VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Objektinių ir reliacinių duomenų bazių taikymas objektiškai orientuoto programavimo kontekste

Application of object and relational databases in context of object oriented programming

Magistro baigiamasis darbas

Atliko:	Piotr Raginia	(parašas)
Darbo vadovas:	Balys Šulmanas	(parašas)
Recenzentas:	Karolis Petrauskas	(parašas)

Vilnius – 2008

Turinys

Santrauka

Šiame magistro baigiamajame darbe lyginamos objektinių ir reliacinių duomenų bazių taikymo galimybės objektiškai orientuoto programavimo kontekste. Duomenų bazių ir objektiškai orientuoto programavimo naudojimas iškelia dvi pagrindines problemas. Pirma problema yra reliacinių duomenų bazių ir objektiškai orientuotų programavimo kalbų nesuderinamumas. Antra – duomenų bazių naudojimas judriųjų programų sistemų kūrimo metodikų aplinkoje.

Norint palyginti reliacinių ir objektinių duomenų bazių taikymo galimybes buvo nuspręsta sudaryti metodų ir įrankių, skirtų iškeltoms problemoms spręsti, vertinimo modelį. Kadangi iškeltos problemos buvo per stambios vertinimo modelio sudarymui, jos buvo detalizuotos. Vertinami įrankiai ir metodai buvo suskirstyti į keturias kategorijas pagal prigimtį. Iš kiekvienos grupės buvo vertinamas vienas įrankis ar metodas. Darbo pabaigoje visi gauti rezultatai agreguoti ir pateiktos objektinių ir reliacinių duomenų bazių taikymo rekomendacijos, išskirti privalumai ir trūkumai.

Beveik pagal visus vertinimo kriterijus objektinės duomenų bazės įvertintos maksimaliais balais. Tačiau iš vertinimo rezultatų paaiškėjo, kad ir jos turi tam tikrų taikymo apribojimų - tokių kaip lėtas duomenų išrinkimas esant dideliam transakcijų skaičiui. Automatinio atvaizdavimo įrankiai prastai tinka projektams, naudojantiems judriąsias programų sistemų kūrimo metodikas, ir sunkiai pritaikomi sudėtingoms hierarchinėms struktūroms atvaizduoti. Reliacinių koncepcijų integravimas į tarpinius objektus tinka tik labai paprastoms objektų schemoms atvaizduoti. Reliacinių koncepcijų integravimo į objektiškai orientuotas programavimo kalbas kategorijoje vertintas Linq yra dar vystomas projektas, tačiau ateityje gali tapti geriausia alternatyva.

Raktiniai žodžiai: objektinės duomenų bazės, RDB ir OOP nesuderinamumas, judriosios programų sistemų metodikos.

Summary

This MA paper aims at object and relational databases application comparison in the context of object oriented programming. Joint usage of databases and object oriented programming raises two problems relevant today. First problem is impedance mismatch of relational databases and object oriented programming languages. Second – database usage in agile software development process environment.

In order to define application area of object and relational databases it was decided to compose assessment model to compare solutions available today. Since chosen problems were too bulky they were detailed. The evaluated methods and tools were categorized into four groups by their nature. One tool or method from each group was evaluated. In the end of this MA paper all results were aggregated into object and relational databases usage guidelines and databases usage pros and cons.

Object databases got the best evaluations in almost all defined criteria. However, evaluation also showed that object databases have some restrictions in application, such as bad performance when using multiple transactions. Object-relational mappers are not suitable for storing complex hierarchical class structures and usage with agile software development processes. Relational concepts integration into intermediate objects can be used only for very simple object schemes. Linq, evaluated in relational concepts integration with object oriented programming languages category, is still in development, but in future it can be the best choice.

Keywords: object databases, RDB and OOP impedance mismatch, agile software development process.

Įvadas

Nuo 1960-ųjų metų, kai atsirado pirmoji objektiškai orientuota programavimo kalba Simula, iki šiandienos objektiškai orientuoto programavimo paradigma populiarėjo. Jau 1990 metais dauguma programavimo kalbų palaikė objektiškai orientuoto programavimo paradigmą, o programuotojai žinojo tokias sąvokas kaip objektas, paveldėjimas, polimorfizmas, informacijos slėpimas (inkapsuliacija) ar abstrakcija. Šiandien objektiškai orientuotas programavimas yra de-facto standartas. De-facto standartu šiandien laikomos ir reliacinės duomenų bazės, kurių koncepcija buvo pasiūlyta 1970-ais metais E. F. Coddo ir iš esmės iki dabar nepasikeitė [Ter04] [Bar02]. Tačiau objektiškai orientuoto programavimo ir reliacinių duomenų bazių pasauliai blogai suderinami: imperatyvus programavimas prieš deklaratyvias užklausas, kompiliatoriaus optimizavimas ir užklausų optimizavimas, null nuorodos prieš null reikšmes, rodančias duomenų nebuvimą, gijos prieš transakcijas, algoritmai ir duomenų struktūros prieš sąryšius tarp duomenų ir indeksavimą [CIb05].

Objektiškai orientuoto programavimo ir reliacinių duomenų bazių nesuderinamumo problema sprendžiama nuo pat šių koncepcijų atsiradimo. Tačiau iki šiol šis klausimas lieka atviras, nors buvo padaryta didelė pažanga ir pasiūlyta daug įvairių technikų: specializuotos duomenų bazių programavimo kalbos, transakcijų modeliai, duomenų prieigos bibliotekos ir vis labiau populiarėjantys objektiniai – reliaciniai atvaizdavimo įrankiai, kurie labiausiai priartina prie problemos sprendimo [CIb05]. Tačiau net naujausi objektinį-reliacinį atvaizdavimą realizuojantys produktai, tokie kaip atviro kodo Hibernate, TopLink (IBM) ar Kodo (BEA Systems), neišsprendžia kai kurių nesuderinamumo problemų. Kaip pavyzdį galima įvardinti paveldėjimo atvaizdavimą į reliacines duomenų bazes. Galimos trys skirtinos paveldėjimo atvaizdavimo į reliacines duomenų bazes strategijos [BEA06]:

1) visos klasės (paveldima ir paveldėtos) atvaizduojamos į vieną reliacinės duomenų bazės lentelę;
2) visos klasės (paveldima ir paveldėtos) atvaizduojamos į skirtingas lentelės taip, kad laukai nesikartotų, ir sukuriami išoriniai raktai iš paveldėtų lentelių į paveldimą;
3) visos klasės (paveldima ir paveldėtos) atvaizduojamos į skirtingas lenteles panaudojant visus reikalingus atskirai klasei laukus.

Visos strategijos turi trūkumų. Nei viena iš jų netinka sudėtingos hierarchinės klasių struktūros saugojimui, kadangi arba duomenų išrinkimas bus lėtas (2), arba bus saugojama daug perteklinių duomenų (1,3)[BEA06]. Reikia pastebėti, kad perteklinių duomenų saugojimas gali sukelti duomenų anomalijas.

Dar viena problema, susijusi su objektiškai orientuotu programavimu ir reliacinių duomenų bazių naudojimu, yra evoliucinė daugumos šiuolaikinių programų sistemų kūrimo procesų prigimtis. Naudojant reliacines duomenų bazes, net ir nedideli kodo ar projekto pakeitimai reikalauja duomenų bazės schemos pakeitimų, kuriuos tenka daryti rašant SQL kalbos sakinius, kadangi dabar nėra automatinių reliacinių duomenų bazių pataisymo įrankių [Amb05]. Ypač tai aktualu judriosioms programų sistemų kūrimo metodikoms, tokioms kaip Ekstremalus Programavimas (Extreme Programming) arba Scrum, nes vienos iteracijos laikas tokiose metodikose yra ypatingai trumpas, vadinasi dažnai tenka keisti duomenų bazės schemą [Amb05]. Judriosios programų sistemų kūrimo metodikos neatsiejamos nuo tokių pasikartojančių veiklų kaip kodo perdarymas (refactoring), detalesnis sistemos perprojektavimas, regresyvus testavimas, testinių aplinkų kūrimas ir sudėtingas konfigūracijų valdymas. Reliacinės duomenų bazės yra labai statiškos ir sunkiai pritaikomos prie šių veiklų [Amb05]. Klasės lauko vardo pakeitimas gali reikalauti, pavyzdžiui, klasės diagramos pakeitimų, priklausomybių-ryšių diagramos pakeitimo, objektinio-reliacinio atvaizdavimo įrankių konfigūravimo, duomenų bazės schemos pakeitimo (visa tai galbūt visose testavimo aplinkose) ir visų pakeitimų sudėjimo į konfigūracijų valdymo sistemas.

Vienas iš galimų šių problemų sprendimų yra objektinės duomenų bazės (ODB), kurių taikymo nagrinėjimas šioms problemoms spręsti ir yra šio darbo vienas iš pagrindinių tikslų. Objektinių duomenų bazių idėja atsirado 1980-ųjų metų pradžioje, bet neišpopuliarėjo, nes programų sistemų kūrimo industrijoje jau spėjo įsitvirtinti geresnį matematinį pagrindą turinčios reliacinės duomenų bazės. Trūko su objektinėmis duomenų bazėmis mokančių dirbti specialistų, taip pat rinkoje buvo daug dagiau įrankių, skirtų reliacinėms duomenų bazėms. Nuo 2004-ųjų metų objektinių duomenų bazių populiarumas pradėjo augti, atsiradus atviro kodo objektinėms duomenų bazėms, kurios buvo parašytos objektiškai orientuotomis kalbomis, dažniausiai C++, java ir c#. Tokios atviro kodo objektinės duomenų bazės kaip db4o (db4objects) ir Perst (McObject) sparčiai išpopuliarėjo dėl savo lengvo panaudojamumo ir prieinamumo. Ilgą laiką objektinių duomenų bazių populiarumą stabdė standartų stoka. ODMG (Object Data Management Group) konsorciumas, kuris ruošė objektinių duomenų bazių ir objektinių-reliacinių atvaizdavimo įrankių standartus, iširo 2001-iais metais. 2006-ųjų vasarį OMG (Object Management Group) anonsavo, kad perduoda ODMG 3.0 (paskutinį ODMG išleistą standartą) vystymą ODBT WG (Object Database Technology Working Group), kuri jau parengė būsimo standarto RFI (Request For Information). Standartas turėtų apjungti naujoves objektinių duomenų bazių srityje, tokias kaip replikavimas, orientuotas indeksavimas ir XML palaikymas [Odb06].

Šiandien objektinės duomenų bazės populiarėja ir randa vis daugiau pritaikymų. Daugelis pasaulinio mąsto kompanijų tokių kaip Boeing, BMW, Bosh, Seagate, Intel naudoja savo veikloje ODB [Dbo08]. Įdomus faktas, kad didžiausia šiuolaikinė duomenų bazė yra būtent objektinė ir jos dydis viršija 1000 TB (Stanford Linear Accelerator Center). Nors objektinių duomenų bazių

instaliacijų skaičius didėja santykinai greičiau negu reliacinių, reiktų pastebėti, kad duomenų bazių sektoriuje vis dar didžiąją dalį rinkos užima reliacinės duomenų bazės [Odb06].

Tokia situacija susidarė ne tik dėl jau minėtų istoriškai susiklosčiusių politinių priežasčių, bet ir dėl aibės techninių objektinių duomenų bazių taikymo apribojimų, tokių kaip sudėtingas užklausų optimizavimas ar ribotos transakcijų mechanizmų palaikymas [Bar07]. Šiuolaikinės objektinės duomenų bazės stipriai patobulėjo ir gali būti taikomos daug platesniam uždavinių ratui, nors ir neišvengia tam tikrų apribojimų.

Šio darbo pagrindinis tikslas yra reliacinių ir objektinių duomenų bazių taikymo galimybių palyginimas objektiškai orientuoto programavimo kontekste, t.y. sprendžiant duomenų bazių ir objektiškai orientuotų programavimo kalbų nesuderinamumo problemas, bei duomenų bazių ir judriųjų programų sistemų kūrimo metodikų naudojimo problemas. Laukiami darbo rezultatai yra iškeltų problemų detalizavimas, vertinimo modelio sudarymas, skirtingų įrankių palyginimas pagal sudarytą vertinimo modelį ir taikymo rekomendacijų pateikimas.

1. Analizė

Norint tinkamai apibrėžti duomenų bazių taikymo galimybės iškeltoms problemoms spręsti, pirmiausia reikalinga atlikti detalesnę iškeltų problemų analizę. Apžvelgsime su kokiais sunkumais susiduriama ir kokie metodai ir praktikos taikomi šiandien iškeltų problemų sprendimui. Taip pat sugrupuokime galimus sprendimo būdus. Analizės metu gauta medžiagą panaudosime alternatyvių iškeltų problemų sprendimo būdų ir objektinių duomenų bazių palyginimui. Gautas vertinimo rezultatas taps pagrindu duomenų bazių taikymo galimybėms apibrėžti, t.y. galėsime:

- išskirti objektinių ir reliacinių duomenų bazių naudojimo privalumus ir trūkumus;
- pateikti rekomendacijas objektinių ir reliacinių duomenų bazių naudojimui.

1.1. DB ir OOP kalbų nesuderinamumas

Šiame skyriuje panagrinėsime DB (duomenų bazių) ir OOP (objektiškai-orientuoto programavimo) kalbų nesuderinamumo problemą detaliau. Apžvelgsime svarbiausius nesuderinamumo aspektus ir jų atsiradimo priežastis.

1.1.1 Modelių atvaizdavimas

OOP ir RDB modeliai yra skirtingi, todėl norint naudoti juos kartu tenka daryti vienokį ar kitokį atvaizdavimą tarp šių dviejų modelių. Atvaizdavimas gali būti daromas tiek automatiškai, naudojant specialius įrankius, tokius kaip Hbernate, TopLink (IBM) ar Kodo (BEA Systems), tiek rankiniu būdu, naudojant bendravimo tarp OOP kalbos ir RDB bibliotekas, tokias kaip JDBC, ADO.NET ir kt. Žinomi ir tarpiniai atvaizdavimo būdai, pavyzdžiui, ActiveRecord. [New06] [BEA06] [CIb05][Hib07] [ARe07] [Map06] Šiame skyrelyje aptarsime bendras atvaizdavimo problemas.

Kaip RDB, taip ir OOP kalbos turi paprastų tipų ir duomenų struktūrų palaikymą. Dažniausiai paprasti tipai ir duomenų struktūros RDB nesutampa su OOP kalbos tipais, pavyzdžiui, SQL-92 standartas nenustato absoliutaus skaitinių tipų dydžių. Operacijos su paprastais duomenų tipais taip pat gali būti skirtingos, pavyzdžiui, simbolių eilučių palyginimo operacijos. [CIb05] Kitas svarbus primityvių tipų atvaizdavimo aspektas yra null nuorodos. Dažniausiai OOP kalbose primityvus tipai, tokie kaip int negali turėti null reikšmės, nors RDB tai leidžia. Kai kurios OOP kalbos turi specialius tipus null reikšmei atvaizduoti, pavyzdžiui, C# yra System.DBNull tipas. Tačiau paprastų tipų nesuderinamumas yra palyginus nedidelė problema atvaizduojant šiuos du modelius vienas į kitą. [BEA06] [New06] [Mis06] [Map06] Panagrinėkime atvaizdavimą detaliau.

Paprastai atvaizdavimas atliekamas taip:

1) esybių-ryšių (angl. *entity-relationship*) modelio atributai, kurie reprezentuojami paprastais tipais ir duomenų struktūromis, atvaizduojami į objektų laukus;
2) esybių-ryšių modelio ryšiai atvaizduojami į nuorodas tarp objektų. Jeigu sąryšis yra daugiareikšmis, jis atvaizduojamas į nuorodų kolekciją.

Objektiniame modelyje ryšiai tarp objektų realizuojami visiškai skirtingai nei esybių-ryšių modelyje. Nors abiejų modelių ryšiai yra vienakrypčiai, tačiau priešingos krypties. Taip atvaizduojant paprastą 1:N ryšį objektiniame modelyje, objektas, į kurį yra nuoroda, nežino apie egzistuojantį ryšį, o esybių-ryšių modelyje yra atvirkščiai - išorinis raktas rodo į kitos lentelės įrašą. Taigi, norint, pavyzdžiui, išrinkti visus su vienu objektu susietus objektus, reikės formuoti laiko atžvilgiu neracionalią dviejų lentelių apjungimo (angl. *JOIN*) užklausą. Situacija dar labiau komplikuojasi kai reikia atvaizduoti N:M objektinio modelio ryšį į esybių-ryšių modelio ryšį. Tokiu atveju reikės kurti papildomą lentelę ir visų objekto sąryšių išrinkimui reikės apjungti jau tris lenteles.

Dar viena atvaizdavimo problema, su kuria susiduriama realizuojant atvaizdavimą, yra paveldėjimas (angl. *inheritance*). [New06] [BEA06] [Mis06] Galimos tris paveldėjimo atvaizdavimo strategijos [Map06]:

1) visos klasės (paveldima ir paveldėtos) atvaizduojamos į skirtingas lentelės panaudojant visus reikalingus atskirai klasei laukus;
2) visos klasės (paveldima ir paveldėtos) atvaizduojamos į skirtingas lentelės taip, kad laukai nesikartotų, ir sukuriami išoriniai raktai iš paveldėtų lentelių į paveldimą;
3) visos klasės (paveldima ir paveldėtos) atvaizduojamos į vieną reliacinės duomenų bazės lentelę.

Pirma strategija yra viena iš paprasčiausiai suvokiamų, tačiau veikianti lėčiausiai, ypač jei paveldėjimo medis yra didelis. Pavyzdžiui, norint išrinkti visus bendriausio tipo objektus, reikės peržiūrėti ir apjungti visas lenteles, kur lentelių skaičius lygus paveldėjimo medžio dydžiui. Dėl pirmos strategijos lėtumo, dažniausiai naudojamos kitos dvi, tačiau ir jos turi didelių trūkumų. Antra strategija denormalizuoja lenteles ir saugomi pertekliniai duomenys. Trečiai strategijai būdingas didžiausiais perteklinių duomenų kiekis, taip pat dauguma laukų turės null reikšmės.

Visos šiame skyrelyje aptartos problemos yra gerai žinomos ir visuotinai pripažintos, tačiau dėl metodų saugojimo iki šiol nėra vieningos nuomonės. [CIb05] [Bla05] Iš vienos pusės elgsenos ir būsenos integravimas yra vienas iš pagrindinių OOP principų, tačiau atvaizdavimas, kuris nesaugo elgsenos (metodų), pažeidžia šį principą. Iš kitos pusės būsenos ir elgsenos atskyrimas yra viena iš geruųjų praktikų, naudojamų projektuojant duomenims imlias sistemas.[Bla05]

Bendru atveju OOP kalbos klasių modelį praktiškai neįmanoma atvaizduoti į RDB esybių-ryšių modelį.

1.1.1.1 Objektinės duomenų bazės

Kadangi naudojant objektinę duomenų bazę lieka vienintelis objektinis modelis, visiškai nelieka modelių atvaizdavimo problemos. Dauguma šiuolaikinių duomenų bazių geba išgauti informaciją apie saugomą objektą per atspindžio (angl. *reflection*) mechanizmus [New07] [Bar07] [Gre06], todėl, norint išsaugoti objektą, tereikia iškviesti saugojimo metodą. Bet kokie klasių pakeitimai lengvai atsispindi objektinėje duomenų bazėje (pavyzdžiui, naujo klasės atributo pridėjimas, ar pašalinimas [Gre06]).

Toks veikimo principas taip pat supaprastina kodo perdarymus (angl. *refactoring*), kas ypač svarbu naudojant judriąsias programų sistemų kūrimo metodikas, kadangi labai dažni kodo perdarymai yra vienas iš labiausiai būdingų tokioms metodikoms bruožų.

Iš kitos pusės tarp objektinės duomenų bazės ir ją naudojančių taikomųjų programų stipriai padidėja sujungimas (angl. *coupling*), todėl bet kokie objektinės duomenų bazės schemos pakeitimai reikalauja kodo pakeitimų.

1.1.2 Schemos priklausomybė

Kalbėdami apie modelių atvaizdavimą neatsižvelgėme į tai, jog kūrimo procese OOP klasių modelis dažnai keičiasi, todėl atitinkamai dažnai reikia keisti RDB schemą. Tačiau praktikoje programų sistemos kūrėjų komanda dažnai negali tiesiogiai valdyti ir keisti schemos, kurios valdymo teisės priklauso kitai grupei – RDB administratoriams. Toks įtakos sferų paskirstymas yra naudingas, nes administratoriai gali netrukdomi rūpintis schemos optimizavimo ir saugumo klausimais. Norėdami padidinti greitaveiką, administratoriai neretai gali denormalizuoti ar kitaip keisti schemą. Bet kuriuo atveju tai nėra patogu kūrėjams, kuriems tenka nuolat keisti OOP modelį. Pradėdami naują projektą kūrėjai turi suprojektuoti schemą ir pateikti RDB administratoriams, tačiau vėliau atvaizduojant OOP klasių modelio pasikeitimus tenka prašyti administratorių atitinkamai pakoreguoti schemą.

Iš esmės ši problema yra ne tiek techninė, kiek politinė, tačiau didesniuose projektuose su ja susiduriama pakankamai dažnai. [New06]

1.1.2.1 Objektinės duomenų bazės

Kadangi objektinės duomenų bazės schema yra objektinis klasių modelis, jos priklausomybės problema tampa visiškai neaktuali.

1.1.3 Schemos sinchronizavimas

Kita svarbi atvaizdavimo problema yra dvigubas RDB schemos saugojimas. Vienas schemos egzempliorius saugomas tiesiogiai (tai gali būti esybių-ryšių diagrama, ar specialus įrankis, leidžiantis saugoti schemą, pavyzdžiui, Oracle Designer, kuris saugo schemą tiesiogiai. DB [Ora07]). Kitas schemos egzempliorius saugomas netiesiogiai OOP klasių modelyje, jei naudojame automatizuotus atvaizdavimo įrankius, arba net užklausų pavidale, jei naudojame tiesiogines užklausas (angl. *explicit query*). [CIb05] [New06] Taigi bet kokie pakeitimai viename modelyje atitinkamai reikalauja pakeitimų kitame. Ši schemos sinchronizavimo problema kažkiek susijusi su ankstesniame skyrelyje aptarta, tačiau yra labiau techninio pobūdžio.

Praktikoje dažniausiai išeities kodas yra priderinamas prie RDB schemos pakeitimų, o ne atvirkščiai, kadangi jau veikiančios schemos pakeitimas dažnai reikalauja duomenų migravimo arba adaptacijos. RDB dažnai naudoja kelios taikomosios programos, todėl dėl vienos programos klasių modelio pasikeitimo keisti DB schemą yra pernelyg brangu. [New06] Situacija dar labiau komplikuojasi jei naudojamas ortogonalus objektų saugojimas (angl. *orthogonal persistance*), t.y. objektas išsaugomas tarp atskirų programų vykdymų, pavyzdžiui, naudojant XML serializavimą. Tokiu atveju nepakanka pakeisti vien klasių modelį, reikia atitinkamai atnaujinti visus serializuotus objektus. [CIb06] Didėjant projekto apimčiai kūrėjai bus priversti atsisakyti DB schemos pakeitimų ir mažinti dviejų modelių – objektinio ir reliacinio – sukibimą [New06]

1.1.3.1 Objektinės duomenų bazės

Kadangi objektinės duomenų bazės naudojasi tuo pačiu objektiniu modeliu kaip ir jas naudojančios taikomosios programos, visiškai nelieka schemos sinchronizavimo problemos.

1.1.4 Objektų identitetas

Kita gerai žinoma OOP kalbų ir RDB nesuderinamumo problema yra objektų identitetas. Iš esmės objekto identitetą nusako nuoroda į jį, o ne pats objektas, t.y. mes galime turėti identiškus objektus, bet jeigu jų nuorodos skiriasi, tai bus du skirtingi objektai. Tokiose programavimo kalbose kaip C# ar Java operacijos „==“ ir „x.equals(y)“ turi visiškai skirtingą prasmę. Reliacinėse duomenų bazėse identitetas nusakomas ne nuoroda, o raktu, t.y. eilutės turiniu. Taigi mes galime turėti du skirtingus objektus, bet tą pačią eilutę. [Hib07] [New06] Toks identiteto skirtumas reikalauja papildomo dėmesio naudojant lygiagretų programavimą. Kadangi dauguma šiuolaikinių automatinių atvaizdavimo įrankių naudoja tarpinį duomenų saugojimą (angl. *caching*), tai identiteto problema tampa dar aktualesnė, nes reikia sekti tarpinių duomenų pasikeitimą ir laiku juos atnaujinti. Tai įmanoma tik tuo atveju, jei duomenų bazė gali signalizuoti apie duomenų pasikeitimą. Toks signalizavimas dažniausiai yra nepalaikomas ir šiam funkcionalumui užtikrinti reikalingas papildomas programavimas, pavyzdžiui, naudojant oracle advanced queueing technologija .[Aka01]

1.1.4.1 Objektinės duomenų bazės

Tam tikra objektų identiteto problema išlieka ir naudojant objektines duomenų bazes, nors formuluojama šiek tiek paprasčiau. Iš tikrųjų, dalis problemos atkrenta vien todėl, kad nereikia rūpintis pirminiais raktais. Objektinės duomenų bazės automatiškai suteikia unikalų identifikatorių (OID, iš angl. *Object ID*) kiekvienam objektui [Gre06]. OOP kalbos dažniausiai identifikuoja objektus pagal atminties adresą. Taigi reikia tikrinti, ar tam tikras objektas yra priskirtas kokiam nors objektinės duomenų bazės objektui. Tokiam patikrinimui dažnai yra specialus statinis metodas [New07].

1.1.5 Duomenų išrinkimas

Išrenkant duomenis susiduriama su tam tikromis atvaizdavimo problemomis. Panagrinėkime jas detaliau.

1.1.5.1 Išrinkimo būdai

Geriausia, kai visi objektai yra konstruojami naudojant specialiai tam skirtus metodus – konstruktorius, tačiau užklausų atveju tai būtų pernelyg neracionalu laiko atžvilgiu. Dažnai prireikia išrinkti ne vieną objektą, o duomenų kolekciją pagal tam tikrą kriterijų. Tai atlikti tik su konstruktorių pagalba praktiškai neįmanoma. Objektų konstravimas iš reliacinės duomenų bazės saugomos esybės dažniausiai vyksta pagal vieną iš trijų scenarijų [New06][CIb05]:

1. užklausa pagal pavyzdį;
2. užklausoms naudojama programavimo sąsaja;
3. užklausoms naudojama specifinė kalba.

Pirmas būdas yra paprasčiausias: išrinkimo metodui perduodamas tam tikras pavyzdinis objektas (užklausos šablonas), kurio laukai dalinai ar pilnai užpildomi reikiamais paieškos kriterijais [New06]:

```
Asmuo a = new Asmuo(); // visi laukai pagal nutylėjimą įgyja null reikšmę
a.Pavarde = "Smith";
ObjectCollection oc = QueryExecutor.execute(a);
```

Toks išrinkimas, žinoma, tinka tik paprasčiausioms užklausoms, kadangi negalima suformuluoti sudėtingesnių užklausų, kurias nepadengia užklausos šablonas. Tai pat neaišku kaip interpretuoti tuos objekto laukus, kurie negali įgyti reikšmės null.

Antras būdas naudoja specialius užklausos objektus, kurie gali būti konstruojami, naudojant specialią sintaksę ir perduodami išrinkimo metodams [New06]:

```
Query q = new Query();
q.From("Asmuo").Where(
new EqualsCriteria("Asmuo.Pavarde", "Smith"));
ObjectCollection oc = QueryExecutor.execute(q);
```

Šis metodas yra kur kas lankstesnis, tačiau ir jis neleidžia sukonstruoti sudėtingesnių užklausų, kaip lentelių apjungimas. Dar viena šiam metodui būdinga problema yra tai, kad neįmanoma atlikti statinio tipų patikrinimo. [New06] [CIb05] Statiškai negalima patikrinti ar užklausos parametras yra reikiamo tipo, ir ar egzistuoja lentelėje „Asmuo“ stulpelis „Pavarde“. Taigi naudojant tokį išrinkimo būdą prisirišama prie RDB schemos ir net menkiausi pakeitimai reikalauja visų užklausų pakeitimo. Galimas ir hibridinis pirmų dviejų išrinkimo būdų variantas, kai pasiliekamos statinio patikrinimo galimybės ir jos papildomos užklausų sudarymo lankstumu [New06]:

```
Query q = new Query();
Field lastNameFieldFromPerson = Asmuo.class.getDeclaredField("Pavarde");
q.From(Asmuo.class).Where(new EqualsCriteria(lastNameFieldFromPerson, "Smith"));
ObjectCollection oc = QueryExecutor.execute(q);
```

Tačiau vis dar negalima atlikti sudėtingesnių užklausų kelioms esybėms.
Trečias išrinkimo būdas yra specifinės užklausų kalbos naudojimas, tokios kaip OQL ar HQL. Ši specifinė užklausų kalba turėtų būti labiau tinkama objektų išrinkimui nei SQL ir palaikyti visas sudėtingas užklausų konstrukcijas. Pirma, šioms kalbo būdingas stiprus prisirišimas prie schemos. Antra, dažniausiai šios kalbos būna tik SQL kalbos tam tikras supaprastinimas ir negali prilygti SQL galingumui, tuo labiau, kad šios specifinės kalbos dažniausiai negali naudoti objektams būdingų savybių, tokių kaip polimorfizmas, išrinkimui. Todėl tokia užklausa yra praktiškai neįmanoma [New06] [CIb05]:

```
SELECT Asmuo p1, Asmuo p2
FROM Asmuo
WHERE p1.sutuoktinis() == null
AND p2.sutuoktinis() == null
AND p1.tinkamasSutuoktinis(p2)
AND p2.tinkamasSutuoktinis(p1);
```

Čia metodas „tinkamasSutuoktinis“ yra skirtingas kiekvienam objektui „Asmuo“. Taigi prarandamas SQL lankstumas ir neįgyjami OOP būdingi privalumai.

1.1.5.2 Dalinis duomenų išrinkimas

Aptardami išrinkimo būdus neišnagrinėjome dar vienos svarbios duomenų išrinkimo problemos – dalinio duomenų išrinkimo.

Yra gerai žinoma, kad lentelių apjungimas yra viena iš brangiausių laiko atžvilgiu operacijų, atliekamų vykdant SQL užklausas. Todėl natūralu, kad stengiamasi išvengti tokio tipo užklausų. Tačiau naudojant atvaizdavimo įrankius dažnai sugeneruojamos neefektyvios kelių lentelių apjungimo užklausos [Oba01] [New06]. Ankstesniuose skyreliuose aptarėme paveldėjimo atvaizdavimo problemas, kurios parodo, kaip paveldėtų objektų išrinkimo užklausos gali būti transformuotos į lėtas daugelio lentelių apjungimo užklausas. Su panašiomis problemomis susiduriama ir tuomet, kai reikia išrinkti tik dalį lentelės duomenų. Nors dauguma šiuolaikinių automatinio atvaizdavimo įrankių leidžia nurodyti dalinių duomenų išrinkimo strategiją [Hib07] [Map06], tačiau toks pasirinkimas yra konfigūruojamas statiškai, todėl praktiškai neįmanoma išvengti situacijų, kai panaudojama strategija priešinga reikiamai. Pavyzdžiui, lentelė „Asmuo“ turi labai daug stulpelių, o mes norime išrinkti visus objektus „Asmuo“ ir mus domina tik stulpelis „vardas“. Automatinis atvaizdavimo įrankis gali elgtis trimis skirtingais būdais [Hib07][New06]:

1. iš karto išrinkti visus objekto stulpelius, taip stipriai padidindamas užklausos vykdymo laiką;
2. išrinkti tik stulpelį „vardas“, paliekant kitus objekto „Asmuo“ laukus lygius null, kas gali prieštarauti modeliui;
3. išrinkti tik stulpelį „vardas“, paliekant kitus objekto „Asmuo“ laukus atidėto išrinkimo būsenoje, t.y. prireikus sužinoti lauko reikšmę, bus įvykdyta papildoma užklausa.

Kad ir kokią strategiją pasirinktume, galime surasti situaciją, kai pasirinktoji strategija bus netinkama. Nors primityvių tipų laukams tai palyginus nedidelės vykdymo laiko sąnaudos, tačiau, jei tarp objekto „Asmuo“ laukų yra kitos objektų kolekcijos, pavyzdžiui, visi asmens pažįstami asmenys, tai tampa rimta problema.

1.1.5.3 Objektinės duomenų bazės

Objektinėms duomenų bazėms būdingos aptartos duomenų išrinkimo problemos. Jeigu su SQL užklausos pagalba mes galime išrinkti bet kokį duomenų kiekį, tai objektiniuose duomenų bazėse mažiausias duomenų kiekis yra vienas objektas, kas kartais būna ne taip efektyvu. [New07] [Gre06] Tačiau objektinėms duomenų bazėms prieš kiekvieną užklausą galima nurodyti susijusių objektų

išrinkimo gylį. Taip, jeigu mes nurodysime vienetinį gylį, bus ištrauktas tik pats objektas, o visos jo nuorodos bus lygios null reikšmei. Toks veikimo principas užtikrina tinkamos išrinkimo strategijos pasirinkimą prieš kiekvieną užklausą, skirtingai nei automatiniai reliaciniai/objektiniai atvaizdavimo įrankiai, kurie naudoja tik vieną pasirinktą strategiją visoms konkrečios klasės objektų užklausoms.

Kaip ir automatiniai objektiniai/reliaciniai atvaizdavimo įrankiai, objektinės duomenų bazės naudoja tris pagrindinius duomenų išrinkimo būdus [New06][CIb05][Gre06]:

1. užklausa pagal pavyzdį;
2. užklausoms naudojama programavimo sąsaja;
3. užklausoms naudojama specifinė kalba.

Dažniausiai duomenų išrinkimui naudojamos specifinės užklausų kalbos, tačiau skirtingai nei atvaizdavimo įrankiai, visos užklausos gali būti patikrintos naudojant statišką tipizavimą, kadangi naudojamas vienas objektinis modelis, kuris gali būti išgautas iš kodo tokiomis priemonėmis kaip atspindžio technologija [Gre06] [New07].

1.2. Judriosios PS kūrimo metodikos

Praktiškai visos judriųjų PS kūrimo metodikų ir duomenų bazių naudojimo problemos susiveda į jau aptartą schemos sinchronizavimo problemą, kuri komplikuojasi dėl to, kad kompiliatoriai negali statiškai patikrinti tipų ir SQL užklausų korektiškumo. Visos užklausų klaidos gali būti rastos tik vykdymo metu. Panagrinėkime kelias būdingiausias judriųjų metodikų veiklas [Ext08] [Ris07].

1.2.1 Kodo perdarymai

Dažni kodo perdarymai yra labai būdingi judriosioms PS kūrimo metodikoms. Tai savo ruožtu gali pareikalauti RDB schemos pakeitimo, taigi iš esmės turime schemos sinchronizavimo problemą [Amb05]. Dažni RDB schemos pakeitimai iškelia užklausų automatizuoto perdarymo problemą [Coo06]. Iš tikrųjų jau parašytos SQL užklausos po schemos pakeitimo reikalauja atitinkamo koregavimo. Tačiau dauguma šiuolaikinių programavimo aplinkų negali automatiškai pakeisti užklausų, pavyzdžiui, lentelių stulpelių vardų, nors ir suteikia galimybę tai daryti su klasių laukais ir metodais. Taigi, sinchronizuojant objektinės ir reliacinės schemas, programuotojui tenka rūpintis ir SQL užklausų sinchronizavimų su RDB schemos pasikeitimais. Kadangi kompiliatorius negali patikrinti tipų ir užklausų korektiškumo toks sinchronizavimas tampa dar sudėtingesnis, o klaidos išgaudomos tik vykdymo metu.

1.2.2 Regresyvus testavimas

Naudojant regresyvius testus (dažniausiai automatinius) naudinga turėti statinį išrenkamų duomenų tipų tikrinimą, idealiu atveju kompiliatorius turėtų mokėti tikrinti SQL užklausų korektiškumą. Tačiau dėl objektinio i reliacinio modelio atskyrimo toks patikrinimas yra praktiškai

neįmanomas, dar didesne problema tampa sudėtingos RDB schemos ir vidinių procedūrų regresyvaus testavimo galimybės [Amb05][Bar07].

1.2.3 Sudėtingas konfigūracijos valdymas

Maži laikotarpiai tarp iteracijų ir išleidimų, net kas savaitę, pastovus reikalavimų pataisymas ir papildymas, skirtingų versijų palaikymas yra neatsiejama judriųjų metodikų dalis [Hib07][New06]. Papildomos pastangos skirtos RDB schemos versijų saugojimui gali kainuoti daug papildomų resursų.

1.2.4 Testinės aplinkos

Testinių aplinkų palaikymas turint kelias RDB schemas ir su jomis dirbantį kodą gali sukelti ypatingai daug sinchronizavimo problemų. Paprasčiausiu atveju, kai turimą testinę aplinką reikia perkelti į naudojimą, neužtenka vien pakeisti programinės įrangos nauja versija, tenka papildomai pernešti testavimo RDB schemą. Toks pernešimas dažnai būna sudėtingas, praktiškai visada tenka stabdyti RDB naudojimą, kas ne visada yra priimtina, ypač kalbant apie sistemas su aukštu pasiekiamumo reikalavimu.

1.3. Sprendimo būdai

Dauguma literatūros šaltinių pateikia viena ar kelis iš šių nesuderinamumo problemos sprendimo būdų [CIb05] [New06] [Oba01][New07] [Bar07]:

1. Atvaizdavimas rankiniu būdu.
2. Atvaizdavimas panaudojant automatizuotus atvaizdavimo įrankius.
3. Reliacinių koncepcijų integravimas į objektines kalbas.
4. Reliacinių koncepcijų integravimas į tarpinius objektus.
5. Objektinių koncepcijų integravimas į duomenų bazes.

Kai kuriuose darbuose, pavyzdžiui [CIb05], bandoma lyginti atskirus nesuderinamumo sprendimo įrankius pasitelkiant į pagalbą aibę skirtingų kriterijų, tokių kaip ortogonalaus objektų saugojimo galimybė, transakcijų buvimas, tiesioginių užklausų vykdymo galimybė, skirtingos optimizavimo galimybės, tačiau kriterijų skalė neatspindi iškeltos problemos išsprendimo laipsnio. Mums svarbiau išskirti, kaip išvardinti metodai gali išspręsti nesuderinamumo problemą, o ne palyginti metodų funkcionalumą. Toliau panagrinėsime kiekvieną išvardintą būdą detaliau.

1.3.1 Atvaizdavimas rankiniu būdu

Šis nesuderinamumo sprendimo būdas vis labiau užleidžia vietą kitiems, kadangi reikalauja didelių pastangų, nors yra lanksčiausias ir pateikia mažiausią objektinio ir reliacinio modelių sukibimą [New07]. Programuotojams tenka rašyti SQL užklausas ir pasinaudojant gautais duomenimis formuoti reikalingus objektus. Realizuojant atvaizdavimą rankiniu būdu, dažniausiai tenka konstruoti gerai išnagrinėtus ir daugelyje automatizuoto atvaizdavimo įrankių realizuotus atvaizdavimo mechanizmus, tokius kaip tarpinių duomenų saugojimas, paveldėjimo atvaizdavimas, išrinkimo strategijų pasirinkimas ir kt. Dažnai naudojamas projektuose, kuriuose iškyla šiame darbe aptarta schemos priklausomybės problema. Naudojant iškyla ir kitos aptartos atvaizdavimo problemos.

1.3.2 Atvaizdavimas panaudojant automatizuotus atvaizdavimo įrankius

Šiuo metu labai populiarus atvaizdavimo būdas. Modelių atvaizdavimas atliekamas pasinaudojant tokiais automatizuoto atvaizdavimo įrankiais kaip Hibernate, TopLink (IBM) ar Kodo (BEA Systems). Nors šie įrankiai ir pateikia didelę dalį reikiamo atvaizdavimo funkcionalumo, naudojant juos neišvengiama daugumos ankščiau aptartų problemų, o tai reikalauja nemenko darbuotojų apmokymo.

1.3.3 Reliacinių koncepcijų integravimas į objektines kalbas.

Nelabai paplitęs, tačiau šio metu sparčiai plėtojamas būdas [Lin08]. Programuotojai naudoja OOP programavimo kalbas, kurios palaiko reliacinių užklausų koncepciją. Šiuo metu didelį dėmesį pritraukė Microsoft LINQ projektas [Lin08]. Neišvengia daugumos aptartų atvaizdavimo problemų, nors dar pakankamai neištobulintas ir gali ateityje tapti viena iš geriausių alternatyvų [New07].

1.3.4 Reliacinių koncepcijų integravimas į tarpinius objektus.

Pakankamai paplitęs metodas: programuotojai naudoja specialius tarpinius objektus, kurie pateikia lentelės abstrakcijas. Tai Microsoft‘o DataSet, Ruby‘s ActiveRecord, Java‘s RowSet ir kt. Vėl gi neišvengiama daugumos aptartų problemų.

1.3.5 Objektinių koncepcijų integravimas į duomenų bazes.

Šiuo metu populiarėjantis geriausiai nesuderinamumo problemas išsprendžiantis būdas.[New07] [Oba01] [Bar07] Vietoje duomenų bazių reliacinių koncepcijų integravimo į objektines programavimo kalbas, integruoja objektines koncepcijas į objektinių duomenų bazių valdymo sistemas. Tokių objektinių duomenų bazių naudojimas, kaip db4object db4o, Versant Object Database, Objectiviy ObjectSpace, Ontos ir kt. minimizuoja iškeltas problemas.

2. Vertinimas

2.1. Vertinimo modelis

Detalizavus problemas galime sudaryti vertinimo modelį, kurį naudosime skirtingų įrankių palyginimui ir objektinių duomenų bazių taikymo galimybių apibrėžimui. Kiekvienas detalizuotas iškeltų problemų aspektas, išskyrus schemos priklausomybės problemą, kuri yra greičiau politinio nei techninio pobūdžio, bus naudojamas kaip vienas iš vertinimo modelio elementų :

- Schemos atvaizdavimas.
- Schemos sinchronizavimas.
- Objektų identitetas.
- Duomenų išrinkimas.
- Judriosios PS kūrimo metodikos.

Kiekvienas iš šių elementų bus suskaidytas į matuojamus kriterijus. Kriterijaus pavadinimas bus sudaromas naudojant elemento pavadinimo paskutinių dviejų žodžių didžiąsias raides ir numerį. Kriterijus gali įgyti reikšmes: 0 (prastai), 1 (vidutiniškai)i, 2(puikiai).

2.1.1 Schemos atvaizdavimas

Šioje kriterijų grupėje surinkti svarbiausi objektinio ir reliacinio pasaulio atvaizdavimo aspektai. Tipų atvaizdavimo kriterijus yra skirtas primityviems objektiškai orientuotų programavimo kalbų tipams ir duomenų struktūroms .

1 lentelė. Schemos atvaizdavimo vertinimo kriterijai

Identifikatorius	Kriterijus	Aprašas
SA1	Tipų atvaizdavimas	DB ir OOP kalbos tipai ir duomenų struktūros pilnai atvaizduojamos.
SA2	Null-nuorodų atvaizdavimas	DB ir OOP Null-nuorodos pilnai atvaizduojamos.
SA3	Objektų sąryšių atvaizdavimas	OOP objektų sąryšiai tokie kaip paveldėjimas ir nuorodos pilnai atvaizduojami į DB.

2.1.2 Schemos sinchronizavimas

Schemos sinchronizavimo problemos yra vienos iš svarbiausių kalbant apie judriąsias programų sistemų kūrimo metodikas, tačiau jos buvo iškeltos į atskira grupę, nes yra pakankamai dažnai sutinkamos kitokio pobūdžio projektuose.

2 lentelė. Schemos sinchronizavimo vertinimo kriterijai

Identifikatorius	Kriterijus	Aprašas
SS1	Schemų sinchronizavimas	Galimybė automatiškai atvaizduoti vienos schemos pakeitimus į kitą (OOP į DB).
SS2	Ortogonalus saugojimas	Galimybė išsaugoti objekto būseną.

2.1.3 Objektų identitetas

3 lentelė. Objektų identiteto vertinimo kriterijai

Identifikatorius	Kriterijus	Aprašas
OI1	Objektų identitetas	Galimybė atskirti duomenų bazėje saugomus objektus pagal nuorodą, o ne pirminį raktą.

2.1.4 Duomenų išrinkimas

4 lentelė. Duomenų išrinkimo vertinimo kriterijai

Identifikatorius	Kriterijus	Aprašas
DI1	Laukų išrinkimas	Galimybė išrinkti tik tam tikrus atskirus objektų laukus.
DI2	Užklausos pagal pavyzdį	Galimybė daryti užklausas pagal šablonų metodą.
DI3	Užklausos naudojant sąsają.	Galimybė daryti užklausas naudojant specialiai tam skirtą sąsają.
DI4	Užklausos naudojant specifinę kalbą.	Galimybė daryti užklausas naudojant specifinę kalbą.
DI5	Laisvas išrinkimas.	Galimybė išrinkti bet kokios kombinacijos skirtingų objektų laukus.
DI6	Statinis tipų tikrinimas.	Tipai gali būti patikrinti statiškai, t.y. prieš vykdymą.
DI7	Išrinkimo greitaveika.	Objektų išrinkimo greitaveika.
DI8	Išrinkimo optimizavimas.	Galimybė optimizuoti objektų išrinkimą.

2.1.5 Judriosios PS kūrimo metodikos

Šiai kriterijų grupei taip pat svarbi statinio tipų tikrinimo galimybė, tačiau toks kriterijus jau įtrauktas į duomenų išrinkimo kriterijų grupę, todėl čia jo nekartosime.

5 lentelė. Naudojimo su judriosiomis PS kūrimo metodikomis vertinimo kriterijai

Identifikatorius	Kriterijus	Aprašas
KM1	Schemos modifikavimas.	Duomenų bazės schemos modifikavimo sudėtingumas.
KM2	Užklausų perdarymas	Galimybė automatizuotai perdarinėti užklausas.
KM3	Schemos konfigūracijos valdymas	Duomenų schemos konfigūracijos valdymo proceso sudėtingumas.
KM4	Testinių aplinkų paruošimas	Testinių aplinkų paruošimo sudėtingumas.

2.2. Įrankių ir metodų grupės

Šiame darbe sudarytas vertinimo modelis yra įrankių palyginimo ir tinkamumo apibrėžtoms problemoms spręsti matas. Toliau parinksime ir įvertinsime pagal pasirinktą vertinimo modelį objektinę duomenų bazę ir po konkretų įrankį iš šiuo metu labiausiai paplitusių įrankių kategorijų [CIb05] [New06] [Oba01][New07] [Bar07].

1. Atvaizdavimas panaudojant automatizuotus atvaizdavimo įrankius. Modelių atvaizdavimas atliekamas pasinaudojant tokiais automatizuoto atvaizdavimo įrankiais kaip Hibernate, TopLink (IBM) ar Kodo (BEA Systems). Vertinime naudosime vieną iš populiariausių šiandien java programavimo kalbai skirtą Hibernate automatizuoto atvaizdavimo įrankį. Vertindami remsimės Hibernate 3.2.2 versija[Hib07].
2. Reliacinių koncepcijų integravimas į objektines kalbas. Nelabai paplitęs, tačiau šiuo metu sparčiai plėtojamas būdas [Lin08]. Programuotojai naudoja OOP programavimo kalbas, kurios palaiko reliacinių užklausų koncepciją. Šiuo metu didelį dėmesį pritraukė Microsoft LINQ projektas [Lin08]. Būtent šis įrankis ir taps vertinimo objektu šioje kategorijoje.
3. Reliacinių koncepcijų integravimas į tarpinius objektus. Pakankamai paplitęs metodas: programuotojai naudoja specialius tarpinius objektus, kurie pateikia lentelės abstrakcijas. Tai Microsoft‘o DataSet, Ruby‘s ActiveRecord, Java‘s RowSet ir kt. Vertinimui naudosime Microsoft DataSet klasę [Dat08].
4. Objektinių koncepcijų integravimas į duomenų bazes. Objektinių koncepcijų integravimas į duomenų bazių valdymo sistemas. Šio metu šis būdas vėl sulaukė daug

dėmesio tiek iš akademinės, tiek iš pramoninės bendruomenės Iš rinkoje populiariausių objektinių duomenų bazių - db4object db4o, Versant Object Database, Objectiviy ObjectSpace, Ontos ir kt. - vertinsime db4o objektinę duomenų bazę[Dbo08].

Atvaizdavimo rankiniu būdu neįtrauksime į vertinimų sąrašą, nes šis nesuderinamumo sprendimo būdas vis labiau užleidžia vietą kitiems, kadangi reikalauja didelių pastangų, nors yra lanksčiausias ir pateikia mažiausią objektinio ir reliacinio modelių sukibimą [New07]. Programuotojams tenka rašyti SQL užklausas ir remiantis gautais duomenimis formuoti reikalingus objektus. Realizuojant atvaizdavimą rankiniu būdu, dažniausiai tenka konstruoti gerai išnagrinėtus ir daugelyje automatizuoto atvaizdavimo įrankių realizuotus atvaizdavimo mechanizmus, tokius kaip tarpinių duomenų saugojimas, paveldėjimo atvaizdavimas, išrinkimo strategijų pasirinkimas.

Tolimesniuose skyreliuose pateikiami pasirinktų įrankių vertinimai.

2.3. Schemos atvaizdavimas

2.3.1 Tipų atvaizdavimas

2.3.1.1 Hibernate

Hibernate automatizuoto atvaizdavimo įrankis turi 11 pagrindinių įsiūtų tipų kategorijų. Šių kategorijų tipai atvaizduoja java kalbos tipus į standartinius duomenų bazių tipus [Hib07][Sql92]:

6 lentelė. Hibernate paprastų tipų atvaizdavimas.

Hibernate tipai	Java tipai	SQL tipai[1]
integer, long, short, float, double, character, byte, boolean, yes_no, true_false	java.lang.Character, java.lang.Byte, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean	INTEGER, CHARACTER, SMALLINT, FLOAT, REAL, DOUBLE
string	java.lang.String	VARCHAR
date, time, timestamp	java.util.Date	DATE, TIME, TIMESTAMP
calendar, calendar_date	java.util.Calendar	TIMESTAMP, DATE
big_decimal, big_integer	java.math.BigDecimal, java.math.BigInteger	NUMERIC

[1] Priklausomai nuo duomenų bazės tipai gali vadintis skirtingai. Iš esmės galimos visos duomenų bazės palaikančios JDBC.

locale, timezone, currency	java.util.Locale, java.utilTimeZone, java.util.Currency	VARCHAR
class	java.lang.Class	VARCHAR
binary	java.lang.Byte[]	BLOB
text	java.lang.String	CLOB,TEXT
serializable	java.io.Serializable	BLOB
clob, blob	java.sql.Blob, java.sql.CLOB	BLOB, CLOB

Hibernate yra numatyta vartotojo tipų atvaizdavimo apibrėžimo galimybė, todėl iš esmės visi java kalbos tipai gali būti atvaizduojami[Hib07]. Tiesa, toks atvaizdavimas reikalauja nemažai pastangų, kadangi reikės apibrėžti norimus tipus Hibernate suprantamu formatu ir, kaip ir standartinių tipų atveju, visą atvaizdavimą aprašyti specialiai suformatuotu XML dokumentu.

Kadangi visi tipai gali būti atvaizduoti, vertiname Hibernate tipų atvaizdavimą 2 (puikiai).

2.3.1.2 Linq

Kadangi Linq yra tik užklausų kalba, vertinti atvaizdavimo galimybes neprasminga.

2.3.1.3 DataSet

Kadangi DataSet naudoja reliacinio modelio objektinės abstrakcijos koncepciją, iš esmės nėra jokio atvaizdavimo iš objektinės schemos į reliacinę, todėl vertinti atvaizdavimo galimybes neprasminga.

2.3.1.4 Db4o

Db4o nereikalauja schemų atvaizdavimo, kadangi naudojama vienintelė objektinė klasių schema.

Vertiname db4o tipų atvaizdavimą 2 (puikiai).

2.3.2 Null-nuorodų atvaizdavimas

2.3.2.1 Hibernate

Visos išvardintos Hibernate tipų kategorijos palaiko java.lang.null semantiką. Tačiau kolekcijų atvaizdavimas yra ribotas ir java.lang.null semantikos nepalaiko, t.y. hibernate negali atskirti tuščios kolekcijos nuo null reikšmės. Dar vienas apribojimas yra tai, kad visi kolekcijos tipo laukai turi būti apibrėžiami tik kaip interfeisai [Hib07].

Šių trūkumų egzistavimas leidžia vertinti Hibernate null-nuorodų atvaizdavimą kaip 1(vidutiniškai).

2.3.2.2 Linq

Kadangi Linq yra tik užklausų kalba, vertinti atvaizdavimo galimybes neprasminga.

2.3.2.3 DataSet

Kadangi DataSet naudoja reliacinio modelio objektinės abstrakcijos koncepciją, iš esmės nėra jokio atvaizdavimo iš objektinės schemos į reliacinę, todėl vertinti atvaizdavimo galimybes neprasminga.

2.3.2.4 Db4o

Kaip ir tipų atvaizdavimo kriterijaus atveju, db4o nereikalauja jokio atvaizdavimo, nes naudoja vienintelę objektinę klasių schemą.

Vertiname db4o null-nuorodų atvaizdavimo galimybę 2 (puikiai).

2.3.3 Objektų sąryšių atvaizdavimas

2.3.3.1 Hibernate

Kaip ir dauguma automatinių atvaizdavimo įrankių, Hibernate palaiko tris pagrindines paveldėjimo atvaizdavimo strategijas[Hib07]:

1) visos klasės (paveldima ir paveldėtos) atvaizduojamos į vieną reliacinės duomenų bazės lentelę;
2) visos klasės (paveldima ir paveldėtos) atvaizduojamos į skirtingas lentelės taip, kad laukai nesikartotų, ir sukuriami išoriniai raktai iš paveldėtos lentelės į paveldimą;
3) visos klasės (paveldima ir paveldėtos) atvaizduojamos į skirtingas lenteles panaudojant visus reikalingus atskirai klasei laukus.

Hibernate kiekvienai iš šių strategijų numato aibę papildomų parinkčių. Vienas iš svarbesnių yra polimorfizmo tipo pasirinkimas: besąlygiškas (angl. „implicit polymorphism“) ar tikslus (angl. „explicit polymorphism“). Šios parinkties pagalba galima nurodyti, kokiu būdu bus išrenkami objektai. Pirmuoju atveju tėvinės klasės objektų išrinkimo užklausa išrenka ir visus paveldėtų klasių objektus. Antruoju atveju tėvinės klasės objektų išrinkimas išrenka tik tėvinės klasės objektus, bet ne paveldėtus (išskyrus klasės, apibrėžtas naudojant <subclass> ir <joinded-subclasss> tagus).

Šių strategijų ir parinkčių pagalba teoriškai galima atvaizduoti bet kokio sudėtingumo klasių hierarchiją, nors praktiškai dideles klasių hierarchijas labai sudėtinga atvaizduoti ir dar sudėtingiau atvaizdavimą valdyti, kadangi kiekviena iš išvardintų strategijų turi savo specifinius panaudojimo apribojimus ir atvaizdavimą sunku modifikuoti [Hib07][Map06][Cib05][Mis06] [Coo06].

Nuorodos iš vienų objektų į kitus (objektų asociacijos) Hibernate atvaizduojamos pasinaudojant vienu iš trijų būdų [Hib07]:

1) tiesiogiai, pasinaudojant išoriniu raktu;
2) pasinaudojant tarpine lentele;
3) naudojant HQL užklausą, kaip filtrą.

Visos asociacijos gali būti kaip vienakryptės (angl. „unidirectional“) arba dvikryptės (angl. „bidirectional“). Taip pat yra galimybė nurodyti asociacijų sąryšius. Hibernate palaiko keturis asociacijų sąryšio tipus - vienas-su-vienu, vienas-su-daug, daug-su-vienu ir daug-su-daug (angl. „one-to-one“, „one-to-many“, „many-to-one“, „many-to-many“).

Tokio rinkinio galimybių pakanka atvaizduoti objektų asociacijas, todėl bendras objektų sąryšių atvaizdavimo įvertis yra 2(puikiai).

2.3.3.2 Linq

Kadangi Linq yra tik užklausų kalba, vertinti atvaizdavimo galimybes neprasminga.

2.3.3.3 DataSet

Kadangi DataSet naudoja reliacinio modelio objektinės abstrakcijos koncepciją, iš esmės nėra jokio atvaizdavimo iš objektinės schemos į reliacinę, todėl vertinti atvaizdavimo galimybes neprasminga.

2.3.3.4 Db4o

Kaip ir tipų atvaizdavimo kriterijaus atveju, db4o nereikalauja jokio atvaizdavimo, nes naudoja vienintelę objektinę klasių schemą.

Vertiname db4o objektų sąryšių atvaizdavimo galimybę 2 (puikiai).

2.4. Schemos sinchronizavimas

2.4.1 Schemų sinchronizavimas

2.4.1.1 Hibernate

Pats Hibernate neturi jokių automatinių reliacinės ir objektinės schemų sinchronizavimo metodų, tačiau yra nemažai kitų gamintojų sukurtų įrankių, kurie leidžia sugeneruoti klasių apibrėžimus ir atvaizdavimą iš reliacinės schemos, taip pat sugeneruoti lentelių apibrėžimus ir atvaizdavimą iš klasių apibrėžimų [Hib07]. Aišku, automatinis generavimas neturi galimybės perdaryti jau turimas klases ir lentelės, jis leidžia tik iš naujo jas apibrėžti. Todėl negalima pakeisti lentelės stulpelių pavadinimų, tik sukurti naują lentelę.

Vertinsime Hibernate schemų sinchronizavimo galimybę 1(vidutiniškai).

2.4.1.2 Linq

Linq neturi jokių priemonių automatizuotam schemų sinchronizavimui. Vertiname Linq schemų sinchronizavimo galimybę 0 (prastai).

2.4.1.3 DataSet

Kaip ir Linq, DataSet neturi jokių priemonių automatizuotam schemų sinchronizavimui. Vertiname Linq automatizuoto schemų sinchronizavimo galimybę 0 (prastai).

2.4.1.4 Db4o

Kadangi db4o naudoja vienintelę objektinę klasių schemą, schemų sinchronizavimo problemą tampa neaktuali.

Vertiname db4o automatinio schemų sinchronizavimo galimybę 2(puikiai).

2.4.2 Ortogonalus saugojimas

2.4.2.1 Hibernate

Hibernate gali serializuoti objektus, palaikančius java.io.Serializable interfeisą, tačiau toks saugojimo būdas negali apsaugoti nuo klasės pasikeitimų, t.y. jeigu serializuotą objektą norėsim deserializuoti po to, kai buvo pakeistas vieno iš laukų pavadinimas, to padaryti nepavyks.

Vertiname Hibernate ortogonalaus saugojimo galimybę 1 (vidutiniškai).

2.4.2.2 Linq

Kadangi Linq yra tik užklausų kalba, ortogonalaus saugojimo galimybės vertinimas neprasmingas.

2.4.2.3 DataSet

Kadangi DataSet naudoja reliacinio modelio objektinės abstrakcijos koncepciją, DataSet neturi jokių priemonių išsaugoti konkrečių objektų būsenas.

Vertiname DataSet ortogonalaus saugojimo galimybę 0 (prastai).

2.4.2.4 Db4o

Db4o pilnai palaiko objektų ortogonalaus saugojimo galimybę, vertiname 2 (puikiai).

2.5. Objektų identitetas

2.5.1.1 Hibernate

Reliacinės duomenų bazės veikimo principas neleidžia Hibernate skirti duomenų bazėje saugomų objektų pagal nuorodą.

Vertiname Hibernate objektų identiteto atskyrimo galimybę 0 (prastai).

2.5.1.2 Linq

Linq turi tas pačias identinteto problemas kaip ir Hibernate, tačiau galimybė naudoti objektinės duomenų bazes suteikia Linq didesnį lankstumą.

Vertiname Linq objektų identiteto atskyrimo galimybę 1 (vidutiniškai).

2.5.1.3 DataSet

DataSet kaip ir kiti įrankiai, dirbantys reliacinių duomenų bazių pagrindu, neišvengia objektų identiteto problemos.

Vertiname DataSet objektų identiteto atskyrimo galimybę 0 (prastai).

2.5.1.4 Db4o

Db4o nebūdingos objektų identiteto problemos dėl kiekvienam saugomam objektui automatiškai generuojamų specialių unikalių identifikatorių, vadinamu OID (angl. „Object Identifier“). Tačiau dėl pirminio rakto idėjos atsisakymo reikia papildomai dėmesio skirti objektų, saugomų duomenų bazėje, keitimui, t.y. norint pakeist, o ne sukurti nauja objektą, iš pradžių tenka patikrinti, ar jis jau išsaugotas duomenų bazėje, ir paskui jį pasirinkti.

Vertiname db4o objektų identiteto atskyrimo galimybę 2 (puikiai).

2.6. Duomenų išrinkimas

2.6.1 Laukų išrinkimas

2.6.1.1 Hibernate

Hibernate galima nesunkiai pritaikyti daliniam atskirų klasės laukų išrinkimui, pavyzdžiui, panaudojant <lazy> semantiką. Toks nurodymas pasako Hibernate, kad laukas turi būti traukiamas tik tuomet, kai bandoma ji panaudoti.

Vertiname dalinio laukų išrinkimo galimybę 2(puikiai).

2.6.1.2 Linq

Linq neturi galimybės išrinkti atskirus klasės laukus, tačiau galima apibrėžti naują klasę su reikalingų laukų rinkiniu ir juos išrinkti.

Vertiname Linq atskirų klasės laukų išrinkimo galimybę 1 (vidutiniškai).

2.6.1.3 DataSet

DataSet nesunkiai gali išrinkti atskirus klasės laukus dėl galimybės išrinkti duomenis vykdant SQL užklausas.

Vertiname DataSet atskirų laukų išrinkimo galimybę 2 (puikiai).

2.6.1.4 Db4o

Db4o neturi galimybės išrinkti atskirus klasės laukus, kadangi mažiausias išrenkamas db4o duomenų vienetas yra objektas.

Vertiname db4o dalinio išrinkimo galimybę 0 (prastai).

2.6.2 Užklausos pagal pavyzdį

2.6.2.1 Hibernate

Hibernate neturi jokių specialių priemonių užklausai pagal pavyzdį atlikti. Nors užklausų pagal kriterijus (angl. „Criteria Queries“) mechanizmas suteikia praktiškai identišką funkcionalumą, jis negali būti priskirtas prie šios strategijos.

Vertiname Hibernate užklausų pagal pavyzdį galimybę 0 (prastai).

2.6.2.2 Linq

Linq neturi jokių priemonių užklausoms pagal pavyzdį atlikti. Vertiname Linq užklausų pagal pavyzdį galimybę 0 (prastai).

2.6.2.3 DataSet

Kaip ir Linq, DataSet neturi jokių priemonių užklausoms pagal pavyzdį atlikti. Vertiname DataSet užklausų pagal pavyzdį galimybę 0 (prastai).

2.6.2.4 Db4o

Db4o turi puikiai išvystytą užklausų pagal pavyzdį mechanizmą. Vertiname Db4o užklausų pagal pavyzdį galimybę 2 (puikiai).

2.6.3 Užklausos naudojant sąsają

2.6.3.1 Hibernate

Hibernate turi gerai ištobulintą užklausų, naudojant specialius objektus sąsają vadinamą užklausomis pagal kriterijus (angl. „Criteria Queries“).

Vertiname Hibernate užklausų naudojant sąsają galimybę 2 (puikiai).

2.6.3.2 Linq

Linq nepalaiko jokių užklausų naudojant sąsają galimybių. Vertiname Linq užklausų naudojant sąsają galimybę 0 (prastai).

2.6.3.3 DataSet

DataSet kaip ir Linq, nepalaiko galimybės daryti užklausas naudojant sąsają. Vertiname DataSet užklausų naudojant sąsają galimybę 0 (prastai).

2.6.3.4 Db4o

Db4o turi galimybę vykdyti užklausas per specialią sąsają, vadinamą SODA („Simple Object Database Access“).

Vertiname db4o užklausų naudojant sąsają galimybę 2 (puikiai).

2.6.4 Užklausos naudojant specifinę kalbą

2.6.4.1 Hibernate

Hibernate turi savo užklausų kalbą, vadinamą HQL („Hibernate Query Language“). Kalba savo struktūra labai panaši į SQL su objektiniais papildymais, tokiais kaip asociacijos, polimorfizmas, paveldėjimas.

Vertiname Hibernate užklausų naudojant specifinę kalbą galimybę 2 (puikiai).

2.6.4.2 Linq

Linq pilnai palaiko užklausas naudojant specifinę kalbą, kadangi Linq ir yra integruota į programavimo kalbą specifinė užklausų kalba („Language integrated query“).

Vertiname Linq užklausų naudojant specifinę kalbą galimybę 2 (puikiai).

2.6.4.3 DataSet

DataSet neturi specifinės užklausų kalbos. Vertiname DataSet užklausų naudojant specifinę kalbą galimybę 0 (prastai).

2.6.4.4 Db4o

Db4o turi į programavimo kalbą integruotą specifinę kalbą („Native queries“). Vertiname Db4o užklausų naudojant specifinę kalba galimybę 2 (puikiai).

2.6.5 Laisvas išrinkimas

2.6.5.1 Hibernate

Hibernate turi galimybę vykdyti SQL užklausas tiesiogiai, todėl galima išrinkti bet kokios kombinacijos skirtingų objektų laukus.

Vertiname laisvo išrinkimo galimybę 2 (puikiai).

2.6.5.2 Linq

Linq neturi jokių priemonių, skirtų išrinkti iš objektinės duomenų bazės bet kokios kombinacijos objektų laukus, tačiau puikiai susitvarko kai išrenkama iš reliacinės.

Vertiname Linq laisvo išrinkimo galimybę 1 (vidutiniškai).

2.6.5.3 DataSet

DataSet gali vykdyti standartines SQL užklausas ir tokiu būdu išrinkti bet kokių laukų kombinaciją.

Vertiname DataSet laisvo išrinkimo galimybę 2 (puikiai).

2.6.5.4 Db4o

Db4o neturi jokių specialių priemonių atskiriems objektų laukams išrinkti. Mažiausias išrinkimo vienetas yra objektas.

Vertiname Db4o laisvo išrinkimo galimybę 0 (prastai).

2.6.6 Statinis tipų tikrinimas

2.6.6.1 Hibernate

Hibernate nenumatyta jokių statinio tipų tikrinimo galimybių. Vertiname įrankį pagal statinio tipų tikrinimo kriterijų 0 (prastai).

2.6.6.2 Linq

Kadangi Linq yra integruota į objektinę programavimo kalbą, kompiliatorius gali patikrinti tipus ir užklausų sintaksę.

Vertiname Linq statinio tipų tikrinimo galimybę 2 (puikiai).

2.6.6.3 DataSet

Kaip ir Hibernate, DataSet neturi jokių statinio tipų tikrinimo priemonių. Vertiname DataSet statinio tipų tikrinimo galimybę 0 (prastai).

2.6.6.4 Db4o

Į objektinę kalbą integruotos db4o užklausos (angl. „native query“) gali būti tikrinamos kompiliavimo metu.

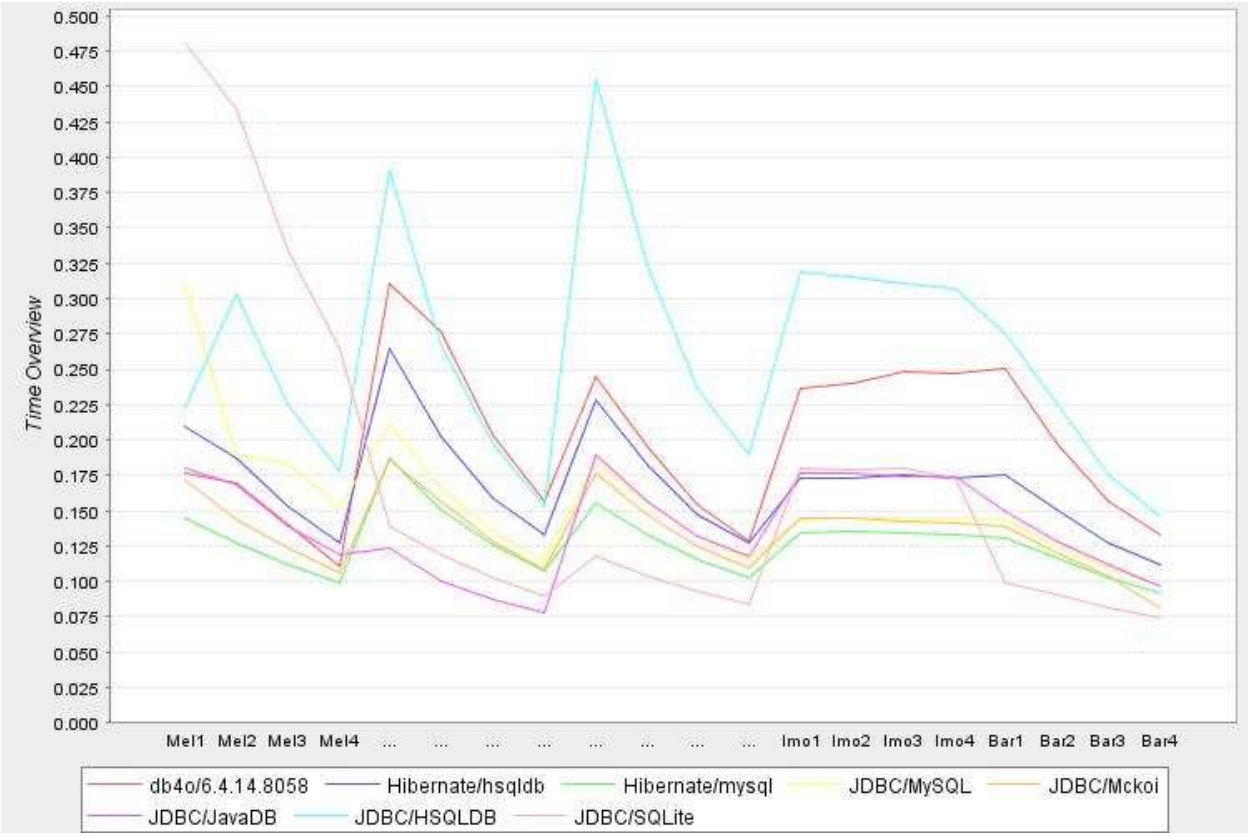
Vertiname db4o statinio tipų tikrinimo galimybę 2 (puikiai).

2.6.7 Išrinkimo greitaveika

Išrinkimo greitaveikos testai vienoje mašinoje neatskleidžia galimų išrinkimo greitaveikos elgesio modelių esant skirtingoms sąlygoms, todėl vertinant išrikimo greitaveika buvo remtasi

viešai prieinamais išrinkimo greitaveikos testais. Vienas iš žinomiausių yra PolePos [Pol08]. Pav. 1 pateikiama bendra išrinkimo greitaveikos PolePos palyginimo diagrama. Pirmame priede pateikiamas pilnas PolePos palyginimo metodų ir vertinimų aprašas.

7 lentelė. PolePos vertinimo santrauka[Pol08].



2.6.7.1 Hibernate

Hibernate išrinkimo greitaveika stipriai priklauso nuo daugelio įvairių sąlygų, tokių kaip išrenkamų objektų klasių hierarchijos sudėtingumas, transakcijų skaičius, išrinkimo strategijos ir pan. Tačiau didesnę dalį išrinkimo greitaveikos skirtumų tarp Hibernate ir kitų sprendimų sudaro duomenų bazės pasirinkimas. Hibernate atveju tai gali būti tik reliacinė duomenų bazė. Palyginus su objektinėmis duomenų bazėmis galima išskirti bendrą tendenciją: kuo paprastesni išrenkami duomenys ir kuo didesnis vienu metu vykdomų transakcijų skaičius, tuo greitesnė bus reliacinė duomenų bazė palyginus su objektine [Edl08][Kin07][Lin07][Lin08][Obj07][Pol08].

Vertiname Hibernate išrinkimo greitaveiką 1 (vidutiniškai).

2.6.7.2 Linq

Kaip ir Hibernate atveju, negalima vienareikšmiškai vertinti išrinkimo greitaveikos. Viskas priklauso nuo daugelio įvairių faktorių.

Vertiname Linq išrinkimo greitaveiką 1 (vidutiniškai).

2.6.7.3 DataSet

Iš esmės DataSet išrinkimo greitaveika yra šiek tiek greitesnė nei Hibernate, tačiau DataSet stipriai pralošia objektinėms duomenų bazėms, kai vienu metu vykdomų transakcijų skaičius yra santykinai mažas.

Vertiname DataSet išrinkimo greitaveiką 1 (vidutiniškai).

2.6.7.4 Db4o

Db4o išrinkimo greitaveika gali būti tiek daug didesnė nei kitų vertinamų įrankių, tiek ir daug mažesnė. Geriausius rezultatus db4o pasiekia, kai vienu metu išrenkama tik iš vienos gijos ar proceso. Tačiau kai vienu metu vykdomų transakcijų skaičius didėja, db4o tampa vis lėtesnė. Tokia situacija iškyla dėl db4o naudojamos optimistinės transakcijų vykdymo strategijos, t.y. kuo dažniau galima patekti į konfliktinę situaciją, tuo lėtesnė bus išrinkimo greitaveika [Edl08].

Vertiname db4o išrinkimo greitaveiką 1 (vidutiniškai).

2.6.8 Išrinkimo optimizavimas

2.6.8.1 Hibernate

Hibernate galima optimizuoti objektų išrinkimą naudojant skirtingas išrinkimo strategijas. Išrinkimo strategijos gali būti pasirinktos apibrėžiant atvaizdavimą, arba tiesiogiai nurodytos prie HQL ir prie užklausų pagal kriterijus. Iš viso yra 10 strategijų [Hib07]:

- išrinkimas panaudojant JOIN operatorių („join fetching“). Išrenkant objektą, objekto asociacijos išrenkamos tame pačiame SELECT sakinyje naudojant OUTER JOIN operatorių;
- išrinkimas panaudojant SELECT operatorių („select fetching“). Išrenkant objektą, užklaustos objekto asociacijos išrenkamos kitame SELECT sakinyje;
- pilnas išrinkimas panaudojant SELECT operatorių („subselect fetching“). Išrenkant objektą visos objekto asociacijos išrenkamos kitame SELECT sakinyje;
- paketinis išrinkimas („batch fetching“). Objektų išrinkimas naudojant unikalių raktų kolekciją;
- tiesioginis išrinkimas („immediate fetching“). Asociacija, kolekcija ar laukas išrenkami tuoj pat po to, kai užkraunamas tėvinis objektas;
- tingus kolekcijų išrinkimas („lazy collection fetching“). Kolekcija išrenkama tik prireikus;
- labai tingus kolekcijų išrinkimas („extra-lazy fetching“). Išrenkami tik reikalingi kolekcijos elementai, o ne visa kolekcija;
- išrinkimas su tarpininku („proxy fetching“). Vienos reikšmės asociacija išrenkama tik tuomet, kai iškviečiamas šią reikšmę naudojantis metodas;

- išrinkimas be tarpininko („no-proxy fetching“). Vienos reikšmės asociacija išrenkama kartu su objektu;
- tingus laukų išrinkimas („lazy atribute fetching“). Atributas ar vienos reikšmės asociacija išrenkami prireikus.

Pirmos keturios strategijos nusako, kada turi būti išrenkami objektai, kitos nusako, kaip jie turi būti išrenkami.

Nepaisant visų galimybių Hibernate negalima imperatyviai nurodyti, kokius indeksus naudoti išrinkimui, todėl vertiname išrinkimo optimizavimo galimybę 1(vidutiniškai).

2.6.8.2 Linq

Linq neturi jokių specialių priemonių užklausų optimizavimui. Kalbos semantika apsiriboja standartiniais JOIN, UNION, INTERSEC, EXCEPT, DISTINCT ir panašiais operatoriais, būdingais visoms užklausų kalboms.

Vertiname Linq užklausų optimizavimo galimybę 0 (prastai).

2.6.8.3 DataSet

Kadangi DataSet naudoja standartines SQL užklausas duomenų išrinkimui, tai suteikia plačias užklausų optimizavimo galimybes.

Vertiname DataSet užklausų optimizavimo galimybę 2 (puikiai).

2.6.8.4 Db4o

Dažniausiai db4o užklausos daromos naudojant integruotų užklausų mechanizmą (angl. „native query“), kurios vėliau pakeičiamos į SODA užklausas. Db4o taip pat suteikia galimybę tiesiogiai rašyti SODA užklausas, kurios yra pakankamai lanksčiai optimizuojamos.

Vertiname db4o užklausų optimizavimo galimybę 2 (puikiai).

2.7. Judriosios PS kūrimo metodikos

2.7.1 Schemos modifikavimas

2.7.1.1 Hibernate

Hibernate neturi specialių priemonių automatizuotam duomenų bazės schemos modifikavimui. Hibernate atvaizdavimo schemos dalis gali būti užrašyta objektinės programavimo kalbos priemonėmis (java annotations), bet didesnę dalį tenka rašyti specialiai suformatuotuose XML failuose. Nors Hibernate ir neturi specialių priemonių automatizuotam duomenų bazės schemos modifikavimui yra sukurta nemažai specialių įrankių, galinčių atlikti objektinės klasių diagramos transformaciją į reliacinį modelį [Hib07].

Vertiname Hibernate schemų modifikavimo sudėtingumą 1 (vidutiniškai).

2.7.1.2 Linq

Kaip ir Hibernate, Linq neturi jokių priemonių automatizuotam duomenų bazės schemos modifikavimui. Tačiau didesnę dalį atvaizdavimo galima užrašyti objektinės programavimo kalbos priemonėmis, tokioms kaip atributai.

Vertiname Linq duomenų bazės schemos modifikavimo sudėtingumą 1 (vidutiniškai).

2.7.1.3 DataSet

Kaip ir Hibernate, DataSet neturi jokių priemonių automatizuotam duomenų bazės schemos modifikavimui, tačiau visas atvaizdavimas gali būti užrašytas objektinės programavimo kalbos priemonėmis.

Vertiname DataSet duomenų bazės schemos modifikavimo sudėtingumą 1 (vidutiniškai).

2.7.1.4 Db4o

Kadangi db4o duomenų bazės schema padaroma pagal klasių aprašus, tai bet kokie objektinės schemos pakeitimai automatiškai keičia duomenų bazę, t.y. pridėjus nauja lauką prie klasės ir pabandžius išsaugoti tokios klasės objekto egzempliorių, prie visų saugomų bazėje tos klasės egzempliorių bus pridėtas naujas laukas.

Vertiname db4o automatinio duomenų schemos modifikavimo sudėtingumą 2 (puikiai).

2.7.2 Užklausų perdarymas

2.7.2.1 Hibernate

Hibernate turi galimybę automatizuotai generuoti ir atlikti perdarymus atvaizdavimo schemai, tačiau neturi jokių priemonių automatizuotam užklausų perdarymui.

Vertiname Hibernate užklausų perdarymo galimybę 0 (prastai).

2.7.2.2 Linq

Kadangi Linq yra integruota į objektinio programavimo kalbą, visos Linq užklausos gali būti statiškai tikrinamos ir perdaromos automatizuotai.

Vertiname Linq automatizuoto užklausų perdarymo galimybę 2 (puikiai).

2.7.2.3 DataSet

Kaip ir Hibernate, Microsoft DataSet neturi jokių priemonių automatizuotam užklausų perdarymui.

Vertiname DataSet automatizuoto užklausų perdarymo galimybę 0 (prastai)

2.7.2.4 Db4o

Db4o integruotos užklausos (angl. „native queries“) rašomos objektine kalba, todėl kompiliatorius gali tikrinti tipus ir sintaksę. Tai leidžia integruotoms programavimo aplinkoms, tokioms kaip Microsoft Visual Studio, automatizuotai perdarinėti užklausas, t.y. pakeitus klasės lauko vardą, gali būti automatiškai pakeistos visos šį lauką naudojančios užklausos.

Vertiname db4o automatizuoto užklausų perdarymo galimybę 2 (puikiai).

2.7.3 Schemos konfigūracijos valdymas

2.7.3.1 Hibernate

Hibernate neturi specialių priemonių konfigūracijos valdymui, tačiau praktiškai visoms reliacinėms duomenų bazėms yra sukurta nemažai įrankių, palengvinančių schemos konfigūracijos valdymą. Be objektinės ir reliacinės schemų, naudojant Hibernate, prireikia valdyti dar vieną schemą – atvaizdavimo. Taigi tokių projektų konfigūracijos valdymas yra labai sudėtingas.

Vertiname Hibernate konfigūracijos valdymo sudėtingumą 0 (prastai).

2.7.3.2 Linq

Naudojant Linq gali tekti valdyti objektinę ir reliacinę schemas. Tačiau skirtingai nei Hibernate, Linq nereikalauja jokių papildomų atvaizdavimo schemų.

Vertiname Linq konfigūracijos valdymo sudėtingumą 1(vidutiniškai).

2.7.3.3 DataSet

Kaip ir Linq atveju, DataSet naudojimas nereikalauja papildomų atvaizdavimo schemų kūrimo ir palaikymo.

Vertiname DataSet konfigūracijos valdymo sudėtingumą 1(vidutiniškai).

2.7.3.4 Db4o

Objektinė klasių schema yra vienintelė schema, kurios konfigūracija tenka rūpintis db4o, t.y. užtenka standartinių kodo konfigūracijos valdymo priemonių, tokių kaip cvs.

Vertiname db4o konfigūracijos valdymo sudėtingumą 2 (puikiai).

2.7.4 Testinių aplinkų paruošimas

2.7.4.1 Hibernate

Daugiausia resursų paruošiant testinę aplinką, naudojančią Hibernate atvaizdavimo įrankį, reikalauja reliacinės duomenų bazės paruošimas. Iš esmės tik objektinės duomenų bazės db4o testinės aplinkos paruošimas yra žymai greitesnis, tačiau Linq gali naudoti objektines duomenų bazes

(taip pat ir db4o). Reliacinė bazė turi būti instaliuota ir tinkamai sukonfigūruota, nemažai laiko gali būti sunaudota duomenų schemos paruošimui.

Vertiname Hibernate testinių aplinkų paruošimo sudėtingumą 0 (prastai).

2.7.4.2 Linq

Kaip ir Hibernate atveju, didesnę dalį resursų Linq naudojančios testinės aplinkos paruošimui sunaudoja duomenų bazės paruošimas, tačiau skirtingai nei Hibernate, Linq gali naudoti ir objektines duomenų bazes.

Vertiname Linq testinių aplinkų paruošimo sudėtingumą 1(vidutiniškai).

2.7.4.3 DataSet

Akivaizdu, kad DataSet atvaizdavimą naudojančios testinės aplinkos paruošiamas užtrunka taip pat ilgai, kaip ir Hibernate, kadangi reliacinės duomenų bazės paruošimas sunaudoja didesnę dalį testinės aplinkos paruošimo laiko.

Vertiname DataSet testinių aplinkų paruošimo sudėtingumą 0(prastai).

2.7.4.4 Db4o

Db4o objektinės bazės naudojimui praktiškai nereikalingas joks paruošimas. Reikia nurodyti duomenų bazės alokaciją diske ir prijungti prie projekto db4o biblioteką [Lin08]. Toks testinių aplinkų paruošimo būdas, visiškai nereikalaujantis instaliavimo ir konfigūravimo, yra greičiausias iš visų nagrinėjamų įrankių.

Vertiname db4o testinių aplinkų paruošimo sudėtingumą 2(puikiai).

3. Vertinimo rezultatai

Toliau esančioje lentelėje pateikiami agreguoti vertinimo rezultatai:

7 lentelė. Vertinimo rezultatai.

Įrankis / Kriterijus	Hibernate	Linq	DataSet	Db4o
SA1. Tipų atvaizdavimas.	**2**	**-**	**-**	**2**
SA2. Null - nuorodų atvaizdavimas.	**1**	**-**	**-**	**2**
SA3. Objektų sąryšių atvaizdavimas.	**2**	**-**	**-**	**2**
SS1. Schemų sinchronizavimas.	**1**	**0**	**0**	**2**
SS2. Ortogonalus saugojimas.	**1**	**-**	**0**	**2**
OI1. Objektų identitetas.	**0**	**1**	**0**	**2**
DI1. Laukų išrinkimas.	**2**	**1**	**2**	**0**
DI2. Užklausos pagal pavyzdį.	**0**	**0**	**0**	**2**
DI3. Užklausos naudojant sąsają .	**2**	**0**	**0**	**2**
DI4. Užklausos naudojant specifinę kalbą.	**2**	**2**	**0**	**2**
DI5. Laisvas išrinkimas.	**2**	**1**	**2**	**0**
DI6. Statinis tipų tikrinimas.	**0**	**2**	**0**	**2**
DI7.Išrinkimo greitaveika.	**1**	**1**	**1**	**1**
DI8. Išrikimo optimizavimas	**1**	**0**	**2**	**2**
KM1. Schemos modifikavimas.	**1**	**1**	**1**	**2**

KM2. Užklausų perdarymas.	**0**	**2**	**0**	**2**
KM3. Schemos konfigūracijos valdymas.	**0**	**1**	**1**	**2**
KM4. Testinių aplinkų paruošimas.	**0**	**1**	**0**	**2**

Gauti rezultatai parodo, kad objektinės duomenų bazės iš tikrųjų yra geriausias būdas spręsti iškeltas duomenų bazių ir objektiškai orientuoto programavimo nesuderinamumo problemas, ir žymiai geriau nei alternatyvos tinka projektams, naudojantiems judriąsias programų sistemų kūrimo metodikas. Automatizuoto atvaizdavimo įrankiai, tokie kaip Hibernate, nestipriai nusileidžia objektinėms duomenų bazėms objektinio ir reliacinio modelio nesuderinamumo problemų sprendimo efektyvume, tačiau prastai pritaikyti judriąsias programų sistemų kūrimo metodikas naudojantiems projektams, dėl sudėtingo schemų sinchronizavimo ir atvaizdavimo. Linq yra kompromisas tarp visų kitų variantų. Nors Linq daugelyje kriterijų negali būti vertinamas ar gavo prasčiausią vertinimą, nereikia pamiršti, kad tai labai jaunas šiuo metu stipriai plečiamas projektas. Iš vertinimo matome, kad dažniausiai Linq yra kitų vertinimų vidurkis, taigi ateityje gali tapti viena iš geriausių alternatyvų. DataSet naudojimas yra mažiausiai priimtinas ir tinka paprasčiausiems uždaviniams, kur objektų sąryšiai yra trivialus, arba jų iš viso nėra.

Iš viso pagal tris kriterijus (DI1, DI5, DI7) objektinės duomenų bazės negavo maksimalaus įvertinimo. Kadangi kriterijų svoris skirtingo tipo projektams gali būti nelygiavertis, į šiuos kriterijus reikėtų atkreipti dėmesį renkantis projekto duomenų bazės tipą. Kitose skyreliuose panagrinėsime šiuos kriterijus detaliau ir pateiksime objektinių ir reliacinių duomenų bazių panaudojimo apribojimus ir rekomendacijas.

.

3.1. Privalumai

Kaip parodė palyginimas, išvardintos problemos geriausiai sprendžiamos objektinių duomenų bazių pagalba, tačiau toks sprendimas tūri savo privalumų ir trūkumų. Šiame skyriuje agreguosime analizės ir vertinimo metu gautą informaciją, pateikdami objektinių ir reliacinių duomenų bazių privalumus naudojant jas objektiškai orientuoto programavimo kontekste. Po kiekvieno įvardinto punkto nurodomas susijusių kriterijų sąrašas.

Toliau išvardinami objektinėms duomenų bazėms būdingi privalumai [Bar07] [CIb06] [Oba01] [Lin07].

1. Nėra papildomų reliacinio ir objektinio modelio atvaizdavimo išlaidų. Dauguma šiuolaikinių projektų atliekami naudojant OOP kalbas, todėl reliacinių duomenų bazių panaudojimas neišvengiamai reikalauja objektinio ir reliacinio modelių atvaizdavimo (SA1-3).

2. Labiau tinka sudėtingų struktūrų saugojimui. Norint saugoti sudėtingas struktūras reliaciniame modelyje, tenka rinktis viena iš dviejų galimybių: saugoti visą struktūrą vienoje didelėje lentelėje su daugybe tuščių laukų, arba turėti daug mažesnių normalizuotų lentelių, sujungtų per išorinius raktus. Objektinės duomenų bazės gali vykdyti užklausas tokioms struktūroms nuo dešimties iki tūkstančio kartų greičiau priklausomai nuo struktūros sudėtingumo(DI1-8).
3. Realaus pasaulio duomenys dažniausiai turi hierarchinę struktūrą. Tokių struktūrų saugojimui geriau tinka objektinės duomenų bazės (SA1-3, DI7-8).
4. Užklausų kalbos naudojimas nebūtinas, tačiau galimas objektinėse duomenų bazėse, kadangi bendravimas su objektine duomenų baze vyksta visiškai skaidriai per kreipinius į objektus (DI2-4).
5. Nereikia rūpintis pirminiais raktais, kadangi objektinė duomenų bazė automatiškai sugeneruoja kiekvienam saugomam objektui unikalų identifikatorių (OI1).
6. Nereikia rūpintis objektinio ir reliacinio modelių sinchronizavimu(SS1-2, KM1-2).
7. Žymiai greičiau apdorojamos užklausos esant mažam transakcijų skaičiui (DI7).
8. Geriau tinka projektams, naudojantiems judriąsias programų sistemų kūrimo metodikas, dėl galimybės automatizuotai perdarinėti užklausas ir schemą, dėl paprastesnio schemos konfigūracijos valdymo ir dėl greitesnio testinių aplinkų paruošimo(KM1-4, SS1).

Toliau išvardinami reliacinėms duomenų bazėms būdingi privalumai [Bar07] [CIb06] [Oba01] [Lin07] [Bar02][Kin07].

1. Labiau tinka paprastų struktūrų saugojimui, kadangi greičiau išrenkami duomenys (DI7).
2. Galimybė išrinkti bet kokios kombinacijos duomenis (DI1, DI5)
3. Išvystytas transakcijų mechanizmas leidžia pasiekti geresnės greitaveikos esant dideliam transakcijų skaičiui (DI7).
4. Standartizuotas ir matematiškai pagristas duomenų saugojimo, keitimo ir išrinkimo modelis (DI1, DI5, DI7).
5. Didelis įrankių užpildančių trūkstamo funkcionalumo spragas pasirinkimas (SA1-3, SS1-2, KM1).
6. Neprisirišama prie konkrečios programavimo kalbos (DI6,KM2).

3.2. Trūkumai

Kaip ir praeitame skyrelyje, šiame agreguosime gautus vertinimo rezultatus ir analizės metu gauta informaciją ir pateisime reliacinių ir objektinių duomenų bazių trūkumus, naudojant

jas objektiškai orientuoto programavimo kontekste. Reikia pastebėti, kad dauguma objektinių duomenų bazių naudojimo privalumų tampa reliacinių duomenų bazių trūkumais ir atvirkščiai.

Toliau pateikiami objektinėms duomenų bazėms būdingi trūkumai [Bar07][CIb06][Oba01] [Cha05] [Kin07].

1. Objektinės duomenų bazės schemos pakeitimai vykdomi daug lėčiau nei reliacinėse duomenų bazėse. Reliacinių duomenų bazių schemos pakeitimas ne visada reikalauja atitinkamo objektinio modelio pakeitimų, objektinių duomenų bazių atveju tai yra neišvengiama (SS1,KM1).
2. Skirtingai nei reliacinėse duomenų bazėse, objektinė duomenų bazė dažniausiai yra pririšta prie tam tikros konkrečios OOP kalbos. (Db4o, kurią naudojome vertinime, palaiko dvi kalbas - Java ir C# - ir iš abiejų galima kreiptis į tą pati db4o egzempliorių) (DI6, KM2)
3. Užklausos, kurios gali būti atliktos objektinėje duomenų bazėje, yra daug griežtesnės nei reliacinėje. Iš esmės reliacinis modelis leidžia bet kokių lentelių apjungimą, leisdamas išrinkti įvairiausius duomenų rinkinius. Objektiniam klasių modeliui gali būti vykdomos užklausos tik atsižvelgiant į sąryšius tarp klasių (DI1, DI5).
4. Objektinės duomenų bazės lėčiau nei reliacinės apdoroja užklausas, esant dideliam transakcijų skaičiui (DI7).

Toliau pateikiami reliacinėms duomenų bazėms būdingi trūkumai.

1. Reliacinio ir objektinio modelių atvaizdavimo išlaidos yra neišvengiamos naudojant reliacinės duomenų bazes(SA1-3).
2. Netinka sudėtingoms struktūroms saugoti, dėl lėto išrinkimo ir komplikuoto sudėtingų struktūrų atvaizdavimo (SA1-3,DI1-8)
3. Prastai tinka projektams naudojantiems judriąsias programų sistemų kūrimo metodikas, nes nėra galimybės statiškai patikrinti užklausų, automatizuotai daryti DB schemos pakeitimus, sudėtinga valdyti konfigūracija ir ruošti testines aplinkas(KM1-4, SS1, DI6).

3.3. Rekomendacijos ir apribojimai

Išvardinti reliacinių ir objektinių duomenų bazių naudojimo privalumai ir trūkumai leidžia sudaryti bendras naudojimo rekomendacijas ir apribojimus. Prieš naudojant gerai pažįstamas reliacinės ar bandant objektines duomenų bazes reikėtų atsižvelgti į tokio tipo duomenų bazių panaudojimo apribojimus ir rekomendacijas [Bar07][CIb06][Oba01][Cha05][Kin07][Edl08] [Dbo08][Bar02]:

- dėl prisirišimo prie programavimo kalbos, objektinės duomenų bazės netinka aplinkoms su įvairiomis technologijomis ir programavimo kalbomis parašytomis programų sistemomis;

- jei duomenų bazėje bus saugomos nesudėtingos duomenų struktūros, geriau pasirinkti labiau pritaikytas tokio tipo duomenims skirtas reliacines duomenų bazes;
- objektinės duomenų bazės netinka didelio transakcijų kiekio sistemose, dėl žymaus objektinių duomenų bazių sulėtėjimo, esant dideliam transakcijų skaičiui, ir dėl nepakankamai išvystytos klasterizavimo technologijos;
- objektinės duomenų bazės netinka sistemoms, kuriose bus daroma daug agreguotų užklausų, pavyzdžiui, OLAP;
- objektinės duomenų bazės puikiai tinka integravimui į portatyvius įrenginius, kadangi toks panaudojimas nereikalauja didelio transakcijų skaičiaus ir sudėtingų agreguotų užklausų, be to objektinių duomenų bazių valdymo sistemos užima daug mažiau vietos atmintyje ir diske;
- reliacinės duomenų bazės puikiai tinka kaip bendros centrinės saugyklos, kadangi tinka mišrioms aplinkoms, palaikomos daugelio jau sukurtų įrankių, turi gerai išvystytus transakcijų, replikavimo, klasterizavimo ir saugumo mechanizmus.

Rezultatai ir išvados

Darbe išanalizuotos reliacinių ir objektinių duomenų bazių ir objektiškai orientuoto programavimo nesuderinamumo bei naudojimo su judriosiomis programų sistemų kūrimo metodikomis problemos. Sugrupuoti šiandien populiariausi ir dažniausiai naudojami išvardintų problemų sprendimo būdai. Sudarytas sprendimo būdų efektyvumo vertinimo modelis. Vertinimo modelis pritaikytas vertinant po vieną įrankį iš kiekvienos analizės metu sudarytos kategorijos. Vertinimo ir analizės metu gauta informacija panaudota išskiriant reliacinių ir objektinių duomenų bazių taikymo privalumus ir trūkumus bei pateikiant naudojimo rekomendacijas ir apribojimus.

Palyginimo metu paaiškėjo, kad kaip šiuolaikinės objektinės duomenų bazės, taip ir reliacinėms duomenų bazėms skirti įrankiai ir metodai ne visada sėkmingai gali būti taikomi duomenų bazių ir objektiškai orientuoto programavimo nesuderinamumams šalinti. Nors objektinės duomenų bazės gavo geriausius vertinimus, tačiau nagrinėdami trūkumus pastebėjome, kad objektinės duomenų bazės turi svarbių taikymo apribojimų - prisirišimą prie konkrečios objektinės programavimo kalbos, prastas sudėtingų agreguojančių užklausų vykdymo galimybes ir lėtą išrinkimą esant dideliam transakcijų skaičiui. Automatinio atvaizdavimo įrankiai (tokie kaip Hibernate) prastai tinka projektams, naudojantiems judriąsias programų sistemų kūrimo metodikas, ir sunkiai pritaikomi sudėtingoms hierarchinėms struktūroms atvaizduoti. Reliacinių koncepcijų integravimas į tarpinius objektus tinka tik labai paprastoms objektų schemoms atvaizduoti. Reliacinių koncepcijų integravimo į objektiškai orientuotas programavimo kalbas kategorijoje vertintas Linq yra kompromisas tarp kitų alternatyvų. Linq yra pakankamai naujas projektas ir šiuo metu aktyviai vystomas, tačiau jau šiandien pretenduoja į universalaus sprendimo vardą.

Šaltinių sąrašas

[Aka01] Akadia. Advacned Queueing, http://www.akadia.com/services/ora_advanced_queueing.html, 2001.

[Amb05] S. Ambler. Agile Techniques for Object Databases, http://www.db4o.com/about/productinformation/whitepapers/db4o Whitepaper Agile Techniques for Object Databases.pdf , 2005.

[ARe07] Active Record design pattern. http://www.martinfowler.com/eaaCatalog/activeRecord.html, 2007.

[Bar02] R. Baronas. Duomenų bazių valdymo sistemos. TEV, 2002.

[Bar07] OODBMS Facts. Barry & Associates, http://www.odbmsfacts.com, 2007

[BEA06] BEA Systems. Inheritance, http://edocs.bea.com/kodo/docs41/full/html/ejb3_overview_mapping_inher.html, 2006.

[Bla05] M. Blaha. The Dilemma of Encapsulation vs. Query Optimization, http://www.odbms.org/download/007.01 Blaha The Dilemma of Encapsulation vs Query Optimization July 2005.pdf , 2005.

[Cha05] Jagadish Charterjee. Introduction to RDBMS, OODBMS and ORDBMS, http://www.aspfree.com/c/a/Database/Introduction-to-RDBMS-OODBMS-and-ORDBMS/1/ , 2005.

[CIb05] William R. Cook, Ali H. Ibrahim. Integrating Programming Languages & Databases: What's the Problem?, http://www.cs.utexas.edu/~wcook/Drafts/2005/PLDBProblem.pdf , 2005.

[Coo06] William R. Cook, Carl Rosenberg. Native Queries for Persistent Objects, http://www.cs.utexas.edu/~wcook/papers/NativeQueries/NativeQueries8-23-05.pdf

[Dat08] .NET Framework Developer Center. DataSet, http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx , 2008

[Dbo08] Db4objects, http://www.db4o.com/ , 2008

[Edl08] Steffan Edlich, Daniel Oltmanns.Db4o in the Mirror of JPA/EJB and Hibernate, http://dndj.sys-con.com/read/497141_1.htm , 2008

[Ext08] Extreme Programming, http://www.extremeprogramming.org/ , 2008.

[Gre06] R. Greene. OODBMS Architectures, http://www.odbms.org/download/028.01 Greene OODBMS Architectures September 2006.PDF, 2006

[Hib07] Hibernate reference documentation. http://www.hibernate.org/hib_docs/v3/reference/en/pdf/hibernate_reference.pdf, 2007.

[Kin07] G. King. In Defence of RDBMS, http://blog.hibernate.org/Bloggers/Everyone/Year/2007/Month/05/Day/23#in-defence, 2007

[Lin07] LinuxDevices.com. Object database explained, http://www.linuxdevices.com/news/NS9637482600.html, 2007

[Lin08] The LINQ Project. http://msdn2.microsoft.com/en-us/netframework/aa904594.aspx, 2008

[Map06] Mapping Objects to Relational Databases: O/R Mapping In Detail. http://www.agiledata.org/essays/mappingObjects.html, 2006

[Mis06] The object-relational impendace mismatch. http://www.agiledata.org/essays/impedanceMismatch.html, 2006

[New06] Ted Neward. The Vietnam of Computer Science, http://www.odbms.org/download/031.01 Neward The Vietnam of Computer Science June 2006.PDF, 2006

[New07] T. Neward. Avoiding the Quagmire, http://www.odbms.org/download/031.02 Neward Avoiding the Quagmire May 2007.PDF, 2007

[Oba01] D. Obasjano. An exploration of object oriented database management systems, http://www.25hoursaday.com/WhyArentYouUsingAnOODBMS.html, 2001

[Obj07] Objectivity. Object Oriented Database vs Relational Database, http://www.objectivity.com/pages/object-oriented-database-vs-relational-database/benchmark.html , 2007.

[Odb06] ODMBS. History , http://www.odbms.org/introduction_history.html , 2006.

[Ora07] Oracle Designer Documentation. - http://www.oracle.com/technology/documentation/designer.html, 2007.

[Pol08] PolePosition. The open source database benchmark, http://www.polepos.org/ , 2008

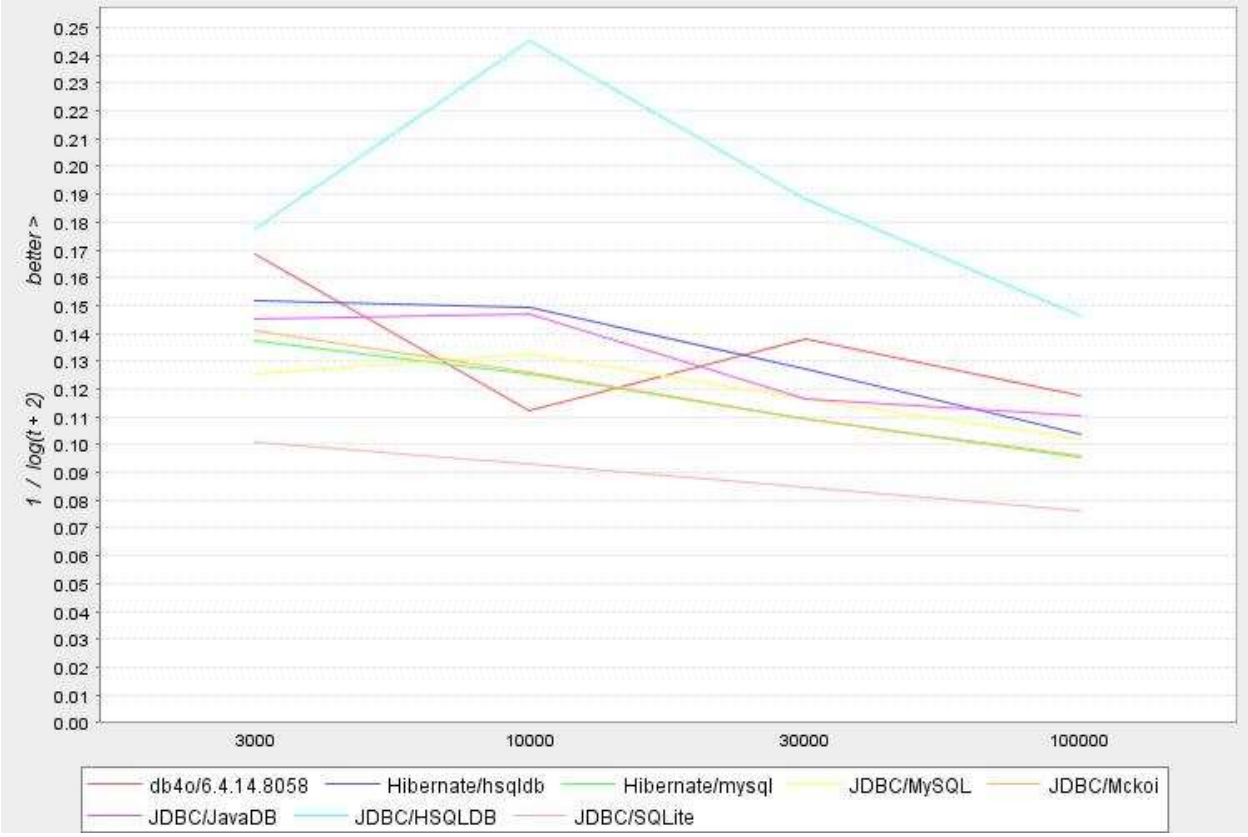
[Ris00] L. Rising. The Scrum Software Development Process for Small Teams http://members.cox.net/risingl1/Articles/IEEEScrum.pdf , 2000

[Sql92] SQL 92, http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt , 1992.

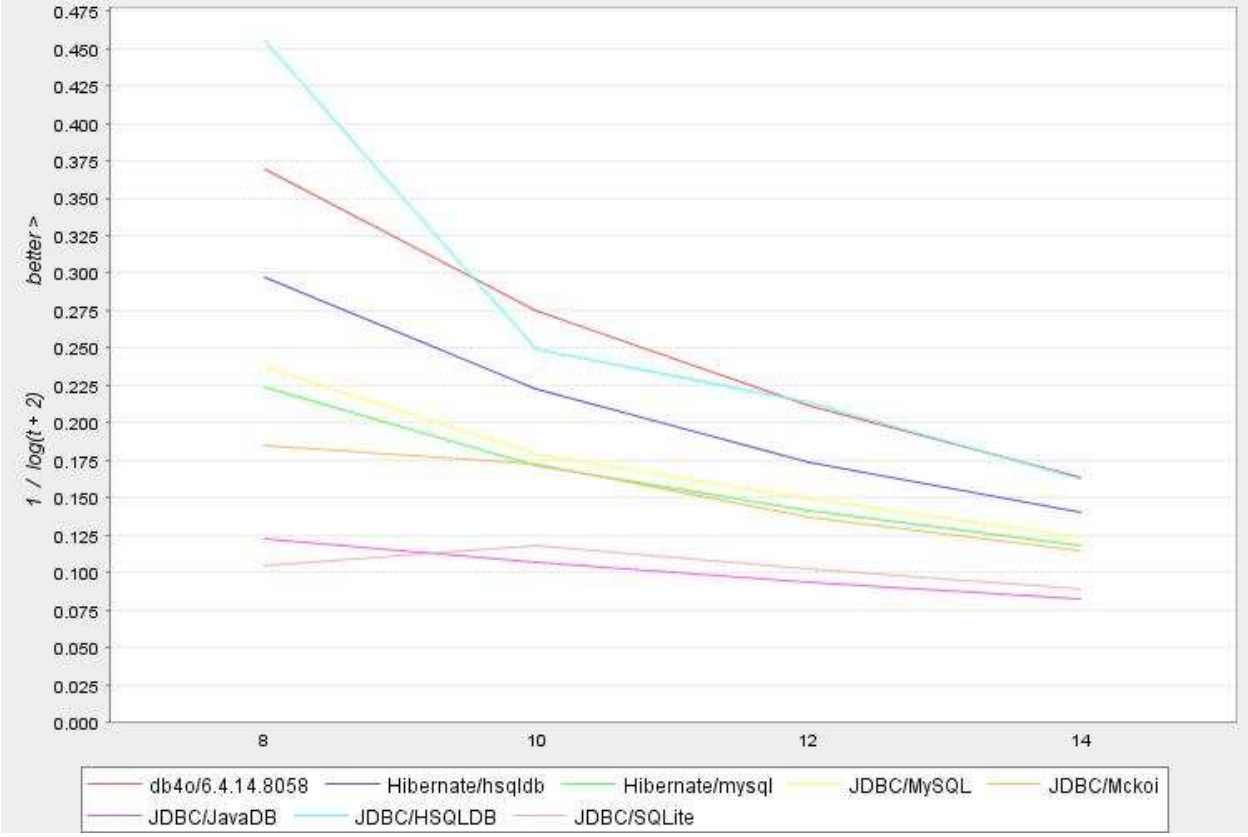
[Ter04] Teradata. RDBMS History, http://www.teradata.com/t/page/127057/index.html, 2006.

1 priedas. PolePos duomenų bazių išrinkimo greitaveikos matavimai.

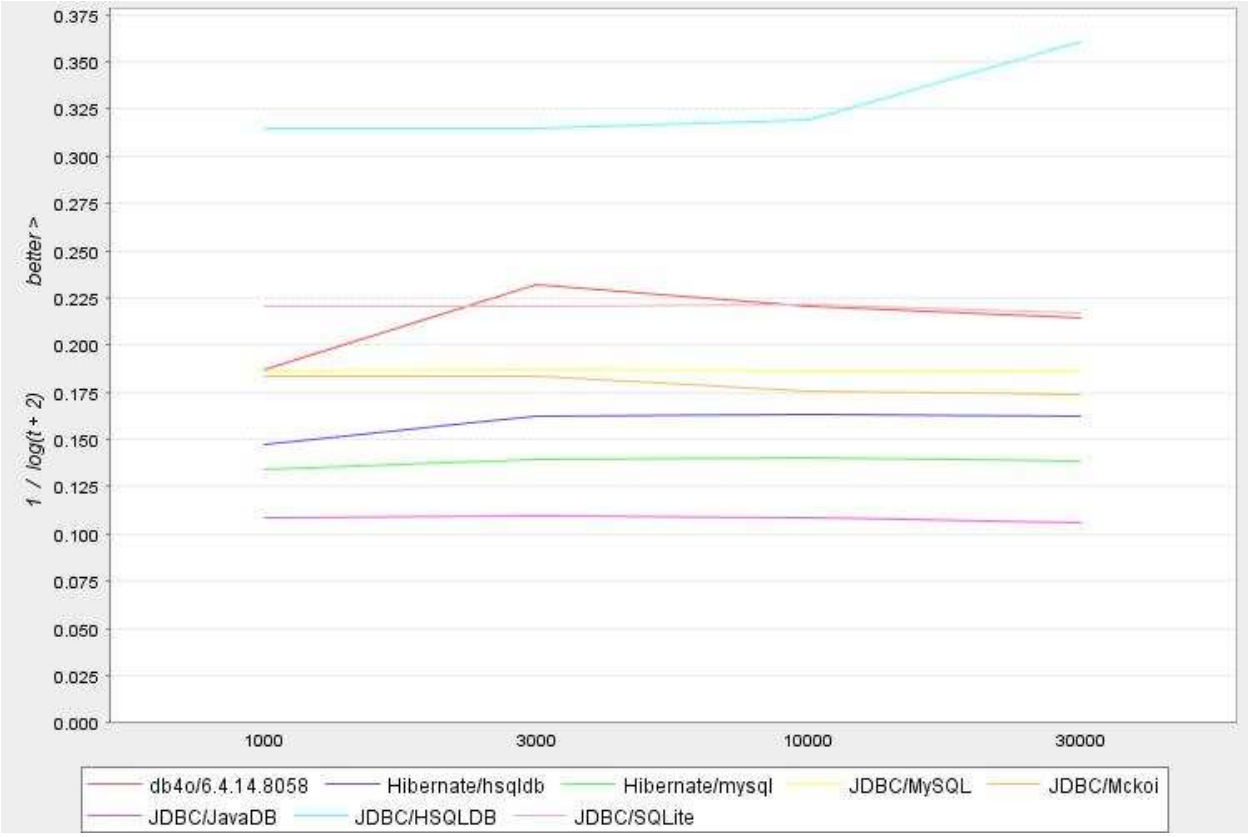
Melbourne testas: paprastų objektų be sąryšių išrinkimas (1 užklausa) (objektų skaičius, laikas).



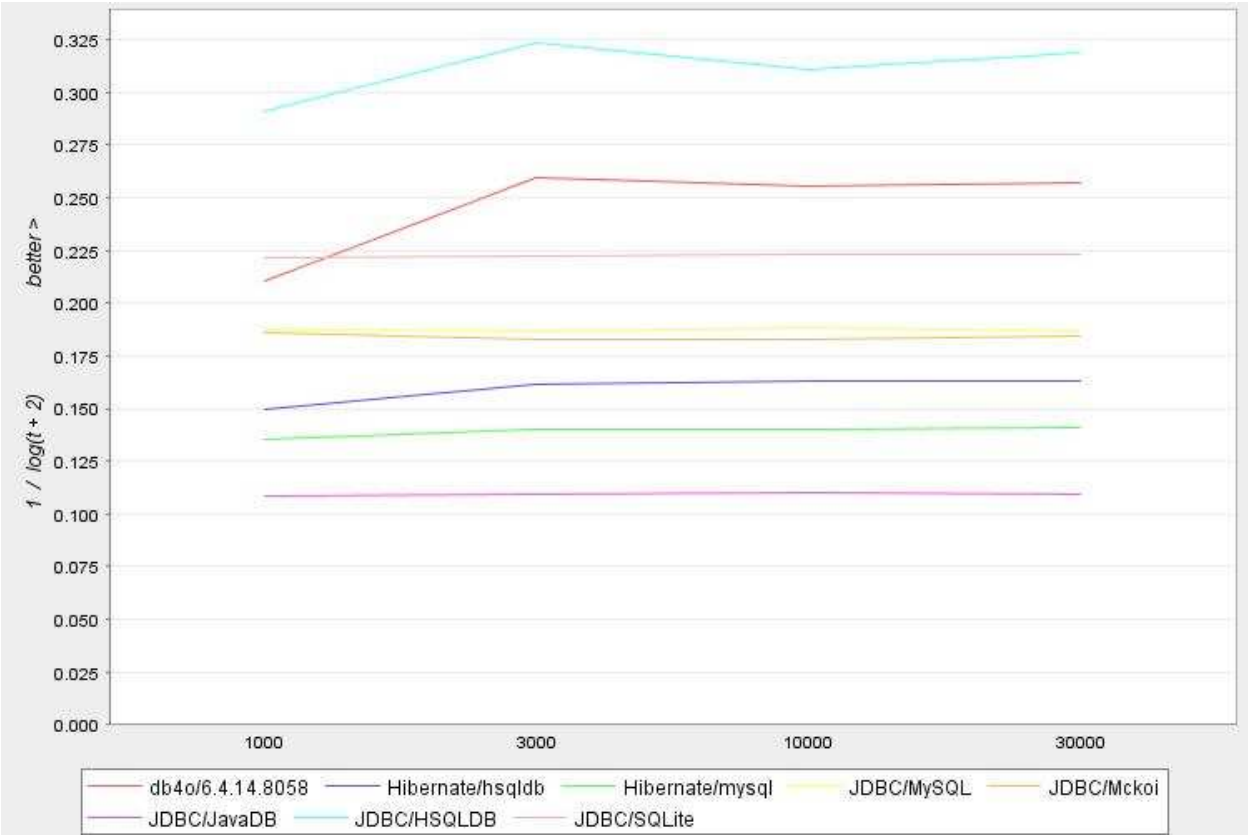
Sepang testas: objektų medžio išrinkimas (1 užklausa) (medžio gylis, laikas).



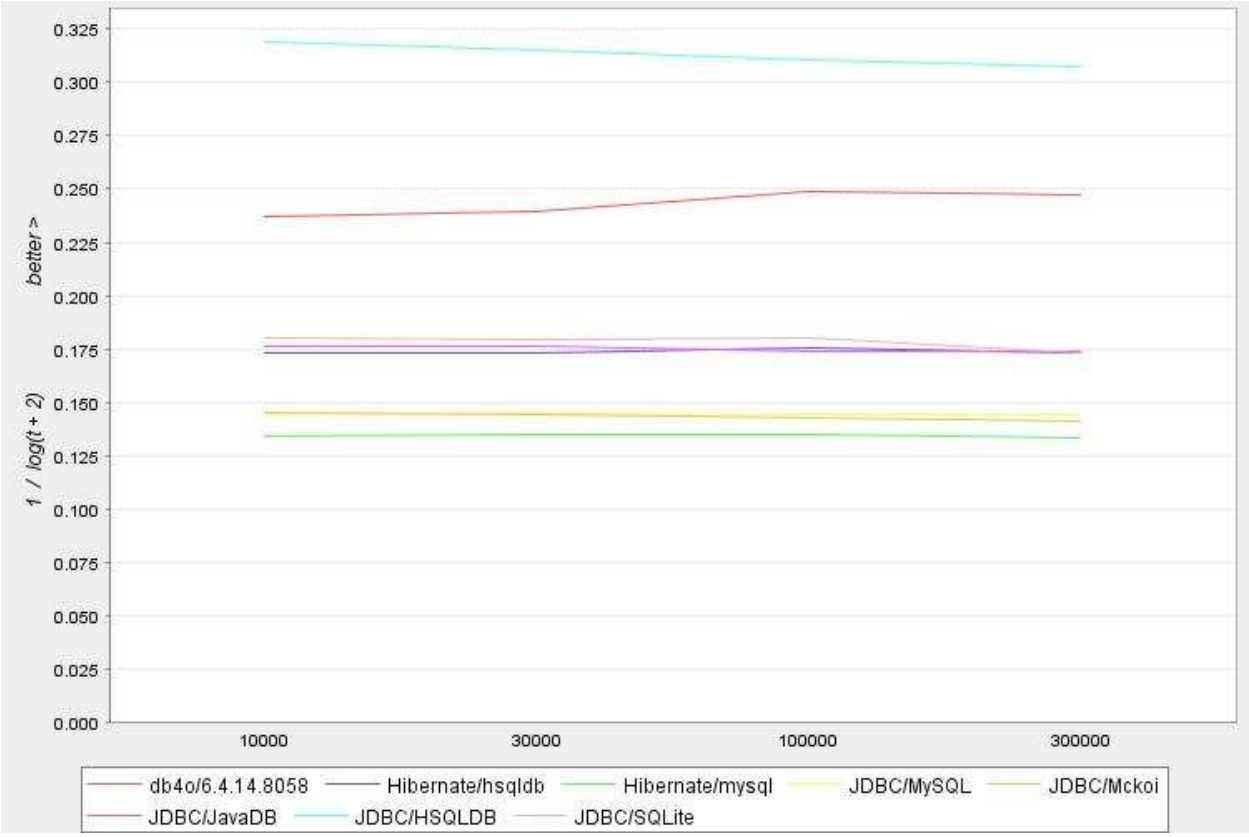
Bahrain testas: paprastų objektų be sąryšių (string tipo) išrinkimas (900 užklausų) (objektų skaičius, laikas).



Bahrain testas: paprastų objektų be sąryšių (int tipo) išrinkimas (900 užklausų) (objektų skaičius, laikas).



Imola testas: objektų išrinkimas pagal ID (5000 užklausų) (objektų skaičius, laikas).



Barcelona testas: objektų išrinkimas iki 5 paveldėjimo lygmens(100 užklausų) (objektų skaičius, laikas).