ŠIAULIŲ UNIVERSITETAS
Technologijos, fizinių ir biomedicinos mokslų fakultetas
Kompiuterių sistemų katedra

Mindaugas Ruzgys

ORM karkasų lyginamosios analizės metodas ir jo taikymas

Magistro baigiamasis darbas

Vadovė doc. dr. S. Ramanauskaitė

Šiauliai, 2016

ŠIAULIŲ UNIVERSITETAS

Technologijos, fizinių ir biomedicinos mokslų fakultetas

Kompiuterių sistemų katedra

TVIRTINU

Kompiuterių sistemų katedros vedėjas
doc. dr. E. Paliulis
2016-06-01

ORM karkasų lyginamosios analizės metodas ir jo taikymas

Informatikos inžinerijos magistro baigiamasis darbas

Vadovas
Kompiuterių sistemų katedros dr. S. Ramanauskaitė
2016 m. gegužės ____ d.

Recenzentas
Kompiuterių sistemų katedros dr. A. Slotkienė
2016 m. gegužės ____ d.

Recenzentas
Kompiuterių sistemų katedros dr A. Drukteinienė
2016 m. gegužės ____ d.

Autorius
ITM-14 gr. studentas M. Ruzgys
2016 m. gegužės 26 d.

Šiauliai, 2016

SANTRAUKA

ORM karkasų lyginamosios analizės metodas ir jo taikymas

Augant reliacinių duomenų bazių populiarumui auga ir kokybiškų ORM karkasų poreikis. Šiuo metu nėra vieningos metodikos ORM karkasų kokybiniam vertinimui. Dauguma atliktų ORM karkasų vertinimų nesilaiko jokios bendros tvarkos ir yra pritaikyti konkrečiam moksliniam darbui.

Šiame darbe pristatoma metodika, kurios pagalba galima palyginti ORM karkasus. Palyginimo metodiką sudaro trys dalys: palyginimas neįtraukiant techninių reikalavimų, griežtas palyginimas įtraukiant techninius reikalavimus ir lankstus palyginimas įtraukiant techninius reikalavimus. Kurį palyginimo būdą naudoti renkamasi pagal konkretaus palyginimo tikslą.

Norint įrodyti siūlomos metodikos teisingumą ir tinkamumą buvo atliekamas pasirinktų ORM palyginimas remiantis siūloma metodika ir ekspertiniu vertinimu. Palyginus palyginimo rezultatus galima spręsti apie metodikos tinkamumą naudoti praktikoje.

SUMMARY

Comparative Analysis Method for ORM (Object Role Modelling) Frameworks and its Application

While the popularity of relational database is increasing, the need of ORM frameworks is increasing too. Currently there is no unified methodology for ORM framework evaluation. Existing works on ORM framework analysis are using different methodologies, which depends on the situation and the purpose of the work. Therefore, it is difficult to compare ORM frameworks, analysed in different works.

In the thesis, a new methodology for ORM comparison presented. The methodology has three parts: comparison when no technical requirements provided; comparison when strict technical requirements are applied; comparison when flexible technical requirements (which might be changed is a better solution exists) are applied. The type of methodology varies as different requirements for ORM framework selection exists.

Three main php ORM frameworks compared according to proposed methodology in order to prove the suitability and correctness of proposed methodology. Obtained results compared to ORM frameworks evaluation results, provided by experts.

TERMINŲ IR SANTRUMPŲ ŽODYNĖLIS

CMD sąsaja	-	Sąsajos tipas kai naudojama komandinė eilutė;
DB	-	Duomenų Bazė;
DBVS	-	Duomenų Bazės Valdymo Sistema;
Eager loading	-	ORM karkaso darbo režimas, kai visi duomenys užkraunami iškart;
GUI	-	Grafinė sąsaja;
IS	-	Informacinė Sistema;
Lazy loading	-	ORM karkaso darbo režimas, kai dalis duomenų užkraunama ne iškart;
Migracija	-	Failas aprašantis duomenų bazės struktūros pokyčius;
ORM karkasas	-	Tarpinė programinė įranga, leidžianti pbjektiškai dirbi su duomenų baze;
Podėlis	-	Laikinai saugomi darbo rezultatai, kuriuos vėliau tikimąsi panaudoti;
RAW užklausa	-	Paprata SQL užklausa, nenaudojant ORM siūlomų įrankių;
Transakcija	-	veiksmų seka, kuri traktuojama kaip vienas loginis vienetas;

PAVEIKSLĖLIŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

TURINYS

Įvadas

Informacinių technologijų amžius keičia ne tik kompiuterinių sistemų vartotojų, bet ir informacinių sistemų kūrėjų, programuotojų įpročius ir naudojamas technologijas. Naujų sistemų kūrimo greičiui padidinti dažnai naudojami karkasai. Jie programuotojui leidžia naudotis kitų sukurtais programiniais komponentais ir juos taikyti, jungti tarpusavyje, nesigilinant į vidines jų veikimo smulkmenas.

Nerimą dėl netinkamo ar kenksmingo karkasų veikimo išsklaido tai, jog egzistuoja didelis atviro kodo ir plačiai naudojamų, kitų programuotojų rekomenduojamų karkasų pasirinkimas. Tačiau naujai pradedantiems programuotojams didelis karkasų pasirinkimas reikalauja laiko sąnaudų tinkamiausių karkasų pasirinkimui. Net ir su karkasais jau dirbantys asmenys kartais nori peržvelgti naujai rinkoje pasirodžiusius karkasus, palyginti juos su jau egzistuojančiais ir taip įvertinti ar jie atitinkamoje situacijoje taiko tinkamiausią karkasą. Todėl šio **darbo objektas** yra objekto-reliacijos susiejimo (angl. Object-Relational Mapping – ORM) karkasų lyginamoji analizė.

ORM karkasai programuotojui leidžia duomenų bazių paslaugas naudoti analogiškai, kaip objektiniame programavime objektus. Taikant ORM karkasus mažiau dėmesio skiriama pačių duomenų bazių valdymo sistemų (toliau DBVS) specifikai, o programinis kodas tampa labiau objektiškai orientuotas.

Kadangi šiuolaikinės informacinės sistemos sunkiai įsivaizduojamos be duomenų bazių panaudojimo, tai ORM karkasai yra vienas iš esminių kuriamos sistemos kokybę įtakojančių aspektų. Tačiau palyginti egzistuojančius ORM karkasus vis dar sudėtinga, nes šiuo metu nėra detalaus metodo, kurio metu būtų galima aiškiai palyginti kelis karkasus tarpusavyje. Šio darbo **tikslas** – padidinti ORM karkasų lyginimo objektyvumą, pasiūlant kiekybinį ORM karkasų lyginimo metodą ir lyginamajai analizei naudojamus duomenų rinkinius.

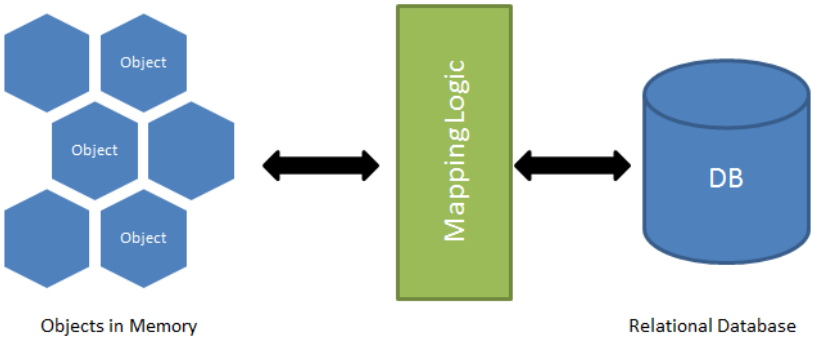
Šiam tikslui pasiekti yra iškelti šie **uždaviniai**:

1. Susipažinti su ORM technologijos principais ir jų karkasų esminėmis savybėmis.
2. Išskirti ORM karkaso pasirinkimą įtakojančius parametrus.
3. Pasiūlyti metodą kiekybiniam ORM karkasų palyginimui.
4. Pritaikyti pasiūlytą metodą pasirinktų ORM karkasų lyginamajai analizei atlikti.

Šio darbo rezultatai leis paprasčiau atlikti kelių pasirinktų ORM karkasų palyginimą ir/arba pasirinkti tinkamiausią karkasą pagal vartotojo turimus reikalavimus. Lyginant su egzistuojančiais ORM karkasų palyginimais, pasiūlytas metodas išsiskirs tuo, kad leis kiekybiškai įvertinti analizuojamus karkasus ir taip padidinti gautų rezultatų objektyvų palyginimą.

1. ORM KARKASAI

Pasaulyje, nuo pat interneto atsiradimo ir HTTP standarto sukūrimo, internetinių svetainių kiekis auga eksponentiškai[1]. Tai reikalauja vis optimalesnio turimų resursų panaudojimo ir pačio internetinių sistemų kūrimo supaprastinimo. Atsiradus ir išplitus modernioms programavimo kalboms, tapo įmanoma kurti dinaminius puslapius, kurie yra interaktyvūs. Absoliuti dauguma dinaminių internetinių puslapių naudoja duomenų bazes saugoti puslapio duomenims bei vartotojo informacijai[2]. Dėl plataus duomenų bazių panaudojimo paplitimo atsirado poreikis sukurti įrankius, kurie palengvina darbą su ja. Informacinių sistemų kūrimui supaprastinti buvo pradėta kurti objekto-reliacijos susiejimo (*angl.* Object-Relational Mapping – ORM) karkasus, kurie leidžia duomenų bazių paslaugomis naudotis analogiškai, kaip objektiniame programavime objektais, mažiau dėmesio skiriant pačių duomenų bazių valdymo sistemų (toliau DBVS) specifikai[2].

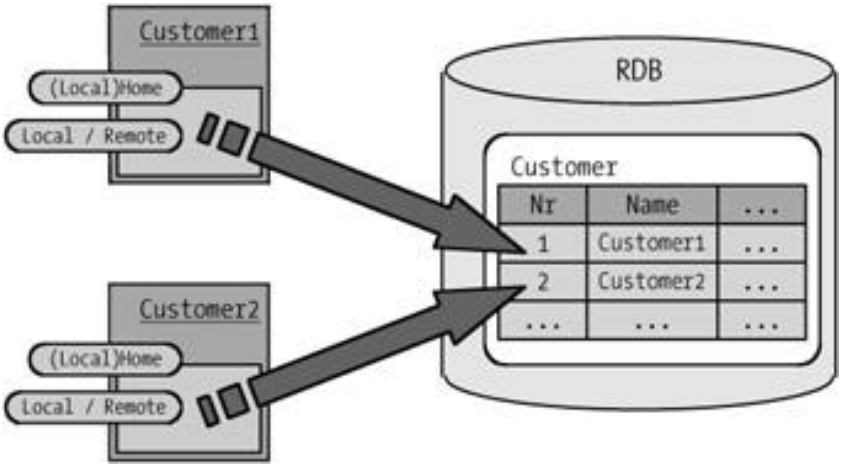


pav. 1 ORM karkaso vieta IS struktūrinėje schemoje[27]

1.1. Technologinė ORM analizė

Norint labiau suprasti ORM karkasų veikimą buvo padaryta literatūros analizė. Tai padės nustatyti svarbiausius veiksnius, kurie įtakoja ORM kakaso darbą.

ORM karkasai leidžia bendrauti su duomenų baze taip, lyg duomenų bazė būtų objektų rinkinys [6, 7]. Paprastai kiekvienai esybei būna sukuriama klasė (Modelis), o ORM tarnauja kaip tarpininkas tarp aplikacijos ir duomenų bazės, nes duomenų bazėje saugomus duomenis kuriamoje sistemoje pateikia kaip objektus, o visas sistemoje vykdomas komandas su objektais verčia atitinkamomis komandomis duomenų bazėje. Tad sistemoje vienas duomenų bazės įrašas atitinka vieną atitinkamo tipo objektą (žr. 2 pav.). Atitinkamai tai programuotojui leidžia dirbti su aiškios struktūros objektu, nebijant sumaišyti įrašo laukų tipus ir kontekstą[6].



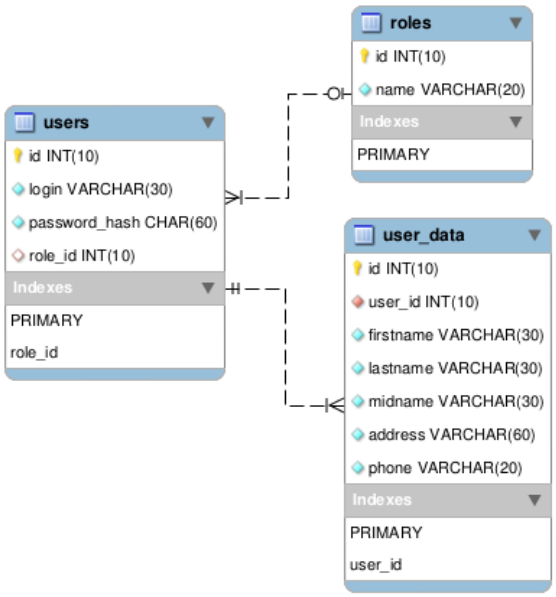
pav. 2 ORM atliekama abstrakcija [8]

1.1.1. ORM technologijoje naudojami ryšių tarp reliacinės duomenų bazės lentelių

Praktikoje duomenų bazių (toliau DB) lentelės dažnai yra viena su kita susijusios[9]. Pavyzdžiui užsakymas gali turėti pirkėjo identifikatorių, kuris jį identifikuoja sistemoje. Tuo tarpu visa informacija apie pirkėją bus pateikta kitoje duomenų bazės lentelėje.

Pagrindiniai sąryšių tarp lentelių yra:

- Vienas su vienu (One To One, arba 1-1)[10, 11] – pats paprasčiausias sąryšio tipas kai vienas įrašas yra priskirtas 1 ir tik vienam kitos lentelės įrašui. Pavyzdžiui sistemos vartotojų duomenys aprašomi per dvi lenteles – vienoje nurodomas tik vartotojo identifikatorius ir prisijungimo duomenys, tuo tarpu kitoje lentelėje pateikiamas vartotojo identifikatorius ir visa asmeninė vartotojo informacija (žr. 3 pav.) ryšį tarp lentelių users ir user_data). Toks duomenų išskaidymas naudingas duomenų atskyrimo požiūriu, nors praktikoje tos dvi lentelės galėtų būti apjungiamos į vieną.

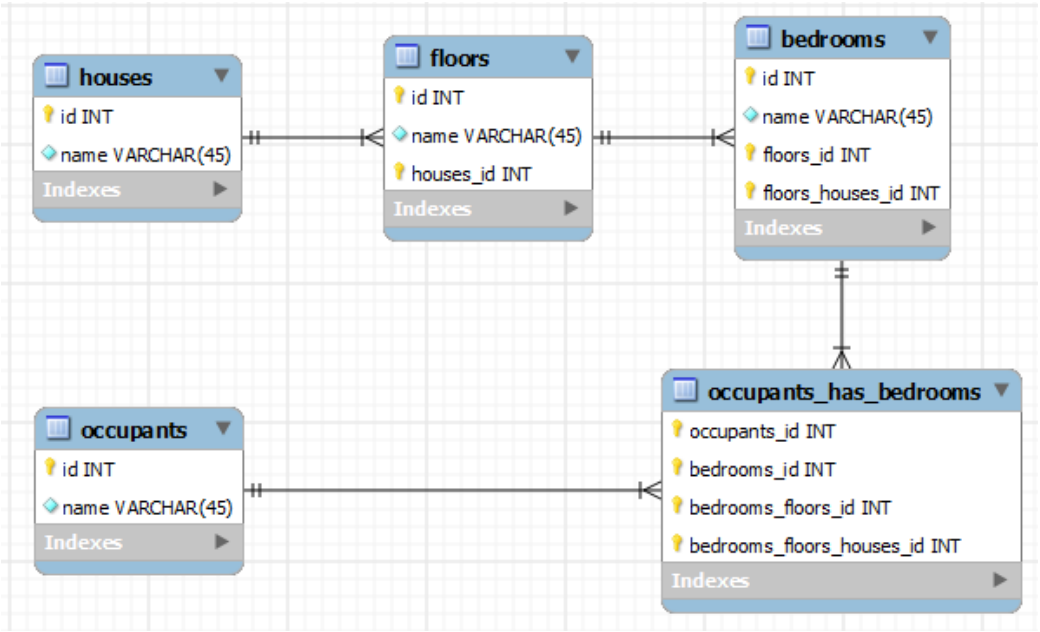


pav. 3 1-1 ir 1-N sąryšių pavyzdžiai[28]

Vienas su daug (One To Many, arba 1-N)[11] – sąryšio tipas kai vienas įrašas gali turėti neribotą kiekį kitų jam priklausančių įrašų. Kaip pavyzdys gali būti vartotojui priskirtos teisės – vienas vartotojas

gali turėti tik vieną jam priskirtą vaidmenį, tuo tarpu vienas vaidmuo gali būti priskirtas keletui skirtingų vartotojų.

- Daug su Daug (Many To Many, arba N-N)[11, 12] – dažnai pasitaikantis, šiek tiek sudėtingesnis sąryšio tipas. Pavyzdžiui viename kambaryje gali gyventi daug gyventojų, o vienas gyventojas (skirtingu laiko momentu, ar net tuo pačiu metu), gali užsisakyti daug kambarių. Tokiam sąryšiui realizuoti reikalinga tarpinė lentelė, kurioje nurodomi sąryšiai tarp įrašų (žr. 4 pav.) lenteles bedrooms ir occupants, kur tarpinė lentelė yra occupants_has_bedrooms).



pav. 4 Sąryšių N-N ir 1-X-N pavyzdys[13]

- Turi daug per (Has Many Through, 1-X-N)[11, 12] – rečiau pasitaikantis sąryšio tipas, leidžiantis patogiau gauti reikalingus duomenis kai yra keletas sąryšių. Tokio sąryšio pavyzdys gali būti namas, kuriame yra daug aukštų, o kiekviename aukšte daug kambarių. Tad norint gauti visus tame name esančius kambarius, nėra tiesioginio sąryšio tarp tų dviejų lentelių, o duomenys tarp jų atrenkami per tarpinę, aukštus nusakančią lentelę.

"Daug su daug" – tai kiek labiau komplikuotas sąryšio tipas nei "vienas su vienu" ar "vienas su daug", tačiau labai dažnai pasitaikantis[12]. Todėl labai svarbu, kad ORM sistema turėtų kuo patogesnį sąryšio realizavimą. Kadangi šiam sąryšiui realizuot reikalinga papildoma sąryšių lentelė, labai svarbu kaip ORM su ja komunikuoja.

Dažnai pasitaikantis atvejis, kai informacinės sistemos (toliau IS) modelio klasė paveldi ORM klasę. Įprastu atveju viena lentelė turi jai priskirtą modelį. Todėl svarbu kad ORM nereikalautų sąryšių lentelei kurti atskiro modelio, nes jis dažnai būna naudojamas tik sąryšiui užtikrinti ir neturi jokios kitos logikos. Tokie modeliai apsunkina sistemos kūrimą ir supratimą. Todėl idealiu atveju sąryšių lentelei neturėtų reikėti atskiro modelio.

1.1.2. ORM karkasų naudojami susiejimo metodai

Kuriant ORM karkasus naudojami keli pagrindiniai realizacijos metodai [14]:

- Active Records.
- Table Data Gateway.
- Row Data Gateway.
- Data Mapper.

Active Records ir Data Mapper yra dažniausiai naudojami principai [14], kai tuo tarpu Table Data Gatey ir Row Data Gateway yra labai retai naudojami ar nagrinėjami [15].

1.1.3. Active Records metodas

Active Record metodas vienas labiausiai naudojamų ORM karkasų realizavimo metodų[14, 15]. Taikant šį metodą kiekviena duomenų bazės lentelė yra apgaubiama (*wrap*) klasės, o kiekviena eilutė duomenų bazėje yra tos klasės objektas. Prielaida kad viena lentelės eilutė atitinka vieną objektą labai palengvina CRUD (pagrindines darbo su įrašais – Create Read Update Delete) operacijas. Naudojant Active Record realziavimo metodą esybės objektas saugo tiek informaciją, tiek logiką, kaip ta informacija manipuliuoti.

Paprastai šio tipo ORM technologija leidžia pakankamai automatizuotą objekto klasės sudarymą. Šis metodas programuotojui kuriant esybės objektą leidžia ne pačiam nurodyti duomenų bazės lentelės laukus, o juos gauti automatiniu būdu iš duomenų bazių valdymo sistemos. Pastarieji laukai yra nustatomi ir duomenų tipai parenkami automatiškai pagal duomenų bazės lentelės schemą. Tai labai palengvina ir pagreitina objektų klasių sukūrimo darbus.

Paprastai esybės objektas paveldi ORM karkaso klasę. Kadangi Active Records atveju, esybės objektas paveldi ORM karkaso klasę, pažeidžiamas *single responsibility* kūrimo principas, kuris teigia, kad klasė turėtų būti tokia, jog jos pokyčius galėtų įtakoti tik viena priežastis [16].

Siekiant skirtinguose vienodo tipo karkasuose naudoti panašią panaudojimo logiką, Active Record esybė dažniausiai turi keletą pagrindinių metodų [17]

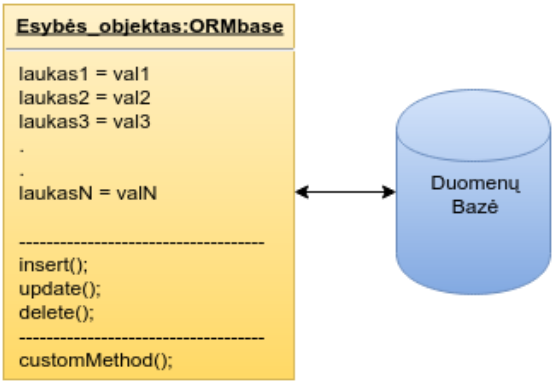
- Objekto sudarymas pagal SQL užklausa gautus duomenis.
- Naujo objekto sukūrimas vėlesniam įrašymui į duomenų bazę.
- Statinis paieškos metodas, kuriame galima nurodyti norimą įvykdyti SQL užklausą ir gauti Active Record objektą.
- Duomenų bazės atnaujinimas ir įrašo įrašymas, pagal Active Record objekto duomenis.
- Atributų reikšmių gavimas ir nustatymas.
- Tam tikros verslo logikos realizavimas.

Tai tik pagrindinis Active Record funkcionalumas, kurį stengiamasi būtinai realizuoti ORM karkase, tačiau gali būti naudojama ir kitų, papildomų funkcijų. Pavyzdžiui atributų reikšmių gavimo ir nustatymo metodai gali atlikti ir papildomus veiksmus, tokius kaip pervertimas iš SQL paremtų tipų į atminties naudojamus tipus [14].

Active Record tipo objekto klasėje galima nurodyti ryšius su kitomis susijusiomis klasėmis, kaip ryšius tarp reliacinių duomenų bazių lentelių[20]. Tokiu būdu sudaromas pilnesnis objektų

tarpusavio sąryšių vaizdas. Atitinkamai jei objektui reikia gauti kito tipo objektus, t.y. duomenis iš kitų, susijusių lentelių, tai tų objektų gavimui nebūtina žinoti identifikavimo raktų, nes ši informacija gali būti gaunama automatiškai pagal duomenų struktūrą, ryšius tarp lentelių. Ši galimybė yra patogi naudojimui, bet kartu palieka ir reliacinės duomenų bazės principų taikymą[14].

Apibendrinant Active Record realizavimo principus galima jį prilyginti pilnam objektiškai orientuotam programavimui, nes kiekviena reikiama esybė iš duomenų bazės aprašoma kaip klasė, su savo struktūra ir metodais. Ryšys su duomenų baze realizuojamas jau pačios klasės, jos metodų lygmenyje, kurie nusako kaip atliekami veiksmai yra susiję su susietąja duomenų baze[12]. Atitinkamai naudojantis tokiu būdu aprašytomis klasėmis didžioji dalis klasių yra objekto atributai ir metodai, tad juos kviečiame ir keičiame lygiai taip pat, kaip ir objektiškai orientuotame programavime[14, 12].



pav. 5 Active Record metodu susiejama klasė su duomenų baze

```
   var product = new Product(); //kuriamas Product tipo objektas
   product->title = 'Personal computer'; //keičiama objekto atributo reikšmė
   product->save(); //tiesiogiai kviečiamas objekto metodas, kuris savyje aprašo
sąsajas su duomenų baze
```

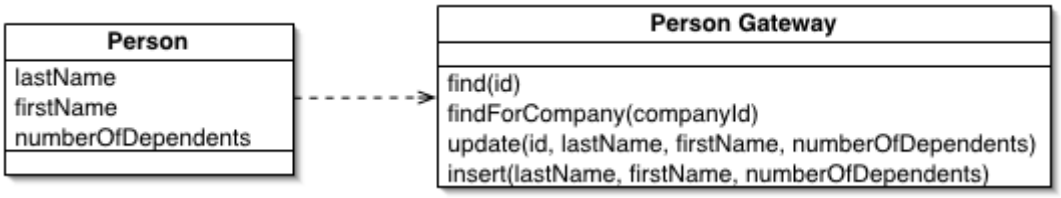
Active Record metodu kuriamo ir naudojamo objekto pavyzdys

Dėl savo savybių Active Record metodas yra tinkamas, kai reikia objekto metodų lygmenyje aprašyti verslo logiką, metodus duomenų skaitymui, rašymui, atnaujinimui ir šalinimui[14,18, 19]. Šis metodas ypač tinkamas, kada dirbama pavienių įrašų lygmenyje. Papildomai jis yra lengvai perprantamas ir paprastas. Tačiau šis metodas geriausiai veikia tik tada, kada Active Record klasė tiesiogiai atitinka vieną duomenų bazės lentelę. Tačiau jei prasideda sudėtingesnė verslo logika, naudojami sudėtingesni sąryšiai tarp duomenų bazių lentelių, tada programuotojui tenka naudoti paveldėjimą, kolekcijas, sąryšius tarp objektų ir pan. Tai nepatyrusio programuotojo rankose gali labai apsunkinti programinio kodo suvokimą ir veikimą[19]. Taip pat šis metodas sudėtingiau keičiamas, kada esama duomenų bazė ir projektas keičiasi, vystosi, nes gali tekti keisti visas klases, nes jos tarpusavyje labai susijusios.

1.1.4. Table or Row Data Gateway metodas

Table ir Row Data Gateway metodas yra labai panašus į Active Record metodą. Skirtumas tarp jų tik tas, kad Active Record klasėje yra aprašoma objekto struktūra ir verslo logika, kai tuo tarpu Table ir Row Data Gateway metodais siūloma aprašyti tik duomenų struktūrą[21]. Dėl šio skirtumo dažnai Table ir Row Data Gateway metodais siūloma statinius duomenų paieškos metodus aprašyti atskiroje

klasėje (žr. 6 pav.). Šis skirtumas neturi aiškios ribos, tad labai dažnai priklausomai nuo situacijos sudėtinga atskirti kuriam metodui reikėtų priskirti atitinkamą kodą[24].



pav. 6 Table Data Gateway panaudojimo struktūros ir logikos atskyrimui pavyzdys [24]

Tuo tarpu skirtumas tarp Table ir Row Data Gateway yra objektų aprašymo lygmenyje. Table Data Gateway siūlo objektą aprašyti kaip vieną klasę. Tuo tarpu Row Data Gateway klasės ir lentelės struktūra nėra tokia griežtai susieta, tad programuotojas gali laisviau aprašyti norimą objekto struktūrą.

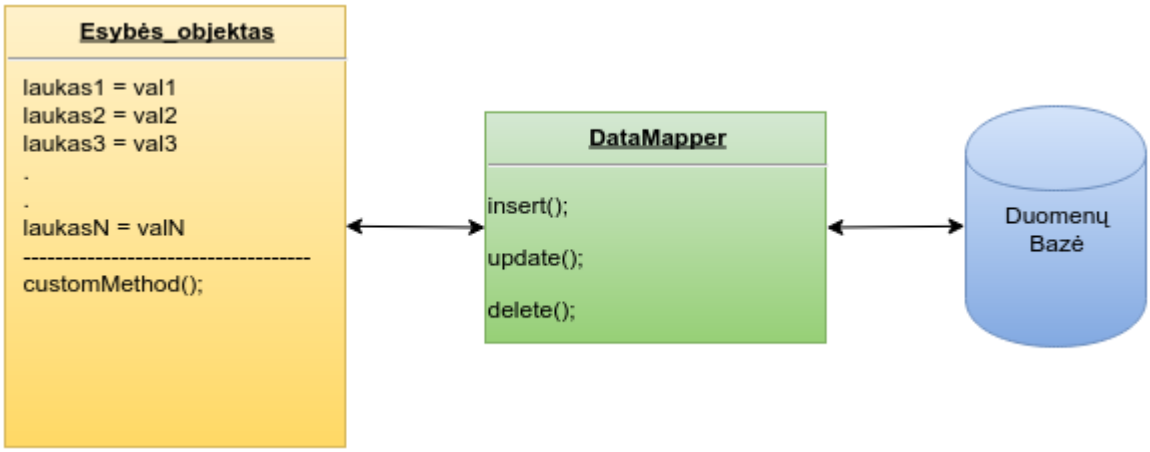
1.1.5. Data Mapper metodas

Situacijose, kada nereikia griežto susiejimo tarp skirtingo tipo objektų ir objektų ir duomenų bazės, yra naudojamas Data Mapper metodas. Šio metodo metu tarp objektų ir duomenų bazės yra naudojamas papildomas sluoksnis, kuris yra ne toks griežtas ir leidžia lanksčiau keisti objektų ir duomenų bazės susiejimą [14].

Objektai ir reliacinė duomenų bazė iš principo yra skirtingos struktūros. Objektiniame programavime dažniausiai gilinamasi į verslo logiką, sudėtingą vidinę objekto struktūrą, paveldėjimą. Tuo tarpu reliacinės duomenų bazės labiau orientuotos į duomenų aprašymą ir saugojimą. Tad šie du modeliai dažnai yra pakankamai nepanašūs.

Kai Active Record metodas objektų susiejimui su reliaciniu modeliu tampa perdaug sudėtingas, reikalingas kitoks sprendimas, kuris nesuteiktų tiek daug objektinio programavimo lankstumo, bet leistų tiksliau aprašyti objektų sąsajas su reliacine duomenų baze, jos lentelėmis ir įrašais.

Data Mapper metodo principas yra papildomo tarpinio sluoksnio įterpimas, kuris būtų atsakingas už objekto ir reliacinės duomenų bazės susiejimą[24] . Atitinkamai objekto klasė savyje saugo pačio objekto struktūrą, tuo tarpu perėjimas iš objektiško duomenų į reliacinį modelį vykdomas tarpiniame sluoksnyje, kuris ir yra atsakingas už visų duomenų bazės įrašų valdymą (įrašymą, atnaujinimą, šalinimą).



pav. 7 Data Mapper metodo realizavimo principas

Tokiu atveju dažnai programiniame kode dirbama su atitinkamais objektais tiesiogiai, tačiau norint, kad objekto duomenys būtų atspindėti ir reliacinėje duomenų bazėje, kreipiamasi į papildomą klasę ar komponentą, kurių pagalba naudojami objektai yra perverčiami į duomenų bazei suprantama formatą.

```
   var product = new Product(); //objekto sukūrimas
   product->title = 'Personal computer'; // objekto savybės reikšmės keitimas
   EntityManager::save(product);  //veiksmų  su  objektų  atlikimas,  per  tarpinį
EntityManager elementą, o ne tiesiogiai
```

Data Mapper metodo panaudojimo pavyzdys

Data Mapper metodo metu itin didelis dėmesys skiriamas tam tarpiniam sluoksniui, kuris yra tarpinė tarp atminties naudojamų tipų nuo SQL kalbos. Šio tarpinio sluoksnio paskirtis perduoti duomenis tarp šių dviejų pusių, bet tuo pačiu izoliuoti vieną nuo kito. Tokiu būdu objekto klasėje dažniausiai visiškai nenaudojamos SQL užklausos, jam praktiškai nebūtina žinoti duomenų bazės schemos ar struktūros.

Nors Data Mapper metodas atrodo paprastesnis objektų sąryšių prasme, tačiau šio metodo esminis sudėtingumas slypi būtent tarpiniame sluoksnyje tarp objektų ir duomenų bazės. Šis sluoksnis gali būti realizuojamas labai įvairiai, tačiau dažniausiai jis atsakingas už tai, kaip pagal vartotojo suskurtus ar modifikuotus objektus sugeneruoti atitinkamą SQL užklausą. Tad nors programavimo prasme programuotojo darbas supaprastėja, nes nebūtinas SQL išmanymas, tačiau pakankamai didelė atsakomybė dėl darbo su duomenų baze našumo skiriama ORM karkaso kūrėjams, jų sugebėjimui optimizuoti tarpinio sluoksnio logiką ir jos veikimą.

Pagrindinis šio metodo skirtumas nuo Active records tas, kad esybės objektas nesaugo jokios informacijos apie duomenų bazę. Tai plain objektai saugantys tik duomenis, bet ne ryšius tarp obejktų. Tokiu būdu esybės objektai yra “lengvesni” (užima mažiau atminties). Tačiau norint objekto informaciją patalpinti duomenų bazėje reikia kviesti papildomą ORM karkaso klasę, kai tuo tarpu naudojant Active Record metodą užteko tiesiogiai iškviesti save() metodą[25].

1.1.6. Duomenų užkrovimo tipai

Dažnu atveju viena esybė siejasi su kita esybe. Pavyzdžiui straipsnis gali turėti komentarus. Tokiu atveju straipsnio esybė turi sąryšį *one to many* su komentaro esybe[11, 12]. Norint gauti Straipsnio esybę iš duomenų bazės iškyla problema, kad pirma reikia išrinkti straipsnį ir tada jo komentarus. Tarkim norint gauti 5 straipsnius su komentarais reikės įvykdyti 6 užklausas: viena užklausa visų 5 straipsnių gavimui, o tada kiekvienam straipsniui dar po užklausą atitinkamo straipsnio komentarų gavimui. Literatūroje tai vadinama N+1 problema[26].

```
   var straipsnis = Straipsnis::get()->limit(5); //5 straipsnių duomenų gavimas
   foreach straipsnis //ciklas kiekvienam straipsniui
            komentarai = straipsnis->getComments(); //komentaro gavimas kiekvienam
straipsniui
```

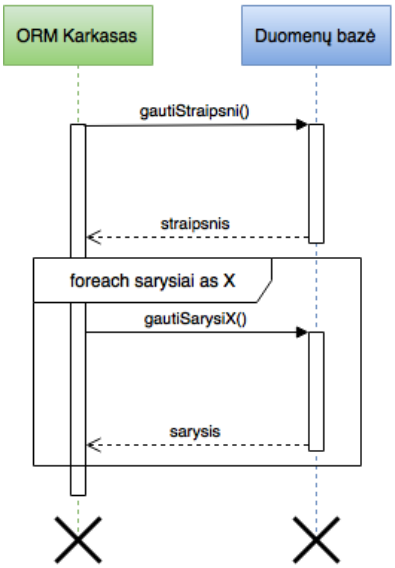
N+1 problemos pavyzdys, kada gaunami 5 straipsniai, su visais jų komentarais

Paprastai ORM karkasai atrinkdami duomenis iš kelių lentelių gali veikti dviem režimais[22]:

- Eager Loading.
- Lazy Loading.

1.1.7. Eager Loading

Eager Loading atveju visi sąryšiai yra išrenkami iš karto, kai tik išrenkama tėvinė esybė. Tokiu atveju kada vartotojas kreipiasi tėvinio objekto gavimui, ORM karkasas iš karto atrenka ir visus dukterinius objektus (žr. 8 pav.). Galimi atvejai, kada ORM karkasas suformuoja vieną užklausą tėvinio objekto duomenų gavimui ir vieną bendrą užklausą visų reikiamų dukterinių objektų išrinkimui. Dar kitas galimas variantas – viena užklausa tėviniam ir visiems reikiamiems dukteriniams objektams išgauti. Toks sprendimas sudėtingesnis formuojant SQL užklausa gautų rezultatų objektus, tačiau leidžia sutaupyti užklausų į duomenų bazę kiekį.



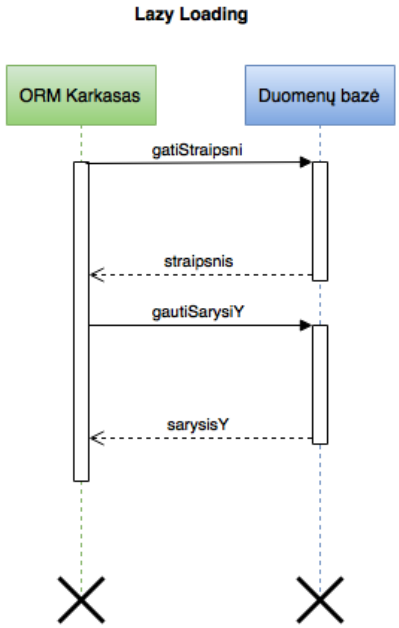
pav. 8 Eager Loading darbo režimo pavyzdys

Šis darbo režimas naudingas, kada tikrai reikės visų dukterinių objektų. Tačiau tais atvejais, kada dukterinių objektų gali ir neprireikti, tai bus neoptimaliai išnaudojami sistemos resursai, nes bus

gaištamas laikas pačių įrašų gavimui iš duomenų bazės, o tuo pačiu ir bus neoptimaliai išnaudojama vieta atmintyje.

1.1.8. Lazy Loading

Naudojant Lazy Loading darbo režimą sąryšiai nėra išrenkami automatiškai to pačio kreipimosi metu. Pirmojo kreipimosi metu gaunamas tik tėvinis objektas. Tuo tarpu dukterinių objektų išrinkimas vykdomas tik tada kai jų pareikalaujama (žr.8 pav.).



pav. 9 Lazy loading darbo pavyzdys

Naudojant Lazy Loading mažiau perteklinės informacijos, kas leidžia neapkrauti sistemos, kada yra maža tikimybė, jog reikės dukterinių objektų. Tačiau jei naudojamas Lazy Loading darbo režimas, bet pagal situaciją reikia gauti visus dukterinius objektus, tada vos tik prireikia dukterinio objekto, bus tikrinama ar jis jau yra išrinktas ir jei nėra – kreipiamasi tuo metu gauti reikiamą įrašą. Tai lėtina sistemos darbą, kai turime daug dukterinių ir dažnai naudojamų įrašų[14].

1.2. Egzistuojantys ORM karkasų lyginimo metodai

Kadangi ORM karkasų naudojimas ir naudojimo poreikis nuolat auga, galima rasti nemažai literatūros. Todėl pirmiausia buvo išanalizuota rasta literatūra ir įvertinti kitų autorių padaryti darbai. Įpatingas dėmesys buvo kreipiamas į pasirinkto lyginimo metodo vertinimą ir tikslumą.

Autorius Jaroslav Orság savo darbe „Object-Relational Mapping“ atliko ORM karkasų lyginamąją analizę[3]. Savo darbe autorius apžvelgė savybes, kurios lemia ORM karkaso kokybę. Galutiniam vertinimui buvo atrinkti 5 ORM karkasai, kurie veikia skirtingose programavimo kalbose. Taigi Siūlomas vertinimo metodas yra universalus.

Kriterijai, kuriais rėmėsi Jaroslav Orság lygindamas ORM karkasus[3]:

- Skaidrumas – parodo kiek pastangų, norėdamas dirbti su pasirinkta DBVS, turi įdėti progrmuotojas;

- Dokumentacija ir palaikymas – tiriama dokumentacijos kokybė. Vertinama kaip greitai galima rasti sprendimus dažniausiai pasitaikančioms problemoms;
- Priklausomybės, Palaikomos DBVS, Palaikomos programavimo aplinkos – tiriama ar ORM karkaso veikimui reikalingi trečiųjų šalių įrankiai. Taipat palaikomos DBV sistemos. Kuo daugiau palaikomų DBVS tuo geriau. Atsižvelgiama ir į programavimo aplinkas kuriose gali veikti ORM karkasas;
- Modelio plėtojimo galimybės – aiškinamasi kaip lengvai atnaujinamas, keičiamas ir kitaip plėtojamas modelis. Tiriama ar reikia atlikti papildomus veiksmus konfigūracijos ir kituose failuose;
- Perprantamumas – tiriama ORM karkaso naudojmo paprastumas naudotojams, kurie anksčiau su tuo karkasu nedirbo.
- Konfigūravimo sudėtingumas – tirima kokiais metodais galima konfigūruoti ORM karkasą ir kaip lengva tai padaryti.
- Modelio priklausomybės – tiriama ar modelis turi papildomų priklausomybių, tokių kaip specifinių klasių paveldėjimas ar interfeisų paveldėjimas.

Visi kriterijai buvo vertinami 4 balų sistema[3]:

- 0 taškų: savybė nepalaikoma;
- 1 taškas:savybė palaikoma, bet yra didelių apribojimų;
- 2 savybė palaikoma;
- 3 savybė pilnai palaikoma;

1 lentelė. Jaroslav Orság ORM palyginimo rezultatai

	DataObjects.NET	**Hibernate**	**TopLink**	**LLBLGenPro**	**Java Data Objects**
Skaidrumas	3	3	3	3	3
Dokumentacija ir palaikymas	3	3	3	3	3
Priklausomybės, Palaikomos DBVS, Palaikomos programavimo aplinkos	2	3	3	3	3
Modelio plėtojimo galimyb	3	3	3	3	3
Perprantamumas	3	3	2	3	2
Konfigūravimo sudėtingumas	0	3	3	2	3
Modelio priklausomybės	2	3	3	1	2

Kaip matome palyginimas vyko tik vertinant pagal septynias savybes. Tai gali įtakoti vertinimo tikslumą, nes galimai svarbios ORM savybės nepatenka į vertinimą, jos svoris vertinime neadekvatus

svarbai. Taipogi tokiu atveju labai svarbus vertintojo/eksperto vaidmuo, nes jam tenka įsisavimti tiriamų ORM karkasų dokumentacijas ir apibendrinti savybes. Pastarasias dar tenka įvertinti naudojama 4 balų vertinimo sistema. Taigi vertinantintojas galimai įtakoja galutinį įvertinimą. Iš kitos pusės toks abstraktus vertinimas leidžia gana greitai palyginti daug ORM karkasų, įpač jei nėra keliama didelių reikalavimų vertinimo tikslumui.

Kitas autorius Oliver Leisalu dviejų ORM karkasų, skirtų PHP kalbai palyginamąją analizę[4]. Palyginimui buvo pasirinkti 2 ORM karkasasi Doctrine bei DomAr, tačiau buvo svarstomi ir kiti galimi variantai. Tyrime gana plačiai apžvelgimos ORM karkasų savybės, tiek funkcinės, tiek kokybinės.

Kriterijų, pagal kurius vertino Oliver Leisalu sąrašas[4]:

- Sąryšiai(*Relationships*) - ar ORM karkasas palaiko sąeryšius tarp klasių?
- Dvikrypčiai sąryšiai(*Bi-directional relationships*) - ar ORM karkasas palaiko dvikrypčius sąryšius tarp klasių. Ar užtikrinamas duomenų integralumas?
- Hierarchiniai duomenys(*Hierarchical data*) - ar ORM karkasas turi įrankius skirtus dirbti su hierarchiniais duomenimis?
- Stulpelių agregacijos paveldėjimas(*Column aggregation inheritance*) – Ar palaiko skirtingų objektų saugojimą vienoje lentelėje?
- Transakcijos(*Transactions*) – Ar ORM karkasas palaiko transakcijas?
- „Nešvarus“ tikrinimas(*Dirty checking*) – Ar ORM karkasas stebi „Nešvarius objektus“? Objektas vadinamas nešvariu, kai pakeičiama nor viena jo savybė.
- Optimizacija(*Optimization*)– Ar ORM karkasas pateikia nurodymus kaip optimizuoti veikimą?
- „Memcache palaikymas“(*Memcache support*) –Ar palaiko objektų saugojimą atmintyje;
- Trigeriai(*Support for hooks / event listeners*)– Ar palaiko trigerius?
- Views palaikymas(*Support for views*) – Ar ORM karkasas palaiko DBVS views?
- Automatinis schemos generavimas(*Automatic generation of schema*) – Ar pagal modelį gali būti sugeneruota DB struktūra?
- Automatinis modelio generavimas(*Automatic generation of model*) – Ar pagal esamą DB schemą įmanoma sugeneruoti modelį?
- Anotacijos(*Annotations*) – Ar galima susieti naudojant anotacijas?
- Duomenų susiejimo kalba(*Data mapping language*) – Susiejimui naudojamas žymėjimas;
- Palaikomos DBVS(*Supported databases*) – Kokias duomenų bazes palaiko?
- Išplečiamumas(*Extendable*) – Ar yra nurodymai kaip išplėsti esamą ORM funkcionalumą.
- Užklausų kalba*(Query language)* – Ar naudojama užklausų kalba?
- Bendruomenė(*Community*) – Ar yra aktyvi ORM karkaso bendruomenė.

Lyginant pastarojo autoriaus(Oliver Leisalu) vertinimo metodiką su Jaroslav Orság siūloma metodika pastebima, kad jį žymiai detalesnė. Jei aptartame tyrime[3] buvo tik 7 vertinimo kriterijai, tai Oliver Leisalu naudoja net 18.

Norint visapusiškiau įvertinti ORM karkasus autorius atliko greičio matavimo testus. Atliekant testavimą buvo naudojamas *Jounneau* testas. Tai testas skirtas palyginti išskirtinai ORM karkasus[4].

Matome, kad gana plačiai analizuojamas ORM karkaso funkcionalumas. Tačiau darytas palyginimkas turi ir trūkumų, nes pavyzdžiui į klausimą „palaikomos užklausos“ buvo atsakyta „Taip“. Tai neleidžia visapusiškai įvertinti ORM savybės.

Taipogi vertinamų savybių gausa gali labai apsunkinti vertinimo procesą, nes esant daug tiriamų ORM karkasų, reikia daug laiko įsigilinti į jų specifikacijas. Įpač jei ORM karkasai veikia skirtingose programavimo aplinkose. Palyginimą komplikuoja ir atliekami spartos vertinimo testai. Tai reikalauja iš vertintojo programavimo kalbų išmanymo. Jei lyginami ORM karkasai veikiantys skirtingose programavimo aplinkose, vertintojas privalo mokėti dirbti visose.

Dar pastebėtina, kad vertinimo metodas neturi skaitinės išraiškos, todėl sunku galutinai pasakyti kuris ORM karkasas yra pranašesnis, nes įvertinimai pateikiami tik atskiroms savybėms.

Dar vienas autorius, Mikael Kopteff, savo darbe „The Usage and Performance of Object Databases compared with ORM tools in a Java environment“ atliko ORM karkasų palyginimą[5]. Tyrime lyginti tik Java kalboje veikiantys ORM. Išvis buvo atrinkti du ORM karkasai: VERSANT ODBMS 7.0.1.3 ir HIBERNATE 3.2.5. Palyginimui naudoti tokie kriterijai[5]:

- Java interfeisai;
- Dvikrypčiai sąryšiai;
- POJO saugojimas;
- Transityvus saugojimas tarp objektų;
- Java-Collections klasių palaikymas;
- Užrakinimo mechanizmai;
- DB prieigos kontrolė;
- Prisijungimai prie EIS
- JTA palaikymas
- Automatinis sesijos valdymas;
- Optimizacija;
- Paskirstymas;
- „Nešvarus“ tikrinimas;
- Automatinis schemos generavimas;
- Užklausų mechanizmas;
- XML palaikymas.

Pastarasis lyginimo metodas taip pat neturi galutinės skaitinės išraiškos. Priek kiekvienos vetinimos savybės parašomas komentaras. Taigi galutinis įvertinimas lieka vertintojo užduotis.

Išanalizavus egzistuojančius ORM vertinimo metodus[3, 4, 5] matome kad nėra siūloma vieningos vertinimo sistemos. Autoriai atlikdavo ORM karkasų lyginimo analizę, tačiau kiekvienu

atveju būdavo pasirenkama vis kitokia vertinimo eiga. Skyrėsi tiek vertinimo metodas tiek eiga, tiek patys vertinimo kriterijai. Tai galėjo įtakoti kelios priežastys:

- Nėra vieningos ORM karkasų lyginimo sistemos. Kiekvienas autorius turėjo pats atrinkti vertinimo kriterijus, bei sudaryti vertinimo metodiką.
- Kiekvienas autorius atsirinko savo tyrimui svarbius kriterijus, o kitų į vertinimą neįtraukė;
- Į kai kurius tyrimus įtraukti kriterijai, kurie aktualųs tik orm karkasams parašytais tam tikra programavimo kalba.

Nei vienas autorius į vertinimo metodiką neįtraukė techninių reikalavimų. Turimi techniniai reikalavimai gali labai įtakotoi ORM karkaso įvertinimą.

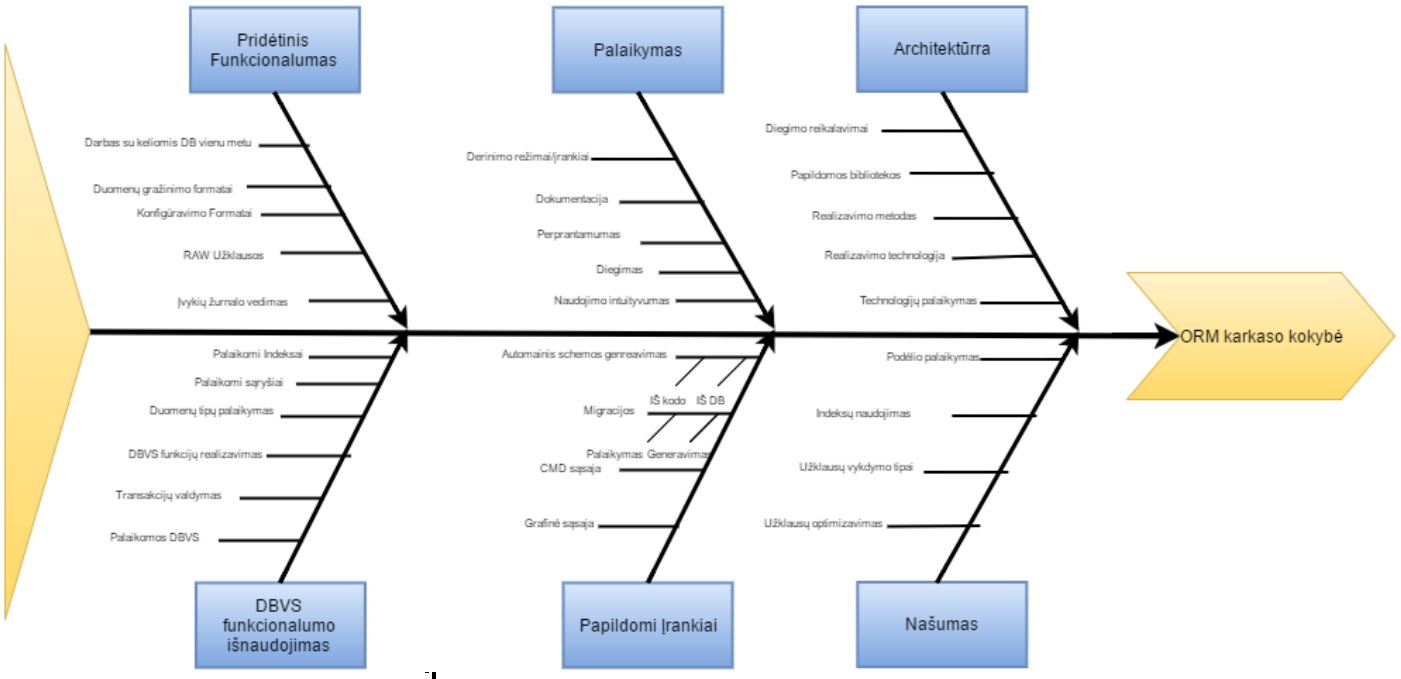
1.2.1. ORM technologijos analizės apibendrinimas

Šiuo metu egzistuoja 4 pagrindiniai metodai ORM realizavimui. Skirtumai tarp kai kurių iš jų pakankamai nežymūs, tad išskiriami tik 2 pagrindiniai tipai – Active Record ir Data Mapping. Abu aptarti metodai yra lygiaverčiai ir kurį pasirinkti konkrečioje situacijoje priklauso nuo kuriamos sistemos dydžio bei specifikos. Active Record yra santykinai paprastesnis ir labiau intuityvus, labiau primenantis objektinio programavimo principus, todėl programuotojui lengviau pradėti su juo dirbti. Tuo tarpu Data Mapper metodas gali būti santykinai sudėtingesnis realizavimo prasme, nes objekto struktūra ir duomenų transformavimas tarp atminties naudojamų tipų ir SQL yra atskirti. Tai reikalauja šiek tiek modifikuoto objektinio programavimo požiūrio, bet tuo pat metu turi daugiau lankstumo, tikslingiau taikomas dideliuose projektuose, kuriuose svarbus atminties naudojimas[14].

ORM karkasai gali ženkliai padidina programavimo produktyvumą, bei aplikacijų kokybę, tačiau pasitaiko atvejų kai ORM naudoti nerekomenduojama. Sistemose, kur ypatingai didelis užklausų skaičius ir duomenų srautai, svarbu kuo labiau optimizuoti visas užklausas. Naudojant ORM tai gali būti gan keblu padaryti. Patogiau rašyti *plain* užklausas žemesniame lygmenyje, nes ORM karkasų panaudojimas įtakoja sistemos sulėtėjimą. Realiai tai įvyksta dėl neefektyvaus užklausų generavimo ir užklausų generavimo iš turimų objektų procesų[14].

2. SIŪLOMAS METODAS ORM KARKASŲ LYGINAMAJAI ANALIZEI

2.1. Veiksniai įtakojanty ORM karkaso kokyę



pav. 11 Faktoriai lemiantys ORM karkaso kokybę

Siekiant Sukurti naują vertinimo metodiką ir išanalizavus lietratūrą |ORM karkasų palyginimo tem, bei įsigilinus į techninius ORM karkasų niuansus buvo išskirtos pagrindinės savybės, kurios įtakoja ORM karkaso kokybę. Pastarosios savybės atvaizduotos diagramoje(žr 11 pav.). Kiekviena savybių grupė bus detalizuojama nurodant jos svarbą.

2.1.1. DBVS funkcionalumo išnaudojimas

Viena iš svarbiausių reliacinių DBVS savybių yra sąryšių realizavimas. Labai svarbu, kad ORM karkasas palaikytų visus pagrindinius sąryšių tipus. Nuo palaikomų saryšių gausos priklauso sistemos plėtojimo galimybės, greitis, bei naudojimo patogumas.

Vienas svarbesnių kriterijų renkantis ORM sistemą yra palaikomų DBVS kiekis. Kuriant IS gali atsirasti poreikis pereiti prie kitos DBVS. Tokiu atveju perėjimo sudėtingumą lemia pasirinkta ORM sistema. Paprasčiausiu atveju, kai nenaudojama jokia ORM sistema, migravimas į naują DBVS būna gana sudėtingas ir reikalauja daug laiko, nes reikia modifikuoti daugumą arba visas užklausas duomenų bazei. Taipogi gali kilti funkcinio nesuderinamumo problemų, kai viena DBVS turi kažkokią funkciją o kita neturi. Tokiu atveju migravimas gali tapti itin sudėtingas. ORM sistema migravimo procesą gali žymiai supaprastinti, jei palaiko abi DBVS: tą kurią IS naudoja šiuo metu ir tą į kurią ketinama migruoti. Tokiu atveju gali pakakti konfigūraciniame faile nurodyti naują DBVS.

Transakcija - veiksmų seka, kuri traktuojama kaip vienas loginis vienetas. Duomenų bazių atveju tai svarbu vykdant keletą užklausų, norint užtikrinti, kad visos užklausos bus ivykdytos sėkmingai. Jei vykdant bent nors viena užklausa įvyksta klaida, anuliuojami visų užklausų rezultatai. Visi veiksmai atliekami sėkmingai, arba neatliekamas nei vienas. ORM karkasas turi turėti galimybę

užtikrinti ACID savybes. Tai labai svarbu norint užtikrinti duomenų bazės vientisumą ir gresnį klaidų valdymą.

Paprastai norint ivykdyti transakciją reikia atlikti tokią veiksmų seką:

- Transakcijos pradėjimas(create). Deklaruojama transakcijos pradžia.
- Vykdomas darbas su duomenų baze.
- Transakcijos patvirtinimas(commit). Jei neįvyko jokių klaidų transakcija patvirtinama. Kol transakcija nėra patvirtinta, duomenų bazėje neatliekami jokie pokeitima;
- Transakcijos atšaukimas(rollback). Jei viena ar kelios užklausos nesėkmingos transakcija atšaukiama ir jokie pakeitimai duomenų bazėje nepadaromi.

Paskutiniai žingsniai(transakcijos patvirtinimas arba atšaukimas), gali būti vykdomas automatiškai arba paliktas programuotojui.

2 lentelė. DBVS funkcionalumo išnaudojimas lemiantis ORM vertinimą

Savybė	**Sąvyės svarba**
Palaikomi indeksai	Indeksai labai svarbi reliacinių DBVS dalis. Indeksų nenaudojimas, arba neteisingas naudojimas, gali drastiškai pailginti užklausų vykdymo laiką.
Palaikomi sąryšiai	Sąryšii viena svarbiausių reliacinių DBVS savybių ,todėl labai svarbu, kad ORM karkasas kokybiškai palaikytų bent pagrindinius sąryšius.
Palaikomos DBVS	Kuo daugiau palaikomų DB, tuo, tikėtina, lengviau pritaikyti konkrečiai IS. Gali pasitaikyti atvejų, kai DBVS keičiama IS kūrimo metu, todėl svarbu, kad ORM karkasas palaikytų kuo daugiau DBVS.
Transakcijų valdymas	Dirbant su reliacinėmis duomenų bazėmis svarbu palaikyti duomenų vintisumą ir integralumą.
DBVS funkcijų realizavimas	Sutrumpina programavimo laiką ir padeda paprasčiau realizuoti standartines funkcijas, tokias kaip COUNT SUM RAND ir pan.

2.1.2. Architektūriniai sprendimai

Architektūriniai sprendimai, vieni svarbiausių faktorių, nulemenčių ORM karkaso greičio bei kitus kokybinius parametrus. Todėl labai svarbu tinkamai įvertinti šiuos parametrus. Pastarieji sprendimai gali nulemti bendrą ORM karkaso našumą, CPU ir RAM atminties sunaudojimą. Taipogi svarbu atkreipti dėmesį ir į realizavimo technologiją, nes nuo to gali priklausyti ar pavyks panaudoti ORM karkasą konkrečioje sistemoje.

3 lentelė. Architektūros sąvybės įtakojančios ORM karkaso vertinimą

Sąvybė	**Sąvybės Svarba**
Realizavimo metodas	Priklausomai nuo realizavimo metodo, gali priklausyti ORM karkaso našumas skirtingo dydžio IS sistemose. Įprastai mažesnės apimties IS, rekomenduojama naudoti Active Record, o didesnės apimties IS Data Mapper. Šis reikalavimas nėra griežtas
Realizavimo technologija	Programavimo kalba kuria parašytas ORM karkasas.
Technologijų palaikymas	Kokiomis programavimo kalbomis galima dirbit ORM karkasu.
CPU naudojimas	Kuo CPU sunaudojama mažiau, tuo sparčiau ir patikimiau veiks IS.
Ram naudojimas	Kuo RAM sunaudojama mažiau, tuo sparčiau ir patikimiau veiks IS.
Papildomų bibliotekų reikalingumas	Jei ORM karkaso veikimui reikia išorinių biliotekų, tuomet mažėja jo patikimumas, nes išorinės bibliotekas gali tapti nebepalaikomos, ar būti nesuderinamos su kuriama IS.

2.1.3. Našumas

Vertinant svarbu atsižvelti į našumą lemiančias savybes, nes nuo jų priklauso, kaip greitai ORM karkasas įvykdys užduotis ir grąžins norimus rezultatus. ORM karkasas pasinaudodamas vidine logika generuoja užklausas kurios yra pritaikytos konkrečiai DBVS. Nuo užklausų kokybės(kompleksiškumo, vykdymo greičio) gali priklausyti ir bendras našumas. Nesudėtingas užklausas dažniausiai generuojamos gana kokybiškai, tačiau daugėjant ssąryšių, bei sąsajų tarp atskirų lentelių užklausų sudėtingumas(dažnai ir vykdymo laikas gali smarkiai išaugti). Tačiau nustatyti generuojamų užklausų kokybę gali būti ne visad paprasta. Jei ORM karkasas leidžia galima peržiūrėti sugeneruotas užklausas ir nuspręsti ar jos adekvačios užduočiai. Tačiau esant sudėtingoms situacijoms gali būti keblu įvertinti ar užklausa yra tinkama ar ne. Dar vienas metodas, įvertinti generuojamoms užklausoms, verinti jų vykdymo greitį. Tačiau tokiu atveju reikia turėti destavimo duomenų rinkinį ir sukonfigūruotą programinę aplinką, kas gali užimti nemažai laiko.

ORM karkasai užklausas dažniausiai gali vykdyti dviem režimais Lazy ir Eager. Pastarieji metodai plačiau analizuojami teoriniame skyriuje, todėl svarbu paminėti, kad lyginant ORM karkasus, jie veiktų vienodame režime. Taipogi geras ORM karkasas turi turėti galimybę keisti užklausų vykdymo režimą.

4 lentelė. Našumą lemiančios savybės

Savybė	Savybės Svarba
Užklausų optimizavimas	Nuo užklausų kokybės ir kompleksiškumo, tiesiogiai priklauso jų vykdymo greitis. Todėl labai svarbu, kad visos generuojamos užklausos būtų vykdomos kuo greičiau
Užklausų vykdymo tipai	Pasirinkus netinkamą užklausų vykdymo tipą, gali nukentėti ORM karkaso greitis, ar stipriai išaugti naudojamų resursų kiekis(RAM, CPU)
Podėlio palaikymas	Podėlio sudarymas(angl. *caching*), tai metodas siekiant paspartinti duomenų gavimą. Todėl tinkamai įgyvendinta podėlio sistema užtikrina spartų ORM karkaso veikimą
Savybė	Savybės Svarba

2.1.4. Papildomi įrankiai

Papildomi įrankiai leidžia pagreitinti ir supaprastinti programavimo darbus naudojant ORM karkasą. Paprastai kuriant IS DB struktūra keičiasi, priklausomai nuo poreikių. Todėl jei ORM karkasas palaiko migracijas, labai supaprastėja DB struktūros keitimo darbai. Programuotojams nebereikia siųsti vienas kitam SQL užklausų. Pakanka gauti naujausius pakeitimus ir ivykdyti atitinkamą komandą.

5 lentelė. Papildomi įrankiai

Savybė	Savybės Svarba
Migracijos	Dažnu atveju DB struktųra keičiasi IS sistemos kūrimo metu. Todėl svarbu, kad programuotojas galėtu programiškai versijuoti DB struktūrą. Tai žymiai supaprastinta DB struktūros keitimo procesą.
DB struktūros generavimas iš kodo	Programuotojui patogu, apsirašius duomenų struktūras, pagal jas sugeneruoti DB struktūrą. Tokiu būdu automatizuojamas darbas.
Kodo generavimas iš DB struktūros	Turint DB struktūrą, patogu pagal ją sugeneruoti Duomenų struktūras aprašančias klases,kurias galima panaudoti programuojant.
CMD sąsaja	Komandinės eilutės dėka galima greitai atlikti pasikartojančias užuotis
Grąfinė sąsaja	Vartotojas gali patogiu būdu valdyti ORM karkaso nustaymus, ar pasinaudoti integruotais įrankiais

2.1.5. Palaikymas

Norint pradėti naudotis ORM karkasu, pirmiausia jį reikai įdiegti. Tai padaryti galiam keliais būdais. Jei naudojamas programavimo karkasas, gali būti, kad jis jau turi įdiegtą vieną ar kitą ORM karkasą. Pavyzdžiui Symfony PHP karkasas turi pagal nutylėjimą įdiegtą Doctrine ORM karkasą, o Laravel PHP karkasas turi įdiegtą Eloquent karkasą.

Norėdami eliminuoti skirtingų PHP karkasų įtaką, PHP karkasai nebus naudojami. Todėl jei įmanoma ORM sistemas reikia įdiegti patiems. Tai padaryti galima keliais būdais, kurie skiriasi sudėtingumu, patogumu, bei tolesniu karkaso valdymu, pavyzdžiu atnaujinimo būdais. Plačiausiai naudojami šitie diegimo būdai:

- Composer - tai įrankis PHP priklausomybių valdymui. Jo pagalba galima paprastai bei greitai įdiegti bei išdiegti norimas PHP bibliotekas ir komponentus iš kuriamos IS. Labai svarbu, kad norint atnaujinti įdiegtą programinę įrangą, užtenka įvykdyti vieną komandą. Tai labai supaprastina sistemos atnaujinimo darbus. Dažnas IS sistemos komponentų Dauguma šiuolaikinių projektų, paremtų PHP kalba naudojamas composer įrankis. Faktiškai dauguma populiariausių PHP karkasų leidžia ir rkomenduoja naudoti Composer įrankį. Todėl svarbu, kad Norimą orm būtų galima įdiegti naudojantis šiuo įrankiu.
- Git - tai versijavimo sistema kurios pagalba taipogi galima įdiegti norimus modulius. Ne toks patogus kaip Composer. Norint pasinaudoti šiuo diegimo metodu sistemoje turi būti įdiegta git versijamo sistema, o norimas komponentas ar biblioteka patalpinti git pagrįstoje sistemoje pvz GitHub ar Bitbucket.
- Manual - Failai yra parsisunčiami ir į IS pajungiami savarankiškai. Šis metodas nereikalauja papildomos programinės įrangos(kaip composer ar GIT), tačiau yra pats lėčiausias. Sudėtingas komponentų atnaujinimas, nes tenka iš naujo paarsiųsti reikalingus failus, o kartais ir pakartoti įjungimo į IS veiksmus. Dėl sudėtingo atnaujinimo kenčia saugumo savybės, nes galimai rečiau atnaujinami sistemos komponentai.

6 lentelė. Palaikymas

Savybė	Savybės Svarba
Perprantamumas	ORM karkasas turėtų būti lengvai perprantamas. Kuo karkasas sunkiau perprantamas, tuo daugiau laiko prireikia jo įsisavinimui.
Derinimo režimai/įrankiai	Labai svarbu, kad ORM karkasas turėtų derinimo(*debug*) režimą. Tai leidžia programuotojui aptikti klaidas ir optimizuoti sistemą.
Diegimas	Kuo daugiau diegimo galimybių, tuo, tikėtina, paprasčiau integruoti įrankį į esamą IS.
Dokumentacija	Dokumentacijos nebuvimas, arba netinkamai parengta/pasenusi dokumentacija, gali būti rimta kliūtimi sprendžiant iškilusias technines problemas. Kuo išsamesnė dokumentacija, tuo, tikėtina, programuotojas greičiau atliks užduotis/
Perprantamumas	Lengviau įsisavinamas įrankis padeda greičiau kurti IS
Intuityvumas	Neintuityvus ORM karkasas, gali lemti lėtą programuotojų darbą, nes reikia nuolat skaityti dokumentaciją.

2.1.6. Pridėtinis funkcionalumas

Pridėtinis funkcionalumas tai papildomos ORM karkaso savybės, kurios lemia naudojimo paprastumą ir patogumą. Kai kurios savybės, pavyzdžiui RAW užklausos yra būtinos, nes paprastai ORM karkasas negali padengti viso DBVS funkcioanalumo. RAW užklausų vykdymas įgalina rašyti užklausas SQL kalba.

7 lentelė. Pridėtinis funkcionalumas

Savybė	Savybės svarba
Darbas su keliomis DB vienu metu	Kartais IS vienu metu dirba su keliomis DB, todėl ORM karkasas turėtų palaikyti, darbą su keliomis DB tuo pačiu metu.
Konfigūravimo formatai	Kuo daugiau konfigūravimo galimybių, tuo labiau tikėtina, kad programuotojas pasirinks labiausiai tinkantį IS specifikai.
RAW užklausos	Šiuo metu ORM karkasai negali padengti ORM funkcionalumo 100%, todėl sudėtingiasniais atvejais svarbu turėti galimybę vykdyti užklausas tiesiogiai
Duomenų gražinimo formatai	Programuojant dažnai prireikia įvairių duomenų formatų, todėl yra svarbu gauti duomenis kuo įvairesniais formatais.
Įvykių žurnalas	Įvykių žurnalas(*log*), gali pasitarnauti sprendžiant iškilusias problemas, bei optimizuojant sistemą.

2.2. Vertinimo kriterijų parinkimas

ORM karkaso kokybę įtakoja visos aptartos savbės. Jos buvo parinktos atsižvelgiant į kitų autorių mokslinius darbus[3, 4, 5], bei remiantis atlika ORM karkasų realizavimo analize[6, 7, 11, 12, 14].

Dalis, anksčiau aptartų, parametrų neįtraukti į vertinimą, siekiant išskirti tik svarbiausius kriterijus, kurie labiausiai įtakoja kokybę. Pasirinkimą įtakojo ir sprendimas metodiką sudaryti taip, kad ORM karkasų vertinimui nereikėtų praktinio jų išbandymo. Toks sprendimas priimtas dėl kelių priežasčių:

- Vertinimas remiantis vien dokumentacija greitenis nei vertinimas įtraukiant ir praktinį išbandymą; Tai įpač aktualu vertinant ORM karkasus kurie veikia skirtingose programavimo aplinkose.
- Vertinimas remiantis vien dokumentacija leidžia eliminuoti papildomas paklaidas ir netikslumus, kuriuos įneša DBVS su kuriomis testuojama. Kiekviena DBVS gali turėti skirtingas konfigūracijas, nuo kurių gali smarkiai priklausyti ORM karkaso našumas ir kiti parametrai;
- Vertininant remiantis vien dokumentacija nereikalaujama, kad vertintojas išmanytu programavimo aplinkas kuriose veikia ORM karkasai.

Taigi visą informaciją, reikalingą ORM vertinimui bus galima rasti ORM karkaso kūrėjų pateikiamoje dokumentacijoje.

8 lentelė. Savybės įtrauktos į vertinimą

Savybės Nr.	Savybė	Tipas	Savybės Nr.	Savybė	Tipas
1	Palaikomos DBVS sistemos	cnt	9	Programinio kodo generavimas iš DB schemos	bin
2	Palaikomi sąryšiai	cnt	10	Transakcijų vykdymas	bin
3	Palaikomi sudėtiniai indeksai	bin	11	Derinimo režimai ir įrankiai	bin
4	Užklausų vykdymo tipai(Eager, Lazy)	cnt	12	Įvykių žurnalas	bin
5	Konfigūravimo metodai	cnt	13	Diegimo galimybės	cnt
6	RAW užklausos	bin	14	Podėlio palaikymas	bin
7	Migracijų palaikymas	bin	15	Duomenų grąžinimo formatai	cnt
8	DB schemos generavimas iš programinio kodo	bin	16	CMD sąsaja	bin

2.2.1. Savybių tipų detalizavimas

Siekiant vertinimą padaryti tikslesnį, savybės buvo suskirstytos į du tipus:

- cnt – savybės, kurias galima įvertinti, suskaičiuojant reikšmių kiekį. Tarkime palaikomos DBVS, palaikomi sąryšiai. Pažymėtina, kad aprašyti vertinimo metodai tinka tik tuomet, kai daugiau savybių yra geriau.
- bin – savybės, kurioms užtenka konstatuoti ar savybė palaikoma(yra/nėra. palaiko/nepalaiko ir t.t.)

Priklausomai nuo tipo, skiriasi ir savybės vertinimas konkrečiu atveju.

Vertinant naudojamas funkcija ***count***, kuri suskaičiuoja baigtinėje aibėje esančius elementus. Pavyzdžiui tarkime, kad turime aibę K kurioje yra tokie elementai *{* r1, r2, r3}. Tokiu atveju count(K) = 3. Formulėse naudojama konstanta 10. Pasirinkimą lėmė sprendimas naudoti dešimtbalę vertinimo sistemą. Konstanta privalo būti teigimas skaičius. Pastaroji konstanta leidžia normalizuoti įvertinimus. Tai reiškia, kad konkrečios savybės įvertinimas visada bus didesnis arba lygus nuliui ir visada mažesnis arba lygus konstantai

2.3. Vertinimas tarpusavyje, neatsižvelgiant į techninius reikalavimus

Šiuo atveju ORM karkasai lyginami tarpusavyje, neturint jokių techninių reikalavimų. Šiuo būdu galima palyginti ORM karkasus tarpusavyje ir nustatyti kuris yra pranašesnis bendru atveju.

2.3.1. Savybių vertinimas(cnt tipas)

Vertinant cnt tipo parametrus pirmiausias sudaroma bendra, lyginamos savybės reikšmių aibė. Turime savybę S. Tarkime, kad:

- ORM1 karkaso S reikšmių aibė: *{ r1, r2, r3}*. Pažymėkime šitą aibę $K1_{match;}$
- ORM2 karkaso S reikšmių aibė *{r2 ir r4}*. Pažymėkime šitą aibę $K2_{match;}$

Tokiu atveju bendra reikšmių aibė bus: *{ r1, r2, r3, r4}*. Šitą reikšmių aibę pažymėsime raide K.

Pažymėkime ORM1 S savybės įvertinimą P_1. Tuomet:

$$P_1 = count(K1_{match}) * \frac{10}{count(K)} = 3 * \frac{10}{4} = 7.5;$$

Pažymėkime ORM2 S savybės įvertinimą P_2. Tuomet:

$$P_2 = count(K2_{match}) * \frac{10}{count(K)} = 2 * \frac{10}{4} = 4;$$

Pateiktame pavyzdyje apskaičiuojame savybės S įvertinimo balus ORM1 ir ORM2 ORM karkasams.

2.3.2. Savybių vertinimas(bin tipas)

Jei savybė yra bin tipo, užtenka nustatyti jos egzistavimą ORM karkase, t.y. ar ORM karkasas turi šią savybę.

Turime savybę S. Tarkime, kad:

- ORM1 karkasas palaiko savybę S; Tokiu atveju $K1_{match}$ reikšmė lygi 1;
- ORM2 karkasas nepalaiko savybės S; Tokiu atveju $K2_{match}$ reikšmė lygi 0;

Pažymėkime ORM1 S savybės įvertinimą P_1. Tuomet:

$$P_1 = K1_{match} * 10 = 1 * 10 = 10;$$

Pažymėkime ORM2 S savybės įvertinimą P_2. Tuomet:

$$P_2 = K1_{match} * 10 = 0 * 10 = 0;$$

$$T = \sum_{i=1}^{count(P)} \begin{cases} K_{match_i} * \frac{10}{count(K_i)}, & type_i = cnt \\ K_{match_i} * 10, & type_i = bin \end{cases}$$

9 lentelė. Balo skaičiavimo formulės kintamųjų detalizavimas

Kintamasis	Paaiškinimas
T	Galutinis ORM karkaso įverinimas balais. Didesnė T reikšmė reiškia geresnį įvertinimą.
P	Tiriamų savybių aibė.
i	Dabartinės tiriamos savybės numeris.
K_{match}	Savybės reikšmių skaičius, kurias turi ORM karkasas.
K	Savybės skirtingų reikšmių kiekis.

2.4. Griežtas ORM karkasų palyginimas, atsižvelgiant į turimus techninius reikalavimus.

Turint tecninių reikalavimų specifikaciją, vieni parametrai tampa svarbesni už kitus. Pavyzdžiui, jei nurodoma, kad IS turi naudoti MySQL ar Postgre SQL DBVS, tai ORM karkaso kitos palaikomos duomenų bazės tampa nebe tokios svarbios. Tačiau svarbu kad palaikytų pastarasias DBVS.

Norint tiksliau palyginti kuris ORM karkasas tinka konkrečiu atveju, buvo nuspresta modifikuoti balo skaičiavimo formulę. Vietoje bendros reikšmių aibės, imama tik reikalaujamų reikšmių aibė.

Kadangi remiamasi technine dokumentacija, gali pasitaikyti atvejų, kai kažkuri vertinama savybė nėra dokumentuota ir/arba jos nereikalaujama. Kad tokios savybės neiškraipytų vertinimo ir

neįtakotų galutinio balo nuspręsta įvesti papildomą koeficientą q, kurio reikšmė gali būti 0 arba 1. Jei koeficiento reikšmė 0, jis panaikina savybės vertinimą ir neįtakoja galutinio rezultato. Svarbu atkreipti dėmesį, kad koeficientas q tai pačiai savybei turi išlikti vienodas nepriklausomai nuo ORM karkaso.

2.4.1. Savybių vertinimas(cnt tipas)

Turime savybę S. Tarkime, kad:

- ORM1 karkaso S reikšmių aibė: *{ r1, r2, r3}*. Čia r1, r2, r3 savybės reikšmės. Pažymėkime šitą aibę K1;
- ORM2 karkaso S reikšmių aibė *{r2 ir r4}* Čia r2, r4 savybės reikšmės.. Pažymėkime šitą aibę K2

Tarkime, kad turime, techninius reikalavimus ir juose nurodoma, kad ORM karkasas privalo palaikyti savybę S kuri turi tokias reikšmes *{r2, r4}*. Šitą reikšmių aibę pažymėsime raide K_{req}. Tai reiškia, kad iš ORM karkasoi reikalaujama kad savybė S turėtų arba *r2* reikšmę arba *r4*. To pasekoje count(K_{req}) lygus 2;

Kadangi savybė privaloma, koeficiento q reikšmė lygi 1;

Pažymėkime ORM1 S savybės įvertinimą P_1. Tuomet:

$$P_1 = q\left(count(K1_{match}) * \frac{10}{count(K_{req})}\right) = 1\left(1 * \frac{10}{2}\right) = 5;$$

Pažymėkime ORM2 S savybės įvertinimą P_2. Tuomet:

$$P_2 = q\left(count(K2_{match}) * \frac{10}{count(K_{req})}\right) = 1\left(2 * \frac{10}{2}\right) = \mathbf{10};$$

Pateiktame pavyzdyje apskaičiuojame savybės S įvertinimo balus ORM1 ir ORM2 ORM karkasams. Kaip matome iš skaičiavimų, šiuo atveju didesnį įvertinimą lėmė, ne savybės reikšmių skaičius, o tai, kiek reikalaujamų reikšmių palaikė.

2.4.2. Savybių vertinimas(bin tipas)

Jei savybė yra bin tipo, užtenka nustatyti jos egzistavimą ORM karkase, t.y. ar ORM karkasas turi šią savybę. Tarkime savybė yra privaloma, taigi koeficientas q lygus 1;

Turime savybę S. Tarkime, kad:

- ORM1 karkasas palaiko savybę S; Tokiu atveju $K1_{match}$ reikšmė lygi 1;
- ORM2 karkasas nepalaiko savybės S; Tokiu atveju $K2_{match}$ reikšmė lygi 0;

Pažymėkime ORM1 S savybės įvertinimą P_1. Tuomet:

$$P_1 = q(K1_{match} * 10) = 1(1 * 10) = 10;$$

Pažymėkime ORM2 S savybės įvertinimą P_2. Tuomet:

$$P_2 = q(K1_{match} * 10) = 1(0 * 10) = 0;$$

$$T\_H = \sum_{i=1}^{count(P)} q\left(\begin{cases} K_{match_i} * \dfrac{10}{count\left(K_{req_i}\right)}, & type_i = cnt \\ K_{match_i} * 10, & type_i = bin \end{cases}\right)$$

10 lentelė. Griežto palyginimo formulės kintamųjų detalizavimas

Kintamasis	Paaiškinimas
T_H	Galutinis ORM karkaso įverinimas balais. idesnė T reikšmė reiškia geresnį įvertinimą.
P	Tiriamų savybių aibė.
i	Dabrtinės tiriamos savybės numeris.
K_{match}	Savybės reikšmių, kurios tenkina techninius reikalavimus skaičius, kurias turi ORM karkasas.
K_{req}	Tinkamų savybės reikšmių skaičius.
type	Savybės tipas.
q	Koeficientas nurodantis savybės aktualumą. Gali įgyti tris reikšmes: 0 – Savybė neaktuali ir į galutinio balo skaičiavimą neįįtraukiama; 0,5 – savybė neprivaloma, bet pageidautina; 1 – savybė aktuali ir turi būti įtraukta į galutinio balo skaičiavimą;

2.5. Lankstus ORM karkasų palyginimas , atsižvelgiant į turimus techninius reikalavimus.

Svarbu suprasti, kad idialiai tinkačių ORM karkasų, vargu ar pavyks surasti - kiekviena IS turi unikalius reikalavimussavo realizacijai. Kad būtų galima parinkti labiausiai tinkantį ORM karkasą, nuspręstą papildyti balo skaičiavimo formulę taip, kad rezultatą nulemtų ir palaikomų reikšmių skaičius, ir techniniai reikalavimai.

2.5.1. Savybių vertinimas(cnt tipas)

Turime savybę S. Tarkime, kad:

- ORM1 karkaso S reikšmių aibė: *{ r1, r2, r3}*. Pažymėkime šitą aibę $ORM_{K1;}$
- ORM2 karkaso S reikšmių aibė *{r2 ir r4}*. Pažymėkime šitą aibę $ORM_{K2;}$ Šitą reikšmių aibę pažymėsime raide

Tarkime, kad turime, techninius reikalavimus ir juose nurodoma, kad ORM karkasas privalo palaikyti savybę S kuri turi tokias reikšmes *{r2, r4}*. Šitą reikšmių aibę pažymėsime raide K_{req}. Tai reiškia, kad iš ORM karkasoi reikalaujama kad savybė S turėtų arba *r2* reikšmę arba *r4*. To pasekoje count(K_{req}) lygus 2; Kadangi savybė pageidautina, bet neprivaloma, koeficiento q reikšmė lygi 0.5;

Pažymėkime ORM1 S savybės įvertinimą P_1. Tuomet:

$$P_1 = \frac{q}{2}\left(count(ORM_{K1}) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = 0.25\left(3 * \frac{10}{4} + 1 * \frac{10}{2}\right) = 3{,}1;$$

Pažymėkime ORM2 S savybės įvertinimą P_2. Tuomet:

$$P_2 = \frac{q}{2}\left(count(ORM_{K2}) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = 0.25\left(2 * \frac{10}{4} + 2 * \frac{10}{2}\right) = 3{,}8;$$

2.5.2. Savybių vertinimas(bin tipas)

Jei savybė yra bin tipo, užtenka nustatyti jos egzistavimą ORM karkase, t.y. ar ORM karkasas turi šią savybę. Kadangi savybė pageidautina, bet neprivaloma, koeficiento q reikšmė lygi 0.5;

Turime savybę S. Tarkime, kad:

- ORM1 karkasas palaiko savybę S; Tokiu atveju $K1_{match}$ reikšmė lygi 1;
- ORM2 karkasas nepalaiko savybės S; Tokiu atveju $K2_{match}$ reikšmė lygi 0;

Pažymėkime ORM1 S savybės įvertinimą P_1. Tuomet:

$$P_1 = q(K1_{match} * 10) = 0{,}5(1 * 10) = 5;$$

Pažymėkime ORM2 S savybės įvertinimą P_2. Tuomet:

$$P_2 = q(K2_{match} * 10) = 0{,}5(0 * 10) = 0;$$

$$T\_S = \sum_{i=1}^{count(P)} q\left(\begin{cases} \dfrac{count(ORM_{K_i}) * \dfrac{10}{count(K_i)} + K_{match_i} * \dfrac{10}{count(K_{req_i})}}{2}, & type_i = cnt \\ K_{match_i} * 10, & type_i = bin \end{cases}\right)$$

11 lentelė. Lankstaus palyginimo formulės kintamųjų detalizavimas

Kintamasis	Paaiškinimas
T_S	Galutinis ORM karkaso įverinimas balais. Didesnė T reikšmė reiškia geresnį įvertinimą.
P	Tiriamų savybių aibė.
i	Dabrtinės tiriamos savybės numeris.
ORM_K	Konkretaus ORM karkaso savybės reikšmių skaičius
K_{match}	Savybės reikšmių, kurios tenkina techninius reikalavimus skaičius, kurias turi ORM karkasas.
K_{req}	Tinkamų savybės reikšmių skaičius.
K	Savybės reikšmių skaičius.
type	Savybės tipas. Gali įgyti dvi reikšmes. cnt – kiekybinis tipas; val - kokybinis tipas.

Nepriklausomai nuo naudojamo vertinimo metodo galutinį įvertinimą gauname skaičių. Pastarasis skaičius nurodo karkaso . Kuo didesnis įvertinimo skaičius tuokarkasas palaiko daugiau funkcijų ir labiau atitinka iškeltus techninius reikalavimus, jeigu tokie buvo iškelti(vertinimas atsižvelgiant į techninius reikalavimus).

Siekiant sukurti ORM karkasų įvertinimo metodiką, buvo išskirtos svarbiausios savybės lemiančios jo kokybę. Pastarosios savybės atrinktos išanalizavus ORM karkasų veikimo principus, bei egzistuojančius ORM palyginimo metodus. Pasiūlytas lyginamosios analizės modelis leidžia itin lanksčiai ir paprastai įvertinti ORM karkasus ir juos palyginti tarpusavyje. Lankstumą lemia tai, kad išskirti trys palyginimo metodai:

- Palyginimas be techninės dokumentacijos;
- Griežtas palyginimas remiantis technine dokumentacija;
- Lankstus palyginimas remiantis technine dokumentacija.

Taigi vertintojas gali pasirinkti labiausiai, konkrečioje situacijoje, tinkantį palygino metodą. Paprastumą lemia tai, kad vertintojui nereikia atlikti praktinio ORM karkasų išbandymo. Taigi vertintojas neprivalo mokėti dirbti tomis technologijomis,kurių pagalba veikia ORM karkasas.

3. PHP KALBAI SKIRTŲ ORM KARKASŲ PALYGINIMAS

Norint išbandyti ORM karkasų vertinimo metodą buvo atlikta keletas eksperimentų. Kad parodyti siūlomos metodologijos veikimo tesingumą eksperimentas susidės iš dviejų dalių. Pirmiausia tiriamieji ORM karkasai įvertinami remiantis sukurta vertinimo metodologija, o vėliau pasitelktas ekspertinis vertinimas. Daroma prielaida, kad jei vertinimas, gautas remiantis metodologija, bus panašus į ekspertų pateiką vertinimą, metodologija yra teisinga.

Atliekant vertinmą pagal siūlomą metodologiją, kiekvienos savybės vertinimo proces buvo smulkiai detalizuotas, aprašant atliekamus veiksmus, bei naudojamus kintamuosius. Taip siekiama pademonstruoti metodologijos veikimą taip palenkvinant tolimesnį jos taikymą.

Norint atlikti sudarytos metodologijos testavimą, buvo reikalinga techninė specifikacija. Norint kuo labiau įvertinti pasiūlyto metodo tinkamumą, buvo pasirinkta realios IS techninė specifikacija. Pastaroji IS buvo kuriama, norint realizuoti komunikavimo ir bendradarbiavimo platformą tarp odontologijos klinikų ir dantų technikų laboratorijų. Iš pastarosios specifikacijos buvo išrinkti metodologijai reikalingi kriterijai ir surašyti į lentelę.

Kadangi iš užsakovo nebūvo konkrečių techninių reikalavimų, buvo svarstomi keli ORM variantai:

- Eloquent;
- Doctrine;
- NotORM.

Pastarieji ir bus analizuojami ir vertinant siūlomą metodologją.

12 lentelė. Techniniai reikalavimai

Savybės Nr.	Savybė	Tipas	Reikalavimai savybei
1	Palaikomos DBVS sistemos	cnt	MySQL arba Oracle
2	Palaikomi sąryšiai	cnt	Visi pagrindiniai
3	Sudėtiniai indeksai	bin	Neprivaloma
4	Užklausų vykdymo tipai(Eager, Lazy)	cnt	Privalo palaikyti Eager ir Lazy tipus
5	Konfigūravimo metodai	cnt	Privaloma, kad palaikytų Array arba YAML
6	RAW užklausos	bin	Privalo palaikyti
7	Migracijų palaikymas	bin	Neprivaloma, bet pageidautina
8	DB schemos generavimas iš programinio kodo	bin	Neprivaloma, bet pageidautina
9	Programinio kodo generavimas iš DB schemos	bin	Nereikalaujama
10	Transakcijų vykdymas	bin	Privalomas palaikynas
11	Derinimo režimai ir įrankiai	bin	Neprivaloma, bet pageidautina
12	Dokumentacijos kokybė	val	Neprivaloma, bet pageidautina
13	Diegimo galimybės	cnt	Privalo palaikyti diegimą composer įrankiu
14	Podėlio palaikymas	bin	Nereikalaujama
15	Duomenų grąžinimo formatai	cnt	Neprivaloma, bet pageidautina
16	CMD sąsaja	bin	Neprivaloma, bet pageidautina

3.1. Palaikomos DBVS sistemos

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

13 lentelė. Lyginamų ORM palaikomos DBVS

ORM Karkasas	Palaikomos DBVS	Palaikomu DBVS kiekis(ORM_K)
Eloquent	MySQL, Postgres, SQL Server, SQLite;	4
Doctrine	MySQL, Postgre, Oracle, SQLite;	4
NotORM	MySQL, SQLite, Postgre, SQL Server, Oracle;	5

Sudarome bendrą reikšmių aibę:

K:{MySQL, Postgres, MS SQL Server, SQLite, Oracle}. Matome, kad aibėje K yra 5 elementai.

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas *q*, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju reikalaujama, kad palaikytų vieną iš dviejų DBVS, taigi savybė yra svarbi. To pasekoje q = 1.

Kadangi reikalaujama, kad ORM karkasas palaikyų vieną iš dviejų DBVS(MySQL arba Oracle), kintamasis K_{req} lygus 2.

14 lentelė. Kintamųjų reikšmės, kurios bendros visuose palyginimuose pagal DBVS

Kintamasis	Reikšmė
count(K)	5
q	1
count(K_{req})	2

3.1.1. Lyginimas be techninių reikalavimų

Lyginant neatsižvelgiant į techninius reikalavimus, kiekviena, ORM karkasui reikia rasti K_{match} reikšmę. Pastaroji reikšmė lygi garkaso palaikomų DBVS kiekiui.

15 lentelė. K_{match} *reikšmės lyginant pagal palaikomas DBVS. Neatsižvelgian į techninius reikalavimus*

ORM karkasas	K_{match} Reikšmė
Eloquent	4 (palaiko 4 DBVS iš 5 galimų)
Doctrine	4 (palaiko 4 DBVS iš 5 galimų)
NotORM	5 (palaiko 5 DBVS iš 5 galimų)

Suskaičiuojame įvertinimus visiems ORM karkasasms:

Eloquent: $T_1 = K_{match} * \frac{10}{count(K)} = 4 * \frac{10}{5} = 8.$

Doctrine: $T_1 = K_{match} * \frac{10}{count(K)} = 4 * \frac{10}{5} = 8.$

NotORM: $T_1 = K_{match} * \frac{10}{count(K)} = 5 * \frac{10}{5} = 10.$

3.1.2. Griežtas lyginimas pagal techninius reikalavimus

Šiuo atveju K_{match} reikšmė bus ORM karkaso savybės reikšmių, kurios tenkina techninius sreikalavimus, skaičius.

16 lentelė. K_{match} *reikšmės lyginant griežtai pagal palaikomas DBVS*

ORM karkasas	K_{match} **Reikšmė**
Eloquent	1 (palaiko 1 DBVS iš 2 reikalaujamų)
Doctrine	1 (palaiko 1 DBVS iš 2 reikalaujamų)
NotORM	2 (palaiko 2 DBVS iš 2 reikalaujamų)

Suskaičiuojame įvertinimus, visiems ORM karkasasms:

Eloquent: $T\_H_1 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(1 * \frac{10}{2}\right) = 5.$

Doctrine: $T\_H_1 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(2 * \frac{10}{2}\right) = 10.$

NotORM: $T\_H_1 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(2 * \frac{10}{2}\right) = 10.$

3.1.3. Lankstus lyginimas pagal techninius reikalavimus

Kiekvienam lyginamam ORM karkasui surandame K_{match} reikšmes:

17 lentelė. K_{match} *reikšmės lyginant lanksčiai pagal palaikomas DBVS*

ORM karkasas	K_{match} **Reikšmė**
Eloquent	1 (palaiko 1 DBVS iš 2 reikalaujamų)
Doctrine	1 (palaiko 1 DBVS iš 2 reikalaujamų)
NotORM	2 (palaiko 2 DBVS iš 2 reikalaujamų)

Eloquent: $T\_S_1 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(4 * \frac{10}{5} + 1 * \frac{10}{2}\right) = 6.5.$

Doctrine: $T\_S_1 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(4 * \frac{10}{5} + 1 * \frac{10}{2}\right) = 6{,}5.$

NotORM: $T\_S_1 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(5 * \frac{10}{5} + 2 * \frac{10}{2}\right) = 10.$

18 lentelė. Palaikomų DBVS palygino rezultatai.

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	8	5	6.5
Doctrine	8	10	6.5
NotORM	10	10	10

3.2. Palaikomi sąryšiai

Savybės tipas – **cnt**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kokius sąryšio tipus palaiko kiekvienas tiriamas ORM karkasas.

19 lentelė. ORM karkasų palaikomi saryšiai

ORM	Vienas su vienu	Vienas su daug	Daug su daug	Turi daug per	Polimorfiniai sąryšiai	Daug su daug polimorfiniai sąryšiai	(ORM_K)
Eloquent	Taip	Taip	Taip	Taip	Taip	Taip	6
Doctrine	Taip	Taip	Taip	Ne	Taip	Ne	4
NotORM	Taip	Taip	Taip	Ne	Ne	Ne	3

Sudarome bendrą reikšmių aibę:

K:{ Vienas su vienu, Vienas su daug, Daug su daug, Turi daug per, Polimorfiniai sąryšiai

, Daug su daug polimorfiniai sąryšiai }. Matome, kad aibėje K yra 6 elementai.

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas q, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju reikalaujama, , kad ORM karkasas palaikytų pagrindinius sąryšio tipus. To pasekoje $q = 1$.

Kadangi reikalaujama, kad ORM karkasas palaikyų Pagrindinius sąryšius, kintamasis K_{req} lygus 3.

20 lentelė. Kintamųjų reikšmės, kurios bendros visuose palyginimuose pagal palaikomus sąryšius

Kintamasis	Reikšmė
count(K)	6
q	1
count(K_{req})	3

3.2.1. Lyginimas be techninių reikalavimų

Lyginant neatsižvelgiant į techninius reikalavimus, kiekviena, ORM karkasui reikia rasti K_{match} reikšmę. Pastaroji reikšmė lygi ORM karkaso palaikomų sąryšių kiekiui.

21 lentelė. Kmatch reikšmės lyginant pagal palaikomus sąryšius. Neatsižvelgian į techninius reikalavimus

ORM karkasas	K_{match} Reikšmė
Eloquent	6 (palaiko 6 Sąryšius iš 6 galimų)
Doctrine	4 (palaiko 4 Sąryšius iš 6 galimų)
NotORM	3 (palaiko 3 Sąryšius iš 6 galimų)

Suskaičiuojame įvertinimus visiems ORM karkasasms:

Eloquent: $T_2 = K_{match} * \frac{10}{count(K)} = 6 * \frac{10}{6} = 10;$

Doctrine: $T_2 = K_{match} * \frac{10}{count(K)} = 4 * \frac{10}{6} = 6{,}6;$

NotORM: $T_2 = K_{match} * \frac{10}{count(K)} = 3 * \frac{10}{6} = 5;$

3.2.2. Griežtas lyginimas pagal techninius reikalavimus

Šiuo atveju K_{match} reikšmė bus ORM karkaso savybės reikšmių, kurios tenkina techninius sreikalavimus, skaičius.

22 lentelė. Kmatch reikšmės lyginant griežtai pagal palaikomus sąryšius

ORM karkasas	K_{match} **Reikšmė**
Eloquent	3 (palaiko 3 DBVS iš 3 reikalaujamų)
Doctrine	3 (palaiko 3 DBVS iš 3 reikalaujamų)
NotORM	3 (palaiko 3 DBVS iš 3 reikalaujamų)

Suskaičiuojame įvertinimus, visiems ORM karkasasms:

Eloquent: $T\_H_2 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(3 * \frac{10}{3}\right) = 10;$

Doctrine: $T\_H_2 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(3 * \frac{10}{3}\right) = 10;$

NotORM: $T\_H_2 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(3 * \frac{10}{3}\right) = 10;$

3.2.3. Lankstus lyginimas pagal techninius reikalavimus

Kiekvienam lyginamam ORM karkasui surandame K_{match} reikšmes:

23 lentelė. Kmatch reikšmės lyginant lanksčiai pagal palaikomussąryšius

ORM karkasas	K_{match} **Reikšmė**
Eloquent	3 (palaiko 3 DBVS iš 3 reikalaujamų)
Doctrine	3 (palaiko 3 DBVS iš 3 reikalaujamų)
NotORM	3 (palaiko 3 DBVS iš 3 reikalaujamų)

Eloquent: $T\_S_2 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(6 * \frac{10}{6} + 3 * \frac{10}{3}\right) = 10;$

Doctrine: $T\_S_2 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(4 * \frac{10}{6} + 3 * \frac{10}{3}\right) = 8.3;$

NotORM: $T\_S_2 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(3 * \frac{10}{6} + 3 * \frac{10}{3}\right) = 7.5;$

24 lentelė. Palaikomų sąryšių palygino rezultatai.

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	10	10
Doctrine	6,66	10	8,33
NotORM	5	10	7,5

3.3. Sudėtiniai indeksai

Savybės tipas – **cnt**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

25 lentelė. Lyginamų ORM sudėtinių indeksų palaikymas

ORM Karkasas	Indeksų palaikymas
Eloquent	Ne
Doctrine	Taip
NotORM	Taip

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas q, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju nereikalaujama, kad palaikytų sudėtinius indeksus, taigi savybė yra nėrasvarbi. To pasekoje q = 0.

26 lentelė. Kintamųjų reikšmės, kurios bendros visuose palyginimuose pagal sudėtinius indeksus

Kintamasis	**Reikšmė**
q	0

3.3.1. Lyginimas be techninių reikalavimų

Lyginant neatsižvelgiant į techninius reikalavimus, kiekviena, ORM karkasui reikia rasti K_{match} reikšmę. Pastaroji reikšmė lygi garkaso palaikomų DBVS kiekiui.

27 lentelė. Kmatch reikšmės, lyginant be techninių reikalavimų, pagal sudėtinius inedksus

ORM karkasas	K_{match} **Reikšmė**
Eloquent	0 (nepalaiko sudėtinių indeksų)
Doctrine	1 (palaiko sudėtinius indeksus)
NotORM	0 (nepalaiko sudėtinių indeksų)

Eloquent: $T_3 = K_{match} * 10 = 0 * 10 = 0$;

Doctrine: $T_3 = K_{match} * 10 = 1 * 10 = 10$;

NotORM: $T_3 = K_{match} * 10 = 1 * 10 = 10$;

3.3.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_3 = 0$;

Doctrine: $T\_H_3 = 0$;

NotORM: $T_H = 0$;

3.3.3. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_3 = 0$;

Doctrine: $T\_S_3 = 0$;

NotORM: $T\_S_3 = 0$;

28 lentelė. Palaikomų DBVS palygino rezultatai.

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	0	0	0
Doctrine	10	0	0
NotORM	10	0	0

3.4. Užklausų vykdymo tipai

Savybės tipas – **cnt**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

29 lentelė. Lyginamų ORM užklausų vykdymo tipai

ORM Karkasas	Užklausų vykdymo tipai	Palaikomu tipų kiekis(ORM_K)
Eloquent	Lazy, Eager	2
Doctrine	Lazy, Eager	2
NotORM	Eager	1

Sudarome bendrą reikšmių aibę:

K:{ Lazy, Eager}. Matome, kad aibėje K yra 2 elementai.

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas *q*, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju reikalaujama, kad palaikytų Lazy ir Eager tipus, taigi savybė yra svarbi. To pasekoje q = 1

Kadangi reikalaujama, kad ORM karkasas palaikyų Lazy ir Eager tipus, kintamasis K_{req} lygus 2.

30 lentelė. Kintamųjų reikšmės, kurios bendros visuose palyginimuose pagal užklausų vykdymo tipus

Kintamasis	**Reikšmė**
count(K)	2
q	1
count(K_{req})	2

3.4.1. Lyginimas be techninių reikalavimų

Lyginant neatsižvelgiant į techninius reikalavimus, kiekviena, ORM karkasui reikia rasti K_{match} reikšmę. Pastaroji reikšmė lygi garkaso palaikomų DBVS kiekiui.

31 lentelė. K_{match} reikšmės „Užklausų vykdymo tipai“ savybei. Be techninių reikalavimų.

ORM karkasas	**K_{match} Reikšmė**
Eloquent	2 (palaiko 2 DBVS iš 2 galimų)
Doctrine	2(palaiko 2 DBVS iš 2 galimų)
NotORM	1 (palaiko 1 DBVS iš 2 galimų)

Suskaičiuojame įvertinimus visiems ORM karkasasms:

Eloquent: $T_4 = K_{match} * \frac{10}{count(K)} = 2 * \frac{10}{2} = 10;$

Doctrine: $T_4 = K_{match} * \frac{10}{count(K)} = 2 * \frac{10}{2} = 10;$

NotORM: $T_4 = K_{match} * \frac{10}{count(K)} = 1 * \frac{10}{2} = 5;$

3.4.2. Griežtas lyginimas pagal techninius reikalavimus

Šiuo atveju K_{match} reikšmė bus ORM karkaso savybės reikšmių, kurios tenkina techninius sreikalavimus, skaičius.

32 lentelė. K_{match} reikšmės „Užklausų vykdymo tipai“ savybei. Griežtas palyginimas

ORM karkasas	**K_{match} Reikšmė**
Eloquent	2 (palaiko 2 DBVS iš 2 reikalaujamų)
Doctrine	2 (palaiko 2 DBVS iš 2 reikalaujamų)
NotORM	1 (palaiko 1 DBVS iš 2 reikalaujamų)

Suskaičiuojame įvertinimus, visiems ORM karkasasms:

Eloquent: $T\_H_4 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(2 * \frac{10}{2}\right) = 10;$

Doctrine: $T\_H_4 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(2 * \frac{10}{2}\right) = 10;$

NotORM: $T\_H_4 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(1 * \frac{10}{2}\right) = 5;$

3.4.3. Lankstus lyginimas pagal techninius reikalavimus

Kiekvienam lyginamam ORM karkasui surandame K_{match} reikšmes:

33 lentelė. K_{match} reikšmės „Užklausų vykdymo tipai“ savybei. Lankstus palyginimas

ORM karkasas	***K_{match}* Reikšmė**
Eloquent	1 (palaiko 1 DBVS iš 2 reikalaujamų)
Doctrine	1 (palaiko 1 DBVS iš 2 reikalaujamų)
NotORM	2 (palaiko 2 DBVS iš 2 reikalaujamų)

Eloquent: $T\_S_4 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(2 * \frac{10}{2} + 2 * \frac{10}{2}\right) = 10;$

Doctrine: $T\_S_4 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(2 * \frac{10}{2} + 2 * \frac{10}{2}\right) = 10;$

NotORM: $T\_S_4 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(2 * \frac{10}{2} + 1 * \frac{10}{2}\right) = 5;$

34 lentelė. Savybės „Užklausų vykdymo tipai“ balų suvestinė.

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	10	10
Doctrine	10	10	10
NotORM	5	5	5

3.5. Konfigūravimo metodai

Savybės tipas – **cnt;**

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

35 lentelė. Lyginamų ORM konfigūravimo metodai

ORM Karkasas	**Palaikomos DBVS**	**Palaikomu DBVS kiekis*(ORM_K)***
Eloquent	XML, YAML, Array	3
Doctrine	XML, YAML, Array	3
NotORM	Array	3

Reikia paminėti, kad Doctrine plaiko Annotations konfigūravimo būdą, bet pastarasis naudojamas tik šio yrankio, todėl į vertinimą neįtrauktas.

Sudarome bendrą reikšmių aibę:

K:{XML, YAML, Array}. Matome, kad aibėje K yra 3 elementai.

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas *q*, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju reikalaujama, kad palaikytų vieną iš dviejų konfigūravimo metodų, taigi savybė yra svarbi. To pasekoje q = 1.

Kadangi reikalaujama, kad ORM karkasas palaikyų vieną iš dviejų konfigūravimo metodų(Array arba YML), kintamasis count(K_{req}) lygus 2.

36 lentelė. Kintamųjų reikšmės, kurios bendros visuose palyginimuose pagal konfigūravimo metodus

Kintamasis	Reikšmė
count(K)	3
q	1
count(K_{req})	2

3.5.1. Lyginimas be techninių reikalavimų

Lyginant neatsižvelgiant į techninius reikalavimus, kiekviena, ORM karkasui reikia rasti K_{match} reikšmę. Pastaroji reikšmė lygi garkaso palaikomų DBVS kiekiui.

37 lentelė. $K_{match\ reikšmės}$ lyginant pagal konfigūravimo metodą. Neatsižvelgiant į dokumentaciją.reikšmės

ORM karkasas	K_{match} Reikšmė
Eloquent	2 (palaiko 2 konfigūravimo būdus iš 3 galimų)
Doctrine	3 (palaiko 3 konfigūravimo būdus iš 3 galimų)
NotORM	1 (palaiko 1 konfigūravimo būdus iš 3 galimų)

Suskaičiuojame įvertinimus visiems ORM karkasasms:

Eloquent: $T_1 = K_{match} * \frac{10}{count(K)} = 2 * \frac{10}{3} = 6.6;$

Doctrine: $T_1 = K_{match} * \frac{10}{count(K)} = 3 * \frac{10}{3} = 10;$

NotORM: $T_1 = K_{match} * \frac{10}{count(K)} = 1 * \frac{10}{3} = 3.3;$

3.5.2. Griežtas lyginimas pagal techninius reikalavimus

Šiuo atveju K_{match} reikšmė bus ORM karkaso savybės reikšmių, kurios tenkina techninius sreikalavimus, skaičius.

38 lentelė. Kmatch reikšmės lyginant pagal palaikomus konfigūravimo būdus.Griežto palyginmo būdas

ORM karkasas	K_{match} Reikšmė
Eloquent	1 (palaiko 1 konfigūravimo būdą iš 2 reikalaujamų)
Doctrine	2 (palaiko 2 konfigūravimo būdus iš 2 reikalaujamų)
NotORM	1 (palaiko 1 konfigūravimo būdą iš 2 reikalaujamų)

Suskaičiuojame įvertinimus, visiems ORM karkasasms:

Eloquent: $T\_H_1 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(1 * \frac{10}{2}\right) = 5;$

Doctrine: $T\_H_1 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(2 * \frac{10}{2}\right) = 10;$

NotORM: $T\_H_1 = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(1 * \frac{10}{2}\right) = 5;$

3.5.3. Lankstus lyginimas pagal techninius reikalavimus

Kiekvienam lyginamam ORM karkasui surandame K_{match} reikšmes:

39 lentelė. Kmatch reikšmės lyginant pagal palaikomus konfigūravimo būdus. Lankstaus palyginmo būdas

ORM karkasas	K_{match} Reikšmė
Eloquent	1 (palaiko 1 konfigūravimo būdą iš 2 reikalaujamų)
Doctrine	1 (palaiko 2 konfigūravimo būdus iš 2 reikalaujamų)
NotORM	2 (palaiko 1 konfigūravimo būdą iš 2 reikalaujamų)

Eloquent: $T\_S_1 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(2 * \frac{10}{3} + 1 * \frac{10}{2}\right) = 5.8;$

Doctrine: $T\_S_1 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(3 * \frac{10}{3} + 2 * \frac{10}{2}\right) = 10;$

NotORM: $T\_S_1 = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(1 * \frac{10}{3} + 1 * \frac{10}{2}\right) = 4.2;$

40 lentelė. Palaikomų konfigūravimo metodų palygino rezultatai.

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	6,66	5	5,83
Doctrine	10	10	10
NotORM	3,33	5	4,2

3.6. RAW užklausos

Savybės tipas – **bin**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

41 lentelė. Lyginamų ORM RAW užklausų palaikymas

ORM Karkasas	RAW užklausų palaikymas
Eloquent	Taip
Doctrine	Taip
NotORM	Ne

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas q, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju reikalaujama, kad palaikytų RAW užklausas, taigi savybė yra svarbi. To pasekoje q = 1.

42 lentelė. Kintamųjų reikšmės savybei: RAW užklausos

Kintamasis	Reikšmė
q	1

Kadangi savybė yra bin tipo, K_{matc} reikšmė išlieka vienoda visiems vertinimo variantams.

43 lentelė. K_{match} reikšmės „RAW užklausos“ savybei

ORM karkasas	K_{match} Reikšmė
Eloquent	1 (palaiko RAW užklausas)
Doctrine	1 (palaiko RAW užklausas)
NotORM	0 (nepalaiko RAW užklausų)

3.6.1. Lyginimas be techninių reikalavimų

Eloquent: $T_6 = K_{match} * 10 = 1 * 10 = 10;$

Doctrine: $T_6 = K_{match} * 10 = 1 * 10 = 10;$

NotORM: $T_6 = K_{match} * 10 = 0 * 10 = 0;$

3.6.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_6 = q(K_{match} * 10) = 1(1 * 10) = 10;$

Doctrine: $T\_H_6 = q(K_{match} * 10) = 1(1 * 10) = 10;$

NotORM: $T\_H_6 = q(K_{match} * 10) = 1(0 * 10) = 0;$

3.6.3. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_6 = q(K_{match} * 10) = 1(1 * 10) = 10;$

Doctrine: $T\_S_6 = q(K_{match} * 10) = 1(1 * 10) = 10;$

NotORM: $T\_S_6 = q(K_{match} * 10) = 1(0 * 10) = 0;$

44 lentelė. Savybės „RAW užklausos“ balų suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	10	10
Doctrine	10	10	10
NotORM	0	0	0

3.7. Migracijos

Savybės tipas – **bin**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

45 lentelė. Lyginamų ORM migracijų palaikymas

ORM Karkasas	Migracijų palaikymas
Eloquent	Taip
Doctrine	Taip
NotORM	Ne

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas *q*, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju migracijų palaikymas nėra privalomas, bet pageidautinas. To pasekoje q = 0,5.

46 lentelė. Kintamųjų reikšmės savybei „Migracijos“

Kintamasis	Reikšmė
q	0,5

Kadangi savybė yra bin tipo, K_{matc} reikšmė išlieka vienoda visiems vertinimo variantams.

47 lentelė. K_{match} reikšmės „Migracijos“ užklausos savybei

ORM karkasas	K_{match} Reikšmė
Eloquent	1 (palaiko migracijas)
Doctrine	1 (palaiko migracijas)
NotORM	0 (nepalaiko migracijų)

3.7.1. Lyginimas be techninių reikalavimų

Eloquent: $T_7 = K_{match} * 10 = 1 * 10 = 10;$

Doctrine: $T_7 = K_{match} * 10 = 1 * 10 = 10;$

NotORM: $T_7 = K_{match} * 10 = 0 * 10 = 0;$

3.7.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_7 = q(K_{match} * 10) = 0{,}5(1 * 10) = 5;$

Doctrine: $T\_H_7 = q(K_{match} * 10) = 0{,}5(1 * 10) = 5;$

NotORM: $T\_H_7 = q(K_{match} * 10) = 0{,}5(0 * 10) = 0;$

3.7.3. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_7 = q(K_{match} * 10) = 0{,}5(1 * 10) = 5;$

Doctrine: $T\_S_7 = q(K_{match} * 10) = 0{,}5(1 * 10) = 5;$

NotORM: $T\_S_7 = q(K_{match} * 10) = 0{,}5(0 * 10) = 0;$

48 lentelė. Savybės „migracijos“ balų suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	5	5
Doctrine	10	5	5
NotORM	0	0	0

3.8. DB schemos generavimas iš programinio kodo

Savybės tipas – **bin**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

49 lentelė. Lyginamų ORM DB schemos generavimas iš programinio kodo palaikymas

ORM Karkasas	DB schemos generavimas iš programinio kodo
Eloquent	Ne
Doctrine	Taip
NotORM	Ne

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas q, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju DB schemos generavimas iš programinio kodo nėra privalomas, bet pageidautinas. To pasekoje q = 0,5.

50 lentelė. Kintamųjų reikšmės savybei „DB schemos generavimas iš programinio kodo“.

Kintamasis	Reikšmė
q	0,5

Kadangi savybė yra bin tipo, K_{matc} reikšmė išlieka vienoda visiems vertinimo variantams.

51 lentelė. K_{match} reikšmės „DB schemos generavimas iš programinio kodo“ užklausos savybei

ORM karkasas	K_{match} Reikšmė
Eloquent	0 (nepalaiko DB schemos generavimo iš programinio kodo)
Doctrine	1 (palaiko DB schemos generavimą iš programinio kodo)
NotORM	0 (nepalaiko DB schemos generavimo iš programinio kodo)

3.8.1. Lyginimas be techninių reikalavimų

Eloquent: $T_8 = K_{match} * 10 = 0 * 10 = 0;$

Doctrine: $T_8 = K_{match} * 10 = 1 * 10 = 10;$

NotORM: $T_8 = K_{match} * 10 = 0 * 10 = 0;$

3.8.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_8 = q(K_{match} * 10) = 0{,}5(0 * 10) = 0;$

Doctrine: $T\_H_8 = q(K_{match} * 10) = 0{,}5(1 * 10) = 5;$

NotORM: $T\_H_8 = q(K_{match} * 10) = 0{,}5(0 * 10) = 0;$

Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_8 = q(K_{match} * 10) = 0{,}5(0 * 10) = 0;$

Doctrine: $T\_S_8 = q(K_{match} * 10) = 0{,}5(1 * 10) = 5;$

NotORM: $T\_S_8 = q(K_{match} * 10) = 0{,}5(0 * 10) = 0;$

52 lentelė. Savybės „DB schemos generavimas iš programinio kodo“balų suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	0	0	0
Doctrine	10	5	5
NotORM	0	0	0

3.9. Kodo generavimas iš DB schemos

Savybės tipas – **bin**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

53 lentelė. Lyginamų ORM Kodo generavimas iš DB schemos palaikymas

ORM Karkasas	Kodo generavimas iš DB schemos
Eloquent	Ne
Doctrine	Taip
NotORM	Ne

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas q, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju kodo generavimo iš DB schemos nereikalaujama . To pasekoje q = 0.

54 lentelė. Kintamųjų reikšmės savybei: kodo generavimo iš DB schemos

Kintamasis	Reikšmė

Kadangi savybė yra bin tipo, K_{matc} reikšmė išlieka vienoda visiems vertinimo variantams.

55 lentelė. K_{match} reikšmės „Kodo generavimas iš DB schemos“ užklausos savybei

ORM karkasas	K_{match} Reikšmė
Eloquent	0 (nepalaiko kodo generavimo iš DB schemos)
Doctrine	1 (palaiko kodo generavimą iš DB schemos)
NotORM	0 (nepalaiko kodo generavimo iš DB schemos)

3.9.1. Lyginimas be techninių reikalavimų

Eloquent: $T_9 = K_{match} * 10 = 0 * 10 = 0;$

Doctrine: $T_9 = K_{match} * 10 = 1 * 10 = 10;$

NotORM: $T_9 = K_{match} * 10 = 0 * 10 = 0;$

3.9.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_9 = 0;$

Doctrine: $T\_H_9 = 0;$

NotORM: $T\_H_9 = 0;$

3.9.3. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_9 = 0;$

Doctrine: $T\_S_9 = 0;$

NotORM: $T\_S_9 = 0;$

56 lentelė. Savybės „Kodo generavimas iš DB schemos“balų suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	0	0	0
Doctrine	10	0	0
NotORM	0	0	0

3.10. Transakcijų vykdymas

Savybės tipas – **bin**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

57 lentelė. Lyginamų ORM transakcijų palaikymas

ORM Karkasas	RAW užklausų palaikymas
Eloquent	Taip
Doctrine	Taip
NotORM	Taip

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas q, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju reikalaujama, kad palaikytų transakcijų vykdymą, taigi savybė yra svarbi. To pasekoje q = 1.

58 lentelė. Kintamųjų reikšmės savybei: Transakcijų vykdymas

Kintamasis	Reikšmė
q	1

Kadangi savybė yra bin tipo, K_{matc} reikšmė išlieka vienoda visiems vertinimo variantams.

59 lentelė. K_{match} reikšmės „Transakcijų vykdymasusos“ savybei

ORM karkasas	K_{match} Reikšmė
Eloquent	1 (palaiko transakcijų vykdymą)
Doctrine	1 (palaiko transakcijų vykdymą)
NotORM	1 (palaiko transakcijų vykdymą)

3.10.1. Lyginimas be techninių reikalavimų

Eloquent: $T_{10} = K_{match} * 10 = 1 * 10 = 10;$

Doctrine: $T_{10} = K_{match} * 10 = 1 * 10 = 10;$

NotORM: $T_{10} = K_{match} * 10 = 1 * 10 = 10;$

3.10.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_{10} = q(K_{match} * 10) = 1(1 * 10) = 10;$

Doctrine: $T\_H_{10} = q(K_{match} * 10) = 1(1 * 10) = 10;$

NotORM: $T\_H_{10} = q(K_{match} * 10) = 1(1 * 10) = 10;$

3.10.3. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_{10} = q(K_{match} * 10) = 1(1 * 10) = 10;$

Doctrine: $T\_S_{10} = q(K_{match} * 10) = 1(1 * 10) = 10;$

NotORM: $T\_S_{10} = q(K_{match} * 10) = 1(1 * 10) = 10;$

60 lentelė. Savybės „Transakcijų vykdymas“ balų suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	10	10
Doctrine	10	10	10
NotORM	10	10	10

3.11. Derinimo režimas/įrankiai

Savybės tipas – **bin**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

61 lentelė. Lyginamų ORM derinimo režimo/įrankių palaikymas

ORM Karkasas	Derinimo režimas/įrankiai
Eloquent	Taip
Doctrine	Taip
NotORM	Taip

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas q, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju ši savybė nėra privaloma, tačiau pageidautina, kad ORM karkasas turėtų derinimo režimą arba įrankius. To pasekoje q = 0,5.

62 lentelė. Kintamųjų reikšmės savybei: Derinimo režimas/įrankiai

Kintamasis	Reikšmė
q	0,5

Kadangi savybė yra bin tipo, K_{matc} reikšmė išlieka vienoda visiems vertinimo variantams.

63 lentelė. K_{match} reikšmės „Derinimo režimas/įrankiai“ savybei

ORM karkasas	K_{match} Reikšmė
Eloquent	1 (turi derinimo režimą/įrankius)
Doctrine	1 (turi derinimo režimą/įrankius)
NotORM	1 (turi derinimo režimą/įrankius)

3.11.1. Lyginimas be techninių reikalavimų

Eloquent: $T_{11} = K_{match} * 10 = 1 * 10 = 10;$

Doctrine: $T_{11} = K_{match} * 10 = 1 * 10 = 10;$

NotORM: $T_{11} = K_{match} * 10 = 1 * 10 = 10;$

3.11.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_{11} = q(K_{match} * 10) = 0.5(1 * 10) = 5;$

Doctrine: $T\_H_{11} = q(K_{match} * 10) = 0.5(1 * 10) = 5;$

NotORM: $T\_H_{11} = q(K_{match} * 10) = 0.5(1 * 10) = 5;$

3.11.3. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_{11} = q(K_{match} * 10) = 0.5(1 * 10) = 5;$

Doctrine: $T\_S_{11} = q(K_{match} * 10) = 0.5(1 * 10) = 5;$

NotORM: $T\_S_{11} = q(K_{match} * 10) = 0.5(1 * 10) = 5;$

64 lentelė. Savybės „Derinimo režimas/įrankiai suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	5	5
Doctrine	10	5	5
NotORM	10	5	5

3.12. Įvykių žurnalas

Savybės tipas – **bin**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

65 lentelė. Lyginamų ORM karkasų įvykių žurna;o palaikymas

ORM Karkasas	Įvykių žurnalas
Eloquent	Ne
Doctrine	Taip
NotORM	Ne

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas q, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju nereikalaujama, kad palaikytų įvykių žurnalą, taigi savybė nėra svarbi. To pasekoje $q = 0$.

66 lentelė. Kintamųjų reikšmės savybei: įvykių žurnalas

Kintamasis	Reikšmė
q	0

Kadangi savybė yra bin tipo, K_{matc} reikšmė išlieka vienoda visiems vertinimo variantams.

67 lentelė. K_{match} reikšmės „Įvykių žurnalas“ savybei

ORM karkasas	K_{match} Reikšmė
Eloquent	0 (nepalaiko įvykių žurnalo)
Doctrine	1 (palaiko įvykių žurnalą)
NotORM	0 (nepalaiko įvykių žurnalo)

3.12.1. Lyginimas be techninių reikalavimų

Eloquent: $T_{12} = K_{match} * 10 = 1 * 10 = 0;$

Doctrine: $T_{12} = K_{match} * 10 = 1 * 10 = 10;$

NotORM: $T_{12} = K_{match} * 10 = 0 * 10 = 0;$

3.12.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_{12} = 0;$

Doctrine: $T\_H_{12} = 0;$

NotORM: $T\_H_{12} = 0;$

3.12.3. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_{12} = 0;$

Doctrine: $T\_S_{12} = 0;$

NotORM: $T\_S_{12} = 0;$

68 lentelė. Savybės „Įvykių žurnalas“ balų suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	10	10
Doctrine	10	10	10

3.13. Diegimo galimybės

Savybės tipas: **cnt**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

69 lentelė. Lyginamų ORM diegimo galimybės

ORM Karkasas	Diegimo galimybės	Diegimo galimybiųkiekis(ORM_K)
Eloquent	Composer, GIT, Manual	3
Doctrine	Composer, GIT, Manual	3
NotORM	GIT, Manual	5

Sudarome bendrą reikšmių aibę:

K:{ Composer, GIT, Manual}. Matome, kad aibėje K yra 3 elementai.

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas *q*, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju reikalaujama, kad palaikytų composer diegimo metodą, taigi savybė yra svarbi. To pasekoje q = 1.

Kadangi reikalaujama, kad ORM karkasas palaikyų vieną diegimo būdąComposer), kintamasis K_{req} lygus 1.

70 lentelė. Kintamųjų reikšmės, kurios bendros visuose palyginimuose pagal diegimo galimybes

Kintamasis	Reikšmė
count(K)	3
q	1
count(K_{req})	1

3.13.1. Lyginimas be techninių reikalavimų

Lyginant neatsižvelgiant į techninius reikalavimus, kiekviena, ORM karkasui reikia rasti K_{match} reikšmę. Pastaroji reikšmė lygi ORM karkaso palaikomų diegimo galimybių kiekiui.

71 lentelė. K_{match} r eikšmės lyginant pagal diegimo galimybes. Neatsižvelgiant į techninius reikalavimus

ORM karkasas	K_{match} Reikšmė
Eloquent	3 (palaiko3 diegimo būdus iš 3 galimų)
Doctrine	3 (palaiko3 diegimo būdus iš 3 galimų)
NotORM	2 (palaiko2 diegimo būdus iš 3 galimų)

Suskaičiuojame įvertinimus visiems ORM karkasasms:

Eloquent: $T_{13} = K_{match} * \frac{10}{count(K)} = 3 * \frac{10}{3} = 10;$

Doctrine: $T_{13} = K_{match} * \frac{10}{count(K)} = 3 * \frac{10}{3} = 10;$

NotORM: $T_{13} = K_{match} * \frac{10}{count(K)} = 2 * \frac{10}{3} = 6.6;$

3.13.2. Griežtas lyginimas pagal techninius reikalavimus

Šiuo atveju K_{match} reikšmė bus ORM karkaso savybės reikšmių, kurios tenkina techninius sreikalavimus, skaičius.

72 lentelė. K_{match} *reikšmės lyginant pagal diegimo galimybse.Griežtas palyginimas.*

ORM karkasas	K_{match} **Reikšmė**
Eloquent	1 (palaiko1 diegimo būdą iš 1 galimo)
Doctrine	1 (palaiko1 diegimo būdą iš 1 galimo)
NotORM	0 (palaiko0 diegimo būdų iš 1 galimo)

Suskaičiuojame įvertinimus, visiems ORM karkasasms:

Eloquent: $T\_H_{13} = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(1 * \frac{10}{1}\right) = 10;$

Doctrine: $T\_H_{13} = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(1 * \frac{10}{1}\right) = 10;$

NotORM: $T\_H_{13} = q\left(K_{match} * \frac{10}{count(K_{req})}\right) = 1\left(0 * \frac{10}{1}\right) = 0;$

3.13.3. Lankstus lyginimas pagal techninius reikalavimus

Kiekvienam lyginamam ORM karkasui surandame K_{match} reikšmes:

73 lentelė. K_{match} *lreikšmės lyginant pagal diegimo galimybes Lankstus palyginimas*

ORM karkasas	K_{match} **Reikšmė**
Eloquent	1 (palaiko1 diegimo būdą iš 1 galimo)
Doctrine	1 (palaiko1 diegimo būdą iš 1 galimo)
NotORM	0 (palaiko0 diegimo būdų iš 1 galimo)

Eloquent: $T\_S_{13} = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(3 * \frac{10}{3} + 1 * \frac{10}{1}\right) = 10;$

Doctrine: $T\_S_{13} = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(3 * \frac{10}{3} + 1 * \frac{10}{1}\right) = 10;$

NotORM: $T\_S_{13} = \frac{q}{2}\left(count(ORM_K) * \frac{10}{count(K)} + K_{match} * \frac{10}{count(K_{req})}\right) = \frac{1}{2}\left(2 * \frac{10}{3} + 0 * \frac{10}{1}\right) = 6.6;$

74 lentelė. Savybės „Diegimo galimybės“ balų suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	10	10
Doctrine	10	10	10
NotORM	6.66	0	6,6

3.14. Podėlio palaikymas

Savybės tipas – **bin**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

75 lentelė. Lyginamų ORM podėlio palaikymas

ORM Karkasas	**Podėlio palaikymas**
Eloquent	Taip
Doctrine	Taip
NotORM	Taip

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas *q*, kuris nurodo ar savybė yra svarbi projektui. Šiuo reikalaujama, kad ORM karkasas palaikytų podėlį. To pasekoje q = 1.

76 lentelė. Kintamųjų reikšmės savybei: Podėlio palaikymas

Kintamasis	**Reikšmė**
q	1

Kadangi savybė yra bin tipo, K_{matc} reikšmė išlieka vienoda visiems vertinimo variantams.

77 lentelė. K_{match} reikšmės „Podėlio palaikymas“ savybei

ORM karkasas	K_{match} **Reikšmė**
Eloquent	1 (palaiko podėlį)
Doctrine	1 (palaiko podėlį)
NotORM	1 (palaiko podėlį)

3.14.1. Lyginimas be techninių reikalavimų

Eloquent: $T_{14} = K_{match} * 10 \ = \ 1 * 10 \ = \ 10;$

Doctrine: $T_{14} = K_{match} * 10 \ = \ 1 * 10 \ = \ 10;$

NotORM: $T_{14} = K_{match} * 10 \ = \ 1 * 10 \ = \ 10;$

3.14.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_{14} = q(K_{match} * 10\ ) \ = \ 1(1 * \ 10) = \ 10;$

Doctrine: $T\_H_{14} = q(K_{match} * 10\ ) \ = \ 1(1 * \ 10) = \ 10;$

NotORM: $T\_H_{11} = q(K_{match} * 10\ ) \ = \ 1(1 * \ 10) = \ 10;$

3.14.3. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_{14} = q(K_{match} * 10\ ) \ = \ 1(1 * \ 10) = \ 10;$

Doctrine: $T\_S_{14} = q(K_{match} * 10\ ) \ = \ 1(1 * \ 10) = \ 10;$

NotORM: $T\_T_{14} = q(K_{match} * 10\ ) \ = \ 1(1 * \ 10) = \ 10;$

78 lentelė. Savybės „Podėlio palaikymas balų suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	10	10
Doctrine	10	10	10
NotORM	10	10	10

3.15. Gražinamų duomenų formatai

Savybės tipas – **cnt;**

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

79 lentelė. Lyginamų ORM grąžinamų duomenų formatai

ORM Karkasas	**Palaikomos DBVS**	**Palaikomu DBVS kiekis(ORM_K)**
Eloquent	Array, JSON	2
Doctrine	Array, JSON	2
NotORM	Array	1

Sudarome bendrą reikšmių aibę:

K:{Array, JSON}. Matome, kad aibėje K yra 2 elementai.

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas *q*, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju nereikalaujama, kad palaikytų konkretų duomenų formatą, taigi savybė nėra svarbi. To pasekoje q = 0.

80 lentelė. Kintamųjų reikšmės, kurios bendros visuose palyginimuose pagal grąžinamų duomenų formatus

Kintamasis	**Reikšmė**
count(K)	2
q	0

3.15.1. Lyginimas be techninių reikalavimų

Lyginant neatsižvelgiant į techninius reikalavimus, kiekviena, ORM karkasui reikia rasti K_{match} reikšmę. Pastaroji reikšmė lygi garkaso palaikomų DBVS kiekiui.

81 lentelė. $K_{match\ reikšmės}$ *lyginant pagal grąžinamus formatus. Neatsižvelgiant į dokumentaciją.reikšmės*

ORM karkasas	K_{match} **Reikšmė**
Eloquent	2 (palaiko 2 duomenų formatus iš 2 galimų)
Doctrine	2 (palaiko 2 duomenų formatus iš 2 galimų)
NotORM	1 (palaiko 1 duomenų formatą iš 2 galimų)

3.15.2. Lyginimas be techninių reikalavimų

Eloquent: $T_{15} = K_{match} * \frac{10}{count(K)} = 2 * \frac{10}{2} = 10;$

Doctrine: $T_{15} = K_{match} * \frac{10}{count(K)} = 2 * \frac{10}{2} = 10;$

NotORM: $T_{15} = K_{match} * \frac{10}{count(K)} = 1 * \frac{10}{2} = 5;$

3.15.3. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_{15} = 0;$

Doctrine: $T\_H_{15} = 0;$

NotORM: $T\_H_{15} = 0;$

3.15.4. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_{15} = 0;$

Doctrine: $T\_S_{15} = 0;$

NotORM: $T\_S_{8} = 0;$

82 lentelė. Grąžinamų duomenų formatų palygino rezultatai.

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	0	0
Doctrine	10	0	0
NotORM	5	0	0

3.16. CMD sąsaja

Savybės tipas – **bin**.

Analizuojant ORM karkasų dokumentacijas išsiaiškiname, kad:

83 lentelė. Lyginamų ORM podėlio palaikymas

ORM Karkasas	**CMD sąsaja**
Eloquent	Taip

Doctrine	Taip
NotORM	Ne

Lyginant atsižvelgiant į techninius reikalavimus atsiranda papildomas koeficientas q, kuris nurodo ar savybė yra svarbi projektui. Šiuo atveju nereikalaujama, kad CMD sąsaja būtų palaikoma privalomai, bet palaikymas pageidautimas. To pasekoje q = 0.5.

84 lentelė. Kintamųjų reikšmės savybei: CMD sąsaja

Kintamasis	**Reikšmė**
q	0.5

Kadangi savybė yra bin tipo, K_{matc} reikšmė išlieka vienoda visiems vertinimo variantams.

85 lentelė. K_{match} reikšmės „CMD sąsaja“ savybei

ORM karkasas	K_{match} **Reikšmė**
Eloquent	1 (palaiko podėlį)
Doctrine	1 (palaiko podėlį)
NotORM	1 (palaiko podėlį)

3.16.1. Lyginimas be techninių reikalavimų

Eloquent: $T_{16} = K_{match} * 10 \ = \ 1 * 10 \ = \ 10;$

Doctrine: $T_{16} = K_{match} * 10 \ = \ 1 * 10 \ = \ 10;$

NotORM: $T_{16} = K_{match} * 10 \ = \ 1 * 10 \ = \ 10;$

3.16.2. Griežtas lyginimas pagal techninius reikalavimus

Eloquent: $T\_H_{16} = q(K_{match} * 10 \ ) \ = \ 1(1 * \ 10) = \ 10;$

Doctrine: $T\_H_{16} = q(K_{match} * 10 \ ) \ = \ 1(1 * \ 10) = \ 10;$

NotORM: $T\_H_{16} = q(K_{match} * 10 \ ) \ = \ 1(1 * \ 10) = \ 10;$

3.16.3. Lankstus lyginimas pagal techninius reikalavimus

Eloquent: $T\_S_{16} = q(K_{match} * 10 \ ) \ = \ 1(1 * \ 10) = \ 10;$

Doctrine: $T\_S_{16} = q(K_{match} * 10 \ ) \ = \ 1(1 * \ 10) = \ 10;$

NotORM: $T\_S_{16} = q(K_{match} * 10 \ ) \ = \ 1(1 * \ 10) = \ 10;$

86 lentelė. Savybės „CMD sąsaja balų“ suvestinė

	Lyginimas be techninių reikalavimų	Griežtas lyginimas pagal techninius reikalavimus	Lankstus lyginimas pagal techninius reikalavimus
Eloquent	10	10	10
Doctrine	10	10	10
NotORM	10	10	10

3.17. Galutinio balo skaičiavimas.

Suskaičiavome atskirų savybių įvertinimus, taigi apskaičiuojame galutinius balus visiems vertinimo metodams. Tam susumuojame visų savybių balus.

3.17.1. Galutinis įvertinimo balas, kai neatsižvelgiama į techninius reikalavimus

87 lentelė. Galutinis įvertinimas, neatsižvelgiant į techninius reikalavimus.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Suma
Eloquent	8	10	0	10	6,6	10	0	0	0	10	10	10	10	10	10	10	**114.6**
Doctrine	8	6,6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	**154.6**
NotORM	10	5	10	5	3,3	0	0	0	0	10	10	10	6,6	10	5	10	**94.9**

Matome, kad didžiausią balą gavo Doctrine ORM karkasas, kuris surinko 154,6 balo iš 160 galimų. Antroje vietoje liko Eloquent ORM karkasas surinkęs 114,6 balo. Trečioje vietoje liko NotORM karkasas su 94,9 balo.

Kadangi lyginant šiuo būdu neatsižvelgiama į techninius reikalavimus, galima teigti jog įvertinimą nulėmė palaikomų funkcijų gausa. Todėl remiantis šiuo vertinimu galima konstatuoti jog Doctrine ORM karkasas yra lyderis pagal palaikomą funkcionalumą.

3.17.2. Galutinis įvertinimo balas, kai neatsižvelgiama į techninius reikalavimus

88 lentelė. Galutinis įvertinimas, vertinant griežtai pagal techninius reikalavimus

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Suma
Eloquent	5	10	0	10	5	10	0	0	0	10	10	5	10	10	0	10	**95**
Doctrine	10	10	0	10	10	10	5	0	0	10	10	5	10	10	0	10	**110**
NotORM	10	10	0	5	5	0	0	0	0	10	10	5	0	10	0	10	**75**

Suskaičiavus įvertinimus matome, kad Doctrin ORM karkasas toliau išliko pirmoje vietoje, tačiau atotrūkis tarp lyginamų ORM karkasų šiektiek sumažėjo.

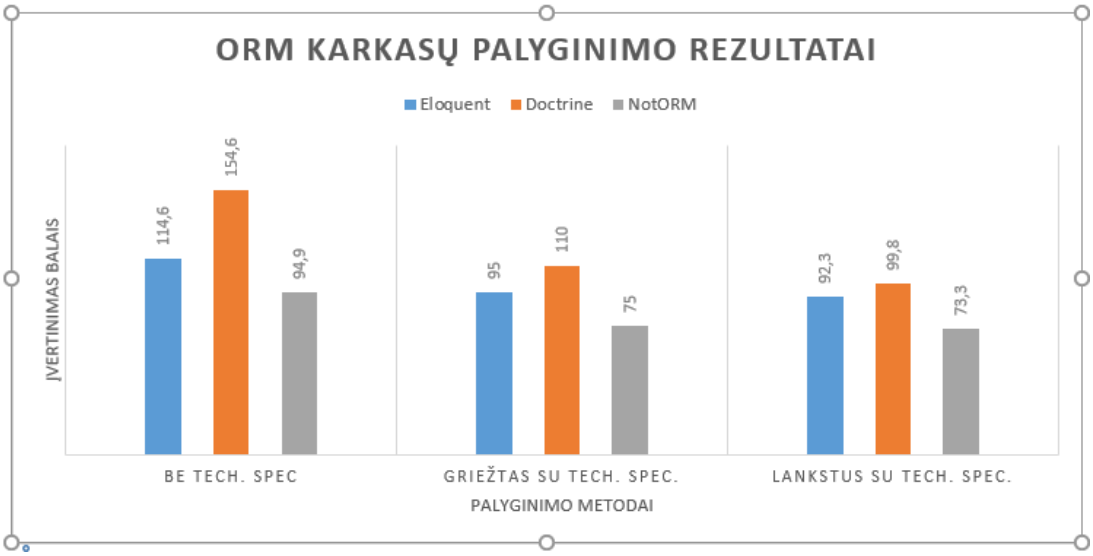
Paminėtina, kad visi lyginami karkasai palaikė visas privalomai reikalaujamas savybes, todėl nebuvo atmestas nei vienas karkasas.

3.17.3. Lankstus vertinimas

89 lentelė. Galutinis įvertinimas, veertinant lanksčiai pagal techninius reikalavimus

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Suma
Eloquent	6,5	10	0	10	5,8	10	0	0	0	5	10	5	10	10	0	10	**92.3**
Doctrine	6,5	8,3	0	10	10	10	5	0	0	5	10	5	10	10	0	10	**99.8**

Suskaičiavus tarpinius balus matome, kad lyderio poziciją toliau išlaikė Doctrine ORM karkasas. Tačiau atotrūkis tarp lyginamųORM karkasų sumažėjo dar labiau.



pav. 12 ORM palyginimo rezultatai

Iš pateikto grafiko(žr. 12 pav.) matome, kad vertinant trimis metodais, visuose trijuose lyderis išliko Doctrine ORM karkasas. Tačiau reikia pastebėti, kad lyginant be techninių reikalavimų Doctrine Pirmavimas buvo labai ryškus, net 40 balų prieš antroje vietoje likusį laravel karkasą. Tai nulėmė palaikomo funkcioanalumo gausa.

Į vertinimą įtraukus techninius reikalavimus galime pasteebėti keletą pokyčių. Pirmiausia sumažėjo bendras bįvertinimas balais. Taip nutiko dėl to, kad kai kurios savybės nebuvo svarbios kuriamai sistemai, todėl koeficientas q buvo 0. To pasekoje tos savybės įvertinimas irgi tapdavo 0. Tokiu būdu įvertinimą formavo tik tos savybės, kurios svarbios remiantis kuriamos IS sistemos reikalavimais.

Doctrine ORM karkasas išliko lyderiu, bet atotrūkis tar lyginamų karkasų labai sumažėjo. Lyginant be techninių reikalavimų atotrūkis buvo 40 balų. Atsiradus techniniams reikalavimams vienu atveju atotrūkis buvo 15 balų(griežtas lyginimas), kitu atveju 7,5 balo(lankstus palyginimas). Pastarąjį pokytį įtakojo techninių reikalavimų įvedimas, ko pasekoje įvertinimą nulėmė ne tik palaikomo funkcionalumo gausa, bet techninių reikalavimų atitikimas.

3.18. Programuotojų apklausa

Norint patvirtinti lyginimo metodo teisingumą nuspresta atlikti PHP programuotojų apklausą. Buvo sukurta anketa, su klausimais apie patirtį programuojant PHP, naudojamus programavimo karkasus it pan. Galiausiai buvo paprašyta įvertinti tiriamus ORM karkasus. Jei siūlome vertinimo metodika teisinga vertinimo balai turėtų gautis panašųs tiek skaičiuojant pagal metodiką, tiek vertinant ekspertams.

Apklausoje dalyvavo 18 respondentų, kurie specializuojasi PHP programavime. Apklausa buvo anoniminė, todėl nereikalauta jokių asmeninių duomenų. Atliekant apklausą buvo prašoma sureitinguoti tiriamus ORM karkasus:

1. Sureitinguokite ORM karkasus nuo geriausio iki blogiausio. Jei neteko dirbti su konkrečiu ORM remkitės dokumentacija:

- Eloquent;
- Doctrine;
- NotORM;

Gautus apklausos rezultatus surašome į lentelę, taip paprasčiau ir patogiau apdoroti duomenis.

90 lentelė. Respodentų apklausos rezultatai

Respondentai	Reitingavimo rezultatai
Respondentas1	Eloquent, Doctrine, NotORM
Respondentas2	Doctrine, Eloquent, NotORM
Respondentas3	Doctrine, Eloquent, NotORM
Respondentas4	Eloquent, Doctrine, NotORM
Respondentas5	Doctrine, Eloquent, NotORM
Respondentas6	Eloquent, Doctrine, NotORM
Respondentas7	Eloquent, Doctrine, NotORM
Respondentas8	Eloquent, Doctrine, NotORM
Respondentas9	Doctrine,Eloquent, NotORM
Respondentas10	Eloquent, Doctrine, NotORM
Respondentas11	Doctrine,NotORMEloquent,
Respondentas12	Eloquent, Doctrine, NotORM
Respondentas13	Doctrine,Eloquent, NotORM
Respondentas14	Doctrine,Eloquent,NotORM
Respondentas15	NotORM, Eloquent, Doctrine
Respondentas16	Doctrine, Eloquent, NotORM
Respondentas17	Doctrine,Eloquent, NotORM
Respondentas18	NotORM, Eloquent, Doctrine

Respondentų buvo paprašyta įvertinti lyginamus ORM, nuo, jų manymu, geriausio iki blogiausio. Net jei respondentas tiesiogiai nedirbo su konkrečiu ORM karkasu, buvo paprašyta peržvelgti pastarojo dokumentaciją ir įvertinti. Pirmoje vietoje esančiam ORM karkasui skiriami 3 taškai, antroje 2, taškaii, o paskutinėje 1 taškas. Reikia paminėti, kad vertinimą galėjo įtakoti eksperto dažniausiai naudojamas programavimo karkasas, nes Simfony karkasas, pagal nutylėjimą naudoja Doctrine, o Laravel karkasas Eloquent ORM karkasą.

Vertinimo rezultatai:

- Eloquent įvertinimas: *3 * 6 + 2 * 9 + 1 * 1 = 40;*
- Doctrine įvertinimas: *3 * 9 + 2 * 7 + 1*2 = 43;*
- NotORM įvertinimas: *3 * 2 + 2 * 1 + 1 * 15 = 21;*

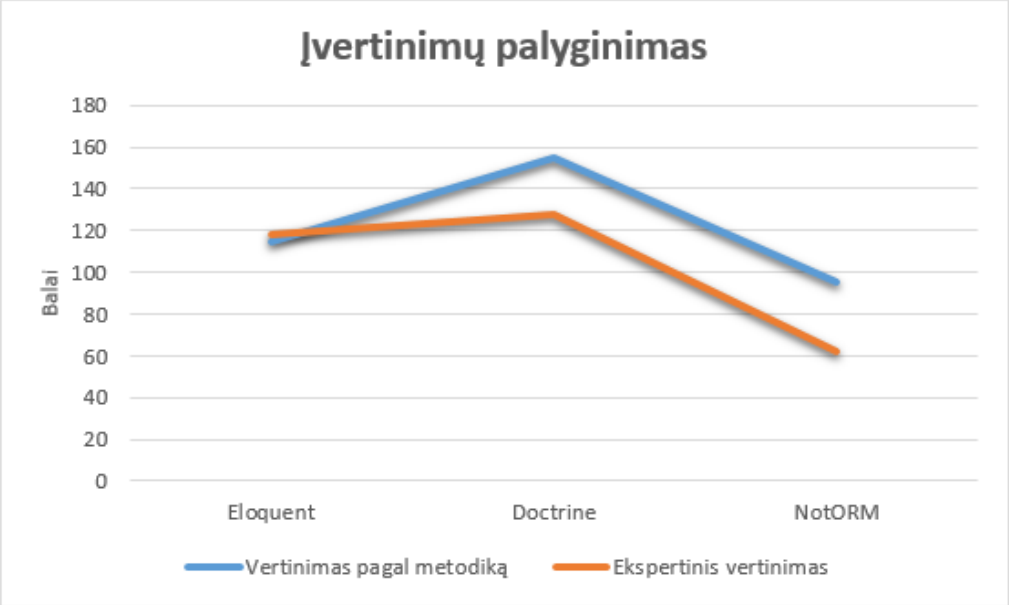
Matome, kad ORM karkasai išsidėstė tokia pat tvarka kaip ir apskaičiavus pagal siūlomą metodoką.

Norint apklausos rezultatus atvaizduoti viename grafike, reikia suvienodinti reikšmių sritis. Vertinant pagal siūlomą metodiką, ORM karkasas gali igyti maksimalią 160 balų reikšmę(*16 * 10*), tuo tarpu vertinant ekspertams, maksimalus įvertinimas gali būti 54(vertino 18 ekspertų, kurie galėjo skirti maksimalų 3 balų įvertinimą, taigi *18 * 3 = 54*). Todėl expertų vertinimą išreikšime proporcingai

siūlomo metodo vertinimui: *54* ekspertų balai atitiks 160 metodikos balų. Perskaičiauojame ekspertinius įvertinimus:

- Eloquent įvertnimas: *40 * 160 / 54 = 118.5;*
- Doctrine įvertinimas: *43 * 160 / 54 = 127.4;*
- NotORM įvertinimas: *21 * 160 / 54 = 62.2;*

Gautas reikšmes galime palygint su siūlomo metodo reikšmėmis, kai lyginama neatsižvelgiant į techninius reikalavimus. Taip yra todėl, kad respondentams nebuvo iškelta jokių konkrečių techninių reikalavimų.



pav. 13 Įvertinimų palyginimas

Matome, kad įvertinimai, nors ir nesutampa idialiai, tačiau kreivė atkartoja viena kitą, todėl galima teigti, kad apklausos dalyvių nuomonė sutampa su rezultatais, gautais naudojant siūlomą lyginimo metodiką.

Siekiant praktiškai išbandyti siūlomą ORM karkasų palyginimo metodiką buvo palyginti trys ORM karkasai. Karkasų pasirinkimą nulėmė realaus igyvendinto projekto specifikacija. Atlikus vertinimą pastarieji ORM karkasai buvo sureitinguoti. Atlikus programuotojų apklausą, paprašyta sureitinguoti tyrime dalyvavusius ORM karkasus. Gauti apklausos rezultatai apytiksliai sutapo su rezultatais, gautais naudojant siūlomą vertinimo metodiką.

DARBO IŠVADOS

1. Susipažinus su ORM karkasų specifika, paaiškėjo, kad ORM karkasų kokybę įtakoja daugybė veiksnių, todėl neįmanoma įvertinti ar palyginti ORM karkasus remiantis viena savybe. Susipažinus su egzistuojančiais vertinimo netodais bei įsigilinus į ORM karkasų realizavimo metodus buvo atrinktos tik tos savybės, kurios labiausiai įtakoja ORM karkaso kokybę.
2. Šiuo metu nėra vieningos sistemos ORM karkasų palyginimui. Esant poreikiui palyginti kelis ORM karkasus vertinimo metodika dažniausiai sugalvojama pačio vertintojo. Dažnu atveju toji metodika būna paviršutiniška, arba įtraukiami mažai reikšmės turinčios ORM karkasų savybės. To pasekoje nukenčia vertinimo tikslumas ir patikimumas. Taip pat gali išaugti laiko sąnaudos norint įvertinti kelis ORM karkasus.
3. Pasiūlytoje ORM karkasų vertinimo metodikoje siūloma ne tik vertinti pastaruosius ORM karkasus tarpusavyje, neatsižvelgiantį jokius kitus faktorius, bet ir įtraukiant techninių reikalavimų specifikaciją. Vertinimas įtraukiant techninius reikalavimus galimas dviem būdais: vertinama griežtai arba vertinama lanksčiai. ORM kokybė ar tinkamumas konkrečiu atveju išriaiškiama skaičiumi. Didesnė skaitinė vertė reiškia gersnį įvertinimą. Tai padaro vertinimo procesą itin lankstų ir leidžia parinkti labiausiai tinkantį ORM karkasą konkrečiam atvejui.
4. Norint praktiškai išbandyti palyginimo metodiką buvo atliktas trijų ORM karkasų palyginimas. Buvo palyginti trys ORM karkasai: Eloquent, Doctrine NotORM. Geriausiai įvertintas liko Doctrine ORM karkasas, kuris vertinant visais trimis būdais išliko pirmoje vietoje. Antroje vietoje liko Eloquent trečioje NotORM. Atlikus programuotojų apklausą, pastebėta, kad ORM karkasų įvertinimo rezultatai apytiksliai sutampa su rezultatais, gautais naudojant siūlomą vertinimo metodiką.

Literatūra

1. Andrew Odlyzko „Internet growth: Myth and reality, use and abuse“ [žiūrėta 2015 metų lapkričio 15 d.] Prieiga internetu: http://www.dtc.umn.edu/~%20odlyzko/doc/internet.growth.myth2.pdf
2. Loup Meurice, Mathieu Goeminne, Tom Mens, Csaba Nagy, Alexandre Decan, and Anthony Cleve „Analyzing the Evolution of Database Usage in Data-Intensive Software Systems“ [žiūrėta 2015 metų lapkričio 16 d.] Prieiga internetu: http://decan.lexpage.net/files/Chapter-ORM-2016.pdf
3. Jaroslav Orság , „OBJECT-RELATIONAL MAPPING“ [žiūrėta 2015 metų gruodžio 6 d.] Prieiga internetu : http://www.dcs.fmph.uniba.sk/diplomovky/obhajene/getfile.php/dp.orsag.orm.pdf?id=86&fid=147&type=application%2Fpdf
4. Oliver Leisalu „Comparative Evaluation of Two PHP Persistence Frameworks“ [žiūrėta 2015 metų gruodžio 4 d.]Prieiga internetu: http://sep.cs.ut.ee/uploads/Main/LeisaluBachelorThesis.pdf
5. Mikael Kopteff „The Usage and Performance of Object Databases compared with ORM tools in a Java environment“ [žiūrėta 2015 metų gruodžio 6 d.] Prieiga internetu: http://www.odbms.org/wp-content/uploads/2013/11/045.01-Kopteff-The-Usage-and-Performance-of-Object-Databases-Compared-with-ORM-Tools-in-a-Java-Environment-March-2008.pdf
6. Gavin King, Christian Bauer „Hibernate Gavin King Christian Bauer Object/Relational Persistence for idiomatic Java“ [žiūrėta 2015 metų gruodžio 15 d.] Prieiga internetu: http://www.immagic.com/eLibrary/ARCHIVES/TECH/JBOSS_US/H040322K.pdf
7. Ant´onio Menezes Leit˜ao „UCL-GLORP—An ORM for Common Lisp“ [žiūrėta 2016 metų vasario 25 d.] prieiga internetu: http://jucs.org/jucs_14_20/ucl_glorp_an_orm/jucs_14_20_3333_3357_leitao.pdf
8. ORM schema prieiga per internetą: [žiūrėta 2016 metų vasario 25 d.] http://flylib.com/books/2/260/1/html/2/images/fig133_01.jpg
9. Peter A. Flach „Inductive characterisation of database relations“ [žiūrėta 2015 metų spalio 7d.] prieiga internetu: https://www.researchgate.net/profile/Peter_Flach3/publication/2704815_Inductive_Characterisation_of_Database_Relations/links/55c2282508aeb975673e3b8a.pdf
10. Cornell Tech „Introduction to Database Systems“ [žiūrėta 2015 metų spalio 23d.] Prieiga internetu: https://courses.cit.cornell.edu/cs5320/lect2.pdf
11. Giorgos Constantinou „Relational Algebra and SQL Query Visualisation“ [žiūrėta 2016 metų kovo 4 d.] prieiga internetu: http://www.doc.ic.ac.uk/~pjm/teaching/student_projects/gc106_report.pdf

12. Mª José Suárez-Cabal, Javier Tuya „Structural Coverage Criteria for Testing SQL Queries“ [žiūrėta 2016 metų balandžio 10d.] prieiga internetu: http://www.jucs.org/jucs_15_3/structural_coverage_criteria_for/jucs_15_03_0584_0619_cabal.pdf
13. Sąryšių diagrama. prieiga internetu: http://i.imgur.com/1vZWr.png
14. Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford „Patterns of Enterprise Application Architecture“ [žiūrėta 2016 metų kovo 19d.] prieiga internetu: http://he-thong-quan-ly-tuong-tac.googlecode.com/svn/trunk/Reference/Patterns_of_Enterprise_Application_Architecture.pdf
15. Sergio Fernandez, Diego Berrueta, Miguel Garc´ıa Rodr´ıguez, Jose E. Labra „Keeping the Semantics of Data Safe and Sound into Object-Oriented Software“ [žiūrėta 2016 metų kovo 29d.] prieiga internetu: http://trioo.wikier.org/resources/doc/paper-trioo-icsoft2010.pdf
16. Robert C. Martin, „Agile Software Development“ [žiūrėta 2016 metų kovo 19d.] prieiga internetu: https://books.google.com.pr/books/about/Agile_Software_Development.html?id=0HYhAQAAIAAJ&hl=lt
17. „Implementing the Active Record pattern“ [žiūrėta 2016 metų kovo 19d.] prieiga internetu: https://www.practicingruby.com/articles/implementing-the-active-record-pattern-1
18. Dragos-Paul Pop „DESIGNING AN OBJECT RELATION MAPPING SYSTEM IN PHP“ [žiūrėta 2016 metų vasario 7d.] prieiga internetu: ftp://ftp.repec.org/opt/ReDIF/RePEc/rau/jisomg/SP11/JISOM-SP11-A24.pdf
19. Paulo Sousa „Learning patterns of application architecture by looking at code“, [žiūrėta 2016 metų vasario 7d.] prieiga internetu: https://www.researchgate.net/publication/228996347_Learning_patterns_of_application_architecture_by_looking_at_code
20. O. Moravcik,D. Petrik, T. Skripcak, and P. Schreiber, „Elements of the Modern Application Software Development“ “, [žiūrėta 2016 metų vasario 10d.] prieiga internetu: https://www.researchgate.net/publication/272912523_Elements_of_the_Modern_Application_Software_Development
21. Michael Carey, Daniela Florescu, Zachary Ives, Ying Lu, Jayavel Shanmugasundaram, Eugene Shekita, Subbu Subramanian „XPERANTO:PublishingObject-RelationalDataasXML“ [žiūrėta 2016 metų vasario 11d.] prieiga internetu: http://xml.coverpages.org/careyXperantoOverview.pdf

22. AngeloCorsaro,DouglasC.Schmidt,RaymondKlefstad,CarlosO'Ryan „ADesignPatternforMemory-ConstrainedEmbeddedApplications“ 2016 metų vasario 13d.] prieiga internetu: https://www.dre.vanderbilt.edu/~schmidt/PDF/virtual-component.pdf
23. Martin Fowler „Table Data Gateway“ [žiūrėta 2016 metų vasario 15d.] prieiga internetu: http://martinfowler.com/eaaCatalog/tableDataGateway.html
24. Fabian Bornhofen, Lauritz Thamsen, Johan UhleFranziska Haeger „InfraRecord: Immediate Feedback on ORM Queries“ [žiūrėta 2016 metų vasario 16d.] prieiga internetu: http://www.freenerd.de/assets/ProgMod_BornhofenThamsenUhle.pdf
25. Philip Brown „What's the difference between Active Record and Data Mapper“ [žiūrėta 2016 metų vasario 16d.] prieiga internetu: http://culttt.com/2014/06/18/whats-difference-active-record-data-mapper/
26. „Nested Loops“ [žiūrėta 2016 metų sausio 3d.] prieiga internetu: http://use-the-index-luke.com/sql/join/nested-loops-join-n1-problem
27. ORM diagrama. [žiūrėta 2016 metų kovo 14d.] prieiga internetu: http://www.agile-code.com/blog/wp-content/uploads/2013/03/ORMMapping.png
28. Venas su vienu sąryšių diagrama [žiūrėta 2016 metų kovo 14d.] prieiga internetu http://i.stack.imgur.com/RkuYs.png