VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Baigiamasis bakalauro darbas

Duomenų migravimas tarp SQL ir NoSQL duomenų bazių

(Data Migration Between SQL and NoSQL Databases)

Atliko: 4 kurso 2 grupės studentas
Juozas Natkevičius (parašas)
Darbo vadovas:
Partn. Doc. Liutauras Ričkus (parašas)
Recenzentas:
Prof., Dr. Rimantas Vaicekauskas (parašas)

Vilnius
2020

Turinys

Įvadas

Daugelis šiuolaikinių programų susiduria su smarkiai išaugusio duomenų kiekio problema bei atitinkamais iššūkiais, susijusiais su šių duomenų apdorojimu bei saugojimu. Ilgą laiką sėkmingai su šiomis užduotimis besitvarkiusios tradicinės reliaciniu modeliu paremtos SQL (angl. *Structured Query Language*) duomenų bazės vis dažniau pasirodo esančios nepakankamai efektyvios ypatingai didelio kiekio nestruktūrizuotų duomenų apdorojimui bei saugojimui vykdyti [RVC+15]. Nepakankamas efektyvumas susidūrus su itin didelės apimties duomenų operacijomis, pasireiškiantis sumažėjusia duomenų bazės sparta ar išaugusiais talpos poreikiais, yra viena iš pagrindinių priežasčių, leidusių išaugti NoSQL duomenų bazių paklausai ir duomenų migravimo klausimo aktualumui [WK12].

NoSQL (angl. *Not only SQL*) yra sparčiai populiarėjančių duomenų bazių rūšis, kurios pagrindinis atributas yra tas, jog skirtingai negu tradicinės reliacinės duomenų bazės, NoSQL nenaudoja lentelių kaip pagrindinių duomenų saugojimo struktūrų. Tai leidžia šioms sistemoms naudoti žymiai lankstesnes struktūras, kurių dėka užtikrinamas kur kas efektyvesnis nestruktūrizuotų, didelės apimties duomenų apdorojimas.

Darbo tikslas - pritaikyti schemos denormalizavimo metodą programos, atliekančios automatizuotą duomenų migravimą iš „MySQL“ į „MongoDB“, kūrimui. Siekiant, kad tikslas būtų pasiektas, o gautos išvados būtų naudingos sprendžiant susijusias duomenų migravimo tarp reliacinių ir NoSQL duomenų bazių problemas, keliami tokie **darbo uždaviniai**:

1. Išanalizuoti literatūroje pristatomus duomenų migravimo iš SQL į NoSQL duomenų bazes metodus.

2. Palyginti reliacinės duomenų bazės schemos migravimo į „MongoDB“ strategijas.

3. Sukurti programą, atliekančią automatinį duomenų migravimą iš „MySQL“ į „MongoDB“ pritaikant schemos denormalizavimo strategiją.

4. Atlikti realios „MySQL“ saugomos reliacinės duomenų bazės migravimą į „MongoDB“ panaudojant sukurtą programą ir įvertinti gautus rezultatus.

Pirmojoje darbo dalyje aptariami esminiai SQL ir NoSQL duomenų bazių privalumai, trūkumai ir skirtumai. Tada analizuojami mokslinėje literatūroje pateikiami duomenų migravimo metodai tarp SQL ir NoSQL duomenų bazių. Išanalizavus metodus pristatomos ir palyginamos rekomenduojamos reliacinių duomenų bazių schemų migravimo į „MongoDB“ strategijos. Sekančiame skyriuje pristatomas detalus sukurtos programos, skirtos automatizuotam duomenų migravimui iš „MySQL“ į „MongoDB“ pritaikant schemos denormalizavimo strategiją, funkcionalumo aprašymas. Tuomet sukurtos programos veikimas yra iliustruojamas migruojant realią „MySQL“ sistemoje saugomą reliacinę duomenų bazę į „MongoDB“ ir pateikiami migravimo proceso rezultatai. Pabaigoje pristatomos darbo metu gautos išvados.

1. SQL ir NoSQL duomenų bazės

SQL (angl. *Structured Query Language*) - tai programavimo kalba, skirta darbui su reliacinėse duomenų bazių valdymo sistemose saugomais duomenimis, todėl tokios sistemos dar vadinamos SQL. Šiose sistemose saugomi duomenys yra struktūrizuoti pagal reliacinį modelį. Pagal reliacinį modelį sumodeliuotoje duomenų bazėje duomenys yra išskaidomi į vieną ar daugiau lentelių. Kiekviena lentelės eilutė reprezentuoja tam tikrą įrašą, o kiekvienas stulpelis to įrašo atributus. Kiekvienas lentelės įrašas privalo turėti unikalų pirminio rakto atributą (angl. *primary key*), pagal kurį tą įrašą galima identifikuoti. Lentelės taip pat gali turėti tarpusavio sąryšius, kurie yra apibrėžiami naudojant išorinio rakto (angl. *foreign key*) atributą, rodantį į susijusios lentelės pirminį raktą. [Ora15]

Ilgą laiką reliacinės SQL duomenų bazės buvo vienareikšmiškai populiariausias būdas saugoti duomenis, tačiau augant socialinių tinklų populiarumui bei verslui keliantis į virtualią erdvę SQL populiarumas pradėjo mažėti. Pagrindinė to priežastis - sumažėjęs reliacinių duomenų bazių efektyvumas, dirbant su itin didelės apimties duomenimis, bei ribotas jų plečiamumas [WK12].

NoSQL atspindi naują požiūrį į duomenų saugojimą ir apdorojimą. NoSQL papildo reliacinį modelį ir sukuria galimybę saugoti itin didelės apimties duomenis lanksčiose duomenų struktūrose užtikrinančiose aukštą duomenų apdorojimo spartą. Vien socialinių tinklų gigantai „Facebook“ ir „Twitter“ kasdien išsaugo ir apdoroja terabaitus duomenų [Bou18]. NoSQL sistemos aukoja reliacinėse duomenų bazėse užtikrinamą vientisumą (angl. *consistency*) vardan kitų faktorių, tokių kaip prieinamumas, plečiamumas ir didesnis našumas, reikalingų dirbant su „Big Data“ duomenimis.

1.1. NoSQL tipai

Šiuo metu pagrindiniai NoSQL duomenų bazių tipai, atsižvelgiant į jose saugomų duomenų struktūrą, yra tokie [MH13]:

- Dokumentų - kiekvienas įrašas yra XML, JSON ar kt. formato objektas.
- Stulpelių - kiekvienas įrašas yra stulpelis, kurį sudaro eilutės.
- Rakto - reikšmes - kiekvienas įrašas yra reikšmė, kurią pasiekti galima pagal jai priskirtą raktą.
- Grafų - duomenys yra saugomi grafinėse struktūrose ir jų viršūnėse bei briaunose.

Kiekvienas iš skirtingų tipų pasižymi sau būdingais duomenų saugojimo bei paieškos ir duomenų vaizdavimo būdais, kurie gali nežymiai skirtis priklausomai nuo konkrečios duomenų bazių valdymo sistemos (DBVS). Duomenų bazių valdymo sistema (DBVS) - tai programinė įranga, suteikianti duomenų bazės vartotojui galimybę dirbti su duomenimis nesigilinant į technines detales. Kitaip tariant, duomenų bazių valdymo sistema atlieka vartotojo sąsajos vaidmenį su duomenų baze.

Šiame darbe pagrindinis dėmesys skiriamas dokumentų (angl. *„Document stores“*) tipo NoSQL duomenų bazėms ir duomenų bazių valdymo sistemoms. Dokumentų duomenų bazės buvo sukurtos dokumentų saugojimui ir apdorojimui. Dokumentai duomenų bazėse yra saugomi tam tikru formatu, tokiu kaip XML (*„Extensible Markup Language“*), JSON (*„Javascript Option Notation)“*), BSON (*„Binary JSON“*) ar kitu, priklausomai nuo pasirinktos DBVS [MH13]. Dokumentų duomenų bazės yra labiausiai tinkamos didelės apimties nestruktūrizuotų dokumentų rinkiniams, tokiems kaip el. laiškai, XML dokumentai, socialinių tinklų žinutės ir kt., saugoti. Jos taip pat yra vienos populiariausių tarp visų tipų NoSQL duomenų bazių.

1.2. „MySQL“ ir „MongoDB“

„MySQL“ yra viena populiariausių reliacinių DBVS, populiarumu nusileidžianti tik „Oracle“ DBVS [DbE20b]. Tai yra atviro kodo (angl. *open source*) DBVS su galimybe įsigyti įmonės (angl. *enterprise*) licenciją. „MySQL“ yra lengva, saugi ir palaikanti visą darbui su šiuolaikinėmis reliacinėmis SQL duomenų bazėmis reikalingą funkcionalumą.

„MongoDB“ yra vienareikšmiškai populiariausia ne tik tarp dokumentų, bet ir tarp visų tipų NoSQL duomenų bazių valdymo sistemų [DbE20a]. Kaip ir „MySQL“, „MongoDB“ taip pat yra atviro kodo, tačiau tik su tam tikromis panaudojimo taisyklėmis ir ribojimais. Joje duomenys yra saugomi reliacines lenteles atitinkančiose struktūrose - dokumentų kolekcijose. Kiekviena kolekcija yra sudaryta BSON formato dokumentų, kurie atitinka įrašus SQL duomenų bazėse.

Straipsnyje „A Comparative Study: MongoDB vs. MySQL“ [GPG+15] autoriai atliko tyrimą, kurio metu buvo kuriamos „MySQL“ ir „MongoDB“ duomenų bazės tiems patiems duomenims saugoti. Autorių pateikiamose išvadose pabrėžiami šie „MongoDB“ pranašumai [GPG+15]:

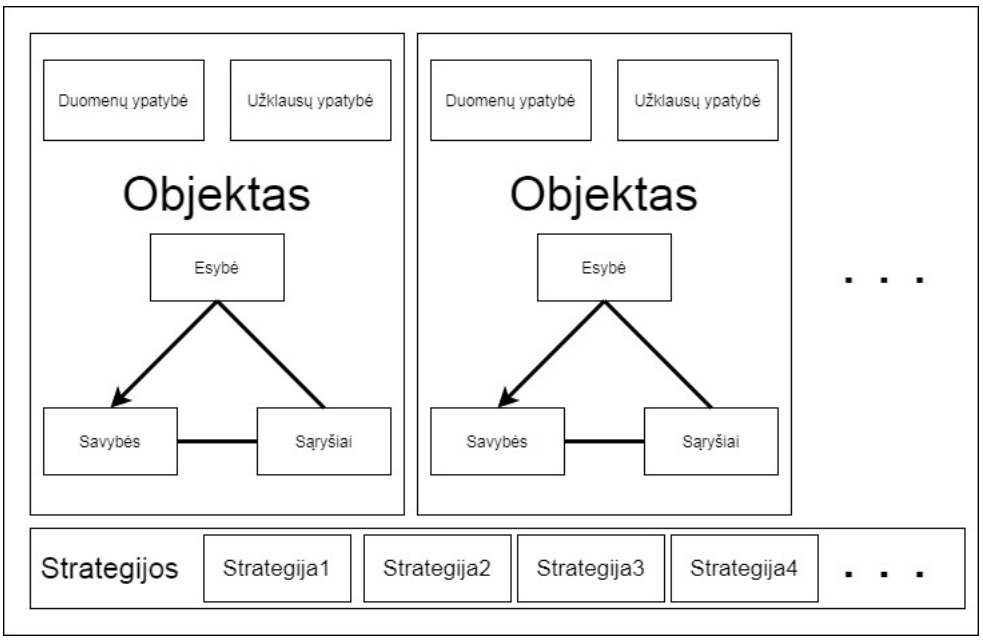
- Lankstesnė duomenų struktūra: vartotojai gali keisti dokumentų struktūrą be privalomo duomenų bazės struktūros modifikavimo.
- Spartesnis užklausų apdorojimas atliekant bazines duomenų operacijas: skaitymo (angl. *read*), rašymo (angl. *write*), trynimo (angl. *delete*) ir modifikavimo (*modify*).

2. Duomenų migravimo metodai

Šioje dalyje analizuojami trys mokslinėje literatūroje pristatomi duomenų migravimo iš reliacinių SQL į NoSQL duomenų bazes metodai bei įrankiai. Pirmajame straipsnyje aprašomas „Mid-model“, leidžiantis atlikti duomenų migravimą iš reliacinės į įvairių tipų NoSQL duomenų bazes. Antrasis metodas yra skirtas duomenų migravimui iš reliacinės į grafų tipo duomenų bazę. Trečiajame straipsnyje pateikiamas įrankio „NoSQLayer“, leidžiančio migruoti duomenis iš „MySQL“ į dokumentų tipo NoSQL DBVS „MongoDB“, aprašymas.

2.1. Duomenų migravimas tarp reliacinių ir NoSQL duomenų bazių panaudojant „Mid-model“

Straipsnyje „Mid-model Design Used in Model Transition and Data Migration between Relational Databases and NoSQL Databases“ [LLD15] autoriai pristato modelį, skirtą duomenų migravimui tarp reliacinių SQL ir NoSQL duomenų bazių. Pagrindinis šio modelio privalumas yra universalumas. Autorių teigimu, modelis suteikia galimybę atlikti duomenų migravimą iš reliacinių į skirtingų tipų NoSQL duomenų bazes.



1 pav. „Mid-model“ struktūra [LLD15].

Iliustracijoje (1 pav.) pavaizduota „Mid-model“ struktūra. Pagrindinis šio modelio elementas yra objektas. Objektą sudaro reliacinėje duomenų bazėje saugomi lentelių metaduomenys. Esybė atitinka reliacinės duomenų bazės lentelę, savybės apibrėžia lentelės turimus stulpelius, o sąryšiai - esybę su kitomis duomenų bazės esybėmis siejančius sąryšius. Sąryšiai gaunami naudojantis pirminiais ir išoriniais lentelės raktais. Kiekvienam objektui taip pat nustatomos tam tikros duomenų ir užklausų savybės. Straipsnio autorių teigimu, tai padeda išlaikyti duomenų integralumą migravimo proceso metu [LLD15]. Kiekvienas objektas gali turėti keletą duomenų ir užklausų savybių.

Duomenų savybės - tai tam tikros esybės charakteristikos, kurios gali būti svarbios atliekant NoSQL duomenų bazės struktūros modeliavimą. Pavyzdžiui, esybė gali turėti dažno įterpimo (angl. *frequent insertion*) savybę, tai reikštų, kad modeliuojant NoSQL duomenų bazės struktūra reikėtų atsižvelgti, kad su šia esybe bus atliekama daug rašymo operacijų. Kitas autorių pateikiamas pavyzdys - duomenų savybė didelis duomenų kiekis (angl. *large data size*), kuri indikuotų apie didelį esybės reliacinės duomenų bazės lentelėje esantį įrašų kiekį [LLD15]. Užklausų savybės saugo dažniausiai su objekto esybe susijusių užklausų informaciją. Ši informacija taip pat yra naudojama modeliuojant NoSQL duomenų bazės struktūrą, kadangi nusako, kokios yra dažniausiai naudojamos užklausos dirbant su kiekviena reliacinės duomenų bazės lentele.

„Mid-model“ taip pat saugo įvairias migravimo strategijas. Šios strategijos, pavaizduotos iliustracijos (1 pav.) apačioje, leidžia nustatyti optimaliausią duomenų migravimo metodą kiekvienai skirtingo tipo NoSQL duomenų bazei. Kiekvieną modelyje išsaugotą strategiją galima modifikuoti, o jei tinkamos strategijos modelyje nėra - yra galimybė pridėti naują [LLD15].“

Pilnas straipsnyje aprašomo modelio veikimo procesas susideda iš keturių etapų [LLD15]:

1. Gaunamas šaltinio duomenų bazės esybių sąryšių modelis.
2. Generuojamos šaltinio duomenų bazės duomenų ir užklausų savybės.
3. Generuojamas „Mid-model“ pagal sugeneruotas šaltinio duomenų bazės savybes ir sąryšių modelį.
4. Generuojamas NoSQL duomenų bazės modelis pagal nustatytą tinkamiausią migravimo strategiją ir atliekamas duomenų migravimas.

Apibrendinant galima teigti, kad autorių siūlomas „Mid-model“ išsiskiria savo universalumu. Modelyje aprašant įvairias migravimo strategijas galima atlikti migravimą į bet kuriuo tipo NoSQL duomenų bazę. Esminė su reliacine šaltinio duomenų baze susijusi informacija yra gaunama vienodai, nepriklausomai nuo to, į kokio tipo NoSQL duomenis norima migruoti. Ši informacija yra saugoma objektuose ir į ją patenka ne tik esybių, bet ir su esybe susijusių duomenų bei užklausų informacija. Šios informacijos pakanka pasirinkto tipo NoSQL duomenų bazės modeliui sukurti, po to seka duomenų migravimas.

2.2. Migravimas iš reliacinės į grafų duomenų bazę panaudojant „R2G“

Šiame poskiryje aprašoma straipsnyje „R2G: a Tool for Migrating Relations to Graphs“ [VMT14] pristatomo įrankio „R2G“, skirto duomenų migravimui iš reliacinės į grafų duomenų bazės valdymo sistemą, metodika bei pagrindiniai veikimo principai.

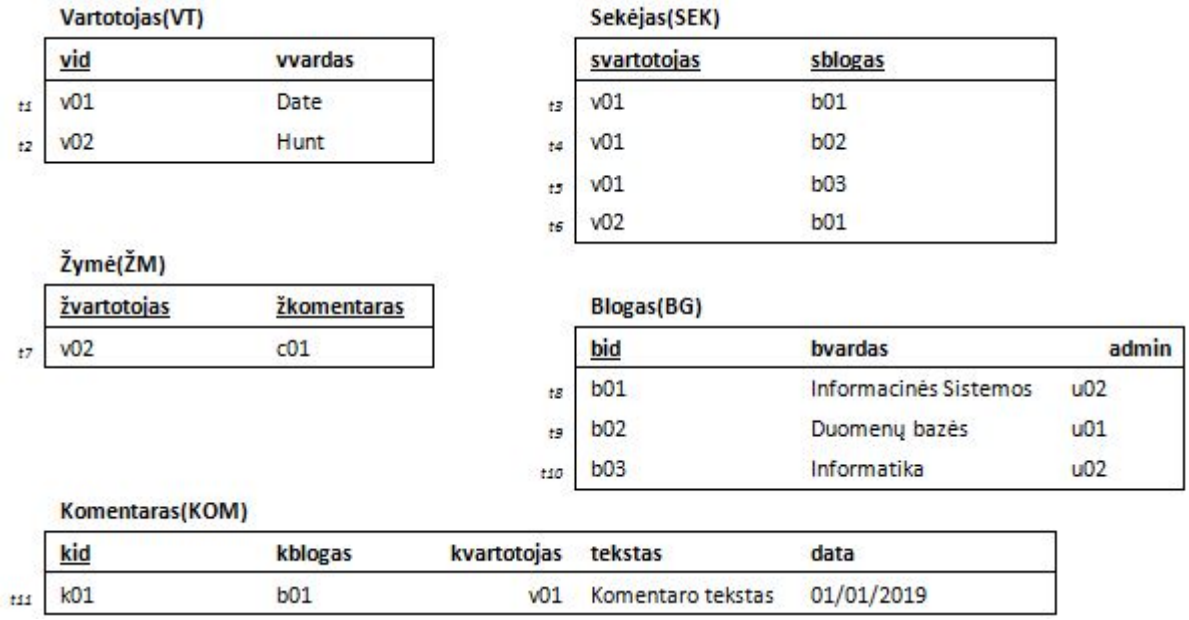
Autorių teigimu, „R2G“ įrankis susideda iš keturių pagrindinių komponentų, vykdančių tam tikras užduotis nustatyta eilės tvarka [VMT14]:

1. MA – metaduomenų analizuotojas (angl. Metadata Analyzer), atliekantis reliacinės šaltinio duomenų bazės schemos analizę ir konstruojantis atitinkamą grafo schemą.

2. DM – duomenų atvaizduotojas (angl. Data Mapper), atsakingas už duomenų atvaizdžio kūrimą pagal MA sukurtą grafo schemą.

3. QM – užklausų atvaizduotojas (angl. Query Mapper), transformuojantis reliacinės duomenų bazės užklausas į grafų duomenų bazės valdymo sistemai suprantamas užklausas pagal MA sukurtą grafo schemą.

4. GM – grafo tvarkytojas (angl. Graph Manager), kuris, pasinaudodamas DM sugeneruotu atvaizdžiu bei QM užklausomis, atlieka reliacinės duomenų bazės migravimą į grafų duomenų bazės valdymo sistemą.

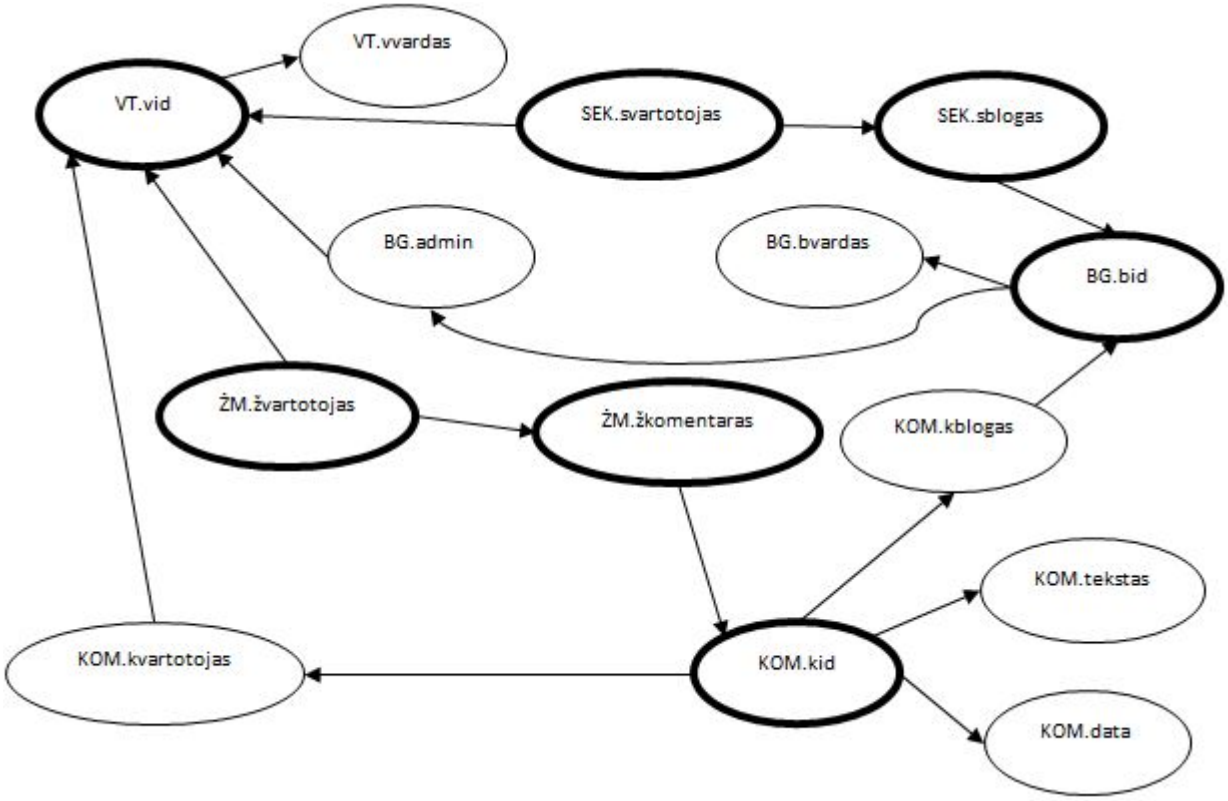


2 pav. Reliacinės duomenų bazės schema [VMT14].

Pirmajame žingsnyje atliekamas grafo schemos kūrimas yra paremtas teiginiu, kad kiekviena reliacinės duomenų bazės schema gali būti pavaizduota kaip grafas, atsižvelgiant į schemoje (8 pav.) nurodomus apribojimus bei pirminius ir išorinius raktus [VMT14]. Reliacinę schemą $\mathcal{R}$ atitinka orientuotas grafas RG (N, E), kur N – viršūnių, o E – briaunų aibės, kuriame:

- Kiekvienam $\mathcal{R}$ priklausančiam atributui yra atitinkama viršūnė $A \epsilon N$.
- Viršūnes jungia briauna (A_i, A_j) ϵ E, jeigu yra patenkinama bent viena iš duotųjų sąlygų:
  - A_i yra lentelės R esančios schemoje $\mathcal{R}$ raktas arba rakto dalis, o A_j yra ne raktinis R atributas.
  - Tiek A_i, tiek A_j priklauso tam pačiam raktui lentelėje R esančioje schemoje $\mathcal{R}$.
  - A_i priklauso R_i, o A_j priklauso R_j ir A_i bei A_j sieja išorinis raktas.

Vadovaujantis šiomis taisyklėmis, straipsnyje pateikiamas pavyzdys, kuriame pagal reliacinės duomenų bazės schemą (8 pav.) yra sukonstruojamas atitinkamas grafas, išsaugantis visus reikalingus atributus bei jų sąryšius (9 pav.).



3 pav. Reliacinės duomenų bazės schema atvaizduota grafe [VMT14].

Antrajame žingsnyje yra atliekamas reliacinių duomenų atvaizdavimas grafo struktūroje. Kitaip tariant, naudojantis pirmajame žingsnyje sukonstruotu grafu (9 pav.), atvaizduojami konkretūs duomenų kortežai (angl. *tuple*) – lentelių eilutės grafinėje struktūroje.

Trečiajame žingsnyje „R2G“ užklausų atvaizduotojas (QM) transformuoja reliacinės duomenų bazės užklausas į tokias, kurios būtų suprantamos grafų duomenų bazių valdymo sistemoms, pirmajame žingsnyje sugeneruoto grafo (9 pav.) pagalba. Pasak autorių, jų sukurtas įrankis geba sėkmingai „versti“ SQL užklausas į kelio apėjimo operacijas grafų duomenų bazėje [VMT14]. Pagal reliacines užklausas, visų pirma, yra kuriamas taip vadinamas užklausos šablonas, kuris nurodo grafo schemos dalį, kurioje yra saugoma užklausos rezultato reikšmė. Tuomet, pritaikius šį šabloną konkrečiam grafui su duomenimis, yra randami užklausos rezultatą atitinkantys duomenys.

Pabaigoje „R2G“ komponentas - grafų tvarkytojas (GM) - atlieka duomenų migravimą į adresato duomenų bazę ir vykdo visas migravimui reikalingas užklausas, pasinaudodamas ankstesniuose žingsniuose duomenų atvaizduotojo (DM) ir užklausų atvaizduotojo (QM) sukurtais atvaizdžiais ir užklausų šablonais.

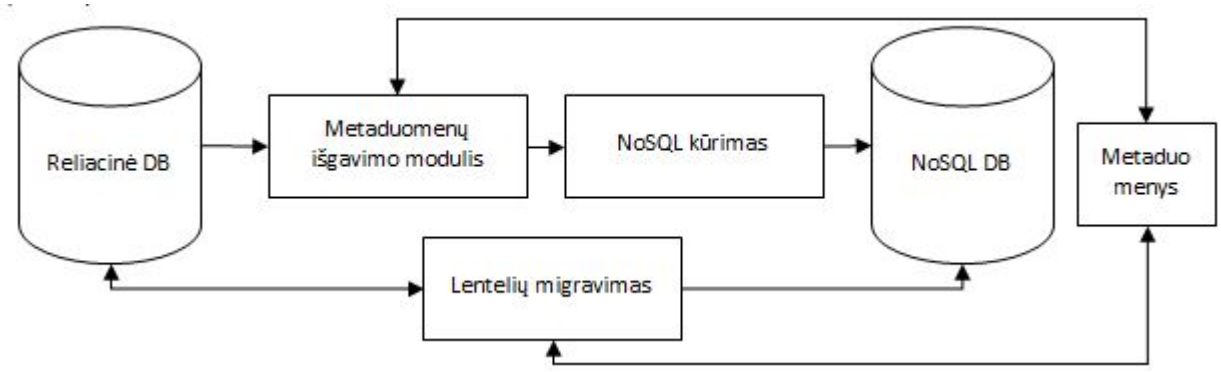
Apibendrinant, straipsnyje [VMT14] aprašomas „R2G“ įrankis, skirtas duomenų migravimui iš reliacinės į grafų duomenų bazių valdymo sistemą, veikia keturių žingsnių principu. Pirmiausia, vadovaujantis apibrėžtomis taisyklėmis, yra sugeneruojamas reliacinės duomenų bazės schemą

atitinkantis grafas, kurio viršūnės atspindi reliacinėse lentelėse saugomus atributus, o briaunos – sąryšius tarp jų. Toliau seka reliacinės duomenų bazės duomenų atvaizdavimas grafinės duomenų bazės struktūroje. Šiame žingsnyje, pasinaudojant įvedama apjungiamumo (angl. unifiability) sąvoka, atsiranda galimybė grafo viršūnėse saugoti ne vieno, o kelių susijusių duomenų kortežų duomenis. Trečiajame žingsnyje, užklausų atvaizduotojas (QM) transformuoja reliacinės duomenų bazės užklausas į grafų duomenų bazių valdymo sistemoms suprantamą formatą, kurdamas užklausų šablonus, kuriuos vėliau galima pritaikyti konkretiems duomenims grafe pasiekti. Pabaigoje grafų tvarkytojas (GM), pasitelkdamas ankstesniuose žingsniuose sugeneruota informacija, atlieka duomenų ir užklausų migravimą.

2.3. „MySQL“ migravimas į „MongoDB“ panaudojant „NoSQLayer“

Šiame poskiryje aprašomas straipsnyje „A Framework for Migrating Relational Datasets to NoSQL“ [RVC+15] pateikiamas metodas duomenų migravimui tarp reliacinės ir NoSQL duomenų bazių. Autoriai pristato sistemą „NoSQLayer“, skirtą automatizuotam ir skaidriam duomenų bei duomenų valdymo modelio migravimui tarp reliacinės „MySQL“ ir NoSQL „MongoDB“ duomenų bazių valdymo sistemų. Pagrindinė idėja, kuria remiantis buvo kuriama ši sistema, yra šaltinio duomenų bazės semantikos išsaugojimas atsižvelgiant ne vien į duomenų migravimą, bet ir užtikrinant, kad darbui su duomenimis būtų galima naudoti tas pačias, šaltinio duomenų bazėje naudojamas užklausas. Šiame skyriuje bus detaliau analizuojama pirmoji „NoSQLayer“ sistemos dalis - Duomenų migravimo modulis (angl. *Data Migration Module*).

Pagrindiniai duomenų migravimo modulio atliekami žingsniai yra automatinis šaltinio duomenų bazės elementų, tokių kaip lentelės, atributai, sąryšiai ir kt. identifikavimas, atitinkamos struktūros adresato NoSQL duomenų bazės sukūrimas bei visų duomenų migravimas. Taip pat straipsnyje pateikiama ir šio modulio veikimą iliustruojanti schema (4 pav.).



4 pav. Duomenų migravimo modulio veikimo schema [RVC+15].

Pirmiausia duomenų migravimo modulyje atliekama reliacinės duomenų bazės analizė, siekiant atrinkti metaduomenis, reikalingus migravimo procesui. Kitaip tariant, reikia identifikuoti visus duomenų bazei priklausančius elementus. Pasak straipsnio autorių, šiai užduočiai atlikti puikiai tinka „Java DatabaseMetaData“ API (angl. *Application Programming Interface*), kurį sudaro

klasės ir metodai, reikalingi metaduomenų išgavimui, o pats API gali būti sėkmingai pritaikomas skirtingoms duomenų bazių valdymo sistemoms [RVC+15]. Šio API pagrindiniai metodai ir jų aprašymai pateikiami lentelėje (lent. 1).

1 lentelė. „Java DatabaseMetaData“ API pagr. metodai [RVC+15]

Metodas	Aprašymas
getTables()	Metodas grąžina visų duomenų bazėje esančių lentelių sąrašą, kuriuo iteruojant galima gauti informaciją apie kiekvieną lentelę.
getColumns()	Metodas grąžina informaciją apie visus nurodomos lentelės atributus (vardai, tipai ir kt.) bei identifikuoja lentelės pirminius raktus.
getIndexInfo()	Metodas naudojamas gauti informaciją apie visus reliacinėje duomenų bazėje sukurtus indeksus.
getMetaData()	Metodas naudojamas kitai reikalingai informacijai, tokiai kaip vientisumo apribojimai ir duomenų tipai, gauti.

Sekantis duomenų migravimo modulio atliekamas žingsnis, surinkus informaciją apie šaltinio duomenų bazės duomenis, yra tinkamos adresato duomenų bazės struktūros sukūrimas. Straipsnyje teigiama, kad „NoSQLayer“ sistema šiuo metu palaiko duomenų migravimą į dokumentų duomenų bazės valdymo sistemą „MongoDB“. Kaip ir kiekviena dokumentų tipo NoSQL duomenų bazė, „MongoDB“ duomenys saugomi kaip dokumentų rinkiniai, todėl adresato struktūra yra kuriama kiekvieną reliacinės šaltinio duomenų bazės lentelę vaizduojant kaip dokumentą, o lentelės atributus – kaip dokumentų laukus. Taip pat kiekvienas lenteles jungiantis sąryšis „MongoDB“ yra atvaizduojamas kaip dokumento nuoroda tam, kad būtų užtikrinamas migruojamų duomenų integralumas ir būtų galima atlikti transakcijas, reliacinėje duomenų bazėje reikalaujančias lentelių jungimo (angl. *join*).

Svarbu paminėti, kad adresato duomenų bazėje saugomi ne tik iš reliacinės šaltinio duomenų bazės migruojami duomenys, tačiau yra sukuriama ir atskira kolekcija saugoti metaduomenims, išgautiems iš šaltinio duomenų bazės pirmojo žingsnio metu. Autorių teigimu, „NoSQLayer“ šią kolekciją sukuria todėl, kad metaduomenys yra svarbūs antrosios sistemos dalies – duomenų atvaizdavimo modulio veikimui, kadangi yra naudojami integralumo apribojimams bei tinkamų duomenų tipų naudojimui užtikrinti [RVC+15].

Paskutinė duomenų migravimo modulyje atliekama funkcija yra pats duomenų migravimas iš šaltinio reliacinės į adresato NoSQL duomenų bazę. Migravimui atlikti, kiekvienai reliacinei lentelei, saugomai „MySQL“ duomenų bazių valdymo sistemoje, naudojamos „SELECT * from TABLE“ užklausos, skirtos duomenims išgauti, kurie tuomet yra dedami į „MongoDB“ kaip atitinkamos dokumentų kolekcijos.

3. „MongoDB“ schemos modeliavimo strategijos

Dokumente „RDBMS to MongoDB Migration Guide“ [Mon18] aprašomi reliacinėje duomenų bazėje esančių sąryšių modeliavimo „MongoDB“ DBVS būdai. Išskiriamos dvi pagrindinės strategijos: įterpimo (angl. *embedding*) arba nuorodų (angl. *referencing*).

Įterpimo strategija yra pagrįsta schemos denormalizavimu - kelių esybių apjungimu į vieną bendrą. SQL duomenų bazių schemos yra normalizuotos siekiant užtikrinti duomenų integralumą ir išvengti duomenų dubliavimo. „MongoDB“ struktūra yra kur kas laisvesnė. Joje galima denormalizuoti schemą ir apjungti esybių duomenis naudojant įterpimo strategiją. Naudojant šią strategiją, kelių tarpusavyje susijusių reliacinių lentelių įrašai yra saugomi viename MongoDB“ dokumente. Pagrindiniai įterpimų naudojimo trūkumai yra duomenų dubliavimo atsiradimas ir dėl to išaugusi duomenų bazės apimtis bei sudėtingesnis rašymo ir modifikavimo operacijų atlikimas. Tačiau pagrindinis privalumas yra efektyvi, paprastai viena užklausa atliekama susijusių duomenų paieška [Mon18].

Nuorodų strategijos esminis siekis yra išlaikyti normalizuotos duomenų bazės schemos semantiką. SQL duomenų bazės lentelės „MongoDB“ DBVS yra modeliuojamos kaip atskirų dokumentų kolekcijos. Sąryšiams atvaizduoti dokumentuose naudojamos nuorodos į susijusius dokumentus. Nuorodos pagal savo prasmę atitinka reliacinėse duomenų bazėse naudojamus išorinius raktus. Pagrindinis nuorodų naudojimo trūkumas pasireiškia kuomet reikia gauti tarpusavyje susijusių dokumentų duomenis, nes reikalingos papildomos užklausos nuorodomis susijusiems dokumentams gauti ir tai mažina duomenų bazės efektyvumą [Mon18]. Kita vertus, nuorodų strategija užtikrina tam tikrų „MongoDB“ esančių apribojimų laikymąsi, pavyzdžiui, nėra rizikos viršyti šešiolikos megabaitų vieno dokumento dydžio limitą. Taip pat normalizuotoje duomenų bazėje efektyviau atliekamos rašymo ir modifikavimo operacijos, kadangi užtenka jas atlikti vienam dokumentui, o ne visoms denormalizavimo metu atsiradusioms kopijoms.

Dokumente pabrėžiama, kad nėra universalios schemos modeliavimo strategijos, tinkančios visoms duomenų bazėms, kadangi kiekviena jų gali turėti skirtingus sąryšių svorius, skaitymo bei rašymo operacijų tendencijas ir kitas savybes, kurios gali daryti įtaką skirtingiems efektyvumo optimizavimo poreikiams [Mon18]. Taip pat dokumente teigiama, kad daugeliu atveju optimaliausia yra naudoti įvairias schemos modeliavimo strategijų kombinacijas, kadangi tiek nuorodų, tiek įterpimų naudojimas turi savo privalumų bei trūkumų. Egzistuoja bendros rekomendacijos sąryšių modeliavimui „MongoDB“ DBVS [Mon18]. SQL duomenų bazėse galimi šie lentelių tarpusavio sąryšiai:

1. 1:1 - vienas lentelės A įrašas gali būti susijęs tik su vienu lentelės B įrašu ir atvirkščiai. Modeliuojant 1:1 sąryšį „MongoDB“ DBVS rekomenduojama naudoti įterpimo strategiją. Tokių būdų susijusių dokumentų informacijai pasiekti pakanka vienos skaitymo operacijos [Mon18].

2. 1:N - vienas lentelės A įrašas gali būti susijęs su daugiau negu vienu lentelės B įrašu, o

vienas lentelės B įrašas gali būti susijęs tik su vienu lentelės A įrašu. Modeliuojant 1:N sąryšį „MongoDB“ DBVS rekomenduojama naudoti įterpimo strategiją. Tačiau yra atvejai, kuomet šiam sąryšiui modeliuoti labiau tinkama nuorodų strategija [Mon18]:

- Dokumentas turi labai daug susijusių dokumentų arba susiję dokumentai yra itin didelės apimties. Tokiais atvejais įterpinėjant dokumentus kyla rizika viršyti 16 megabaitų dokumento limitą.
- Susijęs dokumentas yra labai retai skaitomas ar modifikuojamas, todėl jo įterpimas bereikalingai padidintų susijusio dokumento dydį.

3. N:N - vienas lentelės A įrašas gali būti susijęs su daugiau negu vienu lentelės B įrašu, o vienas B lentelės įrašas gali būti susijęs su daugiau negu vienu lentelės A įrašu. Modeliuojant N:N sąryšį „MongoDB“ DBVS rekomenduojama naudoti vienpusį (angl. *one way*) arba dvipusį (angl. *two way*) įterpimus [Mon18]. Dvipusis nuo vienpusio įterpimo skiriasi tuo, kad naudojant dvipusį įterpimą į abu dokumentus yra atitinkamai įterpiamos susijusių dokumentų reikšmės. Analogiškai 1:N, galimi atvejai kuomet N:N sąryšiui įterpimo metodas nėra optimalus. Tokiais atvejais siūloma naudoti vienpuses arba dvipuses nuorodas.

Straipsnyje „A Study of Normalization and Embedding in MongoDB“ [KGK14] aprašomas tyrimas, kuriame lyginamas normalizuotų ir denormalizuotų schemos modelių „MongoDB“ DBVS skaitymo operacijų efektyvumas. Eksperimentui atlikti buvo naudojamos dvi „MongoDB“ DBVS saugomos skirtingai sumodeliuotos duomenų bazės. Viena iš jų buvo normalizuota - sąryšiams tarp dokumentų apibrėžti buvo naudojama jungimo strategija su išoriniais raktais. Kita duomenų bazė buvo denormalizuota - dokumentai buvo kiek įmanoma labiau apjungti tarpusavyje naudojant įterpimus. Tuomet abiem duomenų bazėms buvo siunčiamos vienodos skaitymo užklausos ir matuojamas rezultatų gavimo laikas. Straipsnio autoriai išvadose teigia, kad su visomis užklausomis denormalizuota „MongoDB“ duomenų bazė užklausas vykdė stabiliai greičiau už normalizuoto modelio, dokumentų nuorodomis paremtą dokumentų duomenų bazę [KGK14].

4. Programa automatizuotam duomenų migravimui iš „MySQL“ į „MongoDB“ taikant schemos denormalizavimo strategiją

Šiame skyriuje pristatoma sukurta programa, skirta automatizuotam duomenų migravimui iš reliacinės „MySQL“ į dokumentų tipo NoSQL duomenų bazių valdymo sistemą „MongoDB“ pritaikant reliacinės schemos denormalizavimo strategiją. Ankstesniame skyriuje išnagrinėtos pagrindinės sąryšių modeliavimo strategijos, naudojamos migruojant SQL schemą į ”MongoDB”. Pagrindinis denormalizavimo strategijos, kuria paremtas aprašomos programos „MongoDB“ schemos modeliavimas, tikslas yra kiek įmanoma labiau apjungti susijusių esybių įrašus tam, kad juose saugomą informaciją būtų galima pasiekti atliekant vieną skaitymo operaciją. Šios programos duomenų migravimo strategija orientuotą į „MongoDB“ DBVS esančių duomenų skaitymo operacijų efektyvumą.

Pristatoma programa automatizuotą duomenų migravimą vykdo dviem etapais:

1. Gaunami ir išsaugomi reliacinės šaltinio duomenų bazės metaduomenys.

2. Reliacinė schema yra denormalizuojama atsižvelgiant į lenteles siejančius sąryšius ir atliekamas duomenų migravimas.

Programa parašyta „Java“ programavimo kalba, todėl prisijungimui ir darbui su „MySQL“ DBVS naudojama JDBC API (angl. *Application Programming Interface*). JDBC (angl. *Java Database Connectivity*) - tai programavimo interfeisas, aprašantis metodus, skirtus komunikavimui su įvairiomis reliacines duomenų bazes valdančiomis sistemomis. Prisijungimui ir darbui su „MongoDB“ naudojama „mongodb-3.4.3“ tvarkyklė (angl. *driver*). Sekančiuose skyriuose detaliau aprašomi kiekvieno etapo metu programos atliekami veiksmai.

4.1. Reliacinės duomenų bazės metaduomenų gavimas

Pirmas programos atliekamas veiksmas šio etapo metu yra prisijungimas prie „MySQL“ DBVS. Naudojantis JDBC API prisijungimui prie „MySQL“ DBVS naudojamas metodas *getConnection()*, kuriame nurodomas prisijungimo URL (*Uniform Resource Locator*) adresas su šaltinio duomenų bazės pavadinimu bei „MySQL“ vartotojo, turinčio skaitymo teises į tą duomenų bazę, prisijungimo vardas ir slaptažodis. Šis metodas grąžina *Connection* tipo objektą, skirtą tolimesniam darbui su duomenų baze. Sekantis veiksmas, prisijungus prie „MySQL“ DBVS, yra šaltinio duomenų bazės metaduomenų gavimas.

Metaduomenų gavimas skirstomas į du žingsnius. Pirmojo žingsnio metu gaunama bendra informacija naudojama duomenų bazę atitinkančios „Java“ klasių struktūros atmintyje sukūrimui, išsaugomi joje esančių lentelių bei stulpelių pavadinimai, sukuriami ir užpildomi atitinkamų objektų sąrašai. Metaduomenų klasių struktūra ir jų sukūrimo procesas atrodo taip:

1. Migruojamos reliacinės šaltinio duomenų bazės pavadinimas paprastai yra žinomas iš anksto. Jai sukuriamame „Java“ klasės objekte taip pat yra saugomas sąrašas visų lentelių klasės objektų, atitinkančių migruojamoje duomenų bazėje esančias lenteles. Naudojantis metodu *getMetaData()* gaunamas visų „MySQL“ esančių duomenų bazių metaduomenų rinkinys. Gautam metaduomenų rinkiniui kviečiamas metodas *getTables()* ir nurodomas duomenų bazės pavadinimas. Gaunamas konkrečios duomenų bazės lentelių rinkinys. Toliau gaunami lentelių metaduomenys (2 žingsnis).

```
class Database {
        String databaseName;
        ArrayList <Table> tables;
}
```

2. Iteruojama gautu lentelių rinkiniu ir kiekvienam lentelės įrašui sukuriamas *Table* klasės objektas, kuriam priskiriamas lentelės pavadinimas. Sukurtas objektas taip pat yra patalpinamas į bendrą duomenų bazės lentelių sąrašą. Imamas lentelės pavadinimas ir bendram metaduomenų rinkiniui kviečiamas metodas *getColumns()* nurodant lentelės ir duomenų bazės pavadinimus. Gaunamas lentelės stulpelių rinkinys. Toliau gaunami stulpelių metaduomenys (3 žingsnis). Įvykdžius paskutinę lentelių rinkinio iteraciją, pirminis metaduomenų gavimas baigiamas.

```
class Table {
        String tableName;
        ArrayList <Column> columns;
        ArrayList <Column> primaryKeys;
        HashMap <String, Column> importedKeys;
        HashMap <String, Column> exportedKeys;
}
```

3. Iteruojama gautu rinkiniu ir kiekvienam stulpelių įrašui sukuriamas *Column* klasės objektas, kuriam priskiriamas stulpelio pavadinimas ir tėvinės lentelės objektas. Sukurtas objektas taip pat yra patalpinamas į bendrą tėvinės lentelės stulpelių sąrašą. Įvykdžius paskutinę stulpelių rinkinio iteraciją, grįžtama į 2 žingsnį.

```
class Column {
        Table parentTable;
        String columnName;
        Boolean primaryKey;
}
```

Antrojo žingsnio metu gaunama informacija apie lentelių pirminius, importuotus ir eksportuotus raktus. Importuotas raktas - tai lentelės išorinis (angl. *external*) raktas, susietas su kitos lentelės

pirminiu (angl. *primary*) raktu. Eksportuotas raktas - tai lentelės pirminis raktas, su kuriuo susietas kitos lentelės išorinis raktas. Informacijai apie raktus gauti iteruojama pirmajame žingsnyje sudarytu duomenų bazės lentelių sąrašu. Bendram metaduomenų rinkiniui kviečiami šie metodai perduodant lentelės pavadinimą:

1. *getPrimaryKeys* - gaunami lentelės pirminių raktų stulpelių pavadinimai. Pagal pavadinimą atrenkamas lentelės stulpelis pažymimas kaip pirminis raktas ir įdedamas į tėvinės lentelės pirminių raktų sąrašą.

2. *getImportedKeys* - gaunami lentelės importuotų raktų stulpelių bei susietų stulpelių ir jų lentelių pavadinimai. Kiekvienas importuoto rakto ir su juo susieto stulpelių poros įdedamos į lentelės importuotų raktų struktūrą.

3. *getExportedKeys* - gaunami lentelės eksportuotų raktų stulpelių bei susietų stulpelių ir jų lentelių pavadinimai. Kiekvienas eksportuoto rakto ir su juo susieto stulpelių poros įdedamos į lentelės eksportuotų raktų struktūrą.

Atlikus abu žingsnius, reliacinės šaltinio duomenų bazės metaduomenų gavimo etapas baigiamas. Surinktų duomenų pakanka pradėti migravimą į „MongoDB“ DBVS atliekant reliacinės schemos denormalizavimą pagal lentelių tarpusavio sąryšius.

4.2. Duomenų migravimas denormalizuojant reliacinę schemą

Šiame etape programa atlieka reliacinės šaltinio duomenų bazės schemos ir duomenų migravimą į „MongoDB“. Procesas vykdomas tokia tvarka:

1. Sukuriamos jungtys su „MySQL“ ir „MongoDB“ DBVS. Prisijungimui prie „MySQL“ naudojamas ankstesniame skyriuje minėtas metodas *getConnection()*. Prie „MongoDB“ prisijungiama kviečiant klasės *MongoClient* konstruktorių ir perduodant prisijungimo URI (*Uniform Resource Identifier*) adresą. Tuomet sukurtam *MongoClient* objektui kviečiame metodą *getDatabase* ir nurodome duomenų bazės pavadinimą. Jeigu „MongoDB“ sistemoje tokia duomenų bazė neegzistuoja, automatiškai sukuriama nauja duomenų bazė ir grąžinamas darbui su ja skirtas objektas *MongoDatabase*. Sukūrus jungtis su „MySQL“ ir „MongoDB“, galima pradėti duomenų migravimo procesą.

2. Iteruojama pirmajame programos veikimo etape sukurtu duomenų bazės lentelių sąrašu *tables*. Analizuojami kiekvienoje iteracijoje gaunamos lentelės sąryšiai (baigus iteravimą migravimas laikomas pabaigtu ir programa baigia darbą):

   (a) Jeigu lentelė neturi importuotų ir eksportuotų raktų (*importedKeys* ir *exportedKeys* struktūros yra tuščios), tai jos įrašams migruoti kuriama nauja „MongoDB“ kolekcija (angl. *collection*) (3 žingsnis).

(b) Jeigu lentelė turi importuotų, tačiau neturi eksportuotų raktų, tai naujos „MongoDB“ kolekcijos nekuriame, nes lentelės įrašai bus įterpiami susijusių lentelių dokumentuose. Grįžtama į 2 žingsnį.

(c) Jeigu lentelė neturi importuotų, tačiau turi eksportuotų raktų, tai jos įrašams migruoti kuriama nauja „MongoDB“ kolekcija (3 žingsnis).

(d) Jeigu lentelė turi ir importuotų, ir eksportuotų raktų, tai jos įrašams migruoti kuriama nauja „MongoDB“ kolekcija (3 žingsnis), **išskyrus** atvejį, kai importuotų raktų yra daugiau negu 2 ir jie sutampa su lentelės pirminiais raktais - tai reiškia, kad lentelė yra jungiamoji, skirta N:N sąryšiui tarp kitų lentelių apibrėžti. Jungiamosioms lentelėms naujos „MongoDB“ kolekcijos nekuriamos (grįžtama į 2 žingsnį), nes N:N sąryšiu susijusių lentelių įrašams migruoti naudojama dvipusio įterpimo strategija.

3. Kviečiamas metodas *getCollection()* su parametru *tableName*, nurodančiu lentelės pavadinimą. Jeigu tokios kolekcijos „MongoDB“ duomenų bazėje nėra, sistema automatiškai ją sukuria. Sukuriama SQL užklausa (angl. *statement*) visiems lentelės įrašams gauti:

   ```
   SELECT * FROM tableName
   ```

   Gaunamas užklausos rezultato duomenų rinkinys (angl. *Result Set*). Jeigu jame nėra įrašų, grįžtama į 2 žingsnį. Jeigu duomenų rinkinys nėra tuščias, einama į 4 žingsnį.

4. Sukuriamas sąrašas *Document* tipo objektams, atitinkantiems „MongoDB“ BSON dokumentus, saugoti. Iteruojama lentelės įrašų duomenų rinkiniu:

   (a) Kiekviena gauto lentelės įrašo stulpelio pavadinimo ir reikšmės pora metodu *put()* pridedama į įrašui sukurtą dokumento objektą. Tikrinama, ar lentelės eksportuotų raktų struktūroje yra elementas, kurio raktinė reikšmė atitinka stulpelio pavadinimą. Jeigu tokių elementų yra, tuomet kiekvienam jų gaunama eksportuotų raktų struktūros reikšmės *Column* tipo objekto tėvinė lentelė *expToTable* ir tikrinama:

      i. Jeigu *expToTable* lentelės eksportuotų raktų sąrašas yra tuščias, vadinasi susiję šios lentelės įrašai bus įterpiami į einamąjį dokumentą. Kuriama SQL užklausą susijusiems įrašams gauti:
      ```
      SELECT * FROM expToTableName WHERE expToColumnName =
      columnValue
      ```
      Čia *expToColumnName* - stulpelio, į kurį eksportuotas raktas, pavadinimas, *columnValue* - einamojo lentelės įrašo stulpelio reikšmė.
      Gaunamas susijusių įrašų sąrašas, kuriuo iteruojant visos visų stulpelių, išskyrus einamojo, pavadinimų ir reikšmių poros dedamos į susijusių įrašų sąrašą. Iteravimo pabaigoje sąrašas įdedamas į einamosios lentelės įrašui sukurtą „MongoDB“ dokumentą, nurodant *expToTable* lentelės pavadinimą.

ii. Jeigu *expToTable* lentelės eksportuotų raktų sąrašas nėra tuščias ir lentelė nėra N:N jungiamoji, vadinasi einamajame dokumente bus sukuriama nuoroda į susijusius šios lentelės įrašus. Kuriama SQL užklausą susijusiems įrašams gauti:
`SELECT expToTablePrimaryKeys FROM expToTableName WHERE expToColumnName = columnValue`
Čia *expToTablePrimaryKeys* - lentelės *expToTable* pirminiai raktai.
Gaunamas susijusių įrašų sąrašas, kuriuo iteruojant visos visų stulpelių, išskyrus einamojo, pavadinimų ir reikšmių poros dedamos į susijusių įrašų sąrašą. Iteravimo pabaigoje sąrašas pridedamas prie einamosios lentelės įrašui sukurto „MongoDB“ dokumento kaip nuoroda, nurodant *expToTable* lentelės pavadinimą.

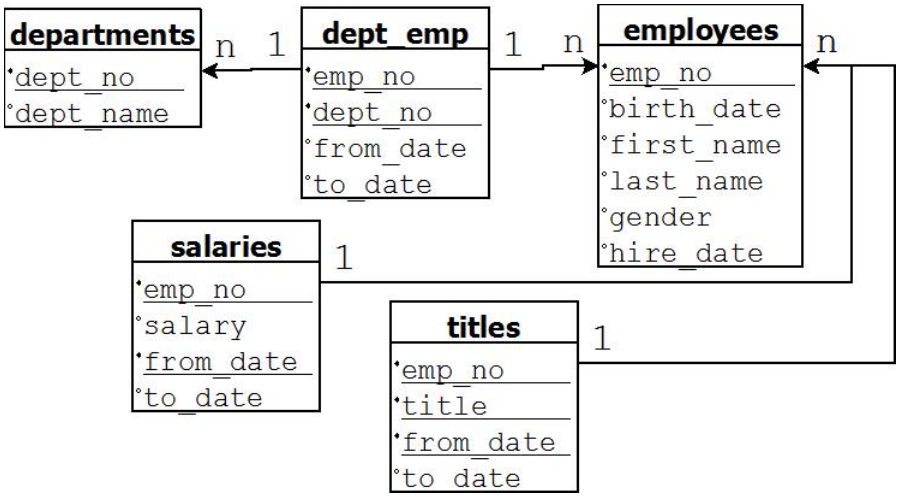
iii. Jeigu *expToTable* lentelė yra N:N jungiamoji, vadinasi į einamąjį dokumentą reikalinga įterpti kitą N:N sąryšio pusę žyminčios lentelės susijusius įrašus. Šiam tikslui pasiekti naudojamas kitas *expToTable* lentelės išorinis raktas, kurio pavadinimas nelygus einamojo stulpelio pavadinimui. Jis gaunamas iš jungiamosios lentelės importuotų raktų struktūros. Kuriama SQL užklausą susijusiems įrašams gauti:
`SELECT * FROM expToTableName INNER JOIN otherForeignKeyParentTable ON expToTableName.expToColumnName = otherForeignKeyParentTable.otherForeignKeyColumnName WHERE columnName = columnValue`
Čia *otherForeignKeyParentTable* - kitą N:N sąryšio pusę žyminčios lentelės pavadinimas, *otherForeignKeyColumnName* - kitą N:N sąryšio pusę žyminčios lentelės eksportuoto rakto stulpelio pavadinimas.
Gaunamas susijusių įrašų sąrašas, kuriuo iteruojant visos visų stulpelių, išskyrus einamojo, pavadinimų ir reikšmių poros dedamos į susijusių įrašų sąrašą. Iteravimo pabaigoje sąrašas pridedamas prie einamosios lentelės įrašui sukurto „MongoDB“ dokumento kaip nuoroda, nurodant *expToTable* lentelės pavadinimą.

(b) Baigus iteruoti per gauto lentelės įrašo stulpelius užpildytas „MongoDB“ dokumentas įdedamas į kolekcijos dokumentų sąrašą. Apdorojus paskutinį gautą įrašą, einama į 5 žingsnį.

5. Kiekvienai sukurtai „MongoDB“ kolekcijai kuriamas indeksas, atitinkantis reliacinės lentelės pirminių raktų laukus. Kuriamas indeksas su parametru *unique*, o tai reiškia, jog į kolekciją nebus galima patalpinti dokumento, kuriame bus aptikta pasikartojanti pirminį raktą reliacinėje lentelėje atitinkančio lauko reikšmė. Taip pat tokiu būdu padidinamas nuorodomis susietų dokumentų paieškos duomenų bazėje efektyvumas, kadangi nuorodos dokumentuose sukuriamos panaudojant lentelės pirminių raktų laukus. Sukūrus indeksą, grįžtama į 2 žingsnį.

4.3. Programos taikymas realios SQL duomenų bazės migravimui į „MongoDB“

Siekiant įvertinti sukurtos programos funkcionalumą, buvo atliekami realių „MySQL“ DBVS saugomų reliacinių duomenų bazių migravimo testai į „MongoDB“. Testavimui buvo naudojamos viešai prieinamos bandomosios reliacinės duomenų bazės, sudarytos iš ne daugiau kaip 10 lentelių, susietų įvairiais sąryšiais. Šiame skyriuje pristatomas SQL duomenų bazės „employees“ migravimo į „MongoDB“ procesas ir rezultatai panaudojant ankstesniame skyriuje aprašytą programą.



5 pav. SQL duomenų bazės „employees“ esybių sąryšių diagrama.

Iš pateiktos esybių diagramos (5 pav.) matoma, kad duomenų bazę „employees“ sudaro penkios lentelės, tarpusavyje apibrėžtos sąryšiais:

1. Vienas lentelės „employees“ įrašas gali būti susietas su daugiau nei vienu lentelės „salaries“ įrašu, o atitinkamai daugiau nei vienas „salaries“ įrašas gali būti susietas su ne daugiau kaip vienu „employees“ įrašu, todėl sąryšis yra 1:N.

2. Vienas lentelės „employees“ įrašas gali būti susietas su daugiau nei vienu lentelės „titles“ įrašu, o atitinkamai daugiau nei vienas „titles“ įrašas gali būti susietas su ne daugiau kaip vienu „employees“ įrašu, todėl sąryšis yra 1:N.

3. Daugiau nei vienas lentelės „employees“ įrašas gali būti susietas su daugiau nei vienu lentelės „departments“ įrašu, o atitinkamai daugiau nei vienas „departments“ įrašas gali būti susietas su daugiau nei vienu „employees“ įrašu. Lenteles siejantis sąryšis N:N normalizuotas naudojant jungiamąją lentelę „dept_emp“ ir iškaidant N:N į du 1:N sąryšius tarp lentelių „departments“ ir „dept_emp“ bei „employees“ ir „dept_emp“.

Sukurta programa automatizuotam duomenų migravimui iš „MySQL“ į „MongoDB“ DBVS, taikant schemos denormalizavimo strategiją, vykdo duomenų bazės „employees“ migravimą skyriuose 4.1. ir 4.2. detaliai aprašytais etapais.

Visų pirma, programa atlieka prisijungimą prie „MySQL“ DBVS ir visų joje saugomų duomenų bazių metaduomenų rinkinio gavimą. Sukuriamas *Database* klasės objektas duomenų bazei „employees“. Pasinaudojant gautu rinkiniu kviečiamas metodas *getTables()* ir nurodomas duomenų bazės pavadinimas „employees“. Iteruojama gautu „employees“ duomenų bazės lentelių metaduomenų rinkiniu. Kiekvienoje iteracijoje sukuriamas lentelę atvaizduojantis *Table* klasės objektas, jame išsaugomas lentelės pavadinimas, o pats objektas įtraukiamas į duomenų bazės lentelių objektų sąrašą. Kiekvienam lentelės metaduomenų rinkinio elementui kviečiame metodą *getColumns()*. Iteruojama gautu rinkiniu ir kiekvienoje iteracijoje sukuriamas lentelės stulpelį atvaizduojamas *Column* tipo objektas, kuriam suteikiamas pavadinimas, priskiriamas tėvinę lentelę atitinkantis objektas, o sukurtas stulpelio objektas įtraukiamas į jos stulpelių objektų sąrašą.

Toliau atliekama antra 4.1. skyriuje aprašyta duomenų bazės metaduomenų gavimo dalis - renkama informacija apie kiekvienos lentelės pirminius, importuotus (išorinius) ir eksportuotus raktus. Ši informacija gaunama iteruojant sudarytu lentelių objektų sąrašu ir bendram duomenų bazės metaduomenų rinkiniui kviečiant skirtingus metodus, skirtus atitinkamų tipų raktams gauti. Gauti pirminių raktų pavadinimus atitinkantys stulpelių objektai išsaugomi lentelės pirminių raktų sąraše, o importuotų ir eksportuotų raktų duomenys jiems skirtose raktų - reikšmių poras saugančiose struktūrose.

Sekantis programos atliekamas veiksmas - 4.2. skyriuje aprašomo duomenų migravimo, denormalizuojant reliacinės „employees“ duomenų bazės schemą, įgyvendinimas. Programa sukuria jungtis su „MySQL“ ir „MongoDB“ DBVS. „MongoDB“ sukuriama nauja duomenų bazė „employees.“ Iteruojama turimu duomenų bazės lentelių objektų sąrašu:

1. Pirmoji lentelė sąraše - „departments“. Ši lentelė neturi importuotų, tačiau turi vieną eksportuotą raktą (pirminio rakto stulpelis „dept_no“ eksportuotas į lentelę „dept_emp“ kaip išorinis raktas „dept_no“), todėl lentelei „departments“ kuriama nauja tokio paties pavadinimo „MongoDB“ kolekcija (2.c.):

   (a) Sukuriama ir įvykdoma SQL užklausa lentelės „departments“ įrašams gauti.
   `SELECT * FROM departments`

   (b) Sukuriamas sąrašas *Document* tipo objektams saugoti. Iteruojama gautu užklausos rezultatų rinkiniu.

   (c) Sukuriamas *Document* tipo objektas gautam įrašui saugoti. Stulpelio „dept_no“ pavadinimas ir reikšmė įraše dedami į sukurtą dokumentą.

   (d) Lentelės „departments“ eksportuotų raktų struktūroje aptinkamas elementas, kurio raktinė reikšmė yra „dept_no“. Raktą struktūroje atitinkanti reikšmė - stulpelio objektas „dept_no“, kurio tėvinė lentelė yra „dept_emp“. Kadangi „dept_emp“ yra jungiamoji N:N sąryšio lentelė, gaunama kita jos importuotų raktų struktūroje saugoma pora, kurios raktinė reikšmė „emp_no“ atitinka stulpelio objektą „emp_no“ rodantį į tėvinę lentelę „employees“ (4.iii.). Sukuriama ir įvykdoma SQL užklausa lentelių „dept_emp“ ir „employees“ duomenims, susijusiems su einamuoju lentelės „departments“ įrašu:

`SELECT * FROM dept_emp INNER JOIN employees ON dept_emp.emp_no = employees.emp_no WHERE dept_no=colVal`

Čia *colVal* - einamojo įrašo stulpelio „dept_no“ reikšmė.

(e) Iteruojama gaunamu susijusių įrašų sąrašu. Kiekvieno stulpelio pavadinimo ir reikšmės pora, išskyrus einamąjį (kadangi jo reikšmė jau įrašyta į sukurtą „MongoDB“ dokumentą), dedama į susijusio įrašo dokumentą, kuris atitinkamai yra dedamas į susijusių įrašų dokumentų sąrašą. Baigus susijusių įrašų iteravimą, užpildytas susijusių dokumentų sąrašas pridedamas prie einamojo dokumento su reikšme „employees“.

(f) Imama sekančio stulpelio „dept_name“ pavadinimas ir reikšmė įraše. Jų pora dedama į sukurtą dokumentą. Eksportuotų raktų struktūroje elemento, kurio raktinė reikšmė yra „dept_name“, nėra.

(g) Kiekvienos iteracijos pabaigoje sukurtas lentelės „departments“ įrašą atitinkantis dokumentas su įterptais „employees“ duomenimis įkeliamas į bendrą „departments“ kolekciją „MongoDB“ sistemoje.

(h) Galiausiai kolekcijai „departments“ „MongoDB“ sistemoje sukuriamas indeksas „PrimaryKey“, kuriame nurodyti visi lentelės pirminių raktų stulpelių pavadinimai. Iš SQL į „MongoDB“ migruoto lentelės „departments“ įrašo dokumento, kuriame įterpta ir susijusių „employees“ įrašų informacija, pavyzdys pateikiamas iliustracijoje (6 pav.).


```
{
    "_id" : ObjectId("5ec9b82f384a5a18ec66cff5"),
    "dept_no" : "d009",
    "employees" : [
        {
            "emp_no" : "10011",
            "from_date" : "1990-01-22",
            "to_date" : "1996-11-09",
            "birth_date" : "1953-11-07",
            "first_name" : "Mary",
            "last_name" : "Sluis",
            "gender" : "F",
            "hire_date" : "1990-01-22"
        },
        {
            "emp_no" : "10038",
            "from_date" : "1989-09-20",
            "to_date" : "9999-01-01",
            "birth_date" : "1960-07-20",
            "first_name" : "Huan",
            "last_name" : "Lortz",
            "gender" : "M",
            "hire_date" : "1989-09-20"
        },
    ],
    "dept_name" : "Customer Service"
}
```


6 pav. Į „MongoDB“ migruoto lentelės „departments“ įrašą, į kurį buvo įterptas susijęs „employees“ lentelės įrašas, atitinkančio dokumento pavyzdys.

2. Antroji lentelė sąraše - „dept_emp“. Lentelė turi ir importuotų, ir eksportuotų raktų, tačiau ji taip pat yra ir jungiamoji sąryšio N:N lentelė, todėl naują „MongoDB“ kolekcija nekuriama (2.d.)

3. Trečioji lentelė sąraše - „employees“. Ši lentelė neturi importuotų, tačiau turi tris eksportuotus raktus (pirminio rakto stulpelis „emp_no“ eksportuotas į lenteles „dept_emp“, „salaries“ ir „titles“ kaip išorinis raktas „emp_no“), todėl lentelei „employees“ kuriama nauja tokio paties pavadinimo „MongoDB“ kolekcija (2.c.):

   (a) Sukuriama ir įvykdoma SQL užklausa lentelės „departments“ įrašams gauti.
   `SELECT * FROM employees`

   (b) Sukuriamas sąrašas *Document* tipo objektams saugoti. Iteruojama gautu užklausos rezultatų rinkiniu.

   (c) Sukuriamas *Document* tipo objektas gautam įrašui saugoti. Stulpelio „emp_no“ pavadinimas ir reikšmė įraše dedami į sukurtą dokumentą.

   (d) Lentelės „employees“ eksportuotų raktų struktūroje aptinkamas elementas, kurio raktinė reikšmė yra „emp_no“. Raktą struktūroje atitinkanti reikšmė - stulpelio objektas „emp_no“, kurio tėvinė lentelė yra „dept_emp“. Kadangi „dept_emp“ yra jungiamoji N:N sąryšio lentelė, gaunama kita jos importuotų raktų struktūroje saugoma pora, kurios raktinė reikšmė „dept_no“ atitinka stulpelio objektą „dept_no“ rodantį į tėvinę lentelę „departments“ (4.iii.). Vykdomas analogiškas, lentelės „departments“ migravime aprašytas procesas.

   (e) Lentelės „employees“ eksportuotų raktų struktūroje aptinkamas dar vienas elementas, kurio raktinė reikšmė yra „emp_no“. Raktą struktūroje atitinkanti reikšmė - stulpelio objektas „emp_no“, kurio tėvinė lentelė yra „salaries“. Kadangi lentelės „salaries“ eksportuotų raktų sąrašas yra tuščias, vadinasi susiję šios lentelės įrašai bus įterpiami į einamąjį dokumentą (4.i.).

   (f) Sukuriama ir įvykdoma SQL užklausa susijusiems lentelės „salaries“ įrašams gauti.
   `SELECT * FROM salaries WHERE emp_no = colValue`

   (g) Sukuriamas sąrašas *Document* tipo objektams saugoti. Iteruojama gautu užklausos rezultatų rinkiniu.

   (h) Iteruojama gaunamu susijusių įrašų sąrašu. Kiekvieno stulpelio pavadinimo ir reikšmės pora, išskyrus einamąjį (kadangi jo reikšmė jau įrašyta į sukurtą „MongoDB“ dokumentą), dedama į susijusio įrašo dokumentą, kuris atitinkamai yra dedamas į susijusių įrašų dokumentų sąrašą. Baigus susijusių įrašų iteravimą, užpildytas susijusių dokumentų sąrašas pridedamas prie einamojo dokumento su reikšme „salaries“.

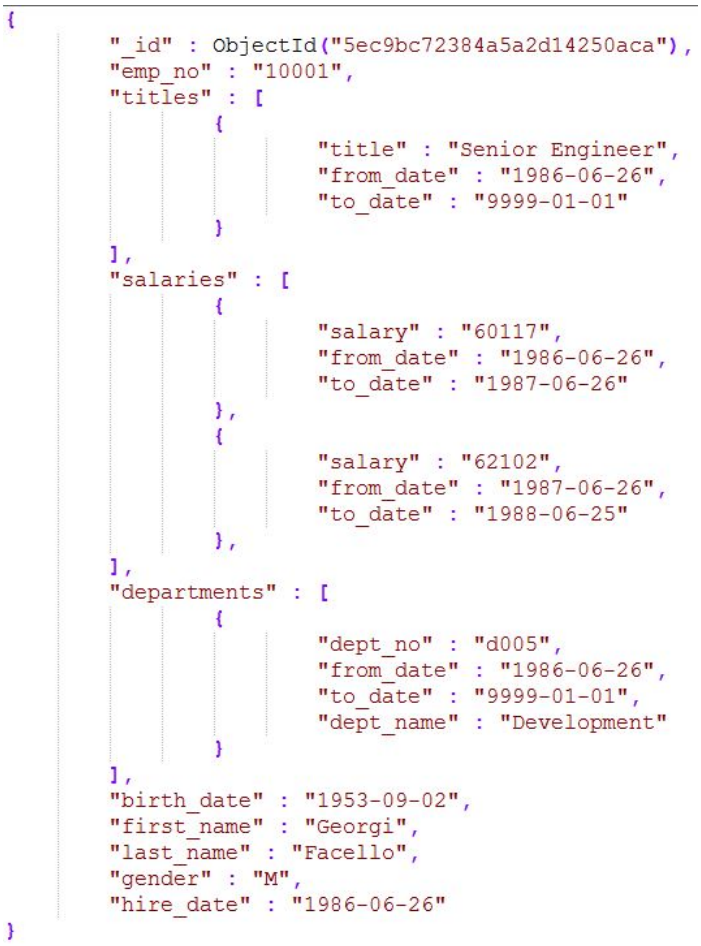
   (i) Lentelės „employees“ eksportuotų raktų struktūroje aptinkamas dar vienas elementas, kurio raktinė reikšmė yra „emp_no“. Raktą struktūroje atitinkanti reikšmė - stulpe-

lio objektas „emp_no“, kurio tėvinė lentelė yra „titles“. Kadangi lentelės „titles“ eksportuotų raktų sąrašas yra tuščias, vadinasi susiję šios lentelės įrašai bus įterpiami į einamąjį dokumentą (4.i.). Atliekami analogiški veiksmai kaip ir su susijusia lentele „salaries“.

(j) Imamos sekančio stulpelių „birth_date“, „first_name“, „last_name“, „gender“ ir „hire_date“ pavadinimai ir reikšmės įraše. Jų poros dedama į sukurtą dokumentą. Eksportuotų raktų struktūroje elemento, kurio raktinė reikšmė yra kuris nors iš minėtų stulpelių, nėra.

(k) Kiekvienos iteracijos pabaigoje sukurtas lentelės „employees“ įrašą atitinkantis dokumentas su įterptais „departments“, „salaries“ ir „titles“ duomenimis įkeliamas į bendrą „employees“ kolekciją „MongoDB“ sistemoje.

(l) Galiausiai kolekcijai „employees“ „MongoDB“ sistemoje sukuriamas indeksas „PrimaryKey“, kuriame nurodyti visi lentelės pirminių raktų stulpelių pavadinimai. Iš SQL į „MongoDB“ migruoto lentelės „employees“ įrašo dokumento, kuriame įterpta ir susijusių „departments“, „salaries“ ir „titles“ įrašų informacija, pavyzdys pateikiamas iliustracijoje (7 pav.).



```
{
    "_id" : ObjectId("5ec9bc72384a5a2d14250aca"),
    "emp_no" : "10001",
    "titles" : [
        {
            "title" : "Senior Engineer",
            "from_date" : "1986-06-26",
            "to_date" : "9999-01-01"
        }
    ],
    "salaries" : [
        {
            "salary" : "60117",
            "from_date" : "1986-06-26",
            "to_date" : "1987-06-26"
        },
        {
            "salary" : "62102",
            "from_date" : "1987-06-26",
            "to_date" : "1988-06-25"
        },
    ],
    "departments" : [
        {
            "dept_no" : "d005",
            "from_date" : "1986-06-26",
            "to_date" : "9999-01-01",
            "dept_name" : "Development"
        }
    ],
    "birth_date" : "1953-09-02",
    "first_name" : "Georgi",
    "last_name" : "Facello",
    "gender" : "M",
    "hire_date" : "1986-06-26"
}
```


7 pav. Į „MongoDB“ migruoto lentelės „employees“ įrašą, į kurį buvo įterpti susiję „departments“, „salaries“ ir „titles“ lentelių įrašai, atitinkančio dokumento pavyzdys.

4. Ketvirtoji lentelė sąraše - „salaries“. Ši lentelė turi importuotų (išorinis raktas „emp_no“), tačiau neturi eksportuotų raktų, todėl jai nauja „MongoDB“ kolekcija nekuriama, nes lentelės įrašai bus įterpiami susijusių lentelių, šiuo atveju lentelės „employees“ dokumentuose (2.b.).

5. Penktoji ir paskutinė lentelė sąraše - „titles“. Ši lentelė turi importuotų (išorinis raktas „emp_no“), tačiau neturi eksportuotų raktų, todėl jai nauja „MongoDB“ kolekcija nekuriama, nes lentelės įrašai bus įterpiami susijusių lentelių, šiuo atveju lentelės „employees“ dokumentuose (2.b.).

6. Baigiamas duomenų bazės lentelių sąrašo iteravimas. Duomenų „MySQL“ į „MongoDB“ laikomas užbaigtu.

Programa sėkmingai atliko automatizuotą reliacinės duomenų bazės „employees“ migravimą į „MongoDB“, panaudojant schemos denormalizavimo strategiją. Reliacinėje „employees“ schemoje esantys 1:N sąryšiai buvo denormalizuoti į lentelės „employees“ įrašus atitinkančius dokumentus įterpiant susijusius lentelių „salaries“ ir „titles“ įrašų dokumentus. Lenteles „employees“ ir „departments“ jungiantis N:N sąryšis, šaltinio duomenų bazėje realizuotas panaudojant jungiamąją lentelę „dept_emp“, „MongoDB“ sukurtoje duomenų bazėje sumodeliuotas atliekant dvipusį įterpimą: tarpusavyje susijusių „departments“ ir „employees“ lentelių įrašų dokumentai buvo įterpiami vienas kito viduje.

Išvados

Atlikta egzistuojančios literatūros, susijusios su duomenų migravimų tarp SQL ir NoSQL duomenų bazių, analizė. Pristatyti trys metodai ir juose naudojami įrankiai, skirti duomenų migravimui iš reliacinės SQL į skirtingų tipų NoSQL duomenų bazių valdymo sistemas. Nustatyta, kad visuose metoduose autorių siūloma veiksmų seka, nors ir įgyvendinama nevienodai, yra gana panaši: pirmiausia surenkama informacija apie reliacinės šaltinio duomenų bazės struktūrą (lenteles, atributus, raktus, indeksus ir kt.), tuomet pagal tai yra modeliuojama atitinkamo tipo NoSQL duomenų bazės struktūra, pritaikyta migruojamiems duomenims ir jų sąryšiams saugoti, ir pabaigoje yra atliekamas duomenų migravimo procesas. Šio darbo kontekste atlikta apžvalga yra aktuali, kadangi rasta įrodymų, palaikančių sprendimą kuriamos programos, skirtos automatiniam duomenų migravimui, atliekamą funkciją skaidyti į etapus: šaltinio SQL duomenų bazės metaduomenų gavimo bei analizės, ir NoSQL schemos modeliavimo ir duomenų migravimo.

Atliktas strategijų, skirtų reliacinės duomenų bazės modeliavimui „MongoDB“ sistemoje, palyginimas. Nustatyta, kad plačiausiai naudojamos ir rekomenduojamos yra schemos normalizavimo ir denormalizavimo strategijos. Normalizavimo strategija yra paremta siekiu išlaikyti migruojamos reliacinės duomenų bazės struktūrą - kiekvienai lentelei sukurti ją atitinkančią dokumentų kolekciją, o tarpusavio sąryšiais susietas lenteles „MongoDB“ duomenų bazėje atvaizduoti dokumentų nuorodomis, atliekančiomis išorinių raktų funkciją. Pagrindiniai normalizuotos schemos naudojimo „MongoDB“ privalumai yra didelis rašymo ir modifikavimo operacijų efektyvumas bei išsaugoma migruojamos reliacinės duomenų bazės semantika. Vis dėlto normalizuotų schemų duomenų bazės kenčia nuo mažesnio skaitymo operacijų efektyvumo, kadangi „MongoDB“ nepalaiko tiesioginių jungimo (*JOIN*) operacijų, todėl susijusių dokumentų duomenims gauti reikia naudoti keletą sudėtinių operacijų.

Šiai problemai eliminuoti rekomenduojama naudoti schemos denormalizavimo strategiją, paremtą tarpusavyje susijusių dokumentų jungimu naudojant įterpimus. Denormalizuojant migruojamą reliacinę schemą, atliekamas tam tikras išankstinis jungimas (*PRE-JOIN*), kuriuo dėka susiję skirtingų lentelių įrašai „MongoDB“ duomenų bazėje yra saugomi tame pačiame dokumente ir yra pasiekiami viena operacija. Tačiau naudojant dokumentų įterpimus kyla rizika viršyti „MongoDB“ vieno dokumento dydžio limitą ir yra kuriamos duomenų kopijos, o tai ne tik didina duomenų bazės apimtį, bet ir mažina rašymo bei modifikavimo operacijų efektyvumą. Prieita prie išvados, kad optimaliausia yra panaudoti abiejų strategijų privalumus ir sukurtoje automatizuoto migravimo programoje naudoti ne vien įterpimų, bet ir nuorodų metodą.

Sukurta programa automatizuotam duomenų migravimui iš „MySQL“ į „MongoDB“ taikant schemos denormalizavimo strategiją. Programa migravimo procesą atlieka dviem etapais: pirmiausia yra gaunama ir struktūrizuojama šaltinio reliacinės duomenų bazės metaduomenų informacija, o tuomet yra atliekamas duomenų migravimas. Duomenų migravimo etape „MongoDB“ schemos modeliavimui yra naudojami lentelių importuotų ir eksportuotų raktų informacija. Pagal apibrėžtas taisykles, atsižvelgiant į importuotų ir eksportuotų raktų kiekį bei į tai, ar lentelės, į kurias eks-

portuojami raktai, pačios eksportuoja raktus į kitas lenteles, yra nustatomas įterpimo ar nuorodų strategijos naudojimas.

Programos funkcionalumas pademonstruotas detaliai aprašant atliktą automatizuotą realios „MySQL“ sistemoje saugomos duomenų bazės migravimą į „MongoDB“. Atlikus migravimą 5 reliacinėse lentelėse saugoma informacija buvo perkelta į 2 dokumentų kolekcijas, denormalizavus reliacinę šaltinio duomenų bazės schemą ir panaudojant vieną dvipusio bei du vienpusio įterpimo metodus. Sėkmingi migravimo bandymų rezultatai leidžia daryti prielaidą, kad sukurtą programą būtų galima nesudėtingai modifikuoti bei pritaikyti duomenų migravimui ir iš kitų reliacinių į dokumentų duomenų bazių valdymo sistemas, kadangi fundamentaliai jų visų funkcionalumas yra pakankamai panašus.

Summary

Nowadays, many programs and software applications are facing a constant growing of data that has to be stored and manipulated efficiently, which has proved to be a tough challenge for the rigid traditional relational SQL databases. This has increased the popularity of much less schema restricted document type NoSQL databases which are much better suited to handle large amounts of unstructured data. However, migrating data from relational to document NoSQL databases can be a challenging task, because it requires knowledge of relational schema to select an optimal migration strategy. Proposed solution to this problem - an application that utilizes schema denormalization strategy to perform automatic data migration from „MySQL“ to „MongoDB“. The automatic migration of data is performed by analyzing existing relationships between relational tables and using embedding method to store related entries as a single „MongoDB“ collection document. In this document the migration process of test „MySQL“ database to „MongoDB“ has been described in detail to provide a better understanding of how the algorithm behind proposed application works.

Keywords: SQL, NoSQL, „MySQL“, „MongoDB“, automatic migration, schema denormalization, document embedding

Literatūra

[Bou18] S. Bouamama. Migration from a relational database to nosql. *International Journal of Knowledge-Based Organizations*, 8:63–80, 2018.

[DbE20a] DbEngines. Db-engines ranking. `https://db-engines.com/en/ranking`, 2020. tikrinta 2020-05-05.

[DbE20b] DbEngines. Db-engines ranking of relational dbms. `https://db-engines.com/en/ranking/relational+dbms`, 2020. tikrinta 2020-05-05.

[GPG+15] C. Gyorodi, G. Pecherle, R. Gyorodi, and A. Olah. A comparative study: mongodb vs. mysql. *Conference Paper: The 13th International Conference on Engineering of Modern Electric Systems, Oradea*, 2015.

[KGK14] A. Kanade, A. Gopal, and S. Kanade. A study of normalization and embedding in mongodb. *Conference Paper: 2014 IEEE International Advance Computing Conference (IACC)*, 2014.

[LLD15] D. Liang, Y. Lin, and G. Ding. Mid-model design used in model transition and data migration between relational databases and nosql databases. *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, 2015.

[MH13] A. B. M. Moniruzzaman and S. A. Hossain. Nosql database: new era of databases for big data analytics - classification, characteristics and comparison. *International Journal of Database Theory and Application*, 6:1–14, 2013.

[Mon18] MongoDB. Rdbms to mongodb migration guide. `https://webassets.mongodb.com/_com_assets/collateral/RDBMStoMongoDBMigration.pdf`, 2018. tikrinta 2020-04-21.

[Ora15] Oracle. Oracle database concepts. `https://docs.oracle.com/cd/E11882_01/server.112/e40540.pdf`, 2015. tikrinta 2020-04-20.

[RVC+15] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, and F. Mourao. A framework for migrating relational datasets to nosql. *Procedia Computer Science*, 51:2593–2602, 2015.

[VMT14] R. De Virgilio, A. Maccioni, and R. Torlone. R2g: a tool for migrating relations to graphs. In *Advances in Database Technology - EDBT 2014*, pp. 640–643, Athens, Greece. OpenProceedings, 2014.

[WK12] B. Walek and C. Klimes. A methodology for data migration between different database management systems. *International Journal of Computer and Information Engineering*, 6:536–541, 2012.