KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Arūnas Bytautas

OBJEKTINIŲ TECHNOLOGIJŲ SU SĄRYŠINĖMIS DUOMENŲ BAZĖMIS SUDERINAMUMO TYRIMAS

Magistro darbas

Vadovas
doc. dr. V. Pilkauskas

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. habil. dr. H. Pranevičius
2005-05-

OBJEKTINIŲ TECHNOLOGIJŲ SU SĄRYŠINĖMIS DUOMENŲ BAZĖMIS SUDERINAMUMO TYRIMAS

Informatikos magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių k. lekt. dr. J. Mikelionienė
2005-05-

Vadovas
doc. dr. V.Pilkauskas
2005-05-

Recenzentas
doc. dr. D. Rubliauskas
2005-05-

Atliko
IFM 9/1 gr. stud.
A. Bytautas
2005-05-

KAUNAS, 2005

KVALIFIKACINĖ KOMISIJA

Pirmininkas: Laimutis Telksnys, akademikas

Sekretorius: Stasys Maciulevičius, docentas

Nariai:

Rimantas Barauskas, profesorius

Raimundas Jasinevičius, profesorius

Jonas Kazimieras Matickas, docentas

Jonas Mockus, akademikas

Rimantas Plėštys, docentas

Henrikas Pranevičius, profesorius

Bytautas A. Objektinių technologijų su sąryšinėmis duomenų bazėmis suderinamumo tyrimas. Magistro darbas. Vadovas: doc. dr. V. Pilkauskas. KTU Informatikos fakultetas Verslo informatikos katedra, Kaunas, 2005, - 59 p.

SANTRAUKA

Šiame magistro darbe pristatomas objektinių technologijų suderinamumo su sąryšinėmis duomenų bazėmis tyrimas kaip aktuali informatikos mokslo ir praktikos problema šiuolaikinėse verslo sistemose. Programuotojams dirbant su objektais, o duomenų saugojimui naudojant sąryšines duomenų bazes, iškyla poreikis apjungti šias dvi teorijas.

Šio magistro darbo tikslas yra apžvelgti objektinių bei sąryšinių technologijų skirtumus, apžvelgiant esmines objektinių bei sąryšinių sistemų savybes, suformuluojant jų panašumus ir skirtumus bei pateikiant realius problemų sprendinius. Taip pat bus pateikti reikalavimai idealiam objektų saugojimo karkasui, kuris naudojamas šioms problemoms spręsti šiuolaikinėse verslo sistemose.

Praktinis šio magistro darbo tikslas yra pasiūlyti objektų saugojimo karkasą realioje kliento – serverio pagrindu veikiančioje sistemoje, atliekant šiuo metu egzistuojančių produktų tyrimą ir išskiriant jų privalumus bei trūkumus. Tam praktinėje darbo dalyje analizuojami šiuo metu rinkoje siūlomi produktai objektų saugojimo sąryšinėse duomenų bazėse problemai spręsti, studijuojamos jų savybės, atliekamas rinkoje egzistuojančių produktų pagal pasirinktus kriterijus palyginimas, atliekamas šių produktų tyrimas kompanijos Exigen draudimo kompanijoms skirtos programinės įrangos kaip tipiškos šiuolaikinės verslo sistemos programinės įrangos, kontekste.

Teiginiams ir rezultatams iliustruoti darbe pateikta 16 paveikslų ir 6 lentelės. Magistro darbo pagrindą sudaro 15 literatūros šaltinių lietuvių ir anglų kalbomis. Magistro darbo gale pateikti 2 priedai.

Bytautas A. Object-Relational mapping research. Master of Science Thesis. Supervisor: doc. dr. V. Pilkauskas. Kaunas University of Technology Faculty of Informatics Department of Business Informatics, Kaunas, 2005, - 59 p.

SUMMARY

Thesis presents object-relational mismatch research as actual problem in the information technology systems targeted for business domain applications. Programmers are working with objects, whereas they use relational databases to make the data persistent. This leads for the need to unite the two theories – object and relational.

The aim of the thesis is to overview object-relational mismatch. Advantages and disadvantages are overviewed and the solutions are given. Requirements for the state-of-the-art object-relational mapping layer will be presented. Those object-relational mapping layers are used in current business processes based system.

Practical aim of the thesis is to propose object-relational mapping framework for real client-server system, based on current the most popular object-relational mapping layers in the market. In the practical part of the thesis feature based analysis of selected object-relational mapping frameworks is being performed. Those object relational mapping layers are being used for tests on real insurance industry based software system.

Theoretical propositions and the results discovered are illustrated with 16 pictures and 6 tables. The thesis is based on 15 different literature sources in the Lithuanian and English languages. Two appendices are given and the end of the work.

TURINYS

PAVEIKSLAI

LENTELĖS

1 ĮVADAS

Šiuo metu plačiausiai duomenų saugojimui naudojamos sąryšinių duomenų bazių sistemos, tačiau programavime labiausiai paplitę yra objektinės sistemos, kadangi būtent objektinės sistemos labiau primena žmogaus mąstymo būdą negu standartinės procedūrinės programavimo kalbos. Dauguma šiandien kuriamų objektinių sistemų naudojasi jau esamais duomenimis sąryšinėse duomenų bazėse. Sunkumai, su kuriais susiduriame, kyla dėl objektinių bei sąryšinių sistemų skirtumo: tarp dviejų „mąstymo“ būdų – objektinio ir sąryšinio. Būtent dėl šio skirtumo ir atsirado tokia sąvoka, kaip „objektinio modelio su sąryšine schema susiejimas“ (angl. *object – relational mapping*). Lengviausias būdas šiai problemai spręsti yra tiesiog naudoti SQL užklausas (angl. *SQL – Structured Query Language*, struktūrizuotų užklausų kalba) rašant objektinę programą, tačiau šis sprendimas yra netinkamas, jei sistema yra bent kiek sudėtingesnė, todėl, kad jis neužtikrina nepriklausomybės tarp duomenų valdymo ir pačios programinės sistemos, todėl, kiek pasikeitus duomenų struktūrai sąryšiniame modelyje, mes esame priversti keisti programinį kodą. Tinkamiausias šios problemos sprendinys yra naudoti „objektų saugojimo sąryšinėse duomenų bazėse karkasą“ (angl. *object persistence framework*), kuris dinamiškai generuoja programos bendravimo su duomenų baze kreipinius ir yra viena iš pagrindinių objektinių daugialypių sistemų dalių. Šiame darbe apžvelgiami sąryšinis ir objektinis modeliai, nurodomos bendros jų savybės bei skirtumai, apžvelgiami šiuolaikinėse sistemose naudojami objektų saugojimo sąryšinėse duomenų bazėse būdai.

Sistemos veikimo metu transformacija tarp objektinės ir sąryšinės schemos atliekama pagal jau nustatytas taisykles. Tos taisyklės apibrėžia kaip objektinis ir sąryšinis modeliai susieti tarpusavyje. Ši transformacija vyksta tarp objektinį modelį palaikančios sistemos į/iš sąryšinį modelį palaikančios sistemos. Kad ši transformacija būtų atlikta nepriekaištingai, reikia gerai suprasti ir žinoti objektinį bei sąryšinį modeliavimą, rasti panašumus ir skirtumus. Idealiu atveju rezultatas – vienas integruotas modelis, kuriame kiekviena dalis atlieka jai priklausantį svarbų vaidmenį. Šiame modelyje turėtų būti užtikrinti ryšiai jungiantys abu skirtingus duomenų pateikimo būdus.

Sunkumai prasideda tuomet, kai mes turime realias sistemas, kuriose duomenims pateikti naudojamos abi schemos – sąryšinė ir objektinė. Realūs atvejai būna daug sudėtingesni ar netgi ne visada telpa į teorinius rėmus. Sąryšinės duomenų bazės gyvuoja jau keletą dešimtmečių; jos sukurtos taip, kad kuo efektyviau būtų įgyvendinti sąryšinės schemos pagrindai. Tuo pačiu objektinis modeliavimas yra labai jaunas, palyginus su sąryšinėmis technologijomis, juolab kad kiekvienoje programavimo kalboje mes galime

išreikšti objektinį modelį vis kitokiais būdais ir priemonėmis. Būtent todėl objektinio modelio su sąryšiniu sąsaja yra gana sudėtinga.

Nežiūrint skirtingo šių technologijų gyvavimo laiko ir joms skirtų uždavinių sričių spręsti, vis dėlto objektinis ir sąryšinis modeliai turi labai daug panašumų. Sąryšinio modelio pagrindas yra duomenų saugojimas, objektinio modelio – veiksmai, kuriuos su duomenimis galima atlikti. Objektinio–sąryšinio modelio sąsajai įgyvendinti reikia nuspręsti, kaip šie du modeliai sąveikaus vienas su kitu ir kokie bus jų ryšiai. Pirmiausia reikėtų išskirti vieno ir kito modelio ypatumus tam, kad būtų galima nuspręsti, kaip būtų galima juos sujungti.

Jeigu objektinei sistemai reikia naudoti sąryšinę duomenų bazę, paprasčiausias būdas yra naudoti SQL kalbos užklausas naudojant jas tiesiogiai programiniame kode. Jei taip padarytume, tuomet sistema tamptų visiškai priklausoma nuo lentelių struktūros sąryšinėje duomenų bazėje. Dar daugiau, programuotojas turi mokėti SQL užklausų kalbą, žinoti apie sąryšines duomenų bazes ir duomenų bazės lentelių struktūrą. Kad daugialypės architektūros sistemos (angl. *multi-tier architecture systems*) būtų kuo paprasčiau palaikomos, duomenų bazės struktūra turi būti paslėpta nuo programuotojo. Programuotojas turėtų matyti ir koncentruotis tik ties savo klasių struktūra ir naudotis objektuose apibrėžta programine sąsaja. Todėl reikalingas objektų saugojimo mechanizmas, kuris užsiimtų objektų saugojimu ir nepriklausytų nuo konkrečios programos ar netgi nuo konkretaus tipo sistemų.

Mokslinis šio darbo tikslas yra pažvelgti į objektinių bei sąryšinių technologijų skirtumus. Tam, kad tai būtų padaryta, bus apžvelgti objektinių bei sąryšinių sistemų pagrindai. Kai turėsime panašumus ir skirtumus, bus pateikti realūs problemos sprendiniai. Taip pat bus pateikti reikalavimai idealiam objektų saugojimo karkasui.

Praktinis šio tyrimo tikslas yra išnagrinėti objektų saugojimo karkasą realioje kliento – serverio pagrindu veikiančioje sistemoje. Bus atliktas šiuo metu egzistuojančių produktų tyrimas ir išskirti jų privalumai bei trūkumai.

2 PROBLEMOS ANALIZĖ

2.1 Objektinių bei sąryšinių technologijų apžvalga

Objektinės ir sąryšinės technologijos skiriasi, kadangi abi šios technologijos buvo sukurtos skirtingoms problemoms spręsti. Sąryšinės technologijos buvo išrastos tam, kad matematiniu pagrindu apibrėžtų duomenų modelį, tuo tarpu objektiškai orientuotos sistemos buvo sugalvotos remiantis tuo, kad objektai su jų savybėmis yra artimesni realiam pasauliui. Toliau apžvelgiamos abiejų modelių charakteristikos ir jų skirtumai.

2.1.1 Sąryšinis modelis

1970 m. E.F. Kodas pateikė sąryšinį duomenų modelį. Šis modelis paremtas prielaida, kad bet kokia informacija gali būti išsaugota viename ar daugiau ryšių.

Sąryšinėse duomenų bazėse ryšiai realizuoti lentelių pavidalu; sąryšinis duomenų modelis nusako tik faktų saugojimo būdą, todėl duomenys ir operacijos yra atskirti vienas nuo kito. Operacijos gali būti realizuotos naudojant papildomas duomenų bazės teikiamas funkcijas, tokias kaip trigeriai (angl. *trigger*) arba sąryšinių duomenų bazių (angl. *stored*) procedūros [2].

Lentelę sudaro stulpeliai ir eilutės. Stulpelių struktūra vadinama lentelės schema. Stulpeliai – tai atributų reikšmės; jie turi nustatytą seką; šią seką nustato lentelės schema kurioje yra stulpelis. Lentelės struktūra (schema) yra apibrėžta (statinė) ir nusako realaus pasaulio esybes. Eilutės, sąryšinėje teorijoje vadinamos įrašais, nusako realaus pasaulio faktus ir yra apibrėžtos stulpelių [2].

Lentelėje visiškai nesvarbus esantis įrašų eiliškumas informacijos požiūriu, tačiau duomenų bazių sistema gali saugoti įrašus tam tikra tvarka, tam, kad įgyvendinti tam tikras funkcijas, pavyzdžiui greitos paieškos ir kita. Sąryšinėje duomenų bazėje raktas yra ryšio poaibis su sekančiomis savybėmis:

- minimali unikaliai identifikuojanti įrašą atributų aibė yra vadinama potencialiuoju raktu. Galimi keli potencialūs raktai; paprastai vienas iš potencialiųjų raktų išrenkamas pirminiu raktu. Raktas, kurį sudaro du ar daugiau atributų, vadinamas sudėtiniu raktu [2].
- Ryšį su kitoje lentelėje esančiu įrašu nusakanti atributų aibė yra vadinama išoriniu raktu. Išoriniai raktai sudaro svarbius ryšius tarp lentelių. Jie naudojami tam, kad surišti vienos lentelės duomenis su kitos lentelės duomenimis. Duomenims sujungti gali būti naudojama SQL kalbos jungimo (angl. *join*) operacija [2].

Pagrindinės bazinės operacijos, kurias galima atlikti su lentelėse esančiais įrašais, yra:

- SELECT – išrinkti duomenis;
- INSERT – įterpti duomenis;
- UPDATE – pakeisti duomenis;
- DELETE – ištrinti įrašą(-us).

Duomenų modelio kūrimo metu ar netgi vėliau, duomenų bazėje gali prireikti normalizuoti lenteles. Duomenų normalizavimu vadinamas procesas, naudojamas duomenų loginiu modeliavimo metu tam, kad užtikrinti tik vieną būdą žinoti tam tikrą faktą, pašalinant visas struktūras, kurios suteikia galimybę sužinoti tą patį faktą iš duomenų bazės lentelės. Normalizavimas naudojamas nereikalingo pertekliškumo kontrolei, bei sumažinti duomenų pridėjimo ir išmetimo metu atsirandančias anomalijas. Duomenų pridėjimo metu anomalija atsiranda tada, jeigu vieno atributo informacijos saugojimas reikalauja papildomos informacijos apie kitą atributą. Trynimo anomalija atsiranda tuomet, kai vieno įrašo trynimo metu netenkame kitam faktui priklausančios informacijos. Yra šešios viena kitą papildančios normalinės formos, aprašytos matematiniu pagrindu. Sakoma, kad lentelė yra normalinėje formoje, jeigu ji tenkina tam tikras sąlygas. Kiekvienai aukštesnio lygio normalinei formai galioja tokios pačios sąlygos kaip ir prieš tai buvusios dar pridedant ir papildomas. Pvz., antrai normalinei formai galioja pirmos normalinės formos sąlygos ir papildomos.

Jei duomenų bazių sistema turės būti palaikoma, kartais gali tekti denormalizuoti lenteles, kadangi kuo aukštesnė normalinė forma, tuo daugiau SQL kalbos jungimo operacijų reikia atlikti tam, kad atkurti faktus, o SQL kalbos jungimo operacijos gali būti palyginti lėtos. Dažniausiai optimaliausia yra trečia normalinė forma. Objektinio – sąryšinio modelių sąsajos požiūriu problemos atsiranda tuomet, jei duomenų bazė yra normalizuojama ar denormalizuojama jau egzistuojančiai sistemai, kadangi esant glaudžiam ryšiui tarp objektinės sistemos ir duomenų bazės reiškia, kad keisdami duomenų bazės schemą mes neišvengsime pakeitimų programoje. Dar didesnė problema iškyla tuomet, kai kliento pusėje yra nuo SQL užklausų priklausomas programinis kodas ir jį reikia keisti – tai padaryti tampa problematiška esant dideliam klientų skaičiui (pvz., 500 – 1000), nes keitimą tektų atlikti kiekvienam klientui.

2.1.1.1 Programinė sąsaja su duomenų bazėmis

SQL (angl. *Structured Query Language* – struktūrinių užklausų kalba) buvo sukurta programinei sąsajai su sąryšinėmis duomenų bazėmis. SQL kalba dirba su aibėmis. Kadangi tokios struktūrinio programavimo kalbos kaip C ar Paskalis nepalaiko operacijų su aibėmis

buvo sugalvotas žymeklis (angl. *cursor*). Žymeklis – tai programinė abstrakcija, suteikianti galimybę kreiptis į bet kurį užklausoje grąžintos aibės elementą, tačiau tik į vieną tuo pačiu metu.

2.1.2 Objektinės sistemos

Objektinių sistemų ištakos siekia 1962 metus, kai Ole-Johan Dahl ir Kristen Nygaard Norvegijos nacionaliniame skaičiavimo centre (angl. *NCC – Norwegian Computing Centre*) Oslo mieste sukūrė modeliavimui skirtą SIMULA. [16]

Objektinis principas yra aprašyti realaus pasaulio objektus, remiantis jų elgsena (metodais) ir esybėmis (duomenys); pvz., liftas turi keliamąją galią, gali būti tuščias, jame gali būti n keleivių ir jis gali judėti aukštyn ir žemyn. Tokiu būdu objektinių sistemų modeliavimas labiau panašus į žmogaus mąstyseną.

Paveldėjimo ir polimorfizmo principai leidžia atlikti pakartotinį kodo panaudojimą, todėl objektinės sistemos yra lengviau išplečiamos ir sumažėja programinio kodo kiekis. Nežiūrint šių privalumų pirmos objektinės programavimo kalbos, kurios įgijo populiarumą ir įsitvirtino iki šių dienų, buvo tik Smalltalk (1980), C++ (1986), Java (1995), C# (2001). Dėl nepriklausomybės nuo operacinės sistemos, Java programavimo kalba šiuo metu yra viena iš populiariausių programavimo kalbų paskirstytosioms sistemoms kurti.

Objektiniai modeliai skiriasi nuo kitų modeliavimo technologijų, kadangi jie kilo iš koncepcijos, jog kintamieji ir abstraktūs duomenų tipai turi būti sujungti į visumą, kurią vadiname objektu. Objektas turi unikalią identifikaciją, būseną ir apibrėžtą elgseną. Objektiniai modeliai sudaromi iš šių objektų. Kad apibrėžti objektus bei jų sąveiką, kitaip tariant, aprašyti modelį, mes turime tokias sąvokas kaip tipas, klasė, paveldėjimas ir asociacija. Nors objektinis modeliavimas tėra tik nedidelis žingsnelis nuo duomenų modeliu pagrįsto programavimo, jis visiškai pakeičia programų struktūrą bei jų rašymo būdą. Objektinio modeliavimo pagrindas yra unikali objekto identifikacija bei jo elgsena, kas yra visiškai nebūdinga sąryšiniam modeliui. Pagrindinės objektų savybės yra paveldėjimas, polimorfizmas ir dinaminis surišimas [4].

Toliau šiame darbe bus apžvelgiamos pagrindinės objektinio programavimo sąvokos. Be šių sąvokų apžvalgos neįmanomas teorinis pagrindas abiejų teorijų apjungimui.

Tipas yra sąsajos, t.y. operacijų bei metodų, kuriuos palaikys objektas, aprašas. Sakoma, kad objektas palaiko tipą, jeigu objektas realizuoja to tipo sąsają. Su visais tokio pat tipo objektais gali būti sąveikaujama per tą pačią sąsają. Objektas gali palaikyti keletą tipų tuo pačiu metu [4: 67].

Tipai gali būti susieti su kitais tipais, t.y. vieno tipo objektai gali būti sujungti ryšiais su kito tipo objektais. Turėdami ryšių grandinę tarp objektų galime pasiekti kitus šiais ryšiais susietus objektus.

Objektas – tai klasės realizacija, kuri yra daugelio to paties tipo objektų specifikacija. Klasė nusako, kokius tipus objektas realizuos, aprašo objektų elgseną ir kokiu būdu objektas turės ir žinos savo būseną. Kiekvienas objektas gali žinoti tik savo paties būseną. Klasė tuo pačiu yra ir tipas.

Objektiniame modeliavime sistemos aprašomos objektais, t.y. sistemos sudarytos iš objektų. Objektas – tai abstraktus programinis vienetas, turintis unikalų identifikatorių, elgseną ir būseną. Objektas – tai abstraktaus duomenų tipo realizacija. Abstraktus duomenų tipas, dar kitaip vadinamas klase, yra duomenų ir kintamųjų bei algoritmų (operacijų, dar kitaip vadinamų metodais), skirtų tiems duomenims apdoroti, visuma. Objektinis modeliavimas taip pat įtraukia daugelį kitų sąvokų, tokių kaip abstrakcija, enkapsuliacija, paveldėjimas, moduliškumas ir kt. [4].

Kadangi objektas gali būti identifikuotas nepriklausomai nuo jo turimos „reikšmės“, jis turi būseną: einamąją reikšmę, susietą su jo unikaliu identifikatoriumi. Objektas viso savo gyvavimo laikotarpiu gali turėti vieną vienintelę būseną (reikšmę), arba gali pereiti per daugelį skirtingų būsenų (turėti daug skirtingų reikšmių). Kadangi objektas turi tokią savybę kaip enkapsuliacija, t.y., jo reikšmė yra „paslėpta“ objekto viduje, mes galime tik tyrinėti jo būseną remdamiesi jo elgsena (naudodamiesi jo turimomis operacijomis). Enkapsuliacija suteikia objektui abstrakcijos lygį, jis tampa „juodoji dėže“. Enkapsuliacija apsaugo, kad trečiosios šalys nematytų objekto vidinės sandaros. Klientai gali sąveikauti tik su išorine objekto struktūra ir tai darydami sužinoti objekto vidinę būseną, tačiau jie nemato, kaip realizuota objekto elgsena ir kokia yra jo sandara [4: 162].

Kiekvienas objektas – tai abstrakti „juodoji dėžė“, kuria galime naudotis tam, kad gautume norimus rezultatus ar atliktume reikalingus veiksmus. Objekto elgsena – tai operacijų kurias jis suteikia (sąsaja, angl. *interface*), šių operacijų gražinamų rezultatų ir šių operacijų įtakotų pasikeitimų objekto būsenai (reikšmei), taip pat ir kitiems sistemos objektams, visuma. Apie objekto būseną galime sužinoti tik atlikdami veiksmus jo sąsaja bei iš tų veiksmų gražinamų reikšmių.

Kai kuriose objektinio programavimo kalbose paveldėjimas gali būti taikomas ir tipams, ir klasėms. Tipų atveju paveldėjimas reiškia, kad jeigu objektas yra „tipo B“ ir jei turime, kad „tipas B“ yra paveldėtas iš „tipo A“, tai mes objektą galime naudoti kaip „tipo A“ objektą. Sakoma kad „tipas B“ atitinka „tipą A“ ir visi „tipo B“ objektai tuo pačiu yra ir „tipo A“ objektai. Paveldėjimo taikymas klasėms reiškia, kad klasė naudoja kitos klasės

realizaciją su galimybe ją pakeisti. Klasių paveldėjimas dažniausiai yra atliekamas siekiant paveldėti tipą, tačiau taip yra ne visada.

Objektas – tai abstraktus programinis vienetas, turintis unikalų identifikatorių, elgseną ir būseną. Unikalų identifikatorių nusako objekto identifikatoriai (programavimo kalbos lygyje tai yra fizinis adresas, kuriuo randasi objektas), objekto elgseną realizuoja jo metodai (operacijos), o būsena priklauso nuo objekto esybių (kintamųjų).

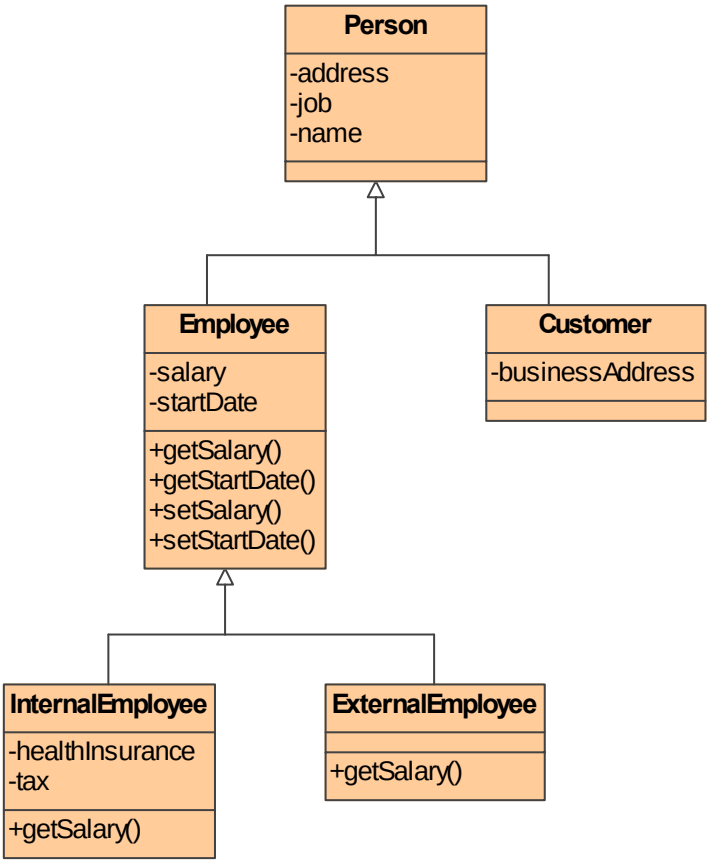
Pagrindinis objektiškumo principas yra surasti esybes ir metodus, kurie yra bendri grupei objektų ir gali būti apjungti vienoje klasėje. Ryšys tarp objekto ir klasės objektiškai orientuotose programavimo kalbose yra toks pat, kaip ir ryšys tarp duomenų tipo ir kintamojo struktūrinėse programavimo kalbose. Programuotojo požiūriu klasė turi vardą ir aprašo kintamuosius (esybes) ir metodus (operacijas).

Veiksmai su esybėmis gali būti atliekami tik per objektui nustatytas operacijas. Tai reiškia, kad objekto būsena priklauso nuo jo esamų esybių reikšmių ir gali būti pakeista kviečiant metodus (atliekant operacijas). Metodų realizacija yra paslėpta klasės viduje; tai vadinama enkapsuliacija. Klasės realizacija ir jos naudojimas yra du skirtingi etapai.

Klasės specifikacija aprašo metodus ir esybes, kurias ji turi. Jei klasė B (dar kitaip vadinama klase–vaiku), yra paveldėta iš klasės A, tai ji turi visas klasės A esybes ir metodus (dar kitaip vadinamos super klase arba pirmine klase). Klasė–vaikas gali savo viduje perrašyti esybes bei metodus, tokiu būdu suteikdama jiems kitokią reikšmę. Taip galimas daugialypis paveldėjimas, t.y. klasė yra paveldėta iš kelių klasių. Paveldėjimas vadinamas „klasė yra X tipo“ ryšiu, kadangi klasė–vaikas turi visa tai, ką turi ir pirminė klasė.

1 paveiksle pateiktas klasių hierarchijos su paveldėjimu pavyzdys. Klasės Employee ir Customer yra paveldėtos iš klasės Person. Klasės InternalEmployee ir ExternalEmployee yra paveldėtos iš klasės Employee ir jose perrašomas metodas getSalary(), nors jie abu gauna atlyginimą, tačiau kiekvienam iš šių objektų jis skaičiuojamas skirtingais būdais, todėl kiekviename objekte šio metodo realizacija bus skirtinga.

Objekto tipo kintamajam gali būti priskirtas žemesnio tipo objektas iš tos pačios objektų klasių hierarchijos. Jei objekto tipo kintamasis yra tipo „klasė A“, jam taip pat gali būti priskirtas „klasės B“ tipo kintamasis, jeigu B klasė paveldėta iš A klasės, nes B yra A (paveldėjimas – yra tipo X ryšys). Objektų kintamieji gali būti realizuoti kaip adresai, kadangi B tipo objekte gali būti daugiau metodų ir esybių negu objekte A, tuo pačiu jis gali pakeisti objekto A metodų ir esybių realizaciją ir B tipo objektui reikės daugiau atminties negu objektui A, o kintamajam šiuo metu priskirto objekto tipą galima nustatyti tik sistemos veikimo metu. Galimybė A tipo kintamajam priskirti B tipo kintamąjį, jei klasė B yra paveldėta iš klasės A, vadinama polimorfizmu.

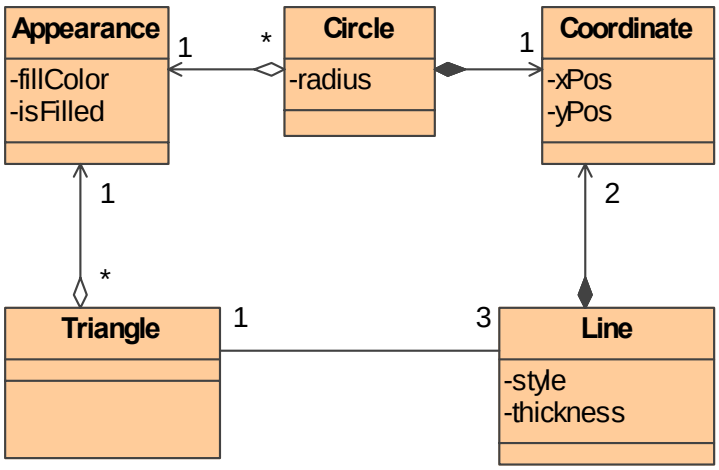


1 pav. Klasių hierarchijos su paveldėjimu pavyzdys (UML notacija)

Paveldėti metodai gali būti perrašyti išlaikant tą patį parametrų kiekį kaip ir pirminėje klasėje, tačiau jų elgsena gali būti visiškai skirtinga. Tokiu būdu sistema tik veikimo metu gali nustatyti kintamajam priskirto objekto tipą ir kurį metodą reikia kviesti; tai vadinama dinaminiu surišimu. Programuotojui tuo rūpintis nereikia, nes sistema tai atlieka automatiškai.

Sąryšiniai ryšiai krypties neturi, tuo tarpu objektiškai orientuotame modelyje ryšiai gali būti dvikrypčiai arba vienakrypčiai.

Objektas gali iškviesti kito objekto metodą išsiusdamas šiam pranešimą. Jei objektai vienas kitam siunčia pranešimus, sakoma, kad tarp jų egzistuoja ryšys. Objektinėje terminologijoje ryšys vadinamas asociacija [8].



2 pav. Skirtingų tipų ryšių tarp klasių pavyzdys

Kaip ir sąryšiniame modelyje, ryšiai turi kardinalumą. Ryšiai gali būti viengubi (rodantys į save) arba dvigubi. Ryšiai tarp klasių dar gali turėti kryptiškumą ir tipą.

Asociacija gali būti dvikryptė arba vienkryptė. Vienkryptė asociacija yra tuomet, kai tarp objekto A ir objekto B egzistuoja ryšys ir arba tik objektas A gali pasiekti objektą B, arba tik objektas B gali pasiekti objektą A, bet ne abu vienas kitą. 2 paveiksle Line objektas gali pasiekti Triangle objektą, tuo pačiu Triangle objektas gali pasiekti objektą Line. Dvikryptė asociacija yra tuomet, kai ir objektas A gali pasiekti objektą B ir objektas B gali pasiekti objektą A. UML diagramose vienkrypčiai ryšiai vaizduojami kaip linija su atvira rodykle viename gale. Linija be rodyklių reiškia arba dvipusį ryšį arba reiškia, kad ryšio kryptis yra nežinoma. 2 paveiksle tik per klasių Circle arba Line objektus galima pasiekti objektą Koordinate, tačiau šis savo ruožtu visiškai nieko nežino apie objektus Circle ir Line.

Asociacijos ryšys tarp klasių skirstomas į du tipus – agregaciją ir kompoziciją:

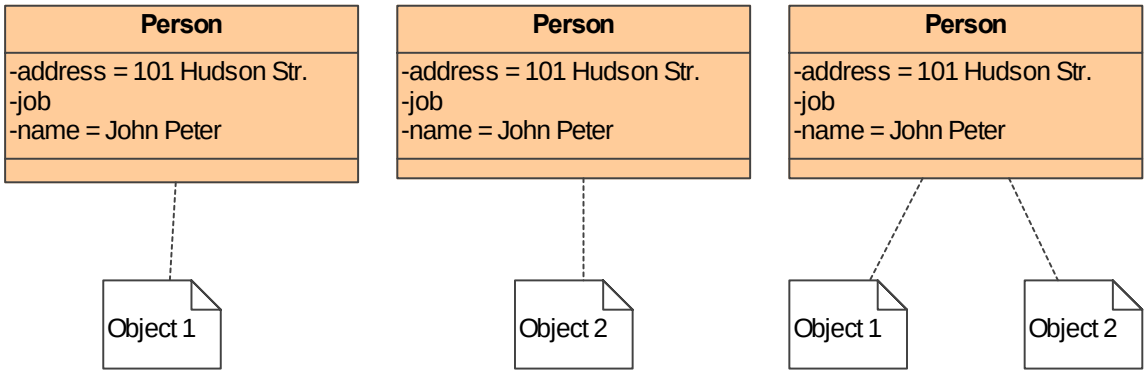
- Asociacija reiškia laisvą ryšį tarp klasių ir reiškia, kad tam tikros klasės objektai gali siųsti pranešimus kito klasės tipo objektams;
- Agregacija yra „yra objekto dalis“ ryšys (angl. *„a part–of–relationship“*) ir ją sudaro objektas, kuris turi kitą objektą, bei turimasis objektas. Nėra aiškaus apibrėžimo, kuris nustatytų skirtumą tarp asociacijos ir agregacijos. [8: 80]. Jeigu triname Triangle ar Circle klasių objektus pagal 2 pav. hierarchiją, tuomet turime tuo pačiu ištrinti ir su jais susietą Appearance objektą, kadangi šio objekto egzistavimas tuomet netenka prasmės;
- Kompozicija – tai „stipresnis“ agregacijos ryšio tipas, kuris nusako, jog abu ryšį sudarantys objektai yra kaip vienas, t.y. jų gyvavimo ciklas yra susietas tarpusavyje: ištrinant vieną, bus ištrintas ir kitas. Pagal 2 pav. Coordinate tipo objektas priklauso

arba Line arba Circle klasės tipo objektui, tačiau tas pats vienas Coordinate tipo objektas negali priklausyti dviems skirtingiems Line ar Circle klasės tipo objektams. Pabandykime įsivaizduoti tokį atvejį – jeigu pakeičiame Circle objekto centrinę koordinatę, o ryšio tipas tarp Coordinate objekto ne kompozicija, tuomet tuo pačiu pasikeistų visų Line objektų susijusių su Coordinate objektu koordinatės, o tai nėra pageidautinas efektas.

Kiekvienas objektinės sistemos objektas turi turėti unikalų identifikatorių – objekto ID. Šis objekto identifikatorius turi būti nepriklausomas nuo paties objekto esybių. Kitaip sakant, jeigu lyginame du objektus, tai turime sulyginti juos dviem sekančiais aspektais:

- Objekto esybių pagrindu – du objekto tipo kintamieji rodo į skirtingus fizinius to paties tipo objektus kurie turi tas pačias esybes;
- Objekto identifikatoriaus unikalumo pagrindu – tam, kad sulyginti ar mūsų turimi du objekto tipo kintamieji rodo į vieną ir tą patį fizinį objektą.

Šis identifikatorius yra pagrindinė savybė, skirianti jį nuo abstraktaus duomenų tipo (klasės). Iš karto po objekto sukūrimo tai yra vienintelis būdas atskirti jį nuo kitų objektų, kadangi objektų būsenos gali sutapti. Objektinėse programavimo kalbose unikalus identifikatorius gali būti skirtingai realizuotas, tačiau jo esmė yra ta pati. Paprastai objektinėje programavimo kalboje unikalus objekto identifikatorius yra jo adresas atmintyje.



3 pav. Objektų lygumo pavyzdys

Užtikrinti objekto identifikatorių unikalumą gali būti gana sudėtinga, jeigu sistemoje yra objektų, esančių skirtingose adresų erdvėse, pvz., paskirstytose sistemose. Šiuo atveju reikalingas globalus modulis, kuris užtikrintų teisingą objekto identifikatorių paskirstymą. Objektinio – sąryšinio keitimo atveju tokia situacija atsiranda tuomet, kai du klientai yra skirtinguose kompiuteriuose, tačiau kreipiasi į vieną serverį, kuris atlieka bendravimą su sąryšine duomenų baze ir per jį atliekamas objektų saugojimas.

2.1.3 Panašumai ir skirtumai

Kaip jau buvo minėta anksčiau, abi technologijos – sąryšinė ir objektinė – buvo sukurtos skirtingiems uždaviniams spręsti. Dabar bus apžvelgti panašumai ir skirtumai tam, kad suprasti, kokios problemos atsiranda abi technologijas naudojant kartu.

Abiems teorijoms yra bendri šie panašumai:

- Struktūra – lentelės nustato įrašų struktūrą taip, kaip klasės aprašo objektų struktūrą;
- Atominiai duomenys – įrašai saugo atominius duomenis taip, kaip objektai turi esybes;
- Ryšiai – sąryšiniame modelyje ryšiai egzistuoja tarp įrašų; objektiniame modelyje ryšiai egzistuoja tarp objektų.

Šie panašumai ir yra pagrindinės savybės, leidžiančios šias teorijas apjungti.

Du nagrinėjamieji modeliai turi panašumų, tačiau jie turi ir esminių skirtumų:

- Elgsena ir duomenys – sąryšiniame modelyje elgsenos nėra; sąryšinis modelis dirba tik su duomenimis, bet ne su elgsena;
- Paveldėjimas – EER modelyje yra apibrėžtos generalizacijos ir specializacijos sąvokos, kurios leidžia sąryšiniame modelyje nusakyti „yra tipo“ ryšį, kuris reiškia duomenų paveldėjimą. Objektinėse sistemose paveldėti galima ne tik duomenis, bet ir elgseną (metodus), kurią galima perrašyti.
- Polimorfizmas – objektinėje sistemoje objektas gali keisti savo vaidmenį sistemos veikimo metu, sąryšiniame modelyje tokios sąvokos visai nėra.
- Vienakrypčiai ir dvikrypčiai ryšiai. Sąryšiniame modelyje įmanomi ryšiai 1:1; 1:n jei naudojame išorinius raktus; ir m:n jei naudojame išorinius raktus ir papildomas lenteles ryšiams. Sąryšinė algebra leidžia atlikti jungimo (angl. *join*) operaciją į abi puses, todėl sąryšiniame modelyje ryšiai neturi krypties. Objektiniame modelyje mes galime turėti vienakrypčius ar dvikrypčius ryšius.
- Unikalumas – sąryšinėse technologijose unikalumą lentelėje galima užtikrinti sąryšinės duomenų bazės sistemos lygyje naudojant pirminius raktus. Jeigu mes nuskaitome duomenis iš lentelės ir iš tų duomenų sukonstruojame objektą, tuo metu, kai informacija palieka duomenų bazę, šį unikalumą mes prarandame, kadangi informacija jau nėra duomenų bazės duomenų integralumo užtikrinimo mechanizmo ribose ir tai gali sukelti nesuderinamumo problemą, jeigu operacijas su tuo pačiu objektu atlieka keletas klientų, esančių skirtingose adresų erdvėse (paskirstymo problema). Objekto unikalumą užtikrina ne pirminis raktas, bet unikalus objekto identifikatorius.

Sąryšinis modelis skirtas darbui su faktais, t.y. duomenimis. Objektinis modelis skirtas darbui su objektais kurie prie savo duomenų dar turi tapatumą, būseną ir elgseną.

Sąryšinis modelis leidžia generalizaciją ir specializaciją, kurias galima vadinti paveldėjimu duomenų lygyje. Priešingai nei objektinis, sąryšinis modelis polimorfizmo nepalaiko.

Ryšiai sąryšiniame modelyje išreiškiami per pirminius ir išorinius raktus; susieti įrašai gali būti saugomi toje pačioje arba atskirose lentelėse; juos galima apjungti naudojant SQL kalbos jungimo (angl. *join*) operaciją. Objektinis modelis naudoja ryšius, kurie gali būti naudojami tiesiogiai, t.y. objektai tiesiai sujungti vienas su kitu. Sąryšinis modelis nesuteikia galimybės turėti tam tikros krypties ryšius, tuo tarpu kai objektiniame modelyje galime turėti vienakrypčius ir dvikrypčius ryšius.

Sąryšinio modelio aprašymo tikslas yra normalizuoti duomenis, išvengiant anomalijų bei pertekliškumo. Objektinio modelio aprašymo tikslas yra sumodeliuoti vadinamąjį verslo procesą (angl. *business process*), sukuriant objektus, kurie atspindėtų realaus pasaulio esybes su elgsena bei duomenimis. Verslo procesas – tai apibrėžtas realaus pasaulio procesas, pvz., paskolos išdavimas, kliento registracija ir pan.

Įrašų unikalumą sąryšinėje duomenų bazėje užtikrina pirminiai raktai. Objektai turi objekto identifikatorių, kuris gali būti unikalus klasės lygyje, klasių hierarchijoje ar netgi visų klasių atžvilgiu.

2.1.4 Objektinio – sąryšinio modelių sąsajos elementai

Sekančiame skyriuje bus nagrinėjama, kaip klases galima susieti su lentele, objekto kintamuosius – su įrašais toje lentelėje; ryšiai tarp klasių gali būti išsaugoti pirminių/išorinių raktų porose. Tačiau kas atsitinka metodams, paveldėtoms klasėms ir specializuotiems ryšiams (asociacijai ir kompozicijai)? Tarp šių dviejų technologijų egzistuoja tiek teoriniai, tiek praktiniai skirtumai. Realus problemos sprendinys turi atsižvelgti ir į tokias problemas kaip spartinančioji atmintis ir tarpiniai objektai (naudojami duomenų užkrovimui pagal poreikį).

3 OBJEKTŲ SAUGOJIMO SĄRYŠINĖSE DUOMENŲ BAZĖSE TEORINIAI SPRENDINIAI

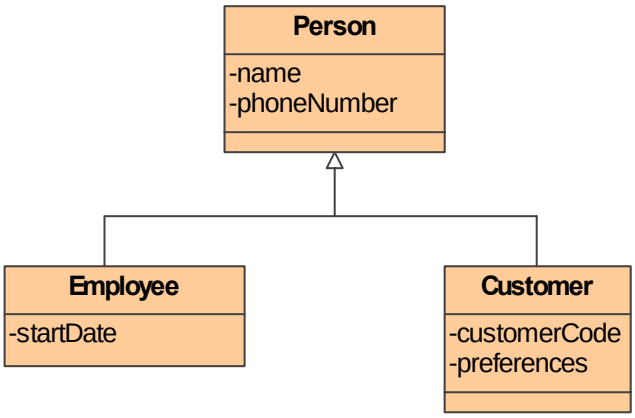
Šiame skyriuje bus nagrinėjamos problemos, atsirandančios dėl skirtumo tarp objektinės ir sąryšinės technologijų, ir bus pateikti šių problemų sprendiniai.

3.1 Klasės kintamųjų sąsaja

Lentelės aprašo įrašų struktūrą panašiai, kaip klasės aprašo objektų struktūrą. Tokiu būdu klasę galima tiesiogiai susieti su lentele. Klasės vardas susiejamas su lentelės pavadinimu, o klasės kintamųjų vardai – su lentelės stulpeliais. Nesuderinamumas atsiranda tada, jeigu klasės kintamasis yra tokio tipo, kuris neturi ekvivalento sąryšinėje duomenų bazės sistemoje.

Sąryšinėje duomenų bazėje įrašų unikalumui užtikrinti naudojami pirminiai raktai, o objektų unikalumui užtikrinti – unikalūs objektų identifikatoriai. Jeigu klasė susiejama su lentelės schema, tuomet unikalus objekto identifikatorius tampa pirminiu raktu.

4 paveiksle pavaizduota klasių hierarchija su klasių vidiniais kintamaisiais.



4 pav. Klasių diagrama su vidiniais kintamaisiais

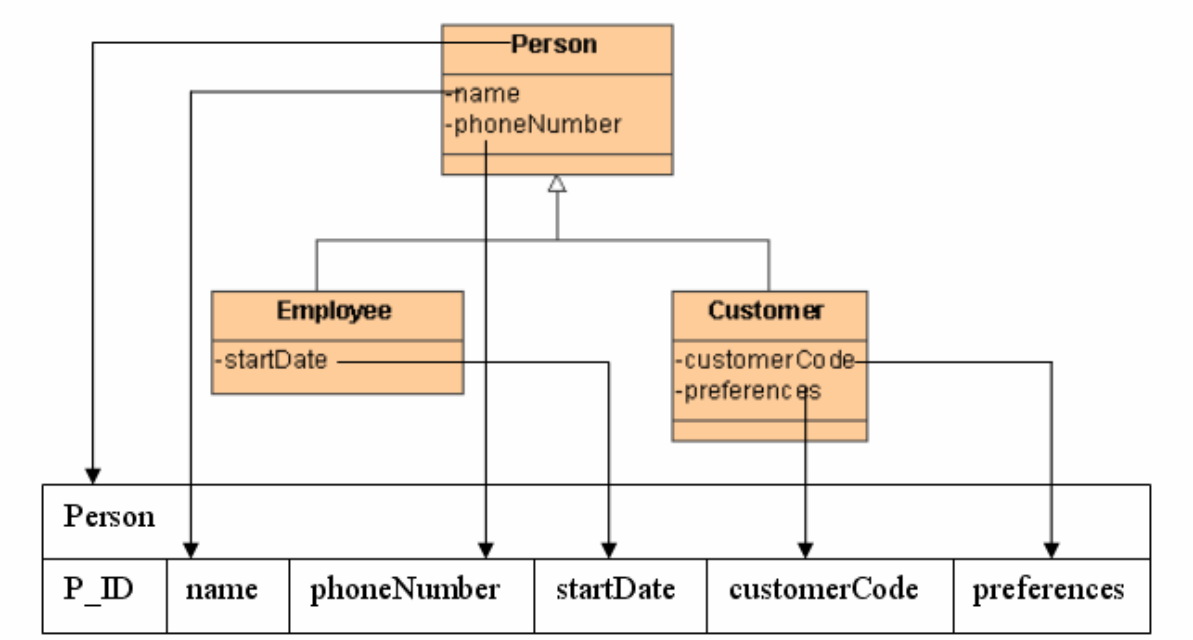
3.2 Klasių hierarchijos sąsaja su sąryšine schema

Klasių hierarchijos susiejimas su sąryšinės duomenų bazės schema yra kiek sudėtingesnis. 4 paveiksle pavaizduota klasių hierarchija. Galimi trys skirtingi klasių hierarchijos susiejimo būdai:

- Viena lentelė visai klasių hierarchijai;
- Viena lentelė galutinei klasei (neabstrakčiai);
- Viena lentelė kiekvienai klasei.

3.2.1 Viena lentelė visai klasių hierarchijai

Šis susiejimas dar vadinamas filtruota sąsaja. Tai lengviausias būdas šiai problemai spręsti, susiejant visą klasių hierarchiją į vieną lentelę. Šioje lentelėje saugomi kintamieji iš visų hierarchijos klasių. 4 paveiksle pateikto pavyzdžio tokiu būdu susietas rezultatas pateiktas 5 paveiksle.



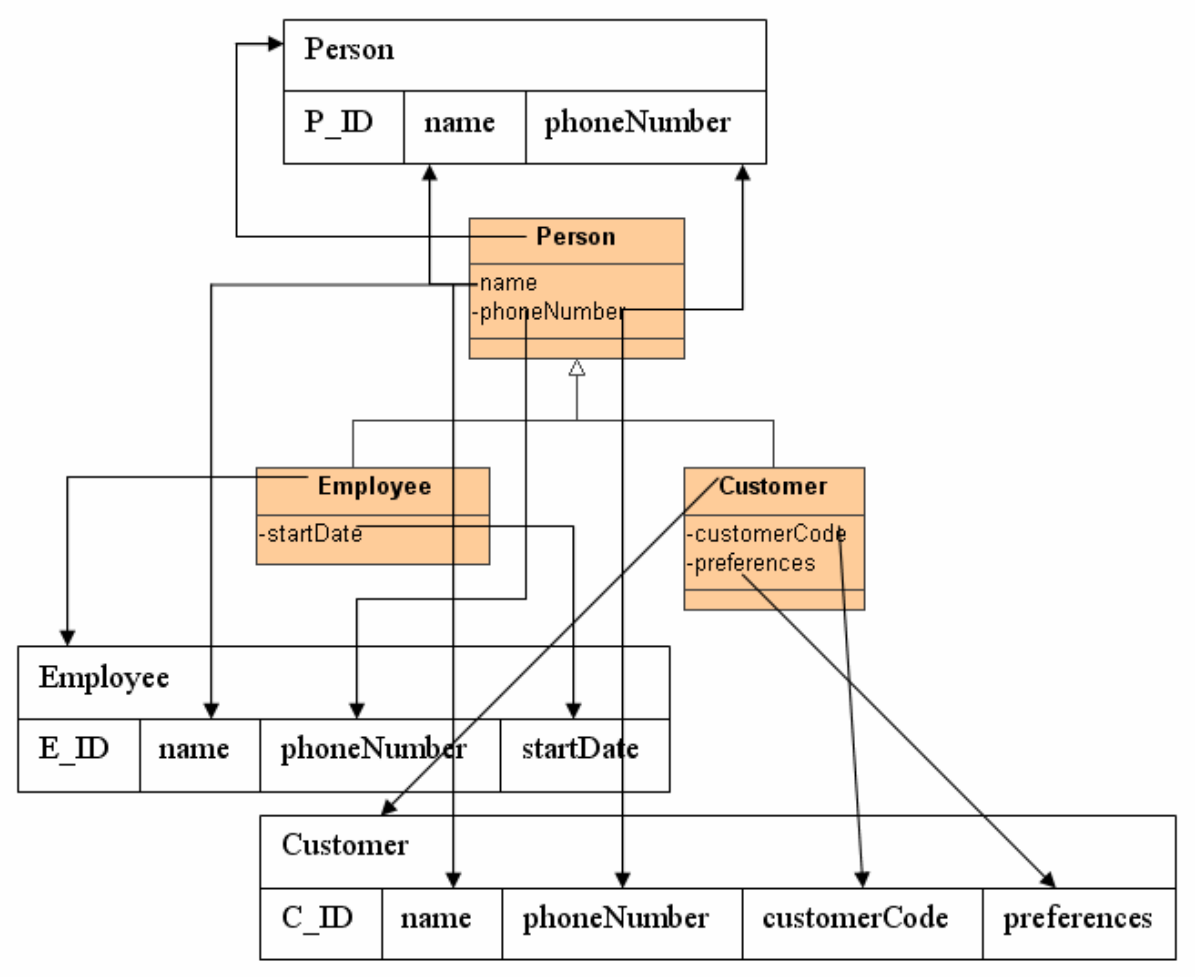
5 pav. „Viena lentelė visai klasių hierarchijai“ sąsajos pavyzdys

Šio susiejimo būdo privalumas yra toks, kad užklausos gali būti suformuotos ir atliktos labai greitai ir efektyviai, nes nereikia atlikti SQL kalbos jungimo (angl. *join*) operacijų. Kitas privalumas – efektyviai palaikomas polimorfizmas, kadangi iš vieno įrašo galima suformuoti bet kokį objektą iš tos pačios klasių hierarchijos.

Šio sprendimo trūkumas yra tai, kad eilutės gali turėti daug tuščių reikšmių, o šios savo ruožtu gali lemti tai, jog pasikeitus bet kurios klasės kintamiesiems gali tekti keisti visus lentelės įrašus. Kitas trūkumas – objekto identifikatorius turi būti unikalus visoje klasių hierarchijoje.

3.2.2 Viena lentelė galutinei (neabstrakčiai) klasei

Tai vadinamoji horizontali sąsaja. Šiuo atveju sukuriama po vieną lentelę kiekvienai galutinei (neabstrakčiai) klasei. Visi klasės kintamieji, įskaitant ir paveldėtus, susiejami su vienos lentelės atributais. 6 paveiksle pavaizduota, kaip tai daroma.



6 pav. „Viena lentelė neabstrakčiai klasei“ sąsajos pavyzdys

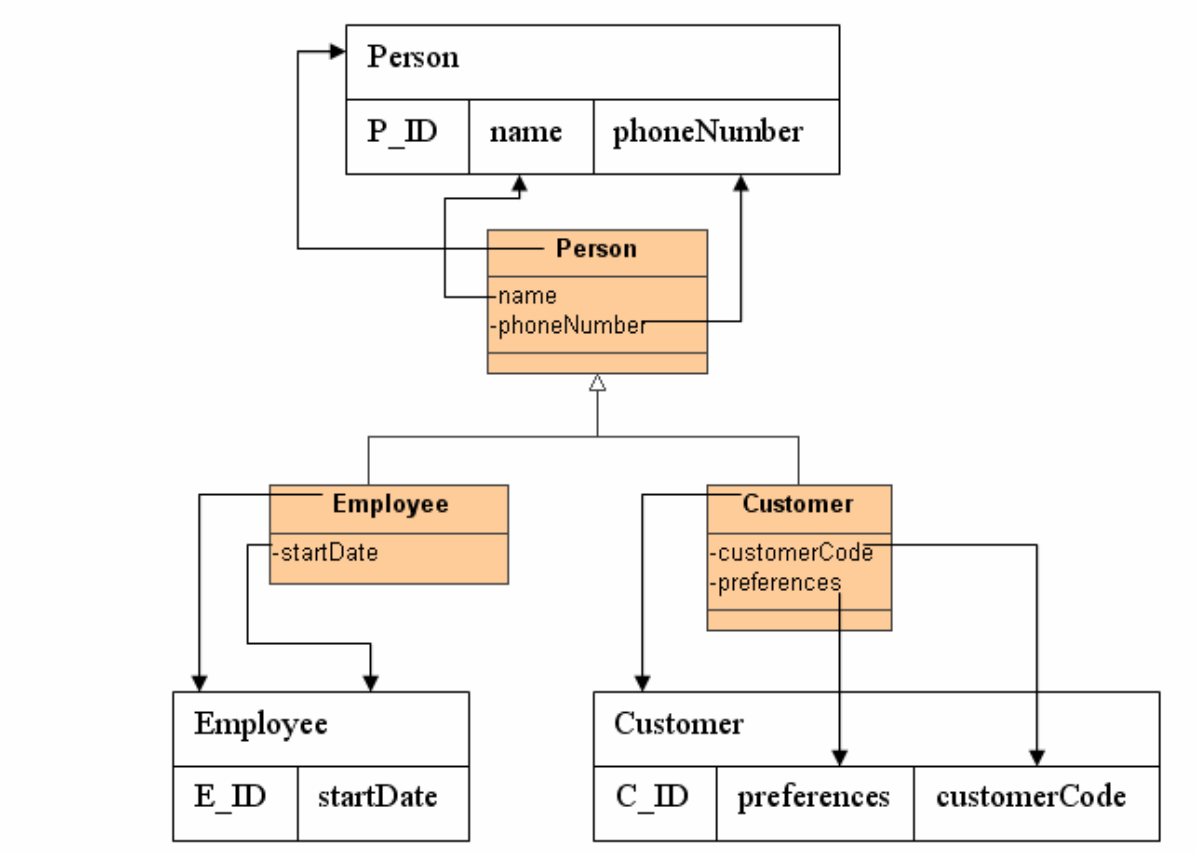
Šio metodo privalumas yra tas, kad kiekvieno tipo objektas turi savo lentelę ir užklausas galima realizuoti gana efektyviai, nes ir čia nereikia SQL kalbos jungimo (angl. *join*) operacijų.

Trūkumas yra tai, kad polimorfizmo negalime išreikšti taip lengvai, kaip kitais dviem atvejais – jei objektas tampa kitokio tipo objektu, telieka ištrinti įrašą iš jo lentelės ir įterpti į kitą. Jei objektas yra kelių tipų, tuomet turės būti keli tam pačiam objektui priklausantys įrašai skirtingose lentelėse. Tampa gana sudėtinga užtikrinti duomenų integralumą. Dar daugiau, jeigu pasikeičia klasės kintamieji, turime pakeisti ne tik šiai klasei priklausančią lentelės schemą, tačiau ir visas lenteles, kurios buvo sukurtos iš tos klasės paveldėtiems objektams saugoti.

3.2.3 Viena lentelė vienai klasei

Tai vadinamasis vertikalus susiejimas. Šiuo atveju kiekvienai klasei sukuriama po lentelę visoje klasių hierarchijoje. Taigi, klasė B automatiškai paveldi klasės A kintamuosius

kurie jau susieti su klasės A lentele. Tai reiškia, kad kiekviena klasė, kuri bus paveldėta iš klasės A, bus susieta su daugiau negu viena lentele (7 paveikslas).



7 pav. „Viena lentelė vienai klasei“ sąsajos pavyzdys

Praktikoje papildomą informaciją apie tipą patogu saugoti pradinės klasės lentelėje.

Ši informacija susiejimo mechanizmui nurodo, kurias lenteles susieti atliekant SQL kalbos jungimo operaciją. 7 paveiksle parodyta, kad lentelė Person naudoja atributą p_type su galimomis reikšmėmis Employee ir Customer.

Toks susiejimo būdas leidžia labai gerai realizuoti polimorfizmo palaikymą, kadangi objektui palaikant kelis tipus, jis bus paskirstytas per visas atitinkamas lenteles. Jeigu klasės kintamųjų reikšmės pasikeis, tai įtakos tik vieną lentelę, kuri bus susieta su atitinkamu atributu.

Šio susiejimo būdo trūkumas yra tai, kad atliekant duomenų, priklausančių paveldėtiems objektams išrinkimą, turi būti atliekama SQL kalbos jungimo operacija. Be to, jei paveldėjimo schema gana sudėtinga, reikia palaikyti daug lentelių.

3.2.4 Paveldėjimo sąsajos būdų palyginimas

Kiekvienas iš galimų susiejimo variantų turi savo privalumų ir trūkumų; kiekvienu atveju reikia nuspręsti, kokią strategiją pasirinkti. Kuo lengvesnis polimorfizmo palaikymas, tuo daugiau reikės atlikti SQL kalbos jungimo operacijų. Pirmoje lentelėje pateikti apibendrinti rezultatai.

1 lent. Paveldėjimo sąsajos būdų palyginimas

Kriterijus	**Filtruota sąsaja**	**Horizontali sąsaja**	**Vertikali sąsaja**
SQL kalbos jungimo operacijų kiekis	mažai	mažai	daug
Keitimų skaičius klasės kintamųjų pasikeitimo atveju	daug	labai daug	mažai
Keitimų skaičius objekto vaidmens pasikeitimo atveju (polimorfizmas)	mažai	daug	daug
Keitimų skaičius jei objektas kelių tipų (paveldėjimas)	mažai	labai daug	daug
Lentelių skaičius	mažai	vidutiniškai	labai daug
Tuščios reikšmės lentelėse	taip	ne	ne

Kaip matome, strategijos pasirinkimas yra labai svarbus žingsnis; netinkamos strategijos pasirinkimas gali lemti labai lėtą sistemos darbą.

3.3 Klasių ryšių susiejimas

Sąryšinėse duomenų bazėse įrašų susiejimui naudojami išoriniai raktai, objektinėse sistemose ryšiai tarp klasių yra tiesioginiai.

Vienakrypčiai ir dvikrypčiai klasių ryšiai skirtingai susiejami su sąryšinių duomenų bazių schema. Tarkime, kad turime dvi dvikrypčiu ryšiu susijusias klases – A ir B. Jeigu iš sąryšinių duomenų konstruojame A tipo objektą, taip pat turime atstatyti ir klasę B, susijusią dvikrypčiu ryšiu su klase A. Jeigu turime vienkryptį ryšį tarp A ir B, tai konstruojant objektą A turime sukurti objektą B ir atstatyti ryšį tarp A ir B. Tačiau, jeigu mums reikalingas tik objektas B, nereikia atstatyti nei ryšio, nei paties objekto A, tuo pačiu išvengiant SQL kalbos jungimo operacijos.

Yra trys pagrindiniai ryšių susiejimo būdai: su ryšių lentelėmis, su išoriniais raktais ir su vidinėmis klasėmis.

Ryšių lentelių metodas. Šiuo būdu galima išsaugoti 1:1, 1:n, m:n ryšius. Ryšių lentelėje saugomi du išoriniai raktai. Šie raktai yra pirminiai raktai iš abiejų susijusių tarpusavyje lentelių įrašų. Šis būdas tinka išsaugoti bet kokio tipo ryšius, tačiau tai nėra labai optimalus sprendimas greičio atžvilgiu, kadangi reikia atlikti net dvi join operacijas.

Išorinių raktų metodas. Šiuo metodu galima susieti 1:1, 1:n ryšius. Vienoje iš lentelių saugomas išorinis raktas į kitos lentelės susijusį įrašą. Objekto konstravimo metu atliekama SQL kalbos jungimo operacija. Šis metodas yra greitesnis negu prieš tai buvęs, kadangi jis reikalauja tik vienos SQL kalbos jungimo operacijos.

Klasės į lentelę metodas. Šiuo atveju dvi klasės susiejamos su viena ir ta pačia lentele. Šis metodas tinkamas ryšiams vienas–su–vienu bei vienas–su–daug, tačiau 1:n atveju turime tiksliai žinoti n. Šis būdas yra labai greitas, nereikia SQL kalbos jungimo operacijų, tačiau vienas–su–daug ryšių atstatymui reikia kiek didesnių pastangų, o daug–su–daug ryšiai iš viso nepalaikomi. Dar didesnis trūkumas yra tai, kad įrašo vieta eikvojama tuščioms reikšmėms. Be to, tokiu atveju lentelė jau nebėra pirmoje normalinėje formoje.

Antroje lentelėje pateikta ryšių sąsajos būdų suvestinė.

2 lent. Ryšių sąsajos būdų palyginimas

Ryšio tipas	**Ryšių sąsajos būdai**		
	Ryšių lentelė	**Įšorinis raktas**	**Klasės į lentelę**
1:1	neefektyvus	Efektyvus	labai efektyvus
1:N	neefektyvus	labai efektyvus	su apribojimais
M:N	efektyvus	Neįmanomas	Neįmanomas

N–mačiai ryšiai paprastai nėra naudojami klasių diagramose, o sąryšiniame modelyje jie realizuoti į ryšių lentelę įrašant visų susijusių lentelių įrašų pirminius raktus. N-matis ryšys visada gali būti išrištas į n dvigubų ryšių ir į papildomą esybę. Paprastai, jei klasių modelyje atsiranda n-matis ryšys, tai reiškia, kad užmiršome kokią nors klasę ir modelyje yra klaida.

Objektiniame pasaulyje yra trys ryšių tipai: asociacija, agregacija ir kompozicija. Asociacija reiškia ryšį tarp objektų, kurie gyvuoja nepriklausomai vienas nuo kito. Prarastas asociacijos ryšys nesukelia integralumo problemų. Išorinio rakto atributui galime priskirti tuščią reikšmę. Agregacija yra tuomet, kai vienas objektas turi kitą objektą. Jei prarandame šį ryšį, tuomet turimo objekto egzistencija netenka prasmės; todėl turime pasirūpinti, kad šalinant objektą–savininką turimasis objektas taip pat būtų ištrintas, t.y. ištrinti visus

objektui–savininkui priklausančius įrašus. Kompozicija yra ryšio tipas, kai turimąjį objektą turi to paties tipo objektas.

Šiuo atveju objekto įrašo visi išoriniai raktai yra įrašų, esančių toje pačioje lentelėje pirminiai raktai.

3.4 Klasės metodų sąsaja

Nepaisant to, kad sąryšinėse duomenų bazėse yra įvestos tokios galimybės kaip trigeriai ir sąryšinių duomenų bazių procedūros (angl. *stored–procedures*), sąryšinis modelis yra skirtas duomenims saugoti. Mūsų atveju saugomi duomenys priklauso objektinėms sistemoms. Sistemos klasių elgsenos nereikia saugoti duomenų bazėje, kadangi ji jau yra pačioje programoje.

Jeigu duomenų bazę naudojanti paskirstyta sistema yra parašyta tik viena programavimo kalba, šia kalba aprašomi ir klasių struktūra bei metodai. Jeigu naudojamos kelios programavimo kalbos su tuo pačiu objektiniu modeliu, kurio objektai saugomi duomenų bazėje, tuomet turi būti naudojamas nuo programavimo kalbos nepriklausomas objektų aprašymas, pvz., XML schema ar CORBA IDL.

Bendravimui su duomenų baze metodai ir kintamieji gali būti realizuoti viena programavimo kalba; kitos programavimo kalbos gali jais naudotis per tam tikrą sąsają, pvz., CORBA arba žiniatinklio paslaugas (angl. *WEB-Services*). Turėdamas nuo programavimo kalbos nepriklausomą klasių modelį, klientas žinos, kaip bendrauti su serveriu. Kadangi darbas su duomenimis atliekamas programavimo kalbos klasių realizacijoje, nėra pagrindo stengtis dubliuoti klasių metodų kodą duomenų bazėje. Šiuo atveju galima tik bandyti perkelti tam tikras kodo dalis į trigerius arba sąryšinių duomenų bazių procedūras, tačiau tai nėra sąsajos su sąryšine schema dalis. Todėl toliau programinio kodo perkėlimo galimybės nebus nagrinėjamos, nors kai kurios duomenų bazės teikiamos galimybės gali praversti, siekiant užtikrinti duomenų integralumą.

3.5 Objektų sąsaja su sąryšine schema

Ankstesniame skyrelyje aprašyti būdai, kaip priklausomai nuo pasirinktos strategijos klasės gali būti susiejamos su viena ar keliomis lentelėmis. Kiekvienas objektas šiose lentelėse gali būti išsaugomas įrašų pavidalu. Kadangi objektai yra dinamiški, keletas jų savybių gali būti žinomos tik sistemos veikimo metu, t.y. ryšių tipą bei objektams priklausančių duomenų integralumą užtikrina unikalių objektų identifikatorių priskyrimo mechanizmas.

3.5.1 Objekto unikalumo ir identiškumo užtikrinimas

Objekto identiškumą galima garantuoti trimis lygiais: pirma, klasės viduje; antra, klasių hierarchijos lygyje; ir trečia, visų klasių atžvilgiu.

- Unikalumas klasės viduje. Šiuo atveju du skirtingų klasių tipo objektai gali turėti tą patį unikalų identifikatorių. Trūkumas yra tai, kad kiltų problemos su polimorfizmu.
- Unikalumas klasių hierarchijos lygyje. Jei objekto identifikatorius yra unikalus klasių hierarchijos lygyje, tuomet poliformizmas palaikomas pilnai, nes objektui keičiant savo vaidmenį, jo identifikatorius vis tiek lieka unikalus.
- Unikalumas visų klasių atžvilgiu. Jei objekto identifikatorius unikalus visų klasių atžvilgiu, tokiu atveju sistemoje nebus dviejų klasių su sutampančiais identifikaciniais numeriais.

Kalbant apie objektų saugojimą duomenų bazėje, yra keletas atvejų, kai kyla su objekto unikalumu susijusios problemos:

- Objekto iš įrašų lentelėse generavimo metu. Sąryšinėje duomenų bazėje objekto unikalumą garantuoja unikalus identifikatorius, kuris tampa pirminiu raktu. Tačiau kas atsitinka tuomet, kai informacija palieka duomenų bazę ir tampa objektu jau nesančiu duomenų bazės integralumo mechanizmo valdose? Atliekant to paties objekto paiešką kelis kartus, turi būti gražinamas tas pats objektas, o ne skirtingos jo kopijos.
- Naujo objekto išsaugojimo duomenų bazėje metu. Šiuo atveju naujas objekto identifikatorius turi būti sugeneruotas taip, kad jis būtų globaliai teisingas. Būtent objektų išsaugojimo sąryšinėje duomenų bazėje mechanizmas turi generuoti naujus identifikacinius numerius, kadangi objekto kūrėjas egzistuoja atskiroje, jam vieninteliam būdingoje adresų erdvėje, kurioje objekto identifikatorių generavimas gali neatitikti integralumo reikalavimų.

Pirmuoju atveju, problemos sprendimas yra naudoti nepriklausomą nuo duomenų bazės objektų spartinančiosios atminties mechanizmą. Gavę užklausą tam tikram objektui pirmiausia patikriname, ar jo dar nėra spartinančioje atmintyje. Jeigu objektas jau egzistuoja spartinančioje atmintyje, tokiu atveju gražiname nuorodą į tą objektą. Jeigu objektas neegzistuoja, tuomet jis kuriamas iš sąryšinių duomenų. Objektų transakcijų mechanizmas saugo informaciją apie objektų dalyvavimą transakcijose. Priklausomai nuo pasirinktos duomenų apribojimo (užrakinimo) strategijos (optimistinis/pesimistinis užrakinimas), gavus kreipinį į tam tikrą objektą apie jo dalyvavimą transakcijoje jį galima leisti arba atmesti.

Norint išspręsti antrąją problemą, reikia suprogramuoti mechanizmą, kuriantį naujus ir unikalius objektų identifikatorius. Šie identifikatoriai gali būti naudojami kaip pirminiai raktai. Tolimesniuose skyriuose bus apžvelgiami keletas tokių identifikatorių kūrimo būdų.

3.5.2 Pirminis raktas ir unikalus objekto identifikatorius

Pirminis raktas užtikrina unikalumą tik duomenų bazės lentelės lygyje. Jeigu ruošiamasi išsaugoti naują objektą, šis objektas turi turėti naują unikalų identifikatorių, kitaip duomenų bazės pirminio rakto tikrinimo mechanizmas atmes naujo įrašo išsaugojimą. Tuomet kyla klausimas: kaip sugeneruoti unikalų objekto identifikatorių, kuris būtų nepriklausomas nuo objekto kintamųjų (kadangi klasės aprašas gali keistis ir tai sukeltų daug problemų susijusių su jau egzistuojančiu objektų panaudojimu atnaujintoje sistemoje) bei globalus?

3.5.3 Unikalių objekto identifikatorių generavimo būdai

Yra du skirtingi būdai, kuriais galima užtikrinti generuojamų objektų identifikatorių unikalumą:

1. Algoritmas, kuris yra paremtas tam tikru kiekių įeinančių unikalių parametrų, pvz., ethernet adresas, laiko matas (angl. *timestamp*), proceso identifikatorius ir kt.
2. Globaliai administruojamas skaitliukas, pvz., speciali lentelė duomenų bazėje, kurioje saugomi naujai kuriamų objektų identifikatoriai.

   Toliau pateikti penki generavimo būdai remiasi vienu iš šių dviejų metodų.

3.5.3.1 Timestamp, proceso identifikatorius bei ethernet adresas

Naudojantis šiuo metodu, trijų parametrų – laiko mato (angl. *timestamp*), proceso identifikatoriaus bei ethernet adreso – kombinacijos pagalba galima sugeneruoti unikalų numerį. Kiekviena tinklo plokštė turi unikalų pasaulyje ethernet adresą, laiko matas kiekvienu duotuoju momentu taip pat yra unikalus, o operacinė sistema užtikrina, kad procesai gautų skirtingus procesų identifikacinius numerius.

3.5.3.2 SQL MAX() funkcija

Šis metodas taip pat paremtas pirmuoju metodu. Šiuo atveju lentelėje pirminis raktas yra sveikasis skaičius. Kiekvieną kartą, kai reikia naujo objekto identifikacinio numerio, naudojama SQL MAX() funkcija, grąžinanti didžiausią šiame stulpelyje esančio įrašo

reikšmę. Ši reikšmė didinama vienetu ir naudojama kaip naujas objekto identifikatorius– pirminis raktas.

Šio metodo trūkumas yra tai, kad priklausomai nuo pasirinktos sąsajos strategijos (filtruotas, horizontalus, vertikalus) unikalumas yra užtikrinamas klasės viduje (horizontali sąsaja), klasių hierarchijoje (filtruota sąsaja), tačiau neužtikrinamas unikalumas tarp visų klasių. Be to, naudojant MAX() funkciją reikia laikinai užrakinti duomenis, o tai mažina sistemos veikimo greitį.

3.5.3.3 Raktas/ Reikšmės lentelė

Šis būdas paremtas antruoju generavimo metodu, kadangi jis naudoja specialiai tam skirtą sekų lentelę. Lentelėje yra du stulpeliai: vienas skirtas lentelės, kurioje saugomi objekto įrašai, pavadinimui, antrasis yra skaitliukas. Kai tam tikrai lentelei (–ėms) arba objektui reikalingas naujas objekto identifikatorius/ pirminis raktas(–ai), eilutėje su lentelės pavadinimu yra kitas objekto identifikatorius. Jei sistemai reikia naujo identifikatoriaus, pasinaudojame sekančiu numeriu sekų lentelėje. Tai padeda išvengti lentelės, iš kurios skaitomi objektui priklausantys duomenys, užrakinimo, tačiau pati sekų lentelė gali lėtinti sistemos veikimą. Problemos esmė yra tokia, kad norint generuoti unikalius identifikacinius numerius klasių hierarchijos lygiu ar netgi visų klasių, sekų lentelėje turi būti įrašas, skirtas visai hierarchijai, arba vienas vienintelis įrašas, skirtas visiems objektų identifikaciniams numeriams generuoti. Šiuo atveju tai gali tapti rimta problema, kadangi kiekvieno naujo objekto kūrimo metu generuojant unikalų identifikatorių šiam objektui, sekų lentelė bus užrakinama.

3.5.3.4 High/Low algoritmas

Šis algoritmas taip pat naudoja sekų lentelę, tačiau lentelėje saugoma pradinė objekto identifikatoriaus reikšmė. Jei sistemai reikia naujo objekto identifikatoriaus tam tikram objektui, nuskaitoma reikšmė iš sekų lentelės; ta reikšmė padidinama tam tikru skaičiumi ir įrašoma atgal į lentelę. Turimas objektų identifikatorių reikšmių diapazonas [n..n+m], kur n – tai iš sekų lentelės gauta reikšmė, m – tam tikras diapazono dydį apibrėžiantis skaičius. Jis naudojamas jau pačioje klasėje – prireikus naujo objekto identifikatoriaus, skaičius padidinamas vienetu. Taip gauname m unikalių objekto identifikatorių. Kai išnaudojamas visas diapazonas, iš sekų lentelės imama nauja reikšmė. Šio algoritmo privalumas yra tas, kad kreipiniai į sekų lentelę sumažėja m kartų.

3.5.3.5 Duomenų bazės teikiamos galimybės

Kai kurios duomenų bazių sistemos, pvz., Oracle, siūlo priemones unikalių identifikacinių numerių generavimui. Tačiau panaudojus šias priemones kiltų problemos, norint pakeisti duomenų bazę. Šias priemones galima naudoti abiems šiame skyriuje paminėtiems objektų identifikatorių/ pirminių raktų generavimo būdams. Trečioje lentelėje pavaizduotas pateiktų penkių unikalių objekto identifikatorių generavimo metodų palyginimas.

3 lent. Identifikatorių generavimo būdų palyginimas

		Palaikymo lygis			
Strategija	**Principas**	**Klasės**	**Klasių hierarchija**	**Visos klasės**	**Trūkumai**
Timestamp/PID/Eth	Algoritmas	Taip	Taip	Taip	Nėra
MAX() funkcija	Skaitliukas	Taip	Taip (filtruotai sąsajai)	Ne	Dažnas duomenų užrakinimas
Raktas/Reikšmė lentelė	Skaitliukas	Taip	Taip	Taip	Sekų lentelė
High/Low algoritmas	Skaitliukas	Taip	Taip	Taip	Sekų lentelė
Duomenų bazės mechanizmai	Algoritmas/ Skaitliukas	Taip	Taip	Taip	Keičiant duomenų bazę reikalingi pakeitimai

3.6 Sąsajos elementų susiejimo būdų apibendrinimas

Klasių kintamieji susiejami su lentelių stulpeliais, klasių vardai su lentelėmis. Objektai išsaugomi kaip įrašai sąryšinės schemos lentelėse. Stulpelių bei objekto kintamųjų tipai turi būti ekvivalentūs arba reikia atlikti tipų konvertavimą.

Jeigu naudojama tik viena objektinė programavimo kalba, kuri naudoja saugomus duomenis, klasių metodų saugoti nereikia, kadangi jie saugomi kartu su programa ir yra įtraukti į klasės aprašą. Jiems sąsaja nereikalinga. Jeigu naudojamos kelios programavimo kalbos, saugotini objektai turi būti aprašyti nepriklausomu nuo kalbos būdu; objektai generuojami panaudojant šį aprašą. Paveldėti kintamieji gali būti išsaugomi trimis būdais: filtruota, horizontalia arba vertikalia sąsaja. Šie skirtingi būdai skiriasi polimorfizmo palaikymu, pakeitimais klasių struktūroje bei lentelių kiekiu. Sąryšinėje duomenų bazėje

ryšiai tarp objektų saugomi kaip išorinių/pirminių raktų poros. Ryšiai tarp objektų skirstomi į asociacijas, agregacijas bei kompozicijas. Jų saugojimas skiriasi tuo, kad įrašuose išoriniai raktai gali būti turėti null reikšmę (asociacija), išoriniai raktai įrašuose negali būti null reikšmė (agregacija) arba gali būti išorinis raktas tik į vieną tam tikrą lentelę (kompozicija). Galimi trys skirtingi ryšių išsaugojimo būdai: vidinės klasės, išorinių raktų ir ryšių lentelės. Vienakrypčių ryšių sąsaja paprastesnė negu dvikrypčių.

Sąryšinėse duomenų bazėse įrašų unikalumą užtikrina pirminiai raktai, tuo tarpu objektų unikalumą užtikrina objektų identifikatoriai. Norint išsaugoti objektą sąryšinėje duomenų bazėje, sugeneruojamas naujas objekto identifikatorius, kuris tuo pačiu naudojamas ir kaip pirminis raktas. Objekto identifikatoriai gali būti unikalūs trimis atžvilgiais: klasės viduje, klasės hierarchijoje arba visų klasių atžvilgiu. Dviejų principų pagrindu yra penki pagrindiniai unikalaus objekto identifikatoriaus generavimo mechanizmai, kurių kiekvienas turi savo privalumus ir trūkumus.

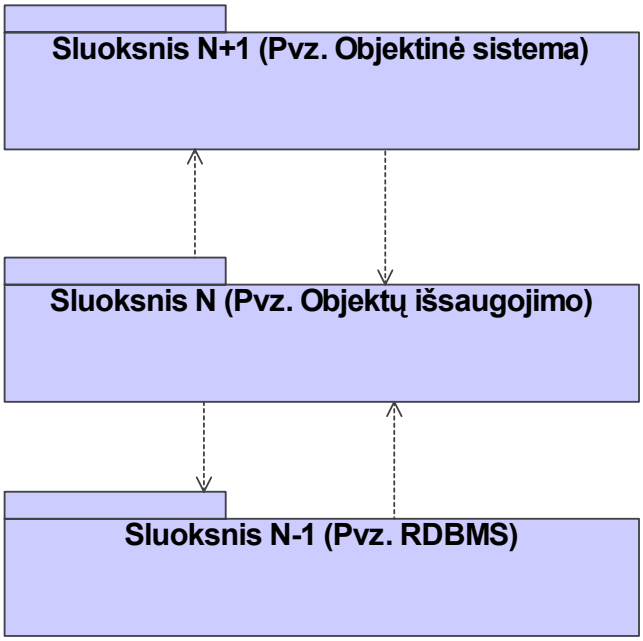
4 TYRIMAI IR JŲ REZULTATAI

4.1 Objektų saugojimo sistemos

Minėti objektinių – sąryšinių sistemų skirtumai yra aktualūs ne vienai sistemai, tačiau bet kokioms objektinėms sistemoms, duomenų saugojimui naudojančioms sąryšines duomenų bazes, yra natūralu galvoti apie šios problemos sprendimui skirtas posistemes. Pagrindinis reikalavimas tokiai posistemei yra tas, kad ji palaikytų CRUD (angl. *Create/Retrieve/Update/Delete* – Sukurti/Išrinkti/Atnaujinti/Ištrinti – pagrindinės duomenų bazės operacijos) savybių aibę objektų lygyje.

4.1.1 Reikalavimai objektų saugojimo posistemiui

Kuriant sistemą pirmiausia būtina nuspręsti, kokia bus jos architektūra. Didesni programiniai moduliai dažniausiai yra suskirstomi į sluoksnius. Sluoksnis – tai didesnis programinis modulis, skirtas bendravimui su daugiausia dviem programiniais sluoksniais – vienu, esančiu virš jo (jei yra), arba po juo (jei yra) per aprašytą programinę sąsają. Sąsajoje apibrėžti visi veiksmai, kuriuos galima atlikti su sluoksniu ir kaip tie veiksmai gali būti atlikti (perduodami parametrai, gražinami tipai, klasių aprašai). Sluoksnio realizacija yra paslėpta sluoksnio viduje. N–tasis sluoksnis skiria n+1 sluoksnį nuo n-1 sluoksnio. Bendravimas tarp N-1 bei N+1 sluoksnių yra negalimas. Naudojant sluoksniavimo metodą galima programas padaryti visiškai nepriklausomas nuo duomenų bazės – visas bendravimas su duomenų baze vyksta per objektų išsaugojimo sluoksnį (8 pav.).



8 pav. Sluoksnių tipo sistemos architektūra

Kita problema – objektų saugojimas. Kadangi sistemos yra rašomos objektinėmis kalbomis, bendravimas su objektų saugojimo sluoksniu taip pat turi būti objektinis, tai yra naudojantis metodais ir kintamaisiais. Remiantis savybe, jog objektų išsaugojimo sluoksnis padaro programą nepriklausomą nuo duomenų bazės, joje galima keisti lentelių struktūrą (pvz., normalizuoti ar denormalizuoti) be jokių pasekmių pačiai programai. Dar viena problema – tai lygiagretumo užtikrinimas, nes paprastai objektų išsaugojimo sluoksniu gali naudotis kelios programos arba netgi ta pati programa iš kelių gijų (angl. *threads*). Sluoksnis turi užtikrinti lygiagretų priėjimą prie duomenų ir turi būti realizuotas taip, kad palaikytų pagrindinį transakcijų mechanizmo principą – ACID (angl. *ACID: Atomicity/ Consistency/ Isolated Execution/ Durability* – dalumas, vientisumas, izoliacija, trukmė – pagrindinės savybės kurias turi palaikyti transakcijų mechanizmas).

Ketvirtas klausimas yra bendravimo tarp sluoksnių būdo pasirinkimas. Objektinėms sistemoms yra įprastas yra kliento/serverio bendravimo mechanizmas. Taigi galime daryti tokias išvadas:

- Sąsajos su sąryšine schema informacija turi būti saugoma struktūrizuotu aprašu. Toks aprašas vadinamas metaduomenimis. Objektų išsaugojimo sluoksnis gali aptarnauti daug posistemių, todėl kiekvienam posistemiui priklausanti metaduomenys turi būti efektyviai ir struktūrizuotai saugoma ir palaikoma;
- Objektinėms programų dalims nereikės SQL kodo, kadangi sluoksniavimo technika leidžia visiškai nuo jų atskirti duomenų bazę. Šis sprendimas yra pats lanksčiausias, palyginus su klasės viduje esančiu SQL kodu ar jo generavimu;
- Kliento/serverio architektūroje objektų lygyje palaikant transakcijų mechanizmą turi būti užtikrinamas lygiagretus priėjimas prie tų pačių išsaugotų objektų.

Atsižvelgiant į šiuos kriterijus iškyla klausimas: ką turi ir privalo užtikrinti objektų išsaugojimo sluoksnis?

4.1.1.1 Būtini reikalavimai objektų išsaugojimo sluoksniui

Pagrindinė objektų išsaugojimo sluoksnio užduotis yra paslėpti objektų sąsajos su sąryšine schema realizaciją, kad realizuojantis verslo procesus programuotojas nesirūpintų tuo, kaip saugomi jo naudojami objektai. Duomenų bazės administratoriaus užduotis yra administruoti duomenų bazę, o objektų išsaugojimo sluoksnis turi užtikrinti tai, kad nereikėtų daryti pakeitimų kode, susijusių su pakeitimais lentelių schemoje. Idealus objektų išsaugojimo sluoksnis turėtų užtikrinti sekančias užduotis:

1. Užtikrinti sistemos nepriklausomumą nuo duomenų bazės. Sistemai tiesiogiai neleidžiama kreiptis į duomenų bazę. Bendravimas su duomenų baze turi vykti tik per objektų išsaugojimo sluoksnį.
2. Išsaugojimo mechanizmo enkapsuliacija. Programinė įranga yra parašyta objektine kalba, bendravimas su objektų išsaugojimo sluoksniu yra atliekamas per metodus. Ši sąsaja turi būti žinoma ir apibrėžta.
3. Skirtingos sąsajos strategijos. Idealus objektų išsaugojimo sluoksnis turėtų suteikti galimybę pasirinkti skirtingus klasių paveldėjimo hierarchijų bei ryšių sąsajos su sąryšine schema būdus, kadangi jie įtakoja sistemos veikimo greitį.
4. Ryšių integralumas. Turi būti užtikrintas toks objektų išsaugojimas, kad juos atstatant išliktų tie patys ryšiai tarp objektų.
5. Objektų identifikatoriai. Bandant išsaugoti naują objektą, objektų išsaugojimo sluoksnis turi sugeneruoti naują objekto identifikatorių, naudodamasis prieš tai aprašytais algoritmais ar kitais saugiais būdais, užtikrinančiais objekto identifikatoriaus unikalumą bei integralumą. Ši užduotis priklauso ne programai, sukūrusiai ir naudojančiai šį objektą, tačiau objektų išsaugojimo sluoksniui.
6. Transakcijų mechanizmas. Kadangi paprastai daugiau negu vienas klientas lygiagrečiai naudojasi tais pačiais duomenimis, reikia užtikrinti ACID transakcijas objektų lygyje.
7. Užklausų kalba. Kadangi virš objektų išsaugojimo sluoksnio esantys programiniai sluoksniai tiesiogiai nebendrauja su duomenų baze, nes naudojasi tik jos suteikiamais metodais, turi būti nustatytas objektų paėmimo iš šio sluoksnio būdas; tam SQL užklausų kalba netinka, kadangi ji: a) priverčia žinoti apie lenteles ir jų stulpelius (o kaip tik šito reikia stengtis išvengti) ir b) ji nėra objektinė. Šiuo atveju reikalinga objektinė užklausų kalba. Tokia kalba gali būti OQL – objektų užklausų kalba (angl. *OQL – Object Query Language*), arba bet kuri kita kalba, turinti tas pačias savybes, t.y. objektų išrinkimas pagal atributus ir kt.

4.1.1.2 Papildomi reikalavimai objektų išsaugojimo sluoksniui

Aukščiau išvardinti reikalavimai yra būtini objektų išsaugojimo sluoksniui. Visi objektų išsaugojimo sluoksniai šiuos reikalavimus užtikrinti privalo. Prie šių reikalavimų galima pridėti keletą kitų, kurie teigiamai įtakotų sistemą.

1. Standartizuotas priėjimas prie duomenų bazės. Jei programa sluoksniuotos architektūros dėka tampa atskirta nuo duomenų bazės ir jei naudojamas standartinis

priėjimas prie duomenų bazės, pvz., toks, kaip Java programavimo kalboje esanti JDBC sąsaja, tuomet būtų galima pakeisti duomenų bazę nekeičiant programos kodo, pvz., Microsoft SQL Server į Oracle.

2. Žymeklio (angl. *cursor*) palaikymas. Šiuo atveju žymeklis naudojamas objektų išrinkimui, o ne leidžia išrinkti norimus sąryšinius įrašus.
3. Tarpiniai (angl. *Proxy*) objektai. Jeigu iš visos ryšiais susijusios hierarchijos reikalingi ne visi, o tik vienas ar keletas objektų, naudojama technika, vadinama tarpiniais objektais. Tarpinis objektas – tai toks objektas, kurio vidiniams kintamiesiems dar nesuteiktos sąryšinėje duomenų bazėje esančios reikšmės, tačiau tai padaroma tuomet, kai jų prireikia, t.y. tuomet, kai į juos atliekamas pirmas kreipinys. Kitaip sakant, objektai yra užkraunami tuomet, kai jie yra reikalingi.
4. Objektų spartinančioji atmintis. Objektų spartinančiosios atminties naudojimas gali labai pagreitinti sistemos darbą, kadangi smarkiai sumažėja kreipinių į duomenų bazę.
5. Centralizuoti metaduomenys. Idealiu atveju metaduomenys turėtų saugoti struktūrinę informaciją. Duomenų bazės pasikeitimo atveju, o klasių struktūrai nekintant, reikėtų keisti tik metaduomenis.
6. Išankstinis duomenų užkrovimas. Jeigu numanoma, kad tuoj reikės tam tikrų duomenų, šie duomenys gali būti užkraunami antrame plane ir susiejami automatiškai. Tai naudojama kartu su tarpiniais objektais ir objektų spartinančiąja atmintimi.
7. Architektūrinis modelis. Objektų išsaugojimo sluoksnis suteikia tam tikrus servisus. Jis gali būti realizuotas kaip serveris, o kiti programų sluoksniai gali būti kaip jo klientai. Yra keletas būdų, kaip realizuoti kliento/serverio architektūros modelį, tačiau esminis dalykas yra kliento/serverio bendravimo protokolas. Didelis privalumas yra paskirstytų OO protokolų palaikymas, tokių kaip CORBA, žiniatinklio paslaugos, ar specifinių Java programavimo kalbai RMI/EJB.
8. Paskirstytos duomenų bazės. Sąryšinės duomenų bazės dažnai būna paskirstytos. Jeigu klasių schema susiejama su dviem arba daugiau sąryšinių schemų, kurios egzistuoja viena nepriklausomai nuo kitos skirtingose duomenų bazėse, į tai turi būti atsižvelgta transakcijų palaikymo mechanizmo realizacijoje.
9. Prisijungimų prie duomenų bazės spartinančioji atmintis (angl. *database connection cache*). Prisijungimų spartinančioje atmintyje saugomas tam tikras kiekis jau esamų prisijungimų prie sąryšinės duomenų bazės. Jei objektų išsaugojimo sluoksniu vienu metu naudojasi daug klientų, esamų prisijungimų kiekis prie duomenų bazės vaidina

svarbų vaidmenį. Prisijungimų spartinančioji atmintis leidžia efektyviai valdyti tam tikrą turimą prisijungimų kiekį, paskirstant prisijungimus pagal kiekvieno kliento poreikius.

10. Sąsajai skirta grafinė vartotojo sąsaja paremta programinė įranga. Galimybė atlikti objektų sąsają su sąryšine schema naudojantis grafinę sąsają turinčia programine įranga yra didelis privalumas. Idealiu atveju ši programinė įranga turėtų leisti ne tik matyti klases, lenteles ir ryšius tarp jų, bet ir juos redaguoti. Svarbi savybė yra tai, kad ši programinė įranga leistų priekinę inžineriją (remiantis įvesta informacija sukurtų lenteles ir klases) ir apgrąžos inžineriją (sąsajos informacijos generacija iš esamų klasių ir lentelių).
11. Užrakinimo būdai. Idealus objektų išsaugojimo sluoksnis turėtų suteikti galimybę valdyti optimistinį/pesimistinį duomenų užrakinimą objekto lygyje.

4.1.1.3 Programinės įrangos įsigijimo būdai

Programinės įrangos įsigijimo būdai yra du – pirkti arba sukurti patiems.

Atsižvelgiant į visus aukščiau išvardintus reikalavimus galima daryti išvadą, kad objektų išsaugojimo sluoksnio sukūrimas yra nelengva užduotis, nes ji reikalauja žinių ir patirties daugelyje sričių, tokių kaip lygiagretus, tinklo, sisteminis programavimas, sąryšinių duomenų bazių programavimas, tvarkyklių programavimas, tokių standartų kaip SQL, OQL, CORBA, žiniatinklio paslaugos, JDBC, EJB, RMI ir kt. išmanymas.

Rinkoje egzistuoja šiam uždaviniui spręsti skirti produktai, tačiau prieš nusprendžiant kurį nors iš jų naudoti, būtina atlikti tyrimą, remiantis aukščiau išvardintais kriterijais bei įvertinant tinkamumą konkrečiu atveju.

Pagrindinis egzistuojančio produkto pirkimo privalumas yra tas, kad jo nereikia daryti patiems. Programuotojams ir duomenų bazių administratoriams turi būti atliekami apmokymai, kaip naudotis produktu, o tiekėjas suteikia techninę pagalbą. Dar vienas privalumas yra tai, kad produktą galima nuodugniai įvertinti prieš atliekant pirkimą, tuo tarpu pačių programavimo įvertinimą galime atlikti tik tada, kai produktas jau baigtas, o tuo metu situacija rinkoje jau gali būti visiškai pasikeitusi.

Nėra daug argumentų, kodėl tokį produktą, kuris jau egzistuoja rinkoje ir tapęs standartu, reikėtų daryti patiems. Šiuo atveju pagrindiniai argumentai yra tokie, kad gali bankrutuoti programinę įranga sukūrusi įmonė ir nebebus galimybės atnaujinti versiją, nebebus techninės pagalbos, o programos kodas ir dokumentacija nebus prieinami. Pagrindiniai veiksniai, lemiantys pasirinkimą daryti objektų saugojimo sluoksnį patiems, yra

didelė rinkoje esančių produktų kaina ir noras patiems kontroliuoti produkto palaikomą sąsają bei funkcijas.

4.2 Egzistuojančio produkto pasirinkimas

Rinkoje yra daug produktų, kurie kuriami skirtingiems uždaviniams spręsti, tuo pačiu jiems keliami ir skirtingi reikalavimai. Todėl tam, kad atlikti jų analizę, reikalingi tam tikri kriterijai, pagal kuriuos būtų galima atlikti palyginimą.

4.2.1 Būtini vertinimo kriterijai

Šiame darbe objektų saugojimo karkasui įvertinti bus naudojami tokie kriterijai:

a) Java palaikymas. Bus nagrinėjami tik Java palaikantys objektų išsaugojimo karkasai.
b) Duomenų bazių palaikymas. Būtina sąlyga yra tai, kad karkasas palaikytų Oracle duomenų bazę bei Microsoft SQL Server.
c) Sąsaja su duomenų baze. Sąsaja su duomenų baze turi būti atliekama per Java JDBC. Karkasas turi palaikyti standartines Oracle bei Microsoft SQL Server tvarkykles.
d) Tik Java programavimo kalba. Karkasas turi būti paremtas tik Java programavimo kalba tam, kad būtų lengvai pernešamas tarp skirtingų platformų.
e) Objektų spartinančioji atmintis. Sistema turi palaikyti daug vartotojų ir didelius duomenų kiekius, todėl būtina, kad karkasas palaikytų spartinančiąją atmintį objektų lygyje.
f) Standartų palaikymas. Yra keletas objektų saugojimo sistemoms skirtų ar bent iš dalies jas liečiančių standartų. Tokie standartai yra JDO, EJB ar ODMG. Kuo daugiau standartų objektų saugojimo karkasas palaiko tuo didesnė tikimybė, kad jį lengvai pavyks pritaikyti jau esamoms sistemoms per trumpesnį realizacijos laiką ir mažesnėmis sąnaudomis.

4.2.2 Šiuo metu rinkoje esantys objektų išsaugojimo karkasai

Šiuo metu rinkoje patys populiariausi yra sekantys objektų su sąryšinėmis duomenų bazėmis sąsajos karkasai:

- Cayenne. Palyginti naujas, nemokamai platinamas produktas, turintis daugelį komerciniams produktams būdingų savybių.

- KodoJDO. Solarmetric kompanija buvo viena iš pirmųjų, kuri realizavo ir pateikė į rinką Sun Microsystems JDO specifikaciją atitinkantį produktą KodoJDO. Šiuo metu tai vienas iš labiausiai paplitusių JDO realizuojančių produktų.
- TopLink. Nuo pat Java programavimo kalbos atsiradimo 1996 metais kompanija „ObjectPeople“ dirbo objektų susiejimo su sąryšinėmis duomenų bazėmis srityje. Šiuo metu kompanijos sukurtas produktas „TopLink“ priklauso Oracle.
- Hibernate. Atvirojo programinio kodo nemokamai platinamas produktas, palaikantis daugybę funkcijų. Nuo pat Hibernate produkto išleidimo pradžios šis produktas tapo vienu pirmaujančių objektinės – sąryšinės sąsajos produktų, labiausiai paplitusiu tarp paskirstytų sistemų programuotojų [3].
- Apache OJB. Apache – tai organizacija, kurioje sukurta daug nemokamai platinamų atvirojo kodo bibliotekų, daugelis iš jų labai plačiai naudojamos ir kai kurios net įtrauktos į Java kalbos programinę realizaciją. Turbūt šiandien nėra Java sukurtos sudėtingos sistemos, kuri nenaudotų bent vienos Apache bibliotekos (XML apdorojimui, programos trasavimo informacijos išvedimui/apdorojimui ir kt.).

Remiantis šių produktų specifikacijomis bei programuotojo vadovais [1,3,6,7,11,15] buvo sudaryta šių produktų savybių lentelė (4 lentelė).

4 lent. Objektų saugojimo karkasų savybių palyginimas

Savybės	Cayenne	KodoJDO	TopLink	Hibernate	JakartaOJB
Nemokama	Taip	Ne	Ne	Taip	Taip
Programos kodas	Taip	Ne	Iš dalies	Taip	Taip
Grafinė vartotojo sąsaja	Taip	Taip	Taip	Ne	Taip
Generuoja SQL	Taip	Taip	Taip	Taip	Taip
Duomenų objektas neturi būti paveldėtas iš specialios klasės/neturi realizuoti specialios sąsajos	Ne	Taip	Taip	Taip	Taip
RDBMS palaikymas (Oracle/ MS SQL Server)	Taip	Taip	Taip	Taip	Taip
Ryšių tarp objektų palaikymas	Taip	Taip	Taip	Taip	Taip
GROUP BY funkcijos palaikymas	Taip	Taip	Taip	Taip	Taip
Duomenų užkrovimas pagal poreikį	Iš dalies	Taip	Taip	Taip	Iš dalies
Gražina tuos pačius objektus jei sutampa objekto identifikatorius	Taip	Taip	Taip	Taip	Ne
Ciklinių ryšių užkrovimas	Ne	Taip	Taip	Taip	Ne
Sąsajos informacijos bei duomenų objektų	Taip	Taip	Taip	Taip	Taip

generavimas					
Sudėtinių pirminių raktų palaikymas	Taip	Taip	Taip	Taip	Ne
1:M, N:M ryšiai	Taip	Taip	Taip	Taip	Taip
Optimistinis užraktas/įrašų versijos	Taip	Taip	Taip	Taip	Taip
Objekto lygio transakcijų palaikymas	Taip	Taip	Taip	Taip	Taip
JDO programinės sąsajos palaikymas	Ne	Taip	Taip	Ne	Ne
Papildomų lentelių poreikis transakcijų bei užraktų mechanizmo užtikrinimui	Taip	Ne	Ne	Ne	Ne
Galima užtikrinti sinchronizaciją tarp paskirstytų sistemų	Taip	Taip	Taip	Taip	Taip
Užklausų spartinančioji atmintis	Ne	Taip	Taip	Taip	Taip
Palaiko vienos klasės sąsają su keliomis lentelėmis	Taip	Taip	Taip	Taip	Taip
Palaiko kelių klasių sąsają su viena lentele	Taip	Taip	Ne	Taip	Taip
Užtikrina objektų egzistavimą praradus ryšį su duomenų baze ir jam kiek vėliau atsiradus	Ne	Taip	Ne	Ne	Ne

SQL užklausų trasavimo galimybės	Taip	Taip	Ne	Taip	Taip
Palaiko objektų spartinančiosios atminties išvalymą (pvz. jei atliekame tiesioginius veiksmus su duomenimis)	Taip	Taip	Ne	Taip	Ne
Surinkti taškai	**18**	**23**	**19**	**22**	**19**

Visi šie objektų išsaugojimo karkasai palaiko pagrindinius tokiam karkasui keliamus reikalavimus. Toliau bus tiriami tik du iš aukščiau aprašytų produktų, surinkę didžiausią taškų skaičių pagal 4 lentelėje išvardintus kriterijus – KodoJDO (23 taškai) ir Hibernate (22 taškai).

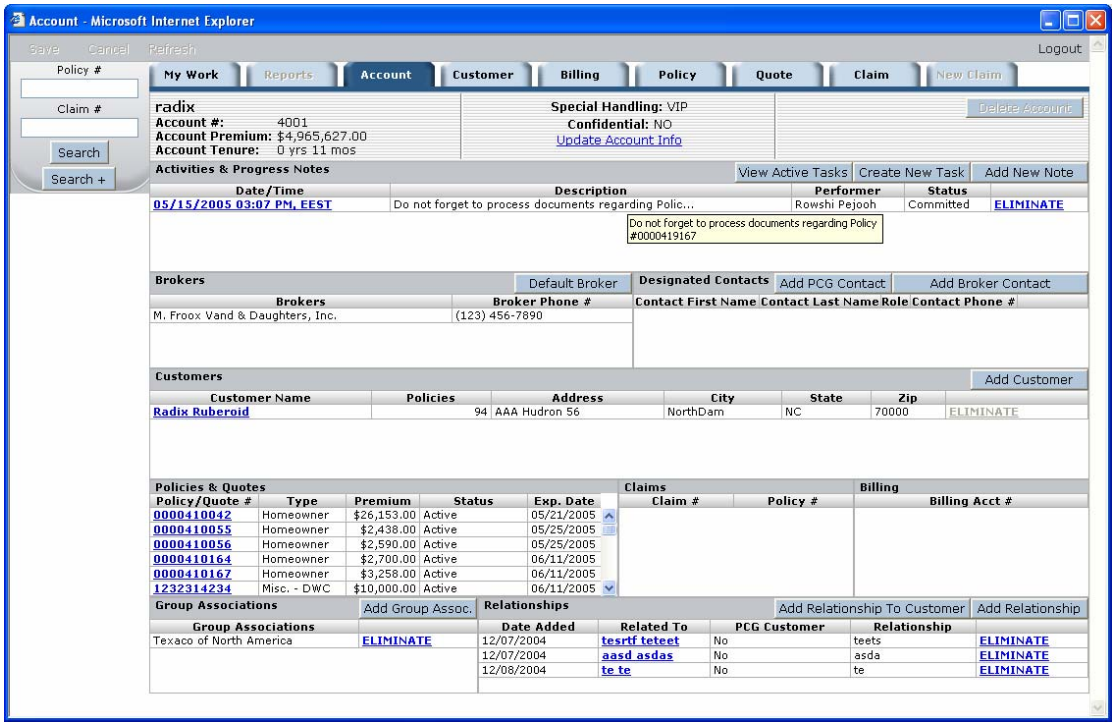
4.3 Pagrindiniai tyrimo uždaviniai

Pagrindiniai šio tyrimo uždaviniai yra du:

1. Išnagrinėti, kaip objektų saugojimo karkasai atlieka ryšių saugojimą;
2. Atlikti sistemos veikimo greičio tyrimą.

4.3.1 Nagrinėjamoji sistema

Testavimui bus naudojama WEB tipo sistema. Pasirinkta kompanijos Exigen draudimo kompanijoms skirta programinė įranga IPB (angl. *Insurance Process Backbone* – Draudimo Procesų Sistemos Karkasas). Tai realiai veikianti sistema, kuria naudojasi viena didžiausių Amerikos draudimo kompanijų. 9 paveiksle pateiktas bendras sistemos vaizdas.



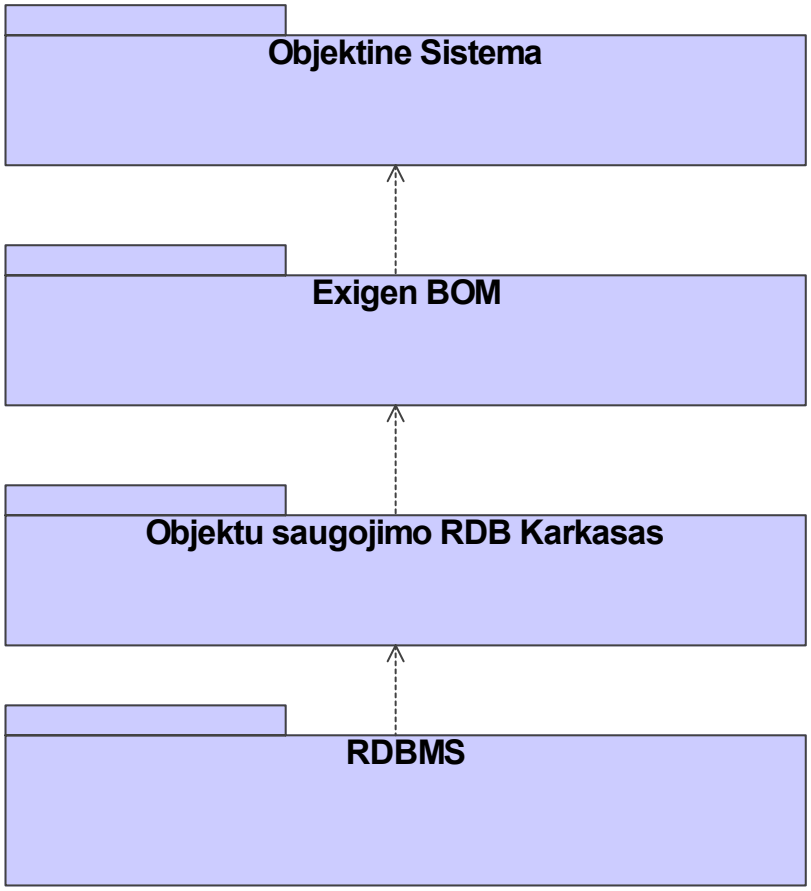
9 pav. Exigen IPB bendras sistemos vaizdas

Sistema sukurta Java programavimo kalba, ir joje realizuota 10 paveiksle pavaizduota architektūra. Sistemos architektūra yra sluoksninė, todėl vieni sluoksniai gali būti keičiami kitais, palaikančiais tą pačią sąsają. Todėl joje galima pakeisti mums testavimui reikalingą parametrą – objektų saugojimo karkasą. Nagrinėjamoji sistema suskirstyta į keletą dalių:

1. RDBMS – Sąryšinė duomenų bazė. Tai gali būti bet kokia RDBMS, kuri būtų suderinama su objektų saugojimo sąryšinėse duomenų bazėse karkasu;
2. Objektų saugojimo sąryšinėse duomenų bazėse karkasas. Tai gali būti bet koks programinis vienetas, kuris galėtų bendrauti su Exigen BOM ir RDBMS;
3. Exigen BOM – tai papildomą abstrakcijos lygį įnešantis programinis vienetas, skirtas objektinės sistemos nepriklausomybei nuo RDBMS bei objektų saugojimo sąryšinėse duomenų bazėse užtikrinimui; pagrindinė šio programinio vieneto savybė – jis yra programuojamas tos pačios įmonės, kuri užsiima vartotojo sistemos realizacija;
4. Objektinė sistema – tai galutiniam vartotojui skirta objektinė sistema, mokanti bendrauti su Exigen BOM sąsaja, užtikrinančia duomenų saugojimą sąryšinėse duomenų bazėse per objektų saugojimo sąryšinėse duomenų bazėse karkasą.

Nagrinėjamoji sistema ypatinga tuo, kad galutinė vartotojo sistema idealiu atveju visiškai nepriklauso nei nuo objektų saugojimo sąryšinėse duomenų bazėse karkaso, nei nuo sąryšinės duomenų bazės tipo ar versijos. Šis atvejis idealus užsibrėžtam tyrimui atlikti,

kadangi galima naudoti skirtingus objektų saugojimo sąryšinėse duomenų bazėse karkasus, nagrinėjamuoju atveju tai KodoJDO ir Hibernate.

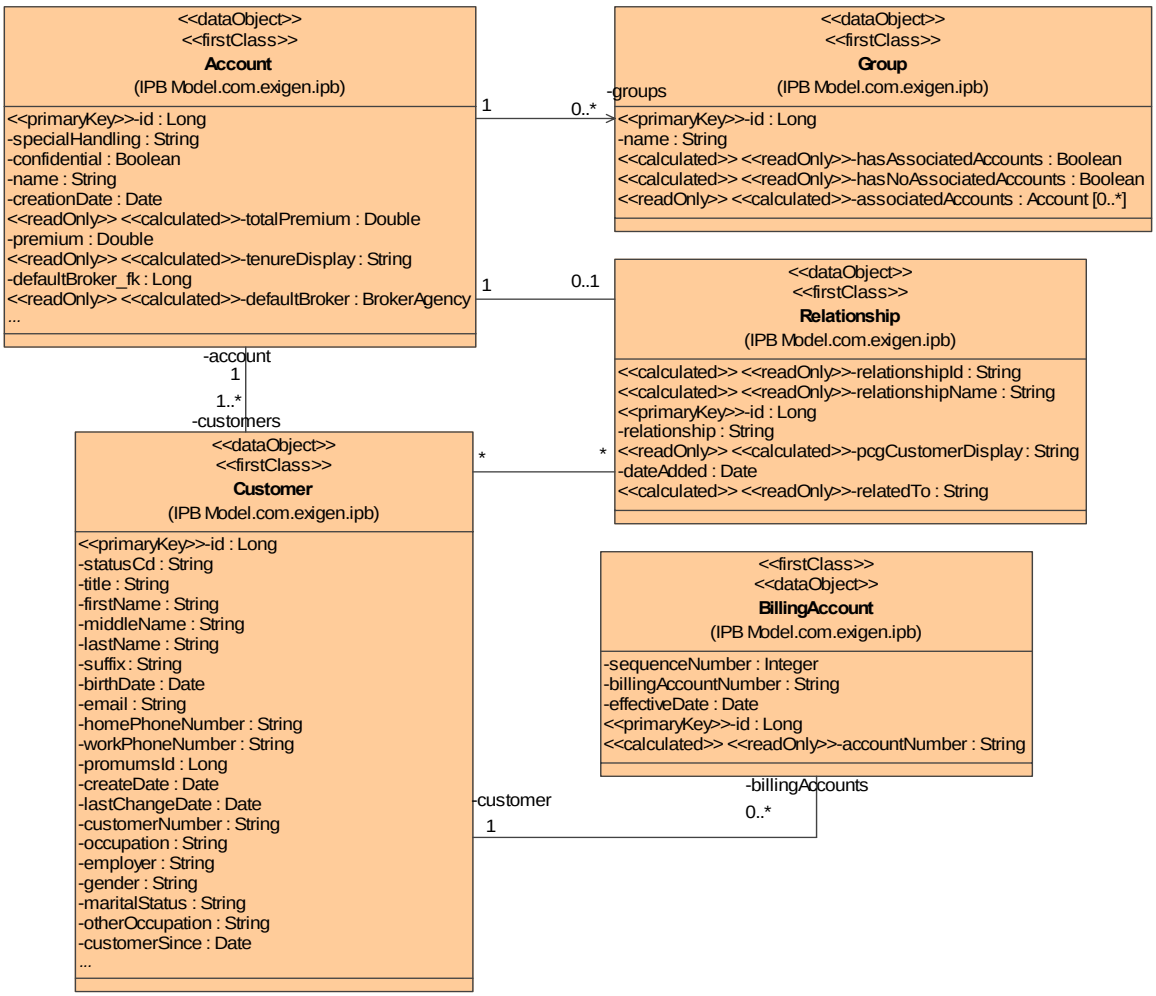


10 pav. Nagrinėjamos sistemos struktūra

4.3.2 Duomenų objektų modelis

11 paveiksle pavaizduota klasių diagrama, kurioje yra nedidelė objektinio duomenų modelio dalis, būdingo daugeliui įmonių [11: 21]. Joje turime:

- Ryšių tipą 1:1 - tarp Account ir Group bei tarp Account ir Relationship;
- Ryšių tipą 1:N – tarp Account ir Group bei tarp Customer ir Billing Account;
- Ryšių tipą N:M – tarp Customer ir Relationship objektų.

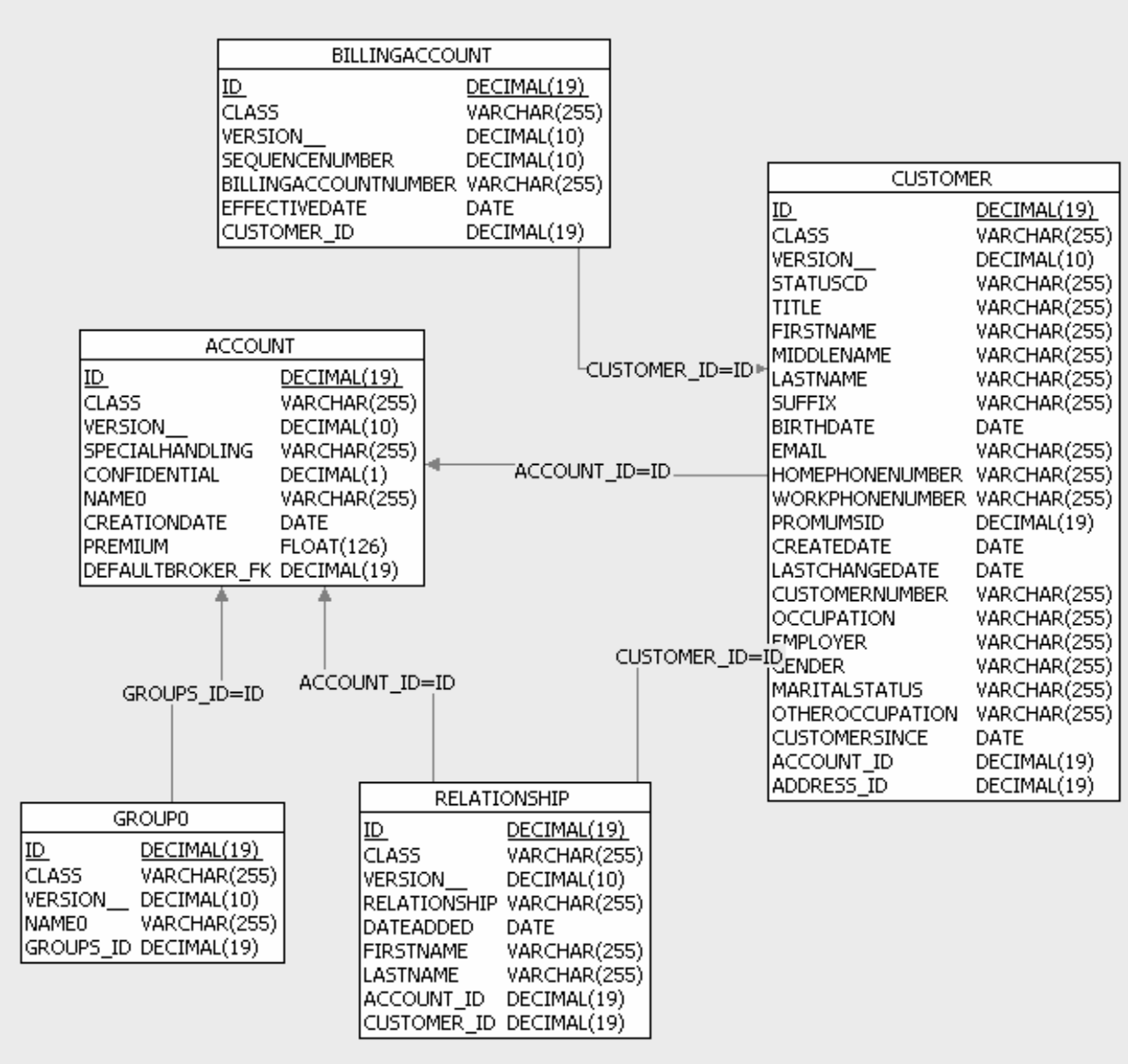


11 pav. Objektinio duomenų modelio klasių diagrama

Toliau šiame darbe bus nagrinėjama, kaip atrodys duomenų bazėje išsaugoti realūs objektai ir visi išvardinti atvejai, be to, bus atliktas sistemos veikimo greičių palyginimo tyrimas su KodoJDO bei Hibernate objektų saugojimo sąryšinėse duomenų bazėse karkasais.

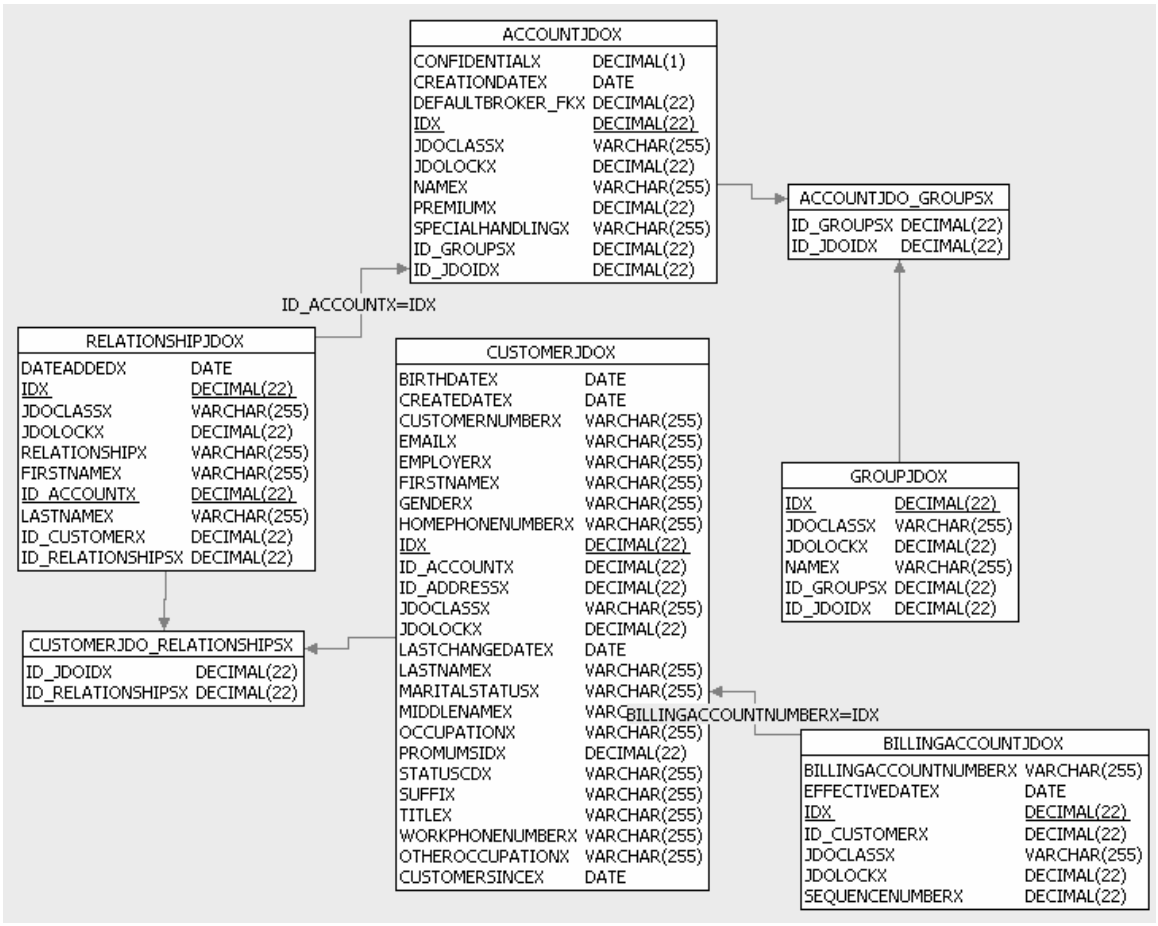
Exigen BOM produktas yra paremtas MDA architektūra (angl. *Model Driven Architecture* – modeliu paremta architektūra). Pirmiausia UML notacijos pagalba aprašomas objektinis modelis, kuriame yra duomenų bei verslo objektai. Duomenų objektai, tai esybės nešančios duomenis – Account, Customer ir kiti verslo objektai. Tai objektai, skirti darbui su duomenų objektais. Duomenų objektams generuojami metaduomenys, jie skirti saugojimui sąryšinėse duomenų bazėse; verslo objektai egzistuoja tik tuomet, kai veikia sistema, t.y. kai ji aktyvi. Pasibaigus sistemos darbui, verslo objektų nelieka, jie duomenų bazėje nesaugomi. Exigen BOM leidžia pasirinkti, kuris objektų saugojimo karkasas bus naudojamas ir priklausomai nuo to sugeneruoja metaduomenis, kuriuose yra saugoma informacija apie tai, kaip objekto esybės bus susietos su sąryšine schema. Taip pat atliekamas ir sąryšinės schemos generavimas, kuris priklauso nuo pasirinkto objektų saugojimo karkaso.

12 paveiksle pateikta Hibernate sugeneruota sąryšinė schema 11 paveikslo klasių diagramai, o 13 paveiksle pateikta sąryšinė schema KodoJDO atveju.



12 pav. Hibernate sugeneruotos sąryšinės schemos ER diagrama

Kaip matyti iš pateiktų paveikslų, abu karkasai sugeneruoja papildomas lenteles ryšių N:M atveju. 14 paveiksle pateikti Account Objektui priklausantys įrašai KodoJDO atveju, o 15 paveiksle – Hibernate atveju. Akivaizdu, kad esminio skirtumo nėra – abu naudoja papildomą stulpelį saugoti duomenų užrakinimo informacijai (JDOLOCKX – KodoJDO, VERSION__ - Hibernate).



13 pav. KodoJDO sugeneruotos sąryšinės schemos ER diagrama

	CONFIDENTIALX	CREATIONDATEX	DEFAULTBROKER_FKX	IDX	JDOCLASSX	JDOLOCKX	NAMEX	PREMIUMX	SPECIALHANDLINGX
1	0		1	9101	com.exigen.ipb.infrastructure.AccountJDO	0	phase2		VIP
2	0		1	0701	com.exigen.ipb.infrastructure.AccountJDO	1	Laurie Munson		VIP
3	0		4	0702	com.exigen.ipb.infrastructure.AccountJDO	0	Christine Nicholasi		VIP
4	0			1501	com.exigen.ipb.infrastructure.AccountJDO	2	MoGilmore		VIP

14 pav. Account objekto įrašai naudojant KodoJDO

	ID	CLASS	VERSION_	SPECIALHANDLING	CONFIDENTIAL	NAME0	CREATIONDATE	PREMIUM	DEFAULTBROKER_FK
1	1	Account.ipb.exigen.com	0	VIP	0	BOM_APT_acc_0001	08-FEB-05		
2	2	Account.ipb.exigen.com	0	VIP	0	BOM_APT_acc_0002	08-FEB-05		
3	3	Account.ipb.exigen.com	0	VIP	0	BOM_APT_acc_0003	08-FEB-05		
4	4	Account.ipb.exigen.com	0	VIP	0	BOM_APT_acc_0004	08-FEB-05		

15 pav. Account objekto įrašai naudojant Hibernate

4.3.3 Sistemos veikimo greičio tyrimas

Web tipo sistemos veikimo greičiui ištirti naudojama programinė įranga, kuri leidžia generuoti HTTP protokolo užklausas, tokiu būdu imituojant interneto naršyklės darbą iš vartotojo kompiuterio. Tyrimui pasirinkta Mercury „Load Runner“. Naudojantis šia programa galima įrašyti veiksmus, atliekamus su interneto naršykle. Programa sugeneruoja

skriptą, kurį vėliau vykdydama „Load Runner“ programa gali imituoti daugelio vartotojų realų darbą su sistema.

4.3.4 Pasirinktas tyrimo scenarijus

Testuosime du atvejus:

1. Kai objektų išsaugojimo karkasas bus Hibernate;
2. Kai objektų išsaugojimo karkasas bus KodoJDO.

Visos kitos testavimo sąlygos bus tokios pačios, t.y.:

- 100 Virtualių vartotojų, pradedant nuo trijų ir kas 8 sekundes prisijungiančių dar 3, kol galutinis vartotojų skaičius esančių sistemoje vienu metu pasiekia šimtą;
- Visi vartotojai atlieka vieną ir tą patį scenarijų;
- Visi vartotojai kartoja tą patį scenarijų šimtą kartų tam, kad būtų galima remtis ne pavieniu rezultatu, o laiko, tenkančio vienai transakcijai atlikti, mediana arba vidurkiu;
- Baigę darbą sistemoje vartotojai atsijungia.

Kiekvieno vartotojo atliekama veiksmų seka yra vienoda. Šią seką sudaro verslo procesai. Kiekvienas verslo procesas yra nedalomas; šiuo atveju kiekvienas verslo procesas yra transakcija.

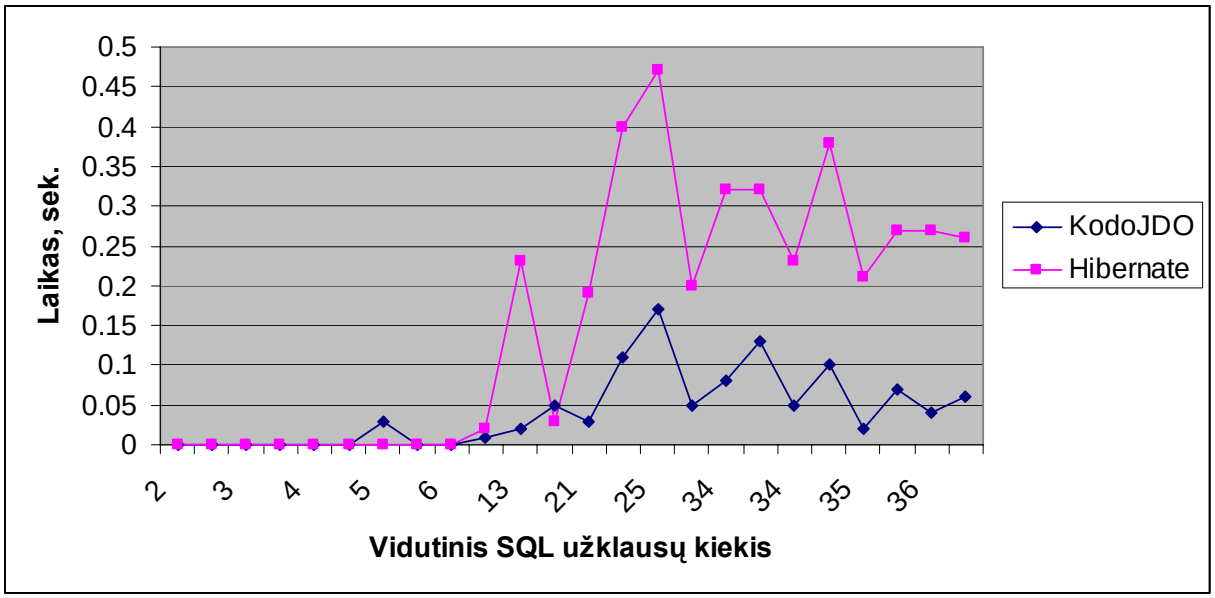
5 lentelėje pavaizduoti transakcijų vykdymo laikai abiem atvejais, vidutinis SQL užklausų kiekis kiekvienos transakcijos metu. Reikia pastebėti, kad atmetama apie 10% transakcijų, labiausiai nutolusių nuo vidurkio. Taip daroma todėl, kad yra daugybė faktorių ir veiksnių, nepriklausančių nuo mūsų nagrinėjamos sistemos bei galinčių lemti sistemos veikimo pagreitėjimą ar sulėtėjimą tam tikrais laiko momentais, pvz., sistemos resursų trūkumas, operatyviosios atminties atlaisvinimas kitoms programoms baigus darbą. Norint gauti kuo tikslesnius rezultatus į tai neatsižvelgti yra netikslinga. Kai kurių transakcijų vykdymo laikas lentelėje yra lygus nuliui. Taip yra todėl, kad transakcija įvyko per trumpesnį negu 0.01 sekundė laiko tarpą, ir „Load Runner“ nespėjo jos užfiksuoti.

5 lent. Objektų saugojimo karkasų testavimo transakcijų laikai, sek.

Transakcija	KodoJDO 90% transakcijų laiko, ms	Hibernate 90% transakcijų laiko, ms	Vidutinis SQL užklausų kiekis
Login	0.08	0.32	34
Search Account	0	0	3
Create Customer	0.05	0.2	26
Create Account	0	0	4

Find Default Broker	0.03	0.19	21
Add Employee	0.02	0.21	35
Create set Contacts	0.01	0.02	8
Find Set Group	0.02	0.23	13
Associate Account Customer	0.07	0.27	35
Save Account	0.13	0.32	34
Create Quote	0.05	0.03	15
Quote Applicant	0.04	0.27	36
Quote Insured Employment	0.05	0.23	34
Quote Location Info	0	0	4
Quote Location Info Update	0	0	4
Quote Location Info Coverages	0	0	3
Quote Location Info Credits	0	0	2
Quote Calculate Premium/Save	0.17	0.47	25
Policy Collect Data	0.03	0	5
Policy Generate Proposal	0.1	0.38	34
Policy Collect Change Data	0	0	6
Policy Bind Quotation	0.11	0.4	23
Policy Collect Issue Data	0	0	6
Policy Issue	0.06	0.26	45

Penktoje lentelėje pateikti duomenys rodo, kad sistema su KodoJDO veikia greičiau. Susumavus transakcijų laikus bei išvedus jų vidurkius gauname, kad sistema su KodoJDO veikia 296%, t.y. beveik tris kartus greičiau negu su Hibernate objektų saugojimo sluoksniu.

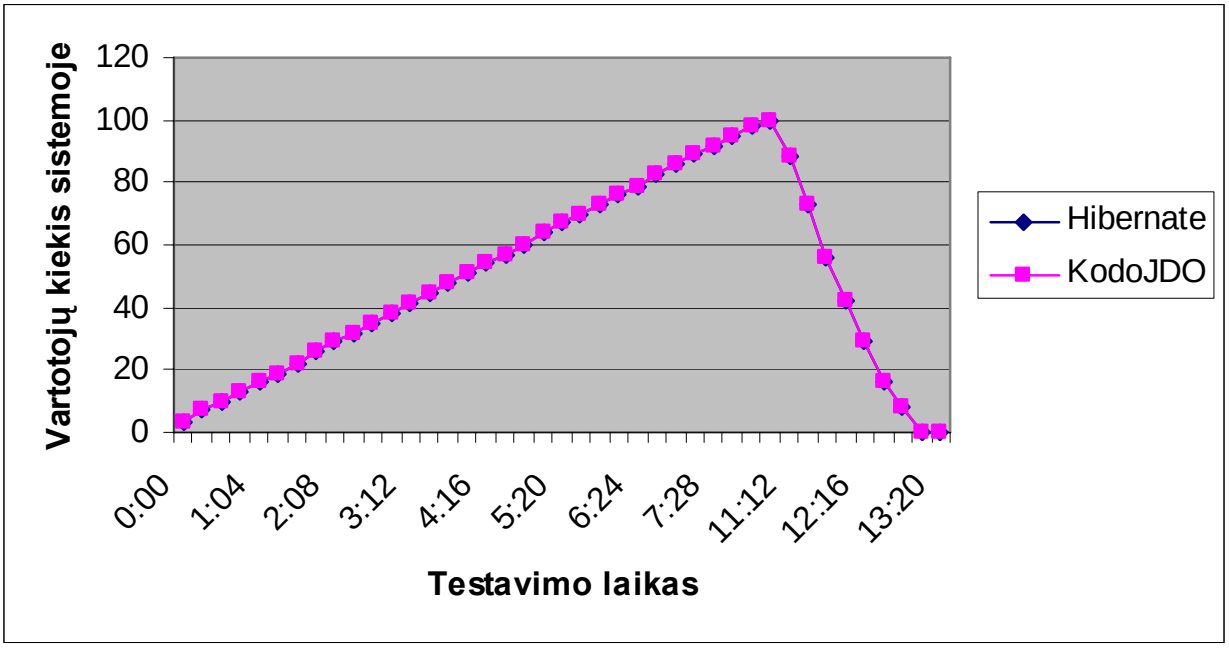


16 pav. Transakcijos laiko priklausomybė nuo SQL užklausų kiekio

17 paveiksle pateiktas transakcijos laiko priklausomybės grafikas nuo SQL užklausų kiekio. 17 paveiksle matyti, kad abiejų karkasų panaudojimo atvejais kuo ilgiau buvo atliekama transakcija KodoJDO atveju, tuo pačiu ir Hibernate ją atlikdavo per didesnį laiko tarpą. Tačiau priklausomybė tarp transakcijos vykdymo laiko ir SQL užklausų kiekio yra labai neapibrėžta. Matome, kad didėjant SQL užklausų kiekiui transakcijų laikas taip pat didėja, tačiau ši priklausomybė nėra tiesinė. Taip yra todėl, kad mūsų nagrinėjamos transakcijos yra ne sąryšinių duomenų bazių lygmenyje o objektų. Tai reiškia, kad transakcijas gali, tačiau nebūtinai sudaro sąryšinių duomenų bazių transakcijos. Daugiausiai laiko užimančiose transakcijose buvo atliekamas naujų objektų saugojimas į duomenų bazę.

4.3.5 Vartotojų kiekis sistemoje laiko atžvilgiu

Analizuojant rezultatus būtina įsitikinti, kad testavimo metu sąlygos buvo visiškai vienodos abiem atvejais. 16 paveiksle pateiktas vartotojų kiekio laiko atžvilgiu sistemoje grafikas. Matyti, kad grafikai yra persidengę, o tai reiškia, kad ši sąlyga abiejų karkasų testavimo atveju yra vienoda.



17 pav. Imituojamų vartotojų kiekis laiko atžvilgiu

4.3.6 Sistemos veikimo greičio tyrimo išvados

Tyrimo metu gauti transakcijų laikai yra tinkantys bet kokiai vidutinio sudėtingumo vidutiniam vartotojų skaičiui skirtai sistemai. Testavimo metu tą patį scenarijų atliko šimtas

virtualių vartotojų tuo pačiu metu. Abu objektų saugojimo karkasai veikė efektyviai, nes sudėtingiausia transakcija buvo atliekama tik 0.5 sekundės. Šiuo atveju apsisprendimas, kurį karkasą naudoti, priklauso nuo turimų lėšų bei kliento pageidavimų: KodoJDO yra komercinis produktas, Hibernate yra nemokamas.

5 IŠVADOS

1. Kuriant šiuolaikines verslo sistemas viena svarbiausių problemų yra objektų saugojimas sąryšinėse duomenų bazėse.
2. Nustatyta ir aprašyta tipinė šiuolaikinėse verslo sistemose naudojama sluoksninė architektūra ir objektų saugojimo karkasas, kaip viena svarbiausių jos dalių, naudojamų sistemos bendravimui su sąryšinėmis duomenų bazėmis.
3. Remiantis literatūros šaltiniais ir objektų saugojimo karkasų vartotojo bei programuotojo vadovais, išskirtos ir palygintos pagrindinės šiuolaikiniams objektų saugojimo karkasams būdingos savybės. Atlikus objektų saugojimo karkasų savybių palyginimą nustatyta, kad pagal taikytus kriterijus geriausi iš tirtų produktų yra KodoJDO ir Hibernate.
4. Nustatyta, kad yra keletas objektų saugojimo karkasams skirtų standartų, o karkasas yra tuo universalesnis, kuo daugiau šių standartų jis palaiko.
5. Nustatyta, kad papildomo sluoksnio įvedimas bei karkaso standartų palaikymas leidžia sistemose panaudoti bet kurį pasirinktą sąsają palaikantį karkasą.
6. Eksperimentinėje dalyje atliktas vieno iš svarbiausių praktinių objektų saugojimo karkasų faktorių – greičio – tyrimas remiantis realia šiuolaikine verslo sistema. Įsitikinta, kad sistemos veikimo greitis daugiau priklauso nuo objektų lygio transakcijų sudėtingumo negu nuo SQL užklausų kiekio.

6 LITERATŪRA

1. Apache DB Project. OJB – Features. 2005 [žiūrėta 2005-10-11]. Prieiga per internetą: http://db.apache.org/ojb/features.html.
2. Baronas R. Duomenų bazių sistemos. Vilnius: TEV, 2002.
3. Bauer C., King G. Hibernate in Action. New York: Manning Publications, 2005.
4. Booch G. Object-Oriented Analysis and Design with Applications. Second Editon. New York: Addison-Wesley, 1993.
5. Cattell R.G.G., Barry D. ir kt. The Object Data Standart: ODMG 3.0. San Diego: Morgan Kaufman, 2000.
6. Cayenne. Cayenne – User Guide. 2005 [žiūrėta 2005-02-17]. Prieiga per internetą: http://objectstyle.org/cayenne/userguide/index.html.
7. Elliott J. Hibernate: A Developer's Notebook. New York: O'Reilly, 2004.
8. Fowler M., Kendall S. UML Distilled: Addison–Wesley, 1998.
9. Imhoff C., Galemmo N., Geiger G. J. Mastering Data Warehouse Design: Relational And Dimensional Techniques. New York: Wiley Publishing, 2003.
10. JDO Expert Group. Java Data Objects. Version 1.0.1. 2003 [žiūrėta 2004-10-10]. Prieiga per internetą: http://jcp.org/aboutJava/communityprocess/final/jsr012/index2.html.
11. Oracle. Oracle AS. TopLink. Feature Overview. 2005 [žiūrėta 2005-02-17]. Prieiga per internetą: http://www.oracle.com/technology/products/ias/toplink/technical/tl10g_fov.htm.
12. Silverston L. The Data Model Resource Book, Volume 1. New York: Wiley Computer Publishing, 2001.
13. Silverston L. The Data Model Resource Book, Volume 2. New York: Wiley Computer Publishing, 2001.
14. Smith D. A Brief History of TopLink. 2005 [žiūrėta 2005-02-26]. Prieiga per internetą: http://www.oracle.com/technology/tech/java/newsletter/articles/toplink/history_of_toplink.html
15. Solarmetric. SolarMetric Kodo™ JDO 3.1.5 Developers Guide. 2005 [žiūrėta 2005-02-16]. Prieiga per internetą: http://www.solarmetric.com/jdo/Documentation/3.1.5/docs/manual.pdf.
16. Szyperski C., Gruntz D., Murer. S. Component Software: Beyond Object-Oriented Programming. Second Edition. New York: Addison–Wesley, 2002.

7 SANTRUMPŲ ŽODYNAS

6 lent. Terminų ir santrumpų žodynas

Trumpinys	Angliškai	Paaiškinimas
ACID	Atomicity, Consistency, Isolation, Durability	Dalumas, vientisumas, izoliacija, trukmė – pagrindinės savybės kurias turi palaikyti transakcijų mechanizmas
CORBA	Common Object Request Broker Architecture	Architektūra, skirta skirtingomis kalbomis parašytų programinių modulių bendravimui
EJB	Enterprise Java Beans	Paskirstytoms sistemoms skirtų Java komponentų specifikacija
JDBC	Java Database Connectivity	Bendravimo su duomenų baze protokolas Java kalboje
JDO	Java Data Objects	Java specifikacija, nusakanti kaip turėtų būti saugomi objektai
MDA	Model Driven Architecture	Modeliu paremta architektūra
UML	Unified Modelling Language	Vieninga modeliavimo kalba
RMI	Remote Method Invocation	Standartas, nusakantis nuotolinį bendravimą tarp Java objektų
SQL	Structured Query Language	Struktūrizuota užklausų kalba
OQL	Object Query Language	Objektų užklausų kalba

1 PRIEDAS. KodoJDO sugeneruotų metaduomenų pavyzdys

```
<?xml version="1.0" encoding="UTF-8"?>
<jdo>
   <package name="com.exigen.ipb.infrastructure">
      <class objectid-class="AccountJDOID" name="AccountJDO" identity-
type="application">
         <extension vendor-name="kodo" value="base" key="jdbc-class-map">
            <extension vendor-name="kodo" value="ACCOUNTJDOX" key="table"/>
         </extension>
         <extension vendor-name="kodo" value="version-number" key="jdbc-version-ind">
            <extension vendor-name="kodo" value="JDOLOCKX" key="column"/>
         </extension>
         <extension vendor-name="kodo" value="in-class-name" key="jdbc-class-ind">
            <extension vendor-name="kodo" value="JDOCLASSX" key="column"/>
         </extension>
         <extension key="data-cache-timeout" value="3600" vendor-name="kodo"/>
         <field name="id" primary-key="true">
            <extension vendor-name="kodo" value="value" key="jdbc-field-map">
               <extension vendor-name="kodo" value="IDX" key="column"/>
            </extension>
         </field>
         <field name="specialHandling">
            <extension vendor-name="kodo" value="value" key="jdbc-field-map">
               <extension vendor-name="kodo"
                  value="SPECIALHANDLINGX" key="column"/>
            </extension>
         </field>
         <field name="confidential">
            <extension vendor-name="kodo" value="value" key="jdbc-field-map">
               <extension vendor-name="kodo" value="CONFIDENTIALX" key="column"/>
            </extension>
         </field>
         <field name="name">
            <extension vendor-name="kodo" value="value" key="jdbc-field-map">
```

```
            <extension vendor-name="kodo" value="NAMEX" key="column"/>
          </extension>
        </field>
        <field name="creationDate">
          <extension vendor-name="kodo" value="value" key="jdbc-field-map">
            <extension vendor-name="kodo" value="CREATIONDATEX"
key="column"/>
          </extension>
        </field>
        <field name="premium">
          <extension vendor-name="kodo" value="value" key="jdbc-field-map">
            <extension vendor-name="kodo" value="PREMIUMX" key="column"/>
          </extension>
        </field>
        <field name="defaultBroker_fk">
          <extension vendor-name="kodo" value="value" key="jdbc-field-map">
            <extension vendor-name="kodo"
              value="DEFAULTBROKER_FKX" key="column"/>
          </extension>
        </field>
        <field name="customers">
          <collection element-type="com.exigen.ipb.infrastructure.CustomerJDO"/>
          <extension vendor-name="kodo" value="many-many" key="jdbc-field-map">
            <extension vendor-name="kodo" value="ID_CUSTOMERSX" key="element-
column.IDX"/>
            <extension vendor-name="kodo" value="ID_JDOIDX" key="ref-
column.IDX"/>
            <extension vendor-name="kodo"
              value="ACCOUNTJDO_CUSTOMERSX" key="table"/>
          </extension>
        </field>
        <field name="customerRelationships">
          <collection element-
type="com.exigen.ipb.infrastructure.CustomerRelationshipJDO"/>
          <extension vendor-name="kodo" value="many-many" key="jdbc-field-map">
```

```
            <extension vendor-name="kodo"
               value="ID_CUSTOMERRELATIONSHIPSX" key="element-
column.IDX"/>
            <extension vendor-name="kodo" value="ID_JDOIDX" key="ref-
column.IDX"/>
            <extension vendor-name="kodo"
               value="ACCOUNTJDO_CUSTOMERRELATIONSHX" key="table"/>
         </extension>
      </field>
      <field name="designatedCompanyContacts">
         <collection element-type="com.exigen.ipb.infrastructure.CompanyContactJDO"/>
         <extension vendor-name="kodo" value="many-many" key="jdbc-field-map">
            <extension vendor-name="kodo"
               value="ID_DESIGNATEDCOMPANYCONTACTSX" key="element-
column.IDX"/>
            <extension vendor-name="kodo" value="ID_JDOIDX" key="ref-
column.IDX"/>
            <extension vendor-name="kodo"
               value="ACCOUNTJDO_DESIGNATEDCOMPANYCX" key="table"/>
         </extension>
      </field>
      <field name="designatedCompanyAgents">
         <collection element-type="com.exigen.ipb.infrastructure.CompanyAgentJDO"/>
         <extension vendor-name="kodo" value="many-many" key="jdbc-field-map">
            <extension vendor-name="kodo"
               value="ID_DESIGNATEDCOMPANYAGENTSX" key="element-
column.IDX"/>
            <extension vendor-name="kodo" value="ID_JDOIDX" key="ref-
column.IDX"/>
            <extension vendor-name="kodo"
               value="ACCOUNTJDO_DESIGNATEDCOMPANYAX" key="table"/>
         </extension>
      </field>
      <field name="externalRelationships">
```

```
            <collection element-
type="com.exigen.ipb.infrastructure.ExternalRelationshipJDO"/>
            <extension vendor-name="kodo" value="many-many" key="jdbc-field-map">
               <extension vendor-name="kodo"
                  value="ID_EXTERNALRELATIONSHIPSX" key="element-column.IDX"/>
               <extension vendor-name="kodo" value="ID_JDOIDX" key="ref-
column.IDX"/>
               <extension vendor-name="kodo"
                  value="ACCOUNTJDO_EXTERNALRELATIONSHX" key="table"/>
            </extension>
         </field>
         <field name="designatedBrokerContacts">
            <collection element-type="com.exigen.ipb.infrastructure.BrokerContactJDO"/>
            <extension vendor-name="kodo" value="many-many" key="jdbc-field-map">
               <extension vendor-name="kodo"
                  value="ID_DESIGNATEDBROKERCONTACTSX" key="element-
column.IDX"/>
               <extension vendor-name="kodo" value="ID_JDOIDX" key="ref-
column.IDX"/>
               <extension vendor-name="kodo"
                  value="ACCOUNTJDO_DESIGNATEDBROKERCOX" key="table"/>
            </extension>
         </field>
         <field name="groups">
            <collection element-type="com.exigen.ipb.infrastructure.GroupJDO"/>
            <extension vendor-name="kodo" value="many-many" key="jdbc-field-map">
               <extension vendor-name="kodo" value="ID_GROUPSX" key="element-
column.IDX"/>
               <extension vendor-name="kodo" value="ID_JDOIDX" key="ref-
column.IDX"/>
               <extension vendor-name="kodo" value="ACCOUNTJDO_GROUPSX"
key="table"/>    </extension>  </field>
      </class>    </package>  </jdo>
```

2 PRIEDAS. Hibernate sugeneruotų metaduomenų pavyzdys

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping package="com.exigen.ipb.infrastructure" default-cascade="save-
update">
<class name="AccountHibernate" discriminator-value="Account.ipb.exigen.com"
optimistic-lock="version" table="ACCOUNT">
<composite-id name="id__" class="AccountHibernateID" access="field">
<key-property name="id" column="ID" access="field"/>
</composite-id>
<discriminator column="class"/>
<version name="version__" column="version__" access="field"/>
<property name="specialHandling" column="SPECIALHANDLING" access="field"/>
<property name="confidential" column="CONFIDENTIAL" access="field"/>
<property name="name" column="NAME0" access="field"/>
<property name="creationDate" column="CREATIONDATE" access="field"/>
<property name="premium" column="PREMIUM" access="field"/>
<property name="defaultBroker_fk" column="DEFAULTBROKER_FK" access="field"/>
<bag name="designatedCompanyContacts" lazy="true" access="field" inverse="true">
<key column="ACCOUNT_ID"/>
<one-to-many class="com.exigen.ipb.infrastructure.CompanyContactHibernate"/>
</bag>
<bag name="groups" outer-join="true" access="field">
<key column="GROUPS_ID"/>
<one-to-many class="com.exigen.ipb.infrastructure.GroupHibernate"/>
</bag>
<bag name="customerRelationships" lazy="true" access="field" inverse="true">
<key column="ACCOUNT_ID"/>
<one-to-many class="com.exigen.ipb.infrastructure.CustomerRelationshipHibernate"/>
</bag>
<bag name="customers" lazy="true" access="field" inverse="true">
<key column="ACCOUNT_ID"/>
```

```
<one-to-many class="com.exigen.ipb.infrastructure.CustomerHibernate"/>
</bag>
<bag name="externalRelationships" lazy="true" access="field" inverse="true">
<key column="ACCOUNT_ID"/>
<one-to-many class="com.exigen.ipb.infrastructure.ExternalRelationshipHibernate"/>
</bag>
<bag name="designatedCompanyAgents" outer-join="true" access="field">
<key column="DESIGNATEDCOMPANYAGENTS_ID"/>
<one-to-many class="com.exigen.ipb.infrastructure.CompanyAgentHibernate"/>
</bag>
<bag name="designatedBrokerContacts" lazy="true" access="field" inverse="true">
<key column="ACCOUNT_ID"/>
<one-to-many class="com.exigen.ipb.infrastructure.BrokerContactHibernate"/>
</bag>
</class>
</hibernate-mapping>
```