ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

შაქრო მელქაძე

დინამიური ძიება

კომპიუტერული მეცნიერების სამაგისტრო პროგრამა

ნაშრომი შესრულებულია კომპიუტერული მეცნიერების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: მიხეილ თუთბერიძე, ფიზ.-მათ. მეცნ. კანდიდატი

თბილისი

2015

სარჩევი

ანოტაცია 3
შესავალი 4
ამოცანის დასმა 6
ამოცანის ობიექტური მოდელი 8
ამოცანის ალგორითმული რეალიზაცია 11
მეთოდი `GenerateSqlScript` 11
მეთოდი `FormFilterScript` 14
მეთოდი `LoOperFilterOperation` 16
პროგრამული უზრუნველყოფის ტესტირება და ექსპლოატაცია 18
პროგრამული უზრუნველყოფის კოდი 24
სამომავლო გეგმები და პერსპექტივები 31
დასკვნა 32

ანოტაცია

წინამდებარე ნაშრომში შესწავლილია მონაცემთა ბაზაში ინფორმაციის ძიების დინამიური მოთხოვნის აგების საკითხი. აღნიშნული ამოცანა წარმოიქმნება მსხვილ ორგანიზაციებში საბოლოო მომხმარებლების მიერ მონაცემთა ბაზის ინტენსიური მოხმარებისას. ნაშრომზე მუშაობის ფარგლებში შექმნილია პროგრამული უზრუნველყოფა, რომელშიც რეალიზებულია მონაცემთა ბაზის შესახებ მიწოდებული მინიმალური ინფორმაციის საფუძველზე მოთხოვნის დინამიურად ფორმირების ალგორითმი. პროგრამულ უზრუნველყოფას გააჩნია ბიბლიოთეკის სახე და მარტივად ინტეგრირდება სხვა აპლიკაციებში. ნაშრომზე მუშაობის ფარგლებში ასევე შექმნილია აპლიკაცია, რომელშიც ინტეგრირებულია მოცემული ბიბლიოთეკა და რომელიც ტესტავს ამ ბიბლიოთეკის ფუნქციონალს.

Shakro Melkadze

Dynamic Search

In the present work the question of construction of dynamical query for information searching in database is investigated. The mentioned problem occurs in large organizations when end users use database systems intensively. In the scope of the present work the software is created in which the algorithm of dynamic construction of query based on minimal input information about database is implemented. The software is developed as dynamic linking library and can be easily integrated into other applications. In the scope of the present work also the application was created, in which the mentioned dynamic linking library is integrated and which tests the library's functional.

შესავალი

თანამედროვე კომპიუტერულ სამყაროში ინფორმაციის ძიება ერთერთი უმნიშვნელოვანესი ამოცანაა. აქ მთავარი არამარტო ძიების ახალი და ეფექტური ალგორითმების შემუშავებაა, არამედ ძიების არსებული მეთოდების უნივერსალური გამოყენების სქემის შემუშავებაც. ხშირ შემთხვევაში სწორედ მისი არარსებობა უქმნის პრობლემას საბოლოო მომხმარებელს, რათა მან ძიების არსებული მეთოდი მოარგოს თავის ამოცანას.

ნებისმიერი ზომის ორგანიზაცია დღითიდღე უფრო და უფრო დამოკიდებული ხდება მონაცემთა ბაზებზე. ეს ტენდენცია უკვე დიდი ხნის განმავლობაში გრძელდება. შესაბამისად, უფრო და უფრო საჭირო ხდება, დაიშვას ამ მონაცემთა ბაზებზე ჩვეულებრივი მომხმარებლები. ესენი არიან ისეთი მომხმარებლები, რომლებსაც არა აქვთ სპეციალური ცოდნა მონაცემთა ბაზების ადმინისტრირების სფეროში. ამ პრობლემის გადაწყვეტისთვის საუკეთესო გამოსავალია შეიქმნას “user-friendly” მომხმარებლისთვის მოსახერხებელი ინტერფეისი, ეს გაუიოლებს საქმეს არაპროფესიონალ მომხმარებლებს, დაამყარონ წარმატებული კავშირი მონაცემთა ბაზებთან თავიანთი სურვილის მიხედვით.

იმისთვის, რომ ბუნებრივი ენა გავხადოთ ხელმისაწვდომი, არსებობს ორი გზა. ერთის მხრივ, მენეჯმენტისთვის სასურველია მოხდეს დაშვება მონაცემთა ბაზებზე ყოველგვარი ექსპერტების და ძვირადღირებული პროცესების ჩარევის გარეშე. სწორედ ამიტომაც, ბუნებრივი ენის ინტერფეისი არის ის მეთოდი, რომელიც ასრულებს ექსპერტის ფუნქციას მონაცემთა ბაზებსა და გამოუცდელ მომხმარებლებს შორის. მეორეს მხრივ, ადამიანების ურთიერთობის პროცესი მონაცემთა ბაზებთან უფრო და უფრო ფართო სპექტრის მოთხოვნებს აყენებს, როგორც სამუშაო, ასევე თავიანთი ინდივიდუალური შესაძლებლობიდან გამომდინარე. ისინი მოელიან, და მოლოდინიც შესაბამისად გამართლებულია, რომ შემდგომში უფრო და უფრო დაიხვეწოს ბუნებრივი ენის ინტერფეისი, და რომ სისტემამ უფრო და უფრო უნდა შეძლოს ადამიანის ენაზე კომუნიკაციის უნარი და არა პირიქით.

თანდათანობით იზრდება კომპეტენციის ის დონე, რაც უნდა გააჩნდეს საბოლოო მომხმარებელს, რათა მან შეძლოს შედარებით მაღალკვალიფიციური აპლიკაციის მოხმარება.

მეორეს მხრივ, ძნელია საბოლოო მომხმარებელს მოსთხოვო კომპეტენციის ამაღლება კომპიუტერული ტექნოლოგიების სფეროში, მაშინ როცა მას არა აქვს შესაბამისი აკადემიური განათლება. ამის გათვალისწინებით, აუცილებელი ხდება აპლიკაციების იმგვარად შემუშავება, რომ საბოლოო მომხმარებელმა შეძლოს აპლიკაციის ისეთნაირად მომართვა, რომ ის მოარგოს თავის ამოცანას.

საბოლოო მომხმარებლის წინაშე ძალიან ხშირად ისმის მონაცემთა ბაზაში ძიების ამოცანები, რომელთა განხორციელებაც რთულდება იმის გამო, რომ საბოლოო მომხმარებელი ბოლომდე არ იცნობს ბაზის სტრუქტურას ან არ იცნობს მონაცემთა ბაზის მართვის სისტემას ან არ იცის SQL ენა. სწორედ ამიტომ საჭირო ხდება რაიმე უნივერსალური მექანიზმის შემუშავება, რომელიც საბოლოო მომხმარებელს მონაცემთა ბაზის მართვის სისტემისა და თავად მონაცემთა ბაზის სტრუქტურის ცოდნის გარეშეც მისცემს საშუალებას, მოიძიოს მონაცემთა ბაზიდან მისთვის საჭირო ინფორმაცია.

წინამდებარე ნაშრომში შემოთავაზებულია მონაცემთა ბაზაში ინფორმაციის უნივერსალური ძიების მეთოდიკა, რომლის გათვალისწინებითაც შემუშავებულია შესაბამისი პროგრამული უზრუნველყოფა დინამიურად მიერთებადი ბიბლიოთეკის სახით. აღნიშნული პროგრამული უზრუნველყოფა არის მუშა მდგომარეობაში და შესაძლებელი მისი ექსპლოატაცია .NET პლატფორმაზე მომუშავე ნებისმიერი კომპიუტერული აპლიკაციიდან.

ამოცანის დასმა

მონაცემთა ბაზაში ინფორმაციის ძიების წარმოებისათვის არსებობს საკმაოდ ეფექტური მექანიზმი, რომელსაც SELECT ინსტრუქციას უწოდებენ. აღნიშნული ინსტრუქცია საშუალებას გვაძლევს, ცხრილიდან ამოვირჩიოთ ჩვენთვის საინტერესო სტრიქონების ნაკრები, რისთვისაც ფილტრის სახით გამოსაყენებელ ლოგიკურ გამოსახულებას ვუთითებთ, თუმცა SELECT ინსტრუქციის გამოყენების დროს ჩვენ ვალდებული ვართ დავაკონკრეტოთ ცხრილის სახელი და ვალდებული ვართ, რომ ასევე დავაკონკრეტოთ სვეტების სახელები, რომლებშიც წარმოებს ძიება. სწორედ ამის გამო აუცილებელი ხდება, რომ გარკვეული წარმოდგენა გვქონდეს მონაცემთა ბაზის აგებულებაზე, რაც საბოლოო მომხმარებლისთვის ცოტა ძნელად მიღწევადია. ამის გათვალისწინებით, აუცილებელი ხდება რომ შეიქმნას რაღაც შუალედური რგოლი საბოლოო მომხმარებელსა და მონაცემთა ბაზის მართვის სისტემას შორის, რომელიც თავის თავზე აიღებს საბოლოო მომხმარებლის მიერ მოსაძებნი მონაცემისათვის შესაბამისი SQL მოთხოვნების ფორმირებას და მათ შესრულებას. შუალედური რგოლი უნდა შეიქმნას, როგორც ავტომატიზებული აპლიკაცია, რომლის შემავალი მონაცემებიც იქნება საძიებო ინფორმაცია, ხოლო გამომავალი მონაცემები იქნება SQL ტექსტი.

არსებობს ბევრი პლატფორმა, რომლითაც ჩვენ შეგვიძლია შევქმნათ მონაცემთა ბაზებზე ორიენტირებული აპლიკაცია, თუმცა, ჩვენი აზრით, განსაკუთრებული მდგომარეობა უკავია ADO.NET ტექნოლოგიას, რომელიც საშუალებას გვაძლევს, რომ დინამიურად ავაგოთ მონაცემთა ბაზის ობიექტური მოდელი და შემდგომ ვიმუშაოთ მონაცემთა ბაზასთან ობიექტური მოდელის გამოყენებით. ეს უკვე თავისთავად მიუნიშნებს იმაზე, რომ SQL ტექსტის დაწერის აუცილებლობა მინიმუმამდეა დაყვანილი. ADO.NET ტექნოლოგია საშუალებას გვაძლევს, რომ ნებისმიერი სირთულის მონაცემთა ბაზიდან ინფორმაცია ამოვიღოთ ბაზის რელაციების და იერარქიულობის გათვალისწინებით. სწორედ ამიტომ, შუალედური რგოლის შესაქმნელად უნდა გამოყენებულ იქნას ADO.NET ტექნოლოგია.

მონაცემთა ბაზასა და საბოლოო მომხმარებელს შორის შუალედური რგოლი უნდა რეალიზებულ იქნას, როგორც დინამიურად მიერთებადი ბიბლიოთეკა, რომელიც

ხელმისაწვდომი იქნება სხვა აპლიკაციებისთვის. ბიბლიოთეკას უნდა გააჩნდეს შემავალი მონაცემები, რომელიც მოიცავს საძიებო მონაცემებს და ასევე გარკვეულ ინფორმაციას მონაცემთა ბაზის აღნაგობის შესახებ. ბიბლიოთეკამ უნდა განახორციელოს მიწოდებული ინფორმაციის ანალიზი და ამის საფუძველზე უნდა მოახდინოს შესაბამისი SQL მოთხოვნების გენერაცია. ბიბლიოთეკის მიერ დაგენერირებული SQL მოთხოვნების ტექსტი ხელმისაწვდომი უნდა გახდეს ბიბლიოთეკის გამომყენებელი სხვა პროგრამისათვის. იმის გათვალისწინებით, რომ ბიბლიოთეკის ამოცანა არის რეალურად SQL ტექსტის გენერაცია და ის არ არის მიჯაჭვული არც ერთ ტექნოლოგიაზე, ამიტომ SQL ტექსტი თავსებადი უნდა იყოს მონაცემთა ბაზის მართვის სისტემების უმეტესობასთან. აქედან გამომდინარე, SQL ტექსტი, რომელიც დაგენერირდება ბიბლიოთეკის მიერ, არ უნდა შეიცავდეს ისეთ ინსტრუქციებს, რომელიც სპეციფიურია კონკრეტული მონაცემთა ბაზის მართვის სისტემისათვის.

ბიბლიოთეკა უნდა იყოს მულტიპლატფორმული, რაც გულისხმობს რომ მისი გამოყენება უნდა შეიძლებოდეს ნებისმიერი აპლიკაციიდან, რომელსაც აქვს COM ინტერფეისის მხარდაჭერა, იგი უნდა რეგისტრირდებოდეს ოპერაციულ სისტემაში, ენიჭებოდეს გარკვეული იდენტიფიკატორი და შემდგომ შესაძლებელი უნდა იყოს მისი ჩატვირთვა სწორედ ამ იდენტიფიკატორით.

ამოცანის ობიექტური მოდელი

ამოცანის კომპიუტერული რეალიზაციის მიზნით გამოყენებულ იქნა ობიექტზე ორიენტირებული დაპროგრამების ენა, რამაც შესაძლებელი გახადა რომ აგებული ყოფილიყო ამოცანის ობიექტური მოდელი, კერძოდ შემუშავებულ იქნა კლასი Query რომელიც წარმოადგენს ინფორმაციის მატარებელს განსახორციელებელი მოთხოვნის შესახებ, კერძოდ მას გააჩნია შემდეგი კომპონენტები:

- EntityName: მონაცემთა ბაზის ცხრილის სახელი, საიდანაც ვაპირებთ მონაცემების ამოღებას;
- Schema: რომელ მონაცემთა ბაზაზეც ვმუშაობთ იმ მონაცემთა ბაზის სქემის სახელი;
- SelectFields: SelelectFields ტიპის ობიექტების კოლექციას, კოლექციაში მოვათავსებთ ამოსაღები ინფორმაციის ველებს SELECT ინსტრუქციაში;
- LogicalGroups:ველი შეიცავს LogicalGroup ტიპის ობიექტების კოლექციას, ამ ველის მიხედვით შეგვიძლია გადავცეთ დინამიურად შედგენილი ფილტრის მონაცემები. ველის დახმარებით აიგება Where ინსტრუქცია;
- SubQueries: ველი შეიცავს SubQuery ტიპის ობიექტების კოლექციას. ამ ველის გამოყენებით შესაძლებელია დინამიურად განვსაზღვროთ ამოსაღები წყაროს სიმრავლე. იგი მონაწილეობას იღებს INNER JOIN ინსტრუქციის ჩამოყალიბებაში.

კლასი SelectField გამოიყენება ცხრილის ველების შესახებ გარკვეული ინფორმაციის შესანახად. იგი შედგება შემდეგი კომპონენტებისაგან:

- FieldFullName – ცხრილის ველის სრული სახელი, ანუ: [ცხრილისსახელი].[ველისსახელი], ხოლო ბრძანების აწყობის დროს ამ შაბლონს წინ დაემატება გადაცემული სქემის სახელი.
- ModificatorId – ველი Int ტიპის არის და ამოიღებს ბაზიდან შესაბამის მოდიფიკატორის სახელს. თუ ამ ველს მივუთითებთ ბრძანების ფორმირების დროს, ეს მნიშვნელობა მონაწილეობას მიიღებს ORDER BY ინსტრუქციის ჩამოყალიბებაში.

ლოგიკური ოპერაციების განსახორციელებლად გამოიყენება კლასი LogicalGroup, რომელიც შედგება შემდეგი ველებისაგან:

- LogicalOperation: ველი ენუმერაციის ტიპის არის და ვუთითებთ ფილტრის ოპერაციას მისი შესაძლო მნიშვნელობებია:
    - NonOperation
    - And
    - Or
- LogLevel: ველი განსაზღვრავს ფილტრის დონეს. 1 ნიშნავს ზედა დონეს
- Clauses: ველი შეიცავს Clause ტიპის ობიექტების კოლექციას. ამ ველის დახმარებით გადავცემთ დასაფილტრი ველების სიას თავისი პრედიკატებით და მნიშვნელობებით
- SubLogicalGroups: ველი შეიცავს LogicalGroup ტიპის ობიექტის კოლექციას, ეს ველი საჭიროა იმისთვის რომ ჩადგმული ფილტრი დავაგენერიროთ.

ქვემოთხოვნის შესანახად გამოიყენება კლასი SubQuery, რომელიც შედგება შემდეგი ველებისგან:

- SetOperation – ველი ენუმერაციის ტიპის არის და გადაეცემა სიმრავლის ოპერაცია. მისი შესაძლო მნიშვნელობებია:
    - Join
    - InnerJoin
    - LeftJoin
    - RigthJoin
    - CrossJoin
- EntityName: ამ ველში ვუთითებთ ცხრილის სახელს რომეზლეც ვახორციელებთ ოპერაციას.
- LogicalGroups: ველი არის LogicalGroup ტიპის ობეიქტების კოლექცია, ამ ველში ვუთითებთ სიმრავლეეების გადაბმის წერტილებს. ეს ველი მონაწილებს On ინსტრუქციის აწყობაში.

გასაფილტრი პირობის ფორმირების მიზნით შექმნილია კლასი Clause, რომელიც შედგება შემდეგი ველებისგან:

- FullFieldName: ველში ეთითება დასაფილტრი ველის სრული სახელი ანუ [ცხრილის სახელი]. [ველის სახელი] ფორმატში.
- ValueText: ეთითება დასაფილტრი ველის მნიშვნელობა.
- ValueType: ველი ენუმერაციაა, ის აღნიშნავს ველის ტიპს, მისი მნიშვნელობებია:
    - BigInt
    - Text
    - Bit
    - DateTime
    - Set
    - Bin
    - Money
    - Object
    - Int
    - Float
    - Double
    - Date
- JoinedFullFieldName – ველში ეთითება ველის მნიშვნელობა, ეს ხდება მაშინ როცაValueText ველი ცარიელია და ორი ბაზის ველი გვინდა შევადაროთ ერთმანეთს. ამ შემთხვევაში შესადარებელი მნიშვნელობაც ბაზისველია.
- PredicatId – ველში ეთითება ბაზაში მყოფი პრედიკატის ID. პრედიკატებია: “=“, “>“, “<“ და ასე შემდეგ...

ამოცანის ალგორითმული რეალიზაცია

ბიბლიოთეკაში რეალიზებულია QOMEngine კლასი, რომელსაც აქვს ერთადერთი public მეთოდი `GenerateSqlScript,` რომელიც პარამეტრად ღებულობს Query კლასის ობიექტს, რისი მეშვეობითაც დგინდება დასაგენერირებელი Sql მოთხოვნის პარამეტრები. გარდა ამისა, კლასს ასევე გააჩნია შემდეგი private მეთოდები :`FormFilterScript` და `LogOperFilterOperation`, რომელთა ფუნქციონალს განვიხილავთ ქვემოთ.

მეთოდი `GenerateSqlScript`

ამოცანის საბოლოო შედეგს იძლევა მეთოდი `GenerateSqlScript`, რომელიც პარამეტრად მიიღებს Query ტიპის ობიექტს, ამ ობიექტში კი ინახება Sql მოთხოვნის დასაგენერირებლად აუცილებელი პარამეტრები. Sql ტექსტის გენერაცია ხდება შემდეგნაირად:

1) იქმნება Sql მოთხოვნის შაბლონი;
2) ხდება Sql მოთხოვნის შაბლონში SELECT ველების ჩამატება;
3) ხდება Sql მოთხოვნის შაბლონში FROM კონსტრუქციის ფორმირება;
4) თუ მომხმარებლის მიერ გადმოცემულია Join ბრძანება, მაშინ ხდება From კონსტრუქციის აგება დინამიურად;
5) ხდება WHERE კონსტრუქციის ფორმირება;
6) ხდება ORDER კონსტრუქციის ფორმირება.

Sql მოთხოვნის შაბლონის ასაგებად გამოიყენება StringBuilder ობიექტი, რომელიც საშუალებას იძლევა, რომ ავაგოთ ტექსტი რესურსების მინიმალური დანახარჯებით.

GenerateSqlScript – ფუნქციის მუშაობის პრინციპი მდგომარეობს შემდგომში:

ფუნქციას გადაეცემა ზემოთ ნახსენები Query ტიპის ობიექტი და მისი დახმარებით დააგენერირებს SQL ბრძანებას და დააბრუნებს მას სტრიქონის სახით. განვიხილოთ კოდის ფრაგმენტები:

StringselFmt = "SELECT {0} FROM {1} {2} WHERE {3} {4}";selFmt ცხრილში მოთავსებულია შაბლონი,თუ როგორ უნდა გამოიყურებოდეს დასაბრუნებელი ბრძანება. თითოეული ბრძანების შესაქმნელად ვიყენებთ:

```
StringBuilder whereClause = new StringBuilder();
//ფილტრისთვის
StringBuilder modifierStr = new StringBuilder();
// რიგითობის განსაზღვრისთვის
StringBuilder subQueries = new StringBuilder();
// წყაროს გენერაციისთვის
StringBuilder selList = new StringBuilder();
// ამოსაღები ველებისთვის
```

whereClause ცვლადის ინიციალიზაციისთვის გამოიძახება FormFilterScript ფუნქცია. ქვემოთ მოყვანილია ფილტრის გენერაციის კოდის ფრაგმენტი:

```
whereClause.Append(FormFilterScript(schema,
    query.LogicalGroups, null));
```

subQueries ცვლადის ინიციალიზაციას აკეთებს შემდეგი კოდის ფრაგმენტი:

```
if (query.SubQueries != null){
    foreach (var sq in query.SubQueries){
    subQueries.Append(GetSetOperation((
    new SetOperationRequestModel{ ID =(int)sq.SetOperation
    })).Name + " ");
subQueries.Append(schema + "." + sq.EntityName + " ON ");
subQueries.Append(FormFilterScript(schema,
sq.LogicalGroups, null));
                }
            }
```

აღნიშნულ კოდის ფრაგმენტში წყაროს გენერაციისთვის გამოიძახება FormFilterScript ფუნქცია და GetSetOperation ფუნქცია, ის დააბრუნეს ბაზიდან მოცემული ოპერაციის მნიშვნელობას.

selList და modifierStr ცვლადებს ორივეს ერთად ინიციალიზაციას უკეთებს:

```
if (query.SelectFields != null)
{
    inti = 0;
    string coma = " ";
    foreach (varsfinquery.SelectFields)
    {
    String str = schema + "." + sf.FieldFullName;
    selList.Append(str);
    i += 1;
          if (sf.ModificatorId != 0)
          {
            modifierStr.Append(coma +
      GetQueryModifier(newQueryModifierRequestModel { ID =
      sf.ModificatorId }).Name + " " + str);coma = ",";
}
if (i<query.SelectFields.Count())
    selList.Append(",");
    }
}
```

კოდის ფრაგმენტი: ეს ფრაგმენტი იყენებს GetQueryModifer ფუნქციას, რომელიც აბრუნებს ბაზაში განსაზღვრულ მოდიფიკატორის მნიშვნელობას.

ბოლოს ხდება selFmt ცვლადის ფორმატში ზემოთ აღნიშნული ცვლადების ჩანაცვლება, კოდის ფრაგმენტში:

```
String res = string.Format(selFmt, selList, entityName,
  subQueries, whereClause, modifierStr);
```

მეთოდი FormFilterScript

ფუნქციას გადაეცემა პარამეტრად მონაცემთა ბაზის სქემის სახელი LogicalGroup ტიპის ობიექტების კოლექცია და LogicalGroup ტიპის ობიექტი. ფუნქცია ააგებს ფილტრს და ამისთვის გამოიყენებს LogOperFilterOperation ფუნქციას. რადგანაც LogicalGroup კლასის სტრუქტურა მოიცავს თავისივე თავს, ანუ შეიძლება ჩადგმული ფილტრები გვქონდეს, გვექნება გარკვეული ხის სტრუქტურა. ამიტომაც ფილტრების აგების დროს FormFilterScript ფუნქცია რეკურსიულად მუშაობს და იძახებს თავისთავს. ქვემოთ მოყვანილია ამ ფუნქციის ფრაგმენტი:

```
StringBuilder flt = new StringBuilder("( ");

string logoper = (lg == null) ? " " :
lg.LogicalOperation.ToString();

string op = " ";
if (lg == null)
{
if (subLgs != null&&subLgs.Count> 0)
{
foreach (var lg1 in subLgs.Where(e =>e.LogLevel == 1))
{
flt.Append(" " + op);
flt.Append(FormFilterScript(schema, subLgs, lg1));
op = " " + lg1.LogicalOperation.ToString() + " ";
        }
     }
}
else
{
flt.Append(LogOperFilterOperation(schema, lg.Clauses,
op, logoper));
```

```
if (lg.SubLogicalGroups !=
null&&lg.SubLogicalGroups.Count> 0)
  {
if (!flt.ToString().Trim().Equals("(")) op = logoper;
foreach (var lg1 in lg.SubLogicalGroups)
         {
flt.Append(" " + op + " ");
flt.Append(FormFilterScript(schema,subLgs, lg1));
op = " " + logoper + " ";
         }
   }
}
returnflt.Append(" )");
```

მეთოდი `LoOperFilterOperation`

ფუნქციას გადაეცემა პარამეტრებად: მონაცემთა ბაზის სქემის სახელი, Clause ტიპის ობიექტების კოლექცია, ოპერაცია და ლოგიკური ოპერატორი, ის ამ პარამეტრების დახმარებით ააწყობს უშუალოდ თითოეულ ფილტრს ცალ-ცალკე. ეს ფუნქცია დამოუკიდებელი არ არის ის გამოიძახება FormFilterScript ფუნქციიდან გარკვეულ დროს. ფუნქცია FieldValueForSqlScript დახმარებით გაარკვევს, რა ტიპის არის გადმოცემული ValueText ის მნიშვნელობა, ხოლო GetPredicat-ის დახმარებით ბაზიდან ამოიღებს PredicatId-ის მიხედვით საჭირო პრედიკატის მნიშვნელობას. პროგრამული კოდი მოყვანილია ქვემოთ:

```
StringBuilder flt = new StringBuilder("");
if (clauseList != null&&clauseList.Count> 0)
{
foreach (var cl in clauseList)
  {
flt.Append(op);
string f1 = schema + "." + cl.FullFieldName;
varpred = GetPredicat(newPredicatRequestModel { ID =
cl.PredicatId });

if (cl.ValueText != null&& !cl.ValueText.Equals(""))
 {
String constVal =
DataValueTransformation.FieldValueForSqlScript(
cl.ValueType, cl.ValueText);
if (pred.TypeID == 1)
   {
flt.Append(f1 + " " + pred.Name + " " + constVal);
   }
else
{
StringBuilder tt = newStringBuilder(pred.FuncTemplate);
tt.Replace("c1", f1);
```

```
string ss = cl.ValueText;
if (ss[0] == '\'') ss = ss.Substring(1, Math.Max(0,
ss.Length - 2));
tt.Replace("c2", ss);
flt.Append(tt);
   }
 }
else
 {
if (pred.TypeID == 1)
{
flt.Append(f1 + " " + pred.Name + " " +
(string.IsNullOrWhiteSpace(cl.JoinedFullFieldName) ? "" :
(schema + "." + cl.JoinedFullFieldName)));
}
else
{
StringBuildertt = new StringBuilder(pred.FuncTemplate);
tt.Replace("c1", f1);
tt.Replace("c2", "+" + schema + "." +
cl.JoinedFullFieldName + "+");
flt.Append(tt);
    }
  }
op = " " + logoper + " ";
}
}
return flt;
```

პროგრამული უზრუნველყოფის ტესტირება და ექსპლოატაცია

ნაშრომზე მუშაობის ფარგლებში შექმნილი პროგრამული უზრუნველყოფის ტესტირების მიზნით შექმნილ იქნა სატესტო აპლიკაცია, რომელშიც ინტეგრირებულია მოცემული პროგრამული უზრუნველყოფა და რომელშიც გათვალისწინებული იქნა პროგრამული უზრუნველყოფის ფუნქციონალის სრული ტესტირება.

სატესტო აპლიკაციის პირველ ეკრანზე ეთითება ლოგიკური ოპერატორი, რომელიც უნდა დაკომპლექტდეს ლოგიკური გამოსახულებებით. ლოგიკური გამოსახულება შედგება შემდეგი კომპონენტებისაგან

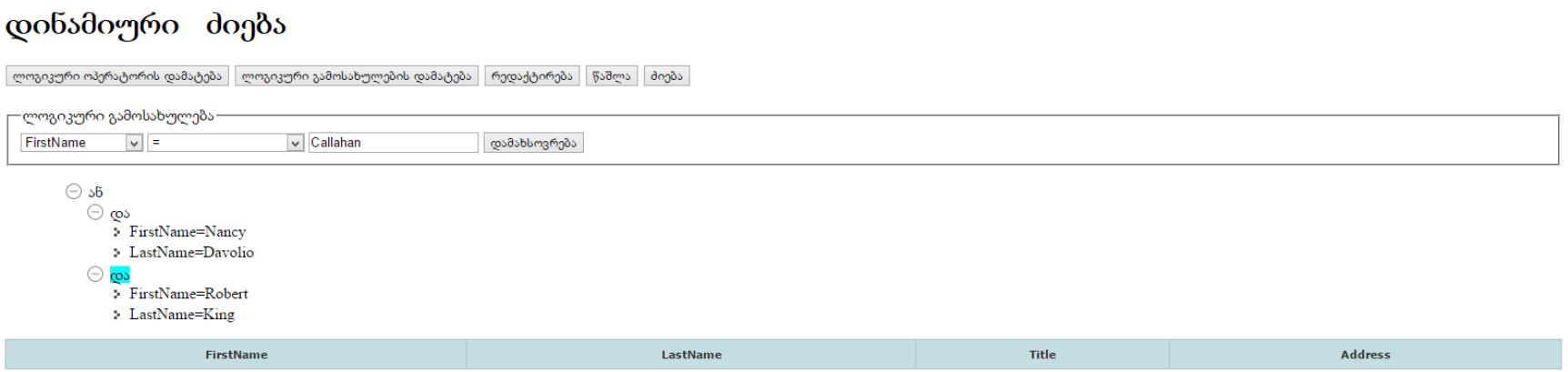
- ველის დასახელება;
- პრედიკატი;
- ველის სასურველი მნიშვნელობა.

აქვე შესაძლებელია, რომ მოხდეს ლოგიკური გამოსახულების რედაქტირება, ან წაშლა. (იხ. სურათი 1)

მას შემდეგ, რაც დავაკომპლექტებთ ლოგიკურ ოპერატორს ლოგიკური გამოსახულებებით, უნდა მოხდეს დაჭერა ღილაკზე „ძიება“, რის შედეგადაც მოხდება მოთხოვნის დინამიური ფორმირება, რომელიც გადაეგზავნება მონაცემთა ბაზას და რომელიც, თავის მხრივ, დაგვიბრუნებს ძიების შედეგებს (სურათი 2).

ძებნის პარამეტრების სხვადასხვა კომბინაციებია მოცემული სურათებზე 3, 4, 5.

განხილული მაგალითები ნათელს ხდის, თუ რა სახის მოთხოვნების დაგენერირებაა შესაძლებელი ნაშრომზე მუშაობის ფარგლებში შექმნილი ბიბლიოთეკით.


დინამიური ძიება
ლოგიკური ოპერატორის დამატება
ლოგიკური გამოსახულების დამატება
რედაქტირება
წაშლა
ძიება
ლოგიკური გამოსახულება
FirstName
=
Callahan
დამახსოვრება
ან
და
FirstName=Nancy
LastName=Davolio
და
FirstName=Robert
LastName=King
FirstName
LastName
Title
Address

სურათი 1

დინამიური ძიება

ლოგიკური ოპერატორის დამატება | ლოგიკური გამოსახულების დამატება | რედაქტირება | წაშლა | ძიება

ლოგიკური გამოსახულება

FirstName | = | Callahan | დამახსოვრება

- ან
  - და
    - FirstName=Nancy
    - LastName=Davolio
  - და
    - FirstName=Robert
    - LastName=King

FirstName	LastName	Title	Address
Nancy	Davolio	Sales Representative	507 - 20th Ave. E. Apt. 2A
Robert	King	Sales Representative	Edgeham Hollow Winchester Way

სურათი 2

დინამიური ძიება

ლოგიკური ოპერატორის დამატება | ლოგიკური გამოსახულების დამატება | რედაქტირება | წაშლა | ძიება

- ან
  - ან
    - და
      - FirstName=Nancy
      - LastName=Davolio
    - და
      - FirstName=Robert
      - LastName=King
  - City=London

FirstName	LastName	Title	Address
Nancy	Davolio	Sales Representative	507 - 20th Ave. E. Apt. 2A
Steven	Buchanan	Sales Manager	14 Garrett Hill
Michael	Suyama	Sales Representative	Coventry House Miner Rd.
Robert	King	Sales Representative	Edgeham Hollow Winchester Way
Anne	Dodsworth	Sales Representative	7 Houndstooth Rd.

სურათი 3

დინამიური ძიება

ლოგიკური ოპერატორის დამატება | ლოგიკური გამოსახულების დამატება | რედაქტირება | წაშლა | ძიება

- ან
  - ან
    - და
      - FirstName=Nancy
      - LastName=Davolio
    - და
      - FirstName=Robert
      - LastName=King
  - City=London

FirstName	LastName	Title	Address
Nancy	Davolio	Sales Representative	507 - 20th Ave. E. Apt. 2A
Steven	Buchanan	Sales Manager	14 Garrett Hill
Michael	Suyama	Sales Representative	Coventry House Miner Rd.
Robert	King	Sales Representative	Edgeham Hollow Winchester Way
Anne	Dodsworth	Sales Representative	7 Houndstooth Rd.

სურათი 4

დინამიური ძიება

ლოგიკური ოპერატორის დამატება | ლოგიკური გამოსახულების დამატება | რედაქტირება | წაშლა | ძიება

- ან
  - ან
    - და
      - FirstName=Nancy
      - LastName=Davolio
    - და
      - FirstName=Robert
      - LastName=King
  - City=London
  - Address=4110 Old Redmond Rd.

FirstName	LastName	Title	Address
Nancy	Davolio	Sales Representative	507 - 20th Ave. E. Apt. 2A
Margaret	Peacock	Sales Representative	4110 Old Redmond Rd.
Steven	Buchanan	Sales Manager	14 Garrett Hill
Michael	Suyama	Sales Representative	Coventry House Miner Rd.
Robert	King	Sales Representative	Edgeham Hollow Winchester Way
Anne	Dodsworth	Sales Representative	7 Houndstooth Rd.

სურათი 5

პროგრამული უზრუნველყოფის კოდი

პროგრამული უზრუნველყოფა შექმნილია დაპროგრამების ენა C#-ზე და მისი კოდის სრული ვერსია მოყვანილია ქვემოთ.

```
public partial class QOMEngine
    {

        public string GenerateSqlScript(Query query)
        {
            string selFmt = "SELECT {0} FROM {1}  {2}  WHERE {3} {4}";
            StringBuilder selList = new StringBuilder();

            StringBuilder whereClause = new StringBuilder();
            StringBuilder modifierStr = new StringBuilder();
            StringBuilder subQueries = new StringBuilder();
            StringBuilder whereFlt = new StringBuilder();

            string schema = query.Schema;

            string entityName = schema + "." +  query.EntityName;

            whereClause.Append(FormFilterScript(schema,
query.LogicalGroups, null));

            if (query.SubQueries != null)
            {
                foreach (var sq in query.SubQueries)
                {
                    subQueries.Append(GetSetOperation((new
SetOperationRequestModel { ID = (int)sq.SetOperation })).Name + " ");
                    subQueries.Append(schema + "." + sq.EntityName + "
ON ");
                    subQueries.Append(FormFilterScript(schema,
sq.LogicalGroups, null));
```

```
                }
            }

            if (query.SelectFields != null)
            {
                int i = 0;
                string coma = " ";
                foreach (var sf in query.SelectFields)
                {
                    string str = schema + "." + sf.FieldFullName;
                    selList.Append(str);
                    i += 1;
                    if (sf.ModificatorId != 0)
                    {
                        modifierStr.Append(coma + GetQueryModifier(new
QueryModifierRequestModel { ID = sf.ModificatorId }).Name + " " +
str);
                        coma = ",";
                    }
                    if      (i      <      query.SelectFields.Count())
selList.Append(",");
                }
            }

            string res = string.Format(selFmt, selList, entityName,
subQueries, whereClause, modifierStr);
            return res;
        }

        private    StringBuilder    FormFilterScript(string    schema,
List<LogicalGroup> subLgs, LogicalGroup lg)
        {
            StringBuilder flt = new StringBuilder("( ");
```

```
            string logoper = (lg == null) ? " " :
lg.LogicalOperation.ToString();

            string op = " ";
            if (lg == null)
            {
                if (subLgs != null && subLgs.Count > 0)
                {
                    foreach (var lg1 in subLgs.Where(e => e.LogLevel
== 1))
                    {
                        flt.Append(" " + op);
                        flt.Append(FormFilterScript(schema, subLgs,
lg1));
                        op = " " + lg1.LogicalOperation.ToString() + "
";
                    }
                }
            }
            else
            {
                flt.Append(LogOperFilterOperation(schema, lg.Clauses,
op, logoper));

                if (lg.SubLogicalGroups != null &&
lg.SubLogicalGroups.Count > 0)
                {
                    if (!flt.ToString().Trim().Equals("(")) op =
logoper;

                    foreach (var lg1 in lg.SubLogicalGroups)
                    {
                        flt.Append(" " + op + " ");
                        flt.Append(FormFilterScript(schema,subLgs,
lg1));
```

```
                    op = " " + logoper + " ";
                }
            }
        }

        return flt.Append(" )");
    }

    private StringBuilder LogOperFilterOperation(string schema,
IList<Clause> clauseList, string op, string logoper)
    {
        StringBuilder flt = new StringBuilder("");
        if (clauseList != null && clauseList.Count > 0)
        {
            foreach (var cl in clauseList)
            {
                flt.Append(op);
                string f1 = schema + "." + cl.FullFieldName;
                var pred = GetPredicat(new PredicatRequestModel {
ID = cl.PredicatId });

                if (cl.ValueText != null &&
!cl.ValueText.Equals(""))
                {
                    string constVal =
DataValueTransformation.FieldValueForSqlScript(cl.ValueType,
cl.ValueText);

                    if (pred.TypeID == 1)
                    {
                        flt.Append(f1 + " " + pred.Name + " " +
constVal);
                    }
                    else
                    {
```

```
                            StringBuilder tt = new
StringBuilder(pred.FuncTemplate);
                            tt.Replace("c1", f1);
                            string ss = cl.ValueText;
                            if (ss[0] == '\'') ss = ss.Substring(1,
Math.Max(0, ss.Length - 2));

                            tt.Replace("c2", ss);
                            flt.Append(tt);
                        }
                    }
                    else
                    {
                        if (pred.TypeID == 1)
                        {
                            flt.Append(f1 + " " + pred.Name + " " +
(string.IsNullOrWhiteSpace(cl.JoinedFullFieldName) ? "" : (schema +
"." + cl.JoinedFullFieldName)));
                        }
                        else
                        {
                            StringBuilder tt = new
StringBuilder(pred.FuncTemplate);
                            tt.Replace("c1", f1);
                            tt.Replace("c2", "+" + schema + "." +
cl.JoinedFullFieldName + "+");
                            flt.Append(tt);
                        }
                    }
                    op = " " + logoper + " ";
                }
            }
            return flt;
        }
    }
```

```
public class Query
    {
        public string EntityName { get; set; }
        public string Schema { get; set; }

        public List<SelectField> SelectFields { get; set; }

        public List<LogicalGroup> LogicalGroups { get; set; }
        public List<SubQuery> SubQueries { get; set; }
    }

public class SelectField
    {
       public string FieldFullName { get; set; }
       public int ModificatorId { get; set; }
    }

public class SubQuery
    {
        public SetOperationType SetOperation { get; set; }
        public string EntityName { get; set; }
        public List<LogicalGroup> LogicalGroups { get; set; }
    }

public class Clause
    {
        //public int LogicalGroupeId { get; set; }
        public string FullFieldName { get; set; }
        public string ValueText { get; set; }
        public FieldType ValueType { get; set; }
        public string JoinedFullFieldName { get; set; }
        public int PredicatId { get; set; }
    }
```

```
public class LogicalGroup
   {
      public LogicalOperationType LogicalOperation { get; set; }
      public decimal? LogLevel { get; set; }
      public List<Clause> Clauses { get; set; }
      public List<LogicalGroup> SubLogicalGroups { get; set; }
   }
```

სამომავლო გეგმები და პერსპექტივები

მნიშვნელოვანია, რომ პროექტს დაემატოს უსაფრთხოების მექანიზმი, რომლის დახმარებითაც თავიდან ავიცილებთ Sql-Injection-ის მსგავს პრობლემებს. ასევე მნიშვნელოვანია პროექტს ქონდეს ზოგადი მომხმარებლის ინტერფეისი, რომლის დახმარებითაც ანალიტიკოსთა და სხვა სპეციალობათა წარმომადგენლები შეძლებენ მართონ თავიანთ მონაცემთა ბაზა სპეციალისტების დახმარების გარეშე. ასევე მნიშვნელოვანი ამოცანაა სამომავლოდ შესაძლებელი იყოს გაუქმდეს ამოსაღები წყაროს განსაზღვრის საშუალება, რადგანაც მომხმარებლებიდან გარკვეულ ცოდნას მოითხოვს ეს პროცესი და უმჯობესია წამოსაღები ველების და დასაფილტრი ინფორმაციის მიხედვით პროგრამამ ავტომატურად შექმნას ოპტიმალური გადასვლის წერტილებით ამოსაღები წყაროს სიმრავლე. სავარაუდოდ, ეს პრაქტიკულად საინტერესო ამოცანა იქნება და მნიშვნელოვნად წინ გადადგმული ნაბიჯი პროექტის განვითარების კუთხით. ასევე ფილტრის გენერაციის დროს სასურველია, არ გავუშვათ ისეთი ფილტრი, რომელშიც თავისთავად ჯდება სხვა ფილტრის ინფორმაცია, ანუ ამოვყაროთ უაზრო და ზედმეტი ფილტრები, რომელიც ლოგიკურად არაა საჭირო. სამომავლოდ შესაძლებლობა უნდა მოგვეცეს, რომ ბიბლიოთეკამ იმუშაოს სხვდასხვა ტიპის მონაცემთა ბაზებთან.

დასკვნა

ამგვარად, წინამდებარე ნაშრომში განხილული იყო ავტორის მიერ შემუშავებული ალგორითმი (შესაბამისი პროგრამული უზრუნველყოფა), რომელიც დინამიურად აფორმირებს მონაცემთა ბაზაში ინფორმაციის საძიებელ მოთხოვნას მიწოდებული ინფორმაციის საფუძველზე მონაცემთა ბაზის სტრუქტურის შესახებ. მოთხოვნის ფორმირებისას არსებით როლს ასრულებს ფილტრის აგება სხვადასხვა ველების პრედიკატების და ლოგიკური ოპერაციების გამოყენებით ასევე გათვალისწინებულია ცხრილებს შორის კავშირის დინამიურად აწყობაც. ალგორითმი რეალიზებულია net პლატფორმის გამოყენებით შექმნილი ბიბლიოთეკის სახით და შესაძლებელი მისი პრაქტიკული გამოყენება სხვადასხვა მონაცემთა ბაზებზე ორიენტირებულ აპლიკაციებში. ავტორის მიერ ასევე შემუშავებულია აპლიკაცია რომელიც იყენებს მოცემულ ალგორითმს და ეს აპლიკაცია თან ერთვის ნაშრომს.

გამოყენებული ლიტერატურა

1. Roberts, S. A., and Gahegan, M. N. Supporting the notion of context within a database environment for intelligent reporting and query optimization. European Journal of Information Systems, I, 1 (1991), 13-22.
2. Sheth, A. P., and Larson, J. A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys, 22, 3 (September 19 90), 183-236.
3. Chen, A. N. K., Goes, P. B. and Marsden, J. R. A Query-Driven Approach to the Design and Management of Flexible Database Systems. Journal of Management Information Systems, Vol. 19, No. 3 (Winter, 2002/2003), 121-154.
4. Krishnan R., Li X., Steier D. and Zhao J. L. On Heterogeneous Database Retrieval: A Cognitively Guided Approach. Information Systems Research. Vol. 12, No. 3 (September 2001), pp. 286-303
5. Panckhurst R. A Database for Linguists: Intelligent Querying and Increase of Data. Computers and the Humanities. Vol. 28, No. 1 (1994), pp. 39-52.
6. Gorman K. and Choobineh J. The Object-Oriented Entity-Relationship Model (OOERM). Journal of Management Information Systems. Vol. 7, No. 3, Management Support Systems (Winter, 1990/1991), pp. 41-65.
7. Clifford J. Formal Semantics and Pragmatics for Natural Language Querying. Cambridge Tracts in Theoretical Computer Science (No. 8). Cambridge University Press, 1990.
8. Ruckhaus E., Ruiz E. and Vidal M. E. Query Evaluation and Optimization in the Semantic Web. Theory and Practice of Logic Programming. Vol. 8, Issue 03 (May 2008), pp. 393-409.
9. Clark D., Giacobazzi R., Mu C. Foreword: Programming Language Interference and Dependence. Mathematical Structures in Computer Science. Vol. 21, Special Issue 06 (December 2011), pp. 1109-1110