საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე

დაპროგრამების ვიზუალური მეთოდები და ინსტრუმენტები

(UML, Borland C++ Builder)

თბილისი 2007

ს ა რ ჩ ე ვ ი

შესავალი

მართვის საინფორმაციო სისტემების დაპროექტება და შემდგომ დაპროგრამება თანამედროვე ვიზუალური კომპიუტერული ტექნოლოგიების საშუალებით მეტად მნიშვნელოვანი და აქტუალურია, რამეთუ საგრძნობლად უმჯობესდება პროექტის რეალიზაციის ხარისხი და მცირდება მისი დამუშავების დრო და ხარჯები.

განსაკუთრებით საყურადღებოა დღეს ფირმა მაიკროსოფთის მიერ შემოთავაზებული დაპროგრამების ახალი პლატფორმა დოტ-NET ტექნოლოგიის სახით, რომელიც Windows- და Web-დანართების ასაგებადაა გამიზნული თავისი ახალი ვიზუალურ-ობიექტ-ორიენტირებული დაპროგრამების ინსტრუმენტებით: VB.NET, C#.NET, C++.NET, ADO.NET, ASP.NET, XML, MS VISIO და ა.შ. [1,2,3].

მეორეს მხრივ, მნიშვნელოვანია პროგრამული ინჟინერიის (Software Engineering) ისეთი ინსტრუმენტის ათვისება, როგორიცაა უნიფიცირებული მოდელირების ენა (UML - Unified Modeling Language), ვინაიდან იგი ითვლება კომპიუტერული პროგრამული პაკეტების შექმნის მეთოდოლოგიურ საფუძვლად.

ესაა დაპროგრამების ობიექტ-ორიენტირებულ მეთოდზე შექმნილი თანამედროვე ინფორმაციული ტექნოლოგია, რომელიც არის მოდულების სპეციფიკაციის, კონსტრუირების, ვიზუალიზებისა და დოკუმენტირების ენა და აღნიშვნათა სისტემა.

დღეს ამ სტანდარტს იყენებს Microsoft, Oracle, Hewlet-Packard და სხვა ცნობილი ფირმები.

დაპროგრამების თანამედროვე ინსტრუმენტები ინტეგრირებული პაკეტებია, რომლებიც აერთიანებს მონაცემთა აღწერისა და მანიპულირების ენებს (მონაცემთა ბაზის სახით), პროცედურების დამუშავების ხერხებს კლასთა თეორიის გამოყენებით და სტანდარტულ ბიბლიოთეკებს.

ამგვარად, მათში რეალიზებულია ობიექტ-ორიენტირებული დაპროგრამების მეთოდი და სტილი: ინკაფსულაციის, კლასთა მემკვიდრეობითობის, პოლიმორფიზმისა და აბსტრაქციის სახით.

დინამიკური პროცესების მოდელირებისა და ანალიზისათვის აქტუალურია პეტრის ქსელების ინსტრუმენტის შესწავლა. მისი დახმარებით აიგება სისტემის მდგომარეობათა დიაგრამა (State Diagrams UML-ში), კომპიუტერული ქსელების პროცესების მართვის მოდელი კონფლიქტური სიტუაციების აღმოფხვრის მიზნით და ა.შ.

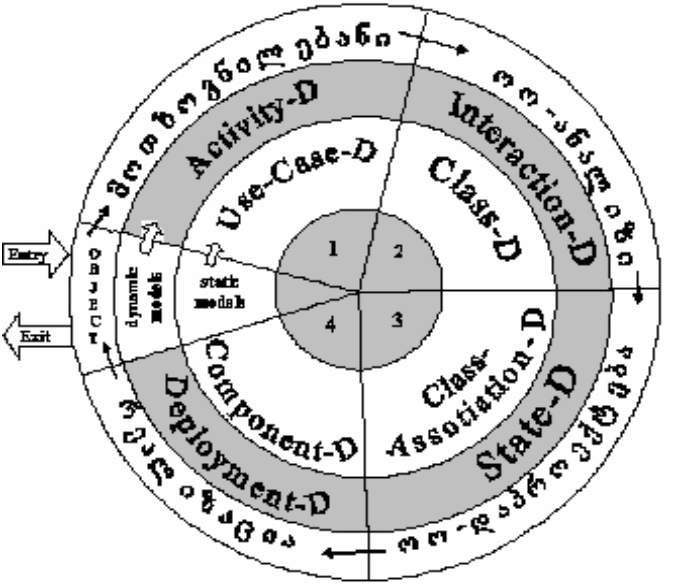
წინამდებარე ნაშრომში შემოთავაზებულია აღნიშნული საკითხების თეორიულ-პრაქტიკული ასპექტები. კონკრეტული საპრობლემო სფეროს მაგალითზე განიხილება მართვის საინფორმაციო სისტემის ობიექტ-ორიენტირებული ანალიზის, მოდელირების, დაპროექტებისა და პროგრამული რეალიზაციის ამოცანები.

I თავი

უნიფიცირებული მოდელირების ენა (UML)

1.1. პროგრამული პაკეტების აგების ეტაპები

პროგრამული პაკეტების შექმნა UML-ტექნოლოგიის მიხედვით ოთხ ეტაპად ხორციელდება (საკვლევი ობიექტის ავტომატიზაციის მოთხოვნილებების დადგენა, მისი ობიექტ-ორიენტირებული (ოო) ანალიზი, ოო-დაპროექტება (დეტალური დონე) და რეალიზაცია (პროგრამული კოდი). 1-ელ ნახაზზე მოცემულია ეს ეტაპები, სადაც ოო-მოდელირება სტატიკური და დინამიკური დიაგრამებით (D) ხორციელდება.

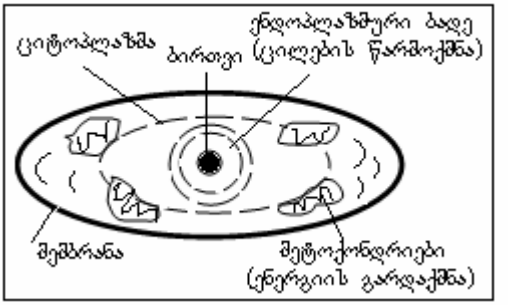


ნახ. 1

1.2. ობიექტ-ორიენტირებული მიდგომა: კლასები და ობიექტები

ობიექტი (Object) განიხილება, როგორც გარკვეული არსი (Entity), რომელიც ხასიათდება მდგომარეობით (მონაცემთა ერთობლიობა) და ქცევით (ფუნქციური პროგრამები). ობიექტის ქცევა ანუ რეაქცია, რომლის დროსაც მისი ახალი მდგომარეობა განისაზღვრება, დამოკიდებულია გარედან მოსულ ინფორმაციაზე, შეტყობინებებზე. ვინაიდან ობიექტი უმთავრესი ცნებაა, იგი საწყისია ობიექტ-ორიენტირებული დაპროგრამებისა, ამიტომ დიდი მნიშვნელობა აქვს მის სწორად გაგებას.

ამისათვის განვიხილოთ ჯერ **ბიოლოგიური უჯრედის** აგებულება (ნახ.1.1).

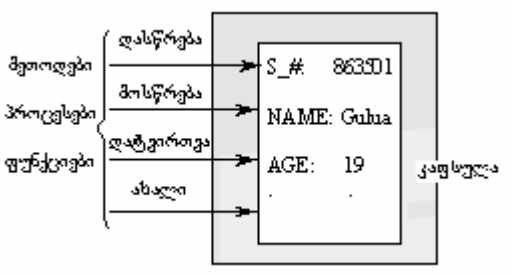


ნახ.1.1

იგი კაფსულირებულია მემბრანის (გარსის) საშუალებით, რომლითაც გამოიყოფა სხვა უჯრედებისა და გარემოსაგან. მას უნარი აქვს გარემოდან მიიღოს ქიმიური სახის ინფორმაცია და გადასცეს უჯრედს შიგნით. ბირთვი უჯრედის ძირითადი ინფორმაციის მატარებელია. ის შედგება ქრომოსომებისაგან, რომლებიც გარკვეული გენეტიკის მქონეა. უჯრედის დაყოფის (გამრავლების) დროს ხდება მემკვიდრული თვისებების გადაცემა. ბირთვის გარშემო დაჯგუფებულია სხვადასხვა ფუნქციის

ელემენტები: მაგ., ენდოპლაზმური ბადე - ცილების წარმოქმნის ფუნქცია, მიტოქონდრები - ენერგიის გარდაქმნის ფუნქცია, ციტოპლაზმა (თხევადი მოძრავი გარემო) - ტრანსპორტირების ფუნქცია და ა.შ.

როგორია ობიექტის ,,ინფორმაციული უჯრედის'' აგებულება. 1.2 ნახაზზე განხილულია ობიექტი-სტუდენტი, რომელიც რეალური სამყაროს ნაწილია.



ნახ.1.2

იგი კაფსულირებულია, რომლის შიგნითაც მოთავსებულია ბირთვი ობიექტის თვისებების აღმწერ მონაცემთა ელემენტები S#(სტუდენტის ნომერი), NAME (გვარი), AGE(ასაკი) და სხვ.

კაფსულაში შეღწევა და მონაცემების დამუშავება გარედან ყველა ფუნქციას არ შეუძლია, არსებობს წინასწარ განსაზღვრული მეთოდები, პროცესები ან ფუნქციები, რომელთაც ზოგადად სერვისული პროგრამები შეიძლება ვუწოდოთ. მათ აქვთ უნარი შეაღწიოს კაფსულაში და დაამუშაოს მონაცემები. ე.ი. ინფორმაციული ობიექტი კაფსულირებული მონაცემების და მათი დასამუშავებელი მეთოდებისაგან შედგება.

მონაცემები განსაზღვრავს ობიექტის მდგომარეობას, ხოლო მეთოდები – ობიექტის ქცევას, მის რეაქციას გარედან მოსულ შეტყობინებაზე.

სტუდენტის მონაცემების დამუშავება შეუძლია მხოლოდ სამ ფუნქციას, როგორებიცაა დასწრება, მოსწრება, დატვირთვა. თუ მოვიდა შეტყობინება სწორედ ამ ინფორმაციის მისაღებად, მაშინ ობიექტი (კაფსულა) ცნობს მათ. სხვა შეტყობინებებისათვის მონაცემები დამალულია. თუ საჭიროა ახალი ფუნქციის დამატება, ის წინასწარ უნდა მოთავსდეს „კაფსულაში“.

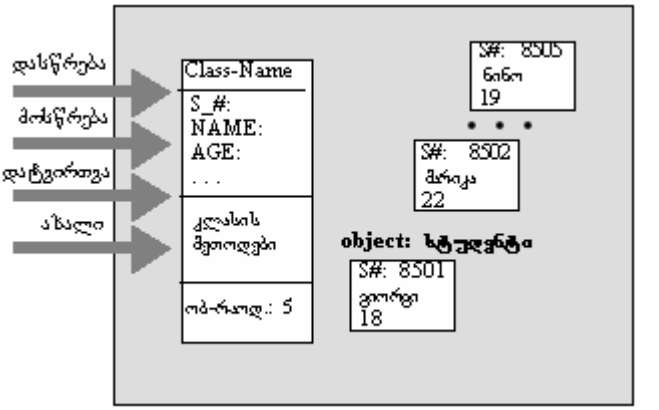
ობიექტები და კლასები ობიექტ-ორიენტირებული დაპროგრამების ძირითადი კომპონენტებია. ობიექტის ცნებას, როგორც ინფორმაციული უჯრედისა, წინა პარაგრაფში გავეცანით. ახლა დავაკონკრეტოთ მისი განსაზღვრება და ავხსნათ ამ ცნების მიმართება კლასის ცნებასთან.

ობიექტი რეალური ან წარმოსახვითი სამყაროს ნაწილი, ინდივიდუალური ეგზემპლარია. ის არსია (Entity) და გააჩნია უნიკალური იდენტიფიკატორი, რომლითაც სხვა არსებისაგან განსხვავდება. ობიქტს აქვს ინდივიდუალური თვისებები, რომლებიც გამოიხატება მონაცემებით (ატრიბუტები, ცვლადები) და ქცევით (მეთოდები, ფუნქციები) გარემოში. ობიექტებს შორის კომუნიკაციები ხორციელდება შეტყობინებების გადაცემით. მიღებული შეტყობინების საფუძველზე ობიექტში აქტიურდება შესაბამისი მეთოდი, რომელიც მის ქცევას განსაზღვრავს და შეუძლია გადაიყვანოს იგი სხვა მდგომარეობაში (იცვლება ატრიბუტების და ცვლადების მნიშვნელობები). ობიექტის მდგომარეობა ხასიათდება სტატიკური კომპონენტით (ატრიბუტები) და დინამიკური კომპონენტით (ატრიბუტთა მნიშვნელობები).

ობიექტის ქცევა მიუთითებს იმაზე, თუ როგორ იცვლება მისი მდგომარეობები და სხვა ობიექტებთან ურთიერთმიმართებანი.

ობიექტის ცნება დაპროგრამების ენაში პირველად გამოყენებულ იქნა Simula-ში, ხოლო მისი ზემოაღნიშნული კლასიკური განმარტება მოგვცა ამერიკელმა მეცნიერმა გრადი ბუჩმა (G. Booch).

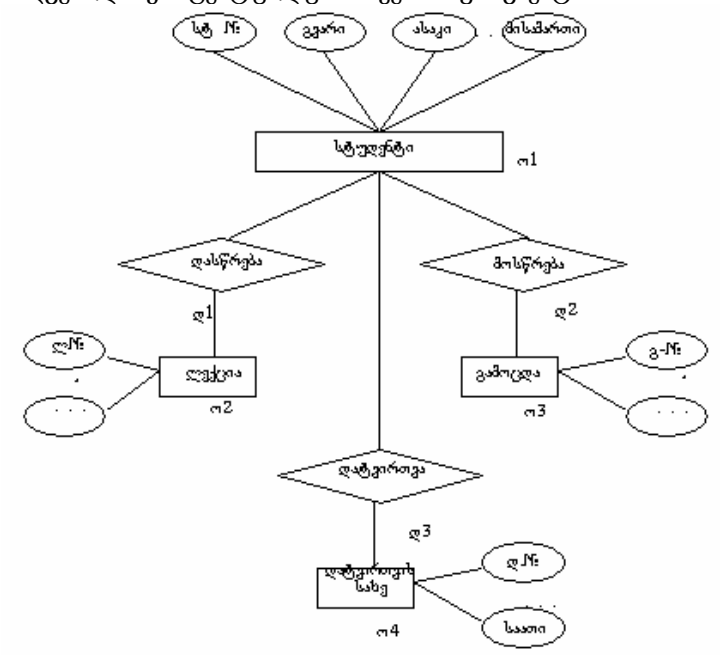
კლასი არის ერთი ტიპის ობიექტების სიმრავლე, რომელთაც აქვთ მსგავსი სტრუქტურა და ქცევა. 1.3 ნახაზზე ნაჩვენებია კლასის მაგალითი.



ნახ.1.3

ვინც იცნობს მონაცემთა რელაციური ბაზების თეორიას და არსთა - დამოკიდებულებათა (Entity - Relationship) მოდელის აგების საკითხებს, ადვილად შეამჩნევს გარკვეულ ანალოგიას საპრობლემო სფეროს კონცეპტუალური მოდელის, ლოგიკური სტრუქტურისა და მისი ფიზიკური ორგანიზაციის რეალიზაციასთან. მსგავსება შემდეგი თვალსაზრისით შეიძლება

იქნეს განხილული: კლასი სტუდენტი ეთანადება 1.4 ნახაზზე წარმოდგენილი კონცეპტუალური სქემის ფრაგმენტს.



ნახ.1.4

ამ კონცეპტუალური სქემის გადატანით მონაცემთა ბაზების ლოგიკურ სტრუქტურაში მივიღებთ სქემას, რომელიც ახლოა კლასის მოდელთან. ლოგიკური სტრუქტურა ატრიბუტებისგან შემდგარი სქემაა, რომელიც ობიექტების სტატიკურ კომპონენტებად მოვიხსენიეთ ზემოთ. კლასის ობიექტები კი ეთანადება ამ ლოგიკური სტრუქტურის ქვეშ მდგარ ფიზიკურ ჩანაწერებს (ჩანაწერის უნიკალური ნომრითა და ველების მნიშვნელობებით).

მონაცემთა რელაციური ბაზების თეორიაში გამოიყენება მონაცემებისა და პროგრამების ერთმანეთისაგან იზოლირების

პრინციპი, რათა განხორციელდეს მათ შორის სრული დამოუკიდებლობა. მონაცემთა აბსტრაქტული სტრუქტურების გამოყენებით ეს საკითხი დადებითად იქნა გადაჭრილი.

ობიექტ-ორიენტირებული დაპროგრამების ენა ფლობს კლასების, ობიექტების, მათი მნიშვნელობების დამუშავების მეთოდების რეალიზაციის საშუალებებს. როგორც აღვნიშნეთ, ძირითადი პრინციპი მონაცემების კაფსულირებაშია. კლასი კი თავისი ბუნებით მონაცემთა ახალი ტიპია, რომელიც იქმნება თვით მომხმარებლის მიერ. როგორ უნდა გვესმოდეს ეს საკითხი?

დაპროგრამების ენებში არის მონაცემთა სტანდარტული ტიპები (მაგალითად, int *a*), რომელიც აცხადებს *a* ცვლადს მთელი ტიპით. ეს *a* პროგრამისათვის ობიექტია. თუ ენას აქვს მონაცემთა ახალი ტიპის შექმნის შესაძლებლობა, ეს მის სიმძლავრეზე მიუთითებს, მაგალითად, C++ ენის ფრაგმენტის მოშველიებით გამოვაცხადოთ Magistrant როგორც ახალი კლასი:

```
class Magistrant {
  private:
        char Name [20];
        int Age;
        char Specification [30];
  public:
        Input-Name (NAME);
        Input-Age (AGE);
        Input-Spec (SPEC); };
     void main(void) {   // და მისი ობიექტებიც
           Magistrant  M1, M2,  M3.....;
           . . . }
```

ამგვარად, **Magistrant** ისეთივე ტიპია, როგორც **int, char** და ა.შ. ყოველი ობიექტი M1,M2,M3..... არის კონკრეტული მაგისტრანტი, რომელიც სტრუქტურას იღებს **class Magistrant**

(....)-იდან **private** და **public** ოპერატორები განსაზღვრავს მონაცემებსა (**Name, Age, Specification**) და ფუნქციებზე (**Input-Name, Input-Age, Input-Spec**) მიმართვის შესაძლებლობას. პირველი ლოკალურია და მალავს ამ მონაცემებს სხვა ობიექტებისათვის. მათთან მიმართვა შეუძლია მხოლოდ მოცემული ობიექტის ფუნქციებს და ზოგჯერ მათ „მეგობრებსაც“-friend). Public იძლევა ნებართვას, რათა მის შემდეგ მდგომი ფუნქციები გამოყენებულ იქნას ობიექტის გარედან. განხილული მაგალითის განზოგადებით შეიძლება დავასკვნათ, რომ კლასების საფუძველს მონაცემთა აბსტრაქტული ტიპები წარმოადგენს.

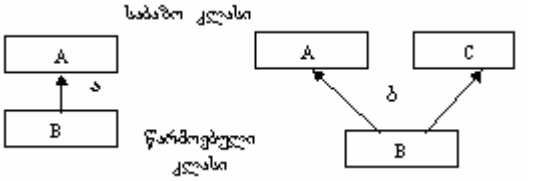
ასხვავებენ პროცედურულ და მონაცემთა აბსტრაქციებს. პირველი ცალ-ცალკე განიხილავს პროცედურის მიზანს, მის შინაგან რეალიზაციას და მონაცემთა აბსტრაქციას. ეს უკანასკნელი ნიშნავს, რომ საჭიროა მხოლოდ იმის ცოდნა, თუ რა ოპერაციებს ასრულებს მოცემული პროგრამული მოდული. არაა აუცილებელი ვიცოდეთ, თუ რომელ მონაცემებს ამუშავებს (ისინი დამალულია) და როგორ სრულდება ეს ოპერაციები.

1.3. კლასთა იერარქია, მემკვიდრეობითობა და პოლიმორფიზმი

ობიექტ-ორიენტირებული დაპროგრამების სტილის საბაზო პრინციპებია ინკაფსულაცია, მემკვიდრეობითობა და პოლიმორფიზმი. პირველი მათგანი წინა პარაგრაფში განვიხილეთ და კლასის ცნებამდე მივედით. კლასებს შორისაც არსებობს გარკვეული დამოკიდებულებანი. რას ნიშნავს ეს ?

თუ არსებობს გარკვეული კლასების ბიბლიოთეკები, რომლებიც შექმნილია ამ მომენტამდე, მაშინ სასურველია მოხდეს მათი გამოყენება ახალი ამოცანების გადასაწყვეტად. ახალი კლასები უნდა განისაზღვროს არსებულის ბაზაზე, მოხდეს მათი გაფართოება და მოდიფიკაცია. ყოველივე ეს მნიშვნელოვნად ამცირებს ახალი სისტემების დაპროექტებისა და რეალიზაციის ვადებს. სწორედ ამაშია ობიექტ-ორიენტირებული დაპროგრამების მეთოდის ეფექტურობის საიდუმლოება.

ორი კლასიდან ერთი ბაზისური, ხოლო მეორე წარმოებულია. 1.5 ნახაზზე ნაჩვენებია მარტივ-მემკვიდრეობითი (**single inheritance**) და მრავალ-მემკვიდრეობითი (**multiple inheritance**) იერარქიული კავშირები.



ნახ.1.5

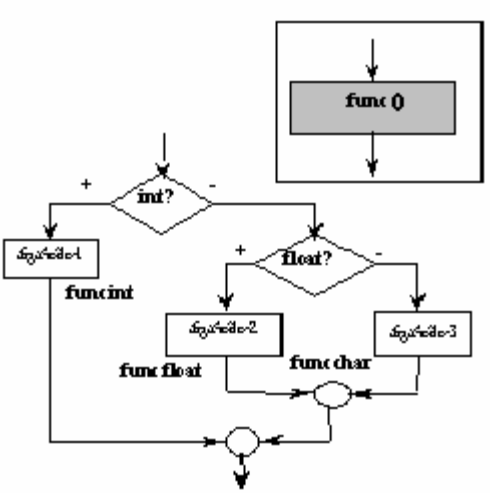
საბაზო კლასი შეიძლება იყოს აბსტრაქტული, რომელსაც თვითონ არ გააჩნია კონკრეტული ეგზემპლარი, მაგრამ გამოიყენება სხვა წარმოებული კლასების მისაღებად.

იერარქიული კავშირების აღწერა შეიძლება გრაფის საშუალებით ორიენტირებული ხის სახით, ციკლების გარეშე. გრაფის წვეროებს კლასები შეესაბამება. ფესვური წვერო ის კლასია, რომელიც აღწერს ყველაზე ზოგად თვისებებს, ისინი კი გადაეცემა ქვედა იერარქიის წარმოებულ კლასებს. ამგვარად შეიძლება დავასკვნათ, რომ ფუნქციური შესაძლებლობების თვალსაზრისით წარმოებული კლასები, უფრო მძლავრია, ვიდრე საბაზო კლასები. ეს იმიტომ, რომ წარმოებულ კლასს შეუძლია თავისი ფუნქციების შესრულებაც და საბაზო კლასისაც, საბაზოს კი – მხოლოდ თავისი. წარმოებულ კლასს შეუძლია გამოიყენოს საბაზო მონაცემებიც (თუ ისინი არაა დამალული სპეციალური ატრიბუტებით, მაგალითად, private) და ა.შ.

პოლიმორფიზმი. ობიექტ-ორიენტირებულ დაპროგრამებაში ინკაფსულაციისა და მემკვიდრეობითობის გვერდით მნიშვნელოვანი ადგილი უკავია პოლიმორფიზმის ცნებას. ის ბერძნული სიტყვაა polymorphism და „მრავალფორმიანობას“ ნიშნავს. ესაა ობიექტის თვისება, რომელიც უზრუნველყოფს ერთსა და იმავე ფუნქციების გამოყენებას სხვადასხვა ამოცანათა გადასაწყვეტად. ამოცანები შეიძლება მოითხოვდეს ფუნქციებისა და მათი არგუმენტების სხვადასხვა ტიპებს, რასაც პოლიმორფიზმი ადვილად წყვეტს ე.წ. ვირტუალური ფუნქციებით (**virtual function**) ან ფუნქციათა გადატვირთვით (**overloaded function**).

მონომორფულ სისტემებში ყოველი ფუნქცია და მისი არგუმენტები მხოლოდ ერთი ტიპითაა შეზღუდული. მაგალითად,

ჩვეულებისამებრ C ენაში საჭიროა დაიწეროს ორი სხვადასხვა ფუნქცია **int - func (int, int)** და **float - func (float, float)**, თუკი გვინდა ორ მთელ რიცხვზე ან ორ ნამდვილ რიცხვზე არითმეტიკული ოპერაციების ჩატარება (ნახ.1.6).

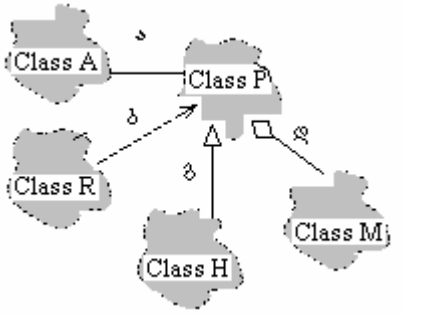


ნახ.1.6

პოლიმორფიზმის იდეა C++ ენაში საშუალებას გვაძლევს დავწეროთ მხოლოდ ერთი func (a,b) ფუნქცია, შემდეგ კი არგუმენტების ტიპების ანალიზის საფუძველზე ენის კომპილატორი თვითონ აირჩევს, თუ რომელი ოპერაციები შეასრულოს.

1.4. კლასთაშორისი კავშირები

კავშირის ასოციაციები კლასებს ან ობიექტებს შორის მრავალი სახისაა, მაგ., 1:1, 1:M, M:N. ესაა ტიპური მათემატიკური ასახვის ფუნქციები, დამოკიდებულებანი კლასებსა და მის ელემენტებს შორის. შეიძლება განვიხილოთ აგრეთვე ისეთი დამოკიდებულებები, რომლებიც სტრუქტურულ მოწესრიგებას ემსახურება, მაგალითად, კლასიფიკაცია და აგრეგაცია. პირველი მათგანი აერთიანებს ობიექტებს გარკვეული კრიტერიუმებით, რომლებიც მსგავს, მაგრამ არაიდენტურ თვისებებს ეყრდნობა. მეორე კი მთელისა და შემადგენელი ნაწილის მიმართების ტიპური ასახვაა. 1.7 ნახაზზე ნაჩვენებია სქემატურად კლასებს შორის ასოციაციური (ა), რელაციური (ბ), მემკვიდრეობითი (გ) და აგრეგირებული (დ) კავშირების გამოსახვა.



ნახ.1.7

- ასოციაციური (Assotiation) ნიშნავს სემენტიკურ კავშირს კლასებს შორის. ის შეიძლება გამოისახოს

ერთ- ან ორმიმართულებიანი (იგივეა, რაც უისრო) ხაზით. ისარი გვიჩვენებს შეტყობინების გადაცემის მიმართულებას. ასოციაციური კავშირის რეალიზება ხდება ერთ კლასში

დამატებით მეორე კლასის ატრიბურტის ჩასმით. ეს ჰგავს პირველადი (Primary) და მეორადი გასაღებური ატრიბუტების შეერთებას.

- რელაციური (Dependency) ნიშნავს ერთი კლასის დამოკიდებულებას მეორეზე. იგი ერთმიმართულებიანი წყვეტილი ისრით გამოიხატება. მასში დამატებითი დამაკავშირებელი ატრიბუტები არ გამოიყენება.

- მემკვიდრეობითი (Generalization) ასახავს „გენეტიკურ“, განზოგადოებულ კავშირებს კლასებს შორის. ასეთ დროს ერთი კლასი („შვილი“) მთლიანად იღებს მეორე კლასის („მშობელი“) ყველა ატრიბუტს, მეთოდს და კავშირებს.

აგრეგირებული (Aggregation) ნიშნავს კავშირს მთელი-ნაწილი. მაგალითად, ავტომობილი - ძარა, ძრავი, საბურავები და ა.შ.

დასასრულს, შევაჯამოთ ძირითადი მოსაზრებანი:

– ობიექტ-ორიენტირებული დაპროგრამების ტექნოლოგიის გამოყენება ეფექტურია დიდი და რთული დასაპროექტებელი საპრობლემო გარემოსათვის;

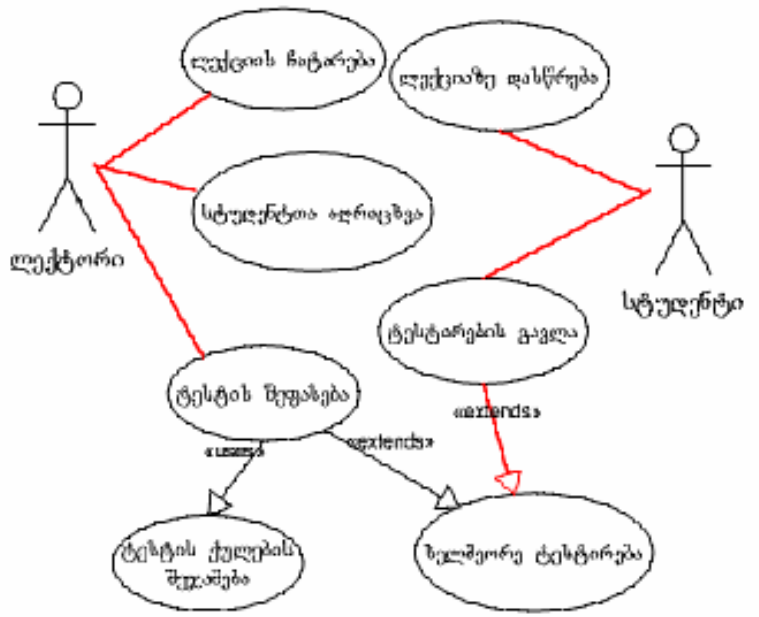
– ობიექტ-ორიენტირებული მოდელირების ძირითად ელემენტს ობიექტი შეადგენს, რომელშიც კაფსულირებულია მისი თვისებრივი მონაცემები და დამუშავების ფუნქციები (მეთოდები);

– ობიექტების აბსტრაქციით, სტრუქტურული მოწესრიგებითა და მათ შორის გარკვეული მემკვიდრეობითი კავშირების ასახვით, იქმნება კლასები, სუბკლასები და მეტაკლასები;

– ობიექტ-ორიენტირებული დაპროგრამების დისციპლინის ძირითადი კვლევის საგანია ობიექტთა მდგომარეობისა და ქცევის მოდელირების პროცესები.

1.5. UML - ეტაპების დიაგრამები

UseCase-D დიაგრამა უჩვენებს როლებს (შემსრულებლებს-Actor), მათ ფუნქციებს (Action) და კავშირებს (ნახ.1.8):



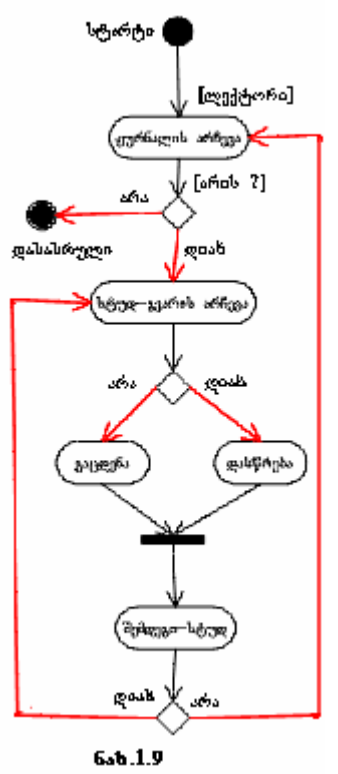
ნახ.1.8. UseCase დიაგრამა: შემსრულებლებია (Actors) და პროცედურებია (Actions)

ესაა დასაპროექტებელი (გამოსაკვლევი) ორგანიზაციული ობიექტის საწყისი (პირველადი) მოდელი, რომელშიც ჩანს, თუ რომელ საპრობლემო სფეროსთან გვაქვს საქმე.

კავშირი communication გამოიყენება როლებსა და ფუნქციებს შორის, ხოლო <<use>> და <<extends>> ფუნქციებს შორის. პირველი ასახავს შემთხვევას, როცა ერთი პროცესი **აუცილებლად** მოითხოვს ჯერ სხვა ქვეპროცესის შესრულებას,

მეორ შემთხვევაში ეს არაა აუცილებელი. კომუნიკაციური კავშირი არ შეიძლება გავავლოთ როლებს შორის.

ყოველ UseCase-ფუნქციას (ოვალს) შეესაბამება ერთი დინამიკური მოდელი, რომელიც **Activity-D** დიაგრამის სახით ფორმირდება (ნახ.1.9).



ნახ.1.9

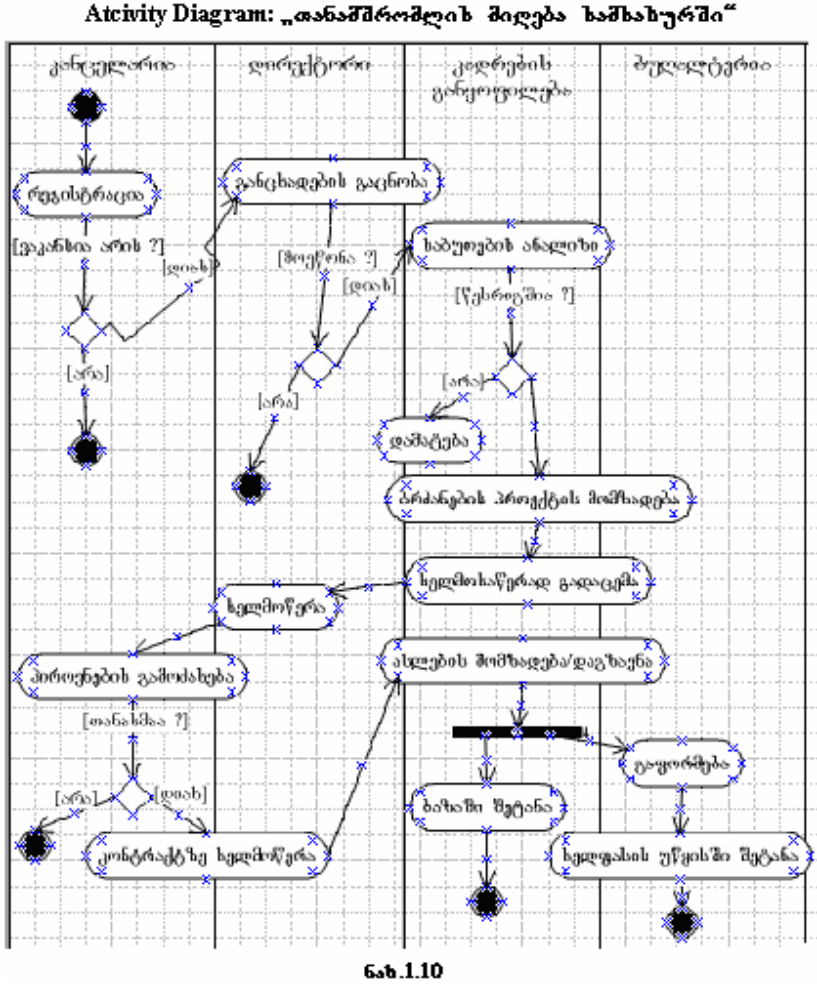
აქტიურობათა დიაგრამას ერთი დასაწყისი და რამდენიმე დასასრული შეიძლება ჰქონდეს. მასში მონაწილეობს რამდენიმე როლის შემსრულებელი (მაგ., ლექტორი, სტუდენტი, დეკანი და ა.შ.).

მრგვალკუთხედებში მოთავსებულია მათ მიერ შესასრულებელ პროცედურათა დასახელებები. პროცედურები შეიძლება შესრულდეს მიმდევრობით ან პარალელურად. გამოიყენება სპეციალური განშტოების, შეერთების და სხვა აღნიშვნები. პროცედურები უკავშირდება ერთმანეთს ისრებით, რომლებიც მოქმედებათა მიმდევრობას ასახავს.

აქტიურობის დიაგრამა შეიძლება შედგებოდეს რამდენიმე პარალელური ზოლისგან (swimline), რომლებშიც თავსდება სხვადასხვა მართვის სფეროს (დეპარტამენტის) როლის შესასრულებელი პროცედურები.

ამგვარად, აქტიურობის დიაგრამა ასახავს კონკრეტული ამოცანის გადაწყვეტის ინფორმაციულ-ტექნოლოგიურ სურათს.

მაგალითად, „ახალი თანამშრომლის მიღება“ (ნახ.1.10), „თანამშრომლის გადაყვანა სხვა განყოფილებაში“, „კონტრაქტის გაფორმება“ და სხვ.

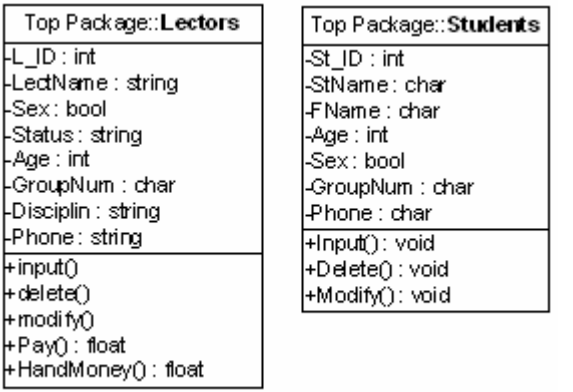


ნახ.1.10

ნახაზზე ჩანს კოლექტიური, დროში განაწილებული შესასრულებელი პროცედურების ლოგიკურ მიმდევრობათა აქტიურობის დიაგრამის ფრაგმენტი.

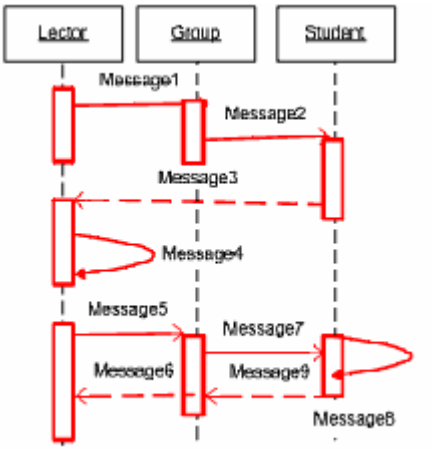
კლასებისა (Class-D) და ინტარაქტიურობათა (Sequence-D, Collaboration-D) დიაგრამები ემსახურება ობიექტ-ორიენტირებული ანალიზის ეტაპს. კლასები საკვლევი ობიექტის სტატიკური მოდელია და მასში აღიწერება კონკრეტული დასახელება, მონაცემები (ატრიბუტები) და ფუნქციები (მეთოდები). ინტარაქტიურობათა დიაგრამები ასახავს ობიექტის დინამიკურ მოდელს, ანუ როგორ ხდება ობიექტის კლასების დამუშავება დროში და სივრცეში. ასეთი პროცესები ცნობილია სცენარების სახელწოდებით: კონკრეტული როლი რა თანამიმდევრობით, რომელი მეთოდებით ამუშავებს რომელი კლასის ობიექტებს.

ამგვარად, კლასების ასაგებად საჭიროა ოო-მოდელირების საფუძველზე განისაზღვროს კლასთა დასახელებები (ნახ.1.11), მათი ატრიბუტები (მონაცემები) და ფუნქციები (მეთოდები).



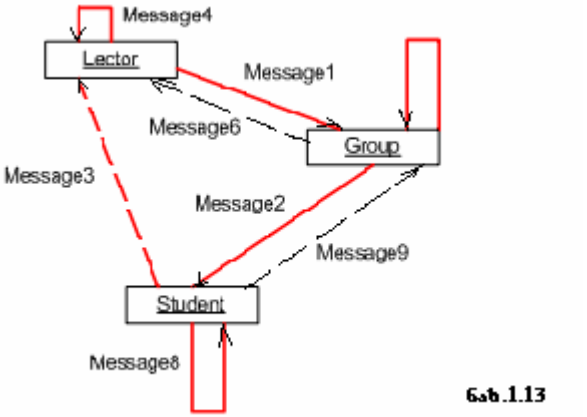
ნახ.1.11. კლასების დიაგრამა

მიმდევრობითობის დიაგრამაზე (ნახ.1.12) შეტყობინებები და ოპერაციები დალაგებულია მათი შესრულების მიმდევრობით, აქ მთავარი დროა.



ნახ.1.12

თანამოქმედების დიაგრამაზე (ნახ.1.13) კარგად ჩანს კლასებს შორის ინფორმაციის გაცვლა შეტყობინებებისა და მეთოდების გამოყენების საფუძველზე.

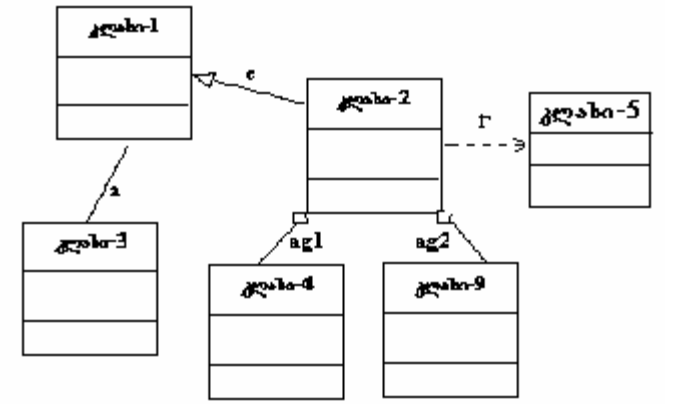


ნახ.1.13

ამ ეტაპზე სასარგებლოა არსთა დამოკიდებულების მოდელის (Entity-Relationship-Model) გამოყენება (ნახ.1.4).

Class-D კლასების დიაგრამა შემდგომში, **ობიექტ-ორიენტირებული დაპროექტების ეტაპზე**, გამოიყენება კლასებისა და მათ შორის კავშირების (Class-Assotiation-D) აღსაწერად.

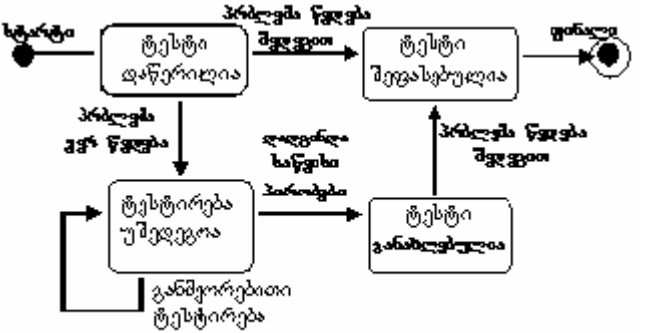
როგორც აღვნიშნეთ (ნახ.1.7), კავშირები კლასებს შორის ოთხი ტიპისაა: ასოციაციური-a, რელაციური-r, აგრეგატული-ag და მემკვიდრეობითი-c. მათი გრაფიკული აღნიშვნები მოცემულია 1.14 ნახაზზე.



ნახ.1.14. კლასთა-კავშირების დიაგრამა

ყოფაქცევის დიაგრამებიდან ჩვენ უკვე განვიხილეთ Activity-D და Interaction-D.

არსებობს აგრეთვე კლასების მდგომარეობათა დიაგრამა ანუ State-D (ნახ.1.15). იგი აღწერს მოქმედებებს, ობიექტთა მდგომარეობებს, მდგომარეობათა გადასვლებს და მოვლენებს. მისი გამოყენება ყველა კლასისთვის არაა საჭირო. აუცილებელია მაშინ, როდესაც კლასი შეიძლება იმყოფებოდეს რამდენიმე მდგომარეობაში და თითოეულ მათგანში იგი იქცევა სხვადასხვანაირად.



ნახ.1.15. მდგომარეობათა დიაგრამა

რეალიზაციის ეტაპზე აიგება კომპონენტების დიაგრამა (Component-D), რომელშიც იგულისხმება პროგრამული კოდების (CPP, H, DLL და ა.შ.) დამუშავება (ნახ.1.16).

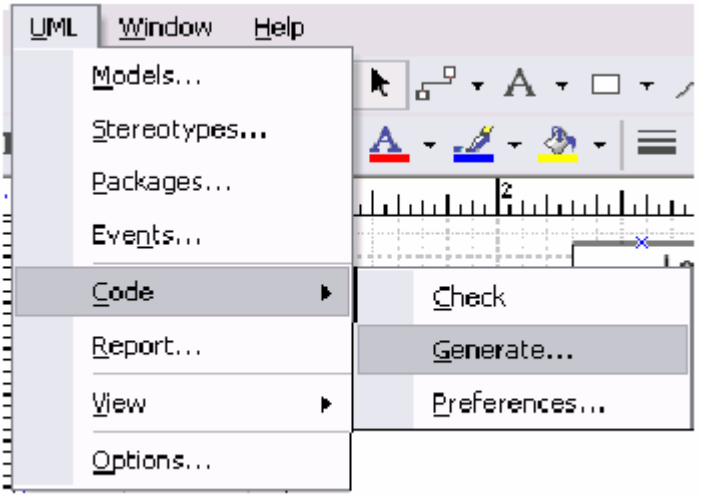
ნახ.1.16. კომპონენტების დიაგრამა

პროგრამული კოდის ავტომატური გენერაცია შესაძლებელია კლასთა კავშირების დიაგრამის აგების შემდეგ (ნახ.17-ა). მენიუდან ავირჩევთ UML | Code | Generate, რის შემდეგაც გამოჩნდება ახალი კადრი (ნახ.1.17-ბ).

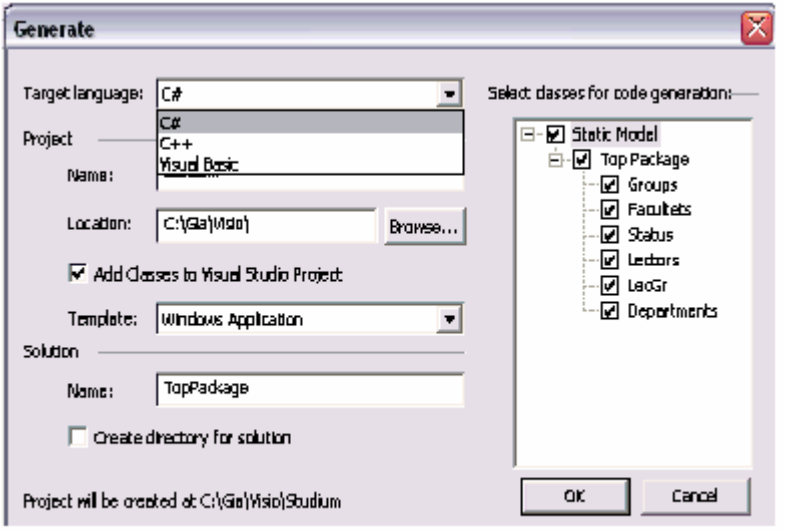
სისტემა შემოგვთავაზებს დაპროგრამების ენის არჩევას (C# , C++, Visual Basic, J++).

აქვე უნდა ავირჩიოთ პროექტის სახელი (Name), Browse-ს გამოყენებით პროგრამული მოდულების ჩასაწერი კატალოგი (Location), მაგ., : D:\Gia\Visio-1\ და მარჯვენა ნაწილში ამოვირჩიოთ კლასები, რომელთა დაპროგრამებასაც ვაპირებთ.

ნახაზზე ყველა კლასია მონიშნული. დავამოწმოთ შედეგი ღილაკით Ok.

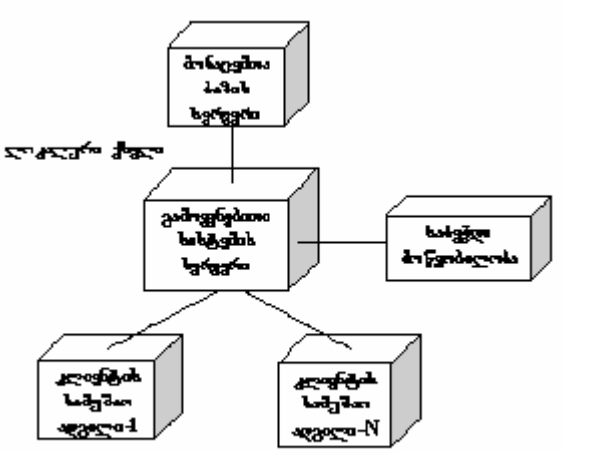


ნახ.17-ა



ნახ.17-ბ

პროექტის ბოლოს აიგება განთავსების დიაგრამა (Deployment-D), რომელიც აღწერს კომპონენტების განაწილებას "კლიენტ-სერვერ" ქსელში (ნახ.1.18).



ნახ.1.18. განთავსების დიაგრამა

ამით დავასრულებთ უნიფიცირებული მოდელირების ენის ძირითადი კომპონენტების (დიაგრამების) თეორიულ განხილვას. მომდევნო თავში შევისწავლით UML-ტექნოლოგიის კონკრეტულ ინსტრუმენტს, მაიკროსოფთის ახალ პაკეტს Ms Visio, რომელმაც საფუძვლიანად დაიმკვიდრა ადგილი Ms Office-პაკეტის შემადგენლობაში Word, Excel, PowerPoint და სხვა პროგრამებთან ერთად.

1.6. საკონტროლო კითხვები და სავარჯიშოები

1. რას წარმოადგენს UML ტექნოლოგია და რა ეტაპებისგან შედგება იგი ?

2. რას არის კლასი და ობიექტი ?

3. რას არის ინკაფსულაცია ?

4. რას ნიშნავს კლასთა მემკვიდრეობითობა ?

5. რას ნიშნავს პოლიმორფიზმი ?

6. რას ნიშნავს ცნება **ობიექტორიენტირებული** ?

7. როგორ ხდება ობიექტის მდგომარეობის სტატიკური მოდელის წარმოდგენა ?

8. როგორ ხდება ობიექტის მდგომარეობის დინამიკური მოდელის წარმოდგენა ?

9. რა არის კლასის მეთოდები ?

10. რა არის შეტყობინება ?

11. კლასთაშორის რომელ კავშირებს იცნობთ ?

12. ააგეთ UseCase დიაგრამა „ლექცია“.

13. ააგეთ Activity დიაგრამა „ცოდნის შეფასება“.

14. ააგეთ Class-თა დიაგრამა სფეროსათვის „მარკეტი“.

13. ააგეთ Sequence და Collaboration დიაგრამები „ბიბლიოთეკაში წიგნების გატანა-დაბრუნება“.

14. ააგეთ Class-Assotiation დიაგრამა საპრობლემო სფეროსათვის „ლექცია“.

15. რას წარმოადგენს Component და Deployment დიაგრამები ?

16. დააპოექტეთ UML-დიაგრამები შემდეგი საპრობლემო სფეროებისათვის:

ა) „კათედრა“, „დეკანატი“, „უნივერსიტეტი“;

ბ) „მინი-მარკეტი“, „სუპერ-მარკეტი“, „მაღაზიათა ქსელი“.

გ) „ანაბრები“, „კრედიტები“, „ბანკი“.

დ) „ხელფასები“, „პენსიები“, „ბუხჰალტერია“.

ე) „პოლიკლინიკა“, „რეგისტრატურა“, „ანალიზების ლაბორატორია“.

ვ) „მიმღები“, „ნოზოლოგიური განყოფილებები“, „საოპერაციო“, „საავადმყოფო“.

ზ) „ფეხბურთელთა საკლუბო გუნდი“, „ჩემპიონატი“ (მაგ., ევროლიგა).

თ) „მუზეუმი“, „თეატრი“, „ტურისტული ბიურო“.

I თავის ლიტერატურა

1. ვ. დეიტელი., კ. დეიტელი. დაპროგრამება C და C++ ენებზე. რუს.ენაზე, მოსკოვი, 2002.

2. გ. სურგულაძე. შესავალი პრაქტიკული დაპროგრამების C ენაში. ნაწ.-1, სტუ, 2003.

3. გ. სურგულაძე, ლ. ყვავაძე. მონაცემთა სტრუქტურების და ფაილების დამუშავება C ენაზე. ნაწ.-2, სტუ, 2004.

4. ჩოგოვაძე გ., გოგიჩაიშვილი გ., სურგულაძე გ., შეროზია თ., შონია ო. მართვის ავტომატიზებული სისტემების დაპროექტება და აგება (თეორიულ-პრაქტიკული ინფორმატიკა). თბ., სტუ, 2001.

II თავი
დაპროგრამების ვიზუალურ-კომპონენტებიანი ინსტრუმენტი Borland C++ Builder

2.1. სისტემის აგებულება

არის: აპლიკაციების სწრაფი დამუშავების (RAD - Rapid Application Develepment) ვიზუალურ კომპონენტებიანი ობიექტ-ორიენტირებული ენა.

აქვს: აგების ორმიმართულებიანი (რევერსული) ტექნოლოგია (Two-Wey Tools), რაც გამოიხატება ვიზუალური დაპროექტების ინსტრუმენტისა და პროგრამული კოდის რედაქტორის სინქრონიზებულ ურთიერთქმედებაში.

შედგება (იხ. ნახ.2.0):

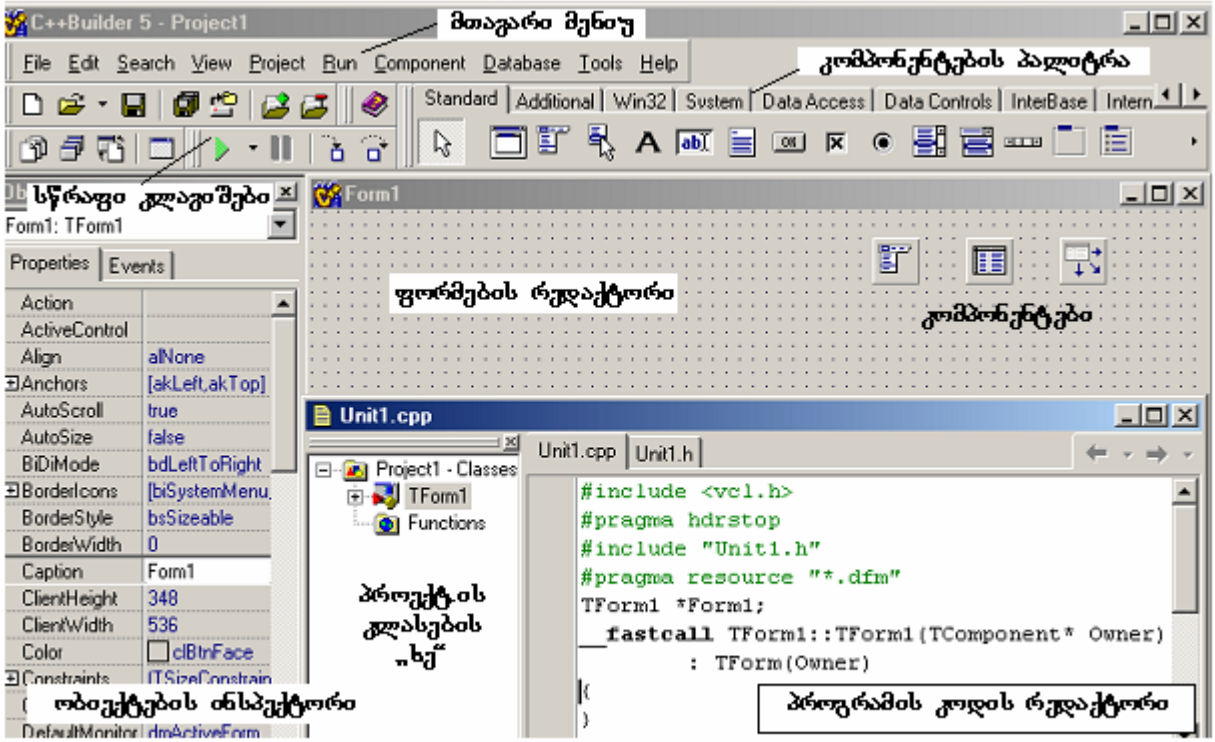
1. სამუშაო გარემოს მთავარი მენიუს;
2. სწრაფი კლავიშების პანელის;
3. კომპონენტების პანელის;
4. ობიექტების ინსპექტორის;
5. ფორმების რედაქტორისა და
6. კოდის რედაქტორისაგან.

გამოიყენებს: "გადათრევის" მეთოდს (drag and drop), რაც მდგომარეობს კომპონენტების პალიტრიდან არჩეული კომპონენტის ფორმაზე გადატანაში.

იყენებს: მონაცემთა ლოკალური (dBase, Paradox) და განაწიებული ბაზების (InterBase) მართვის სისტემების ინსტრუმენტებს. შეუძლია იმუშაოს Ms SQL Server და სხვა ბაზებთან.

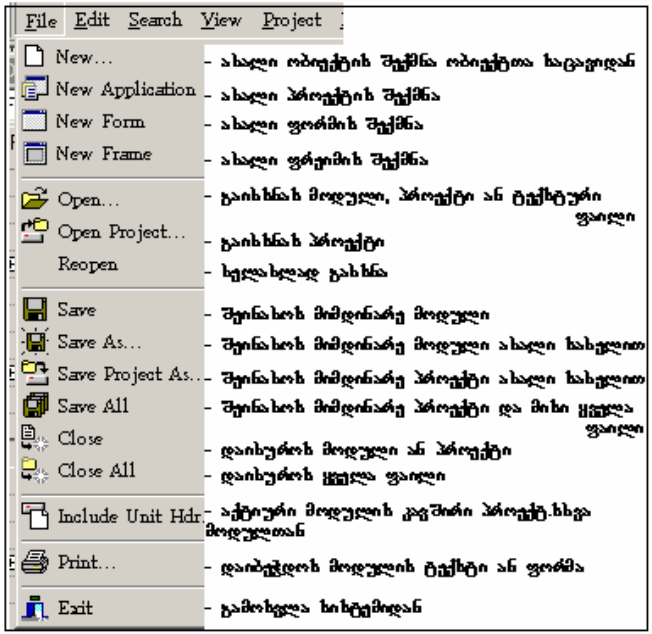
გამოიყენება თანამედროვე ინფორმაციულ ტექნოლოგიებში კლიენტ-სერვერ სისტემების ასაგებად.

ფლობს 200-ზე მეტ ვიზუალურ კომპონენტს და ასეთი კომპონენტების შექმნის ინსტრუმენტულ საშუალებებს.

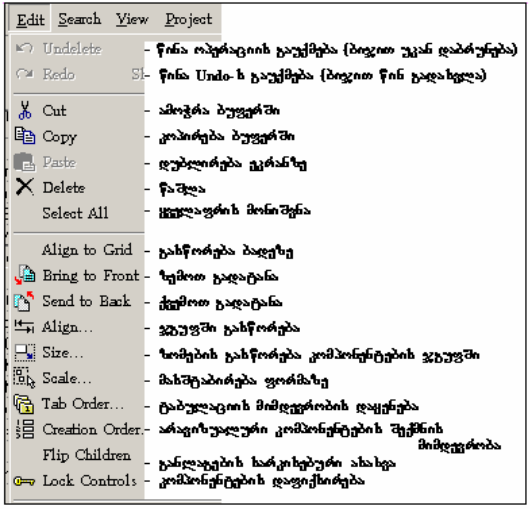


ნახ.2.0. ხამუშაო გარემო BC++Builder

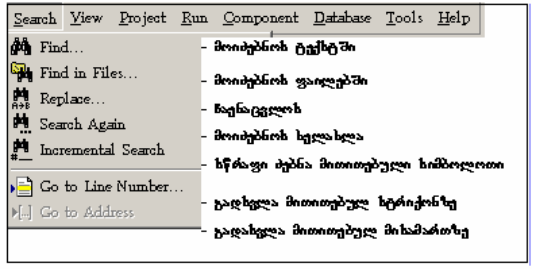
2.2. სისტემის მთავარი მენიუ და ქვემენიუები



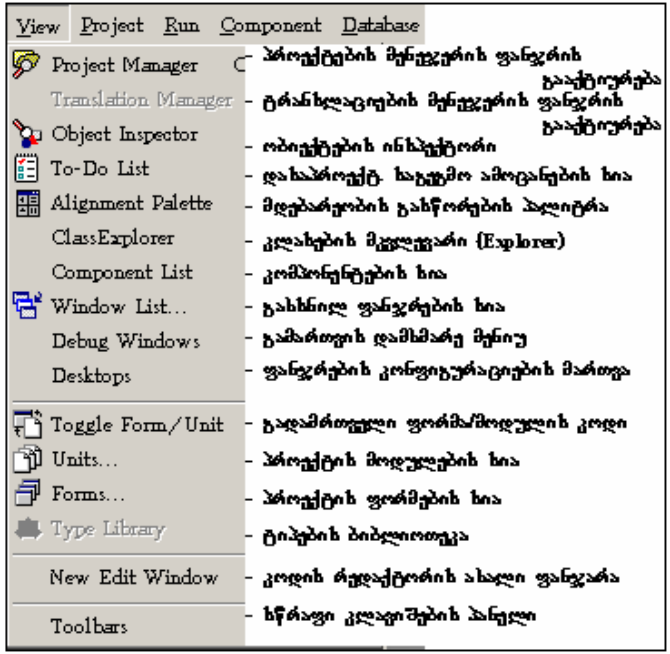
ნახ.2-1. მენიუ - File



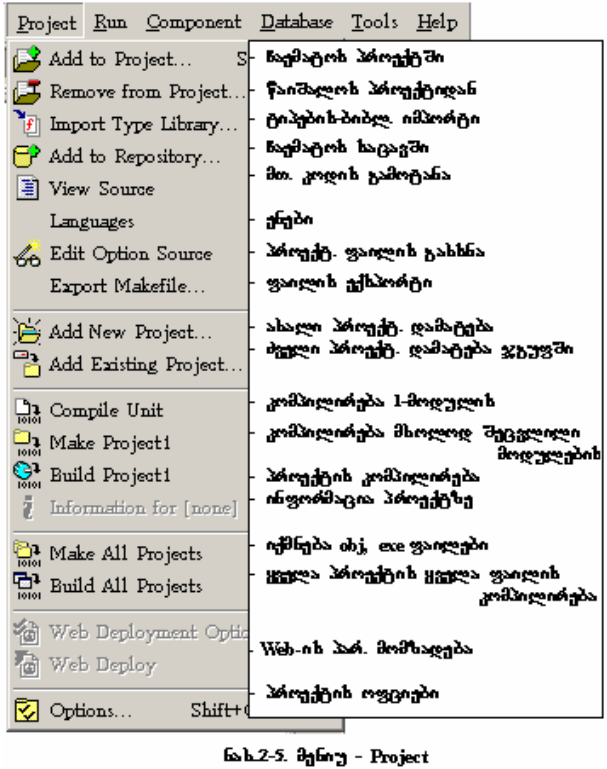
ნახ.2-2. მენიუ - Edit



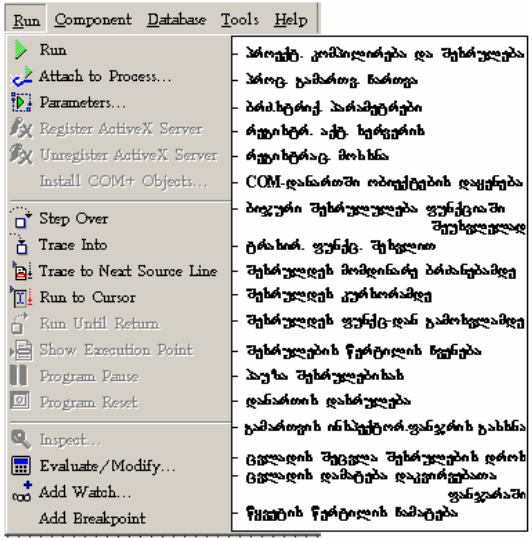
ნახ.2-3. მენიუ - Search



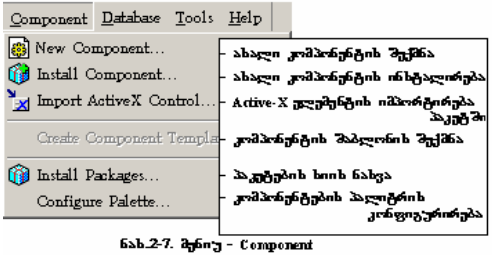
ნახ.2-4. მენიუ - View



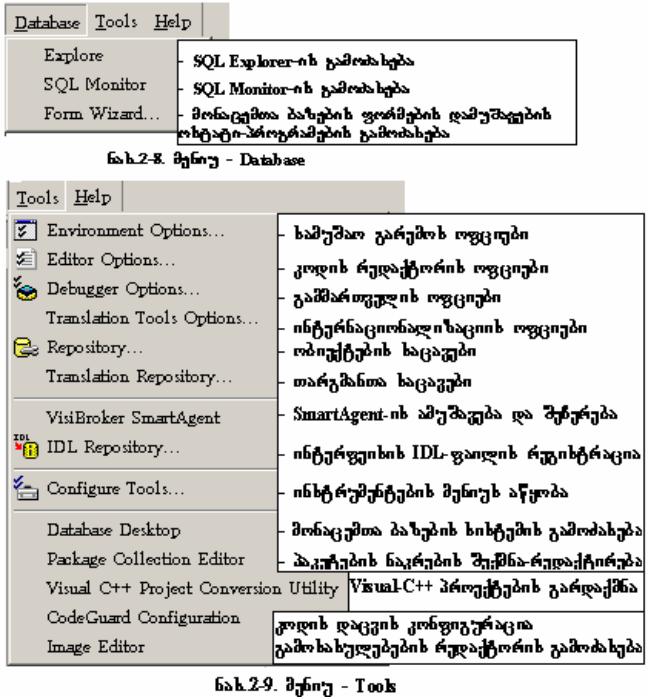
ნახ.2-5. მენიუ - Project



ნახ.2-6. მენიუ - Run



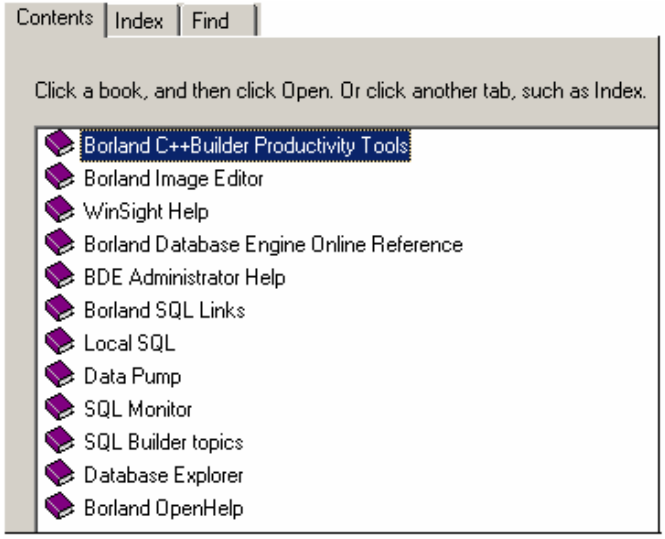
ნახ.2-7. მენიუ - Component



ნახ.2-8. მენიუ - Database

ნახ.2-9. მენიუ - Tools

ნახ.2-10. მენიუ - Help



ნახ.2.10.ა. კონტექსტური ძებნის ფრაგმენტი Help-ფაილში

2.3. სისტემის ვიზუალური კომპონენტები

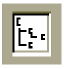
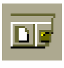
2.3.1. Standard - სტანდარტული კომპონენტები

1		TFrame	კადრია,რომელიც რაღაცნაირ ფორმა-კონტეინერია გამოიყენება სხვა კომპონენტებისათვის
2		TmainMenu	ჭრის ბრძანებათა პანელის ფორმის მთავარი მენიუსათვის
3		TpopUpMenu	ჭრის კონტექსტური მენიუს ფორმისათვის
4		TLabel	დასახელების ტექსტის ახახა ფორმაზე
5		TeEdit	რედაქტირებადი სტრიქონის შეტანის ველი ფორმაზე
6		TMemo	რედაქტირებადი მრავალსტრიქონიანი შეტანის ველი ფორმაზე
7		TButton	ჭრის ღილაკს წარწერით
8		TCheckBox	ჭრის მარკერის ველშეტანის ორი მდგომარეობით
9		TRadioButton	რადიო-ღილაკი ორი მდგომარეობით
10		TListBox	ახახის ტექსტური სტრიქონების hook არის
11		TComboBOx	ჭრის ტექსტური სტრიქონების დამამაგრ hook და რედაქტირების არის

12		TScrollBar	ქმნის ფანჯრის ფორმის ან ხის გადასათვალიერებლად (ვერტიკალური ისრებით)
13		TGroupBox	ქმნის ფორმაზე ლოგიკურად დაკავშირებული კომპონენტების კონტეინერს
14		TRodio Group	ქმნის ლოგიკურად ურთიერთგამომრიცხავ რადიო ღილაკს
15		TPanel	ქმნის ინსტრუმენტების პანელს ან მდგომარეობის სტრიქონს
16		TActionList	მოქმედებათა სია(არავიზუალური ელემენტებისა და პროგრამული ლოგიკის ურთიერთმოქმედების მართვისათვის)

2.3.2. windows32 -ის კომპონენტები

17		Tab Control	ასახავს ველების ერთობლიობას ჩანართების ურთიერთგადაფარვის პრინციპით
18		PageControl	მრავალგვერდიანი დიალოგის ორგანიზება ურთიერთგადაფარვის პრინციპით
19		ImageList	გრაფიკული გამოსახულებათა კოლექციის შექმნა(ერთი ზომის)
20		RichEdit	ფორმატირების "მდიდარი" საშუალებები (Rich text format)
21		rack Bar	სასრიალის მაგვარი სკალის შექმნა
22		Progress Bar	ქმნის მართკუთხა პროგრესს- ინდიკატორს. პროგრამის შესრულების მდგომარეობის საილუსტრაციოდ

23		T Up Down	ქმნის წყვილ ღილაკს ისრებით (ზევით და ქვევით) პარამეტრის შესაცვლელად
24		T Hotkey	სწრაფი კლავიშების გამოყენება (მაგ. Ctrl ,Alt, Shift)
25		TAnimate	ქმნის უხმოდ კომპონენტს ანიმაციური ვიდეო ჩანაწერის შესასრულებლად
26		T DateTime Picker	ქმნის თარიღისა და დროის რედაქტირების არეს
27		T MonthCalendar	ქმნის თვის კალენდარს
28		T TreeView	ასახავს ელემენტების იერარქიულ სტრუქტურას(მაგ. დოკუმენტის სათაურები. ჩანაწერები სარჩევში ,ფაილები და კატალოგები დისკზე
29		T List View	ასახავს სიის ელემენტების სხვადასხვა ცხრილურ ფორმაში
30		T HeaderControl	ქმნის სვეტების სათაურების კონტროლების ცვლადი სიგანის ველებით
31		T Statusbar	ქმნის სტატუს-ინდიკატორის გამოსატან პანელს ფორმის ქვედა ნაწილში
32		T Tool Bar	ქმნის კომპონენტს ინსტრუმენტების ღილაკებით
33		T Cool Bar	ქმნის პანელს მოძრავი ფირფიტებით(Bands)
34		T PageScroller	ქმნის ცვლადი სიგრძის კომპონენტს(პანელს) გადასახვევი ისრებით

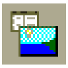
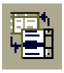
2.3.3. Additional - დამატებითი კომპონენტები

35		TBitBth	ქმნის გრაფიკულ ღილაკს ბიტური გამოსახულებით
36		TSpeedBotton	ქმნის გრაფიკულ ღილაკს ფუნქციის სწრაფი გამოძახების ან რეჟიმის დასაყენებლად
37		TMaskEdit	ქმნის მარაკუთხა არეს მომცემია ფორმატირებული შეტანისთვის
38		TString Grid	ქმნის ბადეს სამუშაოდ გამოსახულებისთვის x და y ღერძებით
39		TDrawGrid	ქმნის ბადეს გრაფიკული მომცემის ორგანიზებულებით მასივისთვის
40		TImage	ქმნის კომპონენტს გრაფიკული გამოსახულებისთვის
41		TShape	ხატავს მარტივ გეომეტრიულ ფიგურებს
42		TBevel	ქმნის მოცულობით ხაზებს ჩარჩოებს
43		TScrollBox	ქმნის ცვლად სფეროს კომპონენტს გადასვლა სამუშაო
44		TCheeKListBox	ასახავს ჩამონათვალი გადასვლა სამუშაო
45		TSplitter	ფორმის დაყოფა სფეროებად
46		TStatic Text	ქმნის სტატიკურ ტექსტის შესატანად არეს
47		TConrolBar	მართვის პანელის ტიპიური კომპონენტია
48		TApplication Events	ამოიცნობს ობიექტს Applicatin-ის მოვლენებს (13მოვლენა)
49		TC Hart	სქემების გრაფიკების და დიაგრამების აგება

2.3.4. DataAccess - მონაცემთა ბაზების კომპონენტები

50		TDataSource	აკავშირებს ცხრილებს,მოთხოვნებს, პროცედურებს დამოკიდებულია მართვის სხვა კომპონენტებს
51		TTable	მიმართავს მონაცემთა ბაზების ცხრილებთან
52		TQuery	SQL -ენაზე მოთხოვნის ფორმირება ბაზებისათვის
53		TStoredProc	სერვერზე შენახული პროცედურების შესრულება
54		TDatabase	კლიენტ/სერვერ სისტემებში მართვის შესაძლებლობა
55		TSession	რამდენიმე ბაზის ჯგუფური შეერთების გლობარული მართვის საშუალებები
56		TBatchMove	პაკეტური ოპერაციები წანაწერთა ჯგუფზე ან მთლიან ცხრილებზე
57		TUpdateSQL	SQL - მოთხოვნის საფუძველზე მონაცემთა ბაზების მონაცემთა განახლება

2.3.5. Data Controls-მონაცემთა მართვის კომპონენტები

58		TDBGrid	ბადისებური ცხრილის (ან მონაცემთა ბაზის მოთხოვნის) ჩანაწერების ასახვა და რედაქტირება
59		TDBNavigatin	ცხრილში (ან ბაზაში მოთხოვნების) ჩანაწერების მიხედვით გადაადგილება შემდგომი რედაქტირებისათვის ან დასათვალიერებლად
60		TDBText	ცხრილში (ან ბაზაში მოთხოვნის) მიმდინარე ჩანაწერის დასახელების სტრიქონი ველის ტექსტის ასახვა
61		TDBEdit	ემსახურება რედაქტირებად 1-სტრიქონიან ველს ცხრილის ან მონაცემთა ბაზის ჩანაწერში შესატანად
62		TDBMemo	ემსახურება მრავალსტრიქონიანი რედაქტირების არეს ცხრილის ან ბაზის ჩანაწერში შესატანად
63		TDBImage	ემსახურება გრაფიკული ფაილის ველის გამოსახულების შესანახად ჩანაწერში
64		TDBListbox	ემსახურება სიის, რომლიდანაც აირჩევა შესაბამისი ცხრილის (ან მონაცემთა ბაზის მოთხოვნის) მიმდინარე ჩანაწერის მნიშვნელობისათვის
65		TDBCombobox	ემსახურება რედაქტირების კომბინირებულ არეს და ჩამოშ-სტრიქონების კომბინირებულ დასვით სიის, რომლიდანაც აირჩევა საჭირო ჩანაწერი
66		TDBChekbox	ემსახურება ბრძანების ველის ორი მდგომარეობით, რომლებიც დაკავშირებულია ცხრილის (ან მონაცემთა ბაზის მოთხოვნის) ჩანაწერის კონკრეტულ ველთან
67		TDBRadioGroup	ემსახურება გრაფიკულ ღილაკებს ურთიერთგამომრიცხავი რამდენიმე ველის ვარიანტისთვის, რომლებიც დაკავშირებულია ცხრილის(ან ბაზის მოთხოვნის) ჩანაწერის ველებთან
68		TDBLookupList	ემსახურება მნიშვნელობის სიის ველების შესავსებად სხვა ცხრილიდან(ან მონაცემთა ბაზის მოთხოვნიდან)
69		TDBLookup ComboBox	ემსახურება რედაქტირებადი არის და ველის ჩამოშლადი სიის მნიშვნელობის კომბინირებული ველების შესავსებად სხვა ცხრილიდან

2.3.6. Dialogs - დიალოგური კომპონენტები

70		OpenDialog	ფაილების გახსნის დიალოგი
71		Savedialog	ფაილების შენახვის ალოგი
72		OpenPictureDialog	გრაფიკული ფაილების ეკრანზე არჩევის დიალოგი
73		SavePictureDialog	გრაფიკული ფაილების შენახვის დიალოგი Save as რეჟიმში
74		FontDialog	შრიფტების და მასთან პარამეტრების არჩევის დიალოგი
75		ColorDialog	ფერების შერჩევის დიალოგი
76		PrintDialog	ბეჭდვის პარამეტრების შესარჩევი დიალოგი
77		PrinterSEtup Dialog	პრინტერის წინასწარ დასაყენებელი (მისამართების) დიალოგი
78		FindDialog	ტექსტის ძებნის დიალოგი
79		ReplaceDialog	ტექსტის მოძებნისა და მისი ჩანაცვლების (შეცვლის) დიალოგი

2.3.7. System - სისტემური კომპონენტები

80		TTimer	ამოქმედება On Timer-ის მოვლენა განსაზღვრული დროითი ინტერვალის გაქრობის შემდეგ
81		TPaintBox	ფერწერული ხატვის შესაძლებლობა
82		TFileListBox	მიმდინარე კატალოგში ფაილების სია
83		TDirectoryListBox	მიცემული დისკის კატალოგების სტრუქტურის ასახვა
84		TDriveComboBox	ასახავს რადაკვირვებადი დისკების სიას
85		TFilterComboBOX	ასახავს რედაქტირებად ფილტრების სიას ფაილების სახეების ასარჩევად(ტიპებია)
86		TMedia Player	მულტიმედიური მოწყობილობების მართვის სტანდარტული პანელის ასახვა
87		TOleContainer	Ole-ობიექტებთან კავშირის დამყარება
88		TDdeClientConv	მოსაცემია დინამიკური გაცვლის რეჟიმის დაყენება კლიენტისათვის
89		TDDEClientItem	მოსაცემია დინამიკური გაცვლის კლიენტის განსაზღვრა კლიენტისათვის
90		TDDEServerConv	მოსაცემია დინამიკური გაცვლის რეჟიმის დაყენება სერვერისათვის
91		TDDEServerItem	მოსაცემია დინამიკური გაცვლის კლიენტის განსაზღვრა სერვერისათვის

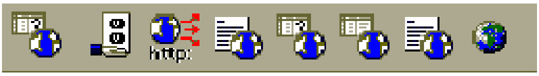
2.3.8. InterBase-ს კომპონენტები

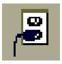
92		IBTable	მონაცემთა ვირტუალური კომპონენტი,რომელიც ანალოგიურია უშუალოდ მონაცემთა ბაზის ცხრილის
93		IBQuery	ახორციელებს InterBase - ის SQL - ოპერატორის მონაცემთა ბაზაში ვიზუალურ ან რამდენიმე ცხრილის მასალებზე
94		IBStoredProc	ანალიტიკური უშუალოდ შენახულ პროცედურის მონაცემთა ბაზის სერვერში
95		IBDatabase	ანალიტიკური უშუალოდ InterBase - ის მონაცემთა ბაზის კავშირს
96		IBTransaction	უზრუნველყოფს დისკრეტული ტრანზაქციის კონტროლის ვიზუალურ ან რამდენიმე მონაცემთა ბაზის კავშირებზე ---
97		IBUpdateSQL	უზრუნველყოფს ობიექტის განსაზღვრებას read-only - ის მქონე მონაცემთა ვირტუალურთანაც
98		IBDataSet	ახორციელებს InterBase - ის SQL - ოპერატორს
99		IBSQL	უზრუნველყოფს ობიექტს InterBase - ის SQL-ოპერატორის შესასრულებლად მინიმალური დანახარჯებით (overhead)
100		IBDatabaseInfo	აბრუნებს ინფორმაციას მიმდინარე მონაცემთა ბაზის შესახებ
101		IBSQLMonitor	აკონტროლებს SQL - დინამიკას InterBase - ის სერვერისათვის
102		IBEvents	უზრუნველყოფს შეიქმნას ასინქრონიზაციის მექანიზმი შეტყობინებაზე (მოვლენაზე) რეაგირებისათვის

2.3.9. Samples ხელმისაწვდომი კომპონენტები

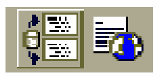
103		Pie	სექტორული დიაგრამის ჩვენა
104		TrayIcon	გამოაქვს პიქტოგრამა ამოცანების პანელზე
105		PerformanceGraph	სისტემის რესურსების (პროცესორი, ოპერატიული მეხსიერება,დისკები) გამოყენების ეფექტურობის გრაფიკი
106		SpinButton	მთელი ღილაკების შექმნა ორი ისრით (მატება, კლება)
107		SpinEdit	ღილაკებით მნიშვნელობების შერჩევა მოცემული დიაპაზონიდან
108		ColorGrid	ფერების პალიტრის შერჩევა
109		CGauge	პროგრესის ინდიკატორი ეკრანზე,რომელიც რამდენიმე ცვლადის ცვლილების მდგომარეობას პროცენტებში გამოსახავს
110		CDirectory	კომპიუტერში დირექტორიების ხე
111		CCalendar	თვის კალენდარის ასახვა
112		IBEventAlerter	**InterBase** - ს მოვლენების შესრულება და შეტყობინებების მიღება სერვერიდან

2.3.10. Internet-ინტერნეტის კომპონენტები



113		ClientSocket	TCP/IP - კლიენტ-პროტოკოლი (Transmission Control Protocol) რომელიც უზრუნველყოფს სოკეტთა ინტერნეტ კავშირს
114		ServerSocket	TCP/IP - სერვერ-პროტოკოლი, რომელიც უზრუნველყოფს კლიენტის მოთხოვნას, რომლის მისგან მოთხოვნის დამუშავებას
115		WebDispatcher	აგებს Web-სერვერის დანართის დისპეტჩერს, რომელიც რეაგირებს კლიენტთა მოთხოვნებზე HTTP-პროტოკოლით
116		PageProducer	HTML - კოდის გენერატორი წინასწარ განსაზღვრული შაბლონით
117		QueryTableProducer	მონაცემების ცხრილური ფორმით გამომტანი გენერატორი, რომელიც მოთხოვნის ალებს Query-ობიექტიდან
118		DataSetTableProducer	მონაცემების ცხრილური ფორმით გამომტანი გენერატორი, რომელიც მოთხოვნის ალებს Data Set-ობიექტიდან
119		DataSetPageProducer	აქტიურ გვერდს, რომელშიც HTML-დოკუმენტის ადგილება ჩანაცვლებს მიმდინარე მონაცემები ბაზის ველებით
120		CppWebBrowser	გახსნის HTML-დოკუმენტს Internet Explorer-ის ფანჯარაში

2.3.11. Internet Express კომპონენტები



უზრუნველყოფს განაწილებული ქსელური გამოყენებითი სფეროს აგებას, რომლის WEB-სერვერიც ასრულებს კლიენტის MIDAS-დანართის როლს.

121		XHLBroker	მოაწყობს XML-პაკეტებს კლიენტის ბრაუზერს და აბრუნებს განახლებულ პაკეტებს
122		MidasPageProducer	გვერდების გენერატორი. კლიენტის ბრაუზერის ფანჯარაში დინამიურად ქმნის HTML-დოკუმენტს რომელშიც მოცემულია ინფორმაცია MIDAS-სერვერული ბაზიდან

2.3.12. Active-X კომპონენტები

123		Chartfx	დიაგრამების შექმნა,კორექტირება
124		VSSpell	მართლწერის შემოწმება
125		F1Book	სამუშაო წიგნი (**WorkBook**),**Excel**-ის მსგავსი
126		VtChart	სამგანზომილებიანი დიაგრამების გამოყენება

2.3.13. ADO - Active Data Objects

წარმოადგენს **Active-X** კომპონენტების სიმრავლეს, რომლებიც გამოიყენება **Microsoft**-ის **OLEDB** ინფორმაციულ ბაზებთან მიმართვისათვის. **ADO**-ს კომპონენტები ალტერნატიულია **Data Access**-ის კომპონენტებისა, რომლებიც **BDE**-ში გამოიყენებოდა

128		ADOConnection	გამოიყენება ADO-მონაცემებთან დასაკავშირებლად. ასრულებს მონაცემთა ურთიერთობის კომპონენტების შესაძლებელი პრობლემების დიაგნოსტირების ფუნქციას
129		ADOCommand	გამოიყენება SQL-პრობლემების შესაკრებლად შედეგების სამრავლის დაბრუნებლად. შეიძლება შეიცავდეს ცვლადებსაც
130		ADODataSet	უნივერსალური მონაცემთა ურთიერთობა. მუშაობს სხვადასხვა რეჟიმებში,რომელთაც შეესაბამება BDE-ს კომპონენტები Table, Query და StoredProc
131		ADOTable	გამოიყენება ერთი ცხრილთან (Table) სამუშაოდ
132		ADOQuery	გამოიყენება მონაცემთა ურთიერთობასთან სამუშაოდ SQL-მოთხოვნის საშუალებით
133		ADOStoredProc	გამოიყენება პროცედურების შესაკრებლად სერვერზე
134		RDS	არის RDS DataSpace-ობიექტი და გამოიყენება მრავალმაგიდიან დამოკიდებულებებში

2.3.14. Decision Cube მრავალმაჩორებელი ანალიზის გადაწყვეტილებათა კუბი

კომპონენტები გამოიყენება მონაცემთა ბაზისა და გადაწყვეტილებათა მიღების სისტემებში მრავალგანზომილებიანი მონაცემების გასაანალიზებლად. ინფორმაციის ასახვა ხორციელდება ჯვარედინი შეკვეცის, პროექტებისა და ჯამების საშუალებით.

135		DecisionCube	უზრუნველყოფს დაკავშირებული ცხრილების მონაცემებთან მუშაობას. ინახავს მრავალგანზომილებიან მონაცემებს
136		DecisionQuery	არის სპეციალიზებული SQL-მოთხოვნაგენერატორი, ამზადებს დაკავშირებული ცხრილების მონაცემებს გადაწყვეტილებათა კუბისთვის
137		DecisionSource	აკავშირებს მართვის გადაწყვეტილებას კონფიგურირებულ მონაცემებსა და გადაწყვეტილებათა კუბს შორის
138		DecisionPivot	მომხმარებელს ახლევს ნებას აკონტროლოს გადაწყვეტილების წყაროს მდგომარეობა
139		DecisionGrid	წარმოადგენს დაკავშირებულ ცხრილებს, მონაცემებს GRID-ცხრილის სახით
140		DecisionGraph	გამოაქვს ეკრანზე დაკავშირებული ცხრილების მონაცემთა გრაფი

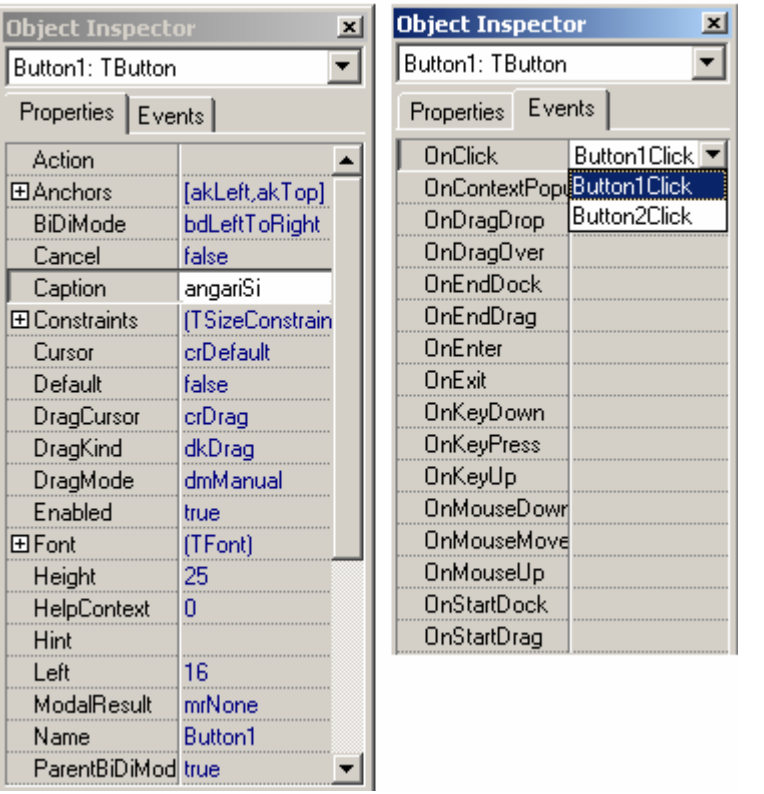
2.3.15. QReport - ანგარიშების შორმირება და ბეჭდვა

141		QuickRep	ანგარიშების დასაბეჭდი ხაშუალების დაკვნება დანართში
142		QRSubDetail	ანგარიშში დაშტებითი მონაცემების ჩართვა
143		QRStringsBand	ანგარიშში დაშტებითი ტექსტის ჩართვა
144		QRBand	ანგარიშის აკვეა შესახებ ხაბეჭდი კომპონენტების განლაგებით
145		QRChildBand	ანგარიშში "შვილი"-ზოლების შექმნა,რომელშიც შეიძლება იყოს ბეჭდვის სხვა კომპონენტებიც
146		QRGroup	მონაცემთა დაჯგუფების კომპონენტი
147		QRLabel	ანგარიშში ტექსტის მოთავსება
148		QRDBText	ანგარიშში ტექსტის მოთავსება მონაცემთა ბაზიდან
149		QRExpr	აგების და ანახავს გამოხახულების (ცვლებისთვის) და სისტემურ სიდიდეების (დრო,თარიღი)
150		QRSysData	სისტემური მონაცემების ანახვა
151		QRMemo	ანგარიშში მრავალსტრიქონიანი ტექსტების განლაგება

152		QR ExprMemo	ანგარიშში მათემატიკური გამოსახულებების ტექსტების განლაგება
153		QR RichText	ანგარიშში მრავალტრანსფორმირებული ტექსტების განლაგება გამდიდრებული ფორმატით Rich Edit
154		QR DBRichText	ანგარიშში მრავალტრანსფორმირებული ტექსტების განლაგება გამდიდრებული ფორმატით Rich Edit მონაცემთა ბაზიდან
155		QR Shape	ანგარიშში გრაფიკული ფორმის დამატება
156		QR Image	ანგარიშში გამოსახულებების ბეჭდვა
157		QR DBImage	ანგარიშში გამოსახულებების ბეჭდვა მონაცემთა ბაზიდან
158		QR CompositeReport	შეერთებული ანგარიშების ამება
159		QR Preview	დოკუმენტის ანგარიშის წინასწარ დათვალიერება ეკრანზე
160		QR Text Filter	ტექსტებისთვის ფილტრის დაწერა
161		QR CSVFilter	ტექსტში გამყოფების ჩაწერა
162		QR HTML Filter	html -ფილტრის ჩაწერა
163		QR Chart	ანგარიშში დიაგრამების ბეჭდვა მონაცემთა ბაზიდან

2.4. ობიექტების ინსპექტორი (Object Inspector)

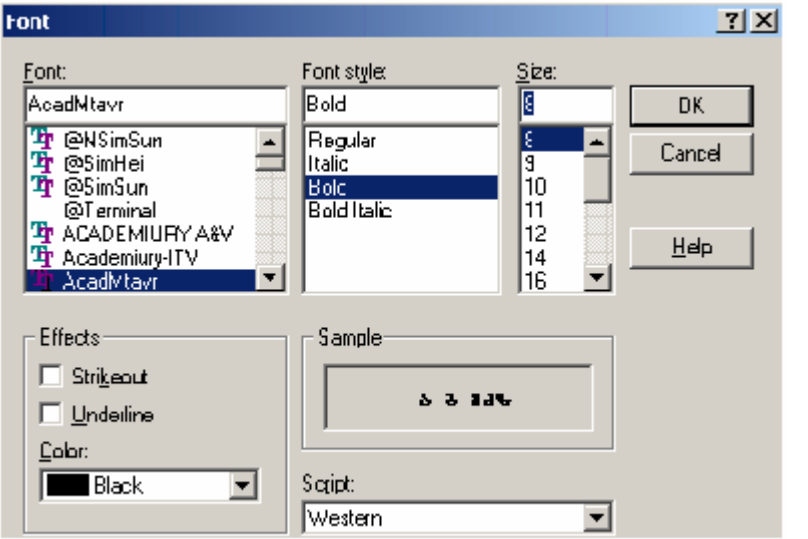
დანიშნულება: უზრუნველყოფს C++Builder ობიექტების თვისებების (Properties) არჩევას (შეცვლას) და მათი მოვლენების (Events) მართვას (მაგ. სისტემის რეაქცია „ღილაკზე“ (ეს ობიექტია) დაჭერისას). 2.11 ნახაზზე მოცემულია Object Inspector-ის ფანჯარა ორი გადასართავი გვერდით (Properties და Events):



ნახ.2.11

ნახაზზე ილუსტრირებულია შემთხვევა, როდესაც ფორმაზე (Form1) დავდეთ ორი კომპონენტი Button1, Button2 პანელიდან Standard და ვცდილობთ მათ დამუშავებას.

პირველი Button1 მონიშნულია, ე.ი. აქტიურია და ობიექტების ინსპექტორის ფანჯარა მას ეკუთვნის. ჩვენ Object Inspector-ში ავირჩიეთ სტრიქონი (თვისება) Font და მის მარჯვენა ნაწილიდან (TFont . . .) გამოვიძახეთ შრიფტების ასარჩევი ფანჯარა (ნახ.2.12):



ნახ. 2.12

მაგ., AcadMtavr ფონტის არჩევის შემდეგ Object Inspector-ში ვდგებით Caption თვისებაზე და ვწერთ სიტყვას „ანგარიში“. მეორე გვერდზე (Events) ჩანს ღილაკებისათვის OnClick თვისების მნიშვნელობათა არჩევის პროცედურა. მაგ., Button1Click შეესაბამება შემთხვევას, როცა უნდა გამოვიძახოთ ანგარიშის პროგრამა Form2–ზე, ხოლო Button2Click

გათვალისწინებულია მე-2 ღილაკისათვის, რომელიც გამოიძახებს სხვა ამოცანას, მაგ., მე-3 ფორმას და ა.შ..

რეალურად ეს მოვლენები (Button1Click, Button2Click) პროგრამირდება სისტემის მიერ ობიექტ-ორიენტირებული დაპროგრამების „მემკვიდრეობითობის“ პრინციპის საფუძველზე TButton-კლასიდან (Template Button). ამგვარად, TButton არის საბაზო კლასი, ხოლო ButtonClick - წარმოებული კლასი, რომლის კონკრეტული ეგზემპლარებია Button1Click, Button2Click ობიექტები.

მათი შესაბამისი პროგრამული კოდები მოცემულია Unit1.cpp პროგრამაში, რომელიც კავშირშია Form1 ფორმის დამუშავებასთან:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{  Form2->Show(); }   // Form2-ის გამოძახება
//---------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{ Form3->Show(); }  // Form3-ის გამოძახება
```

Object Inspector მეტად საჭირო და მნიშვნელოვანი ინსტრუმენტია კომპიუტერული სისტემების სწრაფად ასაწყობად. ის ხშირად იქნება გამოყენებული ჩვენს მიერ სისტემის ცალკეული დეტალების დასამუშავებლად, ამიტომაც აქ დროებით დავასრულებთ მის განხილვას.

2.5. C++Builder-ის პროექტის ფაილები

პროექტის ძირითადი ფაილებიდან ჩვენ ვახსენეთ ზემოთ **.cpp** ფაილი (**Unit1.cpp**-ფორმისათვის). **Project.cpp** არის კომპიუტერული სისტემის WinMain-მთავარი ფუნქცია, რომლითაც შესრულებაზე გაიშვება გამოყენებითი პროგრამა. გარდა ამისა სისტემას აქვს შემდეგი აუცილებელი ფაილები: **.bpr**-პროექტის ფაილი, რომელიც XML-ფორმატით ინახება (ნახ.2.13), ტექსტურია და შეიცავს კომპილირებისათვის საჭირო ყველა ფაილის სიას.

```
<?xml version='1.0' encoding='utf-8' ?>
<!-- C++Builder XML Project -->
<PROJECT>
 <MACROS>
  <VERSION value="BCB.05.03"/>
  <PROJECT value="Project1.exe"/>
  <...>
```

ნახ.2.13. bpr-ფაილის ფრაგმენტი XML-ფორმატში

.res - პროექტის რესურსების ფაილი, რომელიც ორობითია და შეიცავს, მაგ., პიქტოგრამების, კურსორების და სხვ. კომპონენტებს.

.h – სათავო **(header)** ფაილი, მაგ., <iostream.h>, <math.h> და სხვ.

.hpp – კომპონენტის სათავო ფაილი, რომელიც C++Builder-ის ბიბლიოთეკიდან მიუერთდება ჩვენს პროგრამას ახალი კომპონენტის დამატებისას ფორმაზე.

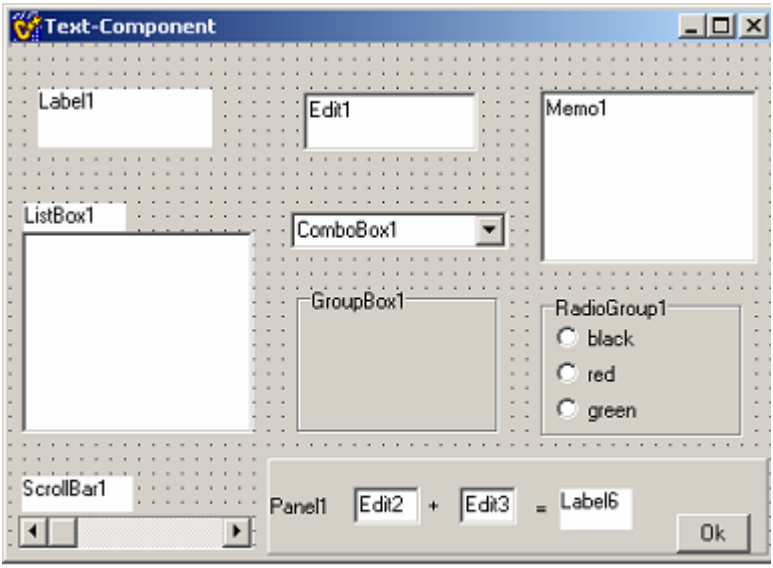
.dfm – ფორმის ფაილია. ყველა ფორმის ფაილს თავისი საკუთარი **cpp** და **h** ფაილები აქვს.

.exe – შესრულებადი (მუშა პროგრამა) ფაილი, **obj** – ობიექტური მანქანური კოდი, **.dll** – დინამიკურად მიერთებადი ბიბლიოთეკა, **.tds** – სიმბოლოთა ცხრილების ფაილი, **.hlp** – დახმარების (help) ცნობათა ფაილი, **.bmp, .wmf, ico** – გრაფიკული საინტერფეისო ფაილები, **.il?** – (ilc, ild, ilf, ils) სპეცფაილები კომპილირებისათვის, **.~bp, ~df, ~cp, ~h** – სარეზერვო ფაილებია.

<u>შენიშვნა:</u> **პროგრამის გადასატანად სხვა კომპიუტერზე აუცილებელია cpp, h, dfm, bpr და res ფაილები, დანარჩენები იქმნება ავტომატურად. ~??, tds, exe, obj, il? ფაილები შეიძლება წაიშალოს.**

2.6. ფორმებთან მუშაობა (Forms)

<u>დანიშნულება:</u> ფორმა C++Builder-ის ძირითადი სამუშაო ადგილი, ფანჯარაა. ფორმაზე ხდება სასურველი გამოყენებითი სისტემის დაპროექტება, კომპონენტების გადმოტანა პანელებიდან, მასზე სასურველი ინტერფეისის დამუშავება, შუალედური და საბოლოო შედეგების ასახვის რეალიზება. შეიძლება ითქვას, რომ ფორმები და კომპონენტების მრავალფეროვანი პანელები ძირითადი „სამშენებლო“ მასალებია პროგრამული პაკეტების შესაქმნელად. ფორმის სრული ფრაგმენტი ნაჩვენებია 2.14 ნახაზზე. მაგალითისათვის მასზედ გადმოტანილია რამდენიმე კომპონენტი Standard პანელიდან.



ნახ.2.14. ფორმის მაგალითი კომპონენტებით

ტექსტის გამოსატანად ფანჯარაში იყენებენ სხვადასხვა სახის კომპონენტს. მაგ., Label-სათაურებისთვის, Edit-ერთსტრიქონიანი რედაქტირებადი ველი. Memo-მრავალსტრიქონიანი რედაქტირებადი ფანჯრა. ListBox-სტანდარტული ფანჯარა სიით, რომლიდანაც აირჩევა პუნქტი Object Inspector-ის Items-თვისებაში მოთავსებული სიიდან. ComboBox-რედაქტირებადი სია (მარჯვენა ისრის დაჭერისას გამოჩნდება სია, რომელიც ასევე Object Inspector-ის Items-თვისებაშია მოთავსებული).

რადიოღილაკების პანელები RadioGroup და GroupBox გამოიყენება ალტერნატიული, ურთიერთგამომრიცხავი პუნქტის ასარჩევად იმ სიიდან, რომელიც Object Inspector-

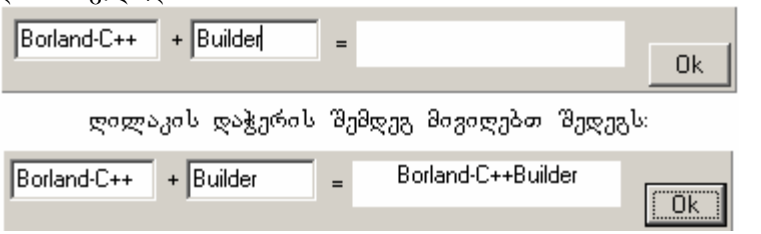
ის Items-თვისებაშია ჩაწერილი (მაქსიმუმ 17 სტრიქონი). ამ ინდიკატორული კომპონენტების ზედა მარცხენა კუთხეში მოთავსებული სახელები შეიტანება Object Inspector-ის Caption-თვისებაში.

გადასახვევი სახაზავი ScrollBar გამოიყენება ჩვეულებრივ Windows-ფანჯრებში გადასაადგილებლად.

Panel-კომპონენტი ლოკალური ადგილია რამდენიმე დაკავშირებული კომპონენტის მოსათავსებლად. პანელის გადატანით სხვა ადგილას, მას ყველა კომპონენტი გადაჰყვება.

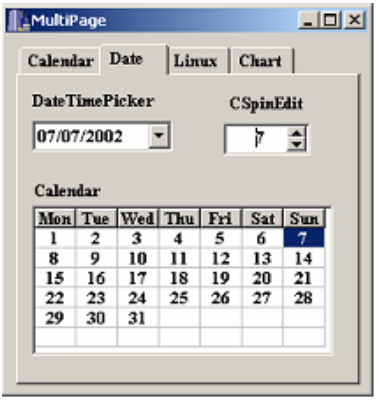
ფორმის ასამუშავებლად C++Builder-ის მთავარი მენიუს Project-პუნქტში ავირჩიოთ Compile Unit (Alt+F9) ან მწვანე სამკუთხედი (Run – F9).

ჩვენს პანელზე მოთავსებულია ორი Edit-კომპონენტი, მაგ., ორი სიტყვის შესატანად და Label-კომპონენტში გადასაბმელად:



ფორმის (ფანჯრის) ფართობის ეფექტურად გამოსაყენებლად შექმნილია მრავალფურცლიანი (მრავალგვერდიანი) პანელის კომპონენტი PageControl. იგი მდებარეობს Win32 პანელზე. 2.15 ნახაზზე შევქმნით 3 - TabSheet: Calendar – MonthCalendar-ით, Date – DateTimePicker-ით და Linux ნახატი Additional-ის Image-კომპონენტით.

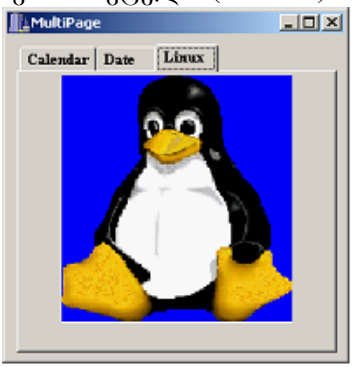
ნახ.2.15-ა



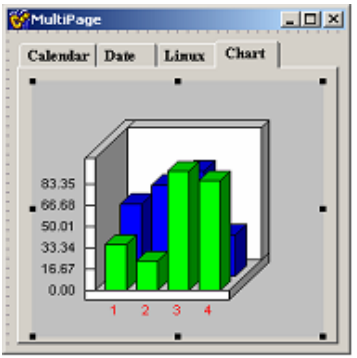
ნახ.2.15-ბ.

ფორმაზე გრაფიკული კომპონენტების გამოყენება ერთ-ერთი მნიშვნელოვანი მიღწევაა ილუსტრირებული ინტერფეისების შესაქმნელად.

მაგალითად, Active-X პანელზე Chartfx-კომპონენტით გამოიძახება დიაგრამების რედაქტორი, რომლითაც შესაძლებელია როგორც მონაცემთა, ასევე დიაგრამების ტიპების შეცვლა (ნახ.2.16)



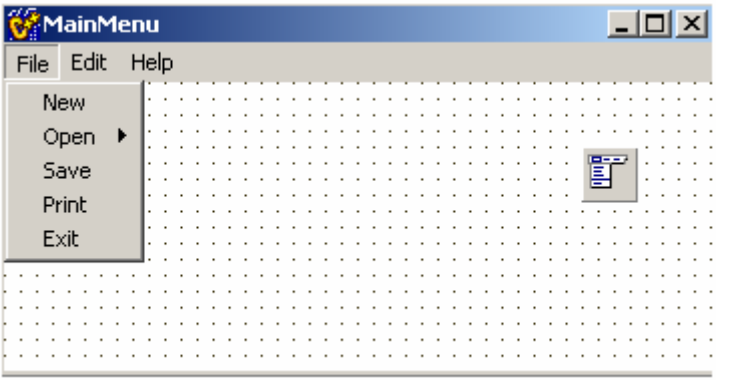
ნახ.2.15.-გ



ნახ.2.16

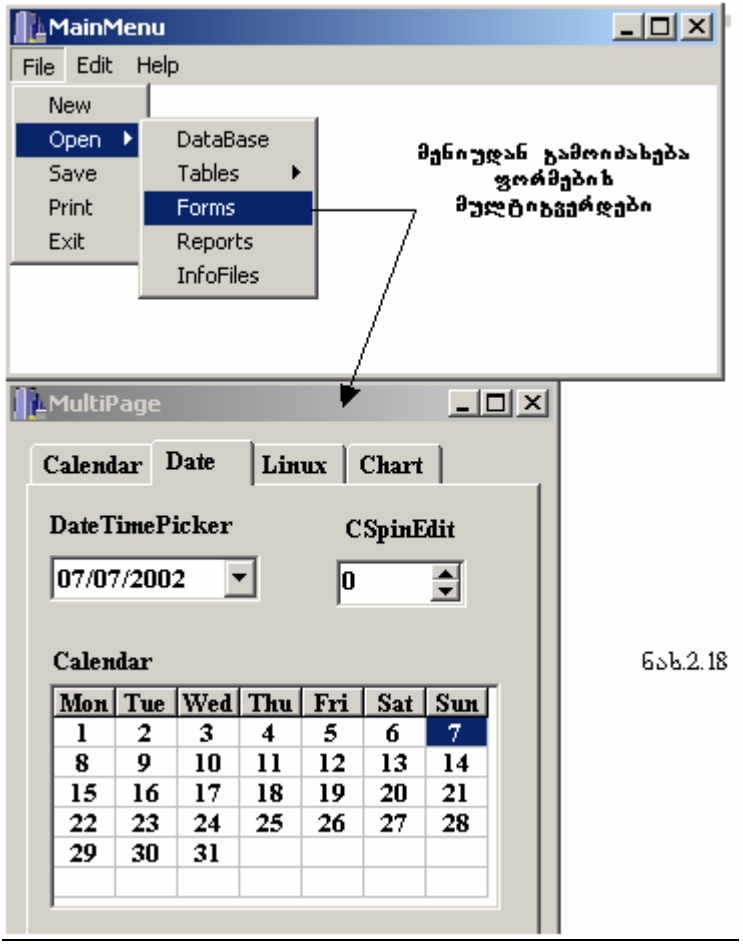
2.7. მთავარი მენიუს აგება ფორმაზე (MainMenu)

<u>დანიშნულება:</u> მთავარი მენიუ ასაგები კომპიუტერული სისტემის ძირითადი და საჭირო ელემენტია. Windows გარემოში სისტემის დაპროექტება სწორედ მენიუთი უნდა დავიწყოთ. აქედან კარგად ჩანს გამოყენებითი სისტემის ამოცანები, საშუალებები, დახმარებები და ა.შ. ფორმაზე გადმოვიტანოთ Standard პანელის MainMenu-კომპონენტი (ნახ.2.17 პიქტოგრამა მარჯვნივ).



ნახ.2.17. მთავარი მენიუს აგება

ფორმა თავიდან ცარიელია. მენიუს პიქტოგრამაზე თაგუს მარცხენა ღილაკის 2-ჯერ დაწკაპუნებით გამოჩნდება დამხმარე ფანჯარა მენიუს პუნქტების მიმდევრობით ჩასაწერად (File, Edit, ... , New, Open, . . .). ვერტიკალური მენუს ქვეპუნქტების შესაქმნელად, მაგ., Open-ისთვის: ვდგებით Open-ზე და თაგუს მარჯვენა ღილაკით გამოვიტანთ დამხმარე მენიუს და ავირჩევთ CreateSubmenu პუნქტს. საბოლოოდ გვექნება ნახ.2.18.



ნახ.2.18

<u>შენიშვნა</u>: მთავარი მენიუს დაკავშირება სხვა ფორმებთან, რომლებიც ამ მენიუდან გამოიძახება. განიხილება სამი მომენტი:

1 – მთავარი მენიუს (Form2) კავშირი მულტი-გვერდების (Form1) ფორმასთან. ვდგებით მენიუს ფორმაზე, სისტემის File-მენიუში ვირჩევთ **Include Unit Hdr** (Alt-12) სტრიქონს. გამოდის ფანჯარა ფორმათა

ჩამონათვალით (ჩვენს შემთხვევაში ერთი Form1). ვირჩევთ მას, Ok და კავშირი ორ ფორმას შორის დამყარდა.

2 – ახლა კონკრეტულად, მთავარი მენიუს File | Open | Forms პუნქტმა გახსნას მულტიგვერდების ფორმა (Form1). ამისათვის ვდგებით Forms-პუნქტზე, 2-ჯერ მარცხენათი დავაწკაპუნებთ და შევდივართ Unit2.cpp პროგრამის კოდში, რომელშიც კურსორი დგება მის მიერვე შექმნილ ფუნქციაში:

```
void __fastcall TForm2::Forns1Click(TObject *Sender)
{
  _                    // Cursor
}
```

კურსორის ადგილას ხელით უნდა ჩავწეროთ საჭირო ოპერატორი:

მაგ.:
```
void __fastcall TForm2::Forns1Click(TObject *Sender)
 {
  Form1->Show();   // ფორმის გამოძახება
 }
```

ამის შემდეგ, თუ ავამუშავებთ სისტემას კომპილაცია-შესრულებაზე (Run), ვნახავთ, რომ შედეგში გამოიტანება Form1-ფორმის (ძველი) შედეგი, Form2 კი არაა ! აქ საჭიროა მე-3 მოთხოვნის შესრულება.

3 - გავხადოთ Form2 მთავარ ფორმად: ამისათვის სისტემის მენიუში Project | Options | Forms –ში Main form-ის მნიშვნელობად ჩავსვათ Form2 და Ok.

Run-პროცედურა შეასრულებს ყველაფერს მოწესრიგებულად და შედეგში მივიღებთ მთავარი მენიუს

ფორმას, რომელშიც File | Open | Forms არჩევით გამოვა ეკრანზე მეორე მულტიფორმაც.

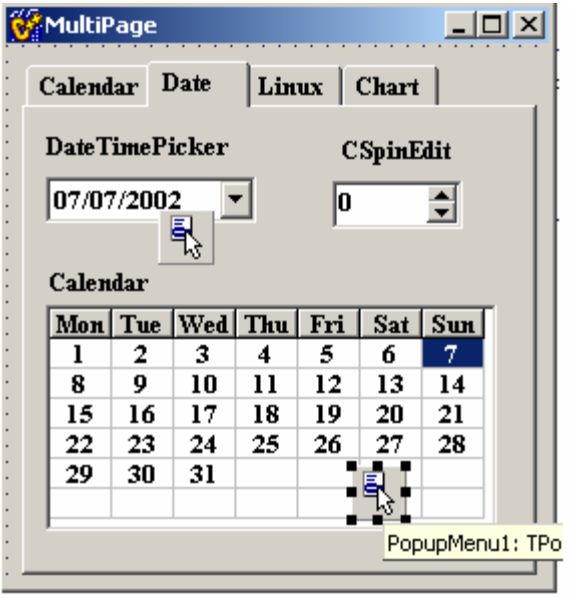
მთავარი მენიუს დანარჩენი პუნქტებიც ასევე უნდა დაუკავშირდეს სხვა ფორმებს.

თუ საჭიროა მულტი-ფორმიდან ისევ მთავარ მენიუში დაბრუნება, მაშინ გამოიყენება ან X – ფანჯრის დახურვის ღილაკი, ან მულტი-ფორმაზე დაიდება ახალი ღილაკი, მაგ., Back. ამ ღილაკზე 2-ჯერ დაწკაპუნებით შევდივართ Unit1.cpp პროგრამის კოდში და ჩავწერთ: Form2->Show(). ამასთანავე, ვდგებით Form1-ზე და File –ში ვირჩევთ **Include Unit Hdr** სტრიქონს და Form2-ს.

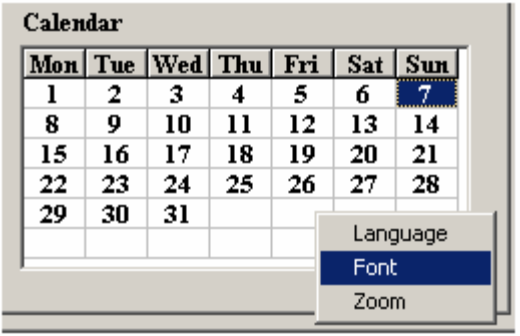
2.8 კონტექსტური მენიუს აგება (PopupMenu)

<u>დანიშნულება:</u> ფორმაზე ყოველ კომპონენტს შეიძლება ჰქონდეს საკუთარი მენიუ, რომელიც ამ კომპონენტზე დადგომითა და თაგუს მარჯვენა ღილაკის დაჭერით გამოიტანება.

ასეთი კონტექსტური მენიუს შექმნა ხდება Standard | PopupMenu კომპონენტით. ის უნდა გადმოვიტანოთ ფორმაზე, 2-ჯერ დაწკაპუნებით ჩაირთვება მენიუს რედაქტორი, ჩავწერთ სიტყვებს. ამის შემდეგ ეს კომპონენტი უნდა დავაკავშიროთ ფორმაზე მდებარე ობიექტს, მაგ., Calendar-ს. ვდგებით კალენდრის ცხრილზე და Object Inspector | Properties | PopupMenu-ში ვირჩევთ PopupMenu1-ს, ასევე DateTimePicker-ისთვის კი PopupMenu2-ს (ნახ.2.19, 2.20).



ნახ. 2.19. Popup Menu - კომპონენტი



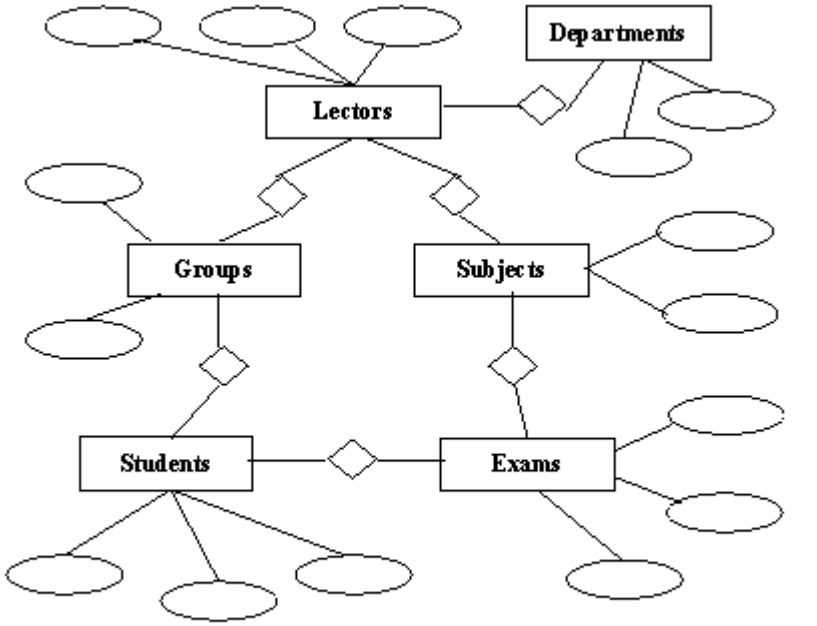
ნახ. 2.20. Popup Menu - შედეგი

2.9. მონაცემთა ლოკალურ ბაზასთან მუშაობა

დანიშნულება: მონაცემთა ფაილების შენახვა, დამუშავება, მოძებნა და მომხმარებელზე (ან პროგრამებზე) მიწოდება. Borland_C++ Builder პაკეტი იყენებს მონაცემთა როგორც ლოკალურ ბაზებს (dbf-ფაილები dBase, Access, Vis-FoxPro, db-ფაილები Paradox ბაზების მართვის სისტემებისათვის), ასევე განაწილებულს (gdb-ფაილები InterBase მბ-სისტემისთვის).

ამჯერად გავარჩიოთ საილუსტრაციო მაგალითი „საუნივერსიტეტო კათედრის“ საპრობლემო სფეროსათვის. კერძოდ, კათედრაზე გვაქვს ინფორმაციული ბაზები ლექტორების, ჯგუფების, სტუდენტების, მაგისტრანტების, სასწავლო დისციპლინების, გამოცდების და ა.შ. მათ შორის არსებობს რელაციური, ფუნქციური და მემკვიდრეობითი კავშირები. როგორ უნდა აიგოს ასეთი საპრობლემო სფეროსათვის კომპიუტერული სისტემა მონაცემთა ბაზების გამოყენებით?

უპირველეს ყოვლისა, დაპროექტების სტადიაზე უნდა განისაზღვროს საპრობლემო სფეროს კონცეპტუალური მოდელი, ანუ აიგოს ER-მოდელი (Entity-Relationship-Model). 2.21 ნახაზზე ნაჩვენებია ეს სქემა ჩვენი საკვლევი ობიექტისათვის.



ნახ. 2.21. საპრობლემო სფეროს ER მოდელი

ობიექტ-ორიენტირებული მოდელირებისა და დაპროგრამების მეთოდების საფუძველზე აღნიშნული არსები (ობიექტები) და რელაციები (კავშირები) ქმნის კლასების სიმრავლეს (სტრუქტურირებული მონაცემებისა და მათი დამუშავების პროცედურების ინკაფსულაციით მიღებული ერთობლიობა). კლასთა კონკრეტული ელემენტები – ობიექტებია.

თუ კლასს განვიხილავთ როგორც არაერთგვაროვანი ატრიბუტებისაგან (ველებისაგან) შედგენილ სტრუქტურას (ცხრილს), მაშინ ობიექტები ამ ცხრილის სტრიქონებია (კორტეჟები). ნახაზზე კლასები მართკუთხედებით, კავშირები რომბებით, ატრიბუტები ოვალებითაა ასახული.

C++ ენაზე დაპროგრამების დროს კლასები, ატრიბუტები და კავშირები სპეციალური ხერხებით აღიწერება, მაგ., Class, Properties, Relationship და ა.შ. ფიზიკურ დონეზე კლასები და ობიექტები თავსდება ორგანზომილებიან ცხრილებში (Tables). მათი აღწერა Header-ფაილებში ინახება. ამგვარად, მონაცემთა ბაზა არის ცხრილთა ერთობლიობა, რომელთაც გააჩნია უნიკალური სახელები და გასაღებური ველები.

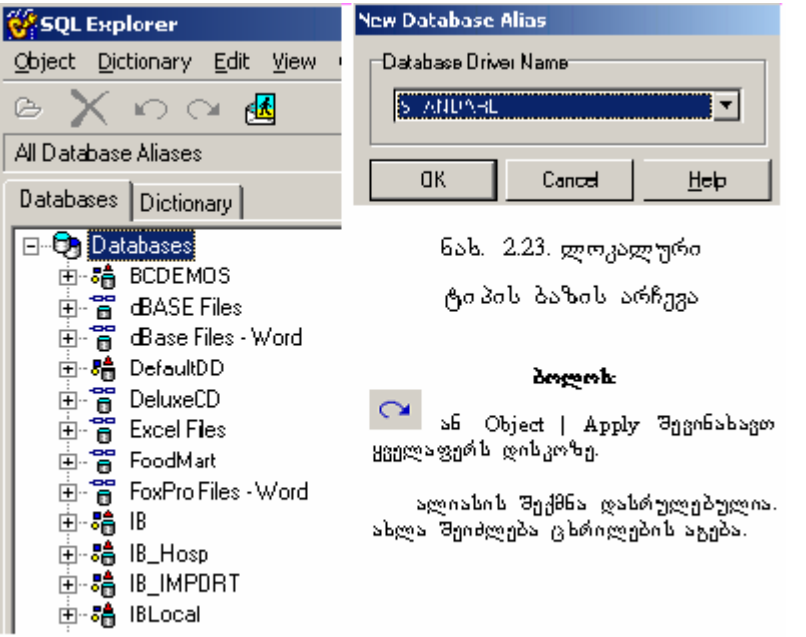
2.10. მონაცემთა ბაზის ფსევდონიმის (ალიასის) შექმნა

<u>დანიშნულება:</u> ფსევდონიმი (Alias) შეიცავს მთელ ინფორმაციას მონაცემთა ბაზის ფაილების შესახებ, უზრუნველყოფს მათ წვდომას და სისტემის მოქნილ ტრანსპორტირებას სხვა კომპიუტერებზე.

მონაცემთა ბაზის ალიასი იქმნება ერთხელ და კომპიუტერული სისტემა მუშაობს მასთან. თუ იცვლება მონაცემთა ფიზიკური განლაგება, კატალოგები ან სხვა ობიექტები, კომპიუტერული სისტემა არ საჭიროებს ყველა ამ ცვლილებების ასახვას, მხოლოდ მიეთითება ალიასის ახალი მდებარეობა (წვდომის გზა).

Borland_C++ Builder სისტემაში ალიასის შექმნის სამი ხერხი არსებობს: Database Desktop, BDE Administrator და Database Explorer. შედეგი სამივე შემთხვევაში ერთნაირია, შესრულების ინსტრუმენტი კი განსხვავებული.

Borland_C++ Builder-ის მენიუში Database | Explore გამოიტანს 2.22 ნახაზზე მოცემულ ფანჯარას. ჩვენი მონაცემთა ბაზის ალიასის შესაქმნელად ვიღებთ: Object | New და 2.23 ფანჯარაში ვირჩევთ STANDARD (ესაა ლოკალური ბაზა Dbase ან Paradox). თუ ვაპირებთ განაწილებულ ბაზებთან მუშაობას, მაშინ ვირჩევთ INTERBASE-ს.



ნახ. 2.22. საწყისი მდგომარეობა

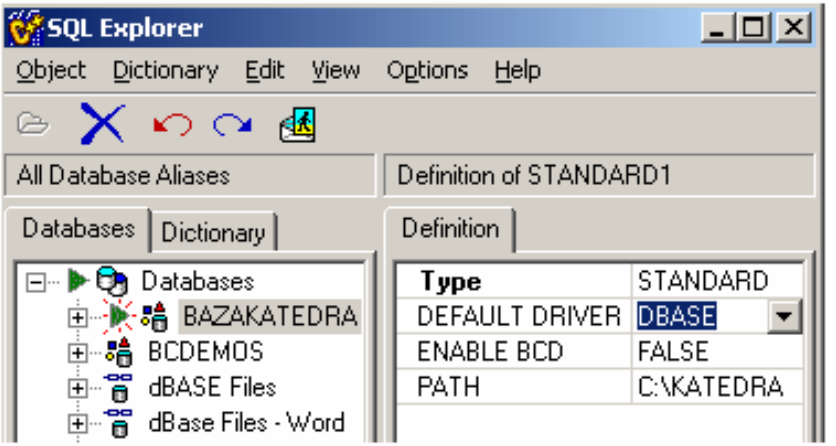
ნახ. 2.23. ლოკალური ტიპის ბაზის არჩევა

ბოლოს:

ან Object | Apply შეცინახავთ ცვლილებებს დისკოზე.

ალიასის შექმნა დასრულებულია. ახლა შეიძლება ცხრილების ატება.

შემდეგ ტექსტი STANDARD შევცვალოთ BAZAKATEDRA ტექსტით (ნახ.2.24). აქვე ფანჯრის მარჯვენა ნაწილში DEFAULT DRIVER-ში შევარჩიოთ DBASE ან PARADOX, ხოლო PATH-ში მივუთითოთ ის გზა კატალოგში, სადაც შეინახება ბაზის ფაილები და კომპიუტერული სისტემა. ასეთია C:\KATEDRA.



ნახ. 2.24. ალიასი BAZAKATEDRA, ტიპი DBASE და კატალოგის გზა C:\KATEDRA

2.11. მონაცემთა ბაზის ცხრილების (Tables) შექმნა

დანიშნულება: ცხრილი (Table) მონაცემთა ბაზის მთავარი კომპონენტია, იგი dbf-ფაილი (dBase for Windows) ან db-ფაილია (Paradox) ბაზებისათვის. ერთ ბაზაში შეიძლება იყოს ბევრი ცხრილი. მათ შორის რეალიზებადია რელაციური ურთიერთკავშირები და ლოგიკური მთლიანობის უზრუნველყოფის ასპექტები.

ცხრილის (Table) შექმნა ხორციელდება Database Desktop ინსტრუმენტით, რომელიც გამოიძახება ორი ხერხით:

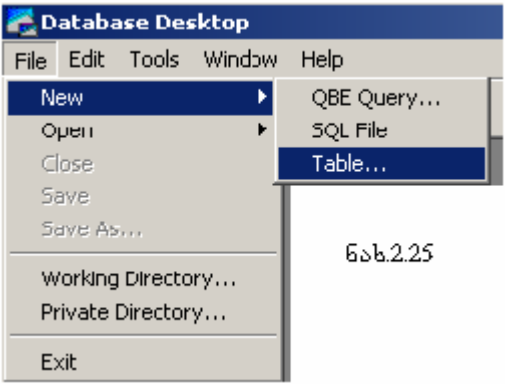
Start | Programs | BorlandC++Builder5 | Database Desktop

ან

Borland_C++Builder სამუშაო გარემოს მთავარი მენიუდან:

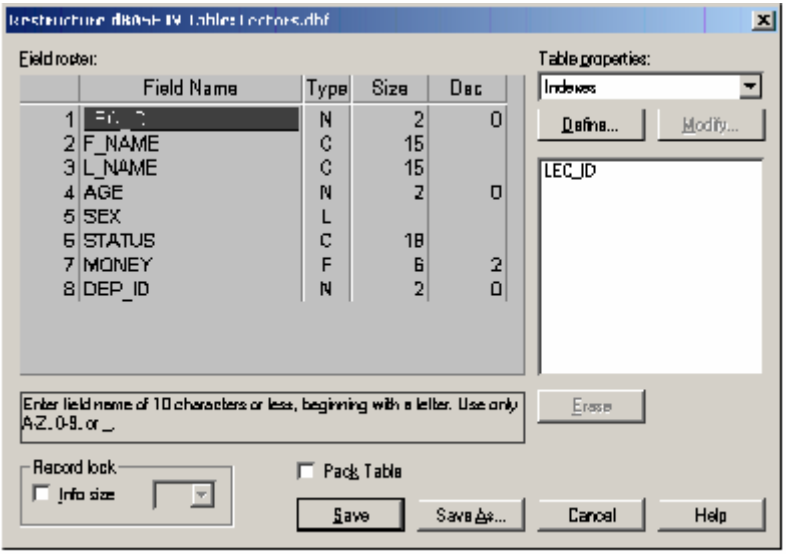
Tools | Database Desktop

რის შემდეგაც გამოვა ახალი ფანჯარა (ნახ.2.25).



ნახ.2.25

2.26 ნახაზზე ილუსტრირებულია თვით Database Desktop–ის სამუშაო გარემო. ჩვენ 2.21 ნახაზზე დავაპროექტეთ საპრობლემო სფეროს კონცეპტუალური მოდელი (ER-M). ახლა თითოეული ობიექტისთვის (მართკუთხედი) უნდა ავაგოთ ცხრილი ატრიბუტებით (ოვალები). მაგალითისათვის 2.26 ნახაზზე ნაჩვენებია Lectors – ფაილის სტრუქტურა. ასევე უნდა აიგოს Groups, Students და სხვა სტრუქტურებიც.

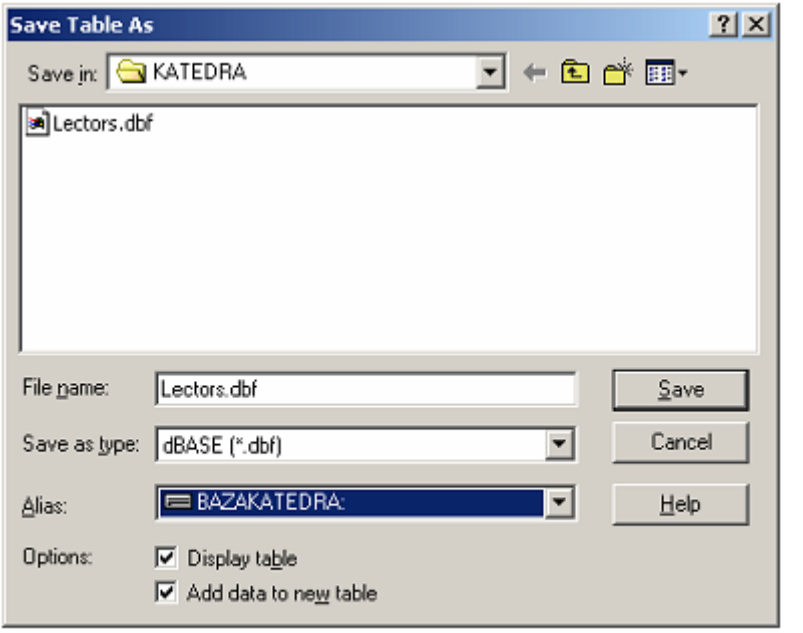


ნახ.2.26

აქ ჩანს ლექტორების ცხრილის Lectors.dbf სტრუქტურის ფაილი ატრიბუტთა დასახელებებით, ტიპებით, სიგრძეებით. მარჯვენა ნაწილში Define ღილაკით შევქმნით უნიკალური (Unique) ინდექსი (პირველადი გასაღებური ატრიბუტი), ესაა LEC_ID. ე.ი. ცხრილში არ იქნება ორი ლექტორი ერთიდაიმავე ინდექსით, ეს გაკონტროლდება ავტომატურად სისტემის მიერ.

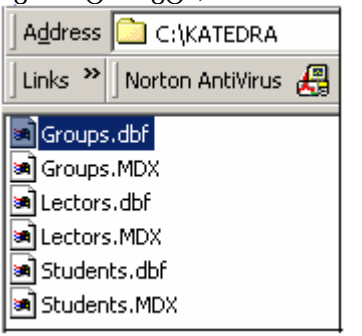
ბოლოს Save As ღილაკით სტრუქტურას შევინახავთ ბაზაში. ამისათვის ფანჯარაში (ნახ.2.27) უნდა მივუთითოთ ჩვენი ალიასი BAZAKATEDRA, რომელიც ავტომატურად გამოიტანს C:\KATEDRA კატალოგს, შევიტანთ ფაილის სახელს Lectors და Save.

ამის შემდეგ თქვენ თვითონ შექმენით ახალი სტრუქტურები Groups, Students და ა.შ. ისინი ჩაემატება 2.27 ნახაზს და 2.28 სახეს მიიღებს.



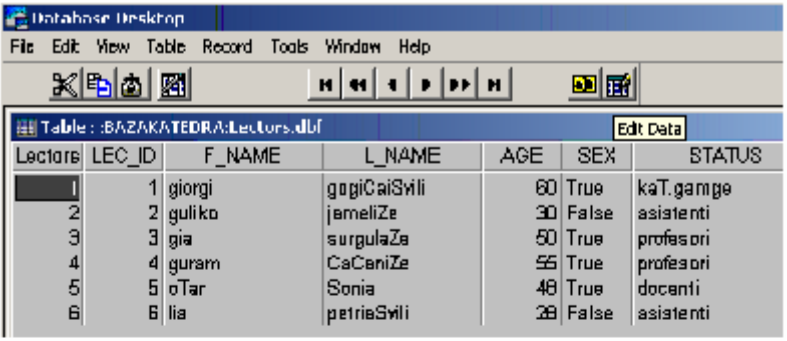
ნახ.2.27

აქ dbf-ფაილები ცნობილია. რაც შეეხება MDX-ფაილებს, ესაა ჩვენს მიერ ინდექსირებული ცხრილები. როგორც ვხედავთ, ყოველ ძირითად ფაილს ახლავს ინდექსირებული ფაილი. ინდექსირება შეიძლება მოხდეს რამდენიმე ატრიბუტით (შედგენილი გასაღებური ატრიბუტი).



ნახ.2.28

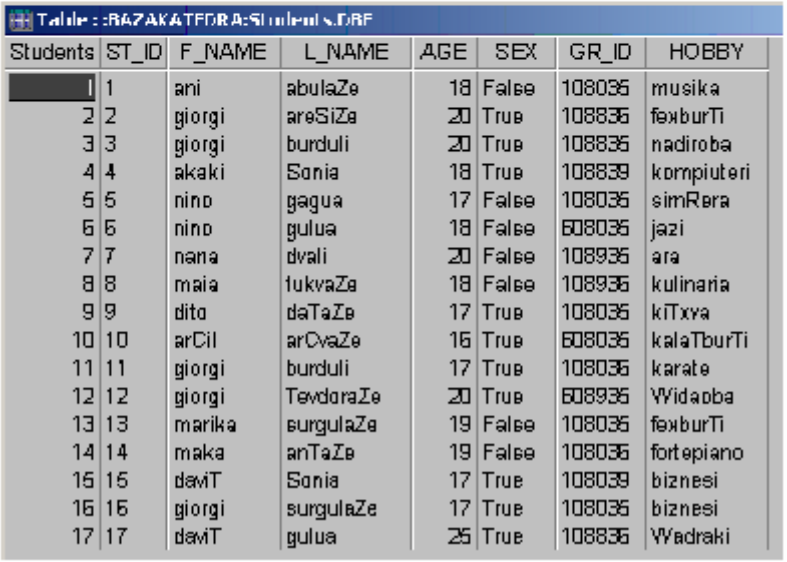
შემდეგი ეტაპია აგებულ ცხრილებში მონაცემების შეტანა. ამისათვის შევალთ ისევ Database Desktop სისტემაში, გავხსნით (Open) ცხრილს Lectors მარჯვენა-ზედა კუთხეში ჩავრთოთ Edit Data (ნახ.2.29). შემდეგ შეიძლება სტრიქონების შეტანა ცხრილში.



Lectors	LEC_ID	F_NAME	L_NAME	AGE	SEX	STATUS
1	1	giorgi	gogiCaiSvili	60	True	kaT.gange
2	2	guliko	jameliZe	30	False	asistenti
3	3	gia	surgulaZe	50	True	profesori
4	4	guram	CaCaniZe	55	True	profesori
5	5	oTar	Sonia	48	True	docenti
6	6	lia	petriaSvili	28	False	asistenti

ნახ.2.29. ცხრილში Lectors მონაცემების შეტანა/კორექტირება

შევიტანოთ მონაცემები ცხრილებში Groups და Students.



Students	ST_ID	F_NAME	L_NAME	AGE	SEX	GR_ID	HOBBY
1	1	ani	abulaZe	18	False	108036	musika
2	2	giorgi	areSiZe	20	True	108836	fexburTi
3	3	giorgi	burduli	20	True	108836	nadiroba
4	4	akaki	Sonia	18	True	108839	kompiuteri
5	5	nino	gagua	17	False	108036	simRera
6	6	nino	gulua	18	False	808036	jazi
7	7	nana	dvali	20	False	108936	ara
8	8	maia	tukvaZe	18	False	108936	kulinaria
9	9	dito	daTaZe	17	True	108036	kiTxva
10	10	arCil	arCvaZe	16	True	808036	kalaTburTi
11	11	giorgi	burduli	17	True	108036	karate
12	12	giorgi	TevdoraZe	20	True	808936	Widaoba
13	13	marika	surgulaZe	19	False	108036	fexburTi
14	14	maka	anTaZe	19	False	108036	fortepiano
15	15	davT	Sonia	17	True	108039	biznesi
16	16	giorgi	surgulaZe	17	True	108036	biznesi
17	17	davT	gulua	25	True	108836	Wadraki

ნახ.2.30. ცხრილი Students

Table ::BAZAKATEDRA:Groups.dbf

Groups	GR_ID	DEP_ID	LANGUAGE	KURS	SPEC_	ST_RAOD
1	108035	94	qarTuli	2	2202	25
2	108036	94	qarTuli	2	2202	30
3	108039	94	rusuli	2	2202	19
4	608035	94	qarTuli	2	2202	20
5	108935	94	qarTuli	3	2202	23
6	108936	94	qarTuli	3	2202	17
7	108939	94	rusuli	3	2202	15
8	608935	94	qarTuli	3	2202	17
9	108835	94	qarTuli	4	2202	30
10	108836	94	qarTuli	4	2202	16
11	108839	94	rusuli	4	2202	18
12	608835	94	qarTuli	4	2202	14

ნახ.2.31. ცხრილი Groups

კავშირი ლექტორებსა და სტუდენტებს შორის არ არსებობს ბაზაში, რაც არაა კარგი. საჭიროა ეს დამოკიდებულება M:N (მრავალი-მრავალთან)შემოვიტანოთ დამატებითი რელაციური ცხრილით Lec_Gr. მასში იქნება ინფორმაცია იმის შესახებ, თუ რომელი ლექტორი რომელ ჯგუფებს ასწავლის, რა საგანს და რომელ სემესტრში (ნახ.2.32-1,2). საგნები (Subjects) კოდირებულია და ცალკე ცხრილში ინახება.

Restructure dBASE IV Table: Lec_Gr.DBF

Field roster:

Table prop

Indexes

Define

LEC_GR

	Field Name	Type	Size	Dec
1	LEC_ID	N	2	0
2	JG_ID	C	6	
3	SUBJECT	N	2	0
4	SEMESTR	N	1	0

ნახ.2.32-1. ცხრილი Groups-ის სტრუქტურა

Table : :BAZAKATEDRA:Lec_Gr.DBF

Lec_Gr	LEC_I	JG_ID	SUBJECT	SEMEST
1	1	108039	5	3
2	1	108939	4	6
3	1	608035	5	3
4	1	608935	4	6
5	2	108135	1	2
6	2	108136	1	2
7	2	108137	1	2
8	3	108035	2	3
9	3	108036	2	3
10	3	608035	2	3
11	3	108935	2	6
12	3	108936	2	6
13	3	608935	2	6
14	4	108035	5	3
15	4	108036	5	3
16	4	108935	4	6
17	4	108936	4	6
18	6	108939	6	6
19	6	608939	6	6
20	1	608939	4	6
21	1	608039	5	3

ნახ.2.32-2. ცხრილი Groups

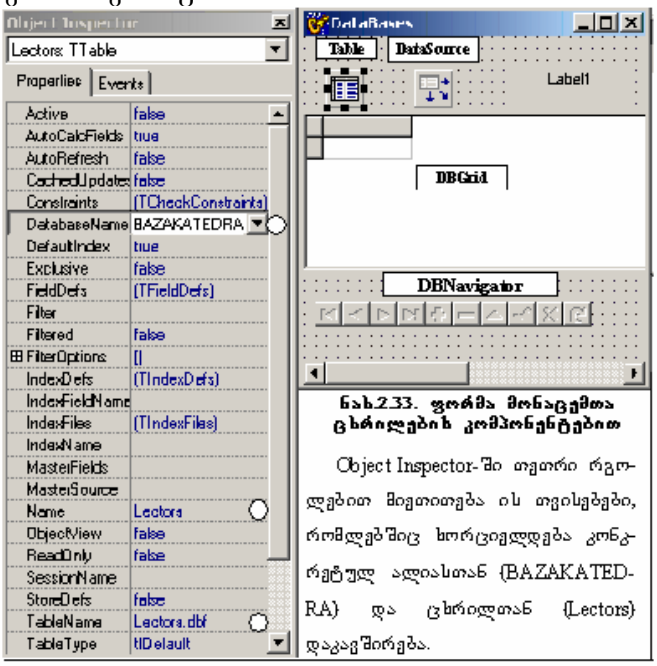
ცხრილში გასაღებური ატრიბუტი LEC_GRშედგენილია ორი ველისაგან LEC_ID და JG_ID.

ამგვარად, განხილულ ცხრილებს შორის კავშირები დამყარებულია პირველადი და მეორადი გასაღებური ატრიბუტების გამოყენებით. ამ მექანიზმის საფუძველზე მომდევნო პარაგრაფში განვიხილავთ ლოგიკური ბაზის მთლიანობის საკითხს და ცხრილებს შორის კასკადური კავშირების გამოყენებას. ახლა დავაკავშიროთ მონაცემთა ბაზა ფორმებს.

2.12. მონაცემთა ბაზის ცხრილების გამოტანა ფორმაზე

<u>დანიშნულება:</u> მონაცემთა ბაზის ფაილთა (ცხრილების) ასახვა Borland_C++ Builder სისტემაში. გარდა მონაცემთა მონიტორინგისა ფორმაზე შესაძლებელია ბაზის ცხრილების დამუშავება ვიზუალური და ობიექტ-ორიენტირებული მეთოდების კომპონენტებით.

კომპონენტების პანელებიდან მონაცემთა ბაზასთან მუშაობს Data Access (Table, DataSource კომპონენტები) და Data Controls (DBGrid, DBNavigator კომპონენტები). ნახ.2.33 ნაჩვენებია ასეთი ფორმა.

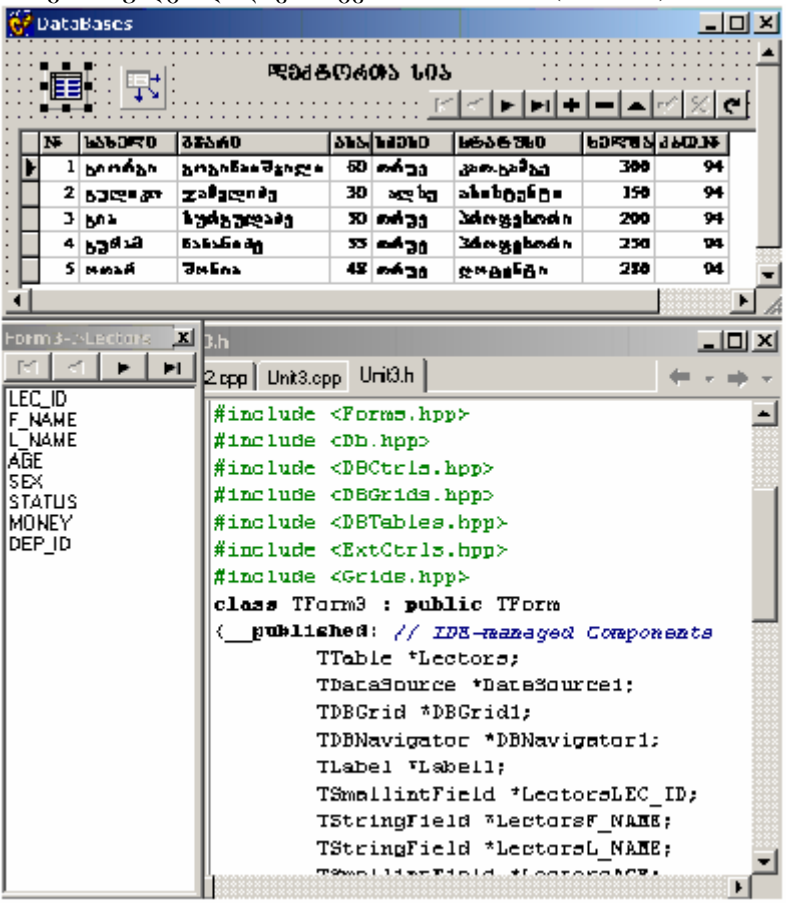


ნახ.2.33. ფორმა მონაცემთა ცხრილების კომპონენტებით

Object Inspector-ში თვითრი რგოლებით მითითება ის თვისებები, რომლებშიც ხორციელდება კონკრეტულ ალიასთან (BAZAKATEDRA) და ცხრილთან (Lectors) დაკავშირება.

ნახ.2.33

ბოლოს Object Inspector-ის Active თვისებაში უნდა ჩავსვათ true. ამის შემდეგ ფორმაზე DBGrid-ში გამოჩნდება მონაცემები (არა-ქართულად). ცხრილში მონაცემების ქართულად მისადებად საჭიროა დავდგეთ DBGrid ობიექტზე და Object Inspector-ის Font თვისებიდან ავირჩიოთ სათანადო შრიფტი. ცხრილის სვეტების გასაქართულებლად ვირჩევთ TitleFont-ს (ნახ.2.34).

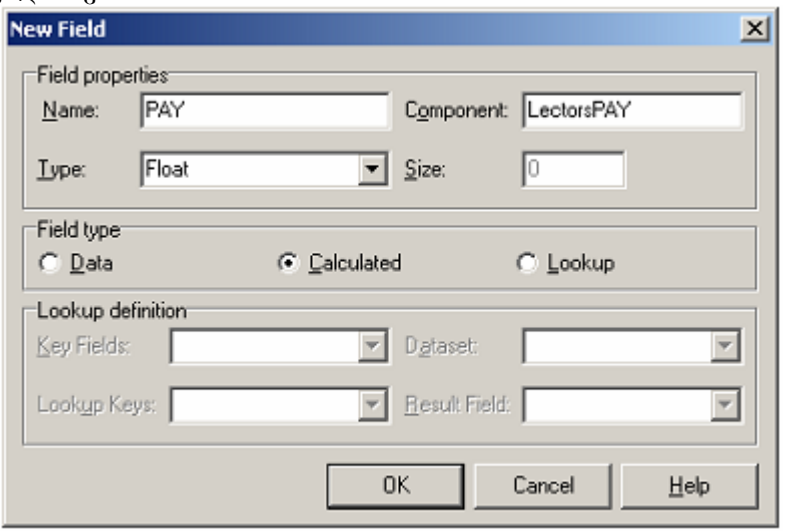


ნახ.2.34.

DBNavigator საჭიროა ცხრილებთან სამუშაოდ. მოძრაობა ჩანაწერებში, ახალი სტრიქონების ჩამატება (+), მოძველებულის წაშლა (-), კორექტირება (Δ), ბაზაში ჩაწერა (v) ან ცვლილებების გაუქმება (x - არ ჩაწერა).

2.34 ნახაზზე Table მონიშნულია, ე.ი. Object Inspector მას ეკუთვნის. 2-ჯერ დაწკაპუნებით Table-ზე გამოიტანება ველების რედაქტორი (Fields Editor), თაგუს მარჯვენა ღილაკით გამოვიტანთ დამხმარე ფანჯარას, რომლიდანაც შეიძლება DBGrid-ში გამოსატანი ატრიბუტების არჩევა, მათ შორის ახალი-გაანგარიშებადი ველებისაც (რომელიც ცხრილში არაა).

მაგალითად, შეიძლება ხელფასიდან საშემოსავლო დაქვითვის ანგარიშის ველის დამატება. ამისათვის Table-ს Field Editor-დან თაგუს მარჯვენა ღილაკით ვირჩევთ Add New Field (ნახ.2.35) და შევავსებთ Field Properties. Field type უნდა იყოს Calculated.



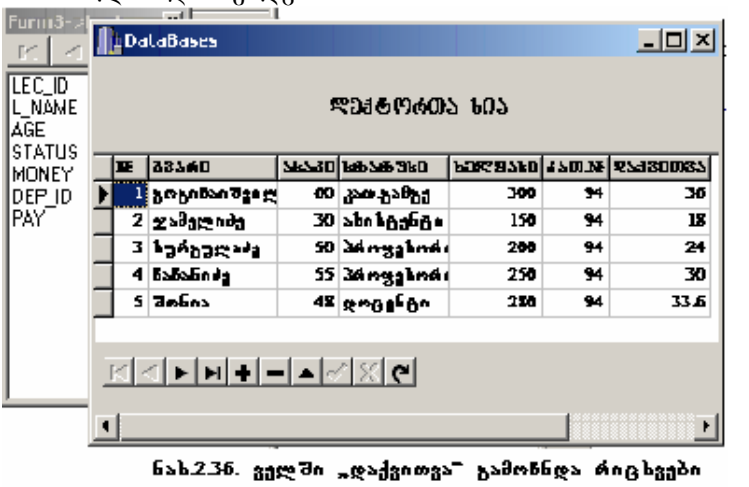
ნახ.2.35. გაანგარიშებადი გელის შექმნა

შედეგად ფორმაზე ცხრილში გამოჩნდება ახალი სვეტი (PAY), რომლის ქართულ მნიშვნელობას ჩავწერთ Object Inspector-ის Properties-ში DisplayLabel : “დაქვითვა“.

შემდეგ ვდგებით Table-ზე და Events-ში OnCalcFields-ში 2-ჯერ დაწკაპუნებით შევალთ პროგრამულ Unit3.cpp კოდში (ვინაიდან ეს Form3-ია) და ჩავწერთ გაანგარიშების ოპერატორს:

```
void __fastcall TForm3::LectorsCalcFields(TDataSet *DataSet)
 {
  // შესატანი
  LectorsPAY->Value= LectorsMONEY->Value*0.12;
 }
```

საბოლოოდ მივიღებთ 2.36 ნახაზს.



ნახ.2.36. ველში „დაქვითვა“ გამოჩნდა რიცხვები

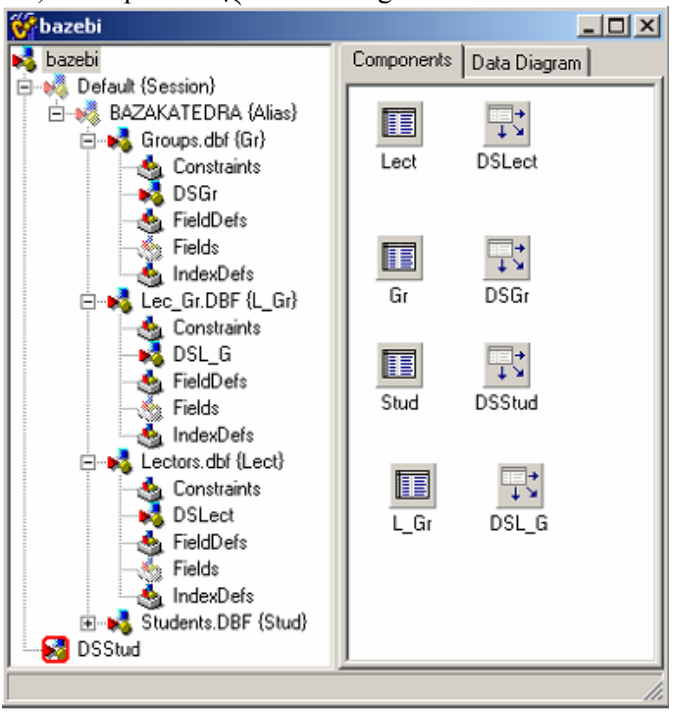
2.34 ნახაზის მარჯვენა ქვედა ნაწილში ჩანს Unit3.h – ფაილის ფრაგმენტი, რომელშიც ჩაწერილია კომპონენტთა კლასები. ტიპებად გამოყენებულია ბიბლიოთეკის შაბლონ-კლასები, ხოლო წარმოებული კლასები აღიწერება მაჩვენებლებით. მაგალითად., TTable *Lectors; TStringField *LectorsF_NAME და ა.შ.

2.13. მონაცემთა ბაზის დაპროექტების ვიზუალური კომპონენტი (TDataModule)

დანიშნულება: გამოიყენება რთული პროგრამული დანართების ასაგებად რამდენიმე მონაცემთა ბაზითა და ცხრილებით Borland_C++ Builder-ში მომხმარებელთა ინტერფეისისა და სისტემის მუშაობის ლოგიკის დიფერენცირებული დაპროექტებისათვის. Data Module Designer - ვიზუალური დაპროექტების ინსტრუმენტი გამოიძახება სისტემის მენიუდან:

File | New | Data Module.

ეკრანზე გამოჩნდება ფანჯარა ორი გვერდით (ნახ.2.37): Components და Data Diagram:

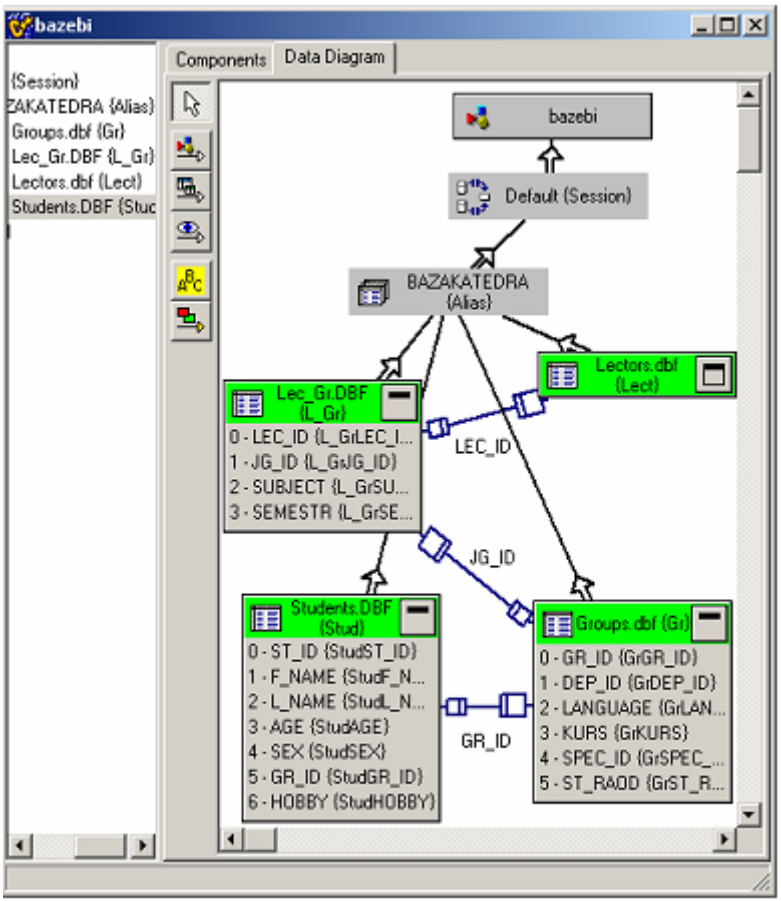


ნახ.2.37. **Data Module Designer**

თავიდან ეს ფანჯარა ცარიელია. Object Inspector-ის Properties-ში შევცვალოთ DataModule1 ჩვენთვის სასურველი სახელით, მაგ., bazebi. ამის შემდეგ Data Access-პანელიდან გადმოვიტანოთ ოთხი წყვილი Table და DataSource კომპონენტებისა (ჩვენ ოთხი ცხრილი გვაქვს მომზადებული). თითოეული Table-სთვის Object Inspector-ის Properties-ში შევცვალოთ მნიშვნელობები, მაგ., DataBaseName: BAZAKATEDRA, TableName: Lectors, Name: Lec. კომპონენტისთვის Data Source1 შევარჩიოთ სახელი, მაგ., Name: DSLect და DataSet: Lect. ასეთივე პროცედურები ჩავატაროთ ჯგუფის, სტუდენტის და ლექტორ-ჯგუფების ცხრილებისათვის. შედეგი არის ნაჩვენები 2.37 ნახაზზე. აქ მარცხენა ფანჯარაში სისტემის მიერ აგებული იერარქიული ხეა მომზადებული.

„ხეში“ პატარა კვადრატებში ჩანს „ - “ სიმბოლოები, მათზე თაგუს ღილაკის დაჭერით, ისინი ‘+“ -ად გადაკეთდება და დაქვემდებარებული სტრიქონები შეიკუმშება (აღარ გამოჩნდება).

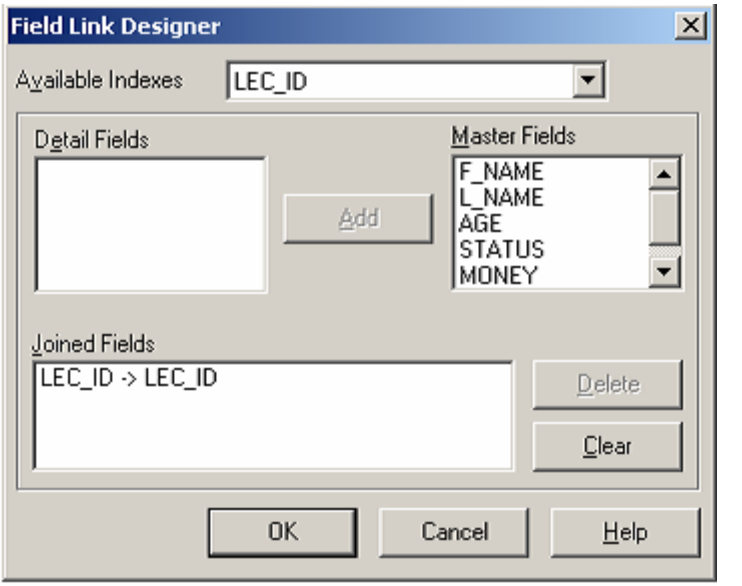
ახლა გადავიდეთ ძირითად დასკვნით ნაწილზე, ოთხ ცხრილს შორის კავშირების დამყარების ვიზუალური კომპონენტის გამოყენებაზე. გადავრთოთ გვერდი Data Diagram-ზე. ფანჯრის მარცხენა ნაწილიდან თაგუს დახმარებით გადმოვიტანოთ მარჯვენა ნაწილში „ხის“ სტრიქონები: bazebi, Dafault (Session), BAZAKATEDRA, Lectors.dbf, Lec_Gr.dbf, Groups.dbf, Students.dbf (ნახ.2.38).



ნახ.2.38 **Data Module Designer**

კავშირები (კონტურული ისრები) ავტომატურად გამოიტანება, იგი იერარქიულად დაქვემდებარებას გულისხმობს. მაგ, ჩვენი ოთხივე ცხრილი (dbf-failebi) არის BAZAKATEDR-ალიასის (ბაზის ფსევდონიმი სახელი) „შვილები“. ბაზები არის DataModule კომპონენტის სახელი („ბაბუა“), მას შეიძლება ჰყავდეს რამდენიმე „შვილი“ (BAZAKATEDR -ის „ძმები“). თითოეული მათგანი შექმნის ცალკე ცხრილთა შტოს - „ოჯახს“. Default Session–ის დანიშნულებაა ბაზის ალიასების მართვა, ანუ მოცემულ

მომენტში რომელი „ძმის“ შტო იქნება აქტიური. ცხრილებს შორის კავშირი ხორციელდება პირველადი და მეორადი გასაღებური ატრიბუტებით. ფანჯრის შუა ნაწილში მე-3 პიქტოგრამა ზემოდან არის Master-Detail კავშირის ასაგებად. მოვნიშნოთ იგი და თაგუთი გავატაროთ ხაზი Lect-ცხრილიდან L_Gr–კენ. გამოჩნდება ახალი ფანჯარა (ნახ.2.39), რომელშიც ავირჩევთ დამაკავშირებელ ველებს, მაგ., LEC_ID -> LEC_ID. პირველი Master Field, მეორე კი Available Indexes –დან ამოირჩევა Detail Fields-ში, ბოლოს OK.



ნახ.2.39. **Master - Detail** კავშირის აგება

გაჩნდება „დიდი-პატარა“ ზომის ორი მართკუთხედი შემაერთებელი კავშირით LEC_ID. ეს უკანასკნელი არის Lector.dbf ცხრილის პირველადი ინდექსი (გასაღები), იგი უნკალურია (Unique), ე.ი. ლექტორების ცხრილში ყოველი ლექტორი მხოლოდ ერთხელაა დაფიქსირებული.

მეორე („სწავლება“), Lec_Gr.dbf ცხრილში პირველადი ინდექსია ორი ატრიბუტი ერთად: Lec_Id + Jg_Id (ვინაიდან აქ რეალიზდება კავშირი M:N, „ერთი ლექტორი ასწავლის რამდენიმე ჯგუფს“ და „ერთ ჯგუფს ასწავლის რამდენიმე ლექტორი“). მეორად ინდექსად (გასაღებად) გამოიყენება Lec_Id ატრიბუტი (Field), ცალკე აღებული. ამგვარად, კავშირი დამყარდა აღნიშნული ორი ცხრილის ორ ველს შორის.

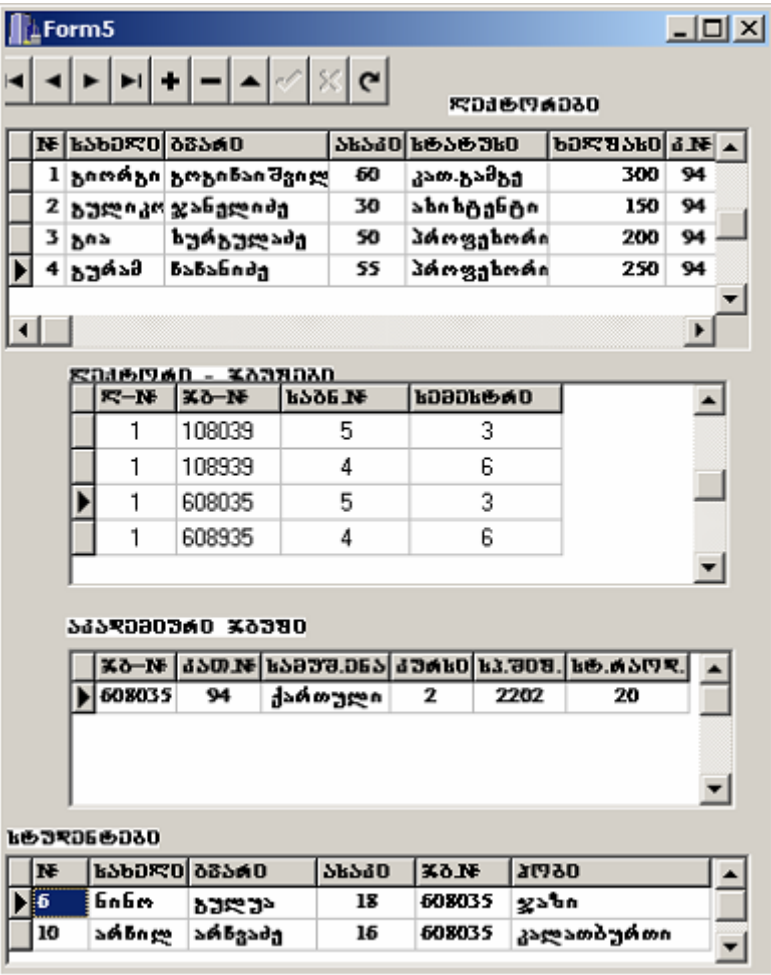
ასეთივე Master-Detail კავშირები ჩანს ნახაზზე „სწავლება“- „ჯგუფი“ და „ჯგუფი“ - „სტუდენტი“ ცხრილებს შორის.

ყოველივე ეს უზრუნველყოფს მონაცემთა ბაზაში ცხრილებს შორის სემანტიკური კავშირების რეალიზებას, რაც გამოიხატება იმაში, რომ თუ, მაგ., კურსორს დავაყენებთ ლექტორ-ცხრილში გვარზე „ჩაჩანიძე“, მაშინ ავტომატურად გამოჩნდება მხოლოდ მასთან დაკავშირებული „აკადემიური ჯგუფები“ და თითოეულ ჯგუფში შემავალი „სტუდენტები“.

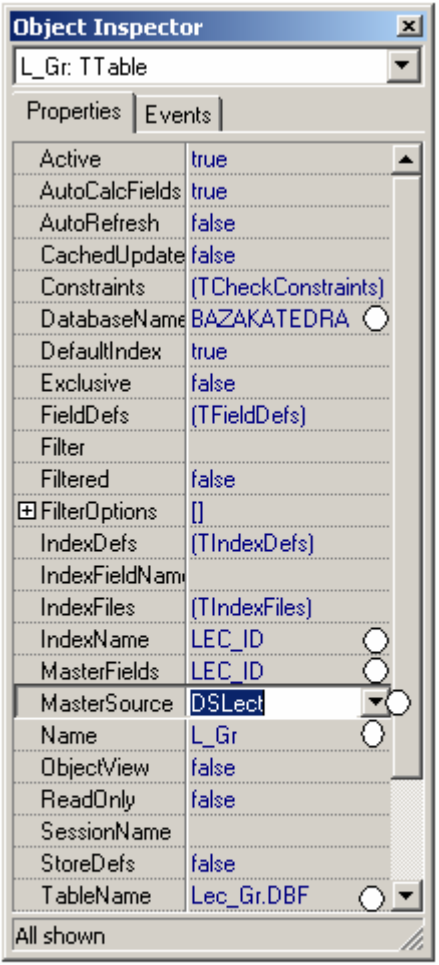
გადავალთ რა სხვა ლექტორზე, შეიცვლება ჯგუფები და სტუდენტები. 2.40 ნახაზზე ილუსტრირებულია ამ პროცესის შედეგები.

Master-Detail კავშირის აგება შესაძლებელია აგრეთვე Object Inspector–დან. მაგ., ლექტორისა (Lectors) და სწავლების (Lec_Gr) ცხრილებს შორის რომ დავამყაროთ ასეთი კავშირი, საჭიროა დავდგეთ Lec_Gr ცხრილზე. გამოჩნდება Object Inspector (ნახ.2.41).

ნახაზზე მცირე ზომის თეთრი რგოლებით ნაჩვენებია აუცილებელი თვისებების მნიშვნელობები (მაგ., MasterSource, IndexName, MasterFields).



ნახ.2.40. რელაციური კავშირების შედეგი



ნახ.2.41. **Master Detail** კავშირის აგება **Object Inspector**-ში

2.14. ცხრილთაშორისი კავშირების აგება ხილვადი ველების გამოყენებით (LookUp Fields)

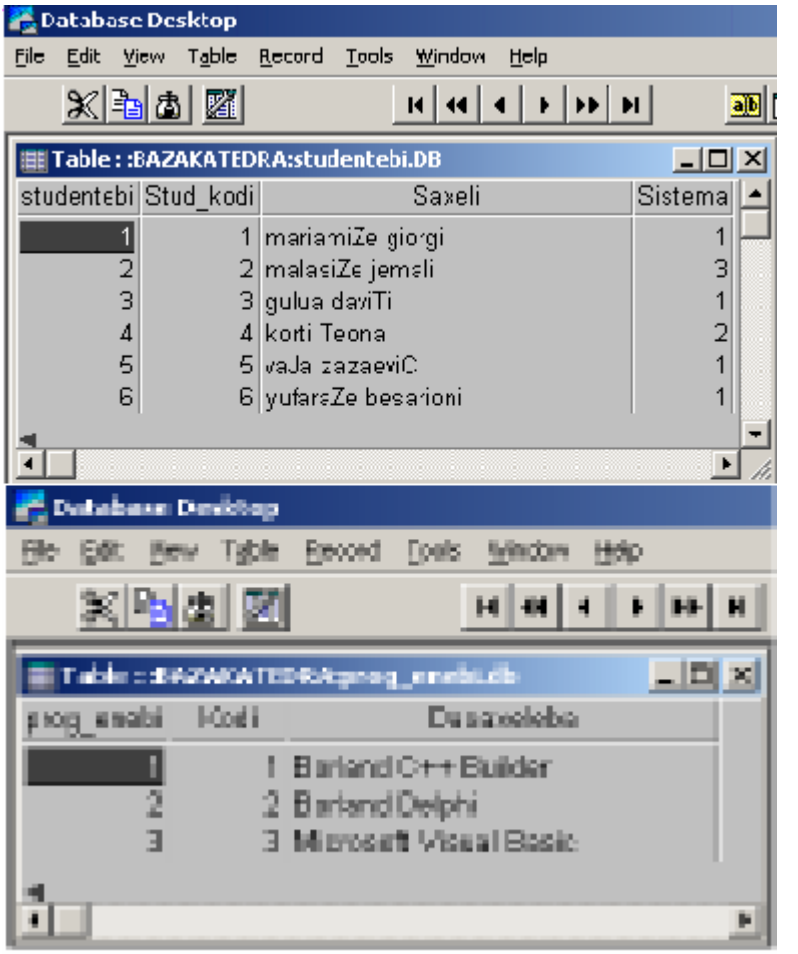
დანიშნულება: გამოიყენება ერთი ცხრილის ველის მნიშვნელობის შესარჩევად მეორე ცხრილიდან.

პირველ ცხრილში ხდება მონაცემთა კორექტირება ან შეტანა, ეს მონაცემები კი ინახება მეორე ცხრილში და იქიდან უნდა ამოირჩეს შესაბამისი მნიშვნელობა.

სამაგალითოდ განვიხილოთ ერთი მარტივი ამოცანა: დავუშვათ გვაქვს სტუდენტების მონაცემთა ბაზა, რომლებსაც სწავლების გარკვეულ ეტაპზე ეძლევათ არჩევანი, ისწავლონ რომელიმე დაპროგრამების სისტემა, მაგალითად **Borland C++ Builder, Delphi** ან **Visual Basic**. საჭიროა ორი ცხრილი - „სტუდენტები” და საცნობარო ტიპის “დაპროგრამების ენები”.

ცხრილთა ნიმუშები მოცემულია 2.42 ნახაზზე, სადაც ვხედავთ, რომ **studentebi** ფაილში გვაქვს ველი **systema**, რომელშიც უნდა ჩაიწეროს სტუდენტისთვის სასურველი დაპროგრამების სისტემის კოდი.

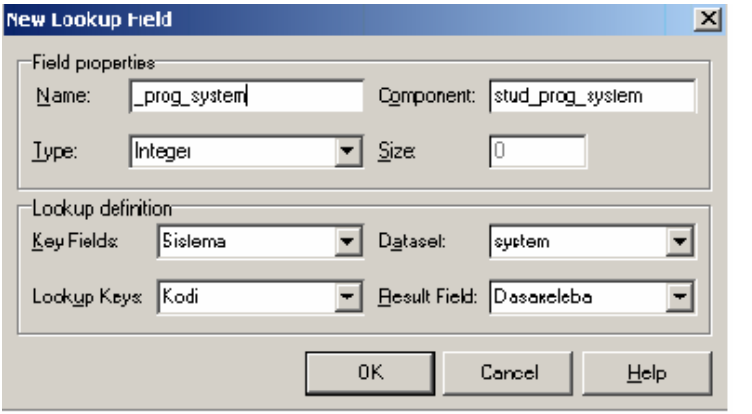
ცხადია, როცა ეს ამოცანა **Builder -** ის ფორმაზე გადაიტანება, მომხმარებლისთვის მოუხერხებელი იქნება უშუალოდ სტუდენტების ბაზაში კოდების ხელით შეყვანა. სასურველია, რომ მან დაპროგრამების სისტემა აირჩიოს სწორედ საცნობარო ცხრილიდან, რომელიც უნდა გავხადოთ მთავარ ცხრილზე დამოკიდებული.



ნახ. 2.42. ცხრილები "სტუდენტები" და "დაპროგრამების სისტემები"

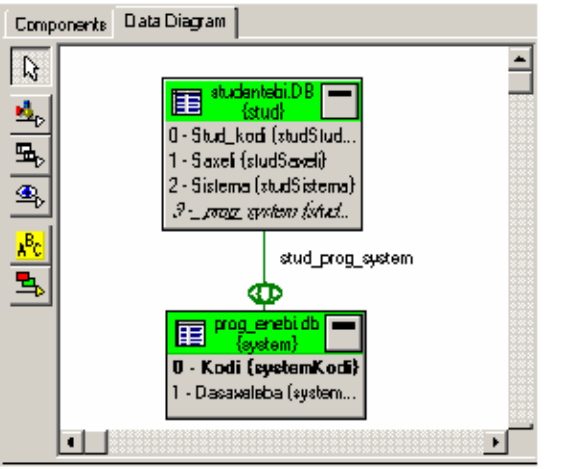
ამ მიზნით ცხრილების შესაბამის კომპონენტებს **(Table, DataSource)** მოვათავსებთ **Data Module**-ზე და გადავალთ **DataModule Designer**-ში, სადაც ავირჩევთ ღილაკს "ისარი თვალით" და გავავლებთ ცხრილიდან **studentebi** ცხრილისკენ **prog_enebi**. დაუყოვნებლივ გამოვა დიალოგი (ნახ. 2.43.), რომელშიც უნდა დავაყენოთ შემდეგი პარამეტრები: **Name** - სახელი, **Type** -

გასაღებური ველის ტიპი, **Key Fields** - მეორადი გასაღებური ველის სახელი მთავარი ცხრილიდან, **LookUp Keys** - პირველადი გასაღებური ველის სახელი საცნობარო ცხრილიდან, **Result Field** - საცნობარო ცხრილის ველი, საიდანაც მნიშვნელობები აიღება.



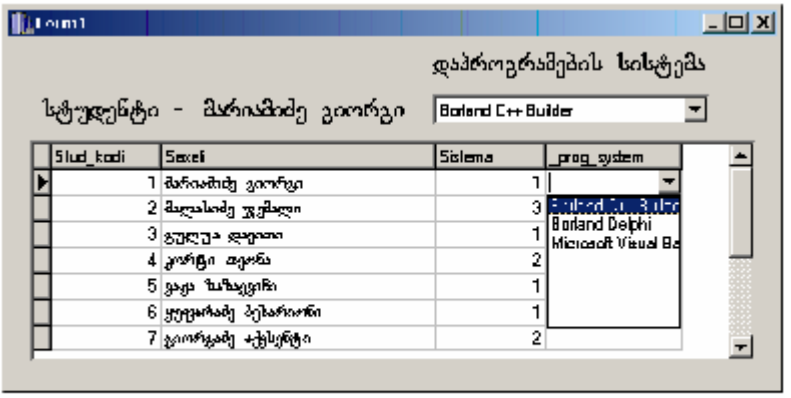
ნახ.2.43. **LookUp** ტიპის კავშირის დამყარება

Ok - ღილაკზე დაჭერის შემდეგ **DataModule Designer** - ის ფანჯარა 2.44 ნახაზზე ნაჩვენებ სახეს მიიღებს.



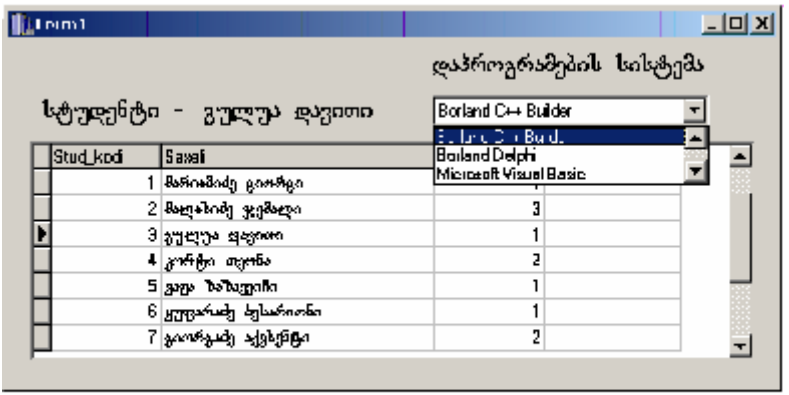
ნახ.2.44. **DataModule Designer** - ის ფანჯარა

ამის შემდეგ შესაძლებელია გადავიდეთ უშუალოდ ფორმაზე (ნახ.2.45-ა). **Data Grid** კომპონენტში გამოვიტანოთ მთავარი ცხრილი **studentebi.**



ნახ.2.45-ა. **Builder** - ის ფორმა. დაპროგრამების ენა აირჩევა **DataGrid** კომპონენტზე

როგორც ამჩნევთ, ველების სიას დაემატა კიდევ ერთი, **_prog_system,** რაც ზემოთ აღწერილი კავშირების დამყარების შედეგია. ამ ველის დახმარებით კომპონენტ **DataGrid** - შივე შეგვიძლია მიმდინარე სტუდენტს დაპროგრამების სისტემა შევუცვალოთ, ხოლო 2.45-ბ. ნახაზზე ნაჩვენებია შემთხვევა, როცა ვიყენებთ სპეციალურ კომპონენტს **DataControls** განყოფილებიდან, რომელსაც ჰქვია **DBLookUpComboBox**.

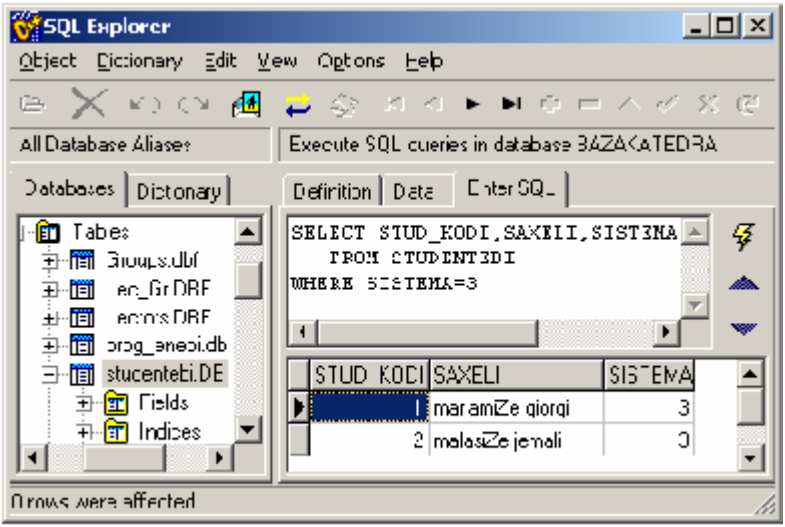


ნახ.2.45-ბ. Builder - ის ფორმა. დაპროგრამების ენა აირჩევა DBLookUpComboBox კომპონენტის საშუალებით

ამ კომპონენტის ასამუშავებლად უნდა დავაყენოთ მისი 5 თვისება. **DataSource** და **DataField** (მთავარი ცხრილის მეორადი გასაღები, რომლითაც ხდება დაკავშირება საცნობარო ბაზასთან) მივამაგროთ მთავარ ცხრილზე (**studentebi**), ხოლო **ListSource, key Field** (საცნობარო ცხრილის პირველადი გასაღები, კოდი) და **ListField** - საცნობაროზე.

ამ შემთხვევაშიც მომხმარებელს შეუძლია მისთვის მოხერხებულ ტექსტურ (და არა კოდების) რეჟიმში შეარჩიოს სასურველი დაპროგრამების სისტემა ყოველი სტუდენტისთვის.

2.46 ნახაზზე ნაჩვენებია მოცემულ ბაზასთან მიმართვის მოთხოვნის ფორმირების ფრაგმენტი SQL ენაზე, რომელსაც ჩვენ მომდევნო პარაგრაფში განვიხილავთ.



ნახ.2.46. მოთხოვნის ფორმირების ფრაგმენტი

2.15. საკონტროლო კითხვები და სავარჯიშოები

1. რას წარმოადგენს ვიზუალური კომპონენტი ?

2. რომელი ძირითადი ბლოკებისგან შედგება **BC++ Builder** სამუშაო გარემო ?

3. როგორაა კლასიფიცირებული ვიზუალური კომპონენტები პანელების მიხედვით?

4. როგორ ხორციელდება ვიზუალური კომპონენტების გამოყენება?

5. ააგეთ მთავარი მენიუ, რომელიც გამოიძახებს ორ ფორმას და აქვს დასასრულის ღილაკი.

6. დააპროექტე ფორმაზე ოთხი ღილაკი: ორი ნამდვილი რიცხვის მიმატების, გამოკლების, გამრავლებისა და გაყოფის, შედეგები აისახოს Label კომპონენტებში.

7. დაახასიათე დეტალურად Standard პანელის კომპონენტები.

8. დაახასიათე Data Access და Data Controls პანელების კომპონენტები მონაცემთა ცხრილებთან სამუშაოდ.

9. რა არის მონაცემთა ბაზის ფსევდონიმი (ალიასი), რისთვის გამოიყენება და როგორ იქმნება იგი ?

10. როგორ შევქმნათ მონაცემთა ბაზის ცხრილები (Tables) კლასებისათვის სტუდენტი, ლექტორი, ჯგუფი და ლექცია ?

11. ააგეთ მრავალგვერდიანი ფორმა ოთხი ფურცლით და გამოიტანეთ თითოეულზე სტუდენტების, ლექტორების, ლექციებისა და ჯგუფების ცხრილები, მათში მონაცემების შეტანისა და კორექტირების შესაძლებლობით.

12. რა არის სისტემური კომპონენტები (System) და როგორ ვიყენებთ მათ ? ააგეთ მარტივი მაგალითები.

13. ვიზუალური კომპონენტის (DataModule) გამოყენებით ააგე ბაზის სამი დაკავშირებული ცხრილი: ქვეყანა, გუნდი, ფეხბურთელი. შედეგები გამოიტანეთ ფორმაზე.

14. ააგეთ დეპარტამენტის თანამშრომელთა ხელფასის უწყისის ცხრილი სამი ძირითადი და ორი გაანგარიშებადი ველით.

15. დაიანგარიშეთ დეპარტამენტის ხელფასის უწყისში თანამშრომელთა ჯამური დარიცხული თანხა, ჯამური დაქვითვა და ჯამური ხელზე ასაღები თანხა.

გამოყენებული ლიტერატურა

1. ჩოგოვაძე გ., გოგიჩაიშვილი გ., სურგულაძე გ., შეროზია თ., შონია ო. მართვის ავტომატიზებული სისტემების დაპროექტება და აგება (თეორიულ-პრაქტიკული ინფორმატიკა). სტუ, თბილისი, 2001.

2. რეისიგი ვ., სურგულაძე გ., გულუა დ. ვიზუალური, ობიექტ-ორიენტირებული დაპროგრამების მეთოდები. სტუ. თბილისი, 2002.

3. Архангельский А. Программирование в Borland C++Builder. Москва, 2001.

4. სურგულაძე გ. დაპროგრამების ვიზუალური მეთოდები და ინსტრუმენტები: UML, Ms Visio, Borland C++Builder . სტუ. თბილისი, 2000.