota deklaratīvā Cypher pieprasījumu valoda, lai pielāgotu grafus. Šīs valodas komandas ir lasāmā formātā un vienkāršas apgūšanā.

2.3.1. Datu modelis

Neo4j Graph Database tanī esošos datus saglabā mezglos un attiecībās. Neo4j izmanto Native GPE (Graph Processing Engine), lai strādātu ar Native Graph uzglabāšanas formātu. Grafu datubāzes datu modeļa galvenie elementi ir mezgli, attiecības un īpašības. Zemāk ir vienkāršs grafa datu modeļa piemērs, kur "Employee" un "Department" ir mezgli, un "works" ir attiecība starp tiem.

2.3.1.1. Grafu datu modeļa piemērs

2.3.2. Īpašības

Zemāk ir aprakstītas Neo4j īpašības:

- ***ACID***

Neo4j atbilst ACID (atomitāte, konsistence, izolācija un izturība) īpašībām, kas ir raksturīgs relāciju datubāzēm nevis NoSQL[10].

- ***Cypher valoda***

Neo4j nodrošina jaudīgu deklaratīvo vaicājumu valodu, kas pazīstama kā Cypher Query Language[10]. Tā izmanto ASCII art, lai attēlotu grafus. Cypher ir viegli iemācīties, un to var izmantot, lai izveidotu un iegūtu attiecības starp datiem, neizmantojot sarežģītus vaicājumus, piemēram, savienojums (Join).

- ***Neo4j Browser***

Neo4j Browser ir interaktīvs izstrādes rīks, kas ļauj mijiedarboties ar grafu un vizualizēt tajā esošo informāciju izmantojot Cypher valodu[10]. Šī pārlūkprogramma ir iekļauta komplektā ar Neo4j un ir pieejama visos sistēmas izdevumos un versijās. Būtībā tas ir izstrādātāja rīks Cypher

vaicājumu veikšanai. Pārlūkprogramma ļauj parādīt vaicājuma rezultātus grafiskā vai tabulas formātā.

2.4. PostgreSQL, MongoDB, Neo4j salīdzinājums

Tā kā bakalaura darba mērķis ir salīdzināt trīs dažādas datubāzes pārvaldības sistēmas, tika izveidota tabula, kurā datubāzes ir salīdzinātas pēc vairākiem parametriem.

2.4.1. *Tabula datubāzes salīdzinājums*

Parametri	**PostgreSQL**	**MongoDB**	**Neo4j**
Īss apraksts	Plaši izmantota relāciju DBPS	Viena no populārākajām dokumentu krātuvēm	Viena no populārākajām grafu DBPS
Datu glabāšanas pamatmodelis	Relāciju datubāze	Dokumentorientēta NoSQL datubāze	Grafu NoSQL datubāze
Papildu datu glabāšanas modelis	Key/Value tipa datubāze	Key/Value tipa datubāze	Grafu tipa datubāze
Izstrādātājs	The PostgreSQL Global Development Group	MongoDB, Inc	Neo Technology
Realizācijas gads	1996	2009	2007
Pašreizējā versija	14.2	5.2	4.4.5
Mākoņkrātuve	Nē	Nē	Jā
Realizācijas valoda	C	C++	Java, Scala
Atbalstītas programmēšanas valodas	.Net, C, C++, Delphi, Java, JavaScript (Node.js), Perl, PHP, Python, Tcl	Actionscript, C, C#, C++, Clojure, ColdFusion, D, Dart, Delphi, Erlang, Go, Groovy, Haskell, Java, JavaScript,	Java, Python, Clojure, Ruby, PHP

		Lisp, Lua, MatLab, Perl, PHP, PowerShell, Prolog, Python, R , Ruby, Scala, Smalltalk	
SQL atbalsts	Jā	Nē	Nē
XML atbalsts	Nē	Nē	Jā
Ārējas atslēgas	Jā	Nē	Nē

Tabulā 2.4.1. ir redzamas galvenās DBPS atšķirības - datu glabāšanas pamatmodelis, izstrādātāji, realizācijas gads, realizācijas valoda, ka arī atbalstītās programmēšanas valodas. No salīdzināmajām datubāzēm tikai PostgreSQL atbalsta SQL un arējās atslēgas.

2.5. DBPS veiktspējas pētījumi

Jautājums par datubāzes veiktspēju laika gaitā kļūst arvien aktuālāks. Jo lielāks ir projekts un sarežģītāks datu modelis, jo vairāk datu tabulās, jo lielāka iespējamība saskaries ar lēnu darbību un nevēlamām problēmām.

Pirms bakalaura darba pētījuma uzsākšanas tika izpētīti jau paveikti citu autoru eksperimenti, kas saistīti ar datubāzes veiktspējas testēšanu. Šajā nodaļa tiek apkopoti pētījumu rezultāti, kā arī konstatēti trūkumi, kas motivēja veikt jaunu eksperimentu bakalaura darba ietvaros.

Viens no pētījumiem[20] analizē PostgreSQL, MongoDB un Neo4j datubāzu sistēmu veiktspēju. Eksperimentā datu shēmai tika paņemtas Mumbajas (pilsēta Indijā) kartes dati, kas sastāv no:

- marķējumiem, kas ir atribūti, kas saistīti ar jebkuru mezglu, veidu vai relāciju;
- mezgliem, kas ir mazākais objekts, ko var uzskatīt par punktu, kur ceļi ir kā malas vai savienojums starp mezgliem;
- attiecībām, kas ir punktu (mezglu) un veidu kopums, kas pārstāv lielāku ceļu vai mezglu kopu.

Pētījuma gaitā tika izveidotas trīs datubāzes – viena katrā no sistēmām. Datu shēma sastāvēja no 9 tabulām, datu importēšana tika nodrošināta caur csv failu. Kopējais mezglu skaits bija 520,689 un datu shēma aizņēma 2 GB. Zemāk ir norādīti divi pieprasījumi, kas tika veikti katrai datubāzei:

- Atrast visus lietotājus, kuri atzīmējuši kādu punktu kartē;
- Atrast visus konkrētā lietotāja izveidotos mezglus.

Pētījuma rezultāti: datubāzei Neo4j ir zemāka veiktspēja, nekā pārējām eksperimentā apskatītajām datubāzēm. Neo4j bija nepieciešams vislielākais laika periods, lai apstrādātu abus pieprasījumus. MongoDB uzrādīja vislabāko rezultātu apstrādājot pieprasījumus visātrāk. Pētījuma autori atzīmē, ka precīzākai analīzei šo eksperimentu jāturpina, izpētot ieraksta ievietošanas gadījumu un atjaunināšanas darbības.

Pētījuma trūkumi:

- testēšanas laikā netika izmantotas visas pieejamās tabulas;
- testēšanai izvelēti tikai "select" pieprasījumi un nav apskatīti ierakstu ievietošanas un atjaunināšanas darbības;
- izmantots tikai viens testēšanas datu apjoms.

Otrā pētījumā[21] PostgresSQL un MongoDB tiek salīdzināti, pārbaudot nestrukturētu datu rakstīšanas ātrumu un datubāzes datu rakstīšanas veiktspēju. Datu shēma sastāv no trim tabulām, kuras saistītas ar arējām atslēgām. Lai pārbaudītu veiktspējas atšķirības starp PostgreSQL un MongoDB nestrukturētiem datiem, šajā pētījumā tika izvērtēts nestrukturētu datu laika patēriņš apstrādei un datubāzes datu apjoma pieaugums. Eksperimenta ietvaros datubāzē tika ierakstīti aptuveni 15000 MB datu, attiecīgi tika reģistrēts katras operācijas patērētais laiks un tiem nepieciešamais atmiņas apjoms datu bāzē.

Pētījuma rezultāti: apstrādāt nestrukturētos datus apjomā 3600, 10600 un 14200 MB PosgtreSQL datubāzei laika patēriņš ir 567, 1841 un 2535 sekundes, savukārt MongoDB – 120, 331 un 438 sekundes. Laikam, kas nepieciešams datu ierakstīšanai PostgreSQL un MongoDB datubāzēs, ir lineāra saistība ar nestrukturēto datu daudzumu. PostgreSQL ierakstīšanas ātrums ir

0,178 s/MB, savukārt MongoDB ātrums ir 0,031 s/MB. MongoDB ir aptuveni 6 reizes efektīvāka nekā PostgreSQL.

Pētījuma trūkumi:

- Eksperimenta gaitā tika aplūkotas tikai ierakstīšanas darbības.

Šim eksperimentam ir līdzīga testēšanas stratēģija, kā autora darbam, bet pētījuma gaitā netika izmantota Neo4j datubāze.

3. DATU SHĒMA

Bakalaura darba ietvaros tika sagatavota nesarežģīta datu shēma ar dažādām tabulām un izveidota PosgtreSQL, MongoDB un Neo4j datubāzēs. Detalizētāk izveides procesi ir aprakstīti nodaļās 3.5-3.7.

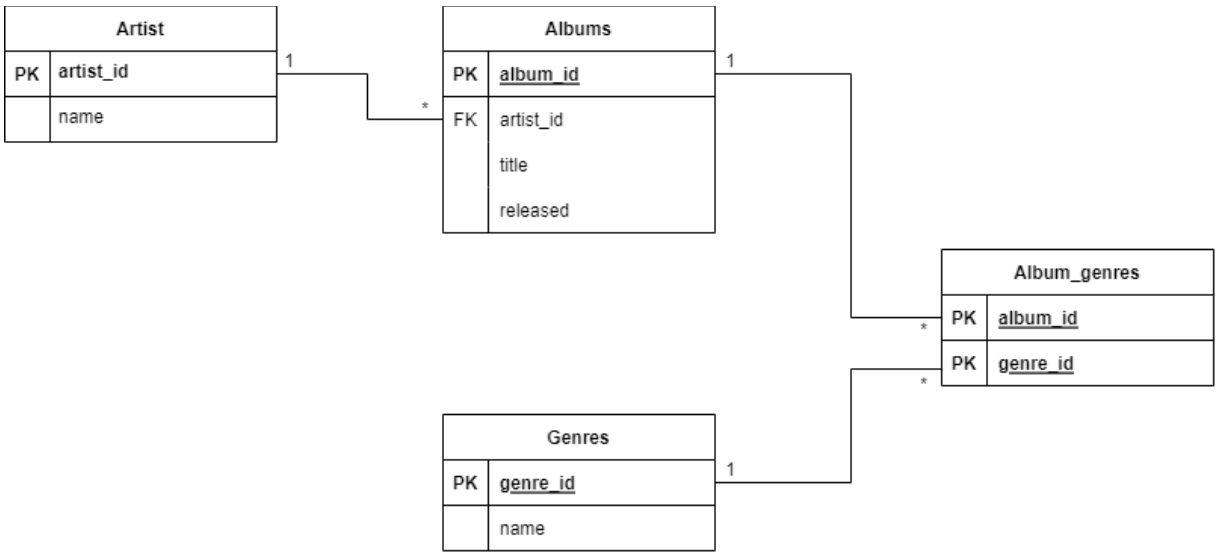
3.1. Datu shēmas apraksts

Turpmākai bakalaura darba pētījuma testēšanai sagatavota datu shēma ar 4 tabulām. Datu shēmas pamatā ir dati par mūziķiem, albumiem un žanriem. Testēšanas dati tika pievienoti csv failos un augšupielādēti katrā no datubāzēm. Detalizētāk testēšanas dati tiek aprakstīti nodaļā 2.4.

Datu shēma ir balstīta uz sekojošiem noteikumiem:

- mūzikas autoram ir vārds;
- autoram var būt daudz albumu (viens pret daudziem), kuriem ir nosaukums un izdošanas datums;
- žanriem ir nosaukums;
- albumi var piederēt daudziem žanriem (daudzi pret daudziem).

3.2. Datu shēmas ER modelis



3.2.1. Att. datu shēmas ER modelis

3.3. Tabulas apraksti

Tabula “Artist” (dati par mūziķiem).

3.3.1. Tabula Artist

Nosaukums	Datu tips	Apraksts
Artist_id	Int	Primārā atslēga
Name	Text	Vārds jeb pseidonīms

Tabula “Albums” (dati par albumiem).

3.3.2. Tabula Albums

Nosaukums	Datu tips	Apraksts
Album_id	Int	Primārā atslēga
Artist_id	Int	Arējā atslēga no tabulas “Artist”
Title	Text	Albuma nosaukums
Released	Date	Izlaišanas datums

Tabula “Albums_genres” (starptabula priekš tabulas “Albums” un “Genres”).

3.3.3. Tabula Ablums_genres

Nosaukums	Datu tips	Apraksts
Album_id	Int	Primārā atslēga, arējā atslēga no tabulas “Albums”
Genre_id	Int	Primārā atslēga, arējā atslēga no tabulas “Genres”

Tabula “Genres” (dati par žanriem).

3.3.4. Tabula Genres

Nosaukums	Datu tips	Apraksts
Genre_id	Int	Primārā atslēga
Name	Text	Žanra nosaukums

3.4. Testēšanas dati

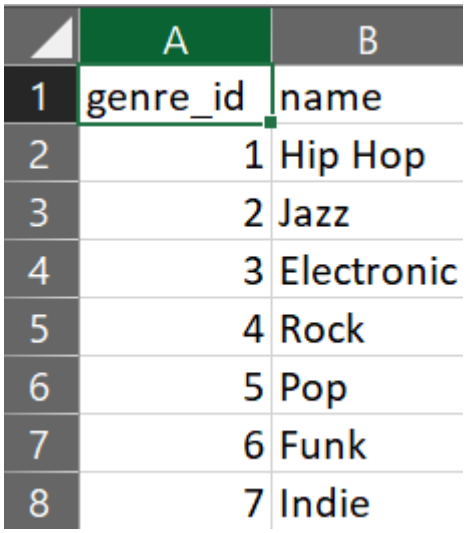
Testēšanas dati tika izveidoti caur programmu Microsoft Excel un importēti kā csv faili. Katrai tabulai tika sagatavoti trīs atsevišķi csv faili ar dažādu datu apjomu: mazs, vidējs un liels.

Tā kā tabulas primāras atslēgas ir veselo skaitļu datu tipa, tie tika ģenerēti ar Microsoft Excel programmas iebūvētas funkciju – "**ROW(row_number)**". Excel ROW funkcija atgriež rindas numuru atsaucei. Piemēram, ROW(C5) atgriež 5, jo C5 ir piektā rinda izklājlapā.

Tabulas lauki ar datu tipu teksts mazam datu apjomam tika sagatavoti manuāli – tika pievienoti īsti muzikantu pseidonīmi, albumu nosaukumi un žanri. Vidējam un lielam datu apjomiem teksta lauki tika ģenerēti ar tiešsaistes csv ģeneratora palīdzību[14].

Lauki "released" ar datu tipu datums un ārējās atslēgas tika ģenerēti ar Microsoft Excel programmas iebūvētas funkciju – "**RANDBETWEEN(bottom,top)".** Funkcija RANDBETWEEN atgriež nejaušu veselu skaitli starp diviem skaitļiem. Izmantojot šo funkciju arī ir iespējams ģenerēt datumus, izmantojot šo sintaksi:
"**RANDBETWEEN(DATE(start_date,end_date),DATE(start_date,end_date))".**

Csv faila piemērs tabulai "Genres":



	A	B
1	genre_id	name
2	1	Hip Hop
3	2	Jazz
4	3	Electronic
5	4	Rock
6	5	Pop
7	6	Funk
8	7	Indie

3.4.1. Att. Csv faila piemērs

Testēšanai tika izmantoti tabulā 3.4.2. norādītie datu apjomi:

3.4.2. Tabula testēšanas datu izmēri

Apjoms	**Ierakstu skaits**			
	Artists	***Albums***	***Genres***	***Albums_genres***
Mazs	200	220	50	220
Vidējs	20000	20000	1527	20000
Liels	200000	500000	1527	500000

3.5. PostgreSQL datu shēmas izveide

Mijiedarboties ar PosgtreSQL datubāzi izmantojot komandlīniju ir iespējams izmantojot "SQL Shell" vai arī grafisko rīku pgAdmin 4, kas tika izmantots bakalaura darbā pētījumā. Tika izveidota datubāze "bakalaura_darbs", kurā tika pievienotas tabulas ar SQL valodas skriptiem.

Skriptu piemēri "Artist" un "Albums" tabulas ģenerēšanai:

```
CREATE TABLE IF NOT EXISTS public."Artist"
(
    artist_id integer NOT NULL,
    name text NOT NULL,
    CONSTRAINT "Artist_pkey" PRIMARY KEY (artist_id)
)
```

```
CREATE TABLE IF NOT EXISTS public."Albums"
(
    album_id integer NOT NULL,
    title text NOT NULL,
    released date NOT NULL,
    artist_id integer NOT NULL,
    CONSTRAINT "Albums_pkey" PRIMARY KEY (album_id),
    CONSTRAINT artist_id FOREIGN KEY (artist_id)
        REFERENCES public."Artist" (artist_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

3.6. MongoDB datu shēmas izveide

Tā kā MongoDB ir NoSQL datubāze, pēc instalēšanas to ir iespējams izmantot caur komandlīniju (command prompt).

Lai uzsāktu darbu ar MongoDB serveri komandlīnijā jāievada sekojošu komandu: **"net start MongoDB"**.

MongoDB ir dokumentorientēta datubāze, tāpēc tabulas un ieraksti šinī sistēmā ir saukti par kolekcijām un dokumentiem. Datubāzi nav iespējams izveidot standarta komandrindā. Lai to izdarītu, jāpalaiž MongoDB komandlīniju – MongoDB shell ar kommandu: **"mongo"**. Mijiedarbība ar MongoDB notiek izmantojot MongoDB Query Language.

No sākuma ir nepieciešams inicializēt datubāzi. Jāievada komandu **"use *database_name*"**. Zemāk datubāzes "bakalaura_darbs" inicializēšana:

```
> use bakalaura_darbs
switched to db bakalaura_darbs
```

3.6.1. *Att. Datubāzes inicializēšana MongoDB*

Kolekciju izveidei ir paredzētas divas komandas:

- **"db.createCollection("*table_name*")" –** šinī gadījumā tiek izveidota tukša kolekcija, kolekcijas lauki netiek definēti, kā arī dokumentus jāpievieno ar atsevišķu komandu;
- **"db.*table_name*.insert({ *field_name*: *value*, …})" –** šinī gadījumā kolekcijas izveide, lauku definēšana un dokumentu pievienošana notiek vienlaicīgi.

Zemāk ir kolekcijas “Artist” un “Albums” inicializēšana:

```
> db.artist.insert({artist_id : 1, name: "lady gaga"})
WriteResult({ "nInserted" : 1 })
> db.artist.find()
{ "_id" : ObjectId("6259a101abbe3124b6dadb5f"), "artist_id" : 1, "name" : "lady gaga" }
> show collections
artist
> db.albums.insert({album_id: 1, title: "bad romance", released: "02.03.2010", artist_id: 1 })
WriteResult({ "nInserted" : 1 })
```

3.6.2. *Att. Tabulas “Artist” un “Albums” definēšana*

Lai pārbaudītu visas datubāzē eksistējošās kolekcijas, jāizmanto kommandu **“show collections”**:

```
> show collections
albums
albums_genres
artist
genres
```

3.6.3. *Att. Datubāzes “bakalaura_darbs” kolekcijas*

3.7. Neo4j datu shēmas izveide

Mijiedarbība ar Neo4j datubāzēm ir iespējama izmantojot Neo4j Desktop, kas ir instalējama lietojumprogramma, kas palīdz strādāt ar Neo4j. Neo4j Desktop piedāvā visas nepieciešamās funkcijas datubāzes organizēšanai i izveidei, rediģēšanai, failu importēšanai, u.c.

Caur Neo4j Desktop ir iespēja palaist Neo4j Browser, kas ir rīks paredzēts darbam ar konkrēto datubāzi.

Mijiedarbība ar datubāzi notiek caur iebūvēto komandrindu Neo4j Browser rīkā, izmatojot Cypher valodu. Grafu definēšanas komandas ir:

```
CREATE (ee:Person { name: “Test”, from: “Riga”, age: 8})
```

CREATE - funkcija grafu izveidei;

( ) – iekavas grafu identificēšanai;

ee:Person – mainīgais “ee” un iezīme “Person”;

{} – figūriekavas grafu īpašību definēšanai.

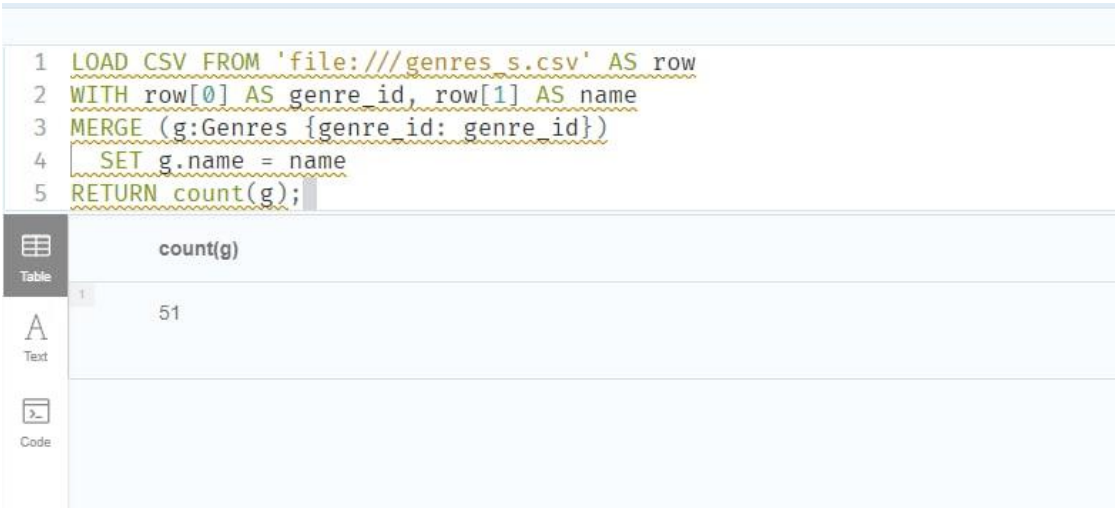
Tā kā eksperimentam ir paredzēti csv faili ar testēšanas datiem, datu ielāde tika veikta ar sekojošu komandu:

```
LOAD CSV FROM 'file:///genres_s.csv' AS row
WITH row[0] AS genre_id, row[1] AS name
MERGE (g:Genres {genre_id: genre_id})
  SET g.name = name
```

LOAD CSV FROM - funkcija datu importēšanai no csv failiem;

WITH row[0] AS genre_id – csv faila pirmās kolonas apzīmējums grafā;

MERGE – funkcija grafa savienojumam ar testēšanas datiem.



3.7.1. *Att. Grafa "Genres" izveide un testēšanas datu ielade Neo4j datubāzē*

4. VEIKSTPĒJAS TESTĒŠANA

Lai iegūtu datus par datubāzes veikto operāciju ātrumu, tika izmantota DBeaver testēšanas sistēma.

Šīs daļas uzdevums bija izpētīt testēšanas rīku DBeaver, kas ir aprakstīts sadaļā 3.1., un salīdzināt DBPS veiktspēju dažādām operācijām, kurām ir nepieciešama vairāku saistītu tabulu shēma. Detalizētāk testēšanas plāns ir aprakstīts sadaļā 3.2. un testēšanas rezultāti 3.3.

4.1. Testēšanas rīks

Datubāzes uzraudzība ir datubāzes veiktspējas mērīšanas un izsekošanas process atbilstoši galvenajiem rādītājiem, kas to ietekmē[13]. Šīs metrikas parasti tiek uzraudzītas reāllaikā, ļaujot noteikt vai prognozēt problēmas. Efektīva datubāzu uzraudzība sniedz arī iespēju uzlabot vai optimizēt datu bāzi, lai palielinātu tās vispārējo veiktspēju. Bakalaura darba ietvaros datubāzes uzraudzības process tiks izmantots kā veids veiktspējas noteikšanai.

Tirgū ir plašas gan maksas, gan bezmaksas rīku alternatīvas, kas paredzētas datubāzu uzraudzībai. Bakalaura darba pētījumam tika izvelēts DBeaver testēšanas rīks[23].

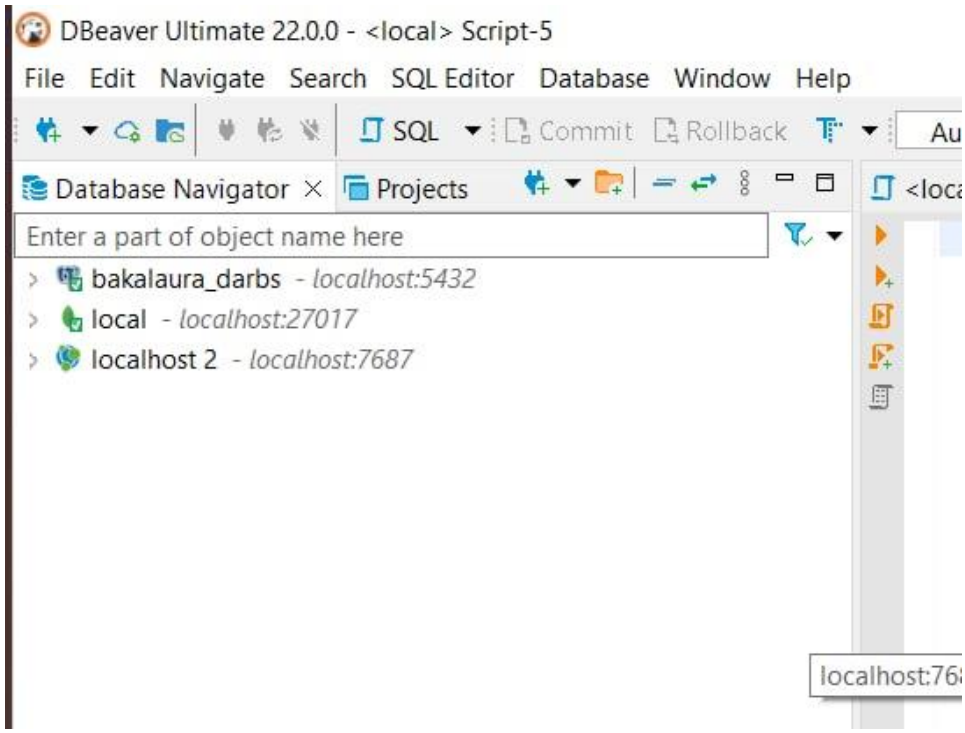
DBeaver ir bezmaksas, atvērtā koda, grafisko datubāzu pārvaldības un uzraudzības rīks datubāzu izstrādātājiem un administratoriem[23]. DBeaver var izmantot, lai izveidotu un pārvaldītu datubāzes dažādās datubāzu pārvaldības sistēmās (DBPS). DBeaver sadarbojas ar lielāko daļu populāro datubāzu pārvaldības sistēmām, piemēram, MySQL, PostgreSQL, MariaDB, SQLite, Oracle, DB2, SQL Server, Sybase. DBeaver ir vairāku platformu rīks – tas darbojas uz Windows, Linux, Mac un Solaris.

Kritēriji, pēc kuriem tika izvēlēts DBeaver rīks:

- ***Grafiskais lietotāja interfeiss*** (Graphical User Interface)[23]. Ļauj mijiedarboties ar datubāzi un to objektu sarakstu. Tas ir pretēji komandrindas rīkam, kurā būtu jāatceras precīzas komandas, lai parādītu datu bāzu sarakstu vai veiktu citus uzdevumus.
- ***Datubāzes analīze***[23]. Ir iespējams analizēt datubāzes procesus reāllaikā, izmantojot ģenerētos grafus. Ietver arī metadatu pārvaldību.

- ***SQL redaktors***[23]. Ļauj izveidot SQL vaicājumus un izpildīt tos datubāzē. Skriptus iespējams arī augšupielādēt un saglabāt pēc nepieciešamības. Ir arī iespējams paralēli palaist vairākus pieprasījumus.
- ***PosgtreSQL, MongoDB un Neo4j atbalsts***[23]. Tirgū ir maz rīku, kuri atbalsta šo trīs datubāzu uzraudzību.

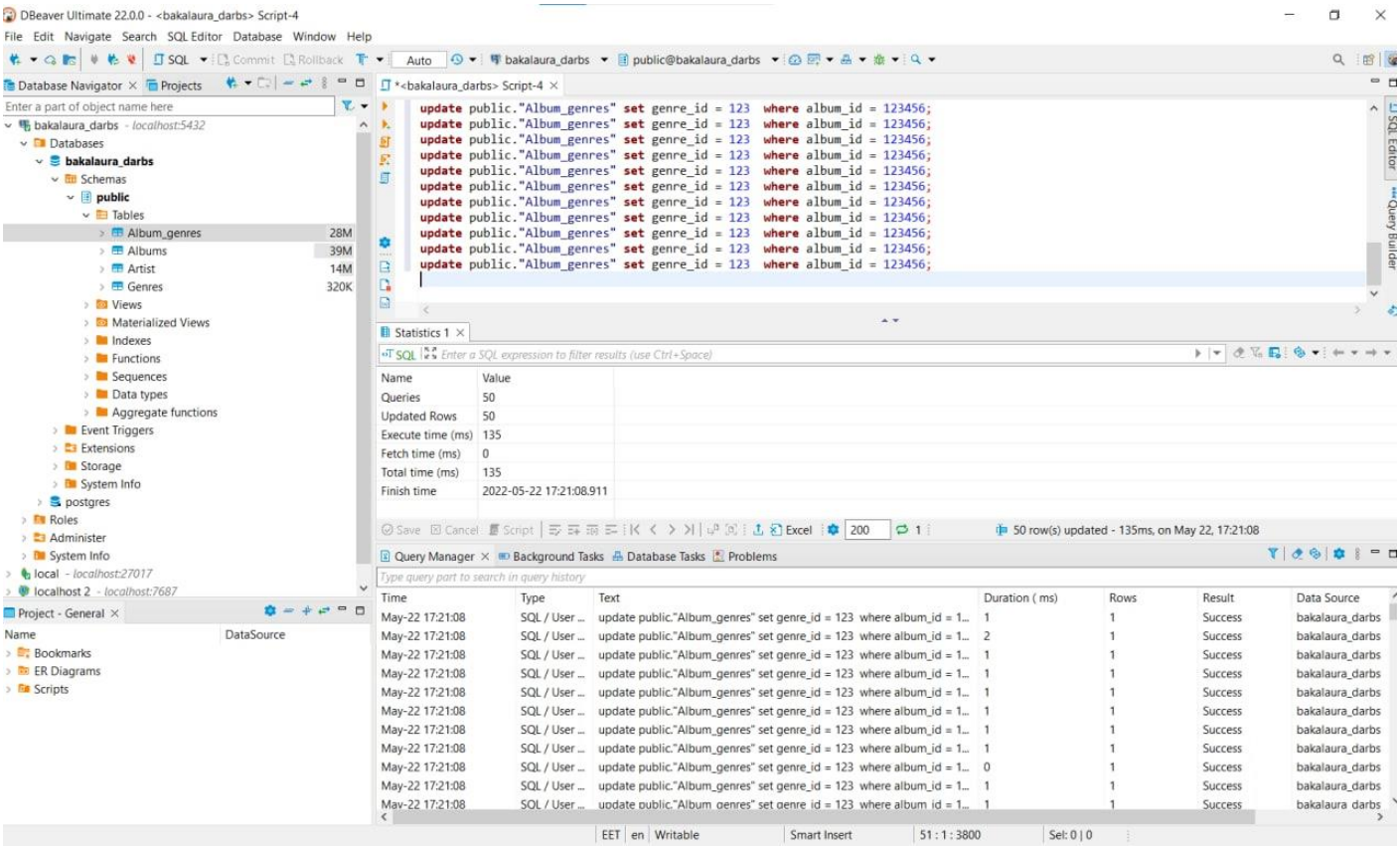
Datubāzes analīzei DBeaver rīkā jākonfigurē savienojumu ar nepieciešamajām datubāzēm. To ir iespējams izdarīt nospiežot pogu "Database" un no saraksta jāizvēlas "New database connection". Atvērsies jauns logs, kur jāizvēlas datubāzes sistēmu un jāievada nepieciešamo informāciju: lokālo adresi, lietotājvārdu un paroli. Zemāk ir DBeaver programmatūras ekrānuzņēmums, kad visas trīs sistēmas bija veiksmīgi savienotas ar rīku.



4.1.1. *Att. DBeaver ekrānskats*

DBeaver rīkam ir SQL skriptu atbalsts visiem datubāzu veidiem, bet šī opcija netika izmantota testēšanas gaitā, lai ietaupītu laiku apstrādājot vaicājumus katrai datubāzei. Visām datubāzes pārvaldības sistēmām tika sagatavoti pieprasījumi attiecīgajā programmēšanas valodā. Pieprasījumi PostgeSQL datubāzei ir aprakstīti tabulā 3.3.1, pieprasījumi parejām datubāzēm ir darba pielikumā.

Lai nosūtītu pieprasījumu sistēmai nepieciešams izmantot SQL skriptu redaktoru. To ir iespējams atvērt nospiežot pogu “SQL”. Paralēlo darbību izpildei skriptu redaktora logā jāievada nepieciešamo skriptu sarakstu un palaist to ar pogu kombināciju: “Ctrl+Alt+Shift+x”. Zemāk ir DBeaver ekrānuzņēmums pēc 150 darbību izpildes.



4.1.2. *Att. DBeaver ekrānskats*

4.2. Testēšanas vide

Eksperiments tika veikts izmantojot datoru, kam ir sekojoši tehniskie paramteri:

4.2.1. Tabula Datora konfigurācija

Operētājsistēma	Windows 10 Pro
Procesors	Intel® Core™ i7-5600U CPU @ 2.60GHz
Brīvpiekļuves atmiņa (RAM)	8 GB
Sistēmas tips	64-bit

Testēšanas laikā tikai izmantotas sekojošas datubāzu versijas:

- PostgreSQL – 14.0;
- MongoDB – 5.0;
- Neo4j – 4.4.7.

Eksperimenta gaitā tika izslēgti visi fona procesi un tika palaists tikai testēšanas rīks DBeaver.

4.3. Testēšanas plāns

Veiktspējas mērījumiem sistēmā tika veiktas šādas operācijas:

- ***Read*** — viena ieraksta lasīšana. Rezultāti ir sniegti tabulai Artist;
- ***Update*** - esoša ieraksta lauku pārrakstīšana. Rezultāti ir sniegti tabulai Albums_genres;
- ***Join*** — lasīšana no divu tabulu ārējās atslēgas savienojumiem. Rezultāti ir sniegti tabulām Artist un Albums;
- ***Count group by*** - ierakstu grupēšana tabulā, izmantojot unikālu ierakstu skaitu. Rezultāti ir sniegti tabulai Albums.

Pieprasījumi tika izpildīti trīs posmos, lai simulētu datubāzes darba slodzi:

1. etaps: katrs SQL skripts palaists vienu reizi;
2. etaps: 50 SQL skripti palaisti paralēli;
3. etaps: 100 SQL skripti palaisti paralēli;
4. etaps: 150 SQL skripti palaisti paralēli.

Tabulā 4.1.1. ir SQL pieprasījumi katrai no operācijām, kas tika izmantoti PostgreSQL testēšanai. Cypher (Neo4j) un MongoDB Query Language pieprasījumi ir darbā pielikumā.

4.3.1. Tabula Testēšanas SQL pieprasījumi

Operācija	Pieprasījums
Read	`SELECT * FROM "Artist" a WHERE`

	`"name"  LIKE 'A%';`
Update	`UPDATE "Album_genres" SET genre_id = 123  WHERE album_id = 123456;`
Join	`SELECT a.artist_id, a.name, a2.album, a2.released FROM "Artist" a JOIN "Albums" a2 on a2.artist_id = a.artist_id;`
Count group by	`SELECT COUNT(album_id), RELEASED FROM "Albums" a GROUP BY RELEASED;`

4.4. Testēšanas rezultāti

4.4.1. PostgreSQL datubāze

Tabulā 4.4.1.1 ir apkopoti PostgreSQL datubāzes testēšanas rezultāt, izmantojot mazo datu apjomu.

4.4.1.1. Tabula PostgreSQL datubāzes testēšanas rezultāti mazām datu apjomam

	Izpildes laiks(ms)			
Pieprasījums	**1 izpilde**	**50 paralēlas izpildes**	**100 paralēlas izpildes**	**150 paralēlas izpildes**
Read	55	61	130	187
Update	20	39	91	132
Join	118	100	190	289
Count group by	61	80	330	523

Tabulā 4.4.1.2 ir apkopoti PostgreSQL datubāzes testēšanas rezultāti izmantojot vidējo datu apjomu.

4.4.1.2. Tabula PostgreSQL datubāzes testēšanas rezultāti vidējām datu apjomam

	Izpildes laiks(ms)			
Pieprasījums	**1 izpilde**	**50 paralēlas izpildes**	**100 paralēlas izpildes**	**150 paralēlas izpildes**

Read	102	121	135	278
Update	12	56	121	233
Join	93	412	650	974
Count group by	82	638	1240	1758

Tabulā 4.4.1.3 ir apkopoti PostgreSQL datubāzes testēšanas rezultāti izmantojot lielo datu apjomu.

4.4.1.3. Tabula PostgreSQL datubāzes testēšanas rezultāti lielām datu apjomam

	Izpildes laiks(ms)			
Pieprasījums	**1 izpilde**	**50 paralēlas izpildes**	**100 paralēlas izpildes**	**150 paralēlas izpildes**
Read	73	76	167	253
Update	12	74	165	243
Join	212	4976	10474	13388
Count group by	330	15625	33172	46393

4.4.2. MongoDB datubāze

Tabulā 4.4.2.1 ir apkopoti MongoDB datubāzes testēšanas rezultāti izmantojot mazo datu apjomu.

4.4.2.1. Tabula MongoDB datubāzes testēšanas rezultāti mazām datu apjomam

	Izpildes laiks(ms)			
Pieprasījums	**1 izpilde**	**50 paralēlas izpildes**	**100 paralēlas izpildes**	**150 paralēlas izpildes**
Read	66	207	381	3207
Update	99	595	1172	2038
Join	167	1924	3473	5082
Count group by	78	189	330	519

Tabulā 4.4.2.2 ir apkopoti MongoDB datubāzes testēšanas rezultāti izmantojot vidējo datu apjomu.

4.4.2.2. Tabula MongoDB datubāzes testēšanas rezultāti vidējam datu apjomam

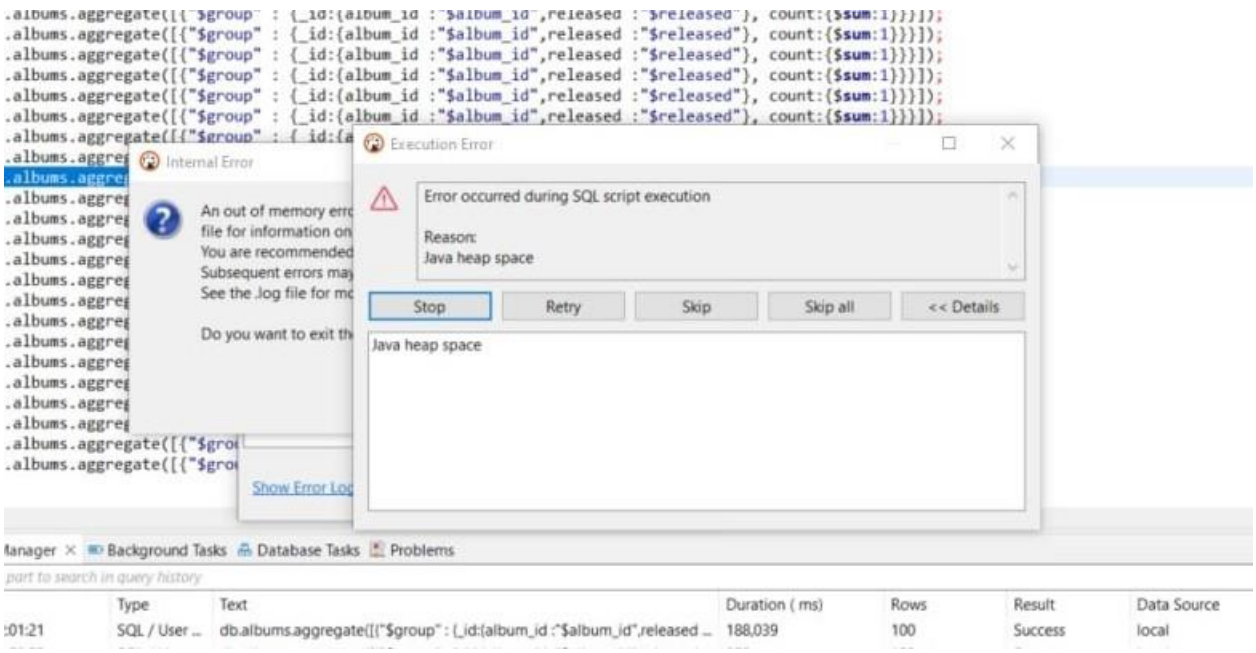
	Izpildes laiks(ms)			
Pieprasījums	**1 izpilde**	**50 paralēlas izpildes**	**100 paralēlas izpildes**	**150 paralēlas izpildes**
Read	69	288	483	709
Update	68	171	310	468
Join	273584	516459	125546789	189654763
Count group by	432	7486	12.640	**Error: Java heap space**

Tabulā 4.4.2.3 ir apkopoti MongoDB datubāzes testēšanas rezultāti izmantojot lielo datu apjomu.

4.4.2.3. Tabula MongoDB datubāzes testēšanas rezultāti lielam datu apjomam

	Izpildes laiks(ms)			
Pieprasījums	**1 izpilde**	**50 paralēlas izpildes**	**100 paralēlas izpildes**	**150 paralēlas izpildes**
Read	64	471	577	919
Update	58	240	329	560
Join	**Error: Java heap space**	**Error: Java heap space**	**Error: Java heap space**	**Error: Java heap space**
Count group by	3.797	**Error: Java heap space**	**Error: Java heap space**	**Error: Java heap space**

Eksperimenta gaitā, izmantojot vidējo un lielo datu apjomu, operācijām ***Join*** un ***Count group by*** rezultāti netika piefiksēti "Java heap space" problēmas dēļ.



4.4.2.4. Att. Java heap space

4.4.3. Neo4j datubāze

Tabulā 4.4.3.1 ir apkopoti Neo4j datubāzes testēšanas rezultāti izmantojot mazo datu apjomu.

4.4.3.1. Tabula Neo4j datubāzes testēšanas rezultāti mazām datu apjomam

	Izpildes laiks(ms)			
Pieprasījums	**1 izpilde**	**50 paralēlas izpildes**	**100 paralēlas izpildes**	**150 paralēlas izpildes**
Read	68	821	1794	1934
Update	72	605	992	1427
Join	110	884	1389	1831
Count group by	77	665	1039	1520

Tabulā 4.4.3.2 ir apkopoti Neo4j datubāzes testēšanas rezultāti izmantojot vidējo datu apjomu.

4.4.3.2. Tabula Neo4j datubāzes testēšanas rezultāti vidējam datu apjomam

	Izpildes laiks(ms)			
Pieprasījums	**1 izpilde**	**50 paralēlas izpildes**	**100 paralēlas izpildes**	**150 paralēlas izpildes**
Read	341	2924	3682	4390
Update	168	1156	1942	2758
Join	635	7749	12407	19409
Count group by	232	4156	7413	11220

Tabulā 4.4.3.3 ir apkopoti Neo4j datubāzes testēšanas rezultāti izmantojot lielo datu apjomu.

4.4.3.3. Tabula Neo4j datubāzes testēšanas rezultāti lielam datu apjomam

	Izpildes laiks(ms)			
Pieprasījums	**1 izpilde**	**50 paralēlas izpildes**	**100 paralēlas izpildes**	**150 paralēlas izpildes**
Read	700	12462	19256	25844
Update	110	817	1386	1798
Join	3768	128179	248201	399337
Count group by	2252	99438	213083	317566

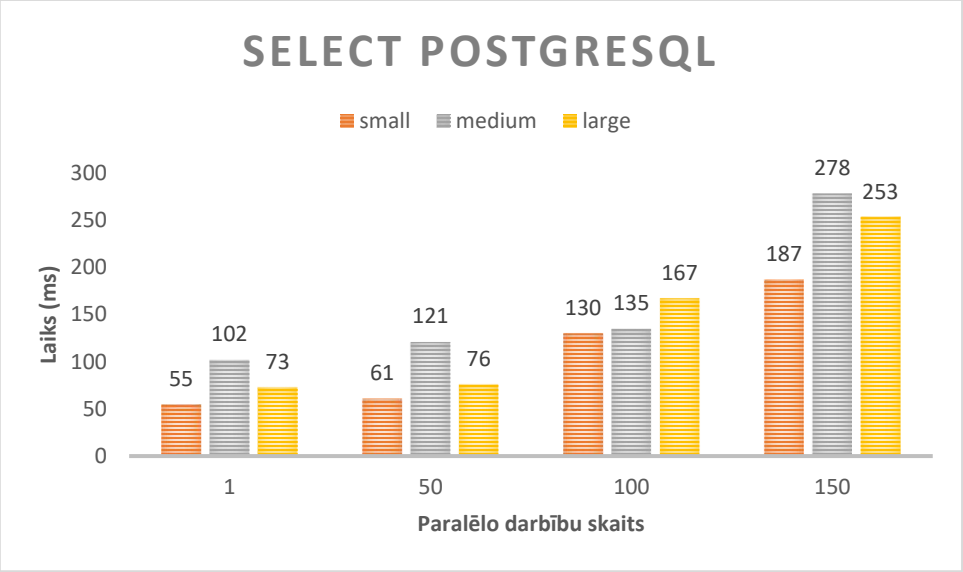
4.4.4. Rezultātu salīdzinājums pa operācijām

Diagrammās 4.4.4.1. – 4.4.4.12. attēlots datubāzes pieprasījumu izpildes laiks atkarībā no paralēlo darbību skaita maziem, vidējiem un lieliem datu apjomiem. Vertikālās ass vērtības atbilst operācijas izpildes laikam vai tā decimāllogaritmam, gadījumā kad vērtība ir liela. Horizontālas ass vērtības ir paralēlo darbību skaits.

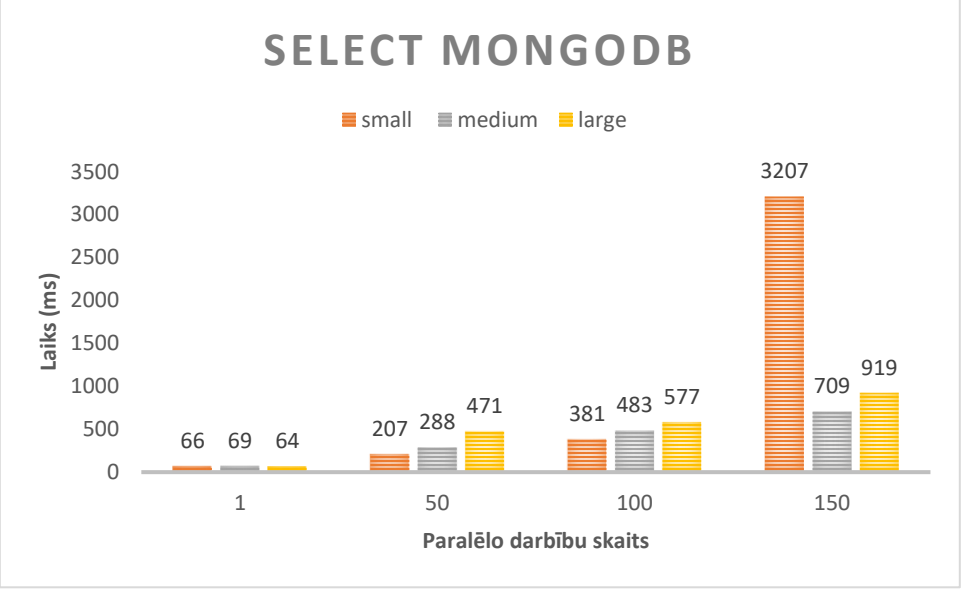
- ***Select***

Select operācijai vislabāko rezultātu uzrādīja relāciju datubāze PostgreSQL - izpildes laiks augs atkarībā no datu apjoma un paralēlo darbību skaita. Diagrammās redzams, ka vislielāko operācijas izpildes laiku visiem datu apjomiem uzrāda Neo4j. MongoDB parādīja nestabilus

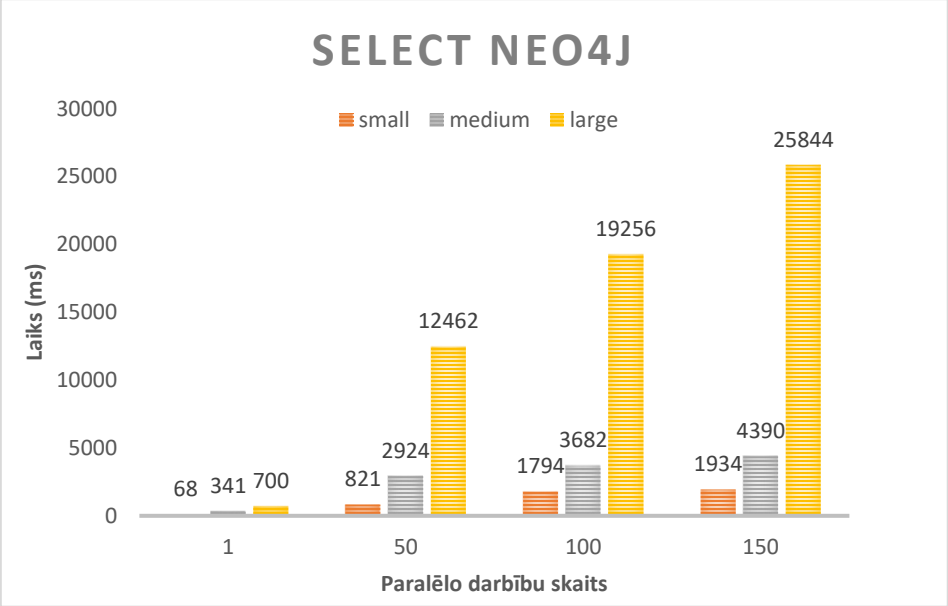
rezultātus – 150 paralēlo darbību izpildes laiks ar mazu datu apjomu bija gandrīz 3 reizes lielāks nekā lielam datu apjomam.



4.4.4.1. Att. PostgreSQL datubāzes rezultāti operācijai Select



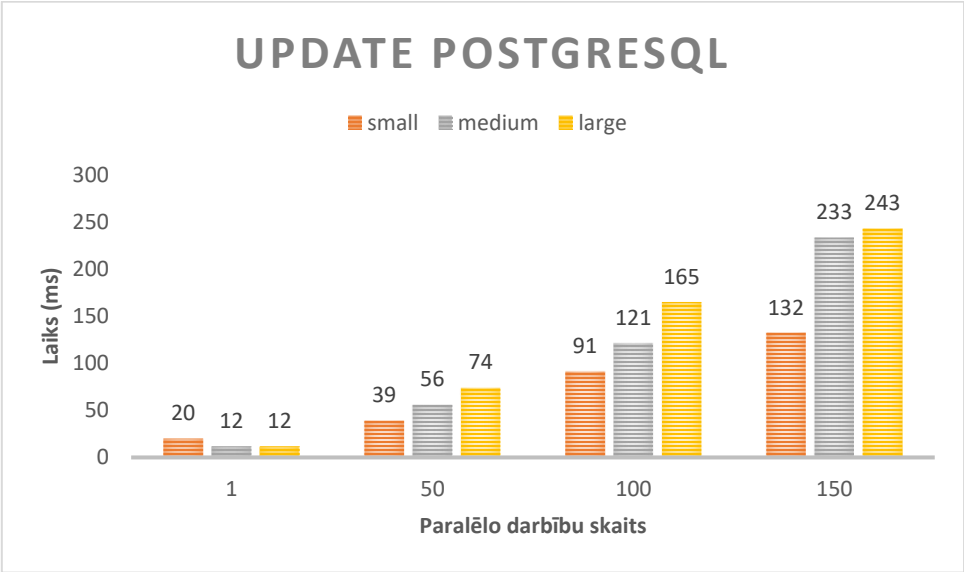
4.4.4.2. Att. MongoDB datubāzes rezultāti operācijai Select



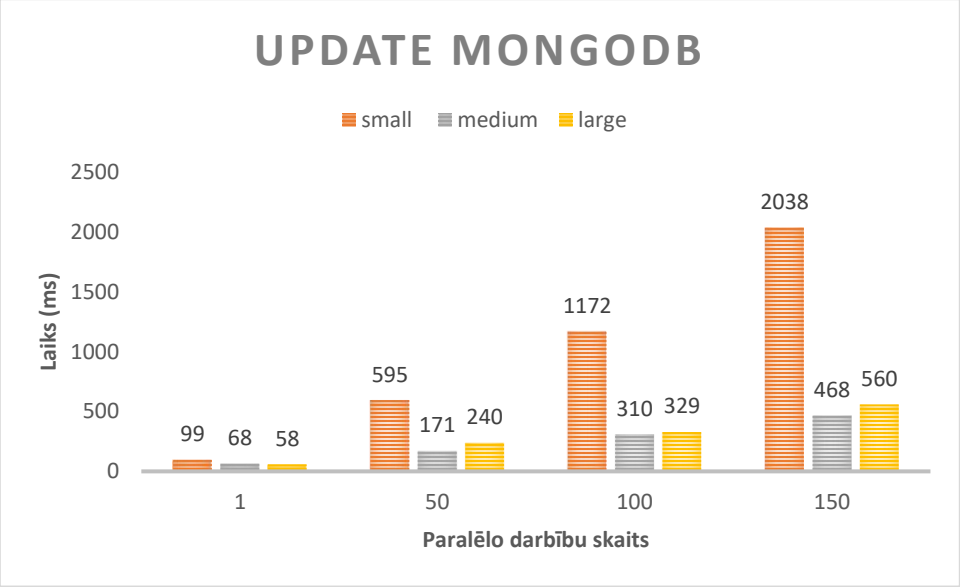
4.4.4.3. Att. Neo4j datubāzes rezultāti operācijai Select

- ***Update***

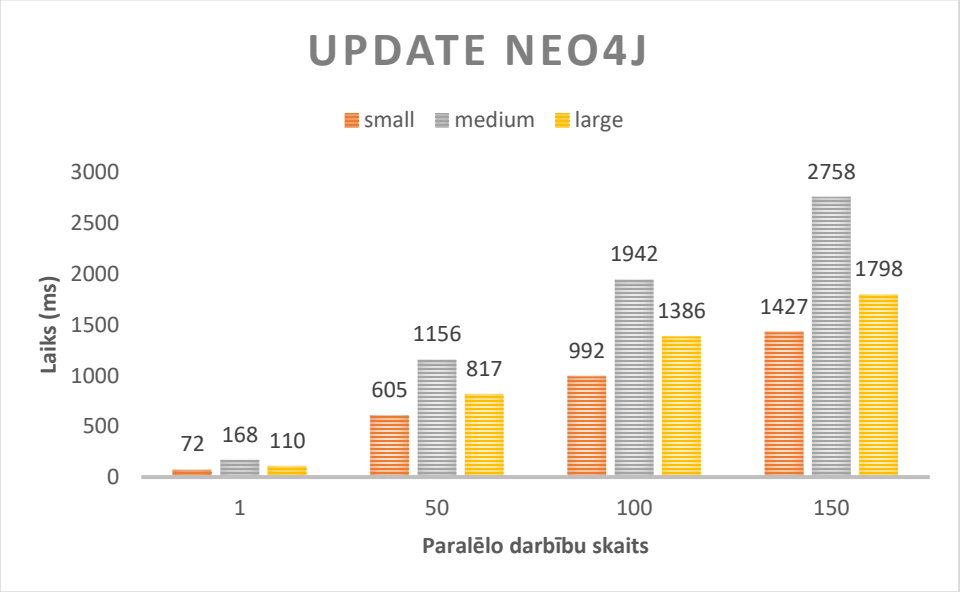
Update operācijas izpildes laiki PostgreSQL datubāzē ir viszemākie. MongoDB izpildes laiks mazam datu apjomam ir vislielākais, salīdzinot ar izpildes laiku vidējam un lielam datu apjomam. Neo4j datubāzei ir vislielākais izpildes laiks visiem datu apjomiem.



4.4.4.4. Att. PostgreSQL datubāzes rezultāti operācijai Update



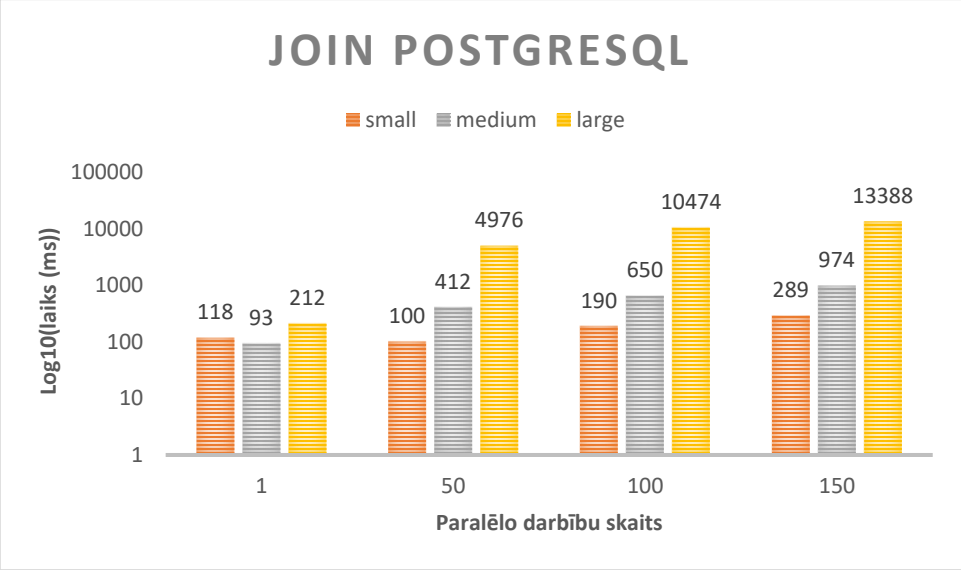
4.4.4.5. Att. MongoDB datubāzes rezultāti operācijai Update



4.4.4.6. Att. Neo4j datubāzes rezultāti operācijai Update

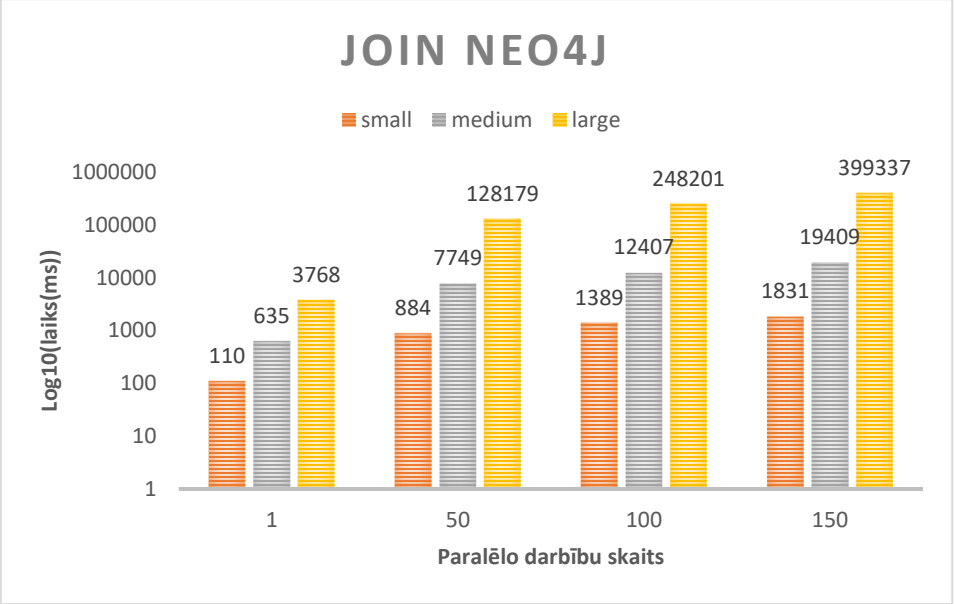
- ***Join***

Diagrammā 4.4.4.8. MongoDB datubāzei nav rezultātu lielam datu apjomam, jo katra Join darbības izpilde beidzās ar kļūdu. Papildus tam, rezultāti mazam un vidējam datu apjomam ir vislielākie.



4.4.4.7. Att. PostgreSQL datubāzes rezultāti operācijai Join

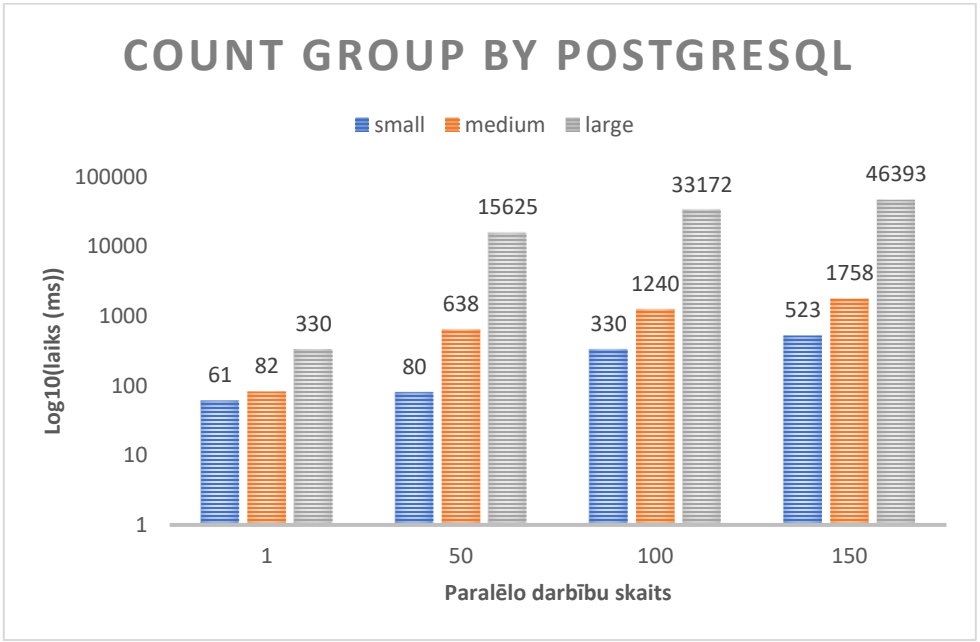
4.4.4.8. Att. MongoDB datubāzes rezultāti operācijai Join



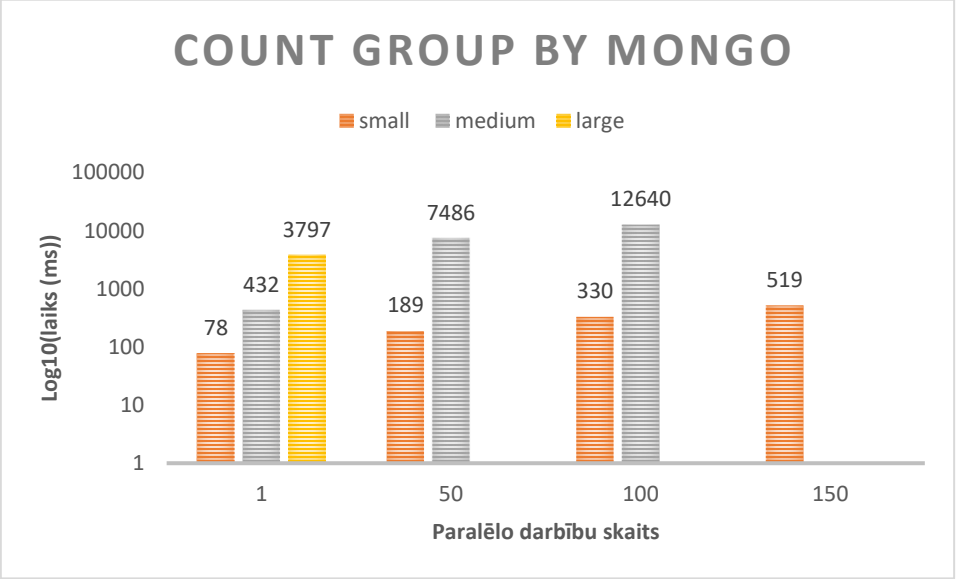
4.4.4.9. Att. Neo4j datubāzes rezultāti operācijai Join

- ***Count group by***

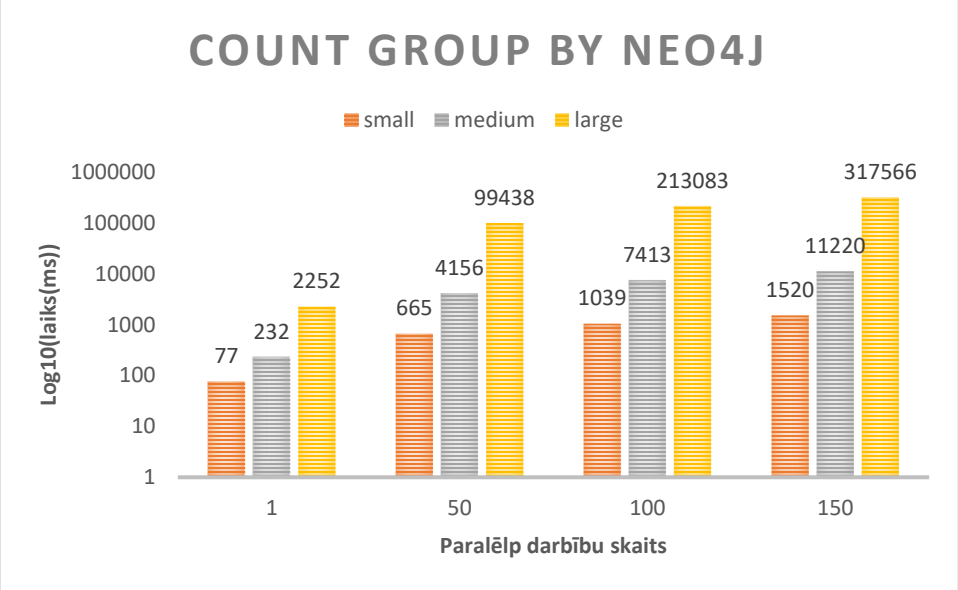
Count group by operācija MongoDB datubāzei tika veiksmīgi izpildīta tikai mazam datu apjomam kļūdas dēļ. Mazam datu apjomam MongoDB un PostgreSQL datubāzes rezultāti ir līdzīgi, bet Neo4j datubāze atkal uzrādīja visilgāko laiku vaicājumu izpildei.



4.4.4.10. Att. PostgreSQL datubāzes rezultāti operācijai Count group by



4.4.4.11. Att. MongoDB datubāzes rezultāti operācijai Count group by



4.4.4.12. Att. Neo4j datubāzes rezultāti operācijai Count group by

4.4.5. Eksperimenta rezultāti

No sistēmu veiktspējas testēšanas rezultātiem var secināt, ka relāciju datubāze PostgreSQL veic eksperimenta operācijas labāk nekā NoSQL datubāzes. Operāciju izpildes laiks bija stabils, tā pieaugums bija saistīts ar datu daudzumu tabulās un paralēlo darbību skaitu.

MongoDB testēšanas gaitā izdevās saņemt rezultātus ne visām operācijām: ***Join*** un ***Count group by*** operācijas bija pārtrauktas ar kļūdu "Java heap memory". MongoDB nav atmiņas datubāze (memory resident database). Tā liberāli izmanto ķešatmiņu, kas nozīmē, ka datu ieraksti saglabā atmiņā ātrai ieguvei, nevis diskā.

Interneta vietnē ir daudz informācijas par to, kā rīkoties, ja serverī pietrūkst atmiņas. Rezumējot, MongoDB process patērēs arvien vairāk atmiņas, jo tas aizpildīs kešatmiņu. Tāpēc cilvēki uzskata, ka, ja datoram pietrūkst atmiņas, nepieciešams ierobežot MongoDB pieejamo atmiņu, taču šī rekomendācija nav pieejama oficiālajā dokumentācijā.

MongoDB savā noklusējuma konfigurācijā ķešatmiņas izmēram izmantos lielāko no 256 MB vai ½ no (RAM — 1 GB)[9].

MongoDB ķešatmiņas lielumu ir iespējams ierobežot pievienojot argumentu cacheSizeGB "/etc/mongod.conf" konfigurācijas failam. Taču MongoDB izmanto gan iekšējo ķešatmiņu, gan failu sistēmas kešatmiņu. Tātad, ierobežojot to vienā vietā, MongoDB patērēs vairāk citā vietā. No tā iespējams secināt, ka tas nav risinājums.

NoSQL Neo4j datubāzes izpildes laika rezultāti ir vislielākie. To iespējams pamatot ar to, ka grafu datubāzes nav tik noderīgas operatīvai lietošanai, jo tās nav efektīvas liela apjoma darījumu apstrādē un tās nespēj labi apstrādāt vaicājumus, kas aptver visu datubāzi.

Džims Vēbers, grāmatas Graph Databases[24] autors, raksta: "Ir svarīgi ņemt vērā grafu datubāzu izmantošanas sekas. Vaicājuma latentums grafā ir proporcionāls tam, cik lielu daļu grafa izvēlaties izpētīt vaicājumā, un tas nav proporcionāls uzglabāto datu apjomam".

Vienkārši sakot, grafu datubāzes ļauj ātri meklēt datus, kas saistīti ar atsevišķu ierakstu (personu, produktu, vietu utt.), tomēr ir problēma - nav iespējams veikt masveida analītiskus vaicājumus visās attiecībās un ierakstos.

5. SECINĀJUMI

Bakalaura darbā tika veikta relāciju datubāzu pārvaldības sistēmu un NoSQL sistēmu analīze, sniegtas to galvenās atšķirības, priekšrocības un trūkumi. Detalizētāk ir apskatītas trīs dažādu veidu datubāzes: PostgeSQL, MongoDB un Neo4j. Darba galvenais mērķis bija salīdzināt relāciju un NoSQL DBPS, izpētīt to veiktspējas esošos pētījumus un veikt veiktspējas testēšanu.

Darba ietvaros tika sagatavota datu shēma, balsoties uz kuru tika izveidotas trīs datubāzes. Katrai no datubāzēm ir atšķirīgs izveides process, kurš tika aprakstīts dokumentā. Papildus tam, pētījumam tika sagatavoti testēšanas dati trijos apjomos - mazs, vidējs un liels.

Viens no darba uzdevumiem bija izpētīt esošos DBPS veiktspējas pētījumus un salīdzināt rezultātus. Uzdevums ir veiksmīgi pabeigts, tika izpētīti divi avoti, kuros bija veikti dažādi eksperimenti ar datubāzēm. Pētījumu rezultāti liecina, ka katrs no datubāzes tipiem uzrādīja dažādus veiktspējas rezultātus atkarībā no eksperimenta. Dažos no eksperimentiem relāciju datubāzēm bija labāks rezultāts, savukārt citos NoSQL uzrādīja labāku veiktspēju. Avotu analīze parādīja, ka, lai iegūtu precīzākus eksperimenta rezultātus, ir nepieciešams veikt plašāku salīdzinājumu, izmantojot dažādas testēšanas metodes.

Bakalaura darba galvenais uzdevums bija veikt datubāzes veiktspējas testēšanu. Pētījumam bija izvelēts DBeaver testēšanas rīks, kas tika aprakstīts dokumentā. Eksperiments parādīja, ka relāciju datubāzei ir vislabākie un stabilākie rezultāti, salīdzinot ar NoSQL datubāzēm. MongoDB izpēte netika veiksmīgi pabeigta divām operācijām vidējam un lielam datu apjomam ķešatmiņas beigšanās dēl. Neo4j datubāzei bija visilgākais izpildes laiks visam operācijām un visiem datu apjomiem.

Izmantotie avoti

1. Datubāze. Pieejams: https://www.britannica.com/technology/database [aplūkots 2022. gada 28. maijā]
2. PostgreSQL. Pieejams: https://www.postgresql.org/files/documentation/pdf/13/postgresql-13-A4.pdf [aplūkots 2022. gada 28. maijā]
3. Michael Madison, Mark Barnhill, Cassie Napier, Joy Godin - "NoSQL Database Technologies"
4. Amazon DynamoDB. Pieejams: https://aws.amazon.com/documentation/dynamodb/ [aplūkots 2022. gada 28. maijā]
5. Apache HBase. Pieejams: https://hbase.apache.org/ [aplūkots 2022. gada 28. maijā]
6. MongoDB. Pieejams: https://www.mongodb.com/ [aplūkots 2022. gada 28. maijā]
7. He Changlin "Survey on NoSQL Database Technology"
8. M.Sandeep Kumar – "Comparison of NoSQL Database and Traditional Database-An emphatic analysis"
9. MongoDB Data Model. Pieejams: https://docs.mongodb.com/manual/core/data-modeling-introduction/ [aplūkots 2022. gada 28. maijā]
10. Neo4j database documentation. Pieejams: https://neo4j.com/docs/ [aplūkots 2022. gada 28. maijā]
11. Craig S. Mullins – "The 5 Key Factors for Database Performance"
12. DSA Editorial – "NoSQL Performance Explained"
13. Craig S. Mullins – "What you need to know about database performance software". Pieejams: https://www.techtarget.com/searchdatamanagement/feature/What-you-need-to-know-about-database-performance-software#:~:text=At%20a%20high%20level%2C%20database,DBMS)%20supplies%20information%20to%20users [aplūkots 2022. gada 28. maijā]
14. Online csv generator. Pieejams: https://bfotool.com/random-csv [aplūkots 2022. gada 28. maijā]

15. Arjun Panwar – “Types of Database Management Systems”. Pieejams: https://www.c-sharpcorner.com/UploadFile/65fc13/types-of-database-management-systems/ [aplūkots 2022. gada 28. maijā]
16. Populārākie relāciju DBPS. Pieejams: https://db-engines.com/en/ranking/relational+dbms [aplūkots 2022. gada 28. maijā]
17. Populārākie dokumentorientētas DBPS. Pieejams: https://db-engines.com/en/ranking/document+store [aplūkots 2022. gada 28. maijā]
18. Populārākie grafu DBPS. Pieejams: https://db-engines.com/en/ranking/graph+dbms [aplūkots 2022. gada 28. maijā]
19. S. Sumathi, S. Esakkirajan – “Fundamentals of Relational Database Management Systems”
20. Monika Sharma, Vishal Deep Sharma, Mahesh M. Bundele “Performance Analysis of RDBMS and No SQL Databases: PostgreSQL, MongoDB and Neo4j”. Pieejams: https://www.researchgate.net/publication/344634192_Performance_Analysis_of_RDBMS_and_No_SQL_Databases_PostgreSQL_MongoDB_and_Neo4j [aplūkots 2022. gada 28. maijā]
21. Yinyi Cheng, Kefa Zhou, Jinlin Wang “Performance Analysis of PostgreSQL and MongoDB Databases for Unstructured Data”
22. DBPS. Pieejams: https://www.britannica.com/technology/database-management-system [aplūkots 2022. gada 28. maijā]
23. DBeaver. Pieejams: https://dbeaver.com/docs/wiki/Dashboards/ [aplūkots 2022. gada 28. maijā]
24. Ian Robinson, Jim Webber, Emil Eifrem “Graph databases”

Pielikumi

1. MongoDB Query Language pieprasījumi datubāzes testēšanai:

Operācija	Pieprasījums
Read	`db.artist.find({name : {$regex: "A" }});`
Update	`db.albums_genres.updateOne({ album_id: 123456 }, { $set: { genre_id: 123 } });`
Join	`db.artist.aggregate([{$lookup: {from: "albums" , localField: "artist_id", foreignField: "artist_id", as: "result"}}]);`
Count group by	`db.albums.aggregate([{"$group" : {_id:{album_id :"$album_id",released :"$released"}, count:{$sum:1}}}]);`

2. Cypher pieprasījumi datubāzes testēšanai:

Operācija	Pieprasījums
Read	`MATCH (ar:Artist) WHERE ar.name STARTS WITH 'A' RETURN ar;`
Update	`MATCH (g:Albums_Genre { album_id: '123456'}) SET g.genre_id: '123' RETURN g;`
Join	`match (ar:Artist), (al:Album) where ar.artist_id = al.artist_id return ar.artist_id, ar.name, al.album, al.released;`
Count group by	`match (al:Album) return count(al.album_id), al.album_id, al.released;`