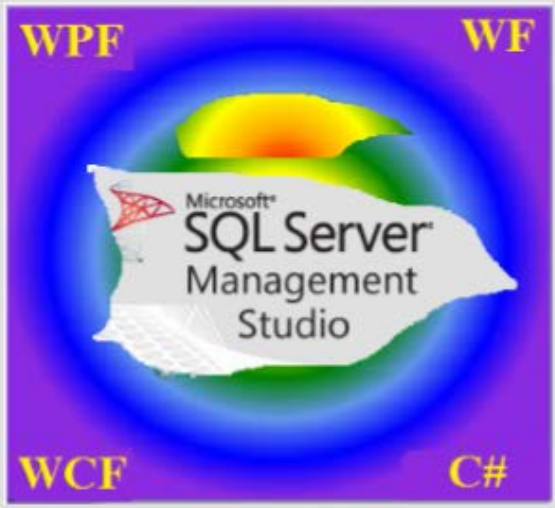
გია სურგულაძე

დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი

(WPF, WF, WCF & Ms SQL Server)



„IT-კონსალტინგის ცენტრი“

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე

დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი

(WPF, WF, WCF & Ms SQL Server)

დამტკიცებულია:
სტუ-ს „IT-კონსალტინგის"
სამეცნიერო ცენტრის
სარედაქციო-საგამომცემლო
კოლეგიის მიერ

თბილისი
2016

უაკ 004.5

განიხილება „მაიკროსოფთის" კორპორაციის ახალი ჰიბრიდული პროგრამული ტექნოლოგიები მართვის საინფორმაციო სისტემების აპლიკაციების ასაგებად. შემოთავაზებულია MsVisual Studio.NET Framework 4.0/5 ინტეგრირებულ გარემოში Windows- და Web- დანართების (კომპიუტერული სისტემების) დაპროგრამების ინსტრუმენტული საშუალებები WPF, Workflow და WCF ტექნოლოგიებით და Ms SQL Server პაკეტის ბაზაზე. იგი ეფუძნება XAML (სისტემის დიზაინის ნაწილი) და C# (სიტემის ლოგიკური ნაწილი) ენების და რელაციური ბაზების კომპლექსურ გამოყენებას. წარმოდგენილია როგორც თეორიული, UML/Agile და ITIL მეთოდოლოგიების, ასევე პრაქტიკული ღირებულების ამოცანები და მეთოდური ინსტრუქციები სხვადასხვა გამოყენებითი სფეროს ავტომატიზებული სისტემების დასაპროგრამებლად.

დამხმარე სახელმძღვანელო განკუთვნილია ინფორმატიკისა და მართვის საინფორმაციო სისტემების სპეციალობის ბაკალავრიატის მაღალი კურსის სტუდენტებისა და მაგისტრანტებისთვის.

რეცენზენტები:

- პროფ. რ. სამხარაძე
- პროფ. ე. თურქია

რედკოლეგია:

მ. ახობაძე, გ. გოგიჩაიშვილი, ზ. ბოსიკაშვილი, ე. თურქია, რ. კაკუბავა, თ. ლომინაძე, ნ. ლომინაძე, ჰ. მელაძე, გ. ნარეშელაშვილი, თ. ობგაძე, რ. სამხარაძე, გ. სურგულაძე, გ. ჩაჩანიძე, ა. ცინცაძე, ზ. წვერაიძე

ISBN 978-9941-0-8692-2

შინაარსი

შესავალი

აპლიკაციების (დანართების) ორი ნაირსახეობაა ცნობილი: ვინდოუს სისტემები, რომელთაც ასევე სამაგიდო აპლიკაციებს უწოდებენ და ვებ-აპლიკაციები, რომელთა გამოყენებაც ინტერნეტ ბრაუზერებიდანაა შესაძლებელი [1-3].

ეს დანართები იქმნება .NET Framework -ის ორი სხვადასხვა პაკეტით. პირველი - Windows Forms კომპონენტებით და მეორე ASP.NET -ის საშუალებით. ორივეს აქვს თავისი უპირატესობები და ნაკლოვანებანი. კერძოდ, სამაგიდო დანართები ძალზე მოქნილი და რეაქციულია, ხოლო Web-დანართები ინტერნეტის საშუალებით იძლევა დისტანციური წვდომის საშუალებას ერთდროულად მრავალი მომხმარებლისთვის. მაგრამ თანამედროვე კომპიუტერული ტექნოლოგიების სამყაროში ამ ორი სახის აპლიკაციებს შორის საზღვრები სულ უფრო და უფრო იშლება [4].

Web-სამსახურების და WCF (Windows Communication Foundation) სერვის-ორიენტირებული არქიტექტურის აგების საშუალებების გაჩენამ განაპირობა სამაგიდო- და ვებ-დანართების ფუნქციონირების შესაძლებლობა ერთიან განაწილებულ გარემოში, სადაც მონაცემთა გაცვლა ხორციელდება როგორც ლოკალურ, ასევე გლობალურ ქსელებში. 1-ელ ნახაზზე ნაჩვენებია ეს გამაერთიანებელი პროცესი.

WPF – Windows Presentation Foundation ერთ-ერთი ასეთი გამაერთიანებელი ტექნოლოგიაა და აგებს ისეთ დანართს, რომელშიც გამორიცხულია დაპირისპირება სამაგიდო აპლიკაციას და ინტერნეტს შორის. WPF-დანართს შეუძლია ფუნქციონირება როგორც სამაგიდო აპლიკაციის, ასევე როგორც ვებ-აპლიკაციას ბრაუზერის შიგნით. არსებობს ასევე WPF-ის შეზღუდული ვერსია, სახელით Silverlight, რომლითაც შესაძლებელია ვებ-დანართში დინამიკური მდგენელის დამატება.

ნახ. 1.

WF (Workflow Foundation) ტექნოლოგია .NET-ში არის სრულიად ახალი პარადიგმა სამუშაო (ბიზნეს) პროცესებზე (workflow) ბაზირებული აპლიკაციების ასაგებად. იგი ფუნდამენტურად ახლად გააზრებული ტექნოლოგიაა.

წიგნში ჩვენ გავეცნობით ორგანიზაციული მართვის სისტემების პროგრამული აპლიკაციების აგების და ექსპლუატაციის ამოცანების გადაწყვეტას მონაცემთა განაწილებული რელაციური ბაზის MsSQL_Server2012-ის საფუძველზე.

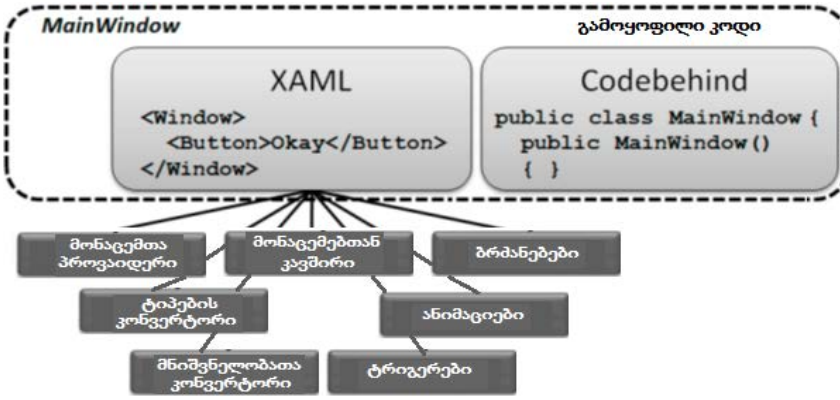
ნაშრომის სამ თავში შემოთავაზებულია დაპროგრამების აღნიშნული ჰიბრიდული ტექნოლოგიების ძირითადი ცნებები და პრაქტიკული მაგალითები, MsSQL_Server2012-ის მოკლე აღწერა, რომელიც ეხება მონაცემთა ახალი ტიპებისა და მათი მენეჯმენტის საკითხებს და ბოლოს, წარმოდგენილია კონკრეტული საპრობლემო სფეროების კომპლექსური ავტომატიზებული სისტემების დაპროგრამების სადემონსტრაციო პაკეტები. ბაკალავრიატის და მაგისტრატურის სტუდენტებს შეუძლიათ მათი გამოყენება პროტოტიპების სახით სხვადასხვა სფეროების მართვის საინფორმაციო სისტემების ასაგებად პროგრამული ინჟინერიის დისციპლინებში.

I თავი: დაპროგრამების ჰიბრიდული ტექნოლოგიები

1.1. Windows Presentation Foundation ტექნოლოგია

პრაქტიკული სამუშაოების შესასრულებლად Windows Presentation Foundation სისტემაში საჭიროა შემდეგი რესურსები:

- **ოპერაციული სისტემა** - .Net Framework-ის 3.0 (და SP1) ვერსიას აქვს Windows XP, Windows Vista და Windows Server 2003–ის მხარდაჭერა. .Net Framework 4.0/4.5-ის გამოჩენის შემდეგ იგი ბევრად სრულყოფილი გახდა. ახალი ფუნქციონალობის გამოსაყენებლად სასურველია კონფიგურაციის გაუმჯობესება:
  - Windows 7 ან Windows 10 (ან Windows Server 2008 R2);
  - გრაფიკული კარტა DirectX-ის მე-9 ვერსიისთვის;
- **ინტეგრირებული სამუშაო გარემო:** .Net Framework 4.0/4.5. 1.1 ნახაზზე ნაჩვენებია ორი ძირითადი პროგრამული ტანდემი (XAML-თავისი მეთოდებით და კოდი).



ნახ.1.1

WPF (ადრე ცნობილი იყო სახელით Avalon) არის ტექნოლოგია, რომელიც საშუალებას იძლევა დაიწეროს პლატფორმაზე დამოუკიდებელი აპლიკაციები, დიზაინისა და ფუნქციონალური

შესაძლებლობების ცხადი დაყოფით. იგი ეფუძნება ადრე არსებულ ისეთ ტექნოლოგიათა გაფართოებულ ცნებებს და კლასებს, როგორიცაა Windows Forms, ASP.NET, XML, GDI+ და ა.შ. ასეთი მსგავსება კარგად ჩანს ვებ-დანართის აწყობისას .NET Framework გარემოში [1,2].

გარდა ამისა WPF-ში მრავლადაა ახალი ფუნქციონალური საშუალებები პროგრამისტებისა და მომხმარებლებისთვის, რაც მისი, როგორც ახალი ტექნოლოგიის განხილვის უფლებას იძლევა. მაგალითად, Silverlight ტექნოლოგიას შეუძლია პროგრამები პირდაპირ ბრაუზერში შეასრულოს. ამ პლაგინის ინსტალაცია აუცილებელია. არაა დიდი მოცულობის, კომპაქტურია. ისე როგორც WPF-ში, აქაც გამოიყენება XAML და C#. მაგრამ WPF პროგრამები უშუალოდ ბრაუზერში ვერ გაიშვება.

WPF მუშაობს ვექტორულ გრაფიკაზე ბაზირებულად და არა პიქსელებზე. მართვის ელემენტები, გრაფიკები, აგრეთვე ნახაზები არ იხაზება პიქსელებით, არამედ აღიწერება მრუდებით და წრფეებით. ამის გამო აღარაა ძლიერი დამოკიდებულება მონიტორებისგან.

1.1.1. WPF აპლიკაციის შექმნა მონაცემთა ბაზებით

კორპორაციული აპლიკაცია (დანართი) პროგრამაა, რომელიც რეალიზაციას უკეთებს განსაზღვრულ ბიზნესამოცანას (ბიზნეს-ფუნქციას). დანართი უნდა მუშაობდეს მონაცემებთან, რომლებიც ინახება საინფორმაციო სისტემის მონაცემთა ბაზაში.

დანართის არქიტექტურა მოიცავს **წარმოდგენის შრეს, ბიზნესლოგიკის შრეს** და **მონაცემთა შრეს.** დანართის თითოეული შრის ფუნქციონალობა ბევრადაა დამოკიდებული საინფორმაციო სისტემის საგნობრივ სფეროზე, თუმცა არსებობს აგრეთვე ზოგადი, ფუძემდებლური ფუნქციები, რომლებიც ახასიათებს პრაქტიკულად ნებისმიერ კორპორაციულ დანართს.

ამგვარად, აპლიკაციაში უნდა დამუშავდეს წარმოდგენის შრე, რომელიც უზრუნველყოფს მომხმარებლის ინტერფეისს სისტემასთან. ინტერფეისი შეიძლება შეიქმნას Windows-ფანჯრებით და WPF გვერდებით, რომლებიც შეივსება მართვის სხვადსხვა ვიზუალური ელემენტით [5].

მართვის ელემენტები უნდა უზრუნველყოფდეს სისტემის ფუნქციონალობის ვიზუალურ წარმოდგენას მომხმარებლისათვის, აწარმოებდეს შესატანი მონაცემების ვერიფიკაციას და ურთიერთქმედებდეს ბიზნესკლასებთან.

ბიზნესლოგიკის შრე უნდა უზრუნველყოფდეს დანართის ძირითად ფუნქციონალობას: დააფორმიროს ბიზნესკლასები, რეალიზება გაუკეთოს მონაცემთა დამუშავების ალგორითმებს, უზრუნველყოს მონაცემებთან მიერთება და მათი კეშირება. ამ შრის რეალიზაცია შეიძლება განხორციელდეს კლასების საფუძველზე, რომლებიც ბიზნესლოგიკას არეალიზებს ინტერფეისული ელემენტების კლასთა მეთოდებით ან მონაცემთა მოდელის კლასთა მეთოდებით.

მონაცემთა შრე უნდა უზრუნველყოფდეს დანართის ურთიერთქმედებას ბაზის მონაცემებთან. კორპორაციულ აპლიკაციებში ამისათვის ყველაზე მიზანშეწონილია გამოყენებულ იქნას პლატფორმა ADO.NET Entity Framework და მოდელი EDM (Entity Data Model). EDM მოდელი აღწერს მონაცემთა სტრუქტურას ფიზიკური შენახვის ფორმისგან დამოუკიდებლად.

კორპორაციული დანართების დაპროექტების საკითხების შესასწავლად განვიხილოთ ძირითადი მიდგომები ინფორმაციული სისტემის ცალკეული ფუნქციების ასაგებად, რომელიც უზრუნველყოფს კომპანიის თანამშრომელთა მონაცემების დამუშავებას.

მაგალითისთვის აქ გამოვიყენებთ მონაცემთა ბაზას TitlePersonal, ცხრილებისა და ველების მცირე რაოდენობით, ხოლო

აპლიკაციის ფუნქციონალობა ითვალისწინებს კომპანიის თანამშრომელთა მონაცემების შეტანას, კორექტირებას და წაშლას. ასაგები დანართი უნდა უზრუნველყოფდეს შემდეგი მონაცემების შენახვას და გადამუშავებას: *გვარი, სახელი, სქესი, დაბადების_თარიღი, თანამდებობა, ტელეფონი, ელ_ფოსტა.*

აპლიკაციის ფუნქციებია:

- თანამშრომელთა მონაცემების დათვალიერება;
- ახალი თანამშრომლის მონაცემთა შეტანა;
- თანამშრომლის მონაცემთა რედაქტირება;
- თანამშრომლის მონაცემების წაშლა;
- მონაცემთა მოძებნა თანამშრომლის შესახებ.

განვიხილოთ „არსთა-დამოკიდებულების“ მოდელის ძირითადი დებულებები. მისი საბაზო კომპონენტებია: არსები (ობიექტები), ასოციაციები (კავშირები) და თვისებები (ატრიბუტები).

სადემონსტრაციო მაგალითზე ვაჩვენოთ მოდელის აგების პროცესი არსებული ბაზის გამოყენებით. აგებული PageEmployee დანართის გვერდისა და მონაცემთა მოდელისთვის ხორციელდება მონაცემთა მიბმა ინტერფეისის ელემენტებთან: ტექსტბოქსის, კომბობოქსის, ლისტბოქსის და თარიღის. განიხილება დანართის ურთიერთქმედების ოპერაციების დაპროექტების საკითხები მონაცემთა ბაზასთან: მონაცეთა რედაქტირება, დამატება და წაშლა. აღიწერება მონაცემთა შემოწმების შესაძლებლობა მათი შეტანისას, მომხმარებლელთა შემოწმების წესების გამოყენებით.

Entity Data Model (EDM) - არსთა მონაცემთა მოდელი („არსთა-კავშირების“ მოდელი). **EDM** მოდელი ძირითად ცნებათა ერთობლიობაა, რომელიც აღწერს მონაცემთა სტრუქტურას მისი კომპიუტერის მეხსიერებაში ფიზიკურად შენახვის ფორმისგან დამოუკიდებლად. EDM მოდელში აღწერილ მონაცემებს შეიძლება ჰქონდეს განსხვავებული სტრუქტურები: რელაციური, ტექსტური ფაილების,

XML-ის, ელექტრონული ცხრილების და რეპორტების. EDM მოდელი აღწერს მონაცემთა სტრუქტურას არსებისა და კავშირების საფუძველზე, რომლებიც დამოუკიდებელია შენახვის სქემებისგან.

ასეთი მიდგომის საფუძველზე მონაცემთა ფიზიკური დამახსოვრება განცალკევებულია დანართისაგან და არ მოქმედებს მის დამუშავებაზე. ამის უზრუნველყოფა ხდება იმის გამო, რომ არსები და კავშირები აღწერს მონაცემთა სტრუქტურებს ისე, როგორც ეს სჭირდება დანართს. EDM მოდელი კონცეპტუალური მოდელია, რომელიც აღწერს მონაცემთა სტრუქტურებს არსების და კავშირების სახით.

EDM მოდელი იყენებს სამ ძირითად ცნებას მონაცემთა სტრუქტურის აღსაწერად:

- არსის ტიპი;
- ასოციაციის ტიპი;
- თვისება.

არსის ტიპი გამოიყენება მონაცემთა სტრუქტურის აღსაწერად EDM მოდელის დახმარებით. კონცეპტუალურ მოდელში არსთა ტიპები აიგება თვისებებისგან და აღწერს ზედა დონის ძირითად კონცეპტუალურ ელემენტთა სტრუქტურას, როგორიცაა მაგალითად, თანამშრომლები და როლები. არსის ტიპი არის შაბლონი არსებისთვის. არსი არის განსაზღვრული ობიექტი (მაგალითად, რომელიმე თანამშრომელი და მისი როლი ბიზნესპროცესში).

ყოველ არსს უნდა ჰქონდეს უნიკალური გასაღები არსთა ერთობლიობის შიგნით. არსთა ერთობლიობა არის განსაზღვრული ტიპის არსის ეგზემპლართა კოლექცია. არსთა ერთობლიობები (და ასოციაციათა ერთობლიობები) ლოგიკურად დაჯგუფებულია არსთა კონტეინერებში.

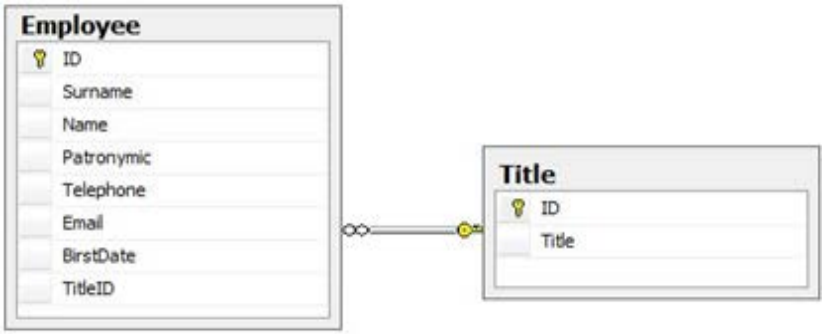
ასოციაციის ტიპი გამოიყენება კავშირთა აღწერის ასაგებად EDM მოდელში. კონცეპტუალურ მოდელში ასოციაცია ასახავს კავშირს ორი ტიპის არსებს შორის. ყოველ ასოციაციას აქვს ორი

ბოლო წერტილი, რომლებიც განსაზღვრავს არსთა ტიპებს. ისინი მონაწილეობს ასოციაციაში. ყოველი ბოლო წერტილი განსაზღვრავს მის ჯერადობას, რაც მიუთითებს არსების შესაძლო რაოდენობაზე ასოციაციის ამ ბოლო წერტილში.

არსთა ტიპები შეიცავს თვისებებს, რომლებიც განსაზღვრავს მათ სტრუქტურას და მახასიათებლებს. მაგალითად, არსის ტიპს „თანამშრომელი“ შეიძლება ჰქონდეს თვისებები: **ID, გვარი, სახელი, სქესი, დაბ_თარიღი, თანამდებობა, ტელეფონი, ელ_მისამართი** და ა.შ.

თვისებები კონცეპტუალურ მოდელში ანალოგიურია პროგრამული აპლიკაციის კლასებისა ან მონაცემთა ბაზების ატრიბუტების. თვისებები შეიძლება შეიცავდეს პრიმიტიულ მონაცემებს (სტრიქონი, მთელი რიცხვი, თარიღი, ლოგიკური მნიშვნელობა) ან სტრუქტურირებულ მონაცემებს (რთული ტიპი).

კონცეპტუალური მოდელი არის მონაცემთა სტრუქტურის სპეციფიკური ასახვა არსებისა და კავშირების სახით. 1.2 ნახაზზე ნაჩვენებია კონცეპტუალური მოდელის გამოსახვა სქემით, ბაზისათვის TitlePersonal – „თანამშრომელი“, ორი ტიპის არსით Employee – თანამშრომელი და Title –როლი/თანამდებობა), და ასოციაციური კავშირით 1:* (ერთი:მრავალთან).



ნახ.1.2. მონაცემთა ბაზის კონცეპტუალური მოდელი

Employee ცხრილი შეიცავს თანამშრომლის მონაცემებს. მისი ატრიბუტებია:

- ID – თანამშრომლის იდენტიფიკატორი;
- Surname – გვარი;
- Name – სახელი;
- Sex –სქესი;
- BirstDate – დაბადების თარიღი;
- Telephone – ტელეფონი;
- Email – ელ_მისამართი;
- TitleID – გარე გასაღები Title ცხრილთან დასაკავშირებლად.

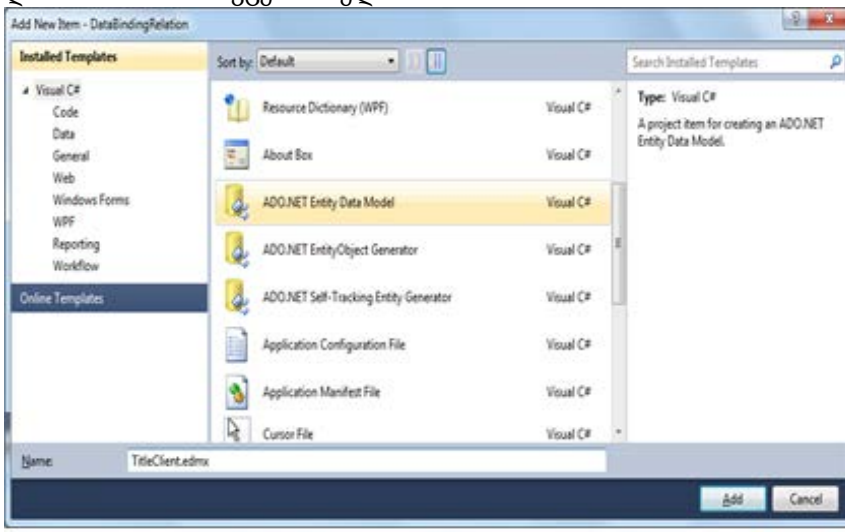
Title ცხრილი თანამდებობათა ცნობარია, რომელიც ამ ორგანიზაციას აქვს. იგი შეიცავს შემდეგ ატრიბუტებს:

- ID – თანამდებობის კოდი;
- Title – თანამდებობის დასახელება.

EDM-მოდელის შესაქმნელად პროექტში Solution Explorer-დან ჩავამატოთ ახალი ელემენტი (ნახ.1.3):

Add -> NewItem -> ADO.NET EDM

და File Name-ში მივცეთ სახელი: TitleClient.

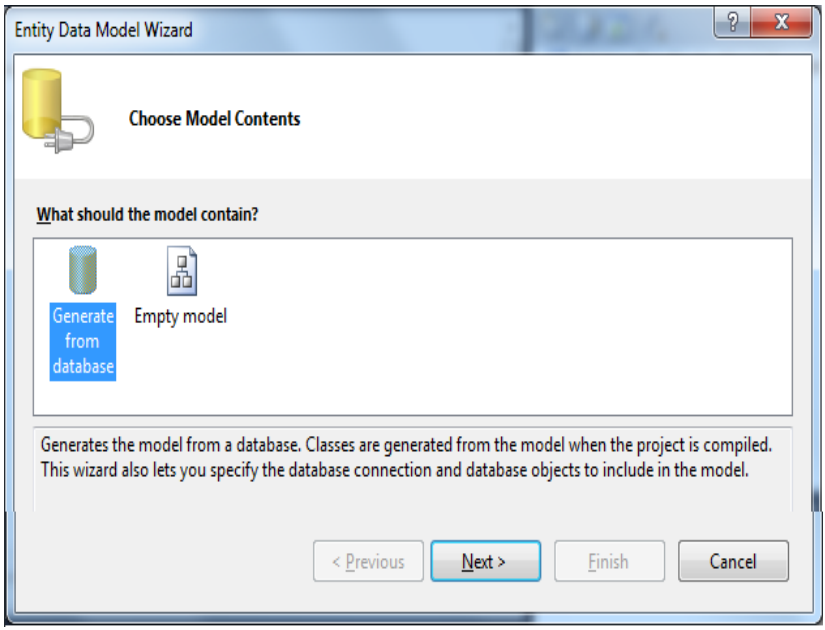


ნახ.1.3. პროექტში EDM მოდელის დამატება

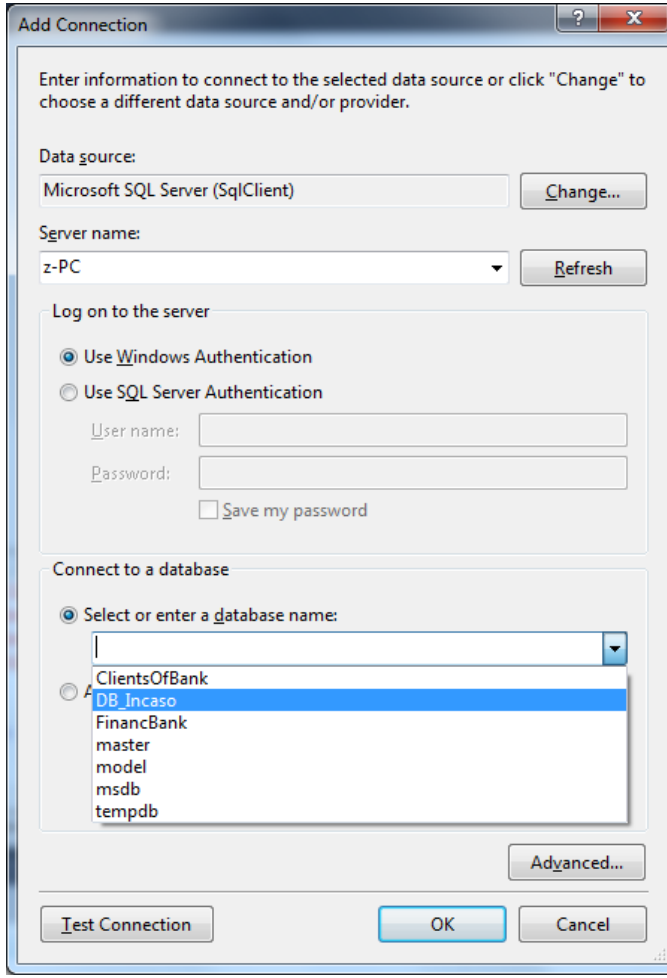
EDM მოდელის შექმნისას 1.4 ნახაზზე ნაჩვენებ Wizard-ში მივუთითოთ „შექმნა მონაცემთა ბაზიდან“, რის შემდეგაც გამოვა 1.5-(ა-გ) ფანჯრები. აქ, მონაცემთა ბაზასთან მისაერთებლად, უნდა მივცეთ:

სერვერის_სახელი.მონაცემთა_ბაზის_სახელი

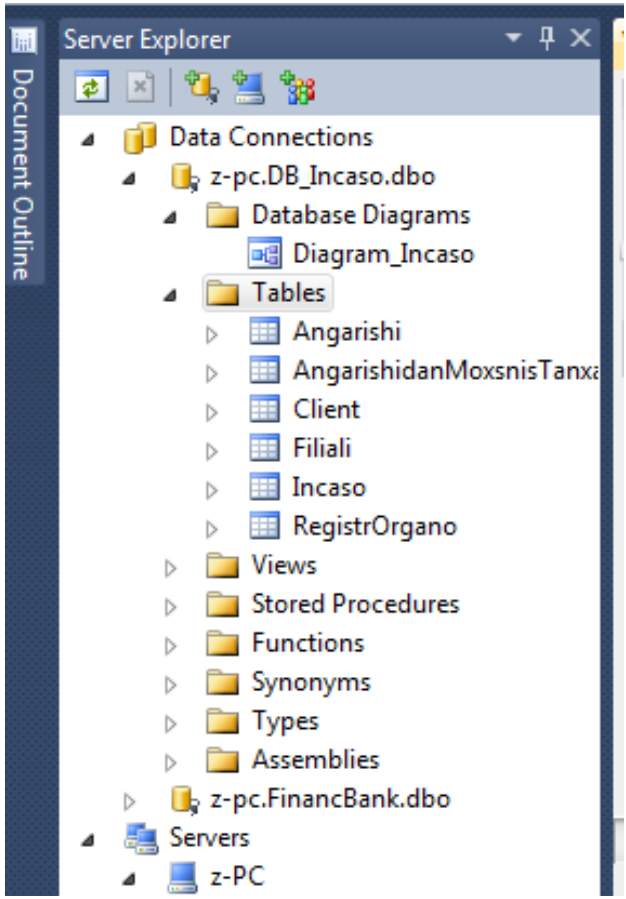
და მონაცემთა მოდელის სახელი - TitlePersonEntities (მიერთების პარამეტრების შენახვა Config-ში).



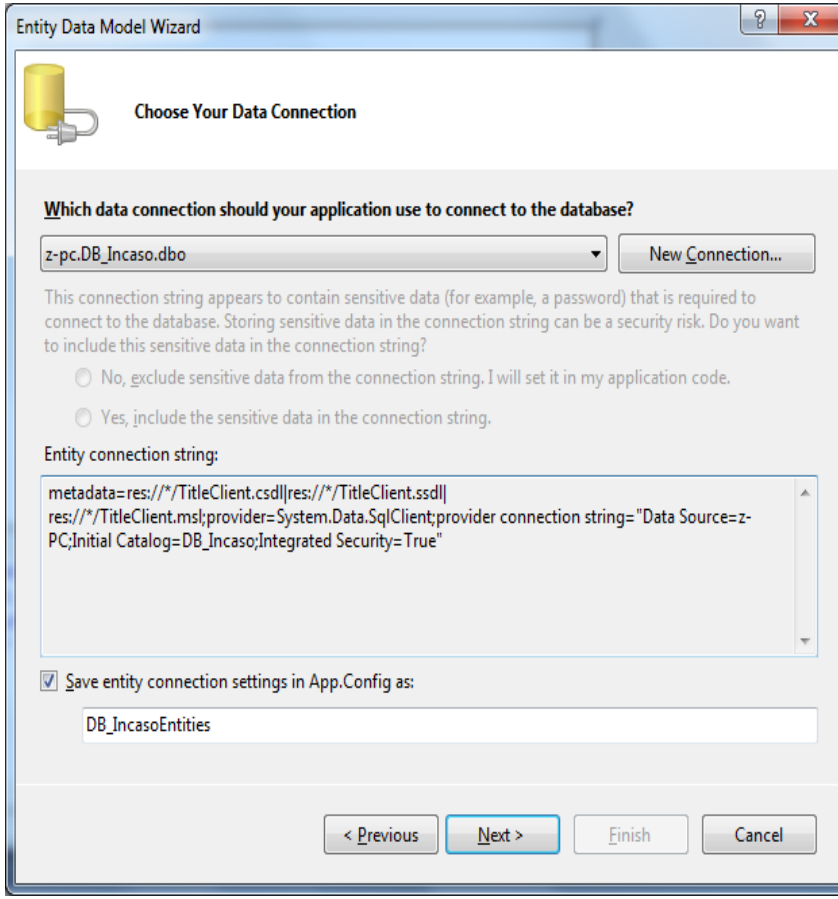
ნახ.1.4. EDM მოდელის შექმნა მონაცემთა ბაზიდან



ნახ.1.5-ა. მონაცემთა ბაზასთა მიერთება: ეტაპი_1

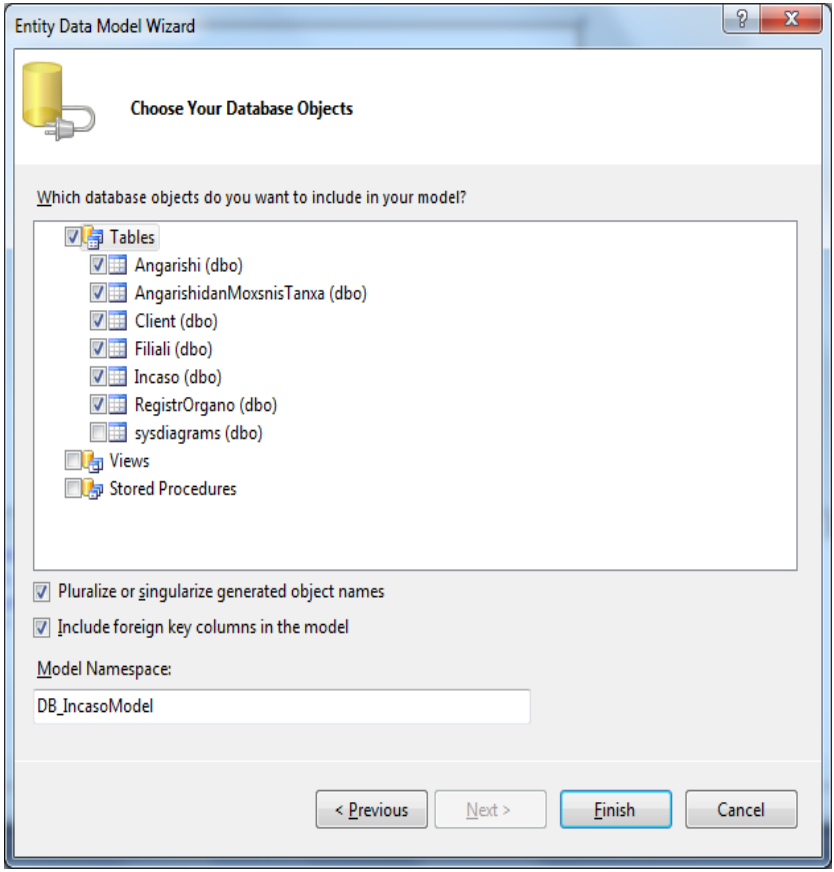


ნახ.1.5-ბ. მონაცემთა ბაზასთა მიერთება: ეტაპი_ 2



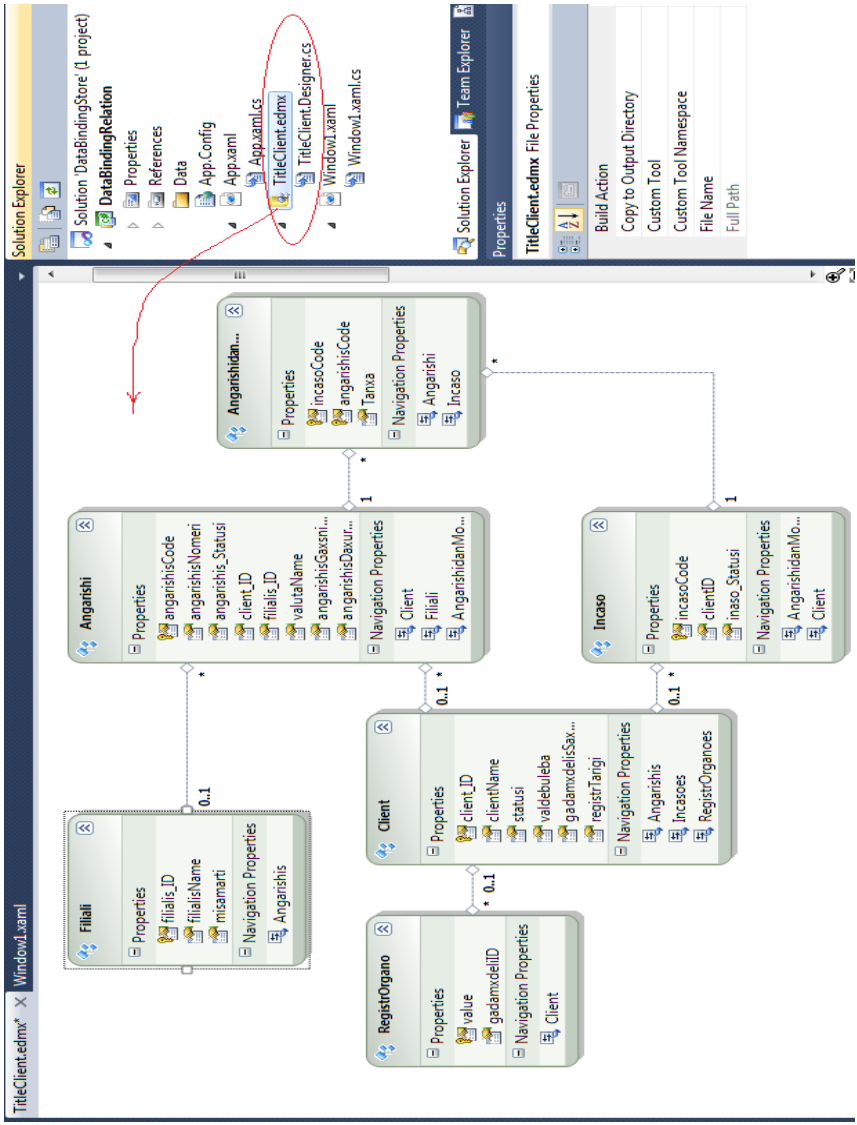
ნახ.1.5-გ. მონაცემთა ბაზასთა მიერთება: ეტაპი_3

Next ღილაკით გამოიტანება 1.6 ფანჯარა, რომლის ჩეკბოქსშიც ვირჩევთ მონაცემთა ბაზის ობიექტებს (მაგალითად, Employee და Title), მივუთითებთ ობიექტების ფორმირებას მხოლობით ან მრავლობით რიცხვში.



ნახ.1.6. მონაცემთა ბაზის ცხრილების არჩევა

EDM მოდელის შექმნის შედეგად პროექტში დამატებული იქნება TitleEmployee.edmx ფაილი (ნახ.1.7). მიმართებები საჭირო ბიბლიოთეკებზე და კონფიგურაციის ფაილი.



ნახ.1.7. პროექტი EDM მოდელით

ავტომატურად გენერირებული კლასი `DB_IncasoEntities`, რომელიც მემკვიდრეა ObjectContext კლასის, ასახავს `TitleClient` მონაცემთა ბაზის არსებს, შეიცავს თვისებებს, რომლებიც ამოდელირებს Tanamshromeli და Tanamdeboba ცხრილებს, კავშირებს მათ შორის.

მონაცემთა მოდელის შექმნისას პროექტში ავტომატურად მოხდა კონფიგურაციის App.Config ფაილის შექმნა, რომელიც შეიცავს მონაცემთა ბაზასთან მიერთების სტრიქონს.

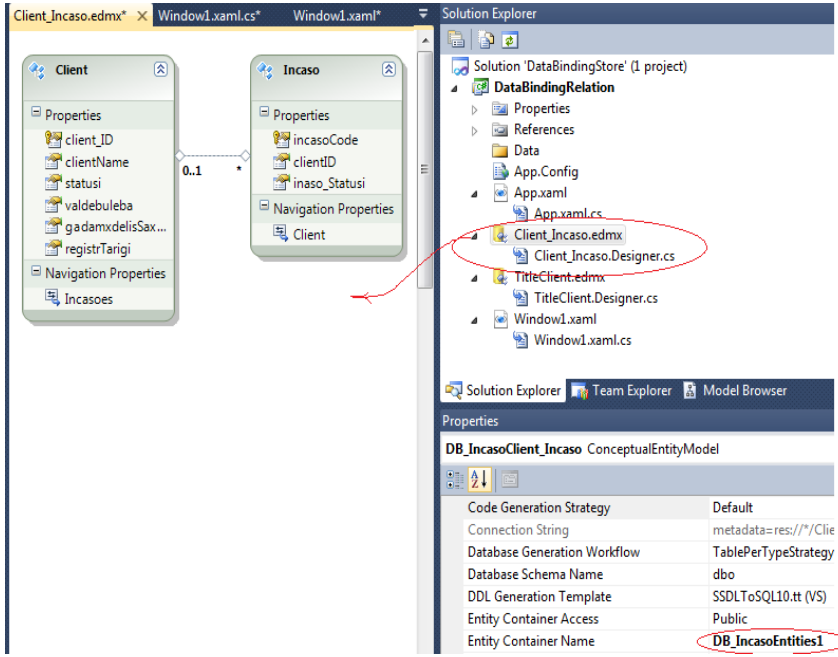
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <connectionStrings>
  <add name="DB_IncasoEntities"
    connectionString="metadata=&amp;quot;res://*/
    TitleClient.csdl|&amp;#xD;&amp;#xA;
    res://*/TitleClient.ssdl|res://*/
    TitleClient.msl&amp;quot;
    provider=System.Data.SqlClient; provider connection
    string=&amp;quot;Data Source=z-PC;Initial
    Catalog=DB_Incaso;&amp;#xD;&amp;#xA;
    Integrated
    Security=True;MultipleActiveResultSets=True&amp;quot;"
    providerName="System.Data.EntityClient" />
   <add name="DB_IncasoEntities1"
    connectionString="metadata=res://*/
    Client_Incaso.csdl|res://*/Client_Incaso.ssdl|res://*/
    Client_Incaso.msl;provider=System.Data.SqlClient;
    provider connection string=&amp;quot;
    Data Source=z-PC;Initial Catalog=DB_Incaso;Integrated
    Security=True;MultipleActiveResultSets=True&amp;quot;"
    providerName="System.Data.EntityClient" />
 </connectionStrings>
```

```
</configuration>
```

Solution Explorer-ში Edm მოდელის C#-ის TitleClient.Designer.cs -კოდის ფრაგმენტი ასეთი სახისაა:

```
// ── ლისტინგი_EDM მოდელი ──────
using System;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using System.Data.EntityClient;
using System.ComponentModel;
using System.Xml.Serialization;
using System.Runtime.Serialization;

[assembly: EdmSchemaAttribute()]
#region EDM Relationship Metadata

[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_Angarishi_Client",
   "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
   typeof(DataBindingRelation.Client), "Angarishi",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.Angarishi), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_Angarishi_Filiali",
   "Filiali",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
   typeof(DataBindingRelation.Filiali), "Angarishi",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.Angarishi), true)]
```

```
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_AngarishidanMoxsnisTanxa_Angarishi",
    "Angarishi", System.Data.Metadata.Edm.RelationshipMultiplicity.One,
    typeof(DataBindingRelation.Angarishi), "AngarishidanMoxsnisTanxa",
System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.AngarishidanMoxsnisTanxa), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_AngarishidanMoxsnisTanxa_Incaso",
    "Incaso", System.Data.Metadata.Edm.RelationshipMultiplicity.One,
    typeof(DataBindingRelation.Incaso), "AngarishidanMoxsnisTanxa",
    System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.AngarishidanMoxsnisTanxa), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_Incaso_Client",
    "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
    typeof(DataBindingRelation.Client), "Incaso",
    System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.Incaso), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_RegistrOrgano_Client",
    "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
    typeof(DataBindingRelation.Client), "RegistrOrgano",
    System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.RegistrOrgano), true)]
#endregion
namespace DataBindingRelation
{
  . . . }
```



ნახ.1.8. EDM მოდელი: Client_Incaso

```
// Client_Incaso.Designer.cs ---------------------
using System;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using System.Data.EntityClient;
using System.ComponentModel;
using System.Xml.Serialization;
using System.Runtime.Serialization;

[assembly: EdmSchemaAttribute()]
#region EDM Relationship Metadata
```

```
[assembly: EdmRelationshipAttribute("DB_IncasoClient_Incaso",
"FK_Incaso_Client",
    "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
       typeof(DataBindingRelation.Client),
    "Incaso", System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
       typeof(DataBindingRelation.Incaso), true)]
#endregion

namespace DataBindingRelation
{
. . .
}
```

1.1.2. მომხმარებლის ინტერფეისის დაკომპლექტება

პროგრამული აპლიკაციის მომხმარებლის ინტერფეისის დაპროექტების დროს აუცილებელია ფანჯარაში (ფორმაზე) ან გვერდზე საჭირო მართვის ელემენტების ფორმირება და შესაბამისი თვისებების განსაზღვრა, ანუ შინაარსის ორგანიზების ჩატარება. ამ პროცესს უწოდებენ დაკომპლექტებას (შედგენას).

WPF-ში დაკომპლექტება ხორციელდება სხვადასხვა კონტეინერით. ყოველ მათგანს თავისი საკუთარი დაკომპლექტების ლოგიკა აქვს. ზოგი ალაგებს ელემენტებს მიმდევრობით სტრიქონში, ზოგი ალაგებს მათ უხილავი უჯრების ბადეში და ა.შ.

ფანჯარას და გვერდს WPF-ში შეუძლია შეიცავდეს მხოლოდ ერთ ელემენტს - კონტეინერს. კონტეინერში შეიძლება განთავსდეს მომხმარებლის ინტერფეისის განსხვავებული ელემენტები და სხვა კონტეინერები. WPF-ში განლაგება განისაზღვრება გამოყენებული კონტეინერის ტიპით. ეს კონტეინერებია; პანელები, წარმოებული System.Windows.Controls.Panel აბსტრაქტული კლასიდან.

აპლიკაციებში გამოიყენება შემდეგი კლასები:

- Grid და UniformGrid – განალაგებს ელემენტებს უხილავი ცხრილის სტრიქონებსა და სვეტებში, შესაბამისად;
- StackPanel – განალაგებს ელემენტებს ჰორიზონტალურ ან ვერტიკალურ სვეტში.
- WrapPanel – განალაგებს ელემენტებს ხელმისაწვდომ სივრცეში, ერთ სტრიქონად ან სვეტად;
- DockPanel – განალაგებს ელემენტებს ერთ-ერთი სასაზღვრო გვერდის შეფარდებით;
- Frame – ანალოგიურია StackPanel-ის, მაგრამ უფრო მოსახერხებელია გვერდების გადასასვლელბის ორგანიზებისთვის.

Grid არის WPF-ის ყველაზე მძლავრი კონტეინერი. ის, რასაც სხვა კონტეინერები ასრულებს ცალკ-ცალკე, შეიძლება Grid-ში შერულდეს. იგი იდეალური ინსტრუმენტია ფანჯრის (გვერდის) დასაყოფად შედარებით მცირე ზომის არეებად, რომელთა მართვა განხორციელდება სხვა პანელებით. Grid ანაწილებს ელემენტებს უხილავი ბადის სტრიქონებსა და სვეტებში. ბადის ერთ უჯრაში მიზანშეწონილია ერთი ელემენტის მოთავსება, რომელიც, საჭიროების შემთხვევაში, თვითონ შეიძლება იყოს სხვა კონტეინერი, რომელშიც განლაგდება საკუთარი მართვის ელემენტთა ჯგუფი.

StackPanel - ერთ-ერთი უმარტივესი კონტეინერია. იგი ალაგებს თავის შვილობილ ელემენტებს ერთ სტრიქონში ან სვეტში.

UniformGrid კონტეინერი, Grid-ისგან განსხვავებით, მოითხოვს მხოლოდ სტრიქონების და სვეტების რაოდენობის მითითებას, და აფორმირებს ერთი ზომის უჯრებს, რომელთაც დაკავებული აქვს ფანჯრის (გვერდის) ან ჩადგმული კონტეინერის ელემენტის მთელი ხელმისაწვდომი არე.

WrapPanel აწესრიგებს ელემენტების განლაგებას პანელზე Orientation თვისების შესაბამისად, ჰორიზონტალურად (Horizontal) ან ვერტიკალურად (Vertical). სტრიქონის ან სვეტის შევსების შემდეგ მომდევნო ელემენტი გადადის ახალ სტრიქონზე ან სვეტზე.

DockPanel უახლოვებს მართვის ელემენტებს მის რომელიმე მხარეს, Dock თვისების შესაბამისად, რომელიც იქნება Left, Right, Top ან Bottom. ელემენტის სიმაღლე განისაზღვრება MaxHeight პარამეტრით.

Frame მართვის ელემენტია შიგთავსისთვის, რომელიც იძლევა შესაძლებლობას შიგთავსზე ან მის ასახვაზე გადასასვლელად. Frame შეიძლება მოთავსდეს სხვა შიგთავსის შიგნით, როგორც სხვა ელემენტები. შიგთავსი შეიძლება იყოს .NET Framework-ის და HTML-ფაილების ნებისმიერი ტიპი. ჩვეულებისამებრ, Frame გამოიყენება შიგთავსის ჩასაწყობად, რომელიც განსაზღვრავს გადასასვლელებს გვერდებზე.

დაკომპლექტების თვისებები განისაზღვრება კონტეინერით, მაგრამ შვილობილი ელემენტებიც ახდენს მასზე გალენას. დაკომპლექტების პანელები მუშაობს შვილობილ ელემენტებთან შეთანხმებით, შემდეგი თვისებების საუძველზე:

- HorizontalAlignment და VerticalAlignment - განსაზღვრავს, თუ როგორ პოზიციონირებს შვილობილი ელემენტი კომპლექტის შიგნით, როცა არსებობს დამატებითი ჰორიზონტალური/ვერტი-კალური სივრცე;
- Margin - ამატებს ცარიელ სივრცეს ელემენტის ირგვლივ;
- MinWidth / MaxWidth აყენებს ელემენტის მაქსიმალურ ზომებს;
- Width и Height – ცხადად აყენებს ელემენტის ზომებს.

WPF-ში არსებობს კონტეინერები, რომლებსაც აქვს მართვის ელემენტები მოცემული კოორდინატების შესაბამისად, რომლებიც ზომებითაა მოცემული. ეს კონტეინერებია Canvas და InkCanvas. ისინი ძირითადად გამოიყენება გრაფიკული პრიმიტივების და ფიგურების ვიზუალიზაციისთვის.

1.1.3. XAML ენის საფუძვლები

მომხმარებელთა ინტერფეისების აგება WPF- და Silverlight-დანართებისთვის (აპლიკაციებისთვის) ხორციელდება XAML (Extensible Application Markup Language - აპლიკაციების გაფართოებადი ფორმატირების ენა) ენის გამოყენებით. XAML-დოკუმენტი შეიცავს ფორმატს, რომელიც აღწერს დანართის ფანჯრის (ან გვერდის) გარეგან სახეს და ქცევას, ხოლო მასთან კავშირში მყოფი C# კოდის ფაილები კი - დანართის ლოგიკას. XAML-ენა უზრუნველყოფს დანართის დიზაინის პროცესის (გრაფიკული ნაწილი) გამოყოფას ბიზნეს-ლოგიკის (პროგრამული კოდი) დამუშავების პროცესისგან, დიზაინერებსა და დეველოპერებს შორის [4,5].

WPF-ის XAML არის XML-ენის ქვესიმრავლე, გაფართოებული დამატებითი ფუნქციებით. იგი უზრუნველყოფს WPF-შიგთავსის აღწერას ისეთი ელემენტებით, როგორიცაა ვექტორული გრაფიკა, მართვის ელემენტები და დოკუმენტები.

XAML-ის საფუძველია XML და მისი სინტაკსი განისაზღვრება შემდეგი წესებით:

- XAML-დოკუმენტის ყოველი ელემენტი აისახება .NET კლასის რომელიღაც ეგზემპლარში. ასეთი ელემენტის სახელი ზუსტად შეესაბამება კლასის სახელს. მაგალითად, <Button> ელემენტი ემსახურება WPF-ინსტრუქციას Button-კლასის ობიექტის აგების მიზნით;
- XAML-ის ელემენტები შეიძლება ერთმანეთში ჩალაგდეს. ელემენტების ჩალაგების ფორმატი ასახავს ინტერფეისის ელემენტების ჩალაგებას;
- კლასის თვისებები განისაზღვრება ატრიბუტებით ან ჩალაგებული დესკრიპტორების დახმარებით, სპეცსინტაკსით.

XAML-ენა ხასიათდება თვითაღწერადობით. XAML-დოკუმენტში ყოველი ელემენტი არის ტიპის სახელი (მაგალითად, Button, Window ან Page) მოცემული სახელსივრცის ჩარჩოებში.

ელემენტთა ატრიბუტები გამოიყენება შესაბამისი ობიექტების თვისებების (მაგალითად, Name, Height, Width და ა.შ.) და მოვლენების (Click, Load და ა.შ) მოსაცემად.

WPF-დანართის MyFirstWpfProject შექმნის დროს VisualStudio აგენერირებს შემდეგ XAML-დოკუმენტს.

```
<Window x:Class="MyFirstWpfProject.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/
                                        2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
  <Grid>
  . . .
  </Grid>
</Window>
```

WPF-დანართის XAML-დოკუმენტი MyFirstWpfProject იწყება დესკრიპტორით < Window...>.

XAML-დოკუმენტის ყველა დესკრიპტორი იწყება „<“ - სიმბოლოთი და მთავრდება „>“ - სიმბოლოთი. ნებისმიერი XAML-დოკუმენტი შედგება XAML-ელემენტებისგან. ყოველი XAML-დოკუმენტი (XAML-ელემენტი) იწყება გახსნის დესკრიპტორით (მაგალითად, < Window >), რომელსაც მოჰყვება დოკუმენტის შიგთავსი (მაგალითად, ტექსტური სტრიქონი ან სხვა XAML-ელემენტები). გახსნის დესკრიპტორში შეიძლება იყოს მოთავსებული ატრიბუტების აღწერა (მაგალითად, Class, xmlns, Title, Height, Width და

სხვ.). XAML-დოკუმენტი (XAML-ელემენტი) უნდა დასრულდეს დახურვის დესკრიპტორით (მაგალითად, „/>“ ან „</Window>“).

XAML-დოკუმენტის ტექსტი უნდა შეიცავდეს ერთ ფესვურ ელემენტს - ჩალაგების უმაღლესი დონის ელემენტი. WPF-დანართის MyFirstWpfProject XAML-დოკუმენტში ასეთი ელემენტია <Window>. ფესვურ ელემენტში შეიძლება დაემატოს XAML-ის სხვა ელემენტებიც. მაგალითში ასეთი ელემენტია <Grid>.

WPF-დანართის XAML-დოკუმენტის კომპილაციის პროცესში სინტაკსურ ანალიზატორს გადაჰყავს XAML ფაილები აპლიკაციის ორობითი ფორმატირების ფაილებში BAML (Binary Application Markup Language), რომლებიც შემდეგ ჩაშენდება პროექტის ნაკრებში რესურსების სახით. WPF-დანართის კლასების ასაგებად სინტაკსური ანალიზატორი გამოიყენებს სახელსივრცეს, რომელიც განსაზღვრულია XAML-დოკუმენტის ფესვურ დესკრიპტორში.

XAML-დოკუმენტში სახელსივრცე მოიცემა xmlns ატრიბუტის საშუალებით. ზემოაღწერილ დოკუმენტში გამოცხადებულია ორი საბაზო სახელსივრცე:

- xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation - ესაა WPF-ის საბაზო სახელსივრცე, რომელიც მოიცავს WPF-ის ყველა კლასს, მართვის ელემენტების ჩათვლით. ისინი გამოიყენება მომხმარებლის ინტერფეისის ასაგებად. ვინაიდან სახელსივრცე ცხადდება პრეფიქსის გარეშე, იგი ვრცელდება მთელი XAML-დოკუმენტისთვის;
- xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" - ესაა XAML-ის სახელსივრცე. იგი შეიცავს XAML უტილიტების სხვადასხვა თვისებებს, რომლებიც გავლენას ახდენს იმაზე, თუ XAML-დოკუმენტი როგორ ინტერპრეტირდება. მოცემული სახელსივრცე აისახება x პრეფიქსზე. ეს პრეფიქსი შეიძლება მოთავსდეს ელემენტის სახელის წინ (მაგ., x:ელემენტის_სახელი).

მეორე სახელსივრცე გამოიყენება XAML-ის სპეციფიური ლექსემების („საკვანძო სიტყვები“) ჩასასმელად. 1.1 ცხრილში მოცემულია შედარებით ხშირად გამოყენებადი ასეთი სიტყვები.

XAML-ის საკვანძო სიტყვები ცხრ.1.1

N	საკვანძო სიტყვა	დანიშნულება
1.	x:Array	წარმოადგენს .NET-ის მასივის ტიპს XAML-ზე
2.	x:Class	XAML-ფაილის კლასის სახელი
3.	x:ClassModifier	უზრუნველყოფს კლასის ტიპის ხილვადობის (internal ან public) განსაზღვრას, რომელიც Class საკვანძო სიტყვითაა აღნიშნული
4.	x:Code	პროგრამული კოდი შეიძლება უშუალოდ ჩაიდოს XAML-კოდში
5.	x:FieldModifier	უზრუნველყოფს ტიპის წევრის ხილვადობის (internal, public, private ან protected) განსაზღვრას ფესვის ნებისმიერი სახელმინიჭებული ელემენტისთვის (საკვანძო სიტყვით Name)
6.	x:Key	უზრუნველყოფს გასაღების მნიშვნელობის დაყენებას XAML ელემენტეისთვის, რომელიც უნდა მოთავსდეს ლექსიკონის ელემენტში
7.	x:Name	უზრუნველყოფს C#-ით გენერირებული სახელის მითითებას მოცემული XAML ელემენტისთვის
8.	x:Null	წარმოადგენს null-მიმთითებელს
9.	x:Shared	შესაძლებელია მხოლოდ იმ რესურსებისთვის, რომლებიც ერთხელ იძლევა ეგზემპლარს. ჩვეულებრივად, ყოველი წვდომისას იწარმოება რესურსის ახალი ეგზემპლარი
10.	x:Static	უზრუნველყოფს ტიპის სტატიკურ წევრზე მიმართვას

11.	x:Subclass	ეს კონსტრუქცია ქმნის წარმოებულ კლასს და გამოდგება დაპროგრამების მხოლოდ იმ ენებისთვის, რომელთაც არ აქვს დანაწევრებული (partielle) კლასების მხარდაჭერა
12.	x:Type	XAML-ეკვივალენტია C#-ია typeof ოპერაციის (იძახებს System.Type მითითებული სახელის საფუძველზე)
13.	x:TypeArgument	უზრუნველყოფს ელემენტის დაყენებას, როგორც განზოგადებული ტიპისას განსაზღვრული პარამეტრებით
14.	x:Uid	x:Name –ს პარალელურად შეუძლია მართვის ელემენტს ამ ატრიბუტით ცალსახა სახელი მიიღოს, რომელიც შემდგომიმთარგმანისთვის იქნება გამოყენებული
15	x:XData	ქმნის „მონაცემთა კუნძულს“ XAML-ის შიგნით, შეუძლია მარტივი XML-მონაცემთა კონსტრუქციით წარმოება
შენიშვნა: 2, 4, 9, 11, 14, 15		VS-2012 ვერსიაში დამატებული საკვანძო სიტყვები

WPF-დანართებში საბაზო სახელსივრცეთა გარდა იყენებენ ასევე სპეციალურს, რომლებიც არააუცილებელია:

- http://schemas.openxmlformats.org/markup-compatibility/2006 - XAML-ის სახელსივრცეა, დაკავშირებული ფორმატირების თავსებადობის პრობლემასთან სამუშაო გარემოსთან. ეს სახელსივრცე გამოიყენება XAML-ის სინტაკსური ანალიზატორის ინფორმირებისათვის იმის შესახებ, თუ რომელი ინფორმაციაა დასამუშავებელი და რომელი საიგნორირო;
- http://schemas.microsoft.com/expression/blend/2008 - XAML-ის სახელსივრცეა, რომელსაც აქვს მხარდაჭერა Expression Blend და Visual Studio პროგრამებიდან. გამოიყენება გვერდის გრაფიკული პანელის ზომების დასაყენებლად.

Window ობიექტის ატრიბუტებში შეიძლება დაემატოს შემდეგი XAML-აღწერები:

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="600"
```

მოცემული XAML-აღწერა აცხადებს არასავალდებულო სახელსივრცეებს პრეფიქსებით mc და d. თვისებები DesignHeight და DesignWidth იმყოფება სახელსივრცეში, რომელსაც აქვს პრეფიქსი d. ეს თვისებები განსაზღვრავს, რომ დანართის პროექტის დამუშავებისას Visual Studio დიზაინერში ფანჯარას უნდა ჰქონდეს ზომები 300x600. თვისება Ignorable მდებარეობს სახელსივრცეში, რომელიც აღნიშნულია პრეფიქსით mc და ის სინტაკსურ ანალიზატორს აინფორმირებს, რომ მან იგნორირება გაუკეთოს XAML-დოკუმენტის ნაწილს, რომელიც აღნიშნულია d პრეფიქსით.

WPF-დანართის XAML-დოკუმენტში ხშირად საჭიროა განხორციელდეს წვდომა პროექტის სხვა რომელიმე სახელსივრცესთან. ამ დროს აუცილებელია ახალი პრეფიქსის განსაზღვრა და მიეცეს სახელსივრცე. თუ პროექტში არის სახელსივრცე MyFirstWpfProject.Commands, მაშინ მის მიერთებას WPF -დანართის XAML-დოკუმენტთან ექნება შემდეგი სახე (command - გამოიყენება პრეფიქსის სახით).

```
xmlns:command="clr-namespace:
 MyFirstWpfProject.Commands"
```

პრეფიქსი (command) გამოიყენება მიმართვისთვის სახელსივრცეზე XAML-დოკუმენტში. clr-namespace ლექსემს ენიჭება სახელსივრცის დასახელება .NET ნაკრებში.

XAML-დოკუმენტში კლასის აღსაწერად გამოიყენება ატრიბუტი Class. XAML-დოკუმენტის სტრიქონი

```
<Window x:Class="MyFirstWpfProject.MainWindow" ...>
```

ითვალისწინებს MyFirstWpfProject.MainWindow კლასის შექმნას Window კლასის ბაზაზე. Class ატრიბუტის x პრეფიქსი განსაზღვრავს იმას, რომ ეს ატრიბუტი თავსდება XAML-ის სახელსივრცეში.

MainWindow კლასი გენერირდება ავტომატურად კომპილაციის დროს. კლასის ნაწილისთვის ავტომატურად გენერირდება კოდი (ნაწილობრივი (partial) კლასი):

```
namespace MyFirstWpfProject
{
  // <summary>
  // ურთიერთქმედების ლოგიკა MainWindow.xaml - ისთვის
  // </summary>
  public partial class MainWindow : Window
   {
    public MainWindow()
     {
       InitializeComponent();
     }
   }
}
```

როდესაც სრულდება დანართის კომპილაცია, XAML-ფაილი, რომელიც განსაზღვრავს მომხმარებლის ინტერფეისს (MainWindow.xaml), ტრანსლირდება CLR ტიპის გამოცხადებაში, რომელიც ერთიანდება დანართის ლოგიკასთან გამოყოფილი კოდის კლასის ფაილიდან (MainWindow.xaml.cs).

InitializeComponent() მეთოდი გენერირდება დანართის კომპილაციის დროს და საწყის კოდში არ თავსდება.

XAML-დოკუმენტში აღწერილი მართვის ელემენტების პროგრამულად სამართავად, აუცილებელია მართვის ელემენტს მიეცეს XAML ატრიბუტი Name. მაგალითად, Grid ელემენტისთვის ჩაიწერება ასე:

```
<Grid Name="grid">

</Grid>
```

მარტივი თვისებები XAML-დოკუმენტში მოიცემა შემდეგი სინტაკსის შესაბამისად:

თვისების_სახელი = "მნიშვნელობა"

მაგალითად, **Name = "grid1"**

თვისების მისაცემად, რომელიც არის სრულფასოვანი ობიექტი, გამოიყენება *რთული თვისებები* „თვისება-ელემენტი“ სინტაკსის შესაბამისად:

მშობელი.თვისების_სახელი

მაგალითად, StackPanel კონტეინერისთვის აუცილებელია მიეცეს გრადიენტული ფუნჯი პანელის შესავსებად, რაც განისაზღვრება Background ატრიბუტით. იგი რეალიზდება დესკრიპტორებით:

<StackPanel.Background> . . . </StackPanel.Background>.

თვისების მნიშვნელობის მისაცემად გამოყოფილი კლასიდან გამოიყენება ფორმატირების გაფართოება, რომელიც უზრუნველყოფს XAML გრამატიკის გაფართოებას ახალი ფუნქციონალობით. ფორმატირების გაფართოება შეიძლება გამოყენებულ იქნას ჩალაგებულ დესკრიპტორებში ან XAML-ატრიბუტებში. როცა იყენებენ ატრიბუტებს, მაშინ აუცილებელია ფიგურული ფრჩხილების {...} გამოყენება.

ფორმატირების გაფართოებები იყენებს შემდეგ სინტაკსს:

{ფორმატირების_გაფართოების_კლასი არგუმენტი}

ფორმატირების გაფართოებები რეალიზდება კლასებით, რომლებიც შვილობილია System.Windows.Markup.MarkupExtention კლასის. MarkupExtention საბაზო კლასს აქვს ProvideValue() მეთოდი, რომელიც იძლევა ატრიბუტისთვის საჭირო მნიშვნელობას. მაგალითად, იმისათვის, რომ Button-ობიექტის Foreground ატრიბუტს მიეცეს სტატიკური თვისება, რომელიც სხვა კლასშია განსაზღვრული, აუცილებელია შემდეგი XAML-აღწერის შექმნა:

```
<Button Foreground="{x:Static
SystemColors.ActiveCaptionBrush}"    />
```

კომპილაციის დროს სინტაკსური ანალიზატორი შექმნის Static Extention კლასის ეგზემპლიარს, შემდეგ გამოიძახებს ProvideValue() მეთოდს, რომელიც ამოიღებს საჭირო მნიშვნელობას და დააყენებს მას Foreground თვისებისთვის.

ფორმატირების გაფართოებები შეიძლება გამოყენებულ იქნას როგორც ჩალაგებული თვისებები.

მიერთებული თვისებები აღწერს თვისებებს, რომელთა გამოყენება შეიძლება რამდენიმე მართვის ელემენტთან, ოღონდ რომლებიც განსაზღვრულია სხვა კლასში. WPF-დანართებში მიერთებული თვისებები ხშირად გამოიყენება ინტერფეისის ელემენტების დაკომპლექტების სამართავად. მიერთებული თვისებების სინტაკსი შემდეგია:

განმსაზღვრელი_ტიპი.თვისების_სახელი

მაგალითად, თუ საჭიროა ღილაკის მოთავსება ბადის 0-ოვან სტრიქონში, მაშინ აუცილებელია შემდეგი XAML აღწერის შექმნა:

```
<Button ... Grid.Row="0" >
....
</Button>
```

აქ მიერთებული თვისებაა Grid.Row, ანუ Grid-ელემენტის Row-თვისება, რომელიც არაა Button ობიექტის თვისება. თვისება Row მიუერთდება Button ობიექტის თვისებებს, ვინაიდან ეს ობიექტი განთავსებულია Grid კონტეინერში.

ობიექტის ატრიბუტები შეიძლება გამოყენებულ იქნას მოვლენათა დამმუშავებლების მისაერთებლად, შემდეგი სინტაკსის გამოყენებით:

მოვლენის_სახელი = "მოვლენის_დამმუშავებლის_მეთოდის_სახელი"

მაგ., ღილაკის Click მოვლენისთვის (მისი დაჭერისას), შეიძლება დაყენდეს მოვლენის დამმუშავებელი Exit_Click.

```
<Button Name="Exit" Content="გამოსვლა"
        Click="Exit_Click" />
```

XAML-აღწერაში Exit_Click დამმუშავებლის განსაზღვრისას, აუცილებელია კლასის კოდში გვქონდეს მეთოდი კორექტული სიგნატურით. ქვემოთ მოყვანილია კოდი, რომელიც გენერირდება ავტომატურად მოვლენის დამმუშავებლის აღწერის შექმნისას XAML-დოკუმენტში.

```
private void Exit_Click(object sender,RoutedEventArgs e)
        {
        . . .
        }
```

1.1.4. Web-გვერდების აპლიკაციების შექმნა

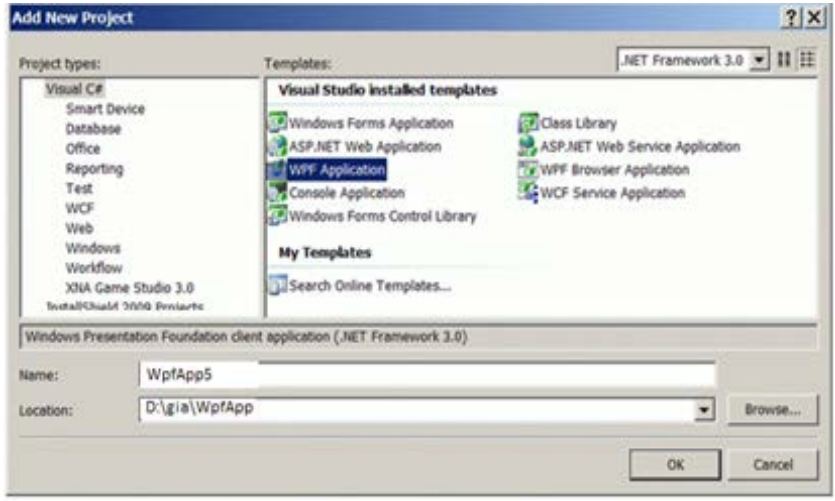
განხილულია WPF-ტექნოლოგიით ვებ-აპლიკაციის დამუშავება მომხმარებელთა ინტერფეისების ვიზუალური მართვის ელემენტებით.

WPF-პლატფორმას აქვს ვებ-გვერდების შექმნის საშუალებები. ასეთი გვერდების გამოყენება ხშირად მომხმარებლისთვის უფრო მოსახერხებელია, ვიდრე ფანჯრული ორგანიზაცია.

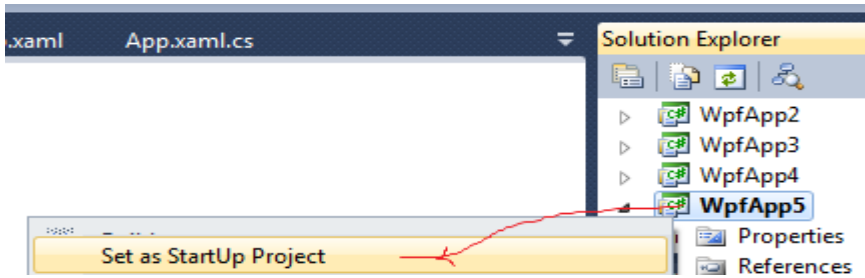
გვერდული დანართის (აპლიკაციის) შიგთავსი ჩაშენდება სპეციალურ ნავიგაციურ კარკასში, რომელსაც აქვს ნავიგაციური კავშირები და ნავიგაციური ჟურნალი.

მისი ძირითადი (ფესვური) კლასია NavigationWindow, რომელიც ამატებს აპლიკაციისთვის **ნავიგაციის სტანდარტულ ინტერფეისს** და მისთვის საჭირო ინფრასტრუქტურას. NavigationWindow კლასი წარმოებულია Window-კლასიდან და აქვს წვდომა დანართის იგივე საშუალებებზე.

1. შევქმნათ ახალი პროექტი WpfApp5 სახელით (ნახ.1.9) და გავხადოთ „სასტარტო“ (ნახ.1.10).



ნახ.1.9



ნახ.1.10

2. წავშალოთ ავტომატურად შექმნილი Window1.xaml ფაილი SolutionExplorer-ში და ჩავამატოთ მის ნაცვლად Window(WPF)-შაბლონით ახალი ფაილი NavExample.xaml სახელით.

3. გავხსნათ App.xaml ფაილი და შევცვალოთ მასში ატრიბუტი StartupUri="NavExample.xaml"

4. ავამუშავოთ პროექტი WpfApp5. დავრწმუნდეთ, რომ არაა შეცდომები.

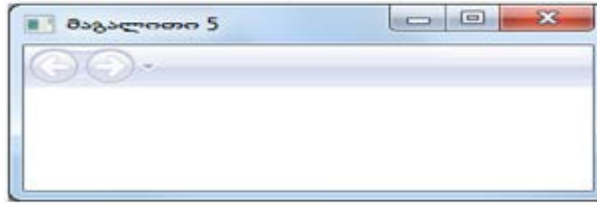
5. გავხსნათ ფაილი NavExample.xaml და შევასწოროთ მასში დესკრიპტორული კოდი:

```
<NavigationWindow x:Class="WpfApp5.NavExample"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="მაგალითი 5"
    Height="300"
    Width="300"
    WindowStartupLocation="CenterScreen"
        >
</NavigationWindow>
```

6. გავხსნათ NavExample.xaml.cs ფაილი და შევასწოროთ C# კოდის ტექსტი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

using System.Windows.Navigation;

namespace WpfApp5
{
    public partial class NavExample : NavigationWindow
    {
        public NavExample()
        {
            InitializeComponent();

            //this.Navigate(new Page1());
        }
    }
}
```

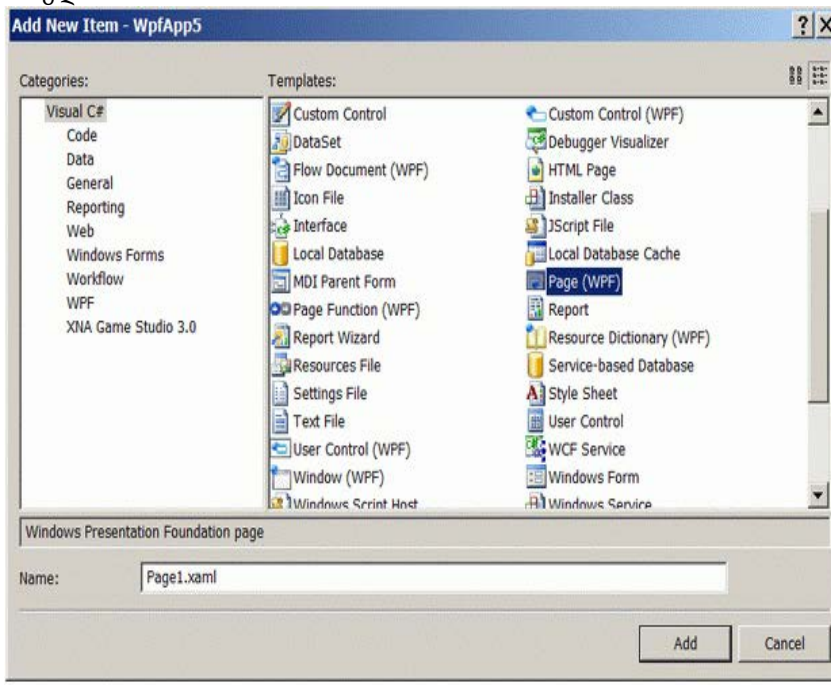
ავამუშავოთ პროექტი, შედეგი იქნება ნახ.1.11, ცარიელი გვერდი ნავიგაციური ელემენტით.



ნახ.1.11

7. ნავიგაციური კვანძის შიგთავსი წარმოდგენილი უნდა იყოს კლასით, რომელიც წარმოებულია ბიბლიოთეკის Page-კლასისგან. შევქმნათ სამი გვერდი და მივაბათ ნავიგაციის კვანძს Navigate() მეთოდის დახმარებით.

• დავამატოთ პროექტს ახალი Page ელემენტი Page1.xaml სახელით.



ნახ.1.12

8. შევასწოროთ Page1.xaml ფაილის ტექსტი:

```
<Page x:Class="WpfApp5.Page1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center" FontSize="24">გვერდი
1</TextBlock>
        <TextBlock></TextBlock>
        <TextBlock>
      <Hyperlink Click="LinkClicked">მე-2 გვერდზე</Hyperlink>
        </TextBlock>
    </StackPanel>
</Page>
```

9. შევასწოროთ Page1.xaml.cs ფაილის კოდი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp5
{
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }
   private void LinkClicked(object sender, RoutedEventArgs e)
        {
            this.NavigationService.Navigate(page2);
        }
    }
}
```

9. დავამატოთ პროექტს ახალი Page ელემენტი Page2.xaml სახელით. შევასწოროთ xaml-კოდი:

```
<Page x:Class="WpfApp5.Page2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page2"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center" FontSize="24">გვერდი
2</TextBlock>
    </StackPanel>
</Page>
```

10. NavExample.xaml ფაილში დავამატოთ Source-ატრიბუტი, რომელიც მიუერთდება კარკასის გაშვებისას Page1.xaml საწყისი გვერდის შიგთავსს.

```
<NavigationWindow x:Class="WpfApp5.NavExample"
    ...
    WindowStartupLocation="CenterScreen"
    Source="Page1.xaml"
          >
</NavigationWindow>
```

11. ავამუშავოთ პროექტი და შევამოწმოთ ღილაკების მუშაობისუნარიანობა.

დასკვნა:

ამგვარად, ჩვენ შევქმენით კარკასი და ორი ცარიელი გვერდი, რომლებიც არაფერს არ აკეთებს. თითოეული გვერდი ავტონომიურია, შეიძლება მათი შევსება ტულბოქსის ელემენტებით.

გვერდებს შორის გადასვლისას საჭიროა ვიცოდეთ ინფორმაციის გადაცემა ერთი გვერდიდან მეორეში. უნდა არსებობდეს საერთო საფოსტო ყუთი, რომელიც არ იქნება დამოკიდებული გვერდებზე.

WPF-ში მონაცემების გადასაცემად გვერდებს შორის იყენებენ ლექსიკონს (წყვილების მასივი: „გასაღები-მნიშვნელობა“)

Application.Current.Properties, ან ინფორმაციის „ჩაკერვას“ უშუალოდ ახალი გვერდის ობიექტში.

(მაგალითის გაგრძელება)

12. Page1-ზე დავამატოთ სახელმინიჭებული ტექსტური ველი შემდეგი სახით:

```
<Page x:Class="WpfApp5.Page1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page1"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center"
                FontSize="24">გვერდი 1</TextBlock>
        <TextBlock></TextBlock>
        <Label>შეიტანეთ თქვენი სახელი: </Label>
        <TextBox Name="nameBox" Width="200"></TextBox>
        <TextBlock></TextBlock>
        <TextBlock>
     <Hyperlink Click="LinkClicked">მე-2 გვერდზე</Hyperlink>
        </TextBlock>
    </StackPanel>
</Page>
```

TextBox-ობიექტს მივანიჭეთ სახელი, რათა შეიძლებოდეს მასზე მიმართვა კოდიდან.

13. შევცვალოთ Page1 გვერდის კოდი შემდეგი სახით;

```
using System;
...
namespace WpfApp5
{
  public partial class Page1 : Page
   {
    public Page1()
        {
            InitializeComponent();
        }
 private void LinkClicked(object sender, RoutedEventArgs e)
        {
```

```
            Page2 page2 = new Page2();
            page2.Message = nameBox.Text + " !!!"; // ინფორმაციის
                                                   //„ჩაკერვა“ ობიექტში
            this.NavigationService.Navigate(page2);
        }
    }
}
```

14. დავამატოთ Page2 გვერდზე სახელმინიჭებული ტექსტური ჭდე და ჰიპერლინკი Page3-ზე გადასასვლელად. აგრეთვე მომამზადეთ გვერდის მოვლენა Loaded და მოვლენა Click ჰიპერლინკისთვის.

```
<Page x:Class="WpfApp5.Page2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page2"
    Loaded="Page_Loaded"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center"
               FontSize="24">გვერდი 2</TextBlock>
        <TextBlock></TextBlock>
        <TextBlock>მოგესალმებით </TextBlock>
        <Label Name="nameLabel"></Label>
        <TextBlock Margin="0,10">   <!--Отступ сверху-->
 <Hyperlink Click="LinkClicked">მე-3 გვერდზე</Hyperlink>
        </TextBlock>
    </StackPanel>
</Page>
```

Label ობიექტს მივანიჭეთ სახელი, რათა კოდიდან შეიძლებოდეს მასზე მიმართვა.

15. დავამატოთ Page2-ის კოდს public თვისება, ტექსტურ ჭდეზე გადაცემული ტექსტის მისანიჭებელი კოდი Loaded-მოვლენაში და შემდეგ გვერდზე გადასვლის კოდი.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
```

```
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp5
{
    public partial class Page2 : Page
    {
        public Page2()
        {
            InitializeComponent();
        }

        string message;
        public string Message
        {
            set { message = value; }
        }
   private void Page_Loaded(object sender, RoutedEventArgs e)
        {
            nameLabel.Content = message;
        }
   private void LinkClicked(object sender, RoutedEventArgs e)
        {
            Page3 page3 = new Page3();
            this.NavigationService.Navigate(page3);
        }
    }
}
```

16. ავამუშავოთ აპლიკაცია. ინფორმაცია გადაეცემა, ოღონდ ღილაკის ამუშავებით.

(!) ნავიგაციის ღილაკით ახალი ინფორმაცია ტექსტური ველიდან არ გადაიცემა. ინფორმაცია აიღება ისტორიის ჟურნალიდან.

17. Page3 გვერდისთვის შეავსეთ xaml-კოდი:

```
<Page x:Class="WpfApp5.Page3"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Page3">
<StackPanel>
    <!--გვერდის კონტექსტის სათაური -->
    <TextBlock TextAlignment="Center"
               FontSize="24">გვერდი 3
        <TextBlock.Margin>0,0,0,10</TextBlock.Margin>
    </TextBlock>
 <!--პირველი ღილაკი-->
    <Button Content="Push Me!" FontSize="22" Width="175"
                        Height="50" Click="Button_Click">
        <Button.Effect>
            <DropShadowEffect />
        </Button.Effect>
    </Button>
 <!--მეორე ღილაკი-->
    <Button FontSize="22" Height="50" Width="175"
                      Margin="0,10" Click="Button_Click">
        "დამკლიკე"
        <Button.Effect>
            <DropShadowEffect />
        </Button.Effect>
        <Button.Foreground>
   <LinearGradientBrush StartPoint="1,0" EndPoint="0,0">
             <GradientStop Color="Red" Offset="0" />
             <GradientStop Color="Orange" Offset=".17" />
             <GradientStop Color="Yellow" Offset=".33" />
             <GradientStop Color="Green" Offset=".5" />
             <GradientStop Color="CornflowerBlue"
                                       Offset=".67" />
            <GradientStop Color="Blue" Offset=".84" />
           <GradientStop Color="BlueViolet" Offset="1" />
           </LinearGradientBrush>
        </Button.Foreground>
        <Button.Background>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
              <GradientStop Color="Red" Offset="0" />
```

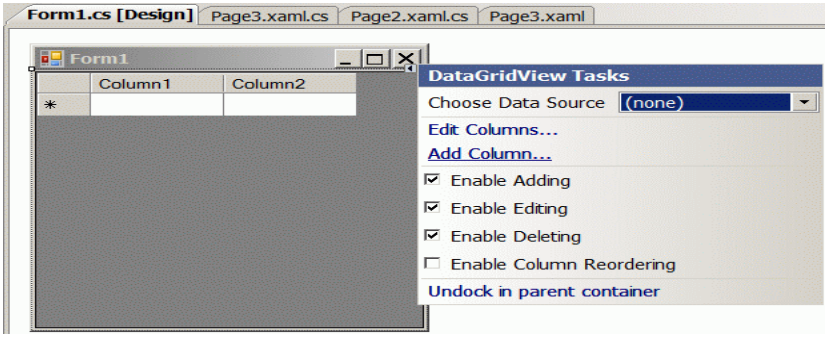
```
                <GradientStop Color="Orange" Offset=".17" />
                <GradientStop Color="Yellow" Offset=".33" />
                 <GradientStop Color="Green" Offset=".5" />
                 <GradientStop Color="CornflowerBlue"
                                          Offset=".67" />
                 <GradientStop Color="Blue" Offset=".84" />
              <GradientStop Color="BlueViolet" Offset="1" />
               </LinearGradientBrush>
          </Button.Background>
      </Button>
  </StackPanel>
</Page>
```

Click - მოვლენის დამმუშავებელი უნდა ჩაიწეროს ხელით, რათა ის შეიქმნას კოდის ნაწილში.

18. სანამ შევავსებთ Page3 გვერდის კოდის ნაწილს, საჭიროა პროექტს დაემატოს ახალი ფორმა. ამით შესაძლებელია WPF და Windows Forms ტექნოლოგიების ერთობლივად მუშაობის დემონ-სტრირება. ღილაკებზე დავდოთ ფორმის ამუშავების კოდი.

19. დავამატოთ WpfApp5-ში ახალი ფორმა Form1.cs

20. Form1 ფორმაზე ტულბოქსიდან დავდოთ DataGridView ელემენტი. დავაყენოთ მისი თვისება Dock=Fill და დავამატოთ ინტელექტუალური დესკრიპტორიდან (SmartTag) ორი სვეტი



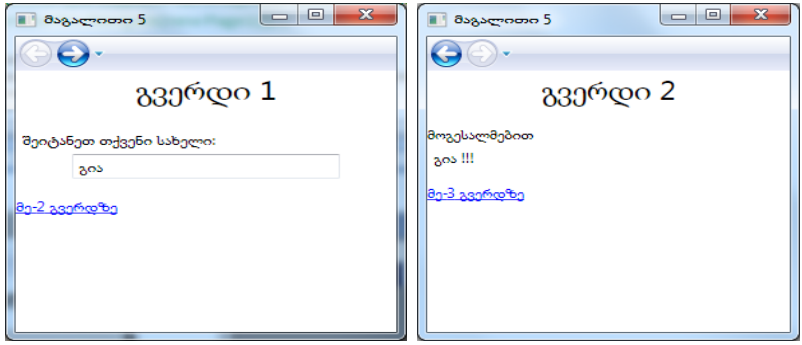
ნახ.1.13

21. ღილაკების საერთო დამმუშავებელი Page3 კოდის ნაწილში შევავსოთ ასე:

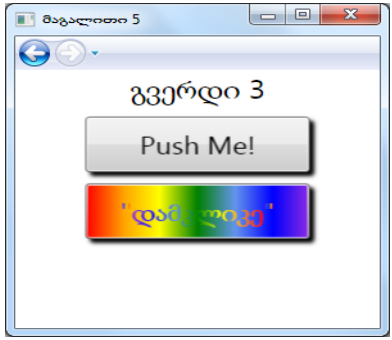
```
//--- ლისტინგი -------
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApp5
{
    public partial class Page3 : Page
    {
        public Page3()
        {
            InitializeComponent();
        }
 private void Button_Click(object sender, RoutedEventArgs e)
        {
            Form1 frm = new Form1();
            frm.ShowInTaskbar = false; // ფორმის ღილაკი არ
                                       //გამოჩნდეს ამოცანების პანელზე
            frm.Show();
        }
    }
}
```

22. ავამუშავოთ პროექტი. დავაკვირდეთ ახალი გვერდის დიზაინს. იხსნება Form1 -იც, რაც ადასტურებს WPF და Windows Form ტექნოლოგიების ერთობლივი მუშაობის შესაძლებლობას.

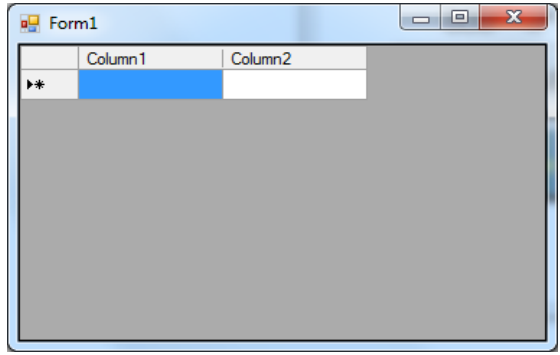
შედეგები (ნახ.1.14–ა,ბ,გ):



ნახ.1.14–ა



ნახ.1.14–ბ



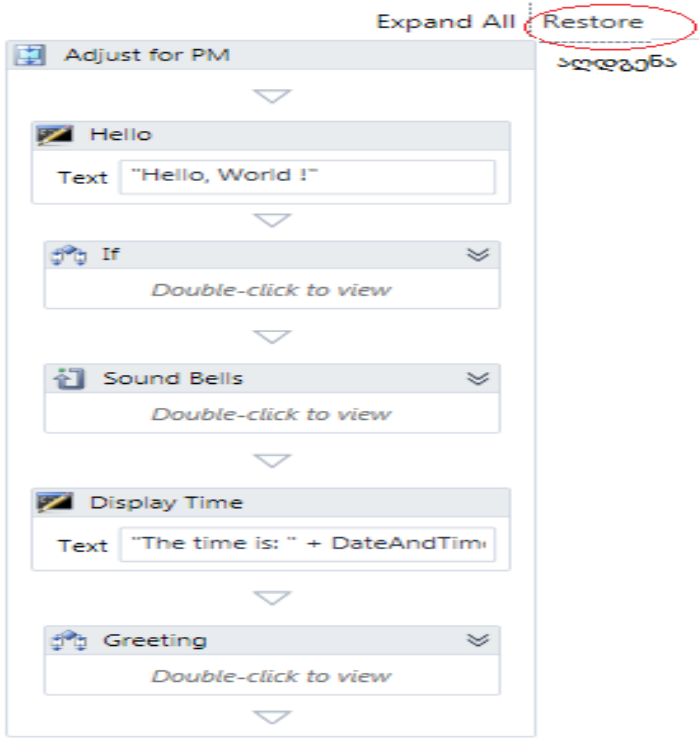
ნახ.1.14–გ

1.2. Workflof Foundation ტექნოლოგია

1.2.1. ბიზნესპროცესების დაპროგრამება WF-ის საფუძველზე

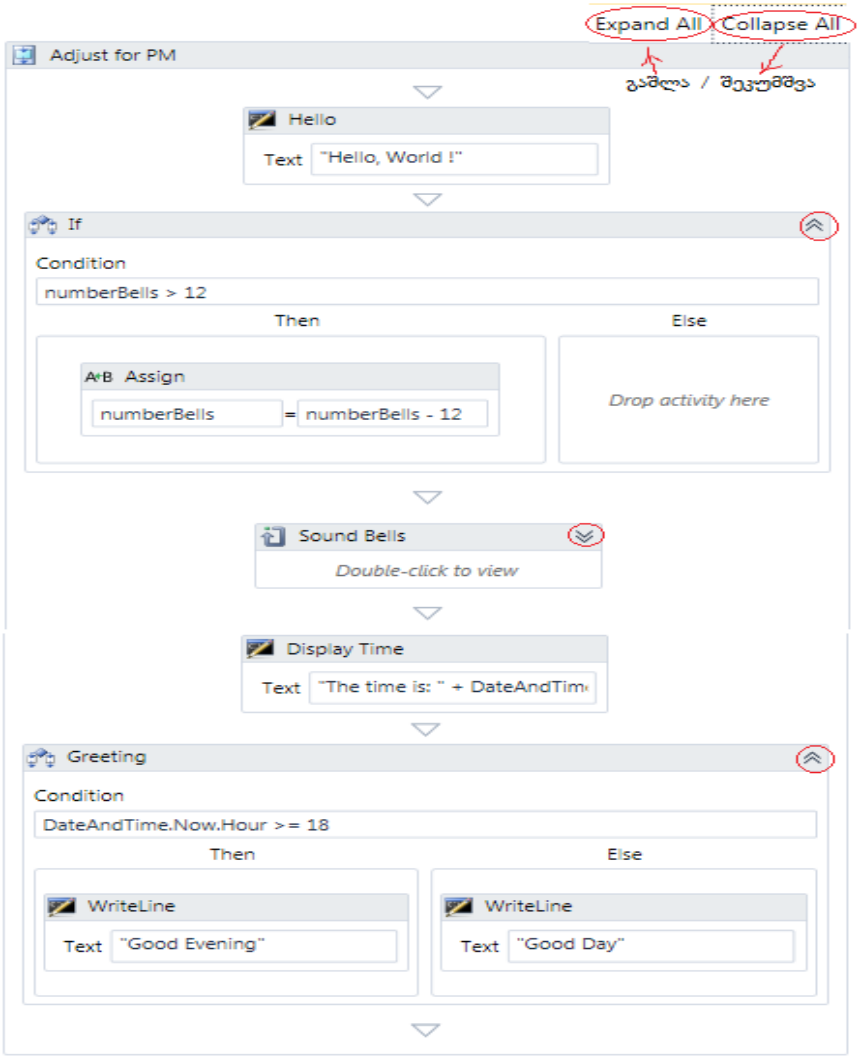
განვიხილოთ Visual Studio.NET პაკეტის WF-ის დიზაინერის სამუშაო გარემოში დიდი პროექტების და სქემების მართვის საშუალებები (Navigating the Designer).

დიდი პროექტების დროს საკმაოდ რთულია დიდი სქემების მართვა. ამისათვის დიზაინერებს ეძლევათ საშუალება ჩაკეცონ სქემის სექციები, მათი მთლიანი სტრუქტურის დათვალიერების და მართვის მიზნით. მაგალითად, ჩვენი პროექტი ჩაკეცვის (Collapse All) შემდეგ ასე გამოიყურება (ნახ.1.15).



ნახ.1.15. შეკუმშული workflow დიაგრამა

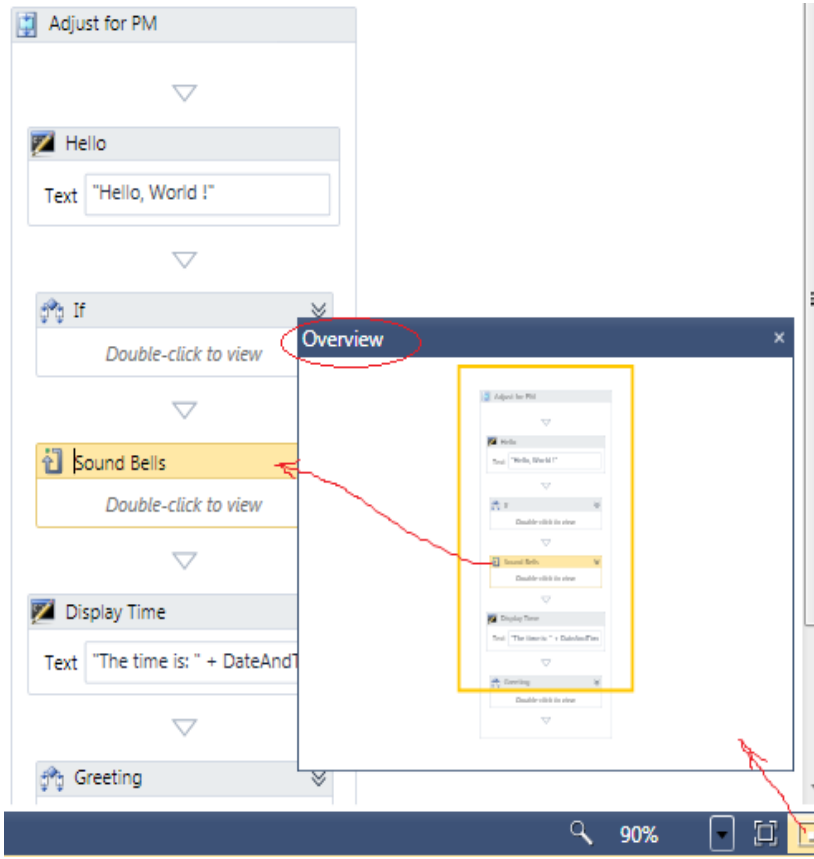
თუ აღდგენის (Restore) ღილაკს გამოვიყენებთ, მაშინ მივიღებთ სურათს შეკუმშვამდე (ნახ.1.16).



ნახ.1.16. ნაწილობრივ შეკუმშული workflow დიაგრამა

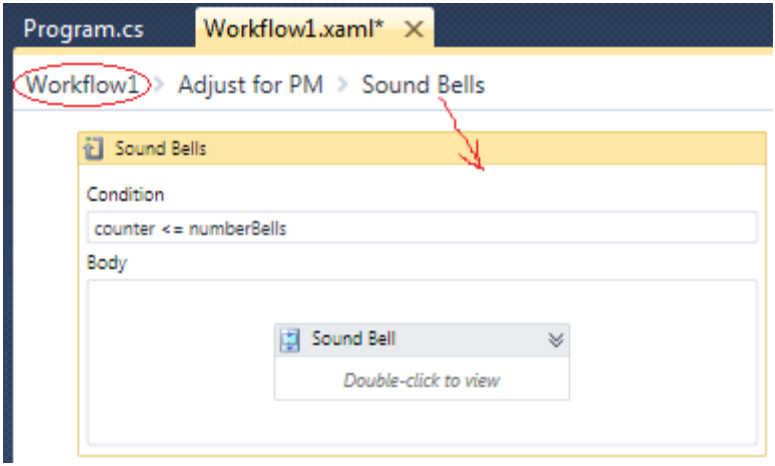
აქ რამდენიმე ბლოკი გაშლილია მთლიანად, რამდენიმე კი ჩაკეცილია (მაგალითად, Bound Bells). Expand All ღილაკით კი გაიშლება ყველა ჩაკეცილი ბლოკი.

დიზაინერის ქვედა მარჯვენა კუთხეში არის Overview–ღილაკი, რომელიც ხსნის პროექტის მონიტორინგის ფანჯარას (ნახ.1.17). მოყვითალო ფერის ზოლი მიუთითებს აქტიურ ხილვად ბლოკზე. აქვეა ეკრანის ზომების ცვლილების (მასშტაბირების) ღილაკებიც.



ნახ.1.17. Overview - ღილაკი

„Sound Bell" ქმედების დაკლიკვით გამოჩნდება მხოლოდ ეს ბლოკი თავისი შვილობილი (ჩადგმული) კომპონენტებით (ნახ.1.18). აქ Window1 გადამრთველის არჩევით დიზაინერში გამოჩნდება ისევ 1.15 ნახაზზე ნაჩვენები სქემა.



ნახ.1.18. გამოყოფილი Sound Bell ბლოკი

- დიზაინერის XAML-კოდი

Solution Explorer-ში დავკლიკოთ Workflow1.xaml და დავათვალიეროთ შესაბამისი კოდის ლისტინგი:

```
<! ---- ლისტინგი -----  >
<Activity mc:Ignorable="" x:Class="WorkflowConsoleHello.Workflow1"
    xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:mv="clr-namespace:Microsoft.VisualBasic;assembly=System"
    xmlns:mva="clr-
namespace:Microsoft.VisualBasic.Activities;assembly=System.Activities"
    xmlns:s="clr-namespace:System;assembly=mscorlib"
```

```
      xmlns:s1="clr-namespace:System;assembly=System"
      xmlns:s2="clr-namespace:System;assembly=System.Xml"
      xmlns:s3="clr-namespace:System;assembly=System.Core"
      xmlns:s4="clr-namespace:System;assembly=System.ServiceModel"
      xmlns:sa="clr-namespace:System.Activities;assembly=System.Activities"
      xmlns:sad="clr-
namespace:System.Activities.Debugger;assembly=System.Activities"

xmlns:sap="http://schemas.microsoft.com/netfx/2009/xaml/activities/presentation"
      xmlns:scg="clr-namespace:System.Collections.Generic;assembly=System"
      xmlns:scg1="clr-
namespace:System.Collections.Generic;assembly=System.ServiceModel"
      xmlns:scg2="clr-namespace:System.Collections.Generic;assembly=System.Core"
      xmlns:scg3="clr-namespace:System.Collections.Generic;assembly=mscorlib"
      xmlns:sd="clr-namespace:System.Data;assembly=System.Data"
      xmlns:sl="clr-namespace:System.Linq;assembly=System.Core"
      xmlns:st="clr-namespace:System.Text;assembly=mscorlib"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <sap:WorkflowViewStateService.ViewState>
    <scg3:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="ShouldCollapseAll">True</x:Boolean>
      <x:Boolean x:Key="ShouldExpandAll">False</x:Boolean>
    </scg3:Dictionary>
  </sap:WorkflowViewStateService.ViewState>

  <Sequence DisplayName="Sequence"

sad:XamlDebuggerXmlReader.FileName="C:\WorkflowConsoleHello\WorkflowCo
nsoleHello\Workflow1.xaml"
    sap:VirtualizedContainerService.HintSize="233,564">
    <Sequence.Variables>
      <Variable x:TypeArguments="x:Int32" Default="1" Name="counter" />
```

```
    <Variable x:TypeArguments="x:Int32" Default="[DateAndTime.Now.Hour]"
Name="numberBells" />
   </Sequence.Variables>
  <sap:WorkflowViewStateService.ViewState>
    <scg3:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="IsExpanded">True</x:Boolean>
    </scg3:Dictionary>
  </sap:WorkflowViewStateService.ViewState>

 <WriteLine DisplayName="Hello"
sap:VirtualizedContainerService.HintSize="211,62"

Text="Hello,  World !" />
   <If Condition="[numberBells &gt; 12]"
sap:VirtualizedContainerService.HintSize="211,52">
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">True</x:Boolean>
        <x:Boolean x:Key="IsPinned">False</x:Boolean>
      </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <If.Then>

      <Assign sap:VirtualizedContainerService.HintSize="291,100">
        <Assign.To>
          <OutArgument
x:TypeArguments="x:Int32">[numberBells]</OutArgument>
        </Assign.To>
        <Assign.Value>
          <InArgument x:TypeArguments="x:Int32">[numberBells -
12]</InArgument>
        </Assign.Value>
      </Assign>
```

```
    </If.Then>
   </If>

   <While DisplayName="Sound Bells"
sap:VirtualizedContainerService.HintSize="211,52">
    <sap:WorkflowViewStateService.ViewState>
     <scg3:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="IsExpanded">False</x:Boolean>
      <x:Boolean x:Key="IsPinned">False</x:Boolean>
     </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <While.Condition>[counter &lt;= numberBells]</While.Condition>
    <Sequence DisplayName="Sound Bell"
sap:VirtualizedContainerService.HintSize="438,100">
     <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
       <x:Boolean x:Key="IsExpanded">False</x:Boolean>
       <x:Boolean x:Key="IsPinned">False</x:Boolean>
      </scg3:Dictionary>
     </sap:WorkflowViewStateService.ViewState>

   <WriteLine sap:VirtualizedContainerService.HintSize="242,62"
Text="[counter.ToString()]" />
     <Assign sap:VirtualizedContainerService.HintSize="242,58">
      <Assign.To>
       <OutArgument x:TypeArguments="x:Int32">[counter]</OutArgument>
      </Assign.To>
      <Assign.Value>
       <InArgument x:TypeArguments="x:Int32">[counter + 1]</InArgument>
      </Assign.Value>
     </Assign>
     <Delay Duration="[TimeSpan.FromSeconds(1)]"
sap:VirtualizedContainerService.HintSize="242,22" />
```

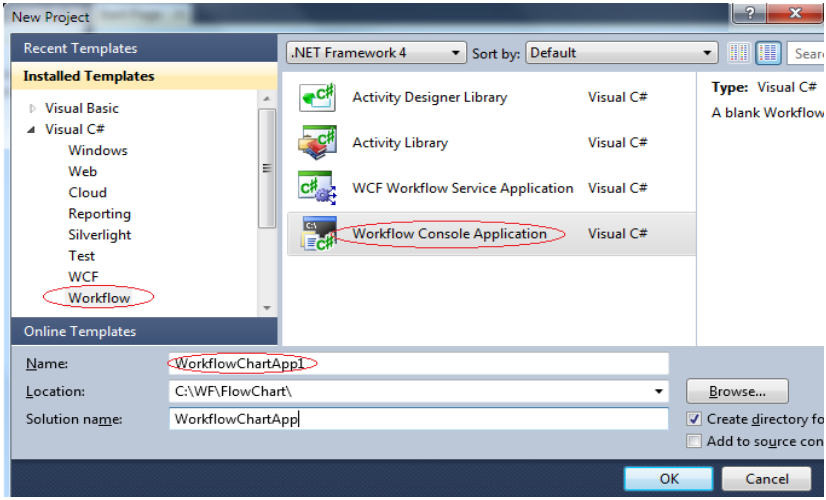
```
    </Sequence>
   </While>

 <WriteLine DisplayName="Display Time"
sap:VirtualizedContainerService.HintSize="211,62"
        Text="[&quot;The time is: &quot; + DateAndTime.Now.ToString()]" />

  <If Condition="[DateAndTime.Now.Hour &gt;= 18]" DisplayName="Greeting"
       sap:VirtualizedContainerService.HintSize="211,52">
    <If.Then>
      <WriteLine sap:VirtualizedContainerService.HintSize="219,100" Text="Good
Evening" />
    </If.Then>
    <If.Else>
      <WriteLine sap:VirtualizedContainerService.HintSize="220,100" Text="Good
Day" />
    </If.Else>
   </If>
  </Sequence>
</Activity>
```

1.2.2. ბიზნესპროცესის დიაგრამა (Flowchart Workflow)

ამჯერად უნდა ავაგოთ ბიზნესპროცესი, რომელიც გამოიყენებს აქტიუროზათა დიაგრამას. როგორც სათაურიდან ჩანს, ქმედებათა დიაგრამა მუშაობს როგორც ბლოკ–სქემა, ქმედებათა სახეები დაკავშირებულია ერთმანეთთან გადაწყვეტილების ხეებით (decision trees). ქმედებათა მიმდევრობითობის გამოყენებით, შვილი–პროცესები სრულდება მიმდევრობით ზემოდან–ქვევით (top-down). ამისდა მიუხედავად, შვილი–ქმედებები შეიძლება შესრულდეს ნებისმიერი მიმდევრობით, გადაწყვეტილების მიმღები პირის მიერ.

- **ბიზნესპროცესის დიაგრამის შექმნა**

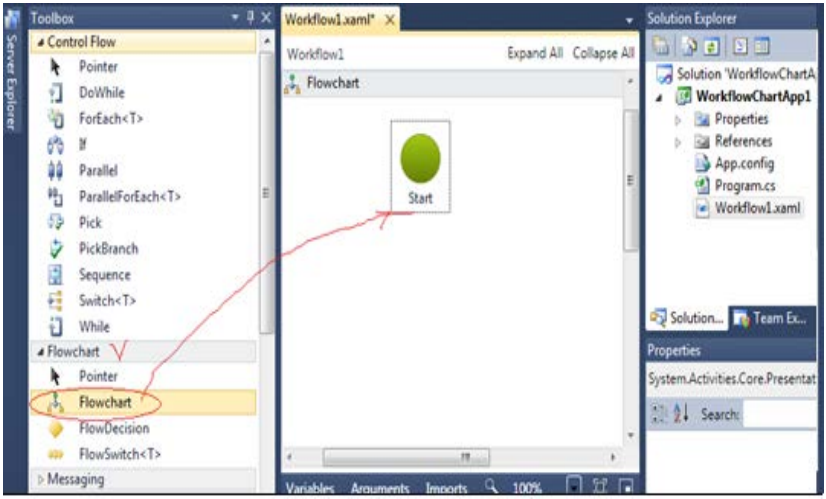
შევქმნათ ახალი პროექტი (solution), ავირჩიოთ workflow და Console Application (ნახ.1.19).



ნახ.1.19

ინსტრუმენტების პანელიდან Flowchart გადმოვიტანოთ ფორმაზე Flowchart-ქმედება. საწყისი მუშა პროცესის დიაგრამა სასტარტო მწვანე წრით მოცემულია 1.20 ნახაზზე. ცარიელი სივრცე მის ქვეშ გამოიყენება ასაგები ბიზნესპროცესის ქმედებების დასამატებლად.

ძირითადი განსხვავება Flowchart ქმედებასა და Sequence ქმედებას შორის ისაა, თუ როგორაა დაკავშირებული შვილი–აქტიურობები. როგორც ცნობილია, ქმედებების დამატებისას მიმდევრობითობაში ისინი სრულდება დაღმავალი (top-down) რიგითობით. აქ შესაძლებელია მიმდევრობის კონტროლი (მართვა), ქმედებათა გადაადგილებით, ოღონდაც ისინი ყოველთვის გასწორებულია ვერტიკალში და განაწილებულია თანაბრად. ისრები ქმედებებს შორის კი აგებულია ავტომატურად.

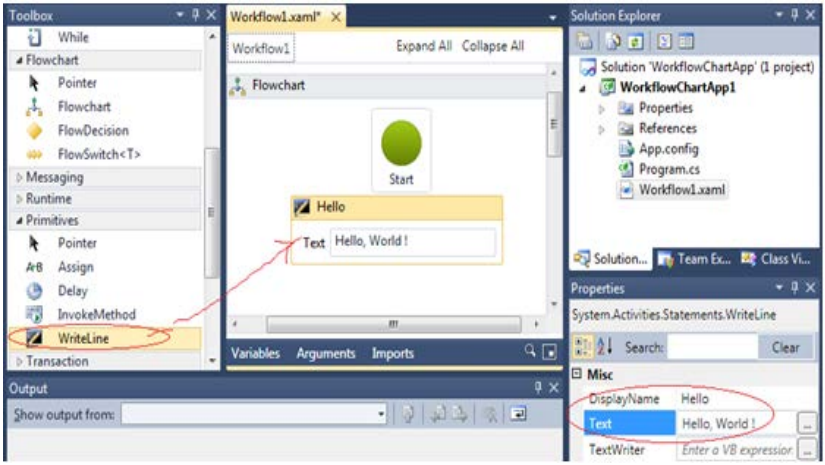


ნახ.1.20

Flowchart აქტიურობით შესაძლებელია ქმედებათა განთავსება პალიტრის ნებისმიერ ადგილას. მნიშვნელოვანია ისიც, რომ აქ ისრები ხელით უნდა გაკეთდეს და შეერთება დასაშვებია უკან, წინა ქმედებასთანაც !

ჩვენ აპლიკაციაში დისპლეიზე უნდა გამოვიტანოთ მისალმებები, შესაბამისად დღე–ღმის დროისა. დავიწყოთ სტანდარტული მისალმების ასახვით „Hello, World".

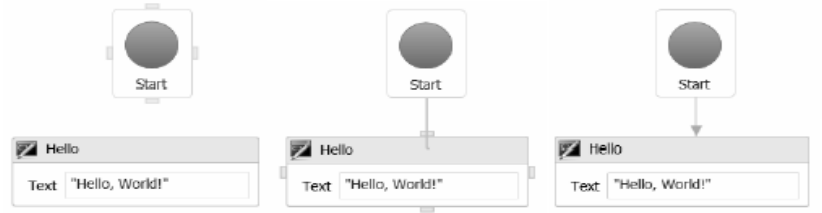
ამისათვის გადმოვიტანოთ WriteLine ქმედება ინსტრუმენტების პანელიდან მწვანე წრის ქვეშ. დავაყენოთ DisplayName-ში "Hello" და Text-ში "Hello, World !" (ნახ.1.21).



ნახ.1.21

- **მიერთების განსაზღვრა**

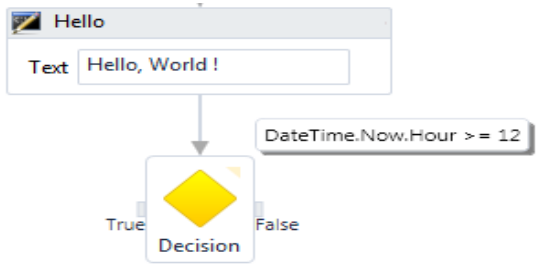
მაუსით მოვნიშნოთ სასტარტო მწვანე წრე, გავატაროთ კავშირი და ავტომატურად შეიქმნება ისარი ორ ქმედებას შორის (ნახ.1.22).



ნახ.1.22. მიერთების საწყისი და ბოლო წერტილები

- **გადაწყვეტილების ნაკადი (FlowDecision)**

გადმოვიტანოთ სქემაზე FlowDecision ქმედება "Hello" ქმედების შემდეგ. FlowDecision ქმედება გამოიყურება ყვითელი ალმასის სახით, როგორც განშტოების სიმბოლო ნორმალურ ბლოკ–სქემაში, თვისებათა ფანჯარაში შევიტანოთ მდგომარეობა (Condition) DateTime.Now.Hour> = 12. მაუსის კურსორის მიტანით FlowDecision ქმედებაზე გამოჩნდება ასეთი სურათი (ნახ.1.23).



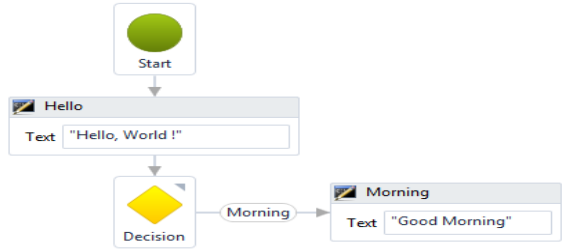
ნახ.1.23

ქმედებას აქვს მარცხნიდან მიერთების წერტილი true შტოსთვის, ხოლო მარჯვნიდან – false შტოსთვის. ზედა მარჯვენა კუთხეში პატარა სამკუთხედით ჩაირთვება პირობის გამოსახულების მუდმივად გამოსაჩენი თვისება (უმაუსოდ).

შეიძლება ტექსტის შეცვლა true/false შტოებისთვის. მაგალითად, FalseLabel–თვის შევიტანოთ Morning, და TrueLabel-თვის Afternoon. მიიღება სურათი:

ნახ.1.24

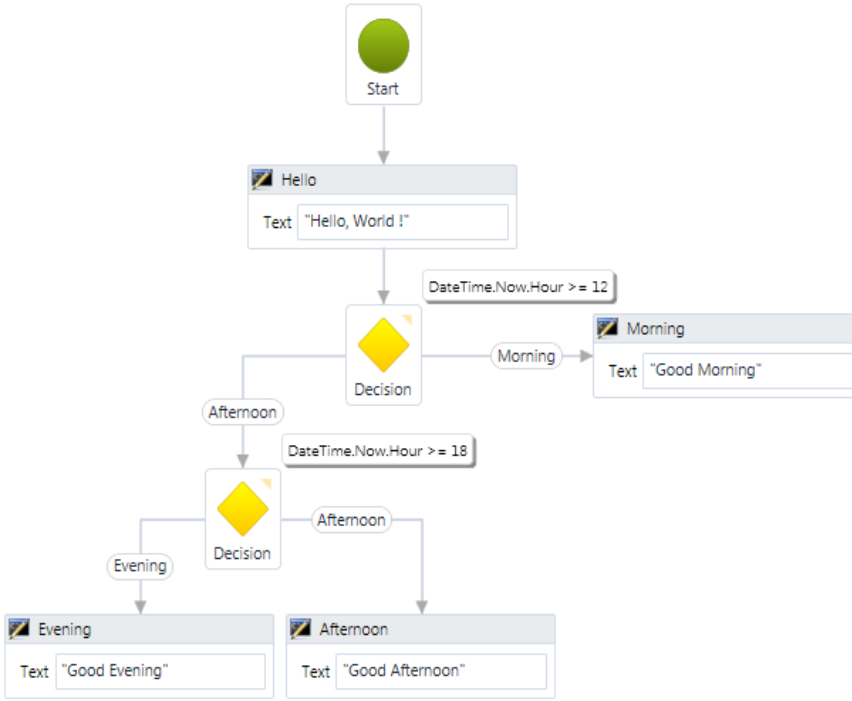
შევაერთოთ FlowDecision ქმედება მარჯვნიდან WriteLine ქმედებას, რომელშიც ჩაწერილი გვაქვს Morning/”Good Morning” (ნახ.1.25).



ნახ.1.25

FlowDecision ქმედებას არ აქვს DisplayName თვისება, მაგრამ შეუძლია პირობის ჩვენება და true/false შტოების შემოწმება და კორექტირება. ამგვარად ამ აქტიურობის მიზანი და ფუნქციონალობა ცალსახად ცხადია.

დავამატოთ FlowDecision ქმედების მარცხენა true შტოს ახალი FlowDecision ქმედება, პირობის გამოსახულებით DateTime.Now.Hour >= 18. მისი მარცხენა TrueLabel შტო იყოს Evening, ხოლო მარჯვენა – ისევ Afternoon. ორივეს მივუერთოთ ახალი WriteLine ქმედებები: Evening და Afternoon. 1.26 ნახაზზე მოცემულია ეს სურათი.

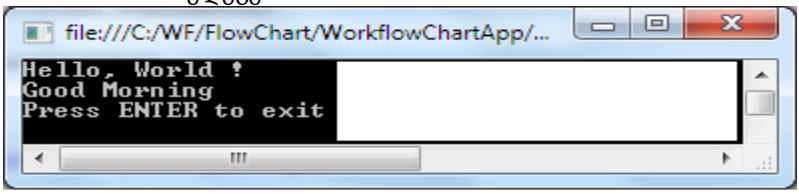


ნახ.1.26

სანამ ავამუშავებთ მიღებულ აპლიკაციას, გავხსნათ Program.cs ფაილი. ჩავამატოთ აქ შემდეგი კოდი WorkflowInvoker კლასის გამოძახების შემდეგ. ის დაგვანახვებს შედეგებს პროგრამიდან გამოსვლამდე.

```
Console.WriteLine("Press ENTER to exit");
Console.ReadLine();
```

აპლიკაციის ამუშავება: F5
შედეგები:

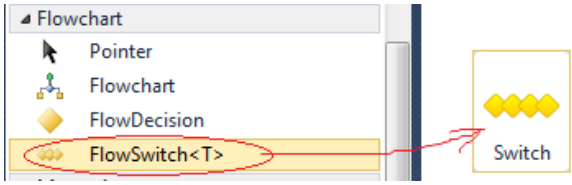


ნახ.1.27

შედეგი დამოკიდებულია იმაზე თუ დღის რომელ მონაკვეთში ავამუშავებთ აპლიკაციას.

- **პროცესის გადამრთველი (Flow Switch)**

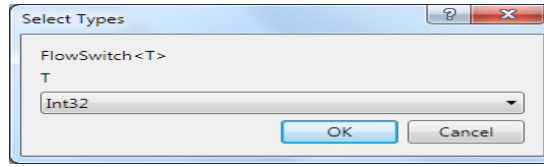
FlowSwitch ქმედება ფუნქციონირებს როგორც FlowDecision იმ გამონაკლისით, რომ არაა შეზღუდვა მხოლდ true/false ორი განშტოებით. შესაძლებელია შტოების ნებისმიერი რაოდენობის განსაზღვრა ისე, როგორც ეს იყო C# ენაში. 1.28 ნახაზზე ნაჩვენებია FlowSwitch აქტიუროობის პიქტოგრამა ინსტრუმენტების პანელზე.



ნახ.1.28. პიქტოგრამა

- FlowSwitch ქმედების დამატება

გადმოვიტანოთ ინსტრუმენტების პანელიდან FlowSwitch აქტიურობა ჩვენი ბოლო პროექტის სქემის ქვეშ. იგი არის <T> კლასის შაბლონი და უნდა მიეთითოს ტიპი. ჩვენ შემთხვევაში ესაა Int32, რომელიც ავტომატურადაა განსაზღვრული. ვირჩევთ OK-ს (ნახ.1.29).



ნახ.1.29. ტიპის განსაზღვრა

შევაერთოთ სქემის Morning, Evening და Afternoon ქმედებები FlowSwitch ქმედებას, რომელსაც აქვს ერთი თვისება გამოსახულებისთვის და იგი განსაზღვრავს გადართვის შტოს ცვლადის მნიშვნელობას (ნახ.1.30). ჩვენ პროექტში განვახორციელებთ გადამრთველის რეალიზებას მისალმების მიზნით წელიწადის დროის მიხედვით.

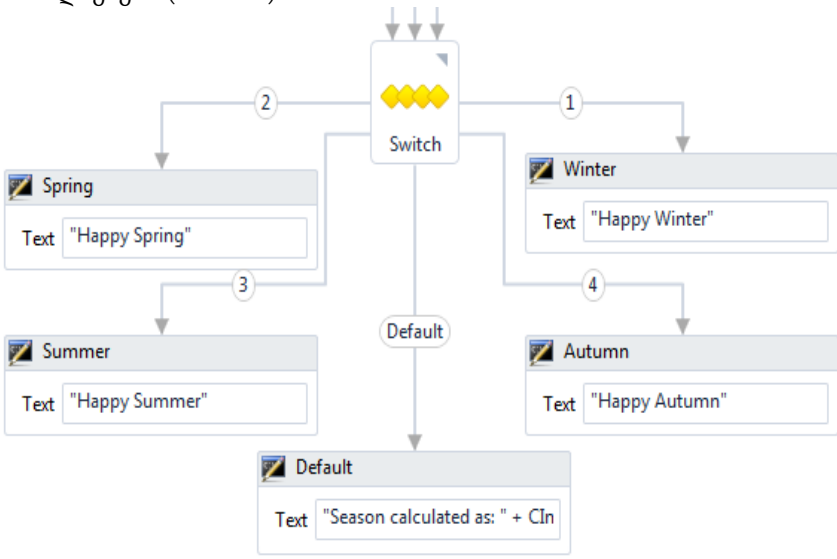
ნახ.1.30

შენიშვნა: გამოსახულების ჩაწერის ფორმა არაა დამოკიდებული გამოყენებული ენის სინტაკსზე. აქ მიღებულია, რომ ეს ფორმა იყოს Visual Basic ენის სინტაკსის მსგავსი.

ჩვენი გამოსახულება განსაზღვრავს წელიწადის დროის (ზამთარი, გაზაფხული, ზაფხული, შემოდგომა) სეზონს. მაგალითად, თუ მიმდინარე (ახლანდელი) თვე არის დეკემბერი, იანვარი ან თებერვალი, მაშინ გამოსახულება გვაძლევს 0–ს. ანალოგიურად მარტი, აპრილი და მაისი მოგვცემს 1–ს და ა.შ. გადამრთველის ოთხ შტოსთვის უნდა განვსაზღვროთ 0,1,2 და 3, ანუ სეზონები.

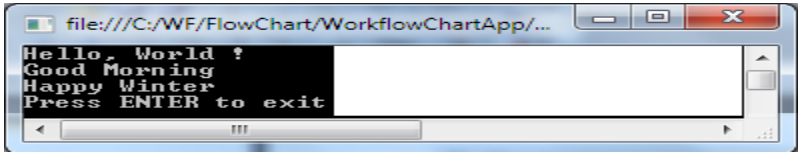
- **FlowStep ქმედების დამატება**

FlowSwitch აქტიურობის ყოველი შტო იძახებს FlowStep ქმედებას. ის Toolbox-ში არაა, ამიტომ მისი ცხადი სახით შექმნა არ შეიძლება. სქემას გადამრთველის გამოსასვლელზე უნდა დაემატოს რამდენიმე ქმედება, მაგალითად, ხუთი WriteLine და მათი შეერთების დროს მოხდება შინაგანად FlowStep–ის მნიშვნელობის განსაზღვრა. WriteLine ქმედებებისთვის ჩავწეროთ DisplayName-ში: Winter, Spring, Summer, Autumn და Default, აგრეთვე შესაბამისი მისალმებები (ნახ.1.31).



ნახ.1.31

პროგრამის ამუშავებით(F5) მივიღებთ შედეგს (ნახ.1.32).



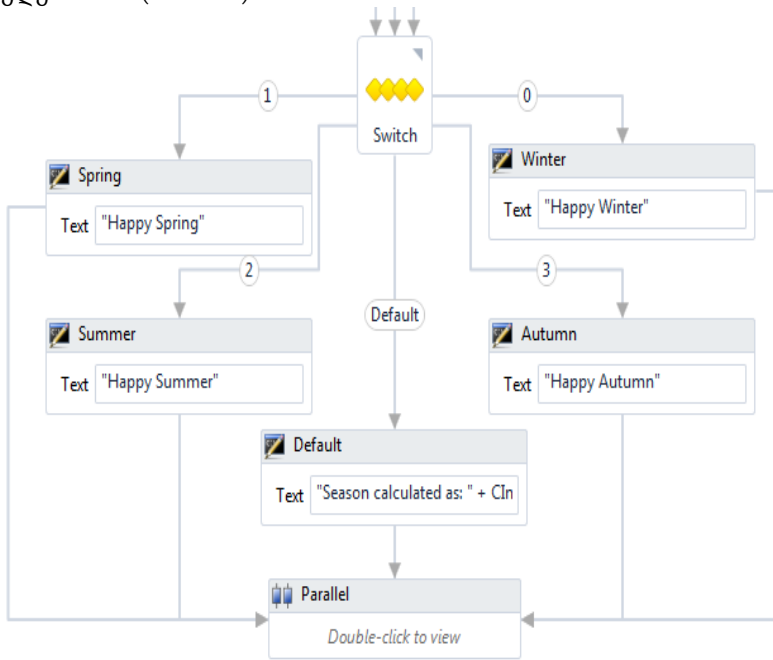
ნახ.1.32. FlowSwitch აპლიკაციის შედეგები

- **პარალელური პროცესები (Parallel)**

ჩვენი პროექტის ფარგლებში განვიხილოთ პარალელური პროცესების საკითხი, Parallel ქმედების მაგალითზე, რომელიც საშუალებას იძლევა განვსაზღვროთ Sequence ქმედებათა რაოდენობა, რომლებიც სრულდება პარალელურად (ერთ-დროულად). ამ პროექტში ყოველი შტო გამოიტანს ინფორმაციის თავის ნაწილს. გამოტანის რიგითობას არ აქვს არსებითი მნიშვნელობა, მთვარია, რომ ისინი მოთავსებულია ერთ Parallel აქტიურობაში და ყველა სრულდება ერთდროულად.

- **Parallel ქმედების დამატება**

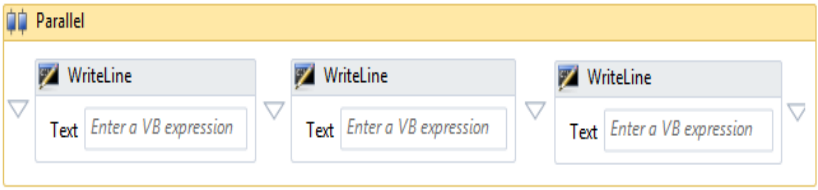
გადმოვიტანოთ Parallel აქტიურობა ჩვენი მუშა პროცესის ბოლოში. ხუთივე WriteLine-დან გავავლოთ კავშირი Parallel ქმედებასთან (ნახ.1.33).



ნახ.1.33

- **შტოების დამატება**

ორჯერ დავკლიკოთ Parallel ქმედება და მის ზედაპირზე გადმოვიტანოთ სამი WriteLine ქმედება (ნახ.1.34).



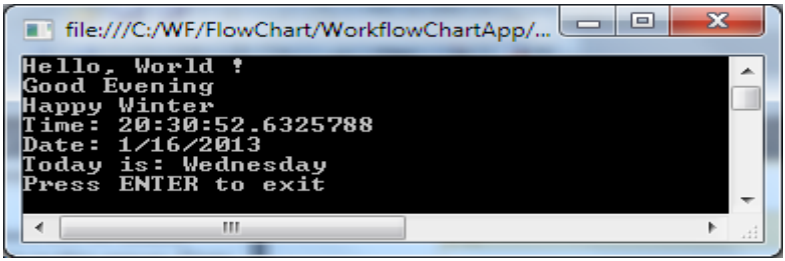
ნახ.1.34

ერთმა უნდა გამოიტანოს მიმდინარე თარიღი, მეორემ – დრო და მესამემ კვირის დღე. მათ Text-ში გამოსახულებებისათვის (expression) შევიტანოთ შემდეგი:

```
"Date: " + DateTime.Now.Date.ToShortDateString()
"Time: " + DateTime.Now.TimeOfDay.ToString()
"Today is: " + DateTime.Now.ToString("dddd")
```

შენიშვნა: Parallel ქმედება ნებას იძლევა მხოლოდ ერთი ქმედება განხორციელდეს ერთ შტოში, რაც ჩვენ მაგალითზე კარგად მუშაობს. თუ გვინდა მულტი–ქმედებების გამოყენება თითოეულ შტოში, მაშინ უნდა ვიხმაროთ Sequence ქმედება. აქ შეიძლება მის შიგნით დავამატოთ ქმედებათა გარკვეული რაოდენობა.

პროგრამის ამუშავების შემდეგ მიიღება ასეთი შედეგები:



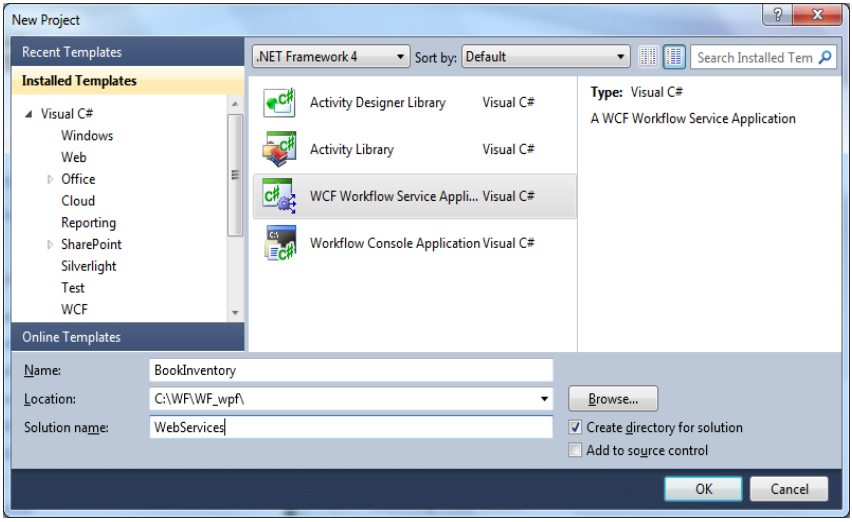
ნახ.1.35

1.3. Web სერვისების დამუშავება WCF-ის ბაზაზე

ბიზნესპროცესები შეიძლება განთავსდეს ვებ-სერვისში, რომელიც უზრუნველყოფს იდეალურ საშუალებას სამუშაო პროცესის გადაწყვეტილების მისაწოდებლად არა-მუშა პროცესის კლიენტებისთვის, როგორიცაა ვებ-აპლიკაციები. ვებ-სერვისი იღებს მოთხოვნას, ასრულებს მის სათანადო გადამუშავებას და აბრუნებს პასუხს. ეს, ბუნებრივია, სრულდება Receive და Send ქმედებებით. ეს აქტიურობები ინტეგრირებულია Windows Communication Foundation თან, ჩვენ შეგვიძლია ადვილად შევქმნათ WCF სერვისები [3,20].

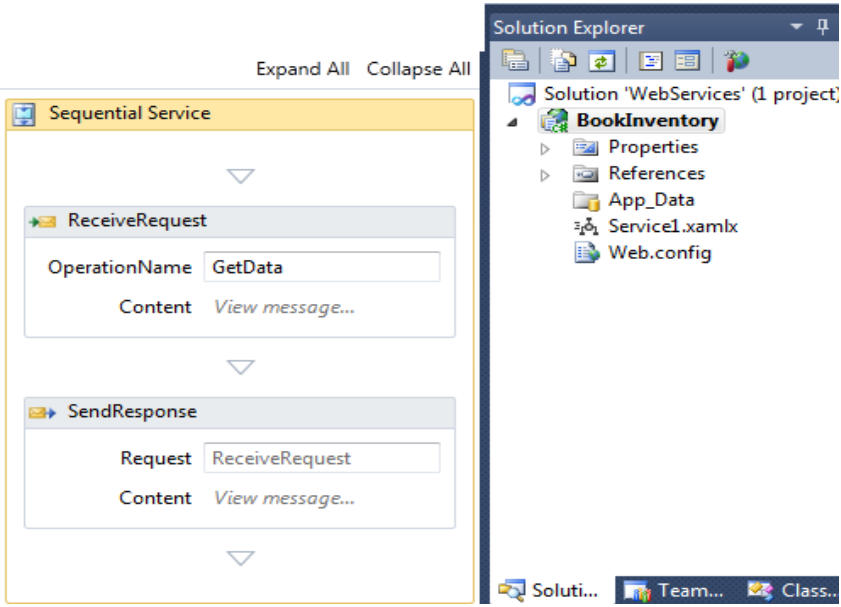
1.3.1. ბიზნესპროცესის სერვისის შექმნა

Visual Studio-ში შევქმნათ ახალი პროექტი WCF Workflow Service Application template გამოყენებთ. შევიტანოთ პროექტის სახელი BookInventory და solution-ის სახელი WebServices.



ნახ.1.36. *Creating a WCF Workflow Service Application*

შეიქმნება საინიციალიზაციო workflow Sequence ბლოკი, რომელიც შეიცავს Receive და SendReply ქმედებებს, როგორც 1.37 ნახაზზეა ნაჩვენები.



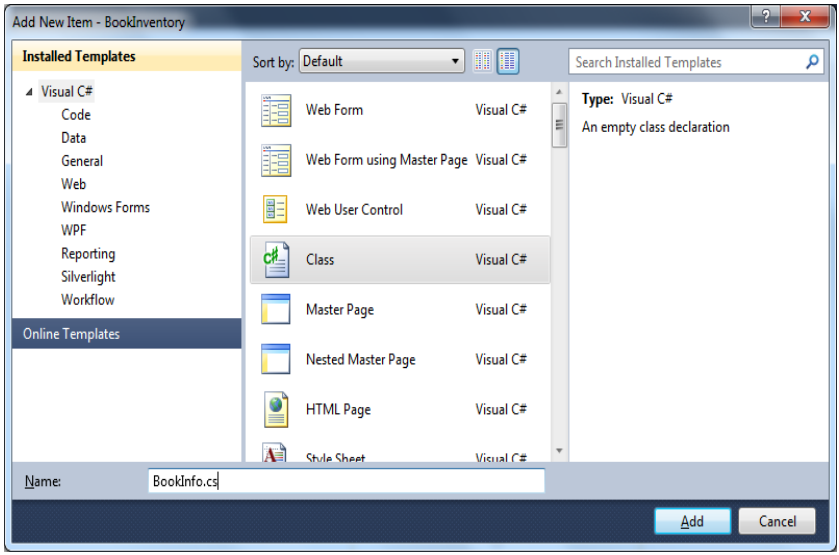
ნახ.1.37. The initial workflow sequence

თავიდან საჭიროა ამ ქმედებების კონფიგურაცია სერვისის კონტრაქტის განსაზღვრის მიზნით, რომელსაც ისინი დააკმაყოფილებს. შემდეგ დავამატოთ დამუშავების მუშა პროცესი, რომელიც განხორციელდება Receive და SendReply ქმედებებს შორის.

შაბლონი შექმნის საწყის მუშა პროცესს ფაილში სახელით Service1.xamlx. შევცვალოთ Solution Explorer-ში ეს სახელი BookInventory.xamlx -ით. ***„სერვისი, რომლის შექმნაც გვინდა, ძებნის მითითებულ წიგნს და აბრუნებს უკან ყოველი ასლის მდგომარეობას, რომელიც ბიბლიოთეკას ეკუთვნის”.***

1.3.2. სერვისის კონტრაქტის განსაზღვრა

Solution Explorer-ში, მარჯვენა ღილაკით BookInventory პროექტზე ავირჩიოთ: Add->Class სახელით BookInfo.cs, რომლის ტექსტი მოცემულია 3.1 ლისტინგში.



ნახ.1.38

```
// ----- ლისტინგი 3.1 -----
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;

namespace BookInventory
{
    /*************************************************/
    // Define the service contract, IBookInventory
    // which consists of a single method, LookupBook()
    /*************************************************/
    [ServiceContract]
```

```
public interface IBookInventory
{
    [OperationContract]
    BookInfoList LookupBook(BookSearch request);
}
/*****************************************************/
// Define the request message, BookSearch
/*****************************************************/
[MessageContract(IsWrapped = false)]
public class BookSearch
{
    private String _ISBN;
    private String _Title;
    private String _Author;

    public BookSearch()
    {
    }

public BookSearch(String title, String author, String isbn)
    {
        _Title = title;
        _Author = author;
        _ISBN = isbn;
    }

    #region Public Properties
    [MessageBodyMember]
    public String Title
    {
        get { return _Title; }
        set { _Title = value; }
    }

    [MessageBodyMember]
    public String Author
    {
        get { return _Author; }
        set { _Author = value; }
    }

    [MessageBodyMember]
    public String ISBN
```

```
    {
        get { return _ISBN; }
        set { _ISBN = value; }
    }
    #endregion Public Properties
}
/*****************************************************/
// Define the BookInfo class
/*****************************************************/
[MessageContract(IsWrapped = false)]
public class BookInfo
{
    private Guid _InventoryID;
    private String _ISBN;
    private String _Title;
    private String _Author;
    private String _Status;

    public BookInfo()
    {
    }

  public BookInfo(String title, String author, String isbn,
        String status)
    {
        _Title = title;
        _Author = author;
        _ISBN = isbn;
        _Status = status;
        _InventoryID = Guid.NewGuid();
    }
    #region Public Properties
    [MessageBodyMember]
    public Guid InventoryID
    {
        get { return _InventoryID; }
        set { _InventoryID = value; }
    }
    [MessageBodyMember]
    public String Title
    {
        get { return _Title; }
        set { _Title = value; }
```

```
        }
        [MessageBodyMember]
        public String Author
        {
            get { return _Author; }
            set { _Author = value; }
        }
        [MessageBodyMember]
        public String ISBN
        {
            get { return _ISBN; }
            set { _ISBN = value; }
        }

        [MessageBodyMember]
        public String status
        {
            get { return _Status; }
            set { _Status = value; }
        }
        #endregion Public Properties
    }
    /*****************************************************/
    // Define the response message, BookInfoList, which
    // is a list of BookInfo classes
    /*****************************************************/
    [MessageContract(IsWrapped = false)]
    public class BookInfoList
    {
        private List<BookInfo> _BookList;

        public BookInfoList()
        {
            _BookList = new List<BookInfo>();
        }

        [MessageBodyMember]
        public List<BookInfo> BookList
        {
            get { return _BookList; }
        }
    }
}
```

სერვისის კონტრაქტი IbookInventory შეიცავს ერთადერთ მეთოდს LookupBook(). იგი მონაცემებს გადასცემს BookSearch კლასს, რომელსაც აქვს სახადასხვა თვისებები, საჭირო წიგნის მოსაძებნად, მაგალითად ავტორს და დასახელებას. ის აბრუნებს უკან BookInfoList კლასს, რომელიც შეიცავს BookInfo კლასების კოლექციას. შესაძლებელია მე-8* თავის ნახვა, სადაც ახსნილია ServiceContract, MessageContract და MessageBodyMember ატრიბუტების გამოყენება.

F6 ამოქმედებით აიგება გადაწყვეტა (solution).

= = = = =

*) MessageContract ატრიბუტი მიუთითებს, რომ ეს კლასი ჩართულ იქნება SOAP ბარათში. SOAP–ის გამოყენების დროს შეტყობინებები გადაიცემა XML–ის მსგავსი ფორმატირებადი ენით. ეს უზრუნველყოფს კლიენტებსა და სერვერს შორის მაღალი ხარისხის ურთიერთქმედების პლატფორმას. SOAP არის სტანდარტული პროტოკოლი, რომლის მხარდაჭერაც აქვს WCF –ს.

არსებობს აგრეთვე MessageBodyMember ატრიბუტი მის ყოველ პუბლიკ–თვისებაზე. ეს აუცილებელია WCF–ფენისთვის რათა სწორად დაფორმატდეს SOAP შეტყობინება.

WCF–ის ბოლო წერტილის განსაზღვრისთვის არსებობს ინფორმაციის სამი პორცია, რომლებიც მითითებულ უნდა იქნას: მიერთება (binding), მისამართი და კონტრაქტი.

მიერთება მიუთითებს იმ პროტოკოლს, რომელიც გამოიყენება (მაგ.,HTTP, TCP ან სხვ.).

მისამართი მიუთითებს თუ სად უნდა ვიპოვოთ ბოლო წერტილი, და მისამართის ტიპს, რომლის გამოყენება დამოკიდებულია მიერთებაზე. მაგალითად, HTTP–მიერთებისას უნდა მიეთითოს URL, ხოლო TCP–თვის მისამართი იქნება სერვერის სახელი ან IP–მისამართი.

კონტრაქტი განისაზღვრება ServiceContract–ით, რომელიც არის ინტერფეისი. იგი განსაზღვრავს მეთოდებს, რომლებიც მიწვდომადია ბოლო წერტილში.

ამგვარად, ჩვენ განვსაზღვრეთ შეტყობინებები, რომლებიც გადაიცემა სერვის–მეთოდების მიერ პარამეტრების სახით.

= = = = = =

1.3.3. Receive და SendReply კონფიგურირება

გავხსნათ BookInventory.xamlx ფაილი და ავირჩიოთ "ReceiveRequest" ქმედება (ნახ.1.39-ა).

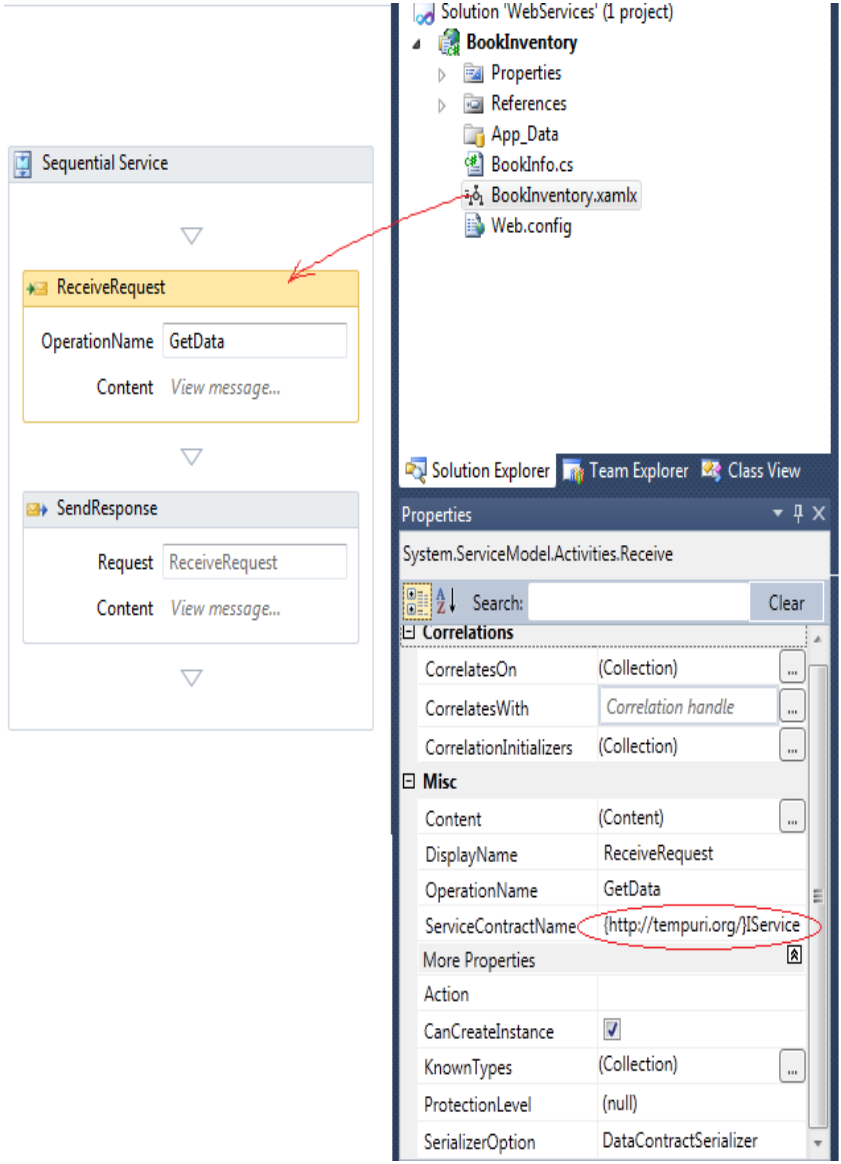
თვისებათა ფანჯარაში ServiceContract თვისებას აქვს default-მნიშვნელობა {http://tempuri.org/}IService. შევცვალოთ Iservice სტრიქონი **IbookInventory**-ით. შევიტანოთ OperationName როგორც **LookupBook.**

ეკრანზე WorkflowService-დიზაინერში ქვედა მარცხენა კუთხეში დავკლიკოთ Variables ღილაკი. გამოჩნდება შაბლონი ორი ცვლადის შესაქმნელად (ნახ.1.39-ბ).

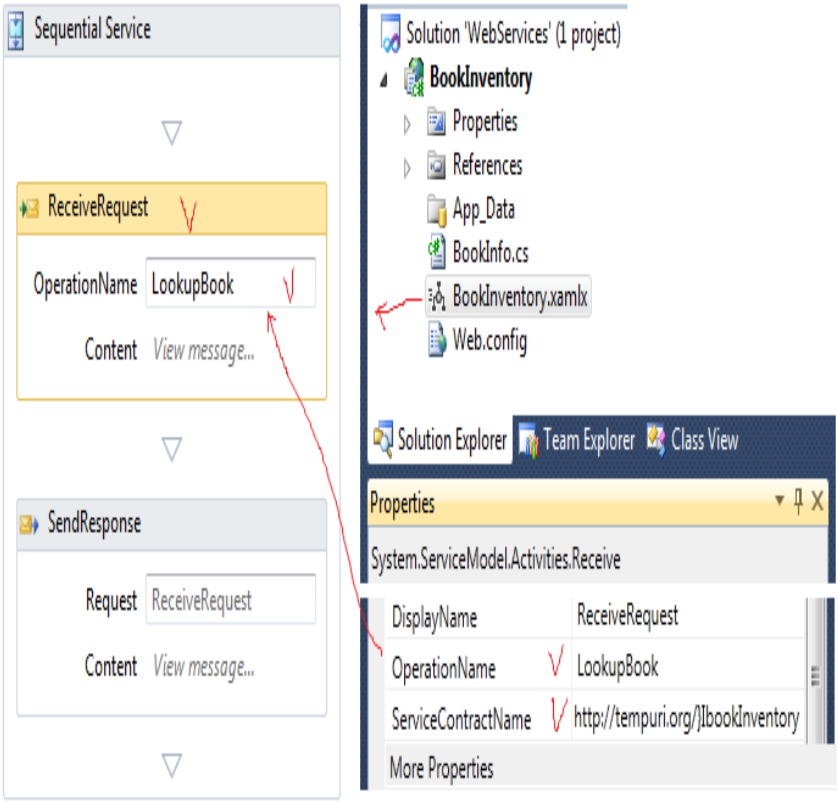
გადასამუშავებელი ცვლადი (handle variable) გამოიყენება პასუხის კორელაციისთვის იმ ეგზემპლართან, რომელმაც გააგზავნა მოთხოვნა.

მონაცემთა ცვლადი იქმნება გადასაცემი მონაცემების (ინფორმაციის) მიზნით. გავასუფთავოთ ცვლადების არე (data) და შევქმნათ ორი ახალი ცვლადის სახელი.

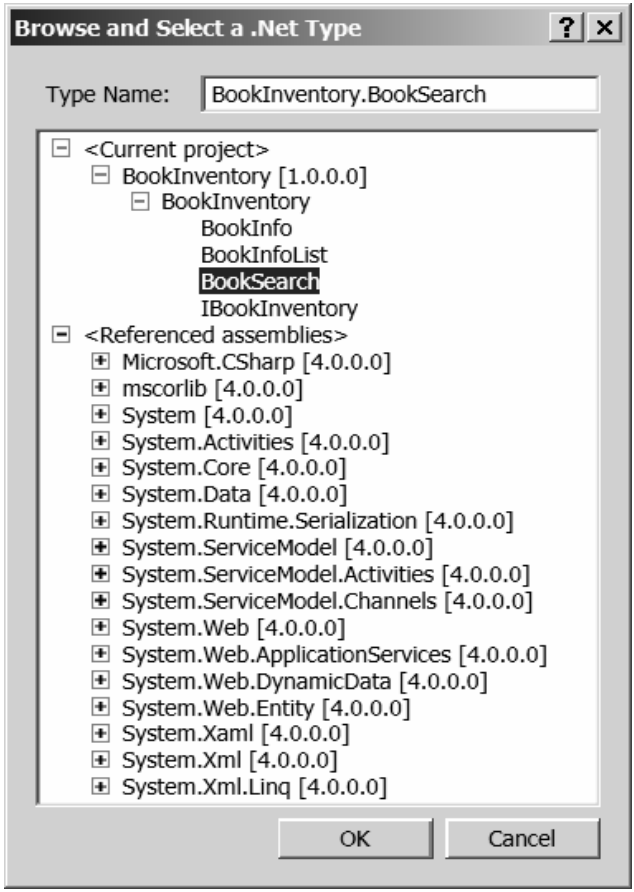
პირველისთვის ავირჩიოთ სახელი Name search და ტიპი - Browse for Typs. ახალ დიალოგურ ფანჯარაში BookInventory ნაკრები გავაფართოვოთ BookSearch-ით (ნახ.1.40).



ნახ.1.39-ა. საწყისი

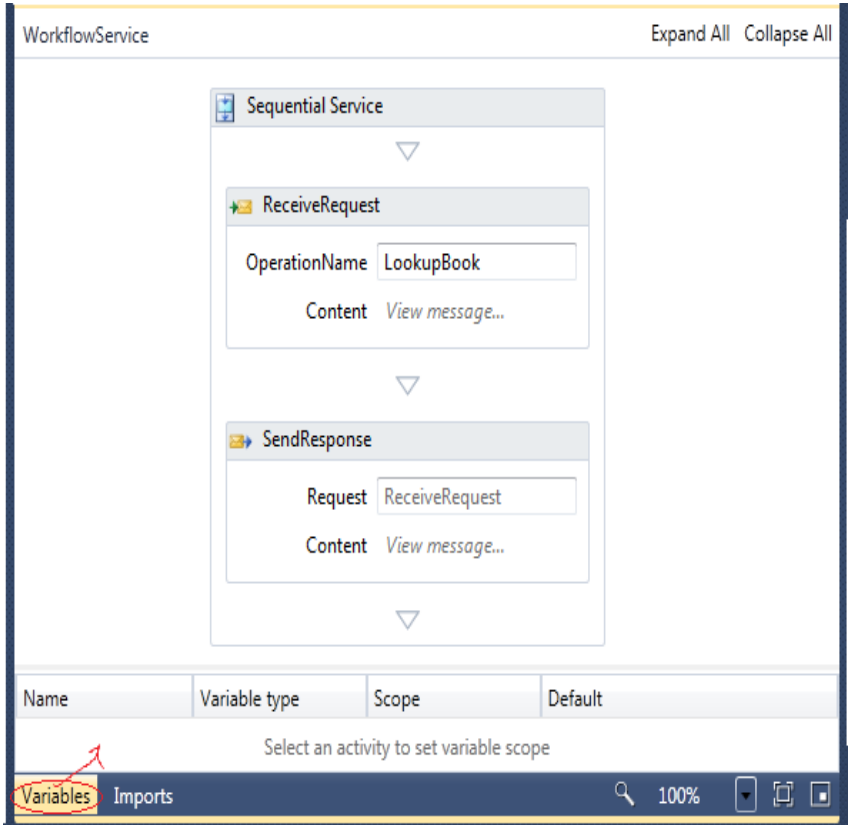


ნახ.1.39-ბ. საბოლოო

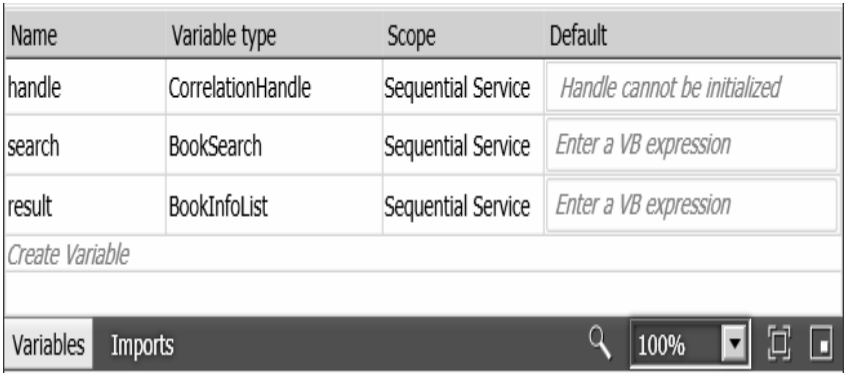


ნახ.1.40

მეორე ცვლადისთვის შეიტანეთ Name: result. ტიპი შეირჩევა Browse-დან, BookInfoList კლასით (ნახ.1.41). მიიღება 1.42 ნახაზზე მოცემული შემთხვევა.



ნახ.1.41. ცვლადები

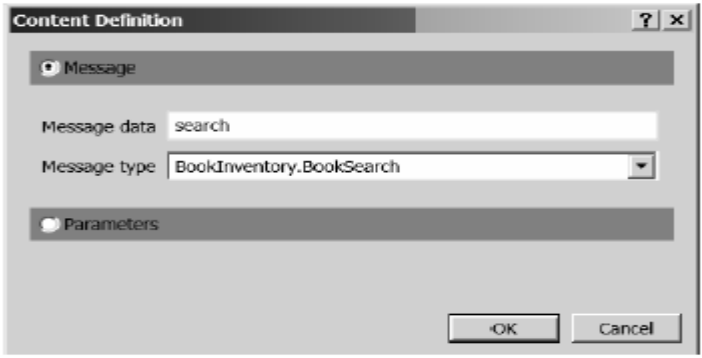


ნახ.1.42. ცვლადები განისაზღვრა მუშა პროცესისთვის

ბიზნესპროცესების დიზაინერში „ReceiveRequest“ (მოთხოვნების მიღების) ქმედებას აქვს view message (შეტყობინების ნახვის) ლინკი შინაარსის თვისებისთვის. მისი ამოქმედებით იხსნება დიალოგური ფანჯარა შემავალი შეტყობინების დასადგენად (შეიძლება ასევე სამ-წერტილიანი ღილაკის გამოყენებაც, თვისების გვერდით). შესასვლელი განისაზღვრება ორი ორი ხერხით: შეტყობინებით ან პარამეტრების ერთობლიობით.

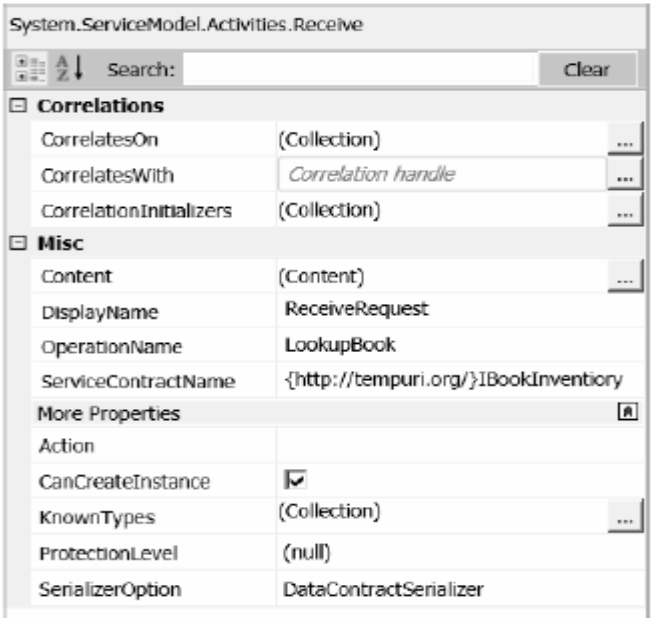
ამ თავში, მოგვიანებით ჩვენ განვიხილავთ მეორე ხერხსაც. ახლა დავაკვირდეთ, რომ რადიობუტონის გადამრთველი შეტყობინებისთვის სწორადაა არჩეული.

Message data თვისებისთვის შევიტანოთ **search.** ის მიუთითებს, რომ შემომავალი შეტყობინება უნდა ინახებოდეს search ცვლადში. Message ტიპისთვის ვირჩევთ BookInventory.BookSearch. დიალოგური ფანჯარა მოცემულია 1.43 ნახაზზე.



ნახ.1.43. შემავალი შეტყობინების განსაზღვრა

თვისებების ფანჯარა უნდა გამოიყურებოდეს 1.44 ნახაზზე ნაჩვენები სახით.



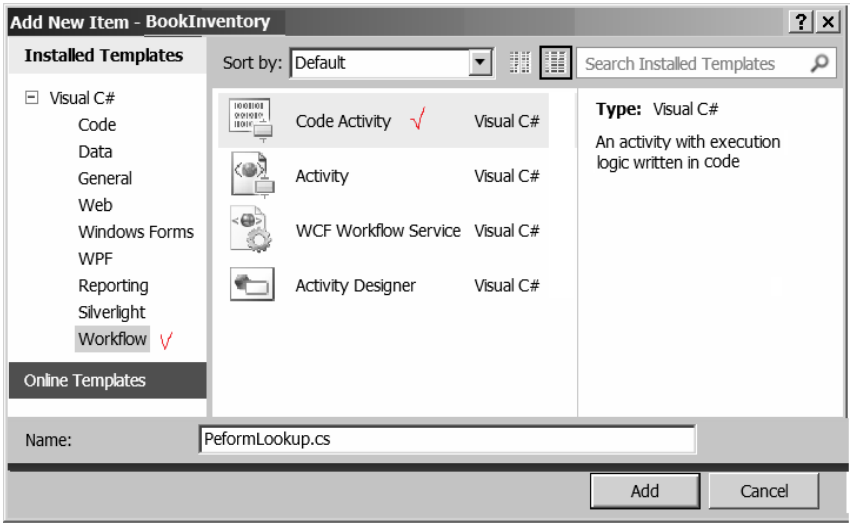
ნახ.1.44. Receive ქმედების თვისებათა ფანჯარა

ვირჩევთ ქმედებას „SendResponse“ და ავამოქმედებთ view message ლინკს. კვლავ შევამოწმოთ, რომ არჩეულია შეტყობინების გადამრთველი. Message data თვისებისთვის შევიტანოთ result, ხოლო Message type თვისებისთვის ვირჩევთ BookInfoList კლასს.

1.3.4. PerformLookup აქტიურობის შექმნა

ამ პროექტისთვის შევქმნით მომხმარებლის ქმედებას (აქტიურობას) “lookup“-ის (ძებნის) შესასრულებლად. ფაქტობრივად, მარტივი იქნება ხისტად-კოდირებული მონაცემების დაბრუნება. რეალურ სიტუაციაში მან ალბათ უნდა შეასრულოს მონაცემთა ბაზისადმი მოთხოვნა საჭირო მონაცემების მისაღებად.

Solution Explorer-ში BookInventory პროექტზე მარჯვენა ღილაკით ვირჩევთ Add ➤ New Item და დიალოგში Workflow კატეგორიისთვის ვირჩევთ Code Activity-ს. Name-ში შევიტანთ PerformLookup.cs სახელს (ნახ.1.45).



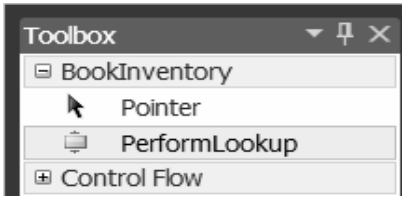
ნახ.1.45. მომხმარებლის ქმედების შექმნა

შევიტანოთ PerformLookup ქმედების რეალიზაციისთვის 3.2 ლისტინგში მოცემული კოდი.

// ლისტინგი 3.2 ----------

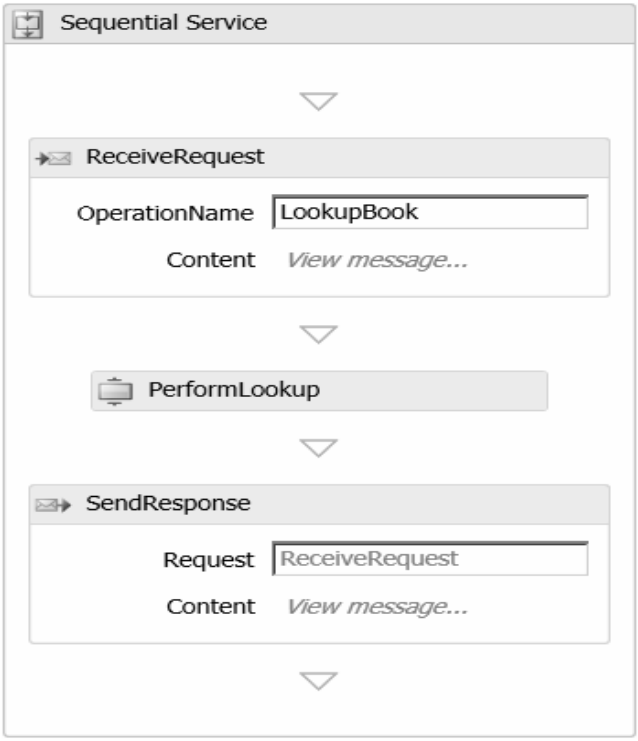
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;

namespace BookInventory
{
   /************************************************/
   // This custom activity creates a BookInfoList class
   // which is a collection of BookInfo classes.  It uses
   // the input parameters (BookSearch class) to "lookup"
   // the matching items. The BookInfoList class is
   // returned in the output parameter.
   /************************************************/
   public sealed class PerformLookup : CodeActivity
   {
      public InArgument<BookSearch> Search { get; set; }
      public OutArgument<BookInfoList> BookList { get; set; }

      protected override void Execute(CodeActivityContext context)
      {
         string author = Search.Get(context).Author;
         string title = Search.Get(context).Title;
         string isbn = Search.Get(context).ISBN;
         BookInfoList l = new BookInfoList();
         l.BookList.Add(new BookInfo(title, author, isbn, "Available"));
         l.BookList.Add(new BookInfo(title, author, isbn, "CheckedOut"));
         l.BookList.Add(new BookInfo(title, author, isbn, "Missing"));
         l.BookList.Add(new BookInfo(title, author, isbn, "Available"));
         BookList.Set(context, l);
      }
   }
}
```

F6-ით განვახორციელოთ აპლიკაციის აღდგენა. გავხსნათ BookInventory.xamlx ფაილი. გასათვალიწინებელია, რომ მომხმარებლის PerformLookup ქმედება არის ToolBox-ზე (ნახ.1.46).



ნახ.1.46

გადმოვიტანოთ PerformLookup ქმედება „ReceiveRequest” და „SendResponse” ქმედებებს შორის (ნახ.1.47).

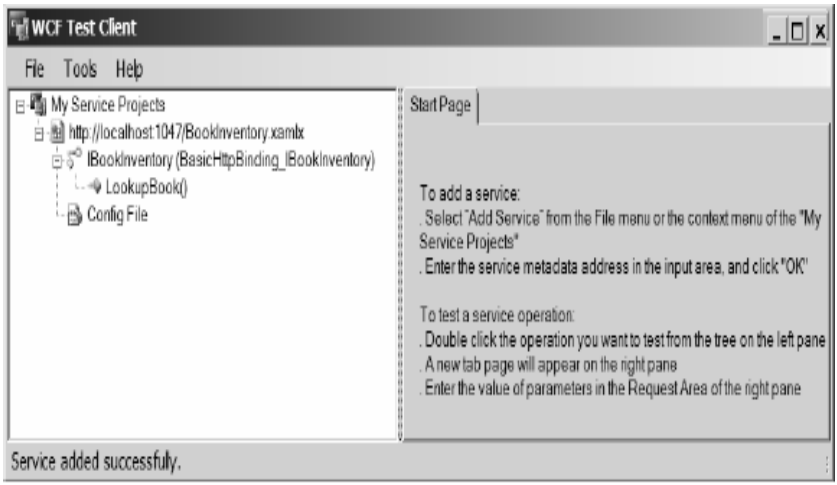


ნახ.1.47

ავირჩიოთ PerformLookup ქმედება. Properties-ის ფანჯარაში, BookList თვისებისთვის შევიტანოთ result; Search თვისებისთვის კი search.

1.3.5. სერვისის ტესტირება

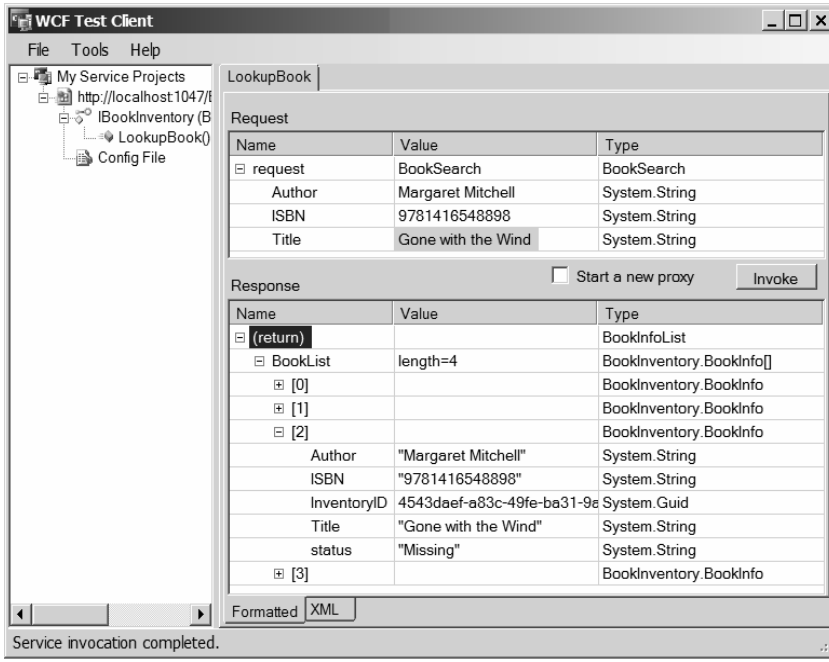
F5-ით ჩავატეროთ სერვისის გამართვა (debug). ვინაიდან ეს Web-სერვისია, Visual Studio ავტომატურად აამუშავებს WCF Test Client-ს. ეს მეტად მოსახერხებელი უტილიტაა. იგი ჩატვირთავს Web-სერვისებს და აღმოაჩენს მეთოდებს, რომლებიც გათვალისწინებულია. ისინი ჩანს 1.48 ნახაზის მარცხენა პანელზე.



ნახ.1.48. WCF Test Client ინიციალიზაციის ფანჯარა

LookupBook() მეთოდზე მაუსის 2-ჯერ დაჭერით მარჯვენა პანელის ზედა ნაწილში გამოიყოფა ადგილი შემოსული შეტყობინების განსათავსებლად. იგი მზადაა რთული შეტყობინებებისთვისაც, რომლებიც შეიცავს კლასების და თვისებების კოლექციებს.

შევიტანოთ ავტორი, ISBN-ნომერი, დასახელება. შემდეგ ავამოქმედოთ Invoke ღილაკი. გამოჩნდება შედეგები, რომლებიც ანალოგიურია 1.49 ნახაზზე ნაჩვენების.

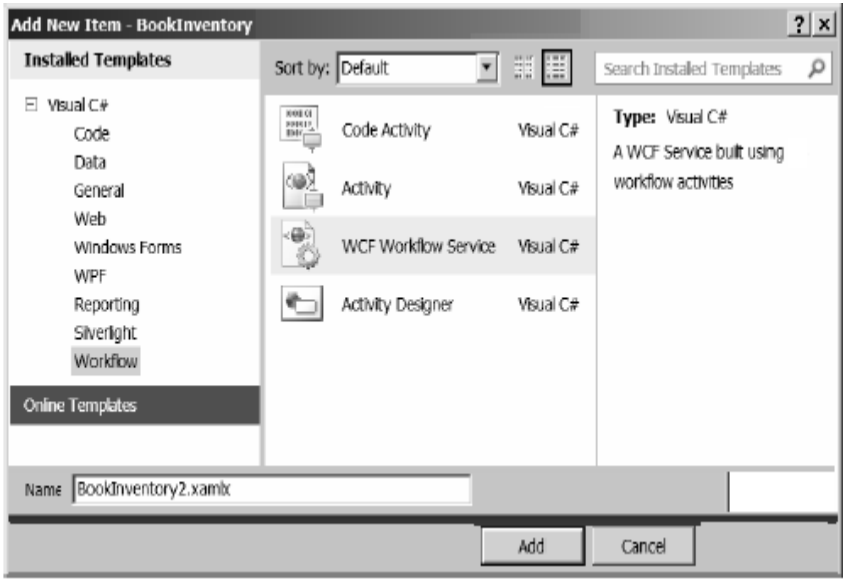


ნახ.1.49. WCF Client ტესტის სერვისის შედეგების ნახვა

სერვისი აბრუნებს BookInfo-ს ოთხ კლასს. 1.50 ნახაზზე მესამე ჩანაწერი გაფართოებულია, რათა დავინახოთ დაბრუნებული მონაცემების მაგალითი. მოცემულ კონკრეტულ ელემენტს აქვს სტატუსი Missing (დაკარგული, ამოვარდნილი).

<u>შენიშვნა:</u> თუ .xamlx ფაილი არის მომდინარე ფაილი Visual Studio-ში, როცა F5-ვაჭერთ, მაშინ WCF Test Client გაიშვება, როგორც აქაა ნაჩვენები. თუ სხვა ფაილია აქტიური, მაშინ გამოჩნდება

შესაბამისი კატალოგი და შედეგები. ის უნდა დაიხუროს და გააქტიურდეს ჩვენთვის საჭირო .xamlx ფაილი.



ნახ.1.50. WCF მუშა პროცესის სერვისის შექმნა

II თავი: მონაცემთა მენეჯმენტი

2.1. მონაცემთა ბაზების მართვის სისტემა Ms SQL Serever 2012

გამოყენებითი სფეროს აპლიკაციის მონაცემთა ბაზა (Database) შედგება ურთიერთდაკავშირებული ცხრილებისგან (Tables). ეს კავშირები ძირითადად რეალიზებულია პირველადი (Primary key - PK) და მეორეული (Foreign key - FK) გასაღებებით. შესაძლებელია ინდექსების გამოყენებაც (ინდექსური ფაილების შესაქმნელად).

ჩვენ ვგულისხმობთ, რომ მკითხველს აქვს საწყისი წარმოდგენა რელაციურ ბაზებთან სამუშაოდ და კერძოდ, MsSQLServer-თან. ამიტომ აქ ჩვენ მოკლედ განვიხილავთ ზოგიერთ საკითხს (გამეორების თვალსაზრისით), რაც დაგვჭირდება პროგრამული აპლიკაციების გასამართად, მომხმარებელთა ინტერფეისების სამუშაოდ მონაცემთა ბაზებთან.

2.1.1. მონაცემთა ძირითადი ტიპები

მონაცემთა ბაზის ცხრილი (Table), როგორც ვიცით სვეტების (ატრიბუტების) და სტრიქონებისგან (კორტეჟებისგან) შედგება. იგი სტრუქტურაა, რომლის ელემენტები შეიძლება მონაცემთა სხვადასხვა ტიპით განისაზღვროს (ნახ.2.1).

R1=„Student”

ID	A1	A2	…	An
მთელრიცხვა	სტრიქონული	ნამდვილ-რიცხვა	სხვა	თარიღი
101	აბესაძე ავთო	19		21/05/76
102	ბურძგლა დიტო	27		03/01/59
115	. . .	. . .	. . .	. . .

MsSQL Server-ში მონაცემთა ტიპები მრავალფეროვანია [10]. ერთი სვეტის (ატრიბუტის) ყველა მნიშვნელობა ერთი ტიპისაა. გამონაკლისია მხოლოდ SQL_VARIANT ტიპი, რომელშიც შესაძლებელია ერთდროულად რამდენიმე ტიპის მონაცემის შენახვა. მაგალითად, რიცხვითი, სტრიქონული ან თარიღის ტიპის მონაცემები.

მონაცემთა ტიპები იყოფა შემდეგ კატეგორიებად:

- რიცხვითი ტიპები;
- სიმბოლური ტიპები;
- დროითი ტიპები (თარიღი და დრო);
- სხვა ტიპები.

რიცხვითი ტიპები ცხრ.2.1

მონაცემთა ტიპი	ბაიტი	აღწერა
INT	4	მთელრიცხვა: -2^{31}-:-$2^{31}-1$
SMALLINT	2	მთელრიცხვა: -2^{15}-:-$2^{15}-1$
TINYINT	1	მთელრიცხვა: 0 - 255
BIGINT	8	მთელრიცხვა: -2^{63}-:-$2^{63}-1$
DEC(p,[s]) ან NUMERIC	5-:-17	p -სიზუსტის წილადი $-2^{38}+1$ 238 -1 s –თანრიგები წერტილის მარჯვნივ
REAL		მცოცავწერტილიანი დადებითი 2,23E –308 -:- 1,79E +308, უარყოფ. –1,18E –38 -:- –1,18E +38
FLOAT[(p)]	4 8	მცოცავწერტილიანი if p < 25 if p >= 25
MONEY	8	ფულის ტიპი: -2^{63} -:- $2^{63}-1$
SMALLMONEY	4	ფულის ტიპი: -2^{31} -:- $2^{31}-1$

სიმბოლური ტიპები ცხრ. 2.2

მონაცემთა ტიპი	ბაიტი	აღწერა
CHAR[(n)]	min 1	ფიქსირებული სიგრძის სტრიქონი n = 1-:- 8000 (სიმბოლო)
VARCHAR[(n)]	min 1	ცვლადი სიგრძის სტრიქონი 0 < n < 8000 (სიმბოლო)
NCHAR[(n)]	min 2	ფიქს.სიგრძის Unicode სტრიქონი n = 1-:- 4000 (სიმბოლო)
NVARCHAR[(n)]	min 2	ცვლადი სიგრძის Unicode სტრიქონი 0 < n < 4000 (სიმბოლო)

დროითი ტიპები (თარიღი და დრო) ცხრ.2.3

მონაცემთა ტიპი	ბაიტი	აღწერა
DATETIME	4	თარიღი და დრო დიაპაზონით: 01/01/1753 -:- 31/12/9999
SMALLDATETIME	2	თარიღი და დრო დიაპაზონით: 01/01/1900 -:- 06/06/2079
DATE	3	თარიღი დიაპაზონით: 01/01/0001-:-31/12/9999 მაგ.: 'mmm dd yyyy' ('Jan 07 2016'). SET DATEFORMAT-ით იცვლება
TIME	3-:-5	დრო 100 ნანოსეკ. სიზუსტით. მაგ.: 'hh:mm' ('21:45')
DATETIME2	6-:-8	თარიღი და დრო დიდი სიზუსტით
DATETIMEOFFSET	6-:-8	თარიღი და დრო დროის სარტყელით

ორობითი და ბიტური ტიპები ცხრ.2.4

მონაცემთა ტიპი	ბაიტი	აღწერა
BINARY[(n)]	=n	ფიქსირ. სიგრძის ბიტების სტრიქონი 0 < n < 8000
VARBINARY[(n)]	n-მდე	ცვლადი სიგრძის ბიტების სტრიქო- ნი 0 < n < 8000
BIT	1 ბიტი	ლოგიკური მნიშვნელობა: FALSE, TRUE და NULL

დიდი ობიექტების ტიპები ცხრ.2.5

მონაცემთა ტიპი		აღწერა
VARCHAR(max)		LOB : ობიექტი 2GB - მდე
NVARCHAR(max)		LOB : ობიექტი 2GB - მდე
VARBINARY(max)		BLOB: ატრიბუტით FILESTREAM შეინახება მონაცემები NTFS ფაილურ სისტემაში

UNIQUEIDENTIFIER - გლობალური უნიკალური იდენტიფიკატორების (Global Unique Identifier, GUID) შენახვის ტიპი;

TIMESTAMP - დროითი შტამპი, რომელიც აფიქსირებს სტრიქონის ყოველი ცვლილების დროს;

HIERARCHYID - ინახავს სრულ იერარქიას (მაგ.: თანამშრომელთა იერარქია, კატალოგში ფოლდერების იერარქია და ა.შ.);

SPARSE – „მეჩხერი“ სვეტების (შეიცავს ბევრ NUL მნიშვნელობას) შენახვის მოცულობის ოპტიმიზაცია.

2.1.2. მონაცემთა ბაზის ობიექტების შექმნა

მონაცემთა ბაზის ობიექტები არის ფიზიკური (დისკებზე), როგორიცაა ფაილები და ფაილთა ჯგუფები, ან ლოგიკური - მომხმარებელთა წარმოდგენები მონაცემთა ბაზის შესახებ. ლოგიკური ობიექტების მაგალითებია ცხრილები (Tables), სვეტები (Columns) და ვირტუალური ცხრილები (Views - წარმოდგენები).

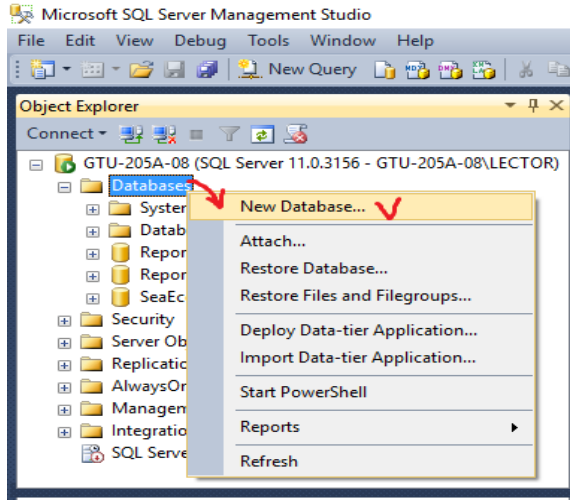
მონაცემთა ბაზის ობიექტი, რომელიც პირველ რიგში უნდა შეიქმნას, არის თვით მონაცემთა ბაზა. Database Engine კომპონენტი მართავს სისტემურ და მომხმარებელთა მონაცემთა ბაზებს. სისტემური ბაზები იქმნება მონაცემთა ბაზის ინსტალირების დროს, ხოლო მომხმარებელთა ბაზები კი - თვით ავტორიზებული მომხმარებლის მიერ.

მონაცემთა ბაზის შექმნა ორი მეთოდითაა შესაძლებელი. პირველი, SQL Server Management Studio-ს დახმარებით, რომლითაც ხდება დიალოგურ რეჟიმში პროცესების წარმართვა (ნახ.2.1, 2.2.).

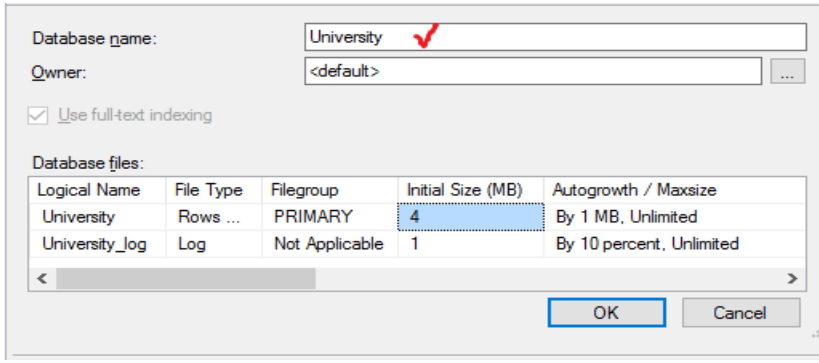
მეორე, Transact-SQL ენის CREATE DATABASE ინსტრუქციის დახმარებით:

```
CREATE DATABASE db_name
    [ON [PRIMARY] {file_spec1},.]
    [LOG ON {file_spec2},.]
        [COLLATE collation_name]
  [FOR {ATTACH ATTACH_REBUILD_LOG}]
```

ჩვენ აქ Transact-SQL ენას დეტალურად არ განვიხილავთ.



ნახ.2.2.1. ახალი ბაზის შექმნის დაწყება



ნახ.2.2.2. იქმნება University ბაზა

საჭიროა აღვნიშნოთ, რომ ერთ სისტემას შეუძლია 32 767 მონაცემთა ბაზის მართვა. ყველა ბაზა ინახება ფაილებში, რომლებიც შეიძლება ცხადად იყოს მითითებული ადმინისტრატორის მიერ ან არაცხადად იყოს წარმოდგენილი სისტემის მიერ. თუ CREATE DATABASE ინსტრუქცია შეიცავს ON პარამეტრს, მაშინ ბაზის ყველა ფაილი ცხადად არის გამოცხადებული.

დიდი ბაზებისთვის სასურველია ფაილთა ჯგუფების გამოყენება. ფაილი ინახავს ერთი ბაზის მონაცემებს. ფაილთა ჯგუფები საშუალებას იძლევა გადანაწილდეს მონაცემები სხადასხვა დისკებზე, შესრულდეს სარეზერვო დუბლირება და მონაცემთა ბაზის ნაწილის აღდგენის პროცედურა.

PRIMARY პარამეტრი მიუთითებს პირველ (მნიშვნელოვან) ფაილზე, რომელიც შეიცავს სისტემურ ცხრილებს და სხვა საჭირო შიგა ინფორმაციას ბაზის შესახებ.

COLLATE ოპციაში მიეთითება მოწესრიგების მიმდევრობა.

FOR ATTACH ოპცია მიუთითებს, რომ მონაცემთა ბაზა იქმნება უკვე არსებული ფაილების მიერთებით.

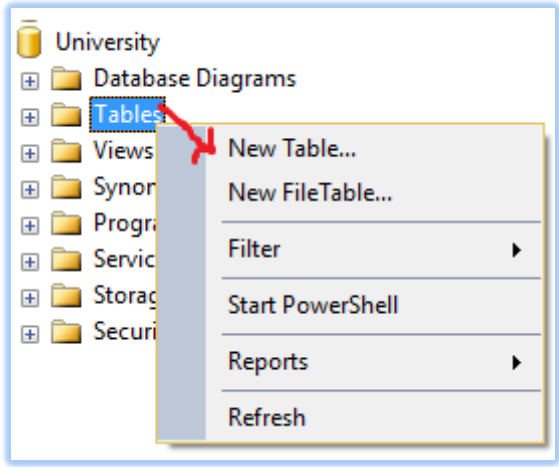
არსებული მონაცემთა ბაზის **მომენტალური სურათის** შექმნა CREATE DATABASE ინსტრუქციით (როცა ბაზაში დასრულებულია გარკვეული ტრანზაქციები და საჭიროა დუბლის შექმნა):

```
CREATE DATABASE database_snapshot_name
    ON (NAME = logical_file_name,
        FILENAME = 'C:\temp\file_name') [,...n ]
        AS SNAPSHOT OF source_database_name
```

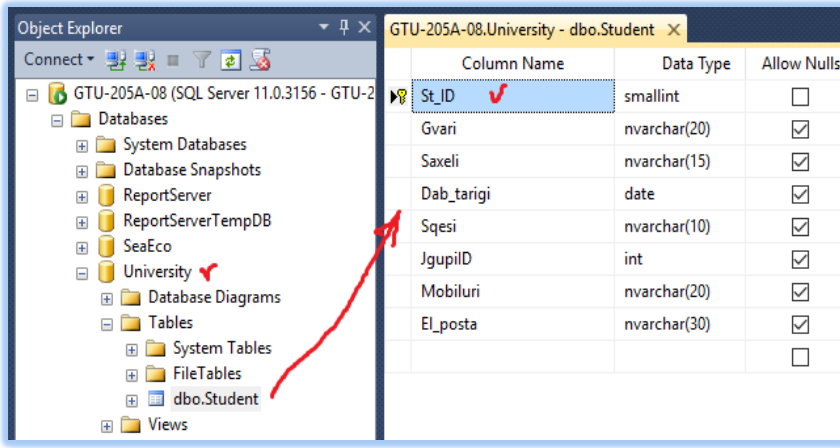
ამისათვის აქ გამოიყენება სტრიქონი AS SNAPSHOT OF. თავიდან უნდა შეიქმნას ბაზის მომენტალური სურათის შესანახი კატალოგი, მაგალითად, C:\temp\... . (NTFS - ფაილურ სისტემაში).

უნდა აღინიშნოს, რომ მომენტალური სურათი შეიცავს მონაცემთა ბაზის მხოლოდ შეცვლილ ნაწილს და ყოველი გადაღებული კადრი არ მოითხოვს დიდ დისკურ მეხსიერებას.

მონაცემთა ბაზის ცხრილის (Table) შექმნა ხორციელდება ინტერაქტიულ რეჟიმში SQL Server Management Studio-ს დახმარებით (ნახ.2.3). მომხმარებელს შეაქვს ცხრილის ატრიბუტები მათი ტიპების და სხვა მახასიათებლების მითითებით (ნახ.2.4). იქმნება რელაციური ცხრილის სტრუქტურა (მაგალითად, Student).



ნახ.2.3. ცხრილის შექმნის დაწყება



ნახ.2.4. Student ცხრილის შექმნა

2.5 ნახაზზე მოცემულია Student ცხრილის შევსების მაგალითი რამდენიმე სტრიქონით.

GTU-205A-08.University - dbo.Student

St_ID	Gvari	Saxeli	Dab_tariqi	Sqesi	JqupilD	Mobiluri	El_posta
1	აბაშიძე	აკაკი	1995-05-17	მამრობითი	108550	577102030	abashidze.ak@gmail.com
2	ბურდული	ბუდუ	1997-01-31	მამრობითი	108550	599707070	burdubu@yahoo.com
3	ბახტაძე	მერაბ	1995-01-01	მამრობითი	108551	591111222	bakhtadze.m@gmail.com
4	გაგუა	ნინო	1998-08-20	მდედრობითი	108551	577223355	gaguani@yahoo.com
5	კაკუბავა	ნინო	1998-05-09	მდედრობითი	108555	595777555	kakunino@gmail.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

ნახ.2.2.5. Student ცხრილის ფრაგმენტი

ცხრილების შექმნის მეორე გზა Transact-SQL ენის CREATE TABLE ინსტრუქციაა, რომლ;ის საბაზო ფორმა ასეთია:

CREATE TABLE table_name
(col_name1 type1 [NOT NULL | NULL)
[{, col_name2 type2 [NOT NULL | NULL]} ...])

აქ col_name ველის სახელია, ხოლო type - შესაბამისი ველის ტიპი.

ცხრილების მაქსიმალური რაოდენობა ერთ ბაზაში შეზღუდულია მონაცემთა ბაზის ობიექტების რაოდენობით, რომელთა რაოდენობა არ უნდა აღემატებოდეს 2 მილიარდს. აქ შედის ცხრილები (Tables), წარმოდგენები (Views), შენახვადი პროცედურები (Store procedures), ტრიგერები (Triggers) და შეზღუდვები (Constraints).

2.1.3. დეკლარაციული მთლიანობის შეზღუდვები

მონაცემთა ბაზების მართვის სისტემის ერთ-ერთი ყველაზე მნიშვნელოვანი მოთხოვნაა მონაცემთა მთლიანობის უზრუნველყოფა. შეზღუდვებს, რომლებიც გამოიყენება მონაცემთა შესამოწმებლად მათი ცვლილების ან დამატებისას, უწოდებენ მთლიანობის უზრუნველყოფის შეზღუდვებს (Integrity constraints).

მონაცემთა მთლიანობის უზრუნველყოფას გამოყენებით პროგრამაში ახორციელებს მომხმარებელი ან თვით მონაცემთა ბაზების მართვის სისტემა. ამ უკანასკნელ შემთხვევაში გვაქვს შემდეგი უპირატესობები:

- მაღლდება მონაცემთა საიმედოობა;
- მცირდება დაპროგრამების დრო;
- მარტივდება ტექნიკური მომსახურება.

მთლიანობის უზრუნველსაყოფად მონაცემთა ბაზების მართვის სისტემაას გააჩნია ორი ტიპის შეზღუდვა: დეკლარაციული შეზღუდვები და პროცედურული შეზღუდვები (ტრიგერებით რეალიზებადი).

დეკლარაციული შეზღუდვები განისაზღვრება DDL ენის CREATE TABLE და ALTER TABLE ინსტრუქციებით, სვეტების ან ცხრილების დონეზე. ყოველ დეკლარაციულ შეზღუდვას ენიჭება სახელი. იგი ენიჭება ცხადად CONSTRAINT ოფციის გამოყენებით CREATE TABLE ან ALTER TABLE ინსტრუქციაში.

დეკლარაციული შეზღუდვები შეიძლება დაჯგუფდეს შემდეგ კატეგორიებში:

- DEFAULT ;
- UNIQUE;
- PRIMARY KEY;
- CHECK;
- FOREIGN KEY.

(ჩავამატო მერე ----)

из главы 5, посредством инструкции CREATE TYPE

2.2. მონაცემთა ბაზების უსაფრთხოების სისტემა

მონაცემთა რელაციური ბაზების მართვის სისტემის ერთ-ერთი მნიშვნელოვანი კომპონენტია Database Engine, რომლის მომხმარებელთა ინტერფეისის დახმარებითაც საგრძნობლად მარტივდება სისტემასთან მუშაობა. აგრეთვე აქვს სხვადასხვა ინსტრუმენტი მონაცემთა ბაზის ობიექტების შესაქმნელად, დანართების ასაწყობად და სისტემური ადმინისტრირების ამოცანების სამართავად.

ჩვენ განვიხილავთ Database Engine კომპონენტის სისტემას მონაცემთა უსაფრთხოების უზრუნველყოფის მიზნით. ასეთია ავტორიზაციის (მონაცემთა ბაზასთან სანქცირებული მიმართვის) საკითხი, აუტენთიფიკაცის (მონაცემებთა მომხმარებელთა წვდომის პრივილეგიების) საკითხი, აგრეთვე მონაცემთა მოდიფიკაციის თანხლების შესაძლებლობები Database Engine კომპონენტით.

ამგვარად, მონაცემთა ბაზების უსაფრთხოების დაცვის ძირითადი კონცეფციებია:

- აუტენთიფიკაცია (მოხმარებლის სახელი, პაროლი);
- შიფრაცია (ინფორმაციის კოდირება, კრიპტოგრაფია);
- ავტორიზაცია (ბაზის რესურსების ფლობის ნებართვა);
- ცვლილებების მონიტორინგი (ბაზაში ყველა მიმართვის და ცვლილების ფიქსირება და დოკუმენტირება).

SQL Server სისტემაში მონაცემთა უსაფრთხოების მოდელი შედგება სამი განსხვავებული კატეგორიისგან:

- პრინციპალები (principals) - სუბიექტებია, რომელთაც გააჩნიათ წვდომის ნებართვა ბაზის განსაზღვრულ არსებთან (entities). არსებობს ასევე Windows-ჯგუფები და SQL Server-როლები. მომხმარებელს მიენიჭება შესაბამისი ჯგუფის ან როლის სააღრიცხვო ჩანაწერი, რაც მის უფლებებს განსაზღვრავს;
- დაცული ობიექტები (securables) - რესურსებია, რომლებზეც წვდომა რეგულირდება ბაზის ავტორიზაციის სისტემით;

❖ ნებართვები (permissions) - ყოველ დაცულ ობიექტს აქვს მასთან დაკავშირებული ნებართვა, რომელიც წარედგინება პრინციპალს.

2.2.1. აუტენთიფიკაცია

Database Engine კომპონენტის უსაფრთხოების სისტემა შედგება ორი განსხვავებული ქვესისტემისგან:

- Windows უსაფრთხოების სისტემები;
- SQL Server უსაფრთხოების სისტემები.

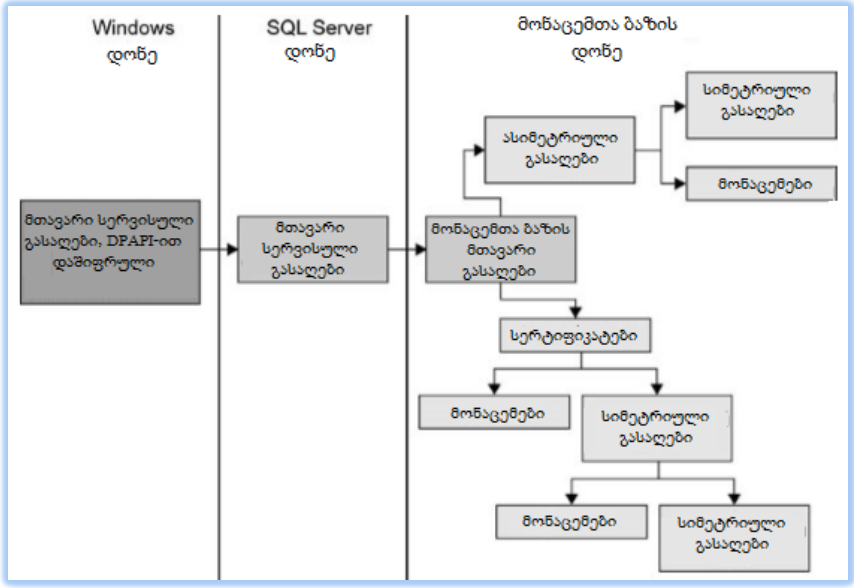
Windows უსაფრთხოების სისტემა განსაზღვრავს უსაფრთხოებას ოპერაციული სისტემის დონეზე. ესაა მეთოდი, რომლის დახმარებითაც მომხმარებელი შედის Windows სისტემაში.

SQL Server უსაფრთხოების სისტემა განსაზღვრავს დამატებით უსაფრთხოებას, რომელიც საჭიროა მონაცემთა ბაზის დონეზე. ესაა ხერხი, რომლითაც მომხმარებელი უკვე Windows სისტემაშია და ცდილობს მონაცემთა ბაზის სერვერთან დაკავშირებას.

2.2.2. მონაცემთა შიფრაცია

შიფრაცია არის მონაცემების კოდირება (მაგალითად, კრიპტოგრაფიის მეთოდებით), რომლის დროსაც გაუგებარი ხდება მათი შინაარსი. Database Engine კომპონენტი უზრუნველყოფს მონაცემთა დაცვას შიფრაციის იერარქიული დონეების მიხედვით და გასაღებების მართვის ინფრასტრუქტურით. შიფრაციის ყოველი დონე იცავს მის მომდევნო დონეს, იყენებს სერტიფიკატების, სიმეტრიული და ასიმეტრიული გასაღებების კომბინაციას (ნახ.2.6).

მთავარი სერვისული გასაღები მართავს ყველა დანარჩენ გასაღებს და სერტიფიკატებს. იგი იქმნება ავტომატურად Database Engine კომპონენტის დაყენებისას. ეს გასაღები დაშიფრულია API-Windows მონაცემთა დაცვის ინტერფეისის დახმარებით (DPAPI – Data Protection API) [11].



ნახ.2.6. შიფრაციის იერარქიული კომპონენტი [11]

მონაცემთა ყოველ ბაზას აქვს თავისი ერთი მთავარი გასაღები, რომელიც იქმნება CREATE MASTER KEY ინსტრუქციით. ეს გასაღები დაცულია სისტემის მთავარი სერვისული გასაღებით, რომელსაც შეუძლია ავტომატურად მისი გაშიფრვა.

მონაცემთა ბაზის მთავარი გასაღებით შესაძლებელია მომხმარებელთა გასაღებების შექმნა: სიმეტრიული, ასიმეტრიული და სერტიფიკატი. სიმეტრიულის დროს ორივე მხარეს (გადამცემი, მიმღები) აქვს ერთი გასაღები, ხოლო ასიმეტრიულის დროს - სხვადასხვა. პირველი უფრო მარტივია, მეორე კი - საიმედო.

SQL Server-ში გამოიყენება მონაცემთა შიფრაციის ორი ხერხი: შიფრაცია სვეტების დონეზე და გამჭვირვალე შიფრაცია (სიმეტრიული გასაღების საფუძველზე). უფრო დეტალურად შეიძლება ამ საკითხების გაცნობა [11,36].

2.3. სივრცითი მონაცემთა ტიპები SQL Serever -ში

21-ე საუკუნის დასაწყისიდან ბიზნესის, გეოსისტემების და სხვა სფეროებში მზარდი მოთხოვნილება გაჩნდა და მნიშვნელოვანი განვითარება დაიწყო სივრცითი მონაცემების მოდელირების, მონაცემთა ბაზაში შენახვის და მათი ასხვის ტექნოლოგიების ტექნოლოგიებმა. ამ პროცესებზე განსაკუთრებით დიდი გავლენა იქონია „მაიკროსოფთის“ კორპორაციის Virtual Earth სისტემის და GPS ინსტრუმენტების (Global Positioning Systems – ნავიგაციის გლობალური სისტემა და ადგილმდებარეობის განსაზღვრა) გამოჩენამ [11].

მოდელები. სივრცითი მონაცემების ასახვის მიზნით დღეისათვის გამოიყენება ორი სახის მოდელი:

- გეოდეზიური სივრცითი მოდელები და
- ბრტყელი სივრცითი მოდელები.

პლანეტები, როგორც რთული ობიექტები შეიძლება წარმოდგენილ იყოს სფეროიდების სახით. ამის ერთ-ერთი კარგი მაგალითია გლობუსი – დედამიწის მოდელი. მისი ზედაპირის კონკრეტული წერტილი აღიწერება განედით (პარალელები, ეკვატორის ჩრდილოეთით და სამხრეთით) და გრძედით (მერიდიანები, მაგალითად, 0-ვანი მერიდიანის დასავლეთით ან აღმოსავლეთით). ასეთ მოდელებს უწოდებენ გეოდეზიურს.

ბრტყელ სივრცით მოდელებში, მაგალითად დედამიწის ასახვის მიზნით, გამოიყენება ორგანზომილებიანი რუქები.

ამგვარად, ჩვენ განვიხილავთ სივრცითი მონაცემების ორ ტიპს:

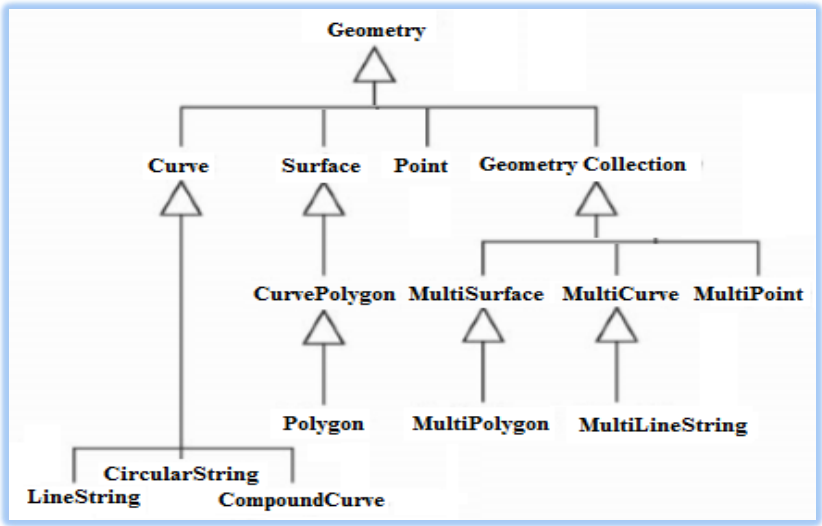
- GEOMETRY – ბრტყელი სივრცითი მოდელი;
- GEOGRAPHY – გეოდეზიური მოდელი.

მონაცემთა ასეთი ტიპები რეალიზებულია და გამოიყენება მონაცემთა ბაზების მართვის სისტემის, SQL Server 2012/14 ვერსიებში.

2.3.1. მონაცემთა ტიპი GEOMETRY

ტერმინი - გეომეტრიული ობიექტი შემოიტანა OGC (Open Geospatial Consortium) კონსორციუმმა, რათა შესაძლებელი ყოფილიყო ისეთი სივრცითი თვისებების ასახვა, როგორიცაა წერტილები და წრფეები. ამიტომაც გეომეტრიული ობიექტი მონაცემებს წარმოადგენს ორგანზომილებიან სივრცეში წერტილების, წრფეების და მრავალკუთხედების სახით.

გეომეტრიული ობიექტის მონაცემთა ტიპს აქვს ქვეტიპები, როგორც ეს 2.6 ნახაზზეა ნაჩვენები [11].



ნახ.2.6. ტიპების იერარქია GEOMETRY ფესვური ტიპით

როგორც ნახაზიდან ჩანს, ქვეკლასები დაყოფილია ორ კატეგორიად: საბაზო გეომეტრიულ ქვეკლასებად და ერთგვაროვან კოლექციათა ქვეკლასებად.

საბაზო გეომეტრიული ქვეკლასები შედგება Point, LineString და Polygon ქვეკლასებისგან, ხოლო ერთგვაროვანი კოლექციები -

MultiPoint, MultiLineString და MultiPolygon ქვეკლასებისგან, მათ საკუთარი თვისებებიც გააჩნია.

ახლა განვიხილოთ სივრცითი მონაცემების აღნიშნული ტიპების მოკლე დახასიათება:

- **Point (წერტილი)** – არის ორგანზომილებიანი გეომეტრიული ობიექტი კოორდინატთა X და Y მნიშვნელობებით, ამ ტიპის ეგზემპლარს შეიძლება ჰქონდეს ორი დამატებითი კოორდინატა: დონე (Z) და ზომა (M). Point ტიპის ეგზემპლარები გამოიყენება რთული სივრცითი ტიპების ასაგებად;
- **MultiPoint (მრავალწერტილოვანი)** – არის კოლექცია ნულოვანი ან მეტი რაოდენობის წერტილებისა. არაა აუცილებელი მრავალ-წერტილოვან ტიპში წერტილები იყოს განსხვავებული;
- **LineString (ტეხილი ხაზი)** – არის გეომეტრიული ობიექტი, რომელსაც აქვს სიგრძე და შეინახება წერტილების მიმდევრობით, რომელიც განსაზღვრავს ხაზის გზას. ტეხილ ხაზს აქვს საკონტროლო წერტილები (გადახრის ადგილი). ტეხილი ხაზი მარტივია (simple), თუ იგი არ კვეთს თავის თავს. თუ ტეხილი ხაზის საწყისი და საბოლოო წერტილები ემთხვევა, იგი ჩაკეტილია (closed). ჩაკეტილ, მარტივ ტეხილს უწოდებენ რგოლს (ring).
- **MultiLineString (მრავალტეხილი ხაზები)** – არის კოლექცია ნულოვანი ან მეტი რაოდენობის ტეხილი ხაზის.
- **Polygon (მრავალკუთხედი)** – არის ორგანზომილებიანი გეომეტრიული ფიგურა ზედაპირით. მრავალკუთხედი ინახება მიმდევრობითი წერტილების სახით, რომლებითაც განისაზღვრება მისი გარეგანი შემომსაზღვრელი პერიმეტრი (ring) და შიგა მრავალკუთხედები (ნული ან რამდენიმე). გარე და შიგა ფიგურბი იძლევა მრავალკუთხედის საზღვრებს, ხოლო სივრცე მათ შორის კი - განსაზღვრავს შიგა ნაწილს.
- **MultiPolygon ()** – მრავალკუთხედების კოლექციაა (0 ან მეტი).

- **GeometryCollection** – გეომეტრიული ეგზემპლარების კოლექციაა 0 ან მეტი გეომეტრიული ობიექტით. ამ ტიპის ეგზემპლარი შეიძლება შეიცავდეს GEOMETRY ტიპის ნებისმიერ ქვეტიპს.

2.6 ნახაზიდან ჩანს, რომ არსებობს სამი ქვეტიპი, რომლისთვისაც შეიძლება ეგზემპლარების შექმნა: CircularString, CompoundCurve და CurvePolygon. ეს ქვეტიპები ახალია SQL Server 2012 -ში და გამოიყენება სივრცით მონაცემებთან სამუშაოთ.

2.3.2. მონაცემთა ტიპი GEOGRAPHY

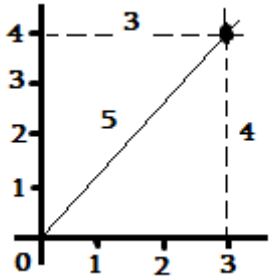
როგორც აღინიშნა, გეომეტრიული ტიპი მონაცემებს ინახავს X და Y კოორდინატებით, ხოლო გეოგრაფიული მონაცემები კი – GPS სისტემის განედისა და გრძედის გამოყენებით. გრძედი არის ჰორიზონტალური კუთხე (-180^0-:- $+180^0$) დიაპაზონში, ხოლო განედი – ვერტიკალური კუთხე (-90^0-:- $+90^0$) დიაპაზონში.

გეომეტრიულისგან განსხვავებით გეოგრაფიული ტიპისთვის უნდა მიეთითოს კონკრეტული კოორდინატთა სივრცითი სისტემა შესაბამისი მთელრიცხვა იდენტიფიკატირით SRID (Spatial Reference ID). მონაცემთა GEOMETRY ტიპის ეგზემპლარები გამოყენება აგრეთვე GEOGRAPHY ტიპისთვის.

განსხვავება GEOMETRY და GEOGRAPHY ტიპებს შორის, როგორც ადრე აღვნიშნეთ ისაა, რომ გეომეტრიული მონაცემები გამოიყენება ბრტყელ სივრცით მოდელებში, ხოლო გეოგრაფიულ კი გეოდეზიურ მოდელებში.

ძირითადი განსხვავება სივრცით მონაცემთა ამ მოდელებს შორის ისაა, რომ GEOMETRY ტიპის მონაცემებისთვის მანძილები და ფართობები მიეთითება განზომილების იმავე ერთეულებში, როგორც ეს გამოიყენებოდა ეგზემპლარებია კოორდინატებში. მაგალითად, მანძილი ორ (0,0) და (3,4) წერტილს შორის იქნება 5 ერთეული

(ნახ.2.7). მართკუთხა სამკუთხედის ჰიპოტენუზა – პითაგორას თეორემის საფუძველზე.



ნახ.2.7. ორ წერტილს შორის სიგრძე გეომეტრიული კოორდინატებში

GEOGRAPHY ტიპის მონაცემებისთვის გამოიყენება ელიპსოიდური კოორდინატები, რომლებიც იზომება განედისა და გრძედის კუთხეებში. ამასთანავე, ამ ტიპის მონაცემებზე გაითვალისწინება გარკვეული შეზღუდვები, მაგალითად, ასეთი ტიპის ყოველი ეგზემპლარი უნდა მოთავსდეს ერთი ნახევარსფეროს შიგნით.

მონაცემთა გარე ფორმატები. SQL Server მონაცემთა ასახვის მიზნით იყენებს სამი ტიპის გარე ფორმატს მათი რეალიზაციის ფორმისგან დამოუკიდებლად:

- WKT (Well-Known Text – ცნობილი ტესტური ფორმატი) – გამოიყენება სივრცით სტრუქტურათა სივრცითი კოორდინატების სისტემების ასახვისათვის და გარდაქმნებისათვის კოორდინატთა სისტემებს შორის;
- WKB (Well-Known Binary – ცნობილი ორობითი ფორმატი) – არის WKT-ს ორობითი ექვივალენტი;
- GML (Geography Markup Language – გეოგრაფიული ფორმატირების ენა) – არის XML ენის გრამატიკა, განსაზღვრული OGC კონსორციუმის მიერ, გეოგრაფიული თვისებების გამოსახვისათვის.

ესაა მონაცემთა გაცვლის ღია ფორმატი გეოგრაფიული ტრანზაქციების შესასრულებლად ინტერნეტში.

საილუსტრაციოდ განვიხილოთ WKT ფორმატის მაგალითები:

– POINT(3,4) – მნიშვნელობები მიუთითებს X და Y კოორდინატებს;

– LINESTRING(0 0, 3 4) – პირველი ორო რიცხვი საწყისი წერტილის X და Y კოორდინატების, ხოლო მესამე და მეოთხე – ბოლო წერტილისა;

– POLYGON(300 0, 150 0, 150 150, 300 150, 300 0) – რიცხვების ყოველი წყვილი არის წერტილი მრავალკუთხედზე. ჩვენ შემთხვევაში პირველი და ბოლო წერტილები ემთხვევა.

2.3.3. სივრცით მონაცემებთან მუშაობა

SQL Server -ში სივრცითი მონაცემებისათვის გამოიყენება, როგორც აღვნიშნეთ, ორი ტიპი: GEOMETRY და GEOGRAPHY. ესაა მომხმარებლის მიერ განსაზღვრული მონაცემთა ტიპები, რომლებიც რეალიზდება SQL Server დანართის სახით CLR გარემოს გამოყენებით.

ამ ტიპებს გააჩნია ქვეტიპები, რომლებიც, რომლებიც შეიძლება იყოს ეგზემპლირებადი ან არაეგზემპლირებადი.

ეგზემპლირებადი ქვეტიპების შემთხვევაში შესაძლებელია ეგზემპლარების შექმნა და მათთან მუშაობა. ეს ეგზემპლარები შეიძლება შენახული იყოს როგორც ცხრილის სვეტები, ან როგორც ცვლადების მნიშვნელობები ან პარამეტრები.

არაეგზემპლირებადი ქვეტიპებისთვის ეგზემპლარების შექმნა არ შეიძლება. კლასთა იერარქიაში ფესვური (აბსტრაქტული) კლასი არაეგზემპლირებადია.

2.3.3.1. GEOMETRY ტიპის მონაცემებთან მუშაობა

მაგალითი_1: ცხრილი „უალკოჰოლო_სასმელები“ შედგება სამი სვეტისაგან. პირველი - id გენერირდება სისტემის მიერ, მეორე - name გამოიყენება სამელის დასახელების შესანახად, მესამე - shape შეიცავს ინფორმაციას ბაზრის სფეროს ფორმის შესახებ, რომელშიც მყიდველები უფრო მეტ უპირატესობას ანიჭებენ რომელიმე კონკრეტულ სამელს. პირველი სამი INSERT ინსტრუქცია ქმნის სამ სფეროს, რომლებშიც კონკრეტული სასმელია პრიორიტეტული. ყოველი ეს სამი სფერო არის მრავალკუთხედი. მეოთხე INSERT ინსტრუქცია სვამს წერტილს, ვინაიდან ეს სპეციფიკური სასმელი იყიდება მხოლოდ ერთ ადგილას.

2.1 ლისტინგზე მოცემულია ცხრილის აგების Transact_SQL პროგრამის ტექსტი.

```
USE sample;
CREATE TABLE beverage_markets
 ( id INTEGER IDENTITY(1,1),
   name VARCHAR(25), shape GEOMETRY);
   INSERT INTO beverage_markets
   VALUES ('Coke1', GEOMETRY::STGeomFromText
    ('POLYGON ((0 0, 150 0, 150 150, 0 150, 0 0))', 0));
   INSERT INTO beverage_markets
   VALUES ('Pepsi', GEOMETRY::STGeomFromText
('POLYGON ((300 0, 150 0, 150 150, 300 150, 300 0))', 0));
   INSERT INTO beverage_markets
   VALUES ('7UP', GEOMETRY::STGeomFromText
('POLYGON ((300 0, 150 0, 150 150, 300 150, 300 0))', 0));
   INSERT INTO beverage_markets
   VALUES ('Almdudler', GEOMETRY::STGeomFromText
('POINT (50 0)', 0));
```

ლისტინგი 2.1

ლისტინგიდან ჩანს მონაცემთა გეომეტრიულ ტიპებთან მუშაობის მეთოდი GEOMETRY::STGeomFromText(). ეს სტატიკური მეთოდი გამოიყენება გეომეტრიული ფიგურების (მრავალკუთხედი, წერტილები) კოორდინატების ჩასასმელად.

ტიპს შეიძლება ჰქნდეს მეთოდების ორი განსხვავებული ჯგუფი: სტატიკური მეთოდები და კლასის ეგზემპლარების მეთოდები. პირველი გამოიყენება მთლიანი კლასისითვის, ხოლო მეორე - კლასის გარკვეული ეგზემპლარებისთვის. ამ ჯგუფთა მეთოდების გამოძახება სხვადა-სხვანაირად ხდება.

სტსტიკურისთვის გამოიყენება სიმბოლო „ :: “, ხოლო ეგზემპლარის მეთოდისთვის კი – „ . “, მაგალითად:

GEOMETRY :: STGeomFromText;

@g.STContains;

გარდა აღნიშნულისა გამოიყენება ასევე სამი სხვა სტატიკური მეთოდებიც:

- STPointFromText() – დააბრუნებს POINT მონაცემთა ტიპის ეგზემპლარის WKT წარმოდგენას;
- STLineFromText() – დააბრუნებს LINESTRING მონაცემთა ტიპის ეგზემპლარის WKT წარმოდგენას, სიმაღლის და ზომის მნიშვნელობათა დამატებით;
- STPolyFromText() – დააბრუნებს MULTIPOLYGON მონაცემთა ტიპის ეგზემპლარის WKT წარმოდგენას, სიმაღლის და ზომის მნიშვნელობათა დამატებით;

სივრცით მონაცემებზე მოთხოვნები სრულდება ისევე, როგორც რელაციურ მონაცემებზე. ჩვენი მაგალითისთვის, დავუშვათ, რომ ვირჩევთ ინფორმაციას beverage_markets ცხრილის shape სვეტიდან.

2.2 ლისტინგში აგებულია მოთხოვნა, რომელიც განსაზღვრავს „არსებობს თუა არა Almdudler სასმელის გამყიდველი მაღაზია იმ რაიონში, სადაც პრიორიტეტულია სასმელი Coce“.

```
DECLARE @g geometry;
DECLARE @h geometry;
SELECT @h = shape FROM beverage_markets
      WHERE name ='Almdudler';
SELECT @g = shape FROM beverage_markets
      WHERE name = 'Coke';
SELECT @g.STContains(@h);
```

ლისტინგი 2.2

ეს მოთხოვნა აბრუნებს პასუხს 0-ს, რაც ნიშნავს, რომ ამ რაიონში ასეთი მაღაზია არაა.

STContains() მეთოდი დააბრუნებს 1-ს, თუ GEOMETRY ტიპის მონაცემის ერთი ეგზემპლარი შეიცავს ამავე ტიპის მეორე ეგზემპლარს, რომელიც მეთოდის პარამეტრშია მითითებული.

2.3 ლისტინგში ასახულია მოთხოვნა, რომელიც განსაზღვრავს „ shape სვეტის სიგრძეს და წარმოდგენას იმ მაღაზიისთვის, რომელიც ყიდის სასმელს Almdudler“.

```
SELECT id, shape.ToString() AS wkt, shape.STLength() AS length
    FROM beverage_markets
    WHERE name = 'Almdudler';
```

ლისტინგი 2.3

შედეგი გამოიტანება შემდეგი სახით:

Id	wkt	Length
4	POINT (500)	0

ამ მაგალითში STLength() მეთოდი აბრუნებს GEOMETRY მონაცემთა ტიპის ელემენტის საერთო სიგრძეს. 0 ნიშნავს, რომ მნიშვნელობა წერტილია. ToString() მეთოდი აბრუნებს სტრიქონს wkt ფორმატში, რომელიც შეიცავს მოცემული ეგზემპლარის ლოგიკურ წარმოდგენას.

2.4 ლისტინგში განიხილება STIntersects() მეთოდის გამოყენება. კერძოდ, მოთხოვნაა: „განისაზღვროს იკვეთება თუ არა რაიონი, რომელშიც იყიდება Coce, რაიონთან, რომელშიც იყიდება Pepsi”.

```
USE sample;
DECLARE @g geometry;
DECLARE @h geometry;
SELECT @h = shape FROM beverage_markets WHERE name = 'Coke';
SELECT @g = shape FROM beverage_markets WHERE name = 'Pepsi';
SELECT @g.STIntersects(@h);
```

ლისტინგი 2.4

STIntersects() მეთოდის გამოყენების შედეგად მიიღება 1 (TRUE), რაც ნიშნავს, რომ ეს ორი გეომეტრიული ფიგურა, რომლებიც ასახავს გაყიდვების სფეროებს, იკვეთება.

2.5 ლისტინგში განიხილება STIntersection() მეთოდის გამოყენება.

```
USE   sample;
DECLARE @poly1 GEOMETRY = 'POLYGON ((1 1, 1 4, 4 4, 4 1, 1 1))';
DECLARE @poly2 GEOMETRY = 'POLYGON ((2 2, 2 6, 6 6, 6 2, 2 2))';
DECLARE Sresult GEOMETRY;
SELECT Sresult = @poly1.STIntersection(@poly2);
SELECT Sresult.STAsText();
```

ლისტინგი 2.5

შედეგად მიიღება შემდეგი სტრიქონი (დამრგვალებული მნიშვნელობებით):

POLYGON ((22, 42, 44, 24, 22))

მეთოდი STIntersection() აბრუნებს ობიექტს, რომელიც ასახავს წერტილებს, სადაც GEOMETRY მონაცემთა ტიპის ეგზემპლარი იკვეთება ამავე ტიპის სხვა ეგზემპლართან. ამიტომაც. ამ მოთხოვნის საფუძველზე მიიღება მართკუთხედი, რომელზეც იკვეთება მრავალკუთხედი (გამოცხადებული @poly1 ცვლადით) მეორე მრავალკუთხედთან (გამოცხადებული @poly2 ცვლადით).

STAsText() მეთოდი აბრუნებს შედეგს wkt ფორმატში.

2.3.3.2. GEOGRAPHY ტიპის მონაცემებთან მუშაობა

GEOGRAPHY ტიპის მონაცემების დასამუშავებლადაც ასევე გამოიყენება იგივე სტატიკური და ეგზემლლარის მეთოდები, რაც იყო GEOMETRY მონაცემთა ტიპებისთვის. 2.6 ლისტინგში ნაჩენებია ერთი მაგალითი:

```
USE AdventureWorks;
SELECT SpatialLocation, City
    FROM Person.Address
    WHERE City = 'Dallas';
```

ლისტინგი 2.6

ამ მოთხოვნის შესრულების შედეგები ნაჩვენებია ქვემოთ, SpatialLocation, City ცხრილში.

AdventureWorks მონაცემთა ბაზის Address ცხრილი შეიცავს SpatialLocation სვეტს, რომელიც არის GEOGRAPHY მონაცემთა ტიპის. ჩვენი მაგალითში მოითხოვება იმ თანამშრომელთა გეოგრაფიული მდებარეობის დადგენა, რომლებიც ცხოვრობე ქალაქ დალასში.

SpatialLocation	City
0xE6100000010C4DD2603933694040 26C0A31BF73458C0	Dallas
0xE6100000010C10A810D1886240403A0F0653663158C0	Dallas
0xE6100000010C4346160AA26440406340F0E64F3 B58C0	Dallas
0xE6100000010C107E16DAAD6540403DA892EAD52C58C0	Dallas
0xE6100000010C8044A1422D5F4040F66D784F983758C0	Dallas
0xE6100000010C8E345943826A4040839B00B8E03358C0	Dallas
0xE6100000010CAA5BBD5FAB69404087866D198D3C58C0	Dallas

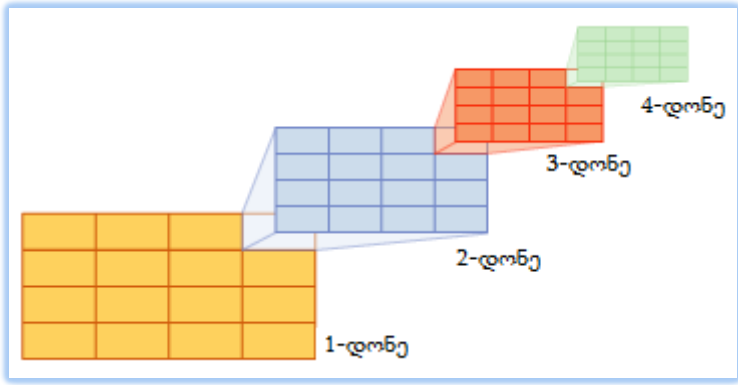
ამ ცხრილის SpatialLocation სვეტის მნიშვნელობებში ასახულია მისამართების გრძედი და განედი 16-ით კოდში თითოეული თანამშრომლისთვის ქალაქ დალასიდან.

შემდგომში ჩვენ განვიხილავთ ამ მაგალითს SQL Server Management Studio -ს გამოყენებით.

2.3.4. სივრცით ინდექსებთან მუშაობა

ინდექსირება, როგორც ცნობილია, გამოიყენება მონაცემებთან სწრაფი მიმართვის განსახორციელებლად. სივრცით მონაცემებთან მიმართვის დასაჩქარებლად კი საჭიროა ე.წ. სივრცითი ინდექსები, რომლებიც განისაზღვრება ცხრილის სვეტისთვის GEOMETRY ან GEOGRAPHY მონაცემთა ტიპით.

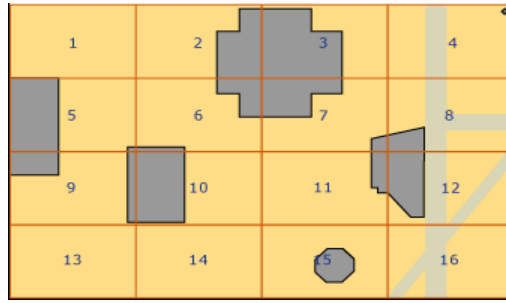
SQL Server-ში ასეთი ინდექსების ასაგებად გამოიყენება B-ხეები. ინდექსები აქ ასახავს ორ განზომილებას B-ხეების წრფივ რიგში. სივრცით ინდექსში მონაცემების მოთავსებამდე სისტემა ყოფს სივრცეს იერარქიულად ერთგვაროვანი წესით. ინდექსის შექმნის პროცესი სივრცეს ყოფს ოთხდონიანი ბადისებრი იერარქიის სახით (ნახ.2.8) [12,13]. ცხრილს უნდა ჰქონდეს კლასტერიზებული პირველადი გასაღები.



ნახ.2.8. სივრცითი ინდექსის ოთხდონიანი ბადისებრი იერარქია

ყველა დონის ბადეს უნდა ჰქონდეს უჯრედების ერთნაირი რაოდენობა (მაგალითად, 4x4 ან 8x8) და ყველა უჯრა ერთი ზომისაა. ნახაზზე ნაჩვენებია ბადის ზედა-მარჯვენა უჯრის დეკომპოზიცია ბადის იერარქიის ზედა დონეებზე. დეკომპოზიცია სრულდება ყველა უჯრედისთვის. ამგვარად, მთლიანი სივრცის დეკომპოზიცია 4x4 ბადეზე ქმნის 65 536 უჯრედს მეოთხე დონეზე.

2.9 ნახაზზე ნაჩვენებია უჯრედების ნუმერაციის მაგალითი და ბადეზე განლაგებული მრავალკუთხედები (ობიექტები მაგალითად, სახლები, ქუჩები და ა.შ.).



ნახ.2.9. სივრცით ბადეზე ობიექტების განთავსება

ბადის სიმკვრივე განისაზღვრება უჯრედების რაოდენობით, რაც მეტია, მით უფრო მკვრივია იგი. მაგალითად, ბადე 8x8 (64 უჯრედით) უფრო მკვრივია, ვიდრე ბადე 4x4 (16 უჯრედით).

სივრცითი ინდექსების შესაქმნელად გამოიყენება ინსტრუქცია CREATE SPATIAL INDEX, რომლის GRIDS წინადადებით შეიძლება განსხვავებული სიმკვრივეების მითითება სხვადასხვა დონეებზე (LOW - 4x4, MEDIUM – 8x8, HIGH. – 16x16).

გამოიყენება დამატებით ოფციები და წინადადებები:

- GEOMETRY_GRID – ეს წინადადება განსაზღვრავს გეომეტრიული ობიექტის ბადის ტესელაციის (tessellations) გამოყენებულ სქემას. ამ პროცესში ობიექტი თავსდება ბადისებრ იერარქიაში, უკავშირდება ბადის უჯრედებს, რომლებთანაც მას აქვს შეხება. სვეტს უნდა ჰქონდეს გეომეტრიული ტიპი;
- BOUNDING_BOX – ეს ოფცია განსაზღვრავს კომპლექტს ოთხი რიცხვითი მნიშვნელობით (მართკუთხედის ოთხი კოორდინატისთვის): Xmin და Ymin (ქვედა-მარცხენა კუთხე) და Xmax და Ymax (ზედა-მარჯვენა კუთხე). ეს პარამეტრი გამოიყენება მხოლოდ GEOMETRY_GRID წინადადებისთვის;
- GEOGRAPHY_GRID – ეს წინადადება განსაზღვრავს გეოგრაფიული ობიექტის ბადის ტესელაციის (tessellations) სქემას. სვეტს უნდა ჰქონდეს გეოგრაფიული ტიპი;

2.7 ლისტინგში ნაჩვენებია სივრცითი ინდექსის შექმნის მაგალითი beverage_markets ცხრილის shape სვეტისთვის.

სივრცითი ინდექსის შექმნა შეიძლება მახოლოდ მაშინ, თუ სივრცითი სვეტის მქონე ცხრილისთვის ცხადად არის განსაზღვრული პირველადი გასაღები. ამ მიზეზით ლისტინგის პირველ ALTER TABLE ინსტრუქციაში ეს შეზღუდვაა განსაზღვრული. შემდეგ ინსტრუქციაში CREATE SPATIAL INDEX იქმნება სივრცითი ინდექსი, გამოიყენება GEOMETRY_GRID.

```
USE sample;
GO
ALTER TABLE beverage_markets
    ADD CONSTRAINT prim_key PRIMARY KEY(id);
    GO
    CREATE SPATIAL INDEX i_spatial_shape
        ON beverage_markets(shape)
        USING GEOMETRY_GRID
         WITH (BOUNDING_BOX = (xmin=0, ymin=0, xmax=500, ymax=200),
         GRIDS = (LOW, LOW, MEDIUM, HIGH),
         PAD_INDEX = ON);
```

ლისტინგი 2.7

პარამეტრი BOUNDING BOX იძლევა საზღვრებს, რომელშიც მოთავსდება shape სვეტის ეგზემპლარი. GRIDS პარამეტრში მიეთითება ბადის სიმკვრივე ტესელაციის სქემის ყოველ დონეზე. პარამეტრი PAD_INDEX კავშირშია FILEFACTOR პარამეტრთან, რომელიც განსაზღვრავს პროცენტებში სივრცის თავისუფალ მოცულობას ინდექსის გვერდების საერთო მოცულობიდან. PAD_INDEX პარამეტრი კი მიუთითებს, რომ FILEFACTOR პარამეტრის მნიშვნელობა გამოიყენება როგორც ინდექსის გვერდებთან, ასევე ინდექსში მონაცემთა გვერდებთან.

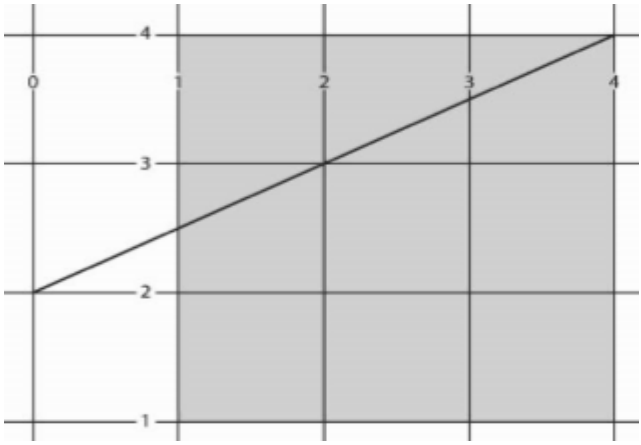
2.3.5. ინფორმაციის ასახვა სივრცითი მონაცემების შესახებ

SQL Server Management Studio 2012 ვერსიიდან ზევით გაფართოვდა სივრცითი მონაცემების ასახვის შესაძლებლობები გრაფიკულ ფორმატში. განვიხილოთ ეს საკითხები GEOMETRY და GEOGRAPHY მონაცემების ტიპებზე, შესაბამისად 2.8 და 2.9 ლისტინგების საფუძველზე.

```
USE sample;
DECLARE @rectangle1 GEOMETRY = 'POLYGON((1 1, 1 4, 4 4, 4 1, 1 1))';
DECLARE @line GEOMETRY = 'LINESTRING (0 2, 4 4)';
SELECT @rectangle1
UNION ALL
SELECT @line
```

ლისტინგი 2.8

ამ მოთხოვნის შესრულების შედეგების მისაღებად პანელზე ვირჩევთ Spatial Results ჩანართს (Results-ის მარჯვნივ). შედეგად ვღებულობთ 2.10 ნახაზზე მიღებულ მართკუთხედს (ცვლადით @rectangle1) და წრფეს (ცვლადით @line).



ნახ.2.10. შედეგის გრფაიკული წარმოდგენა (2.8 ლისტინგის მიხედვით)

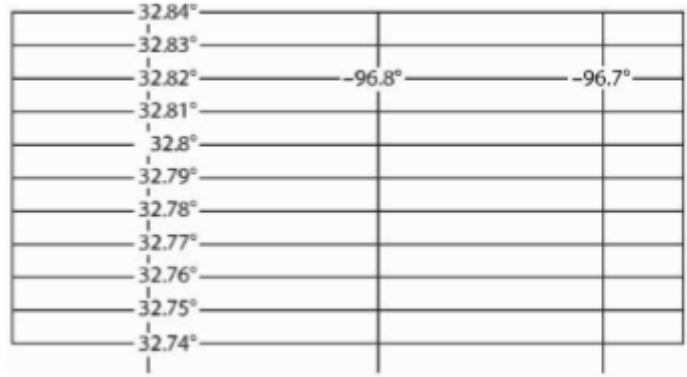
GEOMETRY ტიპის რამდენიმე ობიექტის ერთდროულად ასახვის მიზნით (ერთი ცხრილის რამდენიმე სტრიქონი), 2.8 ლისტინგში გამოყენებულია ორი SELECT ინსტრუქცია, რომლებიც ერთიანდება UNION ALL წინადადებით.

2.9 ლისტინგში ნაჩვენებია მოთხოვნა GEOGRAPHY მონაცემთა ტიპის გრაფიკული ასახვის სადემონსტრაციოდ.

```
USE AdventureWorks;
SELECT SpatialLocation, City
   FROM Person.Address
   WHERE City = 'Dallas';
```

ლისტინგი 2.9

აქაც ვიყენებთ შედეგების პანელზე Spatial Results ჩანართს და მივიღებთ 2.11 ნახაზს.



ნახ.2.11. შედეგის გრფაიკული წარმოდგენა (2.9 ლისტინგის მიხედვით)

2.3.6. სიახლეები სივრცით მონაცემებთან სამუშაოდ

SQL Server 2012 და უფრო ახალ ვერსიებში გაუმჯობესდა სივრცით მონაცემებთან მუშაობის შესაძლებლობები. კერძოდ, შემოტანილია:

- წრის რკალის ახალი ქვეტიპები;
- ახალი სივრცითი ინდექსები;

- ახალი სისტემური შენახვადი პროცედურები, დაკავშირებული სივრცით მონაცემებთან.

განვიხილოთ თითოეული მათგანი:

➢ **წრის რკალის ახალი ქვეტიპები.** ეფუძნება ANSI SQL/MM სტანდარტს. წრის რკალი ზოგადად არის მრუდის ჩაკეტილი სეგმენტი ორგანზომილებიან სიბრტყეზე. ამგვარად, რკალი წრის სეგმენტია, რომელიც მოიცემა დამოუკიდებლად ან წრფის სეგმენტთან ერთად. ამასთანავე, იგი შეიძლება გამოყენებულ იქნას როგორც საფუძველი ახალი ტიპის მრავალკუთხედისა, რომელიც შეიცავს ერთ ან მეტ რკალურ კომპონენტს.

წრის რკალები მხარდაჭერილია GEOMETRY და GEOGRAPHY მონაცემთა ტიპებით და განისაზღვრება WKT, WKB და GML ფორმატებში.

არსებობს წრის რკალების სამი ახალი ტიპი:

- **CircularString;**

2.6 ნახაზზე GEOMETRY ტიპის იერარქიიდან ჩანს, რომ CircularString ტიპი არის Curve-ს ქვეტიპი. ამიტომაც CircularString ტიპის ობიექტები მრუდის საბაზო ქვეტიპია. CircularString ტიპის ობიექტის განსაზღვრისათვის საჭიროა მინიმუმ სამი წერტილის მოცემა. პირველი მიუთითებს რკალის დასაწყისს, მეორე - მის ბოლოს, ხოლო მესამე უნდა იყოს რკალის რომელიმე ადგილზე. CircularString ტიპის ეგზემპლარები შეიძლება გაერთიანდეს, სადაც წინა მრუდის ბოლო წერტილი ხდება მომდევნო მრუდის საწყისი წერტილი. 2.10 ლისტინგში ნაჩვენებია CircularString ტიპის ობიექტის განსაზღვრა @g ცვლადის გამოყენებით.

```
DECLARE @g GEOMETRY;
SET @g =  GEOMETRY::STGeomFromText
      (‘CIRCULARSTRING(0 -12.5, 0 0, 0 12.5)’, 0);
```

ლისტინგი 2.10

- **CompoundCurve;**

განსაზღვრავს ახალ შედგენილ მრუდებს, რომლებიც შედგება ან მხოლოდ CircularString ტიპის ეგზემპლარებისგან ან CircularString და LineString ეგზემპლარებისგან. ყოველი წინა კომპონენტის ბოლო წერტილი უკავშირდება მომდევნო საწყის წერტილს. 2.11 ლისტინგში ნაჩვენებია შედგენილი მრუდის შექმნა CircularString ტიპის რამდენიმე სხვადასხვა ობიექტით.

```
DECLARE @g GEOGRAPHY;
SET @g = GEOGRAPHY::STGeomFromText('
COMPOUNDCURVE(CIRCULARSTRING(0 -23.43778, 0 0, 0 23.43778),
CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),
CIRCULARSTRING(-90 23.43778, -90 0, -90 -23.43778),
CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778) )' ,4326);
```

ლისტინგი 2.11

აქ იქმნება GEOGRAPHY მონაცემთა ტიპის ეგზემპლარი, რომელიც მიენიჭება @g ცვლადს. ეს ცვლადი შედგება CompoundCurve ტიპის ეგზემპლარისგან, რომელიც თავის მხრივ შედგება CircularString ტიპის ეგზემპლარებისგან. საყურადღებოა STGeomFromText() მეთოდის ბოლო არგუმენტის მნიშვნელობა 4326. ესაა SRID (Spatial reference ID) იდენტიფიკატორი GEOGRAPHY მონაცემთა ტიპისთვის და შეესაბამება WGS_82 კოორდინატთა სივრცით სისტემას [11].

- **CurvePolygon.**

CurvePolygon ტიპის ობიექტი შედგება LineString და CircularString ტიპის ობიექტებისგან, აგრეთვე CompoundCurves ტიპის ობიექტებისგან. 2.6 ნახაზიდან ჩანს, რომ CurvePolygon არის უშუალო ქვეტიპი Surfface ტიპისა და სუპერტიპი Polygon ტიპისა. ასეთი წრის

შიგნით CurvePolygon ობიექტის მომდევნო კომპონენტის პირველი წერტილი ემთხვევა მომდევნო კომპონენტის ბოლო წერტილს.

- **ახალი სივრცითი ინდექსები**

SQL Server 2012 -ში GEOMETRY и GEOGRAPHY ტიპის მონაცემებისთვის დამატებულია ახალი სივრცითი ინდექსი auto_grid_index. მისი ფუნქციონალობა ორივესთვის მსგავსია, ამიტომ მხოლოდ ერთს განვიხილავთ.

ამ ინდექსში გამოიყენება ტესელაციის რვა დონე სხვადასხვა ზომის ობიექტების უკეთესი აპროქსიმაციის მიზნით.

2.12 ლისტინგში ნაჩვენებია ავტომატური ინდექსის შექმნა გეომეტრიული ობიექტისთვის.

```
CREATE SPATIAL INDEX auto_grid_index
    ON beverage_markets(shape)
    USING GEOMETRY_AUTO_GRID
        WITH (BOUNDING_BOX = (xmin=0, ymin=0, xmax=500, ymax=200),
        CELLS_PER_OBJECT = 32, DATA_COMPRESSION = page);
```

ლისტინგი 2.12

2.3.7. ახალი სისტემური შენახვადი პროცედურები სივრცითი მონაცემებისთვის

SQL Server 2012 -ში სივრცითი ტიპის მონაცემებისათვის დამატებულია ახალი სისტემური შენახვადი პროცედურები:

- sp_help_spatial_geometry_index;
- sp_help_spatial_geography_index;

რომელთა გამოყენების სინტაქსიც ორივესი მსგავსია.

sp_help_spatial_geometry_index სისტემური შენახვადი პროცედურა აბრუნებს უკან გეომეტრიული ობიექტის სივრცითი ინდექსის მითითებული თვისებების ერთობლიობის დასახელებებს და მნიშვნელობებს. შედეგები აისახება ცხრილურ ფორმატში. აგრეთვე შესაძლებელია ძირითად თვისებათა ერთობლიობის ან ინდექსის ყველა თვისების მოთხოვნა. 2.13 ლისტინგში მოცემულია ამ სისტემური შენახვადი პროცედურის კოდი.

```
DECLARE @query geometry
 ='POLYGON((-90.0 -180.0, -90.0 180.0, 90.0 180.0, 90.0 -180.0, -90.0 -180.0))' ;
EXEC sp_help_spatial_geometry_index 'beverage_markets',
                                    'auto_grid_index', 0, @query;
```

ლისტინგი 2.12

ამ მაგალითში sp_help_spatial_geometry_index სისტემური პროცედურა ასახავს auto_grid_index სივრცითი ინდექსის თვიებებს, რომელიც შეიქმნა 2.12 ლისტინგით.

III თავი: პროგრამული აპლიკაციების აგება ჰიბრიდული ტექნოლოგიებით

3.1. WPF-აპლიკაცია მომხმარებლის სახელის და პაროლის კონტროლისთვის

განიხლება პროგრამული აპლიკაციის უსაფრთხოების (ავტორიზაციის) მიზნით მომხმარებლის სახელისა და პაროლის კონტროლის ამოცანის გადაწყვეტა Visual Studio .NET 2013 სამუშო გარემოში WPF-ტექნოლოგიის გამოყენებით.

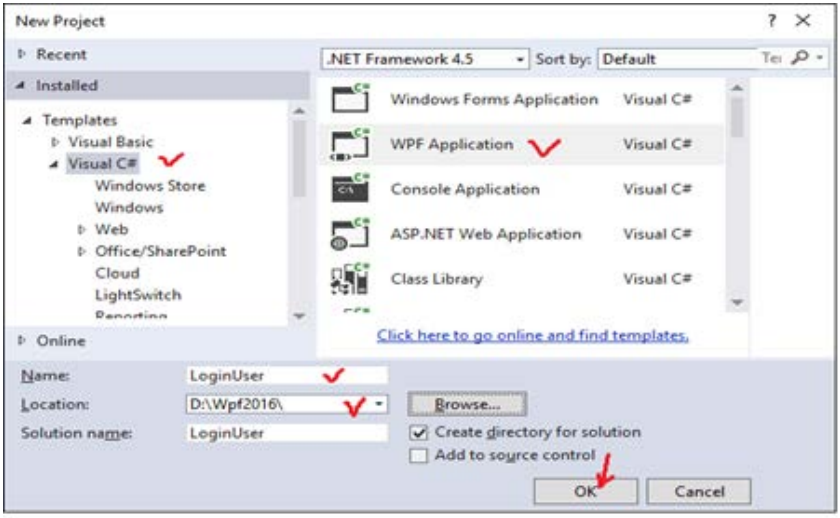
საჭიროა შემდეგი ფუნქციური ეტაპების შესრულება:

1. ახალი WPF-პროექტის შექმნა;
2. მომხმარებლის ინტერფეისის დიზაინის განსაზღვრა;
3. ინტერფეისის დიზაინის აგება ToolBox-ის კომპონენტების საფუძველზე;
4. ინტერფეისის კომპონენტების თვისებების და მათი მნიშვნელობების შერჩევა Properties-ში;
5. XAML ფაილის დაზუსტება ინტერაქტიულ რეჟიმში;
6. აპლიკაციის ლოგიკის შემუშავება ანუ მისი კომპონენტებისთვის C# კოდის დაპროექტება;
7. C# კოდის გამართვა, ტესტირება;
8. აპლიკაციის მუშა ფაილის ამუშავება და შედეგების ანალიზი.

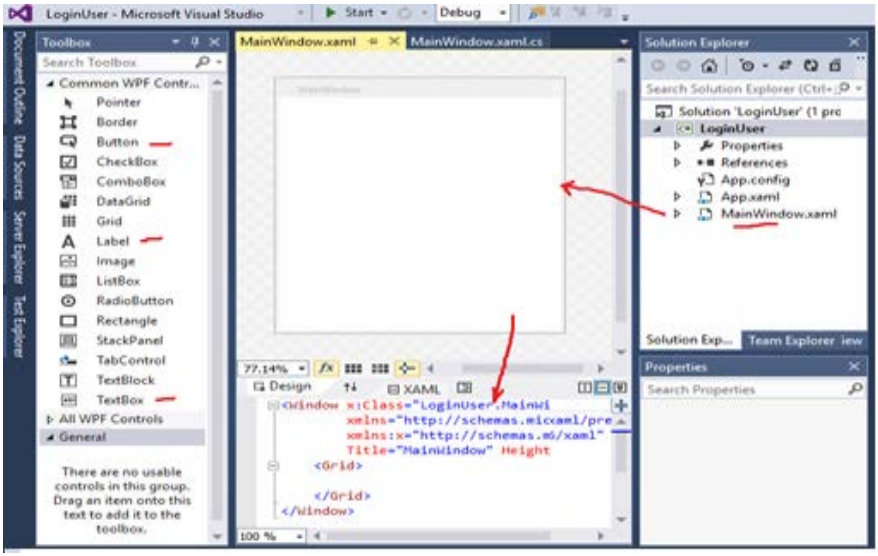
შევქმნათ ახალი პროექტი Wpf Application სახელით LoginUser და შევინახოთ ახლადშექმნილ ფოლდერში Wpf2016 (ნახ.3.1).

მიიღება VisualStudio.NET სამუშაო გარემო SolutionExplorer-ით LoginUser (ნახ.3.2). აქ ჩანს ინსტრუმენტების პანელი (მარცხნივ), რომელსაც გამოვიყენებთ აპლიკაციის დიზაინისთვის, შუა ადგილას WPF-ის სამუსა ფორმა, მის ქვემოთ XAML-ის ფაილი (ფორმის შესაბამისი - თავიდან კომპონენტების გარეშე).

მარჯვნივ ჩანს ასევე Properties - თვისებების ფანჯარა.



ნახ.3.1. პროექტის შექმნა LoginUser სახელით



ნახ.3.2. პროექტი შეიქმნა: იწყება დიზაინერის მუშაობის ეტაპი

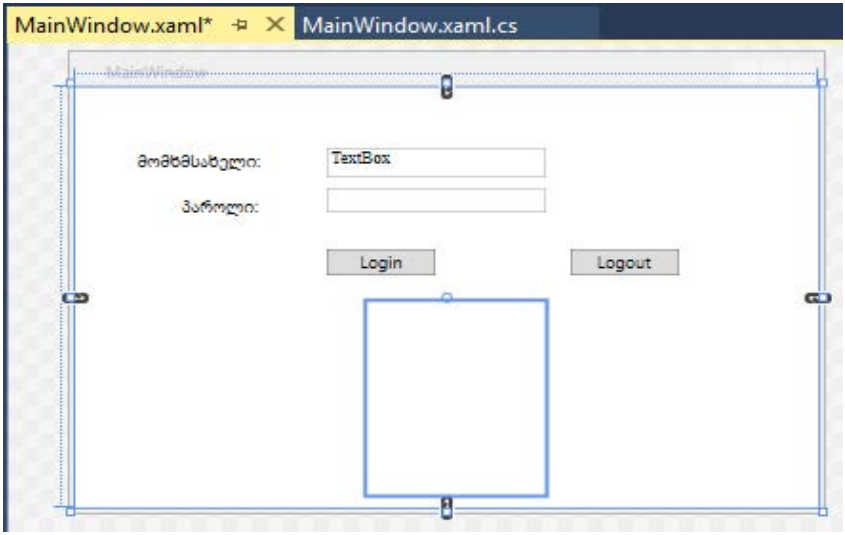
ინტერფეისის ასაგებად ToolBox-იდან გადმოვიტანოთ ორი label (წარწერებისთვის მომხმარებლის სახელი და პაროლი), ერთი TextBox - მომხმარებლის სახელის შესატანად, ერთი PasswordBox - პაროლის შესატანად.

ფორმაზე დავდოთ აგრეთვე ორი Button, ერთი Login და მეორე Logout.

შედეგების საილუსტრაციოდ დავამატოთ ფორმაზე ერთი Image, რომელშიც სწორი პასუხის შემთხვევაში გამოვა ამ მომხმარებლის ფოტო. ამასთანავე გაქრება Login ღილაკი და გამოჩნდება Logout.

თუ სახელი ან/და პაროლი შეცდომითაა, მაშინ გამოვა შეტყობინება არასწორი პასუხის შესახებ MessageBox-ში.

დიზაინის ფორმა ნაჩვენებია 3.3 ნახაზზე.

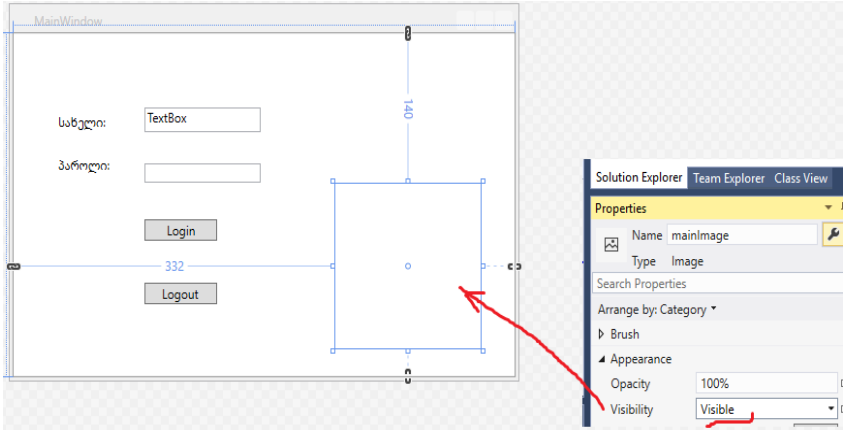


ნახ.3.3. პროექტის დიზაინი ჩვენი პირობებით

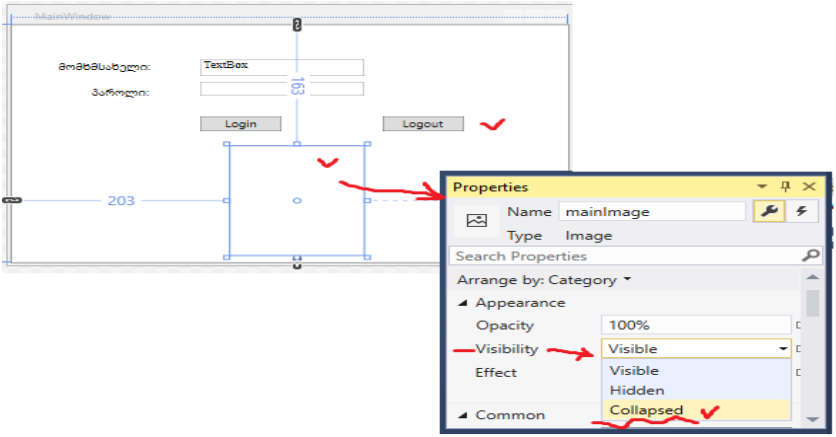
ფორმაზე ელემენტების მდებარეობა შეიძლება შევცვალოთ ჩვენი (ან დამკვეთის) სურვილისამებრ.

Properties-ში Image და Logout Button-ისთვის ხილვადობის თვისების მნიშვნელობა დავაყენოთ (ნახ.3.4):

Visibility="Collapsed" .

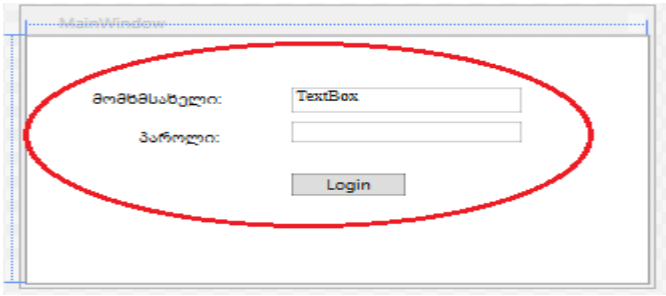


ნახ.3.4-ა. ხილვადობის თვისების მნიშნელობა Properties-ში (თავიდან Visible-შია)



ნახ.3.4-ა. ხილვადობის თვისების მნიშნელობის შეცვლა Collapsed-ით

ინტერფეისის საბოლოო შედეგი მოცემულია 2.5 ნახაზზე.



ნახ.3.5. ფორმაზე Collapsed-ის არჩევამ დამალა სურათი და Logout.

ახლა შეიძლება გავაანალიზოთ დიზაინით აგებული ფორმის შესაბამისი XAML ფაილი. Solution Explorer-ში მას შეესაბამება MainWindow.xaml ფაილი.

```
<Window x:Class="LoginUse.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006
        /xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
 <Grid>
   <Label Content="მომხმ.სახელი:"
        HorizontalAlignment="Left" Margin="38,42,0,0"
        VerticalAlignment="Top" Width="98"/>
  <Label Content="პაროლი:" HorizontalAlignment="Left"
        Margin="69,77,0,0"
        VerticalAlignment="Top" Width="67"/>
  <TextBox x:Name="UserName" HorizontalAlignment="Left"
        Height="23" Margin="175,46,0,0"
        TextWrapping="Wrap" Text="TextBox"
        VerticalAlignment="Top" Width="152"
                  FontFamily="Times New Roman"/>
```

```
      <PasswordBox x:Name="PasswordBox"
           HorizontalAlignment="Left" Margin="175,77,0,0"
           VerticalAlignment="Top" Width="152"/>
      <Button x:Name="LoginBTN" Content="Login"
           HorizontalAlignment="Left" Margin="175,123,0,0"
           VerticalAlignment="Top" Width="75"/>
      <Button x:Name="LogoutBTN" Content="Logout"
           HorizontalAlignment="Left" Margin="345,123,0,0"
           VerticalAlignment="Top" Width="75"
                          Visibility="Collapsed"/>
      <Image x:Name="mainImage" HorizontalAlignment="Left"
           Height="146" Margin="203,163,0,0"
           VerticalAlignment="Top" Width="124"
                          Visibility="Collapsed"/>
    </Grid>
</Window>
```

ამგვარად, ინტერფეისის შესაბამისი ფორმა აგებულია წარმატებით. ახლა უნდა დავწეროთ ფორმაზე მოთავსებული Login-ღილაკის მუშაობის ლოგიკა, ანუ C# კოდი.

Double click ღილაკზე Login და ჩავწეროთ კოდი:

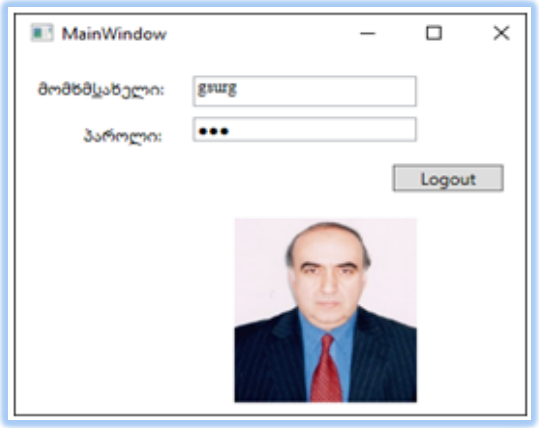
```
// ---- ლისტინგი 3.1 ----- Login -------
private void LoginBTN_Click(object sender, RoutedEventArgs e)
{
  if (!usernameTB.Text.Equals(“”) &&
                    !PasswordBoxPB.Password.Equals(“”))
    {
      if (usernameTB.Text.Equals(“1″) &&
                    PasswordBoxPB.Password.Equals(“1″))
        {
          mainImage.Visibility = Visibility.Visible;
          LoginBTN.Visibility = Visibility.Collapsed;
          logoutBTN.Visibility = Visibility.Visible;    }
```

```
        else
            MessageBox.Show("Wrong Password");
    }
  else
     MessageBox.Show("Wrong Info");
}
```

ავამუშავოთ პროგარმა და შევიტანოთ თავიდან რაიმე არასწორი პაროლი. შედეგად უნდა მივიღოთ 3.6 ნახაზის შესაბამისი ფორმა.

ნახ.3.6. არასწორი პაროლის შეტანის შედეგი

სწორი სახელის და პაროლის შეტანის შემთხვევაში მივიღებთ 3.7 ნახაზის შესაბამის სურათს.



ნახ.3.7. სწორი პაროლის შეტანის შედეგი

ახლა საჭიროა Logout ღილაკის ფუნქციის C#-კოდის აგება. ამ ღილაკზე დავკლიკოთ და შევიტანოთ ასეთი ტექსტი (ლისტინგი 3.2):

```
// ----- ლისტინგი 3.2 ---- Logout ------
private void logoutBTN_Click(object sender, RoutedEventArgs e)
 {
   mainImage.Visibility = Visibility.Collapsed;LoginBTN.Visibility =
      Visibility.Visible;logoutBTN.Visibility = Visibility.Collapsed;
 }
```

საბოლოო C#-პროგრამის კოდი მოცემულია 3.3 ლისტინგში.

```
// ---- ლისტინგი 3.3 --------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace test2Log
{
 public partial class MainWindow : Window
  {
     public MainWindow()
     {
        InitializeComponent();
     }
```

```
    private void LoginBTN_Click(object sender, RoutedEventArgs e)
     {
       if (!UserName.Text.Equals("") && !PasswordBox.Password.Equals(""))
        {
          if (UserName.Text.Equals("gsurg") &&
                                PasswordBox.Password.Equals("123"))
           {
              mainImage.Visibility = Visibility.Visible;
              LoginBTN.Visibility = Visibility.Collapsed;
              LogoutBTN.Visibility = Visibility.Visible;
           }
           else
            MessageBox.Show("მომხმ_სახელი ან პაროლი არასწორია !");
        }
        else
           MessageBox.Show("ინფორმაცია არაა !");
     }
   private void LogoutBTN_Click(object sender, RoutedEventArgs e)
     {
        mainImage.Visibility = Visibility.Collapsed;
        LoginBTN.Visibility = Visibility.Visible;
        LogoutBTN.Visibility = Visibility.Collapsed;
     }
   }
}
```

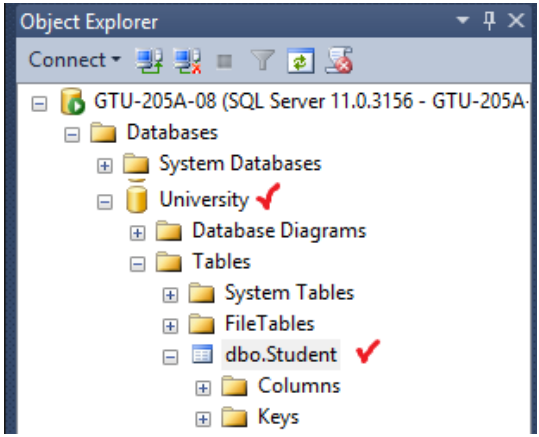
3.2. WPF-აპლიკაცია მონაცემთა ბაზასთან სამუშაოდ SQL Server 2012 -ის საფუძველზე

განვიხილოთ wpf პროგარმული აპლიკაციის პროექტის აგება მონაცემთა ბაზასთან ინტერაქტიულ რეჟიმში სამუშაოდ. კერძოდ, უნდა ავაგოთ მომხმარებლის ინტერფეისი Visual Studio.NET-ის WPF ტექნოლოგიით, რომლითაც შესაძლებელი იქნება SQL Server მონაცემთა ბაზასთან დაკავშირება და მასში მონაცემთა მენეჯმენტის (მონაცემების ჩამატება, მოდიფიკაცია, წაშლა) განხორციელება. მონაცემთა ბაზის ფაილი წინასწარ მზად უნდა იყოს გამოსაყენებლად.

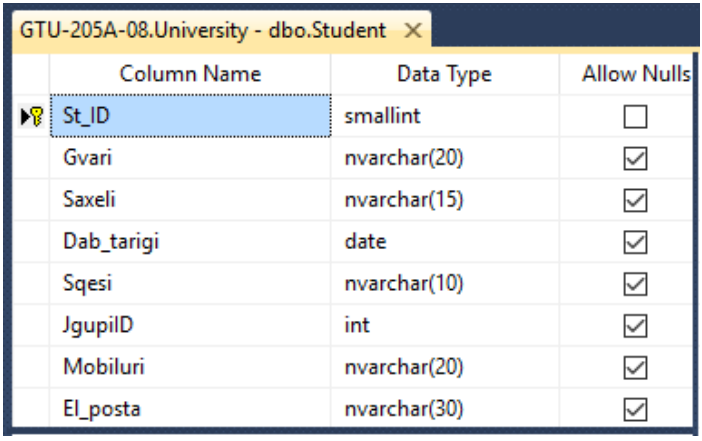
საჭიროა შემდეგი ფუნქციური ეტაპების შესრულება:

1. SQL Server Management Studio-ში მონაცემთა ბაზის და შესაბამისი ცხრილების აგება;
2. მომხმარებლის ინტერფეისის აგება ინსტრუმენტების პანელის სტანდარტული ელემენტებით Visual Studio.NET-ის გარემოში WPF პროექტით;
3. ინტერფეისის შესაბამისი xaml ფაილის გამართვა;
4. ინტერფეისის დაკავშირება ბაზასთან (Server Explorer);
5. ინტერფეისის ღილაკების (Add, Update, Delete, Clear) მეთოდების დაპროგრამება;
6. პროგრამის გამართვა და ექსპერიმენტის ჩატარება კონკრეტულ მონაცემებთან ბაზაში:
   - ჩასამატებლად;
   - შესაცვლელად;
   - წასაშლელად;
   - შესატანი ტექსტბოქსების გასაწმენდად.
7. პროგრამული აპლიკაციის ფუნქციონირების ტესტირება.

Ms SQL Server Management Studio 2012 გამოყენებით შევქმნათ მონაცემთა ახალი ბაზა - „უნივერსიტეტი“ და ცხრილი - „სტუდენტი“, რომლების 3.8 და 3.9 ნახაზებზეა მოცემული.



ნახ.3.8. მმონაცემთა ბაზის შექმნა - „უნივერსიტეტი“



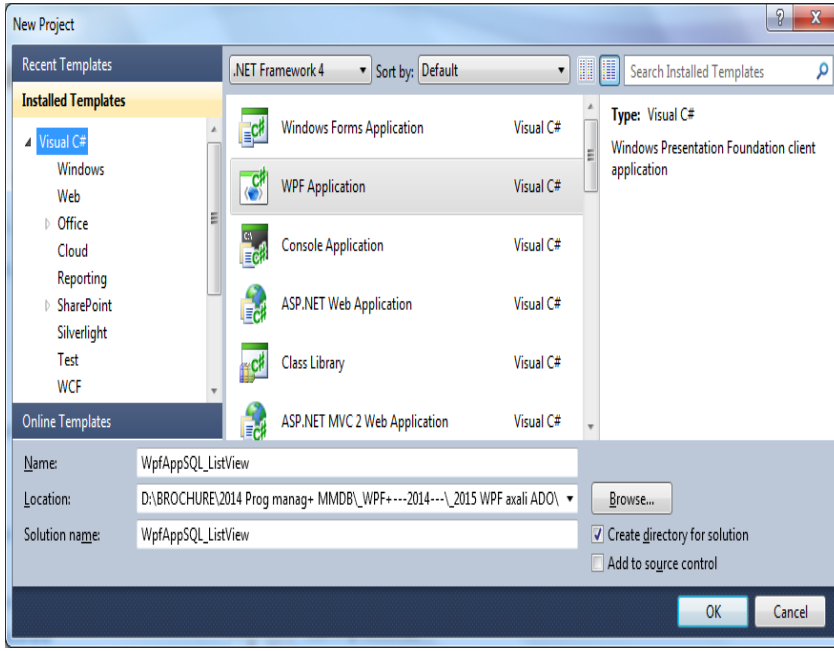
ნახ.3.9-ა. მბ-ის Student ცხრილის ატრიბუტების ფორმირება

GTU-205A-08.University - dbo.Student

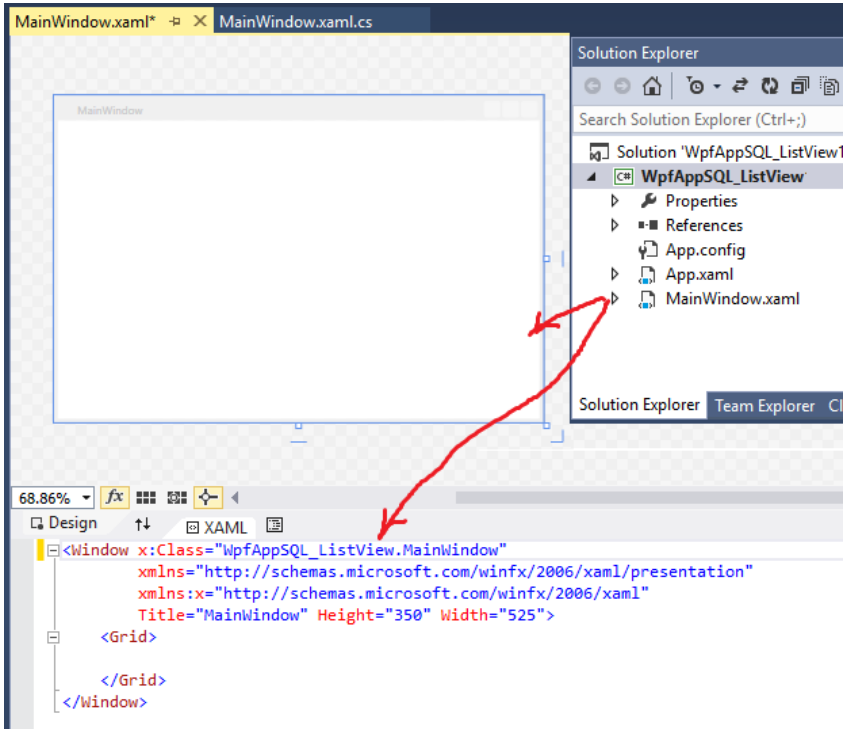
St_ID	Gvari	Saxeli	Dab_tariqi	Sqesi	JqupilD	Mobiluri	El_posta
1	აბაშიძე	აკაკი	1995-05-17	მამრობითი	108550	577102030	abashidze.ak@gmail.com
2	ბურდული	ბუდუ	1997-01-31	მამრობითი	108550	599707070	burdubu@yahoo.com
3	ბახტაძე	მერაბ	1995-01-01	მამრობითი	108551	591111222	bakhtadze.m@gmail.com
4	გაგუა	ნინო	1998-08-20	მდედრობითი	108551	577223355	gaguani@yahoo.com
5	კაკუბავა	ნინო	1998-05-09	მდედრობითი	108555	595777555	kakunino@gmail.com

ნახ.3.9-ბ. მბ-ის Student ცხრლის ჩანაწერები

მონაცემთა ბაზა შექმნილია. ახლა გადავიდეთ Ms_Visual Studio.NET 2013 სამუშაო გარემოში. შევქმნათ ახალი პროექტი სახელით WpfAppSQL_ListView (ნახ.3.10).



ნახ.3.10. ახალი პროექტის შექმნა



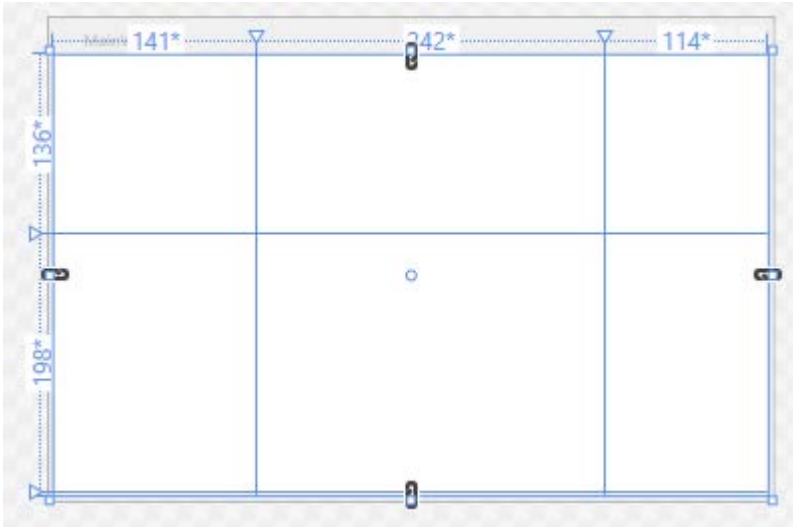
ნახ.3.11. პროექტის საწყისი მდგომარეობა XAML ფაილით

ჩავამატოთ ასეთი კოდი ცხრილის სვეტების და სტრიქონების ზომების განსაზღვრის მიზნით:

```
<Grid.ColumnDefinitions>
        <ColumnDefinition Width="141*" />
        <ColumnDefinition Width="242*" />
        <ColumnDefinition Width="114*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
        <RowDefinition Height="136*" />
        <RowDefinition Height="198*" />
```

```
            <RowDefinition Height="2*" />
        </Grid.RowDefinitions>
```

მივიღებთ:



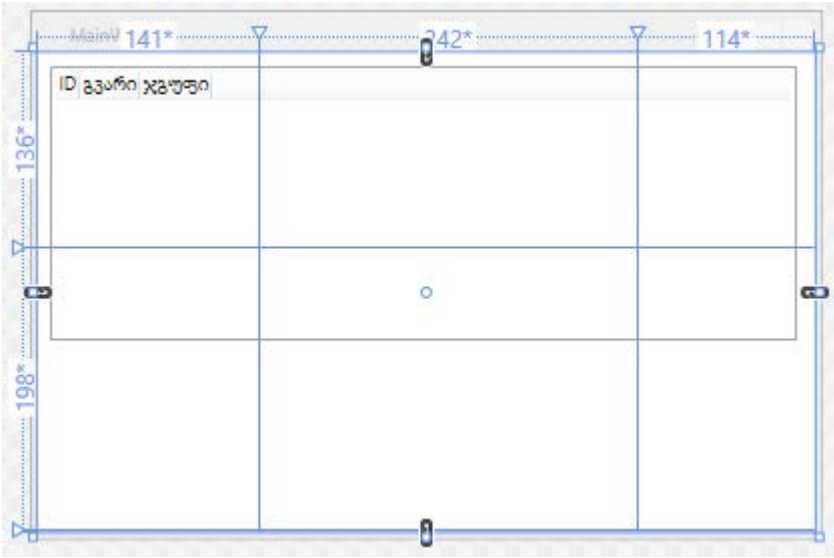
ნახ.3.12. სვეტების და სტრიქონების ზომების განსაზღვრა

ჩავამატოთ

```
<ListView Margin="8,9,12,125" Name="listView1" ItemsSource="{Binding}"
          MinWidth="250" MinHeight="100" Grid.ColumnSpan="4"
                Grid.RowSpan="2">
  <ListView.View>
    <GridView>
      <GridViewColumn Header="ID" DisplayMemberBinding=
                  "{Binding Path=St_ID}"></GridViewColumn>
      <GridViewColumn Header="გვარი" DisplayMemberBinding=
                  "{Binding Path=Gvari}"></GridViewColumn>
```

```
    <GridViewColumn Header="ჯგუფი" DisplayMemberBinding=
                "{Binding Path=JgupiID}"></GridViewColumn>
  </GridView>
 </ListView.View>

</ListView>
```

მივიღებთ:



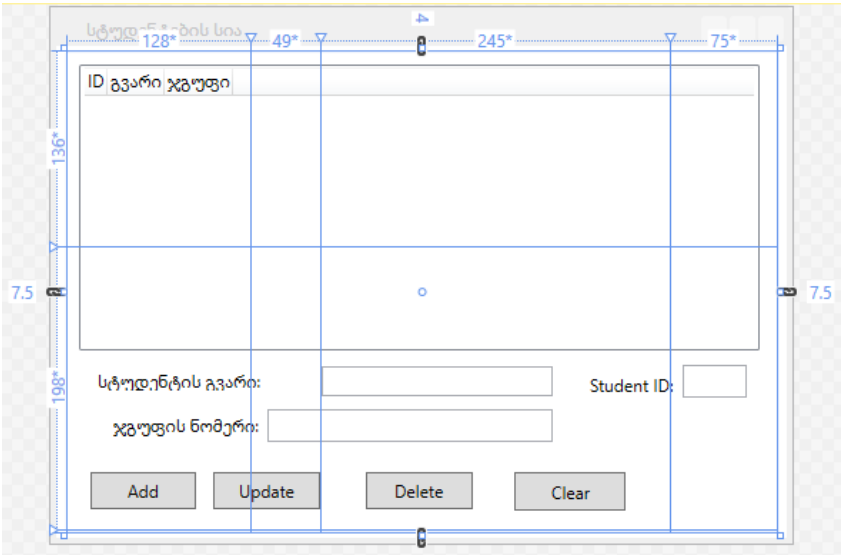
ნახ.3.13. ListView სვეტების დასახელებით

- დავალაგოთ ფორმაზე 2 Label, 2-TextBox და 4 Button (იხ. კოდი):

```
<TextBox Margin="1,84,82,93" Name="textBox1" DataContext=
       "{Binding ElementName=listView1,Path=SelectedItem}"
          Text="{Binding Path=Username}" Grid.Row="1"
          Grid.Column="2" />
<TextBox Height="23" Margin="0,0,82,61" Name="textBox2"
        VerticalAlignment="Bottom" DataContext=
       "{Binding ElementName=listView1,Path=SelectedItem}"
         Text="{Binding Path=Password}" Grid.ColumnSpan="2"
```

```
          Grid.Row="1" HorizontalAlignment="Right"
          Width="200" Grid.Column="1" />
<Label Margin="16,81,0,92" Name="label1" Grid.Row="1"
    Content="სტუდენტის გვარი:" Grid.ColumnSpan="2"></Label>
<Label Height="29" Margin="27,0,0,57" Name="label2"
    VerticalAlignment="Bottom" Grid.Row="1"
    Content="ჯგუფის ნომერი:" Grid.ColumnSpan="2"></Label>
<Button Height="26" Margin="16,0,0,14" Name="btnAdd"
    VerticalAlignment="Bottom" Click="btnAdd_Click"
    Grid.Row="1" HorizontalAlignment="Left"
    Width="74">Add</Button>
<Button Height="26" Margin="102,0,0,14" Name="btnUpdate"
    VerticalAlignment="Bottom" Click="btnUpdate_Click"
    HorizontalAlignment="Left" Width="75"
    Grid.ColumnSpan="2" Grid.Row="1">Update</Button>
<Button Height="26" Margin="32,0,0,14" Name="btnDelete"
    VerticalAlignment="Bottom" Click="btnDelete_Click"
    Grid.Column="2" HorizontalAlignment="Left" Width="75"
    Grid.Row="1">Delete</Button>
<Button Height="27.5" Margin="136,0,31,13" Name="btnClear"
    VerticalAlignment="Bottom" Click="btnClear_Click"
    Grid.Column="2" Grid.Row="1">Clear</Button>
<Label Content="Student ID:" Grid.Column="2" Grid.Row="1"
    Height="26" HorizontalAlignment="Left"
    Margin="184,84,0,0" Name="label3"
    VerticalAlignment="Top" Width="70"
    Grid.ColumnSpan="2" />
<TextBox Grid.Column="3" Grid.Row="1" Height="23"
    HorizontalAlignment="Left" Margin="9,83,0,0"
    Name="textBox3" VerticalAlignment="Top" Width="45" />
```

მიიღება:



ნახ.3.14. ინტერფეისი მონაცემთა ბაზის ცხრილის მენეჯმენტისთვის

საჭიროა ღილაკების ფუნქციების დაწერა და მონაცემთა SQL Server ბაზასთან კავშირის დამყარება მონაცემების დასამატებლად (Add), შესასწორებლად (Update), წასაშლელად (Delete) და შესატანი ტექსბოქსების გასასუფთავებლად (Clear).

ამგვარად, XAML ფაილს მთლიანობაში ექნება შემდეგი სახე:

```
<Window x:Class="WpfAppSQL_ListView.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presenta
tion"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title ="სტუდენტების სია" Height="375" Width="520"
Loaded="Window_Loaded" Background="White">
        <Grid Height="336" Width="497" Background="White" >
        <Grid.ColumnDefinitions>
```

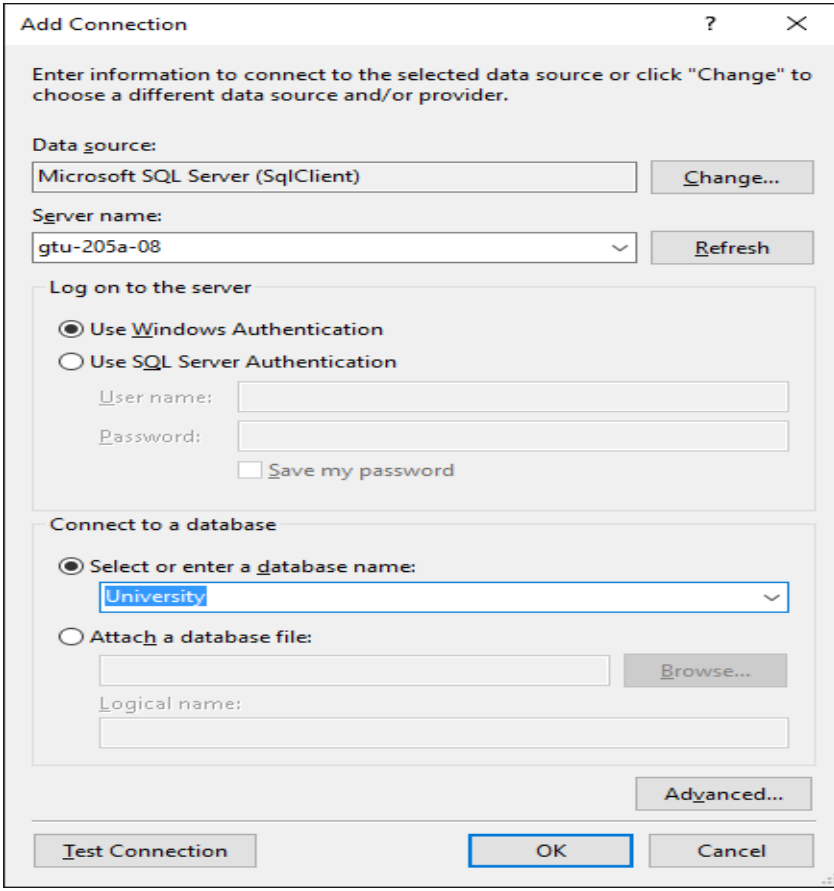
```
            <ColumnDefinition Width="128*" />
            <ColumnDefinition Width="49*" />
            <ColumnDefinition Width="245*" />
            <ColumnDefinition Width="75*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="136*" />
            <RowDefinition Height="198*" />
            <RowDefinition Height="2*" />
        </Grid.RowDefinitions>
        <ListView Margin="8,9,12,125" Name="listView1"
ItemsSource="{Binding}" MinWidth="250" MinHeight="100"
Grid.ColumnSpan="4" Grid.RowSpan="2">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="ID"
DisplayMemberBinding="{Binding
Path=St_ID}"></GridViewColumn>
                    <GridViewColumn Header="გვარი"
DisplayMemberBinding="{Binding
Path=Gvari}"></GridViewColumn>
                    <GridViewColumn Header="ჯგუფი"
DisplayMemberBinding="{Binding
Path=JgupiID}"></GridViewColumn>

                </GridView>
            </ListView.View>
        </ListView>
        <TextBox Margin="1,84,82,93" Name="textBox1"
DataContext="{Binding
ElementName=listView1,Path=SelectedItem}" Text="{Binding
Path=Username}" Grid.Row="1" Grid.Column="2" />
        <TextBox Height="23" Margin="0,0,82,61"
Name="textBox2" VerticalAlignment="Bottom"
DataContext="{Binding
```

```
ElementName=listView1,Path=SelectedItem}" Text="{Binding
Path=Password}" Grid.ColumnSpan="2" Grid.Row="1"
HorizontalAlignment="Right" Width="200" Grid.Column="1" />
        <Label Margin="16,81,0,92" Name="label1"
Grid.Row="1" Content="სტუდენტის გვარი:"
Grid.ColumnSpan="2"></Label>
        <Label Height="29" Margin="27,0,0,57" Name="label2"
VerticalAlignment="Bottom" Grid.Row="1" Content="ჯგუფის
ნომერი:" Grid.ColumnSpan="2"></Label>
        <Button Height="26" Margin="16,0,0,14" Name="btnAdd"
VerticalAlignment="Bottom" Click="btnAdd_Click" Grid.Row="1"
HorizontalAlignment="Left" Width="74">Add</Button>
        <Button Height="26" Margin="102,0,0,14"
Name="btnUpdate" VerticalAlignment="Bottom"
Click="btnUpdate_Click" HorizontalAlignment="Left"
Width="75" Grid.ColumnSpan="2" Grid.Row="1">Update</Button>
        <Button Height="26" Margin="32,0,0,14"
Name="btnDelete" VerticalAlignment="Bottom"
Click="btnDelete_Click" Grid.Column="2"
HorizontalAlignment="Left" Width="75"
Grid.Row="1">Delete</Button>
        <Button Height="27.5" Margin="136,0,31,13"
Name="btnClear" VerticalAlignment="Bottom"
Click="btnClear_Click" Grid.Column="2"
Grid.Row="1">Clear</Button>
        <Label Content="Student ID:" Grid.Column="2"
Grid.Row="1" Height="26" HorizontalAlignment="Left"
Margin="184,84,0,0" Name="label3" VerticalAlignment="Top"
Width="70" Grid.ColumnSpan="2" />
        <TextBox Grid.Column="3" Grid.Row="1" Height="23"
HorizontalAlignment="Left" Margin="9,83,0,0" Name="textBox3"
VerticalAlignment="Top" Width="45" />
    </Grid>
</Window>
```

გადავიდეთ C# კოდის აგებაზე. ჯერ ჩავამატოთ:

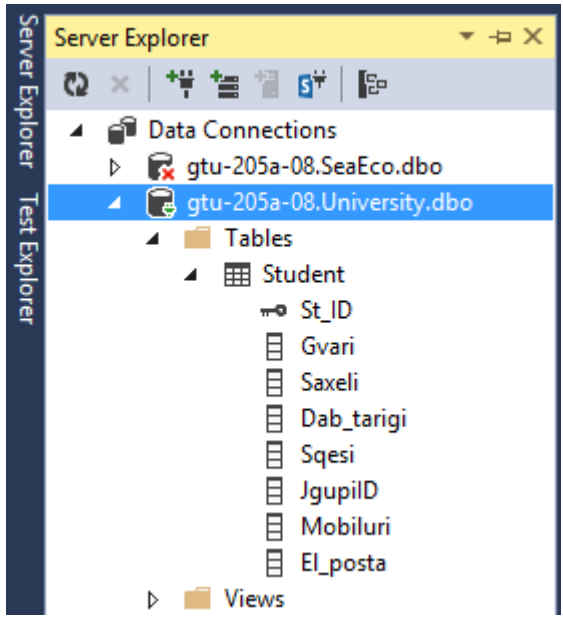
```
using System.Data.SqlClient;
using System.Data;
```

გავააქტიუროთ მთავარი მენიუს View და Server Explorer. ავირჩიოთ სერვერი და ბაზა:

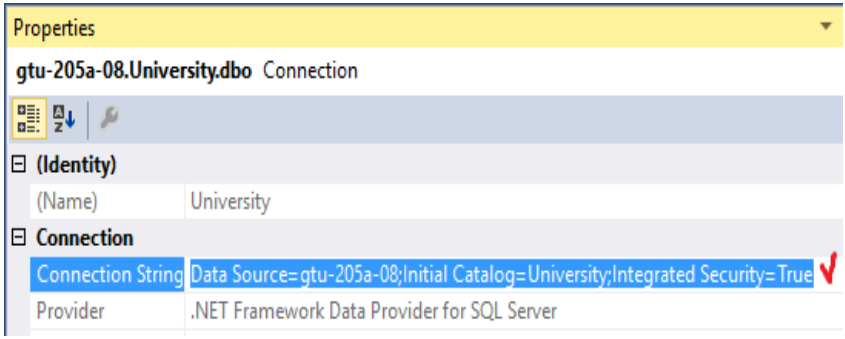


ნახ.3.15. ბაზასთან დაკავშირება

გამოჩნდება მიერთებული ბაზა University.dbo და ცხრილი Student მისი ატრიბუტებით:



ნახ.3.16. ბაზის Student -ცხრილი



ნახ.3.17. WPF პროექტის ბაზასთან დაკავშირების სტრიქონი Properties-ის Connection String-ში

ListView-ში ბაზის სტრიქონების გამოსატანად დავწეროთ მეთოდი ShowData(), რომელიც თავიდან ჩაიტვირთებვა ასე:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
    {
       ShowData(); // მეთოდი
    }
```

თვით ამ მეთოდის ტექსტი ასეთია:

```
public void ShowData()
  {
   SqlConnection con = new SqlConnection(@"Data Source=
                gtu-205a-08;Initial Catalog=University;
               Integrated Security=True");
   con.Open();
   SqlCommand comm = new SqlCommand("Select St_ID, Gvari,
                  JgupiID from Student", con);
   DataTable dt = new DataTable();
   SqlDataAdapter da = new SqlDataAdapter(comm);
   da.Fill(dt);
   listView1.DataContext = dt.DefaultView;
  }
```

- Add-მეთოდი:

```
private void btnAdd_Click(object sender, RoutedEventArgs e)
  {
    string Gvari = textBox1.Text;
    string JgupiID = textBox2.Text;
    string St_ID = textBox3.Text;
    SqlConnection con = new SqlConnection(@"Data Source=
        gtu-205a-08;Initial Catalog=University;
        Integrated Security=True");
     con.Open();
    SqlCommand comm = new SqlCommand("insert into
                         Student(Gvari,JgupiID, St_ID)
          values(@Gvari, @JgupiID, @St_ID)", con);
    comm.Parameters.AddWithValue("@Gvari", textBox1.Text);
    comm.Parameters.AddWithValue("@JgupiID", textBox2.Text);
```

```
    comm.Parameters.AddWithValue("@St_ID", textBox3.Text);
    comm.ExecuteNonQuery();
    con.Close();
    ShowData();
  }
```

- Update - მეთოდი:

```
private void btnUpdate_Click(object sender, RoutedEventArgs e)
 {
  if (listView1.SelectedItems.Count > 0)
   {
      DataRowView drv = (DataRowView)listView1.SelectedItem;
      string id = drv.Row[0].ToString();
      SqlConnection con = new SqlConnection(@"Data Source=
          gtu-205a-08;Initial Catalog=University;Integrated
                  Security=True");
      con.Open();
      SqlCommand comm = new SqlCommand("update Student set
Gvari=@Gvari,JgupiID=@JgupiID where St_ID=@St_ID", con);
    comm.Parameters.AddWithValue("@St_ID", id);
    comm.Parameters.AddWithValue("@Gvari", textBox1.Text);
    comm.Parameters.AddWithValue("@JgupiID", textBox2.Text);
    comm.ExecuteNonQuery();
    con.Close();
    ShowData();
   }
 }
```

- Delete-მეთოდი:

```
private void btnDelete_Click(object sender, RoutedEventArgs e)
 {
  if (listView1.SelectedItems.Count > 0)
   {
     DataRowView drv = (DataRowView)listView1.SelectedItem;
     string id = drv.Row[0].ToString();
     SqlConnection con = new SqlConnection(@"Data Source=
          gtu-205a-08;Initial Catalog=University;Integrated
                      Security=True");
```

```
     con.Open();
     SqlCommand comm = new SqlCommand("delete from Student
                         where St_ID=@St_ID", con);
     comm.Parameters.AddWithValue("@St_ID", id);
     comm.ExecuteNonQuery();
     ShowData();
   }
}
```

- Clear -მეთოდი:

```
private void btnClear_Click(object sender, RoutedEventArgs e)
  {
    textBox1.Text = "";
    textBox2.Text = "";
    textBox3.Text = "";

  }
```

ამგვარად, მთლიანი C# კოდის ლისტინგი მოცემულია ქვემოთ:

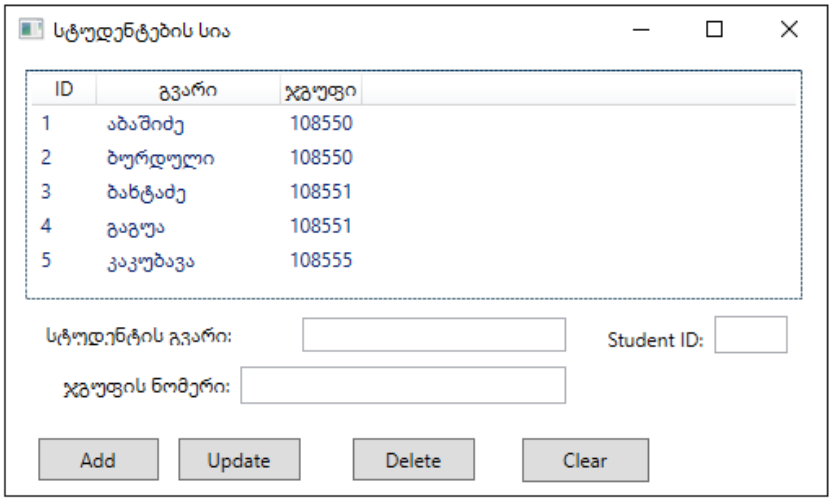
```
// --- C# კოდის ლისტინგი ------------

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Data.SqlClient;   // !!!
using System.Data;             // !!!

namespace WpfAppSQL_ListView
{
    /// <summary>
```

```
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
      public MainWindow()
        {
            InitializeComponent();
        }
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
       ShowData();
   }
public void ShowData()
  {
    SqlConnection con = new SqlConnection(@"Data Source=
                gtu-205a-08;Initial Catalog=University;
                Integrated Security=True");
   con.Open();
   SqlCommand comm = new SqlCommand("Select St_ID, Gvari,
                   JgupiID from Student", con);
   DataTable dt = new DataTable();
   SqlDataAdapter da = new SqlDataAdapter(comm);
   da.Fill(dt);
   listView1.DataContext = dt.DefaultView;
  }

private void btnAdd_Click(object sender, RoutedEventArgs e)
  {
    string Gvari = textBox1.Text;
    string JgupiID = textBox2.Text;
    string St_ID = textBox3.Text;
    SqlConnection con = new SqlConnection(@"Data Source=
        gtu-205a-08;Initial Catalog=University;
        Integrated Security=True");
     con.Open();
    SqlCommand comm = new SqlCommand("insert into
                          Student(Gvari,JgupiID, St_ID)
          values(@Gvari, @JgupiID, @St_ID)", con);
```

```
    comm.Parameters.AddWithValue("@Gvari", textBox1.Text);
    comm.Parameters.AddWithValue("@JgupiID", textBox2.Text);
    comm.Parameters.AddWithValue("@St_ID", textBox3.Text);
    comm.ExecuteNonQuery();
    con.Close();
    ShowData();
  }

private void btnUpdate_Click(object sender, RoutedEventArgs e)
 {
  if (listView1.SelectedItems.Count > 0)
   {
      DataRowView drv = (DataRowView)listView1.SelectedItem;
      string id = drv.Row[0].ToString();
      SqlConnection con = new SqlConnection(@"Data Source=
          gtu-205a-08;Initial Catalog=University;Integrated
                   Security=True");
      con.Open();
      SqlCommand comm = new SqlCommand("update Student set
Gvari=@Gvari,JgupiID=@JgupiID where St_ID=@St_ID", con);
    comm.Parameters.AddWithValue("@St_ID", id);
    comm.Parameters.AddWithValue("@Gvari", textBox1.Text);
    comm.Parameters.AddWithValue("@JgupiID", textBox2.Text);
    comm.ExecuteNonQuery();
    con.Close();
    ShowData();
   }
 }

private void btnDelete_Click(object sender, RoutedEventArgs e)
 {
  if (listView1.SelectedItems.Count > 0)
   {
     DataRowView drv = (DataRowView)listView1.SelectedItem;
     string id = drv.Row[0].ToString();
     SqlConnection con = new SqlConnection(@"Data Source=
          gtu-205a-08;Initial Catalog=University;Integrated
                       Security=True");
     con.Open();
     SqlCommand comm = new SqlCommand("delete from Student
                        where St_ID=@St_ID", con);
```

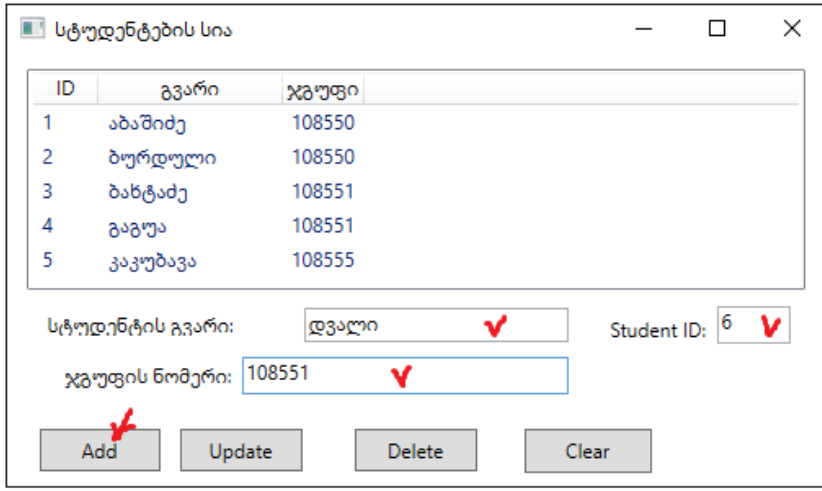
```
      comm.Parameters.AddWithValue("@St_ID", id);
      comm.ExecuteNonQuery();
      ShowData();
    }
}

private void btnClear_Click(object sender, RoutedEventArgs e)
  {
    textBox1.Text = "";
    textBox2.Text = "";
    textBox3.Text = "";
  }
 }
}
```

ექსპერიმენტის ჩასატარებლად გავმართოთ აგებული აპლიკაციის პროგრამა და ავამუშავოთ იგი. მიიღება 3.18 ნახაზზე ნაჩვენები საწყისი ცხრილი:

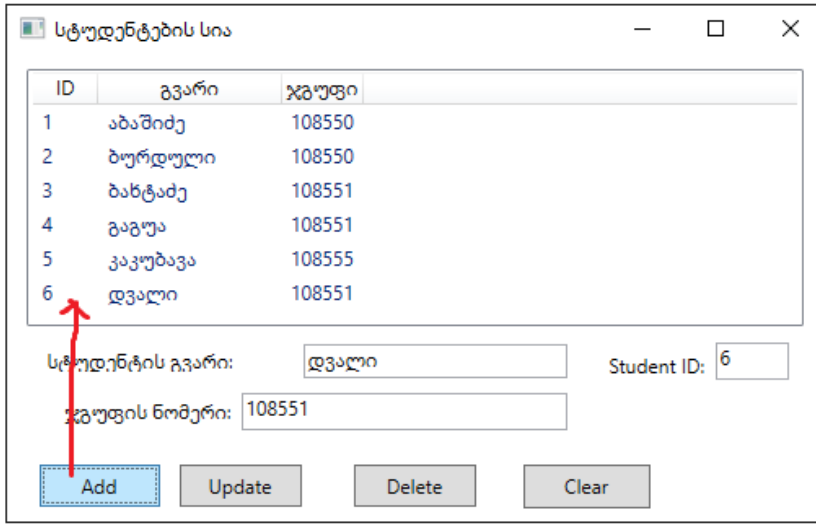


ნახ.3.18. SQL Server-ის University-ბაზაში Student-ცხრილის მიმდინარე ჩანაწერები

- ჩავამატოთ ცხრილში ახალი სტუდენტი (Add) - ნახ.3.19.

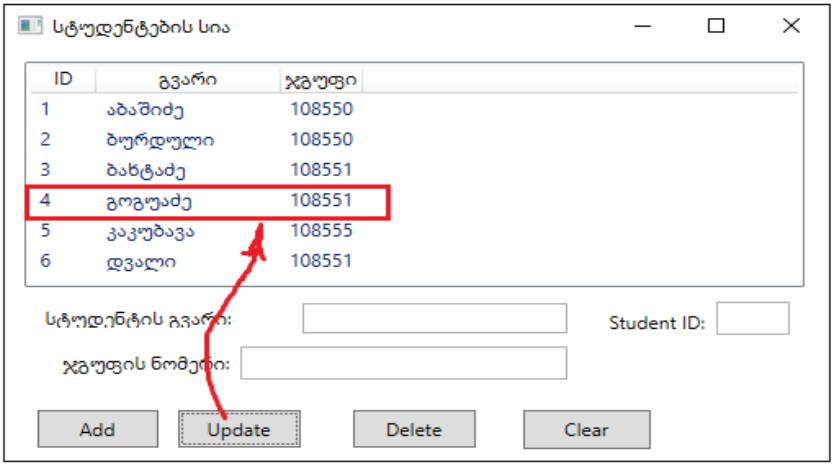


ნახ.3.19-ა. ცხრილში ახალი სტუდენტის ჩამატების პროცედურა



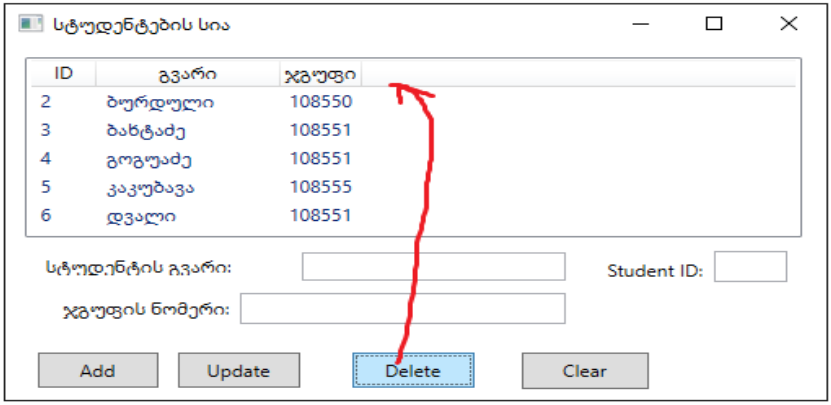
ნახ.3.19-ბ. ცხრილში ჩამატების შედეგი

- შევცვალოთ ცხრილში მონაცემის მნიშვნელობა (Update). მაგალითად, სტუდენტი „გაგუა“ შეცდომითაა, უნდა იყოს „გაგუაძე“.



ნახ.3.20. მონაცემის შეცვლის შედეგი

- წავშლოთ ცხრილიდან სტუდენრი „აბაშიძე“, ვინაიდან ის გადავიდა სხვა უნივერსიტეტში.



ნახ.3.21. მონაცემების წაშლის შედეგი

3.3. WPF-აპლიკაციის აგება საპრობლემო სფეროსთვის „უნივერსიტეტი“

საუნივერსიტეტო მართვის საინფორმაციო სისტემის შექმნა ან არსებული სისტემის მოდიფიკაცია ინტეგრაციის პრინციპების საფუძველზე მოითხოვს ამ საპრობლემო სფეროს ობიექტორიენტირებული, პროცესორიენტირებული და სერვისორიენტირებული მიდგომების კომპლექსურ გამოყენებას [24]. სისტემის შესაბამისი ინფრასტრუქტურის დასამუშავებლად კი აუცილებელია დღეისათვის არსებული ისეთი სტანდარტებისა და მეთოდოლოგიების გამოყენება, როგორიცაა BSI, ITIL, COBIT [25], რაც საბოლოო ჯამში უზრუნველყოფს უსაფრთხო განაწილებული ინფორმაციული სისტემის შექმნას, მის შემდგომ მასშტაბირებას და განვითარებას.

მეორეს მხრივ, ინტეგრაციის პროცესში სისტემის ცალკეული კომპონენტების დასამუშავებლად დროითი პარამეტრების და პროგრამული პროდუქტის ხარისხის გასაუმჯობესებლად აუცილებელი ხდება თანამედროვე CASE ტექნოლოგიების გამოყენება, როგორიცაა მაგალითად, Enterprise Architect (UML-ის ინსტრუმენტული საშუალება), Natural Object Role Modeling Architect (NORMA) და სხვ., რომლებიც მაიკროსოფთის Visual Studio .NET Framework პაკეტის სამუშაო გარემოს თავსებადია [14,26,27].

წინამდებარე პარაგრაფში გადმოცემულია საუნივერსიტეტო მართვის საინფორმაციო სისტემის საპილოტო ვერსიის მაგალითზე მონაცემთა განაწილებული ბაზის და მომხმარებელთა ინტერფეისების დაპროექტების და პროგრამული რეალიზაციის პროცედურების დეტალური აღწერა ზემოაღნიშნული ახალი ტექნოლოგიებისა და სერვისორიენტირებული არქიტექტურის საფუძველზე [16].

პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის ეტაპების შესაბამისად (ანალიზი, დაპროექტება, დეველოპმენტი, ტესტირება, დანერგვა), საუნივერსიტეტო მართვის საინფორმაციო

სისტემის შექმნა, UML-ტექნოლოგიით, ითვალისწინებს სისტემის ფუნქციონალური და არაფუნქციონალური მოთხოვნილებების განსაზღვრას, ობიექტორიენტირებული ანალიზის და დაპროექტების განხორციელებას, შესაბამისად სისტემასთან მომხმარებელთა მუშაობის ინტერაქტიული სცენარების, კლასთა-ასოსიაციების და მდგომარეობათა დიაგრამების აგებით სხვადსხვა შემთხვევით მოვლენათა შესაბამისად.

ესაა ცოდნა სამართავი ობიექტის შესახებ, მისი სტატიკური (მდგომარეობათა სიმრავლე) და დინამიკური (ქცევათა სიმრავლე) მოდელებით. თუ ობიექტორიენტირებული მოდელირების ტერმინებით ვისარგებლებთ, დასმული ამოცანის გადაწყვეტის „გასაღებს“ კლასების, ობიექტების, კლასთაშორისი კავშირების, ობიექტ-როლური და არსთა-დამოკიდებულების მოდელებისა და სხვა სახის დიაგრამების აგება წარმოადგენს. ხოლო შემდეგ, კლასთა-ასოციაციებისა და არსთა-დამოკიდებულების დიაგრამათა საფუძველზე განხორციელდება მიზნობრივი სისტემის პროგრამული კოდების რეალიზაციის ავტომატიზებული პროცესი ტესტირებით [28].

ამგვარად, ჩვენ განვიხილვთ კონკრეტული მართვის საინფორმაციო სისტემის („უნივერსიტეტი“) აგების ამოცანების სპექტრს და ამ პროცესში გამოვყოფთ პროგრამული სისტემის ტესტირების ფუნქციურ ამოცანას და მის რეალიზაციას [16].

3.3.1. სისტემის ობიექტ-როლური მოდელის დაპროექტება

განვიხილოთ ზოგადად „უნივერსიტეტის“ კლასის ობიექტისათვის მონაცემთა განაწილებული ბაზის დაპროექტების ამოცანა. უნივერსიტეტის საპრობლემო სფეროს არაფორმალიზებული აღწერის ობიექტებია (ტერმინთა ლექსიკონი): ფაკულტეტი, დეპარტამენტი, სტუდენტი, ლექტორი, საგანი (აკადემიური

დისციპლინები, რომლებიც იკითხება შესაბამისი კათედრებსა და სპეციალობების მიხედვით), სასწავლო გეგმები, სილაბუსები (პროგრამები), ლექციები, პრაქტიკული და ლაბორატორიული სამუშაოები, აუდიტორიები, გამოცდები, ტესტირება და ა.შ.

ობიექტ-როლური მოდელირების თეორიის წესების თანახმად საჭიროა აიგოს უნივერსიტეტის სასწავლო პროცესის შესაბამისი ORM-დიაგრამა. ამისათვის ფაქტ-შეზღუდვების ერთობლიობით (რომელსაც ადგენს სისტემური ანალიტიკოსი და საბოლოო მომხმარებელი), რომლებშიც ასახულია საპრობლემო სფეროს შესახებ ცოდნა (კლასებისა და ობიექტების ძირითადი ტერმინები და ქცევის წესები, დებულებით დადგენილი კანონზომიერებები და სხვ.), გადაიტანება Ms_Visual Studio.NET Framework სამუშაო გარემოში, ობიექტ-როლური მოდელის, NORMA-პაკეტის (Natural ORM Architect) ინტერფეისზე [14,15]. მაგალითად, ასეთი ფაქტები შეიძლება იყოს:

f1 : ლექტორს აქვს გვარი
f2 : ლექტორი მუშაობს დეპარტამენტში
f3 : ლექტორს აქვს თანამდებობა
f4 : ლექტორს აქვს ტელეფონი
f5 : ლექტორს აქვს ელ-ფოსტა
f6 : ლექტორს აქვს ხარისხი
f7 : ლექტორი კითხულობს საგანს
f8 : ლექტორი ასწავლის #-ჯგუფს
f8 : სტუდენტი არის #-ჯგუფში
. . .
f50 : ლექტორი მუშაობს #-დეპარტამენტში
f51 : დეპარტამენტი ეკუთვნის #-ფაკულტეტს
f52 : #-ჯგუფი ეკუთვნის #-დეპარტამენტს
f53 : ფაკულტეტს აქვს დასახელება
. . .

f100 : სტუდენტი არ შეიძლება იყოს ერთზე მეტ ჯგუფში

f101 : ლექტორი არ შეიძლება იყოს სრულ შტატზე ერთზე მეტ დეპარტამენტში

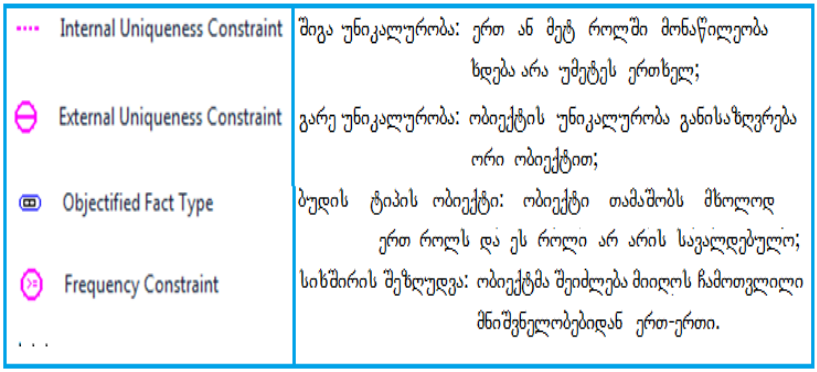
f102 : ლექტორი დროის ერთ მომენტში არ შეიძლება იყოს ორ სხვადასხვა აუდიტორიაში

და ა.შ.

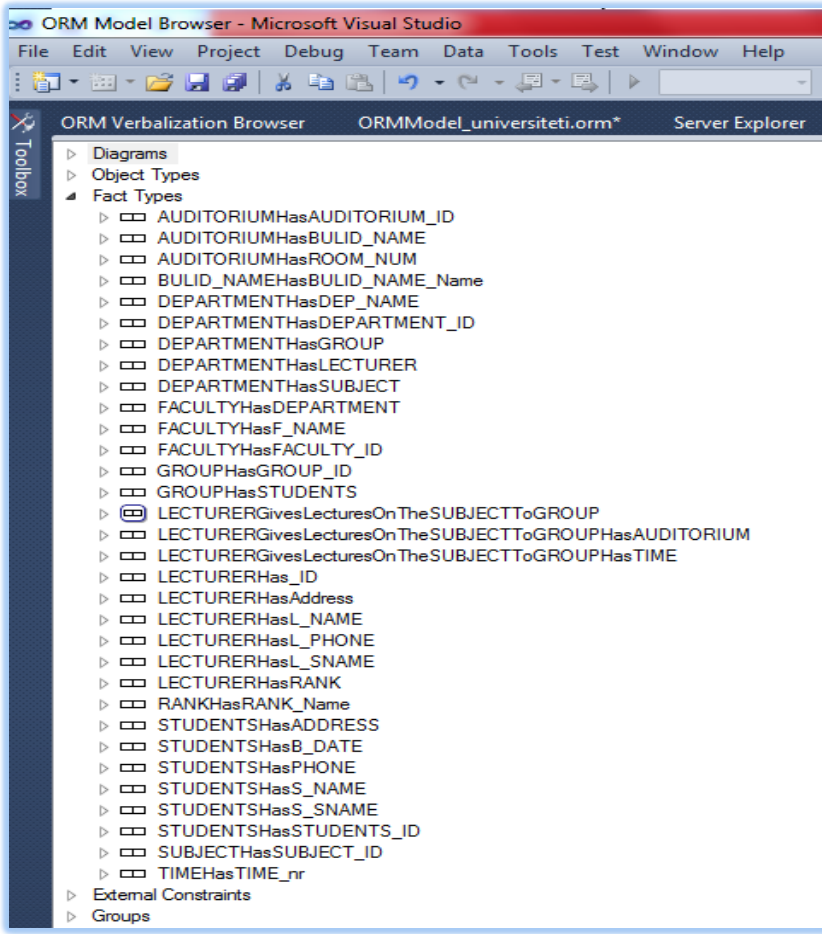
3.22 ნახაზზე წარმოდგენილია ობიექტ-როლური მოდელირების თეორიაში არსებული ზოგიერთი შეზღუდვის მაგალითი, რომლებიც გამოყენებულია ჩვენ მაგალითში.

შეზღუდვების აღწერა ხდება კატეგორიალური მიდგომის საფუძველზე, რომელიც აერთიანებს სალაპარაკო ენის ფორმალური გრამატიკის და ლოგიკურ-ალგებრულ წესებს. შედეგად მიიღება პრედიკატები, რომლებიც შეიძლება იყოს ერთ-, ორ- ან n-ადგილიანი [67].

3.23 ნახაზზე მოცემულია Visual Studio.NET სამუშაო გარემოში ORM ინსტრუმენტის გამოყენებით მიღებული შედეგები „უნივერსიტეტის“ ფაქტების შეტანის საფუძველზე.

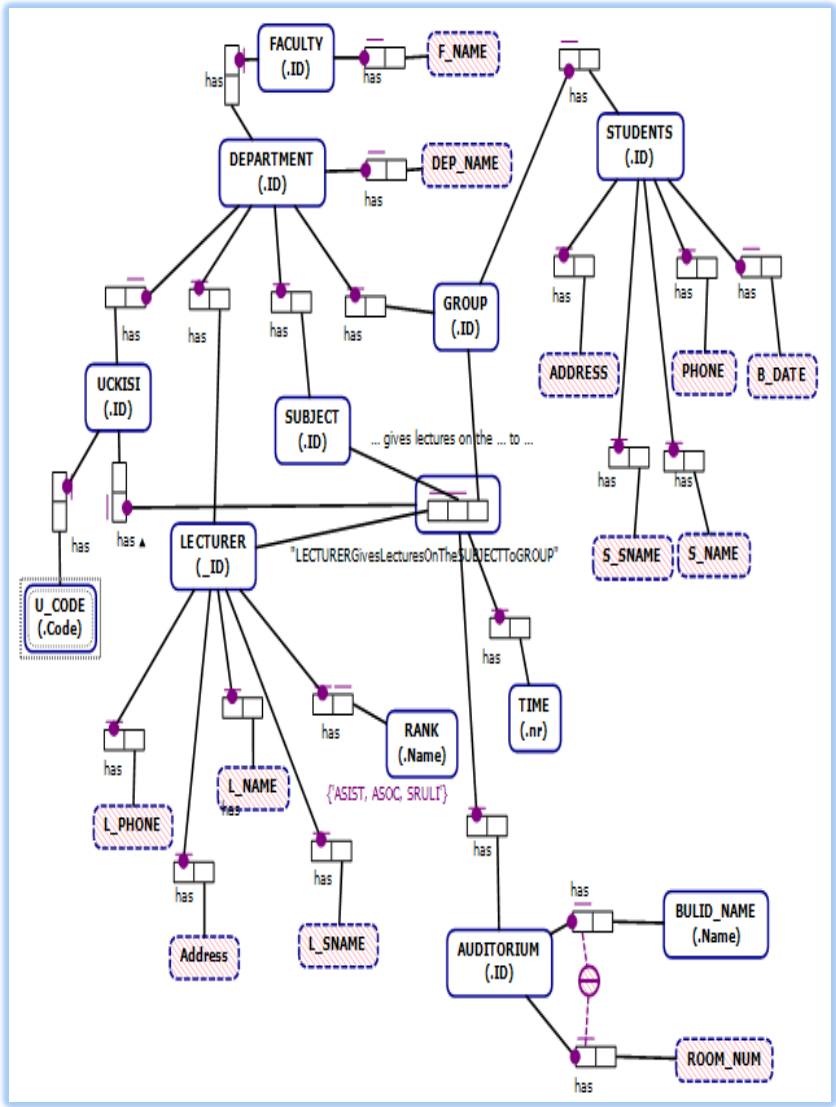


ნახ.3.22. შეზღუდვების აღწერის მაგალითები ORM-დიაგრამაზე



ნახ.3.23. „უნივერსიტეტის“ ფაქტების აღწერის ფრაგმენტი

ზემოჩამოთვლილი ფაქტებიდან NORMA ინსტრუმენტი გვაძლევს შემდეგი სახის ORM-დიაგრამას (ნახ.3.24). აქ შესაძლებელია ახალი ფაქტის დამატება, არსებულის მოდიფიკაცია / წაშლა.

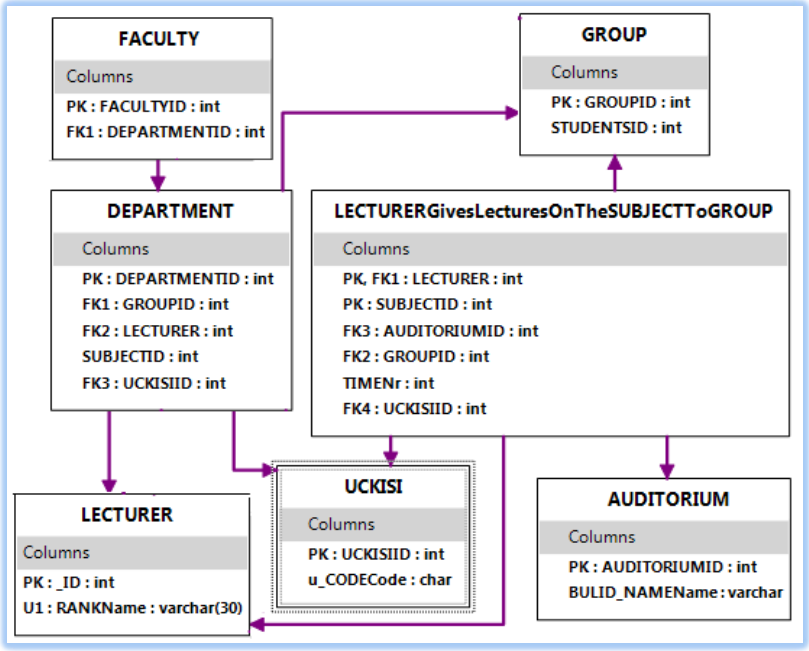


ნახ.3.24. „უნივერსიტეტის“ ORM დიაგრამის ფრაგმენტი
(საპრობლემო სფეროს კონცეპტუალური მოდელი-1)

3.3.2. არსთა-დამოკიდებულების ER მოდელის დაპროექტება

მომდევნო ეტაპზე განხორციელდება ობიექტ-როლური მოდელის ავტომატიზებული გადაყვანა არსთა-დამოკიდებულების მოდელში. სისტემის დამპროექტებელი გააქტიურებს NORMA პაკეტის მენიუდან გენერაციის პროცედურას. ORM-დიაგრამიდან გენერირებული ERM-დიაგრამა მოცემულია 3.25 ნახაზზე.

შესაბამისად გამოკვეთილია შვიდი ობიექტი (არსი - Entity): ფაკულტეტი (Faculty), დეპარტამენტი (Department), ჯგუფი (Group), სტუდენტი (Students), ლექტორი (Lecturer), საგანი (Subject), აუდიტორია (Auditorium), საგამოცდო_უწყისი (Uckisi).

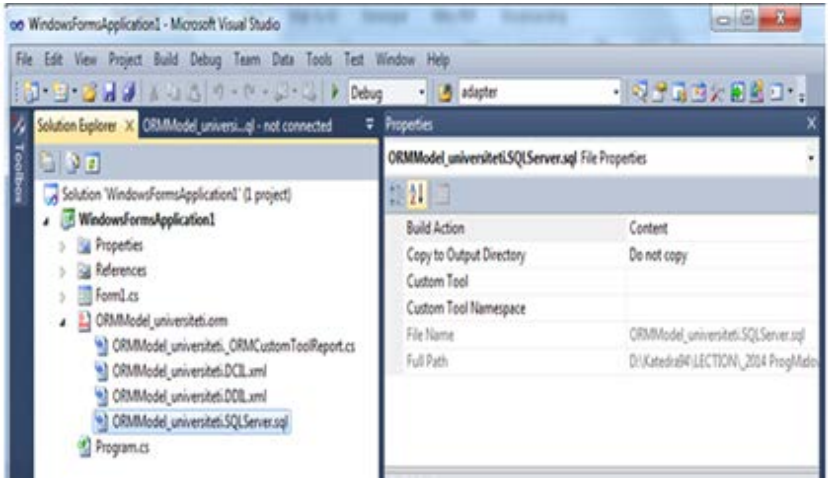


ნახ.3.25. ORM დიაგრამიდან გენერირებული ER-მოდელი (საპრობლემო სფეროს კონცეპტუალური მოდელი-2)

დიალოგში შესაძლებელია ER-მოდელის ობიექტების (Tables) განლაგების შეცვლა, რათა ვიზუალურად უფრო მოხერხებული მდებარეობა მიიღოს თითოეულმა, ობიექტთაშორისი კავშირების რაც შეიძლება ნაკლები გადაკვეთებით. ცხრილებში ჩანს ობიექტის სახელი და ატრიბუტთა დასახელებები, ტიპების მითითებით. აგრეთვე ასახულია პირველადი (PK) და მეორეული (FK) გასაღებური ატრიბუტები და ა.შ.

3.3.3. მონაცემთა ბაზის სერვერზე განთავსება

მონაცემთა ბაზის კონცეპტუალური ERM სქემის აგების შემდეგ საჭიროა მის საფუძველზე სისტემის მიერ დაიწეროს შუალედური ტექსტური ტიპის DLL–ფაილი, რომელიც მომავალში SQL Server მონაცემთა ბაზების მართვის სისტემამ უნდა გამოიყენოს. ამგვარად, ER-დიაგრამიდან, ცხრილებითა და ატრიბუტებით, ავტომატურად გენერირდება .DDL ფაილები, რომლებიც შემდგომ სტრუქტურულად მოთავსდება SQL Server მონაცემთა ბაზაში. 3.26 ნახაზზე ნაჩვენებია ამ ეტაპის პროცესის ინიცირების დიალოგური სქემა.



ნახ.3.26. DDL ფაილის გენერაცია ER-მოდელიდან

3.1 ლისტინგში მოცემულია ავტომატურად გენერირებული DDL-ფაილის ტექსტის ფრაგმენტი.

```
!-- Listing_3.1 -----DDL file --------------------------
CREATE SCHEMA ORMModel1
GO
GO
CREATE TABLE ORMModel1.FACULTY
(  FACULTYID INTEGER IDENTITY (1, 1) NOT NULL,
   DEPARTMENTID INTEGER NOT NULL,
   CONSTRAINT FACULTY_PK PRIMARY KEY(FACULTYID)  )
GO
CREATE TABLE ORMModel1.DEPARTMENT
(  DEPARTMENTID INTEGER IDENTITY (1, 1) NOT NULL,
   GROUPID INTEGER NOT NULL,
   LECTURER INTEGER NOT NULL,
   SUBJECTID INTEGER IDENTITY (1, 1) NOT NULL,
   CONSTRAINT DEPARTMENT_PK PRIMARY KEY(DEPARTMENTID)   )
GO
CREATE TABLE ORMModel1."GROUP"
(  GROUPID INTEGER IDENTITY (1, 1) NOT NULL,
   STUDENTSID INTEGER IDENTITY (1, 1) NOT NULL,
   CONSTRAINT GROUP_PK PRIMARY KEY(GROUPID)       )
GO
CREATE TABLE ORMModel1.LECTURER
(  "_ID" INTEGER IDENTITY (1, 1) NOT NULL,
   RANKName NATIONAL CHARACTER VARYING(30) NOT NULL,
   CONSTRAINT LECTURER_PK PRIMARY KEY("_ID"),
   CONSTRAINT LECTURER_UC UNIQUE(RANKName),
   CONSTRAINT LECTURER_RANKName_RoleValueConstraint1 CHECK
(RANKName IN (N'ASIST, ASOC, SRULI'))  )
GO
CREATE TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
(  LECTURER INTEGER NOT NULL,
   SUBJECTID INTEGER IDENTITY (1, 1) NOT NULL,
   AUDITORIUMID INTEGER NOT NULL,
   GROUPID INTEGER NOT NULL,
   TIMENr INTEGER NOT NULL,
   CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_PK
PRIMARY KEY(LECTURER, SUBJECTID)     )
GO
CREATE TABLE ORMModel1.AUDITORIUM
(  AUDITORIUMID INTEGER IDENTITY (1, 1) NOT NULL,
```

```
   BULID_NAMEName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
   CONSTRAINT AUDITORIUM_PK PRIMARY KEY(AUDITORIUMID)
)
GO
ALTER TABLE ORMModel1.FACULTY ADD CONSTRAINT FACULTY_FK FOREIGN
KEY (DEPARTMENTID) REFERENCES ORMModel1.DEPARTMENT
(DEPARTMENTID) ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.DEPARTMENT ADD CONSTRAINT DEPARTMENT_FK1
FOREIGN KEY (GROUPID) REFERENCES ORMModel1."GROUP" (GROUPID) ON
DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.DEPARTMENT ADD CONSTRAINT DEPARTMENT_FK2
FOREIGN KEY (LECTURER) REFERENCES ORMModel1.LECTURER ("_ID") ON
DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
ADD CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK1
FOREIGN KEY (LECTURER) REFERENCES ORMModel1.LECTURER ("_ID") ON
DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
ADD CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK2
FOREIGN KEY (GROUPID) REFERENCES ORMModel1."GROUP" (GROUPID) ON
DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
ADD CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK3
FOREIGN KEY (AUDITORIUMID) REFERENCES ORMModel1.AUDITORIUM
(AUDITORIUMID) ON DELETE NO ACTION ON UPDATE NO ACTION
GO
GO
```

შეიძლება ჩვთვალოთ, რომ ამ DDL ფაილის კოპირებით Ms SQL Server–ში შეიქმნება შესაბამისი ბაზა, ცხრილებით, ატრიბუტებით და კავშირებით.

რა თქმა უნდა, შესაძლებელია აქაც დამპროექტებლის ჩარევა ბაზის სტრუქტურის ზოგიერთი კომპონენტის შესასწორებლად, საჭიროების შემთხვევაში.

3.3.4. ბიზნესპროცესის სერვისის შექმნა

როგორც ცნობილია, ბიზნესპროცესი შეიძლება განთავსდეს ვებ-სერვისში, რომელიც უზრუნველყოფს ბიზნესპროცესის გადაწყვეტილების (შედეგის) მიწოდებას კლიენტებისთვის (ვებ-აპლიკაციებისთვის). ვებ-სერვისი იღებს მოთხოვნას კლიენტისგან, ასრულებს მის დამუშავებას და უბრუნებს პასუხს. ეს პროცედურები სრულდება Receive და Send ქმედებებით.

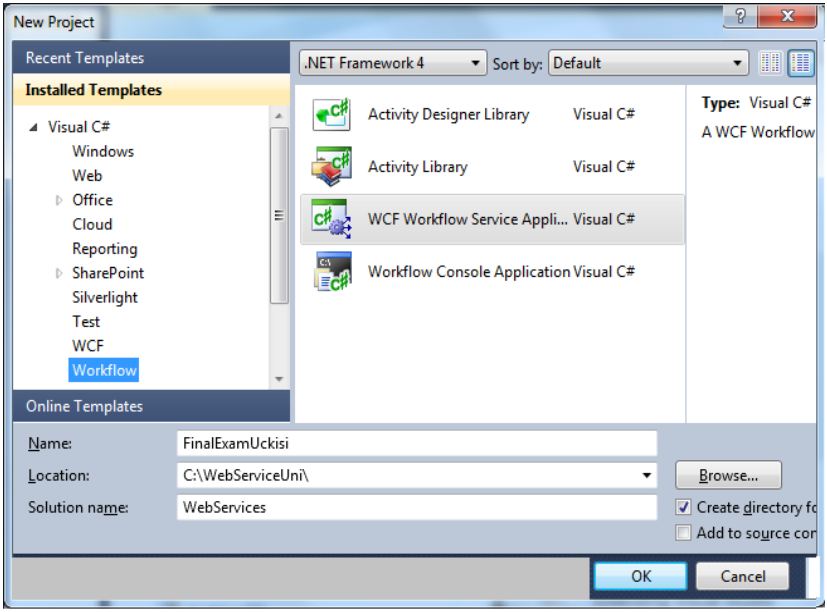
განვიხილოთ ჩვენი მაგალითის რეალიზაცია ჰიბრიდული ტექნოლოგიების, WF (Workflow Foundation) და WCF ბაზაზე (Windows Communication Foundation) [69]. ისევე, როგორც WPF (Windows Presentation Foundation) ტექნოლოგია, Visual Studio .NET Framework 4.0/4.5 გარემოში ქმნის მომხმარებელზე ორიენტირებულ მაღალი ხარისხის დიზაინის აპლიკაციებს C#.NET (ლოგიკის ნაწილი) და XAML (დიზაინის ნაწილი) ენების საფუძველზე [1-3].

ავამუშავოთ Visual Studio და შევქმნათ ახალი პროექტი WCF Workflow Service Application template გამოყენებთ. შევიტანოთ პროექტის სახელი FinalExamUckisi და solution-ის სახელი WebServices (ნახ.3.27).

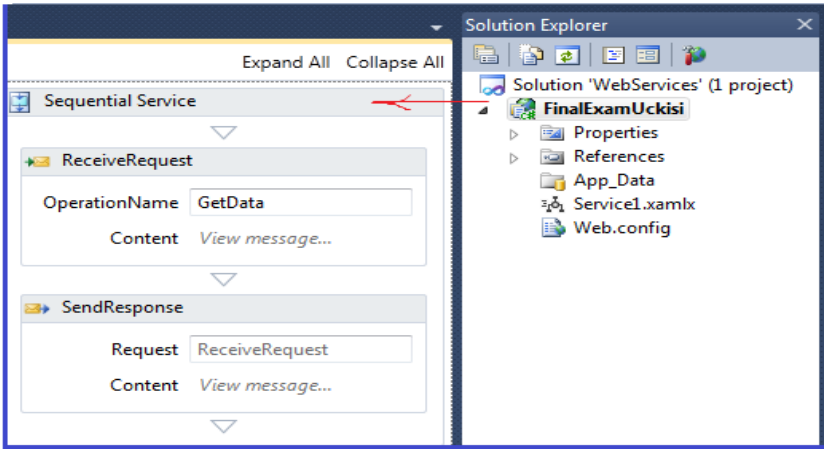
შეიქმნება საინიციალიზაციო workflow Sequence ბლოკი, რომელიც შეიცავს Receive და SendReply ქმედებებს, როგორც 3.28 ნახაზზეა ნაჩვენები.

თავიდან საჭიროა ამ ქმედებების კონფიგურაცია სერვისის კონტრაქტის განსაზღვრის მიზნით, რომელსაც ისინი დააკმაყოფილებს. შემდეგ დავამატოთ დამუშავების ბიზნესპროცესი, რომელიც განხორციელდება Receive და SendReply ქმედებებს შორის.

შაბლონი შექმნის საწყის ბიზნესპროცესს ფაილში, სახელით Service1.xamlx. შევცვალოთ Solution Explorer-ში ეს სახელი Final-ExamUckisi.xamlx -ით.



ნახ.3.27. Visual Studio.NET –ში WCF-პროექტის შექმნა



ნახ.3.28. FinalExamUckisi პროექტის workflow Sequence ბლოკი

სერვისი, რომელიც მაგალითის სახით უნდა შევქმნათ, „საფინალო გამოცდისთვის“, ძებნის შესაბამის უწყისებს მითითებული აკადემიური ჯგუფის, ლექტორის და აკადემიური საგნის მიხედვით. ელექტრონული საგამოცდო უწყისები ეკუთვნის ფაკულტეტის დეკანატს, ხოლო მათი დამუშავება ხდება დეპარტამენტთა ლექტორების მიერ”.

3.3.5. სერვისის კონტრაქტის განსაზღვრა

WCF სისტემებში იყენებენ კონტრაქტების სამ დონეს: მონაცემთა კონტრაქტი, შეტყობინებათა კონტრაქტი და სერვისის კონტრაქტი [3,16].

მონაცემთა კონტრაქტის დანიშნულებაა შეთანხმება კლიენტსა და სერვისს შორის ერთმანეთთან გასაცვლელ მონაცემებზე (ითვალისწინებენ მონაცემთა სტრუქტურებს, პარამეტრებს, მოწეს-რიგებას და ა.შ.).

შეტყობინებათა კონტრაქტი უზრუნველყოფს SOAP (Simple Object Access Protocol) შეტყობინებების კონტროლს, რომელიც გამოიყენება ქსელში სხვადასხვა შეტყობინების გასაცვლელად XML ფორმატში. იგი უზრუნველყოფს აგრეთვე ინფორმაციის გაცვლის უსაფრთხოებას შეტყობინებების დონეზე.

სერვისის კონტრაქტი (ანუ კონტრაქტი მომსახურებისთვის) განსაზღვრავს ოპერაციათა სახეებს, რომლებსაც უზრუნველყოფს სერვისი. იგი კლიენტს აწვდის ასევე ინფორმაციას: შეტყობინებაში მონაცემთა ტიპების შესახებ, ოპერაციათა ადგილმდებარეობის შესახებ, ინფორმირების პროტოკოლისა და სერიალიზაციის ფორმატის შესახებ, შეტყობინებათა გაცვლის შაბლონების შესახებ (ცალმხრივი, ორმხრივი ან კითხვა/პასუხის ტიპებით).

ჩვენი პროექტისთვის სერვისის კონტრაქტის ასაგებად უნდა შევქმნათ საგამოცდო უწყისის ინფორმაციის კლასი C# ნაზე - UckisiInfo.cs. ამისთვის Solution Explorer-ში მარჯვენა ღილაკით

FinalExamUckisi პროექტზე ვირჩევთ Add->Cla სახელით UckisiInfo.cs, რომლის ტექსტი მოცემულია 3.2 ლისტინგში.

```
// ----- ლისტინგი_3.2 – Service Contract ---
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;

namespace FinalExamUckisi
{  // ---- სერვისის კონტრაქტის განსაზღვრა ---
IfinalExamUckisi,   რომელიც
   // --- შედგება ერთი მეთოდისგან  -  LookupUckisi () ------
     [ServiceContract]
   public interface IFinalExamUckisi
     {   [OperationContract]
         UckisiInfoList LookupUckisi(UckisiSearch request);
     }
//-- მოთხოვნის შეტყობინების განსაზღვრა , UckisiSearch ---
     [MessageContract(IsWrapped = false)]
    public class UckisiSearch
      {  private String _JGUPI;
         private String _Sagani;
         private String _Lectori;
         public UckisiSearch() {  }
         public UckisiSearch(String sagani, String lectori,
                             String jgupi)
         {
             _Sagani  = sagani;
             _Lectori = lectori;
             _JGUPI   = jgupi;       }
         #region Public Properties
         [MessageBodyMember]
         public String Sagani
         {   get { return _Sagani; }
```

```
        set { _Sagani = value; }    }
    [MessageBodyMember]
    public String Lectori
    {   get { return _Lectori; }
        set { _Lectori = value; }   }
    [MessageBodyMember]
    public String JGUPI
    {   get { return _JGUPI; }
        set { _JGUPI = value; }     }
    #endregion Public Properties
}

// --- UckisiInfo კლასის განსაზღვრა -----
    [MessageContract(IsWrapped = false)]
public class UckisiInfo
{   private Guid _ExamUckisiID;
    private String _JGUPI;
    private String _Sagani;
    private String _Lectori;
    private String _Status;
    public UckisiInfo() {   }
    public UckisiInfo(String sagani, String lectori,
                      String jgupi, String status)
    {   _Sagani = sagani;
        _Lectori = lectori;
        _JGUPI = jgupi;
        _Status = status;
        _ExamUckisiID = Guid.NewGuid();    }
    #region Public Properties
    [MessageBodyMember]
    public Guid ExamUckisiID
    {   get { return _ExamUckisiID; }
        set { _ExamUckisiID = value; }     }
    [MessageBodyMember]
    public String Sagani
```

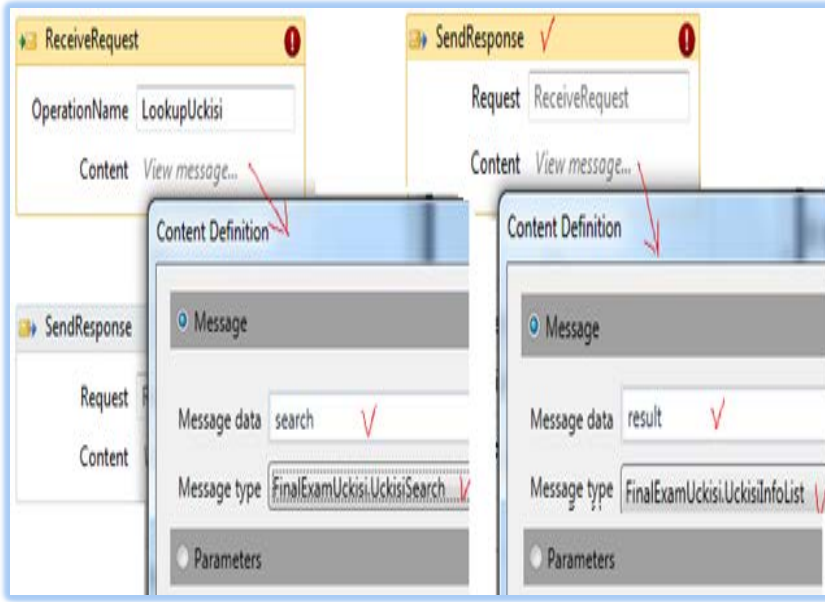
```
        {   get { return _Sagani; }
            set { _Sagani = value; }      }
        [MessageBodyMember]
       public String Lectori
        {   get { return _Lectori; }
            set { _Lectori = value; }     }
        [MessageBodyMember]
       public String JGUPI
        {   get { return _JGUPI; }
            set { _JGUPI = value; }       }
        [MessageBodyMember]
       public String status
        {   get { return _Status; }
            set { _Status = value; }      }
        #endregion Public Properties
    }
    //--საპასუხო შეტყობინების განსაზღვრა --- UckisiInfoList---
       [MessageContract(IsWrapped = false)]
    public class UckisiInfoList
    {   private List<UckisiInfo> _UckisiList;
        public UckisiInfoList()
        {  _UckisiList = new List<UckisiInfo>(); }
        [MessageBodyMember]
        public List<UckisiInfo> UckisiList
        {    get { return _UckisiList; }    }
    }
}
```

სერვისის კონტრაქტი IFinalExamUckisi შეიცავს ერთადერთ მეთოდს LookupUckisi(). იგი მონაცემებს გადასცემს UckisiSearch კლასს, რომელსაც აქვს სხვადასხვა თვისებები, საჭირო საგამოცდო უწყისის მოსაძებნად, მაგალითად, ლექტორი, საგანი, ჯგუფი. ის აბრუნებს უკან UckisiInfoList კლასს, რომელიც შეიცავს UckisiInfo კლასების კოლექციას. F6 ამოქმედებით აიგება გადაწყვეტა (solution).

ამგვარად, ჩვენ განვსაზღვრეთ შეტყობინებები, რომლებიც გადაიცემა სერვის–მეთოდების მიერ პარამეტრების სახით.

3.3.6. Receive და SendReply კონფიგურირება

შემდეგ ეტაპზე FinalExamUckisi.xamlx-ში ReceiveRequest ქმედებისთვის OperationName თვისებაში ვათავსებთ LookupUckisi სახელს. WorkflowService-დიზაინერში შევქმნით ორ ახალ ცვლადს (Variables): search ცვლადი, შემომავალი შეტყობინების შესანახად და result ცვლადი, გამოსატანი შედეგისთვის (ნახ.3.29). Message ბუტონი უნდა იყოს ჩართული და ტიპებიც არჩეული.



ნახ. 3.29. შემავალი მოთხოვნის შეტყობინების და გამომავალი შედეგის ცვლადების განსაზღვრა

3.3.7. PerformLookup ქმედების აგება (ძებნის შესრულება)

ძებნის განსახორციელებლად (მონაცემთა ბაზებთან მიმართვის მიზნითაც) ვქმნით ახალ კლასს PerformLookup.cs, რომელიც Solution Explorer-ში FinalExamUckisi-პროექტისთვის აირჩევა Add->NewItem, Workflow-კატეგორიაში, Code Activity-ით. ამ ახალი კლასის ტექსტი მოცემულია 3.3 ლისტინგში.

```
// ---- ლისტინგი_3.3 --- PerformLookup ----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
namespace FinalExamUckisi
{
 public sealed class PerformLookup : CodeActivity
  {
   public InArgument<UckisiSearch> Search { get; set; }
   public OutArgument<UckisiInfoList> UckisiList {get;set;}
   protected override void Execute(CodeActivityContext
                                                    context)
    { string lectori = Search.Get(context).Lectori;
      string sagani = Search.Get(context).Sagani;
      string jgupi = Search.Get(context).JGUPI;

      UckisiInfoList l = new UckisiInfoList();

     l.UckisiList.Add(new UckisiInfo(sagani, lectori,
          jgupi, "Available"));
```

```
    l.UckisiList.Add(new UckisiInfo(sagani, lectori,
          jgupi, "CheckedOut"));
    l.UckisiList.Add(new UckisiInfo(sagani, lectori, jgupi,
          "Missing"));
    l.UckisiList.Add(new UckisiInfo(sagani, lectori, jgupi,
          "Available"));
      UckisiList.Set(context, l);        }
   }
}
```

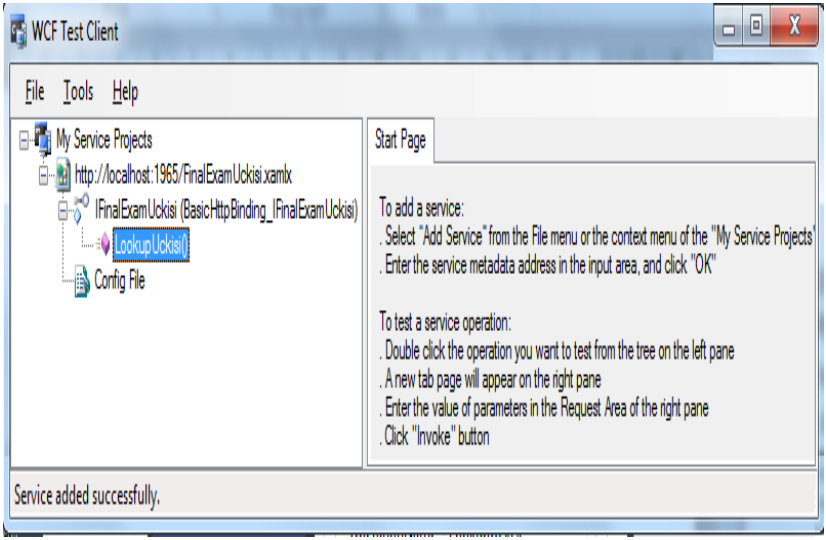
PerformLookup ქმედება ჩაემატება ინსტრუმენტების პანელზე. იგი უნდა გადმოვიტანოთ „ReceiveRequest" და „SendResponse" ქმედებებს შორის შესასრულებლად. ამასთანავე მისი Search თვისებისთვის ჩავწერეთ search, ხოლო UckisiList თვისებისთვის კი - result.

3.3.8. სერვისის ტესტირება

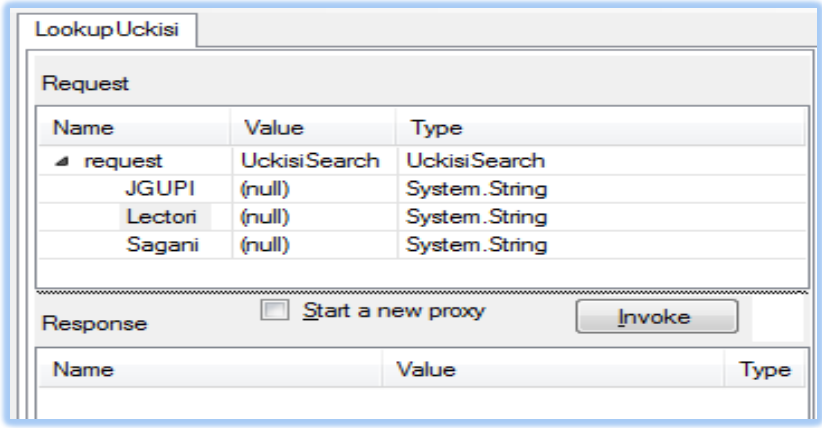
აპლიკაციის საბოლოო გამართვამდე უნდა მოვახდინოთ აგებული სერვისების ტესტირება. F5-ით ავამუშავოთ სერვისის გამართვის პროცედურა (debug). ვინაიდან ეს Web-სერვისია, Visual Studio ავტომატურად აამუშავებს WCF Test Client-ს [20,28].

ეს მეტად მოსახერხებელი უტილიტაა. იგი ჩატვირთავს Web-სერვისებს და აღმოაჩენს მეთოდებს, რომლებიც გათვალისწინებულია. ეს შედეგი ჩანს 3.30 ნახაზის მარცხენა პანელზე.

LookupUckisi() მეთოდზე 2-ჯერ დაჭერით მარჯვენა პანელის ზედა ნაწილში გამოიყოფა ადგილი შემოსული შეტყობინების განსათავსებლად (ნახ.3.31).

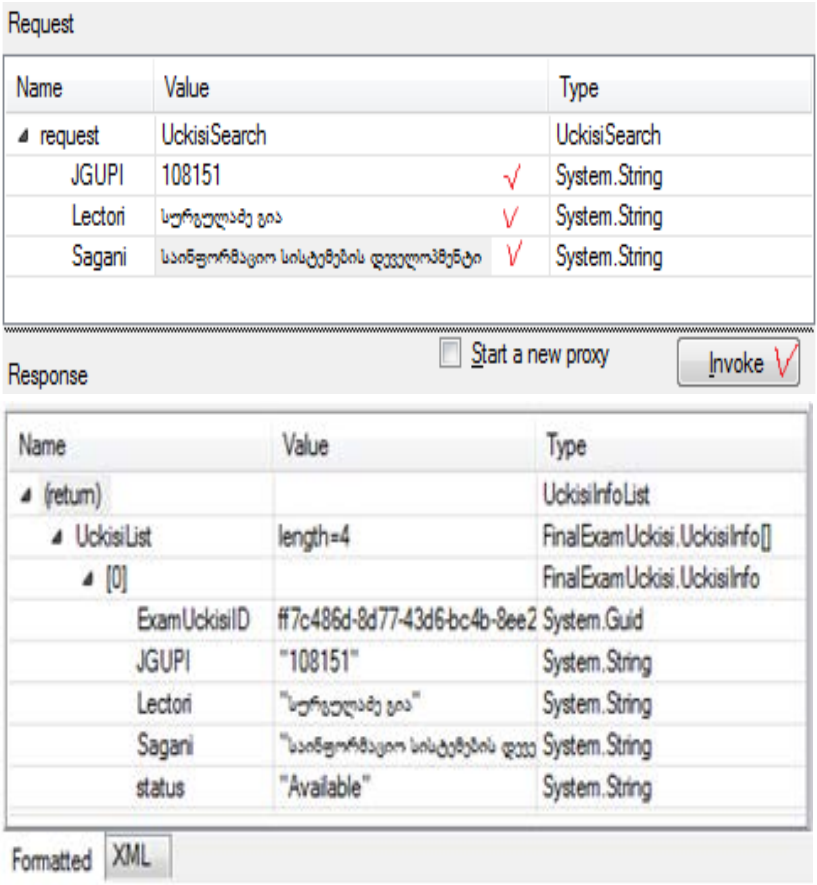


ნახ.3.30. ტესტირების პროცედურა



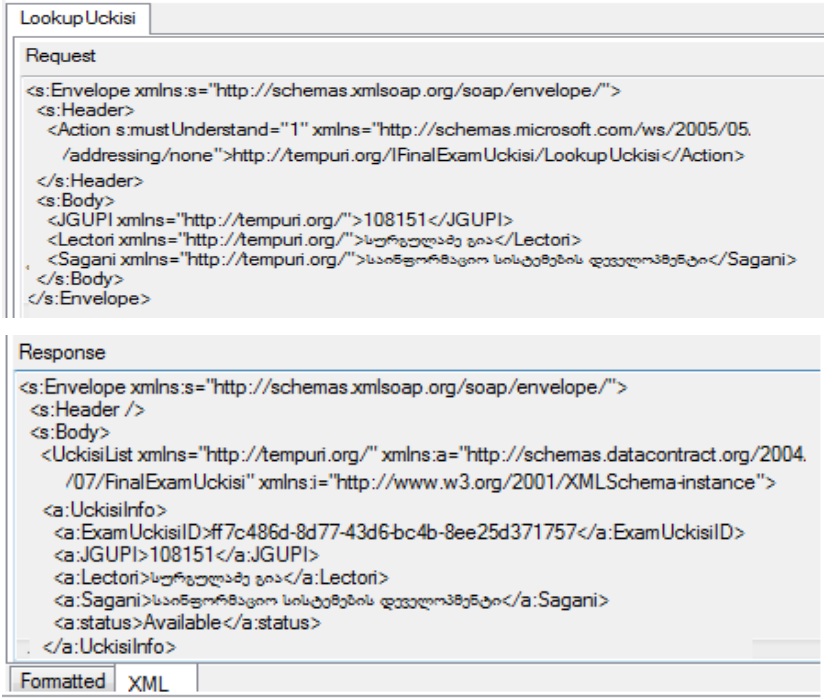
ნახ.3.31. საწყისი მონაცემების შესატანი ფანჯარა

შევიტანოთ კონკრეტული მნიშვნელობები Lectori, JGUPI, Sagani და ავამოქმედოთ Invoke ღილაკი. ცხრილებში გამოჩნდება შედეგები (ნახ.3.32).



ნახ.3.32. WCF Client Test უტილიტით სერვისის ტესტირების შედეგების ნახვა

3.33 ნახაზზე ნაჩვენებია Request და Response ცხრილების შესაბამისი ფაილები XML ფორმატში.



ნახ.3.33. ტესტირების შედეგების XML ტექსტები

3.4. WPF-აპლიკაციის აგება საპრობლემო სფეროსთვის „საფინანსო ბანკი“

3.4.1 ინფორმაციის გაცვლის პროგრამული რეალიზაციის ამოცანა SOA-ისთვის

ბიზნესპროცესების შესრულებისას ერთ-ერთი მნიშვნელოვანი ასპექტია ურთიერთობა (კომუნიკაცია) აპლიკაციებს შორის, კლიენტებსა და სერვერებს შორის, აგრეთვე მუშა-პროცესებსა და ჰოსტ-დანართებს შორის [3].

წინამდებარე პარაგრფში აღწერილი გვაქვს თუ როგორ შეიძლება ბიზნესპროცესების გამოყენებით გამარტივდეს და კოორდინაცია გაეწიოს კომუნიკაციათა სხვადასხვა სცენარებს. აპლიკაციის მაგალითის სახით განიხილება პროექტის აგება საწარმოო ფირმებსა და ბანკებს შორის, რომელშიც მოთხოვნილი ინფორმაცია (მაგალითად, იურიდიული პირის მიერ საკრედიტო განაცხადის წარდგენა) გადაეცემა ბანკს. იგივე აპლიკაციას შეუძლია მოთხოვნის გაგზავნა (სხვა ბანკში) და ასევე პასუხის გაცემა სხვა ორგანიზაციების მოთხოვნებზე [29].

მთავარი ქმედებები, რომლებიც კომუნიკაციისთვის გამოიყენება, არის Send და Receive ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply). ეს ქმედებები გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და სამეთვალყურეოდ [3,20].

ჩვენ ავაგებთ მარტივ WPF-აპლიკაციას (Windows Presentation Foundation), რომელიც გამოიყენებს კომუნიკაციას სხვადასხვა აპლიკაციათა ბიზნესპროცესებს შორის. ბიზნეს-პროცესები შეიძლება განთავსდეს ვებ-სერვისში, რომელიც უზრუნველყოფს იდეალურ საშუალებას მუშა პროცესის გადაწყვეტილების მისაწოდებლად არა-მუშა პროცესის კლიენტებისთვის, როგორიცაა ვებ-აპლიკაციები.

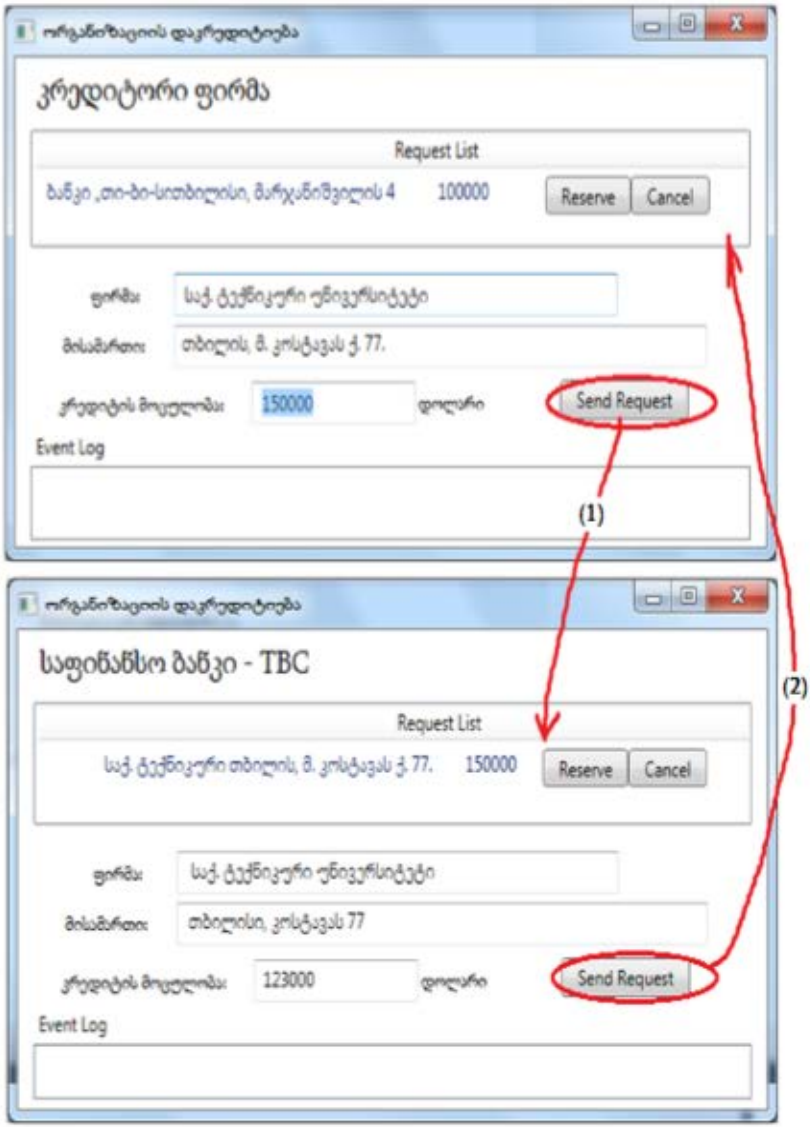
ვებ-სერვისი იღებს მოთხოვნას, ასრულებს მის სათანადო გადამუშავებას და აბრუნებს პასუხს. ეს, ბუნებრივია, სრულდება

Receive და Send ქმედებებით. ვინაიდან ეს აქტიუროზები ინტეგრირებულია Windows Communication Foundation (WCF) -თან, ჩვენ შეგვიძლია ადვილად შევქმნათ WCF სერვისები.

3.4.2. Window Form –ის განსაზღვრა

ავაგოთ მომხმარებელთა ინტერფეისის ფორმა, რომელიც გამოდგება, მაგალითად ბანკებისთვის. მისი მაკეტი მოცემულია 3.34 ნახაზზე. შესაბამისი ფორმა აგებულია XAML ფაილის სახით დიზაინის რეჟიმში, როგორც ზემოთ აღვნიშნეთ, WPF ტექნოლოგიის გამოყენებით. ინტერფეისის დაპროეტქება და პროგრამული რეალიზაცია ხდება Visual Studio.NET Framework 4.0 გარემოში, C#.NET ენის საფუძველზე. გამოიყენება ინტერფეისის დიზაინისთვის XAML და პროგრამის ლოგიკისთვის C# ენები. ქვემოთ, 3.4 ლისტინგში მოცემულია XAML ფაილის ტექსტი. ფორმის ზედა ნაწილში „მოთხოვნების სია“ (Request List) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად, მაგალითად, ბანკში გამოიყენება ველები ფორმის შუაში. აქ მიეთითება: - „ფირმა“, - „მისამართი“ და - მოთხოვნილი „კრედიტის მოცულობა“.

შემდეგ ავამოქმედოთ ღილაკი „მოთხოვნის გაგზავნა“ (Send Request). ქვედა მარცხენა კუთხეში Event Log ასახავს ბიზნესპროცესის შეტყობინებას ისე, როგორც კონსოლის რეჟიმშია.



ნახ.3.3.34. მომხმარებელთა ინტერფეისები

Crediting.xaml ფაილის ლისტინგი ასეთია:

```
<!– – ლისტინგი_3.4 – – –  –>
<Window x:Class="FirmsCrediting.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="ორგანიზაციის დაკრედიტება" Height="480" Width="650"
      Loaded="Window_Loaded" Unloaded="Window_Unloaded">
  <Grid>
      <Label Height="40" HorizontalAlignment="Left" Margin="12,0,0,0"
        Name="lblBranch" FontSize="20" VerticalAlignment="Top"
            Width="462" FontStretch="Expanded"
             FontFamily="Sylfaen">კრედიტორი ფირმა</Label>
      <ListView x:Name="requestList" Margin="12,42,12,5" Height="150"
             VerticalAlignment="Top" ItemsSource="{Binding}">
        <ListView.View>
          <GridView>
            <GridViewColumn Header="Request List" Width="610">
              <GridViewColumn.CellTemplate>
                <DataTemplate>
                  <StackPanel Orientation="Horizontal">
       <TextBlock Text="{Binding Requester.BranchName}" Width="100"/>
       <TextBlock Text="{Binding FirmName}" Width="95"/>
       <TextBlock Text="{Binding Adress}" Width="180"/>
       <TextBlock Text="{Binding CreditQ}" Width="90"/>
       <Button Content="Reserve" Tag="{Binding InstanceID}"
                      Click="Reserve" Width="65"/>
       <Button Content="Cancel"  Tag="{Binding InstanceID}"
                      Click="Cancel" Width="60"/>
                  </StackPanel>
                </DataTemplate>
              </GridViewColumn.CellTemplate>
```

```
            </GridViewColumn>
          </GridView>
        </ListView.View>
    </ListView>
  <Label Height="30" Margin="27,0,0,205" Name="label5"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="73" HorizontalContentAlignment="Right"
          Content="ფირმა:" FontFamily="Sylfaen"></Label>
  <Label Height="30" Margin="27,0,0,176" Name="label2"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="77" HorizontalContentAlignment="Right"
          Content="მისამართი:" FontFamily="Sylfaen"></Label>
  <Label Height="30" Margin="13,0,0,142" Name="label3"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="151" HorizontalContentAlignment="Right"
          Content="კრედიტის მოცულობა:" FontFamily="Sylfaen">
  </Label>
  <TextBox Height="25" Margin="121,0,0,210"  Name="txtFirmName"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="400" /> <TextBox Height="25" Margin="121,0,0,180"
          Name="txtAdress" VerticalAlignment="Bottom"
          HorizontalAlignment="Left" Width="468" />
  <TextBox Height="25" Margin="0,0,329,147" Name="txtCreditQ"
          VerticalAlignment="Bottom" HorizontalAlignment="Right"
          Width="125" />
  <Button Height="23" Margin="477,0,0,150" Name="btnRequest"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="98" Click="btnRequest_Click">Send Request</Button>
  <Label Height="27" HorizontalAlignment="Left" Margin="11,0,0,121"
          Name="label4" VerticalAlignment="Bottom" Width="76">
          Event Log</Label>
```

```
    <ListBox Margin="12,0,12,12" Name="lstEvents" Height="111"
             VerticalAlignment="Bottom" FontStretch="Condensed"
             FontSize="10" />
     <Label Content="დოლარი" Height="28" HorizontalAlignment="Left"
             Margin="302,269,0,0"  Name="label1" VerticalAlignment="Top"
             Width="67" FontFamily="Sylfaen" />
  </Grid>
</Window>
```

ფორმის ზედა ნაწილში „მოთხოვნების სია“ (Request List) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად მაგალითად, ბანკში გამოიყენება ველები ფორმის შუაში. აქ მიეთითება „ფირმა“, „მისამართი“ და მოთხოვნილი „კრედიტის მოცულობა“, შემდეგ გაგზავნის ღილაკი „მოთხოვნის გაგზავნა“ (Send Request). ქვედა მარცხენა კუთხეში Event Log ასახავს ბიზნესპროცესის შეტყობინებას ისე, როგორც კონსოლის რეჟიმშია.

3.4.3. სერვისის პროგრამული რეალიზაცია

ჩვენი სისტემის ClientService.cs კოდის რეალიზაცია ნაჩვენებია 3.5 ლისტინგში.

```
// ----- ლისტინგი 3.5 --- ClientService.cs ---------------
using System;
using System.ServiceModel;   // !!!
namespace FirmsCrediting
{
   public class ClientService : ICreditReservation
   {
      public void RequestCredit(CreditingRequest request)
      {
         ApplicationInterface.RequestCredit(request);
      }
      public void RespondToRequest(CreditingResponse response)
```

```
      {
         ApplicationInterface.RespondToRequest(response);
      }
   }
}
```

ეს რეალიზაცია იყენებს ApplicationInterface სტატიკურ კლასს, რომელიც უკვე შექმნილია ჩვენს მიერ. ყოველი მეთოდი უბრალოდ იძახებს ApplicationInterface კლასის შესაბამის მეთოდს.

ApplicationInterface.cs ფაილი მოცემულია 3.6 ლისტინგში.

```
//  ---- ლისტინგი 3.6 --- ApplicationInterface.cs ფაილისთვის ----
using System;
using System.Windows.Controls;
using System.Activities;
namespace FirmsCrediting
{
  public static class ApplicationInterface
   {
     public static MainWindow _app { get; set; }
     public static void AddEvent(String status)
      {
        if (_app != null)
         {
           new ListBoxTextWriter(_app.GetEventListBox()).WriteLine(status);
          }
     }
  public static void RequestCredit(CreditingRequest request)
   {
      if (_app != null)
         _app.RequestCredit(request);
   }
  public static void RespondToRequest(CreditingResponse response)
   {
      if (_app != null)
```

```
        _app.RespondToRequest(response);
    }
  public static void NewRequest(CreditingRequest request)
   {
     if (_app != null)
       _app.AddNewRequest(request);
  }
 }
}
```

ეს მეთოდები, თავის მხრივ, იძახებს შესაბამის მეთოდებს აპლიკაციაში სტატიკური მიმთითებლის გამოყენებით. საჭირო იქნება ამ მეთოდების რეალიზება Crediting.xaml.cs ფაილში.

3.4.4. ServiceHost -ის რეალიზაცია

აპლიკაციისთვის აუცილებელია ServiceHost-ის რეალიზაცია შემავალი შეტყობინებების მისაღებად. იგი პროექტის Crediting.xaml.cs ფაილში თავსდება კონსტრუქტორის წინ კლასის წევრის სახით (ლისტინგი_3.7).

```
//-- ლისტინგი_3.7 ------
public partial class MainWindow : Window
  {
    private ServiceHost _sh;  // !!!
    public MainWindow()
    {   InitializeComponent();
        ApplicationInterface._app = this;
    }
  ...
```

ServiceHost იწყება მაშინ, როცა ფანჯარა ჩატვირთულია და იხურება, როცა ფანჯარა ამოტვირთულია. მეთოდების დამატება ნაჩვენებია 3.8 ლისტინგში MainWindow კლასისთვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

```
 // --- ლისტინგი 3.8 ------------------
private void Window_Loaded(object sender, RoutedEventArgs e)
 {
  // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და მისი
  // ქსელური მისამართი
 Configuration config = ConfigurationManager
                  .OpenExeConfiguration(ConfigurationUserLevel.None);
 AppSettingsSection app = (AppSettingsSection)config
                    .GetSection("appSettings");
 string adr = app.Settings["BranchAddress"].Value;
      // ფილიალის სახელის გამოტანა ფორმაზე
      lblBranch.Content = app.Settings["Branch Name"].Value;

      // ServiceHost-ის შექმნა
      _sh = new ServiceHost(typeof(ClientService));

     // დასასრულის წერტილის (Endpoint) დამატება
     string szAddress = "http://localhost:" + adr + "/ClientService";
     System.ServiceModel.Channels.Binding bBinding =
           new BasicHttpBinding();
    _sh.AddServiceEndpoint(typeof(ICreditReservation),bBinding,
                  szAddress);
    // ServiceHost-ის გახსნა შეტყობინებების მისაღებად (listen)
   _sh.Open();
 }
private void Window_Unloaded(object sender, RoutedEventArgs e)
 {
   // service host-ის დატოვება
     _sh.Close();
 }
```

მოვლენის დამმუშავებელი Loaded ხსნის კონფიგურაციის ფაილს და ათავსებს ფილიალის სახელს lblBranch -მართვის ელემენტში, ამიტომაც ფორმა ასახავს ფირმის სახელს. შემდეგ იქმნება ServiceHost თანამგზავრი (passing) ClientService კლასის. შემდეგ იგი აკონფიგურირებს დასასრულის წერტილს ServiceHost-თვის, იყენებს რა ცნობილი მისამართის, მიბმის და კონტრაქტის სამეულს.

Unloaded მოვლენის დამმუშავებელი უბრალოდ ხურავს ServiceHost-ს, ასე რომ მეტი აღარ მოხდება შეტყობინებების მიღება.

აპლიკაციის ამუშავება: საჭიროა აპლიკაციის რამდენიმე კოპიოს ერთად გაშვება, თითოეული თავისი კონფიგურაციის ფაილის ვერსიით. თავიდან საჭიროა F6 კლავიშის ამოქმედება Solution-ის (გადაწყვეტის) აღსადგენად და კომპილატორის შენიშვნების აღმოსაფხვრელად.

შევქმნათ ახალი ფოლდერი BankRequest -ფოლდერის ქვეშ, რომელიც იძახებს ბანკებს. შემდეგ დავაკოპიროთ ბანკის ფოლდერში ფაილები, რომლებიც ზემოთ შევქმენით;

```
FirmsCrediting.exe    // Application
FirmsCrediting.exe.config   // XML Configuration file
FirmsCrediting.pdb   // program debug database
```

სისტემის ამუშავების შემდეგ გვექნება ორი მუშა ფანჯარა, მაგალითად, ერთი კრედიტორი ფირმის, მეორე საფინანსო ბანკის. მათ შორის შესაძლებელი იქნება ინფორმაციის და შეტყობინებების გაცვლა, რისი მიღწევაც გვინდოდა (ნახ.3.34).

3.5. WPF-აპლიკაციის აგება საპრობლემო სფეროსთვის „სატრანსპორტო გადაზიდვები“

3.5.1 მულტიმოდალური გადაზიდვების მართვის საინფორმაციო სისტემის აგების ამოცანა

განიხილება ტვირთების მულტიმოდალური გადაზიდვების მართვის ავტომატიზებული სისეტმის მონაცემთა ბაზის დაპროექტების, მისი პროგრამული რეალიზაციის და მომხმარებელთა ინტერფეისების აგების საკითხები დაპროექტების CASE - და დაპროგრამების ჰიბრიდული ტექნოლოგიებით [8].

წინამდებარე პარაგრაფში შემოთავაზებულია მულტიმოდალური გადაზიდვების (გემი, რკინიგზა, ავტო- და საჰაერო ტრანსპორტი) საპრობლემო სფეროს კონცეპტუალური სქემები კლიენტის (ტვირთის მფლობელი), ტვირთის (გადაზიდვის ობიექტი) და მიმწოდებელის (გადამზიდავი) ცხრილებით, ობიექტ-როლური და არსთა დამოკიდებულების მოდელირების (ORM/ERM) ინსტრუმენტებით Visual Studio.NET გარემოში და Ms SQL Server პაკეტით [22,23]. აგებულია მონაცემთა ბაზის განახლების ფუნქციების ინტერფეისი დაპროგრამების (WPF, C#, XAML) ინტეგრირებულ გარემოში [1].

მულტიმოდალური გადაზიდვების პროცესი, რომლის ძირითადი მიზანი ტვირთების ტრანსპორტირებაა მიმწოდებლიდან დამკვეთამდე, არის მომსახურების განაწილებული სისტემა. მარტივად რომ წარმოვიდგინოთ, მიმწოდებელი (Supplier_ID) აგზავნის ტვირთს (Freight_ID) დამკვეთის (Cleint_ID) მისამართზე (Client_Address) [22].

როგორც ცნობილია, კლენტსა (ტვირთის მფლობელი) და მიმწოდებელს (გადამზიდავი) შორის ხელშეკრულებას აფორმებს ექსპედიტორი (შუამავალი), რომელსაც გააჩნია საჭირო ინფორმაცია ადგილობრივი და საერთაშორისო გადაზიდვების

აგენტების, მარშრუტებისა და შესაბამისი ფასების შესახებ (ამ უკანასკნელის ცვლილებების შესახებაც) და სხვა.

ქვემოთ მოცემული გვაქვს მულტიმოდალური გადაზიდვების პროცესის ინფრასტრუქტურის ძირითადი ობიექტების და მათი თვისებების (ატრიბუტების) სემანტიკური აღწერა, რაც მომავალში გამოყენებულ იქნება ავტომატიზებული სისტემის მონაცემთა ბაზის ასაგებად [30].

ტვირთი: იდენტიფიკატორი, ტიპი, მდგომარეობა, შეფუთვის ტიპი, ერთეულის ზომები (სიგრძე, სიგანე, სიმაღლე), ერთეულის მოცულობა, ჯამური მოცულობა, ერთეულის წონა, ერთეულის რაოდენობა, ჯამური წონა, უსაფრთხოობა, საბაჟო კოდი, გამგზავნი, მიმღები, საწყისი მდებარეობა, საბოლოო მდებარეობა და სხვა.

კლიენტი: იდენტიფიკატორი, დასახელება/ვინაობა, იურიდიული/ფიზიკური პირი, მისამართი, ტელეფონი, ელ_მისამართი და სხვა;

მიმწოდებელი: იდენტიფიკატორი, დასახელება, იურიდიული/ფიზიკური პირი, მისამართი, ტელეფონი, ელ_მისამართი, ფაქსი, ტრანსპორტის სახე და სხვა;

გემი: იდენტიფიკატორი, ტიპი, ამწეკრანით/უკრანო, მდგომარეობა, სასაწყობო ლიმიტი, ტვირთამწეობა, ტვირთმოცულობა, ადგილმდებარეობა და სხვა;

თვითმფრინავი: იდენტიფიკატორი, ტიპი, მდგომარეობა, ტვირთმოცულობა, გადასაზიდი ერთეულის დასაშვები ზომები (სიგრძე, სიგანე, სიმაღლე) ადგილმდებარეობა და სხვა.

ავტოტრანსპორტი: იდენტიფიკატორი, ტიპი, მდგომარეობა, ტვირთმოცულობა, გადასაზიდი ერთეულის დასაშვები ზომები (სიგრძე, სიგანე, სიმაღლე), გადასაზიდი ერთეულის დასაშვები წონა, მაქსიმალური დატვირთვა, ადგილმდებარეობა და სხვა.

სარკინიგზო სატვირთო ვაგონი: იდენტიფიკატორი, ტიპი, ტვირთამწეობა, მოცულობა, დასაშვები დატვირთვა, ადგილმდებარეობა, მიმწოდებლის იდენტიფიკატორი, მდგომარეობა და სხვა;

საწყობი: იდენტიფიკატორი, სახე, ფართობი, სართული, დაკავებულობის პროცენტი, დასაშვები დატვირთვა, ადგილმდებარეობა, მისამართი, მიკუთვნება რაიონზე და სხვა;

გადაზიდვის ხელშეკრულება კლიენტთან: იდენტიფიკატორი, საწყისი მდებარეობა, თარიღი_1, საბოლოო მდებარეობა, თარიღი_2, გადაზიდვის ღირებულება, გადახდილი თანხა, გადახდის_თარიღი, მდგომარეობა და სხვა.

3.5.2. მონაცემთა ბაზის დაპროექტება და რეალიზაცია

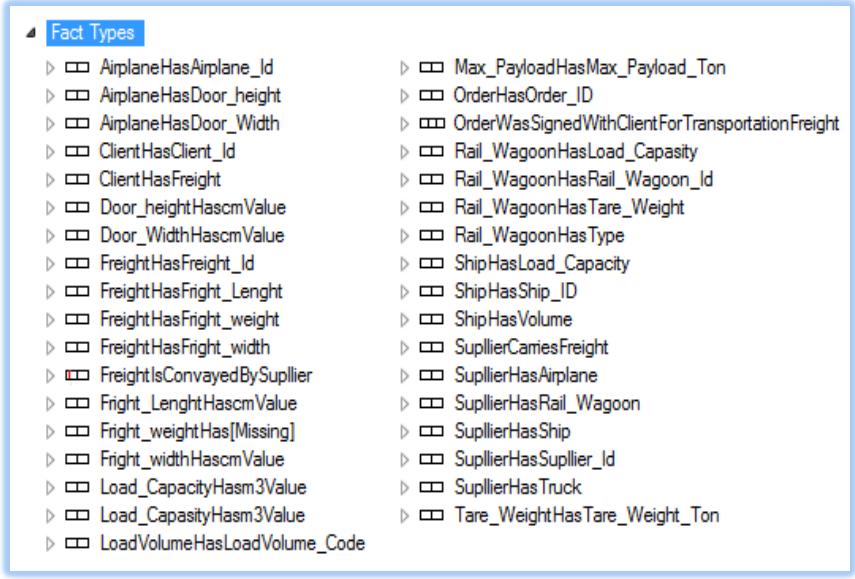
მულტიმოდალური გადაზიდვების საპრობლემო სფეროს მონაცემთა ბაზის ასაგებად საჭიროა მისი ობიექტების, ობიექტთაშორისი რელაციური კავშირების (პრედიკატების) და კონკრეტული ინფორმაციის გადატანა ბაზაში. ამისათვის ჩვენ ვიყენებთ ობიექტ-როლური მოდელირების CASE ინსტრუმენტულ საშუალებას, როგორიცაა Natural ORM Archchitect [14]. ამ სფეროში არსებული ფაქტების აღწერა ხორციელდება კატეგორიალური მიდგომისა (სალაპარაკო ენის გრამატიკული წესები) და ალგებრულ-ლოგიკური თეორიის კანონების საფუძველზე [15]. მაგალითად,

- **f1: კლიენტს აქვს ტვირთი;**
- **f2: კლიენტს აქვს იდენტიფიკატორი;**
- **f3: ტვირთს აქვს იდენტიფიკატორი;**
- **f4: ტვირთს აქვს გადასატზნი მისამართი;**
- **f5: მიმწოდებელს აქვს ტრანსპორტი;**

და ა.შ.

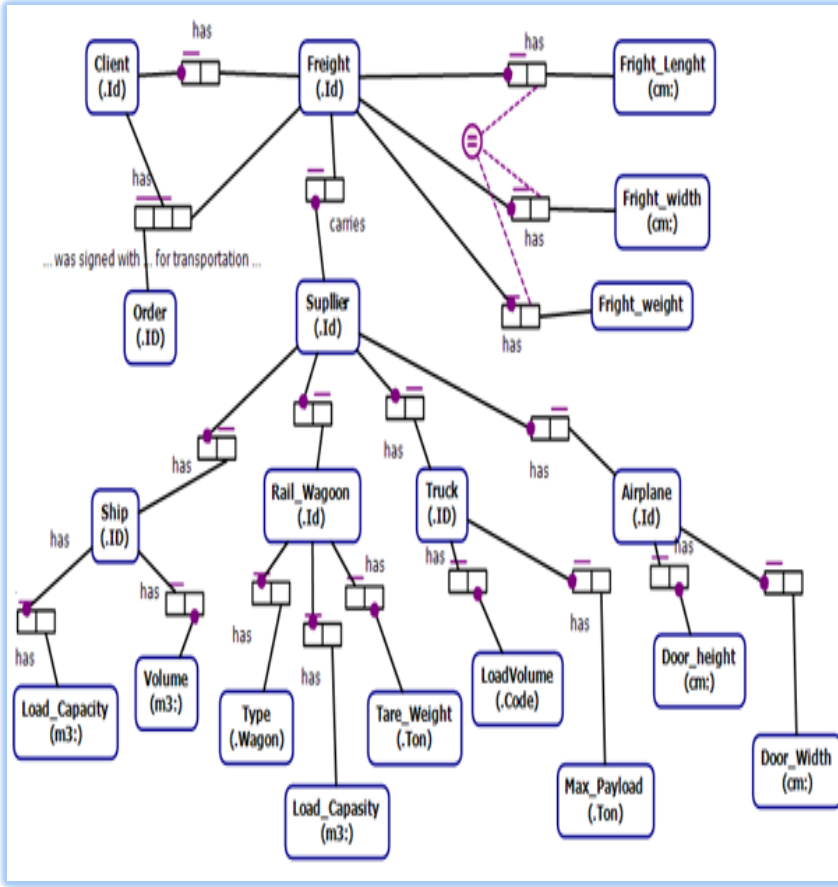
1-ელ ცხრილში მოცემულია ჩვენი ობიექტების შესაბამისი ფაქტების აღწერის ფრაგმენტი NORMA გარემოში.

ფაქტის ტიპები (პრედიკატები) ცხრ.3.1



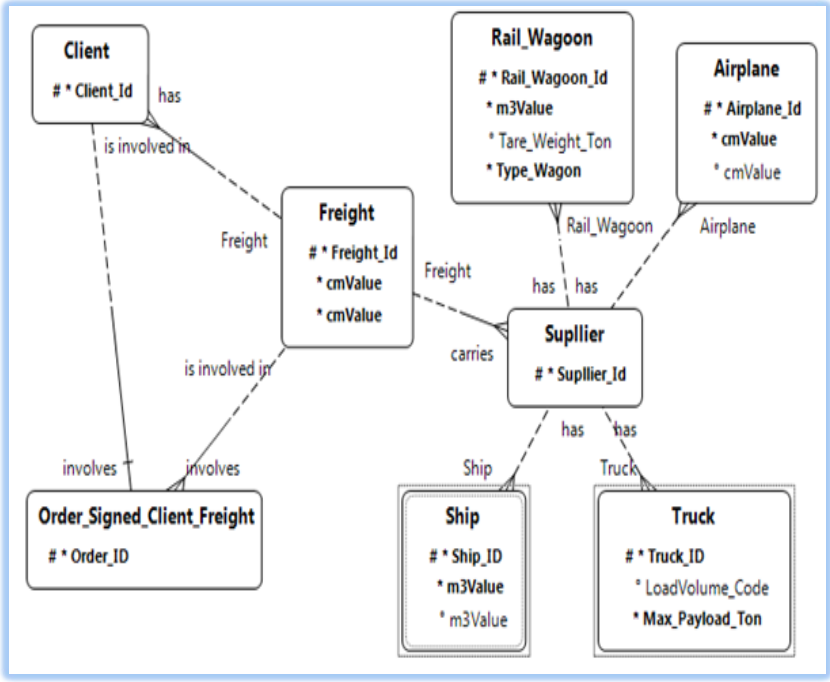
მულტიმოდალური გადაზიდვების საპრობლემო სფეროს ფაქტების აღწერისა და მათი Visual Studio.NET + NORMA ინტეგრირებულ სამუშაო გარემოში გადატანის შემდეგ ვღებულობთ 3.35 ნახაზზე ნაჩვენებ კონცეპტუალურ სქემას, რომელიც ობიექტ-როლური მოდელია.

დიდი მართკუთხედებით აისახება ობიექტები (მაგალითად, Client, Freight, Airplane და სხვა) და მათი თვისებები (Load_Capacity, Volume, Freight_Lenght და სხვა), პატარა მართკუთხედებით კი – პრედიკატები მათ შორის (მაგალითად,,,Ship has Load_Capacity" და ა.შ.). აქ „მრგვალთავა" კავშირის ხაზებით და პრედიკატებზე `ხაზგასმით" გვეძლევა დამატებითი ინფორმაცია ობიექტებსშორის მრავლობითი კავშირების შესახებ, როგორიცაა მაგალითად, 1:1, 1:N და M:N.



ნახ.3.35. საპრობლემო სფეროს კონცეპტუალური სქემა: ობიექტ-როლური მოდელი

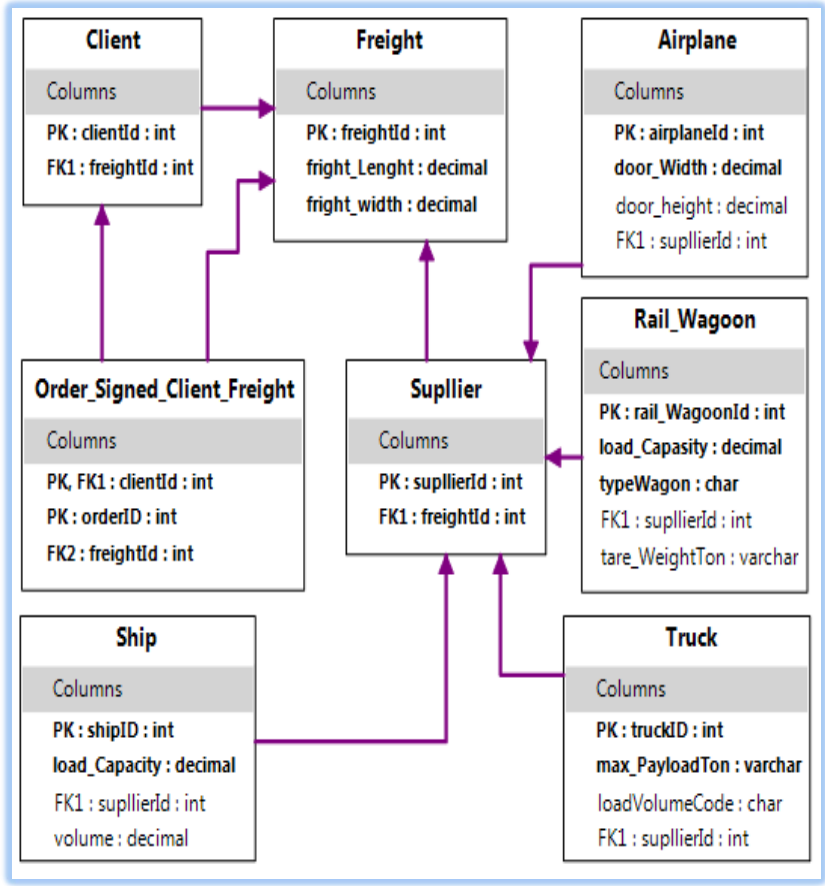
აქვე შესაძლებელია მივიღოთ, ე.წ. რიჩარდ ბარკერის დიაგრამა, რომელშიც ასეთი კავშირები ობიექტებს შორის უკეთესად ჩანს (ნახ.3.36). იგი ჩვეულებრივი არსთა-დამოკიდებულების მოდელია, რომელსაც იყენებენ Oracle CASE მიმდევრები [17].



ნახ.3.36. ბარკერის დიაგრამა (Barker ER view)

ჩვენი შემდეგი ბიჯი დაკავშირებულია OღM დიაგრამით მიღებული კონცეპტუალური სქემიდან ეკვივალენტური ER მოდელის დაპროექტებასთან. ეს პროცესი დიალოგურ რეჟიმში ხორციელდება, სადაც მომხმარებელს შეუძლია სისტემის მიერ შემოთავაზებულ სქემაში შეიტანოს საჭირო ცვლილებები.

3.37 ნახაზზე ნაჩვენებია მულტომოდალური გადაზიდვების საპრობლემო სფეროს ER სქემა. იგი ახლოსაა რელაციური ბაზების დაპროექტების ობიექტ-ორიენტირებული მიდგომის ტრადიციულ დიაგრამებთან, რომლებიც აიგება MsVisio, Enterprise Architect, Rational Rose და სხვა ინსტრუმენტებით [27].



ნახ.3.37. კონცეპტუალური სქემა (ER Model)

ჩვენ აღვწერეთ კონცეპტუალური სქემის ავტომატიზებული ფორმირების პროცესი ORM->ERM. აქვე შეიძლება აღვნიშნოთ, რომ 3.37 ნახაზზე მოცემულია მხოლოდ ფრაგმენტი და მისი სრული

ვერსიის მიღება შესაძლებელია პირველ ეტაპზე ფაქტების თანდათანობით დამატებით. 3.2 ცხრილში მოცემულია სისტემის მიერ ფორმირებული მულტიმოდალური გადაზიდვების საპრობლემო სფეროს კონცეპტუალური სქემის შესაბამისი ფაქტების ვერბალიზაციის ლისტინგი.

ფაქტების ვერბალიზაცია ცხრ.3.2

Freight is an entity type. Reference Scheme: Freight has Freight_Id. Reference Mode: .Id. Fact Types: Freight has Freight_Id. Freight is convayed by Suplier. Supllier carries Freight. Freight has Freight_Lenght. Freight has Freight_width. Freight has Freight_weight. Client has Freight. Order was signed with Client for transportation Freight. **Client** is an entity type. Reference Scheme: Client has Client_Id. Reference Mode: .Id. Fact Types: Client has Client_Id. Client has Freight. Order was signed with Client for transportation Freight. Each Client has exactly one Freight. It is possible that more than one Client has the same Freight.	**Supplier** is an entity type. Reference Scheme: Supllier has Supplier_Id. Reference Mode: .Id. Fact Types: Supllier has Supplier_Id. Freight is convayed by Supplier. Supplier carries Freight. Supplier has Ship. Supplier has Rail_Wagoon. Supplier has Truck. Supplier has Airplane. **Order** is an entity type. Reference Scheme: Order has Order_ID. Reference Mode: .ID. Fact Types: Order has Order_ID. Order was signed with Client for transportation Freight. For each Order and Client,that Order was signed with that Client for transportation at most one Freight. This association with Order, Client provides the preferred identification scheme for Order was signed with Client for Transportation Freight.

Truck is an entity type. Reference Scheme: Truck has Truck_ID. Reference Mode: .ID. Fact Types: Truck has Truck_ID. Truck has LoadVolume. Truck has Max_Payload. Suplier has Truck. Each Suplier has some Truck. For each Truck, at most one Suplier has that Truck. It is possible that the same Suplier has more than one Truck.	**Ship** is an entity type. Reference Scheme: Ship has Ship_ID. Reference Mode: .ID. Fact Types: Ship has Ship_ID. Supplier has Ship. Ship has Load_Capacity. Ship has Volume. Each Supplier has some Ship. For each Ship, at most one Suplier has that Ship. It is possible that the same Suplier has more than one Ship.
Airplane is an entity type. Reference Scheme: Airplane has Airplane_Id. Reference Mode: .Id. Fact Types: Airplane has Door_height. Airplane has Door_Width. Airplane has Airplane_Id. Suplier has Airplane. Each Suplier has some Airplane. For each Airplane, at most one Suplier has that Airplane. It is possible that the same Suplier has more than one Airplane.	**Rail_Wagoon** is an entity type. Reference Scheme: Rail_Wagoon has Rail_Wagoon_Id. Reference Mode: .Id. Fact Types: Rail_Wagoon has Rail_Wagoon_Id. Supplier has Rail_Wagoon. Rail_Wagoon has Type. Rail_Wagoon has Load_Capasity. Rail_Wagoon has Tare_Weight. Each Suplier has some Rail_Wagoon. For each Rail_Wagoon, at most one Suplier has that Rail_Wagoon. It is possible that the same Supplier has more than one Rail_Wagoon.

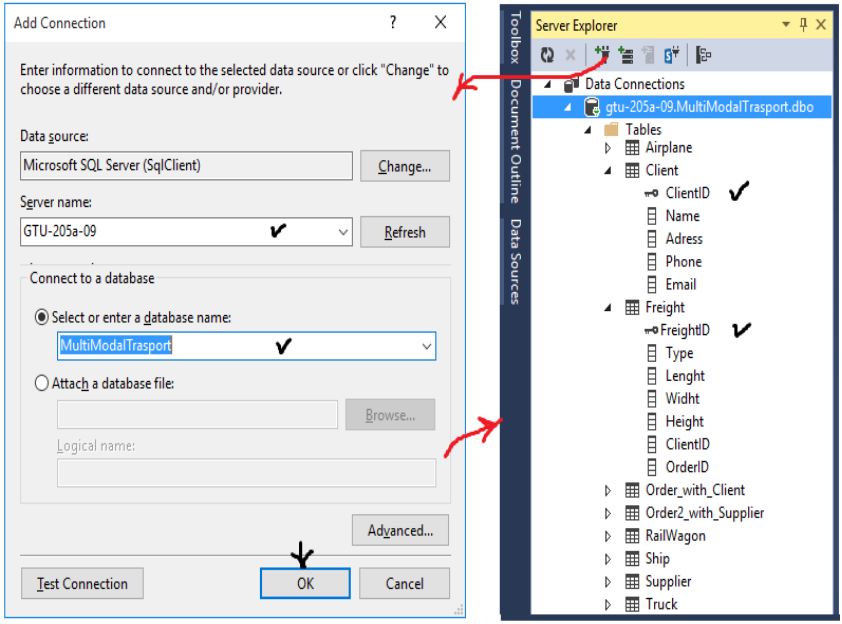
მომდევნო ბიჯზე, მიღებული ER-მოდელის შესაბამისად, სისტემა გვაძლევს DDL-ფაილს (მონაცემთა აღწერის ენა), რომელიც ჩვენ შემთხვევაში გამოიყენება SQL Server ბაზის ასაგებად [30].

3.5.3. მონაცემთა ბაზასთან მუშაობის ინტერფეისის აგება

მულტიმოდალური გადაზიდვების ბაზის დაპროექტებისა და მისი DLL-ფაილით SQL Server 2012-ში რეალიზაციის შემდეგ, საჭიროა აიგოს მომხმარებლის ინტერფეისის პროგრამა Visual Studio.NET 2013 ინტეგრირებულ გარემოში დაპროგრამების ჰიბრიდული ტექნოლოგიის გამოყენებით, WPF (Windows

Presentation Foundation) [8]. ჰიბრიდული ტექნოლოგიის კონცეფცია გულისხმობს, რომ ამ მეთოდოლოგიით შესაძლებელია როგორც ვინდოუსის, ასევე ვებ-აპლიკაციების აგება (შედეგები აისახება ვინდოუსის ფორმაზე ან რომელიმე ინტერნეტ ბრაუზერში).

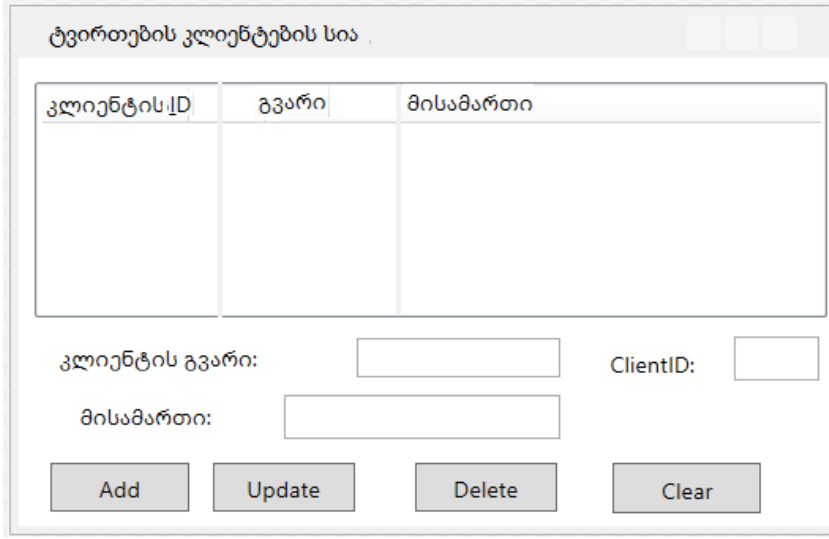
თავდაპირველად საჭიროა ახალი პროექტის შექმნა Visual Studio.NET-ში, შემდეგ კი ჩვენი მონაცემთა ბაზის მიერთება ამ პროექტთან. 3.39 ნახაზზე ნაჩვენებია ბაზის მიერთების (Data Connections) პროცედურა. განახლდება სერვერის სახელი (მაგალითად, GTU-205a-09) და აირჩევა მასში მონაცემთა ბაზის სახელი (MultiModalTransport), ბოლოს OK და შედეგად მივიღებთ ნახაზის Server Explorer-ში და Airplane, Client, Freight სხვა ცხრილებს.



ნახ.3.39. მონაცემთა ბაზის დაკავშირება C#.NET პროგრამასთან

მონაცემთა ბაზის ცხრილებთან მუშაობის ინტერფეისის სქემა წინასწარ შეთანხმებულია პროგრამული სისტემის მომხმარებელთან. მომხმარებელს, რომელმაც არ იცის მონაცემთა ბაზასთან მუშაობა დაპროგრამების ენების გამოყენებით (მაგალითად, DDL და DML), მაშინ მისთვის უნდა აიგოს დიალოგში სამუშაო ინტერფეისი, რომელიც ადვილად შეასრულებს ბაზაში მონაცემების ძებნის (Select) ან ცხრილების განახლების ოპერაციებს (Add, Update, Delete).

3.40 ნახაზზე ნაჩვენებია ასეთი ინტერფეისის ერთ-ერთი ვარიანტი, რომელშიც შესაძლებელია მომხმარებელმა, როგორც ბაზის ადმინისტრატორმა, განახორციელოს თავისი ფუნქციები (რეალურ მონაცემთა ბაზასთან უშუალოდ წვდომის გარეშე), ამჯერად, მაგალითად, კლიენტების ბაზასთან.



ნახ.3.40. ინტერფეისის საილუსტრაციო მაგალითის მაკეტი

ამ ფორმის დიზაინი აიგება Visual Studio.NET გარემოში WPF ინსტრუმენტების პანელის და XAML-ენის გამოყენებით. შესაბამისი კოდის ფრაგმენტი მოცემულია 3.9 ლისტინგში.

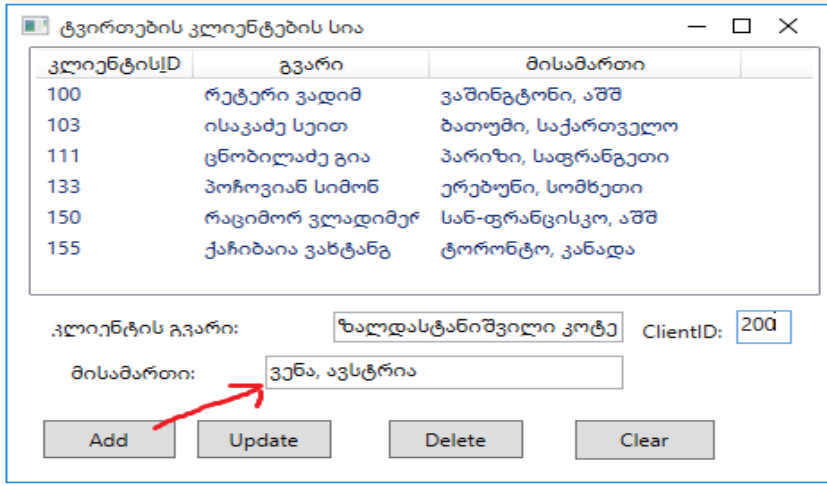
```
<-- ლისტინგი 3.9 ---- XAML ფაილი ------>
<Window x:Class="WpfAppSQL_ListView.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title ="ტვირთების კლიენტების სია"
          Height="375" Width="520" Loaded="Window_Loaded"
               Background="White">
      . . .
     <GridView>
        <GridViewColumn Header="კლიენტის_ID"
            DisplayMemberBinding=
                    "{Binding Path=ClientID}"></GridViewColumn>
        <GridViewColumn Header="გვარი" DisplayMemberBinding=
                    "{Binding Path=Name}"></GridViewColumn>
        <GridViewColumn Header="მისამართი"
                              DisplayMemberBinding=
                    "{Binding Path=Adress}"></GridViewColumn>
    </GridView>
      . . .
</Window>
```

ინტერფეისის ფორმაზე ჩანს LisBox სამი სვეტით, კლიენტების სიაში რამდენიმე მონაცემის გამოსატანად, სამი TextBox, საძებნი ან დასამატებელი სტრიქონის მონაცემების ჩასაწერად და ოთხი Button, ბაზაში სხვადასხვა ფუნქციების შესასრულებლად (Add, Update, Delete). Clear ღილაკი გამოიყენება მონაცემთა შესატანი ტექსტბოქსების გასაწმენდად.

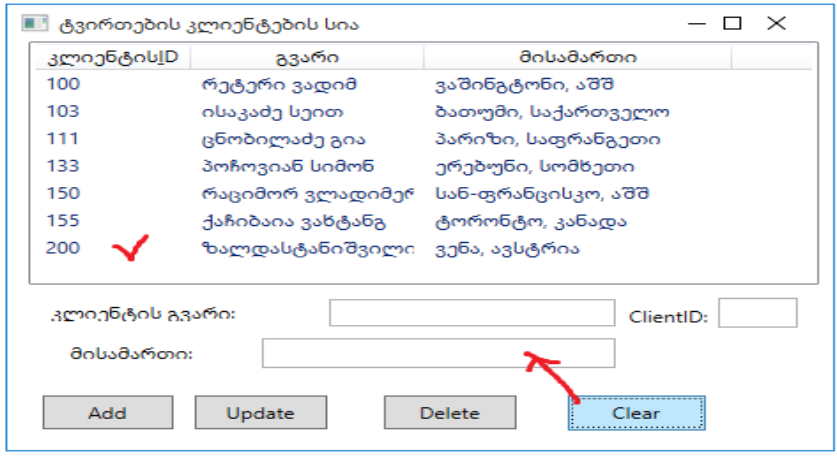
საჭიროა პროგრამის მუშაობის ლოგიკის დაპროგრამება C#.NET ენაზე, კერძოდ უნდა დაიწეროს მეთოდები, რომლებიც მოემსახურება შესაბამისი ღილაკების ამოქმედებით

ინიციალიზირებული მოვლენების დამუშავებას. განვიხილოთ მაგალითები.

დავამატოთ მონაცემთა ბაზას ახალი კლიენტი (ნახ.3.41-ა). უნდა შეივსოს კლიენტის_გვარი, კლიენტის ID და მისამართი.



ნახ.3.41-ა. ახალი ჩანაწერის დამატება (Add ღილაკით)



ნახ.3.41-ბ. ახალი ჩანაწერი დაემატა და ტექსტბოქსები გაიწმინდა

Add – ღილაკის პროგრამის ტექსტი მოცემულია 3.10 ლისტინგში.

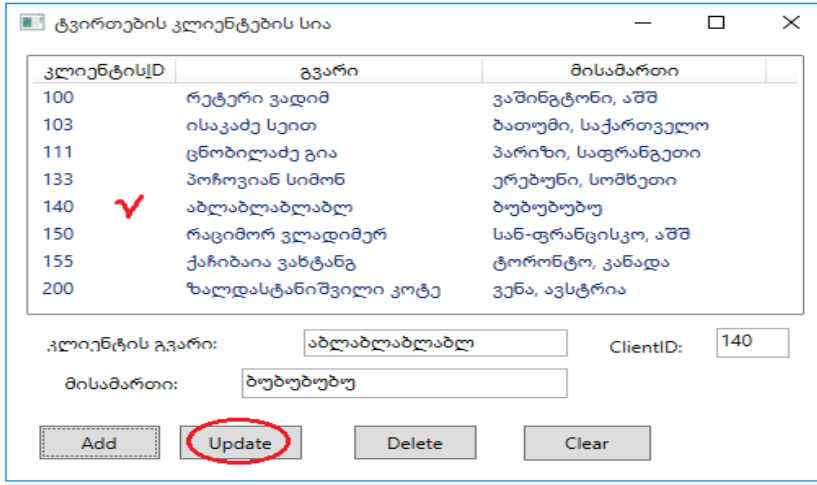
```
// --- ლისტინგში_3.10 – Add ----------
private void btnAdd_Click(object sender, RoutedEventArgs e)
 {
   string Name = textBox1.Text;
   string Adress = textBox2.Text;
   string ClientID = textBox3.Text;
   SqlConnection con = new SqlConnection (@"Data Source=GTU-205a-09;
      Initial Catalog=MultiModalTrasport;Integrated Security=True");
   con.Open();
   SqlCommand comm = new SqlCommand("insert into Client(Name,Adress,
       ClientID) values(@Name, @Adress, @ClientID)", con);

   comm.Parameters.AddWithValue("@Name", textBox1.Text);
   comm.Parameters.AddWithValue("@Adress", textBox2.Text);
   comm.Parameters.AddWithValue("@ClientID", textBox3.Text);
   comm.ExecuteNonQuery();
   con.Close();
   ShowData();
}
```

შენიშვნა: SqlConnection სტრიქონით ხდება MultiModalTrasport მონაცემთა ბაზასთან ავტომატური დაკავშირება, ხოლო SqlCommand–ით ხორციელდება Insert ოპერაციის შესრულება. აქ ShowData()მეთოდია, რომელიც უზრუნველყოფს შედეგების ასახვას ინტერფეისის ფორმაზე; მისი შესაბამისი C#- პროგრამის კოდი ნაჩვენებია 3.11 ლისტინგში.

```
//--- ლისტინგი_3.11 --- ShowData() ----
public void ShowData()
 {
  SqlConnection con = new SqlConnection(@"Data Source=GTU-205a-09;
       Initial Catalog=MultiModalTrasport;Integrated Security=True");
  con.Open();
  SqlCommand comm = new SqlCommand("Select ClientID, Name, Adress
                                                  from Client", con);
  DataTable dt = new DataTable();
```

```
  SqlDataAdapter da = new SqlDataAdapter(comm);
  da.Fill(dt);
  listView1.DataContext = dt.DefaultView;
}
```

მონაცემთა ბაზაზში საჭიროა შეიცვალოს ინფორმაცია, შეცდომაა 140-კლიენტის ჩანაწერში (ნახ.3.42-ა). საჭიროა Update ღილაკის დაპროგრამება, რომლის ტექსტი 3.12 ლისტინგშია მოცემული.

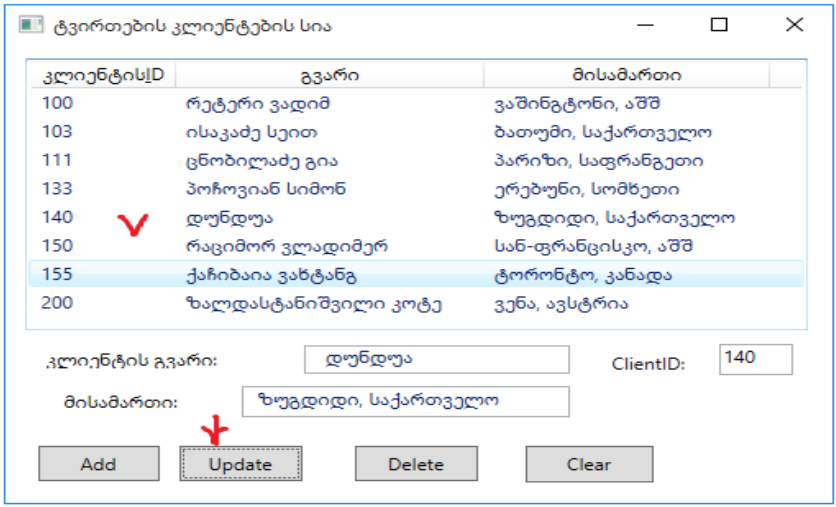


ნახ.3.42-ა. მონაცემთა შეცვლა – ბაზის განახლება (Update ღილაკით)

```
//---  ლისტინგი 3.12 ---  Update ----------
private void btnUpdate_Click(object sender, RoutedEventArgs e)
 {
  if (listView1.SelectedItems.Count > 0)
   {
    DataRowView drv = (DataRowView)listView1.SelectedItem;
    string id = drv.Row[0].ToString();
    SqlConnection con = new SqlConnection(@"Data Source=GTU-205a-09;
         Initial Catalog=MultiModalTrasport;Integrated Security=True");
```

```
    con.Open();
    SqlCommand comm = new SqlCommand("update Client set
          Name=@Name,Adress=@Adress where ClientID=@ClientID", con);
   comm.Parameters.AddWithValue("@ClientID", id);
   comm.Parameters.AddWithValue("@Name", textBox1.Text);
   comm.Parameters.AddWithValue("@Adress", textBox2.Text);
   comm.ExecuteNonQuery();
     con.Close();
     ShowData();
  }
}
```

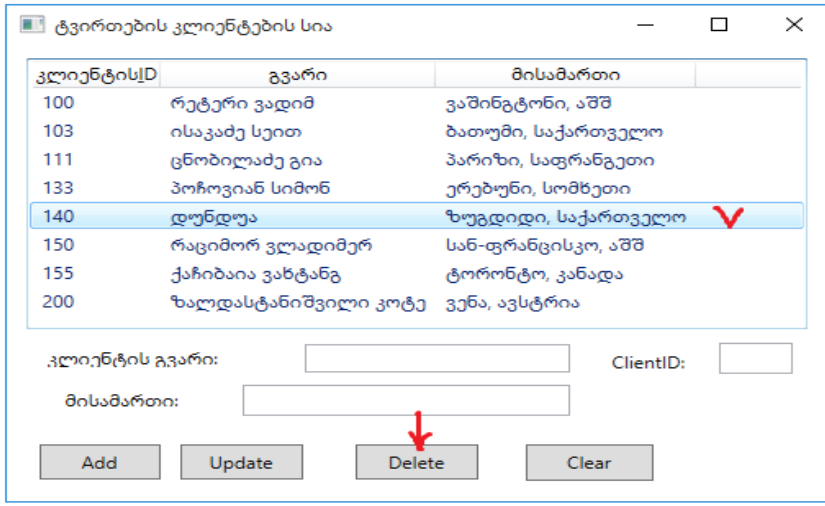
შედეგად მივიღებთ 3.42-ბ ნახაზზე მოცემულ სიტუაციას, ბაზაში 140-ე კლიენტის გვარი და მისამართი განახლებულია.



ნახ.3.42-ბ. მონაცემები შეიცვალა ბაზაში ჩაწერით

დავუშვათ, რომ გვჭირდება კლიენტის მონაცემების წაშლა, რომლის გვარია „დუნდუა”.

უნდა მოვნიშნოთ ეს სტრიქონი (ნახ.3.43-ა).



ნახ.3.43-ა. წასაშლელად მომზადებული სტრიქონი

ამჯერად უნდა დაპროგრამირდეს ელეტე ღილაკი. კოდის ტექსტი მოცემულია 3.13 ლისტინგში.

```
//-----ლისტინგი 3.13---Delete --------
private void btnDelete_Click(object sender, RoutedEventArgs e)
 {
  if (listView1.SelectedItems.Count > 0)
   {
    DataRowView drv = (DataRowView)listView1.SelectedItem;
    string id = drv.Row[0].ToString();
    SqlConnection con = new SqlConnection(@"Data Source=GTU-205a-09;
        Initial Catalog=MultiModalTrasport;Integrated Security=True");
    con.Open();
    SqlCommand comm = new SqlCommand("delete from Client where
                          ClientID=@ClientID", con);
     comm.Parameters.AddWithValue("@ClientID", id);
     comm.ExecuteNonQuery();
     ShowData();
 } }
```

ახალი მონაცემების შეტანის დროს Add ღილაკის ამოქმედების შემდეგ თუ ტექსტბოქსებში დარჩა უკვე შეტანილი მონაცემები, მაშინ ისინი უნდა გასუფთავდეს, გამზადდეს ახალი მონაცემების შესატანად. ამის ფუნქციას ასრულებს Clear ღილაკი, რომლის კოდის ტექსტი მოცემულია 3.14 ლისტინგში.

```
//---- ლისტინგი 3.14 -- Clear -----------
private void btnClear_Click(object sender, RoutedEventArgs e)
 {
   textBox1.Text = "";
   textBox2.Text = "";
   textBox3.Text = "";
 }
```

3.6. WPF-აპლიკაციის აგება საპრობლემო სფეროსთვის „ელექტრონული არჩევნები“

ელექტრონული საარჩევნო სისტემა მიეკუთვნება რთული და დიდი სისტემების კლასს, რომლის ობიექტორიენტირებული მოდელირების, ანალიზის, დაპროექტების და შემდგომი პროგრამული რეალიზაციის საკითხები მეტად მნიშვნელოვანია. იგი ეფუძნება ქვეყნის დემოკრატიული პრინციპებისა და საზოგადოების მაღალზნეობრივი კულტურის განვითარებას.

განიხილება დაცული სახელმწიფო ქსელის აგების კონცეფცია და მისი არქიტექტურა. დაპროექტებულია ელექტრონული საარჩევნო სისტემისათვის საჭირო და აუცილებელი მულტიმედიური მონაცემთა რელაციური ბაზები. ამ თვალსაზრისით განიხილება მოდელირების კატეგორიალური მეთოდების და დაპროექტების ობიექტორიენტირებული, CASE-ტექნოლოგიების გამოყენება.

სისტემური ანალიზის შედეგად შერჩეულია ელექტრონული საარჩევნო სისტემისათვის აუთენტიფიკაციის სხვადასხვა ტიპების კომბინირება და მათი ინტეგრირება მულტიმედიურ მონაცემთა

ბაზებში. ესენია, თითის ანაბეჭდის სკანირების კომპონენტი, ხმის აუდიო ჩანაწერი, ბიომეტრული ფოტოსურათი და ელექტრონული ხელმოწერა [31-33].

ზემოხსენებული მოდულების ინტეგრაციით იზრდება უსაფრთხოება და ამავდროულად პირდაპირპროპორციულად მატულობს საიმედოობისა და ნდობის ხარისხი.

დამუშავებულია კლიენტ-სერვერ არქიტექტურის ლოგიკურად ერთიანი და ფიზიკურად განაწილებული მონაცემთა რელაციური ბაზების სისტემა ობიექტ-როლური მოდელირების პრინციპების საფუძველზე და შესაბამისი გრაფო-ანალიზური ინსტრუმენტების გამოყენებით [14,15].

შემუშავებულია აგებული სისტემის მომხმარებელთა ინტერ-ფეისები, ინსტრუქციები, დანერგვის და ექსპლუატაციის პროცესების ორგანიზაციული, ტექნიკური და იურიდიული ასპექტები.

3.6.1 ელექტრონული არჩევნების მართვის საინფორმაციო სისტემის აგების ამოცანა

ელექტრონული საარჩევნო სისტემის ძირითადი მიზანია [31]:

- საარჩევნო სიების სრულყოფა;
- საარჩევნო სიაში არსებული ყველა ამომრჩევლის შესახებ ამომწურავი ინფორმაციის მოგროვება და გამდიდრება;
- საარჩევნო პროცესის ავტომატიზაცია, რაც სამომავლოდ გაუიაფებს სახელმწიფოს არჩევნების ჩატარების ხარჯებს.

ნაშრომში ხორციელდება ელექტრონული საარჩევნო სისტემის ბიზნესპროცესების მართვის დაპროექტება და რეალიზაცია სერვის-ორიენტირებული არქიტექტურით.

მიზნის მისაღწევად განიხილება შემდეგი ძირითადი ამოცანები:

- არსებული თანამედროვე ელექტრონული საარჩევნო სისტე-მების ანალიზი და შესაბამისი ინფორმაციული ტექნოლოგიების

კლასიფიკაცია, ობიექტ-, პროცეს- და სერვის-ორიენტირებული დაპროექტების პრინციპებით;

- მულტიმედიური მონაცემთა ბაზების დაპროექტება და აგება კლიენტ-სერვერული ტექნოლოგიის გამოყენებით SQL Server-ის ბაზაზე;

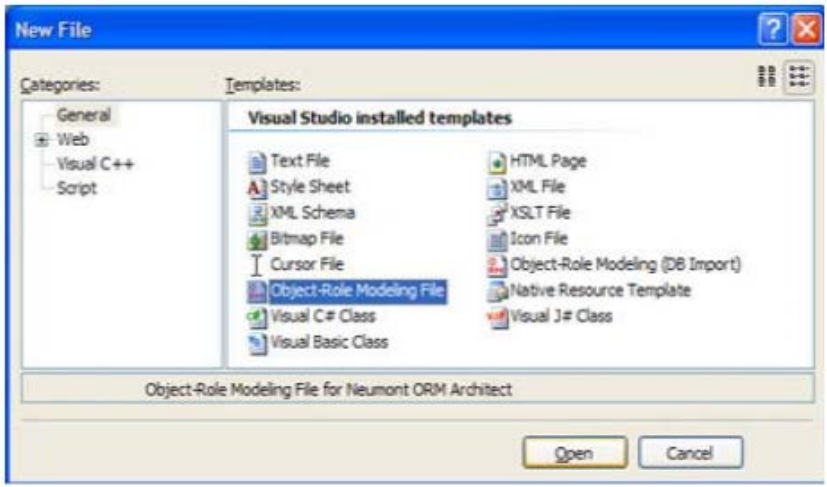
- სერვისორიენტირებული მონაცემთა განაწილებული ბაზების სტრუქტურების დასაპროექტებლად ობიექტ-როლური მოდელების (ORM) აგება და კვლევა რევერსიული CASE ტექნოლოგიების გამოყენებით;

- პროექტის შედეგების საფუძველზე ესპერიმენტული პროგრამული სისტემის რეალიზაცია .NET პლატფორმაზე, C#.NET, Natural ORM Architect და SQL Server პროგრამული პაკეტების გამოყენებით.

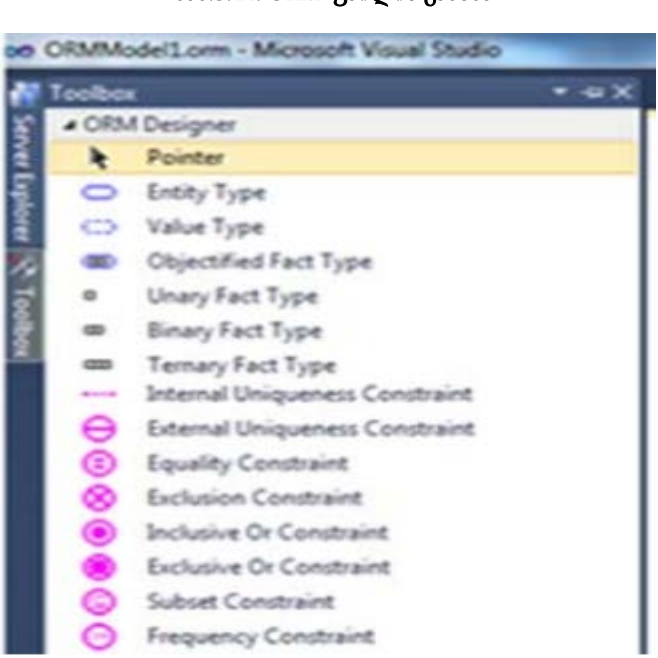
3.6.2 ელექტრონული არჩევნების სისტემის კონცეპტუალური მოდელის აგება ORM/ERM ტექნოლოგიით

მონაცემთა ბაზების ავტოამტიზებულ რეჟიმში დაპროექტება შესაძლებელია ობიექტ-როლური ORM (Object-Role Modeling) მოდელირების ტექნოლოგიის საშუალებით. Natural ORM Architect (NORMA) ინსტრუმენტი წარმოადგენს Microsoft Visual Studio.NET Framework-ის plugin-ს [14,15]. ORM-დიაგრამის შესაქმნელად საჭიროა General კატეგორიიდან Object-Role Modeling File template-ის ამორჩევა (ნახ.3.44) [32].

ORM-დიაგრამის აგება ხდება Document Window ფანჯარაში, სადაც სახეზეა დიაგრამის ასაგებად საჭირო ყველა ინსტრუმენტი: არსი (Entity), კავშირი (Connector) და შეზღუდვები (Constraints). ეს ნაჩვენებია 3.45 ნახაზზე.

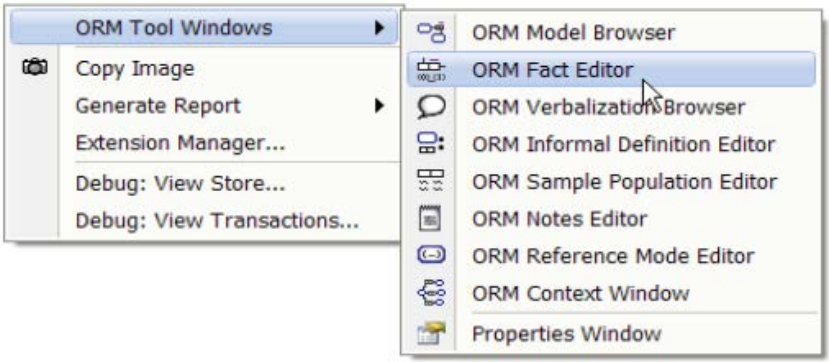


ნახ.3.44. ORM-ფაილის გახსნა



ნახ.3.45. ORM-დოკუმენტის ფანჯარა

ფაქტების შესატანად საჭიროა FACT EDITOR-ის გააქტიურება (ნახ.3.46).



ნახ.3.46. ORM Fact Editor-ის ამორჩევა

თავდაპირველად ხდება საპრობლემო სფეროს (კვლევის ობიექტის) მოთხოვნილებათა ანალიზი, ტექნიკური დავალების განსაზღვრის მიზნით, საიდანაც ჩამოყალიბდება ფაქტები.

ფაქტი არის კატეგორიალური ცნება, რომელშიც აისახება ობიექტის სემანტიკური (შინაარსობრივი) შედგენილობა და სტრუქტურა [15, 34]. იგი ენის გრამატიკისა და ლოგიკური ალგებრის სიმბიოზია.

ფორმალური სახით ფაქტის აღწერა ხდება შემდეგი სქემით:

„ობიექტი_1“ პრედიკატი „ობიექტი_2“ (ან „მნიშვნელობა“)

ესაა ორადგილიანი პრედიკატის ჩაწერის მაგალითი. შესაძლოა 3,4 და ზოგადად n-ადგილიანი პრედიკატების გამოყენებაც. სწორედ ამ ელემენტარული ფაქტების საშუალებით განისაზღვრება ORM-მოდელი. FACT EDITOR-ის ფანჯარაში შეგვაქვს დანარჩენი ფაქტები:

f1: Voter has VoterName

f2: Voter has FingerPrint

f3: Voter voted for a Majoritary_Deputy

f4: Voter has Voter_ID

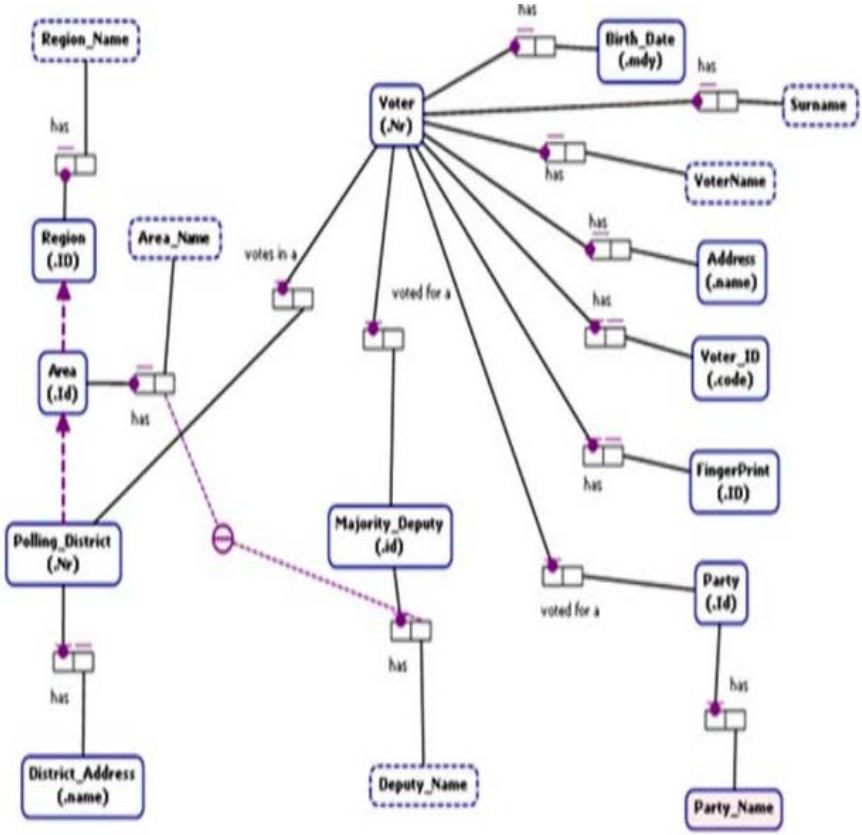
f5: Voter has Address

f6: Voter has Surname

f7: Majoritary_Deputy has Deputy_Name

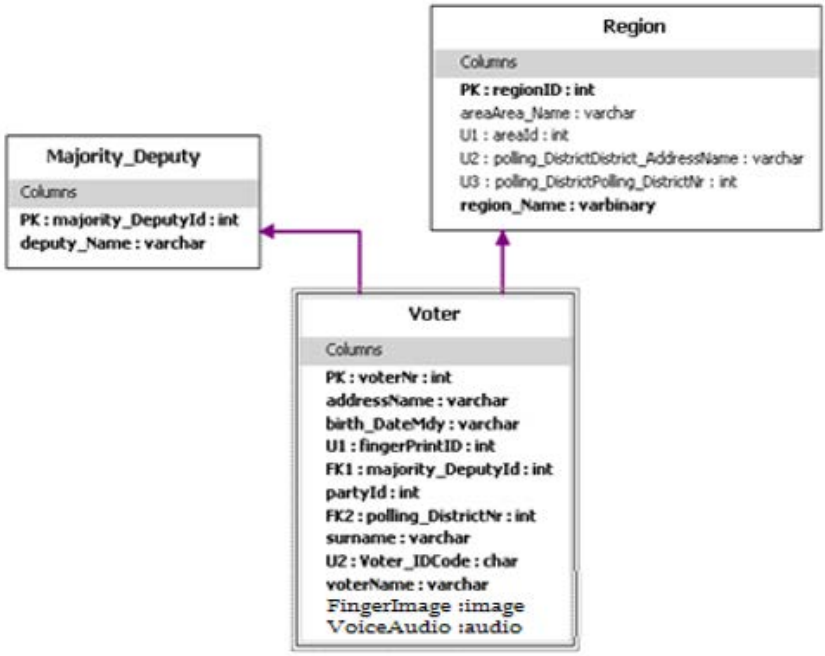
f8: Voter votes in a Polling_District . . . და ა.შ.

ზემომოყვანილი ფაქტების შეტანის შემდეგ ORM-დიაგრამა მიიღებს 3.47 ნახაზზე ნაჩვენებ სახეს.



ნახ.3.47. ORM-დიაგრამის ფრაგმენტი

აგებული დიაგრამიდან ავტომატიზებულ რეჟიმში ვახდენთ არსთადამოკიდებულების მოდელის ER (Entities-Relationship Model) კონსტრუირებას Extension manager-ის საშუალებით და მიიღება ER-მოდელის დიაგრამა, რომელიც ნაჩვენებია 3.48 ნახაზზე.



ნახ.3.48. ავტომატიზებულ რეჟიმში მიღებული ER-მოდელის ფრაგმენტი 3 ცხრილით

დიაგრამაზე გამოსახულია სამი არსი (Entities) და ორი ლოგიკური რელაციური კავშირი, რომელთა პროგრამული რეალიზაცია ხორციელდება ცხრილების (Tables) და ცხრილთა-შორისი კავშირებით პირველადი (Primary key) და მეორეული (Foreign key) გასაღებების გამოყენებით.

ჩვენ მაგალითში ამ მიზნით გამოყენებულია:

- PK: majority_DeputyId,
- PK: regionId,
- PK: voterNr,
- FK1: majority_DeputyId,
- FK2:pooling_DistrictNr,
- U1:fingerPrintID,
- U2:voter_IDCode.

Natural ORM Architect პაკეტის საშუალებით, ასევე ავტომატიზებულ რეჟიმში, ვახდენთ მონაცემთა აღწერის ენის შესაბამის DDL-კოდის გენერაციას.

Visual Studio.NET პაკეტის Solution Explorer-ის საშუალებით შესაძლებელია SQL Server მონაცემთა ბაზისთვის ჩვენ მიერ გენერირებული კოდის ნახვა.

მონაცემთა ბაზის ეს DDL-ფაილის ფრაგმენტი შემდეგნაირად გამოიყურება:

```
CREATE SCHEMA ORMModel1
GO
CREATE TABLE ORMModel1.Voter
( voterNr INTEGER NOT NULL,
  fingerPrintID INTEGER IDENTITY (1, 1) NOT NULL,
  Voter_IDCode NATIONAL CHARACTER(4000) NOT NULL,
  voterName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  surname NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  addressName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  partyId INTEGER IDENTITY (1, 1) NOT NULL,
  birth_DateMdy NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  majority_DeputyId INTEGER NOT NULL,
  polling_DistrictNr INTEGER NOT NULL,
  CONSTRAINT Voter_PK PRIMARY KEY(voterNr),
  CONSTRAINT Voter_UC1 UNIQUE(fingerPrintID),
  CONSTRAINT Voter_UC2 UNIQUE(Voter_IDCode)
```

```
 )
GO
CREATE TABLE ORMModel1.Majority_Deputy
( majority_DeputyId INTEGER IDENTITY (1, 1) NOT NULL,
  deputy_Name NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  CONSTRAINT Majority_Deputy_PK PRIMARY KEY(majority_DeputyId)
)
GO
CREATE TABLE ORMModel1.Region
(  regionID INTEGER IDENTITY (1, 1) NOT NULL,
   region_Name BINARY VARYING(MAX) NOT NULL,
   areaId INTEGER IDENTITY (1, 1),
   polling_DistrictDistrict_AddressName NATIONAL CHARACTER
                                VARYING(MAX),
   polling_DistrictPolling_DistrictNr INTEGER,
   areaArea_Name NATIONAL CHARACTER VARYING(MAX),
   CONSTRAINT Region_PK PRIMARY KEY(regionID),
   CONSTRAINT Region_Area_MandatoryGroup CHECK(areaId IS NOT
   NULL AND areaArea_Name IS NOT NULL OR
        polling_DistrictDistrict_AddressName IS NULL AND
        polling_DistrictPolling_DistrictNr IS NULL AND areaId IS NULL AND
        areaArea_Name IS NULL), CONSTRAINT
        Region_Polling_District_MandatoryGroup CHECK
       (polling_DistrictDistrict_AddressName IS NOT NULL AND
       polling_DistrictPolling_DistrictNr IS NOT  NULL OR
       polling_DistrictDistrict_AddressName IS NULL AND
       polling_DistrictPolling_DistrictNr IS NULL)
   )
GO
CREATE VIEW ORMModel1.Region_UC1 (areaId)
WITH SCHEMABINDING
AS
SELECT  areaId
```

```
FROM    ORMModel1.Region
WHERE areaId IS NOT NULL
GO
CREATE UNIQUE CLUSTERED INDEX Region_UC1Index ON
ORMModel1.Region_UC1(areaId)
GO
CREATE VIEW ORMModel1.Region_UC2
(polling_DistrictDistrict_AddressName)
WITH SCHEMABINDING
AS
SELECT polling_DistrictDistrict_AddressName
FROM   ORMModel1.Region
WHERE polling_DistrictDistrict_AddressName IS NOT NULL
GO
CREATE UNIQUE CLUSTERED INDEX Region_UC2Index ON
ORMModel1.Region_UC2 (polling_DistrictDistrict_AddressName)
GO CREATE VIEW ORMModel1.Region_UC3
(polling_DistrictPolling_DistrictNr)
WITH SCHEMABINDING
AS
SELECT polling_DistrictPolling_DistrictNr
FROM   ORMModel1.Region
WHERE polling_DistrictPolling_DistrictNr IS NOT NULL
GO
CREATE UNIQUE CLUSTERED INDEX Region_UC3Index ON
ORMModel1.Region_UC3 (polling_DistrictPolling_DistrictNr)
GO
ALTER TABLE ORMModel1.Voter ADD CONSTRAINT Voter_FK1
FOREIGN KEY (majority_DeputyId) REFERENCES
ORMModel1.Majority_Deputy(majority_DeputyId) ON
 DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.Voter ADD CONSTRAINT Voter_FK2
```

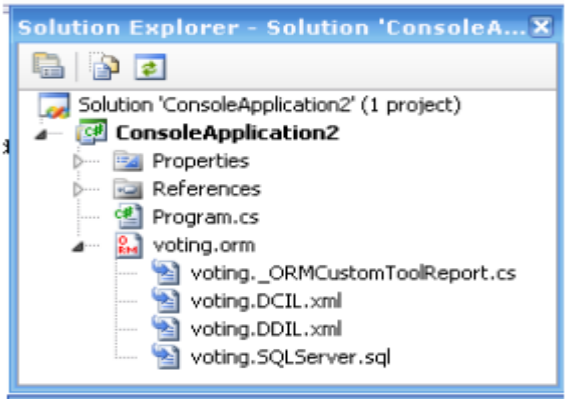
```
FOREIGN KEY (polling_DistrictNr) REFERENCES
ORMModel1.Region(polling_DistrictPolling_DistrictNr)ON
DELETE NO ACTION ON UPDATE NO ACTION
GO
```

3.6.3. მონაცემთა ბაზის აგების პროგრამული რეალიზაციის ავტომატიზებული პროცედურა

Natural ORM Architect პროგრამული პაკეტის დახმარებით ავტომატიზებულ რეჟიმში მოვახდინეთ DDL კოდის გენერირება, რათა აგვეგო დასაპროექტებელი სისტემის რელაციური სქემა.

Solution Explorer-ში ნაჩვენებია კოდი, რომელიც დაგენერირდა SQL Server-ისათვის (ნახ.3.49).

აქ ილუსტრირებულია ConsoleApplication2 პროექტის სტრუქტურა, რომლის voting.orm ფაილი რამდენიმე კომპონენტისგან (.cs, .xml და .sql ფაილები) შედგება [34].



ნახ.3.49. VisualStudio.NET გარემოში პროექტი ConsoleApplication2

პროგრამის ტექსტი, რომელიც ავტომატიზებულ რეჟიმში იქნა მიღებული და აღწერს ჩვენი კვლევის ობიექტის ანუ ელექტრონული საარჩევნო სისტემის მონაცემთა მოდელის შინაარსს, შემდეგი სახისაა:

ელ_საარჩევნო სისტემის დამახასიათებელი ფაქტები ცხრ.3.3

Voter **is an entity type**.
Reference Scheme: Voter has Voter_Nr.
Reference Mode: .Nr.
Fact Types:
Voter has Voter_Nr.
Voter has VoterName.
Voter has Surname.
Voter has Address.
Voter has FingerPrint.
Voter voted for a Party.
Voter voted for a Majority_Deputy.
VoterName.
Voter has Surname.
Each Voter has **exactly one** Surname.
It is possible that more than one Voter has **the same** Surname.
Address **is an entity type**.
Reference Scheme: Address has Address_name.
Reference Mode: .name.

Fact Types:
Address has Address_name.
Voter has Address.
Voter has Address.
Each Voter has **exactly one** Address.
It is possible that more than one Voter has **the same** Address.
Surname **is a value type**.
Portable data type: Text: Variable Length.

Fact Types:
Voter has Surname.
FingerPrint **is an entity type**.
Reference Scheme: FingerPrint has FingerPrint_ID.
Reference Mode: .ID.

Fact Types:
FingerPrint has FingerPrint_ID.
Voter has FingerPrint.
Voter has FingerPrint.
Each Voter has **exactly one** FingerPrint.
For each FingerPrint, **at most one** Voter has **that** FingerPrint.
Voter voted for a Party.
Each Voter voted for a **exactly one** Party.
It is possible that more than one Voter voted for a **the same** Party.
Party **is an entity type**.
Reference Scheme: Party has Party_Id.
Reference Mode: .Id.

Fact Types:
Party has Party_Id.

Voter has Voter_ID.
Voter has Birth_Date.
Voter votes in a Polling_District.
VoterName **is a value type**.
Portable data type: Text: Variable Length.

Fact Types:
Voter has VoterName.
Voter has VoterName.
Each Voter has **exactly one** VoterName.
It is possible that more than one Voter has **the same**
Reference Scheme: Majority_Deputy has Majority_Deputy_id.
Reference Mode: .id.

Fact Types:
Majority_Deputy has Majority_Deputy_id.
Voter voted for a Majority_Deputy.
Majority_Deputy has Deputy_Name.
Voter voted for a Majority_Deputy.
Each Voter voted for a **exactly one** Majority_Deputy.
It is possible that more than one Voter voted for a **the same** Majority_Deputy.
Deputy_Name **is a value type**.
Portable data type: Text: Variable Length.

Fact Types:
Majority_Deputy has Deputy_Name.
Majority_Deputy has Deputy_Name.
Each Majority_Deputy has **exactly one** Deputy_Name.
It is possible that more than one Majority_Deputy has **the same** Deputy_Name.
Voter_ID **is an entity type**.
Reference Scheme: Voter_ID has Voter_ID_code.
Reference Mode: .code.

Fact Types:
Voter_ID has Voter_ID_code.
Voter has Voter_ID.
Voter has Voter_ID.
Each Voter has **exactly one** Voter_ID.
For each Voter_ID, **at most one** Voter has **that** Voter_ID.
Birth_Date **is an entity type**.
Reference Scheme: Birth_Date has Birth_Date_mdy.
Reference Mode: .mdy.

Fact Types:
Birth_Date has Birth_Date_mdy.
Voter has Birth_Date.
Voter has Birth_Date.
Each Voter has **exactly one** Birth_Date.
It is possible that more than one Voter has **the same**

Voter voted for a Party.
Party has Party_Name.
Party has Party_Name.
Each Party has **exactly one** Party_Name.
It is possible that more than one Party has **the same** Party_Name.
Majority_Deputy **is an entity type.**
Region has Region_Name.
Each Area is an instance of Region.
Region_Name **is a value type.**
Portable data type: Raw Data: Variable Length.

Fact Types:
Region has Region_Name.
Region has Region_Name.
Each Region has **exactly one** Region_Name.
It is possible that more than one Region has **the same** Region_Name.
Area **is an entity type.**
Reference Scheme: Area has Area_Id.
Reference Mode: .Id.

Fact Types:
Area has Area_Id.
Area has Area_Name.
Each Polling_District is an instance of Area.
Each Area is an instance of Region.
Area_Name **is a value type.**
Portable data type: Text: Variable Length.

Fact Types:
Area has Area_Name.
Area has Area_Name.
Each Area has **exactly one** Area_Name.
It is possible that more than one Area has **the same** Area_Name.
Polling_District **is an entity type.**
Reference Scheme: Polling_District has Polling_District_Nr.
Reference Mode: .Nr.

Birth_Date.
Region **is an entity type.**
Reference Scheme: Region has Region_ID.
Reference Mode: .ID.

Fact Types:
Region has Region_ID.
Fact Types:
Polling_District has District_Address.
Each Polling_District is an instance of Area.
Polling_District has Polling_District_Nr.
Voter votes in a Polling_District.
District_Address **is an entity type.**
Reference Scheme: District_Address has District_Address_name.
Reference Mode: .name.

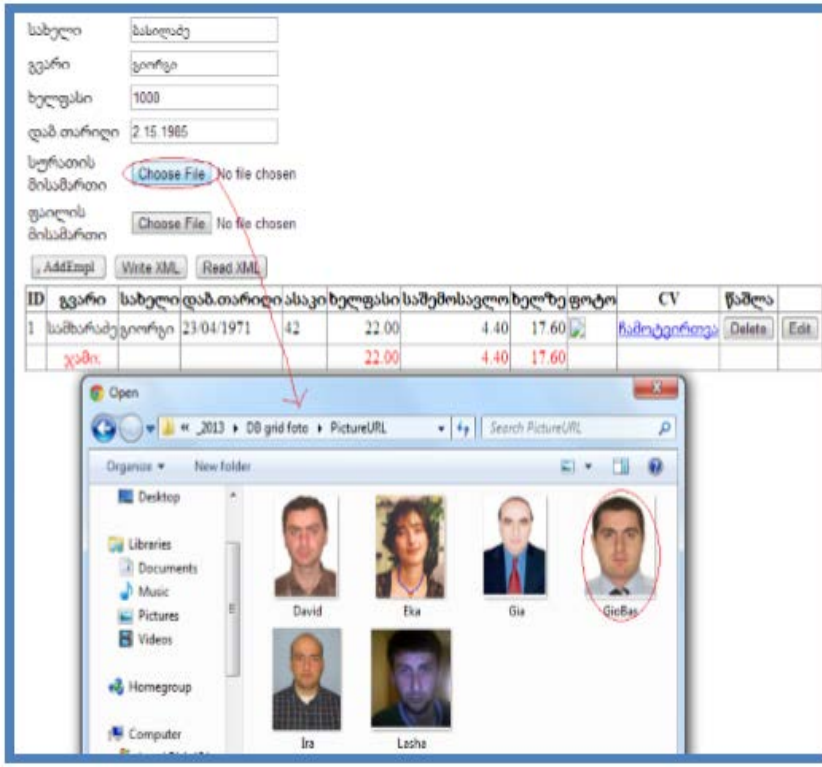
Fact Types:
District_Address has District_Address_name.
Polling_District has District_Address.
Polling_District has District_Address.
Each Polling_District has **exactly one** District_Address.
For each District_Address, **at most one** Polling_District has **that** District_Address.
Voter votes in a Polling_District.
Each Voter votes in a **exactly one** Polling_District.
It is possible that more than one Voter votes in a **the same** Polling_District.
C Area has Area_Name; Majority_Deputy **ontext:** has Deputy_Name.

In this context, each Area_Name, Deputy_Name **combination is unique.**
Model Error: Constraint 'ExternalUniquenessConstraint1' in model 'ORMModel1' requires a join path.
Each Area is an instance of Region.
Each Polling_District is an instance of Area.

3.6.4. სისტემის პროგრამული რეალიზაცია Visual Studio.NET Framework გარემოში

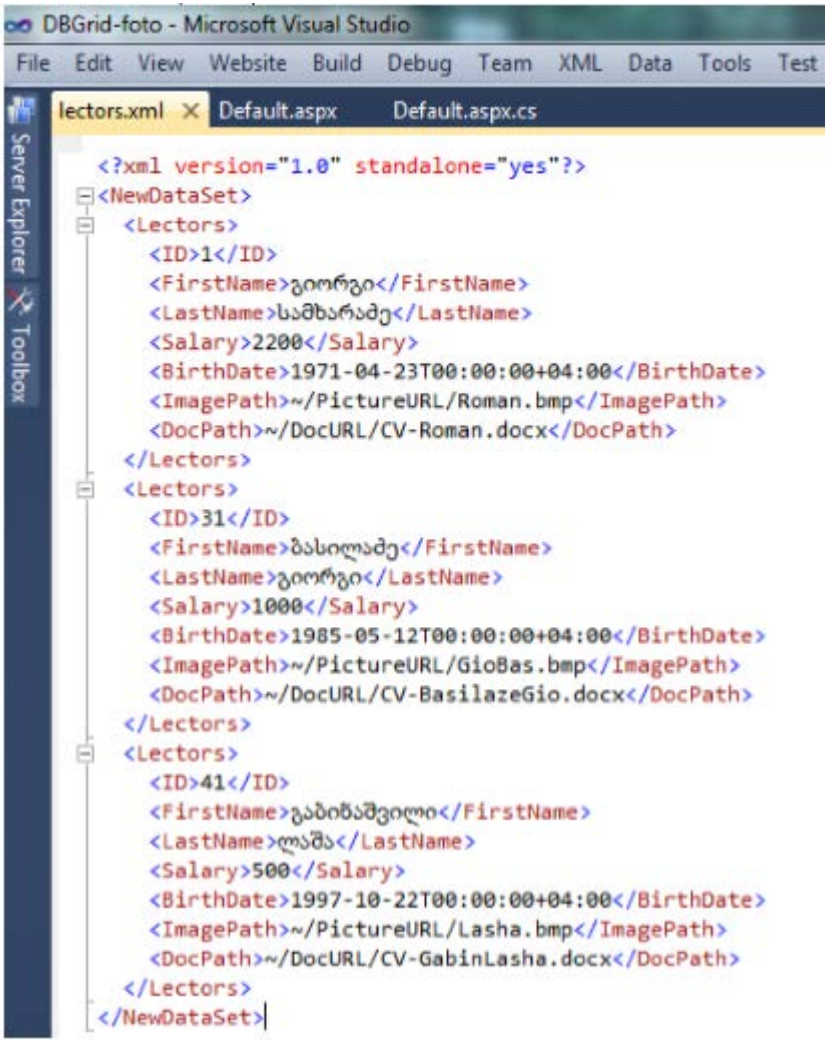
3.50 ნახაზზე ნაჩვენებია ინტერნეტ ბრაუზერში მომუშავე სისტემის ინტერფეისი - ამომრჩეველთა რეგისტრაცია. გრაფიკული

და ტექსტური ინფორმაციის შესატანად გამოყენებულია შესაბამისი დიალოგური პროცედურები.



ნახ.3.50. ინტერფეისის ფრაგმენტი ფოტო ინფორმაციის მიერთებით

Write XML ღილაკით განახლებული ცხრილი ჩაიწერება XML-ფაილში (მონაცემთა ბაზაში). Read XML ღილაკი უზრუნველყოფს ამ ბაზის ხელახლად წაკითხვას. 3.51 ნახაზზე ნაჩვენებია XML ფაილის ლისტინგი.



ნახ.3.51. XML ფაილის შიგთავსის სტრუქტურა

3.6.5. მომხმარებელთა ინტერფეისების დამუშავება

დაპროექტებული და აგებულია ელექტრონული საარჩევნო სისტემის მხარდამჭერი პროგრამული უზრუნველყოფა, რომლის ჩაწერაც მოხდება საარჩევნო უბნებზე არსებულ რეგისტრატორ-თათვის განკუთვნილ კომპიუტერებზე.

ელექტრონული რეგისტრაციის ფორმა შექმნილია ახალი ტექნოლოგიების გამოყენებით, როგორებიცაა Windows Presentation Foundation (WPF) და Metro Style App, რომელიც დღეისათვის ინოვაციას წარმოადგენს და გამოიყენება Microsoft Windows 8-ში [35].

ჩვენ შევეცადეთ, რომ პროგრამა ყოფილიყო შედარებით მარტივი და ადვილად სამართავი, რათა ნებისმიერ საარჩევნო უბნის რეგისტრატორს გაადვილებოდა მასთან მუშაობა და მომხდარიყო დროის მაქსიმალურად გამოყენება. ასევე გათვალისწინებულია მომხმარებელთა ინტერფეისების დამუშავება საქართველოს სომეხი, აზერბაიჯანელი, ოსი და აფხაზი ეროვნების წარმომადგენელთათვის [1,35].

აღნიშნული პროგრამული უზრუნველყოფის გამოყენება გვაძლევს გარანტიას სრულად გამოვრიცხოთ ისეთი ტერმინები და მისგან გამომდინარე შექმნილი უხერხულობები, როგორიც არის საარჩევნო სიების გაყალბება, მკვდარი სულები საარჩევნო სიებში, გამორჩენები საარჩევნო სიებში და არჩევნების მიმდინარეობის პროცესში - „ კარუსელები “ (რაც გამოიხატება ერთი ამომრჩევლის ან ამომრჩეველთა ჯგუფის მიერ რამდენიმე საარჩევნო უბანზე საკუთარი ხმის დაფიქსირებაში).

პროგრამა არ იძლევა უფლებას, რომ ერთმა ადამიანმა ერთზე მეტ საარჩევნო უბანზე გაიაროს რეგისტრაცია და მისცეს ხმა. იმ შემთხვევაში თუ ამომრჩეველს უკვე გავლილი აქვს რეგისტრაცია და შესაბამისად მისი კონსტიტუციური უფლება - მისცეს ხმა სასურველ

კანდიდატს უკვე აღსრულებულია, ხელმეორედ ნებისმიერ საარჩევნო უბანზე მისვლა აღმოჩენილი იქნება პროგრამის მიერ და ეცნობება საუბნო საარჩევნო კომისიის რეგისტრანტ წევრს იმის შესაებ თუ რა დროს და რომელ უბანზე გაიარა უკვე რეგისტრაცია ამა თუ იმ ამომრჩეველმა.

კომისიის წევრი ვალდებულია აღკვეთოს ხელმეორედ ხმის მიცემის ფაქტი და აცნობოს სამართალდამცავ ორგანოებს ზემოაღნიშნული, შესაბამისი რეაგირებისათვის.

პროგრამული უზრუნველყოფისათვის გამოვიყენეთ უახლესი ტექნიკა და თანამედროვე ტექნოლოგიები: თითის ანაბეჭდის სკანერი, ელექტრონული ხელმოწერის პანელი, ხმის ჩამწერი და შემდეგ მისი ამომცნობი სისტემები და ფოტოაპარატი, აგრეთვე ბიომეტრული სურათების შედარების სისტემები.

ამ ყველაფრის ხარჯზე, თუ მონაცემები არ იქნება გატარებული მონაცემთა ბაზაში თქვენ არ მოგეცემათ არჩევნებში მონაწილეობის უფლება. პროგრამა ითვალისწინებს გარდა პირადი ნომრით ამომრჩევლის იდენტიფიცირებისა, ისეთი დამატებითი იდენტიფიკატორების შემოტანას და მათ რეალიზაციას, როგორიცაა ამომრჩევლის თითის ანაბეჭდი, ამომრჩევლის ხმა, ამომრჩევლის ხელმოწერა და ამომრჩევლის ბიომეტრული სურათი.

ახლა დეტალურად განვიხილოთ პროგრამის მუშაობის ძირითადი პრინციპი:

ცენტრალურ საარჩევნო კომისიას ექნება წინასწარ ფორმირებული მონაცემთა ბაზა, თუ რომელ საარჩევნო უბანზე რომელი კომისიის წევრი არის მიმაგრებული და რა ფუნქცია მოვალეობები აქვს ნაკისრი.

საარჩევნო უბნის გახსნისას ამომრჩევლთა მიღების დაწყებამდე, როდესაც რეგისტრატორი გახსნის პროგრამას, პირველ რიგში თვითონ გაივლის იდენტიფიკაციას და როგორც კი მისი აუთენტურობა დადგინდება, როგორც რეგისტრატორი, ამის შემდეგ

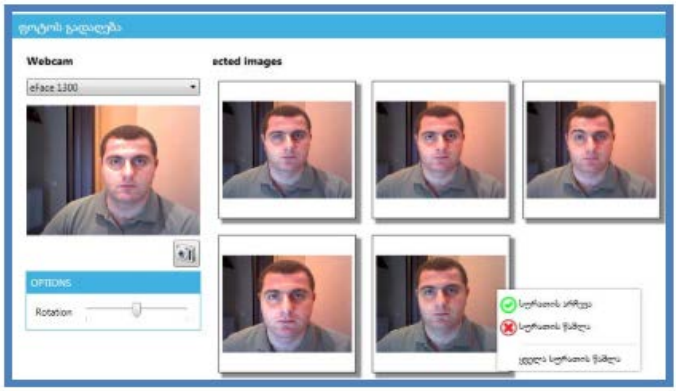
მოხდება მოთხოვნა Access Code(AC)-ის, რომელიც იქნება უნიკალური და წინასწარ დალუქული კონვერტით ექნება მიღებული საუბნო კომისიის თავჯდომარეს.

უნდა შედგეს აღნიშნული კონვერტის გახსნის ოქმი, რომელზეც კომისიის თავმჯდომარესთან ერთად ხელმომწერები იქნებიან თავჯდომარის მოადგილე და საუბნო კომისიის წევრები. აღნიშნული AC კოდის შეტანის შემდეგ მიეცემათ უფლება დაიწყონ ამომრჩეველთა მიღება და რეგისტრაცია. ყოველივე ეს აძლიერებს და ამყარებს უსაფრთოების ხარისხს და მეტ დამაჯერებლობას სძენს ელეტრონულ საარჩევნო სისტემას.

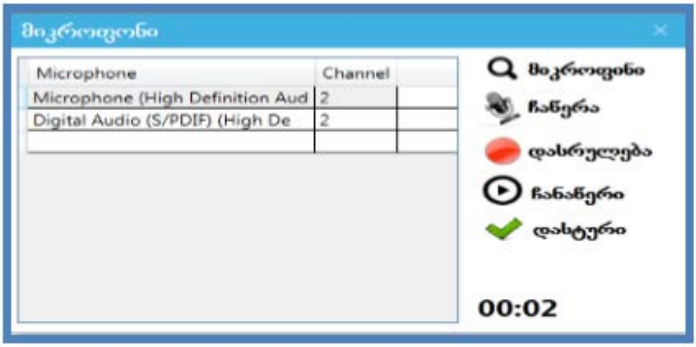
ამომრჩევლის უბანზე მისვლისას ხდება მისთვის ბიომეტრული სურათის გადაღება ვებ-კამერის მეშვებით, რეგისტრატორს შეუძლია ერთი ან რამდენიმე სურათის გადაღება და არჩევანის გაკეთება. ვებ-კამერა იმართება პროგრამიდან და აკეთებს დროებით ჩანაწერებს კომპიუტერში, სადაც ინახება მოქალაქის სურათი თავისივე უნიკალური დასახელებით, შერჩეული სურათის დაშლა ხდება ბიტებად და მონაცემთა ბაზაში ჩაწერა, ხოლო დანარჩენი სურათები ავტომატურად იშლება მყარი დისკიდან. კამერის სამართავად პროგრამული უზრუნველყოფა იყენებს კომპიუტერში არსებულ დრაივერს, რომელსაც პოულობს და ავტომატურად აყენებს. (ნახ.3.52).

ხმის ჩასაწერად და მიკროფონზე წვდომის განსახორციელებლად გამოვიყენეთ ყველასთვის კარგად ცნობილი და გამოცდილი ბიბლიოთეკა, როგორიცაა Naudio. მისი მეშვეობით ხდება წვდომა მიკროფონზე და ძიება ყველა არსებული მიკროფონის, რომელიც მიერთებულია კომპიუტერზე (ნახ.3.53) [35].

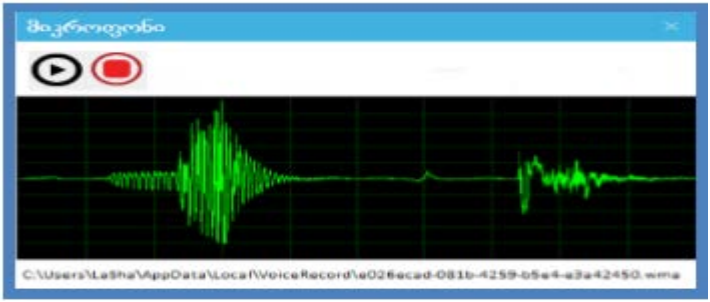
რეგისტრატორმა უნდა აირჩიოს მიკროფონი და ჩაიწეროს ხმა, რომელიც ასევე დროებით ჩაწერას აკეთებს კომპიუტერის მეხსიერებაში. ჩაწერის შემდეგ შესაძლებელია ჩაწერილი ხმის მოსმენა და მისი გრაფიკულად ნახვა (ნახ.3.54).



ნახ.3.52. მოქალაქის სურათი თავისივე უნიკალური დასახელებით



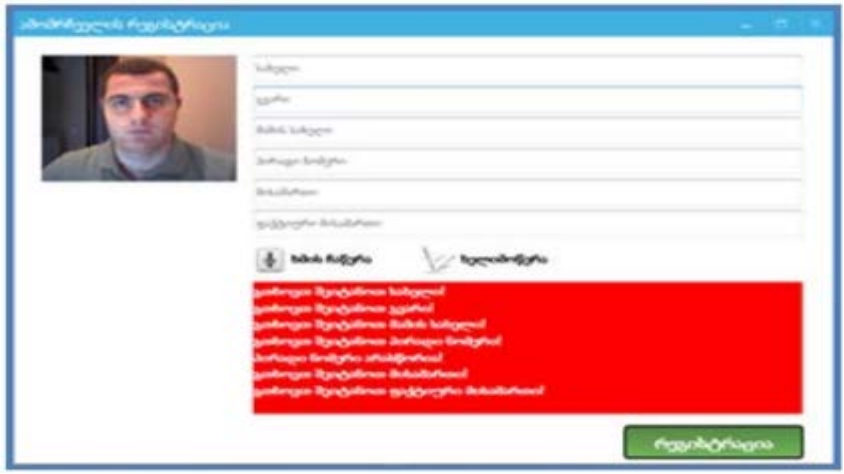
ნახ.3.53. კომპიუტერთან მიერთებული მიკროფონების ნუსხა



ნახ.3.54. ხმის გრაფიკული გამოსახულება

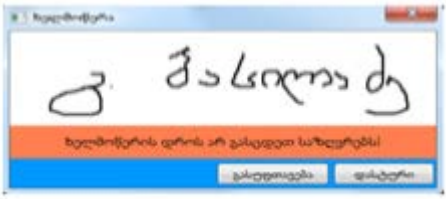
პროგრამული უზრუნველყოფა შეიცავს აგრეთვე მთელ რიგ ვალიდაციებს, რომელიც არ მისცემს რეგისტრატორს რაიმე სახის შეცდომის დაშვების უფლებას ამომრჩეველთა რეგისტრაციის დროს. კერძოდ, სახელის, გვარის, მამის სახელის, მისამართის და ფაქტიური მისამართის შეტანა და რეგისტრაცია მოხდება წინასწარ განსაზღვრული ფონტით-Sylfaen. რაც შეეხება პირად ნომერს, იგი აუცილებლად იქნება 11 ნიშნა რიცხვითი მონაცემი.

აღნიშნული შეცდომების დაშვების დროს მივიღებთ შემდეგი სახის გამაფრთხილებელ დიალოგურ ფანჯარას (ნახ.3.55).



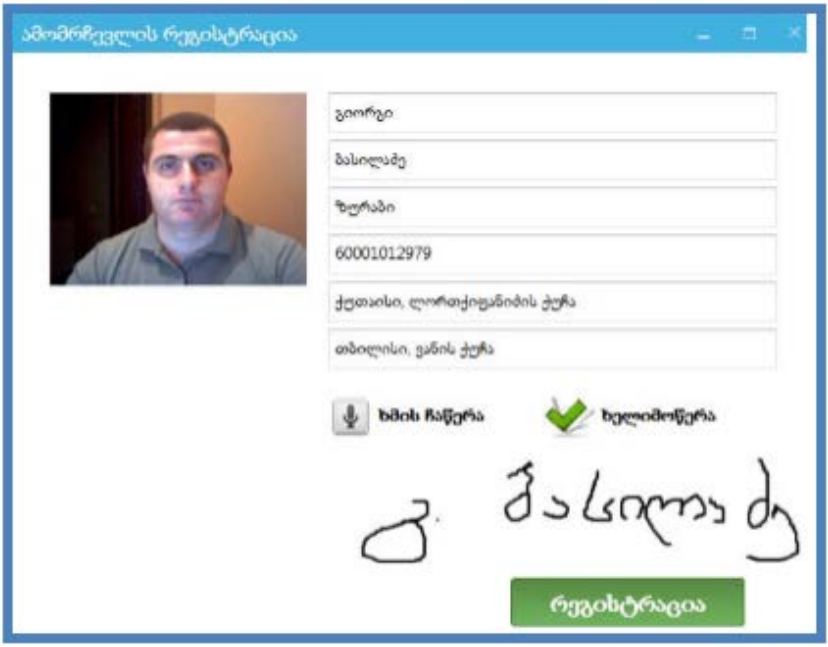
ნახ.3.55. გამაფრთხილებელი ფანჯარა

პროგრამა გამორიცხავს ამომრჩევლის შესახებ არასრული ინფორმაციის შემთხვევაში ბაზაში ჩანაწერის გაკეთებას, რაც თავის მხრივ ხელს უშლის არჩევნებზე დარეგისტრირებული ამომრჩეველთა რიცხვის ხელოვნურ ზრდას. პროგრამული უზრუნველყოფა ითვალისწინებს აგრეთვე ელექტრონული ხელმოწერის შეტანა-შენახვის მოდულს (ნახ.3.56) და თითის ანაბეჭდის შეტანა-შენახვაიდენტიფიცირებას.



ნახ.3.56. ელექტრონული ხელმოწერის შეტანა

პროექტის ფარგლებში შემუშავებულია მომხმარებელთა ინსტრუქციებიც [31]. მაგალითად, 3.57 ნახაზი გვიჩვენებს რეგისტრაციის ფორმის შევსებულ, საბოლოო ვარიანტს.



ნახ.3.57. რეგისტრაციის ფორმა

3.6.6. კომუნიკაცია: WCF - ტექნოლოგია

ბიზნესპროცესების შესრულებისას ერთ–ერთი მნიშვნელოვანი ასპექტია ურთიერთობა (კომუნიკაცია) აპლიკაციებს შორის, კლიენტებსა და სერვერებს შორის, აგრეთვე მუშა–პროცესებსა და ჰოსტ–დანართებს შორის.

აპლიკაციის მაგალითის სახით განიხილება პროექტის აგება საარჩევნო სისტემისათვის, რომელშიც მოთხოვნილი ინფორმაცია (მაგალითად, ამომრჩევლის ან დეპუტატის შესახებ) გადაიცემა ფილიალებს (საარჩევნო სისტემის იერარქიული რგოლები) შორის.

მთავარი ქმედებები, რომლებიც კომუნიკაციისთვის გამოიყენება, არის Send და Receive ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply). ეს ქმედებები გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და მონიტორინგისთვის.

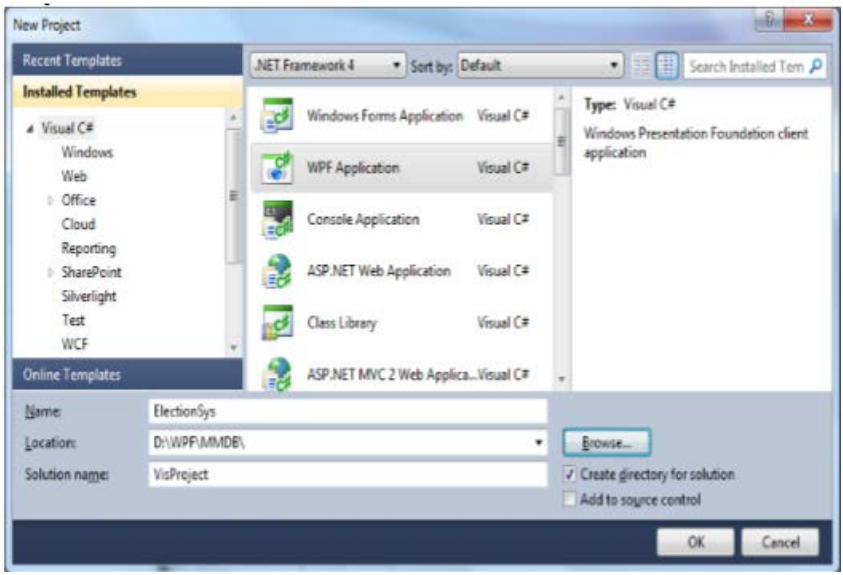
ავაგოთ მარტივი WPF-აპლიკაცია (Windows Presentation Foundation), რომელიც გამოიყენებს კომუნიკაციას სხვადასხვა აპლიკაციათა ბიზნესპროცესებს შორის.

- **WPF პროექტის შექმნა.**

1. ახალი პროექტის შექმნა WPF აპლიკაციის სახით იწყება პროექტის სახელის ElectionSys და Solution-ის დასახელების VisProject შერჩევით (ნახ.3.58).

2. Solution Explorer-ში ElectionSys პროექტზე მაუსის მარჯვენა ღილაკით ავირჩიოთ Add Reference და .NET ცხრილიდან დავამატოთ შემდეგი კავშირები:

- System.Activities
- System.Configuration
- System.ServiceModel
- System.ServiceModel.Activities



ნახ.3.58. WPF აპლიკაციის შექმნა

3. Solution Explorer-ში გენერირდება ვინდოუსის ფაილი სახელით Window1.xaml, რომელსაც ვცვალით Election.xaml - ით.

App.xaml ფაილი განსაზღვრავს ვინდოუსის startup-ს. ესაა ახლა Windows1 ფაილი, რომელიც უნდა შევცვალოთ ასევე: StartupUri=" Election.xaml".

- **ახალი config-ფაილის და Class-ების შექმნა**

რამდენიმე აპლიკაციის კონფიგურირების მიზნით (საარჩევნო ფილიალების რაოდენობის შესაბამისად) შეიქმნება App.config ფაილების დუბლები, რომლებშიც მოთავსებულ იქნება შესაბამისი ფილიალის ფიზიკური მონაცემები. ElectionSys–პროექტის Solution Explorer–დან ვირჩევთ Add New Item ➤. დიალოგურ ფანჯარაში General ჯგუფში ვირჩევთ Application Configuration File. ფაილის

სახელი ავტომატურად არის App.config(). კონფიგურაციის ფაილში შევიტანთ საჭირო მონაცემებს 1-ელი ლისტინგის მსგავსად.

```
<!— ლისტინგი_1 ———— –>
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Branch Name" value="Regioni GURIA"/>
    <add key="ID" value="{43E6DADD-4751-4056-8BB7-
                          7459B5C361AB}"/>
    <add key="Address" value="8000"/>
    <add key="Request Address" value="8730"/>
  </appSettings>
</configuration>
```

AppSettings სექციას აქვს მნიშვნელობები ფილიალის სახელი, ID (უნიკალური იდენტიფიკატორი) და მისამართი (პორტის ნომერი, რომელსაც აპლიკაცია იყენებს). მოთხოვნის მისამართი განსაზღვრავს პორტის ნომერს, საითაც იქნება მოთხოვნები გაგზავნილი. შესაძლებელია სხვა პორტების გამოყენებაც, თუ ეს აუცილებელი იქნება.

შემდეგ საჭიროა შეიქმნას კლასი, რომელიც განსაზღვრავს შეტყობინებებს აპლიკაციებს შორის. Solution Explorer-იდან Add -> Class და ჩავწეროთ კლასის სახელი Election.cs. ამ ფაილში დავამატოთ სახელსივრცეები (namespaces):

```
using System.Runtime.Serialization;
using System.ServiceModel;
```

Election.cs ფაილში განვსაზღვროთ სამი კლასი:

- Branch: განსაზღვრავს მონაცემებს საარჩევნო სისტემის ფილიალის მდებარეობის შესახებ;

- ElectionRequest: განსაზღვრავს ფილიალის მოთხოვნას;
- ElectionResponse: განსაზღვრავს პასუხს მოთხოვნის შესაბამის ფილიალისთვის. 3.4 ლისტინგში მოცემულია შესაბამისი კოდი:

```
//--- ლისტინგი 3.4 --- Election.cs ------
using System;
using System.Runtime.Serialization;
using System.ServiceModel;
namespace ElectionSys
{ /*****************************************/
  // ფილიალის მონაცემთა სტრუქტურის განსაზღვრა
  /*****************************************/
public class Branch
 {   public String BranchName { get; set; }
     public String Address { get; set; }
     public Guid BranchID { get; set; }
     #region Constructors
     public Branch() { }
     public Branch(String name, String address)
        {
          BranchName = name;
          Address = address;
          BranchID = Guid.NewGuid();
       }
     public Branch(String name, String address, Guid id)
      {
         BranchName = name;
         Address = address;
          BranchID = id;
      }
    public Branch(String name, String address, String id)
     {
        BranchName = name;
       Address = address;
```

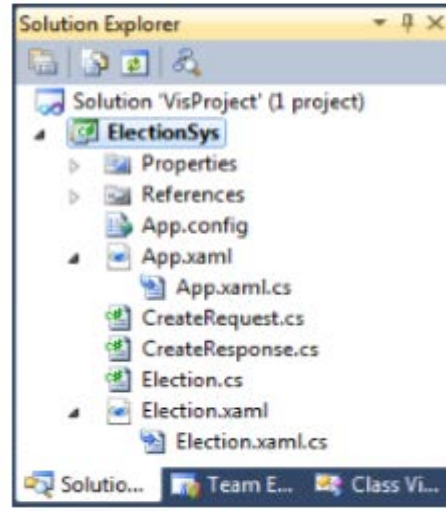
```
        BranchID = new Guid(id);
      }
     #endregion Constructors
   }
/*************************************************/
// მოთხოვნის შეტყობინების  განსაზღვრა, ElectionRequest
 /*************************************************/
[MessageContract(IsWrapped = false)]
public class ElectionRequest
 {
   private String _Region;
   private String _ArealName;
   private String _MajorDeputy;
   private Guid _RequestID;
   private Branch _Requester;
   private Guid _InstanceID;
   #region Constructors
   public ElectionRequest() {  }
public ElectionRequest(String arealname, String majordeputy,
          String region, Branch requester)
 {
   _ArealName = arealname;
   _MajorDeputy = majordeputy;
   _Region = region;
   _Requester = requester;
   _RequestID = Guid.NewGuid();
 }
public ElectionRequest(String arealname, String majordeputy,
           String region, Branch requester, Guid id)
     {   _ArealName = arealname;
          _MajorDeputy = majordeputy;
          _Region = region;
```

```
      _Requester = requester;
      _RequestID = id;
    }
  #endregion Constructors
  #region Public Properties
 [MessageBodyMember]
   public String Title
     {  get { return _ArealName }
        set { _ArealName = value; }
     }
  [MessageBodyMember]
   public String ISBN
    {  get { return _Region; }
       set { _Region = value; }
    }
  [MessageBodyMember]
    public String Author
     {  get { return _MajorDeputy; }
        set { _MajorDeputy = value; }
     }
    [MessageBodyMember]
     public Guid RequestID
      {  get { return _RequestID; }
         set { _RequestID = value; }
      }
     [MessageBodyMember]
      public Branch Requester
    {   get { return _Requester; }
    set { _Requester = value; }
  }
   [MessageBodyMember]
```

```
        public Guid InstanceID
            {  get { return _InstanceID; }
               set { _InstanceID = value; }
             }
         #endregion Public Properties
    }
/*************************************************/
// მოთხოვნის შეტყობინების განსაზღვრა:ElectionResponse
/*************************************************/
  [MessageContract(IsWrapped = false)]
  public class ElectionResponse
   {
      private bool _Reserved;
      private Branch _Provider;
      private Guid _RequestID;
      #region Constructors
      public ElectionResponse() {  }
      public ElectionResponse(ElectionRequest request, bool reserved,
                              Branch provider)
       {  _RequestID = request.RequestID;
          _Reserved = reserved;
          _Provider = provider;
        }
       #endregion Constructors
       #region Public Properties
       [MessageBodyMember]
        public bool Reserved
         {   get { return _Reserved; }
             set { _Reserved = value; }
         }
         [MessageBodyMember]
          public Branch Provider
            {  get { return _Provider; }
               set { _Provider = value; }
            }
```

```
        [MessageBodyMember]
         public Guid RequestID
           {   get { return _RequestID; }
               set { _RequestID = value; }
           }
        #endregion Public Properties
     }
/*************************************************/
// სერვისის კონტრექტის განსაზღვრა, IElectionSys
// რომელიც ორი მეთოდისგან შედგება: RequestElinfo() და
// RespondToRequest()
/*************************************************/
[ServiceContract]
public interface IElectionSys
 {
    [OperationContract(IsOneWay = true)]
    void RequestElinfo(ElectionRequest request);

    [OperationContract(IsOneWay = true)]
   void RespondToRequest(ElectionResponse response);
   }
 }
```

შედეგად, Solution Explorer-ს ექნება ასეთი სახე (ნახ.3,59).



ნახ.3.59

➢ **Window Form –ის განსაზღვრა**

გავხსნათ Election.xaml ფაილი და ავირჩიოთ XAML tab. ჩავანაცვლოთ კოდი შემდეგი 3.5 ლისტინგის ტექსტით:

```
<!-- ლისტინგი 3.5 --- Election.xaml ---- -->
<Window x:Class="ElectionSys.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml
        /presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ელ-საარჩევნო სისტემა" Height="480" Width="650"
    Loaded="Window_Loaded" Unloaded="Window_Unloaded">
 <Grid>
    <Label Height="40" HorizontalAlignment="Left"
       Margin="12,0,0,0" Name="lblBranch" FontSize="22"
       VerticalAlignment="Top" Width="276"
       FontStretch="Expanded">ოლქი: ოზურგეთი</Label>
   <ListView x:Name="requestList" Margin="12,42,12,5"
      Height="150" VerticalAlignment="Top"
      ItemsSource="{Binding}"
      SelectionChanged="requestList_SelectionChanged">
    <ListView.View>
     <GridView>
       <GridViewColumn Header="Request List" Width="610">
         <GridViewColumn.CellTemplate>
           <DataTemplate>
             <StackPanel Orientation="Horizontal">
               <TextBlock Text="{Binding
                     Requester.BranchName}" Width="100"/>
```

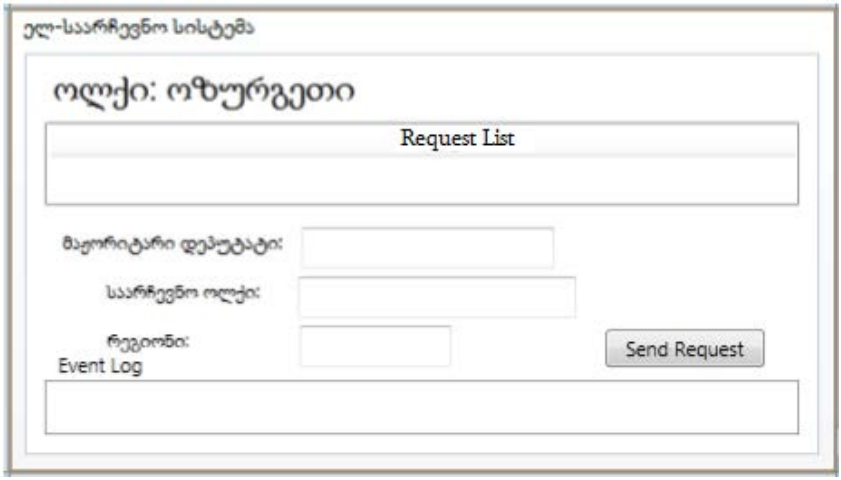
```
            <TextBlock Text="{Binding MajorDeputy}"
                  Width="95"/>
            <TextBlock Text="{Binding ArealName}"
                  Width="180"/>
            <TextBlock Text="{Binding Region}"
                  Width="90"/>
            <Button Content="Reserve" Tag="{Binding
              InstanceID}" Click="Reserve" Width="65"/>
            <Button Content="Cancel" Tag="{Binding
              InstanceID}" Click="Cancel" Width="60"/>
           </StackPanel>
          </DataTemplate>
         </GridViewColumn.CellTemplate>
        </GridViewColumn>
       </GridView>
      </ListView.View>
     </ListView>
  <Label Height="30" Margin="15,0,0,210" Name="label5"
        VerticalAlignment="Bottom"
        HorizontalAlignment="Left" Width="148"
         HorizontalContentAlignment="Right">მაჟორიტარი დეპუტატი:</Label>
<Label Height="30" Margin="34,0,479,180" Name="label2"
          VerticalAlignment="Bottom"
          HorizontalContentAlignment= "Right">საარჩევნო ოლქი:</Label>
<Label Height="30" Margin="45,25,0,150" Name="label3"
           VerticalAlignment="Bottom"
           HorizontalAlignment="Left" Width="60"
           HorizontalContentAlignment="Right">რეგიონი:</Label>
<TextBox Height="25" Margin="169,0,0,210"
           Name="txtMajorDeputy"
           VerticalAlignment="Bottom"
           HorizontalAlignment="Left" Width="200" />
<TextBox Height="25" Margin="0,0,161,180"
           Name="txtArealName"
           VerticalAlignment="Bottom"
           HorizontalAlignment="Right" Width="300" />
```

```
    <TextBox Height="25" Margin="168,0,0,150"
             Name="txtRegion"
             VerticalAlignment="Bottom"
             HorizontalAlignment="Left" Width="100" />
    <Button Height="23" Margin="497,0,0,150"
             Name="btnRequest"
             VerticalAlignment="Bottom"
             HorizontalAlignment="Left" Width="98"
          Click="btnRequest_Click">Send Request</Button>
    <Label Height="27" HorizontalAlignment="Left"
          Margin="15,0,0,137" Name="label4"
          VerticalAlignment="Bottom"
          Width="76">Event Log</Label>
    <ListBox Margin="12,0,12,12" Name="lstEvents"
          Height="130" VerticalAlignment="Bottom"
          FontStretch="Condensed" FontSize="10" />
    </Grid>
</Window>
```

<!– – ლისტინგი 3.5-ის დასასრული -----

შემდეგ ავირჩიოთ Design Tab-ს. ფორმას უნდა ჰქონდეს შემდეგი სახე (ნახ.3.60). ფორმის ზედა ნაწილში მოთხოვნების სია (RequestList) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად რეგიონალურ ცენტრში გამოიყენება ველები ფორმის შუაში.

აქ მიეთითება მაგალითად, მაჟორიტარი დეპუტატის გვარი_ს., საარჩევნო_ოლქი და რეგიონი, შემდეგ გაგზავნის ღილაკი „მოთხოვნის გაგზავნა“ (Send Request). „მოვლენათა რეგისტრაცია“ (Event Log) ქვედა მარცხენა კუთხეში ასახავს ბიზნესპროცესების შეტყობინებებს.



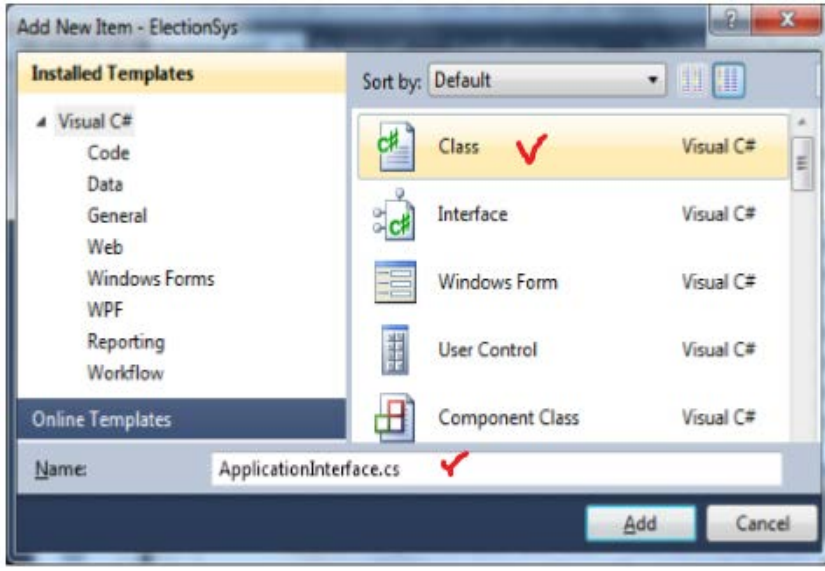
ნახ.3.60. ინტერფეისი „საარჩევნო ოლქი“

- **ტექსტის ჩამწერის რეალიზაცია**

WriteLine ქმედებისთვის, რომელიც გამიყენება ტექსტის გამოსატანად ეკრანზე, არ დაგვიყენებია თვისება TextWriter. კონსოლის რეჟიმში ტექსტი ავტომატურად გამოიტანება, ფორმაზე გამოსატანად კი უნდა გავეცნოთ TextWriter კლასს.

- **აპლიკაციის სტატიკური მიმთითებლის უზრუნველყოფა**

თავიდან უნდა შეიქმნას სტატიკური კლასი, რომელიც უზრუნველყოფს აპლიკაციის ფანჯარასთან წვდომას. Solution Explorer-ში ElectionSys–ზე მაუსის მარჯვენა ღილაკით ვირჩევთ Add -> Class (ნახ.3.61). კლასის სახელია ApplicationInterface.cs, რომლის პროგრამული რეალიზაცია მოცემულია 3.6 ლისტინგში.



ნახ.3.61. Class-ის არჩევა

```
// – – ლისტინგი_3.6 – – ApplicationInterface.cs  – –
using System;
using System.Windows.Controls;
using System.Activities;
namespace ElectionSys
{
  public static class ApplicationInterface
   {
      public static MainWindow _app { get; set; }
      public static void AddEvent(String status)
       {
          if (_app != null)
            {
```

```
            new ListBoxTextWriter(_app.GetEventListBox())
                    .WriteLine(status);
        }
      }
  public static void RequestElinfo(ElectionRequest request)
   {
      if (_app != null)
         _app.RequestElinfo(request);
   }
   public static void RespondToRequest(ElectionResponse response)
     {
        if (_app != null)
          _app.RespondToRequest(response);
     }
   public static void NewRequest(ElectionRequest request)
     {
        if (_app != null)          _app.AddNewRequest(request);
     }
    }
  }
```

ApplicationInterface კლასს აქვს სტატიკური მიმთითებელი (_app) აპლიკაციის ფანჯარაზე (MainWindow კლასი). სტატიკური AddEvent() მეთოდი ქმნის ListBoxTextWriter კლასის ეგზემპლარს, რომელიც შემდგომში იქნება რეალიზებული და იძახებს მის WriteLine() მეთოდს.

გავხსნათ Election.xaml.cs ფაილი და დავამატოთ შემდეგი სახელსივრცეები:

```
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
```

```
using System.ServiceModel.Description;
using System.ServiceModel.Channels;
using System.Activities;
using System.Xml.Linq;
using System.Configuration;
```

დავამატოთ MainWindow()-ში კონსტრუქტორის შემდეგი კოდი: ApplicationInterface._app = this;

აქ this აინიციალიზებს _app მიმართვას ApplicationInterface კლასში. ვინაიდან იგი სტატიკური კლასია, მასში იქნება მხოლოდ ერთი ეგზემპლარი, რომელსაც ექნება მიმართვა MainWindow კლასზე. დავამატოთ შემდეგი მეთოდები Election.xaml.cs ფაილში.

```
public ListBox GetEventListBox()
 {
    return this.lstEvents;
 }
private void AddEvent(string szText)
  {
    lstEvents.Items.Add(szText);
  }
```

GetEventListBox() მეთოდი აბრუნებს უკან მიმთითებელს ListBox–ის ფაქტიური კონტროლისთვის, რომელმაც უნდა ასახოს ეს მოვლენები. ამ მეთოდს იყენებს ApplicationInterface კლასი. AddEvent () მეთოდს იყენებს აპლიკაცია მაშინ, როცა მას სჭირდება მოვლენის დამატება.

- **ListBoxTextWriter–ის რეალიზაია**

დავამატოთ პროექტს კლასი ListBoxTextWriter.cs, რომლის რეალიზაცია 3.7 ლისტინგშია მოცემული.

```
// – – ლისტინგი_3.7 --- ListBoxTextWriter.cs ------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Controls;
namespace ElectionSys
 {
    public class ListBoxTextWriter : TextWriter
     {
       const string textClosed = "This TextWriter must be opened before use";
       private Encoding _encoding;
       private bool _isOpen = false;
       private ListBox _listBox;
       public ListBoxTextWriter()
        {
          // Get the static list box
          _listBox = ApplicationInterface._app.GetEventListBox();
         if (_listBox != null)
             _isOpen = true;
        }
       public ListBoxTextWriter(ListBox listBox)
          {
            this._listBox = listBox;
            this._isOpen = true;
          }
        public override Encoding Encoding
        {
        get
```

```
    {
     if (_encoding == null)
       {
         _encoding = new UnicodeEncoding(false, false);
       }
     return _encoding;
   }
 }
public override void Close()
  {
   this.Dispose(true);
 }
protected override void Dispose(bool disposing)
  {
     this._isOpen = false;
     base.Dispose(disposing);
   }
 public override void Write(char value)
    {
     if(!this._Open)
    throw new ApplicationException(textClosed); ;

   this._listBox.Dispatcher.BeginInvoke
      (new Action(() =>
        this._listBox.Items.Add(value.ToString())));
  }
public override void Write(string value)
  {
    if (!this._isOpen)
       throw new ApplicationException(textClosed);
```

```
                ;
 if (value != null)
     this._listBox.Dispatcher.BeginInvoke
             (new Action(() =>
                     this._listBox.Items.Add(value)));
     }
  public override void Write(char[] buffer, int index,
                         int count)
   {
     String toAdd = "";
     if (!this._isOpen)
        throw new ApplicationException(textClosed); ;
      if (buffer == null || index < 0 || count < 0)
          throw new ArgumentOutOfRangeException("buffer");
      if ((buffer.Length - index) < count)
         throw new ArgumentException("The buffer is too small");
              for (int i = 0; i < count; i++)
                  toAdd += buffer[i];
         this._listBox.Dispatcher.BeginInvoke
           (new Action(() => this._listBox.Items.Add(toAdd)));
     }
  }
}
```

ListBoxTextWriter კლასი არის მიღებული აბსტრაქტული TextWriter კლასიდან და უზრუნველყოფს Write() მეთოდის განხორციელებას, რომელიც ამატებს სტრიქონს ListBox-ში (თუ გსურთ განახორციელოთ Write() მეთოდის სამი გადატვირთვა, იმისთვის რომ იყოს მიღებული, როგორც char ან string ან char [] მასივი).

არსებული კონსტრუქტორი იყენებს სტატიკურ ApplicationInterface კლასს MainWindow-ის lstEvents კონტროლის მისაღებად. იგი ასევე უზრუნველყოფს კონსტრუქტორს, რომელშიც ListBox შეიძლება შესრულდეს. ეს კონსტრუქტორი გამოიყენება ApplicationInterface კლასის AddEvent () მეთოდით.

ListBox-ის Add() მეთოდი ხორციელდება აპლიკაციის შესრულების ნაკადის (thread) შემდეგაც. იგი აკეთებს ამას Dispatcher-ის BeginInvoke() მეთოდის გამოყენებით, რომელიც ასოცირდება lstEvents მართვის ელემენტთან. ეს საშუალებას აძლევს მეთოდს იმუშაოს სხვადასხვა ნაკადებიდან გამოძახების დროსაც.

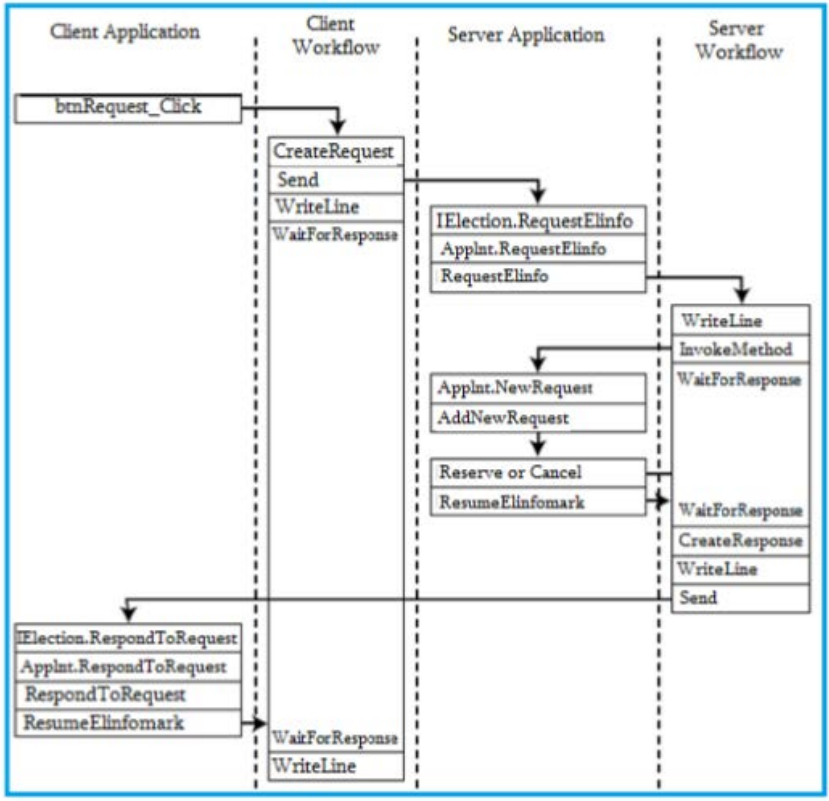
იმის გამო, რომ ListBoxTextWriter კლასი არის მიღებული TextWriter-დან, შეიძლება მისი მითითება, როგორც TextWriter თვისებისა ნებისმიერი WriteLine ქმედებისთვის. და სტატიკური ApplicationInterface კლასის გამო, ListBoxTextWriter კლასს შეუძლია წვდომა lstEvents ელემენტზე აპლიკაციის გარედანაც კი.

ასე რომ, არსებობს სამი გზა lstEvents მართვის ელემენტში ტექსტის დასამატებლად:

- აპლიკაციის შიგნიდან, გამოიყენება ლოკალური AddEvent () მეთოდი;
- აპლიკაციის გარედან, გამოიყენება ApplicationInterface კლასის AddEvent () მეთოდი;
- WriteLine ქმედებიდან, მიეთითება TextWriter თვისება ListBoxTextWriter-ზე.

3.6.7. ბიზნესპროცესის რეალიზაცია

3.62 ნახაზზე ნაჩვენებია ზოგადი ლოგიკა და შეტყობინებათა ნაკადი. ამ სქემის ელემენტები მოგვიანებით იქნება ახსნილი, ამჯერად კი განხილულ იქნება ძირითადი



ნახ.3.62. მოდულების მუშაობის ლოგიკა და შეტყობინებათა ნაკადი

გავხსნათ Election.cs ფაილი, რომელშიც გამოჩნდება ინტერფეისის შემდეგი განსაზღვრება:

```
[ServiceContract]
 public interface IElectionSys
  {  [OperationContract]
     void Elinfo(ElectionRequest request);
     [OperationContract]
    void RespondToRequest(ElectionResponse response);  }
```

საჭიროა მცირე ცვლილების ჩატარება. OperationContract-ს უნდა დაემატოს (IsOneWay = true). ქვემოთ ნაჩვენებია ეს:

```
[ServiceContract]
public interface IElectionSys
  {
    [OperationContract(IsOneWay = true)]
    void RequestElinfo(ElectionRequest request);

   [OperationContract(IsOneWay = true)]
   void RespondToRequest(ElectionResponse response);
  }
```

შეტყობინება იგზავნება ბიზნესპროცესთან ერთად, მაგრამ პასუხი მიიღება ServiceHost-ით აპლიკაციის შიგნით. ასე, რომ ეს არაა ტქნიკური ორმხრივი საუბარი. არსებობს შეტყობინებები ორივე მიმართულებით. რადგან გაგზავნის და მიღების საბოლოო წერტილები სხვადასხვაა, WCF ამას აფიქსირებს როგორც ცალკე ერთმიმართულებიანი შეტყობინებები.

3.6.8. სერვისის კონტრაქტის რეალიზაცია

სერვისის კონტრაქტი განსაზღვრავს მხოლოდ ხელმისაწვდომ მეთოდებს, იგი არ უზრუნველყოფს მათ იმპლემენტაციას (რეალიზაციას). ჩვენი პროექტისთვის აუცილებელია ამ საკითხის გადაწყვეტა. ამიტომაც, Solution Explorer-ში ElectionSys -ზე მარჯვენა ღილაკით ვირჩევთ Add Class. მივუთითებთ კლასის სახელს ClientService.cs. მისი კოდის რეალიზაცია ნაჩვენებია 3.8 ლისტინგში.

```
//---- ლისტინგი 3.8 ----- ClientService.cs -----
using System;
using System.ServiceModel;
```

```
namespace ElectionSys
 {
   public class ClientService : IElectionSys
     {
       public void RequestElinfo(ElectionRequest request)
         {
           ApplicationInterface.RequestElinfo(request);
         }
       public void RespondToRequest(ElectionResponse response)
         {
           ApplicationInterface.RespondToRequest(response);
         }
     }
}
```

ეს რეალიზაცია იყენებს ApplicationInterface სტატიკურ კლასს, რომელიც უკვე შექმნილია ჩვენს მიერ. ყოველი მეთოდი უბრალოდ იძახებს ApplicationInterface კლასის შესაბამის მეთოდს. გავხსნათ ApplicationInterface.cs ფაილი და დავამატოთ შემდეგი მეთოდები:

```
public static void RequestElinfo(ElectionRequest request)
 {
    if (_app != null)
  _app.RequestElinfo(request);
 }
public static void  RespondToRequest(ElectionResponse response)
 {
   if (_app != null)
   _app.RespondToRequest(response);
 }
```

ეს მეთოდები თავის მხრივ იძახებს შესაბამის მეთოდებს აპლიკაციაში სტატიკური მიმთითებლის გამოყენებით. საჭირო ინება ამ მეთოდების რეალიზება Election.xaml.cs ფაილში, რასაც მოგვიანებით დავუბრუნდებით.

3.6.9. ServiceHost -ის რეალიზაცია

აპლიკაციისთვის აუცილებელია ServiceHost-ის რეალიზაცია შემავალი შეტყობინებების მისაღებად (მოსასმენად). გავხსნათ Election.xaml.cs ფაილი და დავამატოთ შემდეგი კლასის წევრები:

```
private ServiceHost _sh;
```

იგი უნდა მოთავსდეს კონსტრუქტორის წინ. ასე:

```
public partial class MainWindow : Window
 {
    private ServiceHost _sh;
    public MainWindow()
     {
         InitializeComponent();
        ApplicationInterface._app = this;
     }
```

ServiceHost იწყება მაშინ, როცა ფანჯარა ჩატვირთულია და იხურება, როცა ფანჯარა ამოტვირთულია. მეთოდების დამატება ნაჩვენებია 3.9 ლისტინგში MainWindow კლასისთვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

```
// -- ლისტინგი 3.9 --- The Loaded and Unloaded Event Handlers ----
private void Window_Loaded(object sender, RoutedEventArgs e)
 {
  // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და
  // მისი ქსელური მისამართი
   Configuration config = ConfigurationManager.OpenExeConfiguration
                          (ConfigurationUserLevel.None);
```

```
    AppSettingsSection app = (AppSettingsSection)config.GetSection
            ("appSettings");
    string adr = app.Settings["Address"].Value;
   // ფილიალის სახელის გამოტანა ფორმაზე
  lblBranch.Content = app.Settings["Branch Name"].Value;

  // ServiceHost-ის შექმნა
   _sh = new ServiceHost(typeof(ClientService));
  // დასასრულის წერტილის (Endpoint) დამატება
    string szAddress = "http://localhost:" + adr + "/ClientService";
   System.ServiceModel.Channels.Binding bBinding =
                                          new BasicHttpBinding();
 _sh.AddServiceEndpoint(typeof(ILibraryReservation),bBinding,
                    szAddress);
// ServiceHost-ის გახსნა შეტყობინებების მისაღებად (listen)
  _sh.Open();    // ListBoxTextWriter -ის ტესტირება
   //ListBoxTextWriter lbtw = new ListBoxTextWriter();
  //lbtw.Write("ეს არის ტესტი - This is a test");
}
private void Window_Unloaded(object sender, RoutedEventArgs e)
 {
   // service host-ის დატოვება
    _sh.Close();
}
```

მოვლენის დამმუშავებელი Loaded ხსნის კონფიგურაციის ფაილს და ათავსებს საარჩევნო უბნის (ფილიალის) სახელს lblBranch - მართვის ელემენტში, ამიტომაც ფორმა ასახავს ლოკალური ფილიალის სახელს. შემდეგ იქმნება ServiceHost თანამგზავრი (passing) ClientService კლასისა, რომელიც ახლახანს შევქმენით როგორც მისი რეალიზაცია. შემდეგ იგი აკონფიგურირებს

დასასრულის წერტილს ServiceHost-თვის, იყენებს რა ცნობილი მისამართის, მიბმის და კონტრაქტის სამეულს. Unloaded მოვლენის დამმუშავებელი უბრალოდ ხურავს ServiceHost-ს, ასე რომ მეტი აღარ მოხდება შეტყობინებების მიღება.

3.6.10. SendRequest ბიზნესპროცესის რეალიზაცია

ახლა შევასრულოთ მუშა პროცესების რეალიზაცია. Solution Explorer-ის ElectionSys ზე მარჯვენა ღილაკით ავირჩიოთ Add -> Class. სახელი ElectionWF.cs. კოდის რეალიზაცია მოცემულია 3.10 ლისტინგში.

```
// --- ლისტინგი 3.10 --- ElectionWF.cs ---
using System;
using System.Activities;
using System.Activities.Statements;
using System.ServiceModel.Activities;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Runtime.Serialization;
using System.Xml.Linq;
using System.IO;

namespace ElectionSys
{
  // ეს ფაილი შეიცავს ორი workflow-ის განსაზღვრას:
  // SendRequest - აინიციალიზირებს  ახალ მოთხოვნებს,
  // ProcessRequest - ამუშავებს შემოსულ მოთხოვნებს
 public sealed class SendRequest : Activity
   { // შემავალი და გამომავალი არგუმენტების განსაზღვრა----
 public InArgument<string> ArealName { get; set; }
```

```
public InArgument<string> MajorDeputy { get; set; }
public InArgument<string> Region { get; set; }
public InArgument<TextWriter> Writer { get; set; }
public OutArgument<ElectionResponse> Response {get; set;}
public SendRequest()
 {
     // ცვლადების განსზღვრა ამ workflow -სთვის ---
  Variable<ElectionRequest> request = new Variable<ElectionRequest>
          {Name="request" };
  Variable<string> requestAddress = new Variable<string> { Name =
          "RequestAddress" };
  Variable<bool> reserved = new Variable<bool> { Name =  "Reserved" };
  // SendRequest workflow -ის განსაზღვრა ---
   this.Implementation = () => new Sequence
     {
       DisplayName = "SendRequest", Variables = {request, requestAddress,
              reserved
     },
 Activities =
  {
    new CreateRequest
     {
       ArealName = new InArgument<string>
          (env =>  ArealName.Get(env)),
       MajorDeputy = new InArgument<string>
          (env =>  MajorDeputy.Get(env)),
      Region = new InArgument<string>
          (env => Region.Get(env)),
      Request = new OutArgument<ElectionRequest>
           (env => request.Get(env)),
```

```
 RequestAddress = new OutArgument<string>
      (env => requestAddress.Get(env))
},
new Send
 {
  OperationName = "RequestElinfo",
  ServiceContractName = "IElectionSys",
  Content = SendContent.Create
     (new InArgument<ElectionRequest>(request)),
  EndpointAddress = new InArgument<Uri>
        (env => new Uri("http://localhost:" +
        requestAddress.Get(env) + "/ClientService")),
  Endpoint = new Endpoint
   {
     Binding = new BasicHttpBinding()
    },
 },
new WriteLine
 {
  Text = new InArgument<string>
      (env => "Request sent; waiting for response"),
  TextWriter = new InArgument<TextWriter>
      (env =>  Writer.Get(env))
 },
new WaitForInput<ElectionResponse>
 {
   ElinfomarkName = "GetResponse",
   Input = new OutArgument<ElectionResponse>
        (env => Response.Get(env))
},
```

```
    new WriteLine
    {
      Text = new InArgument<string> (env => "Response received from  " +
      Response.Get(env).Provider.BranchName + " [" +
      Response.Get(env).Reserved.ToString() + "]"),
      TextWriter = new InArgument<TextWriter> (env =>  Writer.Get(env))
    },
   }
  };
 }
}       // აქ უნდა დაემატოს 3.11 ლისტინგი -  დასასრული“ }
```

უნდა აღვნიშნოთ, რომ ყოველ WriteLine ქმედებას აქვს დამატებითი თვისება:

```
TextWriter = new ListBoxTextWriter()
```

ის მიუთითებს იმაზე, რომ ახალი კლასი ListBoxTextWriter, რომელიც იქნა რეალიზებული, უნდა იქნას გამოყენებული ამ ტექსტის დისპლეიზე გამოსატანად. ეს გამოიწვევს ტექსტის ასახვას lstEvents მართვის ელემენტში.

სხვა განსხვავება იმაშია, რომ მომხმარებლის ქმედება WaitForInput გამოიყენება Receive ქმედების ნაცვლად. აპლიკაცია მიიღებს საპასუხო შეტყობინებას უშუალოდ (პირდაპირ). როცა მიღებულ იქნება პასუხი, მაშინ აპლიკაცია აღადგენს მუშა პროცესს, რომელიც მიმდინარეობს ElectionResponse კლასში.

ყურადსაღებია, რომ მომხმარებლის ქმედება განისაზღვრება როგორც WaitForInput <ElectionResponse>, მიუთითებს რა, რომ გადასაცემი მონაცემები იქნება ElectionResponse კლასის.

3.6.11. ProcessRequest ბიზნესპროცესის რეალიზაცია

ProcessRequest ბიზნესპროცესი განსაზღვრება მოცემულია 3.11 ლისტინგში. ჩავამატოთ ეს კოდი ElectionWF.cs ფაილში.

```
// ---- ლისტინგი 3.11 ---------ElectionWF.cs დამატება ------
public sealed class ProcessRequest : Activity
 {
  public InArgument<ElectionRequest> request {get; set;}
  public InArgument<TextWriter> Writer { get; set; }
  public ProcessRequest()
   {
   // ცვლადების განსზღვრა ამ workflow-სთვის ---
   Variable<ElectionResponse> response =      new
              Variable<ElectionResponse> { Name = "response" };
   Variable<bool> reserved = new Variable<bool> { Name = "Reserved" };
   Variable<string> address = new Variable<string> { Name = "Address" };
// ProcessRequest workflow-ს განსაზღვრა ---
   this.Implementation = () => new Sequence
   {
     DisplayName = "ProcessRequest",
     Variables = { response, reserved, address },
     Activities =
     {
       new WriteLine
        {
          Text = new InArgument<string>(env => "Got request from: " +
                  request.Get(env).Requester.BranchName),
          TextWriter =  new InArgument<TextWriter>
                  (env  => Writer.Get(env))
     },
```

```
  new InvokeMethod
  {
    TargetType = typeof(ApplicationInterface),
    MethodName = "NewRequest",
    Parameters =
    {
      new   InArgument<ElectionRequest>(env   =>   request.Get(env))
    }
  },
new WaitForInput<bool>
    {
      ElinfomarkName = "GetResponse",
      Input = new OutArgument<bool>(env => reserved.Get(env))
    },
new CreateResponse
    {
      Request = new InArgument<ElectionRequest>
                (env => request.Get(env)),
     Reserved = new InArgument<bool>(env =>  reserved.Get(env)),
     Response = new OutArgument<ElectionResponse>
               (env => response.Get(env))
    },
new WriteLine
   {
     Text = new InArgument<string>(env => "Sending response to: " +
        request.Get(env).Requester.BranchName),
     TextWriter = new InArgument<TextWriter>
          (env => Writer.Get(env))
   },
```

```
new Send
 {
  OperationName = "RespondToRequest",ServiceContractName =
   "ILibraryReservation", EndpointAddress = new InArgument<Uri>
                (env => new Uri("http://localhost:" +
                 request.Get(env).Requester.Address +
                "/ClientService")),
   Endpoint = new Endpoint
    {
     Binding = new  BasicHttpBinding()
    },
  Content = SendContent.Create
           (new InArgument<ElectionResponse>(response))
         }
      }
    };
 }
}
```

იმის მაგივრად, რომ დაწყება იყოს Receive ქმედებით, რათა მიღებულ იქნას შემავალი მოთხოვნა, ElectionRequest გადასცემს მუშა პროცესს შემავალი არგუმენტის გამოყენებით. WriteLine ქმედება, რომელიც მოსდევს მას, ცნობს შემავალ მოთხოვნას.

InvokeMethod ქმედება გამოვიყენოთ მონაცემთა გადასაცემად აპლიკაციაში. ApplicationInterface კლასი მოხერხებულადაა შესრულებული ამ მიზნით. იგი უზრუნველყოფს მუშა პროცესს, რათა განხორციელდეს გამოძახება აპლიკაციაში. InvokeMethod ქმედება იძახებს ApplicationInterface კლასის NewRequest() მეთოდს ElectionRequest კლასში გადასაცემად.

გავხსნათ ApplicationInterface.cs ფაილი და დავამატოთ მეთოდი, რომელიც უბრალოდ იძახებს AddNewRequest ()-ს აპლიკაციაში:

```
public static void NewRequest(ElectionRequest request)
  {
    if (_app != null)
          _app.AddNewRequest(request);
  }
```

შემდეგი აქტიურობაა მომხმარებლის WaitForInput ქმედება, რომელიც გამოიყენებოდა SendRequest მუშა პროცესში.

ამჯერად იგი ელოდება Bool-შესატან მითითებას, იყო თუ არა დაჯავშნული დასახელება (სათაური). CreateResponse და WriteLine ქმედებები იგივეა. აქ გამოიყენებოდა SendReply ქმედება, ვინაიდან იგი იყო დაკავშირებული საწყის Receive ქმედებასთან.

ამ პროექტში, ვინაიდან არაა არავითარი Receive ქმედება, ჩვენ გამოვიყენებთ Send ქმედებას. საყურადღებოა, რომ EndpointAddress აწყობილია მისამართის გამოყენებით (პორტის ნომერი), რომელიც გათვალისწინებულია შესატან მოთხოვნაში.

6.3.12. აპლიკაციის რეალიზაცია

შემდეგი ბიჯი არის აპლიკაციის რეალიზაცია. არსებობს მოვლენათა რამდენიმე დამმუშავებელი (event handlers), რომელთა რეალიზაცია აუცილებელია, აგრეთვე მეთოდები, რომლებიც გამოიძახება სტატიკური ApplicationInterface კლასით.

6.3.13. ბიზნესპროცესების ეგზემპლარების მხარდაჭერა

აპლიკაცია ახდენს ბიზნესპროცესის ეგზემპლარების მონიტორინგს, ამიტომაც მას შეუძლია განაახლოს სწორი ეგზემპლარი. ამის შესრულება შესაძლებელია მარტივად ობიექტის ლექსიკონით.

გავხსნათ Election.xaml.cs ფაილი და დავამატოთ კლასის წევრები მომხმარებლის ServiceHost _sh სტრიქონის ქვემოთ:

```
private IDictionary<Guid, WorkflowApplication> _incomingRequests;

private IDictionary<Guid, WorkflowApplication> _outgoingRequests;
```

ისინი იყენებს ბიზნესპროცესის ეგზემპლარის იდენტიფიკატორს, როგორც ლექსიკონის გასაღებს და WorkflowApplication ობიექტს, როგორც მნიშვნელობას. ვინაიდან აპლიკაცია ამუშავებს ორივე მუშა პროცესს SendRequest და ProcessRequest, ამიტომაც საჭირო იქნება ლექსიკონის ორი ობიექტი. დავამატოთ კონსტრუქტორში კოდი ამ ობიექტების ინიციალიზებისთვის:

```
_incomingRequests = new Dictionary<Guid, WorkflowApplication>();

_outgoingRequests = new Dictionary<Guid, WorkflowApplication>();
```

საჭიროა კიდევ ერთი მცირე ცვლილება მომხმარებლის CreateRequest ქმედებაში. ბიზნესპროცესის ეგზემპლარის ID გამოყენებულ უნდა იქნას როგორც ElectionRequest კლასის RequestID ველი. აპლიკაცია მას გამოიყენებს პროცესის განახლების დროს. გავხსნათ CreateRequest.cs ფაილი და შევცვალოთ გამოძახება, რომელიც ქმნის ElectionRequest კლასს, ალტერნატიული კონსტრუქტორის გამოსაყენებლად, რომელიც ღებულობს მეხუთე პარამეტრს RequestID -თვის. დავამატოთ მუქი სტრიქონი კოდის შემდეგ ტექსტში:

```
// -- ElectionRequest კლასის შექმნა და მისი შევსება
// შესატანი არგუმენტებით -----
ElectionRequest r = new ElectionRequest
    (  ArealName.Get(context),
       MajorDeputy.Get(context),
       Region.Get(context),
      new Branch
```

```
    {
      BranchName = app.Settings["Branch Name"].Value,
      BranchID = new Guid(app.Settings["ID"].Value),
      Address = app.Settings["Address"].Value
    },
  context.WorkflowInstanceId  // ეს დაემატა!!!
 )
```

6.3.14. მოვლენათა დამმუშავებელი (Event Handlers)

ახალი მოთხოვნის შესაქმნელად მომხმარებელი შეავსებს:

მაჟორ _ დეპუტატის _ გვარის , ოლქის _ დასახელების , რეგიონის _ სახელის

ველებს და აამოქმედებს Send Request ღილაკს. ამ მოვლენის ღილაკის რეალიზება მოცემულია 3.12 ლისტინგში, Election.xaml.cs ფაილში.

```
// --- ლისტინგი 3.12----- Click Event ის რეალიზება -----
private void btnRequest_Click(object sender,RoutedEventArgs e)
 {
   // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად ---
   Dictionary<string, object> parameters = new Dictionary<string, object>();
   parameters.Add("MajorDeputy", txtMajorDeputy.Text);
   parameters.Add("ArealName", txtArealName.Text);
   parameters.Add("Regioni", txtRegioni.Text);
   parameters.Add("Writer", new ListBoxTextWriter(lstEvents));
  WorkflowApplication i = new WorkflowApplication(new SendRequest(),
                  parameters);
     _outgoingRequests.Add(i.Id, i);
      i.Run();
}
```

ამ მეთოდის პირველი ნაწილი ჩვენთვის ნაცნობია. იგი იყენებს ობიექტის ლექსიკონს შემავალი არგუმენტების შესანახად, რომლებიც

უნდა გადაეცეს ბიზნესპროცესს. შემდეგ იგი ქმნის WorkflowApplication-ს, რომლის კონსტრუქტორსაც გადაეცემა პარამეტრები:

ბიზნესპროცესების დეფინიცია
ობიექტის ლექსიკონი, რომელიც შეიცავს შემავალ არგუმენტებს

WorkflowApplication-ი შემდეგ ემატება _outgoingRequests კოლექციას. ბოლოს, ეგზემპლარი გაიშვება Run () მეთოდით.

მოთხოვნების სიის ფორმაზე მოთავსებულია ღილაკები Reserve და Cancel, რომელთაც იყენებს მომხმარებელი იმის მისათითებლად, თუ რომელი ელემენტი იყო გამოყენებადი.

3.13 ლისტინგი აღწერს ამ ღილაკებისთვის მოვლენათა დამმუშავებლების რეალიზაციას. დავამატოთ ეს მეთოდები Election.xaml.cs კლასში.

```
// -- ლისტინგი 3.13 --Reserve და Cancel ღილაკების რეალიზაცია --
 // -- Reserve ღილაკის მოვლენის დამმუშავებელი ---
private void Reserve(object sender, RoutedEventArgs e)
 {
    // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ----
   FrameworkElement fe = (FrameworkElement)sender;
   Guid id = (Guid)fe.Tag;
   ResumeBookmark(id, true);
 }
  // -- Cancel ღილაკის მოვლენის დამმუშავებელი ---
    private void Cancel(object sender, RoutedEventArgs e)
      {
        // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ----
       FrameworkElement fe = (FrameworkElement)sender;
       Guid id = (Guid)fe.Tag;          ResumeBookmark(id, false);
      }
```

```
private void ResumeBookmark(Guid id, bool bReserved)
  {
    WorkflowApplication i = _incomingRequests[id];
    try
      {
        i.ResumeBookmark("GetResponse", bReserved);
      }
   catch (Exception e)
     {
       AddEvent(e.Message);
     }
  }
```

მოვლენის დამმუშავებლები იღებს ბიზნესპროცესის ეგზემპლარის ID-ს ღილაკის Tag თვისებიდან. შემდეგ იძახებს ResumeBookmark () მეთოდს, მიაწოდებს true-ს ან falseს, იმისდა მიხედვით, თუ რომელი ღილაკი იყო ამოქმედებული.

ResumeBookmark () მეთოდი მიიღებს WorkflowApplication-ს _incomingRequestsკოლექციიდან და გამოიძახებს მის ResumeBookmark () მეთოდს. გადაეცემა სანიშნის სახელი (bookmark name) და მნიშვნელობა, რომელშიც ეგზემპლარი განახლდება (resumed).

6.3.15. ApplicationInterface მეთოდები

ჩვენ განვსაზღვრეთ ApplicationInterface კლასის სამი მეთოდი. ახლა უნდა უზრუნველვყოთ მათი რეალიზაცია MainWindow კლასში. გავხსნათ Election.xaml.cs ფაილი და ჩავამატოთ ამ მეთოდების რეალიზაცია 3.14 ლისტინგის მიხედვით.

```
//--ლისტინგი 3.14 --ApplicationInterface კლასის მეთოდების რეალიზაცია--
public void RequestElinfo(ElectionRequest request)
```

```
 {
   // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად ---
  Dictionary<string, object> parameters = new Dictionary<string, object>();
  parameters.Add("request", request);
  parameters.Add("Writer", new  ListBoxTextWriter(lstEvents));
 WorkflowApplication i = new
            WorkflowApplication(new ProcessRequest(), parameters);
  request.InstanceID = i.Id;
 _incomingRequests.Add(i.Id, i);
   i.Run();
}
public void RespondToRequest(ElectionResponse response)
{
  Guid id = response.RequestID;
  WorkflowApplication i = _outgoingRequests[id];
   try
    {
      i.ResumeBookmark("GetResponse", response);
    }
   catch (Exception e2)
     {
       AddEvent(e2.Message);
     }
   }
 public void AddNewRequest(ElectionRequest request)
 {
    this.requestList.Dispatcher.BeginInvoke
              (new Action(() =>  this.requestList.Items.Add(request)));
 }
```

RequestBook () მეთოდი ანალოგიურია btnRequest_Click () მეთოდის. იგი გამოიძახება მაშინ, როცა შემავალი შეტყობინება მიღებულია ServiceHost -დან და სერვის კონტრაქტის RequestBook მეთოდი მითითებულია. ის აგებს ობიექტის ლექსიკონს ერთი არგუმენტის შესანახად, ქმნის WorkflowApplication-ს, ამატებს მას _incomingRequests კოლექციაში, ხოლო შემდეგ აამოქმედებს მუშა პროცესს.

RespondToRequest () მეთოდი ასევე გამოიძახება ServiceHost-დან მიღებული შეტყობინებით. იგი გამოიძახება მაშინ, როცა RespondToRequest მეთოდია მითითებული. ეს ხდება მაშინ, როცა სხვა ფილიალები აგზავნის უკან პასუხს შემოსულ მოთხოვნაზე.

იგი ღებულობს WorkflowApplication-ს _outgoingRequests კოლექციიდან და აღადგენს სანიშნეს, გამავალს ElectionResponse კლასში.

AddNewRequest() გამოიძახება ProcessRequest მუშა პროცესით, როცა მიიღბა ახალი შეტყობინება. ეს ხდება InvokeMethod ქმედების დახმარებით. იგი უბრალოდ დაამატებს ჩანაწერს ListView-კონტროლის RequestList-ელემენტში. ვინაიდან ის გამოძახებულ უნდა იქნას ბიზნესპროცესის შესრულებად ნაკადში, Dispatcher კლასი გამოიყენებს შესასრულებლად Add () მეთოდს main window-ის შესრულებადი ნაკადით. Election.xaml.cs-ის სრული რეალიზაცია მოცემულია 3.15 ლისტინგში.

```
// -- ლისტინგი 3.15 -- Election.xaml.cs საბოლოო სახე -----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
```

```
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging; using System.Windows.Navigation;
using System.Windows.Shapes;
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
 using System.ServiceModel.Description;
using System.ServiceModel.Channels;
using System.Activities;
using System.Xml.Linq;
using System.Configuration;
namespace ElectionSys
  {
    public partial class MainWindow : Window
    {
     private ServiceHost _sh;
     private IDictionary<Guid, WorkflowApplication>  _incomingRequests;
     private IDictionary<Guid, WorkflowApplication>   _outgoingRequests;
     public MainWindow()
     {
       InitializeComponent();
       ApplicationInterface._app = this;
      _incomingRequests = new Dictionary<Guid, WorkflowApplication>();
      _outgoingRequests = new Dictionary<Guid,  WorkflowApplication>();
     }
private void Window_Loaded(object sender, RoutedEventArgs e)
  {  // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და
     // მისი ქსელური მისამართი ---
```

```
    Configuration config = ConfigurationManager.OpenExeConfiguration
                (ConfigurationUserLevel.None);
    AppSettingsSection app =
                (AppSettingsSection)config.GetSection("appSettings");
    string adr = app.Settings["Address"].Value;
 // ფილიალის სახელის გამოტანა ფორმაზე ----
   lblBranch.Content = app.Settings["Branch Name"].Value;
// ServiceHost-ის შექმნა ----
   _sh = new ServiceHost(typeof(ClientService));
// დასასრულის წერტილის (Endpoint) დამატება ----
   string szAddress = "http://localhost:" + adr +  "/ClientService";
   System.ServiceModel.Channels.Binding bBinding =
                        new BasicHttpBinding();
_sh.AddServiceEndpoint(typeof(IElectionSys), bBinding,  szAddress);
  // ServiceHost-ის გახსნა შეტყობინებების მისაღებად (listen) ---
 _sh.Open();
 }
private void Window_Unloaded(object sender, RoutedEventArgs e)
  {     // service host-ის დატოვება
   _sh.Close();
  }
 // ----- მოვლენათა დამმუშავებელი -----Event Handler-----
private void btnRequest_Click(object sender,RoutedEventArgs e)
 {    // Setup a dictionary object for passing parameters
   Dictionary<string, object> parameters = new Dictionary<string, object>();
   parameters.Add("MajorDeputy", txtMajorDeputy.Text);
   parameters.Add("ArealName", txtArealName.Text);
   parameters.Add("Regioni", txtRegioni.Text);
   parameters.Add("Writer", new ListBoxTextWriter(lstEvents));
```

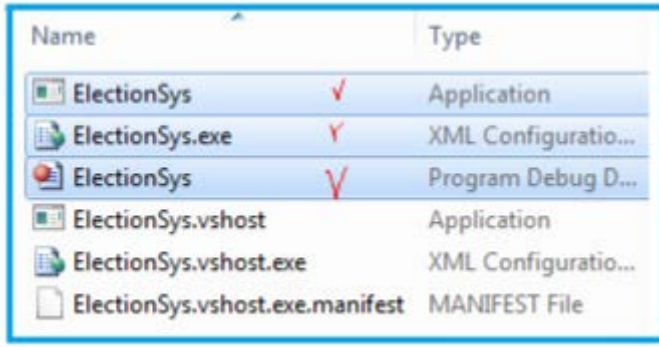
```
 WorkflowApplication i =  new WorkflowApplication(new SendRequest(),
                        parameters);
      _outgoingRequests.Add(i.Id, i);
        i.Run();
}
  // -- Reserve ღილაკის მოვლენის დამმუშავებელი ---
private void Reserve(object sender, RoutedEventArgs e)
  {   // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ----
    FrameworkElement fe = (FrameworkElement)sender;
    Guid id = (Guid)fe.Tag;         ResumeBookmark(id, true);
   }
  // -- Cancel ღილაკის მოვლენის დამმუშავებელი ---
 private void Cancel(object sender, RoutedEventArgs e)
  {   // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ---
    FrameworkElement fe = (FrameworkElement)sender;
    Guid id = (Guid)fe.Tag;
    ResumeBookmark(id, false);
  }
 private void ResumeBookmark(Guid id, bool bReserved)
  {
     WorkflowApplication i = _incomingRequests[id];
     try
      {
         i.ResumeBookmark("GetResponse", bReserved);
      }
    catch (Exception e)
       {              AddEvent(e.Message);
       }
   }
public void RequestElinfo(ElectionRequest request)
```

```
    { // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად ---
    Dictionary<string, object> parameters = new Dictionary<string,
                    object>();
     parameters.Add("request", request);
     parameters.Add("Writer", new ListBoxTextWriter(lstEvents));
     WorkflowApplication i =
              new WorkflowApplication(new ProcessRequest(), parameters);
     request.InstanceID = i.Id;
     _incomingRequests.Add(i.Id, i);
      i.Run();
   }
public void RespondToRequest(ElectionResponse response)
  {
     Guid id = response.RequestID;
    WorkflowApplication i = _outgoingRequests[id];
     try
      {
        i.ResumeBookmark("GetResponse", response);
      }
   catch (Exception e2)
      {
        AddEvent(e2.Message);
      }
   }
 public void AddNewRequest(ElectionRequest request)
  {
    this.requestList.Dispatcher.BeginInvoke (new Action(() =>
            this.requestList.Items.Add(request)));
    }
public ListBox GetEventListBox()
```

```
  {
    return this.lstEvents;
  }
  private void AddEvent(string szText)
    {
      lstEvents.Items.Add(szText);
    }
  }
}
```

6.3.16. აპლიკაციის ამუშავება

პროგრამული სისტემის ასამუშავებლად საჭიროა აპლიკაციის რამდენიმე ასლის (კოპიოს) ერთად გაშვება, თითოეული თავისი კონფიგურაციის ფაილის ვერსიით. თავიდან საჭიროა F6 კლავიშის ამოქმედება solution-ის (გადაწყვეტის) აღსადგენად და კომპილა-ტორის შენიშვნების აღმოსაფხვრელად. შევქმნათ ახალი ფოლდერი ElectionSys-ფოლდერის ქვეშ, რომელიც იძახებს ფილიალებს. შემდეგ დავაკოპიროთ ფილიალის ფოლდერში ფაილები, რომლებიც 3.63 ნახაზზეა ნაჩვენები.



ნახ.3.63. ფილიალის ფოლდერში ფაილების კოპირება

გავხსნათ ElectionSys.exe.config ფაილი (ფილიალის ქვეფოლდერში) და შევასწოროთ შემდეგნაირად:

<?xml version="1.0" encoding="utf-8" ?> <configuration> <appSettings> <add key="Branch Name" value="Olqi Ozurgeti"/> <add key="ID" value="{43E6DADD-4751-4056-8BB7-7459B5C361AB}"/>

<add key="Address" value="8730"/> <add key="Request Address" value="8000"/> </appSettings> </configuration>

შენიშვნა: თუ შედეგი მიღებულია შეცდომით, უნდა ვცადოთ აპლიკაციის გაშვება ადმინისტრატორის უფლებებით (!).

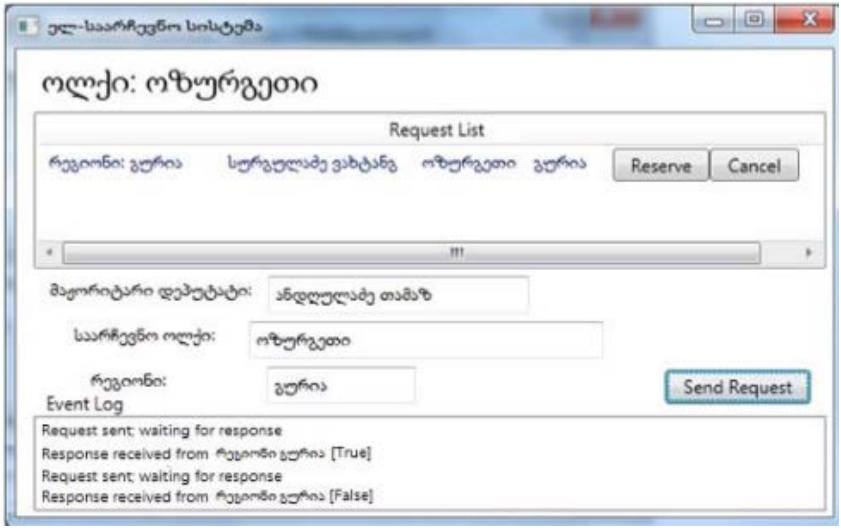
ფილიალის ფოლდერში ElectionSys.exe ფაილი ორჯერ დავკლიკოთ. აპლიკაცია ასე გამოიყურება (ნახ.3.64).

ნახ.3.64. ფილიალის ფორმა

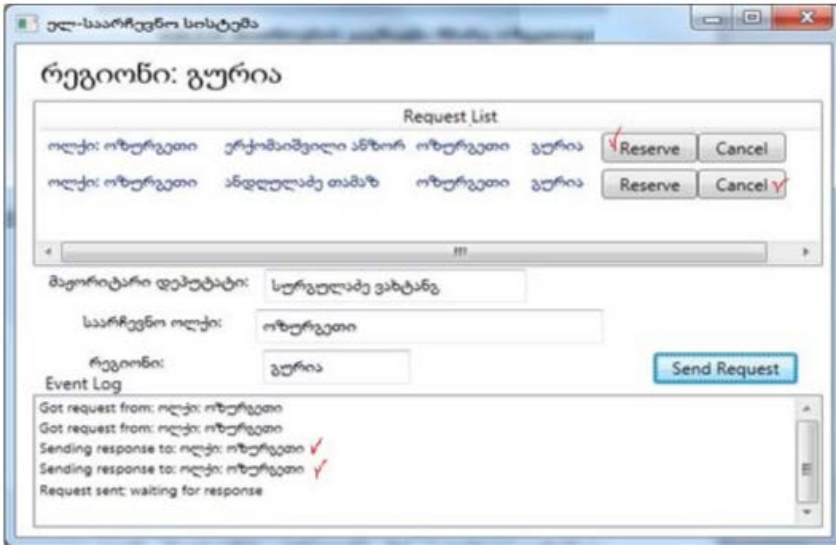
Visual Studio-ში F5-ით გავმართოთ აპლიკაცია. ანალოგიური ფანჯარა უნდა მივიღოთ, ოღონდ სათაურით - „საარჩევნო ოლქი“ (ან

„რეგიონალური საარჩევნო კომისია“ ან „ცენტრალური საარჩევნო კომისია“ და ა.შ.).

ერთ-ერთ აპლიკაციაში შევიტანოთ მაჟორიტარი დეპუტატის გვარი, მხარე (ოლქი), რეგიონი და ავამოქმედოთ ღილაკი „მოთხოვნის გაგზავნა“ (ნახ.3.65). მოთხოვნა უნდა გამოჩნდეს მეორე ფანჯრის მოთხოვნების სიაში. დააჭირეთ Reserve ღილაკს მეორე აპლიკაციაში (ნახ.3.66). გამოჩნდება შეტყობინება პირველი ფანჯრის მოვლენების ჟურნალში, რომ პასუხი მიღებულია.



ნახ.3.65. მოთხოვნის გაგზავნა ოლქიდან „ოზურგეთი“



ნახ.3.66. მოთხოვნის დამუშავება რეგიონში „გურია“

ვცადოთ რამდენიმე მოთხოვნის გაგზავნა ორივე ფანჯრიდან. ასევე შევამოწმოთ Cancel ღილაკი და დავრწმუნდეთ, რომ საპასუხო შეტყობინება მოვლენათა ჟურნალში (მეორე აპლიკაციისთვის) იქნება [False].

3.6.17. დასკვნა

ამრიგად, ჩატარებული სამუშაოების შედეგად შეიძლება შემდეგი სახის დასკვნების ჩამოყალიბება:

- ელექტრონული საარჩევნო სისტემა, როგორც კორპორაციული მართვის ობიექტი ელექტრონული მთავრობის შემადგენლობაში, მიეკუთვნება რთული და დიდი სისტემების კლასს. მისმა კომპლექსურმა ანალიზმა გვიჩვენა, რომ ასეთი სისტემების დასაპროექტებლად აუცილებელია ობიექტ-ორიენტირებული, პროცესორიენტირებული და სერვის-ორიენტირებული მოდელირების მეთოდების გამოყენება;

- საქართველოში არსებული ტრადიციული საარჩევნო სისტემის დიაგნოსტიკური გამოკვლევის საფუძველზე, აგრეთვე საზღვარგარეთული დემოკრატიული ინსტიტუტების გამოცდილების გათვალისწინებით ამ

სფეროში, განსაზღვრულ იქნა ძირითადი პრობლემები და ამოცანები, შემუშავდა მათი გადაწყვეტის გზები და ერთიანი ელექტრონული საარჩევნო სისტემის აგების კონცეფცია;

- ელექტრონული საარჩევნო სისტემის დაპროექტება, აპარატურული და პროგრამული რეალიზაცია უნდა განხორციელდეს მულტიმედიალური საშუალებების, განაწილებული, რელაციური ბაზების, კლიენტ-სერვერ არქიტექტურის და უსაფრთხო, საიმედო ქსელების ბაზაზე;

- ელექტრონული საარჩევნო სისტემის მულტიმედიალური მონაცემთა რელაციური ბაზების ეფექტურად დასაპროექტებლად და ასაგებად მიზანშეწონილია თანამედროვე CASE-ტექნოლოგიების გამოყენება, მოდელირების კატეგორიალური თეორიის და დაპროექტების ობიექტ–ორიენტირებული მეთოდების საფუძველზე;

- კლიენტ–სერვერ არქიტექტურის ლოგიკურად ერთიანი და ფიზიკურად განაწილებული მონაცემთა რელაციური ბაზების სისტემის დამუშავება შესაძლებელი გახდა ობიექტ–როლური მოდელირების პრინციპების საფუძველზე და შესაბამისი გრაფო–ანალიზური ინსტრუმენტების გამოყენებით;

- შემუშავებულია აგებული სისტემის მომხმარებელთა ინტერფეისები, ინსტრუქციები, დანერგვის და ექსპლუატაციის პროცესების ორგანიზაციული, ტექნიკური და იურიდიული ასპექტები. განსაზღვრულია სისტემისათვის საჭირო ტექნიკური საშუალებები და მათი შესაძლებლობები.

ლიტერატურა:

1. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: WPF ტექნოლოგია (ნაწ.1). სტუ, თბ., 2014.

2. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: Workflow ტექნოლოგია (ნაწ.2). სტუ, თბ., 2015.

3. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: WCF ტექნოლოგია (ნაწ.3). სტუ, თბ., 2015.

4.. Мак-Дональд М. WPF: Windows Presentation Fundation в .NET 3.5 с примерами на C# 2008 для профессионаюв. 2-е издание: Пер. с англ. - М. : ООО "И.Д. Вильяме". 2008

5. Petzold Ch. Applications=Code+Markup. A Guide to the MicroSoft Windows Presentation Foundation. St-Petersburg. 2008

6. Долженко А.И. Разработка приложений на базе WPF и Silverlight. –М., 2011, www.intuit.ru/department/se/dawpfsl/

7. Eberhardt C. WPF DataGrid Practical Examples. 2009. http://www.codeproject.com/Articles/30905/WPF-DataGrid-Practical-Examples.

8. სურგულაძე გ., თურქია ე., თოფურია ნ. ჰიბრიდული აპლიკაციების დაპროგრამების ავტომატიზაცია. საერთაშ. სამეცნიერო კონფერენცია „საინფორმაციო და კომპიუტერ. ტექნოლოგიები, მოდელირება და მართვა“ (მიძღვ. ივ.ფრანგიშვილი-85 წლისთავისადმი). სტუ, თბ., 2015. გვ. 69-73

9. სურგულაძე გ., ოხანაშვილი მ., კაშიბაძე მ., ნეფარიძე მ. თანამედროვე საინფორმაციო ტექნოლოგიები მარკეტინგული პროცესების და წარმოების მენეჯმენტში. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(17), თბილისი, 2014, გვ. 64-71.

10. სამხარაძე რ. SQL სერვერი. სტუ. „ტექნიკური უნივერსიტეტი“. თბ., 2009.

11. Петкович Д. Microsoft SQL Server 2012. Руководство для начинающих: Пер. с англ. — СПб.: БХВ-Петербург, 2013.

12. Create Spatial Index (Transact-SQL). https://msdn.microsoft.com-/en-us/library/bb934196.aspx

13. Spatial Indexes Overview. https://msdn.microsoft.com/en-us/en-ue/library/bb895265.aspx#decompose

14. Halpin T. ORM 2 Graphical Notation, Neumont University. 2005. http://www.orm.net/ pdf/ORM2_TechReport1.pdf.

15. სურგულაძე გ., ვედეკინდი ჰ., თოფურია ნ. განაწილებული ოფის-სისტემების მონაცემთა ბაზების დაპროექტება და რეალიზაცია UML-ტექნოლოგიით. `ტექნიკური უნივერსიტეტი", თბ., 2006

16. სურგულაძე გ., თოფურია ნ., ბაკურია კ., ლომიძე მ. საინფორმაციო სისტემის დაპროექტება ობიექტ-როლური მოდელირების და სერვისორიენტირებული არქიტექტურის ბაზაზე. სტუ-ს შრ.კრ., „მას".N1(17). 2014. გვ. 32-45.

17. Barker R. CASE Method: Entity Relationship Modelling. Reading, MA: Addison-Wesley Professional. 1990

18. Booch G., Jacobson I., rambaugh J. Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara, 1996

19. Anurag S. WCF: From a Beginner's perspective & a Tutorial. V, 4 Apr 2013. http://www.codeproject.com/Articles/566691/WCF-From-a-Beginners-perspective-a-Tutorial

20. Collins M.J. Beginning WF: Windows Workflow in .NET 4.0. ISBN-13 (pbk): 978-1-4302-2485-3 Copyright © 2010. USA. http://www.ebooks-it.net/ebook/beginning-wf.

21. სურგულაძე გ. ვიზუალური დაპროგრამება C#_2010 ენის ბაზაზე. სტუ, თბ., 2011

22. Surguladze Gia, Topuria N., Petriashvili L., Surguladze Giorgi. (2015). Modelling of Designing a Conceptual Schema for Multimodal

Freight Transportation Information System. WASET, World Academy of Scientific, Engineering and Technology, v.9, N11. ISSN 1307-6892, Spain, pp. 204-207

23. გოგიჩაიშვილი გ., სურგულაძე გიორგი. (2014). მულტი-მოდალური გადაზიდვების ბიზნესპროცესების ავტომატიზე-ბული მართვის კონცეფცია. სტუ-ს შრ.კრ. „მას“ N2(18). გვ.45-50

24. სურგულაძე გ., ბულია ი. კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. მონოგრ., სტუ. თბ., 2012

25. სურგულაძე გ., ურუშაძე ბ. საინფორმაციო სისტემების მენეჯმენტის საერთაშორისო გამოცდილება (BSI, ITIL, COBIT). სტუ. თბ., 2014

26. გოგიჩაიშვილი გ., ბოლხი გ., სურგულაძე გ., პეტრიაშვილი ლ. მართვის ავტომატიზებული სისტემების ობიექტ-ორიენტი-რებული დაპროექტების და მოდელირების ინსტრუმენტები (MsVisio, WinPepsy, PetNet, CPN). სტუ. თბ., 2013

27. თურქია ე. ბიზნეს–პროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. სტუ. თბ., 2010

28. სურგულაძე გ., გულიტაშვილი მ., კივილაძე ნ. Web-აპლიკაციათა ტესტირება, ვალიდაცია და ვერიფიკაცია. მონოგრ., სტუ. თბ., 2015

29. სურგულაძე გ., ც. ფხაკაძე, ა.კეკენეძე. ორგანიზაციული მართვის ბიზნესპროცესების მოდელირება და დაპროექტება. სტუ, თბილისი, 2016

30. სურგულაძე გიორგი. მულტიმოდალური გადაზიდვების ბიზნესპროცესების მართვის სისტემის ინფრასტრუქტურა და მისი იმიტაციური მოდელი. სტუ-ს შრ.კრ.„მას“, 2(20). 2015. გვ.108-123

31. მეიერ-ვეგენერი კ., სურგულაძე გ., ბასილაძე გ. საინფორმაციო სისტემების აგება მულტიმედიური მონაცემთა ბაზებით. მონოგრაფია. სტუ. თბილისი. 2014

32. სურგულაძე გ., თოფურია ნ., ბასილაძე გ., კივილაძე ნ., ნეფარიძე მ. ელექტრონული საარჩევნო სისტემა მულტიმედიური მონაცემთა ბაზებით და კლიენტ-სერვერ არქიტექტურით. GESJ: Computer Science and Telecommunications, 2014, N2(42). გვ.39-86

33. სურგულაძე გ., თოფურია ნ., ბასილაძე გ., ურუშაძე ბ., ლომიძე მ., გაბინაშვილი ლ. პროგრამული სისტემების მენეჯმენტი მულტიმედიალური აპლიკაციების დასაპროექტებლად და ასაგებად. VI საერთ. სამეცნ.პრაქტ. კონფ. „ინტერნეტი და საზოგადოება“. აკ.წერეთლის სახ.უნივ. ქუთაისი, 2013. გვ. 66-70

34. სურგულაძე გ., თოფურია ნ. მონაცემთა ბაზების მართვის სისტემები: ობიექტ-როლური მოდელირება (ORM / ERM, SQL Server). დამხმ.სახ., სტუ, თბ., 2007

35. ბასილაძე გ., სურგულაძე გ., გაბინაშვილი ლ. მულტი-მედიალური ელექტრონული საარჩევნო სისტემის პროგრამული უზრუნველყოფის დამუშავება. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(14), თბ., 2013, გვ. 234-239

36. SQL Server Security. https://msdn.microsoft.com/en-us/library/bb669074(v=vs.110).aspx

გადაეცა წარმოებას 10.06.2016 წ. ხელმოწერილია დასაბეჭდად 12.06.2016. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 17. ტირაჟი 100 ეგზ.

სტუ-ს „IT კონსალტინგის ცენტრი“ (თბილისი, მ.კოსტავას 77)