

RĪGAS TEHNISKĀ UNIVERSITĀTE
Datorzinātnes un informācijas tehnoloģijas fakultāte
Datorsistēmu institūts
Informātikas un programmēšanas katedra

Sergejs TERENCEVS

Datorsistēmas bakalaura programmas students
(stud. apl. nr. 061RDB140)

DATU BAŽU IZSTRĀDE .NET VIDĒ

Bakalaura darbs

Zinātniskais vadītājs
Dr. sc. ing., docente
M. Uhanova

Rīga, 2010

Bakalaura darbs izstrādāts **Informātikas un programmēšanas katedrā**

Darba autors: stud.
(paraksts, datums)

Zinātniskais vadītājs:
(paraksts, datums)

Bakalaura darbs ieteikts aizstāvēšanai:

IP katedras vadītājs, as. prof. J.Lavendels
(paraksts, datums)

Bakalaura darbs aizstāvēts Lietišķo datorsistēmu programmatūras virziena komisijas 2010. g
_____ sēdē un novērtēts ar atzīmi _____ ()

Lietišķo datorsistēmu programmatūras virziena komisijas sekretārs: _____

Anotācija

Darba ir apskatītas problēmas saistītas ar objektorientētas paradigmas un relāciju datu modeļa atšķirībām. Dotas problēmas balstās uz objektu asociāciju un tabulu attiecību, ka arī datu modeļu koncepciju atšķirībām, objektu ielādes optimizēšanu, datu tipu neatbilstību un objektu identitātes nodrošināšanu.

Bakalaura darba mērķis ir novērtēt .NET platformas piedāvātos risinājumus datu bāžu un programmēšanas valodu integrācijas problēmu risināšanai, veicot esošo līdzekļu izpēti un to salīdzinošo analīzi ar alternatīvam tehnoloģijām.

Darbā ietvaros tiek identificēts datu bāžu un programmēšanas valodu risināmo integrācijas problēmu kopums. Ir apskatītas .NET un Java (J2EE) platformu tehnoloģijas, atklājot risinājumu kopumu, kurš ir piedāvāts esošo problēmu novēršanai. Tiek novērtēta minētu risinājumu lietderība un apkopoti doto tehnoloģiju priekšrocības un trūkumi. Pamatojoties uz doto informāciju, ir veikta esošo risinājumu salīdzinoša analīze.

Bakalaura darbā ir izstrādāta programmatūra, kura demonstrē esošo risinājumu pielietošanu informācijas sistēmu izstrādē. Programmatūras izstrāde tika veikta C# programmēšanas valodā, izmantojot ADO.NET Entity Framework tehnoloģiju.

Iegūtie rezultāti sniedz priekšstatu par sasniegtiem panākumiem programmēšanas valodu un datu bāžu savietojamības problēmu novēršanai, rāda priekšstatu par pašreizējo pētījumu virzienu un kalpo par kritērijiem piemērotākas tehnoloģijas izvēlē.

Darbs satur 75 lappuses, 49 attēlus, un 8 tabulas. Literatūras saraksts satur 24 informācijas avotus.

Аннотация

В работе рассматриваются проблемы совместимости объектно-ориентированной и реляционной модули. Такими проблемами являются различия между отношениями таблиц и ассоциациями объектов, различия в концепциях модели данных, оптимизирования загрузки объектов, противоречивость типов данных и управление идентичностью объектов.

Цель бакалаврской работы оценить предоставленные решения в платформе .NET, исследовав существующие средства и выполнив их сравнительный анализ с альтернативными технологиями.

В работе идентифицируются проблемы интеграции баз данных и языков программирования. Рассматриваются технологии платформ .NET и Java (J2EE), в процессе чего особое внимание уделяется предоставленным решениям по устранению данных проблем. В работе анализируются целесообразность упомянутых решений, и рассматриваются возможности и недостатки данных технологий. Основываясь на данную информацию, выполняется сравнительный анализ существующих средств.

В работе разработано программное обеспечение, которое демонстрирует применения существующих решений при разработке информационных систем. Программное обеспечение было разработано в языке программирования C#, с использованием технологии ADO.NET Entity Framework.

Получение результаты отображают достигнутый прогресс в области решений проблем совместимости баз данных и языков программирования, указывают направление текущих исследований, и служат критериями для выбора приемлемой технологии.

Работа состоит из 75 страниц, 49 изображений и 8 таблиц. В списке литературы указано 24 источников литературы.

SATURS

SATURS.....	5
LIETOTI SAĪSINĀJUMI.....	7
IEVADS	8
1. DATU BĀŽU UN PROGRAMMĒŠANAS VALODU INTEGRĀCIJAS PROBLĒMAS ..	10
2. DATU PIEJAS TEHNOLOĢIJAS.....	18
2.1. Microsoft datu pieejas tehnoloģijas	18
2.1.1. Datu pieejas tehnoloģija ADO.NET	19
2.1.1.1. ADO.NET datu gādnieki.....	19
2.1.1.2. ADO.NET objektu modelis.....	20
2.1.1.3. ADO.NET pielietojums paradigmu savietojamības problēmu risināšanā	22
2.1.2. Paplašinājums LINQ	25
2.1.3. Risinājums LINQ to SQL	27
2.1.3.1. LINQ to SQL arhitektūra	27
2.1.3.2. LINQ to SQL pielietojums paradigmu savietojamības problēmu risināšanā...31	
2.1.4. Risinājums ADO.NET Entity Framework.....	33
2.1.4.1. ADO.NET Entity Framework arhitektūra.....	33
2.1.4.2. ADO.NET Entity Framework pielietojums paradigmu problēmu risināšanā ..36	
2.2. Java platformas datu pieejas tehnoloģijas	39
2.2.1. Datu pieejas tehnoloģija JDBC	39
2.2.1.1. JDBC draiveru tipi	39
2.2.1.2. Tehnoloģijas JDBC objektu modelis.....	40
2.2.1.3. JDBC pielietojums paradigmu savietojamības problēmu risināšanā	41
2.2.2. Objekt-relācijas kartēšanas tehnoloģija Hibernate	44
2.2.2.1. Hibernate arhitektūra.....	44
2.2.2.2. Hibernate objektu modelis	48
2.2.2.3. Hibernate pielietojums paradigmu savietojamības problēmu risināšanā	48
3. DATU PIEJAS TEHNOLOĢIJU SALIDZĪNOŠA ANALĪZE	53
4. TEHNOLOĢIJAS ADO.NET ENTITY FRAMEWORK PRAKTISKS PIELIETOJUMS ..56	
4.1. Sistēmas funkcionālās prasības	56

4.2.	Sistēmas modeļa projektēšana	57
4.3.	Sistēmas implementēšana	61
4.4.	Izstrādātās programmatūras apraksts	63
SECINĀJUMI		72
LITERATŪRA		74

LIETOTI SAĪSINĀJUMI

Saīsinājums	Angliski	Latviski
ADO.NET	ActiveX Data Objects .NET	ActiveX datu objekti .NET
COM	Common Object Model	Kopējais objektu modelis
CSDL	Conceptual Schema Definition Language	Konceptuālas shēmas uzdošanas valoda
JDBC	Java Database Connectivity	Java datu bāzu savietojamības saskarne
JDK	Java Development Kit	Valodas Java programmizstrādes aprīkojums
LINQ	Language Integrated Query	Integrēta vaicājumu valoda
MSL	Mapping Specification Language	Kartēšanas specifikācijas valoda
ODBC	Open Database Connectivity	Atvērtā datu bāzu savietojamības saskarne
OLAP	Online Analytical Processing	Tiešsaistes analītiska apstrāde
OLE DB	Object Linking and Embedding Database	Objektu sasaistes un iegultes datu bāzu saskarne
SQL	Structured Query Language	Strukturēta vaicājumuvaloda
SSDL	Store Schema Definition Language	Datu avota shēmas uzdošanas valoda
XML	Extensible Markup Language	Paplašināmās iezīmēšanas valoda

IEVADS

Mūsdienās relāciju datu bāzes tiek plaši pielietotas korporatīvo uzņēmumu datu glabāšanas vajadzībām. Savukārt, programmatūras analīzes, projektēšanas un izstrādes jomā plašu ievērību ir ieguvusi objektorientēta metodoloģija. Veicot programmatūras izstrādi uzņēmumu datu apstrādes vajadzību apmierināšanai, izstrādātājiem ir jāparedz datu pieejas līmenis starp datu bāzi un programmatūru, kurš ļauj pārvarēt savietojamības problēmas starp diviem atšķirīgiem relācijas un objektu datu modeļiem.

Dotas problēmas ir saistītas ar programmas objektu glabāšanas nepieciešamību, kas nodrošinātu objektu stāvokļu atjaunošanas iespējas nākamās programmas izpildes laikā. Relāciju datu bāzes nav labi piemērotas doto uzdevumu veikšanai. Vairākos gadījumos programmatūras projektētāji un izstrādātāji ir paši atbildīgi par objektu klašu un tabulu pretstatījumu programmatūras projektā. Rodas nepieciešamība tādu problēmu risināšanai, ka datu tipu neatbilstība, iekapsulēšanas, mantošanas un polimorfisma principu nodrošināšana, transakciju vadības realizēšana, asociāciju nodrošināšana starp relāciju tabulām un objektiem, vaicājumu pieprasījumu skaita samazināšanai proporcionāli objektu asociācijām ($n + 1$ selects problem) u.tml. Dotais problēmu kopums, kas ir specifisks objektu un relāciju modeļiem tiek apzīmēts ar terminu „savietojamības zuduma” problēma (*impedance mismatch problem*) [1].

Mūsdienās eksistē liels daudzums dažādu datu pieejas tehnoloģiju, kuras spēj mazināt un dažos gadījumos pat novērst vairākas no esošām problēmām, kamēr pilnīgi visas problēmas netiek atrisinātas. Katrai no esošām tehnoloģijām ir raksturīgs savs risinājumu kopums, savas priekšrocības un trūkumi. Dažās no tām balstās uz abstrakcijas līmeņa izveidošanu starp datu bāzi un programmēšanas valodu, piemēram, objekt-relācijas kartēšanas tehnoloģijas Hibernate, ADO.NET Entity Framework, citās paplašina programmēšanas valodas sintaksi ar konstrukcijām līdzīgam SQL vaicājumu valodai, piemēram, LINQ u.t.t. Dotas tehnoloģijas ļauj samazināt izmaksas un laika patēriņu programmatūras izstrādei, taču var prasīt papildus atmiņas resursu izmantošanu un ietekmēt uz programmas veiktspēju. Vairākos gadījumos izstrādātājiem nākas optimizēt programmas pirmtekstu veiktspējas uzlabošanai un ierobežot programmas funkcionalitāti, atteicoties no iespējamam objektorientētas metodoloģijas koncepcijām.

Bakalaura darba mērķis ir novērtēt .NET platformas piedāvātos risinājumus datu bāzu un programmēšanas valodu integrācijas problēmu risināšanai, veicot esošo līdzekļu izpēti un to salīdzinošo analīzi ar alternatīvam tehnoloģijām.

Darba uzdevumi:

- izpētīt datu bāzu un programmēšanas valodu integrācijas problēmas;
- apskatīt risinājumus, kuri ir piedāvāti .NET programmatūras izstrādes platformā;
- salīdzinot esošos risinājumus ar alternatīvam tehnoloģijām;
- demonstrēt praktisko risinājumu pielietošanu.

Pirmajā nodaļā tiek apskatītas datu bāzu un programmēšanas valodu integrācijas problēmas. Ir apskatīta problēmu būtība un rašanas cēloņi, ka arī pievērsta uzmanība objektorientētam datu bāzēm. Otrajā nodaļā tiek apskatītas .NET un Java platformu datu pieejas tehnoloģijas: ADO.NET, ADO.NET Entity Framework, LINQ, JDBC, Hibernate. Ir izpētītas tehnoloģiju arhitektūras, atklāts risinājumu kopums, kurš tiek piedāvāts apskatīto problēmu risināšanai. Trešā nodaļā tiek veltīta kritērijiem, pēc kuriem var secināt par tehnoloģiju lietderību problēmu novēršanai, tiek apkopotas tehnoloģiju priekšrocības un trūkumi. Ceturtā nodaļā ir apskatīts izstrādātas programmatūras projektēšanas un izstrādes posms, praktiski demonstrēti teorētiska materiāla atklātie risinājumi. Ir definētas funkcionālas prasības un sastādīts īss programmatūras apraksts.

1. DATU BĀŽU UN PROGRAMMĒŠANAS VALODU INTEGRĀCIJAS PROBLĒMAS

Jā par pamatu datu bāžu un programmēšanas valodu integrācijas problēmu izskaitīšanai ņemt relāciju un objektorientēto paradigmu, tad savietojamības problēmas rodas vairākos programmatūras izstrādes un projektēšanas posmos. Darbā ir apskatītas sekojošas problēmas:

- Primitīvo datu tipu neatbilstība,
- Programmatūras modeļa kartēšana datu bāzes shēmā,
- Objektu identitātes nodrošināšana,
- Objektu asociāciju un tabulu attiecību saistīšana,
- Objektu ielādes optimizēšana.

Primitīvo datu tipu neatbilstība. Relāciju datu bāzes un objektorientētas programmēšanas valodās atbalsta dažādus primitīvos datu tipus. Piemēram, datu bāzēs bieži vien virknēm (*strings*) tiek definēts ierobežots garums, kas nav raksturīgs tādām programmēšanas valodām, kā C# vai Java. Problēma rodas, ja tiek mēģināts ieglabāt virkni ar 150 rakstzīmēm tabulas laukā, kurš akceptē tikai 100 rakstzīmes. Cits vienkāršs piemērs ir *Boolean* datu tips, kurš nav paredzēts relāciju datu bāzēs.

Problēmas rodas arī ar *null* vērtībām. Relāciju datu bāzēs *null* vērtība izsaka nezināmo vērtību [1], kura var tikt piešķirta jebkurām tabulas laukam ar dažādu datu tipu. Savukārt, objektorientētas programmēšanas valodas *null* vērtība var tikt piešķirtā tikai objektu atsaucēm, kas nav raksturīgs vesela vai daļveida skaitļu datu tiem. Dažas programmēšanas valodas, piemēram, C++ vai C# ļauj definēt konstrukcijās, kuras atbilst datu bāžu semantikai, taču šāda iespēja nepastāv visas objektorientētas programmēšanas valodas [1]. Piemēram, 1.1 attēlā ir attēlots programmēšanas valodas C# pirmteksts, kurā vesela skaitļu tipa mainīgā i tiek piešķirta *null* vērtība.

01	<code>int? i = null;</code>
----	-----------------------------

1.1. att. *Null* vērtības piešķiršanas piemērs.

Primitīvi datu tipu var būt atšķirīgi arī starp dažādam datu bāzēm, kas ir saistīts ar to, ka standarts SQL-92 neprecizē vairāku datu tipu absolūto izmēru [1].

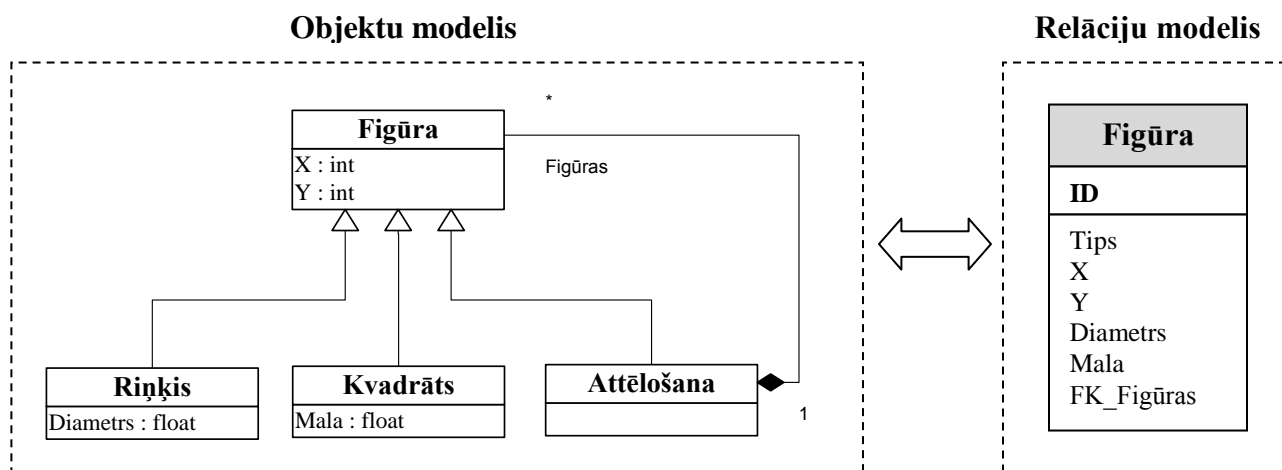
Objektu metožu glabāšana. Objekts ir klases eksemplārs, kas reprezentē kādu noteiktu dabas objektu. Tām ir raksturīgas īpašības un uzvedība. Relāciju datu bāzēs ir iespējams operēt ar datiem vienīgi ar SQL vaicājumu un glabājamo procedūru palīdzību, taču tādas koncepcijas kā objektu metodes datu bāžu kontekstā neeksistē. Relāciju datu bāzēs dati un kods ir stingri nodalīti.

Objektu metožu glabāšanas problēma ir atrisināta objektorientētas datu bāzēs, kuras tika izstrādātas 90.g. sākumā ar nolūku nodrošināt sarežģītas objektu hierarhijas glabāšanas iespēju, ievērojot objektorientētas paradigmas koncepcijas. Objektorientētas datu bāžu vadības sistēmas pilda automātisko programmas pirmteksta analīzi, uz kuras pamatā datu bāzē tiek izveidotas struktūras objektu glabāšanai [2]. Par cik katram vienas un tās pašas klases eksemplāram ir raksturīgs tāds pats metožu kopums, atšķirība no objektu atribūtu vērtībām, metodes tiek saglabātas tikai vienu reizi un tiek glabātas atsevišķi [2]. Saskaņā ar standartu ODMG-93, objektorientētas datu bāzes pilda unikālo objektu identifikatoru un atsauču uz citiem objektiem glabāšanu un vadību, definē iespējamās operācijas starp objektu asociācijām un nodrošina mantošanu, abstrakciju u. c. objektorientētas paradigmas koncepciju ievērošanu [2].

Objektorientētas datu bāzes nav ieguvušas plašu komerciālo pielietojumu salīdzinājumā ar relāciju datu bāzēm. Iespējamais iemesls ir tāds, ka objektorientētas datu bāzes reprezentē nevis autonomo datu glabātuvi, bet vairāk kalpo kā savdabīgs objektorientētu programmēšanas valodu paplašinājums objektu patstāvības nodrošināšanai [3]. Šādi dotas datu bāzes nenodrošina tādas relāciju datu bāzēm raksturīgas iespējas, ka datu analīze ar OLAP kubu palīdzību, atskaišu veidošana u.tml. Galvenās objektorientēto datu bāžu priekšrocības ir cieša integritāte ar objektorientētam programmēšanas valodām un samēra liela veiktspēja [2].

Programmatūras modeļa kartēšana datu bāzes shēmā. Objektorientēta paradigma ievieš tādas koncepcijas, ka mantošana, polimorfisms, abstrakcija un iekapsulēšana. Izmantojot esošās koncepcijas ir iespējams izstrādāt programmatūras struktūru, kura atbilst funkcionālām prasībām kādas noteiktās uzņēmējdarbības vai reālas pasaules problēmsfēras jomā. Problēma rodas, ja ir nepieciešams atveidot izstrādāto klašu hierarhiju relāciju datu modelī, kurš ir balstīts uz relāciju algebru, attiecību normalizāciju starp datu bāzes tabulām, datu glabāšanu tabulu struktūrās u.tml. Par piemēru izskatīsim vienkāršu UML klašu diagrammu, kura ir attēlota 1.2 attēlā. Dotajā klašu

hierarhijā ir izstrādāta bāzes klase „figūra”, no kuras izmantojot kompozīcijas asociāciju tiek izdalīta apakšklase „attēlošana”. Ka arī tiek atveidots mantošanas princips, kurā apakšklases „riņķis”, „kvadrāts” un „attēlošana” manto bāzes klases „figūra” atribūtus. Par cik relāciju datu bāzes neatbalsta tādas koncepcijas, ka mantošana un kompozīcija, tad nav iespējams reprezentēt datus līdzīga veida starp diviem atšķirīgiem modeļiem. Šajā gadījuma piemērotākais risinājums ir izstrādāt tikai vienu tabulu, kas ļaus izvārties no liekas datu dublēšanas starp iespējamajām datu bāzes tabulām un uzlabos datu izgūšanas veiktspēju. Kaut gan dotajā situācijā rodas papildus grūtības, kuras nākas pārvarēt programmatūras izstrādātājiem, lai nodrošinātu objektu stāvokļu saglabāšanu datu bāzes tabulā un pretēji objektu hierarhijas atveidošanu atmiņā, pamatojoties uz izgūtiem tabulas datiem.



1.2. att. Objektu modeļa kartēšanas problēmas piemērs [4].

Atsevišķos gadījumos objektu modelis tiek projektēts iespējami līdzīgi relāciju datu bāzes shēmai, taču šajā gadījumā nākas atteikties no objektorientētas paradigmas piedāvātajam priekšrocībām. Dotās problēmas būtību var izteikt šādi: jā mēs uzvaram viena pusē, tad mēs zaudējam cita pusē [4].

Objektu identitātes nodrošināšana. Objektorientētas programmēšanas valodās katram objektam tiek piešķirt unikāls identifikators, kurš ir derīgs līdz objekta iznīcināšanas brīdim. Dota objektu identitāte nodrošina sekojošas likumsakarības [5]:

- Jā divas objektu atsauces rada uz vienu un to pašu objektu, tad programmas izpildes laikā tās radīs uz vienu un to pašu objekta instanci datora atmiņā;

- Jā ar vienas atsauces palīdzību tiks veiktas izmaiņas objekta stāvoklī, tad arī ar otras atsauces palīdzību būs redzamas objektā veiktas izmaiņas.

Relāciju datu bāzēs identitāte ir raksturīga tikai tabulas ierakstiem, kas tiek panākta ar primāro atslēgu palīdzību. Primāra atslēga satur unikālo lauka vērtību, kura identificē kādu noteiktu tabulas ierakstu un kura nav raksturīga citiem tabulas ierakstiem. Primāras atslēgas izslēdz tabulas ierakstu dublēšanas iespējas, nodrošina tabulas ierakstu kārtošanu un attiecību definēšanu starp datu bāzes tabulām [2].

Problēma rodas objektu identitātes nodrošināšana, ja kāds noteikts tabulas ieraksts tiek atgriezts atkārtoti [3, 5]. Šajā gadījumā tiks izveidoti divi vienādi objekti, kuri reprezentēs vienu un to pašu ierakstu datu bāzes tabulā. Šāda situācija kļūst problemātiska, ja programmas izpildes laikā abas objekta instances tiek modificētas, ka rezultātā nav iespējams viennozīmīgi noteikt, kuru no esošam objektu instancēm glabāt datu bāzes tabulā. Šāda situācija nav vēlama arī dēļ neracionālas datora atmiņas izmantošanas.

Vairākas datu pieejas tehnoloģijas, piemēram, LINQ to SQL vai Hibernate, nodrošina objektu identitātes vadību, t.i., ja tiks atklāts ka pieprasītais objekts ir jau izveidots, objekta atsaucei tiks piešķirta norāde uz jau eksistējošo objekta instanci [3, 5]. Taču šāda objektu identitāte tiek garantēta tikai vienas sesijas (savienojumā ar datu bāzi) ietvaros. Jā tiek plānots izmantot objektus neatkarīgi no sesijas un nodrošināt to identitāti arī citas sesijās, tad ir ieteicams implementēt *GetHashCode* un *Equals* metodes programmēšanas valodā C# vai *equals* un *hashCode* programmēšanas valodā Java [3].

Objektu asociāciju un tabulu attiecību saistīšana. Relāciju datu bāzēs tabulu saistīšana tiek panākta ar attiecību definēšanu starp datu bāzes tabulām. Attiecības nodrošina datu piekļūšanas, rediģēšanas, meklēšanas, izgūšanas u.c. operācijas starp abpusēji saistītām tabulām [2]. Ar to palīdzību tiek veikta ievaddatu kontrole, kas nodrošina korektas informācijas glabāšanu tabulās [2]. Relāciju datu modelī attiecību definēšana ir balstīta uz primāro un sekundāro atslēgu saistīšanu.

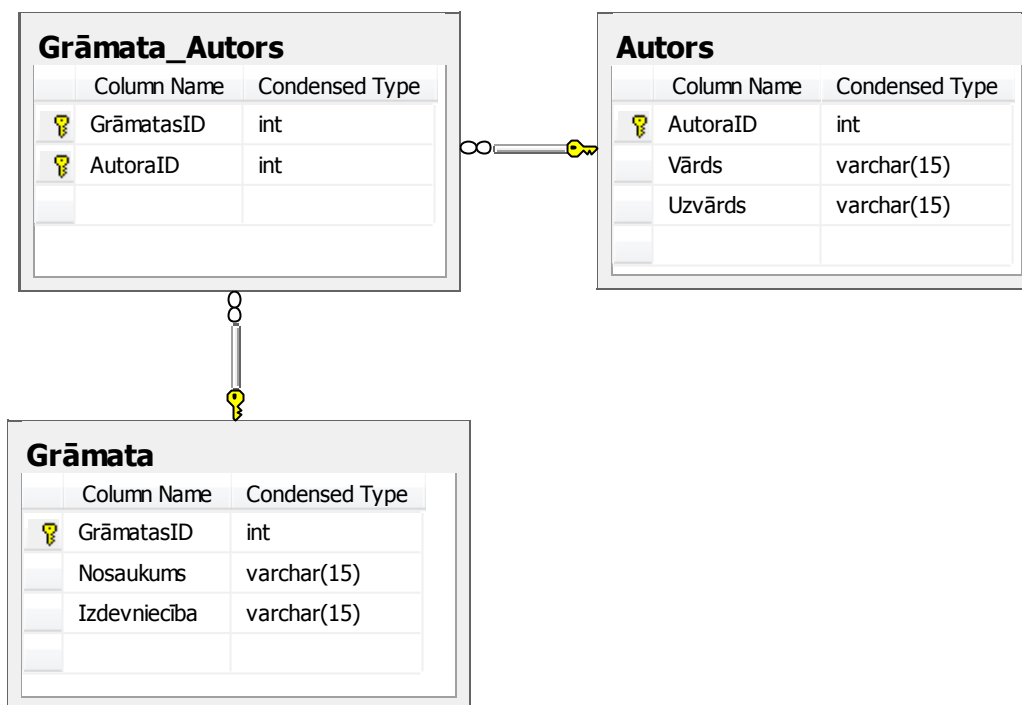
Objektorientētas programmēšanas valodās asociācijas starp objektiem tiek definētas ar atsauču palīdzību [3]. Saskaņā ar šo rodas šādas konstruktīvas pretrunības [3]:

- Objektu atsauces definē asociācijas no viena objekta uz otru (tās ir vienvirziena), taču attiecībām starp datu bāzes tabulām nav raksturīgs kāds noteikts virziens

(izmantojot tabulu apvienošanas un projekcijas pamatoperācijas ir iespējams iegūt vēlamo rezultātu kopu). Lai nodrošinātu abpusēju objektu saistību, ir nepieciešams katra no objekta klasēm definēt atbilstošo asociāciju.

- Relāciju modelī attiecības raksturo savstarpējas priekšmetu (tabulu) saistības. Ir iespējamas attiecības – „viens pret vienu”, „viens pret daudziem”, „daudzi pret daudziem”. Objektorientētas programmēšanas valodās ir iespējams definēt līdzīgas asociācijas ar objektu atsauču un kolekciju palīdzību. Pretrunība rodas attiecības „daudzi pret daudziem” nodrošināšana, kas relāciju datu bāzēs tiek nodrošināta ar papildus tabulas izveidošanu starp saistītām tabulām, taču objektorientētas programmēšanas valodas, piemēram, Java vai C#, šāda attiecība ir pieļaujama, ļaujot sava starpā kombinēt divas saistītas objektu kolekcijas.

Minētas pretrunības var parādīt ar sekojoša piemēra palīdzību. Pieņemsim, ka ir jāizstrādā sistēma, kura veic informācijas par grāmatām un to autoru vadību. Par cik vienai grāmatai var būt vairāki autori un vienam autoram var būt vairākas grāmatas, tad minētiem priekšmetiem ir raksturīga „daudzi pret daudziem” attiecība. Iespējama datu bāzes shēma ir parādīta 1.3 attēlā.



1.3. att. Attiecības „daudzi pret daudziem” realizēšanas piemērs relāciju modelī.

Atšķirība no datu bāzes shēmas objektorientētas programmēšanas valodās ir iespējams realizēt tikai divas atbilstošās klases, realizējot starp tām attiecīgo asociāciju ar kolekciju palīdzību. Šādas realizācijas piemērs programmēšanas valodā C# ir attēlots 1.4 attēlā. Dotajā piemērā asociācijas uzdošana tiek realizēta ar interfeisa *IList* palīdzību (5. un 19. rindkopa).

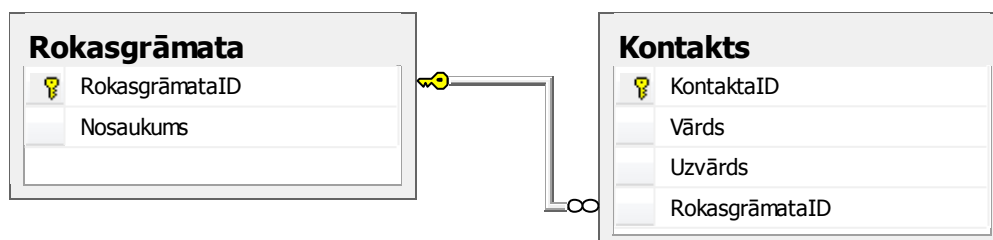
```
01 class Grāmata {
02     private int grāmatasID;
03     private string nosaukums;
04     private string izdevniecība;
05     private IList<Autors> autori;
06
07     public string Nosaukums {
08         get { return this.nosaukums; }
09         set { this.nosaukums = value; }
10     }
11
12     // Citas īpašības, biznes metodes.
13 }
14
15 class Autors {
16     private int autoraID;
17     private string vārds;
18     private string uzvārds;
19     private IList<Grāmata> grāmatas;
20
21     public IList<Grāmata> Grāmatas {
22         get { return this.grāmatas; }
23         set { this.grāmatas = value; }
24     }
25
26     // Citas īpašības, biznes metodes.
27 }
```

1.4. att. Asociācijas „daudzi pret daudziem” realizēšanas piemērs valodā C#.

Objektu ielādes optimizēšana. Relāciju datu bāzēs datu izgūšana no tabulām tiek veikta ar SQL vaicājumu palīdzību, bet objektorientētas programmēšanas valodās pārvietošanas starp objektiem tiek veikta, sekojot pēc to atsaucēm. Piemēram, lai piekļūtu informācijai par autora grāmatu, mēs varam veikt šādu pierakstu `autors.Grāmatas[0].Nosaukums`, kur *Grāmatas* un *Nosaukums* ir klasēs realizētas īpašības, lai nodrošinātu pieeju pie iekapsulētiem klašu atribūtiem. Diemžēl šāds pieejas veids ir problemātisks datu izgūšanas ziņā, jo prasa vairāku vaicājumu izpildi objektu secīgai ielādei [3]. Piemēram, dotajā gadījuma vispirms tiks izpildīts

vaicājums autora instances ielādēšanai atmiņā, tad izmantojot ielādēto objektu tiek realizēta pieeja saistītam objektam, ka rezultātā tiks izpildīts nākošais vaicājums grāmatas instances ielādēšanai u.tml.

Šāds pieejas veids var novests arī pie „ $n + 1$ selects” problēmas, kura izpaužas ar n vaicājumu izpildes nepieciešamību katrai objektu instancei [3]. Piemēram, pieņemsim ka ir divas tabulas „Rokasgrāmata” un „Kontakts” ar attiecību viens pret daudziem (vienai rokasgrāmatai var būt vairāki kontakti). Datu bāzes shēma ir attēlota 1.5 attēlā.



1.5. att. Datu bāzes shēma.

Jā ir nepieciešams izvadīt katras rokasgrāmatas kontaktus, tad saskaņā ar šādu pieeju, vispirms tiek ielādēta rokasgrāmatu objektu kolekcija, tad izmantojot iterāciju tiek apskatīta katra rokasgrāmata atsevišķi, kurai iteratīvi tiek izvadīts katrs atsevišķs kontakts. Minēta uzdevuma risinājuma piemērs ir attēlots 1.6 attēlā.

```

01 using (DbContext db = new DbContext()) {
02     /* Rokasgrāmatu izgūšana no datu bāzēs */
03     IQueryable<Rokasgrāmata> rokasgrāmatas =
04         from r in db.Rokasgrāmatas select r;
05
06     /* Katras rokasgrāmatas kontaktu izvade */
07     foreach (Rokasgrāmata r in rokasgrāmatas)
08         foreach (Kontakts k in r.Kontakts)
09             Console.WriteLine("{0}: {1} {2}",
10                             r.Nosaukums, k.Vārds, k.Uzvārds);
11 }

```

1.6. att. Uzdevuma izpildes piemērs, izmantojot LINQ paplašinājumu.

Dotajā gadījumā tiek izpildīts viens vaicājums rokasgrāmatu kolekcijas ielādei un n vaicājumi katras rokasgrāmatas kontaktu kolekcijas ielādei, kur n ir kopējais rokasgrāmatu skaits. Attēlā 1.6 attēlota pirmteksta izpildes rezultātā ģenerēti SQL vaicājumi ir parādīti 1.7

attēlā, kuri ir iegūti ar utilitātes SQL Server Profiler palīdzību. Dota utilīta pilda SQL Server datu bāzes notikumu pārtveršanu, piedāvājot vaicājumu trasēšanas iespējas iespējamo problēmu cēloņu atklāšanai.

01	SELECT [t0].[RokasgrāmataID], [t0].[Nosaukums]
02	FROM [dbo].[Rokasgrāmata] AS [t0]
03	
04	exec sp_executesql N'SELECT [t0].[KontaktaID], [t0].[Vārds],
05	[t0].[Uzvārds], [t0].[RokasgrāmataID]
06	FROM [dbo].[Kontakts] AS [t0]
07	WHERE [t0].[RokasgrāmataID] = @p0',N'@p0 int',@p0=1
08	
09	exec sp_executesql N'SELECT [t0].[KontaktaID], [t0].[Vārds],
10	[t0].[Uzvārds], [t0].[RokasgrāmataID]
11	FROM [dbo].[Kontakts] AS [t0]
12	WHERE [t0].[RokasgrāmataID] = @p0',N'@p0 int',@p0=2
13	
14	...

1.7. att. Programmas izpildes laikā ģenerēti SQL vaicājumi.

Šādas problēmas ir iespējams atrisināt, veicot datu izgūšanu ar tabulu apvienošanas operāciju palīdzību [3]. Šāda pieeja garantē izpildāmo vaicājumu skaita samazināšanu, kas uzlabo programmas kopējo veikspēju. Taču šādu vaicājumu konstruēšanai vispirms ir jāzina kādus saistītos objektus ir jāielādē līdz ar pieprasīto objektu, t.i., kādiem objektiem programmas izpildes laikā tiks realizēta pieeja.

Objektorientētas paradigmas un relāciju datu modeļa savstarpējais pielietojums rāda vairākas problēmas, kuras ir spiesti risināt programmatūras izstrādātāji. Dotie modeļi balstās uz dažādam koncepcijām, jā objektu modelis ievieš tādas koncepcijas kā mantošana, polimorfisms, abstrakcija u.c., kas ļauj atspoguļot kādas reālas pasaules problēmas sfēru, tad relāciju modelis ir balstīts uz relāciju algebru, attiecību normalizāciju starp tabulām u.tml. Lai atrisinātu vairākas problēmas, kuras ir saistītas ar doto modeļu atšķirībām, ir nepieciešams ieguldīt liela laiku patēriņu un piepūles. Vairākas mūsdienu datu pieejas tehnoloģijas piedāvā vairākus risinājumus doto problēmu novēršana, ļaujot izstrādātājiem koncentrēties tikai uz programmatūras business modeļa izstrādi un uzdevumu veikšanu.

2. DATU PIEJAS TEHNOLOĢIJAS

Mūsdienās .NET Framework un Java 2 Enterprise Edition (J2EE) ir vienas no vadošajām programmatūras izstrādes platformām. Dots platformas piedāvā vairākas tehnoloģijas dažādu komerciālo uzdevumu risināšanai, tanī skaitā arī datu pieejas tehnoloģijas. Šīs tehnoloģijas nodrošina pieeju pie dažādiem datu avotiem un risina vairākus uzdevumus, kuri ir saistīti ar informācijas apmaiņu starp šiem datu avotiem un lietojumprogrammatūru.

2.1. Microsoft datu pieejas tehnoloģijas

Pirms .NET platformas iznākšanas kompānija Microsoft piedāvāja tādas datu pieejas tehnoloģijas kā ODBC, OLE DB, DAO, RDO un ADO. Visām dotām tehnoloģijām bija raksturīgs viens kopējs trūkums: lai strādātu ar kādu noteiktu datu avotu, šīm tehnoloģijām bija nepieciešams patstāvīgi atvērts savienojums, kurš tika aizvērts tikai tad kad bija izpildītas visas nepieciešamas operācijas [6]. Šāds trūkums kļuva par problēmu, kad 90. g. beigās plaši kļuva pieejams globālais tīmeklis. Šādu tehnoloģiju pielietojums Web lietojumos prasīja jauna savienojuma izveidošanu ar datu avotu kartēja lietotāja vajadzībām, kā rezultātā datu bāzu serverim nācās apstrādāt milzīgi daudz pieprasījumu no katra lietotāja pusēs [6]. Šajā gadījumā problemātiski bija arī nodrošināt savstarpējo datu atjaunošanas paralelītāti starp vairākiem lietotājiem, kas, piemēram, ADO gadījumā tika risināts ar ierakstu bloķēšanu datu bāzēs vai esošo funkciju aizliegšanu Web lapās [6]. Citā svarīga problēma, kas rādās līdz ar globāla tīmekļa publisko pieeju ir datu serializēšanas (pārraides) iespēja starp atšķirīgām sistēmām, kuras tika izstrādātas uz atšķirīgo tehnoloģiju pamatā, piemēram, starp sistēmām uz ADO un DCOM un sistēmām uz JDBC un CORBA bāzēs [7]. Šāda problēma radīja standartu ieviešanas nepieciešamību, ka rezultātā tika izstrādāts protokols SOAP un XML [7]. Kaut gan dažās tehnoloģijās, piemēram, ADO, tika ieviests esošo standartu atbalsts, taču tās nav tik pilnvērtīgas, ja dotas tehnoloģijas tiktu izstrādātas sākotnēji doto standartu atbalstām [8].

Līdz ar .NET platformas iznākšanu kompānija Microsoft piedāvāja jauno datu pieejas tehnoloģiju ADO.NET, kura gan risina iepriekšējām tehnoloģijām raksturīgas problēmas, gan pārņem to labākas īpašības. Taču izmantojot doto tehnoloģiju, programmatūras izstrādātāji ir spiesti patstāvīgi rūpēties par savietojamības līmeņa izstrādi starp objektu un relāciju modeļiem.

Kas tiek novērsti dažus gadus vēlāk, kad kļūst pieejamas tehnoloģijas ADO.NET Entity Framework un LINQ to SQL.

2.1.1. Datu pieejas tehnoloģija ADO.NET

Par ADO.NET dēvē klašu un interfeisu kopumu, kuri ir paredzēti darbam ar dažādiem datu avotiem un ir apvienoti viena *System.Data* vārdu telpā [8]. Doto vārdu telpu veido vairākas atsevišķas vārdu telpas, kuras ir uzskaitītas tabulā 2.1 [9].

2.1. tabula

ADO.NET vārdu telpas

Vārdu telpa	Apraksts
<i>System.Data</i>	Satur vispārējas klases un interfeisus
<i>System.Data.Common</i>	Satur abstraktas klases, kuras manto katrs atsevišķs datu gādnieks
<i>System.Data.OleDb</i>	Satur OLE DB datu gādnieka klases
<i>System.Data.ODBC</i>	Satur ODBC datu gādnieka klases
<i>System.Data.SqlClient</i>	Satur SQL Server datu bāzes klienta (datu gādnieka) klases
<i>System.Data.OracleClient</i>	Satur Oracle datu bāzes klienta (datu gādnieka) klases
<i>System.Data.SqlTypes</i>	Satur SQL Server datu tipu realizācijas .NET platformā
<i>System.Data.ProviderBase</i>	Satur datu gādnieka bāzes klasi, kuru implementē citi datu gādnieki
<i>System.Data.Sql</i>	Satur jaunas klases pieejas nodrošināšanai SQL Server datu bāzei

Dotas vārdu telpas ir specifiskas gan ADO.NET 2.0, gan ADO .NET 3.5 versijai.

2.1.1.1. ADO.NET datu gādnieki

Kā uzrāda tabulas 2.1. dati ADO.NET tehnoloģija piedāvā ODBC, OLE DB, SQL Server un Oracle datu gādniekus, kuru realizācijas ir ietvertas atsevišķas vārdu telpās.

ODBC un OLE DB datu gādnieki ir lietojumprogrammu saskarņu (API) ODBC un OLE DB realizācijas .NET platformā. ODBC datu gādnieks dod iespēju izmantot ODBC datu pieejas tehnoloģijas draiverus, bet OLE DB datu gādnieks tehnoloģijas OLE DB datu gādniekus (COM bibliotēkas). ODBC draiveri un OLE DB COM bibliotēkas satur specifisko funkciju kopumu, kurš nodrošina darbu ar noteiktu datu avotu [2]. Dotās tehnoloģijas bija savlaicīgi plaši atbalstītas, t.i., katrs datu bāzu izstrādātājs rūpējās par attiecīga ODBC draivera vai COM

bibliotēkas izstrādi sava produkta atbalstām, kā rezultātā dotas tehnoloģijas spēj darboties ar daudziem datu avotiem [2]. Atšķirība no tehnoloģijas ODBC, OLE DB nodrošina pieeju ne tikai relāciju datu bāzēm, bet arī hierarhiskām datu bāzēm, lietojumprogrammas Excel tabulām, dažādām datnēm u.c. datu avotiem. [2].

SQL Server un Oracle datu gādnieki ir specializēti darbam ar kādu noteiktu datu avotu. SQL Server klienta gadījumā tiek nodrošināta pieeja pie SQL Server datu bāzēm, bet Oracle klienta gadījumā pie Oracle datu bāzēm. Abi datu gādnieki izmanto datu bāzēm specifiskos protokolus, kas ļauj sasniegt samēra augstu datu pārraides veikspēju [8]. Atšķirība no OLE DB un ODBC datu gādniekiem, kuri arī var tikt pielietoti darbam ar šādām datu bāzēm, dotie datu gādnieki ir labāk piemēroti specializēto datu bāzes funkciju izpildei un darbam ar specifiskiem datu bāžu datu tipiem [8].

2.1.1.2. ADO.NET objektu modelis

ADO.NET objektu modeli var iedalīt divu veidu objektos: autonomajos un no savienojuma atkarīgos. Vienīgais izņēmums ir *DataAdapter* objekts, kurš ir starpnieks starp šiem objektiem.

Autonomie objekti reprezentē datu bāžu struktūras, kuros tiek ievietoti datu bāzes dati [8]. Ar doto objektu palīdzību ir iespējams neatkarīgi no datu bāzes apstrādāt tās datus, veikt datu atjaunošanas, kārtošanas u.c. operācijas, kas bieži vien tiek darīts klientā pusē. ADO.NET autonomie objekti ir uzskaitīti tabulā 2.2 [8, 9]. Doto objektu klases ir pieejamās *System.Data* vārdu telpā [9].

2.2. tabula

ADO.NET autonomie objekti

Klase	Apraksts
<i>DataSet</i>	Objekts ir paredzēts datu bāzes shēmas atveidošanai. Tās var saturēt vairākus <i>DataTable</i> objektus, definējot attiecīgas attiecības starp tiem ar objekta <i>DataRelation</i> palīdzību.
<i>DataTable</i>	Objekts reprezentē datu bāzes tabulu, kuru veido vairāki <i>DataRow</i> un <i>DataColumn</i> objekti.
<i>DataRow</i>	Objekts ir tabulas ieraksta (rindas) analogs, kuram atsevišķa lauka vērtības piekļuvei ir paredzēta īpašība <i>Item</i> .

<i>DataColumn</i>	Objekts atbilst tabulas kolonnai, kurš satur informāciju par kolonnas struktūru (kolonnas nosaukumu un datu tipu).
<i>DataRelation</i>	Objekts raksturo attiecības starp diviem <i>DataTable</i> objektiem. Ar tā palīdzību ir iespējams veikt ievaddatu kontroli un caurskatīt saistītos <i>DataRow</i> objektus.
<i>Constraint</i>	Objekts definē ierobežojumus <i>DataColumn</i> objektam. Tās garantē, ka katrs kolonnas lauks saturēs unikālo vērtību vienā <i>DataTable</i> objekta ietvaros.
<i>DataView</i>	Objekts ir skata analogs relāciju datu bāzēs, tas var tikt pielietots datu kārtošanas un filtrēšanas operācijās.

No savienojuma atkarīgie objekti tieši darbojas ar kādu noteiktu datu avotu. Tie nodrošina SQL vaicājumu izpildi un rezultātu kopas atgriešanu, transakciju vadību u.c. operācijas.

Katrs ADO.NET datu gādnieks implementē doto objektu klases konkrētai datu bāzei specifiska veidā. Tās nozīme, ka *System.Data.Common* vārdu telpā ir pieejas doto objektu abstraktās klases, piemēram, *DbConnection*, *DbCommand* u.c., kuras implementē katrs atsevišķs datu gādnieks, lai realizētu šajās klases noteikto kopējo funkcionalitāti katrai datu bāzei vai arī OLE DB un ODBC datu avotiem raksturīga veidā [8]. Šādi, piemēram, savienojuma izveidošanai tieši ar SQL Server datu bāzi ir realizēta *SqlConnection* klase, ar Oracle datu bāzi *OracleConnection* klase, ar OLE DB datu avotu *OleDbConnection* klase u.t.t. Visas šāda tipa objektu klases ir uzskaitītās tabula 2.3, kurā arī ir iekļauts *DataAdapter* objekts, jo tās arī tiek implementēts katram noteiktam datu avotam atsevišķi [8, 9].

2.3. tabula

ADO.NET no savienojuma atkarīgie objekti

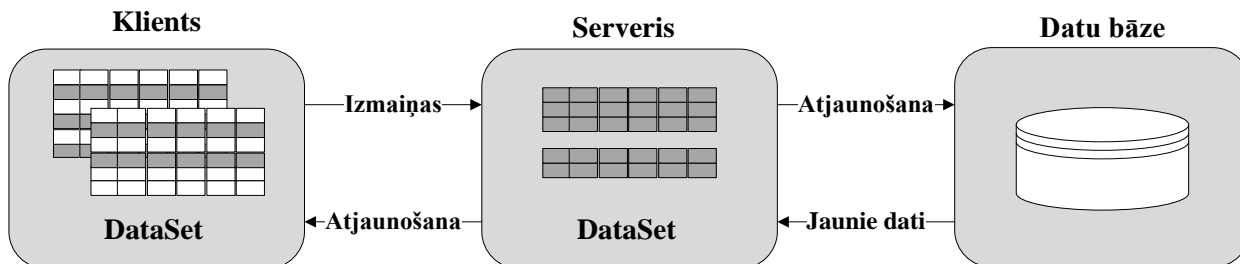
Klases	Apraksts
<i>SqlCommand</i> , <i>OleDbCommand</i> , <i>OracleCommand</i> , <i>ODBCCommand</i>	Objekts nodrošina SQL vaicājumu izpildi, kuri programmatūras pirmtekstā tiek uzdoti kā rakstzīmju virknes. Objekts veic arī glabājamo procedūru izsaukumus.
<i>SqlCommandBuilder</i> , <i>OleDbCommandBuilder</i> , <i>OracleCommandBuilder</i> , <i>ODBCCommandBuilder</i>	Objekts tiek pielietots SQL modifikācijas vaicājumu ģenerēšanai, lai saskaņotu <i>DataSet</i> objekta saturu ar datu bāzes shēmu.

<i>SqlConnection,</i> <i>OleDbConnection,</i> <i>OracleConnection,</i> <i>ODBCConnection</i>	Objekts nodrošina savienojuma izveidošanu ar noteiktu datu avotu un pilda izveidota savienojuma vadību. Savienojuma izveidošanai, objektam ir nepieciešama informācija par datu avotu, pieejas tiesībām u.c.
<i>SqlDataAdapter,</i> <i>OleDbDataAdapter,</i> <i>OracleDataAdapter,</i> <i>ODBCDataAdapter</i>	Objekts ievieto <i>SqlCommand</i> objekta vaicājuma izpildes rezultāta iegūto ierakstu kopu <i>DataTable</i> vai <i>DataSet</i> objektos un nodrošina šajos objektos veikto izmaiņu atjaunošanu datu bāzē.
<i>SqlDataReader,</i> <i>OleDbDataReader,</i> <i>OracleDataReader,</i> <i>ODBCDatareader</i>	Objekts tiek pielietots tabulas ierakstu lasīšanai. Objekts lasa katru nākamo ierakstu bez iespējas atgrieztos uz iepriekšējo ierakstu.
<i>SqlParameter,</i> <i>OleDbParameter,</i> <i>OracleParameter,</i> <i>ODBCParameter</i>	Objekts ļauj iekļaut SQL vaicājumos speciālus parametrus (mainīgos), kuru vērtības var tikt piešķirtas pirms paša vaicājuma izpildes.
<i>SqlTransaction,</i> <i>OleDbTransaction,</i> <i>OracleTransaction,</i> <i>ODBCTransaction</i>	Objekts pilda transakciju vadību, piedāvājot metodes veikto izmaiņu atcelšanai un fiksēšanai.

2.1.1.3. ADO.NET pielietojums paradigmu savietojamības problēmu risināšanā

ADO.NET tehnoloģija tika izstrādāta datu pieejas nodrošināšanai globālajā tīmeklī. Tajā tika ieviests autonomais objektu modelis, ar kuru palīdzību ir iespējams darboties ar datu bāzu datiem un to shēmām neatkarīgi no datu bāzēm. Šajā tehnoloģijā XML valoda tiek pielietota par pamata mehānismu *DataSet*, ka arī atsevišķo *DataTable* objektu saturu serializēšanai starp klientu un serveri [8]. Tā arī nodrošina efektīvu datu atjaunošanas mehānismu, piedāvājot nodod datu bāzei tikai objektos veiktas izmaiņas nevis visu doto objektu saturu [9]. Piemēram, objekta *DataSet* gadījumā, šādas izmaiņas tiks nodotas jaunizveidotajā *DataSet* objektā, kurš var tikt atgriezts atkārtoti klientam ar izmaiņām, kuras tika veiktas datu bāzē bez lietotāja ziņas. Dota

objekta saturs tiks apvienots ar lokāla *DataSet* objekta saturu. Šādu operāciju virkne ir attēlota attēla 2.1.



2.1. att. Datu bāzes un *DataSet* objekta atjaunošanas procedūra [9].

Taču dota tehnoloģija nav piemērota relāciju un objektu datu modeļu savietojamības problēmu risināšanai. Izmantojot doto tehnoloģiju, izstrādātāji ir spiesti tērēt ap 40% no kopēja programmatūras izstrādes laikā, lai risinātu objektu glabāšanas un izgūšanas problēmas, nodrošinātu attiecīgas asociācijas starp objektiem un pārvarētu problēmas, kuras ir saistītas ar relāciju datu bāzes shēmas un programmatūras klašu hierarhijas modeļu atšķirībām [4]. Grāmatu kolekcijas izgūšanas piemērs, izmantojot doto tehnoloģiju, ir attēlots 2.2 attēlā (skatīt 1.3 un 1.4 attēlus).

```

01 List<Grāmata> grāmatas = new List<Grāmata>();
02 using (SqlConnection sqlConn = new SqlConnection("...")) {
03     sqlConn.Open();
04     SqlCommand sqlComm = sqlConn.CreateCommand();
05     sqlComm.CommandText =
06         @"SELECT GrāmatasID, Nosaukums, Izdevniecība
07         FROM Grāmata
08         WHERE Izdevniecība = @Izdevniecība";
09     sqlComm.Parameters.AddWithValue("@Izdevniecība", "ZvaigzneABC");
10
11     using (SqlDataReader sqlReader = sqlComm.ExecuteReader()) {
12         while (sqlReader.Read()) {
13             Grāmata grāmata = new Grāmata();
14             grāmata.GrāmatasID = sqlReader.GetInt32(0);
15             grāmata.Nosaukums = sqlReader.GetString(1);
16             grāmata.Izdevniecība = sqlReader.GetString(2);
17             grāmatas.Add(grāmata);
18         }
19     }
20 }

```

2.2. att. Grāmatu ierakstu izgūšanas piemērs pielietojot tehnoloģiju ADO.NET.

Apskatot attēlā attēloto pirmtekstu var ievērot vairākus trūkumus, kuri ir raksturīgi šādam datu pieejas veidam [4, 10, 11]:

- 1) SQL vaicājumi tiek uzdoti rakstzīmju virknēs, kuras netiek verificētas gan ar moderno izstrādes vides rīku palīdzību, gan arī programmas kompilācijas laikā. Vaicājumos pielaistas kļūdas tiek atklātas vienīgi programmas izpildes laikā, kad vaicājums tiek nodots datu bāzes vadības sistēmai. Šāds pirmteksts ir grūti uzturams, uzlabojams un pārnesams, kas neatbilst mūsdienu programmatūras izstrādes prasībām, piemēram, spējas programmatūras izstrādes (*agile software development*) metodoloģijai. Programmas pirmtekstā šāda SQL vaicājuma rakstzīmes virkne ir attēlota 6., 7. un 8. rindkopā.
- 2) Dotas problēmas ir raksturīgas arī tehnoloģijas piedāvātām metodēm ierakstu vērtību izgūšanai un parametru uzdošanai, jo gan programmēšanas valodas kompilators, gan arī izstrādes vide neveic uzdoto datu tipu atbilstības pārbaudi. Piemēram, jā datu bāzes tabulā *GrāmatasID* laukam tiktu noteikts rakstzīmju virknes datu tips – *varchar*, tad izpildoties metodei *GetInt32* (pārveido izgūta ieraksta lauka vērtību 32 bitu skaitļa vērtībā) kļūdas ziņojums tiktu saņemts tikai programmas izpildes laikā (ierakstu vērtību izgūšana ir attēlota 14., 15. un 16. rindkopā). Doto situāciju var attiecināt arī parametru uzdošanai, kas var radīt problēmas jā attiecīga datu bāzes lauka datu tips vai pat ieraksta kolonnas nosaukums tiktu mainīts (parametru uzdošana ir attēlota 9. rindkopā). Strādājot ar doto tehnoloģiju, programmatūras izstrādātājiem ir precīzi jāzina kādus programmēšanas valodas datu tipus ir nepieciešams pretstatīt atbilstošiem datu bāzes datu tiptiem.
- 3) Vienkārša uzdevuma izpildīšanai ir jāparedz vairākas operācijas: objektu izveidošana, savienojuma atvēršana, komandas objekta piesaiste savienojuma objektam u.tml.

ADO.NET tehnoloģija piespiež programmatūras izstrādātājus galvenokārt darboties ar relāciju datu modeli. Tā piedāvā vairākus relāciju datu bāžu struktūras objektus ar kuru palīdzību ir iespējams apstrādāt un glabāt no datu bāzes izgūtos datus. Datas tehnoloģijas pielietošanai

izstrādātājiem ir jāapgūst ne tikai attiecīgo programmēšanas valodu, bet arī SQL vaicājumu valodu.

2.1.2. Paplašinājums LINQ

LINQ ir .NET programmēšanas valodu paplašinājums, kurš papildina programmēšanas valodas ar SQL vaicājumu valodai raksturīgām konstrukcijām un izmantojot tās ļauj darboties ar dažādiem datu avotiem [9]. LINQ paredz atmiņā esošas objektu kolekcijas iteratīvo apstrādi, nodrošina kārtošanas, grupēšanas, atlases u.c. SQL vaicājumu valodai raksturīgas operācijas. LINQ pēc būtības ir karkass, kuru veido *System.Linq* vārdu telpā noteikts interfeisu un klašu kopums. Galvenās LINQ vaicājumu valodas klases ir [4]:

- *Enumerable* – satur *Select*, *Where*, *OrderBy*, *GroupBy* u.c. metožu realizācijas, kuras darbojas ar interfeisa *IEnumerable<T>* datu avotiem – masīviem un kolekcijām, kuras implementē interfeisā *GetEnumerator* metodi (Šeit T ir kolekcijas vai masīva datu tips). Dota metode nodrošina kolekciju vai masīvu elementu iteratīvo lasīšanu, kuru izmanto *foreach* operators. Klasē realizētas metodes vienmēr atgriež interfeisa *IEnumerable<T>* datu avotu, kas ļauj metodēm savstarpēji sadarboties.
- *Queryable* – satur līdzvērtīgo metožu *Select*, *Where*, *Distinct* u.tml. realizācijas, kuras darbojas ar interfeisa *IQueryable<T>* datu struktūrām (entītiju klašu objektiem). Dotais interfeiss ir atvasināts no *IEnumerable<T>* interfeisa, piedāvājot iespēju iteratīvi lasīt izgūtas objektu kolekcijas elementus. Šeit T ir objekta tips (klases nosaukums).

LINQ vaicājumu valoda tiek pielietota darbam ar dažādiem datu avotiem, pieejā kuriem tiek nodrošināta ar atsevišķo LINQ gādnieku (*LINQ provider*) palīdzību. .NET platformā iekļautie LINQ gādnieki, ka arī to datu avoti ir attēloti 2.3 attēlā.

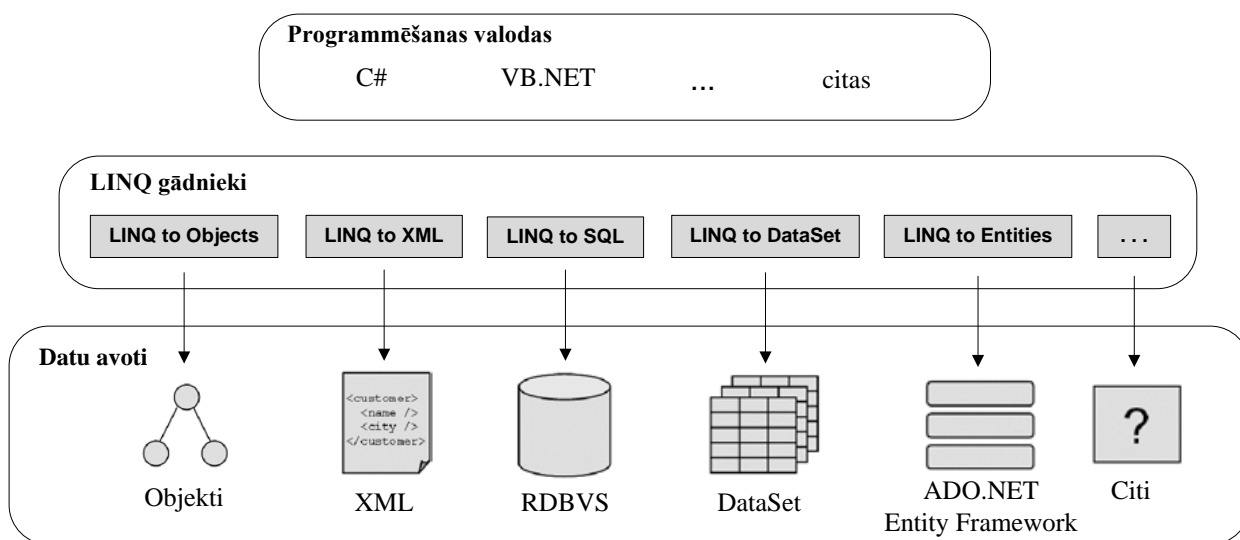
Gādnieks LINQ to Objects reprezentē interfeisu *IEnumerable<T>*. Katra kolekcija vai masīvs, kurš implementē doto interfeisu var tikt apstrādāts ar *Enumerable* klasē noteiktam LINQ vaicājumu valodas metodēm.

Gādnieki LINQ to XML un LINQ to DataSet nodrošina LINQ vaicājumu valodas pielietošanu darbam ar XML datu avotiem un *DataSet* objektā saturu. Dotie gādnieki arī

implementē interfeisu *IEnumerable<T>*, ļaujot darboties ar XML datnes elementiem un *DataSet* ierakstu kopas objektiem līdzvērtīgi objektu kolekcijām [4].

Gādnieks LINQ to SQL nodrošina pieeju SQL Server datu bāzei, pārveidojot LINQ vaicājumus datu bāzes SQL vaicājumu valodas dialektā (T-SQL vaicājumus) [4]. LINQ to SQL paredz arī objekt-relācijas kartēšanas risinājumu, kurš veido abstrakcijas līmeni starp relāciju datu bāzi un objektorientēto lietojumprogrammatūru. Dotais gādnieks implementē interfeisu *IQueryable<T>*, kas ļauj LINQ vaicājumu valodas operatoriem darboties ar dota risinājuma entītijas klašu objektiem [4].

Gādnieks LINQ to Entities ļauj pielietot LINQ vaicājumu valodu darbam ar ADO.NET Entity Framework entītijas klašu objektiem. ADO.NET Entity Framework ir objekt-relācijas kartēšanas tehnoloģija, kura piedāvā vairākus risinājumus objektorientētas paradigmas un relāciju datu modeļa savietojamības problēmu novēršanai. Dota tehnoloģija izmanto vairākus ADO.NET tehnoloģijas datu gādniekus, kas ļauj to pielietot ar dažādam relāciju datu bāzēm.



2.3. att. LINQ gādnieki un to datu avoti [4].

Papildus .NET platformā iekļautiem LINQ gādniekiem, eksistē arī citu izstrādātāju gādnieki, piemēram, LINQ to MySQL, LINQ to Amazon, LINQ to SharePoint u.tml. [9].

LINQ vaicājuma piemērs tiek demonstrēts 2.4 attēlā, kurā tiek izgūtas visas masīva *dienas* nedēļas *dienas*, kuru garums sastāda 9. rakstzīmes. Vaicājuma *from* sadaļā tiek definēts mainīgais *diena*, kurš reprezentē katru masīva *dienas* elementa vērtību. Sadaļās *where* tiek uzdots atbilstošs ierobežojums un *orderby* tiek norādīts rezultātu kārtēšanas veids (ir iespējams

kārtot dilstoša vai augoša secība). *Select* sadaļā nosaka kādas vērtības ir jāiegūst izpildoties vaicājumam un kāda formā tās prezentēt. Dotajā piemēra visiem elementiem tiek uzdots lielo burtu reģistrs. Vaicājums tiks izpildīts tikai *foreach* blokā ietvaros, kad tiks realizēta pieeja vaicājuma iegūtam vērtībām [4]. Šajā gadījumā C# kompilators automātiski izsauks klases *Enumerable* *Where*, *OrderBy*, *Select* metodes, nododot tām atbilstošos parametrus [4].

```
01 String[] dienas = { „Svētdiena”, „Sestdiena”, „Pirmdiena”, „Otrdiena” };
02
03 IEnumerable<string> query =
04     from diena in dienas
05     where diena.Length == 9
06     orderby diena
07     select diena.ToUpper();
08
09 foreach (string s in query) Console.WriteLine(s);
```

2.4. att. LINQ to Objects vaicājuma pielietošanas piemērs.

LINQ ļauj izvairīties no dažādu valodu, piemēram, XML, SQL u.tml., pielietošanas programmas pirmtekstā. Izmantojot LINQ izstrādātājiem vairs nav nepieciešams apgūt Xpath, ADO.NET u.c. līdzekļus darbam ar dažādiem datu avotiem. LINQ piedāvā pilnīgi objekt-orientētu pieeju, kas tiek realizēta ar pašas programmēšanas valodas līdzekļiem.

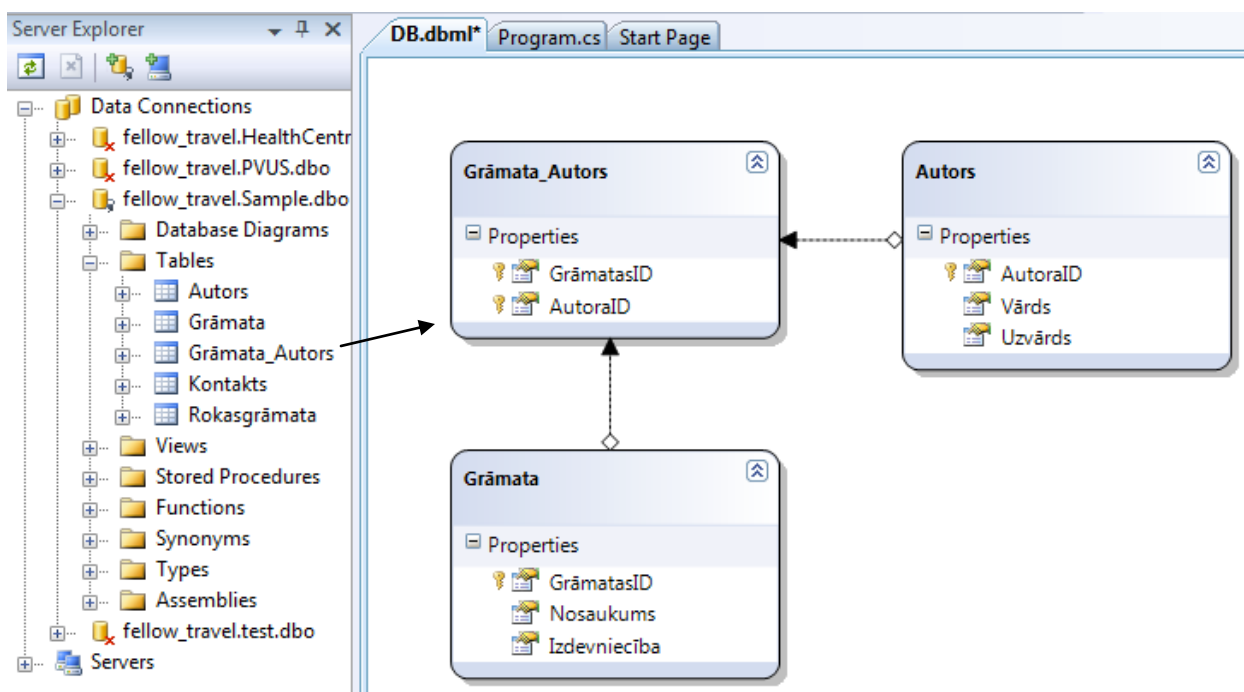
2.1.3. Risinājums LINQ to SQL

LINQ to SQL ir pirmais kompānijas Microsoft objekt-relācijas kartēšanas produkts, kurš nodrošina lietojumprogrammatūras business modeļa projektēšanu, pamatojoties uz klašu (entītiņu) hierarhiju, kuras reprezentē SQL Server datu bāzes tabulas [12]. LINQ to SQL atbalsta tādas relāciju datu bāzes struktūras kā skati un glabājamās procedūras, kā arī pilda transakciju vadību.

2.1.3.1. LINQ to SQL arhitektūra

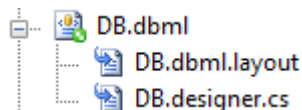
LINQ to SQL arhitektūru veido vizuālais objekt-relācijas kartēšanas konstruktors, kurš abstrahē programmatūras izstrādātājus no objektorientētas paradigmas un relāciju datu modeļu savietojamības problēmām, veicot lielāko daļu no nepieciešama darba divu atšķirīgo modeļu saistīšanai. Dotais konstruktors ir cieši saistīts ar izstrādes vidi Visual Studio. Lai izstrādātu nepieciešamo klašu hierarhiju ir nepieciešams atvērt logu *Server Explorer*, kurš atveido datu bāzes struktūru un konstruktorā ievilkt attiecīgo datu bāzes tabulu. Šāds process tiek attēlots 2.5

attēlā, kurā tiek projektēts klašu modelis, pamatojoties uz attēla 1.3 attēloto datu bāzes struktūru. Tabulas ievilkšanas rezultātā tiek automātiski ģenerēta tabulu reprezentējoša klase un asociācijas starp saistītam klasēm. Konstruktors ļauj uzdot mantošanas attiecības starp klasēm, piedāvā iespējas izstrādāt neatkarīgo klasi un veikt tās saistīšanu ar kādu noteiktu tabulu, ļauj mainīt klašu un to atribūtu pieejas modifikatorus u.tml.



2.5. att. Klašu modeļa projektēšana, izmantojot konstruktoru LINQ to SQL.

LINQ to SQL konstruktoru sastāda trīs atsevišķi komponenti, kuri ir attēloti 2.5 attēlā (šeit DB ir konstruktora nosaukums).



2.6. att. LINQ to SQL konstruktora komponenti.

DB.designer.cs ir LINQ to SQL konstruktora pirmteksta klase, kas tiek izveidota automātiski līdz ar citiem komponentiem. Tā satur vairākas īpašības, kartēšanas atribūtus, konstruktora ģenerēto klašu pirmtekstus, asociāciju deklarējumus un metodes, kuras nosaka mijiedarbšanās veidu ar datu bāzes tabulām [12]. Dota klase tiek atvasināta no *DataContext*

klases, kura nodrošina objektu identitātes un transakciju vadību, seko objektu veiktajām izmaiņām, nodrošina izmaiņu sinhronizēšanu ar datu bāzes tabulām, piedāvā metodes objektu saglabāšanai, dzēšanai, ka arī to saturu atjaunošanai u.tml. [12]. Lai darbotos ar datu bāzi vispirms ir nepieciešams izveidot dota konstruktora klases objektu, kā rezultātā tiks atvērts savienojums ar datu bāzi, tiks nodrošinātas visas iepriekšminētas operācijas, ka arī objektu izgūšanas iespējas ar LINQ vaicājumu valodas palīdzību (skatīt attēlu 1.6). Konstruktorā ģenerētas grāmatas klases pirmteksta piemērs ir attēlots 2.7 attēlā.

```

01 [Table(Name="dbo.Grāmata")]
02 class Grāmata {
03     [Column(DbType="Int NOT NULL", IsPrimaryKey=true)]
04     public int GrāmatasID { get; set; }
05     [Column(DbType="VarChar(15) NOT NULL", CanBeNull=false)]
06     public string Nosaukums { get; set; }
07     [Column(DbType="VarChar(15) NOT NULL", CanBeNull=false)]
08     public string Izdevniecība { get; set; }
09     [Association(ThisKey = "GrāmatasID", OtherKey = "GrāmatasID")]
10     public EntitySet<Grāmata_Autors> Grāmata_Autors { get; set; }
11 }

```

2.7. att. Tabulas „Grāmata” reprezentējošas klases pirmteksts.

Pēc attēla ir redzams, ka tabulas un klases saistīšana tiek panākta ar kartēšanas atribūtu palīdzību. Atribūts *Table* identificē relāciju datu bāzes tabulu, ar kuru tiek saistīta dota klase. Atribūts *Column* identificē klases īpašības (atribūtus), kuri atbilst saistāmas klases kolonnām. Dotais atribūts raksturo kolonu datu tipus, nosaka pieļaujamo lauku izmērus un *null* vērtības piešķiršanas iespējas. Atribūts satur arī citas īpašības, piemēram, *Name* saistāmas kolonnas identificēšanai (dotajā piemēra klases īpašību un kolonnu nosaukumi sakrīt, tāpēc šādas īpašības norādīšana nav obligāta), *IsDbGenerated* primāras atslēgas kolonas lauka vērtības piešķiršanas stratēģijas noteikšanai (nosaka vai laukam tiek piešķirta datu bāzes ģenerēta vērtība) u.tml. [12]. Atribūts *Association* veic klasē definētas asociācijas saistīšanu ar atbilstošo attiecību starp datu bāzes tabulām. Tās norāda primāras atslēgas un saistāmas klases sekundāras atslēgas atribūtus, raksturojot objektu savstarpējo saistīšanas principu. LINQ to SQL neatbalsta „daudzi pret daudziem” attiecību, ģenerējot starpniek tabulas klasi (atbilstoši datu bāzes shēmai). Dotajā piemēra šāda starpniek klase ir *Grāmata_Autors*, ar kuru arī tiek definēta atbilstoša „viens pret daudziem” asociācija. Šeit *EntitySet* ir klase, kura reprezentē saistāmo objektu kolekciju (1.4

attēlā šāda asociācija ir uzdota ar interfeisa *IList* palīdzību). Turpretim, klasē *Grāmata_Autors* tiek uzdota pretēja atsauce uz doto klasi ar klases *EntityRef* palīdzību. Klases *EntitySet* un *EntityRef* pilda saistīto objektu ielādes vadību (nodrošina „atlikta ielādēšanas” stratēģiju) [12].

Komponents *DB.dbml.layout* satur vairākus XML elementus, kuros tiek definēti metadati konstruktora vizuālai attēlošanai.

DB.dbml ir galvenais konstruktora komponents, kas reprezentē lietojumprogrammatūras klašu modelī. Komponenta satura fragments ir attēlots attēla 2.8.

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <Database Name="Sample" Class="DBDataContext"
03   xmlns="http://schemas.microsoft.com/linqtosql/dbml/2007">
04   <Connection Mode="AppSettings"
05     ...
06     ConnectionString="Data Source=FELLOW_TRAVEL;Initial Catalog=Sample;
07       Integrated Security=True"
08     Provider="System.Data.SqlClient" />
09   <Table Name="dbo.Grāmata" Member="Grāmatas">
10     <Type Name="Grāmata">
11       <Column Name="GrāmatasID" Type="System.Int32" CanBeNull="false"
12         DbType="Int NOT NULL" IsPrimaryKey="true" />
13       ...
14     </Type>
15   </Table>
16 </Database>
```

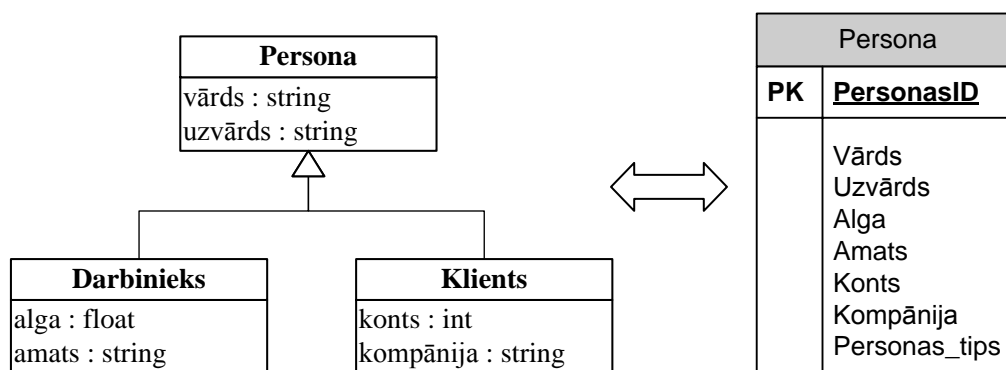
2.8. att. *DB.dbml* XML datnes satura fragments.

Tās satur informāciju par datu bāzi un konstruktora klasi, kura projicē programmas klašu modeli pamatojoties uz dotas datu bāzes shēmu. Šeit tiek norādīts ADO.NET tehnoloģijas datu gādnieks un savienojuma parametru virkne, kuru izmanto gādnieks, lai izveidotu savienojuma ar datu bāzi. ADO.NET datu gādnieks nodrošina LINQ to SQL gādnieka translējamo vaicājumu izpildi un izpildes rezultātā iegūtas ierakstu kopas atgriešanu, kurā tiek kartēta noteiktos klašu objektos vai arī tiek pārveidota anonīmos tipus, jā LINQ vaicājuma ir uzdota projekcija (anonīms tips ir kompilatora ģenerēta klase, kura satur tikai pieprasītos objektu atribūtus). Bez savienojuma iestatījumiem, šeit tiek veikta arī tabulu un klašu saistīšana, izmantojot iepriekš apskatītos kartēšanas atribūtus (LINQ to SQL konstruktors veic tabulu un klašu saistīšanu gan dotajā komponentā, gan arī pirmteksta klasē, kas nav obligāti – bieži vien ir pietiekami definēt kartēšanas atribūtus tikai viena no komponentiem).

2.1.3.2. LINQ to SQL pielietojums paradigmu savietojamības problēmu risināšanā

Kompānija Microsoft pozicionē savu produktu LINQ to SQL kā ātro līdzekli programmu izstrādei, kas ir pielāgots SQL Server datu bāzes funkcionalitātes atbalstām [12]. LINQ to SQL nav paredzēts sarežģīto uzdevumu veikšanai, piedāvājot vienkāršo objekt-relācijas kartēšanas mehānismu, kurā programmatūras modelis tiek projektēts iespējami identiski relāciju datu bāzes shēmai. Dotais risinājums ir paredzēts datu bāžu struktūru atbalstām un ļauj abstrahēties no paradigmu savietojamības problēmām, veicot lielāko daļu no darba divu atšķirīgo modeļu saistīšanai. LINQ to SQL nodrošina [12]:

- Asociāciju un relāciju attiecību kartēšanu. LINQ to SQL atbalsta „viens pret vienu”, „viens pret daudziem” asociācijas, kuras tiek uzdotas pamatojoties uz primāro un sekundāro atslēgu atribūtu vērtībām.
- Mantošanas hierarhijas kartēšanu. LINQ to SQL atbalsta vienīgi „tabula par klašu hierarhiju” (*Table per Class Hierarchy*) mantošanas stratēģiju, kura paredz klašu hierarhijas kartēšanu viena tabulā. Šāda tabula ietver sevī vairākus atvasināto objektu tipus. Katrs atsevišķs objekta tips tiek noteikts, vadoties pēc speciāla tabulas kolonas identifikatora (diskriminatora) vērtības, kurš viennozīmīgi identificē katru objekta tipu. Dotas mantošanas stratēģijas kartēšanas paņēmieni ir attēlots 2.9 attēlā.



2.9. att. Mantošanas kartēšanas stratēģija „tabula par klašu hierarhiju”.

Attēlā parādīta shēmā šāda diskriminatora tabulas kolonna ir *Personas_tips*, vadoties pēc kuras objekt-relācijas kartēšanas tehnoloģija veic attiecīga objekta tipa izgūšanu.

- Objektu identitātes vadību. LINQ to SQL vienas darba sesijas ietvaros uztur informāciju par izgūtiem objektiem, izmantojot kuru pārbauda vai pieprasītais objekts ir izgūts un atrodas atmiņā. Jā tiek konstatēts, ka pieprasītais objekts ir jau izgūts, LINQ to SQL atgriež atsauci uz eksistējošo objekta instanci.
- Objektu ielādes optimizēšanu. LINQ to SQL atbalsta divas objektu ielādēšanas stratēģijas:
 - „Atlikta ielādēšana” (*lazy fetching*) – objekta saistītais (asociētais) objekts vai objektu kolekcija tiek ielādēta tikai tad, kad tiem pirmo reizi tiek realizēta pieeja (veicot objekta izgūšanu tiek izgūts tikai dots objekts). LINQ to SQL izmanto doto stratēģiju pēc noklusēšanas. Tā ļauj darboties tikai ar nepieciešamiem objektiem, optimizējot datora resursus. Taču tai ir raksturīgi arī trūkumi – automātiskais saistīto objektu vai objektu ielādes mehānisms var novest pie „*n + 1 selects*” problēmas. Dots problēma ir apskatīta 1.6 un 1.7 attēlos, kuros vispirms tiek iegūta rokasgrāmatu objektu kolekcija, tad tiek realizēta pieeja katras rokasgrāmatas saistītam kontakta objektam, kā rezultātā tiek izpildīts $1 + n$ vaicājumi. Microsoft terminoloģijā dota stratēģija tiek dēvētā par „kavēto ielādi” (*deffered loading*).
 - „Enerģiska ielādēšana” (*eager fetching*) – objekts tiek ielādēts kopā ar saistīto objektu vai objektu kolekciju, izmantojot SQL apvienošanas (*outer, inner join*) operācijas. LINQ to SQL paredz *DataLoadOptions* klasi, kura ļauj definēt objektu ielādes saistību (kādi saistītie objekti vai objektu kolekcijas tiks ielādētas līdz ar nepieciešamo objektu). Dots klases pielietojums ļauj novērst „*n + 1 selects*” problēmu, jo saistītie objekti tiek izgūti vienā vaicājumā. Klases *DataLoadOptions* pielietojums ir parādīts 2.10 attēlā, kurā līdz ar rokasgrāmatu objektu kolekciju tiek ielādēts arī katras rokasgrāmatas saistītais kontakta objekts. Pēc attēla ir redzams, ka dotas klases objekta norādīta informācija tiek nodota konstruktora pirmteksta klases eksemplāram, kurš vadoties pēc tās zina kādus saistītos objektus ir jāielādē kopā. Microsoft terminoloģijā dota stratēģija tiek dēvētā par „tūlītējo ielādi” (*immediate loading*).

01	<code>DatabaseContext context = new DatabaseContext();</code>
02	<code>DataLoadOptions options = new DataLoadOptions();</code>
03	<code>options.LoadWith<Rokasgrāmata>(r => r.Kontakts);</code>
04	<code>context.LoadOptions = options;</code>
05	
06	<code>List<Rokasgrāmata> rokasgrāmatas = context.Rokasgrāmatas.ToList();</code>

2.10. att. „Energiskas ielādēšanas” stratēģijas realizēšanas piemērs tehnoloģijā LINQ to SQL.

LINQ to SQL risinājums nav adoptētas augstās abstrakcijas līmeņa paraudzēšanai, tās nodrošina datu tipu atbilstības kontroli un paredz pamata risinājumus objektorientētas paradigmas un relāciju modeļa savietojamības problēmu novēršanā.

2.1.4. Risinājums ADO.NET Entity Framework

ADO.NET Entity Framework (turpmāk ADO.NET EF) ir jaunākas kompānijas Microsoft produkts, kas ir adoptēts sarežģīto, uzņēmējdarbības mērogā sistēmu izstrādei [12]. ADO.NET EF arhitektūru veido vairāki abstrakcijas līmeņi, kas ļauj konstruēt sarežģītos klašu hierarhijas modeļus liela mērā abstrahējoties no datu bāzes shēmām. Piemēram, izmantojot ADO.NET EF ir iespējams saistīt vienu klasi ar vairākām tabulām vai skaitiem un pretēji vienu tabulu, skatu ar vairākām klasēm. ADO.NET EF atbalsta vairākas mantošanas stratēģijas un objektu asociācijas, ka arī piedāvā neatkarīgo datu gādnieku modeli, kurš nodrošina pieeju dažādam relāciju datu bāzēm.

2.1.4.1. ADO.NET Entity Framework arhitektūra

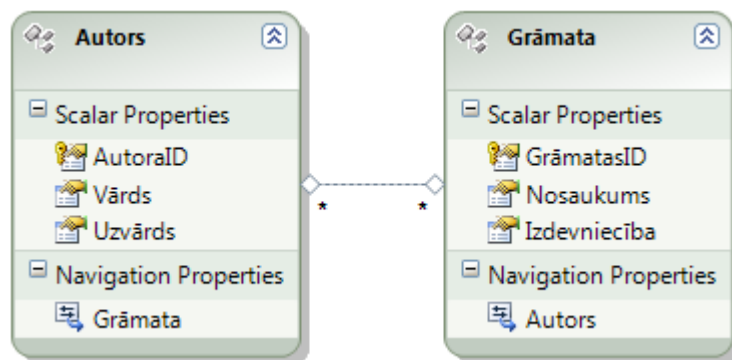
ADO.NET EF arhitektūru veido trīs abstrakcijas līmeņi [9, 12]:

- Loģiskais – tiek uzdots SSDL valodā, kura apraksta relāciju datu bāžu struktūras. Dota valoda satur XML atribūtus, kuri raksturo relāciju tabulu kolonnas un to datu tipus, precizē lauku pieļaujamos izmērus un *null* vērtības piešķiršanas iespējas, nosaka primāro un sekundāro atslēgu kolonnu laukus, ka arī apraksta attiecības starp datu bāzes tabulām. Šeit tiek norādīts savienojuma izveidošanai nepieciešamais ADO.NET tehnoloģijas datu gādnieks.
- Konceptuālais – tiek uzdots ar CSDL valodas palīdzību, kura apraksta lietojumprogrammas klašu hierarhijas modeli. To veido vairāki XML elementi, kuri

raksturo klašu atribūtu īpašības, to datu tipus un asociācijas starp attiecīgo klašu objektiem.

- Kartēšanas – tiek uzdots ar MSL valodas palīdzību, kura saista SSDL un CSDL shēmās uzdotos relāciju un objektu modeļus. Dotajā līmenī tiek nodrošināta mantošanas hierarhijas kartēšana, tabulu kolonnu saistīšana ar klašu atribūtu īpašībām, objektu asociāciju un tabulu attiecību kartēšana.

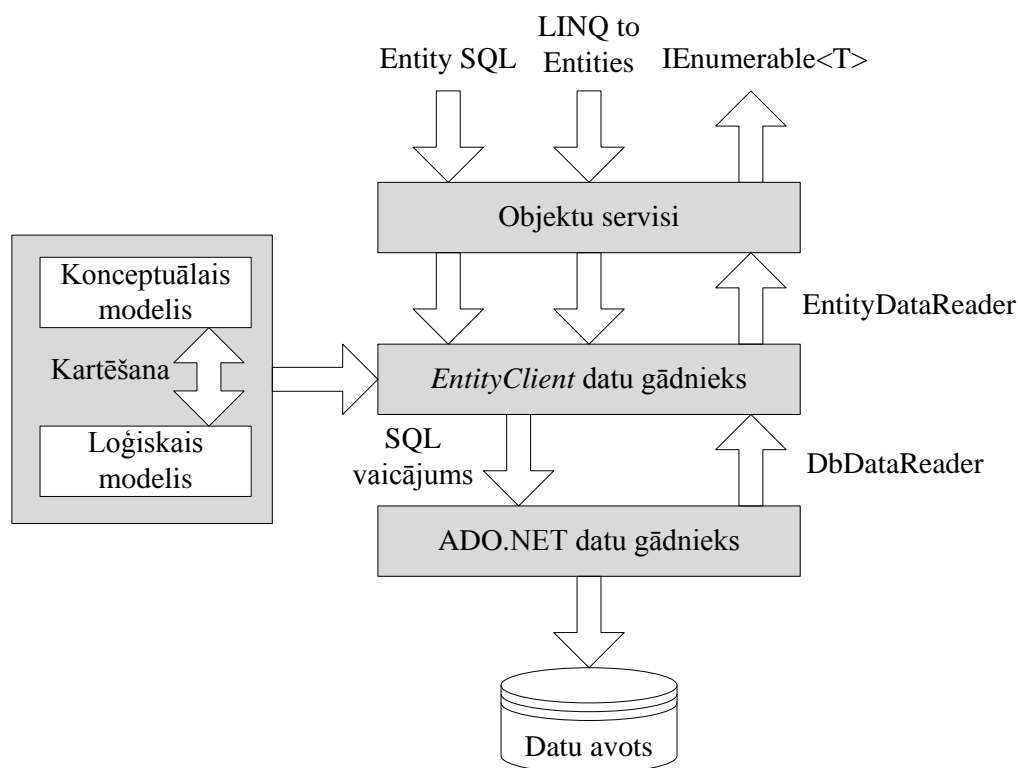
Doto valodu elementi tiek ietverti vienā XML datnē ar paplašinājumu *edmx*, kura arī veido ADO.NET EF vizuālo objekt-relācijas kartēšanas konstruktoru. Dotais konstruktors ir attēlots 2.11 attēlā, kurā tiek projektēts lietojumprogrammas klašu modelis, pamatojoties uz attēla 1.3 attēloto datu bāzes struktūru.



2.11. att. Klašu modeļa projektēšana, izmantojot konstruktoru ADO.NET EF.

ADO.NET EF konstruktors satur pirmteksta klasi, kurā tiek izvietoti ģenerēto entītiju klašu pirmteksti (klases, kuras reprezentē datu bāzes tabulas un tiek aprakstītas ar CSDL valodu), konstruktori, kuri nodrošina dotas pirmteksta klases eksemplāra inicializēšanu u.tml. Dota klase tiek atvasināta no *ObjectContext* klases, kura nodrošina savienojuma izveidošanu ar datu avotu, pilda objektu identitātes un transakciju vadību, seko objektos veiktām izmaiņām un nodrošina šo izmaiņu sinhronizēšanu ar datu bāzes tabulām, piedāvā objektu saglabāšanas, izgūšanas, atjaunošanas u.c. iespējas (veido objektu glabāšanas un izgūšanas kontekstu) [12]. Lai darbotos ar datu bāzi, izstrādātājiem ir jāparedz konstruktorā pirmteksta klases (*ObjectContext* klases apakšklases) eksemplāra izveidošanu. Saskaņā ar šo tiks izveidots abstrakcijas līmenis, piedāvājot iespēju izstrādātājiem darboties ar entītiju klašu objektiem un paredzot visu veikto darbību saskaņošanu ar datu bāzes saturu.

ADO.NET EF atbalsta LINQ vaicājumu valodu, piedāvājot atsevišķo LINQ to Entities gādnieku un ievieš jauno objektorientēto vaicājumu valodu Entity SQL jeb ESQL. Izmantojot dotas objektorientētas vaicājumu valodas ir iespējams izgūt un darboties ar entītijū klāšu objektiem. Šādi vaicājumu tiek translēti SQL vaicājuma valodā ar atsevišķa ADO.NET EF arhitektūra iekļauta *EntityClient* datu gādnieka palīdzību, kas tiek darīts pamatojoties uz abstrakcijas līmeņos aprakstīto informāciju. Kopēja ADO.NET EF arhitektūra ir attēlota 2.12 attēlā.



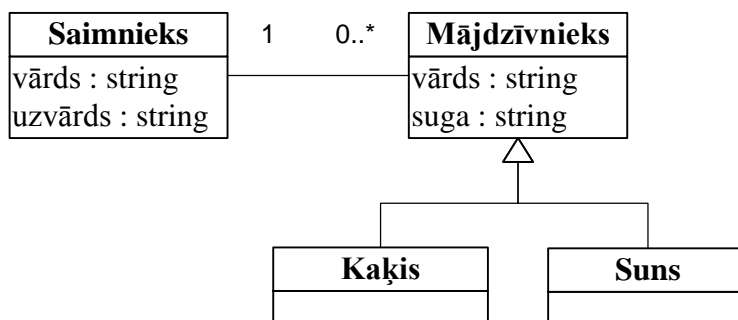
2.12. att. ADO.NET EF arhitektūra [13].

Pēc attēla ir redzams, ka LINQ, ESQL vaicājumi izpildās objektu servisu kontekstā, kuru veido *ObjectContext* klase, entītijū klāšu objekti u.c. komponenti, kuri sastāda abstrakcijas līmeni [14]. Šādi vaicājumi tiek nodoti *EntityClient* datu gādniekam, kuri tiek pārveidoti SQL vaicājumos un nodoti ADO.NET datu gādniekam. Pēc SQL vaicājumu izpildes iegūta ierakstu kopa tiek atgriezta *DbDataReader* implementējamās klases veidolā, piemēram, *SqlClient* gadījumā šāds objekts ir *SqlDataReader*. Šāda ierakstu kopa augstākajā līmenī tiek pārveidota par interfeisa *IEnumerable<T>* objektu kolekciju.

2.1.4.2. ADO.NET Entity Framework pielietojums paradigmu problēmu risināšanā

Kompānija Microsoft pozicionē ADO.NET EF tehnoloģiju kā bagātināto līdzekli relāciju un objektorientētas paradigmas savietojamības problēmu novēršanai, kas var tikt pielietots darbam ar dažādam relāciju datu bāzēm [12]. ADO.NET EF nodrošina [9, 12]:

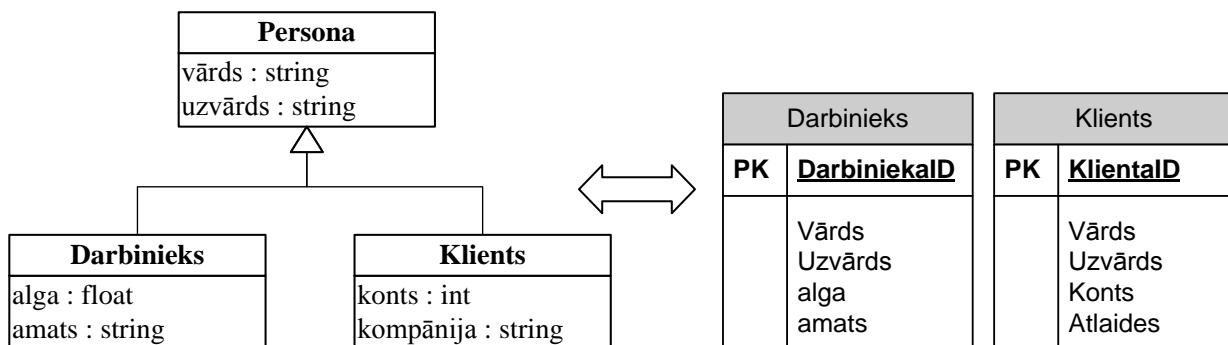
- Objektu asociāciju un tabulu attiecību kartēšanu. ADO.NET EF atbalsta „viens pret vienu”, „viens pret daudziem” un „daudzi pret daudziem” asociācijas, ka arī ļauj uzdot tām polimorfisma attiecības. Šādas asociācijas tiek dēvētas par „polimorfiskām asociācijām” (*polymorphic associations*), kuras ļauj objektam veidot asociāciju ar abstraktas klases apakšklases objektu [3]. Piemēram, 2.13 attēlā attēlotajā klašu hierarhijā ADO.NET EF nodrošinās iespējas piekļūt pie saimnieka objekta saistītam mājdzīvnieku klases apakšklašu instancēm. Pati mājdzīvnieku klase ir abstrakta, kurai nevar tikt izveidota instance.



2.13. att. Klašu hierarhijas piemērs.

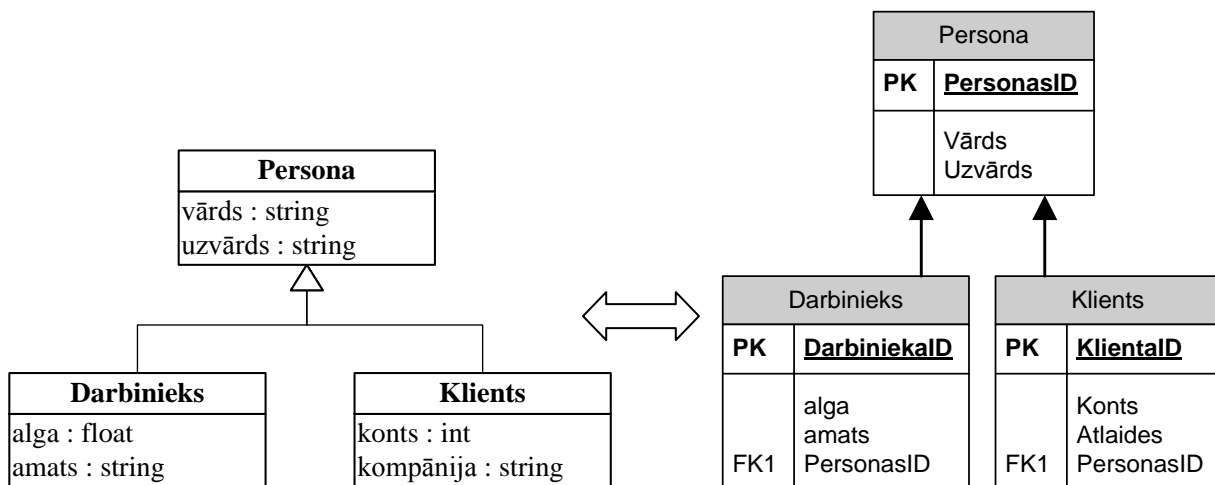
- Objektu identitātes vadību. ADO.NET EF objektiem ir raksturīga *EntityKey* īpašība, kas satur primāras atslēgas lauka vērtību. ADO.NET EF veic objektu identitātes vadību, vadoties pēc doto īpašību vērtībām. Jā tiek noskaidrots, ka pieprasītais objekts ir jau izgūts, tiek atgriezta doto objekta izveidota instance.
- Mantošanas hierarhijas kartēšanu. ADO.NET atbalsta trīs galvenās mantošanas kartēšanas stratēģijas:
 - „Tabula par konkrēto klasi” (*Table per Concrete Class*). Stratēģija paredz atvasināmo klašu kartēšanu atsevišķajās tabulas (katra atsevišķa tabula tiek saistīta ar kādu noteiktu atvasināto klasi). Dotajā stratēģijā abstrakta

bāzes klase netiek kartēta. Šīs stratēģijas kartēšanas paņēmieni ir attēlots 2.14 attēlā.



2.14. att. Mantošanas kartēšanas stratēģija „tabula par konkrēto klasi”.

- „Tabula par klasi” (*Table per Class*). Stratēģija paredz visu klašu kartēšanu atsevišķās tabulās. Dotās mantošanas stratēģijas kartēšanas paņēmieni ir attēlots 2.15 attēlā. Pēc attēla ir redzams, ka starp bāzes klases tabulu un apakšklašu tabulām tiek paredzēta „viens pret viens” attiecība, kas ļauj izgūt apakšklases objekta instanci, izmantojot apvienošanas operācijas.



2.15. att. Mantošanas kartēšanas stratēģija „tabula par klasi”.

- „Tabula par klašu hierarhiju”. Dota mantošanas stratēģija ir aplūkota jau iepriekš un ir attēlota 2.9 attēlā.

- Objektu ielādes optimizēšanu. ADO.NET EF līdzīgi LINQ to SQL tehnoloģijai atbalsta „atliktas ielādēšanas” un „enerģiskas ielādēšanas” objektu izgūšanas stratēģijas. Taču atšķirība no LINQ to SQL, ADO.NET EF pēc noklusējuma neveic ielādēta objekta saistītas objektu kolekcijas vai objekta ielādēšanu, kad tiem tiek realizēta pieeja. Šāds paņēmieni ļauj izvairīties no iespējamam veiktspējas problēmām, kuras var rasties, piemēram, „*n + 1 selects*” problēmas gadījumā. Lai realizētu šādu iespēju, ir nepieciešams vai nu *ObjectContext* klases īpašībai *LazyLoadingEnabled* piešķirt vērtību *true*, vai arī izmantot metodi *Load*, kas veikts norādīto saistīto objektu ielādēšanu. Savukārt, „enerģiskas ielādēšanas” stratēģija ADO.NET EF tehnoloģijā tiek realizēta ar metodes *Include* palīdzību, kas norāda kādos saistītos objektus ir jāielādē līdz ar izgūstāmo objektu. Doto metožu pielietošanas piemērs ir attēlots 2.16 attēlā.

```
01 using (EntitiesContext db = new EntitiesContext()) {
02     List<Grāmata> grāmatas = db.Grāmata.ToList();
03     //Metodes Load pielietošanas piemērs
04     foreach (Grāmata grāmata in grāmatas) {
05         if (!grāmata.Autors.IsLoaded)
06             grāmata.Autors.Load();
07         //...
08     }
09     // "Enerģiskas ielādēšanas" stratēģijas piemērs
10     IQueryable<Autors> autors = db.Autors.Include("Grāmata");
11 }
```

2.16. att. Objektu ielādes piemērs tehnoloģijā ADO.NET EF.

ADO.NET EF tehnoloģija ir paredzēta sarežģīto sistēmu izstrādei, kurās ir jāparedz augstās abstrakcijas līmenis. Tā piedāvā vairākus risinājumus problēmu novēršanai, veic datu tipu atbilstības pārbaudi un ļauj pielietot glabājamās procedūras objektu glābšanai, dzēšanai un tabulu ierakstu atjaunošanai [12].

2.2. Java platformas datu pieejas tehnoloģijas

2.2.1. Datu pieejas tehnoloģija JDBC

JDBC ir standarta datu pieejas tehnoloģija darbam ar relāciju datu bāzēm Java lietojumprogrammās. Dota lietojumprogrammas saskarne tika izstrādāta kompānijā Sun Microsystems un pirmo reizi tika iekļauta sastāvā JDK 1996. gadā. JDBC apvieno sevī labākas Java platformas īpašības, tā ir platformneatkarīga saskarne, kura spēj darboties dažādās sistēmās [15].

2.2.1.1. JDBC draiveru tipi

Tehnoloģijas JDBC datu pieejas mehānisms ir balstīts uz tehnoloģijas ODBC datu pieejas principu, tā arī satur savu draiveru menedžera realizāciju, kurš pilda attiecīgo JDBC draiveru ielādēšanu. Katrs JDBC draiveris pieder konkrētam kompānijas Sun Microsystems noteiktām draiveru tipam, kuri ir aprakstīti tabulā 2.4 [2, 15].

2.4. tabula

JDBC draiveru tipi

Draivera tips	Apraksts
Pirmais draivera tips	JDBC/ODBC tilta draiveri nodrošina iespēju izmantot ODBC draiverus Java lietojumprogrammās. Dotie draiveri novērš savietojamības problēmas starp ODBC draiveriem, kuri tiek rakstīti C/C++ programmēšanas valodās un Java programmēšanas valodu, paredzot draiveros realizēto funkciju izsaukšanu ar JDBC saskarnes palīdzību.
Otrais draivera tips	Draiveri nodrošina iespējas strādāt ar datu bāžu specifiskām lietojumprogrammas saskarnēm, piemēram, ar Oracle datu bāzes lietojumprogrammas saskarni darbam ar Oracle datu bāzi.
Trešais draivera tips	Draiveri izmanto neatkarīgo no datu bāzēm protokolu, kurš veic JDBC pieprasījumu translēšanu konkrētai datu bāzes vadības sistēmai. Šāds pieejas veids nodrošina klienta sadarbšanās iespējas tīmeklī ar vairākiem servera datu avotiem.
Ceturtais draivera tips	Draiveri pilda JDBC pieprasījumu translēšanu specifiskām datu bāzes vadības sistēmas protokolam. Šāds datu pieejas veids nodrošina klienta tiešo sadarbšanos tīmeklī ar datu bāzes vadības sistēmu.

Trešā un ceturtā tipa JDBC draiveri bieži vien tiek pielietoti Java Web sīklīkietotņu (*applet*) un serversīklīkietotņu (*servlet*) izstrādē. Dotie draiveri izmanto tīkla protokolus, kuri nodrošina pieeju gan lokālam, gan arī attālinātām datu avotam. 1. un 2. tipa JDBC draiveri nodrošina pieeju tikai lokālam datu avotam, kurš atrodas uz viena datora ar Java programmu.

2.2.1.2. Tehnoloģijas JDBC objektu modelis

JDBC lietojumprogrammas saskarni veido vairākas klases un interfeisi, kuri ir noteikti *java.sql* pakotnē. Dotas pakotnes interfeisi nosaka kopējo funkcionalitāti, kuru implementē katrs atsevišķs JDBC draiveris konkrētai datu bāzei specifiska veidā. Galvenie tehnoloģijas JDBC interfeisi ir apkopoti tabula 2.5 [15, 16].

2.5. tabula

Tehnoloģijas JDBC interfeisi

Interfeiss	Implementējamās klases apraksts
<i>Driver</i>	Satur metodes savienojuma izveidošanai ar datu avotu, kuras izsauc draiveru menedžeris (<i>DriverManager</i> klase <i>java.sql</i> pakotnē) pēc attiecīga JDBC draivera ielādes
<i>Connection</i>	Pilda savienojuma un transakciju vadību, nodrošina SQL vaicājumu izpildi un ierakstu kopas atgriešanu.
<i>Statement</i>	Satur metodes SQL vaicājumu izpildei. Dotas metodes nodrošina datu izgūšanas, atjaunošanas, ievietošanas un dzēšanas operācijas.
<i>PreparedStatement</i>	Nodrošina SQL vaicājumu izpildi, kuros tiek definēti parametri. Dota klase ir atvasināta no <i>Statement</i> bāzes klases.
<i>CallableStatement</i>	Nodrošina glabājamo procedūru izsaukumus. Dota klase ir atvasināta no <i>PreparedStatement</i> klases.
<i>ResultSet</i>	Nodrošina ierakstu glabāšanas un lasīšanas iespējas. Dotajā objekta tiek ievietota vaicājuma izpildes rezultātā iegūta ierakstu kopa.
<i>ResultSetMetaData</i>	Sniedz informāciju par objektā <i>ResultSet</i> ietverto ierakstu kolonnu tipiem un īpašībām.
<i>ParameterMetaData</i>	Sniedz informāciju par objekta <i>PreparedStatement</i> parametru īpašībām un to tipiem.
<i>DatabaseMetaData</i>	Sniedz informāciju par datu bāzi.

JDBC datu pieejas tehnoloģija līdzīgi ADO.NET tehnoloģijai piedāvā autonomo objektu modeli. Doto modeli veido *javax.sql.rowset* pakotnē iekļautie interfeisi, kura tika ieviesta JDBC 2.0 versijā. Dotie interfeisi manto *Rowset* interfeisā noteiktas metodes, kurš, savukārt, tiek atvasināts no *ResultSet* interfeisa. Kompānija Sun Microsystems paredz doto interfeisu bāzes implementācijas (klases), kuras ir ietvertas atsevišķā *com.sun.rowset* pakotnē. Kaut gan vairākos gadījumos datu bāzes izstrādātāji ir spiesti izstrādāt savas klases, lai realizētu viņu produktam raksturīgo funkcionalitāti. Dotie interfeisi ir apkopoti tabula 2.6 [15, 17].

2.6. tabula

Tehnoloģijas JDBC autonomie interfeisi

Interfeiss	Implementējamās klases apraksts
<i>CachedRowSet</i>	Objekts ir paredzēts ierakstu kopas autonomai glabāšanai, piedāvājot datu modifikācijas operācijas. Tās arī paredz metodes, ar kuru palīdzību objektā veiktas izmaiņas tiek saglabātas datu bāzē.
<i>WebRowSet</i>	Objekts reprezentē autonomu ierakstu kopu, kurš var tikt saglabāts XML datnē. Dota datne var tikt nodota citām lietojumprogrammas komponentam un ielasīta ar citu <i>WebRowSet</i> objektu.
<i>FilteredRowSet</i>	Dota objekta klase ir atvasināta no <i>CachedRowSet</i> klases, kura papildus nodrošina datu atlasēšanas operācijas.
<i>JoinRowSet</i>	Objekts definē attiecības starp diviem interfeisa <i>RowSet</i> objektiem, piemēram, <i>CachedRowSet</i> objektiem.

Ar doto objektu palīdzību ir iespējams nodod vaicājuma izpildes rezultātā iegūto ierakstu kopu kādai citai iekārtai, piemēram, mobilam telefonam, vai arī citām programmas līmenim. Dots operācijas nav iespējams veikt ar *ResultSet* objektu, par cik dotais objekts ir saistīts ar savienojumu.

2.2.1.3. JDBC pielietojums paradigmu savietojamības problēmu risināšanā

JDBC datu pieejas tehnoloģijai ir raksturīgas ADO.NET tehnoloģijas problēmas. Izmantojot doto tehnoloģiju programmatūras izstrādātāji ir spiesti patstāvīgi rūpēties par pirmteksta izstrādi objektorientētas paradigmas un relāciju modeļa savietojamības problēmu novēršanai. Šāda darba veikšana prasa milzīgi liela laika patēriņu, kura rezultātā iegūtais

risinājums ne vienmēr ir pilnvērtīgs – grūti pielāgojams izmaiņām [3]. Turklāt tehnoloģijas JDBC datu pieejas pirmtekstam ir raksturīgi ADO.NET tehnoloģijas trūkumi. Grāmatu ierakstu izgūšanas piemērs, izmantojot doto tehnoloģiju, ir parādīts 2.17 attēlā (datu bāze ir attēlota 1.3 attēlā).

```
01 String connectionString = "jdbc:sqlserver://localhost:1433;" +
02     "databaseName=Sample;integratedSecurity=true";
03 Connection con = null;
04 PreparedStatement stmt = null;
05 ResultSet rs = null;
06 Set<Grāmata> grāmatas = new HashSet<Grāmata>();
07
08 try {
09     Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
10     con = DriverManager.getConnection(connectionString);
11
12     String SQL = "SELECT GrāmatasID, Nosaukums, Izdevniecība " +
13         "FROM dbo.Grāmata " +
14         "WHERE Izdevniecība = ?";
15
16     stmt = con.prepareStatement(SQL);
17     stmt.setString(1, "ZvaigzneABC");
18     rs = stmt.executeQuery();
19
20     while (rs.next()) {
21         Grāmata grāmata = new Grāmata();
22         grāmata.setGrāmatasID(rs.getInt(1));
23         grāmata.setNosaukums(rs.getString(2));
24         grāmata.setIzdevniecība(rs.getString(3));
25         grāmatas.add(grāmata);
26     }
27 }
28 catch (Exception e) {
29     e.printStackTrace();
30 }
31 finally {
32     if (rs != null) try { rs.close(); } catch (Exception e) {}
33     if (stmt != null) try { stmt.close(); } catch (Exception e) {}
34     if (con != null) try { con.close(); } catch (Exception e) {}
35 }
```

2.17. att. Grāmatu ierakstu izgūšanas piemērs pielietojot tehnoloģiju JDBC.

Pēc attēla ir redzams, ka uzdevuma izpildei ir nepieciešams veikt vairākus soļus [2, 15]:

- Izpildīt JDBC draivera ielādēšanu (skatīt 9. rindkopu). Dotajā piemērā tiek pielietots kompānijas Microsoft JDBC draiveris, kurš nodrošina pieeju SQL Server datu bāzei. Draivera ielādēšana tiek panākta ar metodes *forName* palīdzību, kuras izpildes rezultāta tiek izveidota jauna JDBC draivera instance.
- Izveidot savienojumu ar datu avotu (skatīt 10. rindkopu). Savienojuma izveidošana ar datu avotu tiek panākta ar draiveru menedžera (*DriverManager* klases) metodes *getConnection* palīdzību. Dota metode izsauc JDBC draivera klases *Driver* metodi *connect*, kura pārbauda uzdoto datu bāzes savienojuma parametru virkni (URL) un jā izdodas izveidot savienojumu ar datu avotu, atgriež savienojuma (*Connection*) objektu. Dota piemēra šāda datu bāzes savienojuma parametru virkne ir uzdota atsevišķa *connectionURL* mainīgajā.
- SQL vaicājumu izpilde (skatīt no 16. - 18. rindkopai). SQL vaicājuma izpildei ir nepieciešams izveidot *PreparedStatement* objektu, kuram tiek piešķirts parametrs, kas kalpo par kritēriju grāmatu ierakstu atasei. Vaicājuma izpildes rezultāts tiek piešķirts *ResultSet* objektam.
- Ierakstu kopas apstrāde (skatīt no 20. – 25. rindkopai). Ierakstu lasīšana tiek veikta *while* ciklā ar objekta *ResultSet* metodes *next* palīdzību (pārveidojas starp ierakstiem). Ciklā ietvaros tiek izveidots jauns grāmatas objekts, kura atribūtiem tiek piešķirtas objekta *ResultSet* pārveidojumu metožu *getString*, *getInt* iegūtas vērtības. Vērtību piešķiršana tiek panākta ar objekta klasē realizētam piešķiršanas īpašībām – *setGrāmatasID* u.tml. Tad objekts tiek ievietots grāmatu kolekcijā.

Salīdzinot doto pirmtekstu ar ADO.NET pirmtekstu šāda uzdevuma veikšanai (skatīt 2.2 attēlu) var ievērot, ka abos gadījumos uzdevuma izpildes pirmteksts ir gandrīz identisks. Abos gadījumos SQL vaicājumi arī tiek uzdoti rakstzīmju virknēs, tiek uzdotas metodes ierakstu vērtību datu tipu pārveidošanai u.tml. Šāds pirmteksts tieši darbojas ar datu avotu, kas nevar tikt verificēts ar programmēšanas izstrādes vidēs palīdzību. Tās ir nedrošs, jo pieļauj kļūdaino operāciju izpildi. Nelielos projektos šāda problēma nav būtiska, jo bieži vien programmatūras izstrādātājs un datu bāzes projektētājs ir viena persona, taču lielas komandās šāda pirmteksta uzturēšana prasa papildus disciplīnas ieviešanu, kas atvieglots pirmteksta atjaunošanu atbilstoši datu bāzes shēma veiktajām izmaiņām, kuras tiek veiktas ar kādu noteiktu personu [11].

2.2.2. Objekt-relācijas kartēšanas tehnoloģija Hibernate

Hibernate ir platformas Java objekt-relācijas kartēšanas risinājums, kurš nodrošina datu reprezentēšanu starp objektu un relāciju datu modeļiem. Hibernate pilda objektu klāšu saistīšanu ar relāciju datu bāzes tabulām, veic datu tipu atbilstības pārbaudi un nodrošina datu saglabāšanas, atjaunošanas, izgūšanas u.c. operācijas. Tās ir atvērts produkts, kurš veido abstrakcijas līmeņi starp relāciju datu bāzēm un objektorientēto lietojumprogrammatūru, ļaujot programmatūras izstrādātājiem koncentrēties tikai uz programmatūras biznesa loģikas uzdevumu veikšanu. Dota tehnoloģija ir pieejama arī .NET platformas izstrādātājiem kā atsevišķa versija ar nosaukumu NHibernate.

2.2.2.1. Hibernate arhitektūra

Tehnoloģija Hibernate nosaka vairākus atribūtus, kuras raksturo klašu un relāciju tabulu saistīšanas tehniku. Atribūti satur informāciju, kura nosaka kādos tabulas laukos ir nepieciešams veikt objektu atribūtu saglabāšanu, raksturo attiecības starp datu bāžu tabulām, definē objektu ielādes stratēģiju u.tml. [3]. Šādi atribūti tiek uzdoti atsevišķas XML kartēšanas datnēs, kurām ir raksturīgs hbm.xml paplašinājums. Attēlā 2.19 ir parādīts šādas datnes piemērs, kura veic grāmatu tabulas saistīšanu ar atbilstoša objekta klasi (datu bāzes shēma ir parādīta 1.3 attēlā). Savukārt, attiecīga POJO (*Plain Old Java Object*) klase ir attēlota 2.18 attēlā.

```
01 public class Grāmata {
02     private int grāmatasID;
03     private String nosaukums;
04     private String izdevniecība;
05     private Set<Autors> autori = new HashSet<Autors>();
06
07     public Grāmata() {}
08
09     public int getGrāmatasID() {
10         return grāmatasID;
11     }
12
13     public void setGrāmatasID(int grāmatasID) {
14         this.grāmatasID = grāmatasID;
15     }
16
17     // Citas īpašības un business metodes.
18 }
```

```

19     public boolean equals(Object o) {
20         if (this == o) return true;
21         if (!(o instanceof Grāmata)) return false;
22         Grāmata grāmata = (Grāmata)o;
23         if (!this.getNosaukums().equals(grāmata.getNosaukums()))
24             return false;
25         if (!this.getIzdevniecība().equals(grāmata.getIzdevniecība()))
26             return false;
27         return true;
28     }
29
30     public int hashCode() {
31         int result = 17;
32         result = 37 * result + getNosaukums().hashCode();
33         result = 37 * result + getIzdevniecība().hashCode();
34         return result;
35     }
36 }

```

2.18. att. Grāmatas klases realizācijas piemērs valodā Java.

Pēc attēlā ir redzams, ka klase satur tabulas kolonam raksturīgus atribūtus, kuriem ir uzdotas īpašības. Klasē ir realizēta asociācija ar autora klasi, kura reprezentē autoru tabulu datu bāzē (skatīt 5. rindkopu). Par cik starp tabulām ir uzdota „daudzi pret daudziem” attiecība, klasē tiek definēta saistīto autoru objektu kolekcija (līdzvērtīga asociācija tiek definēta arī autora klasē). Klasē ir implementētas arī *equals* un *hashCode* metodes. Doto metožu implementēšana nodrošina objektu identitāti starp dažādam sesijām (savienojumiem ar datu avotu), kas ir nepieciešams, ja tiek plānots strādāt ar objektiem kolekcijās (izslēdz objekta atkārtotas pievienošanas iespējas) [3]. Metodes *hashCode* gadījumā tiek izpildīta operāciju virkne, kuras rezultātā tiek iegūts patvaļīgs skaitlis. Savukārt, metodes *equals* gadījumā tiek pārbaudīts vai divas objektu atsauces rada uz vienu atmiņas apgabalu (objekta instanci), vai objekts ir grāmatas klases eksemplārs un tiek veikta objektu atribūtu atbilstības pārbaude.

```

01 <?xml version="1.0"?>
02 <!DOCTYPE hibernate-mapping PUBLIC
03     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
04     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
05 <hibernate-mapping package="com.example.hibernate">
06     <class name="Grāmata" table="Grāmata" catalog="Sample" schema="dbo">
07         <id column="GrāmatasID" name="grāmatasID" type="int">
08             <generator class="assigned"></generator>
09         </id>

```

10	<code><property name="nosaukums" column="Nosaukums" type="string"</code>
11	<code>not-null="true" length="15"/></code>
12	<code><property name="izdevnieciba" column="Izdevnieciba" type="string"</code>
13	<code>not-null="true" length="15"/></code>
14	<code><set name="autori" table="Grāmata_Autors" cascade="all"></code>
15	<code><key column="GrāmatasID"/></code>
16	<code><many-to-many column="AutoraID" class="Autors"/></code>
17	<code></set></code>
18	<code></class></code>
19	<code></hibernate-mapping></code>

2.19. att. Hibernate objektu klases un relāciju tabulas saistīšanas datne.

Attēlā 5. rindkopā *package* atribūtam tiek uzdots pakotnes nosaukums, kurā ir izvietotas objektu klases (ļauj izvairīties no klašu atrašanas ceļu norādīšanas). 6. rindkopā tiek norādīti saistāmas klases un tabulas nosaukumi, ka arī datu bāzes un shēmas nosaukums (pēdējie divi atribūti ir nepieciešami SQL Server datu bāzes gadījumā). Nākamajā 7. rindkopā tiek norādīta primāras atslēgas kolonna un klases atribūts, kurš atbilst dotai kolonnai. Šeit arī tiek norādīts speciāls Hibernate kartēšanas datu tips, kurš kalpo par norādi datu tipu pārveidošanai [16]. Apakš elementā *generator* atribūtam *class* tiek norādīta primāras atslēgas vērtības piešķiršanas stratēģija, dotajā gadījumā vērtība tiek piešķirta patstāvīgi. Hibernate atbalsta datu bāžu ģenerētas atslēgas un piedāvā vairākus atslēgu ģenerēšanas algoritmus [18]. No 10. līdz 13. rindkopai tiek norādītas saistāmas tabulas kolonnas un klases atribūti, noteikts datu tips, precizēts tabulas lauku pieļaujamais izmērs un *null* vērtības piešķiršanas iespēja. No 14. līdz 17. rindkopai tiek uzdota asociācija „daudzi pret daudziem”. Šeit tiek norādīts tabulas nosaukums, kura saista autoru un grāmatu tabulas datu bāzē, klasē definētas kolekcijas nosaukums un operācijas, kuras ir nepieciešams veikt datu bāzē atbilstoši veiktajām operācijām ar asociāciju starp objektiem [18]. Vērtība *cascade="all"* paredz saglabāšanas, atjaunošanas un dzēšanas operāciju izpildīšanu tabulā Grāmata_Autors, ja attiecīga asociācija starp objektiem tiek definēta, modificēta vai dzēsta. Apakš elementā *key* atribūtam *column* tiek norādīts sekundāras atslēgas lauks. Atbilstoši elements *many to many* satur norādi uz saistāmo autoru klasi un specifificē dotas tabulas sekundāras atslēgas lauku tabulā Grāmata_Autors. Lai asociācija starp objektiem būtu abpusēja, ir nepieciešams līdzīgi definēt attiecīgo asociāciju arī klases Autors kartēšanas XML datnē.

Cita svarīga tehnoloģijas arhitektūras komponente ir XML konfigurācijas datne, kurai ir raksturīgs hibernate.cfg.xml nosaukums. Šajā datnē tiek norādīti savienojuma iestatījumi ar datu avotu, datu bāzēm specifiskie parametri u.tml. [3]. XML konfigurācijas datnes piemērs ir parādīts 2.20 attēlā.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <!DOCTYPE hibernate-configuration PUBLIC
03     "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
04     "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
05 <hibernate-configuration>
06     <session-factory name="sqlserver">
07         <property name="hibernate.connection.driver_class">
08             net.sourceforge.jtds.jdbc.Driver
09         </property>
10         <property name="hibernate.connection.url">
11             jdbc:jtds:sqlserver://localhost:1433;databaseName=Sample;
12             integratedSecurity=true;
13         </property>
14         <property name="hibernate.dialect">
15             org.hibernate.dialect.SQLServerDialect
16         </property>
17         ...
18         <mapping resource="com/example/hibernate/Grāmata.hbm.xml"/>
19         <mapping resource="com/example/hibernate/Autors.hbm.xml"/>
20     </session-factory>
21 </hibernate-configuration>
```

2.20. att. Hibernate konfigurācijas XML datne.

Pēc attēlā ir redzams, ka tehnoloģija pielieto JDBC draiveri savienojuma izveidošanai ar datu avotu. JDBC draivera un savienojuma parametru virknes uzdošana ir attēlota 8., 11. un 12. rindkopā. Hibernate nodod dotos parametrus draiveru menedžera klasei, kura veic norādīta JDBC draivera ielādi [19]. 15. rindkopā tiek norādīta datu bāzei specifiska SQL vaicājumu valodas variācija (dialekts), uz kura bāzēs tiek veikta SQL vaicājumu ģenerēšana. Hibernate atbalsta datu bāžu DB2, HSQLDB, Oracle, SQL Server, MySQL, PostgreSQL, SAP DB, Sybase un TimesTen dialektus [20]. Papildus minētām datu bāzēm dota tehnoloģija tiek testēta arī citām datu bāzēm, piemēram, Access, Informix, Firebird u.c. 18. un 19. rindkopās tiek norādītas klāšu Grāmata un Autors XML kartēšanas datnes.

2.2.2.2. Hibernate objektu modelis

Tehnoloģijas objektu modeli veido vairāki objekti, kuri ir iekļauti *org.hibernate* pakotnē. Biežāk izmantojamie objekti ir aprakstīti tabulā 2.7 [3].

2.7. tabula

Tehnoloģijas objekti

Objekts	Apraksts
<i>SessionFactory</i>	Objekts pēc būtības ir objektu <i>Session</i> kolekcija. Tās veic <i>hibernate.cfg.xml</i> (XML konfigurācijas) datnes ielasīšanu, kā rezultātā tiek izveidots attiecīgs <i>Session</i> objekts.
<i>Session</i>	Objekts reprezentē vienu darba vienību ar datu bāzi. Tās veido objektu izgūšanas un glabāšanas kontekstu, sinhronizē veiktas izmaiņas ar datu bāzi, nodrošina transakciju vadību un garantē objektu identitāti, turot atmiņā izgūtos objektus. Objekts izmanto XML kartēšanas datnes objektu klašu un relāciju datu bāzu tabulu kartēšanai.
<i>Query</i>	Objekts ir paredzēts Hibernate objektorientētas vaicājumu valodas (HQL) vaicājuma izpildes rezultātā iegūtas objektu kolekcijas saņemšanai. Dotais objekts tiek atgriezts objekta <i>Session</i> metodes <i>createQuery</i> izsaukšanas rezultātā.
<i>Criteria</i>	Objekts piedāvā objektorientēto pieeju vaicājumu izpildei. Tās tiek pielietots kā alternatīva <i>Query</i> objektam un tiek atgriezts objekta <i>Session</i> metodes <i>createCriteria</i> izpildes rezultātā.
<i>Transaction</i>	Objekts reprezentē datu bāzes transakciju, ar tā palīdzību ir iespējams atcelt vai apstiprināt izpildītas operācijas.

2.2.2.3. Hibernate pielietojums paradigmu savietojamības problēmu risināšanā

Hibernate objekt-relācijas kartēšanas rīks ir ieguvis plašu pielietojumu Java lietojum-programmatūras izstrādē. Hibernate piedāvā vairākus risinājumus objektorientētas paradigmas un relāciju datu modeļa savietojamības problēmu novēršanai. Hibernate nodrošina [3]:

- Tabulu attiecību un objektu asociāciju kartēšanu. Hibernate atbalsta „viens pret vienu”, „viens pret daudziem”, „daudzi pret daudziem” asociācijas, kuram var tikt uzdots abpusējs vai tikai viens virziens no objekta uz objektu. Hibernate atbalsta

arī „polimorfiskas asociācijas”, kas ļauj objektam veidot asociāciju ar abstraktas klases apakšklases objektu. „Polimorfiskas asociācijas” var tikt uzdotas jebkuram asociāciju veidam – „viens pret daudziem” u.tml.

- Mantošanas hierarhijas kartēšanu. Hibernate atbalsta trīs galvenās mantošanas kartēšanas stratēģijas: „tabula par klašu hierarhiju”, „tabula par konkrēto klasi” un „tabula par klasi”.

Dota tehnoloģija atbalsta arī atsevišķo mantošanas stratēģiju „Tabula par konkrēto klasi ar netiešo polimorfismu” (*table per concrete class using implicit polymorphism*) [21]. „Netiešs polimorfisms” ir paņēmieni, kurš nodrošina atvasināmo objektu instanču atgriešanu, ja vaicājumā tiek uzdots bāzes klases nosaukums (izmantojot bāzes klasi tiek iegūti atvasinātie objekti) [18]. Vienkārša „tabula par konkrēto klasi” stratēģijā šāda iespēja netiek piedāvāta. Kaut gan dotajā stratēģijā šādas iespējas atbalsts nav pilnvērtīgs – rodas zināmas problēmas, kad starp atvasināmo klašu tabulām nav definētas attiecības, ka arī neeksistē bāzes klases tabula, kura satur attiecības ar dotām atvasināto klašu tabulām [3]. Vairākos gadījumos šādu atvasināto objektu instanču izgūšana tiek veikta ar vairāku vaicājumu izpildi.

Hibernate atļauj sava starpā arī kombinēt vairākas mantošanas stratēģijas, piemēram, pielietot „tabula par klašu hierarhiju” stratēģiju kopā ar „tabula par klasi” stratēģiju [21].

- Objektu identitātes vadību. Hibernate garantē objektu identitāti tikai vienas sesijas ietvaros.
- Objektu ielādes optimizēšanu. Hibernate atbalsta „atlikta ielādēšanas” un „enerģiskas ielādēšanas” objektu izgūšanas stratēģijas, ka arī piedāvā „sērijas ielādēšanas” (*batch fetching*) stratēģiju, kura paredz saistīto objektu vai objektu kolekciju sērijas izgūšanu vienā vaicājumā. Dota stratēģija novērš „*n + 1 selects*” problēmu, kura ir raksturīga „atlikta ielādēšanas” stratēģijai. Dota problēma ir apskatīta 1.6 un 1.7 attēlos (LINQ to SQL un Hibernate tehnoloģijas „atlikta ielādēšanas” stratēģija ir noteikta pēc noklusējuma), kur vispirms tiek iegūta rokasgrāmatu objektu kolekcija, tad tiek realizēta pieeja katras rokasgrāmatas saistītam kontakta objektam, kā rezultātā tiek izpildīts $1 + n$ vaicājumi. Pielietojot

doto stratēģiju ģenerējama SQL vaicājumu virkne šāda situācijā varētu izskatīties līdzīgi 2.21 attēlā attēlotai vaicājumu virknei, kur N ir XML kartēšanas datnē noteikts izgūstamo objektu sērijas skaits [3].

01	SELECT * FROM Rokasgrāmata;
02	
03	SELECT * FROM Kontakts k WHERE k.RokasgrāmataID IN (1, 2, ..., N);

2.21. att. Ģenerējamās SQL vaicājumu virknes piemērs.

Papildus apskatītām objektu ielādes stratēģijām Hibernate pilda arī objektu ielādes vadību. Šādas vadības pamatā ir mehānisms, kurš nepilda objekta izgūšanu no datu bāzes, kad attiecīgais objekts tiek pieprasīts. Tā vietā tiek atgriezts speciāls starpnieks (*proxy*) objekts, kurš satur visas pieprasīta objekta metodes. Kad viena no pieprasīta objekta metodēm tiek izsaukta, starpnieks objekts veic dota objekta izgūšanu un izsauc tā attiecīgu metodi. Šāda mehānisma lietderība tiek demonstrēta 2.22 attēlā attēlotā pirmtekstā [3].

01	SessionFactory sessionFactory = new
02	Configuration().configure().buildSessionFactory();
03	
04	Session session = sessionFactory.openSession();
05	
06	Grāmata grāmata = (Grāmata)session.load(Grāmata. class , 1);
07	Autors autors = (Autors)session.load(Autors. class , 2);
08	grāmata.getAutori().add(authors);
09	
10	session.close();

2.22. att. Asociācijas uzdošanas piemērs starp izgūtiem objektiem.

Attēlā 5. un 6. rindkopā tiek izsaukta objekta *Session* metode *load*, kura ir atbildība par objektu izgūšanu pēc to identifikatoriem (primāram atslēgām). Savukārt, 8. rindkopā starp izgūtiem objektiem tiek uzdota asociācija. Programmas pirmteksta izpildes rezultāta ģenerējamie SQL vaicājumi ir attēloti 2.23 attēlā (Hibernate piedāvā iestatījumus ģenerējamo vaicājumu izvadei).

01	select grāmata0_.GrāmatasID as GrāmatasID0_0_,
02	grāmata0_.Nosaukums as Nosaukums0_0_,
03	grāmata0_.izdevniecība as izdevnie3_0_0_
04	from Sample.dbo.Grāmata grāmata0_ where grāmata0_.GrāmatasID=?

05	
05	select autori0_.GrāmatasID as GrāmatasID0_1_,
06	autori0_.AutoraID as AutoraID1_,
07	autors1_.AutoraID as AutoraID2_0_,
08	autors1_.Vārds as Vārds2_0_,
09	autors1_.Uzvārds as Uzvārds2_0_
10	from Grāmata_Autors autori0_ inner join
11	Sample.dbo.Autors autors1_ on autori0_.AutoraID=autors1_.AutoraID
13	where autori0_.GrāmatasID=?

2.23. att. Programmas izpildes laikā ģenerējamie SQL vaicājumi.

Spriežot pēc attēla, ir redzams kā Hibernate neveic objektu izgūšanu, izpildoties metodei *load*. Tās izpildes rezultātā tiek atgriezti starpniek objekti. Objektu izgūšana notiek 2.22 attēla 8. rindkopā, kad tiek izsaukta grāmatas objekta īpašība *getAutori*. Hibernate vispirms ģenerē SQL vaicājumu grāmatas objekta izgūšanai (1. SQL vaicājums 2.23 attēlā), tad ģenerē vaicājumu saistītas autoru objektu kolekcijas izgūšanai (2. SQL vaicājums 2.23 attēlā) un tad pārbauda vai šāds autora objekts ar norādīto primāro atslēgu ir jau saistīts ar izgūto grāmatas objektu. Par cik dotajā gadījumā šāds autora objekts atrodas izgūtajā kolekcijā (ir saistīts), Hibernate neģenerē SQL vaicājumu autora objekta izgūšanai. Kā redzams, šāds mehānisms ļauj samazināt izpildāmo SQL vaicājumu skaitu, kas uzlabo programmas kopējo veiktspēju.

Hibernate piedāvā divas objektorientētas vaicājumu valodas [3]:

- HQL (*Hibernate Query Language*) – valoda liela mēra atgādina objektorientēto datu bāžu OQL (*Object Query Language*) vaicājumu valodu. Tā balstās uz SQL vaicājumu valodas sintaksi, taču atšķirība no tās, darbojas ar objektu klasēm un to atribūtiem. Piemēram, 2.24 attēlā attēlotajā pirmtekstā tiek konstruēts HQL vaicājums, kurš izgūst saistīto grāmatas un autoru objektu atribūtu vērtības (šeit *from* blokā tiek norādīts klases nevis tabulas nosaukums; autori, nosaukums, vārds u.tml. ir klašu atribūtu nosaukumi).

01	Query query = session.createQuery(
02	"select grāmatas.nosaukums, autori.vārds, autori.uzvārds " +
03	"from Grāmata grāmatas " +
04	"join grāmatas.autori autori");

2.24. att. HQL vaicājumu valodas piemērs.

- Criteria – datu pieejas karkass, kurš satur vairākus SQL vaicājumu konstruēšanas objektus. Piemēram, 2.25 attēlā tiek konstruēts vaicājums, kurš veic grāmatu izgūšanu, kuru autora uzvārds ir Ziedonis. Dotais vaicājums tiek veidots pamatojoties uz diviem *Criteria* objektiem (tiek izveidoti metodes *createCriteria* izpildes rezultātā), kuriem 3. rindkopā tiek uzdots ierobežojums.

01	<code>Criteria criteria = session.createCriteria(Grāmata.class)</code>
02	<code>.createCriteria("autori")</code>
03	<code>.add(Restrictions.eq("uzvārds", "Ziedonis"));</code>

2.25. att. Criteria vaicājuma piemērs.

Hibernate vaicājumu valodas nodrošina grupēšanas, projekcijas, kārtotāšanas u.c. SQL vaicājumu valodai raksturīgas operācijas [3]. Tas nodrošina arī apvienošanas operācijas, kas var kalpot par alternatīvo risinājumu „*n + 1 selects*” problēmas novēršanai (nepieciešamie objekti tiek izgūti uzreiz) [22]. Taču dotam valodām ir raksturīgi vairāki trūkumi:

- Vaicājumu semantika balstās uz argumentu norādīšanu rakstzīmju virknēs, kuras netiek uzturētas mūsdienu izstrādes vidēs (rodas uzturēšanas problēmas, piemēram, ja klašu modelī tiek mainīts atribūta *uzvārds* nosaukums).
- Vaicājumos tiek realizēta pieeja slēgtiem (*private*) klašu atribūtiem, izslēdzot iespēju darboties ar atribūtu īpašībām. Šāda pieeja lauž iekapsulēšanas principus, nenodrošinot visas objektorientētas paradigmas iespējas.

Hibernate atbalsta arī SQL vaicājumu valodu, kura var tikt pielietota objektu izgūšanai, ka arī datu bāžu glabājamo procedūru izsaukšanai [3].

3. DATU PIEJAS TEHNOLOĢIJU SALIDZĪNOŠA ANALĪZE

Datu pieejas tehnoloģiju salīdzinoša analīze ir veikta balstoties tehnoloģiju piedāvātiem risinājumiem pirmajā nodaļā apskatīto problēmu novēršanai. Salīdzinošas analīzes iegūtie rezultāti ir apkopoti tabulā 3.1. Tabula lietoti apzīmējumi ir sekojoši:

- „+” – atbalsta;
- „-” – neatbalsta;
- „+ / - ” – daļējs atbalsts.

3.1. tabula

Datu pieejas tehnoloģiju salīdzinoša analīze

	LINQ to SQL	ADO.NET EF	Hibernate	JDBC	ADO.NET
Objektu identitātes vadība	+	+	+	-	-
Objektorientēta vaicājumu valoda	+	+	+ / -	-	-
Vairāku datu bāzu atbalsts	-	+	+	+	+
Datu tipu atbilstības pārbaude (saistīšana)	+	+	+	-	-
Mantošanas kartēšana					
„Tabula par klasi”	-	+	+	-	-
„Tabula par konkrēto klasi”	-	+	+	-	-
„Tabula par klašu hierarhiju”	+	+	+	-	-
„Tabula par konkrēto klasi ar netiešo polimorfismu”	-	-	+	-	-
Mantošanas kartēšanas stratēģiju kombinēšana	-	-	+	-	-
Objektu ielādes optimizēšana					

„Atlikta ielādēšana”	+	+ / -	+	-	-
„Enerģiska ielādēšana”	+	+	+	-	-
„Sērijas ielādēšana”	-	-	+	-	-
Objektu ielādes vadība	-	-	+	-	-
Asociāciju un attiecību kartēšana					
„Viens pret vienu”	+	+	+	-	-
„Viens pret daudziem”	+	+	+	-	-
„Daudzi pret daudziem”	-	+	+	-	-
„Polimorfiskas asociācijas”	-	+	+	-	-

Spriežot pēc tabulas datiem ir redzams, ka tehnoloģija LINQ to SQL ir optimālākais risinājums objektorientētas paradigmas un relāciju datu modeļu savietojamības problēmu novēršanai. Tās der sistēmām, kuras tiek plānots projektēt pēc iespējas identiski datu bāzes shēmai. Tās nodrošina datu tipu saistību, pilda objektu identitātes vadību, piedāvā tīri objektorientēto pieeju (LINQ vaicājumu valodu) datu izgūšanai un piedāvā vairākas stratēģijas objektu ielādes optimizēšanai. LINQ to SQL risinājums ir optimizēts SQL Server datu bāzes funkcionalitātes atbalstām, tās ļauj strādāt ar datu bāzes glabājamam procedūrām, atbalsta datu bāzes primāras atslēgas ģenerēšanas stratēģiju u.tml. Saskaņā ar šo dotais risinājums neatbalsta vairākas citas datu bāzes, ko var uzskatīt par tā galveno trūkumu. Dotas tehnoloģijas izvēlē būs piemērotāka situācijas, kad sistēmas datu bāze ir SQL Server, nav nepieciešams paredzēt augstas abstrakcijas programmatūras līmeni, tās ļaus ietaupīt laiku paredzot automātisko objektorientēta un relāciju modeļu saistīšanu.

Tehnoloģijas Hibernate un ADO.NET EF paredz vairākus risinājumus objektorientētas paradigmas un relāciju datu modeļu savietojamības problēmu novēršanai. Tās atbalsta galvenās mantošanas kartēšanas un objektu ielādes stratēģijas, paredz augsta abstrakcijas līmeņa izveidošanu, nodrošina datu tipu saistīšanu, pilda objektu identitātes vadību un var tikt pielietoti sarežģīto sistēmu izstrādei.

Tehnoloģija ADO.NET EF pēc noklusējuma nepilda saistīto objektu ielādi, kad tiem tiek realizēta pieeja. Šādam paņēmienam ir gan savi trūkumi, gan arī priekšrocības. Par priekšrocību

var uzskatīt objektu ielādes kontroles iespēju (izstrādātājs vienmēr var kontrolēt to, kas tiks ielādēts), bet no citās pusēs tās prasa papildus pirmteksta uzrakstīšanu un rāda grūtības, pārejot no tehnoloģijas LINQ to SQL [23]. Dotas tehnoloģijas izvēle ir piemērota gadījumos, kad tiek plānots paredzēt sarežģīto objektu hierarhijas modeli un par sistēmas datu bāzi nav izvēlēta SQL Server datu bāze. Par dotas tehnoloģijas trūkumu var uzskatīt tās salīdzinoši nelielo veikspēju salīdzinājumā ar citām tehnoloģijām, piemēram, LINQ to SQL [24].

Tehnoloģija Hibernate piedāvā vairākus papildus risinājumus relāciju datu modeļa un objektorientētas paradigmas savietojamības problēmu risināšanā. Tā pilda objektu ielādes optimizēšanu, piedāvā „sērijas ielādēšanas” un „tabula par konkrēto klasi ar netiešo polimorfismu” stratēģijas, ļauj kombinēt galvenās mantošanas kartēšanas stratēģijas. Hibernate trūkumi galvenokārt ir saistīti ar tās objektorientētam programmēšanas valodām, kuras nepiedāvā tīri objektorientēto pieeju datu izgūšanai (vaicājumu semantika netiek pārbaudīta ar kompilatora palīdzību, vaicājumu realizē pieeju slēgtajiem klases atribūtiem). Priekš .NET platformas eksistē LINQ to NHibernate gādnieks, kas ļauj pielietot LINQ vaicājumu valodu darbam ar tehnoloģijas NHibernate (Hibernate versija priekš .NET platformas) objektiem. Java platformā ir uzsākta LINQ alternatīvas izstrāde, uz doto brīdi pazīstamākas ir Quaere, JaQu un JaQue.

Tehnoloģijas JDBC un ADO.NET der nelielam sistēmām, kurās netiek plānots projektēt objektorientēto programmatūras modeli. To priekšrocība salīdzinājumā ar dotām tehnoloģijām ir samēra augsta veikspēja un iespēja tieši darboties ar datu avotu. Dotas tehnoloģijas izmanto objekt-relācijas kartēšanas tehnoloģijas, lai nodrošinātu pieeju kādam noteiktām datu avotam un veiktu ģenerēto SQL vaicājumu izpildi. Piemēram, Hibernate šādām nolūkam izmanto JDBC draiveri, savukārt, LINQ to SQL *SqlClient* datu gādnieku.

4. TEHNOLOĢIJAS ADO.NET ENTITY FRAMEWORK PRAKTISKS PIELIETOJUMS

Darbā praktiska daļā ir izstrādāta poliklīnikas uzskaites vadības sistēma. Sistēmas mērķis ir uzlabot un atvieglot poliklīnikas ikdienas darbību, nodrošinot iekšējas informācijas vadību, kas saistīta ar jauno pacientu, darbinieku un slimību reģistrēšanu, darbinieku darbību un pacientu ārstēšanu. Sistēma ļauj uzturēt organizācijas iekšējo kartību, piedāvājot katrai lietotāju grupai atbilstošas pieejas tiesības un darbības klāstu, nodrošina ārstēšanas kursu standartizēšanu un nepieciešamo datu uzskaiti, ka arī glabāšanu.

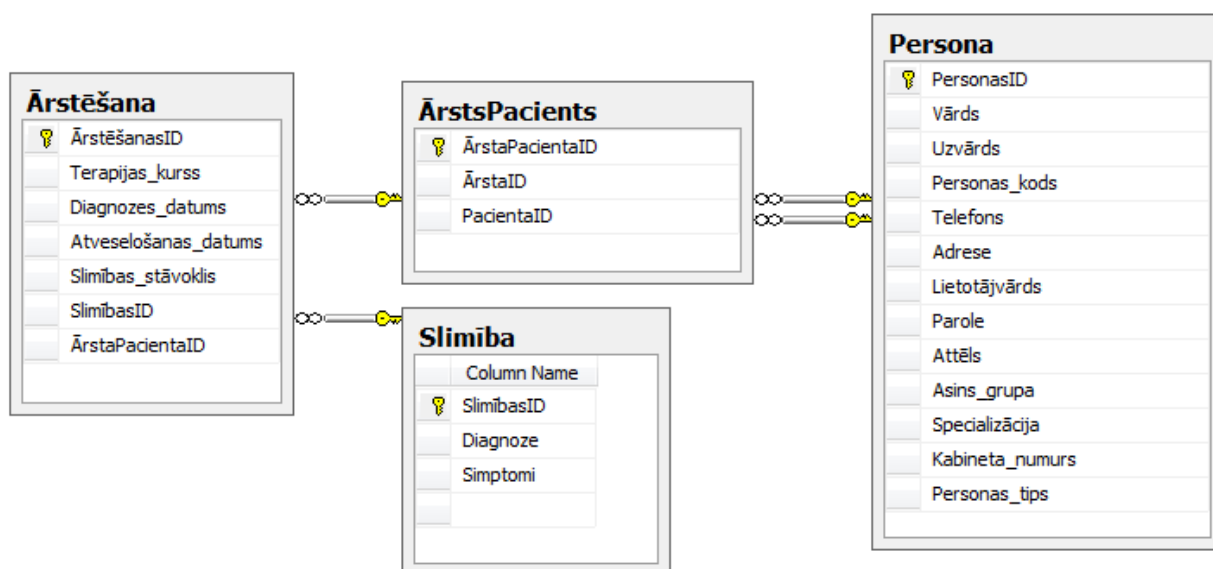
4.1. Sistēmas funkcionālās prasības

1. Lietotāju klases.
 - 1.1. Sistēmā tiks nodrošināts darbs četrām lietotāju grupām: ārstiem, reģistratūras darbiniekiem, pacientiem un administratoram.
 - 1.2. Sistēma nodrošinās lietotāju autentifikāciju – lietotājvārda un paroles ievadīšanu.
2. Sistēmas administratora iespējas.
 - 2.1. Sistēma nodrošinās iespēju administratoram reģistrēt jaunus darbiniekus – ārstus un reģistratūras darbiniekus.
 - 2.2. Sistēma ļaus administratoram modificēt jebkuru darbinieku datus.
 - 2.3. Sistēmas administrators var dzēst darbinieku datus no datu bāzes.
3. Reģistratūras darbinieku iespējas.
 - 3.1. Sistēma nodrošinās iespējas reģistratūras darbiniekiem reģistrēt jaunus pacientus un norīkot dotos pacientus pie konkrētiem ārstiem.
 - 3.2. Sistēma nodrošinās iespējas reģistratūras darbiniekiem modificēt pacientu personīgos datus (vārds, uzvārds u.tml.).
 - 3.3. Reģistratūras darbinieki varēs apskatīties un modificēt savus datus.
4. Ārstu iespējas.
 - 4.1. Sistēma nodrošinās iespējas ārstiem apskatīt tikai viņu pacientu ārstēšanas datus.
 - 4.2. Ārsti varēs uzstādīt diagnozes pacientiem, noteikt ārstēšanas kursus un vadīt slimību norises gaitās, šos reģistrētos datus viņi varēs labot.

- 4.3. Ārsti varēs reģistrēt sistēmā jaunas slimības, norādot to nosaukumus un tām raksturīgos simptomus.
- 4.4. Sistēma nodrošinās iespējas ārstiem pārlūkot savus profilus un nepieciešamības gadījumā modificēt tos.
5. Pacientu iespējas.
 - 5.1. Pacienti varēs apskatīties savu slimību vēsturi.
6. Ārstu, reģistratūras darbinieku un administratora iespējas.
 - 6.1. Sistēma nodrošinās iespējas ārstiem, reģistratūras darbiniekiem, pacientiem un administratoram atlasīt un pārskatīt datus pēc vairākiem kritērijiem (uzvārdiem, diagnozēm).

4.2. Sistēmas modeļa projektēšana

Vadoties pēc funkcionālām prasībām ir izstrādāta 4.1 attēlā attēlota datu bāzes struktūra.



4.1. att. Sistēmas datu bāzes struktūra.

Datu bāzes struktūru veido četras tabulas. Tabula „Slimība” satur datus par ārstējamu slimību. Šādi dati ir slimībai raksturīgie simptomi un tās nosaukums, kas tiek pielietots pacienta slimības diagnozes uzstādīšanai.

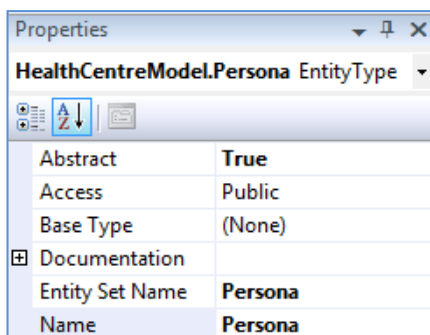
Tabula „Persona” ietver sevī četrus lietotāja tipus: administrators, ārsts, reģistratūras darbinieks un pacients. Tabulas atribūti „Specializācija” un „Kabineta_numurs” ir raksturīgi vienīgi ārstu lietotāju grupai. Savukārt, atribūts „Asins_grupa” ir raksturīgs tikai pacientiem. Pārējie tabulas lauki ir raksturīgi visam lietotāju grupām. Tabula satur atribūtu „Personas_tips”, kas ir diskriminators (speciāls tabulas kolonnas lauks, kas viennozīmīgi identificē katru tabulas objektu). Izmantojot to tiks realizēta „tabula par klašu hierarhiju” mantošanas kartēšanas stratēģija. Ir pieņemti sekojoši tabulas ierakstu tipu (objektu) identifikatori:

- 0 – administrators;
- 1 – ārsts;
- 2 – reģistratūras darbinieks,
- 3 – pacients.

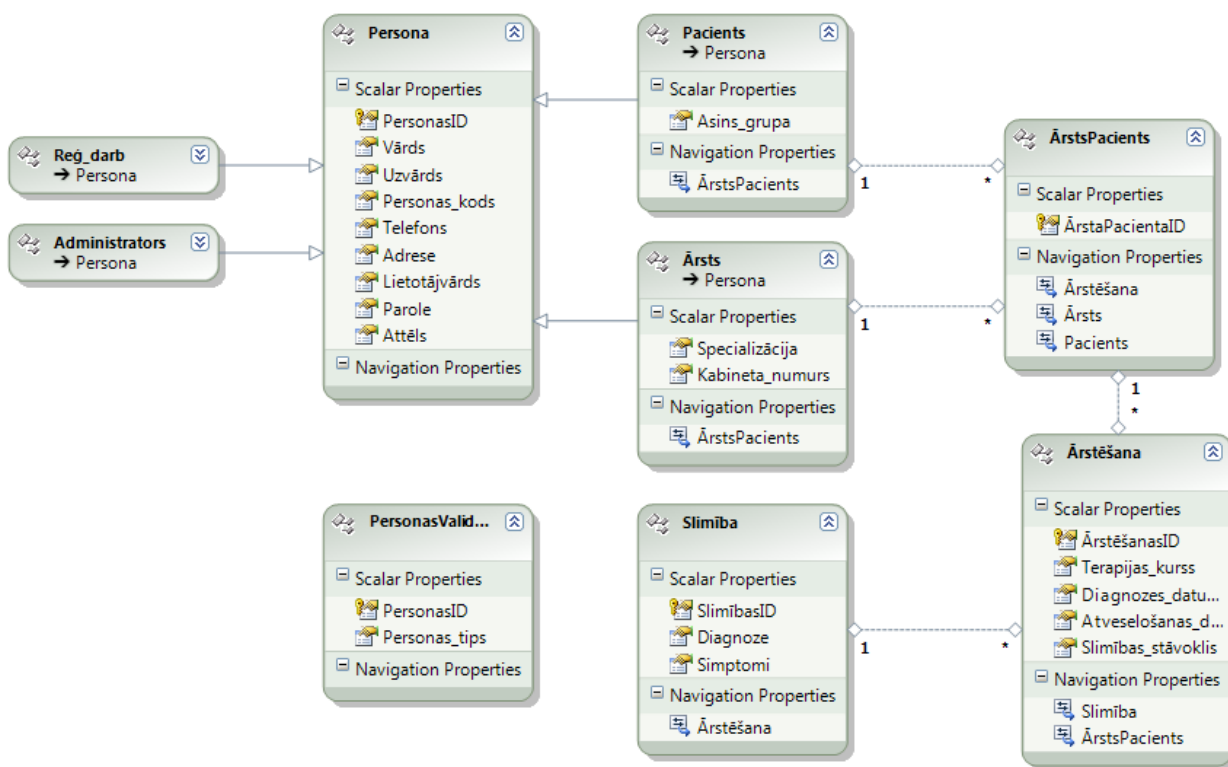
Tabula „ĀrstsPacients” ir papildus tabula, kas tiek izveidoto realizējot attiecību „daudzi pret daudziem”. Tā savstarpēji saista ārstus un pacientus, kuri tiek norīkoti pie viņiem.

Tabula „Ārstēšana” satur datus par pacientu ārstēšanas epizodi. Šādi dati ir diagnozes uzstādīšanas, atveseļošanas datums, terapijas kurss (medikamentu, rehabilitācijas u.tml.) un slimības stāvokļa (slimības norises gaitas) aprakstus.

Balstoties uz datu bāzes struktūru un pielietojot ADO.NET EF tehnoloģiju ir izstrādāts 4.2 attēlā attēlots programmatūras klašu modelis. Pēc attēla ir redzams, ka personu klasei ir paredzētas vairākas apakšklases – „Ārsts”, „Pacients”, „Reģ_darb” (reģistratūras darbinieks) un „Administrators”, kuras atbilst funkcionālas prasības definētam lietotāju grupām. Bāzes klasē ir izslēgts „Personas_tips” atribūts, kas izslēdz situācijas, kuras objektu identifikatori var tikt modificēti. Papildus klasei ir piešķirts arī *Abstract* modifikators, kas neļauj izveidot dotas klases eksemplārus. Dota modifikatora piešķiršana ir attēlota 4.3 attēlā.



4.3. att. Modifikatora *Abstract* piešķiršana bāzes klasei.



4.2. att. Programmatūras klašu hierarhijas modelis.

Atvasināto klašu izveidošana tiek panākta ar komandas *Add*→*Entity* izpildi, ka rezultātā tiks atvērta 4.4 attēlā attēlota forma. Tajā ir jānorāda atvasinātas klases nosaukums un bāzes klase, no kuras tā tiks atvasināta. Pēc apakšklases izveidošanas, tajā no bāzes klases ir jāievelk apakšklasei piederoši atribūti, piemēram, pacientu apakšklases gadījumā šāds atribūts ir „Asins_grupa”.

Visu klašu un to attiecīgo tabulu kartēšanu var veikt izstrādes vides Visual Studio logā *Mapping Details*. Tajā var norādīt kādus klašu īpašību datu tipus ir nepieciešams pretstatīt tabulas kolonnas datu tiptiem, ka arī apakšklases gadījumā ir jānorāda diskriminatora tabulas kolonnas lauks un dotas apakšklases identificējoša diskriminatora vērtība. Dotais logs ir attēlots 4.5 attēlā, kas raksturo kartēšanas nosacījumus priekš apakšklases „Arsts”. Pēc attēla ir redzams, ka dotas apakšklases diskriminatora vērtība tiek uzdota ar „*When Personas_tips = 1*” nosacījumu, ka rezultāta tehnoloģija zinās ka ārsta apakšklases eksemplāru izgūšanai ir nepieciešams konstruēt SQL vaicājumu, noradot attiecīgo izgūšanas kritēriju.

Izmantojot doto logu un izpildot komandu *Add*→*Association* ir iespējams uzdot arī asociācijas starp objektiem. 4.2 attēlā ir parādīts „polimorfiskas asociācijas” piemērs, kas saista ārsta (pacienta) apakšklasi ar „ĀrstsPacients” klasi.

The 'Add Entity' dialog box is shown with the following details:

- Entity name:** Entity1
- Base type:** Persona
- Entity Set:** Persona
- Key Property:**
 - ☐ Create key property
 - Property name:** Id
 - Property type:** Int32
- Buttons:** OK, Cancel

4.4. att. Klases (entītijas) izveidošanas forma.

The 'Mapping Details - Ārsts' window displays the following table:

Column	Operator	Value / Property
Tables		
Maps to Persona		
When Personas_tips	=	1
<Add a Condition>		
Column Mappings		
Asins_grupa : tinyint	↔	Specializācija : String
Specializācija : varchar	↔	Kabineta_numurs : Int16
Kabineta_numurs : smallint	↔	
Personas_tips : tinyint	↔	
<Add a Table or View>		

At the bottom of the window, there are tabs for 'Error List', 'Mapping Details', and 'Output'.

4.5. att. Apakšklases „Ārsts” kartēšanas nosacījumi.

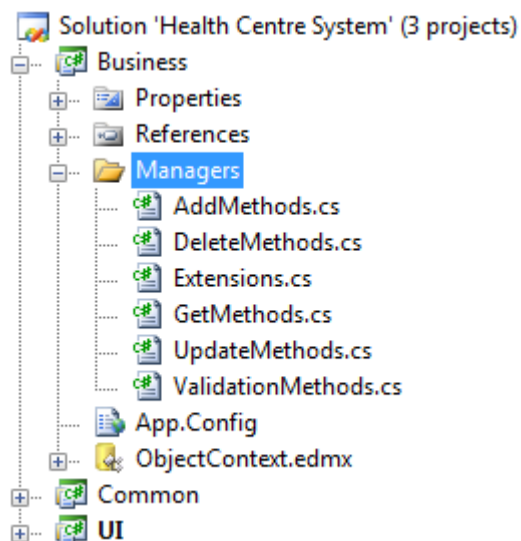
Izmantojot ADO.NET EF vizuāla konstruktora piedāvātos līdzekļus (*Mapping Details* logu, īpašību uzdošanas logu u.tml.), ir iespējams izvairīties no patstāvīgas kartēšanas nosacījumu specificēšanas SSDL, MSL, CSDL valodās. Visas konstruktorā veiktas izmaiņas tiks automātiski ģenerētas doto valodu shēmās un konstruktora pirmteksta klasē.

4.3. Sistēmas implementēšana

Programmatūras kopējo projektu veido trīs apakš projekti:

- UI (*user interface*) – satur lietotāja saskarnes formas un to pirmtekstus;
- Common – satur klases, kuras ir realizētas metodes kopējai lietošanai.
- Business – realizē programmatūras komerciālo (business) loģiku. Tās satur klases, kuras ir realizētas metodes objektu saglabāšanai, dzēšanai, izgūšanai, darbību validācijai (pārbaude vai norādītais lietotājvārds eksistē, lietotājvārda un paroles pārbaude u.t.t.) u.c. metodes. Tās arī satur ADO.NET EF konstruktoru, kurā ir realizēta programmatūras modeļa kartēšana ar datu bāzes struktūru (datne ar paplašinājumu *edmx*).

Projekta apakš projektu struktūra ir attēlota 4.6 attēlā.



4.6. att. Projekta struktūra.

Dažas no šādam objektu izgūšanas, atjaunošanas, saglabāšanas un dzēšanas metodēm ir attēlotas 4.7 attēlā.

```

01 // Objektu izgūšanas metode
02 public static IList<Ārsti> GetDoctorsByLastname(string lastname) {
03     using (MapConnection db = new MapConnection()) {
04         var doctors = from d in db.Persona.OfType<Ārsti>()
06             where d.Uzvārds.StartsWith(lastname)
07             orderby d.Specializācija
08             select d;
09
10         return doctors.ToList();
11     }
12 }
13
14 // Objekta saglabāšanas metode
15 public static void AddDoctor(Ārsti doctor) {
16     using (MapConnection db = new MapConnection()) {
17         db.AddToPersona(doctor);
18         db.SaveChanges();
19     }
20 }
21
22 // Objekta dzēšanas no datu bāzes metode
23 public static void DeleteTreatment(int treatmentID) {
24     using (MapConnection db = new MapConnection()) {
25         EntityKey treatmentKey = new EntityKey(
26             "MapConnection.Ārstēšana", "ĀrstēšanasID", treatmentID);
27         var disease = db.GetObjectByKey(diseaseKey);
28         db.DeleteObject(disease);
29         db.SaveChanges();
30     }
31 }
32
33 // Objekta stavokļa atjaunošanas metode
34 public static void UpdateDoctor(Ārsti doctor) {
35     using (MapConnection db = new MapConnection()) {
36         db.AttachUpdated(doctor);
37         db.SaveChanges();
38     }
39 }

```

4.7. att. Metodes darbam ar ADO.NET EF konstruktora ģenerētam entītiņu klašu objektiem.

Attēlā attēlota izgūšanas metode veic ārstu objektu izgūšanu no datu bāzēs, vadoties pēc lietotāja norādīta ārsta uzvārda. Metode *OfType* specificē kādu bāzes klases apakšklases objektu ir nepieciešams izgūt (pēc tās vadoties, tiek ģenerēts attiecīgs SQL vaicājums, kurā tiek norādīta atvasināta objekta diskriminatora vērtība).

Objekta saglabāšanas metode veic ārsta objekta saglabāšanu datu bāzē. Tajā tiek izsaukta ADO.NET EF ģenerēta konstruktora metode *AddToPersona*, kas nodrošina dota objekta pievienošanu izveidotajām objektu vadības, glabāšanas un dzēšanas kontekstam. Klases *ObjectContext* metode *SaveChanges* pilda veikto izmaiņu saglabāšanu datu bāzes struktūrā.

Objekta dzēšanas metode pilda objekta pacienta ārstēšanas kursa dzēšanu no datu bāzēs. Tajā tiek izveidots *EntityKey* objekts ar kura palīdzību tiek mēģināts izgūt objektu no atvērta kontekstā ar datu bāzi. Jā pieprasītais objekts nav iepriekš ielādēts, tiks izpildīts vaicājums tā ielādēšanai. Šādu nosacījumu (lai izdzēstu objektu, tam ir jābūt izgūtam – objekta dzēšanai ir jāizpilda divi vaicājumi) var uzskatīt par objekt-relāciju kartēšanas tehnoloģiju trūkumu salīdzinājuma ar vienkāršām tehnoloģijām, piemēram, JDBC un ADO.NET.

Objekta stāvokļa atjaunošanas metode pilda ārsta objekta satura atjaunošanu datu bāzes tabulā. Dotajā metodēs ķermenī tiek izsaukta metode *AttachUpdated*, kas mēģina pievienot objektu pašreizējam kontekstam, saskaņojot tā saturu ar tā paša objekta saturu, kas atrodas datu bāzē. T.i., dotais objekts tika izgūts kāda iepriekšēja sesijā (kontekstā) ar datu bāzi un tika neatkarīgi no tās modificēts, lai ADO.NET EF spētu atjaunot dota objekta saturu, tai ir nepieciešams izgūt to pašu objektu no datu bāzes un šo divu objektu saturu salīdzinājumu. Metode *AttachUpdated* ir patstāvīgi izstrādāta, tās saturs ir attēlots 4.8 attēlā.

```
01 public static void AttachUpdated(this ObjectContext obj,  
02     EntityObject objectDetached) {  
03     if (objectDetached.EntityState == EntityState.Detached) {  
04         object original = null;  
06         if (obj.TryGetObjectByKey(objectDetached.EntityKey,  
07             out original))  
08             obj.ApplyPropertyChanges(  
09                 objectDetached.EntityKey.EntitySetName,  
10                 objectDetached);  
11     Else  
12         throw new ObjectNotFoundException();  
13 }
```

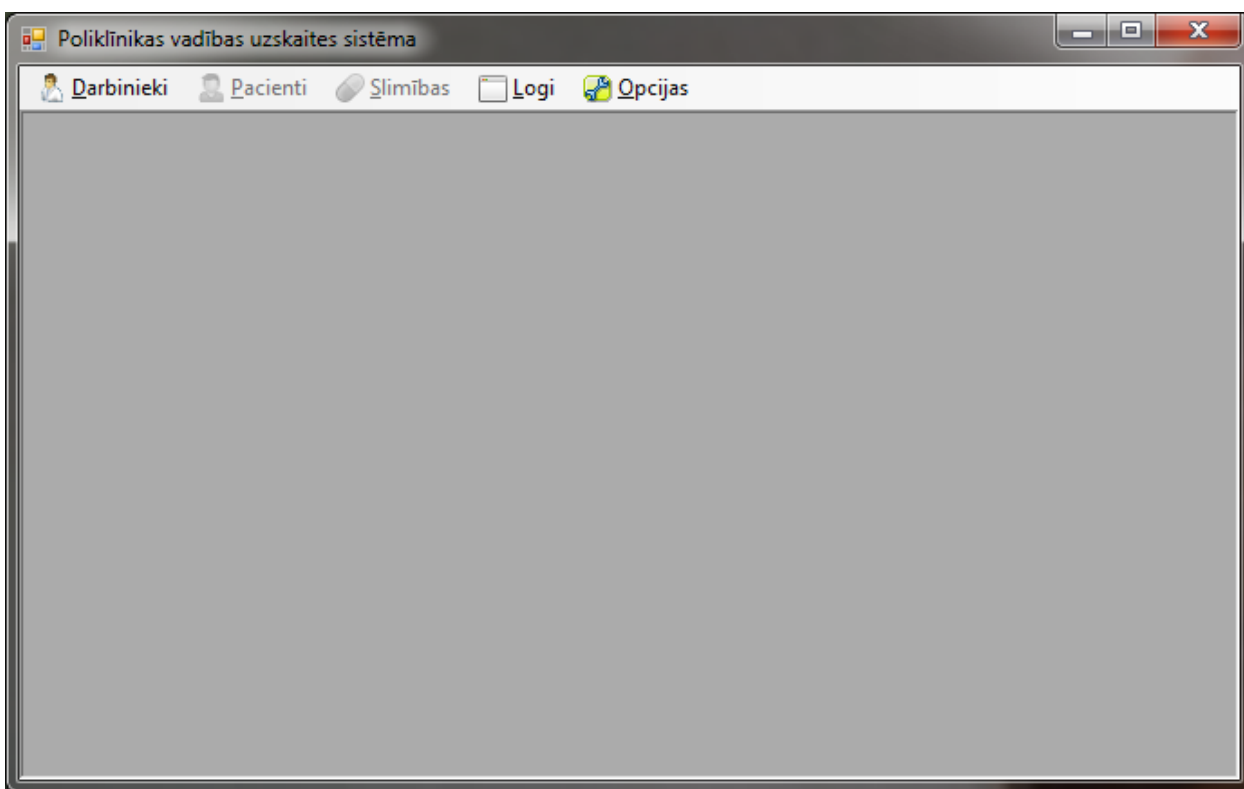
4.8. att. Metodes *AttachUpdated* realizācija.

4.4. Izstrādātās programmatūras apraksts

Pēc programmas palaišanas tiek atvērta autorizācijas forma, kurā lietotājiem ir jāievada savi autentifikācijas dati: lietotājvārds un parole. Lietotāju autorizācijas forma ir attēlota 4.9 attēlā.

4.9. att. Lietotāju autorizācijas forma.

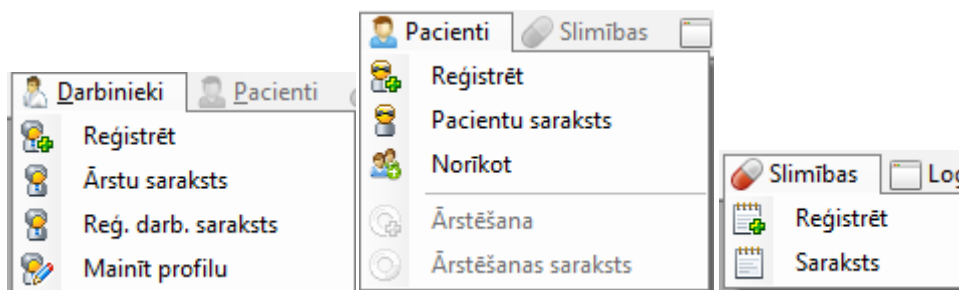
Jā lietotājsvārds vai parole būs ievadīti nepareizi, tiks izvadīts kļūdas paziņojums, pretēji tiks atvērta galvenā lietojumprogrammas forma. Tā kalpo par apakšformu konteineri, kurā var tikt atvērtas un uzturētas vairākas dažādas formas. Dota forma ir attēlota 4.10 attēlā.



4.10. att. Lietojumprogrammas galvenā forma.

Galvenā forma satur izvēlnes „Darbinieki”, „Pacienti” un „Slimības”, kuras atkarība no sistēmā pieteikta lietotāja grupas, liedz viņam nepiederošas darbības. Doto izvēlņu piedāvātais darbības klāsts ir attēlots 4.11 a), 4.11 b) un 4.11 c) attēlos. Izvēlnes „Logi” un „Opcijas” ir

pieejamas jebkuram lietotāja tipam. Izvēlne „Logi” piedāvā galvenajā forma atvērto apakš formu grupēšanas un izvietojšanas iespējas, savukārt, izvēlne „Opcijas” paredz sistēmas attieksšanās (iziešanas no sistēmas darbību, neaizvērot programmu) un aizvēršanas darbības, ka arī informācijas par sistēmu izvadi.




4.11. att. a) Izvēlne „Darbinieki” (pa kreisi), b) Izvēlne „Pacienti” (pa vidu), c) Izvēlne „Slimības” (pa labi).

Izvēlne „Darbinieki” ir pieejama tikai administratoram, reģistratūras darbiniekiem un ārstiem. Ārstu un reģistratūras gadījumā visas izvēlne pieejamas darbības, izņemot darbību „Mainīt profilu” būs slēgtas.

Izpildot darbību „Reģistrēt” tiks atvērta darbinieku reģistrēšanas forma. Tajā ir iespējams norādīt tādas darbinieka personas datus kā vārds, uzvārds, personas kods, telefons un adrese. Darbinieku reģistrēšanas forma piedāvā darbinieka fotoattēla norādīšanas, lietotājvārda un paroles, ka arī amata specificēšanas iespējas. Jā reģistrējama darbinieks ir ārsts, tad sistēmas administratoram obligāti ir jānorāda arī viņa specializācija un kabineta numurs. Dota forma ir attēlota 4.12 attēlā.

Izpildot darbību „Mainīt profilu” tiks atvērta iepriekš aplūkota forma (4.12 attēls), kurā darbinieks var mainīt savus datus, piemēram, administratora noteikto paroli.

Darbības „Ārstu saraksts” un „Reģ. darb. saraksts” pilda sistēma reģistrēto ārstu un reģistratūras darbinieku izvadi. Reģistratūras darbinieku saraksta izvades forma ir attēlota 4.13 attēlā. Tā piedāvā sekojošas darbības:


- Labošana () – nospiežot doto pogu, tiek atvērta darbinieku reģistrēšanas (profilu modifikācijas) forma, kura saturēs izvēlēta darbinieka datus. Veicot nepieciešamo izmaiņu ievadi, lietotājs var saglabāt tās, ka rezultātā saraksts tiks automātiski atjaunots.

- Dzēšana (🗑️) – darbība paredz no saraksta izvēlēta reģistratūras darbinieka dzēšanu.
- Saraksta atjaunošana (🔄) – darbība paredz saraksta atjaunošanu (atkārtoto darbinieku objektu izgūšanu no datu bāzes).
- Atlase (🔍) – darbība paredz objektu atlases operācijas izpildi, vadoties pēc lietotāja norādīta atlases kritērija. Reģistratūras darbinieku saraksta formā šāds kritērijs ir darbinieka uzvārds.

Ārstu saraksta izvades forma ir identiska iepriekš apskatītai reģistratūras darbinieku izvades formai. Tā piedāvā līdzvērtīgas darbības darbam ar ārstu objektiem.

Darbinieks

Dati par personu



Sameklēt

Darbinieka profils

Vārds: Aleksejs

Uzvārds: Saveljevs

Person. k.: 121163-22222

Tel.: 20999007

Adrese: Rīga, Daugavpils 79

Lietotājevārds: Aleksejs

Parole | **Amats**

☒ Ārsts ☐ Reģ. darb. ☐ Administrators

Specializācija: Kardiologs

Kabin. num.: 97

Aizvērt Saglabāt datus

4.12. att. Darbinieku reģistrēšanas (profilu modifikācijas) forma.

Vārds	Uzvārds	Personas kods	Telefons	Adrese	Lietotājvārds	Parole
Mihails	Bavikins	121141-90088	23445689	Ogre, Dzimavu 177	Mihails	Mihails
Alisija	Dadze	171281-28977	29888889	Rīga, Meza 5	Alisija	Alisija
Anastasija	Nikulina	130981-12345	23445567	Rīga, Motoru 333	Anastasija	Anastasija
Inese	Preikule	170979-12199	29002999	Rīga, Elizabetes 4	Inese	Inese

4.13. att. Reģistratūras darbinieku izvades forma.

Izvēlne „Pacienti” nodrošina darbu tikai trim lietotāju grupām: ārsts, pacients un reģistratūras darbinieks (administratora gadījuma izvēlne kļūst nepieejama). Reģistratūras darbiniekiem ir pieejamas tikai trīs darbības no izvēlnes: „Reģistrēt”, „Pacientu saraksts” un „Norīkot”, savukārt, ārstu gadījumā atļautas darbības ir „Ārstēšana” un „Ārstēšanas saraksts”. Pacients var izvēlēties tikai „Ārstēšanas saraksts” darbību.

Izpildot darbību „Reģistrēt”, tiks atvērta pacientu reģistrēšanas forma, kura arī tiek pielietoto reģistrēto datu modificēšanai. Šādi dati ir pacienta vārds, uzvārds, personas kods, telefons, asins grupa u. tml. Dota forma ir attēlota 4.14 attēlā.

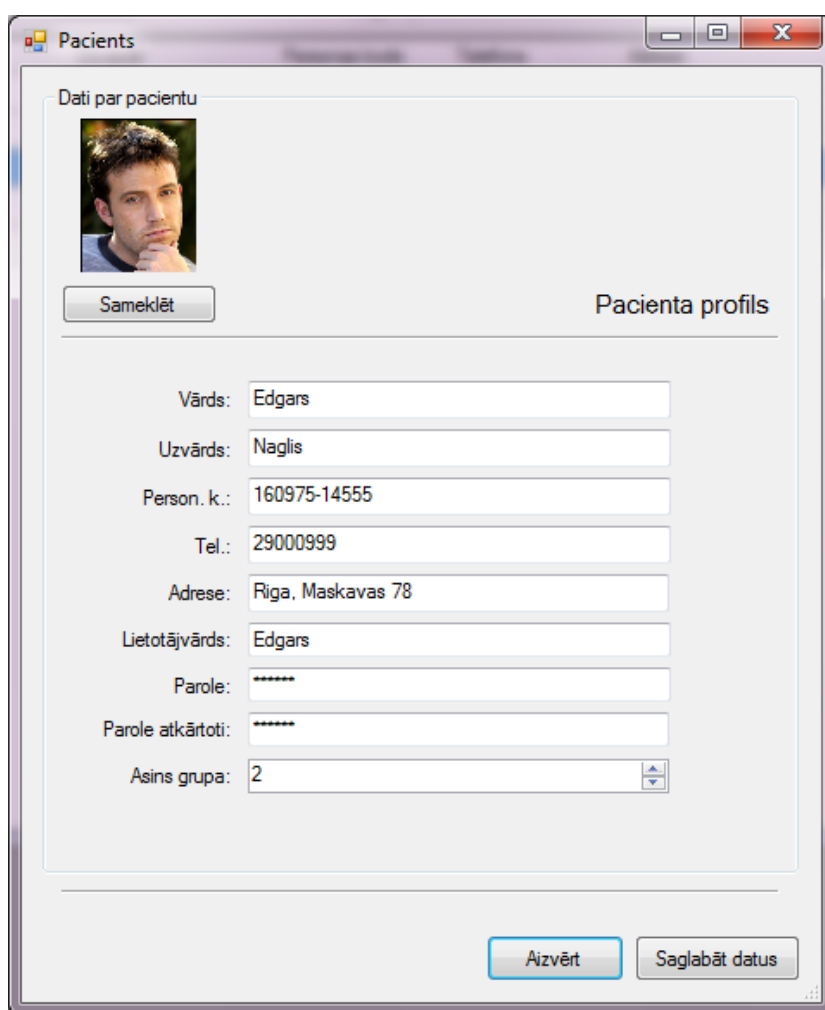
Darbība „Pacientu saraksts” paredz reģistrēto pacientu sarakstu izvadi. Šādu funkciju realizē pacientu saraksta izvades forma, kura piedāvā reģistratūras darbinieku izvades formai līdzvērtīgo operāciju klāstu, t.i., pacientu dzēšanas, esošo datu modifikācijas, atlases un saraksta atjaunošanas operācijas.

Darbība „Norīkot” piedāvā pacientu norīkošanas iespējas konkrētiem ārstiem. Izpildot doto darbību, tiek atvērta 4.15 attēlā attēlota forma. Tajā pacienta norīkošanas funkcijas izpildei, lietotājiem ir jāizpilda šāds darbības klāsts:

- Jāizvēlas nepieciešamais pacients no iznirstoša pacientu sarakstā;
- Jāizvēlas ārsts no ārstēšanas sarakstā;

- Nepieciešamības gadījumā ir jānorāda vēlama ārsta specializācija. Dots darbības izpildes rezultātā ārstu sarakstā tiks atlasīti tikai tie ārsti, kuriem ir raksturīga dota specializācija;
- Jānospiež pogu „Norīkot”.

Šeit tiek arī piedāvāts saraksts izpildīto norīkojumu izvadei. Lietotājs var ievadīt pacienta uzvārdu, ka atlases kritēriju nepieciešama norīkojuma atrašanai, ka arī dzēst eksistējošo norīkojumu. Forma neļauj atkārtoti norīkot pacientu tie tā paša ārsta.



The screenshot shows a software window titled "Pacients". Inside, there is a section "Dati par pacientu" which includes a small portrait photo of a man and a "Sameklēt" button. To the right of this section is the heading "Pacienta profils". Below this heading is a form with several input fields, each with a label and a value:






- Vārds: Edgars
- Uzvārds: Naglis
- Person. k.: 160975-14555
- Tel.: 29000999
- Adrese: Rīga, Maskavas 78
- Lietotājavārds: Edgars
- Parole: *****
- Parole atkārtoti: *****
- Asins grupa: 2 (with a dropdown arrow)

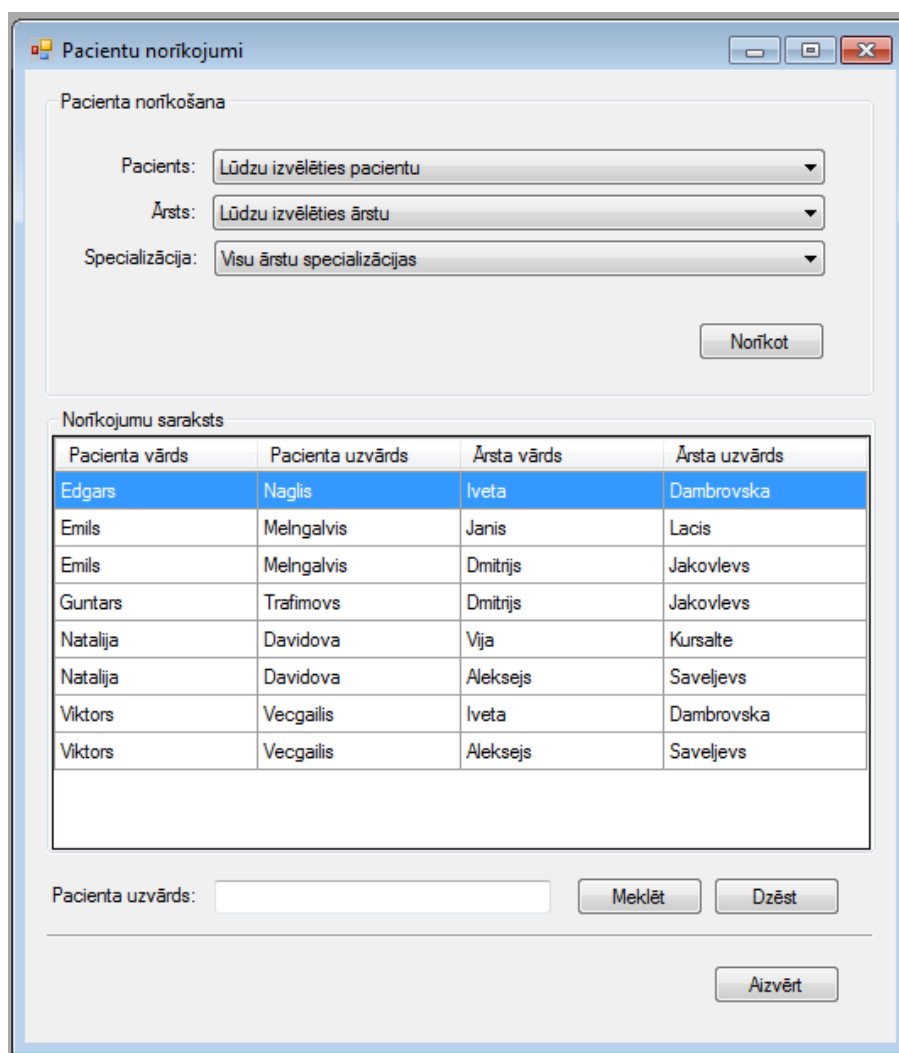
At the bottom right of the window are two buttons: "Aizvērt" and "Saglabāt datus".

4.14. att. Pacientu reģistrēšanas (datu modifikācijas) forma.

Darbība „Ārstēšana” paredz ārstēšanas epizodes aprakstu. Izpildot doto darbību, ārsti pēc pacienta apskatīšanas var uzstādīt attiecīgo diagnozi, aprakstīt pacienta veselības stāvokli, norādīt diagnozes uzstādīšanas datumu un ārstēšanai nepieciešamo terapijas kursu, ka arī pēc

pacienta atveseļošanas norādīt atveseļošanas datumu. Šādas visas darbības tiek veiktas 4.16 attēlā attēlota formā. Tajā ir pieejamas arī sekojošas papilddarbības:

- Jauns ieraksts () – darbība dzēš visos laukos ievadīto datus, piedāvājot veikt jauno ierakstu;
- Fonta maiņa () – darbība piedāvā fonta iestatījumu maiņu;
- Izgriezt () – darbība ļauj izgriezt iezīmēto teksta virkni.
- Kopēt () – darbība ļauj kopēt iezīmēto teksta virkni.
- Ielīmēt () – darbība ļauj ievietot norādīta vietā izgriezto vai kopēto tekstu.



Pacientu norīkojumi

Pacienta norīkošana

Pacients: Lūdzu izvēlēties pacientu

Ārsts: Lūdzu izvēlēties ārstu

Specializācija: Visu ārstu specializācijas

Norīkot

Norīkojumu saraksts

Pacienta vārds	Pacienta uzvārds	Ārsta vārds	Ārsta uzvārds
Edgars	Naglis	Iveta	Dambrovskā
Emils	Melngalvis	Janis	Lacis
Emils	Melngalvis	Dmitrijs	Jakovlevs
Guntars	Trafimovs	Dmitrijs	Jakovlevs
Natalija	Davidova	Vija	Kursalte
Natalija	Davidova	Aleksejs	Saveljevs
Viktors	Vecgailis	Iveta	Dambrovskā
Viktors	Vecgailis	Aleksejs	Saveljevs

Pacienta uzvārds: Meklēt Dzēst

Aizvērt

4.15. att. Pacientu norīkošanas forma.

Ārstēšana

Ārstēšanas informācija

Pacients: Edgars Naglis

Slimība: Hronisks bronhīts

Slim. uzstād. dat.: trešdiena, 2010. gada 5. maijā

Atvaseļošanas dat.: Lūdzu izvēlēties datumu

Veselības stāvoklis

Bronhīta neobstruktīva forma - nav spazmatiska elementa, gļotādas tūska ir mazāk izteikta. Bronhu caurlaidība nav traucēta. Bet ir izteikts elpas trūkums.

Terapijas kurss

Obligāta krūskurvjā masāža. Atkrepošanas līdzekļi - acetilcisteīns, bromheksīns. Svarīga regulāra lietošana, pietiekams šķidruma daudzums.

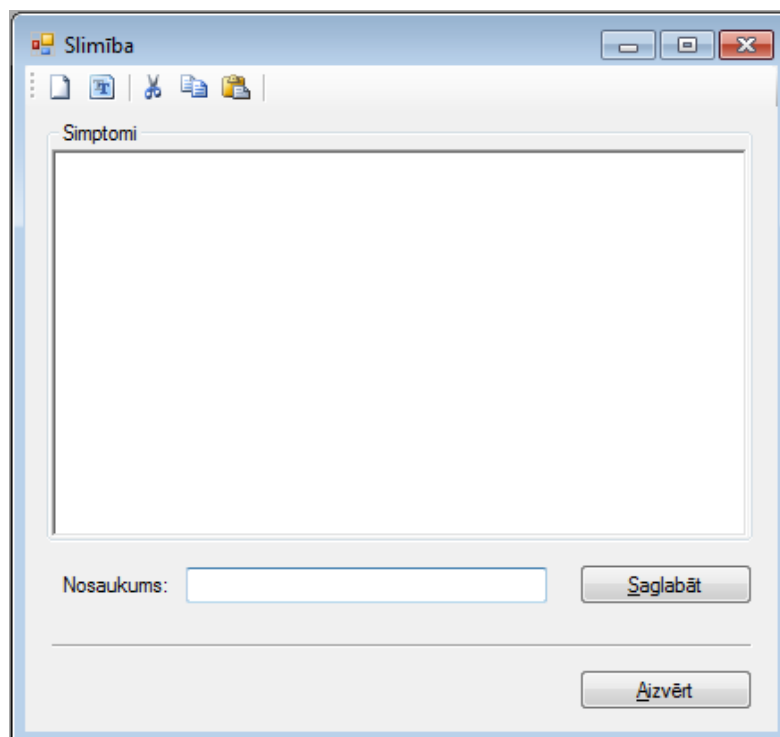
Aizvērt Saglabāt

4.16. att. Ārstēšanas forma.

Darbība „Ārstēšanas saraksts” paredz ārstēšanas epizožu izvadīti. Ārsta gadījumā forma izvadā pacienta vārdu, uzvārdu, fiksēto veselības stāvokli, slimības uzstādīšanās, atvaseļošanas datumus, terapijas kursu un diagnozi (slimības nosaukumu). Savukārt, pacienta gadījumā tiek izvadīta visa iepriekšminētā informācija, izņemot pacienta vārdu un uzvārdu, to vietā tiek izvadīts ārsta vārds un uzvārds. Dota forma ļauj pacientam apskatīties savu slimību vēsturi. Tā piedāvā līdzvērtīgu 4.13 attēla attēlotai formai darbību klāstu, kurš ir pieejams tikai ārstam. Šādas darbības ir ārstēšanas epizodes labošanas un dzēšana. Gan ārsti, gan pacienti var atlasīt sarakstā esošos datus pēc ārstu/pacientu uzvārdiem un slimību nosaukumiem.

Izvēlne „Slimība” kļūst pieejama tikai tad, kad sistēma ir autorizējies ārsts. Tā piedāvā darbības „Reģistrēt” un „Saraksts” (skatīt 4.11 c) attēlu).

Izpildot darbību „Reģistrēt”, tiks atvērta slimību reģistrēšanas forma. Tajā ārstam ir jānorāda slimības nosaukums (tiek pielietots diagnozes uzstādīšanai) un simptomi, kas raksturīgi dotai slimībai. Pēc slimības datu ievades, lietotājs var veikt šo ievaddatu saglabāšanu. Slimību reģistrēšanas forma ir attēlota 4.17 attēlā.



4.17. att. Slimību reģistrēšanas forma.

Darbība „Saraksts” paredz reģistrēto datu par slimībām izvadi. Tā piedāvā slimību dzēšanas, labošanas, saraksta atjaunošanas un atlases operācijas (atlase tiek veikta pēc slimības nosaukuma). Dota slimību saraksta forma ir identiska 4.13 attēlā attēlotai formai.

SECINĀJUMI

Mūsdienu datu pieejas tehnoloģiju industrijā ir novērojama tendence risinājumu kopas paredzēšanai datu bāžu un programmēšanas valodu integrācijas problēmu novēršanai. Eksistē divu veidu risinājumi: objektorientētas datu bāzes un objekt-relācijas kartēšanas tehnoloģijas. Bakalaura darba „datu bāžu izstrāde .NET vidē” pētījums ir saistīts ar .NET platformā piedāvāto līdzekļu izpēti šādu problēmu novēršanā.

Darba ietvaros ir identificēts relāciju datu un programmēšanas valodu integrācijas problēmu kopums. Pamatojoties uz relāciju datu modeļa un objektorientētas paradigmas atšķirībām, tiek noskaidroti doto problēmu rāšanas cēloņi. Ir aprakstīta problēmu būtība un praktiski demonstrētas dažas no šādām problēmām

Darbā ir izpētītas .NET un Java platformas datu pieejas tehnoloģijas. Ir atklāta to savstarpēja saistība. Bastoties uz identificētam problēmām, tiek noteikts risinājumu spektrs, ko piedāvā dotas tehnoloģijas problēmu novēršanai. Izmantojot tās, ir praktiski demonstrēti principi relāciju datu bāzes struktūras un objektorientēta lietojumprogrammas modeļa saistīšanai. Ir izpētītas tehnoloģiju arhitektūras un objektu modeļi. Īpaša uzmanība ir veltīta LINQ vaicājumu valodai, apskatot to plašu pielietojumu spektru un nozīmi ADO.NET, JDBC tehnoloģiju datu pieejas mehānismu trūkumu risināšanā.

Pētījumu rezultāta ir veikta apskatīto tehnoloģiju salīdzinoša analīze, kura ir balstīta uz tehnoloģiju piedāvātiem risinājumiem identificēto datu bāžu un programmēšanas valodu integrācijas problēmu novēršanai. Analizējot doto informāciju ir izvirzīti pieņēmumi, kuros gadījumos katras tehnoloģijas pielietošana būtu vispiemērotākā, ir izcelti tehnoloģiju trūkumi un priekšrocības, ka arī noteikta to lietderība.

Bakalaura darba praktiskā daļā ir izstrādāta programmatūra, kura demonstrē teorētiska materiālā apskatīto risinājumu praktisko pielietojumu. Darba gaitā ir definētas sistēmas funkcionālas prasības, vadoties pēc kurām ir izstrādāta datu bāzes shēma. Programmatūras projektēšanas posmā ir realizēts programmatūras objektu modelis, kura ir paredzēta mantošanas, „polimorfisko asociāciju” u.c. koncepciju realizācija. Ir sastādīts īss programmatūras apraksts.

Apkopojot bakalaura darbā sniegto informāciju, var secināt, ka tā ir lietderīga gan sistēmas izstrādātājiem, gan projektētājiem, kuriem nākas sastapties ar šādam problēmām. Izklāstīta informācija ļauj gūt priekšstatu par esošam paradigmu savietojamības problēmām,

tehnoloģijām, kuras piedāvā risinājumus, doto problēmu novēršanai, par pašreizējo pētījumu virzienu un var kalpot par kritēriju piemērotākas tehnoloģijas izvēlē.

LITERATŪRA

- [1] Cook R. W., Ibrahim H. A. Integrating Programming Languages & Databases: What's the Problem? – Department of Computer Sciences, University of Texas at Austin, 2005. – 18 lpp.
- [2] Крёнке Д. Теория и практика построения баз данных 8-е издание. – Москва: Питер, 2003. – 799 lpp.
- [3] Bauer C., King G. Java Persistence with Hibernate. – New York: Manning Publications Co., 2007. - 841 lpp.
- [4] Marguerie F., Eichert S., Wooley J. LINQ in Action. – Greenwich: Manning Publications Co., 2008. - 542 lpp.
- [5] MSDN: Object Identity (LINQ to SQL) / Internets. – <http://msdn.microsoft.com/en-us/library/bb399376.aspx>.
- [6] Вилдермьюс Ш. Практическое использование ADO.NET. Доступ к данным в Internet. – Москва: Издательский дом «Вильямс», 2003. – 288 lpp.
- [7] Кларк Д. Объектно-ориентированное программирование в Visual Basic .NET – Санкт-Петербург: «Питер», 2003. – 350 lpp.
- [8] Малик С. Microsoft ADO.NET 2.0 для профессионалов – Москва: Издательский дом «Вильямс», 2006. – 553 lpp.
- [9] C# 2008 и платформа .NET 3.5 для профессионалов / К. Нейгел, Б. Ивсен, Д. Глинн и др. – Москва: «Диалектика», 2009. – 1738 lpp.
- [10] Blakeley J., Campbell D., Grim J. etc. Next-Generation Data Access: Making the Conceptual Level Real / Internets. – [http://msdn.microsoft.com/en-us/library/aa730866\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/aa730866(VS.80).aspx),
pēdēja modifikācija 2006. gada jūnijs.
- [11] Neward T. Comparing LINQ and Its Contemporaries / Internets. – <http://msdn.microsoft.com/en-us/library/aa479863.aspx>,
pēdēja modifikācija 2005. gada decembris.
- [12] Mehta P. V. Pro LINQ Object Relational Mapping with C# 2008 – New York: Springer-Verlag, 2008. – 383. lpp.
- [13] MSDN: Working with Entity Data / Internets. – <http://msdn.microsoft.com/en-us/library/bb399760.aspx>.

- [14] Flasko E. Achieve Flexible Data Modeling With The Entity Framework / Internets. – <http://msdn.microsoft.com/en-us/magazine/cc700331.aspx>.
- [15] Хорстманн С. К., Корнелл Г. Java™ 2. Том II. Тонкости программирования. 7-е издание – Москва: Издательский дом «Вильямс», 2007. – 1166 lpp.
- [16] Java™ 2 Platform, Standard Edition, v 1.4.2. API Specification: Package java.sql / Internets – <http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html>.
- [17] Java™ 2 Platform Standard Edition 5.0 API Specification: Package javax.sql.rowset / Internets – <http://java.sun.com/j2se/1.5.0/docs/api/javax/sql/rowset/package-summary.html>
- [18] King G., Bauer C., Andersen R. M. etc. Hibernate Reference Documentation: Chapter 5. Basic O/R Mapping / Internets. – <http://docs.jboss.org/hibernate/core/3.3/reference/en/html/mapping.html>.
- [19] King G., Bauer C., Andersen R. M. etc. Hibernate Reference Documentation: Chapter 3. Configuration / Internets. – <http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html>.
- [20] Patricio A. Supported Databases / Internets. – <http://community.jboss.org/wiki/SupportedDatabases>,
пēdēja modifikācija 2010. gadā 15. martā.
- [21] King G., Bauer C., Andersen R. M. etc. Hibernate Reference Documentation: Chapter 9. Inheritance mapping / Internets. – <http://docs.jboss.org/hibernate/stable/core/reference/en/html/inheritance.html#inheritance-tablepersubclass>
- [22] Hibernate Pitfalls: Avoiding The N + 1 Selects Problem / Internets. – <http://www.realsolve.co.uk/site/tech/hib-tip-pitfall.php?name=n1selects>,
пēdēja modifikācija 2005. gadā 17. janvārī.
- [23] MSDN: Migrating from LINQ to SQL to Entity Framework: Deferred Loading / Internets. – <http://blogs.msdn.com/b/adonet/archive/2008/10/16/migrating-from-linq-to-sql-to-entity-framework-deferred-loading.aspx>,
пēdēja modifikācija 2008. gadā 16. oktobrī.
- [24] MSDN: Improving Entity Framework Performance / Internets. – <http://blogs.msdn.com/b/dparys/archive/2009/04/06/improving-entity-framework-performance.aspx>,
пēdēja modifikācija 2009. gadā 6. aprīlī.