საქართველოს ტექნიკური უნივერსიტეტი
1922 წლიდან
GEORGIAN TECHNICAL UNIVERSITY
SINCE 1922

გია სურგულაძე, სოფიკო პაპავაძე,
ოთარ მაჩალაძე

ინფორმაციული საზოგადოება და ინფორმატიკის დიდაქტიკა

„სტუ-ის IT-კონსალტინგის სამეცნიერო ცენტრი"

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, სოფიკო პაპავაძე,
ოთარ მაჩალაძე

ინფორმაციული საზოგადოება და ინფორმატიკის დიდაქტიკა

დამტკიცებულია:
სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრის“ სარედაქციო კოლეგიის მიერ - ოქმი N8 3.07.23

თბილისი - 2023

უაკ 004.5

განიხილება ინფორმაციული საზოგადოების ფორმირების პროცესში ინფორმატიკის დიდაქტიკის როლი. საჯარო სკოლებსა და უნივერსიტეტებში ინფორმატიკის საგანმანათლებლო პროგრამით აკადემიური კურსების სწავლების ძირითადი პრინციპები და მეთოდოლოგია. სასწავლო პროცესის მენეჯმენტის მხარდამჭერი საინფორმაციო სისტემის პროექტირებისა და პროგრამული აპლიკაციის დეველოპმენტის, CASE ტექნოლოგიისა და დაპროგრამების ავტომატიზაციის საკითხები. მონოგრაფიაში ასახულია ასეთი სისტემების ასაგებად პროგრამული ინჟინერიის და ხელოვნური ინტელექტის თანამედროვე მეთოდების გამოყენება, მათი ინსტრუმენტული საშუალებების შექმნა. წიგნი განკუთვნილია საგანმანათლებლო სფეროს ინფორმატიკის დიდაქტიკით დაინტერესებული მკითხველის, დოქტორანტების და სტუდენტებისთვის.

რეცენზენტები:

- პროფ. რომან სამხარაძე - სტუ, ტ.მ.დ.
- ასოც.პროფ. თინათინ დავითაშვილი - ივ. ჯავახიშვილის სახ. თსუ

რედკოლეგია:

ა. ფრანგიშვილი (თავმჯდომარე), ზ. აზმაიფარაშვილი, მ. ახობაძე, გ. გოგიჩაიშვილი, ზ. ბოსიკაშვილი, ე. თურქია, რ. კაკუბავა, ვ. კვარაცხელია, თ. ლომინაძე, ნ. ლომინაძე, ჰ. მელაძე, ლ. პეტრიაშვილი, გ. სურგულაძე. ბ. შანშიაშვილი, ო. შონია, ზ. წვერაიძე

ISBN 978-9941-8-5443-9

Georgian Technical University

Surguladze Gia, Papavadze Sopiko,
Otar Machaladze

INFORMATION SOCIETY AND DIDACTICS OF INFORMATICS

The role of didactics of informatics in the process of formation of the information society is considered. Basic principles and methodology of teaching academic courses in public schools and universities with an informatics educational program; Issues of designing and software application development of the information system supporting the management of the educational process. The monograph describes the use of modern methods of software engineering and artificial intelligence for the construction of such systems, as well as the creation of their tools. The book is intended for readers, doctoral students and students interested in didactics of informatics in the educational field.

ISBN 978-9941-8-5443-9

ეძღვნება:

საქართველოს ტექნიკური უნივერსიტეტის
„ინფორმაციული საზოგადოების“ კათედრის დაარსების მე-20 წლისთავს (2003-2023) და მისი დამაარსებლის, აკადემიკოს ***გოჩა ჩოგოვაძის*** ნათელ ხსოვნას (1941-2022)

DEDICATED:

to the 20th Anniversary of the establishment *"Information Society"* department of Georgian Technical University (2003-2023) and the bright memory of its founder, academician ***Gocha Chogovadze*** (1941-2022)

საქართველოს ტექნიკური უნივერსიტეტის რეფორმატორი რექტორი (1988-1994), იუნესკოს (პარიზი) კულტურისა და განათლების დეპარტამენტის დირექტორი (1981-1988). დიდი საზოგადო და სახელმწიფო მოღვაწე,

საქართველოს იუნესკოს საქმეთა ეროვნული კომისიის ვიცე-პრეზიდენტი (2004-2012), ევროპის და რუსეთის მრავალი უნივერსიტეტის საპატიო დოქტორი და საერთაშორისო აკადემიების წევრი, დიპლომატი, საქართველოს საგანგებო და სრულუფლებიანი ელჩი საფრანგეთსა და ესპანეთის სამეფოში (1994-2004), სტუ-ს „მართვის ავტომატიზებული სისტემების“ (1971) და სტუ-ს UNESCO-ს „ინფორმაციული საზოგადოების“ (2003) კათედრების დამაარსებელი, მათი პირველი გამგე.

საქართველოს მეცნიერებათა ეროვნული აკადემიის ნამდვილი წევრი (1994-დან), ტექნიკის მეცნიერებათა დოქტორი (1975) - საპატიო პროფესორი – *გოჩა გიორგის ძე ჩოგოვაძე* – მრავალი სამეცნიერო წიგნის, პროექტისა და სტატიის ავტორი, მრავალი სტუდენტის აღმზრდელი და ახალგაზრდა მეცნიერის „სამეცნიერო ნათლია“, ქართული და უცხოური ლიტერატურის, პოეზიის, მუსიკისა და სიმღერის დიდი მოყვარული, საერთაშორისო მეგობრობის დიდოსტატი, სუფრის ორიგინალური თამადა, მოსიყვარულე და გულისხმიერი მეუღლე, მამა და ბაბუა, მუდამ სიკეთის მთესველი, განუმეორებელი *„ქართველი ფაუსტი"* .

შინაარსი

ავტორების შესახებ:

გია სურგულაძე – ტექნიკის მეცნიერებათა დოქტორი, გაეროსთან არსებული „ინფორმატიზაციის საერთაშორისო აკადემიის“ ნამდვილი წევრი (1994 წლიდან, IIA), საქ. ტექნიკური უნივერსიტეტის პროფესორი. ინფორმატიკის ფაკულტეტის „პროგრამული ინჟინერიის“ აკადემიური დეპარტამენტის უფროსი. 100 წიგნის ავტორი (მათ შორის 25 მონოგრაფია, 20 სახელმძღვანელო, 55 დამხმარე და მეთოდური სახელმძღვანელო), 80 ელ-სახელმძღვანელო მოთავსებულია სტუ-ს ვებ გვერდზე. 250-ზე მეტი სამეცნიერო ნაშრომის ავტორი მართვის საინფორმაციო სისტემების პროგრამული ინჟინერიის, მონაცემთა რელაციური და NoSQL ბაზების, დაპროგრამების ჰიბრიდული ტექნოლოგიების, იმიტაციური მოდელირების (პეტრის ფერადი ქსელებით) და სხვ. სფეროში. *ჭვლილი:* წინამდებარე ნაშრომში შეიმუშავა „ინფორმაციული საზოგადოების“ სამეცნიერო მიმართულებაში ინფორმატიკის დიდაქტიკის როლი ახალი ციფრული ტექნოლოგიების გამოყენების საფუძველზე, საინფორმაციო სისტემის ბიზნეს-პროცესების უნიფიცირებული მოდელები, სისტემის ინფრასტრუქტურა სუფთა არქიტექტურის ბაზაზე და მომხმარებელთა ინტერფეისები.

სოფიკო პაპავააძე – აკადემიური დოქტორი ინფორმატიკაში. ივ. ჯავახიშვილის თსუ-ის ბაკალავრი და მაგისტრი, სტუ-ს „ინფორმატიკის“ პროგრამის დოქტორანტი (2019-2022) /მეცნ. ხელმძღ, გ. სურგულაძე). 1 მონოგრაფიის, 5 სამეცნიერო სტატიის ავტორი. 2 საერთაშორისო კონფერენციის მონაწილე-მომხსენებელი, პროგრამისტ-დეველოპერი. *ჭვლილი:* საჯარო სკოლების მიმდინარე საგანმანათლებლო პროცესების შესწავლა, მათი ტექსტური აღწერა, ფორმალიზაცია და ობიექტ-ორიენტირებული ანალიზი. სისტემის მომხმარებელთა ინტერფეისების და მათი ფუნქციონირების სცენარების აგება, მონაცემთა ბაზის და პროგრამული სერვისების გამართვა, მანქანური დასწავლის ალგორითმების პროგრამირება.

ოთარ მაჩალაძე – სტუ-ის „ინფორმატიკის“ პროგრამის დოქტორანტი, თსუ-ს „კომპიუტერული მეცნირების“ მაგისტრი, პროგრამული პროექტების დეველოპერი (სისტემის არქიტექტორი). 1 მონოგრაფიის და 2-სამეცნიერო სტატიის თანაავტორი. საერთაშორისო კონფერენციების მონაწილე-მომხსენებელი. უნარები და ინტერესები: საინფორმაციო სისტემების ბიზნეს-პროცესების ანალიზი და პროექტირება, ITIL-მეთოდოლოგია, Jira Service Management, დაპროგრამების ავტომატიზაცია.

შესავალი

XX საუკუნის ბოლო წლებში მნიშვნელოვნად იმატა ახალი ციფრული ტექნოლოგიების განვითარების ინტენსივობის დონემ, კერძოდ, ინფორმაციული და კომუნიკაციური ტექნოლოგიების (ICT) სახით; მათი გამოყენების სწრაფად მზარდმა დინამიკამ ბიზნესის, ეკონომიკის, განათლების და სხვ. სფეროებში. 1998 წელს UNESCO-ს ინიციატივით შეიქმნა ამ სფეროში საერთაშორისო მონიტორინგის მექანიზმი – ***„ინფორმაციული საზოგადოების“*** ობსერვატორია [1]. იგი ახორციელებს ახალი ეთიკური, სამართლებრივი და სოციალური პრობლემების კვლევას რამდენიმე მიმართულებით: ინფორმაციის წვდომა საზოგადოებრივ სფეროებში; ელექტრონული კომერცია; პირადი ინფორმაცია და კონფიდენციალობა კიბერსივრცეში; ძალადობა კიბერსივრცეში.

დღეს ეს ფუნქციები მნიშვნელოვნადაა გაფართოებული, შექმნილია იუნესკოს ცენტრების ერთობლივი პლატფორმა *ინფორმაციული საზოგადოების კვლევის* საკითხებზე, კერძოდ, განათლების, თავისუფალი წვდომის პროგრამული უზრუნველყოფის, მრავალენოვანი კიბერსივრცის, პროექტების მონაცემთა ბაზების და საიტების სახით და ა.შ.

ასეთ გამოწვევათა მნიშვნელობა იმდენად საყურადღებო იყო, რომ 2006 წელს გაერთიანებული ერების გენერალურმა ასამბლეამ მიიღო რეზოლუცია, რომლითაც ***17 მაისი გამოცხადდა „ინფორმაციული საზოგადოების“ საერთაშორისო დღედ*** [2].

ჟენევაში (შვეიცარია) ყოველწლიურად იმართება სამიტის *WSIS* ფორუმები (*World Summit on the Information Society*), სადაც განიხილება ინფორმაციული საზოგადოების მდგრადი განვითარების აქტუალური საკითხები ინფორმაციულ ტექნოლოგიებსა და კომუნიკაციებში (ITC) [3]. განსაკუთრებით მნიშვნელოვანი იყო ასეთი სამიტების დისტანციური ჩატარება კოვიდ-19 პანდემიის

პერიოდში (2020-2022 წწ.). 2023 წლის ფორუმი „Information and Knowledge Societies“ გაიმართა 13-17 მარტს, ხოლო ვირტუალური ვორკშოპები – აპრილ-მაისის ჩათვლით [3]. ფორუმის მთავარი ამოცანაა ამ მიმართულებით სტრატეგიული განვითარების გეგმის შემუშავება 2030 წლამდე.

ამგვარად, ჩვენი წიგნიდან გვინდა მივმართოთ ინფორმატიკოსებს და არაინფორმატიკოსებს – *„მოსწავლეებო და სტუდენტებო, პროფესორებო და მასწავლებლებო, ყველა დარგის მეცნიერებო და პრაქტიკოსებო, სახელმწიფო და კერძო სტრუქტურების თანამშრომლებო, ექიმებო, მსახიობებო და სპორტსმენებო, ძვირფასო პენსიონერებო და დროებით უმუშევრებო – ყველას, ვისაც გიყვართ კომპიუტრთან ჯდომა და ინტერნეტში მოგზაურობა, არ გრძნობთ თავს იზოლაციაში, თქვენ წინ გადაშლილია დედამიწის ბუნებრივი სილამაზე არქტიკიდან-ანტარქტიდამდე, გალაქტიკების სისტემა, ზღვებისა და ოკეანეების წყალქვეშა სამყარო..., მოსაწყენად არავის სცალია ...,, განსაკუთრებით ძალიან პატარებს, რომლებმაც ჯერ ლაპარაკიც არ იციან და უკვე „კარგად ერკვევიან“ კომპიუტერისა და მობილურის თამაშებში ... და* ***ყველანი ერთად – თქვენ ქმნით „ინფორმაციულ საზოგადოებას“!***“

ჩვენი ქვეყნის სოციალურ-ეკონომიკური მდგრადი განვითარების ერთ-ერთი მნიშვნელოვანი მიმართულება ***განათლებაა,*** რომელიც ქვეყნის პროგრესული წინსვლისთვის გამორჩეულად პრიორიტეტული, გრძელვადიან შედეგზე გათვლილი სფეროა. საქართველოს წარმატება მნიშვნელოვნადაა დამოკიდებული განათლების სფეროს ეფექტიანობაზე. სწორედ მან უნდა უზრუნველყოს, ერთი მხრივ, დემოკრატიული ღირებულებების მქონე მოქალაქეთა აღზრდა, მეორე მხრივ კი – მოამზადოს შრომის ბაზრის მოთხოვნების შესაბამისი კადრები საზოგადოების განათლების უზრუნველსაყოფად.

დღეს მსოფიოში განათლებისა და მეცნიერების სფეროს განვითარება მაღალი ტემპით მიმდინარეობს. მის ფონზე საქართველოსთვისაც მეტად მნიშვნელოვანი და აქტუალურია აღნიშნულ სფეროთა სწრაფი პროგრესული წინსვლა.

განათლების სისტემის განვითარების ძირითადი მიმართულებები მიზნად ისახავს სწავლებისა და კვლევის ხარისხის, რელევანტურობისა და ხელმისაწვდომობის სრულყოფას გრძელვადიანი ეკონომიკური ზრდისა და სოციალური სტაბილურობის ხელშესაწყობად [4, 5]. ამიტომაც, მეტად უნდა ვიზრუნოთ მოსწავლეთა განათლების ხარისხის გაუმჯობესებაზე, მენეჯმენტის შესაბამისი პროცესების ავტომატიზაცია/გაციფრულებაზე და გამჭირვალობაზე.

მონოგრაფიის ფარგლებში შემუშავებული პროექტის შედეგები საშუალებას მოგვცემს ჩამოვაყალიბოთ როგორც მოსწავლეებსა და მშობლებზე, ისე პედაგოგ-თანამშრომლებზე მორგებული მარტივი და მოქნილი მართვის სისტემა. მისი დახმარებით მოსწავლეებისა და მშობელთათვის მოსახერხებელი და გამარტივებული იქნება სხვადასხვა მიმდინარე პროცესში მეტად ჩართულობა. წიგნში ასახულია მისი თანაავტორების (სტუ-ს დოქტორანტი ს. პაპავაძე და მეცნიერ-ხელმძღვანელი, პროფ. გ. სურგულაძე) მიერ საპროექტო კვლევის პერიოდში მიღებული შედეგები [6, 7].

კვლევის ფარგლებში განხორციელდა სასწავლო დაწესებულებებში არსებული (ადრე დანერგილი) სისტემების შესწავლა, პრობლემების აღმოჩენა და შემდეგ მათი გადაჭრის გზების მოძიება. ყურადღება მახვილდებოდა ისეთ პროცესებზე, რომლებიც პედაგოგებს გაუმარტივებდა სასწავლო პროცესის მართვას, აგრეთვე ხელს შეუწყობდა მოსწავლეთა განვითარებასა და წინსვლას.

ნაშრომის აქტუალურობას განაპირობებს თანამედროვე მსოფლიოში მიმდინარე განათლების პროცესების კრიტიკული

გადახედვა და მათი სრულყოფისა და გამარტივების მცდელობები. თუნდაც, ხელოვნური ინტელექტის (AI) მეთოდების დანერგვა საგანმანათლებლო სისტემებში, რაც ერთ-ერთი ურთულესი, მაგრამ ეფექტიანი გამოწვევაა.

რეალურად, AI-ის დანერგითა და სწორად გამოყენებით შესაძლებელია არეერთი პროცესის გამარტივება, დროის ნაკლები რესურსის დახარჯვა იმ საკითხებზე, რომელთა პროგნოზირება შესაძლებელია მანქანური დასწავლის (ML-ის) მეთოდებისა და ალგორითმების დახმარებით [7, 8].

ჩვენი ნაშრომის მიზანია საგანმანათლებლო დაწესებულებისთვის (მაგალითად, სკოლა, უნივერსიტეტი) ისეთი ავტომატიზებული მენეჯმენტის სისტემის შექმნა, რომელიც ხელს შეუწყობს შესაბამის ორგანიზაციაში მიმდინარე პროცესების მართვის გამარტივებას და სრულყოფას.

პირველ თავში გადმოცემულია ინფორმაციული საზოგადოების ფორმირების პროცესის არსი, ინფორმატიკის დიდაქტიკის როლი ამ პროცესში. საგანმანათლებლო დაწესებულებებში ახალი ინფორმაციული ტექნოლოგიებისა და სისტემების მნიშვნელობა, სკოლისა და უნივერსიტეტის მრავალფეროვანი პროცესების განხილვა, როგორც რთული პრობლემური კვლევის სფეროსი. ამ პროცესებიდან ჩვენ გამოვყოფთ და განვიხილავთ რამდენიმე მნიშვნელოვანს, მაგალითად, (დანარჩენებიც შესაძლებელია მსგავსად გადაწყდეს და სისტემურად ინტეგრირდეს მათთან მომავალში). ასეთებია: მოსწავლეთა მეცადინეობებსა და სხვა ღონისძიებებზე დასწრების კონტროლის პროცესი; მათი შეფასებების ანალიზის, შედეგების დამუშავების პროცესი, მოსწავლეთა მიღწევების ანალიზისა და პროგნოზის პროცესები და ა.შ.

წიგნის *მეორე თავში* განხილულია საგანმანათლებლო დაწესებულების ძირითადი და დამხმარე სასწავლო პროცესების სისტემური ანალიზისა და უნიფიცირებული მოდელირების

საკითხები. კერძოდ, წარმოდგენილია შემდეგი მნიშვნელოვანი ამოცანების გადაწყვეტა:

- საგანმანათლებლო დაწესებულების ზემოაღნიშნული პროცესების შესახებ ინფორმაციის წარმოდგენის ეფექტური მოდელის ფორმირება;
- საგანმანათლებლო სისტემებში სამუშაო პროცესების მართვის სისტემის ტექნიკური მახასიათებლების განსაზღვრა და მოდელირება;
- სამუშაო პროცესის მენეჯმენტის სისტემის პრაქტიკული რეალიზებისთვის დიზაინის შემუშავება;
- სისტემის ფუნქციონირების ტექნოლოგიის შემუშავება შესაბამისი მეთოდებით და ინსტრუმენტული საშუალებებით;
- თვითდასწავლადი სისტემის აგება;

მეორე და მესამე თავებში დასმული პრობლემების გადასაწყვეტად ნაშრომში გამოყენებულია საგანმანათლებლო სისტემებში მიმდინარე კონკრეტული პროცესების ობიექტ- და პროცეს-ორიენტირებული ანალიზის და პროექტირების მეთოდები, აგრეთვე უნიფიცირებული მოდელირების ენა (UML) [9,10], მონაცემთა რელაციური და NoSQL ბაზების თეორია [11-13], მიკროსერვისებზე ორიენტირებული არქიტექტურა დომენზე ორიენტირებული დიზაინის (DDD) გამოყენებით [14-17], ხელოვნური ინტელექტის და მანქანური დასწავლის მეთოდები [18,19], ტექსტების დადარების ალგორითმები და ა.შ. [20].

მეოთხე თავი მთლიანად დათმობილი აქვს მანქანური დასწავლის მეთოდების და ალგორითმების დანერგვის საკითხებს სასწავლო პროცესში,

მონოგრაფიის ორიგინალობა მდგომარეობს ისეთი ინოვაციური საკითხების გადაწყვეტაში, როგორიცაა სუფთა არქიტექტურის მოდელის და პრობლემებზე (დომენზე) ორიენტირებული

დიზაინის (DDD) მორგება საგანმანათლებლო სისტემებში მიმდინარე პროცესების ბიზნეს რეალობაზე; ხელოვნური ინტელექტის მეთოდების დანერგვა საქართველოს საგანმანათლებლო ობიექტებზე (მანქანური დასწავლის ალგორითმების საფუძველზე; სასწავლო პროცესების მენეჯმენტის გამარტივება და ახალი შესაძლებლობების გაჩენა); ტექსტების დადარების ახალი ალგორითმის შექმნა, რომელიც რიგ შემთხვევებში იძლევა მეტად ოპტიმალურ შედეგს, ვიდრე არსებული, ხშირად გამოყენებადი ალგორითმები, როგორიცაა მაგალითად, soundex, jaro similarity, jaro_winkler [20].

ნაშრომში მიღებულ შედეგებს აქვს (ჩვენი ექსპერიმენტული კვლევებით) მაღალი პარქტიკული მნიშვნელობა და მომავალშიც ექნება მას საქართველოში მოქმედი საგანმანათლებლო დაწესებულებების და განათლების სისტემაში ჩართული ორგანიზაციებისათვის განსაკუთრებული როლი.

შექმნილია მოქნილი, სწრაფი და თვითდასწავლადი სისტემა, რომელიც მოახდენს საგანმანათლებლო ობიექტებზე მიმდინარე პროცესების ავტომატიზებულ მართვას, მათ სრულყოფას გამარტივების საფუძველზე, ხელოვნური ინტელექტის მეთოდების და სუფთა არქიტექტურის გამოყენებით, უმოკლეს დროში მიაწვდის მომხმარებელს რელევანტურ ინფორმაციას გადაწყვეტილების მისაღებად.

წიგნის ავტორები მადლობას უხდიან წინასწარ გულისხმიერ მკითხველს და ყურადღებით მიიღებენ მათგან შენიშვნებს და რეკომენდაციებს მომავალი სამუშაოების შემდგომი განვითარების და სრულყოფის მიზნით:

g.surguladze@gtu.ge

sopopapavadze@yahoo.com

თავი 1
ინფორმაციული საზოგადოების ფორმირების კონცეფცია და მისი მნიშვნელობა

ტერმინი – „ინფორმაციული საზოგადოება“ ახალი არაა, იგი თავის ისტორიულ წინაპრად „ინდუსტრიულ საზოგადოებას“ მიიჩნევს (ამ სფეროს მეცნიერთა შეხედულება მრავალი ლიტერატურული წყაროდან და ინტერნეტიდან) [22-24].

მისი განსაზღვებაც არაა ცალსახა.

მაგალითად, შედარებით ზოგადი განსაზღვრება ასეთია: „ინფორმაციული საზოგადოება არის საზოგადოება, სადაც ინფორმაციის გამოყენება, შექმნა, გავრცელება, მანიპულირება და ინტეგრირება მნიშვნელოვანი საქმიანობაა. მისი მთავარი მამოძრავებელი ძალაა საინფორმაციო და საკომუნიკაციო ტექნოლოგიები (ICT – Information and Communication Technology), რამაც გამოიწვია ინფორმაციის სხვადასხვა ფორმის სწრაფი ზრდა…”.

ამ სფეროში მოღვაწე მეცნიერთა უმრავლესობას მიაჩნია, რომ „ინდუსტრიულიდან“ - ტრანსფორმაციის პერიოდი „ინფორმაციულზე“ გასული საუკუნის 70-იან წლებში დაიწყო. ხოლო XX-XXI საუკუნეთა მიჯნა ამ სფეროს მნიშვნელოვანი გარდატეხისა და სწრაფი განვითარების პერიოდი აღმოჩნდა. ინფორმაციული ტექნოლოგიების და ინტერნეტის გამოყენების პრინციპებმა გავლენა იქონიეს სხვა სფეროებზე, მათ შორის მთავრობაზე, განათლებაზე, ეკონომიკასა და ჯანმრთელობაზე (გავიხსენოთ „კოვიდ-19“ პანდემიის პერიოდიც...). თითქმის მთელი მსოფლიოს ეკონომიკა (ზოგადად) გადასული იყო დისტანციურ ფუნქციონირებაზე... მოკლედ, რომ ვთქვათ, ამ საჭირბოროტო პანდემიამ, თავის მხრივ, ხელი შეუწყო ადამიანთა ფართო მასების თავმოყრას ინტერნეტული ქსელისა და კომპიუტერული საზოგადოების გლობალურ ფორუმში.

ჩვენი შეხედულებითაც, თუ ვინაა „ინფორმაციული საზოგადოების“ წევრი... ასე მიგვაჩნია [23]: „ინფორმატიკოსებო და არაინფორმატიკოსებო – *„მოსწავლეებო და სტუდენტებო, პროფესორებო და მასწავლებლებო, ყველა დარგის მეცნიერებო და პრაქტიკოსებო, სახელმწიფო და კერძო სტრუქტურების თანამშრომლებო, ექიმებო, მსახიობებო და სპორტსმენებო, ძვირფასო პენსიონერებო და დროებით უმუშევრებო – ყველას, ვისაც გიყვართ კომპიუტრთან ჯდომა და ინტერნეტში მოგზაურობა, არ გრძნობთ თავს იზოლაციაში, თქვენ წინ გადაშლილია დედამიწის ბუნებრივი სილამაზე არქტიკიდან-ანტარქტიდამდე, გალაქტიკების სისტემა, ზღვებისა და ოკეანეების წყალქვეშა სამყარო..., მოსაწყენად არავის სცალია ...,, განსაკუთრებით ძალიან პატარებს, რომლებმაც ჯერ ლაპარაკიც არ იციან და უკვე „კარგად ერკვევიან“ კომპიუტერისა და მობილურის თამაშებში ... და **ყველანი ერთად – თქვენ ქმნით „ინფორმაციულ საზოგადოებას“** !“*

არ ვიცით, რამდენად სწორია ჩვენი ხედვა ფილოსოფიურად ასეთ საკითხებზე, მაგრამ მეტაფორულ პერეფრაზირებას თუ გამოვიყენებთ, „რანი ვიყავით და სად მივედით ?“ - შეიძლება ვთქვათ, რომ ადრე „ადამიანი შედიოდა სამყაროში“, ახლა კი – „სამყარო შედის ადამიანში (მის გონებაში)“ - და ეს ყველაფერი ახალი ციფრული ტექნოლოგიების, მწვანე კომპიუტინგის, ეკოსისტემების და .ა.შ. გლობალური ინტერნეტული ობობას ბრალია,,, ადამიანებს აინტერესებთ, აწყობთ ეს და სულ უფრო ეფლობიან მის წიაღში... ვირტუალური რეალობა თავისი ჯადოსნური სათვალით, გვაძლევს უფლებას რაჭის სუფთა ჰაერზე, ხვანჭკარის ბაკლით ხელში, ვიგულავოთ კოპენჰაგენის ქუჩებში ან დავისვენოთ კანარის კუნძულებზე (ზედმეტი ხარჯის გარეშე).

მსოფლიო ინფორმაციული საზოგადოების ფორმირება ჯერ არ დასრულებულა, წინ მრავალი საინტერესო მოვლენაა !

1.1. ინფორმატიკის დიდაქტიკა, როგორც მეცნიერება: მიზანი და პრინციპები

დიდაქტიკა – მეცნიერებაა სწავლების პროცესის შესახებ [25-27]. ჩვენი მიზანია სასწავლო პროცესის შემდგომი განვითარება და სრულყოფა ICT (Information and Communication Technology) ბაზაზე.

აშშ-სა და ევროპაში 21-საუკუნის დასაწყისიდან კომპიუტინგის, ინფორმატიკის, საბუნებისმეტყველო და სოციალურ მეცნიერებათა ფაკულტეტებზე შეიქმნა ახალი აკადემიური დეპარტამენტები (კათედრები), რომელთა პროფესორ-პედაგოგები მეცნიერულად შეისწავლიან, იკვლევენ და ასწავლიან ახალ თაობას (სკოლა, კოლეჯი, უნივერსიტეტი) საინფორმაციო საზოგადოების ფორმირებისა და ინფორმატიკის დიდაქტიკის როლს ამ სფეროში.

ასეთ ინოვაციურ გამოწვევას დიდად შეუწყო ხელი „კოვიდ-19“ პანდემიის ფაქტორმა და საერთაშორისო ინფორმაციული საზოგადოების მსოფლიო სამიტმა (WSIS - ჟენევის მსოფლიო ფორუმი /ბოლო-2023/) [3].

კლასიკური და თანამედროვე განათლების თვალსაზრისით გამოიკვეთება შემდეგი დიდაქტიკური პრინციპები:

- მეცნიერული;
- სისტემურობის;
- სისტემატურობის;
- ვიზუალურობის (ხილვადობის);
- წვდომადობის;
- თეორიის კავშირის პრაქტიკასთან;
- სწავლა ცხოვრებასთან;
- შედეგების სიმძლავრის და ეფექტურობის;
- შემოქმედებითი აქტივობის;
- მოსწავლეთა/სტუდენტთა დამოუკიდებლობის მასწავლებლის წამყვანი როლით;

- ინტერდისციპილინურობის;
- მუშაობის კოლექტიური ფორმების და ა.შ.

მეთოდური სიახლეების თვალსაზრისით თუ განვიხილავთ პროცესებს, *ინოვაციური დიდაქტიკის მოდელის* მაგალითებია: „გადაბრუნებული კლასი“ [28], სწავლების გუნდური მეთოდი (ორგანიზაციული მიდგომა), მოსწავლე აქტიურია – თვითონ სწავლობს, მასწავლებელი ეხმარება მიგნებაში, აძლევს მიმართულებას და ა.შ. (სტატიკური მოდელები !);

არსებობს დინამიკური მოდელიც, იგი იყენებს დიდაქტიკურ ციკლს, რომელიც შედგება სწავლების პროცესის ურთიერთდაკავშირებული ეტაპებისგან – დროითი პარამეტრით.

დიდაქტიკური ციკლის ეტაპებია:

1) საერთო დიდაქტიკური მიზნის დასახვა (შემეცნებითი დავალება), სკოლის მოსწავლეებში დადებითი მოტივაციის შექმნა მისი გადაწყვეტისთვის, მოსწავლეების მიერ დავალების მიღება;

2) სასწავლო მასალის ახალი ნაწილის წარდგენა და მისი ცნობიერი აღქმისა და პირველადი ათვისების პირობების შექმნა;

3) მოსწავლეთა ორგანიზება და თვითორგანიზება სასწავლო მასალის გააზრებისა და შემდგომი ათვისების პროცესში ამ ციკლში საჭირო და შესაძლო დონეზე;

4) უკუკავშირის ორგანიზება, სასწავლო მასალის შინაარსის ათვისებაზე კონტროლი და თვითკონტროლი;

5) მოსწავლეთა მომზადება დამოუკიდებელი (კლასგარეშე) სამუშაოსთვის.

დიდაქტიკა ავითარებს: განათლების შინაარსის მოდელებს, რომლებიც იძლევა პასუხს კითხვაზე – განათლების შინაარსის შედგენლობისა და სტრუქტურის შესახებ, მისი შერჩევის ალგორითმზე, სასწავლო საგნის წარმოდგენის შესახებ, სხვადასხვა დიდაქტიკურ მიდგომებზე – მაგალითად, ინტეგრირებული

გაკვეთილის შესახებ და ა.შ. [26, 29], ანუ დიდაქტიკა ამუშავებს გაკვეთილის მოდელს.

საქართველოს ტექნიკური უნივერსიტეტის ინფორმატიკისა და მართვის სისტემების ფაკულტეტის „პროგრამული ინჟინერიის“ დეპარტამენტში (მართვის ავტომატიზებული სისტემების კათედრა 1972-2022) პროფ. გ. სურგულაძის და მისი გაზრდილი სტუდენტების (ბაკალავრები, მაგისტრანტები და დოქტორანტები) მიერ ბოლო 50 წლის მანძილზე შექმნილია *ინფორმატიკის დიდაქტიკის მეცნიერული მიმართულების* უნიკალური მეთოდური უზრუნველყოფის ლიტერატურული ბაზა (100 წიგნი: 25 მონოგრაფია, 20 სახელმძღვანელო, 55 დამხმარე და მეთოდური სახელმძღვანელო სტუდენტებისა და სპეციალისტებისათვის თეორიული, პრაქტიკული და გამოყენებითი ინფორმატიკის სფეროში) [21].

აღნიშნულ წიგნებში ასახვა ჰპოვა იმ სამეცნიერო და აკადემიურმა შედეგებმა, რომლებიც ავტორების მიერ წლების განმავლობაში იქნა შემუშავებული, განვითარებული და ფორმალიზებული საბოლოო პროდუქტამდე მსოფლიო ბანკის, USAID-ის და საქართველოს განათლებისა და მეცნიერების სამინისტროს შოთა რუსთაველის ეროვნული სამეცნიერო ფონდის პროექტების ბაზაზე. მნიშვნელოვანი წვლილი ქართულენოვანი სამეცნიერო და საგანმანათლებლო ლიტერატურის შექმნაში (კომპიუტინგის სფერო) შეიტანეს ჩვენი ფაკულტეტის დოქტორანტებმა, რომელთა უმრავლესობამ გაიარა 1-წლიანი სამეცნიერო სტაჟირება გერმანიის მოწინავე უნივერსიტეტებში, განახორციელეს თეორიული და ექსპერიმენტული კვლევები, წარმატებით დაიცვეს სადოქტორო დისერტაციები.

ამასთანავე, გერმანელი პროფესორებიც არიან ჩვენი წიგნების თანაავტორები.

წიგნები ინახება სტუ-ის, პარლამენტის და სხვა უნივერსიტეტების ბიბლიოთეკებში (ბათუმი, ქუთაისი, თელავი და სხვ.). 80 ელ-წიგნი მოთავსებულია სტუ-ს Web-გვერდზე: https://gtu.ge/Learning/ElBooks/ims_books.php, აგრეთვე საქართველოს პარლამენტის ციფრულ ბიბლიოთეკაში - „ივერიელი".

1.2. საგანმანათლებლო პროცესების მიმოხილვა

საგანმანათლებლო დაწესებულებებში მიმდინარეობს მრავალი სახის პროცესი. მათგან ამჯერად მხოლოდ რამდენიმეს, ჩვენთვის მნიშვნელოვანს განვიხილავთ. ესენია :

- მოსწავლეთა საერთო აღრიცხვის, მეცადინეობებსა და სხვა ღონისძიებებზე დასწრების კონტროლის პროცესი;
- მოსწავლეთა შეფასებების ანალიზისა და მათი დამუშავების პროცესი. მის საფუძველზე პროგნოზული სისტემის შექმნა;
- მოსწავლეთა მიღწევების ანალიზისა და პროგნოზის პროცესები;
- სწავლის საფასურის გადახდების ისტორიის მონიტორინგის და პროგნოზირების პროცესი;
- ბიბლიოთეკაში მიმდინარე პროცესი და სხვ.

1.2.1. მოსწავლეთა დასწრების კონტროლის პროცესი

მოსწავლის მშობელს შესაძლებლობა აქვს აკონტროლოს შვილის დასწრება, დაათვალიეროს დაგვიანებები, ადრე გასვლები, გაცდენები, შენობაში ყოფნის ხანგრძლივობა, შეამოწმოს მოსწავლის შენობაში ყოფნის მიმდინარე სტატუსი. მშობელს შეეძლება ნახოს კონკრეტულად რომელ შენობაში (თუ რათქმაუნდა არსებობს სკოლის სტრუქტურულად დაყოფილი ერთეულები) და კონკრეტულად რომელ კარზე განხორციელდება დაშვების ბარათის გამოყენება. ნებისმიერი დარღვევის (დაგვიანება, ადრე გასვლა, გაცდენა) შესახებ ინფორმაციას მშობელი მიიღებს ტექსტური

შეტყობინების სახით. ასევე სკოლაში არსებულ პასუხისმგებელ პირს ექნება შესაბამისი ინფორმაციის ნახვის საშუალება. სისტემის ფარგლებში შესაძლებელი იქნება მომავალი დაგვიანებების, ადრე გასვლების, გაცდენების რაოდენობის პროგნოზირება.

1,2,2, მოსწავლეთა შეფასებების ანალიზისა და მათი დამუშავების პროცესი

მშობელს შესაძლებლობა აქვს აკონტროლოს შვილის ნიშნები თითოეულ საგანში ელექტრონული ჟურნალის დახმარებით და ნახოს მასწავლებლის მიერ დაწერილი სიტყვიერი შეფასება, ანუ კონკრეტული ნიშნის შესახებ დეტალური ინფორმაცია. ასევე დაათვალიეროს ჩაბარებული დავალებებისა და გამოცდების კონკრეტული საგნისა და თარიღების მიხედვით. ასევე სასწავლო დაწესებულების თანამშრომელს შესაძლებლობა აქვს კონკრეტული საგნის ჭრილში გამოცდების შედეგების მიხედვით მომავალი გამოცდის შედეგების პროგნოზირება და ასევე გადაწერის (პლაგიატის) მცდელობების აღმოჩენა.

1.2.3. მოსწავლეთა მიღწევების ანალიზის პროცესი

სისტემის ფარგლებში მოცემული იქნება სასწავლო წლებისა და საგნების მიხედვით სტატისტიკა, ასევე რეიტინგები მოსწავლეებს შორის:

- ❖ მოსწავლეებს შორის რეიტინგები მიმდინარე სემესტრის კონკრეტული საგნის ჭრილში;
- ❖ მოსწავლეებს შორის რეიტინგები მიმდინარე სემესტრში ყველა საგნის შედეგების გათვალისწინებით;
- ❖ მოსწავლის სტატისტიკა კონკრეტული საგნის ჭრილში მიმდინარე და განვლილი სემესტრების მიხედვით;

❖ მოსწავლის სტატისტიკა მიმდინარე და განვლილი სემესტრების მიხედვით ყველა კურსის (საგნის) შედეგების გათვალისწინებით.

სისტემის ფარგლებში შესაძლებელი იქნება მომავალი სტატისტიკების პროგნოზირებაც.

1.2.4. სწავლის საფასურის გადახდების ისტორიის მონიტორინგის და ანალიზის პროცესი

მშობელს შესაძლებლობა აქვს მართოს სწავლის საფასურის გადასახადი და აკონტროლოს დავალიანებები. ასევე სასწავლო დაწესებულების თანამშრომელს სისტემა ეხმარება არსებული მდგომარეობის შეტყობინებით და მომავალი გადახდების სავარაუდო დარღვევების პროგნოზირებაში.

1.2.5. ბიბლიოთეკაში მიმდინარე პროცესი

მოსწავლეს შესაძლებლობა აქვს მისთვის საჭირო წიგნი გაიტანოს ბიბლიოთეკიდან კონკრეტული პერიოდით. სისტემა ინახავს ბიბლიოთეკაში არსებული წიგნების შესახებ ინფორმაციას და ახდენს წიგნის გატანების ისტორიის აღრიცხვას. ასევე იქნება შემდეგი სახის სტატისტიკები: „ყველა მოთხოვნადი წიგნები“, „თვის წიგნი’, „ყველა წიგნის გატანის სტატისტიკა“. მოსწავლის მიერ გატანილი წიგნების ისტორიის მიხედვით სისტემა მის მომხმარებელს საშუალებას მისცემს წინასწარი პროგნოზირების, თუ რომელი მოსწავლე, რომელ წიგნზე შეაჩერებს არჩევანს.

1.3. საგანმანათლებლო პროცესების აღმრიცხავი სისტემების კვლევა და ანალიზი

საგანმანათლებლო ორგანიზაციების პროცესების კვლევის შედეგად გამოიკვეთა, რომ საქართველოს სკოლებში არსებობს სასწავლო პროცესების მენეჯმენტის რამდენიმე სახის სისტემა, რომელთაც ასევე აქვს მოსწავლეთა აღრიცხვის, ცხრილების შენახვის და შეფასებების გაწერის შესაძლებლობა და სხვ.

მაგრამ ჩვენი მოძიებული ინფორმაციის მიხედვით, არცერთ მათგანს არ აქვს, მაგალითად, ხელოვნური ინტელექტის (AI-ს) დახმარებით სწავლების გზების გამარტივების და უკეთ წარმართვის ინსტრუმენტული საშუალებები.

განხილული ყოველი სისტემა ემსახურება მოსწავლეთა მონიტორინგს და არცერთი არ არის მოწყობილი, თუნდაც, მასწავლებელთა განვითარების ან ხელშეწყობისათვის.

ჩვენი სისტემა ეხმარება როგორც მოსწავლეს, ისე მასწავლებელს განვითარებაში და სწავლისა, თუ სწავლებისთვის საუკეთესო გზების მოძიებაში, შემდგომ ამ გზის განხორციელებაში.

ამჯერად, გავარჩევთ სხვადასხვა სფეროში გამოყენებად მანქანური დასწავლის (ML – Machine Learning) რამდენიმე, ალგორითმის მუშაობის პრინციპებს, აგრეთვე ახალი ინფორმაციული ტექნოლოგიების საშუალებებს, რომელთაც სასწავლო პროცესების მენეჯმენტის სისტემებში დიდი გამოყენება აქვს .

ჩვენ მიგვაჩნია, რომ ასეთი კომპლექსური ციფრული ტექნოლოგიები (ML+IT+IS+SE) ერთ-ერთი საუკეთესო გზაა საგანმანათლებლო ორგანიზაციებში უკეთესი სწავლების გზების დანერგვისა და სასწავლო პროცესის სრულყოფისათვის.

თავი 2
ინფორმაციული საზოგადოება და ახალი ციფრული ტექნოლოგიები

2.1. საინფორმაციო სისტემის პორტალის აგების მეთოდოლოგია

საქართველოში, ისევე როგორც ამერიკისა და ევროპის კოლეჯებსა და უნივერსიტეტებში, ახალი საუკუნის დასაწყისიდან სწრაფად ვითარდება ციფრული ტექნოლოგიების, მათ შორის ხელოვნური ინტელექტის სისტემების გამოყენება. ასეთივე მდგომარეობაა თითქმის ყველა სფეროში: ბიზნესი, ეკონომიკა, ჯანდაცვა, სოფლის მეურნეობა და ა.შ.

ახალი ინფორმაციული ტექნოლოგიების ფართოდ დანერგვით განათლების სისტემაში მნიშვნელოვნად უწყობს ხელს გლობალურ მიზანს – ***ინფორმაციული საზოგადოების*** ფორმირებას.

სტუ-ის ინფორმატიკისა და მართვის სისტემების ფაკულტეტზე აქტიურად მუშავდება ახალი ციფრული ტექნოლოგიების, დისტანციური სწავლების მეთოდოლოგიისა და მანქანური დასწავლის მეთოდებისა და ალგორითმების შესაბამისი ინსტრუმენტული საშუალებების შექმნა და დანერგვა [26, 30, 32].

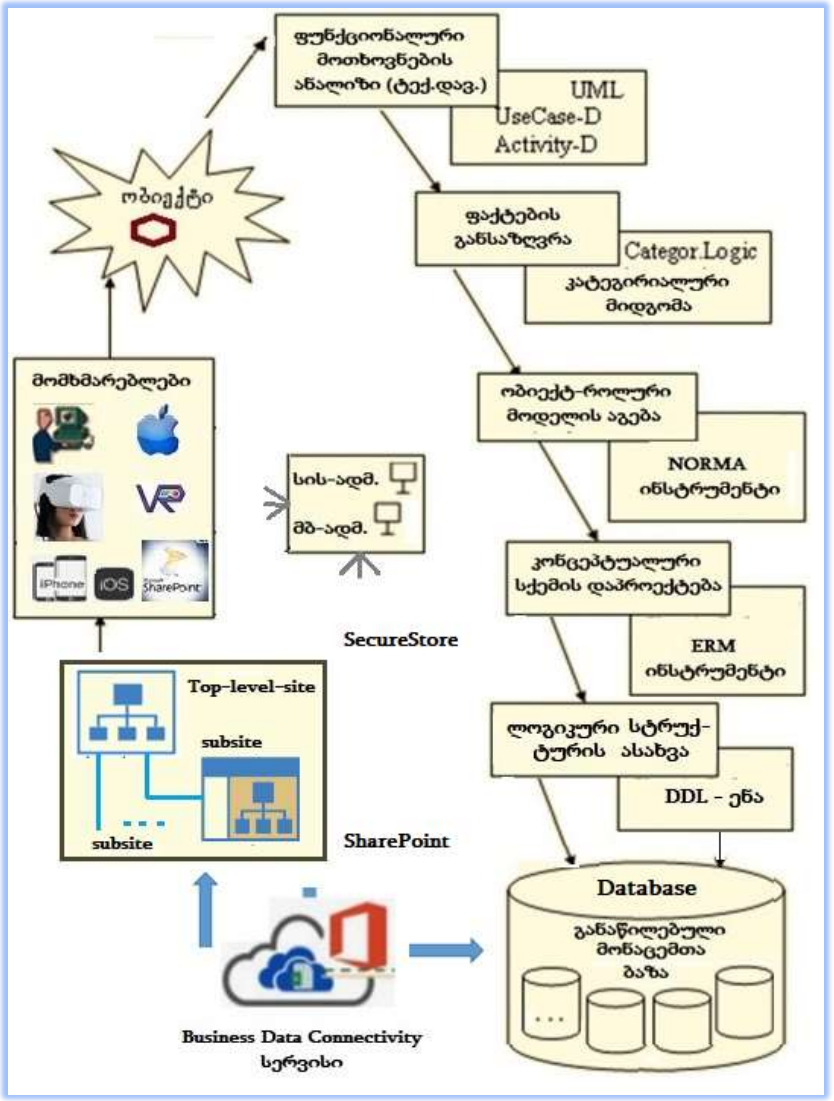
სტუ-ს UNESCO-ს კათედრის (აკად. გ. ჩოგოვაძე), მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერიის) დეპარტამენტის (პროფ. გ. სურგულაძე, პროფ. ლ. პეტრიაშვილი, ასოც. პროფ. ნ. თოფურია) და ქართული ფილოლოგიისა და მედიატექნოლოგიების დეპარტამენტის (პროფ. თ. ჯაგოდნიშვილი) ერთობლივი ძალისხმევით 2016-2021 წლებში შემუშავდა სხვადასხვა სფეროს ობიექტებისათვის საინფორმაციო სისტემების ისეთი (ზოგადი) მეთოდოლოგია, რომელიც ორიენტირებულია, საბოლოო მომხმარებელზე. ეს ტექნოლოგია შეიძლება განვიხილოთ როგორც *ინფორმაციული საზოგადოების ფორმირების* მხარდამჭერი [1].

ასეთი პროექტების შემუშავებაში მნიშვნელოვანი წვლილი შეიტანეს მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერიის) დეპარტამენტის დოქტორანტებმა: გ. კივილაძემ (2018 - ექსტრემალური სიტუაციების მართვის სისტემის აგება „112“-სთვის), მ. ხარიტონაშვილმა (2019 - ინტეგრირებული გაკვეთილე-ბის მონაცემთა ბაზა და მათი დანერგვა სკოლებში ვირტუალური რეალობის გამოყენებით), ბ. კახელი (2019 - ეკოსისტემის შექმნა დიდ მონაცემთა შესანახად ღრუბლოვანი ინფრასტრუქტურის არქიტექტურით MongoDB-ს ბაზაზე) და სხვ. [26, 32, 33].

2.1. ნახაზზე წარმოდგენილია ზემონახსენები ზოგადი მეთოდოლოგიის სქემა ჩვენი კონცეფციის გათვალისწინებით, რომელიც აგებულია ვებ-პორტალის სახით, მომხმარებელთა ინტერფეისების შესაქმნელად და ღრუბლოვანი არქიტექტურის გამოყენებით [26].

განვიხილოთ ამ მეთოდოლოგიის ძირითადი ეტაპების არსი, რომლის დეტალური რეალიზაცია გადმოცემული იქნება წიგნის მომდევნო თავებში.

- ***UML დიაგრამები:*** კომპიუტერული სისტემის ასაგებად პირველ ეტაპზე საჭიროა განისაზღვროს პროგრამული აპლიკაციის ბიზნეს-მოთხოვნები. ამისათვის გამოიყენება უნიფიცირებული მოდელირების ენა (UML), კერძოდ, მისი „როლებისა და ფუნქციების“ (UseCase) და აქტიურობის (Activity) დიაგრამები (აგებული VisualStudio.NET-2015 პლატფორმაზე). შეიძლება აგრეთვე ისეთი UML CASE ინსტრუმენტის გამოყენება, როგორიცაა SparX Systems ფირმის Enterprise Architect პაკეტი, რომელიც თავსებადია Visual Studio .NET-ის და აქვს C# კოდის გენერაციის საშუალება [34]. ან Object Management Group (OMG)-ის ინსტრუმენტი Visual Paradigm პაკეტი Java ენით [35].



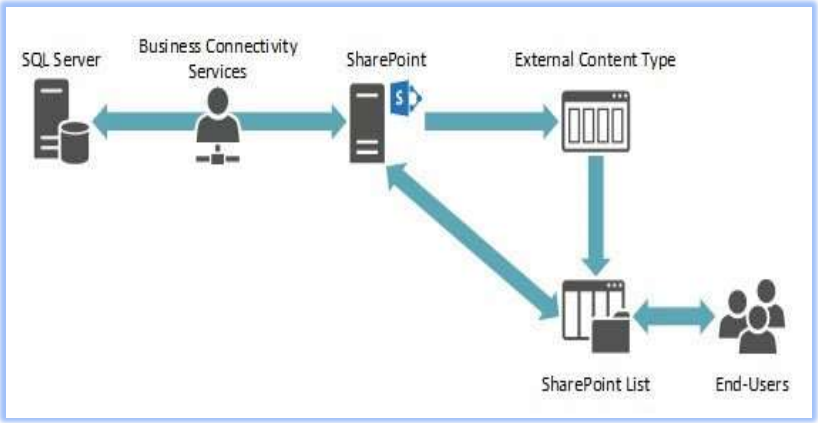
ნახ.2.1. საინფორმაციო სისტემის პორტალის აგების მეთოდოლოგიის სქემა

- ***ORM / ERM დიაგრამები:*** კომპიუტრეული სისტემის მონაცემთა ბაზის ასაგებად ვიყენებთთ ობიექტ-როლური მოდელირების CASE-ინსტრუმენტი (NORMA -Natural Object Role Modeling Architect) [36]. იგი თავსებადია VisualStudio.NET ფრეიმვორკის. აკადემიური კურსის (საგნის) პედაგოგი ამ ინსტრუმენტის ინტერფეისში (მცირე ტრენინგის გავლის შემდეგ), შეიტანს „საკუთარი საგნის“ ობიექტების შესახებ „ფაქტებს“. ესაა მათი ცოდნის ასახვის ეტაპი ობიექტებისა და ობიექტთაშორისი (პრედიკატული) კავშირების შესახებ. მაგალითად, *მეცნიერი არის ქვეყნიდან. მეცნიერს აქვს გვარი, სახელი. ქვეყანას აქვს დასახელება; ქვეყანას აქვს დედაქალაქი; დედაქალაქს აქვს დასახელება;*... და ა.შ.

NORMA ინსტრუმენტი თვითონ აგებს შეტანილი ფაქტების შესაბამის გრაფს. ესაა კონცეპტუალური სქემა-1, ანუ ORM-მოდელი, რომელშიც გადატანილია პედაგოგთა ცოდნა სხვადასხვა საგნების სფეროების შესაბამისად. სემანტიკური გადაკვეთა სრულდება საგანთა ტერმინების სინტაქსური ანალიზის საფუძველზე. მომდევნო ეტაპზე ORM-დიაგრამიდან ავტომატიზებულ რეჟიმში პროექტირდება ERM-დიაგრამა ანუ არსთა-დამოკიდებულების მოდელი. მისი კორექტირება შესაძლებელია მომხმარებლის მიერ. მოდელის ბოლო ვერსიის საფუძველზე სისტემა ავტომატურად აგენერირებს DDL (Data Definition Language) - ფაილს, რომელიც Ms SQL Server ბაზაში გადატანით უპრობლემოდ (პედაგოგების ჩარევის გარეში) ქმნის ფიზიკურ მონაცემთა ბაზას (სტრუქტურას) და ცხრილებს (Tables). მათი შევსების შემდეგ კონკრეტული ჩანაწერებით, მონაცემთა ბაზა მზად იქნება გამოსაყენებლად.

- ***Sharepoint: ვებ-პორტალის აგება.*** ვებ-პორტალის დასაპროექტებლად გამოყენებულია Ms SharePoint პაკეტი, რომელიც მომხმარებლებს თანამშრომლობის და ჯგუფური სერვისების გამოყენების მოქნილ შესაძლებლობას სთავაზობს. SharePoint-ის პორტალზე შესვლა შესაძლებელია ნებისმიერი ბრაუზერის საშუალებით.

SharePoint-ის დაკავშირება SQLServer-თან ხორციელდება SharePoint Designer-ის დახმარებით (ნახ.2.2) [26].



ნახ.2.2. SharePoint Designer

ამ ნახაზზე Business Connectivity Services ცენტრალიზებული ინფრასტრუქტურაა, რომელიც მონაცემებთან მუშაობის ინტეგრირებულ გადაწყვეტილებებს უზრუნველყოფს. ეს სერვისი უზრუნველყოფს მონაცემებთან წვდომას, რომლებიც განთავსებულია SharePoint-კონტენტის გარეთ (მაგ., SQL Server-ზე).

ამასთანავე აუცილებელია Secure Store სერვისის კონფიგურირება (ნახ.2.1). *მონაცემთა უსაფრთხოდ შენახვის სამსახური* (Secure Store) მოიცავს მონაცემთა ბაზას, სადაც ინახება მომხმარებელთა სააღრიცხვო ჩანაწერები და პაროლები იმ აპლიკაციების იდენტიფიკატორებისათვის, რომლებიც გამოიყენება საერთო რესურსებთან ავტორიზებული მიმართვის დროს.

მაგალითად, SharePoint Server-ის უსაფრთხოდ შენახვის მონაცემთა ბაზა შეიძლება გამოყენებულ იქნას სააღრიცხვო ჩანაწერების შესანახად / ამოსაღებად გარე მონაცემებთან მიმართვისას.

ამგვარად, Secure Store სამსახური ინახავს კონფიდენციალურ მონაცემებს, ამიტომ საჭიროა მისი დაშიფვრა. თავდაპირველად აუცილებელია იმ გასაღების გენერირება, რომლითაც მოხდება შინაარსის დაშიფვრა. ასევე აუცილებელია დაშიფვრის გასაღების არქივაცია. ზემოჩამოთვლილი ეტაპების განსახორციელებლად საჭიროა Sharepoint Central Administration-ის გააქტიურება და ამ სერვისების კონფიგურირება.

აგებული ვებ-პორტალის გამოყენება პედაგოგებს შეუძლიათ საკუთარი მობილურებიდანაც. მაგალითად, მონაცემთა შეტანა Windows Phone, iOS, Android ოპერაციულ სისტემებიდან სწრაფად აისახება პორტალის ვებ-გვერდზე და SQL Server-ის ბაზაში [37].

2.2. უნიფიცირებული მოდელირების ენა: მიზანი, კონცეფცია, სტრუქტურა და დიაგრამები

უნიფიცირებული მოდელირების ენა – UML (Unified Modeling Language) არის უნივერსალური (სტანდარტული) გრაფო-ანალიზური ვიზუალური ინსტრუმენტი ობიექტ-ორიენტირებული მოდელირების, პროგრამირების ავტომატიზაციისა და ტექსტური დოკუმენტირებისათვის [9, 38,39].

UML პრაქტიკულად არის გრადი ბუჩის მეთოდის (Booch Method) [9], ობიექტის მოდელირების ტექნიკის (Object-modeling technique – OMT) და ობიექტორიენტირებული პროგრამული უზრუნველყოფის ინჟინერიის (Object-oriented software engineering - OOSE) სინთეზი და გვევლინება როგორც ერთი, საერთო და ფართო გამოყენების მოდელირების ენა [40].

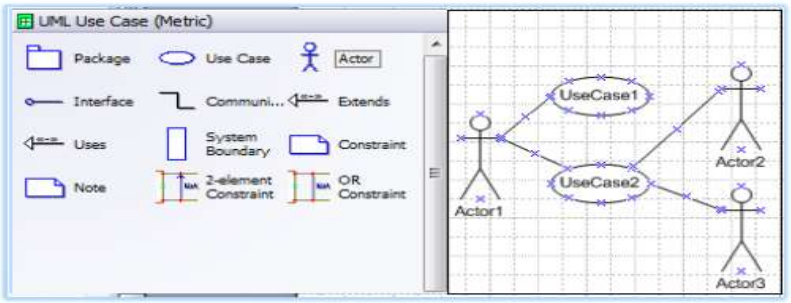
UML არის მსოფლიოში ერთ-ერთი ყველაზე ფართოდ გამოყენებადი მეთოდოლოგია კომპიუტერული საინფორმაციო სისტემების დიდი პროექტების ასაგებად. იგი შექმნილია საერთაშორისო ასოციაციის, ობიექტების მართვის ჯგუფის (Object Management Group – OMG) მიერ [41], ქმნის ღია სტანდარტებს ობიექტორიენტირებული აპლიკაციებისათვის.

UML აერთიანებს მონაცემთა მოდელირების (Entity relationship diagrams), ბიზნესმოდელირების (Work flows), ობიექტების და კომპონენტების მოდელირების მეთოდებს. იგი გამოიყენება პროგრამული უზრუნველყოფის და მისი ტექნიკური რეალიზაციის მთელი სასიცოცხლო ციკლის განმავლობაში.

უნიფიცირებული მოდილირების ენა შედგება დიაგრამების ინტეგრირებული ნაკრებისგან. ის შემუშავებულია სისტემისა და პროგრამული უზრუნველყოფის დეველოპერების დასახმარებლად პროგრამული აპოლიკაციების ნიმუშების დაზუსტების, ვიზუალიზაციის, კონსტრუირების და დოკუმენტირების პროცესებში; აგრეთვე ბიზნესპროცესების მოდელირების და სხვა არაპროგრამული ამოცანების გადასაწყვეტად. ეხმარება პროექტის გუნდებს კომუნიკაციაში, ბიზნეს-მოთხოვნების შესწავლასა და პროგრამული უზრუნველყოფის არქიტექტურული დიზაინის განსაზღვრაში [42].

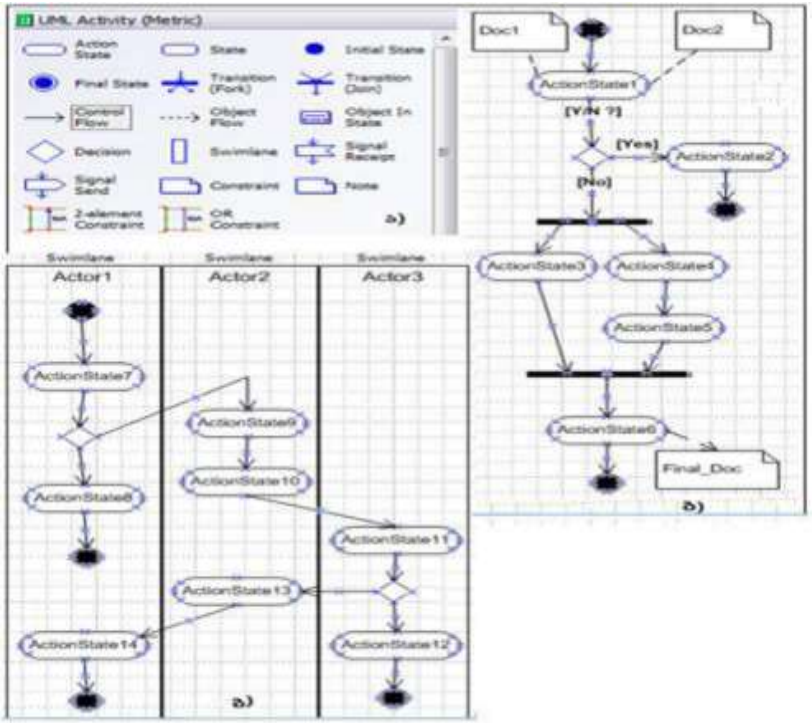
2.2.1. სისტემის ობიექტ-ორიენტირებული ანალიზი

➢ ***Use Case დიაგრამა.*** პირველად უნდა აიგოს სისტემის მომხმარებელთა დიაგრამა – „როლები-ფუნქციები“, ანუ ვინ მონაწილეობს (კაცუნა) და რა ფუნქციებით (ოვალი). ესაა UseCase ანუ გამოყენებით შემთხვევათა დიაგრამა (Actors-Actions) (ნახ. 2.3).



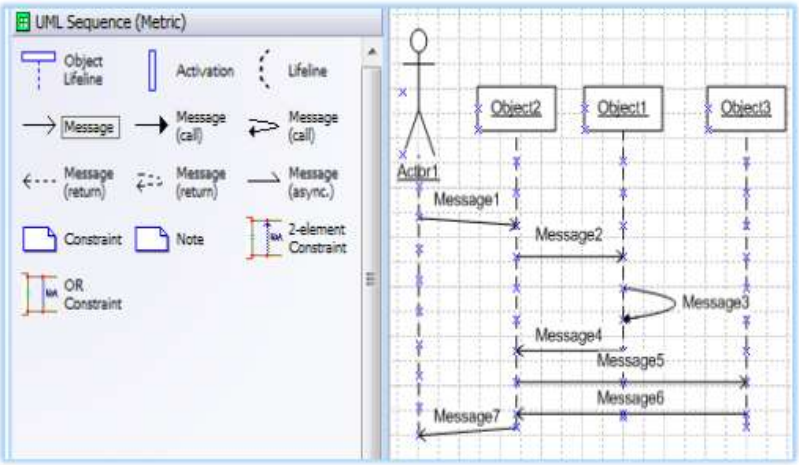
ნახ.2.3. UseCase დიაგრამის აგების ინტერფეისი

➢ **Activity დიაგრამა.** მეორეა აქტიურობათა დიაგრამა (ოვალის შესაბამისად ქმედებების ერთობლიობა). იგი კონკრეტული როლის ფუნქციაა, რომელიც შედეგება იერარქიულად სივრცესა და დროში დალაგებული მიმდევრობით ან პარალელურად შესასრულებელი სუბქმედებებისგან. მას აქვს ერთი დასაწყისი და რამდენიმე შესაძლო დასასრული. იგი მიეკუთვნება პროცესების აღწერის დინამიკურ მოდელს. Use Case და Activity დიაგრამის ერთობლიობის საფუძველზე ხდება დასკვნების გაკეთება საავტომატიზაციო ობიექტის მართვის სისტემის ფუნქციური და არაფუნქციური მოთხოვნილებების განსაზღვრის შესახებ (ნახ.2.4).



ნახ.2.4. Activity დიაგრამის: ა) ინსტრუმენტების პანელი; ბ) ქმედებათა სქემა (მარტივი); გ) ქმედებათა სქემა (როლების ბილიკებით)

➢ ***Sequence და Collaboration დიაგრამები.*** საპრობლემო სფეროს კონკრეტული ამოცანის *შესრულების სცენარს* აღწერს ***მიმდევრობითობის*** დიაგრამა. შესაბამისი დიაგრამის ფარგლებში ხდება როლის სისტემასთან ურთიერთქმედების ბიზნესპროცესის ქმედებათა და მათი მაინიცირებელ, სინქრონულ ან ასინქრონულ შეტყობინებათა დროში მიმდევრობით განლაგება (ნახ.2.5).

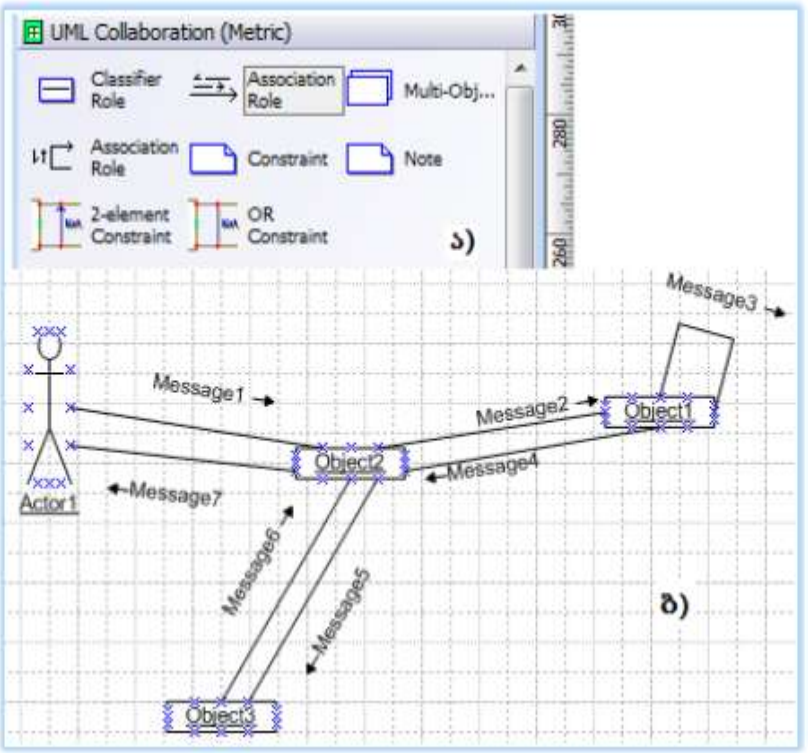


ნახ.2.5. Sequence დიაგრამის აგების ინტერფეისი

ნახაზზე გამოტანილია Sequence დიაგრამის აგების ინსტურმენტების პანელი (მარცხენა) და ასევე თავად აგებული დიაგრამა (მარჯვენა),

შესაბამისი მიმდევრობითობის დიაგრამის ტრანსფორმაციით მიიღება ავტომატურად თანამოქმედების დიაგრამა (ნახ.2.6). დიაგრამაზე არ არის მონაცემთა გაცვლის და შეტყობინებათა მიმდევრობა დროში დალაგებული, სამაგიეროდ კარგად ჩანს კლასის ობიექტებს შორის კავშირებისა და ასევე ინფორმაციული ნაკადების გაცვლის სემანტიკა. CASE ტექნოლოგიის მრავალ

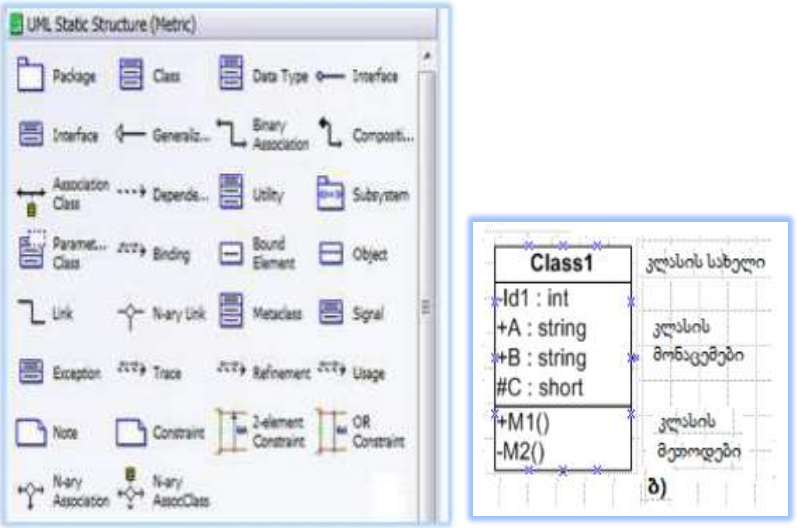
პროდუქტში (მაგალითად, ფირმა Rational Rose) თანამიმდევრობითობის დიაგრამის აგება ხდება ავტომატიზებულ რეჟიმში. ანუ ჯერ აიგება მიმდევრობითობის დიაგრამა და შემდეგ ინსტრუმენტის რომელიმე სწრაფი ღილაკით (მაგალითად, F5) სისტემა თვითონ ააგებს თანამოქმედების დიაგრამას].



ნახ.2.6. Collaboration დიაგრამის აგების ინტერფეისი (ა) და სქემის მაგალითი (ბ)

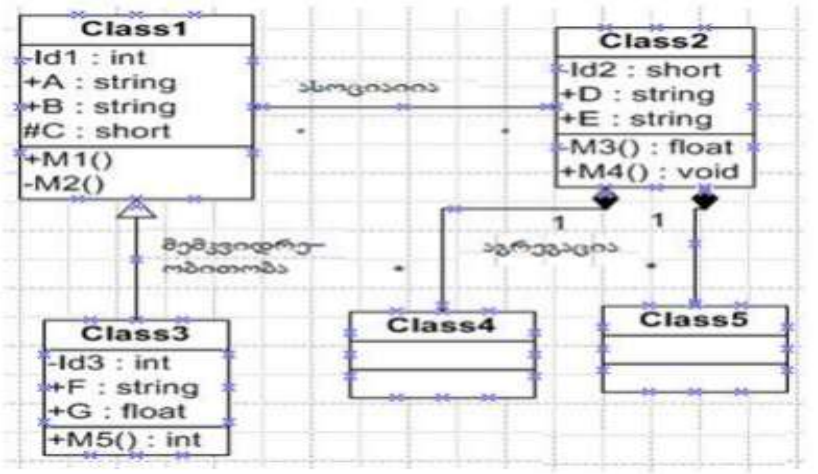
2.2.2. სისტემის ობიექტ-ორიენტირებული დაპროექტება

➢ **კლასების (Class) დიაგრამა.** პროგრამული უზრუნველყოფის ინჟინერიაში UML კლასის დიაგრამა სტატიკური სტრუქტურაა. იგი აღწერს კლასის მონაცემებს, როგორიცაა: მისი დასახელება, ატრიბუტები, ოპერაციები (ან მეთოდები). ესაა კლასის ინკაფსულაცია (ნახ.2.7).



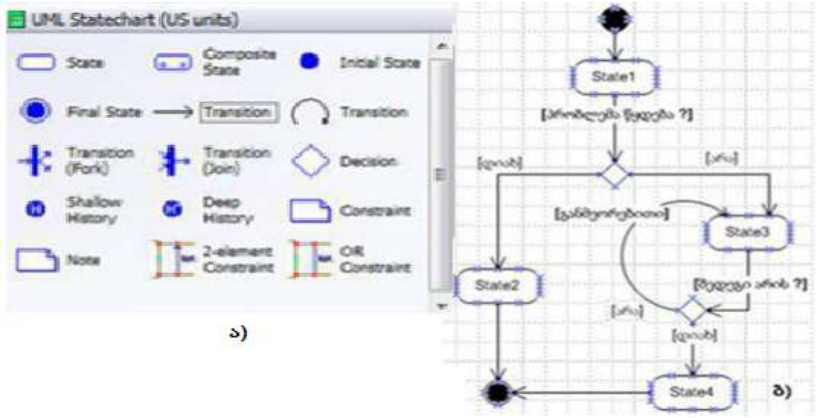
ნახ.2.7. ინსტრუმენტების პანელი და კლასი

➢ **Class-Assotiation დიაგრამა.** კლასთაშორისი კავშირები შეიძლება იყოს: მემკვიდრეობითი, აგრეგატული, რელაციური და ასოციაციური. 2.8 ნახაზზე ნაჩვენებია კლასთა ასოციაციის დიაგრამა ასეთი კავშირების საფუძველზე.



ნახ.2.8. Class-Assotiation დიაგრამა

➢ ***მდგომარეობათა (Statechart) დიაგრამა*** არის ყოფაქცევის დიაგრამა Activity და Sequence, Collaboration დიაგრამების მსგავსად. იგი აღწერს ქმედებებს, მდგომარეობებს, მდგომარეობათა გადასვლებს და მოვლენებს (ნახ.2.9).



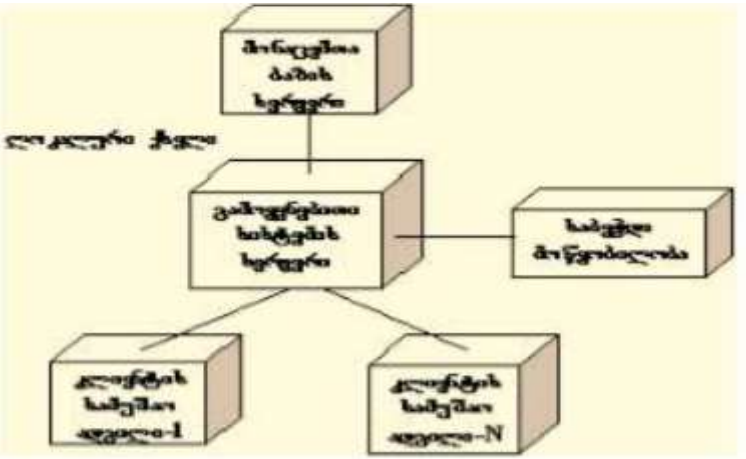
ნახ.2.9. Statechart-ის პანელი (ა) და ზოგადი დიაგრამა (ბ)

მისი გამოყენება ყველა კლასისათვის არაა საჭირო. აუცილებელია მხოლოდ მაშინ, როდესაც კლასი შეიძლება იმყოფებოდეს რამდენიმე მდგომარეობაში და თითოეულ მათგანში მისი ქცევა უნდა იყოს სხვადასხვანაირი.

- ***კომპონენტების (Components) დიაგრამა*** აიგება პროგრამული რეალიზაციის ეტაპზე, რომელშიც იგულისხმება პროგრამული კოდების დამუშავება (მაგ., dll,. cs, .cpp, .h, .exe და ა.შ.) (ნახ. 2.10.)

ნახ.2.10. კომპონენტების დიაგრამა

- ***განთავსების (Deployment) დიაგრამა*** აიგება განაწილებული სისტემის რეალიზაციის ბოლო ეტაპზე, რომელიც აღწერს კომპონენტების განაწილებას „კლიენტ სერვერის" ქსელში (ნახ.2.11).



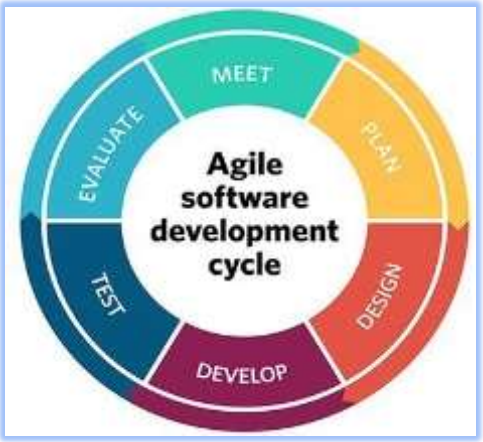
ნახ.2.11. განთავსების დიაგრამა

2.3. Agile მეთოდოლოგიის არსი

„მოქნილი (სწრაფი) მოდელირება“ *(Agile Modeling AM)* – არის პროგრამული უზრუნველყოფის შექმნის გუნდის მუშაობის ეფექტური ორგანიზების ხერხი (მეთოდოლოგია), განსაკუთრებით დამკვეთების მოთხოვნილებათა დასაკმაყოფილებლად (შუალედური ცვლილებების გათვალისწინებით).

გუნდში აქტიურად მონაწილეობს დამკვეთის წარმომადგენელი, სისტემის ანალიზის, დაპროექტებისა და აგების პროცესებში. AM-გუნდის მუშაობის მთავარი მიზანია ეფექტუროზა, დამკვეთის მეტი წვლილის ჩადება საბოლოო პროდუქტში, შესაძლოდ მარტივი მოდელების აგება, *სამუშაო სისტემის შექმნა და არა თეორიის* ! [43].

Agile მოდელირების ეტაპები მოცემულია 2.12 ნახაზზე.



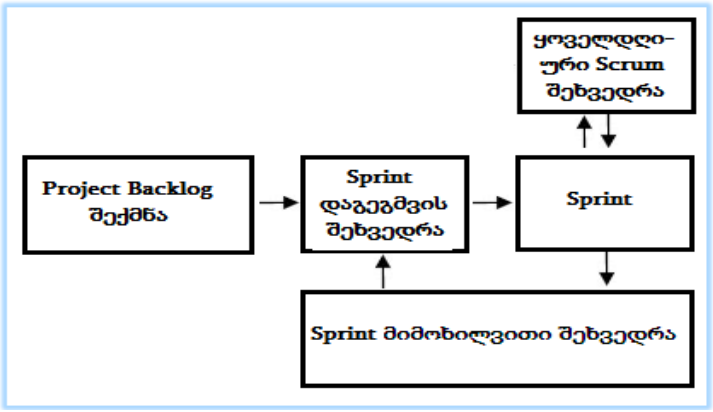
ნახ.2.12. Agile-მეთოდოლოგიის ეტაპები

დაპროგრამების მოქნილი მეთოდოლოგია (Agile Software Methodology) განიხილავს განსხვავებულ Agile-მეთოდებს, როგორიცაა მაგალითად, Extreme Programming (XP), Scrum, Kanban/Lean, Agile Unified Process (AUP), Test-Driven Development (TDD), Rapid Application Development (RAD) და სხვ. [44-47].

მოქნილი პროგრამირების ზოგიერთ მეთოდს აქ მოკლედ შევეხებით, კერძოდ Scrum და Kanban/Lean მეთოდების მაგალითზე [48].

2.3.1. Scrum – მოქნილი მეთოდი

„Scrum - მოქნილი მეთოდის ფრეიმვორკია“. *იგი* გუნდური მიდგომაა ახალი სერვისებისა და პროდუქტების დასამუშავებლად (არა მხოლოდ პროგრამირების სფეროში) [46]. მეთოდის ძირითადი არსი მდგომარეობდა მცირე ზომის (5-9 კაცი) უნივრსალური გუნდის შეკრულ, თანამიმდევრულ მუშაობაში, რომელიც იტერაციულად, ე.წ. Sprint-ებით ამუშავებს პროექტის ყველა ფაზას (ნახ.2.13).



ნახ.2.13. Scrum-მეთოდი Sprint-ბიჯებით

როგორც წესი, დასაწყისში პროდუქტის მფლობელი, Scrum-ოსტატი, გუნდი, ასევე დამკვეთის წარმომადგენელი და სხვა დაინტერესებული პირები განსაზღვრავენ, თუ რომელი მოთხოვნებია Project Backlog-იდან უფრო პრიორიტეტული და რომელი უნდა განხორციელდეს მოცემული სპრინტის ფარგლებში. ფორმირდება Sprint-Backlog. შემდეგ Scrum-ოსტატი და Scrum-გუნდი განსაზღვრავენ, თუ როგორ უნდა იქნას მიღწეული დასმული

მიზნები Sprint-Backlog-დან. მისი ყოველი ელემენტისათვის დადგინდება ამოცანათა სია და შეფასდება მათი შრომატევადობა;

იმართება ყოველდღიური 15-წუთიანი Scrum თათბირი, რომლის მიზანია იმის გარკვევა, თუ რა მოხდა წინა თათბირის შემდეგ, კორექტირდეს სამუშაო გეგმა დღევანდელი დღის შესაბამისად და განისაზღვროს არსებული პრობლემების გადაწყვეტის გზები.

Scrum-გუნდის ყოველი წევრი პასუხობს სამ კითხვას:

- *რა გააკეთა წინა თათბირის შემდეგ,*
- *რა პრობლემები აქვს და*
- *რა უნდა გააკეთოს მომდევნო შეხვედრამდე.*

Sprint მიმოხილვის შეხვედრა იმართება ყოველი სპრინტის დამთავრების შემდეგ. თავიდან Scrum-გუნდი წარმოადგენს პროდუქტის დემონსტრაციას, რომელიც ამ სპრინტის დროს განხორციელდა. აქ მოწვეული იქნება დამკვეთის ყველა დაინტერესებული წარმომადგენელი.

Scrum-გუნდი აანალიზებს ბოლო სპრინტის დროს ერთობლივი მუშაობის დადებით და უარყოფით მომენტებს, გამოიტანს დასკვნებს და იღებს მნიშვნელოვან გადაწყვეტილებებს შემდგომი მუშაობისათვის. Scrum-გუნდი ასევე ეძებს გზებს მომავალი სამუშაოს ეფექტურობის ასამაღლებლად. შემდეგ ციკლი მეორდება.

2.3.2. Kanban/Lean – მოქნილი მეთოდი

„ეკონომიური მოქნილი მეთოდი Kanban“ – პროგრამული უზრუნველყოფის დამუშავების ეკონომიური მეთოდია (Lean method of software development) [47]. სიტყვა kanban იაპონურად არის „სასიგნალო ბარათი“ (kan - სიგნალი, ban - ბარათი). იგი Toyota-ს ავტომანქანების წარმოების ფირმის ტექნოლოგიაა, რომელიც შეიქმნა წარმოების სტაბილური ნაკადის უზრუნველსაყოფად და

მარაგების დონის შესამცირებლად. იყენებენ ასეთ ახსნასაც – „დაუმთავრებელი წარმოების მოცულობის შემცირება“. Kanban-ის გამოყენების ფუძემდებლად ინფორმაციული ტექნოლოგიების სფეროში ითვლება *დევიდ ანდერსონი* (2007) [47].

Kanban-მეთოდი იყენებს სამუშაო მსვლელობის (ნაკადის) ვიზუალიზაციას. კერძოდ, ღირებულებათა ჯაჭვი პროცესის სხვადასხვა ეტაპისათვის (მაგალითად, *მოთხოვნილების განსაზღვრა, პროგრამირება, დოკუმენტირება, ტესტირება, დანერგვა*) კარგადაა ვიზუალიზირებული ყველა მონაწილისათვის. ეს ხორციელდება Kanban-დაფის (Kanban-Board) დახმარებით, რომელზეც სხვადასხვა კვანძები (Stations) აისახება სვეტების სახით (ნახ.2.14.) [48].

ნახ.2.14. Kanban-ის დაფა (იყენებს Software Development Life Cycle-ს)

ინდივიდუალური მოთხოვნილებები (ამოცანები, ფუნქციები, მომხმარებელთა ისტორიები, მინიმალური საბაზრო მახასიათებლები და ა.შ.) ჩაიწერება საჯრიცხვო ბარათებში („ბილეთები“, მიმაგრებული დაფის უჯრაზე, Tiket – TK). ისინი დროის და შესრულებული ბიჯის შესაბამისად გადაადგილდება დაფაზე მარცხნიდან მარჯვნივ.

ვიზუალიზირება და WiP-ის (Work in Progress) შეზღუდვები მარტივი საშუალებაა, რომლითაც სწრაფად ხდება თვალსაჩინო, თუ რა სისწრაფით მოძრაობს ბილეთები სხვადასხვა კვანძებში და სად გროვდება (იჭედება) ისინი. იმ კვანძებს, სადაც გროვდება ბილეთები და ამ დროს მომდევნო კვანძი თავისუფალია, უწოდებენ *ვიწრო ადგილებს*.

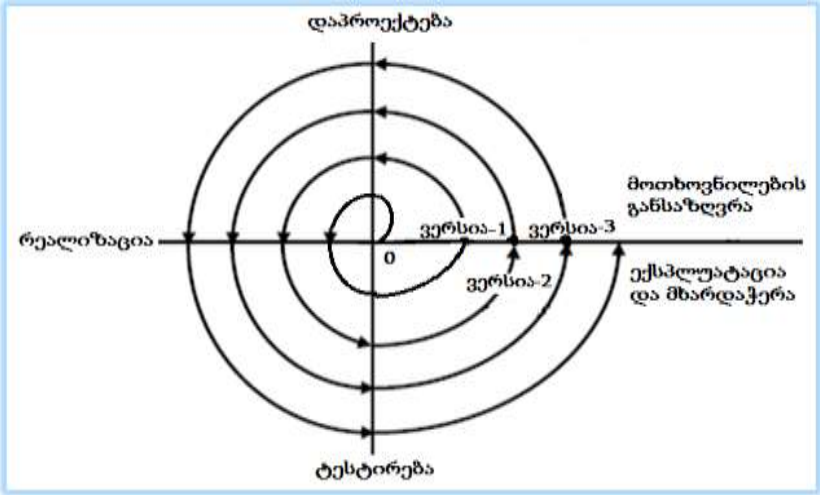
Kanban-დაფის ანალიზით შესაძლებელია ზომების მიღება მაქსიმალურად თანაბარი ნაკადის მისაღწევად. მაგალითად, შეზღუდვები შეიძლება შეიცვალოს ცალკეული კვანძებისათვის, შეიძლება შემოტანილ იქნას ბუფერები (განსაკუთრებით ვიწრო ადგილების გაჩენამდე, რაც გამოწვეულია დროებით რესურსების წვდომის გამო), კვანძებზე მომუშავეთა რაოდენობა შეიძლება შეიცვალოს, აღმოიფხვრას ტექნიკური პრობლემები და ა.შ. სრულყოფის უწყვეტი პროცესი არის Kanban-ის განუყოფელი ნაწილი.

2.4. UML მეთოდოლოგია და ინფორმატიკის დიდაქტიკა - ინტერდისციპლინური მიდგომის საფუძველი

განვიხილავთ საერთაშორისო საუნივერსიტეტო განათლების ინფორმატიკის (მსგავსია კომპიუტერული მეცნიერების) პროგრამის ძირითადი აკადემიური კურსების სწავლების სისტემის სრულყოფის გზებს, *საგანთშორისი კავშირების* გათვალისწინებით.

შემოთავაზებულია არსებული, სამსაფეხურიანი უმაღლესი განათლების სისტემისთვის (ბაკალავრიატი, მაგისტრატურა, დოქტორანტურა) ინტერდისციპლინური სწავლების კონცეფცია და მისი რეალიზაციის პროგრამა ევრო-ამერიკული უნივერსიტეტების გამოცდილებისა და UML/Agile მეთოდოლოგიებით [48, 49].

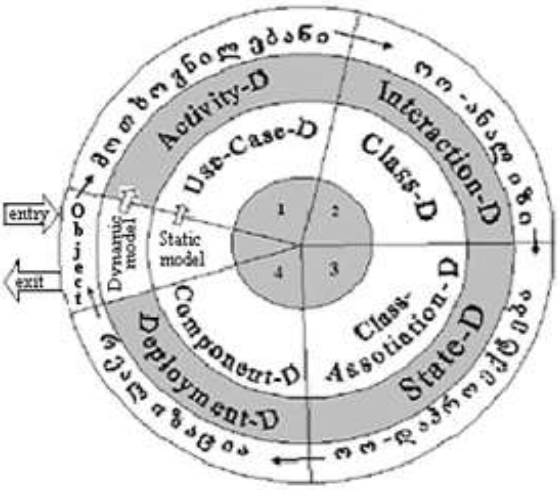
2.15 ნახაზზე ნაჩვენებია პროგრამული სისტემის შექმნის სასიცოცხლო ციკლის 4-ეტაპი სპირალური მოდელისათვის: მოთხოვნილებათა განსაზღვრა, პროექტირება, რეალიზაცია, ტესტირება. მიიღება 1-ელი მუშა ვერსია, შემდეგ ციკლზე მე-2 და ა.შ.



ნახ.2.15. სასიცოცხლო ციკლის სპირალური მოდელი

პროგრამული სისტემის ვერსიები (1,2,3,..) იქმნება პროტოტიპების სახით, იგი ვითარდება სპირალურად და ყოველი ახალი ვერსია სულ უფრო ახლოა დამკვეთი-მომხმარებლების მოთხოვნილებებთან. ტესტირების და საექსპლუატაციო დანერგვის პროცესში მონაწილეობენ სისტემური ანალიტიკოსები და დამკვეთები, რაც საშუალებას იძლევა ოპერატიულად განისაზღვროს საჭირო ცვლილება / დამატებები შემდეგი ვერსიისათვის.

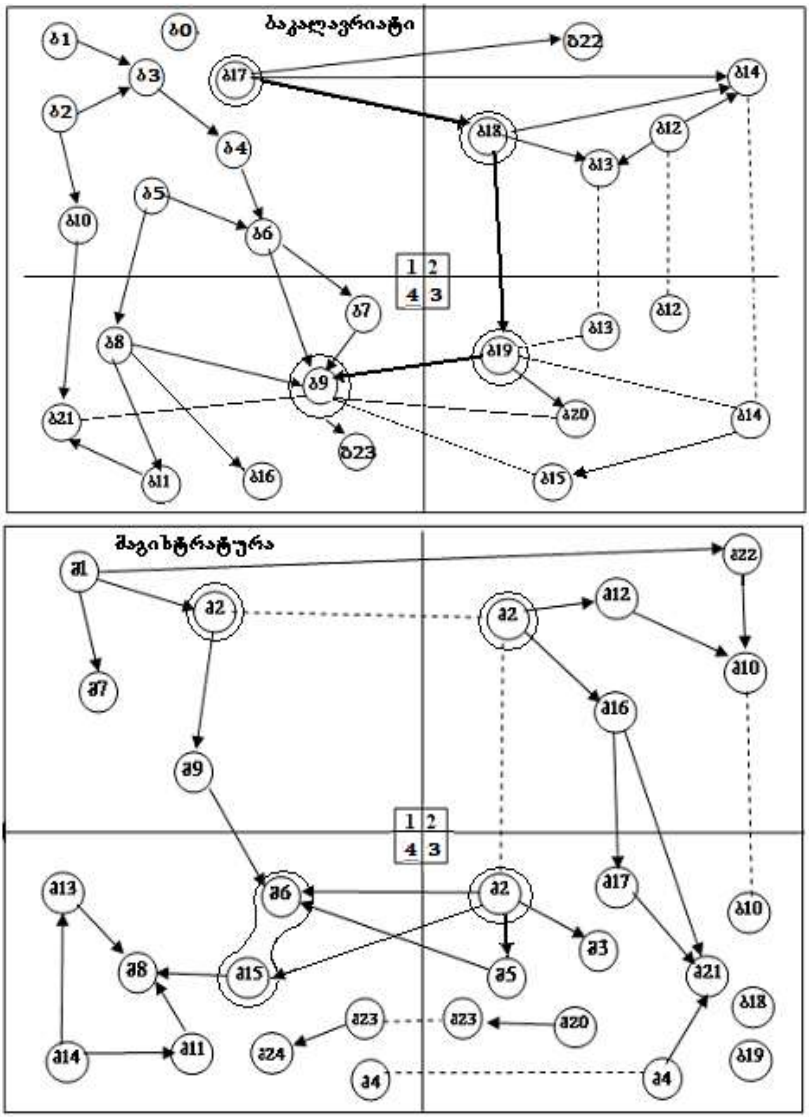
UML მეთოდოლოგიის კლასიკურ სქემაზე (გ. სურგულაძის ინტერპრეტაციის მოდელი - ნახ,2,16 [50-52]) ნაჩვენებია პროგრამული სისტემის სასიცოცხლო ციკლის ზემოგანხილული 4 ეტაპისა და უნიფიცირებული მოდელირების 4-ეტაპის და შესაბამისი სტატიკურ-დინამიკური მოდელების (დიაგრამების) კავშირი. ეს ორი მეთოდოლოგია შეიძლება ჩაითვალოს პროგრამული აპლიკაციების შექმნის იზომორფულ სისტემებად. მათი შინაგანი სემანტიკა ეფუძნება ობიექტ-ორიენტირებული მოდელირების თეორიას [48, 53].



ნახ.2.16. UML მეთოდოლოგიის 1-4 ეტაპები

ჩვენ 2.2 პარაგრაფში დეტალურად განვიხილეთ კლასიკური UML-ის ეს 8 დიაგრამა. UML/2-ის ვერსიაში შემოტანილია კიდევ 7 ახალი დიაგრამა, რომელთა მიზანი მოდელის სემანტიკური გაფართოებაა [49]. ამერიკა/ევროპის უნივერსიტეტების „პროგრამული ინჟინერიის“ დეპარტამენტებში UML და Agile მეთოდოლოგიები ძირითად, პრიორიტეტულ კურსებად არის აღიარებული.

ახლა განვიხილოთ თუ როგორ ხდება სტუ-ში ინფორმატიკის დიდაქტიკის პროცესის წარმართვა შესაბამისი აკადემიური კურსების გათვალისწინებით (მაგალითად, პროგრამული ინჟინერიის კონცენტრაციით). ამ შემთხვევაში UML-მეთოდოლოგია (სისტემების ოო-ანალიზი, ოო-პროექტირება და ოო-რეალიზაცია და დანერგვა) განიხილება როგორც ინტერდისციპლინური კავშირების მთავარი საგანი. ჩვენ შემთხვევაში ვიხილავთ საუნივერსიტეტო განათლების ბაკალავრიატისა და მაგისტრატურის საფეხურებს „ინფორმატიკის“ პროგრამით (ნახ. 2.17).



ნახ.2.17. ეტაპების მიხედვით კურსების გადაწილება (2019-2021 წწ.).
/პროგრამებში დასაშვებია მცირე მოდიფიკაცია/

2.1 ცხრილში ნაჩვენებია მაგალითად, ბაკალავრიატის „ინფორმატიკის“ პროგრამის (პროგრამული ინჟინერიის კონცენტრაციის) აკადემიური კურსების სია, რომლებსაც შეესაბამება 2.17 ნახაზზე მოყვანილი საგანთა იდენტიფიკატორები (ნომრები).

ბაკალავრიატი **ცხრ.2.1**

№	დასახელება	UML / სემესტრ.
ბ0	შესავალი ინფორმატიკაში / კომპიუტერის არქიტექტურა	1/1 1/2
ბ1	კომპიუტერული უწარები	1/1
ბ2	დაპროგრამების საფუძვლები	1/1
ბ3	მონაცემთა სტრუქტურები და ალგორითმები	1/2
ბ4	შესავალი მოწაცემთა ბაზებში	2/2
ბ5	ოო- დაპროგრამება (C++/C#/Py/J++)	4/4
ბ6	ოო-დაპროგრამება & აპლიკაციები	4/5
ბ7	საკურსო პროექტი ოოდ & აპლიკ.	4/5
ბ8	კომპიუტერ. და ქსელის არქიტექტურა	4/3,6
ბ9	პროგრამული პროდუქტების დეველოპმენტი	4/7
ბ10	Web – ტექნოლოგიების საფუძვლები	4/4
ბ11	Web-პროგრ. (XML, Angular,JQuery)	4/5
ბ12	მულტიპარადიგმული .პროგრამირება (Java ენაზე)	2,3/5
ბ13	მობილური პროგრამირება (Android)	2,3/6
ბ14	საბაკალავრო პროექტი	2,3/8
ბ15	მონაცემთა ბაზების დაპროექტება (SQL-სერვერი, No SQL)	3/2,6
ბ16	ინფორმაც. უსაფრთხოების საფუძ.	3/7
ბ17	პროგრ. ინჟინერიის საფუძვლები	1/5
ბ18	პროგრ.აპლიკ. & მონაც.მენეჯმენტი	2/6
ბ19	სისტემების ოო-ანალიზი & ოო-დაპროექტება (UML)	3/6
ბ20	კონტენტ-მენეჯმენტის სისტემები	3/7
ბ21	Web-პროგრამირება (PHP/MySQL)	4/7
ბ22	საკურსო პროექტი პროგრ.ინჟინერ.	1/5
ბ23	საკურს.პრ. პროგ.პროდ.დეველოპ.	4/7

მაგისტრატურა **ცხრ.2.2**

№	დასახელება	UML-სემესტრ
მ1	საინფორმაციო სისტემების მენეჯმენტი	1,4/1
მ2	სისტ. ობ-ანალიზი და ობ-დაპროექტება	1,2,3/1
მ3	გადაწყვეტილების მიღების ხელშემწყობი სისტემების დაპროექტება	3,4/2
მ4	უსაფრთხოება, საიდუმლოება და ქსელური სისტემები	3,4/3
მ5	პროგრ. უზრუნველყოფის დაპროექტება: პრინციპები, მოდელები, ნიმუშები	3,4/2
მ6	განაწილებული სისტემების დაპროგრამება C#.NET da VB.NET პლატფორმაზე	4/2,3
მ7	სამაგიდო აპლიკაციები მენეჯერებისთვის	1,4/3
მ8	Web-აპლიკაციების დაპროგრამება	4/1
მ9	განაწილებული სისტემების დაპროგრამების კონცეფცია Ms.NET ტექნოლოგიებით (ა)	1/1
მ10	ელ-ბიზნესის/კომერციის სისტემების მოდელირება და დაპროექტება (ა)	2,3/1
მ11	ქსელური არქიტექტურები ბიზნესისათვის (ა)	4/3
მ12	ბიზნეს-პროცესების მოდელირება პეტრის ქსელებით (ა)	2/4
მ13	დაპროგრამება Java ენაზე ინტერნეტ-ბიზნესისათვის	4/2
მ14	დაპროგრამება XML ენაზე	4/2
მ15	Web-აპლიკაციების დამუშავება მონაცემთა ბაზების საფუძველზე (ADO.NET, ASP.NET)	4/3
მ16	მონაცემთა ბაზების მართვა საინფორმაციო სისტემებში	2/1
მ17	მონაცემთა ბაზების ადმინისტრირება	3/2
მ18	სტატისტიკური მოდელების თეორია და პრაქტიკა, პროგნოზირების მეთოდები	3/3
მ19	ბიზნესის მართვის ოპტიმალური მეთოდები	3/3
მ20	ხელოვნური ინტელექტის სისტემები	3/1
მ21	მონაცემთა საცავები კორპორაციულ სისტემებში და მათი მენეჯმენტი (ა)	3,4/3
მ22	ბიზნესის ანალიზის და ინტელექტუალური მართვის ტექნოლოგია (BI) (ა)	2,3/3
მ23	ექსპერტული სისტემების დაპროექტება და რეალიზაცია (ა)	3,4/4
მ24	ტექსტების ანალიზის ლინგვისტური სისტემები (ა)	4/3

შენიშვნა: მაგისტრატურის მოცემულ ცხრილშიც დასაშვებია საგანთა მოდიფიკაცია სასწავლო წლების მიხედვით.

2.5. მონაცემთა საცავები და ბაზები

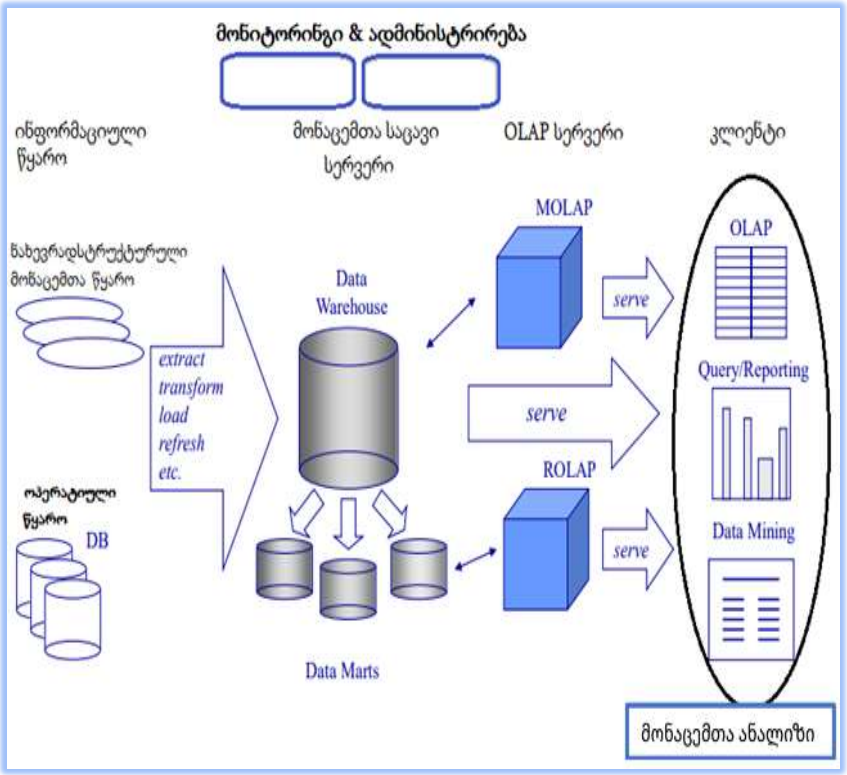
ინფორმატიკის დიდაქტიკის ერთ-ერთი ძირითადი და მნიშვნელოვანი მიმართულებაა ორგანიზაციის მენეჯმენტის სისტემის ინფორმაციული უზრუნველყოფის, კერძოდ, მონაცემთა საცავის და მონაცემთა ბაზის შესწავლა.

- მონაცემთა საცავი (Data Warehouse) არის საგნობრივ სფეროზე ორიენტირებული, ინტეგრირებული, ქრონოლოგიურად დალაგებული, ხანგრძლივი დროის (10-15 წლის მონაცემები) სპექტრში მიღებული არასტაბილურ მონაცემთა ერთობლიობა [54]. იგი გამოიყენება ოპტიმალური გადაწყვეტილებების მისაღებად მენეჯმენტის პროცესების სწორი ორგანიზაციისა და საოპერაციო მონაცემთა საფუძველზე.

მონაცემთა საცავში იყენებენ ინფორმაციის ონლაინ ანალიზს OLAP (Online Analytical Processing) ინსტრუმენტით, რომლის ფუნქციონირება და მოთხოვნის შესრულების ხარისხი საკმაოდ მაღალია. იგი გადაწყვეტილების მიღების მხარდამჭერი ტექნოლოგიაა, აქვს მონაცემთა მოწესრიგების, ინტეგრაციის და ონლაინ ანალიზის პროცესები. მონაცემთა ონლაინ ანალიზის ტექნოლოგია უზრუნველყოფს მონაცემთა აგრეგაციას და შემაჯამებელი მნიშვნელობის ვიზუალიზაციას ინფორმაციის ანალიზის სხვადასხვა ჭრილში.

მონაცემთა ბაზაში ინახება განსხვავებული ტიპის მონაცემები. საცავი არის მრავალჯერადი ჰეტეროგენული მონაცემთა წყაროს რეპოზიტორი, რომელიც ორგანიზებულია ერთიანი სქემის ქვეშ და ხელს უწყობს მმართველი გადაწყვეტილების მიღებას.

2.18 ნახაზზე მოცემულია მონაცემთა საცავის ზოგადი არქიტექტურა [55]. საცავში ბოლო ეტაპზე ხდება მულტიფუნქციური ოპერატიული მონაცემთა ბაზებიდან ინფორმაციის მიღება და მრავალგანზომილებიანი ონლაინ-ანალიზი,



ნახ.2.18. მონაცემთა საცავის ზოგადი ინფრასტრუქტურა

მონაცემთა საცავში, ინფორმაციის განახლების ყველა ეტაპზე, ხდება მონაცემთა არქივაცია. მონაცემთა საცავის დაპროექტება რთული პროცესია, რომლის დროსაც წყდება:

– საცავის არქიტექტურის, შენახვის სერვისების, მონაცემთა წყაროს, დაგეგმვის მოცულობის და OLAP სერვერის მოცულობის განსაზღვრა;

– სერვერების, სამომხმარებლო ინსტრუმენტების და სანახების ინტეგრირება;

– საცავის დიზაინის და ხედების სქემატური განსაზღვრა;

– საცავის ფიზიკური სტრუქტურის დადგენა, მონაცემთა ორგანიზება და შესაბამისი მეთოდების განსაზღვრა;

– მონაცემთა წყაროსთან დაკავშირება შესაბამისი Driver-ებით;

– მონაცემთა ტრანსფორმაციის, განახლების და ჩამოტვირთვის საჭირო დიზაინის და სკრიპტების იმპლემენტაცია;

– რეპოზიტორის შევსება სქემების, სკრიპტების და სხვა მეტამონაცემებით;

– სამომხმარებლო აპლიკაციის და დიზაინის იმპლემენტაცია;

– მონაცემთა საცავის და შესაბამისი აპლიკაციის ფართოდ გამოყენება.

მონაცემთა რელაციური ბაზების „მამამ“, ედგარ კოდმა ჩამოაყალიბა 12 წესი, რომელსაც უნდა აკმაყოფილებდეს განაწილებული სისტემა მონაცემთა საცავით, რათა ჩატარდეს სრულფასოვანი ოპერატიული ანალიზი [56, 57]:

1) *მონაცემთა მრავალგანზომილებიანი კონცეპტუალური ხედვა.* მომხმარებელ-ანალიტიკოსი საპრობლემო სფეროს, თავისი ბუნების მიხედვით, ხედავს როგორც მრავალგანზომილებიანს. შესაბამისად OLAP- მოდელიც უნდა იყოს მრავალგანზომილებიანი. ასეთი ტიპის კონცეპტუალური სქემა (მომხმარებელთა წარმოდგენები) აიოლებს მოდელირებას, ანალიზს და გამოთვლებს;

2) *გამჭვირვალეობა.* OLAP წარმოდგენილი უნდა იყოს ღია არქიტექტურის კონტექსტში, სადაც მომხმარებელს საშუალება ექნება ნებისმიერ დროს ანალიზური ინსტრუმენტის საშუალებით დაუკავშირდეს სერვერს საჭირო ინფორმაციის მისაღებად;

3) *წვდომადობა.* OLAP – ის მომხმარებელ ანალიზატორს უნდა ჰქონდეს ანალიზის ჩატარების საშუალება, რომელიც ემყარება საერთო კონცეპტუალურ სქემას. აქ განთავსებულია რელაციური მონაცემთა ბაზა საწარმოთა შესახებ არსებული ყველა ახალი და ძველი მონაცემებით. ეს ნიშნავს, რომ OLAP-მა უნდა წარმოადგინოს

თვისი საკუთარი ლოგიკური სქემა, რათა შეასრულოს შესაბამისი გარდაქმნა და მომხმარებელს წარუდგინოს მონაცემები. გარდა ამისა აუცილებელია წინასწარ იმაზე ზრუნვა, თუ სად, როგორ და როგორი სახის ფიზიკური ორგანიზაციის მონაცემები იქნას გამოყენებული. OLAP სისტემამ უნდა შეასრულოს ისეთი მონაცემების დამუშავება, რომელთა მოთხოვნაც რეალურად არსებობს;

4) *სტაბილური ანგარიშგების წარმადობა.* თუ ანალიტიკოსის მიერ ჩატარებული გაზომვათა რაოდენობა ან მონაცემთა ბაზების რიცხვი მნიშვნელოვნად იზრდება, მომხმარებელ ეს პროცესი უნდა დარჩეს შეუმჩნეველი და არ უნდა აისახებოდეს საწარმოო პროცესების წარმადობის შემცირებაზე;

5) *კლიენტ-სერვერული არქიტექტურა.* OLAP ინსტრუმენტების სერვერის კომპონენტი უნდა იყოს საკმარისად ინტელექტუალური, რომ კლიენტებმა შეძლონ მიერთება მინიმალური რესურსებით. სერვერს უნდა შეეძლოს განსხვავებული ტიპების მონაცემთა ასახვა და კონსოლიდაცია სხვადასხვა მონაცემთა ბაზებს შორის;

6) *ზოგადი განზომილება.* მონაცემთა ყველა განზომილება უნდა იყოს ეკვივალენტური თავისი სტრუქტურით და ოპერატიული შესაძლებლობებით;

7) *დინამიკური მართვა* გამონთავისუფლებული რეჟიმით. OLAP– ინსტრუმენტის ფიზიკური სქემა უნდა ადაპტირდებოდეს სპეციფიკურ ანალიტიკურ მოდელთან, რათა ოპტიმალურად მართოს გამონთავისუფლებული მატრიცა. ერთი ცარიელი მატრიცისთვის არსებობს ერთადერთი ოპტიმალური ფიზიკური სქემა. OLAP – ინსტრუმენტის ბაზური ფიზიკური მონაცემები პრაქტიკული ოპერაციებისთვის, რომელთაც აქვს დიდი ანალიზური მოდელი, უნდა კონფიგურირდებოდეს ნებისმიერ ქვესიმრავლესთან. თუ OLAP–ინსტრუმენტს არ შეუძლია გააკონტროლოს და დაარეგულიროს გასაანალიზებელი მონაცემების მნიშვნელობები, ის ჩაითვლება უსარგებლოდ და არასაიმედოდ;

8) *მრავალი მომხმარებლის მხარდაჭერა.* ხშირ შემთხვევაში მომხმარებელ-ანალიტიკოსი დასმულ მოთხოვნებს აყენებს ერთ ანალიზურ მოდელთან ან ქმნის განსხვავებულ მოდელს ერთი სახის მონაცემებიდან. OLAP ინსტრუმენტი უნდა უზრუნველყოფდეს ერთდროულად მონაცემთა მოძიებასა და განახლებას, წვდომას, მთლიანობას და უსაფრთხოებას;

9) *შეუზღუდავი გადამკვეთი ოპერაციები.* მონაცემთა შემოწმების სხვადასხვა დონე და გაერთიანების გზა, მათი იერარქიული ბუნების გათვალისწინებით მჭიდრო კავშირშია OLAP - მოდელთან ან დანართთან. თვითონ ინსტრუმენტი უნდა მოიაზრებოდეს შესაბამის გამოთვლებთან და არ უნდა მოსთხოვოს მომხმარებელს თავიდან განსაზღვროს გამოთვლები და ოპერაციები. გამოთვლები მოითხოვს რომელიმე გამოყენებულ ენაში განსხვავებული ფორმულების განსაზღვრას. ასეთი ენა შეიძლება გამოვიყენოთ ნებისმიერი სიდიდის მონაცემთა მანიპულირებისთვის და არ შეზღუდოს მონაცემები არსებული კუბის უჯრედებს შორის და კონკრეტული უჯრედების საერთო ატრიბუტებზე;

10) *მონაცემთა ინტუიციური მანიპულაცია.* მონაცემთა დეტალიზაციის, გაერთიანების და სხვა მანიპულაციები უნდა იყენებდეს ცალკეულ უჯრედებზე ანალიზური მოდელის შედეგებს და არ უნდა იყენებდეს მომხმარებლის ინტერფეისებს. მომხმარებელ ანალიტიკოსს უნდა ჰქონდეს ყველა აუცილებელი პირობა იმისა, რომ მიიღოს სრულყოფილი ინფორმაცია;

11) *ანგარიშების მიღების მოქნილი საშუალება.* შეტყობინებათა დამუშავება და პასუხის გაცემა უნდა იყოს მოქნილი და ელასტიური. მომხმარებელს უნდა შეეძლოს მონაცემთა კომბინირება და გაანალიზება. მოქნილობა მნიშვნელოვანია, რათა ყურადღება გამახვილდეს მონაცემთა განმასხვავებელ ნიშნებზე. ანუ სისტემის ანგარიშგების ინსტრუმენტულმა საშუალებებმა უნდა წარმოადგინოს ინფორმაცია ისე, როგორც მომხმარებელს სურს მისი ნახვა;

12) *შეუზღუდავი ზომები და აგრეგაციის დონეები.* მონაცემთა მხარდაჭერილ განზომილებათა რაოდენობა ყველა მიზნისთვის უნდა იყოს შეუზღუდავი. გამოკვლევებმა აჩვენეს, რომ აუცილებელი გაზომვა ერთდროულად შეიძლება ჩატარდეს 19-ჯერ. აქედან გამომდინარე შეიძლება ვთქვათ, რომ ანალიტიკური ინსტრუმენტი საშუალებას გვაძლევს ერთდროულად ვაწარმოოთ 15-დან 20-მდე გაზომვა, ამასთან თითოეული გაზომვის მცდელობა არ არის შემოსაზღვრული დადგენილი რიცხვით.

ეს პირობები შეიძლება ჩავთვალოთ OLAP-ის (ოპერატიული ანალიზური დამუშავების) თეორიულ ბაზისად. როგორც უკვე აღვნიშნეთ, OLAP-ში ჩადებულია მონაცემთა დამუშავების მრავალგანზომილებიანი სტრუქტურის იდეა. როდესაც ვსაუბრობთ მასზე, უნდა ვიგულისხმოთ, რომ ეს არის მონაცემთა ლოგიკური სტრუქტურის მრავალგანზომილებიანი ანალიზური ინსტრუმენტი.

❖ **მონაცემთა ბაზების სიტემა** (Databases System) არის მონაცემთა ორგანიზებული კოლექცია (ერთობლიობა), რომელიც ინახება კომპიუტერის დისკურ (გარე) მეხსიერებაში, კომპიუტერულ კლასტერებში ან ღრუბლოვან საცავებში.

იგი შედგება თვით მონაცემთა ბაზის (მბ – DB) და ამ მონაცემთა ბაზის მართვის სისტემისგან (მბმს – DBMS). ესაა პროგრამული უზრუნველყოფა, რომელიც აერთიანებს:

– *მონაცემთა აღწერის ენას* (DDL – Data Description [ან Definition] Language) – განსაზღვრავს მონაცემთა ტიპებს, ცხრილების შექმნას, მოდიფიკაციას და წაშლას, აგრეთვე ცხრილთაშორის კავშირებს;

– *მონაცემთა მანიპულირების ენა* (DML – Data Manipulation Language) – ასრულებს დავალებებს, როგორიცაა მონაცემთა ჩასმა, განახლება ან წაშლა;

– მონაცემთა მართვის ენა (DCL – Data Cntrol Language) – მართავს მონაცემებზე წვდომის პროცესს;

– მონაცემთა მოთხოვნების ენა (DQL – Data Query Language) – ახორციელებს ინფორმაციის მოძიებას და/ან მის გამოთვალას.

საბოლოო მომხმარებლებთან, პროგრამებთან და თავად მონაცემთა ბაზებთან ინფორმაციის შეგროვებისა და ანალიზისთვის. DBMS პროგრამული უზრუნველყოფა დამატებით მოიცავს მბ-ის ადმინისტრატორის ფუნქციებს და ინსტრუმენტებს.

შეიძლება ითქვას, რომ მონაცემთა ბაზების სისტემას (იერარქიული, ქსელური, რელაციური და სხვ.) საკმაოდ დიდი ისტორია აქვს [58]. განსაკუთრებით პრიორიტეტული გახდა (70-ანი წლების შემდეგ - დღემდე „ბატონობენ“) რელაციური ტიპის მონაცემთა ბაზები (მაგალითად, Oracle, Ms SQL Server, MySQL, PostgreSQL და სხვ.) [59].

ინტერნეტის განვითარებასთან ერთად დიდი გამოყენება ჰპოვა XML-ტიპის ბაზებმა, რომლებიც ინახება XML (Extended Markup Language) ენის ფორმატში და ადვილად ექსპორტირდება სხვა ტიპის ძირითად ბაზებში (და პირიქით). ასეთი მონაცემთა ტიპის ბაზების გადაცემა ინტერნეტ ქსელში ძალზე ხელსაყრელია.

და ბოლოს, 2000 წლის შემდეგ, კომპიუტერული ტექნოლოგიების განვითარებამ (ოპერატიული მეხსიერების მოცულობის უპრობლემო ზრდამ) დასაბამი მისცა ახალი ტიპის მონაცემთა ბაზების გამოსვლას ბაზარზე. ესენია NoSQL, NewSQL, გრაფიკული ბაზები და სხვ. [12, 54].

შემდეგ პარაგრაფში მოკლედ განვიხილავთ მონაცემთა ბაზების ასეთი განვითარების ტენდენციებს.

2.5.1. მონაცემთა ბაზის ტიპების განხილვა

პროგრამული ინდუსტრიის ბაზარზე არსებული მოთხოვნებიდან გამომდინარე, შეიძლება ჩამოვთვალოთ მონაცემთა ბაზების შემდეგი ტიპები: Centralised database; Distributed database; Personal database; End-user database; Commercial database; NoSQL database; Operational database; Relational database; Cloud database; Object-oriented database; Graph database და ა.შ. [60, 61].

❖ *ცენტრალიზებული მონაცემთა ბაზა (Centralised Database)*

ინფორმაცია ინახება ცენტრალიზებულად, ერთ ადგილას. სხვა ადგილას მყოფ მომხმარებლებს შეუძლიათ ამ მონაცემებზე წვდომა. მონაცემთა ბაზის ამ ტიპს აქვს აპლიკაციების პროცედურები, რომლებიც მომხმარებლებს დისტანციურად ეხმარება მონაცემებზე წვდომაში,

სხვადასხვა სახის ავტორიზაციის პროცედურები გამოიყენება საბოლოო მომხმარებლების გადამოწმებისა და ვალიდაციისთვის, ანალოგიურად სარეგისტრაციო ნომერი მოცემულია აპლოკაციის პროცედურებით, რომლებიც თვალყურს ადევნებს და აღრიცხავს მონაცემთა გამოყენებას.

❖ *განაწილებული მონაცემთა ბაზა* (Distributed Database)

განაწილებული მონაცემთა ბაზა მოთავსებულია ფიზიკურად (გოეგრაფიულად) სხვადასხვა ადგილას, მაგრამ იგი ლოგიკურად ერთი მთლიანი ბაზაა [58], ადგილობრივი კომპიუტერების მიერ აღებული ინფორმაცია. მონაცემები არ არის ერთ ადგილზე და გადანაწილებულია ორგანიზაციის სხვადასხვა საიტებზე. ეს საიტები უკავშირდება ერთმანეთს საკომუნიკაციო ბმულების საშუალებით, რაც მათ ეხმარება ადვილად მიიღონ განაწილებული მონაცემები. განაწილებული მონაცემთა ბაზა არსებობს ორი სახის, ანუ ერთგვაროვანი (Homogeneous) და არაერთგვაროვანი (Heterogeneous) [40].

❖ *პერსონალური მონაცემთა ბაზა* (Personal Database)

მონაცემები გროვდება და ინახება პერსონალურ კომპიუტერებზე, რაც მცირეა და ადვილად მართვადი. მონაცემებს ძირითადად იყენებს ერთი პიროვნება ან ორგანიზაციის ერთი დეპარტამენტის თანამშრომელთა მცირე ჯგუფი.

❖ *საბოლოო მომხმარებლის მონაცემთა ბაზა* (End User Database)

როგორც წესი, ამ ტიპის ბაზას იყენებენ არაინფორმატიკოსები. კერძოდ, ბიზნეს-პროფესიონალები ან სხვა ინდმომხმარებლები (ეკონომისტაები, ბუღალტრები, დეპარტამენტის თანამშრომლები). ეს არის საერთო მონაცემთა ბაზა, რომელიც სპეციალურად შექმნილია საბოლოო მომხმარებლისთვის. შესაბამისად საერთო გამოყენების მთლიანი, აგრეგირებული ინფორმაციაა ამ მონაცემთა ბაზაში.

❖ *კომერციული მონაცემთა ბაზა* (Commercial Database)

ესაა კომპანიის დიდი მონაცემთა ბაზის ფასიანი ვერსიები, რომლებიც შექმნილია მხოლოდ იმ მომხმარებლებისთვის, რომელთაც სურთ მათი ყიდვა. ეს ბაზები, როგორც წესი, უფრო მდიდარი და ძლიერია, ვიდრე უფასო ან ღია კოდის მონაცემთა ბაზები. მათი გამოყენება შეიძლება ბიზნესში, მთავრობასა და განათლებაში. იგი რეალიზებულია ხშირად ინტერნეტის სერვისის სახით.

❖ არარელაციური ბაზები (NoSQL Databases, Json ფორმატით)

შესაბამისი მონაცემთა ბაზის ტიპები გამოიყენება განაწილებული მონაცემების მომხმარებელთა დიდი ჯგუფებისთვის. არსებობს დიდ მონაცემთა დამუშავების ამოცანები, რომლებსაც ეფექტურად ვერ ამუშავებს რელაციური მონაცემთა ბაზები, ისინი უფრო ადვილად და სწრაფად იმართება NoSQL ბაზებით [33, 54]. ისინი ძალზე ეფექტურია დიდი ზომის არასტრუქტურიზებული მონაცემების ანალიზისას, რომლებიც შეიძლება ინახებოდეს Cloud-ის მრავალ ვირტუალურ სერვერზე.

❖ *ოპერატიული მონაცემთა ბაზა* (Operational Database)

ესაა საოპერაციო მონაცემთა ბაზა, სადაც ინახება ორგანიზაციის ყოველდღიურ საქმიანობასთან დაკავშირებული ოპერაციების შესახებ ინფორმაცია. მაგალითად, ისეთი ფუნქციური ამოცანები, როგორიცაა კლიენტების შეკვეთები, ინვენტარის დონეები და ფინანსური ტრანზაქციები. ისინი ხშირად საჭიროებს მონაცემებზე სწრაფ წვდომას და მონაცემთა *რეალურ დროში* განახლების შესაძლებლობას.

❖ *რელაციური მონაცემთა ბაზები* (Relational Databases)

40 წელია გამოიყენება და დღესაც ძირითადი ბაზებია (Oracle, SQL Server, MySQL და სხვ.) [54, 58, 62]. ამ მონაცემთა ბაზების კატეგორიზაცია ხდება ცხრილების მიხედვით, სადაც მონაცემები თავსდება წინასწარ განსაზღვრულ ადგილას. ცხრილი შედგება სტრიქონებისა და სვეტებისგან. ყველა რელაციური ბაზა იყენებს SQL (Structured Query Language)-ს, მოთხოვნების დამუშავებისთვის. რელაციური მონაცემთა ბაზის შექმნისას დიზაინერი გადის გარკვეულ ეტაპებს, ერთ-ერთ მათგანია მონაცემთა მოდელირება. მოდელირების ფაზა განსაზღვრავს ობიექტებს (Entities), თითოეული ობიექტის ატრიბუტებს (Attributes) და სხვადასხვა ობიექტებს შორის კავშირებს, რომლებიც უნდა არსებობდეს მონაცემთა ბაზაში. ამ საკითხებს დეტალურად განვიხილავთ მომდევნო პარაგრაფებში.

❖ *ღრუბლოვანი მონაცემთა ბაზები* (Cloud Databases)

დღეს აქტუალური და ძალზე პოპულარულია (ეკონომიკურადაც მისაღები) მონაცემების შენახვა „ღრუბლებში“ (Cloud-ზე), რომლიც ასევე ცნობილია როგორც ვირტუალური გარემო. Cloud მონაცემთა ბაზა ოპტიმიზირებულია ან აგებულია ასეთი ვირტუალიზებული გარემოსთვის [63, 64]. Cloud მონაცემთა ბაზის სხვადასხვა უპირატესობა არსებობს, რომელთაგან ზოგიერთი

შენახვის მოცულობისა და გამტარუნარიანობის საფასურის გადახდაა ერთ მომხმარებელზე დაყრდნობით და უზრუნველყოფს მოთხოვნის მასშტაბურობას, მაღალ წვდომადობასთან ერთად.

მაგალითად, Azure SQL მონაცემთა ბაზა ასეთი ტიპისაა. იგი რეალიზაებულია სერვისის სახით (DaaS - Desktop as a Service), რომელსაც გვთავაზობს Azure. ეს საშუალებას იძლევა განხორციელდეს SQL მონაცემთა ბაზა პროგრამული უზრუნველყოფის ან აპარატურის ინსტალაციის გარეშე. შესაძლებელია სარეზერვო ასლის გრძელვადიანი შენახვა (10 წლამდე). ღრუბლოვანი პროვაიდერები ავრცელებენ და მართავენ ვირტუალურ სამუშაო მაგიდას საკუთარი მონაცემთა ცენტრიდან.

❖ *ობიექტ-ორიენტირებული ბაზები* (Object-Oriented Databases)

ასეთ მონაცემთა ბაზაში მონაცემები ორგანიზებული ავტონომიური ობიექტების სახით, რომლებიც შეიცავს როგორც მონაცემებს, ასევე მეთოდებს [33], ამ მონაცემებთან სამუშაოდ. ობიექტ-ორიენტირებული მონაცემთა ბაზები ემსახურება მონაცემთა რთული სტრუქტურების შექმნის და მართვის მხარდაჭერას. ისინი ხშირად გამოიყენება აპლიკაციებში, რომლებიც საჭიროებენ დიდი რაოდენობით სტრუქტურირებული და ნახევრად სტრუქტურირებული მონაცემების მანიპულირებას.

❖ *მულტიმედიური მონაცემთა ბაზა* (Multimedia Database)

იგი განვითარდა ობიექტ-ორიენტირებული ბაზების საფუძველზე. აქვს მულტიმედიურ მონაცემთა ერთ ან მეტი ტიპი (მაგალითად, ტექსტი, სურათები, გრაფიკული ობიექტები, აუდიო, ვიდეო და ანიმაციური). მონაცემთა ეს ტიპები Text, Image, ასევე Graphics, Audio და Video განიხილება როგორც MediaObject-ის ქვეკლასები, და მემკვიდრეობით ღებულობს მის მეთოდებს. ისინი აფართოებს ამ მეთოდებს საკუთარი ოპერაციებით, რომლებიც

მორგებულია სპეციალურ მულტიმედიურ მონაცემთა ტიპებზე, მაგალითად, image-ს დროს height, ან Text-ის დროს length [65, 66].

❖ *მულტიმოდალური მონაცემთა ბაზა* (Multimodal Database)

ასეთ ბაზას შეუძლია მონაცემთა სხვადასხვა მოდელის მხარდაჭერა, მათ შორის რელაციურის, ობიექტზე ორიენტირებულის, „გასაღები-მნიშვნელობა“ („key : value“) მოდელის, ფართო სვეტის მოდელების, დოკუმენტების და გრაფული მოდელების. ამ სახის ბაზას შეუძლია სტრუქტურირებული, არასტრუქტურირებული და ნახევრადსტრუქტურირებული მონაცემთა შენახვა და მართვა, მულტიმოდალური მონაცემთა ბაზები ხშირად გამოიყენება აპლიკაციებში, რომლებიც საჭიროებს მონაცემთა ინტეგრაციას სხვადასხვა წყაროდან, როგორიცაა ტექსტური დოკუმენტები, სურათები და სოციალური მედიის მონაცემები [61].

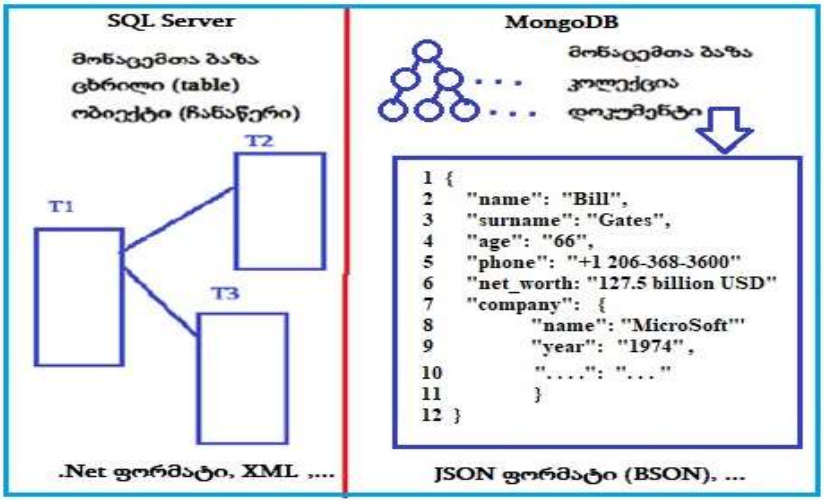
❖ *გრაფული მონაცემთა ბაზები* (Graph Databases)

გრაფი არის წვეროების და წიბოების კრებული, სადაც თითოეული წვერო გამოიყენება *ობიექტის* წარმოსადგენად, ხოლო თითოეული წიბო აღწერს ობიექტებს შორის *კავშირს (პრედიკატს)* [33]. გრაფულ-ორიენტირებული მონაცემთა ბაზა, ან გრაფული მონაცემთა ბაზა, NoSQL მონაცემთა ბაზის ტიპია, რომელიც იყენებს გრაფთა თეორიას. გრაფული მონაცემთა ბაზა ძირითადად გამოიყენება ურთიერთკავშირის ანალიზისთვის. მაგალითად, კომპანიებმა შეიძლება გამოიყენონ გრაფული მონაცემთა ბაზა სოციალური მედიიდან მომხმარებლების შესახებ მონაცემების შესაგროვებლად.

შემდგომში ჩვენ განვიხილავთ დეტალურად „ობიექტ-როლურ მოდელების“ (ORM) სტრუქტურას, სადაც კარგად ჩანს გრაფული ტიპის ბაზის კონცეპტუალური სქემის იდეა [67].

❖ *ძირითადი განსხვავება SQL და NoSQL ტიპის ბაზებს შორის*

როგორც აღვნიშნეთ, არსებობს მონაცემთა ბაზების სხვადასხვა ტიპები და პაკეტები, როგორიცაა კლასიკური რელაციური (SQL-ტიპის) და არარელაციური (NoSQL-ტიპის). მათ შორის არის საკმაო განსხვავება ტერმინოლოგიის, შინაგანი სტრუქტურის და გამოყენების თვალსაზრისით [33]. ამის საილუსტრაციოდ წარმოდგენილია 2.19 ნახაზი [68].



ნახ.2.19. ცხრილური და იერარქიული მოდელების განსხვავება

დიდი მონაცემებისა და ღრუბლოვანი ტექნოლოგიების აპლიკაციების თვალსაზრისით და განვითარების დინამიკის გათვალისწინებით, უპირატესობა NoSQL-ის მხარეზეა, თუმცა რელაციური ბაზები ინარჩუნებს პრივილეგიებს კორპორაციულ სისტემებში. უმეტეს შემთხვევაში მსოფლიოს პრიორიტეტული კორპორაციები და ორგანიზაციები მონაცემთა ბაზების ორივე ტიპის აქტიურად გამოყენებას უჭერს მხარს (გადასაწყვეტი ამოცანების მიზნებიდან გამომდინარე).

2.5.2. არსებული მონაცემთა ბაზების დადებითი და უარყოფითი მხარეების ანალიზი

დღეისათვის არსებობს რამდენიმე ყველაზე ცნობილი და ფართოდ გამოყენებადი მონაცემთა ბაზა.

❖ *რელაციური ბაზებიდან* გამოვარჩევთ ოთხ მათგანს: Ms SQL server, Oracle, MySQL და PostgreSQL (ნახ.2.20). მოკლედ შევეხებით ზოგადად მათ დადებით და უარყოფით მხარეებს, არსებული განსხვავებებს [50-67].

ნახ.2.20. არჩეული რელაციური ბაზები

➢ **Oracle**

დადებითი მხარეები :

- საიმედოობა;
- უსაფრთხოება;
- ფუნქციონალურობა;
- კარგი მომსახურება;
- დიდი მონაცემების მართვა.

უარყოფითი მხარეები :

- ღირებულება;
- საჭიროებს დიდი ზომის მეხსიერებას.

იდეალურია მსხვილი ორგანიზაციებისთვის, რომლებიც დიდ მონაცემთა ბაზებს უმკლავდებიან და სჭირდებათ სხვადასხვა ფუნქციონალი.

- **MS SQL Server**

დადებითი მხარეები :

- სიმარტივე;
- უსაფრთხოება;
- სისწრაფე;
- .Net მხარდაჭერა;
- Microsoft-ის სხვა პროდუქტებთან მარტივი მუშაობა;
- ვერსიებს შორის სტაბილურობა;
- გრაფიკული query-ის ანალიზატორი;
- დამხმარე პროგრამები (Profiler, BI Tools და ა.შ.)

უარყოფითი მხარეები:

- რეგულირება (აწყობა -Tuning);
- ფასი;
- Microsoft-ზე დამოკიდებული.

იდეალურია მსხვილი ორგანიზაციებისთვის, რომლებიც იყენებენ Microsoft-ის პროდუქტებს.

- **MySQL**

დადებითი მხარეები:

- open source;
- მარტივი;
- დიდი ფუნქციონალი;
- სისწრაფე;
- განსხვავებული რილისები;
- NoSQL საპორტი;
- ბევრი მომხმარებელი;
- თავსებადია ყველა ჰოსტინგის პროვაიდერთან;
- დამოუკიდებელი პლატფორმა;

- მასობრივი მონაცემების მანიპულირება;
- დიდი გამოცდილება და ინტერნეტში მრავალი მასალის მოძებნის საშუალება.

უარყოფითი მხარეები:

- არასტაბილურობა ვერსიებს შორის;
- ნაკლები დამხმარე ინსტრუმენტები;
- შეზღუდული ფუნქციონალი.

იდეალურია ორგანიზაციებისთვის, რომელთაც სჭირდებათ ძლიერი მონაცემთა ბაზების მართვის ინსტრუმენტი, მაგრამ აქვთ შეზღუდული ბიუჯეტი.

- **PostgreSQL**

დადებითი მხარეები :

- open source;
- საიმედოობა და სტაბილურობა (საიმედო კომპანიები ამბობენ, რომ დიდი დატვირთვის დროსაც კი, არასდროს არ გათიშვიათ ბაზა, მრავალწლიანი მუშაობის განმავლობაში);
- მხარს უჭერს JSON-ს'
- არსებობს სხვადასხვა წინასწარგანსაზღვრული ფუნქციები;
- მრავალი ინტერფეისია ხელმისაწვდომი;
- განკუთვნილია მაღალი მოცულობის გარემოსთვის.

უარყოფითი მხარეები :

- არასრული დოკუმენტაცია;
- კონფიგურაცია შეიძლება დამაბნეველი იყოს;
- შესაძლებელია სიჩქარე დავარდეს მრავალი ოპერაციის ერთად შესრულების შემთხვევაში;
- უფრო ნელია ვიდრე MySQL;
- რეპლიკაციები.

იდეალურია ორგანიზაციებისთვის, რომელთაც აქვთ შეზღუდული ბიუჯეტი და რომელთაც სურთ ჰქონდეთ ინტერფეისის არჩევის და JSON-ის გამოყენების შესაძლებლობა.

- **MS SQL vs Oracle**

- Oracle ანალიტიკის მხრივ არის შედარებით ცუდი ბაზისთვის, სადაც არის ძალიან დიდი მონაცემები, რადგან სჭირდება გადაერთოს PL / SQL-ზე. ამიტომ თუ ჩვენ გვაქვს რამდენიმე რთული ფუნქცია, რომელთა გაშვებაც გვინდა ჩვენ მონაცემებზე, სჯობს არ გამოვიყენოთ Oracle. მეორეს მხრივ, Ms SQL არის ძალიან კარგი ამ დროს, რადგან მას არ სჭირდება PL / SQL -ზე გადართვა და ასევე აქვს ინტეგრაცია .NET-ზე.

- MS SQL-ი თავსებადია, მხოლოდ Linux-თან და Windows-თან, ხოლო Oracle კი ნებისმიერ ოპერაციულ სისტემასთან.

სხვა განსხვავებანი:

1) Oracle იყენებს PL/SQL-ს, MS SQL იყენებს T-SQL-ს;

2) Oracle-ში შესაძლებელია პროცედურების დაჯგუფება და ა.შ. პაკეტებში, MS SQL-ში კი არ არსებობს პაკეტების ცნება;

3) PL/SQL უფრო კომპლექსური და მძლავრია, T-SQL უფრო მარტივი;

4) Oracle-ში ყოველი ბრძანება განიხილება როგორც ტრანზაქცია. MS SQL-ში ოპერატორების ჯგუფი შეიძლება განიხილებოდეს როგორც ტრანზაქცია;

5) Oracle-ში არაფერი არ ინახება მეხსიერებაში, თუ არ დაკომიტდა. Ms SQL - ში დაკომიტება არ არის აუცილებელი, ისედაც იგულისხმება.

6) Oracle-ში ნაკლებია შეცდომის დაშვების და მონაცემების დაზიანების ალბათობა. ვიდრე Ms SQL -ში;

7) სინტაქსი განსხვავდება (მაგალითად, begin statement rollback განსხვავებულია);

8) Oracle - ში access არის სქემის დონეზე. Ms SQL -ში access არის მონაცემთა ბაზის დონეზე;

9) Oracle - ში ყველა მონაცემთა ბაზა საერთოა, გაზიარებულია ყველა სქემისთვის და მომხმარებლისთვის, მაგრამ access კონტროლირებადია. Ms SQL - ში საერთოდ არ არის სქემის ცნება. მომხმარებლებს ენიჭებათ access პირდაპირ ბაზაზე;

10) Oracle - ს აქვს უკეთესი, უფრო დოკუმენტირებული პარალელიზმის მოდელი ვიდრე Ms SQL - ს;

11) Oracle - ის შემთხვევაში .NET ინტეგრაცია არ არის ისეთი მარტივი, როგორც Ms SQL - ის შემთხვევაში;

12) Oracle იყენებს sequences-ს, ხოლო Ms SQL – auto increment-ს;

13) Oracle -ში დაცვა არის მაღალ დონეზე, ვიდრე MS SQL -ში;

14) Oracle - უფრო გამოდგება რეალურად დიდი აპლიკაციებისთვის, ვიდრე Ms SQL. მაგრამ თუ რეალურად დიდი აპლიკაცია არ არის საჭირო, მაშინ Ms SQL-თან მუშაობა უფრო მარტივია.

- **Oracle vs MySQL**

1) MySQL უფასოა, Oracle კი ფასიანი (და საკმაოდ ძვირი);

2) Oracle მუშაობს შემდეგ ოპერაციულ სისტემებთან : Windows, Mac OS X, Linux, UNIX, z/OS. ხოლო MySQL Windows, Mac OS X, Linux, UNIX, z/OS, BSD, Symbian, AmigaOS;

3) Oracle იყენებს შემდეგ ინდექსებს : Full-text, Hash, R-/R+ Tree, Bitmap, Expression, Partial and Reverse. MySQL კი - Full-text, Hash, R-/R+ Tree;

4) MySQL უზრუნველყოფს საიტის და ტელეფონის მხარდაჭერას, ხოლო Oracle მხოლოდ უზრუნველყოფს ფორუმების მხარდაჭერას;

5) Oracle მუშაობს როგორც დინამიკურ, ისე სტატიკურ სისტემებთან, მაშინ როცა MySQL მუშაობს მხოლოს სტატიკურთან;

6) Oracle-ს მეტი ფუნქციონალი აქვს, ვიდრე MySQL-ს.

- **Ms SQL vs MySQL**

1) MySQL თავდაპირველად მუშაობს PHP -სთან, ხოლო MS SQL .NET-თან;

2) Ms SQL-ის ლიცენზია ფასიანია, ხოლო MySQL უფასოა, მაგრამ თუ რაიმე მხარდაჭერაა საჭირო, მაშინ support-ის მისაღებად საჭიროა შესაბამისი ღირებულების გადახდა;

3) Storage engine – Ms SQL < MySQL;

4) MySQL-ს აქვს შენახვის რამდენიმე მექანიზმი, რომელიც ხდის მას მეტად მოქნილს, ვიდრე არის Ms SQL;

5) Filtering - Ms SQL > MySQL;

6) MySQL -ის შემთხვევაში უნდა გავფილტროთ ცხრილები ინდივიდუალურად. რამდენიმე query-ს გაშვებით, ხოლო Ms SQL-ს აქვს სტრიქონზე დაფუძვნებული ფილტრაცია, რაც რა თქმა უნდა უმჯობესია;

7) Backup - Ms SQL > MySQL;

8) Ms SQL, MySQL-სგან განსხვავებით, არ ბლოკავს მონაცემთა ბაზას მონაცემების რეზერვაციის დროს;

9) Stop Query Execution - Ms SQL > MySQL;

10) MySQL არ აძლევს მომხმარებელს query-ის გაჩერების უფლებას, როცა გაშვებულია;

11) Security - Ms SQL > MySQL;

დამუშავების წესი ხდის Ms SQL Server -ს უფრო უსაფრთხოს, ვიდრე MySQL-ს.

- **Oracle vs PostgreSQL**

1. PostgreSQL უფასოა, Oracle - არა;

2. მომსახურება (support) PostgreSQL-ზე უფასოა, მაგრამ დრო დასჭირდება, სანამ პროგრამისტების ჯგუფი მოაგვარებს პრობლე-მას. ასევე შესაძლებელია პროფესიონალების მომსახურების მიღება,

რაც შესაბამისად ფასიანი იქნება, მაგრამ მაინც ბევრად იაფი, ვიდრე Oracle-ის მომსახურება;

3. პროდუქტიულობა Oracle-ს უფრო მაღალი აქვს. Oracle ასრულებს მეტ ტრანზაქციას წამში, ვიდრე PostgreSQL.

4. Oracle უფრო უსაფრთხოა, ვიდრე PostgreSQL;

5. Oracle დიდი მოცულობის მონაცემებს უფრო ეფექტურად ამუშავებს, ვიდრე PostgreSQL.

❖ არარელაციური ბაზებიდან განვიხილოთ NoSQL ტიპ;ის: MongoDB Compass და Atlas, Redis, neo4j, OrientDB და Apache Cassandra (ნახ.2.21) [33].

ნახ.2.21. არჩეული NoSQL ბაზები

NoSQL მონაცემთა ბაზებს ზოგადად აქვს შემდეგი თვისებები:

- მაღალი მასშტაბურობა (scalable);
- განაწილებული გამოთვლების გამოყენება;

- ეფექტუროობა; მაღალი მწარმოებლუროობა, სწრაფი შესრულება (ოპერატიულ მეხსიერებაში მუშაობის ხარჯზე);
- მოქნილობა (Flexibility): მხარს უჭერს დინამიკურ სქემატურ არქიტექტურას;
- შეუძლია როგორც არასტრუქტურირებული, ისე ნახევრად სტრუქტურირებული მონაცემების დამუშავება;
- არ არსებობს რთული რელაციური კავშირები, როგორც ეს ცხრილებს შორისაა RDB ბაზებში.

➢ **MongoDB**

იგი დოკუმენტ-ორიენტირებული მონაცემთა ბაზაა (***დომბ***), გამოიყენება შინაარსის მართვის სისტემებში (CMS-Content Management System), საგამომცემლო საქმეში, დოკუმენტების საძიებო სისტემებში და სხვ. ***დომბ***-ის მთავარი ცნებაა „დოკუმენტი“, რომელიც განისაზღვრება როგორც მონაცემთა ინკაფსულაცია ინფორმაციის კოდირების სტანდარტული ფორმატებისა და მეთოდების გამოყენების საფუძველზე. ასეთი ფორმატებია: XML, JSON, BSON, YAML. ზოგ შემთხვევაში შესაძლებელია PDF, Ms Office და მსგავსი დოკუმენტების ბინარული ფორმატით შენახვაც.

MongoDB Compass მბ-ის პაკეტის მომხმარებელს შეუძლია მუშაობა Mongo_Compass და Mongo_Shell ინტერფეისებით [68]. მაღალი სიჩქარე/შესრულება: MongoDB-ის დოკუმენტზე ორიენტირებული ბუნება საშუალებას აძლევს მას შესთავაზოს უფრო მაღალი სიჩქარე, ვიდრე ტრადიციული რელაციური მონაცემთა ბაზები. მარტივი სამუშაო გარემო და სწრაფი კონფიგურაცია, მოქნილობა და მაღალი მასშტაბირება (Sharding-ის გამოყენებით, ესაა ჰორიზონტალური მასშტაბირება, ანუ თუ მონაცემთა დიდი ნაკრებია, ის ნაწილდება სხვადასხვა კომპიუტერულ კლასტერებში.

საბოლოო ჯამში, იზრდება შენახვის მოცულობა). MongoDB იყენებს ქეშს და RAM-ს, რაც უზრუნველყოფს უფრო სწრაფ მოძიებას და უკეთეს შესრულებას. მონაცემებზე სწრაფი წვდომადობა (Availability) ხორციელდება მათი დამატებითი ასლების შექმნით სერვერზე (replication).

უარყოფითი მაჩვენებელია მონაცემთა მაღალი დუბლირება, რაც ართულებს მონაცემთა ეფექტურად მართვას. მოითხოვს ოპერატიული მეხსიერების საკმაოდ დიდ მოცულობას.

MongoDB Atlas არის წამყვანი გლობალური *ღრუბლოვანი მონაცემთა ბაზის სერვისი* თანამედროვე აპლიკაციებისთვის. Atlas-ის გამოყენებით, დეველოპერებს შეუძლიათ განათავსონ სრულად მართული ღრუბლოვანი მონაცემთა ბაზები AWS, Azure და Google Cloud-ში. აქვს ამ ტიპის ბაზებში მონაცემთა უსაფრთხოებისა და კონფიდენციალურობის სტანდარტების საუკეთესო პრაქტიკა, რაც ნიშნავს, რომ დეველოპერებს შეუძლიათ მყისიერი წვდომა მონაცემებზე, მასშტაბირებაზე და შესაბამისობა ვალიდურობაზე, რომელიც მათ სჭირდებათ კორპორაციული დონის აპლიკაციის განვითარებისთვის.

MongoDB Atlas აადვილებს მონაცემთა ბაზის მასშტაბირებას, უზრუნველყოფს მონაცემთა ბაზის მაჩვენებლების და მწარმოებლურობის ოპტიმიზაციის ინსტრუმენტების ხილვადობას რეალურ დროში, აგრეთვე დეველოპერების არსებულ რესურსებს მონაცემთა ბაზის მაჩვენებლების შესაბამისად.

ექსპერტების აზრით, კარგი იქნება, თუ მომდევნო ვერსიებისთვის უფრო ადვილი გახდება MongoDB Atlas-ის ინტეგრირება AWS სერვისებთან. ტექნიკური მხარდაჭერაც MongoDB Atlas-ისთვის მომავალში შეიძლება უკეთესი გახდეს. ამჟამად არაა შექმნილი ისეთი ინტერფეისი, რომელიც არაპროფესიონალ მომხმარებელს მიცემს მონაცემთა ბაზის მარტივად განახლების საშუალებას [].

- **Neo4j –** არის NoSql ტიპის გრაფული ბაზა

კომპანია Neo Technology-ის პროდუქტი, დამუშავებულია java ენაზე და ერთ-ერთი ყველაზე პოპულარული გრაფული ბაზაა [69]. აპლიკაციების დაპროგრამების ინტერფეისი მონაცემთა ბაზისთვის რეალიზებულია მრავალი ენისთვის, მათ შორის: Java, Python, Ruby, PHP და სხვ.

აქვს მარტივი, მოქნილი და ძლიერი მონაცემთა მოდელი. მისი მარტივად მორგება შესაძლებელია აპლიკაციისა და ბიზნესის მოთხოვნების შესაბამისად. იგი იძლევა შედეგებს მონაცემებზე დაყრდნობით რეალურ დროში. მონაცემთა კონფიდენციალურობა - რეალიზებულია მომხმარებელთა პერსონალური მონაცემების დონეზე. შეესაბამება მონაცემთა დაცვის საერთაშორისო კანონებს და რეგულაციებს.

Neo4j-ს აქვს დიაგრამის ზომის ზედა ზღვარი და შეუძლია ათობით მილიარდი კვანძის, თვისებისა და კავშირის მხარდაჭერა ერთ გრაფიკში. არ აქვს უსაფრთხოება უზრუნველყოფა მონაცემთა დონეზე, არხდება მონაცემთა დაშიფვრა. უსაფრთხოების აუდიტი მიუწვდომელია Neo4j-ში.

- **OrientDB –** გრაფული- და დოკ.-ორიენტირებული ბაზაა

დამუშავებულია Java ენაზე Orient Technologies LTD ფირმის მიერ Windows, Linux, Mac და სხვა ოპერაციული სისტემებისთვის [70]. მოთხოვნების ენისათვის აქვს SQL-ის მხარდაჭერა (ამიტომაც მას NewSQL ბაზასაც მიაკუთვნებენ). იგი არ იყენებს JOIN ოპერაციას. მის მაგივრად აქვს სუპერ-სწრაფი მუდმივი მიმთითებლები ჩანაწერებს შორის, რომლებიც გრაფული ბაზებისთვისაა დამახასიათებელი. ეს უზრუნველყოფს ჩანაწერების ცალკეული ან მთლიანი ხეების და გრაფების გადასინჯვას რამდენიმე მილიწამის ფარგლებში. OrientDB იყენებს უსაფრთხოების მოდელს, რომელიც დაფუძნებულია მომხმარებლებისა და როლების კონცეფციებზე.

ანუ მონაცემთა ბაზას ჰყავს თავისი მომხმარებლები. თითოეულ მომხმარებელს აქვს ერთი ან მეტი როლი.

OrientDB-ის დადებითი მხარე მდგომარეობს იმაში, რომ ის არის მოქნილი და ძლიერი ოპერაციული მბმს. OrientDB უზრუნველყოფს მოქნილ ხაზოვან მასშტაბურობას და ძალიან კარგად შეეფერება ღრუბლოვან გარემოს.

ნაკლოვანებათა სახით, რომელიც OrientDB-ის ექსპერტებისა და მომხმარებელთა მიერ იქნა დაფისირებული შემდეგია:

- ხშირარდ არაადეკვატური და გაფანტულია გამოსაყენებელი დოკუმენტაცია;
- რთულია მასშტაბირების და მისი წარმოებაში გამოყენების პროცესის რეალიზაცია. ეს პრობლემა, სავარაუდოდ, კავშირშია გრაფში პარალელიზმის დამუშავების სირთულესთან, აგრეთვე თავსებადობის საკითხებთან, რომლებიც წარმოიქმნება სხვადასხვა რეჟიმების შერევისას, როგორიცაა დოკუმენტი, გრაფი, SQL ან Tinkerpop. (Apache TinkerPop არის გრაფული გამოთვლების პლატფორმაა - გრაფული მონაცემთა ბაზებისთვის (OLTP) და გრაფების ანალიზისთვის (OLAP) [71]. როდესაც მონაცემთა სისტემა მხარს უჭერს TinkerPop-ს, მაშინ მის მომხმარებელს შეუძლია თავისი პრობლემური სფეროს (დომაინის) მოდელირება გრაფის სახით და შემდეგ მისი ანალიზის ჩატარება [72]) .
- მწარმოებლურობის (Performance) პრობლემა. OrientDB-ში მომუშავე ზოგიერთ მომხმარებელს უწევს დროის ხარჯვა დამატებით ოპერაციებზე, როგორიცაა ფაილების დაყოფა, ან ფაილების გახსნა-დახურვის ორგანიზება. აგრეთვე, მოთხოვნები-სათვის, რომლებიც იყენებს გრაფს, კვანძებისა და კიდეების დიდი რაოდენობის მქონეს, შესრულებისათვის დასჭირდება ხანგრძლივი დრო, რაც ამცირებს სისტემის მწარმოებლურობას.

- **Redis –** არის NoSQL ტიპის მონაცემთა ბაზა

იგი ღია კოდის მეხსიერების საცავია, რომელიც გამოიყენება როგორც მეხსიერებაში „key–value"-ით განაწილებული მონაცემთა ბაზა, იმის გამო, რომ ის ინახავს ყველა მონაცემს მეხსიერებაში და მისი დიზაინის გამო, Redis გვთავაზობს კითხვა-ჩაწერის ოპერაციებს მცირე დროში, რაც მას განსაკუთრებით შესაფერისს ხდის მათთვის, ვისაც სჭირდება ქეშის გამოყენება. Redis იყენებს Twitter, Yahoo, Adobe, Amazon და სხვ. კომპანიები [73].

Redis-ში რეალიზებულია სხვადასხვა სახის აბსტრაქტულ მონაცემთა სტრუქტურები: სტრიქონები, სიები, რუქები, კომპლექ-ტები, ბიტმაპები, ნაკადები და სივრცითი ინდექსები.

იგი ხშირად გამოიყენება ვებ გვერდების ქეშირებისთვის და სერვერის დატვირთვის შესამცირებლად. Redis-ს ასევე აქვს რამდენიმე ფუნქცია, რაც მიმზიდველს ხდის მას მონაცემთა ბაზად გამოყენებას, როგორიცაა ტრანზაქციების მხარდაჭერა და შეტყობინებების გამოქვეყნება/გამოწერა.

Redis-ის ნაკლოვანებებია [74]:

- შეზღუდული დისკური მეხსიერება, ACID ტრანზაქციების ნაკლებობა;
- მონაცემთა პოტენციური დაკარგვა ავარიის შემთხვევაში;
- არ აქვს მოთხოვნების ენა (მხოლოდ ბრძანებებია);
- ყველა მონაცემზე წვდომა უნდა იყოს გათვალისწინებული დეველოპერის მიერ და უნდა იყოს შემუშავებული მონაცემთა წვდომის სათანადო ბილიკები. ამის გამო სისტემის მოქნილობა დაქვეიტებულია.

- **Apache Cassandra** - NoSQL ტიპის მონაცემთა ბაზა

Cassandra არის NoSQL მონაცემთა ბაზის მძლავრი სისტემა, რომელიც გვთავაზობს: მასშტაბურობას, მაღალ წვდომადობას და მტყუნებებზე (შეცდომებზე) მდგრადობას.

იგი რთული სისტემაა, რომელიც მოითხოვს ფრთხილად (საფუძვლიანად) დაგეგმვას და კონფიგურირებას ოპტიმალური მუშაობის უზრუნველსაყოფად.

Apache Cassandra-ს უპირატესობები:

მას შეუძლია ეფექტურად და ეფექტურად გაუმკლავდეს უზარმაზარ რაოდენობას მრავალ სერვერზე. გარდა ამისა, მას შეუძლია სწრაფად ჩაწეროს დიდი რაოდენობით მონაცემები წაკითხვის ეფექტურობაზე გავლენის გარეშე. Cassandra მომხმარებლებს სთავაზობს „საოცრად სწრაფ წერას“ და სიჩქარეზე ან სიზუსტეზე გავლენას არ ახდენს დიდი მოცულობის მონაცემები.

Cassandra-ს უარყოფითი მხარეები:

- ერთი და იგივე მონაცემები შეიძლება შენახული იყოს რამდენჯერმე, რადგან არ გამოყენება სქემები/დამოკიდებულებები;
- მონაცემთა წაკითხვა ნელია, რადგან Cassandra ოპტიმიზებულია უფრო სწრაფი ჩაწერისთვის;
- Apache-ს დოკუმენტების ნაკლებობა - მომხმარებელს უხდება სხვა კომპანიების დოკუმენტების მოძებნა.

2.5.3. Ms SQL Server-ის მონაცემთა უსაფრთხოება

ჩვენი წიგნის ორიგინალურ ნაწილში, სადაც შემუშავებული გვაქვს საგანმანათლებლო დაწესებულების მენეჯმენტის პროცესების ავტომატიზაციის სპეციალური ალგორითმები და პროგრამები, ძირითადად ვიყენებთ რელაციურ მონაცემთა Ms SQL Server ბაზას, ხოლო რიგი საკითხებისთვის – NoSQL-ს, ამიტომ აქ დამატებით შევეხებით SQL-ის მონაცემების უსაფრთხოების ასპექტებს.

ორგანიზაციას, რომელსაც სურს შეინახოს დიდი მოცულობის მონაცემები, მისთვის ერთ-ერთი საუკეთესო გადაწყვეტაა

Ms SQL Server-ი. გარდა ამისა მას აქვს მონაცემთა დაცვის კარგი მექანიზმებიც.

ზოგადად, იხილევენ სისტემის ოთხ ძირითად ასპექტს: მწარმოებლურობას, წვდომადობა, მასშტაბირებას და უსაფრთხოებას. ვინაიდან აღნიშნულ ასპექტებს ჩვენ წინა პარაგრაფში შევეხეთ, ამჯერად უსაფრთხოების საკითხს შევეხებით მონაცემთა ბაზის ახალ ვერსიაში.

უსაფრთხოება არის ერთ-ერთი ისეთი სფერო, სადაც SQL Server-ის მომხმარებლები ელოდებიან საიმედო და მუდმივ განვითარებად სრულყოფილებას. Microsoft უწყვეტად აგრძელებს მომხმარებლის კონფიდენციალური ინფორმაციის დაცვას. ძალზე მნიშვნელოვანია მომხმარებელმა თავი იგრძნოს დაცულად.

SQL Server 2016 -ში გამოჩნდა უსაფრთხოების რამდენიმე ინოვაცია. როგორც მისი სახელწოდებიდან უნდა გამომდინარეობდეს, Always Encrypted ამატებს უნიკალურ შესაძლებლობას მონაცემების დაშიფვრის, შენახვისა და გამოყენების დროს და ასევე შიფრაციის დროს მონაცემთა მოთხოვნის უნარს. ესაა ოპტიმალური შიგა შესაბამისობისთვის, განსაკუთრებით კონფიდენციალური მონაცემების დასამუშავებლად, სადაც ყველაფერი ხორციელდება მინიმალური შეყოვნებებით.

Row-Level Security არის კიდევ ერთი ფუნქცია, რომელიც უზრუნველყოფს დაშვების კონტროლს ცხრილის სტრიქონებზე პირობების საფუძველზე, რომლებსაც მომხმარებელი აწესებს.

დინამიკური მონაცემების მასივი ასევე ახალი იყო SQL Server 2016-ში. რამდენიმე სხვა გაუმჯობესება, როგორიცაა მონაცემების გამჭვირვალე შიფრაცია OLTP მეხსიერებით და აუდიტის შესაძლებლობათა გაფართოება, ანიჭებს მომხმარებელს ახალ შესაძლებლობებს, რათა დაკმაყოფილდეს უსაფრთხოების საჭიროებები.

Always Encrypted (ყოველთვის დაშიფრული) – ფუნქციაა, რომელიც იცავს მონაცემებს და ინახავს მათ მეხსიერებაში დაშიფრული სახით. ამის საპირისპიროდ, ტრადიციული დაშიფვრა გამოიყენება მხოლოდ შენახული მონაცემების დროს (ანუ იმ მონაცემების, რომელთა გამოყენებაც არ ხორციელდება), რის გამოც მონაცემები საფრთხის ქვეშაა, როდესაც მოხდება მათი გამოყენება. ხოლო Always Encrypted იცავს მონაცემებს შიგა და გარე თაღლითებისგან. ესაა ფუნქცია, რომელიც გამოიყენება კონფედენციალური მონაცემების დასაცავად, როგორიცაა მაგალითად, ნაციონალური საიდენტიფიკაციო ნომერი და ა.შ.

მოკლედ რომ ვთქვათ, ყველა ის ინფორმაცია, რომლის გავრცელებაც არ სურს მომხმარებელს. ის აძლევს საშუალებას მომხმარებელს დაშიფროს მონაცემები კლიენტის აპლიკაციაში, ისე, რომ არ გაუმხილოს SQL Server-ს შიფრაციის გასაღები. ამის შედეგად, ის უზრუნველყოფს გამოყოფას მათ შორის, ვინც ფლობს მონაცემებს (და შეუძლიათ მათი დათვალიერება) და ვინც მართავს მონაცემებს (მაგრამ არ უნდა ჰქონდეთ ამის უფლება). არსებითად, ის იცავს მონაცემთა ბაზას თავდასხმისგან.

Dynamic Data Masking (DDM) მასივი, უზრუნველყოფს მგრძნობიარე მონაცემების კონტროლის შეზღუდვის მექანიზმს, იმის მიხედვით თუ როგორი გამომავალი მონაცემები გამოჩნდება მონაცემთა ბაზაში. Masking წესები შეიძლება განისაზღვროს მონაცემთა ბაზის გარკვეულ სვეტში, რომლის მიხედვითაც ამ Masking დაკმაყოფილებისას შესაბამისი Mask გამოიყენება.

ამდენად, მგრძნობიარე მონაცემები ხდება ნიღბიანი იმ მომხმარებლების მოთხოვნით, რომელთაც არ სჭირდებათ ამ მონაცემების ნახვა ბიზნესში.

დინამიკურ მონაცემთა მასკირების გამოყენება შეიძლება განაცხადში მონაცემების გაუქმებაზე, რათა შეზღუდული

მონაცემები არ გამოვლინდეს. მაგალითად, ქოლ-ცენტრის აპლიკაცია, რომელიც გამოიყენება წარმომადგენლის მიერ მომხმარებლების გადახდის შესახებ ინფორმაციის იდენტიფიცირებაზე, უნდა იყოს გამოვლენილი მხოლოდ კლიენტის საკრედიტო ბარათის ნომრის ბოლო ოთხი ციფრი. მარტივი DDM წესმა შეიძლება უზრუნველყოს, რომ ნებისმიერ დროს, საკრედიტო ბარათის ნომრის საველე გაეგზავნება აპლიკაციით, ნომერი იქნება მასკირებული, გარდა ბოლო ოთხი ციფრისა.

DDM-ის გამოყენების კიდევ ერთი სცენარი საშუალებას აძლევს ინჟინრებს ან ოპერაციულ პერსონალს მიიღონ წვდომა წარმოების მონაცემთა ბაზაში არეულობის თავიდან ასაცილებლად, ან მომსახურების გაწევისათვის, რითიც საფრთხის ქვეშ არ უნდა ჩააყენონ პირადი საიდენტიფიკაციო ინფორმაცია მონაცემთა ბაზაში. DDM- ის პოლიტიკას შეუძლია ამ მონაცემების რეალურ დროში გაუქმება, რათა ასეთი ამოცანები მიღწეული იქნას კონფიდენციალურობის პოლიტიკის დარღვევის გარეშე.

2.6. CASE ტექნოლოგია: ORM/ERM და Ms SQL server-ის DDL - ფაილის გენერირება

ინფორმატიკის დიდაქტიკის ერთ-ერთი მნიშვნელოვანი თემა კომპიუტერული სისტემის მონაცემთა ბაზის დაპროექტების და მისი პროგრამული რეალიზაციის, ანუ სამუშაო ბაზის აგების ამოცანაა. CASE (Computer-Aided Software Engineering) ტექნოლოგია (მეთოდი და ინსტრუმენტული საშუალება) უზრუნველყოფს პროგრამული უზრუნველყოფის დეველოპმენტის პროცესების ავტომატიზაციას. CASE-ის დანერგვის მიზანია სოფთის შემუშავების დროისა და ღირებულების შემცირება და აგებული სისტემის ხარისხის ამაღლება [75].

ამჯერად, ჩვენ ვამახვილებთ ყურადღებას მონაცემთა ბაზის (მაგალითად, Ms SQL Server-სთვის) დაპროექტების და აგების პროცესების ავტომატიზაციაზე. განსაკუთრებით მნიშვნელოვანია ასეთი პროცესების ფორმალიზაცია და იმპლემენტაცია თვით (ბაზების აგების არაპროფესიონალი) საბოლოო მომხმარებლების მიერ, ამერიკელი მეცნიერის ტ. ჰალპინის NORMA (Natural ORM Architect) ტექნოლოგიის საფუძველზე [23, 67].

Object-Role Modeling (ORM) არის გრაფიკული კონცეპტუალური მოდელირების ტექნიკა, რომელიც გამოიყენება ძირითადად მონაცემთა ბაზის ანალიზისა და დიზაინისთვის. მისი დახმარებით ხდება საპრობლემი სფეროს კონცეპტუალური მოდელის აგების პროცესების ავტომატიზაცია.

ცოდნა, რომელიც საპრობლემო სფეროს შესახებ გააჩნია მომხმარებელს, სპეციალური ინტერფეისების საშუალებით, რომელთა საფუძველს ფორმალური ენის გრამატიკის კატეგორიები და ლოგიკურ-ალგებრული მეთოდები შეადგენს, გადაეცემა ობიექტ-როლური მოდელირების კომპიუტერულ პროგრამას [76],

ჩვენი წიგნის 2.1 ნახაზზე ნაჩვენები იყო საპრობლემო სფეროს მოდელირებისა და მონაცემთა ბაზაში ავტომატიზებული ასახვის ამოცანის კონცეფცია.

მონაცემთა ბაზის დაპროექტება მოიცავს საპრობლემო სფეროს ფორმალური მოდელის აგებას. ობიექტ-როლური მოდელირება (ORM) ამარტივებს დაპროექტების პროცესს, იყენებს რა ბუნებრივ, სალაპარაკო ენას, ასევე ინტუიციურ დიაგრამებს, რომელთა შევსებაც შეიძლება მაგალითების საშუალებით და შესაძლებელია ინფორმაციის შემოწმება მარტივ, ელემენტარულ ფაქტებზე დაყრდნობით. ვინაიდან მოდელი გამოსახულია ისეთ ბუნებრივ ტერმინებში, როგორიცაა *ობიექტი და როლი*, იგი უზრუნველყოფს მოდელირების კონცეპტუალურ მიდგომას [76].

კონცეპტუალური მოდელირება მიიღწევა არსთა დამოკიდებულების (ER) მოდელითაც, თუმცა იგი ნაკლებადაა შესაფერისი სემანტიკური ფორმალიზაციისათვის, ER-დიაგრამა შორსაა ბუნებრივი ენისაგან, ვერ ხერხდება ამა თუ იმ მოვლენის შევსება ფაქტით, დამალულია ინფორმაცია იმ სემანტიკური დომენების შესახებ, რომლებიც ქმნის მოდელს.

ამრიგად, კონცეპტუალური მოდელირების განვითარებული ტექნიკაა „ობიექტ-როლური“ მიდგომა. სწორედ ORM-ს შეუძლია უზრუნველყოს სხვადასხვა პროფესიის ადამიანების შეთანხმებული მუშაობა, რომელთა მომზადების დონე ინფორმაციული სისტემების დაპროექტების სფეროში შეიძლება მნიშვნელოვად განსხვავდებოდეს.

საინფორმაციო სისტემების აგების სასიცოცხლო ციკლი, როგორც ცნობილია, მოიცავს რამდენიმე ეტაპს: ტექნიკურ-ეკონომიკური დასაბუთება, მოთხოვნილებათა ანალიზი, მონაცემებისა და ოპერაციების კონცეპტუალური დაპროექტება; ლოგიკური დაპროექტება; პროგრამული რეალიზაცია, ტესტირება და კორექტირება; მომსახურება (თანხლება [77].

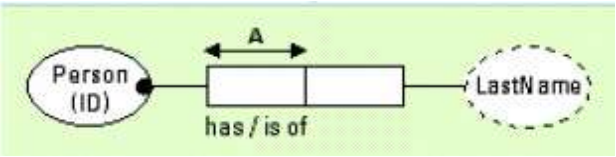
ORM-ის კონცეპტუალური სქემის მოდელირების პროცედურა ყურადღებას ამახვილებს მონაცემების ანალიზსა და დაპროექტებაზე. ეს სქემა აღწერს აპლიკაციის ინფორმაციულ სტრუქტურას: ფაქტების ტიპები, რომლებიც შეესაბამება კვლევის სფეროს. არსებული წესები და შეზღუდვები ქმნის წინაპირობას არსებული ფაქტებიდან ახალი ფაქტის მისაღებად.

- განვიხილოთ *ORM დიაგრამაში გამოყენებული ძირითადი შეზღუდვები:*

ამჯერად ჩვენი მიზანია ფაქტების საფუძელზე ობიექტ-როლური დიაგრამის აგების წესების შესწავლა.

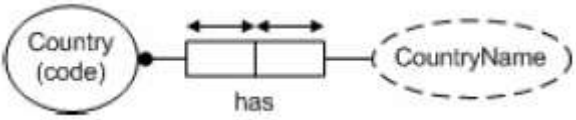
1) შიგა უნიკალურობის და შიგა იძულების შეზღუდვები.

შიგა უნიკალურობის შეზღუდვას (internal uniquess) ასევე უწოდებენ „უნიკალურობას პრედიკატის შიგნით". იგი გვიჩვენებს, რომ მოცემული ფაქტისათვის ერთ ან მეტ როლში მონაწილეობა ხდება *მხოლოდ ერთხელ* ე.ი. არის უნიკალური. ORM_დიაგრამაზე გამოისახება ასეთი სახით (მაგ., ნახ. 2.22-ა).



ნახ.2.22-ა, შიგა უნიკალურობის შეზღუდვა

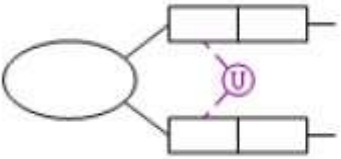
შიგა იძულების შეზღუდვა (Internal simple mandatory constraint) გვიჩვენებს რომ ობიექტი აუცილებლად უნდა ასრულებდეს გარკვეულ როლს. მაგალითად, იმისათვის რათა აღვწეროთ ფაქტი, რომ თითოეულ ქვეყანას აუცილებელია ჰქონდეს სახელი გამოისახება ასეთი სახით (ნახ.2.22-ბ).



ნახ.2.22-ბ, შიგა იძულების შეზღუდვა

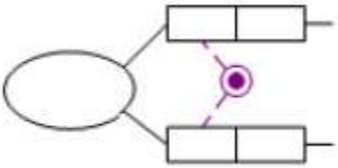
2) გარე უნიკალურობის და გარე იძულების შეზღუდვები.

გარე უნიკალურობის შეზღუდვას (External uniqueness constraint) უწოდებენ ასევე პრედიკატებსშორის უნიკალურობის შეზღუდვას. გარე უნიკალურობის შეზღუდვა გამოიყენება სხვადასხვა ტიპის ფაქტებში ორი ან მეტი როლისათვის. მას იყენებენ ობიექტის უნიკალურად იდეტიფიცირებისათვის. გარე უნიკალურობის შეზღუდვა აღინიშნება ასეთი სახით (ნახ.2.22-გ).



ნახ.2.22-გ, გარე უნიკალურობის შეზღუდვა

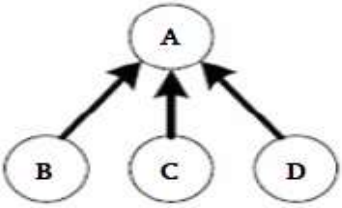
გარე იძულების შეზღუდვა (Disjunctive mandatory constraint) გვიჩვენებს, რომ ობიექტი ვალდებულია შეზღუდვაში ასრულებდეს სულ მცირე ერთ როლს მაინც. იძულების დიზუნქციის შეზღუდვა აღინიშნება ასეთი სახით (ნახ. 2.22-დ).



ნახ.2.22-დ, გარე იძულების შეზღუდვა

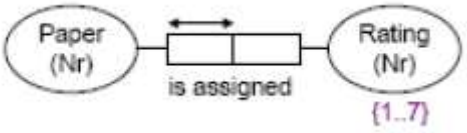
3) ქვეტიპის და მნიშვნელობის შეზღუდვები.

ქვეტიპის შეზღუდვა (Subtype constraint) შემოტანილია იმისათვის, რათა გამოიყოს გარკვეული ტიპის ობიექტები, რომლებიც თამაშობს სპეციფიურ როლებს. ქვეტიპის შეზღუდვა აღინიშნება ასეთი სახით (ნახ. 2.22-ე).



ნახ.2.22-ე, ქვეტიპის შეზღუდვა

მნიშვნელობის შეზღუდვა (Value constraint) გამოიყენება მაშინ, როდესაც საჭიროა განისაზღვროს მოცემული ობიექტისთვის დასაშვებ მნიშვნელობათა ერთობლიობა. ასეთი შეზღუდვა, მაგალითად, „გაზეთის რეიტინგი ფასდება ქულებით“ გამოისახება შემდეგი სახით (ნახ.2.22-ვ).



ნახ.2.22-ვ, მნიშვნელობის შეზღუდვა

დამატებითი ინფორმაციის მისაღებად შეზღუდვების სხვა ტიპების შესახებ, შეიძლება მიმართოთ ზემოაღნიშნილ ლიტერატურულ ან სხვა წყაროებს [67, 76].

- *კონცეპტუალური მოდელირების პროცესი* ORM-სქემისთვის შედგება შვიდი ბიჯისაგან [67, 76]:

ბიჯი_1: *ელემენტარული ფაქტების ფორმირება და მათი ადექვატურობის შემოწმება.* ამ მნიშვნელოვან სტადიაზე ხდება ინფორმაციის შეგროვება, ბუნებრივ სალაპარაკო ენაზე. ასეთი ინფორმაცია ხშირად არის შემავალი და გამომავალი ფორმები, ან ხელნაწერი. სხვა შემთხვევაში მოდელის დამპროექტებელი მუშაობს უშუალოდ დამკვეთთან, რათა ზუსტად ჩამოყალიბდეს, თუ რა მოეთხოვება სისტემას. ექსპერტ-კონსულტანტი უნდა იცნობდეს აპლიკაციას;

ბიჯი_2. ფაქტების ტიპებისათვის დიაგრამის აგება და სისრულის შემოწმება;

ბიჯი_3. იმ ობიექტთა ტიპების შემოწმება, რომლებიც უნდა გაერთიანდეს და მათი მათემატიკური წარმომავლობის დაფიქსირება;

ბიჯი_4. დაემატოს უნიკალურობის შეზღუდვა და შემოწმდეს ფაქტების ტიპების ოპერანდების რაოდენობა;

ბიჯი_5. დაემატოს როლების იძულებითი შეზღუდვები და შემოწმდეს მათი ლოგიკური წარმომავლობა;

ბიჯი_6. დაემატოს ელემენტები, სიმრავლეთა შედარება და ქვეტიპის შეზღუდვები;

ბიჯი_7. დაემატოს სხვა შეზღუდვები და მოხდეს საბოლოო შემოწმება.

ფაქტები შეიძლება ასე ჩაიწეროს:

• f1: თანამშრომელს, ნომრით 5, ***აქვს*** გვარი „შონია“;

• f2: თანამშრომელი, ნომრით 30, ***მუშაობს*** დეპარტამენტში კონტრაქტით თარიღამდე ‘01.10.2023’

თითოეული ფაქტი არის ბინარული დამოკიდებულება ორ ობიექტს შორის. მუქი შრიფტით გამოყოფილია ლოგიკური პრედიკატი, რომელიც ახდენს ობიექტებს შორის კავშირების იდენტიფიცირებას. იმ შემთხვევაში თუ განისაზღვრება ობიექტის მხოლოდ ერთი თვისება, საქმე გვაქვს ერთადგილიან პრედიკატთან (unary fact). პრედიკატს შეიძლება ჰქონდეს (1,2,3,..) ოპერანდი, თუმცა 3-4 ოპერანდზე მეტი პრაქტიკულად იშვიათია.

მე-2 ბიჯზე ხდება ფაქტების ტიპებისათვის დიაგრამის აგება. ობიექტები გამოისახება ელიფსებით, პრედიკატები მართკუთხედებით, მნიშვნელობის ტიპი გამოისახება წყვეტილი ელიფსით. პრედიკატი იკითხება მარცხნიდან-მარჯვნივ და ზემოდან-ქვემოთ, მანამ სანამ არ შეხვდება ნიშანი “<<”, რომელიც ცვლის წაკითხვის მიმართულებას საწინააღმდეგო მიმართულებით.

შემდეგ ბიჯებზე ხდება შეზღუდვების დაწესება, რაც მოკლედ განვიხილეთ ამ პარაგრაფის დასაწყისში.

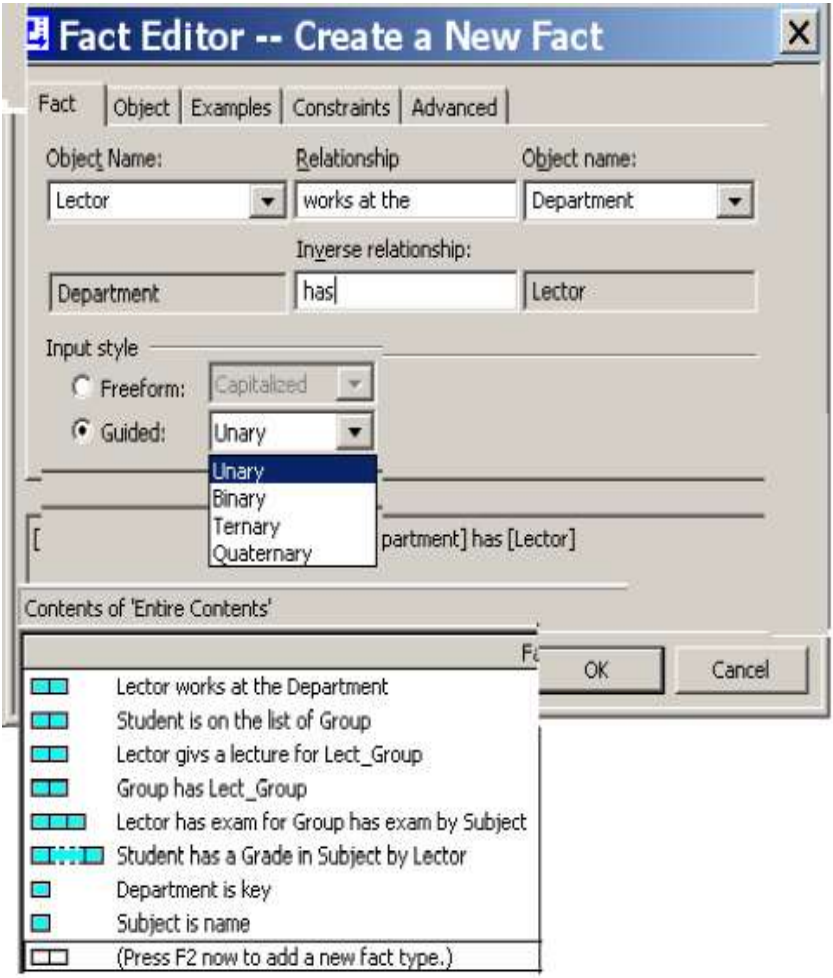
ახლა განვიხილოთ კონკრეტული ამოცანა: ORM-დიაგრამის ასაგებად უნივერსიტეტის მაგალითზე, რომელც საფუძვლად დაედება მის კონცეპტუალურ სქემას.

უნივერსიტეტის საპრობლემი სფეროს ობიექტებია (ტერმინთა ლექსიკონი): ფაკულტეტი, კათედრა, სტუდენტი, ლექტორი, საგნები (აკადემიური კურსი), რომელიც იკითხება სპეციალობების მიხედვით. საგანმანათლებლო პროგრამა, სილაბუსი, ლექცია, პრაქტიკული, ლაბორატორიული, საკურსო პროექტი, გამოცდა, ტესტირება და ა.შ.

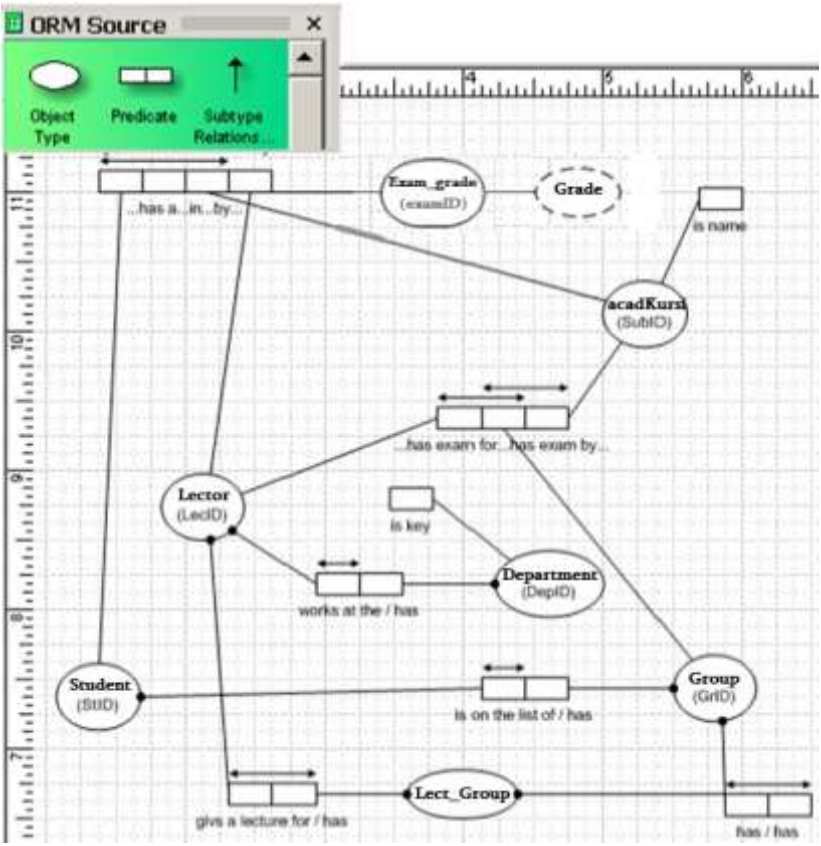
მისი ფაქტების ერთობლიობა იქნება:

f1 - თანამშრომელს (ლექტორს) აქვს გვარი, სახელი.
f2 - თანამშრომელი მუშაობს დეპარტამენტში.
f3 - თანამშრომელს დაკავებული აქვს ოთახი.
f4 - თანამშრომელი მუშაობს კონტრაქტით თარიღამდე.
f5 - თანამშრომელს უკავია თანამდებობა
f6 - თანამშრომელმა მიიღო ხარისხი უნივერსიტეტიდან.
f7 - თანამშრომელი ამოწმებს თანამშრომელს.
f8 - თანამშრომელს აქვს წოდება
f9 - თანამშრომელი ასწავლის აკადემიურ კურსს (საგანს).
ff0 - თანამშრომელი ატარებს რეიტინგს.
f11 - პროფესორი ხელმძღვანელობს დეპარტამენტს.
f10 - დეპარტამენტს აქვს ბიუჯეტი.
f11 - პროფესორს უკავია თანამდებობა კათედრაზე.
f12 - მასწავლებელი არის თანამშრომელი.
f13 - პროფესორი არის თანამშრომელი
და ა შ.

ზემოაღწერილი წესების თანახმად საჭიროა აიგოს ORM-დიაგრამა. ამისათვის ფაქტ-შეზღუდვები, რომლებშიც ასახულია საპრობლემო სფეროს შესახებ ცოდნა (კლასებისა და ობიექტების ძირითადი ტერმინები და ქცევის წესები), ჩვენს მიერ გადაიტანება Ms_Visio ინსტრუმენტის საშუალებით ობიექტ-როლურ მოდელში (ნახ. 2.23-ა და ბ).



ნახ.2.23-ა. ფაქტების ფორმალური ჩაწერა (ობიექტებით და პრედიკატებისთ) უნივერსიტეტის მაგალითზე

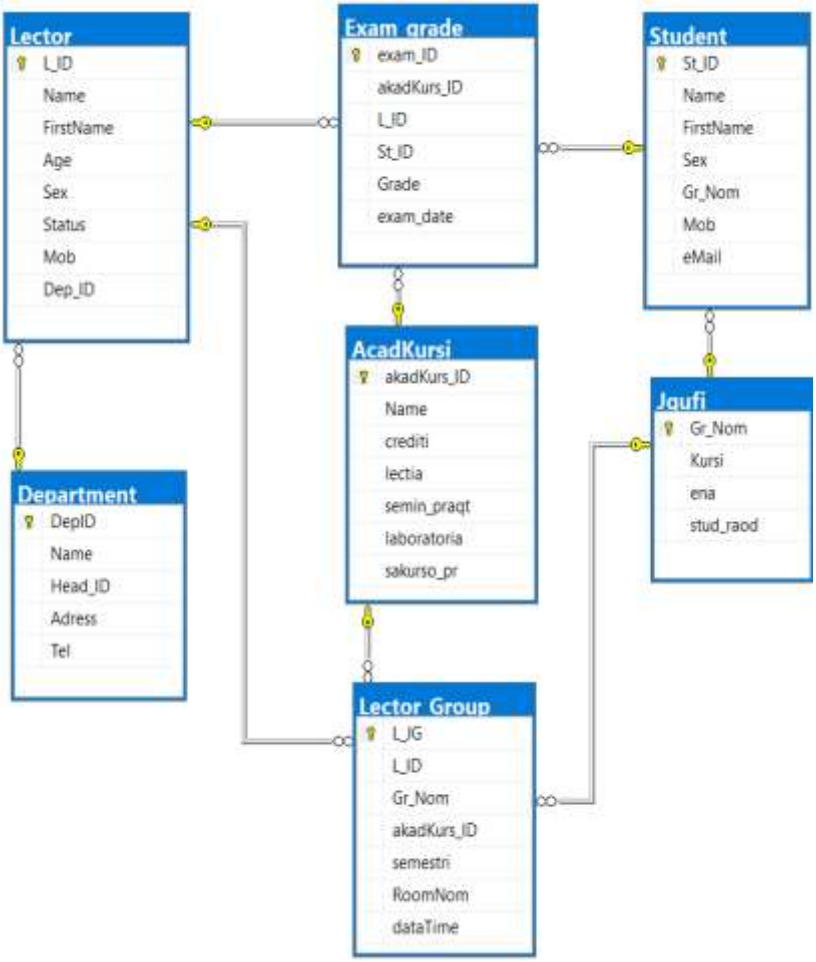


ნახ..2.23-ბ. *ORM*-დიაგრამა *MsVisio*-გარემოში

ნახაზზე კარგად ჩანს 2, 3 და 4 ადგილიანი პრედიკატები, რომლებითაც განხორციელებულია ობიექტებს შორის კავშირები.

მომდევნო ეტაპზე განხორციელდა ORM მოდელის ავტომატური გადაყვანა ERM მოდელში. 2.24 ნახაზზე ნაჩვენებია არსთა

დამოკიდებულების (კონცეპტუალური სქემის) ფრაგმენტი რამდენიმე ცხრილით (ORM დიაგრამის შესაბამისად).



ნახ. 2.24. ERM დიაგრამა მონაცემთა ბაზისათვის

ახლა საჭიროა Ms SQL - ბაზისთვის გამზადდეს .DDL ფაილი, რომლითაც ბოლოს მოხდება რეალური ფიზიკური ბაზის შექმნა თავისი ცხრილებით.

- *SQL Server ბაზის სკრიპტული ფაილის მომზადება.*

უნივერსიტეტის საპრობლემო სფეროს მონაცემთა ბაზის კონცეპტუალური მოდელების დაპროექტების შემდეგ საჭიროა განისაზღვროს სკრიპტული ტიპის DDL–ფაილი, რომლითაც SQL Server მონაცემთა ბაზის მართვის სისტემა ავტომატურად შექმნის რეალურ მონაცემთა ბაზას თავისი ცხრილებით.

ეს პროცესიც ავტომატიზებულია ტ. ჰალპინის NORMA ინსტრუმენტის საშუალებით. ისევე როგორც ORM და ERM მოდელების პროექტირების ეტაპები. ჩვენ მხოლოდ უნდა ვუკარნახოთ სისტემას, რომ გვჭირდება SQL Server-ის DDL ფაილი.

ქვემოთ მოცემულია ავტომატურად გენერირებული DDL-ფაილის ტექსტის ძირითადი ფრაგმენტი.

```
/*** SQL-ბაზის და ცხრილების შექმნის  სკრიპტ ფაილის ფრაგმენტი ***/
USE [master]
GO
/****** Object:  Database [GTUni]    Script Date: 11.08.2023 21:27:29 ******/
CREATE DATABASE [GTUni]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'GTUni', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\GTUni.mdf' , SIZE = 8192KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB )
 LOG ON
( NAME = N'GTUni_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\GTUni_log.ldf' , SIZE = 8192KB ,
MAXSIZE = 2048GB , FILEGROWTH = 65536KB )
 WITH CATALOG_COLLATION = DATABASE_DEFAULT
GO
 . . . . .
/*** Object:  Table [dbo].[Student]    Script Date: 11.08.2023 21:27:30 ***/
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Student](
	[St_ID] [int] NOT NULL,
	[Name] [nchar](15) NULL,
	[FirstName] [nchar](15) NULL,
	[Sex] [nchar](10) NULL,
	[Gr_Nom] [nchar](10) NULL,
	[Mob] [nchar](10) NULL,
	[eMail] [nchar](20) NULL,
 CONSTRAINT [PK_Student] PRIMARY KEY CLUSTERED
(
	[St_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Lector]    Script Date: 11.08.2023 21:27:30 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Lector](
	[L_ID] [smallint] NOT NULL,
	[Name] [nchar](25) NULL,
	[FirstName] [nchar](20) NULL,
	[Age] [smallint] NULL,
	[Sex] [nchar](10) NULL,
	[Status] [nchar](20) NULL,
	[Mob] [nchar](10) NULL,
	[Dep_ID] [smallint] NULL,
 CONSTRAINT [PK_Lector] PRIMARY KEY CLUSTERED
(
	[L_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/*** Object:  Table [dbo].[Department]    Script Date: 11.08.2023 21:27:30 ***/
```

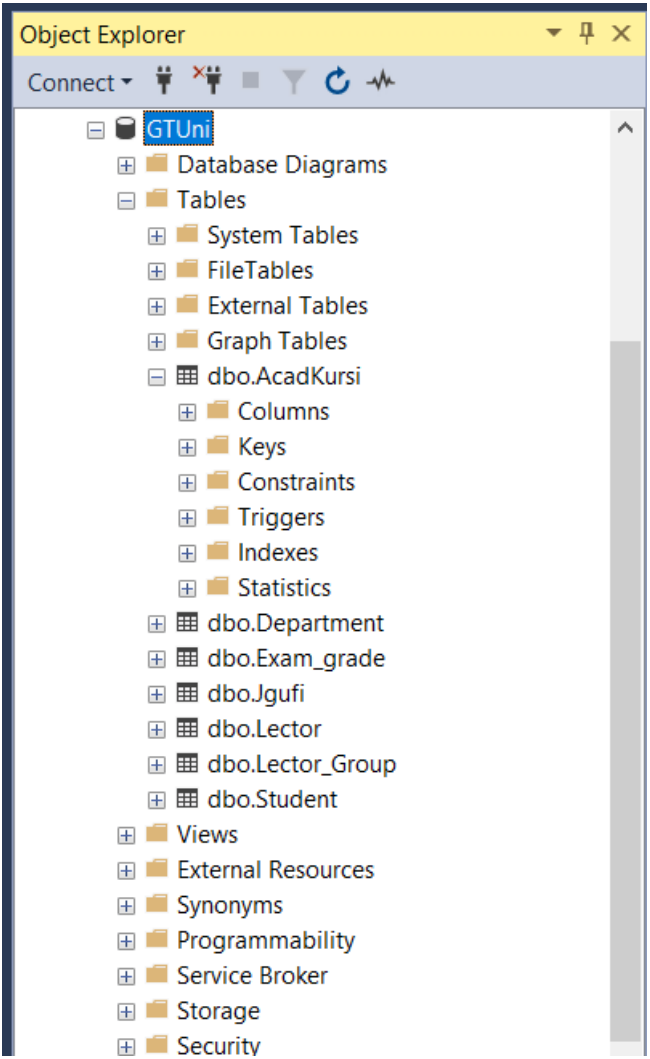
```
. . .
/*** Object:  Table [dbo].[Jgufi]    Script Date: 11.08.2023 21:27:30 ***/
. . .
/*** Object:  Table [dbo].[AcadKursi]    Script Date: 11.08.2023 21:27:30 ***/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[AcadKursi](
	[akadKurs_ID] [smallint] NOT NULL,
	[Name] [nchar](50) NULL,
	[crediti] [smallint] NULL,
	[lectia] [smallint] NULL,
	[semin_praqt] [smallint] NULL,
	[laboratoria] [smallint] NULL,
	[sakurso_pr] [smallint] NULL,
 CONSTRAINT [PK_AcadKursi] PRIMARY KEY CLUSTERED
(
	[akadKurs_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/*** Object:  Table [dbo].[Lector_Group]    Script Date: 11.08.2023 21:27:30 ***/
. . .
/*** Object:  Table [dbo].[Exam_grade]    Script Date: 11.08.2023 21:27:30 ***/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Exam_grade](
	[exam_ID] [smallint] NOT NULL,
	[akadKurs_ID] [smallint] NULL,
	[L_ID] [smallint] NULL,
	[St_ID] [int] NULL,
	[Grade] [smallint] NULL,
	[exam_date] [date] NULL,
 CONSTRAINT [PK_Exam_grade] PRIMARY KEY CLUSTERED
(
	[exam_ID] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Exam_grade]  WITH CHECK ADD  CONSTRAINT
[FK_Exam_grade_AcadKursi] FOREIGN KEY([akadKurs_ID])
REFERENCES [dbo].[AcadKursi] ([akadKurs_ID])
GO
ALTER TABLE [dbo].[Exam_grade] CHECK CONSTRAINT [FK_Exam_grade_AcadKursi]
GO
ALTER TABLE [dbo].[Exam_grade]  WITH CHECK ADD  CONSTRAINT
[FK_Exam_grade_Lector] FOREIGN KEY([L_ID])
REFERENCES [dbo].[Lector] ([L_ID])
GO
ALTER TABLE [dbo].[Exam_grade] CHECK CONSTRAINT [FK_Exam_grade_Lector]
GO
ALTER TABLE [dbo].[Exam_grade]  WITH CHECK ADD  CONSTRAINT
[FK_Exam_grade_Student] FOREIGN KEY([St_ID])
REFERENCES [dbo].[Student] ([St_ID])
GO
ALTER TABLE [dbo].[Exam_grade] CHECK CONSTRAINT [FK_Exam_grade_Student]
GO
. . .
```

ასეთი სრულყოფილი სკრიპტული ტექსტის მოთავსებით SQL Server Management Studio-ში და ამუშავებით, მბმს თვითონ ავტომატურად შექმნის უნივერსიტეტის მონაცემთა ბაზას, სკრიპტში აღწერილი ტექსტის საფუძველზე.

Management Studio-ს Object Explorer-ში გამოჩნდება ეს, ახლადშექმნილი ბაზა და ცხრილები, თავისი ატრიბუტებით, პირველადი და მეორეული გასაღებური ველებით, რელაციური კავშირებით და ა.შ.

2.25 ნახაზზე ნაჩვენებია Object Explorer-ის ფანჯარა და უნივერსიტეტის GTUni მონაცემთა ბაზის იერარქია შვიდი ცხრილის (Tables), ბაზის სქემების (Database Diagrams) და სხვა სისტემური კომპონენტებით.



ნახ.2.25. Object Explorer-ფანჯარა ახალი შექმნილი GTUni -მონაცემთა ბაზით

შევიტანოთ ექსპერიმენტული მონაცემები რამდენიმე ცხრილში და შევამოწმოთ მათი მნიშვნელობები. ცხრილების შიგთავსები ასახულია 2.26, ა-დ ნახაზებზე.

DESKTOP-IG2O44L... - dbo.Exam_grade | DESKTOP-IG2O44LG...I - dbo.AcadKursi

akadKurs_I_	Name	credit	lectia	semin_praqt	laboratoria	saxurso_pr
1	დაპროგრამების საფუძვლები	6	15	0	45	0
2	შესავალი ჭკვიან ტექნოლოგიებში	5	15	30	0	0
3	ინფო-ტექნოლოგიები	5	15	0	30	0
4	პროგრამული ინჟინერიის საფუძვლები	6	15	0	30	15
5	ალგორითმების დაპროგრამება და მონაცემთა მენეჯმენტი	5	15	30	0	0
6	პროგრამული პროდუქტების დეველოპმენტი	6	15	0	30	15
7	ღრუბლოვანი ტექნოლოგიები	5	15	0	15	15
8	მონაცემთა ბაზების დაპროექტება	5	145	15	0	15
9	ქსელური ინტერნეტის საფუძვლები	5	15	30	0	0
10	კიბერუსაფრთხოების საფუძვლები	5	15	30	0	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL

ნახ.2.26-ა. აკადემიური კურსების ცხრილის ფრაგმენტი

DESKTOP-IG2O44LGTUni - dbo.Lector

L_ID	Name	FirstName	Age	Gender	Status	Mob	Dep_I
1	სურგულაძე	გია	55	მამრ	პროფ	599373737	1
2	ბიბლაძე	თენგიზ	53	მამრ	პროფ	577535353	1
3	თოფურია	ნინი	30	მდედრ	ასოც.პროფ	555000200	1
4	ღვინეფაძე	გელა	57	მამრ	პროფ	577666777	1
5	თუმჩია	ეკა	31	მდედრ	მოწვ.პროფ	577252525	1
6	პატრიაშვილი	ლია	33	მდედრ	პროფ	577404040	1
31	ლეონიშვილი	ბადრი	58	მამრ	პროფ	599112233	3
32	ქართველიშვილი	სოსო	32	მამრ	პროფ	599445566	3
33	ამილახვარი	ზვიად	37	მამრ	პროფ	599332211	3
51	კიკნაძე	მზია	37	მდედრ	პროფ	591377373	2
52	კაკუბავა	იაკობი	45	მამრ	პროფ	591515151	2
60	ახობაძე	მერაბ	55	მამრ	პროფ	577111111	4
61	გაზიბაძე	მარინა	34	მდედრ	ასოც.პროფ	599775577	4
62	ოხანაშვილი	მაია	31	მდედრ	ასოც.პროფ	599557755	4
80	ჩხაიძე	მარიამ	33	მდედრ	პროფ	591335577	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

ნახ.2.26-ბ. ლექტორთა ცხრილის ფრაგმენტი

DESKTOP-IG2O44L.GTUni - dbo.Student

St_ID	Name	FirstName	Gender	Gr_Nom	Mob	eMail
1	ბერულავა	ანა	მდედრ	108851	599123456	aberul@gtu.ge
2	გულუა	დავით	მამრ	108850	577123456	dgulua@gtu.ge
3	დოლიძე	სანდრო	მამრ	108850	593123456	sdoli93@gtu.ge
4	ბაბია	გიორგი	მამრ	108852	599001122	gbakhi@gmail.com
5	თურჩია	ქეთი	მდედრ	108852	555334455	kturkia@gtu.ge
6	კოსტავა	დავით	მამრ	108851	577454545	dkosta@gtu.ge
7	მაისურაძე	რატი	მამრ	108851	577131313	rmaisu@gmail.com
8	მესხური	ეთნე	მამრ	108853	577252525	tsotne7@gmail.com
9	ნებულიშვილი	ამიკო	მამრ	108850	599112112	anebuli@gtu.ge
10	ულრელიძე	ვაჩო	მამრ	108850	593669977	vughrel@gtu.ge
11	ხრიკული	მაია	მდედრ	108853	591222324	mkhriku@gtu.ge
21	ჯაჩნაძე	საბა	მამრ	108950	577303030	svachna@gtu.ge
22	თოფურია	გიორგი	მამრ	108950	593757575	gtopuri@gmail.com
23	მამედოვა	სამირა	მდედრ	108950	577008899	smamed@gtu.ge
31	დუმბაძე	ნანა	მდედრ	108951	555777999	ndumba@gtu.ge
32	თუთბერიძე	დიტო	მამრ	108951	593445566	dtutber@gtu.ge
41	ბაბაჭიშვილი	რამაზ	მამრ	108952	599222444	rbabak@gtu.ge
42	ყაველაშვილი	მათე	მამრ	108953	593565656	mkavela@gtu.ge
43	გოგინავა	გიორგი	მამრ	108953	577151515	ggogina@gtu.ge
44	ჩახვაძე	ზატია	მდედრ	108953	555001002	khatiach@gtu.ge
60	მაჰმუდბეგი	მაჰმუდ	მამრ	108058	599002003	mahmud@gmail.com
61	ავაჭოვი	ტიგრან	მამრ	108039	577654321	avatigr@mail.ru
NULL	NULL	NULL	NULL	NULL	NULL	NULL

ნახ.2.26-გ. სტუდენტთა ცხრილის ფრაგმენტი

DESKTOP-IG2O44L....- dbo.Exam_grade | DESKTOP-IG2O44L.G...i - dbo.AcadKursi

exam_ID	akadKurs_ID	L_ID	St_ID	Grade	exam_date
1	1	1	1	65	2023-02-22
2	1	1	2	95	2023-02-22
3	1	1	3	51	2023-02-22
4	1	1	4	100	2023-02-22
5	2	2	1	81	2023-02-16
6	2	2	2	91	2024-02-16
7	2	2	3	58	2023-02-16
8	2	2	4	94	2023-02-16
9	3	1	1	70	2023-02-27
10	3	1	2	91	2023-02-27
11	3	1	3	62	2023-02-27
12	3	1	4	0	2023-02-27
13	1	1	7	25	2023-02-22
14	1	1	8	42	2023-02-22
15	2	2	7	30	2023-02-16
16	3	1	7	0	2023-02-27
NULL	NULL	NULL	NULL	NULL	NULL

ნახ.2.26-დ. გამოცდის შედეგების ცხრილის ფრაგმენტი

ახალი ტიპის ციფრული ტექნოლოგიების გამოყენებით, რომლებიც მოდელირების უნიფიცირებული ენის (UML) ინსტრუმენტებზეა ორიენტირებული, შესაძლებელია მოდელირებისა და დაპროგრამების პროცესების ავტომატიზაცია, რაც საბოლოო ჯამში შესაძლებელს ხდის მონაცემთა ბაზების მართვის სისტემების (მაგალითად, MsSQL Server, SyBase, Oracle, ADO.NET და სხვ.) და დაპროგრამების საინტერფეისო ენების (მაგალითად, C#.NET, VB.NET, Pithon და სხვ.) ინტეგრირებული გამოყენებით ვიზუალური მოდელირების პაკეტებთან ერთად (მაგალითად, Ms Visio, Rational Rose, Paradigm+, Enterprise Architect) დაპროექტდეს და რეალიზებულ იქნას ხარისხიანად და სწრაფად მართვის საინფორმაციო სისტემების Windows- და Web-აპლიკაციები.

თავი 3
სუფთა არქიტექტურა და დომენზე ორიენტირებული დიზაინი

საგანმანათლებლო დაწესებულებათა მართვის ინფორმაციული სისტემების პროცესების ავტომატიზაცია თანამედროვე ციფრული ტექნოლოგიების ბაზაზე მოითხოვს დეველოპერებისაგან საბოლოო პროგრამული პროდუქტის ხარისხის სრულყოფას. აქ მნიშვნელოვანი როლი სისტემის არქიტექტურის სწორად განსაზღვრას და თვით პროგრამულ დეველოპმენტს და ტესტირებას ეკუთვნის [78, 79].

პროგრამული უზრუნველყოფის არქიტექტურისადმი სერიოზული დამოკიდებულება გულისხმობს იმის ცოდნას, თუ რა არის კარგი არქიტექტურა. სისტემის შესაქმნელად, რომლის დიზაინი და არქიტექტურა ხელს უწყობს შრომის ხარჯების შემცირებას და პროდუქტიულობის ზრდას, საჭიროა იმის განსაზღვრა თუ არქიტექტურის რა ელემენტებით უნდა განხორციელდეს იგი [80].

დომენზე ორიენტირებული დიზაინი (Domain Driven Design – DDD, ან საპრობლემო სფეროზე ორიენტირებული დიზანი) არის სუფთა არქიტექტურის მსგავსი მეთოდოლოგია პროგრამული აპლიკაციის ასაგებად.

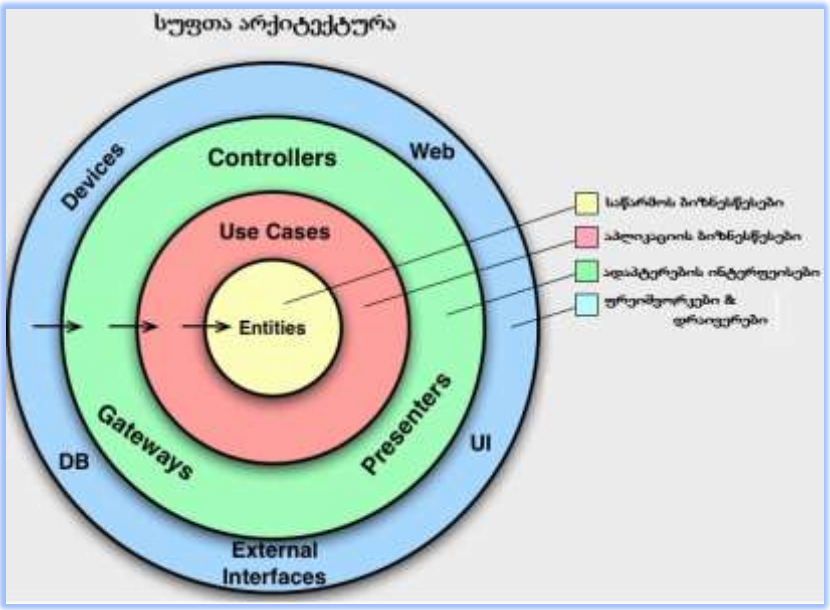
სუფთა არქიტექტურა შესაფერისია პროექტებისთვის, სადაც არქიტექტურული საზრუნავი (პრობლემების გამიჯვნა და ტესტირება) უმთავრესია. ხოლო DDD კი – როდესაც დომენის სირთულე და მის ექსპერტებთან თანამშრომლობა უფრო მნიშვნელოვანია პროგრამული პროდუქტის დიზაინის ასაგებად [81, 82].

წინამდებარე თავში დეტალურად განვიხილავთ *სუფთა არქიტექტურის და DDD მეთოდოლოგიების* ძირითად პრინციპებს, მათ გამოყენებას განათლების მენეჯმენტის საინფორმაციო სისტემებში.

3.1. სუფთა არქიტექტურის მოდელი (CLAM)

ტერმინი „სუფთა არქიტექტურა“ (Clean Architecture) პროგრამული ინჟინერიის მიმართულების ამერიკელი მეცნიერის რობერტ ს. მარტინის („ბიძია ბობი“) შემოტანილია. იგი არის, ამასთანავე, Agile-მანიფესტის ერთ-ერთი თანაავტორი (კენტ ბეკთან ერთად) [43, 82].

სუფთა არქიტექტურის მთავარი იდეაა ის, რომ არსები (entities) და გამოყენების შემთხვევები (UseCases) დამოუკიდებელია ფრეიმვორკების, მომხმარებლის ინტერფეისის, მონაცემთა ბაზისა და გარე სერვისებისგან (ნახ. 3.1) [83].



ნახ.3.1. სუფთა არქიტექტურის მოდელი

- ***დამოკიდებულების წესი*** (Dependency Rule). კონცენტრული წრეები არის პროგრამული უზრუნველყოფის სხვადასხვა სფერო.

ზოგადად, რაც უფრო ცენტრისკენაა წრე, მით უფრო მაღალი დონის პროგრამული უზრუნველყოფაა. ამგვარად, გარე წრეები მექანიზმებია. შიგა კი - პოლიტიკა.

მთავარი წესი, რომლითაც ეს არქიტექტურა მუშაობს, არის *დამოკიდებულების წესი*. იგი ამბობს, რომ შიგა წრეში ვერაფერმა შეიძლება იცოდეს რაიმე გარე წრის შესახებ. კერძოდ, გარე წრეში გამოცხადებული რამის სახელი არ უნდა იყოს მითითებული შიგა წრის კოდში. მაგალითად, ფუნქციები, კლასები. ცვლადები ან სხვ.

ამავე პრინციპით, გარე წრეში გამოყენებული მონაცემთა ფორმატები არ უნდა იყოს გამოყენებული შიგა წრის მიერ, განსაკუთრებით თუ ეს ფორმატები გენერირებულია გარე წრის ფრეიმვორკის მიერ [83, 84].

ამგვარად, მთავარია, რომ გარე წრიდან რაიმემ გავლენა არ იქონიოს შიგა წრეებზე. დამოკიდებულებები ყოველთვის უნდა იყოს მიმართული შიგნით, რაც იძლევა მაღალი მოდულარობის და გარე კომპონენტების მარტივად ჩანაცვლების ან მოდიფიკაციის საშუალებას. განვიხილოთ სუფთა არქიტექტურის ეს ფენები.

- ***არსთა ფენა*** (Entities layer) აერთიანებს საპროექტო ორგანიზაციის ბიზნესპროცესებს და ბიზნესწესებს. არსი შეიძლება იყოს ობიექტი თავისი მეთოდებით ან მონაცემთა სტრუქტურების და ფუნქციების ერთობლიობა. ეს ფენა შეიცავს საპრობლემო სფეროს ობიექტებს, რომლებსაც იყენებს Use Cases ფენა თავისი ბიზნეს ლოგიკის შესასრულებლად. ეს ობიექტები, როგორც წესი, არის ჩვეულებრივი ძველი მონაცემთა ობიექტები, რომლებიც ინახავს მონაცემებს სისტემისთვის და ასევე შეიძლება შეიცავდეს ამ მონაცემების მანიპულირების მეთოდებს. მაგალითად, *არსის კლასი* შეიძლება იყოს სტუდენტის საბანკო ანგარიში და მოიცავდეს მეთოდებს მისი ელფოსტის მისამართის ან პაროლის დასადასტურებლად;

➢ ***გამოყენების შემთხვევების ფენა*** (Use Cases layer – ან პრეცედენტების ფენა) პასუხისმგებელია აპლიკაციის ბიზნეს ლოგიკის მართვაზე. სწორედ აქაა განსაზღვრული და დანერგილი აპლიკაციის ძირითადი ფუნქციონირება.

Use Cases ფენაში ასახულ;ია ბიზნესპროცესში მონაწილე როლები (Actors) და მათი ფუნქციები (Actions). როგორც წესი, იგი შედგება პრეცედენტთა კლასებისგან, რომელთაგან თითოეული შეესაბამება კონკრეტულ ბიზნეს მოთხოვნას ან მოქმედებას (Activity), რაც უნდა შეასრულოს აპლიკაციამ (როლის შესაბამისად). Use Cases ფენის დანიშნულებაა სისტემის ბიზნესლოგიკის ინკაფსულაცია და სისტემის სხვა ფენების ინტერფეისის უზრუნველყოფა.

ბიზნესლოგიკის ცალკე ფენაში შენარჩუნებით, ბევრად უფრო ადვილი ხდება კოდის გაგება და შეცვლა, ასევე სისტემის ტესტირება. ის არ უნდა იყოს დამოკიდებული სისტემის რომელიმე სხვა ფენაზე და მხოლოდ მათთან უნდა ურთიერთობდეს კარგად განსაზღვრული ინტერფეისების საშუალებით. ეს უზრუნველყოფს სისტემის ბიზნეს ლოგიკის განცალკევებას ინფრასტრუქტურისა და მომხმარებლის ინტერფეისის დეტალებისგან. ეს აადვილებს ამ კომპონენტების შეცვლას სისტემის ძირითად ფუნქციონირებაზე გავლენის გარეშე.

➢ ***ინტერფეისის ადაპტერების ფენა*** (Interface Adapters layer) - პასუხისმგებელია არსებისა და პრეცედენტების ფენათა ობიექტების ადაპტაციაზე გარე სამყაროსთან. ანუ იგი გარდაქმნის არსებისა და პრეცედენტების მონაცემთა ფორმატებს ისეთ ფორმატებში, რომლებიც უფრო ხელსაყრელია გარე სისტემებისთვის (მონაცემთა ბაზები, ვებ-ინტერფეისები და სხვ.). მაგალითად, ინტერფეისის ადაპტერის კლასმა შეიძლება შეასრულოს მომხმარებლის ანგარიშის ობიექტის გარდაქმნა Json ობიექტად, რომელიც უნდა გაიგზავნოს ვებ-კლიენტთან.

- ***ფრეიმვორკების და დრაივერების ფენა*** (frameworks and DriverS Layer) – ესაა სისტემის ყველაზე გარე ფენა და იგი პასუხისმგებელია ინფრასტრუქტურისა და სერვისების უზრუნველყოფაზე, რომლებზეც სხვა ფენებია დამოკიდებული. ეს ფენა შეიცავს კოდს მონაცემთა ბაზებთან, ვებ სერვისებთან და სხვა გარე სისტემებთან ურთიერთობისთვის. მაგალითად, ფრეიმვორკმა ან დრაივერის კლასმა შეიძლება შეასრულოს SQL Server-ის მონაცემთა ბაზასთან დაკავშირების პროცესი ან ვებ-სამსახურისთვის მოთხოვნის გადაცემა.

და ბოლოს, შეიძლება ითქვას, რომ სუფთა არქიტექტურით დაპროექტებული სისტემა (აპლიკაცია) უფრო ადვილად ტესტირებადია. ვინაიდან სისტემის სხვადასხვა ფენა ერთმანეთისგან დამოუკიდებელია, შესაძლებელია თითოეული ფენის ცალ-ცალკე დატესტვა. რაც ბევრად აადვილებს სისტემის გამართულ მუშაობას. ეს განსაკუთრებით მნიშვნელოვანია დიდ, რთულ სისტემებში, სადაც საკმაოდ ძნელი ხდება მთლიანი სისტემის ტესტირება.

მართვის საინფორმაციო სისტემების დაპროექტებისათვის სუფთა არქიტექტურის მეთოდოლოგია მეტად მნიშვნელოვანია. ასევე საყურადღებოა კიდევ ერთი – დომენებზე ორიენტირებული დიზაინის (DDD) მთოდოლოგია. ამ თავის შესავალში ჩვენ მოკლედ დავახასიათეთ სისტემების არქიტექტურის ორივე მონათესავე მიდგომა. მათ ბევრი საერთო, მაგრამ პრინციპულად განსხვავებული იდეოლოგია აქვს.

მომდევნო პარაგრაფში ვეცდებით DDD-მეთოდოლოგიის ძირითადი ასპექტების მოკლედ განხილვას, დაპროექტების, დანერგვის და გამოყენების ეფექტიანობის თვალსაზრისით.

3.2. დომენზე ორიენტირებული დიზაინი (DDD)

ინფორმატიკის დიდაქტიკის ერთ-ერთი მნიშვნელოვანი ნებისმიერი საკითხია მართვის საინფორმაციო სისტემის სწორი არქიტექტურის განსაზღვრა. წინა პარაგრაფში ჩვენ განვიხილეთ „სუფთა არქიტექტურის“ მეთოდოლოგია.

ამჯერად გვინდა გავამახვილოთ ყურადღება მის ერთ-ერთ ალტერნატიულ მიდგომაზე, რომელსაც საკმაოდ დიდი პრაქტიკული გამოყენება აქვს პროგრამული ინჟინერიის სფეროში, კერძოდ დომენზე ორიენტირებული დიზაინი (DDD – Domen-Driven Design).

ამავდროულად, ამ მეთოდოლოგიის გამოყენებით შევეცდებით ავაგოთ სისტემის არქიტექტურა საგანმანათლებლო დაწესებულების ობიექტების ავომატიზაციისათვის.

დომენზე (ან საპრობლემო სფეროზე) ორიენტირებული დიზაინი მხარს უჭერს ისეთ მოდელირებას, რომელიც დაფუძნებულია ბიზნესის რეალობაზე. იგი შეესაბამება დამკვეთის მოთხოვნებს, აღწერს დამოუკიდებელ პრობლემურ ზონებს, როგორც შემოსაზღვრულ კონტექსტებს. თითოეული ასეთი კონტექსტი უკავშირდება მიკრომომსახურებას და უზრუნველყოფს საერთო სასაუბრო ენას.

განვიხილოთ შიგა ნიმუშების დიზაინი და განხორციელება: ზოგჯერ ეს DDD ტექნიკური წესები და პატერნები აღიქმება, როგორც დაბრკოლებები, რომლებიც საჭიროებს კომპლექსურ შესწავლას ასეთი მიდგომის განსახორციელებლად. მაგრამ მნიშვნელოვანი ნაწილი არა პატერნებია, არამედ კოდის მომზადება ისე, რომ ის შეესაბამებოდეს ბიზნეს პრობლემებს და იყენებდეს იგივე ბიზნეს ტერმინებს (საყოველთაო ენას). ამასთანავე, DDD მიდგომა უნდა იქნას გამოყენებული იმ შემთხვევაში, თუ გვაქვს რთული მიკროსერვისები მნიშვნელოვანი ბიზნეს წესებით [15, 17].

ვინაიდან სასწავლო დაწესებულებებში მიმდინარე პროცესების ავტომატიზაციის მიზნით უნდა გავითვალისწინოთ მრავალი სხვადასხვა სახის ფუნქციონალი, საჭიროა მათი ანალიზი და კლასიფიკაცია. შემდეგ მათი ფუნქციური დანიშნულების შესაბამისად უნდა მოვაქციოთ საერთო გამოყენების მიკროსერვისებში.

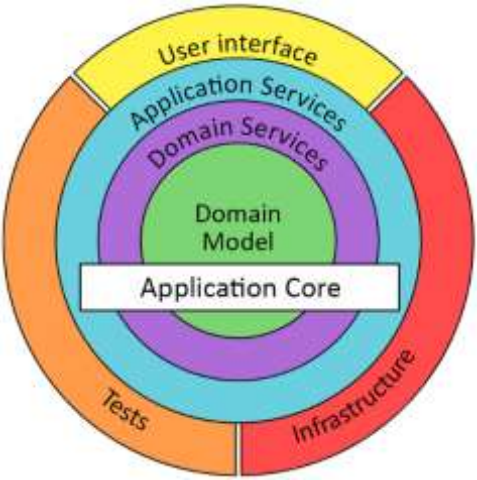
პირველ რიგში, თავიდანვე უნდა ვეცადოთ შევქმნათ რაც შეიძლება ნაკლები მიკროსერვისი, თუმცა ეს არ უნდა იყოს მთავარი მიზანი, რადგან უნდა ვეცადოთ ერთ მიკროსერვისში გავაერთიანოთ საერთო იდეის მატარებელი ნაწილები. ჩვენი ამოცანის გადაწყვეტისას ჯერ უნდა შევძლოთ საერთო სავარაუდო მიკროსერვისების წარმოდგენა.

შესაძლებელია შემდეგ ამ მიკროსერვისებს შორის ხდებოდეს კომუნიკაცია. საჭიროა მაქსიმალურად დაბალანსდეს მიკროსერვისების ურთიერთგამოყენების გზები. კერძოდ, თუ ორ მიკროსერვისს სჭირდება ხშირი ურთიერთქმედება, მაშინ ისინი ერთ მიკროსერვისში უნდა მოთავსდეს [14, 85].

ჩვენი საპრობლემო სფერო საჭიროებს დიდი ზომის მონაცემების შენახვას და შემდეგ მათ დამუშავებას. ამიტომ აუცილებელია მაქსიმალურად მოხდეს მათზე ზემოქმედება და შემდეგ შესაბამისი ბიზნესლოგიკის სწორად გაწერა.

არქიტექტურულად, DDD ფენები მიკროსერვისებში ლოგიკური ნიმუშებია და არაა დაკავშირებული სერვისების გამოყენებასთან (ნახ. 3.2) [81, 88]. ისინი არსებობს, რათა დაეხმაროს დეველოპერს კოდის სირთულის მართვაში.

ნახაზზე ასახული მოდელის დონეები სხვადასხვა ტიპისაა, რაც მოითხოვს ტიპთა გარდაქმნის პროცედურების რეალიზაციის ინსტრუმენტების არსებობას [15, 86].



ნახ. 3.2. DDD არქიტექტურის მოდელი

- *საგნობრივი სფეროს (დომენის) მოდელის ფენა (Domain Model Layer)*

დომენის მოდელის ფენა (ან შრე): პასუხისმგებელია ბიზნესის კონცეფციების, ბიზნეს სიტუაციის შესახებ ინფორმაციისა და ბიზნესის წესების წარმოდგენაზე. მდგომარეობა, რომელიც ასახავს ბიზნეს მდგომარეობას, აქ კონტროლდება და გამოიყენება, მიუხედავად იმისა, რომ მისი შენახვის ტექნიკური დეტალები დელეგირებულია ინფრასტრუქტურაზე. ეს ფენა არის ბიზნეს პროგრამული უზრუნველყოფის გული.

დომენის მოდელის ფენა არის სადაც ბიზნესი გამოხატულია. როდესაც თქვენ დანერგავთ მიკროსერვისის დომენის მოდელის ფენას .NET-ში, ეს ფენა დაშიფრულია, როგორც კლასის ბიბლიოთეკა დომენის (საგნობრივი სფეროს) არსებით, რომლებიც აგროვებენ მონაცემებს და ქცევას (ლოგიკას მეთოდებით).

მდგრადობისა და ინფრასტრუქტურის იგნორირების პრინციპების შესაბამისად, ამ ფენამ მთლიანად უნდა უგულებელყოს მონაცემთა მდგრადობის დეტალები. მდგრადობის ეს ამოცანები უნდა შესრულდეს ინფრასტრუქტურის ფენის მიერ. ამიტომ, ამ ფენას არ უნდა ჰქონდეს პირდაპირი დამოკიდებულება ინფრასტრუქტურაზე, რაც ნიშნავს, რომ მნიშვნელოვანი წესია, რომ დომენის მოდელის არსის კლასები უნდა იყოს POCO (Plain Old Class Objects). ესაა .NET-ის მონაცემთა სტრუქტურა, რომელიც შეიცავს მხოლოდ საჯარო თვისებებს ან ველებს. POCO არ უნდა შეიცავდეს სხვა წევრებს, როგორიცაა: მეთოდები, მოვლენები და დელეგატები.

დომენის არსებს არ უნდა ჰქონდეს რაიმე პირდაპირი დამოკიდებულება მონაცემთა წვდომის ნებისმიერ ინფრასტრუქტურზე.

- ***აპლიკაციის ფენა*** *(Application Layer)* განსაზღვრავს დავალებებს, რომლებიც უნდა შეასრულოს პროგრამულმა უზრუნველყოფამ (აპლიკაციამ). ეს ამოცანები მნიშვნელოვანია ბიზნესისთვის ან აუცილებელია სხვა სისტემების აპლიკაციის ფენებთან ურთიერთობისთვის. აპლიკაციის ფენა არ შეიცავს ბიზნესწესებს ან ცოდნას. იგი მხოლოდ კოორდინაციას უწევს ამოცანებს და დელეგირებს დომენის ფენის სამუშაოს მომდევნო ფენაში. მას არ აქვს ბიზნეს სიტუაციის ასახავის მდგომარეობები. მაგრამ შეიძლება ჰქონდეს მდგომარეობა, რომელიც ასახავს ამოცანის შესრულების კოდს მომხმარებლის ან პროგრამისთვის.

მიკროსერვისის აპლიკაციის ფენა .NET-ში ჩვეულებრივ კოდირებულია, როგორც ASP.NET Core Web API პროექტი. იგი ახორციელებს მიკროსერვისების ურთიერთქმედებას, დისტანციურ ქსელში წვდომას და გარე ვებ API-ებს, რომლებიც გამოიყენება UI ან კლიენტის აპებიდან; მიკროსერვისის მიერ მიღებულ ბრძანებებს და მიკროსერვისებს შორის მოვლენებზე ორიენტირებულ კომუნიკაციას (ინტეგრაციის მოვლენები). ASP.NET Core Web API, რომელიც

არის აპლიკაციის ფენა, არ უნდა შეიცავდეს ბიზნესწესებს ან დომენის ცოდნას (განსაკუთრებით დომენის წესებს ტრანზაქციების ან განახლებისთვის); ისინი უნდა ფლობდეს დომენის მოდელის კლასის ბიბლიოთეკას. აპლიკაციის ფენამ მხოლოდ ამოცანების კოორდინაცია უნდა მოახდინოს. არ უნდა ჰქონდეს (ან განსაზღვროს) საგნობრივი სფეროს რაიმე მდგომარეობა (დომენის მოდელი). იგი ბიზნესწესების შესრულებას ავალებს თავად დომენის მოდელის კლასებს (აგრეგირებული ფესვები და დომენის არსები), რაც საბოლოოდ განაახლებს მონაცემებს ამ დომენის არსებში (Entities).

ძირითადად, აპლიკაციის ლოგიკა არის – სადაც ხდება ყველა გამოყენებითი შემთხვევის (Use case) რეალიზება (რომლებიც დამოკიდებულია ამ გარე ინტერფეისზე). მაგალითად, იმპლემენტაცია, რომელიც დაკავშირებულია Web API სერვისთან.

მიზანია, რომ საგნობრივი სფეროს ლოგიკა დომენის მოდელის ფენაში, მისი ინვარიანტები, მონაცემთა მოდელი და მასთან დაკავშირებული ბიზნესწესები იყოს სრულიად დამოუკიდებელი პრეზენტაციისა და აპლიკაციის ფენებისგან. უპირველეს ყოვლისა, დომენის მოდელის ფენა პირდაპირ არ უნდა იყოს დამოკიდებული რაიმე ინფრასტრუქტურაზე [15].

- ***ინფრასტრუქტურის ფენა*** (Infrastructure Layer) - არის ის, თუ როგორ ხდება საგნობრივი სფეროს (დომენის) არსთა შესაბამისი მეხსიერებიდან მონაცემთა შენახვა მონაცემთა ბაზაში ან სხვა მუდმივ საცავში. ამის მაგალითია Entity Framework Core კოდის გამოყენება Repository-ის ნიმუშის კლასების დასანერგად, რომლებიც იყენებს DBContext-ს რელაციურ მონაცემთა ბაზაში მონაცემების შესანარჩუნებლად. მდგრადობისა (Persistence) და ინფრასტრუქტურის იგნორირების ზემოხსენებული პრინციპების შესაბამისად, ინფრასტრუქტურის ფენა არ უნდა „აბინძურებდეს"

საგნობრივი სფეროს (დომენის) მოდელის ფენას. საჭიროა დომენის მოდელის არსთა კლასების დამოუკიდებლად შენახვა იმ ინფრასტრუქტურისგან, რომელიც ახორციელებს მონაცემთა შენახვას. დომენის მოდელის ფენის კლასთა ბიბლიოთეკაში უნდა იყოს მხოლოდ ამ დომენის კოდი, მხოლოდ POCO არსების კლასები, რომლებითაც იქმნება პროგრამული უზრუნველყოფის „გული" და გამოყოფილია ინფრასტრუქტურული ტექნოლოგიებისგან.

ამგვარად, ჩვენ დავახასიათეთ DDD-არქიტექტურის შრეები. ახლა კი შევაჯამოთ DDD-ს ძირითადი კომპონენტები [87]:

1) საერთო ენა Ubiquitous Language) - იყენებენ როგორც საგნობრივი სფეროს (დომენის) ექსპერტები, ასევე დეველოპერები. იგი საჭიროა მოდელის განსაზღვრისა და საგნობრივ სფეროში კომუნიკაციისათვის. ენა უნდა ეფუძნებოდეს დომენის ტერმინებსა და ცნებებს და უნდა იყოს გამოყენებული თანმიმდევრულად მთელი პროექტის განმავლობაში;

2) შეზღუდული კონტექსტი (Bounded Context) - მკაფიოდ განსაზღვრული საზღვარები, რომლის ფარგლებშიც გამოიყენება კონკრეტული მოდელი. სხვადასხვა კონტექსტს შეიძლება ჰქონდეს საკუთარი მოდელები და მათ შორის ურთიერთობები ფრთხილად უნდა კონტროლირდებოდეს, რათა თავიდან იქნას აცილებული შეუსაბამობები და დაბნეულობა.

3) არსები (Entities) - საგნობრივი სფეროს (დომენის) მოდელის ობიექტებია, რომლებსაც აქვს უნიკალური იდენტურობა და სასიცოცხლო ციკლი. არსები გამოიყენება დომენის ძირითადი კონცეფციების წარმოსაჩენად და მათთან დაკავშირებული ბიზნესწესების ინკაფსულაციისთვის;.

4) ობიექტთა მნიშვნელობა (Value Objects). ობიექტები, რომლებიც ატრიბუტებია ან გაზომვები დომენში, მაგრამ არ აქვთ უნიკალური იდენტურობა. ისინი უცვლელია და თავისუფლად შეიძლება შეიცვალოს ექვივალენტური ეგზემპლარებით;

5) აგრეგატები (Aggregates) - არის არსების და ობიექტთა მნიშვნელობის დაკავშირებული კლასტერი, რომლებიც განიხილება როგორც ერთი მთლიანი. აგრეგატები პასუხისმგებელია საკუთარი უცვლელობის ან ბიზნესწესების შენარჩუნებაზე და ეხმარება მოდელის ფარგლებში თანმიმდევრულობის დაცვას.

6) საცავები (Repositories) - აბსტრაქციები, რომლებიც მართავენ აგრეგატების მდგრადობას და მოძიებას, რაც უზრუნველყოფს მკაფიო გამიჯვნას საგნობრივ სფეროს (დომენის) მოდელსა და მონაცემთა შენახვის საბაზო მექანიზმს შორის.

7) საგნობრივი სფეროს მოვლენები (Domain Events:) - ასახავს დომენის მდგომარეობის მნიშვნელოვან ცვლილებას. მათი გამოყენება შეიძლება სისტემის დასაყოფად ნაწილებად და მოვლენებზე ორიენტირებული არქიტექტურის გასააქტიურებლად.

8) საგნობრივი სფეროს სერვისები (Domain Services) - სერვისები, მდგომარეობათა შენახვის გარეშე, რომლებიც აერთიანებს დომენურ ლოგიკას, რომელიც ბუნებრივად არ ჯდება არსებში ან ობიექტის მნ იშვნელობებში.

ამ პრინციპებისა და შაბლონების დაცვით, Domain-Driven Design მეთოდოლოგია მიზნად ისახავს შექმნას პროგრამული სისტემები, რომლებიც უფრო მდგრადი, მოქნილი და შესაბამისი იქნება ბიზნესის მოთხოვნების, რაც საბოლოოდ გაუმჯობესებს პროგრამული პროდუქტის საერთო ხარისხს და ეფექტიანობას.

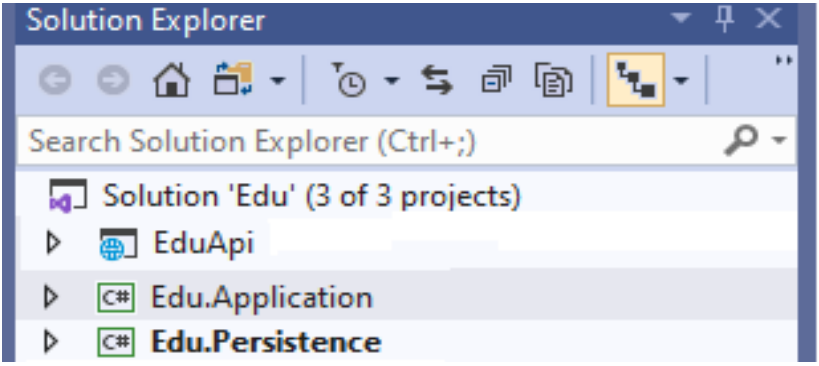
შეიძლება ითქვას, რომ სუფთა არქიტექტურის მეთოდოლოგიის ნაირსახეობაა DDD-მიდგომა. განსხვავება კი ისაა, რომ სუფთა არქიტექტურის მთავარი საზრუნავია პროექტებისთვის პრობლემების გამიჯვნა და აპლიკაციის ტესტირება. ხოლო DDD-სთვის კი – პროგრამული პროდუქტის დიზაინის აგება საგნობრივი სფეროს (დომენის) სირთულის გათვალისწინებისა და მის ექსპერტებთან (და საბოლოო მომხმარებლებთან) მჭიდრო თანამშრომლობით.

3.3. DDD-ს იმპლემენტაცია სისტემაში

სასწავლო დაწესებულებებში მიმდინარე პროცესების ავტომატიზებული მართვისთვის აუცილებელია არსებული მონაცემების სწორად ჩაწერა/წაკითხვა და მათი დამუშავება. საჭიროა კარგად გვესმოდეს გადასაწყვეტი არსებული ამოცანა და შეგვეძლოს მის გადასაჭრელად ადექვატური მოდელების შექმნა. იმისთვის რომ ზუსტად დავყოთ დონეებად ჩვენი პროექტი, ვირჩევთ დომენზე ორიენტირებულ დიზაინს (DDD), რომელიც მთლიანად იქნება მორგებული საგნობრივი სფეროს საქმიანი პროცესების რეალობაზე. აგებული აპლიკაციები ერთმანეთთან კავშირს დაამყარებს DDD მეთოდოლოგიის საფუძველზე.

დეტალურად განვიხილოთ ჩვენი სისტემისთვის შექმნილი DDD არქიტექტურის შრეები (ფენები - Layers).

3.3 ნახაზზე ნაჩვენებია Visual Studio.NET-ში აგებული Edu პროექტის Solution Explorer- ის ფრაგმენტი.



ნახ. 3.3. DDD შრეები

ჩვენ შევქმენით DDD-არქიტექტურის სამი შრე: Persistence (მდგრადობის), Application (დანართის) და Api (აპლიკაციის პროპგრამირების ინტერფეისის).

3.3.1. მდგრადობის შრე

მდგრადობის (Persistence) შრე: პროგრამული აპლიკაციის ასაგებად ვიყენებთ SQL Server მონაცემთა რელაციურ ბაზას თავისი Entity Framework-ით (EF). ესაა ობიექტ-რელაციური წარმოდგენის ფრეიმვორკი (Object Relational Mapping - ORM) [89].

იგი სთავაზობს დეველოპერებს ბაზაში მონაცემთა შენახვისა და წვდომის ავტომატიზებულ მექანიზმს. მისი საშუალებით ხორციელდება აპლიკაციისთვის მისაღები მდგრადობის შრის შემუშავება და დანერგვა. EF მხარს უჭერს მოთხოვნების ინტეგრირებულ ენას (LINQ – Language Integrated Query) და უზრუნველყოფს მკაცრად ტიპიზირებულ ობიექტებს მოდელში, აგრეთვე მონაცემთა ბაზის გამარტივებულ მდგრადობას [90, 91].

Persistence შრეს ვიყენებთ ბაზასთან საკომუნიკაციოდ. შესაბამის ნაწილში ხდება ნებისმიერი მოთხოვნის გაგზავნა ბაზაში და შესაბამისი პასუხების მიღება. Persistence - ში გვაქვს DBObject, რომელშიც კლასების სახით აღვწერთ მონაცემთა ბაზის ცხრილებს.

აქვეა რეპოზიტორებიც, რომლებიც იქმნება ბაზის ყოველი ცხრილისთვის. სწორედ რეპოზიტორებში ხდება შესაბამის ცხრილებზე სხვადასხვა სახის ოპერაციების განხორციელება. Repository პატერნის გამოყენებისას ხდება მონაცემების საცავში განთავსების და მათი ამოღების შესახებ დეტალების დამალვა. მონაცემთა საცავი შეიძლება იყოს მონაცემთა ბაზა, xml ფაილი და ა.შ. [92-95]. ჩვენ შემთხვევაში ეს არის Ms SQL Server.

განვილიხილოთ ერთი კონკრეტული მონაცემთა ბაზის ობიექტი Marks, რომელსაც Persistence შრის DBObject-ში აქვს შესაბამისი კლასი :

```
public class Mark
  {
    public int markID { get; set; }
```

```
        public int cseId { get; set; }
        public int pupilId { get; set; }
        public int mark { get; set; }
        public DateTime markDate { get; set; }
        public DateTime recDate { get; set; }
        public int markTypeId { get; set; }
        public int userId { get; set; }
        public int branchId { get; set; }
    }
```

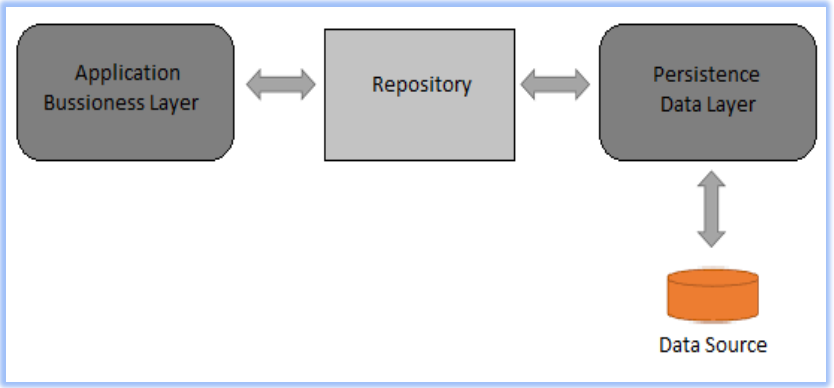
ასევე აქვე გვაქვს შესაბამის ობიექტზე ზემოქმედებისთვის საჭირო რეპოზიტორი და შესაბამისი ინტერფეისი :

```
public interface IMarkRepository
{
    Task<Mark> GetMarkById(int markId, int branchId);
    Task<List<Mark>> GetMarks(int branchId);
    Task<int> SaveMark(Mark mark);
}

  public class MarkRepository : IMarkRepository
    {
      private IConfiguration Configuration { get; }
      public IDbConnection Connection => new  SqlConnection
        (Configuration.GetConnectionString("EduApiConnectionString"));
      public MarkRepository(IConfiguration configuration)
      {
          Configuration = configuration;
      }
      public async Task<Mark> GetMarkById(int markId, int branchId)
      {
          using var cn = Connection;
```

```
        var queryString = $"SELECT * FROM dbo.Marks(NOLOCK)
WHERE MarkId ={markId}";
        cn.Open();
        var result = await cn.QueryAsync<Mark>(queryString);
        // ცვლადის ტიპი დგინდება კომპილაციის დროს -----
        return result.FirstOrDefault();
    }

    public async Task<List<Mark>> GetMarks(int branchId)
    {
        using var cn = Connection;
        var queryString = $"SELECT * FROM dbo.Marks(NOLOCK)
WHERE BranchId =

{branchId}";
        cn.Open();
        var result = await cn.QueryAsync<Mark>(queryString);
        return result.ToList();
    }

    public async Task<int> SaveMark(Mark Mark)
    {
        using var cn = Connection;
        cn.Open();
        var p = new DynamicParameters();
        p.Add("markID", Mark.markID);
        p.Add("cseId", Mark.cseId);
        p.Add("pupilId", Mark.pupilId);
        p.Add("mark", Mark.mark);
        p.Add("markDate", Mark.markDate);
```

```
            p.Add("recDate", Mark.recDate);
            p.Add("status", Mark.status);
            p.Add("markTypeId", Mark.markTypeId);
            p.Add("userId", Mark.userId);
            p.Add("branchId", Mark.branchId);
            p.Add("RecId",
dbType:DbType.Int32,direction:ParameterDirection.Output);
            await cn.ExecuteAsync("dbo.AddEditMark", p, commandType:
CommandType.StoredProcedure);
            return p.Get<int>("RecId");
        }
    }
```

Persistence შრეში მიღებულ მონაცემებს უკვე ვიყენებთ Application Layer-ში (ნახ. 3.4.)

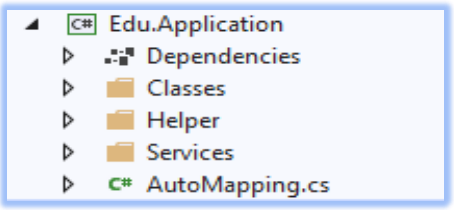


ნახ. 3.4. Persistence, Application შრის და ბაზის კავშირი

3.3.2. აპლიკაციის შრე

Application მოდელის შრე: პასუხისმგებელია ბიზნესის კონცეფციების, ბიზნეს სიტუაციის შესახებ ინფორმაციისა და ბიზნესის წესების წარმოდგენაზე. შესაბამის შრეში ხდება ბიზნეს ლოგიკის გაწერა, მაშინაც კი, როცა ტექნიკური დეტალები შესანახად გადაეცემა Persistence შრეს.

აპლიკაციის შრე არის ბიზნესის პროგრამული უზრუნველყოფა. როდესაც ხდება .NET-ში მიკროსერვისების დომენის (საგნობრივი სფეროს) მოდელის ფენის იმპლემენტაცია, ეს ფენა იწერება, როგორც კლასის ბიბლიოთეკა დომენის ობიექტებით, რომლებშიც ხდება მეთოდების გაწერა ბიზნესის შესაბამისი ლოგიკის მიხედვით (ნახ. 3.5).



ნახ. 3.5. Application შრე

შესაბამის შრეში გვაქვს ობიექტების შესაბამისი კლასები, დამხმარე კლასები და სერვისები (ინტერფეისები და შესაბამისი იმპლემენტაციები). სწორედ სერვისების კლასებში გვაქვს გაწერილი ბიზნეს ლოგიკები.

მნიშვნელოვანი კლასია AutoMapping, რომელშიც ხდება Persistence და Application შრეებში არსებული ტიპების თარგმანი. მაგალითად, Application Layer-ში გვაქვს Persistence Layer -ში არსებული Mark კლასის მსგავსი კლასი - Mark, რომელსაც მხოლოდ პირველი ველი აქვს განსხვავებული (markID-ის ნაცვლად არის ID ველი) :

```
public class Mark
  {
     public int ID { get; set; }
     public int cseId { get; set; }
     public int pupilId { get; set; }
     public int mark { get; set; }
     public DateTime markDate { get; set; }
     public DateTime recDate { get; set; }
     public int markTypeId { get; set; }
     public int userId { get; set; }
     public int branchId { get; set; }
  }
```

სწორედ ამ სხვაობის გამო აუცილებელია Persistence კლასი გადავთარგმნოთ Application კლასისთვის:

```
CreateMap<Edu.Persistence.DBObjects.Mark,
Edu.Application.Classes.Mark.Mark>()
     .ForMember(dest => dest.ID, opt => opt.MapFrom(src =>
src.markID))
     .ReverseMap();
```

აქვე გვაქვს ბიზნეს ლოგიკა, Mark კლასის მეთოდების სერვისი და შესაბამისი ინტერფეისი Application შრეში:

```
public interface IMarkService
  {
     Task<ResponseData<Mark>> GetMarkById(int MarkId,
ClaimUserInfo claimUserInfo);
     Task<ResponseData<List<Mark>>> GetMarks(ClaimUserInfo
claimUserInfo);
```

```
        Task<ResponseData<List<Mark>>> GetMarksByPupil(int pupilId,
ClaimUserInfo
claimUserInfo);
        Task<ResponseData<List<Mark>>> GetMarksByEmployees(int
pupilId, ClaimUserInfo claimUserInfo);
        Task<ResponseData<List<Mark>>> GetMarksBySubject(int pupilId,
ClaimUserInfo claimUserInfo);
        Task<ResponseData<List<Mark>>> GetMarksBySemester(int
pupilId, ClaimUserInfo claimUserInfo);
        Task<ResponseData<List<Mark>>> GetMarksHistoryBySubject(int
pupilId, ClaimUserInfo claimUserInfo);
        Task<ResponseData<List<Mark>>> GetMarksHistoryByPupil(int
pupilId, ClaimUserInfo claimUserInfo);
        Task<ResponseData<List<Mark>>> GetMarksHistoryByEmplyee(int
pupilId, ClaimUserInfo claimUserInfo);
        Task<ResponseData<List<Mark>>> GetMarkDescription(int pupilId,
ClaimUserInfo claimUserInfo);
        Task<ResponseData<Mark>> SaveMark(Mark Mark, ClaimUserInfo
claimUserInfo);
        Task<ResponseData<int>> DeleteMark(int MarkId, ClaimUserInfo
claimUserInfo);
    }
```

შესაბამისი ინტერფეისის იმპლემენტაცია :

```
public class MarkService : IMarkService
    {
        private readonly IMarkRepository _repo;
        private readonly ILogService _log;
        private readonly IMapper _mapper;
```

```
        public MarkService(IMarkRepository repo, IMapper mapper,
ILogService log)
        {
            _repo = repo;
            _mapper = mapper;
            _log = log;
        }

        public async Task<ResponseData<Mark>> GetMarkById(int markId,
ClaimUserInfo claimUserInfo)
        {
            try
            {
                var Mark = await _repo.GetMarkById(markId,
claimUserInfo.BranchId);
                var p = _mapper.Map<Mark>(Mark);
                return new ResponseData<Mark>(p);
            }
            catch (Exception ex)
            {
                _log.LogError($"GetMarkById({markId}) exception
:{Environment.NewLine} {ex.Message}");

                return new ResponseData<Mark>
                {
                    Error = new ErrorProvider
                    {
                        Message = "exception",
                        StatusCode =
System.Net.HttpStatusCode.ExpectationFailed
```

```
            }
        };
    }
}
public async Task<ResponseData<List<Mark>>>
GetMarksByPupil(int pupilId, ClaimUserInfo claimUserInfo)
{
    var list = (await
_repo.GetMarks(claimUserInfo.BranchId)).Where(m => m.pupilId ==
pupilId);
    var result = list.Select(x => _mapper.Map<Mark>(x)).ToList();
    return new ResponseData<List<Mark>>(result);
}

public async Task<ResponseData<List<Mark>>>
GetMarks(ClaimUserInfo claimUserInfo)
{
    var list = await _repo.GetMarks(claimUserInfo.BranchId);
    var result = list.Select(x => _mapper.Map<Mark>(x)).ToList();
    return new ResponseData<List<Mark>>(result);
}
public async Task<ResponseData<Mark>> SaveMark(Mark mark,
ClaimUserInfo claimUserInfo)
{
    var recId = await
_repo.SaveMark(_mapper.Map<Persistence.DBObjects.Mark>(mark));
    return await GetMarkById(recId, claimUserInfo);
}
...
}
```

ClaimUserInfo – კი არის დამხმარე კლასი, რომელიც ივსება სისტემის მომხმარებლის ავტორიზაციის შედეგად და მასში ინახება შემდეგ ინფორმაცია:

```
public class ClaimUserInfo
  {
     public string JtiToken { get; set; }
     public int UserId { get; set; }
     public int BranchId { get; set; }
     public string UserName { get; set; }
     public string ApAddress { get; set; }
  }
```

3.3.3. Api შრე

აპლიკაციის პროგრამირების ინტერფეისი (Application Programming Interface – Api) არის ორი ან მეტი კომპიუტერული პროგრამის ერთმანეთთან კომუნიკაციის საშუალება.

Api შრე განსაზღვრავს დავალებებს, რომლებიც პროგრამულმა უზრუნველყოფამ უნდა შეასრულოს. ამოცანები, რომლებზეც პასუხისმგებელია ეს ფენა, მნიშვნელოვანია ბიზნესისთვის ან საჭიროა სხვა სისტემების Api შრეებთან ურთიერთქმედებისთვის.

იგი არ შეიცავს ბიზნესის წესებსა და ცოდნას, მაგრამ კოორდინაციას უწევს ამოცანებს. როგორც უკვე აღვნიშნეთ, შესაბამის შრეში არ ხდება ბიზნეს ლოგიკის გაწერა, მაგრამ აქ შეგვიძლია სხვადასხვა სახის ამოცანის შესრულების ასახვა.

ეს შრე .NET - ში, როგორც წესი, იწერება როგორც ASP.NET Core Web API - ის პროექტი. ის ახორციელებს მიკროსერვისების ურთიერთქმედებას, დისტანციურ ქსელურ წვდომას და გარე Web API- ს, რომელიც გამოიყენება UI ან კლიენტის აპებიდან.

შესაბამის API შრეში გვაქვს კონტროლერები, მაგალითად განვიხილოთ კვლავ Mark ობიექტის კონტროლერი:

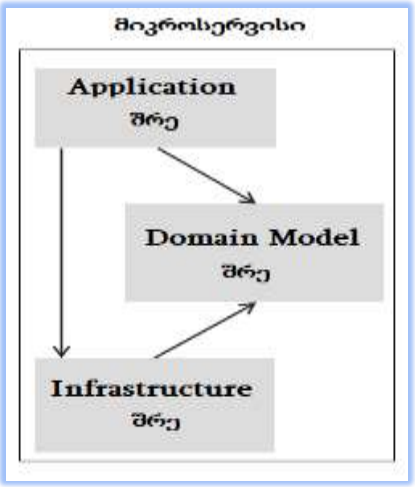
```
[Route("api/[controller]/[action]")]
  [ApiController]
  [Authorize]
  [EnableCors("AllowAll")]
  public class MarkController : ControllerBase
  {
    private readonly IMarkService _MarkService;
    public MarkController(IMarkService MarkService)
    {
      _MarkService = MarkService;
    }

    [HttpGet("{id}")]
    public async Task<ResponseData<Mark>> GetMark(int id)
    {
      var claimUserInfo =
ClaimHelper.GetClaimUserInfo(HttpContext.User);
      return await _MarkService.GetMarkById(id, claimUserInfo);
    }

    [HttpGet]
    public async Task<ResponseData<List<Mark>>> GetMarks()
    {
      var claimUserInfo =
ClaimHelper.GetClaimUserInfo(HttpContext.User);
      return await _MarkService.GetMarks(claimUserInfo);
    }
```

```
    [HttpPost]
    public async Task<ResponseData<Mark>> AddMark([FromBody]
Mark Mark)
    {
        var claimUserInfo =
ClaimHelper.GetClaimUserInfo(HttpContext.User);
        Mark.ID = 0;
        return await _MarkService.SaveMark(Mark, claimUserInfo);
    }

    [HttpPost]
    public async Task<ResponseData<bool>> AddMarks([FromBody]
List<Mark> Marks)
    {
        var claimUserInfo =
ClaimHelper.GetClaimUserInfo(HttpContext.User);
        foreach (Mark Mark in Marks)
        {
            Mark.ID = 0;
            await _MarkService.SaveMark(Mark, claimUserInfo);
        }

        return new ResponseData<bool>(true);
    }

    [HttpPut("{id}")]
    public async Task<ResponseData<Mark>> EditMark(int id,
[FromBody] Mark Mark)
```

```
    {
        var claimUserInfo =
ClaimHelper.GetClaimUserInfo(HttpContext.User);
        Mark.ID = id;
        return await _MarkService.SaveMark(Mark, claimUserInfo);
    }

    [HttpPut("{id}")]
    public async Task<ResponseData<Mark>> DeleteMark(int id, bool
deleteStatus)
    {
        var claimUserInfo =
ClaimHelper.GetClaimUserInfo(HttpContext.User);
        Mark Mark = _MarkService.GetMarkById(id,
claimUserInfo).Result.Result;
        if (Mark != null)
        {
            Mark.status = deleteStatus;
        }
        return await _MarkService.SaveMark(Mark, claimUserInfo);
    }
            ...
}
```

3.3.4. შრეთაშორის დამოკიდებულებები DDD სერვისში

3.6 ნახაზზე წარმოდგენილია DDD-ს ზემოგანხილული სამი შრის დამოკიდებულებები მიკროსერვისებში [15].



ნახ.3.6. შრეებს შორის დამოკიდებულებები

აპლიკაციის შრე დამოკიდებულია დომენზე და ინფრასტრუქტურაზე, ხოლო ინფრასტრუქტურის შრე დამოკიდებულია დომენზე, მაგრამ დომენის შრე არაა დამოკიდებული არცერთ შრეზე. დიზაინის ეს შრე დამოუკიდებელი უნდა იყოს თითოეული მიკროსერვისისთვის.

როგორც უკვე აღინიშნა, შეიძლება რეალიზებულ იქნას ყველაზე რთული მიკროსერვისები DDD შაბლონების მიხედვით, ხოლო მონაცემების მართვის მარტივი მიკროსერვისები (მაგალითად, CRUD ოპერაციები ერთ შრეში) გადაწყდეს უფრო მარტივი საშუალებით.

.3.4. დომენის და აპლიკაციის სერვისები

განსხვავება დომენის სერვისსა და აპლიკაციის სერვის შორის მცირეა, მაგრამ ძალზე მნიშვნელოვანი:

– დომენური (ანუ საგნობრივი სფეროს) სერვისები მეტად დაწვრილებითია, მაშინ როცა აპლიკაციის სერვისი წარმოადგენს ფასადს, რომლის მიზანია API- ს უზრუნველყოფა;

– დომენური სერვისები შეიცავს ბიზნესლოგიკას, მაშინ როცა აპლიკაციის სერვისები თავად არ ახდენს ბიზნესლოგიკის შესრულებას, არამედ მხოლოდ უზრუნველყოფს მისი შესრულებისთვის საჭირო ნაწილების ორგანიზებას;

– დომენის სერვისების მეთოდებს შეიძლება ჰქონდეს სხვა დომენის ელემენტები, როგორც ოპერანდები და დასაბრუნებელი მნიშვნელობები, ხოლო Application Service მოქმედებს ტრივიალურ ოპერანდებზე, როგორიცაა identity მნიშვნელობები და პრიმიტიულ მონაცემთა სტრუქტურები;

– აპლიკაციის სერვისები დამოკიდებულია ინფრასტრუქტურულ სერვისებზე, რათა შეძლოს დომენური ლოგიკების შესრულება;

– აპლიკაციების სერვისები ქმნის API-ს, რომელიც აერთიანებს პროგრამის ბირთვს, ხოლო დომენზე ორიენტირებული დიზაინის შემთხვევაში, ისინი საბოლოოდ ორგანიზებას უწევს და დელეგირებს ძირითად ობიექტებს, ობიექტის მნიშვნელობასა და დომენის სერვისებს [16].

ნებისმიერი ამოცანის გადასაჭრელად შესაძლებელია სხვადასხვა მიდგომის არჩევა და გამოყენება, მთავარია, რომელი მოერგება ჩვენ მოთხოვნებს ყველაზე მეტად. ვინაიდან სასწავლო დაწესებულებებში მიმდინარე პროცესების ავტომატიზებული მართვის სისტემაში უნდა გავითვალისწინოთ მრავალი სხვადასხვა

სახის ფუნქციონალი. ამიტომ აზრობრივად უნდა დავყოთ შესაბამისი ფუნქციები და ისინი, რომლებიც ერთსადაიმავე იდეურ ფარგლებში ხვდება, უნდა მოვაქციოთ საერთო სივრცეში.

ამიტომ გადავწყვიტეთ, რომ დომენზე ორიენტირებული დიზაინის გამოყენება. ყველაზე მეტად დაგვეხმარებოდა, რადგან ის მხარს უჭერს მოდელირებას, რომელიც დაფუძნებულია ბიზნესის რეალობაზე, რაც შეესაბამება ჩვენს მოთხოვნებს.

იგი აღწერს დამოუკიდებელ პრობლემურ ზონებს, როგორც შემოსაზღვრულ კონტექსტებს (თითოეული შემოსაზღვრული კონტექსტი უკავშირდება მიკრო მომსახურებას) და უზრუნველ-ყოფს საერთო სასაუბრო ენას [37-39].

თავი 4
სასწავლო პროცესის მართვის საინფორმაციო სისტემის დაპროექტება და აგება

ორგანიზაციული მართვის (მენეჯმენტის) საინფორმაციო სისტემების ასაგებად, როგორც მეორე თავში აღვნიშნეთ, პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის ეტაპების შესაბამისად, უპირველეს ყოვლისა საჭიროა საგნობრივი სფეროს ბიზნეს-მოთხოვნილებების განსაზღვრა, ანუ რა ფუნქციური ამოცანები უნდა გადაწყვიტოს ავტომატიზებულმა სისტემამ. ამის შემდგეგ შესაძლებელია სისტემის მხარდამჭერი ინფრასტრუქტურის (არქიტექტურის) დაპროექტება, შესაბამისი ინფორმაციული და პროგრამული უზრუნველყოფით. ჩვენ შემთხვევაში ვიხილავთ საგანმანათლებლო დაწესებულების (სკოლა, უნივერსიტეტი და სხვ.) ამოცანებს. კერძოდ, საჯარო სკოლის მაგალითზე. მათი გამოყენება შესაძლებელია უნივერსიტეტის და კოლეჯისთვისაც, შესაბამისი მცირე ადაპტაციის საფუძველზე.

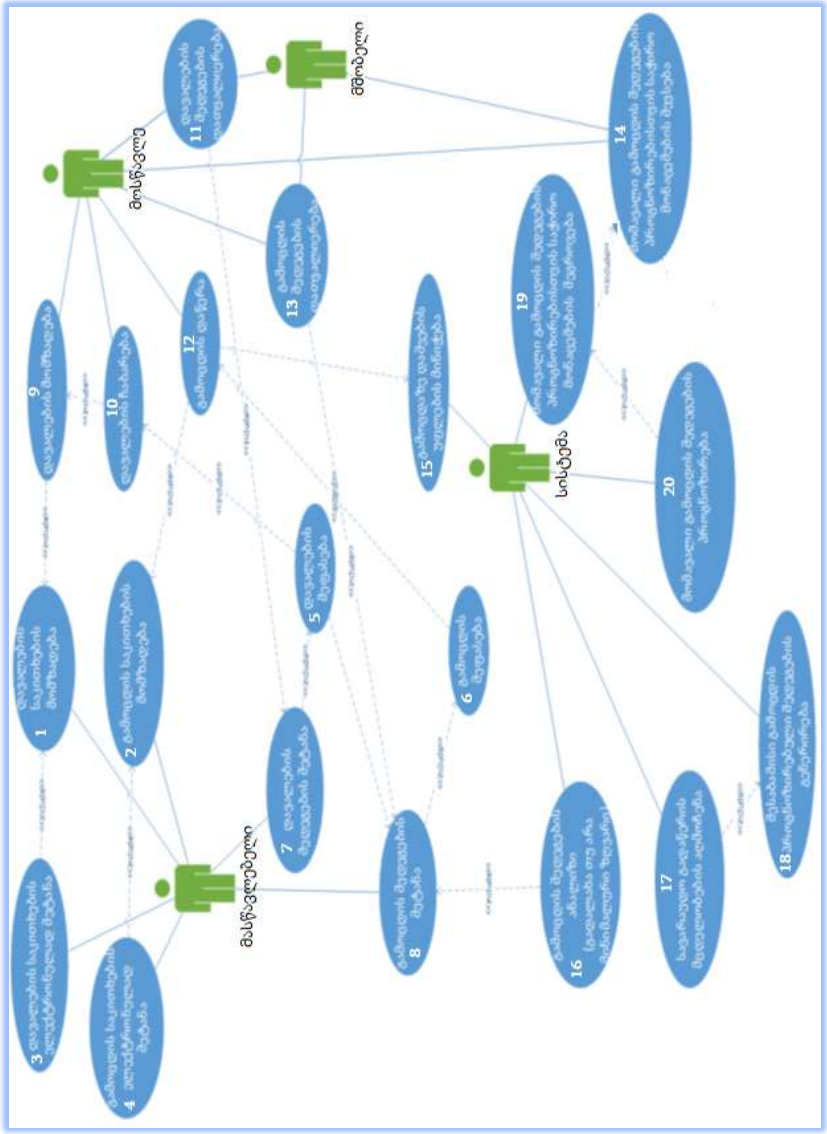
4.1. მოსწავლის შეფასების პროცესის უნიფიცირებული მოდელირება

განვიხილოთ საჯარო სკოლის მაგალითზე ერთ-ერთი მნიშვნელოვანი საქმიანი პროცესის, კერძოდ მასწავლებლის მიერ მოსწავლეთა შეფასების ამოცანა, დავალებების და საგამოცდო საკითხების შესრულების საფუძველზე. საჭიროა განისაზღვროს ამ პროცესის განსახორციელებლად აუცილებელი რესურსები: როლები (ვინ მონაწილეობს), ფუნქციები (რა ბიზნესპროცესები სრულდება) და როგორ (რა ბიზნესწესებით ხდება მართვა).

ამისათვის უნდა გამოვიყენოთ ჩვენი საგნობრივი სფეროს (ამ შემთხვევაში სკოლა) საქმიანი პროცესის უნიფიცირებული მოდელირება, რომელიც ეყრდნობა სისტემების ობიექტ-ორიენტირებულ ანალიზს და ობიექტ-ორიენტირებულ დაპროექტებას [9, 26, 40].

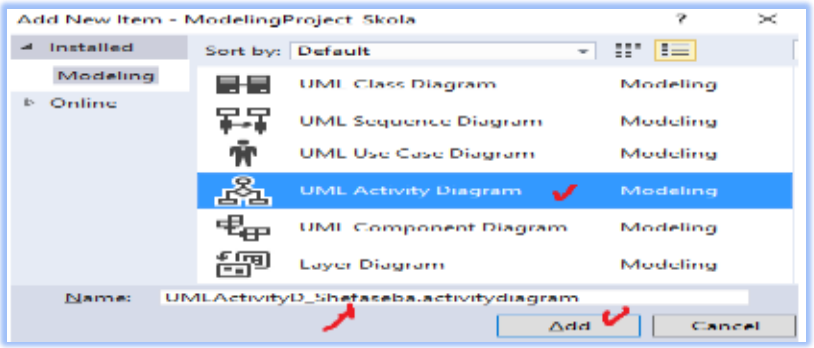
4.1 ნახაზზე მოცემულია UML-ის UseCase დიაგრამა 4 როლით და 20 ფუნქციით (შეიძლება იყოს ალტერნატიული ვარიანტიც).

- *მასწავლებელი:*
  1. დავალების საკითხების მომზადება;
  2. გამოცდის საკითხების მომზადება;
  3. დავალების საკითხების ელ-შეტანა;
  4. გამოცდის საკითხების ელ-შეტანა;
  5. დავალების საკითხების შეფასება ;
  6. გამოცდის საკითხების შეფასება;
  7. დავალების შედეგების ელ-შეტანა;
  8. გამოცდის შედეგების ელ-შეტანა.
- *მოსწავლე:*
  9. დავალების მომზადება (შესრულება);
  10. დავალების ჩაბარება;
  11. დავალების შედეგების დავალიერება;
  12. გამოცდის დაწერა;
  13. გამოცდის შედეგების დავალიერება;
  14. მომავალი გამოცდის შედეგების პროგნოზირებისთვის საჭირო მონაცემების შეგროვება;
- *მშობელი:* 11, 13. 14
- *სისტემა:*
  15. გამოცდაზე დაშვების უფლების მინიჭება;
  16. გამოცდის შედეგების ანალიზი (გადალახა თუ არა მინიმალური ზღვარი);
  17. სავარაუდო გადაწერის მცდელობის აღმოჩენა;
  18. შესაბამისი გამოცდის პროგნოზული შედეგების გენერირება;
  19. მომავალი გამოცდის შედეგების პროგნოზირებისთვის საჭირო მონაცემების შეგროვება;
  20. მომავალი გამოცდის შედეგების პროგნოზირება.



ნახ. 4.1. შეფასების სისტემის UseCase დიაგრამა (სკოლა)

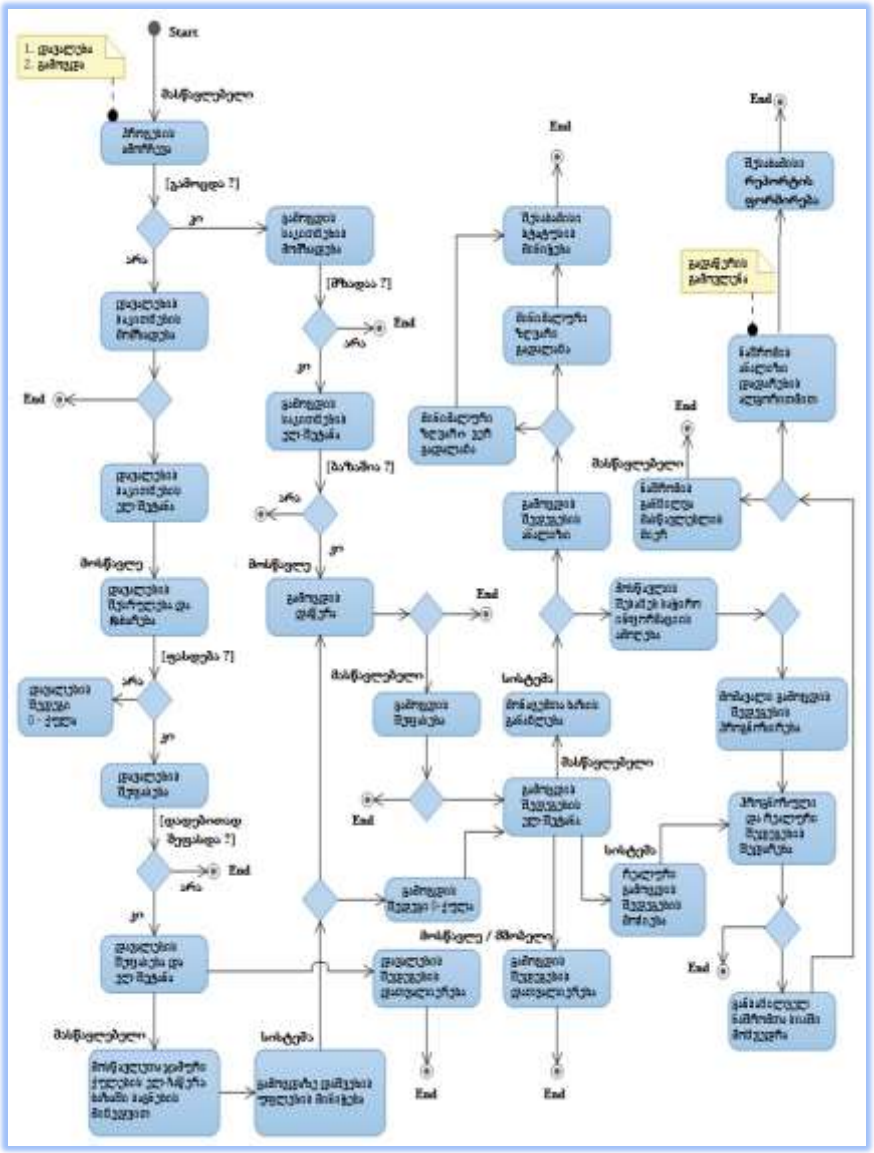
UML-ის Activity-დიაგრამა ავაგოთ Visual Studio.NET 2015-ის პლატფორმაზე, პროექტის სახელით ModelingProject_Skola, ახალი ლოკაციით. შემდეგ პროექტში ვამატებთ Add New Item-ით UMLActivityD_Shefaseba.activitydiagram (ნახ.4.2).



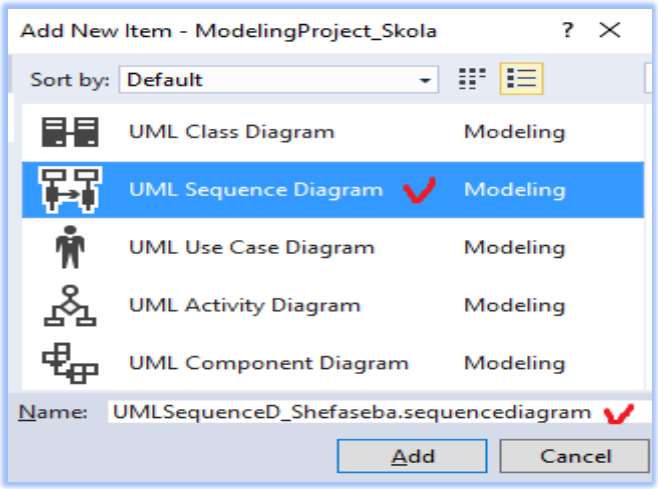
ნახ. 4.2. Activity დიაგრამის აგების რეჟიმი

➢ 4.3 ნახაზზე წარმოდგენილია UML-ის *აქტიურობათა დიაგრამა* მოსწავლეთა (სტუდენტთა) შეფასების პროცესისათვის. იგულისხმება მათ მიერ სავალდებულო (სემესტრალური) დავალებების შესრულების და ბოლოს, გამოცდის ჩაბარების პროცესების შეფასება. ნახაზზე ჩანს ამ პროცესში მონაწილე როლები და მათი ფუნქციები. რომბებში ჩადებულია პროცესის ბიზნესწესები, რომელთა საფუძველზე ხდება გადაწყვეტილების მიღება და მომდევნო პროცესის არჩევა. დიაგრამას აქვს ერთი დასაწყისი (Start) და რამდენიმე დასარული (End).

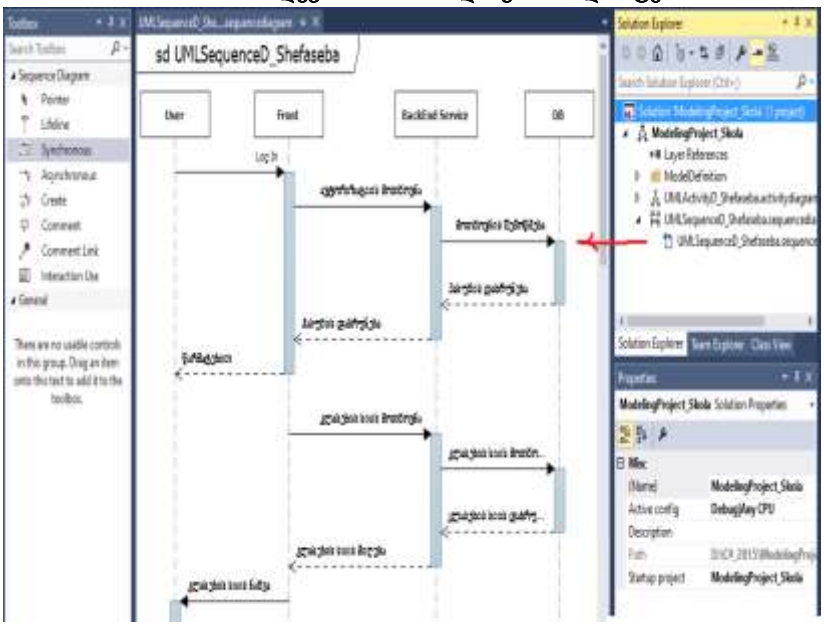
➢ პროექტს დავამატოთ განსახილველი პროცესის „სცენარი“ - *მიმდევრობითობის (Sequence) დიაგრამა,* ნიმუშით (ნახ.4.4-ა,ბ), მოსწავლეთა გამოცდის შედეგების შეფასების პროცესის მოდელი შედგება კლასთა ობიექტების და მათი მოქმედების ბილიკებისგან, სადაც თავსდება ფუნქციური ქმედებები (შეტყობინებები) დროში მიმდევრობით (ზემოდან-ქვემოთ) (ნახ.4.5).



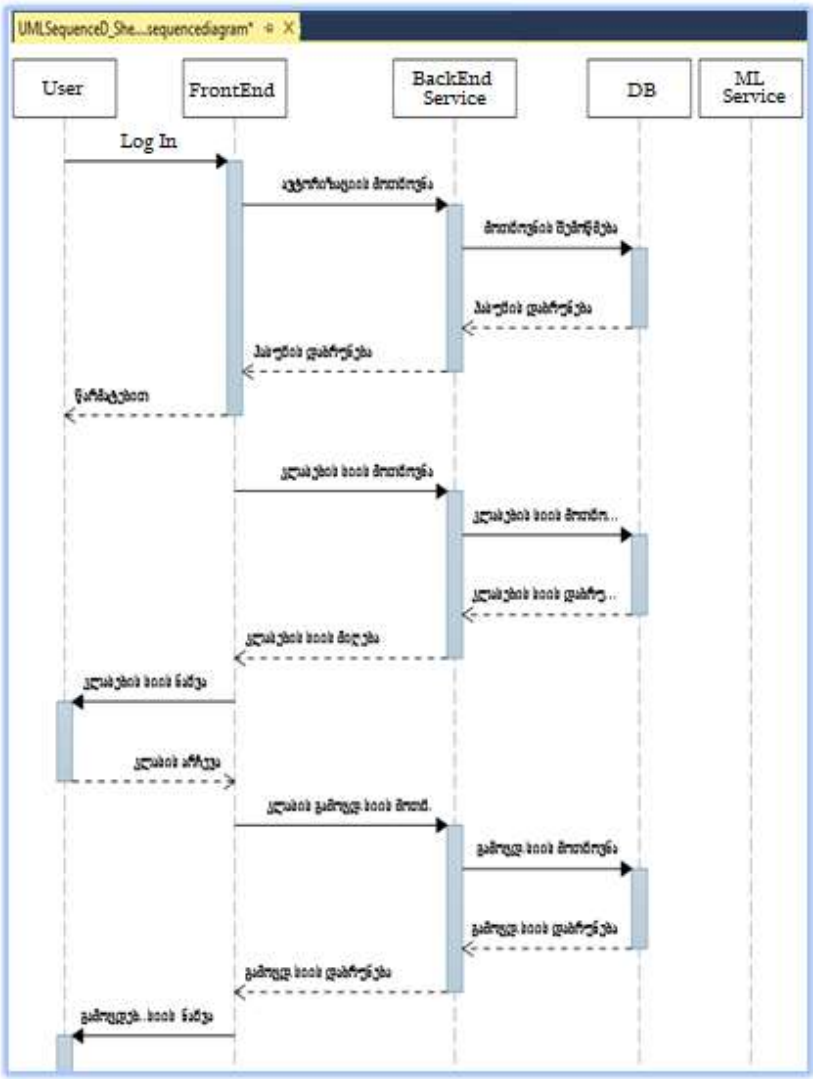
ახ. 4.3. მოსწავლის შეფასების Activity დიაგრამა



ნახ.4.4-ა. მიმდევრობითობის დიაგრამის დამატება

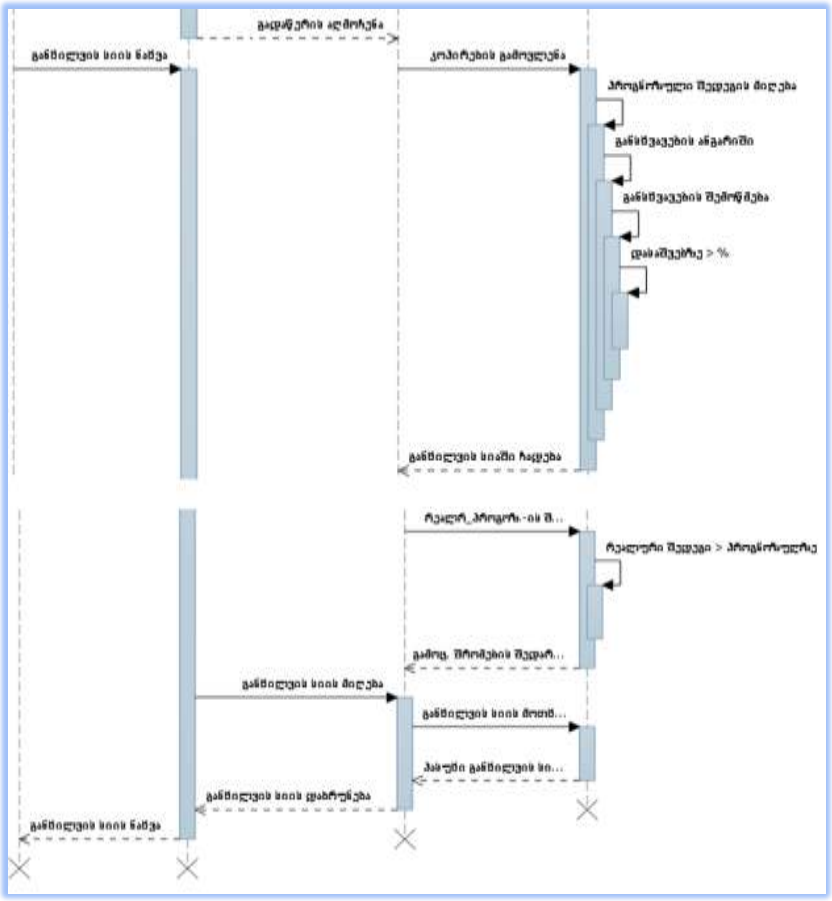


ნახ.4.4-ბ. მიმდევრობითობის დიაგრამის ნიმუში



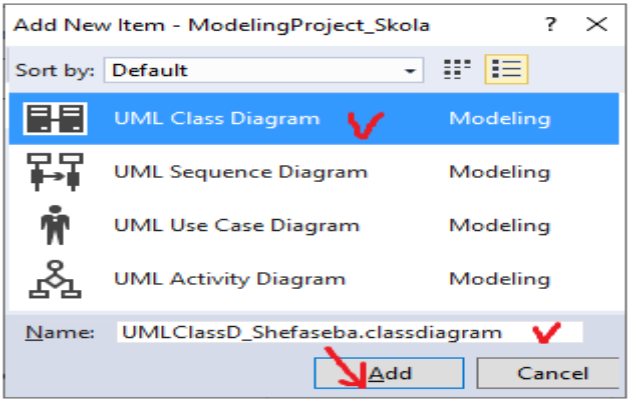
ნახ. 4.5. მოსწავლეთა შეფასების პროცესის სცენარის წარმოდგენა Sequence დიაგრამით (დასაწყისი)

ნახ. 4.5. მოსწავლეთა შეფასების პროცესის სცენარის წარმოდგენა Sequence დიაგრამით (გაგრძელება)



ნახ. 4.5. მოსწავლეთა შეფასების პროცესის სცენარის წარმოდგენა Sequence დიაგრამით (დასასრული)

➢ 4.6-ა ნახაზზე მოცემულია პროექტში *კლასთა დიაგრამის* დასამატებელი ფანჯარა. 4.7-ზე კი ჩვენი აობიექტის კლასთა-ასოციაციის დიაგრამა, რომელიც უნიფიცირებული მოდელირების ობიექტ-ორიენტირებული პროექტირების ეტაპს ეხება.



ნახ.4.6, Visual Studio.NET-გარემოში Add Item-თ ემატება ახალი Class Diagrqam

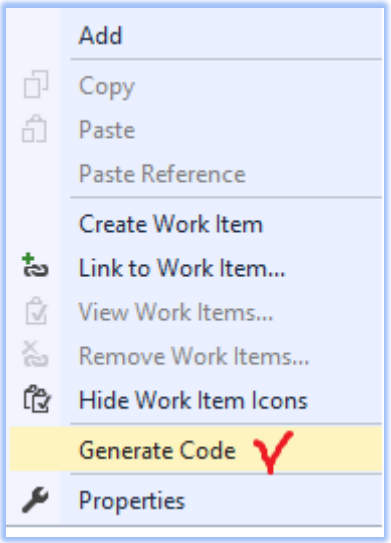
აქვე ნაჩვენებია იმ კლასთა სია, რომლებიც კომპონენტების სახით უნდა ჩაემატოს კლასთა-ასოცოაციის მოდელს. ეს უკანასკნელი არის კლასების დაკავშირებით მიღებული ერთიანი სქემა „მოსწავლეთა შეფასების“ კონკრეტული ამოცანისათვის (ცხრ.4.1). კავშირები შეიძლოება იყოს: ასოციაციური, მემკვიდრეობითი, აგრეგატული, რელაციური და სხვ.

მოსწავლის შეფასების პროცესის კლასთა სია **ცხრ.4.1**

Class Name	კლასის სახ.	Class Name	კლასის სახ.
Pupils	მოსწავლე	Classes_subj_emps	კლ-საგ-თან
Employees	თანამშრომელი	Exam	გამოცდა
Users	მომხმარებელი	ExamProperties	გამოცდის
Branches	სფერო (განყოფ.)	ExamTypes	გამოცდის თვისებები
Subjects	აკად.საგანი	Marks	ნიშანი
Groups	ჯგუფი	MarkTypes	ნიშნის ტიპები
Classes	კლასი	MarkDescriptions	ნიშნის აღწერა

ნახ.4.4.7. მოსწავლეთა საგამოცდო შეფასების ამოცანის კლასთა დიაგრამა (აგებული Visual Studio.NET პლატფორმაზე)

დაგვრჩა CASE ტექნოლოგიის ბოლო, მნიშვნელოვანი ბიჯი - C# *კოდის გენერაცია.* მაუსის მარჯვენა ღილაკით გამოიტანება კონტექსტური მენიუ და ვირჩევთ *Generate Code* (ნახ.4.8). იგი ავტომატურად წერს C# პროგრამის ტექსტს Cveni კლასთა დიაგრამისათვის. ესაა დაპროგრამების ავტომატიზაციის საფეხური, როცა სისტემა მოდელიდან აგებს პროგრამულ კოდს.



ნახ.4.8.

4.9 ნახაზზე ნაჩვენებია ჩვენი ModelingProject_Skola პროექტის Solution Explorer ფანჯარა, სადაც ჩანს სისტემის მიერ აგებული 14 კოდი C# ენაზე (კლასების დიაგრამაზე 14 კლასია).

4.1 – 4.5 ლისტინგზე ნაჩვენებია მათგან ზოგიერთი კოდი.

```
// ---- ლისტინგი 4.1.  Pupils.cs --- (მოსწავლეები)---------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
public class Pupils
    {
        public virtual object Pupil_ID { get; set; }
        public virtual object firdtName { get; set; }
        public virtual object lastName { get; set; }
        public virtual object personalID { get; set; }
        public virtual object gender { get;  set; }
        public virtual object citizenship { get; set; }
        public virtual object birthDate { get; set; }
        public virtual object address { get; set; }
        public virtual object MobNum { get; set; }
        public virtual object sMail { get; set; }
        public virtual object classID { get; set; }
        public virtual object branchID { get; set; }
        public virtual object recDate { get; set; }
        public virtual object User_ID { get; set; }
        public virtual object Branch_ID { get; set; }
        public virtual Users Users { get; set; }
        public virtual Branches Branches { get; set; }
        public virtual void insert() {
            throw new System.NotImplementedException();
        }
        public virtual void update() {
            throw new System.NotImplementedException();
        }
        public virtual void delete() {
            throw new System.NotImplementedException();
        }
}
```

```
// ------ ლისტინგი 4.2  Classes.cs -----  კლასები -------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Classes
{
    public virtual object Class_ID { get; set; }
    public virtual object name { get; set; }
    public virtual object ClassCategory_ID { get; set; }
    public virtual object User_ID { get; set; }
    public virtual object Branch_ID { get; set; }
    public virtual IEnumerable<Groups> Groups { get; set;}
    public virtual void insert() {
                    throw new System.NotImplementedException();
          }
   public virtual void update() {
                    throw new System.NotImplementedException();
          }
   public virtual void delete() {
                    throw new System.NotImplementedException();
          }
}

// ------ ლისტინგი 4.3  Subjects.cs -----  აკად. საგნები კლასები -------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
public class Subjects
{
        public virtual object Subject_ID {    get; set; }
        public virtual object name { get; set; }
        public virtual object description { get; set; }
        public virtual object recDate { get; set; }
        public virtual Groups Groups { get;   set; }

        public virtual IEnumerable<Class_Subject_Emps>
                Class_Subject_Emps { get; set; }

        public virtual void insert() {
                throw new System.NotImplementedException();
        }
        public virtual void update() {
                throw new System.NotImplementedException();
        }
        public virtual void delete() {
                throw new System.NotImplementedException();
        }
}
```

// -- ლისტინგი 4.4 Class_Subject_Emps.cs --- 3 კლასის ნაკრები ------

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

public class Class_Subject_Emps
{
        public virtual object Class_ID { get; set; }
```

```
        public virtual object Group_ID { get; set; }
        public virtual object Subject_ID { get; set; }
        public virtual object Empl_ID { get; set; }
        public virtual object Semester_ID { get; set; }
        public virtual object activationDate { get; set; }
        public virtual object deactivationDate { get; set; }
        public virtual object activeStatus      { get; set; }
        public virtual object User_ID { get; set; }
        public virtual object Branch_ID { get; set; }
        public virtual object recDate { get; set; }
        public virtual object CSE_ID { get;    set; }
        public virtual Employees Employees { get; set; }
        public virtual Classes Classes { get; set; }
        public virtual Groups Groups { get;   set; }
        public virtual void insert() {
                throw new System.NotImplementedException();
        }
        public virtual void update() {
                throw new System.NotImplementedException();
        }
        public virtual void delete() {
                throw new System.NotImplementedException();
        }
}
```

// -- ლისტინგი 4.5 Class_Subject_Emps.cs --- 3 კლასის ნაკრები ------

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

```
public class Marks
{
        public virtual object Mark_ID { get; set; }
        public virtual object CSE_ID {get; set; }
        public virtual object Pupil_ID { get; set; }
        public virtual object mark { get; set; }
        public virtual object markDate { get; set; }
        public virtual object MarkType_ID { get; set; }
        public virtual object User_ID { get; set; }
        public virtual object Branch_ID { get; set; }
        public virtual object recDate { get; set; }
        public virtual IEnumerable<Class_Subject_Emps>
Class_Subject_Emps { get; set; }
        public virtual MarkTypes MarkTypes { get; set; }
        public virtual IEnumerable<Users> Users { get; set; }
        public virtual Branches Branches { get; set; }
        public virtual void insert() {
                throw new System.NotImplementedException();
        }
        public virtual void update()            {
                throw new System.NotImplementedException();
        }
        public virtual void delete()            {
                throw new System.NotImplementedException();
        }
}
```

და ა.შ., შესაძლებელია დანარჩენის ნახვაც.

შეიძლება ითქვას, რომ კლასების ცვლადები პროგრამაში ასახულია კარგად, მაგრამ მეთოდები მოითხოვს პროგრამის დეველოპერისგან კოდის ტექსტის ხელით ჩაწერას.

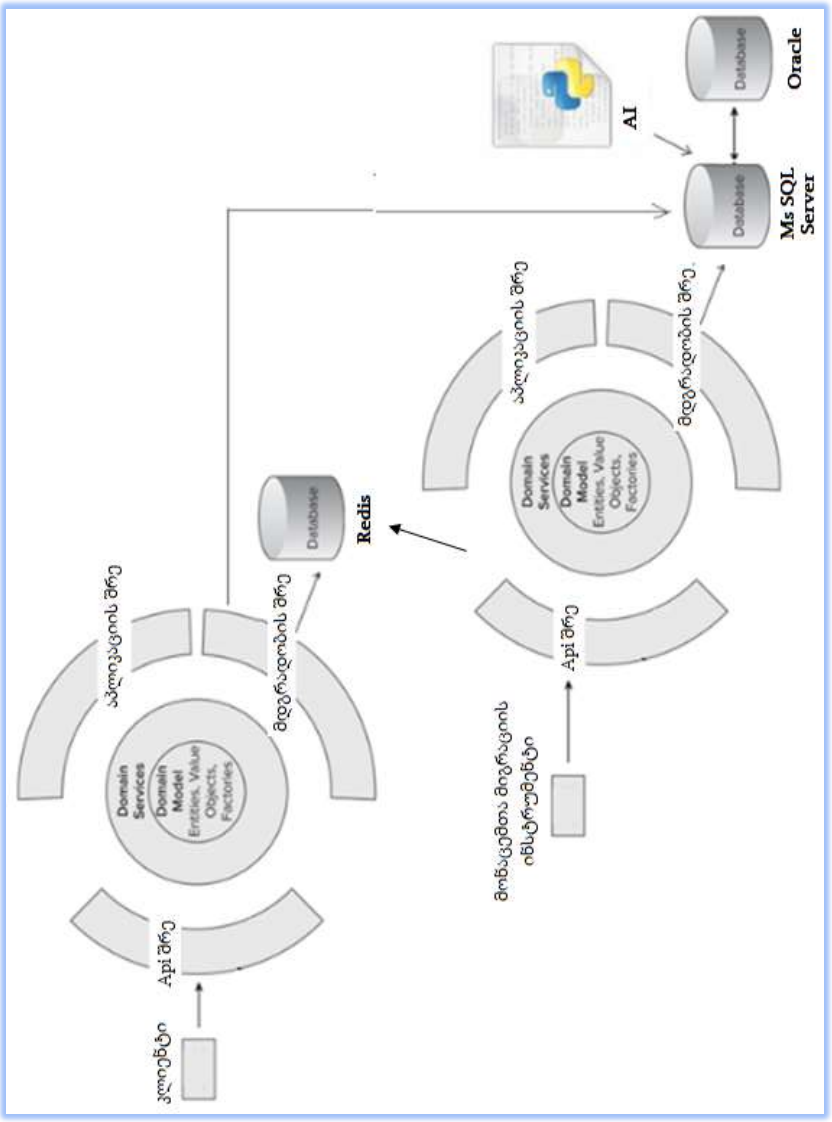
4.2. სისტემის მხარდამჭერი არქიტექტურა და მისი ტესტირება

წინა პარაგრაფში ჩატარებული საკვლევი სისტემის ობიექტ-ორიენტირებილი ანალიზის საფუძველზე (UseCase, Activity Sequence -დიაგრამები) განისაზღვრა მომავალი საპროექტო სისტემის ბიზნესმოთხოვნები (თუ რა ფუნქციები უნდა იქნას სისტემის მიერ განხორციელებული). ამისათვის კი საჭიროა დადგინდეს ასეთი სისტემის მხარდამჭერი კომპონენტების ერთობლიობა (მონაცემთა ბაზები, პროგრამები, ციფრული და აპარატურული უზრუნველყოვა და სხვ.). ჩვენ წინაშე დგება სისტემის არქიტექტურის განსაზღვრის ამოცანა. ამ მიზნით ჩვენ წინა თავში დეტალურად შევეხეთ სუფთა არქიტექტურის და DDD მეთოდოლოგიების კონცეფციებს და თეორიულ საკითხებს.

4,9 ნახაზზე წარმოდგენილია ჩვენი სისტემის სავარაუდო არქიტექტურა (DDD-ს ბაზაზე): გვაქვს front, ორი service, სამი სხვადასხვა სახის მონაცემთა ბაზა.

ბაზის მხარეს გამოყენებულია Ms SQL, Oracle, და Redis. როდესაც საიტზე გამოსატანი მონაცემების გაანგარიშება იყო საჭირო, ფრონტი სერვისის გავლით Ms SQL მონაცემთა ბაზაზე გადიოდა და უკვე ერთხელ დათვლილი ინფორმაციის ხელახლა დათვლა წარმოებდა. ეს ტვირთავდა მონაცემთა ბაზას და შედეგის დაბრუნებაც (ბევრი მომხმარებლის შემთხვევაში) გვიანდებოდა.

სწორედ ამიტომ გამოვყავით ისეთი მომართვები, რომელთა წინასწარ განსაზღვრა და დამუშავება იქნებოდა შესაძელებელი. შემდეგ დამუშავებული მონაცემები იგზავნება Ms SQL-დან Redis-ში სერვისის გავლით. სერვისი, რომელიც დგას ამ ორ ბაზას შორის, მუდმივად ცდილობს დატის წაღებას, ხოლო გასატან პორციებს ვაკონტროლებთ Ms SQL-ის მხარეს.



ნახ.4.9. სისტემის არქიტექტურის მოდელი

პროცესის უკეთ ასახვის მიზნით განვიხილოთ ერთი შემთხვევა. მაგალითად, მომხმარებელს სურს დაათვალიეროს გამოცდის შედეგები. ვინაიდან გამოცდის შედეგები ფრონტენდის მხარეს ვიზუალურად რამდენიმე გვერდზე ნაწილდება და ყველაზე მოთხოვნადი არის პირველი გვერდი (სადაც გამოტანილია ბოლო 10 გამოცდის შედეგი), Ms SQL-ის მხარეს წინასწარ ხდება შესაბამისი მონაცემების დამუშავება და გატანა Redis-ში.

ხოლო ყველა დანარჩენი გვერდის მონაცემებს ვიღებთ სერვისის გავლით Ms SQL-ზე მიმართვით. ყოველი ახალი გამოცდის შედეგის Ms SQL-ში შეტანის დროს ხდება შესაბამისი მოსწავლის/სტუდენტის იდენტიფიკატორის „Redis-ში გასატან რიგში“ ჩასმა (რიგში მომსახურებაა FIFO). მის მიხედვითა სისტემა ხვდება, რომელი მოსწავლისთვის/სტუდენტისთვის შეიცვალა მონაცემები და გენერირდება ახალი დატა Redis-ში გასატანად.

სწორედ ამიტომ Redis-ში მუდმივად განახლებული (live) მონაცემებია ჩაწერილი. შესაბამისი მიდგომა უზრუნველყოფს core ბაზაზე ნაკლებ დატვირთვას და შედეგების ბაზიდან მიღების დროის ოპტიმიზაციას, რაც განპირობებულია წინასწარ დამუშავებული (დათვლილი) მონაცემების Redis-იდან წაკითხვით.

იგი არის in-memory მონაცემთა ბაზა და გამოიყენება სწრაფი ჩაწერა-წაკითხვისთვის. შესაბამისად უმეტესობა მომართვება გადის პირდაპირ რედისზე. რედისში მონაცემების შენახვა ხდება key-value სტრუქტურით. შესაბამისი key ყოველი call-სთვის გენერირდება სერვისის მხარეს, რომელიც დგას Ms SQL-სა და Redis-ს შორის. ფრონტი კი, კონკრეტული მოთხოვნის გაკეთებისას, მიმართავს სერვისს, რომელმაც იცის კონკრეტული call-ის შემთხვევაში არსებული key. იგი მას ავსებს ფრონტიდან გადმოცემული პარამეტრებით და შემდეგ მიღებული key-ის გამოყენებით იღებს Redis-იდან value-ს, რომელიც ჩაწერილია შესაბამის ბაზაში Json ფორმატში.

Ms SQL-დან ასევე გარკვეული მონაცემები გადის Oracle-ში, რადგან იგი მეტად მძლავრი მონაცემთა ბაზაა ტექსტებთან სამუშაოდ. მას ამისთვის აქვს მრავალი ჩაშენებული მეთოდი. ასევე Ms SQL ბაზიდან ვიღებთ მონაცემებს მანქანური დასწავლის ალგორითმების სამუშაოდ Python-ის გამოყენებით.

- ***სისტემის სტრეს-ტესტის შედეგები:***

არსებობს პროგრამების შესრულების (performance - წარმადობა, მწარმოებლურობა, გამტარუნარიანობა) ტესტირების რამდენიმე სახე. მაგალითად, *დატვირთვის ტესტირება* (Load testing) და *სტრეს ტესტირება* (Stress testing). ისინი ამოწმებს, თუ როგორ მუშაობს პროგრამული აპლიკაცია, როდესაც მას ერთდროულად იყენებენ დიდი რაოდენობის მომხმარებლები [96].

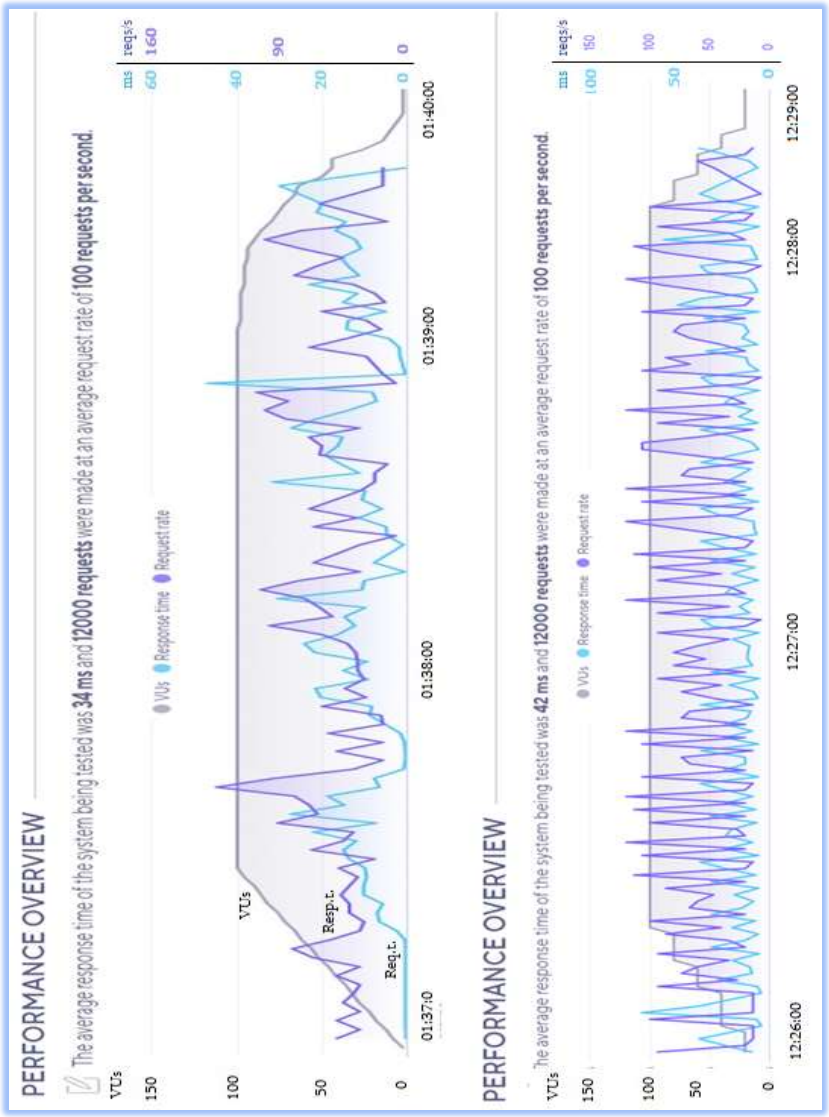
დატვირთვის ტესტირება ახდენს აპლიკაციის რეალური დატვირთვის სიმულაციას, სტრეს-ტესტი კი არის წარმადობის (შესრულების) ტესტის ტიპი, რომელიც ამოწმებს პროგრამული სისტემის ზედა ზღვარს ექსტრემალური დატვირთვის დროს.

სტრეს-ტესტები ამოწმებს, თუ როგორ იქცევა სისტემა ინტენსიური დატვირთვის დროს და როგორ აღდგება იგი ნორმალურ რეჟიმზე დაბრუნებისას; არის თუ არა ეფექტიანობის ძირითადი ინდიკატორები, გამტარუნარიანობა და რეაგირების დრო, იგივე, რაც დატვირთვის მატებამდე და სხვ.

პროექტის ფარგლებში არქიტექტურის გასატესტად ჩავატარეთ სტრეს-ტესტირება. გავუშვით ვირტუალური იუზერები წუთში 6000 მოთხოვნა (Request), ანუ წამში 100 მოთხოვნა.

4.10 ნახაზზე გამოტანილია 2 წუთის მუშაობის შედეგები ორი ვარიანტისათვის - 34 და 42 მილიწამების შემთხვევაში.

ამგვარად, 12000 მოთხოვნა გავიდა და უკან დააბრუნა პასუხი. 34 მწმ (შემდეგ 42 მწმ) დაითვალა თითო მიმართვის მუშაობის საშუალო დრო 12000 Request-ისთვის. (3 წუთი ჩანს დიაგრამაზე, მაგრამ ერთი წუთი დაჭირდა ჯერ იუზერების გენერირებას და შემდეგ ტესტის გაშვებას).



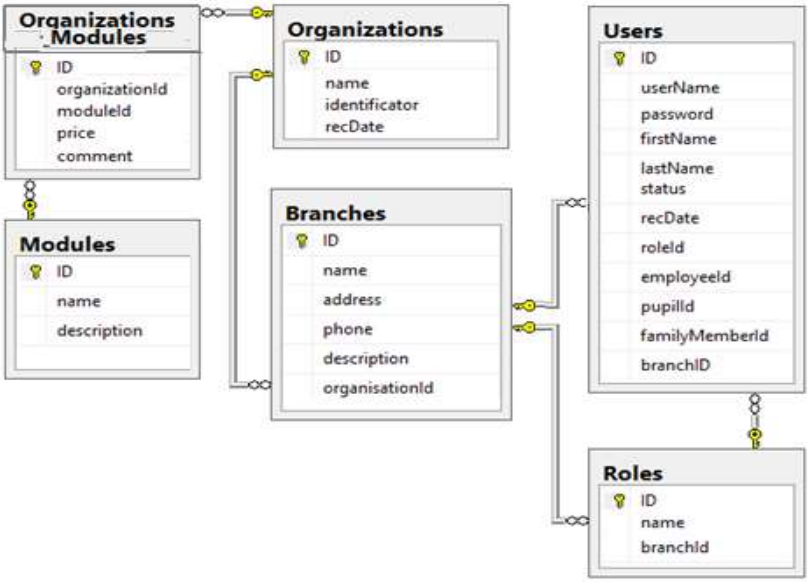
ნახ. 4.10. სისტემის ტესტირების შედეგები
34 მწმ. და 42 მ.წმ შემთხვევაში

4.3. მონაცემთა ბაზის სტრუქტურა და ოპტიმიზაცია

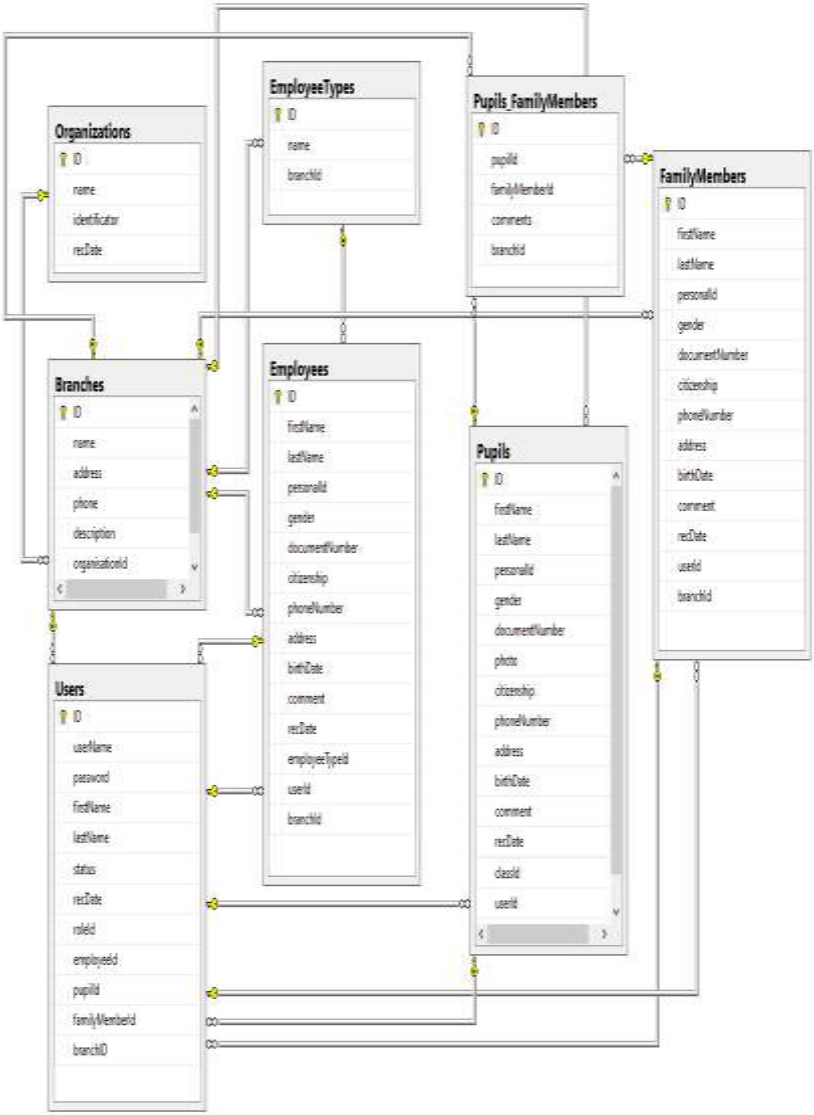
ჩვენი პროექტის მიზანია შევქმნათ მოქნილი, სწრაფი და თვითდასწავლადი სისტემა, რომელიც უმოკლეს დროში მიაწვდის მის მომხმარებელს საჭირო ინფორმაციას. შესაბამისი სისტემა უნდა იყოს მოქნილი და დაცული.

➢ *რელაციური მბ-ის სტრუქტურა* (Ms SQL Server-ის მაგალითზე). სისტემის მომხმარებელს, რომელსაც სურს შეინახოს დიდი მოცულობის მონაცემები, მისთვის ერთ-ერთი საუკეთესო გადაწყვეტაა მათი შენახვა Ms SQL Server-ის მონაცემთა ბაზაში, რომელსაც დიდი ზომის მეხსიერების გარდა აქვს მონაცემთა დაცვის მექანიზმებიც.

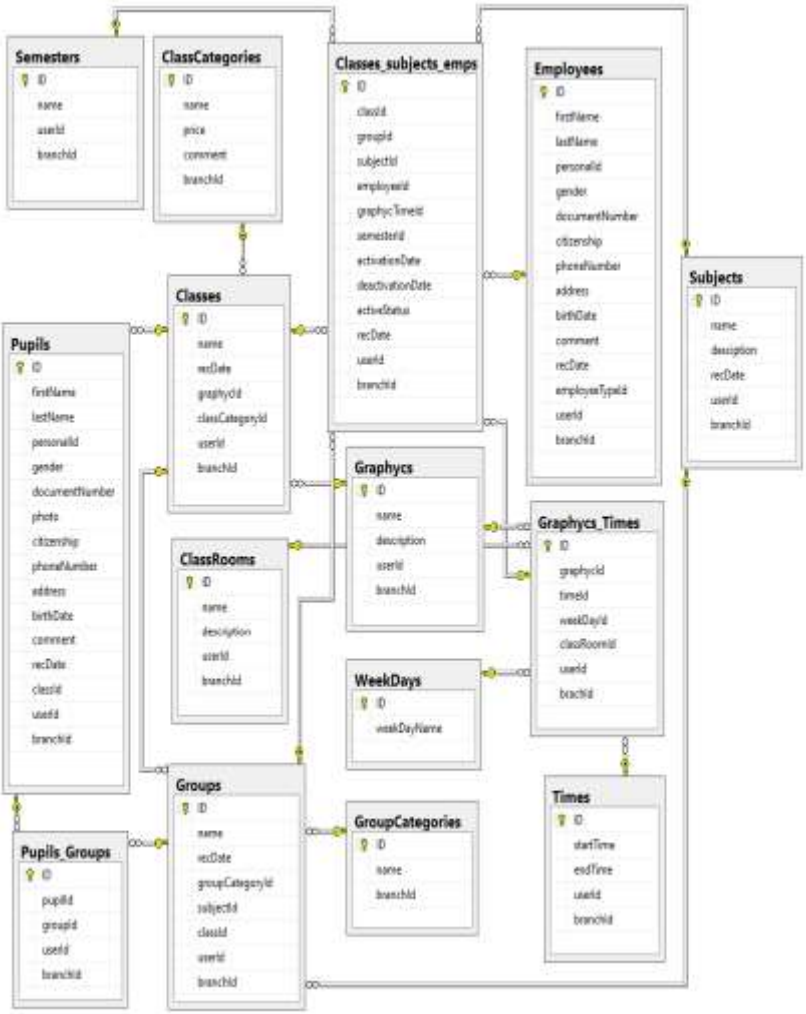
მონაცემთა ბაზის ძირითადი ცხრილების სქემა და ცხრილთა-შორისი კავშირები (ER-მოდელები) ასახულია 4.11-4.15 ნახაზებზე.



ნახ. 4.11. ორგანიზაციის სტრუქტურის ER-მოდელი

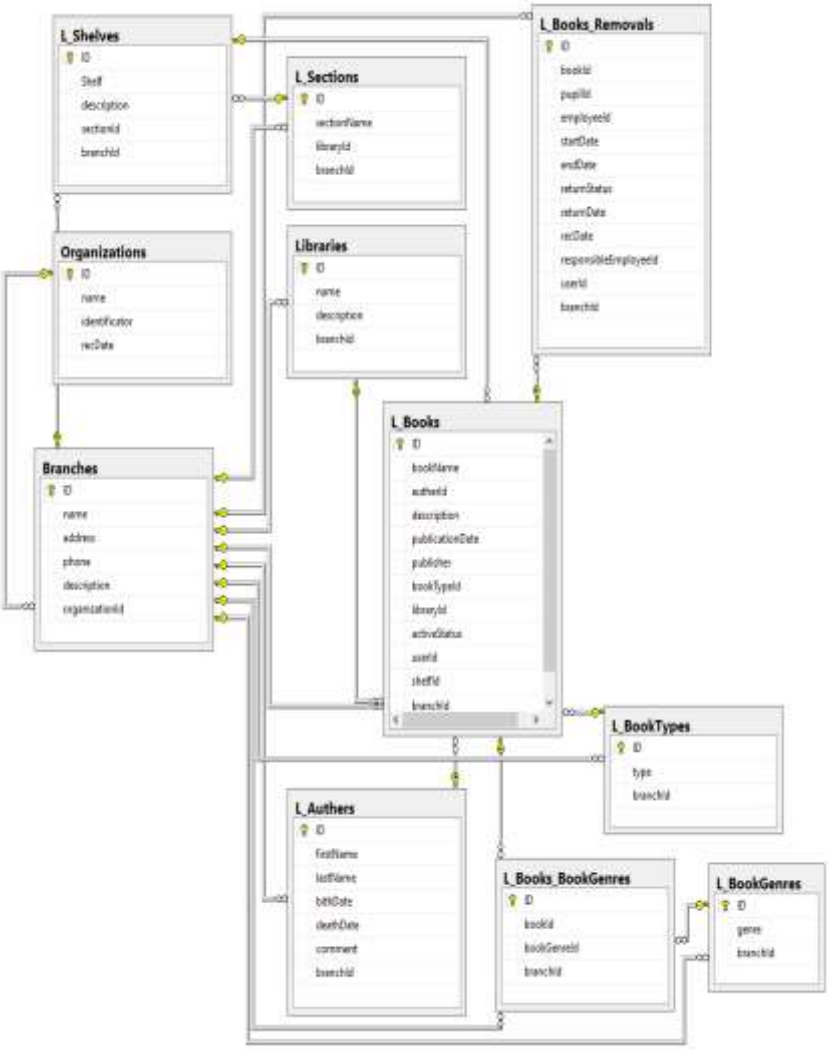


ნახ. 4.12. ორგანიზაციის წევრების ER-მოდელი



ნახ.4.13. სასწავლო პროცესის ER-მოდელი

ნახ. 4.14. მოსწავლეთა შეფასების ER-მოდელი



ნახ. 4.15. ორგანიზაციის ფილიალის ბიბლიოთეკის ER-მოდელი

და ა.შ. შესაძლებელია ბაზის ER-მოდელების გაფართოება და კორექტირება საგანმანათლებლო ორგანიზაციის შესაბამისად.

➢ მონაცემთა ბაზების რელაციური მოდელის კონცეფცია და მისი ოპტიმიზაციის ამოცანა ნორმალურ ფორმათა თეორიის საფუძველზე, რომელიც ეყრდნობოდა სიმრავლეთა თეორიისა და მათემატიკური ლოგიკის საფუძვლებს (ტოპოლოგიას) - დასვა და განავითარა პროფესორმა ედგარ კოდმა 1970-1972 წლებში [97, 98]. მისი ეს თეორიული იდეა მოედო მთელი მსოფლიოს წამყვან უნივერსიტეტებს ამერიკის, გერმანიის, საფრანგეთის და სხვ. ქვეყნებიდან [99-102]. კარძოდ, ფრანგი კ. დელობელის (გრენობლი), გერმანელი ჰ. ვედეკინდის და ტ. ჰერდერის (დარმშტადტი) და ა.შ. ამერიკელების - ჯ. ულმანის, კ. ვონგის, ფ. ბერნშტაინის და სხვ. მიერ დამუშავდა მონაცემთა ბაზების ფუნქციონალურ დამოკიდებულებათა სინთეზის მეთოდი, რომელთა საფუძველს ბულის ალგებრისა და დამოკიდებულებათა თეორიების იზომორფიზმი შეადგენდა. მსგავსი ამოცანები გამოკვლეულ იქნა აგრეთვე ვ. ბრუდნოს, ლ. კალინიჩენკოს, მ. ცალენკოს, ჩოგოვაძის, გ. სურგულაძის და ვ. ქაჩიბაიას (სტუ) ნაშრომებში [103-106].

სტუ-ის (პოლიტექნიკური ინსტიტუტის) „მართვის ავტომატიზებული სისტემების“ ასპირანტმა გ. სურგულაძემ (ამ წიგნის თანაავტორი) 1976 წ. იანვარში (გერმანიაში, მაგდებურგის ტექნიკურ უნივერსიტეტში 1 წლიანი სამეცნიერო სტაჟირების გავლის შემდეგ) წარმოადგინა დისერტაცია „რელაციური ბაზის სტრუქტურების აგების ტექნოლოგია“, რომლის სიახლეც იყო რელაციური ბაზების სტრუქტურების ოპტიმიზაციის პროცესის პროგრამირების ავტომატიზაცია [103].

ახალი ტიპის, NoSQL ბაზების გამოჩენამ, დიდ მონაცემთა დამუშავებისა და ღრუბლოვანი ტექნოლოგიების გამოყენების კონცეფციამ ახალი გამოწვევები გააჩინა. თითქოს კონკურენტი გაუჩნდა 40 წლის „გამეფებულ“ რელაციურ ბაზებს [12, 54].

როგორც ცნობილი ფირმებისა და გამოცდილი მეცნიერ-პრაქტიკოსების კვლევებმა აჩვენა, რომ SQL (რელაციურ) და NoSQL

(იერარქიულ) ტიპის ბაზებს თავიანთი დადებითი და უარყოფითი მხარეები აქვს. საუკეთესო ვარიანტი კი მდგომარეობს მათ ერთობლივ გამოყენებაში. ორგანიზაციებს შეუძლია იქონიოს ორივე ტიპის ბაზა და გამომდინარე კონკრეტული ამოცანიდან, გამოიყენონ ან ერთი ან მეორე.

ასეთ ვითარებაში კვლავ ამოტივტივდა მონაცემთა ბაზის სტრუქტურების ოპტიმიზაციის ამოცანა. გადასაწყვეტი პრობლემის ოპტიმიზაციის თვალსაზრისით, მონაცემთა ძებნის დროისა (Tძ) და მონაცემთა მოცულობის (Vმ) კომპრომისული მნიშვნელობის განსაზღვრის მიზნით, იყენებენ ინდექსირებული ფაილების შექმნის ხერხს [107]. ახლა განვიხილოთ ინდექსაციის საკითხი, ხოლო შემდეგ პარაგრაფში კი ნორმალიზაცია-დენორმალიზაციის ამოცანა SQL/NoSQL ბაზებთა მიმართებაში [108, 109].

4.3.1. ინდექსაცია და ცხრილების დაყოფა

მონაცემთა ბაზის წარმადობის გაზრდისთვის, მონაცემების დამუშავების საგძნობლად დაჩქარებისათვის და მათთან უფრო მარტივად სამუშაოდ ვიყენებთ სხვადასხვა შესაძლებლობას, მაგალითად ინდექსაციას, ცხრილების დაყოფას და ა.შ..

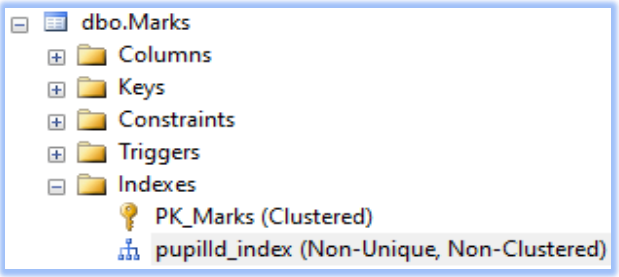
ინდექსი (index) არის ცხრილთან ან წარმოდგენასთან (View) დაკავშირებული სტრუქტურა და განკუთვნილია შესაბამის ცხრილში ან წარმოდგენაში ინფორმაციის ძებნის დასაჩქარებლად.

ინდექსი განისაზღვრება ერთი ან მეტი ველისათვის, რომლებსაც *ინდექსირებული სვეტები* ეწოდება. ინდექსი შეიცავს ინდექსირებული ველის ან ველების მოწესრიგებულ მნიშვნელობებს საწყისი ცხრილის ან წარმოდგენის შესაბამის სტრიქონზე მიმართვებთან ერთად.

წარმადობის ზრდა მიიღწევა სვეტის მონაცემების მოწესრიგების ხარჯზე. თუ სვეტი მოუწესრიგებელია, მაშინ საჭირო მნიშვნელობის საპოვნელად მიმდევრობით უნდა გადაისინჯოს

ყველა მნიშვნელობა. როცა საჭიროა ინდექსირებულ სვეტში მნიშვნელობის პოვნა, მაშინ მისი ძებნა სრულდება ინდექსში. ინდექსების გამოყენება მნიშვნელოვნად ზრდის მონაცემების ძებნის მწარმოებლურობას, მაგრამ ითხოვს დამატებით სივრცეს მონაცემთა ბაზაში [11, 13].

მაგალითად, ინდექსი გვიდევს Marks ცხრილის Pupil_ID ველზე, რაც დაგვეხმარა შესაბამის ცხრილში ძებნის პროცესების დაჩქარებაში და შესაბამისად წარმადობის გაზრდაში.



ნახ. 4.16. ინდექსაციის მაგალითი

დიდ და ისტორიული ტიპის ცხრილებთან სამუშაოდ ვიყენებთ Partitions, რომლის დახმარებითაც მოთხოვნები, რომლებიც მიმართავს მხოლოდ მონაცემების ნაწილს სრულდება უფრო სწრაფად, რადგან სკანირებისათვის საჭიროა ნაკლები მონაცემები (განთავსებული მხოლოდ მიმდინარე partion-ებში),

ანუ კონკრეტული ბრაძანების შესრულების დროს არ ხდება არასაჭირო / ან ძველი ისტორიული მონაცემების განხილვა, მათ შორის ეს გზა გვაძლევს საშუალებას ავამუშავოთ ძველი მონაცემების არქივაციის პოლიტიკაც - Data Retention. მაგალითად, Marks ცხრილი დაყოფილია RecDate-ის მიხედვით, რომ კონკრეტული მიმართვის დროს მოხდეს მხოლოდ კონკრეტული თარიღის მიხედვით მონაცემების ამოღება.

4.3.2. ნორმალიზაცია / დენორმალიზაცია

როგორც ცნობილია, რელაციური ტიპის SQL ბაზებში ნორმალიზაციის პროცესი (რელაციის დაყოფა მე-2, მე-3 და უფრო მაღალი რიგის ცხრილებად) ითვლება მონაცემთა ოპტიმიზციის მეთოდად [98, 101,108]. გადასაწყვეტი პრობლემის თვალსაზრისით, ოპტიმიზაცია გულისხმობს მონაცემთა ძებნის დროისა ($T_ა$) და მონაცემთა მოცულობის ($V_ა$) კომპრომისული მნიშვნელობის განსაზღვრას. ამ მიზნით იყენებენ *ინდექსირებული* ფაილების შექმნის ხერხს, რომელსაც ქვემოთ განვიხილავთ.

მეორეს მხრივ, არარელაციური NoSQL ბაზები მიეკუთვნება იერარქიული სტრუქტურის მქონე ბაზებს. ასეთი ბაზებისთვის უფრო ეფექტურია დენორმალიზაცია, ანუ რაც უფრო ნაკლებადაა დაყოფილი სტრუქტურა, მით ეფექტურია მოთხოვნის დამუშავება დროის მინიმიზაციის თვალსაზრისით [109].

განვიხილოთ, რაში მდგომარეობს და როგოს ხდება ნორმალიზაცია - დენორმალიზაციის პროცესი, როგორ ხდება პროცესის შედეგების შეფასება.

ზოგადად, ასეთი კლასის ოპტიმიზაციის ამოცანების გადაწყვეტა დამოკიდებულია სისტემის მომხმარებელთა მოთხოვნების ანალიზის შედეგებზე. კერძოდ, სამი (ან მეტი) დამოუკიდებელი ცხრილი, მიმთითებლიანი კავშირებით (reference) ერთმანეთთან, იკავებს მონაცემთა მოცულობის ($V_{ამონაც.}$) თვალსაზრისით მინიმალურ ადგილს მეხსიერებაში (ნაკლებია მონაცემთა დუბლირება, სიჭარბე). მაგრამ მოთხოვნისათვის მონაცემთა ძებნის დრო არაა მინიმალური (ცხრილთაშორისი კავშირების/გადართვების/ მეტი რაოდენობის გამო). მოდელი შეიძლება ასე ავსახოთ:

ანფ -> 1ნფ -> 2ნფ -> 3ნფ -> 4ნფ -> 5ნფ -> . ? . -> ბნფ

სადაც **ანფ** არანორმალიზებული ფორმაა, **1ნფ** - პირველი ნფ, ... , **ბნფ** - ბინარული ნფ.

ნიშანი " **?** " მიუთითებს იმაზე, რომ მონაცემთა დამოკიდებულებათა თვისებების კვლევა შეიძლება გაგრძელდეს მათი შემდგომი ოპტიმიზაციის მიზნით. დღეისათვის მრავლად არსებობს სხვა სახის ნფ-ებიც, მაგრამ ნორმალიზაციის კლასიკურ თეორიაში ისინი ნაკლებადაა ასახული [62].

მონაცემთა ცხრილების დეკომპოზიციის დროს ბნფ-ები მიიღება სქემის დაპროექტების ცალკეულ ეტაპზე.

არადეკომპონირებადი ცხრილისთვის კი საჭიროა ხელოვნურად *ფიქტიური ატრიბუტის* (ნატურალურ რიცხვთა სასრული სიმრავლე) შემოტანა. ინფორმაციის სემანტიკური მთლიანობის უზრუნველყოფა დეკომპონირებულ დამოკიდებულებათა შორის ხორციელდება ინდექსური კავშირების დუბლირების საშუალებით, ე.ი. შემოტანილია გარკვეული სიჭარბე.

რაც მაღალია ნფ-ის რიგი, მით ნაკლებია ინფორმაციული და მეტია ინდექსური სიჭარბე (!):

1ნფ -> 2ნფ ->. . . -> ბნფ, 1ნფ -> 2ნფ-> . . . ->ბნფ,
V_1-ინფ >= V_2-ინფ >=. . . > = $V_ბ$–ინფ, V_1-ინდ <= V_2-ინდ <=. . .<= $V_ბ$-ინდ

მონაცემთა სტრუქტურის ოპტიმალური სიჭარბის განსაზღვრის ამოცანა განახლების დროის მინიმიზაციის მოთხოვნით - არის გადასაწყვეტი (მიზანი).

მისი ამოხსნის შედეგად შესაძლებელია მონაცემთა ბაზის სქემის ცხრილების ოპტიმალური ნფ-ების დადგენა, ანუ სად გადის „ნორმალიზაცია / დენორმალიზაციის“ ზღვარი.

საჭიროა შესაბამისი ალგორითმის აგება და პროგრამული რეალიზაცია, რის საფუძველზეც ავტომატიზებულად დაპროექტდება საპრობლემო სფეროს კონცეპტუალური სქემა (ER-მოდელი).

საწყის ცხრილთა სტრუქტურა შეიძლება ასე ჩავწეროთ:

$$\begin{cases} R_1(k_1, k_2, \ldots k_{n1}, A_1, A_2, \ldots, A_{a1}) \\ R_2(k_1, k_2, \ldots k_{n2}, B_1, B_2, \ldots, B_{a2}) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ R_l(k_1, k_2, \ldots k_{nl}, Z_1, Z_2, \ldots, Z_{al}) \end{cases} \quad (4.1)$$

სადაც *k* ატრიბუტების გასაღებური, ხოლო *A -:-Z* არაგასაღებური ნაწილებია.

სემანტიკური მთლიანობა არის იმ ფაქტის შედეგი, რომ სიმრავლე მიღებულია ერთი უნივერსუმის დეკომპოზიციით. თუ ჩავთვლით, რომ

$$k_1, k_2, \ldots k_{n1} \supseteq k_1, k_2, \ldots k_{n2} \supseteq k_1, k_2, \ldots k_{nl}$$

(4.1) სისტემის კომპოზიციით (*დენორმალიზაციით*), მაშინ შესაძლებელია ერთი შედარებით დაბალი ნფ-ის მიღება (რაც NoSQL-ის დროსაც გვჭირდება):

$$R(k_1 \ldots k_{n1}, A_1 \ldots A_{a1}, B_1 \ldots B_{a2}, \ldots Z_1 \ldots Z_{al}) \quad (4.2)$$

დავუშვათ აგრეთვე, რომ წინასწარ ცნობილია R_i-ის ცვლილების რაოდენობა μ_i დროის განსაზღვრულ ინტერვალში და მართებულია შემდეგი მოწესრიგება:

$$\mu_1 \geq \mu_2 \geq \ldots \geq \mu_l$$

(1) და (2) - სთვის მონაცემთა განახლების მოცულობები შესაბამისად გამოითვლება ასე:

$$Q_{dec} = \sum_{i=1}^{l} \mu_i * (n_i + a_i) \text{ და } Q_{com} = \mu_1 * (n_1 + \sum_{j=1}^{l} (a_j - r))$$

სადაც *r* – ატრიბუტების ის რაოდენობაა, რომლითაც სრულდება შეერთების (“Join”) ოპერაცია. შემდგომში შეიძლება იგნორირება.

თუ დავუშვებთ, რომ (1) და (2) ნფ-ებს შორის არსებობს შუალედური ნფ, მაშინ მისთვის განახლებათა მოცულობა შეადგენს:

$$Q=\sum_{j=1}^{s}\mu_j*(n_j+\sum_{k=1}^{l}a_k)$$

სადაც S ფდ-ების რაოდენობაა შუალედურ ნფ-ში. მაშინ, მართებულია გამოსახულება:

$$\mu_1*(n_1+\sum_{k=1}^{l}a_k)\geq\ldots\geq\sum_{j=1}^{s}\mu_j*(n_j+\sum_{k=1}^{l}a_k)\geq\ldots\geq\sum_{i=1}^{s}\mu_i*(n_i+a_i) \quad (4.3)$$

სადაც უტოლობის მარცხენა მხარე ეთანადება *(i-1)*ნფ-ს, ხოლო მარჯვენა მხარე - *(i+1)*ნფ-ს, ცენტრალური კი – *i* ნფ-ს.

გავაანალიზოთ დეტალურად ორი მოსაზღვრე ნფ, მაგალითად, i და i+1. (4.3)-დან შეიძლება მივიღოთ:

$$\sum_{j=1}^{s}\mu_j n_j+\sum_{j=1}^{s}\sum_{k=1}^{l}\mu_j a_k\geq\sum_{i=1}^{l}\mu_i n_i+\sum_{i=1}^{l}\mu_i a_i \quad (4.4)$$

ამის საფუძველზე მართებულია შემდეგი გამოსახულებები:

$$\sum_{i=1}^{l}\mu_i n_i-\sum_{j=1}^{s}\mu_j n_j=\sum_{i=s+1}^{l}\mu_i n_i \quad (4.5)$$

$$\sum_{j=1}^{s}\sum_{k=1}^{l}\mu_j a_k-\sum_{i=1}^{l}\mu_i a_i=\sum_{j=1}^{s}\sum_{k=1,j\neq k}^{l}\mu_j a_k-\sum_{i=s+1}^{l}\mu_i a_i \quad (4.6)$$

თუ (4.5) და (4.6) გამოსახულებათა მარჯვენა ნაწილებს ჩავსვამთ (4.4)-ში, მივიღებთ:

$$\sum_{j=1}^{s}\sum_{k=1,j\neq k}^{l}\mu_j a_k\geq\sum_{i=s+1}^{l}\mu_i n_i+\sum_{i=s+1}^{l}\mu_i a_i \quad (4.7)$$

უტოლობის ორივე მხარე გავყოთ $\sum_{i=s+1}^{l}\mu_i a_i$ -ზე:

$$\frac{\sum_{j=1,k=1}^{s}\sum_{j\neq k}^{l}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i} \geq \frac{\sum_{i=s+1}^{l}\mu_i n_i}{\sum_{i=s+1}^{l}\mu_i a_i} + \frac{\sum_{i=s+1}^{l}\mu_i a_i}{\sum_{i=s+1}^{l}\mu_i a_i} \qquad (4.8)$$

ვინაიდან $[1:l]=[1:s]\cup[s+1:l]$ ამიტომ:

$$\frac{\sum_{j=1}^{s}\sum_{k=1,j\neq k}^{l}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i} = \frac{\sum_{j=1}^{s}\sum_{k=1,j\neq k}^{s}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i} + \frac{\sum_{j=1}^{s}\mu_j \sum_{k=s+1}^{l} a_k}{\sum_{i=s+1}^{l}\mu_i \sum_{i=s+1}^{l} a_i}$$

საბოლოოდ მივიღებთ:

$$\frac{\sum_{j=1}^{s}\sum_{k=1,j\neq k}^{l}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i} + \frac{\sum_{j=1}^{s}\mu_j}{\sum_{i=s+1}^{l}\mu_i} \geq \sum_{i=s+1}^{l}\frac{n_i}{a_i} + 1 \qquad (4.9)$$

სადაც $l \geq 2, s \geq 1$ და $l > s$.

პრაქტიკაში ხშირად გამოიყენება შემთხვევა, როცა *l*=2 და *s*=1, მაშინ (4.9)-ს ექნება ასეთი სახე:

$$\frac{\mu_1}{\mu_2} \geq \frac{n_2}{a_2} + 1$$

ესაა ვონგ-ვედეკინდის მოდელი (1,2 და 3 ნფ განხილვისას) [101]. იგი არის (4.9) გამოსახულების (ვედეკინდ-სურგულაძის მოდელის) კერძო შემთხვევა, როცა გვაქვს *n*ნფ და n>=3 [108].

მაშასადამე, თუ ცხრილების (კოლექციების) გასაღებური ატრიბუტების ცვლილებების სიხშირე მაღალია, მაშინ სასურველია დაბალი რიგის ნფ-ების გამოყენება, ხოლო თუ არაგასაღებურ ატრიბუტთა ცვლილების სიხშირეა განმსაზღვრელი, მაშინ - შედარებით მაღალი რიგის ნფ-ების გამოყენებაა მიზანშეწონილი.

4.3.3. შენახვადი პროცედურები და ფუნქციები

მონაცემთა ბაზასთან დასაკავშირებლად გამოიყენება დრაივერი ADO.NET. ბაზასთან მიმართვისას მას არ ეგზავნება ბრძანებები შესასრულებლად, არამედ გამოიძახება Stored procedure-ები და function-ები, რათა უფრო სწრაფი და საიმედო გახდეს მოთხოვნილი დავალებების შესრულება.

მაგალითად, დაგვიანებების შესახებ ინფორმაციის მისაღებად არსებული შენახვადი პროცედურის გამოყენების მაგალითი მოცემულია შემდეგ ლისტინგში:

```
ALTER proc [dbo].[GetLate]
@ListPupils as PupilsIdList readonly,
@ListDates as DatesList readonly,
@branchId int
as
Begin
        begin--ცვლადები და ცხრილები
         declare @date as date
         declare @CIdDay as int
         declare @MIdDay as int
         declare @pupilId as int
         declare @CIdpupil as int
         declare @MIdpupil as int

        create table #TempReturnData
        (
        [ID] [int] IDENTITY(1,1) NOT NULL,
        pupil nvarchar(500),
```

```
privateNumber nvarchar(50),
currentDate date,
firstIn time(0),
graphycStartTime time(0),
late time(0)
)
create table #DatesTable
(
[ID] [int] IDENTITY(1,1) NOT NULL,
Day date
)
 insert into #DatesTable
 select * from @ListDates Order by cast(ListItem as datetime) asc

 create table #pupilsTable
(
[ID] [int] IDENTITY(1,1) NOT NULL,
pupilId int
)
 insert into #pupilsTable
 select * from @ListPupils
 CREATE TABLE #TempTableHolidays
(
[ID] [int] IDENTITY(1,1) NOT NULL,
HolidayDate date
)
end--ცვლადები და ცხრილები (დასასრული)

set @CIdDay=1
set @MIdDay=(select MAX(ID) from #DatesTable)
```

```
        while @CIdDay<=@MIdDay
        begin--დღეების while
        set @date=(select Day from #DatesTable where ID=@CIdDay)
        set @CIdpupil=1
        set @MIdpupil=(select MAX(ID) from #pupilsTable)
        while @CIdpupil<=@MIdpupil
        begin--pupil-ის while
        set @pupilId=(select pupilId from #pupilsTable where
ID=@CIdpupil)
        declare @graphycStartTime as time,
              @firstIn as time
        set @graphycStartTime = (select
[dbo].[GetGraphycStartTimeByPupilAndDate](@pupilID,@date,@branc
hId))
        set @firstIn = (select
[dbo].[GetPupilFirstInByDate](@pupilId,@date,@branchId))
        if(@firstIn > @graphycStartTime)
        begin
            insert into #TempReturnData (pupil,
                                         PrivateNumber,
                                         CurrentDate,
                                         FirstIn,
                                        GraphycStartTime,
                                        Late)
            select firstName+' '+lastName,
                        personalId,
                        @date,
                        @firstIn,
                        @graphycStartTime,
                CONVERT(CHAR(8),DATEADD(second,datediff(SECOND,
```

```
                @graphycStartTime, convert(time,@firstIn)),0),108)
                  from Pupils where ID=@pupilId
              end
set @CIdpupil=@CIdpupil+1
end--pupil-ის while (დასასრული)
set @CIdDay=@CIdDay+1
end--დღეების while (დასასრული)
select pupil,privateNumber,currentDate,firstIn,graphycStartTime,late
from #TempReturnData
End
```

რომელიც თავის მხრივ იყენებს Scalar-valued ფუნქციას :

```
ALTER FUNCTION [dbo].[GetGraphycStartTimeByPupilAndDate]
(
  @pupilId as int,
  @date as date,
  @branchId int
)
RETURNS time
AS
BEGIN
 DECLARE
  @startTime time,
  @WeekDayId as int
set @WeekDayId= DATEPART(weekday,@date)
set @startTime=(select min(startTime) as 'startTime'
                                    from
[dbo].[GetPupilGraphycInfoById](@pupilId,@branchId) g
inner join WeekDays w on   g.weekDay=w.weekDayName
                                    where w.ID=@weekDayId)
```

```
RETURN @startTime
END
```

შესაბამისი ფუნქცია კი იყენებს შემდეგ Table-valued ფუნქციას :

```
ALTER FUNCTION [dbo].[GetGraphycInfoById]
(
  @graphycId int,
  @branchId int
)
RETURNS TABLE
AS
RETURN
   (select
ROW_NUMBER() OVER ( partition by w.ID ORDER BY w.ID asc) AS
weekDayRn,
            w.weekDayName as 'weekDay',
            t.startTime,
             t.endTime ,
            s.name as 'subject',
            e.firstName +' '+ e.lastName as 'teacher',
            cr.name as 'classroom',
            gr.ID 'groupId',
            gr.name as 'group'
            from Graphycs g
            inner join Graphycs_Times gt on g.ID=gt.graphycId
            left join WeekDays w on gt.weekDayId=w.ID
            left join Times t on gt.TimeID=t.ID
            left join Classes_subjects_emps cse on gt.ID=cse.graphycTimeId
            left join Subjects s on cse.subjectId=s.ID
            left join Employees e on cse.employeeId=e.ID
```

```
        left join ClassRooms cr on gt.classRoomId=cr.ID
        left join groups gr on cse.groupId=gr.ID
    where g.ID=@graphycId and cse.activeStatus=1 and
g.branchId=@branchId)
```

4.4. ღრუბლოვანი საცავები და მონაცემთა ბაზები

ორგანიზაციებში ინფორმაციის მოცულობის ექსპონენციალური მატება მოითხოვს მისი ტექნიკური რესურსების მნიშვნელოვან ექსტენსიონალურ ზრდას, რაც გარკვეულ ფინანსურ, საკადრო და სამუშაო ფართის პრობლემებთანაა დაკავშირებილი. ახალი ციფრული ტექნოლოგიების განვითარებამ და შესაბამისი სერვისული მომსახურების გაფართოებამ, განსაკუთებით ბოლო ათწელეულში, წარმოშვა და დაამკვიდრა უპრეცედენტო მოთხოვნილება ე.წ. „ღრუბლოვანი საცავების“ სახით (Cloud Data Warehouse) [68]. ეს კი რესურსების გამოყენების ინტენსიონალური მიდგომაა.

იგი მონაცემთა (ბაზების) ერთობლიობაა, რომელიც ინახება მართვადი სერვისის სახით საერთო გამოყენების ღრუბელში და ოპტიმიზებულია მასშტაბირებადი ბიზნესანალიზისათვის (BI) [68].

ამგვარად, ღრუბლოვანი მონაცემთა საცავების (არქიტექტურების), დანერგვამ დიდად შეუწყო ხელი, მეტწილად, მცირე და საშუალო ბიზნესის ობიექტებს, შეზღუდული დანახარჯებით გაეგრძელებინათ დიდ მონაცემა შენახვა და ანალიზი. შესაძლებელი გახადა ინფორმაციის განსხვავებული წყაროებიდან, როგორიცაა ERP, CRM, IoT და სხვ. მონაცემების უწყვეტი მიღება.

მონაცემთა ღრუბლოვანი საცავების ფუნქციონირება (ინფორმაციის შეტანა/გამოტანა და დამუშავება) ხორციელდება მრავალი

სერვერის პარალელური მუშაობის რეჟიმში, მათი დატვირთვების გადანაწილებით და მართვით.

სვეტოვანი მონაცემთა საცავები (Columnar data stores) ყველაზე მოქნილი და ეკონომიურია ანალიტიკისთვის. ასეთი ბაზები ინახავს და ამუშავებს მონაცემებს სვეტების (ველების) მიხედვით, ნაცვლად სტრიქონებისა. იგი ახორციელებს მთლიანი მოთხოვნების მკვეთრად უფრო სწრაფად მუშაობას და ამიტომაც ხშირად გამოიყენება ანგარიშგებისთვის.

ამგვარად, ღრუბლოვანი არქიტექტურა აერთიანებს სამ ძირითად ელემენტს: *მონაცემთა შენახვის სიმძლავრეს, დიდ მონაცემთა პლატფორმის მოქნილობას და ღრუბლის ელასტიურობას.* ეს საკითხები უშუალოდაა დაკავშირებული გადაწყვეტილების მიღებასთან - თუ რომელი ვარიანტი იქნას არჩეული ჩვენი კონკრეტული დასაპროექტებელი სისტემისთვის,

აქ ერთ-ერთი მნიშვნელოვანი საკითხია მონაცემთა ღრუბლოვანი საცავების ავტომატიზაცია (Cloud Data Warehouse Automation).

მონაცემთა ინტეგრაციის ზოგიერთ თანამედროვე პლატფორმაზე ავტომატიზიებულია მონაცემთა საცავის მთელი სასიცოცხლო ციკლი, რათა დაჩქარდეს ანალიტიკის პროცესისთვის მზა მონაცემების წვდომადობა.

მოდელზე ორიენტირებული მიდგომა ეხმარება მონაცემთა ბაზების და საცავების ინჟინრებს შექმნან, განათავსონ და მართონ ღრუბლოვანი მონაცემთა საცავები.

მომდევნო პარაგრაფებში დეტალურად განვიხილავთ SQL და NoSQL ტიპის ბაზების შექმნის, განვითარებისა და გამოყენების საკითხებს ღრუბლოვან საცავებში.

4.4.1. NoSQL ბაზა MongoDB Atlas

NoSQL ტიპის მონაცემთა ბაზების მოკლე მიმოხილვა ჩვენ წარმოვადგინეთ მეორე თავის 2.5.1 პარაგრაფში [12]. ამ სახის დოკუმენტური ტიპის ბაზებიდან ამჯერად განვიხილავთ MongoDB Atlas ბაზას, რომელიც ღრუბლოვანი ტექნოლოგიის სერვისია [68].

აქ წარმოდგენილი თემა სპეციალურად იქნა შემუშავებული ინფორმატიკის დიდაქტიკის კურსში და დაინერგა სტუ-ის სტუდენტებისთვის სასწავლო პროცესში პროგრამული ინჟინერიის კონცენტრაციის პროგრამით [68].

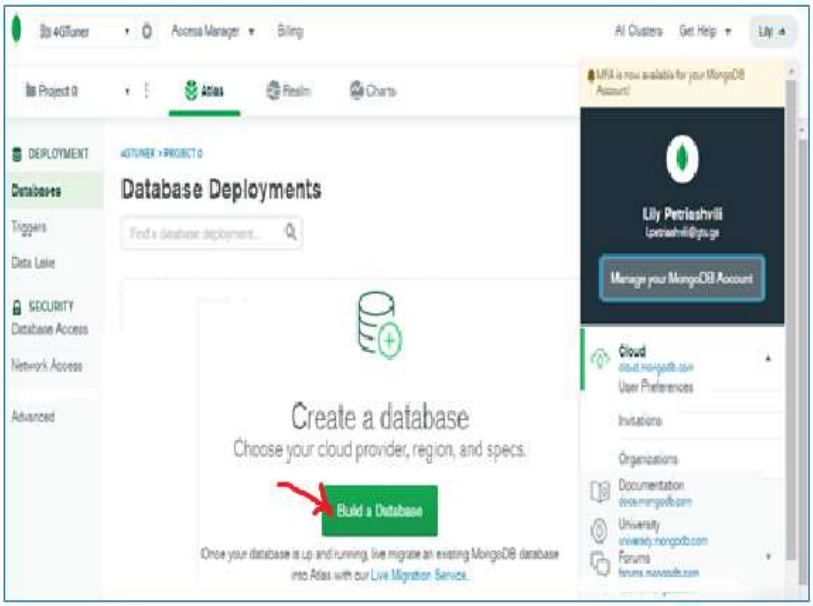
MongoDB Atlas პირველი ღრუბლოვანი ვერსიაა ამ ტიპის მონაცემთა ბაზებიდან. იგი მომხმარებელს აძლევს საშუალებას წარადგინოს ერთდროულად რამდენიმე აპლიკაცია მსხვილ ღრუბლოვან პროვაიდერებთან. MongoDB-ს მონაცემთა ბაზა შესაძლებელია განთავსდეს ისეთ ღრუბლოვან სერვერებზე, როგორიცაა AWS (Amazon Web Services), Azure ან GCP (Google Cloud Platform) [110, 111].

MongoDB Atlas არის მულტიღრუბლოვანი მონაცემთა ბაზა სერვისებით. იგი ახორციელებს მომხმარებელთა მონაცემთა ბაზების განლაგებას ღრუბლებში და მათ მართვას, ხელს უწყობს ღრუბლოვან პროვაიდერებზე მდგრადი და ეფექტიანი გლობალური აპლიკაციების შექმნას. იგი სთავაზობს დეველოპერებს კლასტერიზაციის საშუალებით ისარგებლონ, ყოველი პროვაიდერის მიერ წარმოდგენილი საუკეთესო აპლიკაციით, რაც უმარტივებთ და ეფექტურს ხდის მათ მუშაობას.

MongoDB Atlas პლატფორის სერვისი ავტომატურად აკონფიგურირებს მონაცემთა ბაზას და განათავსებს მას როგორც

უნიკალურს. ეს სერვისი მომხმარებელს ათავისუფლებს NoSQL მონაცემთა ბაზის დაპროექტების და მართვისთვის საჭირო სამუშაოების შესრულებისგან და საშუალებას აძლევს კონცენტრირდეს უშუალოდ აპლიკაციის მართვაზე.

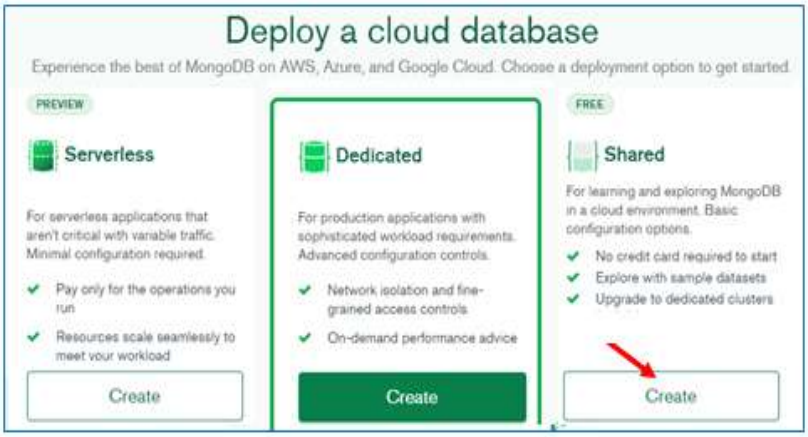
MongoDB Atlas პლატფორმის გამოსაყენებლად საჭიროა დარეგისტრირება (https://www.mongodb.com/cloud/atlas) [68]. ამის შემდეგ შესაძლებელია მონაცემთა ბაზის აგება (ნახ. 4.17).



ნახ.4.17

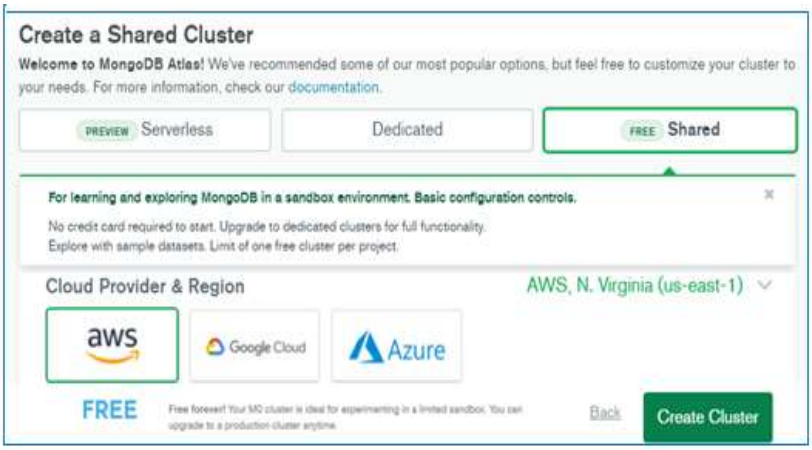
Build a Database ღილაკით გადავდივართ გარემოში, სადაც შესაძლებელია შევარჩიოთ პლატფორმის გამოყენების სხვადასხვა სერვისი. ჩვენ შემთხვევაში ვაკეთებთ თავისუფალ არჩევანს,

რომელიც განკუთვნილია არაკომერციული სასწავლო პროცესისთვის და ნიმუშის შესაბამისად არჩევანს ვადასტურებთ ღილაკით Create (ნახ. 4.18).



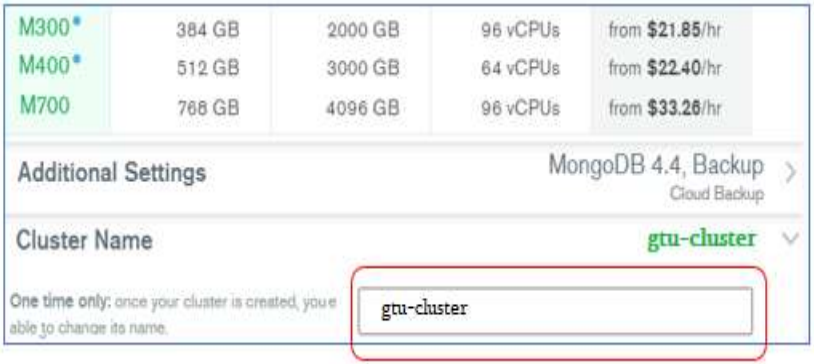
ნახ.4.18

შემდეგ გადავდივართ კლასტერის შექმნის და ფორმირების ეტაპზე (ნახ. 4.19).



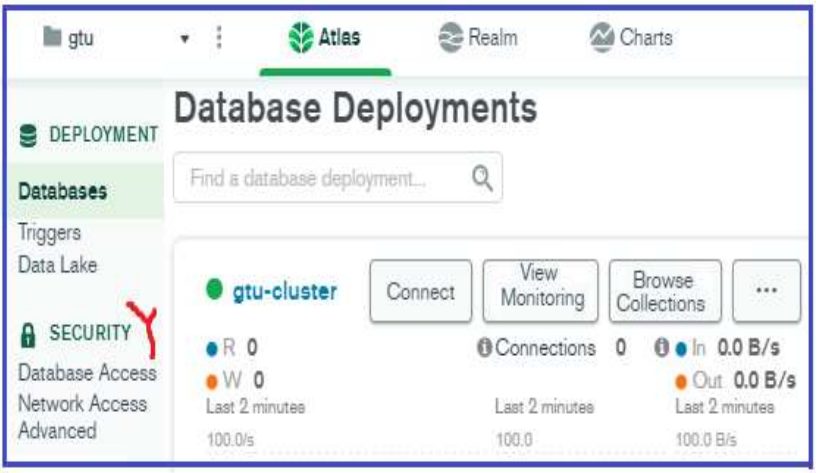
ნახ.4.19

ვირჩევთ მაგალითად, AWS ღრუბლოვან პროვაიდერს. ფანჯრის ქვედა ნაწილის ველში Cluster Name ჩავწერთ კლასტერის დასახელებას, gtu-claster (ქვედა რეგისტრით და რიცხვები, Space-ს გარეშე) (ნახ. 4.20).



ნახ.4.20

Create Claster-ით გადავალთ ფანჯარაში ჩვენ მიერ შექმნილი კლასტერით - „gtu-cluster" (ნახ. 4.21).



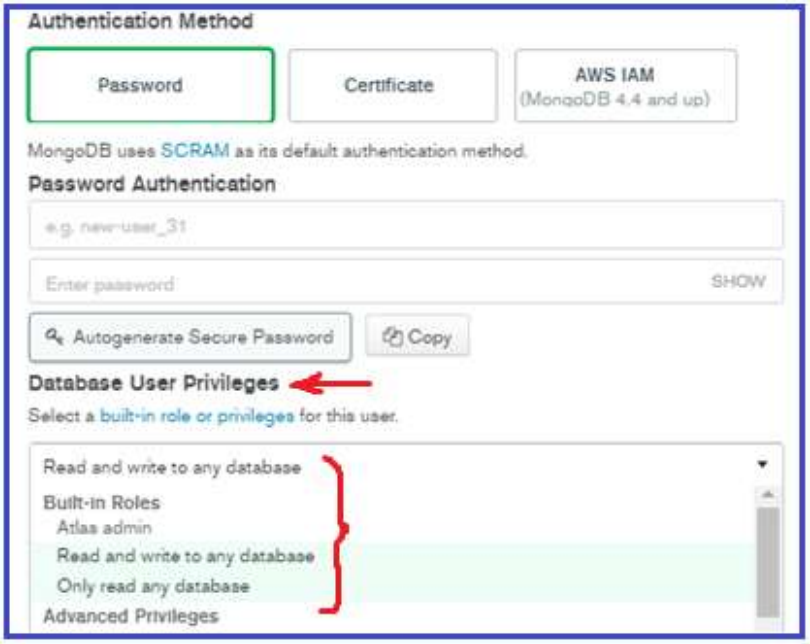
ნახ.4.21

კლასტერის შექმნის შემდეგ ფანჯრის მარცხენა მხარეს მოთავსებული **SECURITY** – **Database Access** ჩანართიდან უნდა განისაზღვროს სისტემის მომხმარებელთა ჩამონათვალი, ვისაც შეეძლებათ ინფორმაციის მიღება და ჩაწერა. აღნიშნული ჩანართის გააქტიურებისას ქვემოთ მოცემულ ფანჯარაში ვამატებთ Database Users – მომხმარებლებს (ნახ. 4.22).

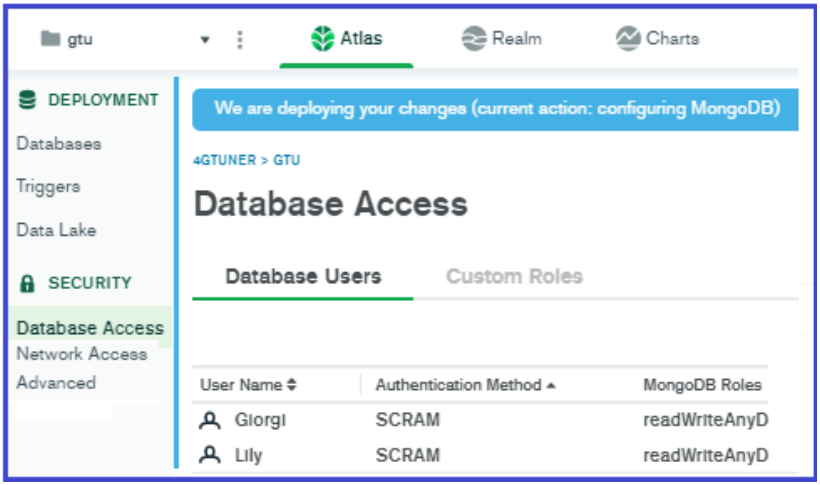
ნახ.4.22

ახალი მომხმარებლების შესახებ მონაცემთა განსაზღვრისას ვააქტიურებთ ფანჯრის ქვედა მარჯვენა ნაწილში მოთავსებულ ღილაკს Add New Database User. შედეგად მიიღება ფანჯარა, რომლის ველებშიც შეგვაქვს მომხმარებლის სახელი და პაროლი (ნახ. 4.23).

ამ ფანჯრის ჩანართიდან **Database User Privileges** ყოველ მომხმარებელს შესაძლებელია მივანიჭოთ ან შევუზღუდოთ უფლებები, აღნიშნულ ჩანართში მოცემული შეთავაზებების შესაბამისად. მაგალითად, 4.24 ნახაზზე წარმოდგენილია ორი მომხმარებელი, რომელთაც წვდომა ექნება ჩვენს ღრუბლოვან მონაცემთა ბაზაზე.

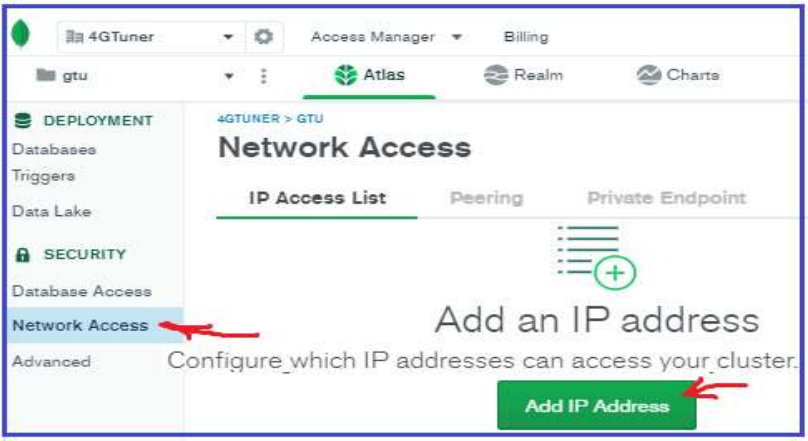


ნახ.4.23



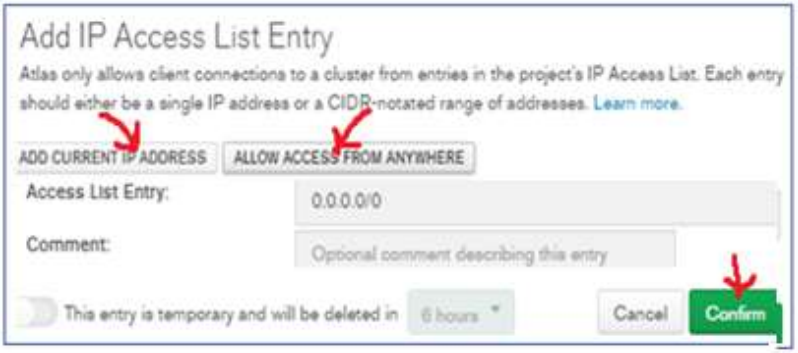
ნახ.4.24

ინტერფეისზე მომდევნო ჩანართია **Network Access,** სადაც უნდა განისაზღვროს მომხმარებლის IP მისამართები. ჩანართის გააქტიურების შედეგად მიიღება ფანჯარა (ნახ. 4.25).



ნახ.4.25

Add IP Addresse ღილაკის გააქტიურებით მიიღება ფანჯარა, სადაც ემატება ის IP მისამართები, რომელთაც Atlas-ი აძლევს სისტემასთან წვდომის უფლებას (ნახ. 4.26).



ნახ.4.26

კონკრეტულ მომხმარებელთა მისამართებს ვამატებთ ღილაკით ADD CURRENT IP ADDRESS, ხოლო თუ არ არის ამის აუცილებლობა, მაშინ ვააქტიურებთ ღილაკს ALLOW ACCESS FROM ANYWERE (დასაშვებია წვდომა ნებისმიერი ადგილიდან). ბოლოს ვადასტურებთ Confirm ღილაკით.

დავამატეთ ორი IP მისამართი, რომელიც არის აქტიური და აქვს წვდომა Atlas პლატფორმაზე. შესაბამისი საილუსტრაციო მაგალითი წარმოდგენილია 4.27 ნახაზზე.

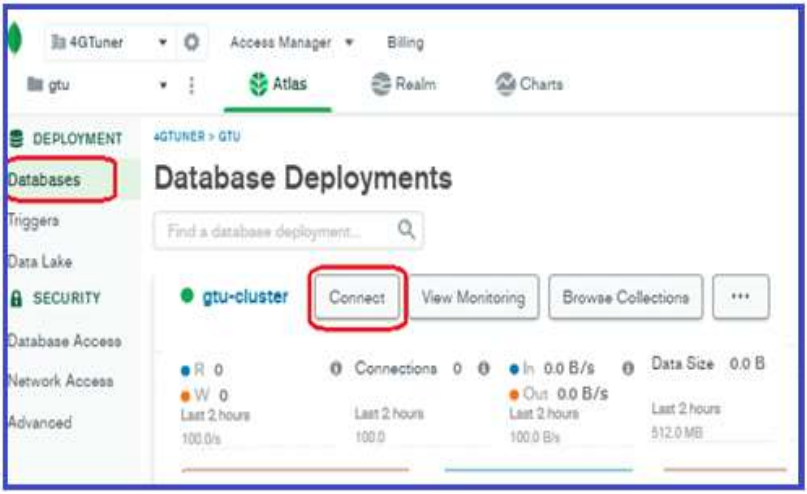
ნახ.4.27

ამგვარად, შექმნილია კლასტერი, დამატებულია მომხმარებლები და განსაზღვრულია IP მისამართები.

შემდეგი ეტაპია MongoDB ბაზასთან დაკავშირება და მონაცემებთან წვდომა.

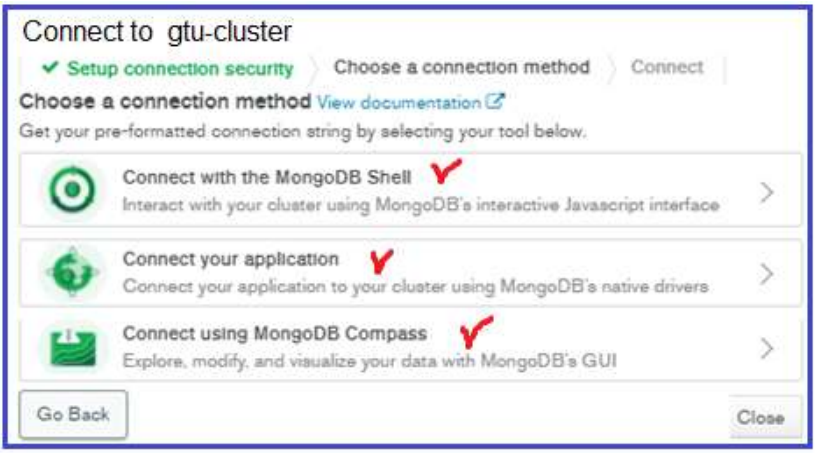
- **დაკავშირება მონაცემთა ბაზასთან.**

მონაცემთა ბაზასთან დასაკავშირებლად ვააქტიურებთ ჩანართს Databases და შემდეგ ვირჩევთ Connect-ს (ნახ. 4.28).



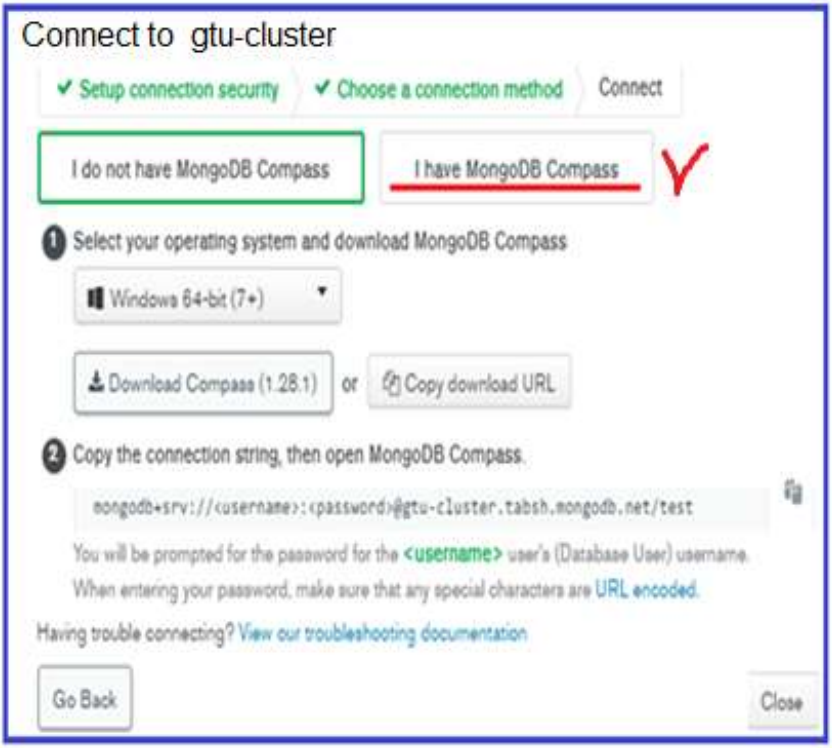
ნახ.4.28

მიიღება ფანჯარა, საიდანაც კლასტერის დაკავშირება შესაძლებელია, როგორც MongoDB Shell-ის, ასევე MongoDB Compass-გარემოსთან (ნახ. 4.29).



ნახ.4.29

კლასტერი დავაკავშიროთ MongoDB Compass-თან. გავააქტიუროთ ღილაკი Connect using MongoDB Compass. შედეგად მიიღება ფანჯარა ორი ღილაკით (ნახ. 4.30), პირველი - მომხმარებელს არა აქვს MongoDB Compass და შეუძლია ჩამოტვირთოს იგი. მეორე, როდესაც არის MongoDB Compass და შესაძლებელია სარგებლობა. ჩვენ შემთხვევაში მოვნიშნოთ ღილაკი I have MongoDB Compass.

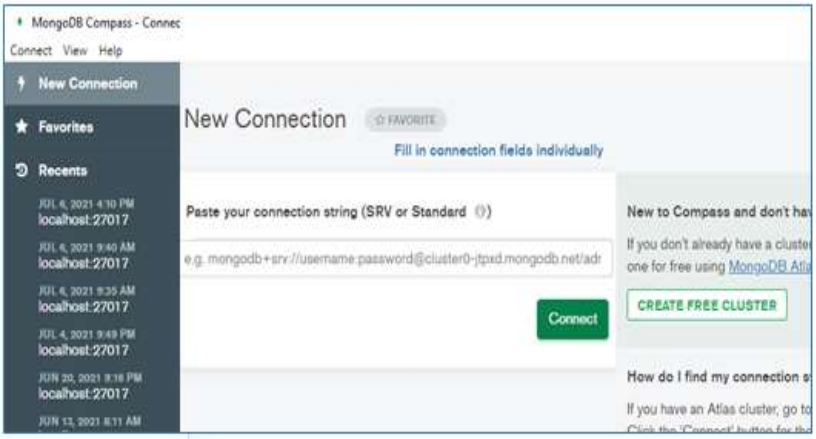


ნახ. 4.30

შედეგად მიიღება ფანჯარა, საიდანაც ვაკოპირებთ სტრიქონს MongoDB Compass-ისთვის (ნახ. 4.31).

ნახ. 4.31

პარალელურად, ლოკალურად ვხსნით ჩვენს MongoDB Compass გარემოს (ნახ. 4.32).



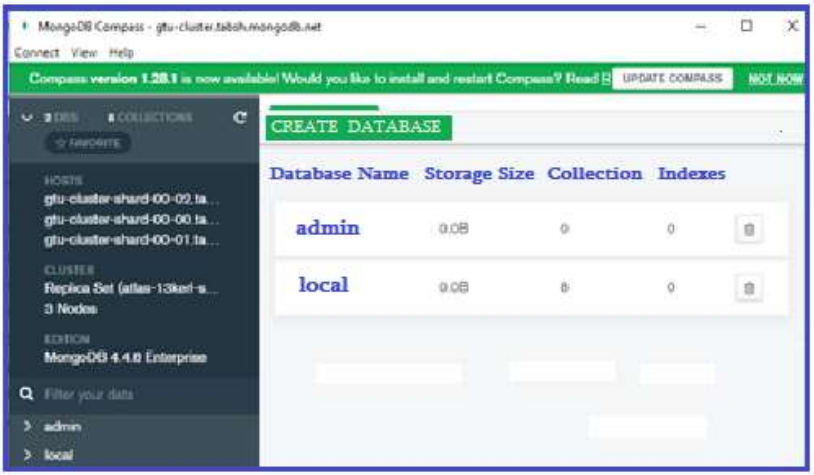
ნახ. 4.32

დაკოპორებულ სტრიქონს ჩავსვამთ სპეციალურად გამოყოფილ არეში (ნახ. 4.33).

ნახ. 4.33

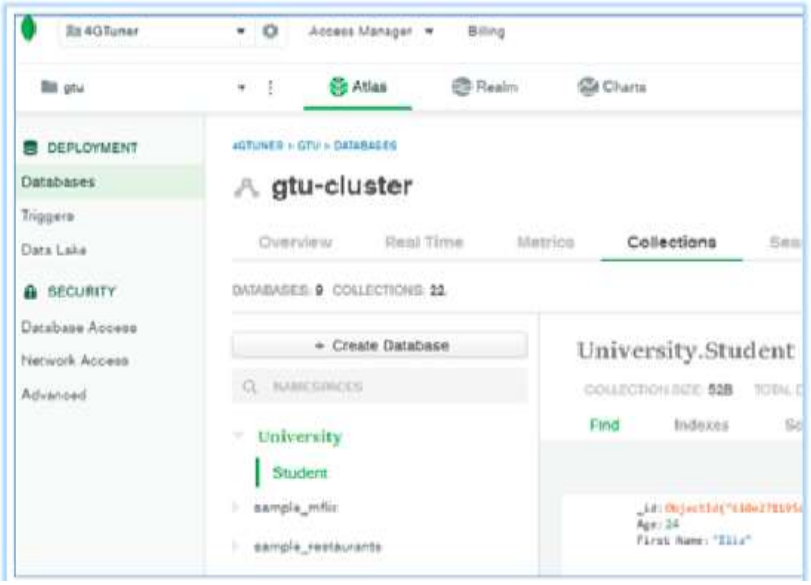
დავაკორექტირებთ მომხმარებლის სახელს და ჩავწერთ შესაბამის პაროლს, რასაც ვადასტურებთ Connect ღილაკით.

შედეგად მოხდება დაკავშირება, მიიღება ფანჯარა, სადაც ვამატებთ ახალ ბაზას. ვქმნით კოლექციებს (ნახ. 4.34).



ნახ. 4.34

MongoDB Atlas გარემოში განახლების ღილაკის გააქტიურების შემდეგ აისახება ჩვენს მიერ შექმნილი, მაგალითად, Universty ბაზა, რომლის კოლექციის დასახელებაა Student (ნახ. 4.35).



ნახ. 4.35

მიღებული შედეგის თანახმად ლოკალური MongoDB Compass გარემოდან ავსახეთ ღრუბლოვან MongoDB Atlas მონაცემთა ბაზაში ჩვენ მიერ ფორმირებული მონაცემთა ბაზა, რომლის შემდგომი ანალიზისა და დამუშავებისათვის შესაძლებელია გამოყენებულ იქნას ყველა თანამედროვე ტექნოლოგია და ინსტრუმენტული საშუალება.

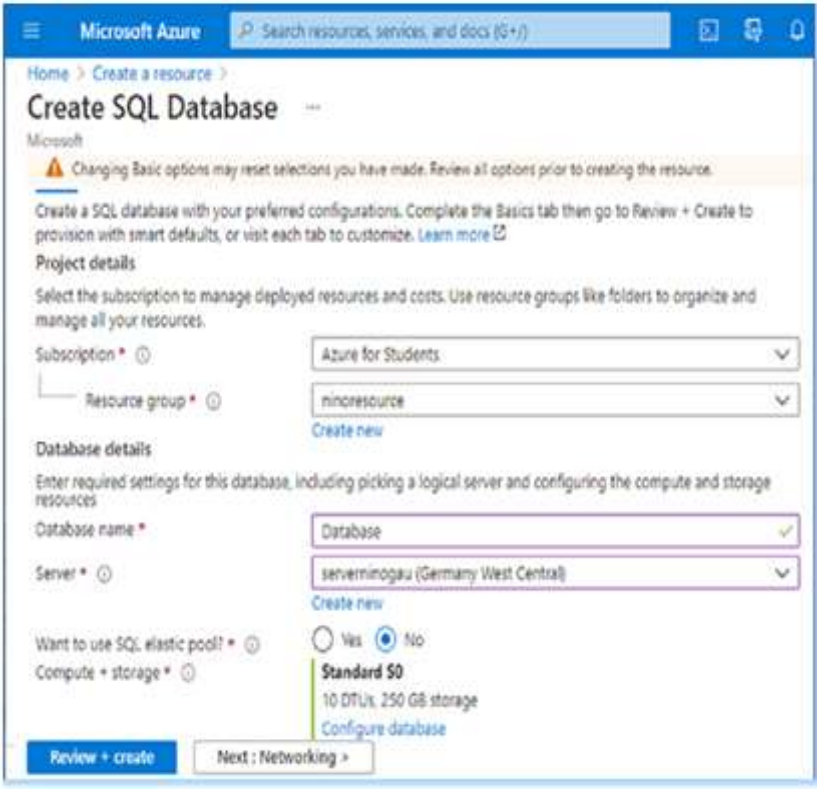
4.4.2. Microsoft Azure SQL

Ms Azure SQL მართული ღრუბლოვანი მონაცემთა ბაზაა. იგი მუშაობს ღრუბლოვან გამოთვლით პლატფორმაზე და მასზე წვდომა უზრუნველყოფილია სერვისის სახით. მართული

მონაცემთა ბაზის სერვისები ზრუნავს მონაცემთა ბაზის მასშტაბურობაზე, სარეზერვო და მაღალ წვდომადობაზე.

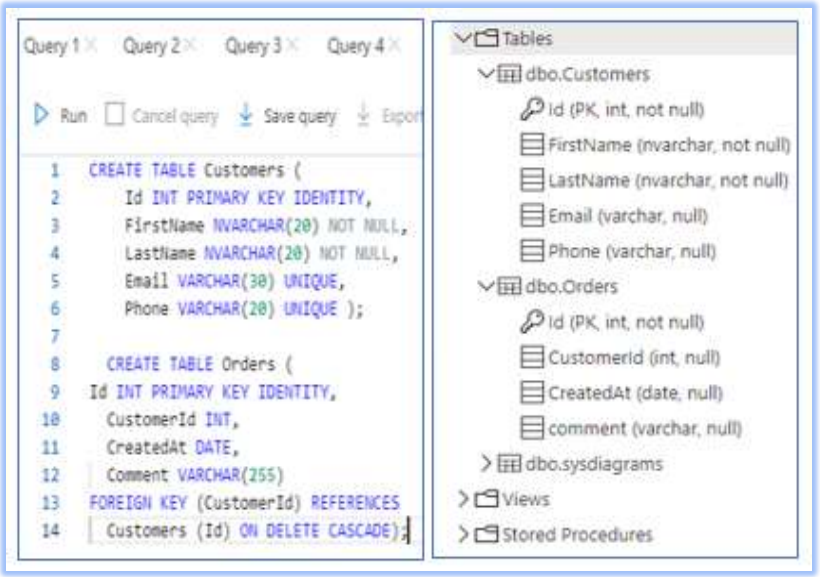
პროგრამული აპლიკაცია თუ front_end-ია, Azure SQL არის back_end. იგი გამოიყენება OLTP-ში (ტრანზაქციების ონლაინ-დამუშავება) დროის რეალურ რეჟიმისთვის. Azure SQL *მონაცემთა საცავი* კი ოპტიმიზებულია მონაცემთა ანალიზის ამოცანების შესასრულებლად, დიდ მონაცემთა დასამუშავებლად.

Azure პორტალზე ავირჩიოთ ბრძანება SQL Database →Create და შევავსოთ დიალოგური ფანჯარა (ნახ. 4.36-ა) [63, 68].



ნახ. 4.36-ა. მონაცემთა ბაზის შექმნა Azure SQL-ში

გამოყენებითი პროგრამული აპლიკაციის შექმნის პროცესის უკეთ გააზრების მიზნით განვიხილავთ კონკრეტულ მაგალითს ბიზნეს-ამოცანისათვის, ბაზის ცხრილებით Customers და Orders. ნახ. 4.36-ბ. [63].



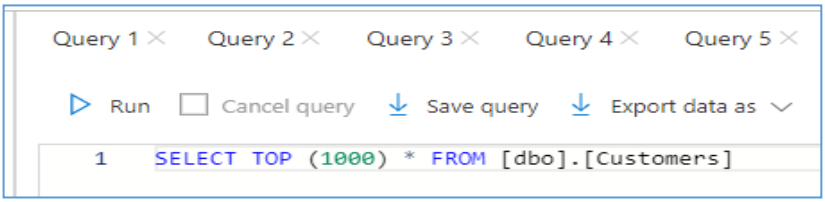
ნახ.4.36-ბ. ცხრილების დაპროექტება Azure SQL-ში

ბაზის Customers ცხრილი შევავსოთ ჩანაწერებით:

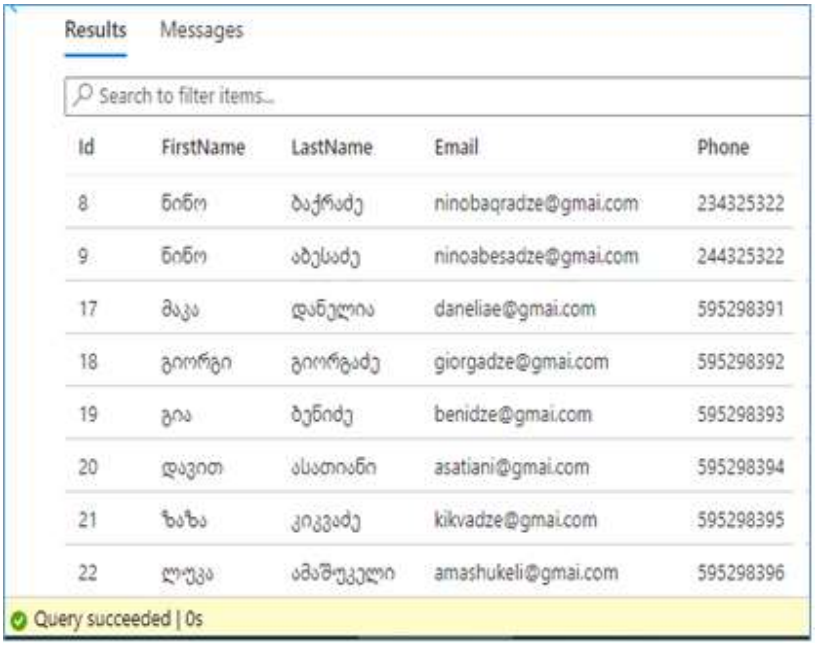
```
insert into [dbo].[Customers] values (N'მაკა', N'დანელია',
                                'daneliae@gmail.com', 595298391)
insert into [dbo].[Customers] values (N'გიორგი', N'გიორგაძე',
                                ‘giorgadze@gmail.com', 595298392)
insert into [dbo].[Customers] values (N'გია', N'ბენიძე',
                                'benidze@gmail.com', 595298393)
insert into [dbo].[Customers] values (N'დავით', N'ასათიანი',
                                 'asatiani@gmail.com', 595298394)
```

```
insert into [dbo].[Customers] values (N'ზაზა', N'კიკვაძე',
                                      'kikvadze@gmail.com', 595298395)
insert into [dbo].[Customers] values (N'ლუკა', N'ამაშუკელი',
                                      'amashukeli@gmail.com', 595298396)
```

შედეგები ნაჩვენებია 4.37-ა,ბ ნახაზებზე.



ნახ.4.37-ა



ნახ.4.37-ბ

Office 365-ის ერთ-ერთი აპლიკაციაა *Power Automate.* იგი ღრუბლოვანი სერვისია, რომლის საშუალებით მომხმარებელს შეუძლია დამოუკიდებლად შექმნას სამუშაო პროცესები, განახორციელოს განმეორებადი, ხელით შესასრულებელი სამუშაოების ავტომატიზაცია. ეს კი ამარტივებს ბიზნეს პროცესებს და მათ ეფექტურად მართვას [112].

Power Automate დაფუძნებულია ტრიგერებზე (triggers) და მოქმედებებზე (actions). ტრიგერის საშუალებით იწყება ნაკადი ანუ ვირჩევთ მოვლენას, რომლის მიხედვითაც იგეგმება მოქმედება, ანუ ის, რაც ხდება ნაკადის გააქტიურების შემდეგ. ეს შეიძლება იყოს დავალების შექმნა ან ერთი ან მეტი მოქმედების შესრულება.

არსებობს *ხუთი ტიპის ნაკადის (flow)* შექმნის საშუალება:

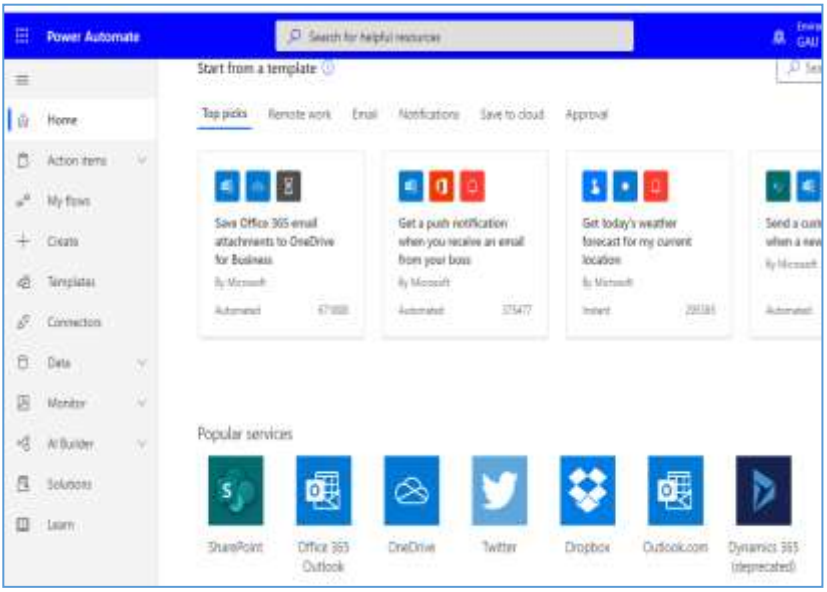
- *ავტომატიზებული* – მოვლენის შედეგად გამოწვეული ნაკადი, მაგალითად, გაიგზავნოს მეილი, თუ SharePoint სიის ელემენტი შეიცვალა;
- *მყისიერი* – მომხმარებელს საშუალებას აძლევს ხელით იმოქმედოს მობილურიდან ან აპლიკაციიდან ღილაკი. მაგალითისთვის, მარტივად გაგზავნოს შეხსენების მეილი ორგანიზაციის წევრებთან შეხვედრის დაწყებამდე;
- *დაგეგმილი* – იწყებს მუშაობას გარკვეულ დროს;
- *ბიზნესპროცესების ნაკადი,* რომელიც ემყარება განსაზღვრულ მოქმედებათა ერთობლიობას;
- *UI (user interface) ნაკადები,* რომლებიც გამოიყენება Windows და Web პროგრამებში განმეორებადი ამოცანების ავტომატიზაციისათვის.

➢ **ბიჯი 1:** *Power Automate გააქტიურება*

რომელიმე ინტერნეტ-ბრაუზერში მივცეთ: **portal.office.com** (ნახ.4.38), შევიტანოთ მომხმარებლის სახელი და პაროლი.

ნახ.4.38

გააქტიურდება Office 365 მთავარი გვერდი, ავირჩიოთ **Power Automate** (ნახ.4.39).

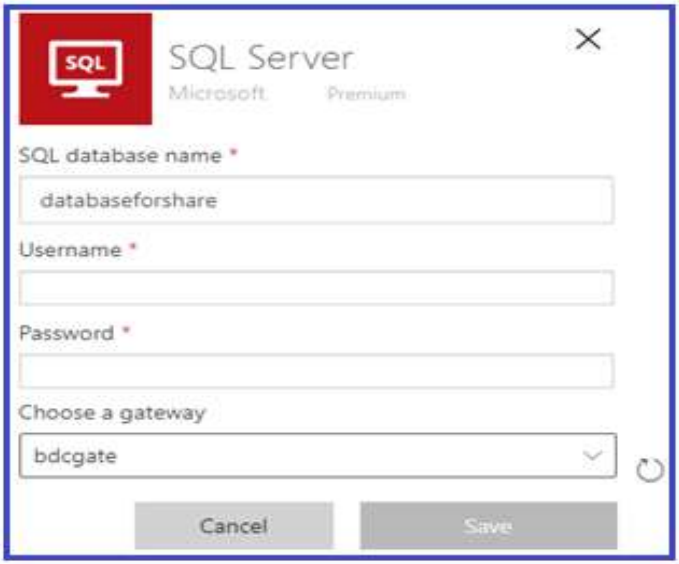


ნახ.4.39

➢ **ბიჯი 2:** *ეკრანზე გამოჩნდება Power Automate-ს შაბლონები. ძებნის ველში ვუთითებთ Sharepoint. მიიღება* ფანჯარა (ნახ.4.40).

ნახ.4.40

შევავსოთ 4.41 ნახაზზე ნაჩვენები ფანჯრის საჭირო ველები,



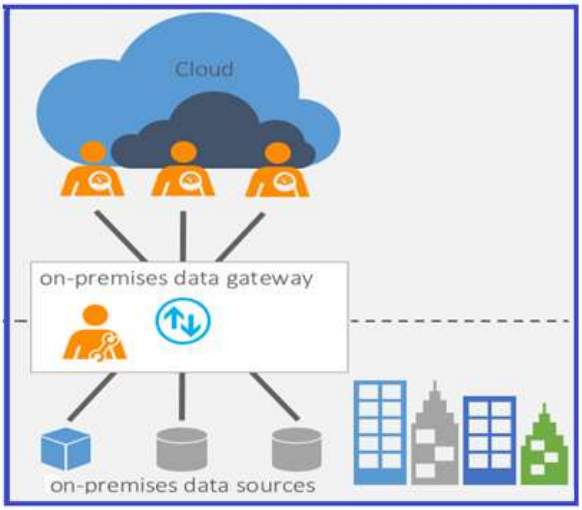
ნახ.4.41

მაიკროსოფთის საიტიდან ჩამოვტვირთოთ და დავაინსტალიროთ Gateway.

on-premises data gateway მოქმედებს როგორც ხიდი, რათა უზრუნველყოს მონაცემთა სწრაფი და უსაფრთხო გადაცემა შიგა მონაცემებს შორის (მონაცემები, რომლებიც არ არის ღრუბელში) და Microsoft-ის სხადასხვა ღრუბლოვან სერვისს შორის.

ეს ღრუბლოვანი სერვისები მოიცავს (ნახ.4.42):

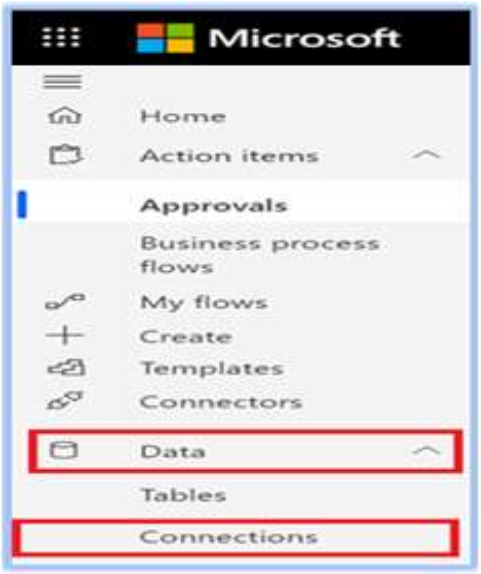
- Power BI, Power Apps;
- Power Automate;
- Azure Analysis Services და
- Azure Logic Apps.



ნახ.4.42

გამოვიყენოთ Power Automate სერვისი სამუშაო პროცესების ავტომატიზაციის მიზნით. კერძოდ, მოვახდინოთ Sharepoint-ის სიაში შეტანილი მონაცემთა სინქრონულად გადატანა Azure SQL-ში.

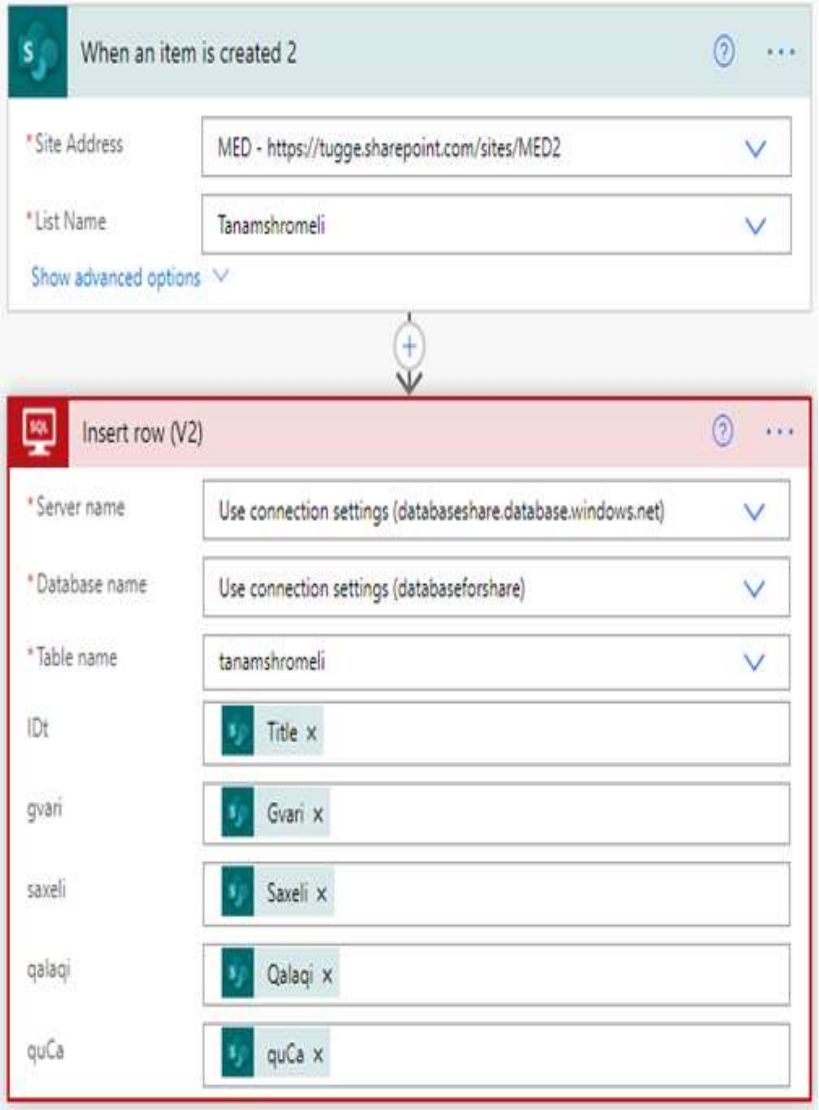
დავამატოთ ახალი კავშირი (ნახ.4.43).



ნახ.4.43

1) Power Automate-ის საშუალებით გავიაროთ ავტორიზაცია;
2) ავირჩიოთ
   Data -> **Connections -> New connection;**
3) ავირჩიოთ **SharePoint**

მოვახდინოთ სამუშაო ნაკადის კონფიგურაცია. 4.44 ნახაზზე ნაკადი დეტალურად აღწერს Azure SQL-ის tanamshromeli-ს ცხრილის ველების შევსების პროცესს.


When an item is created 2
*Site Address
MED - https://tugge.sharepoint.com/sites/MED2
*List Name
Tanamshromeli
Show advanced options
Insert row (V2)
*Server name
Use connection settings (databaseshare.database.windows.net)
*Database name
Use connection settings (databaseforshare)
*Table name
tanamshromeli
IDt
Title x
gvari
Gvari x
saxeli
Saxeli x
qalaqi
Qalaqi x
quCa
quCa x

ნახ.4.44

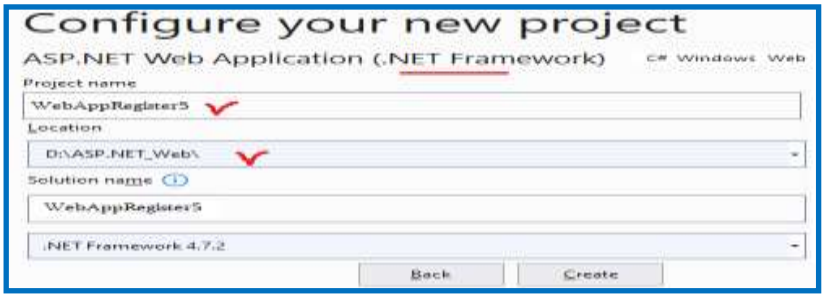
4.5. Web-ინტერფეისის აგება ASP.NET-პაკეტით, უსაფრთხოება და სერვისები

4.5.1. სერვისის ინტერფეისის Web-გვერდის აგება

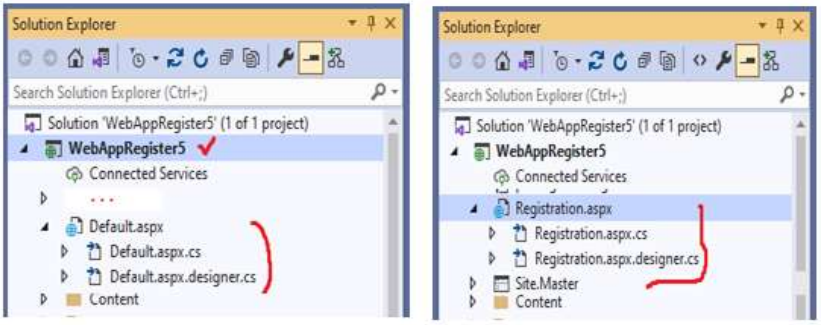
ამჯერად განვიხილოთ ამოცანა მომხმარებლის ინტერაქტიული ინტერფეისის დასაპროგრამებლად Web-აპლიკაციის სახით. იგი შეიძლება გამოიყენოს მასწავლებელმა, მოსწავლემ და სხვა როლებმა, ვისაც აქვს ამის ნებართვა.

ამოცანა: Visual Studio .NET პლატფორმაზე ASP.NET-ის გამოყენებით შევქმნათ Web-გვერდი, რომელზეც მომხმარებელი შეიტანს საკუთარ მონაცემებს და გადააგზავნის სერვერზე [68].

– შევქმნათ ASP.NET აპლიკაცია WebAppRegister5 სახელით (ნახ.4.45) და Default.aspx შევცვალოთ Registration-ით (ნახ.4.46).

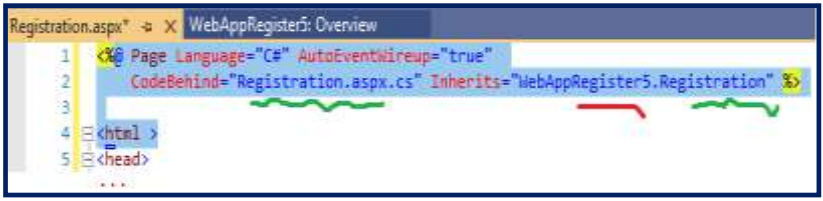


ნახ.4.45



ნახ.4.46

– Registration.aspx ფაილის 1-ელ სტრიქონს ექნება 4.47 ნახაზზე ნაჩვენები სახე, სადაც ასახულია პროექტის და .aspx და .aspx.cs პროგრამების სახელები.

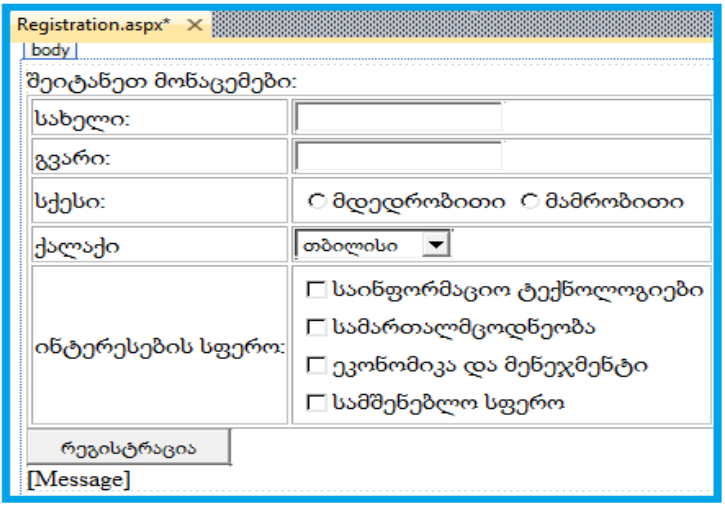


ნახ.4.47

– Registration.aspx.cs ფაილის საწყისი ტექსტის ლისტინგი ასე გამოიყურება;

```
// ----- ლისტინგი_4.7  Registration.aspx.cs.------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace WebAppRegister5
{
   public partial class Registration : Page
   {
      protected void Page_Load(object sender, EventArgs e) {  }
   }
}
```

– Web-გვერდის სარეგისტრაციო ფორმის მაკეტი ნაჩვენებია 4.28 ნახაზზე.



ნახ.4.48. სერვისის ინტერფეისი

ფორმაზე მოთავსებულია სერვერული მართვის ელემენტები form, asp:TextBox, asp:DropDownList, asp:CheckBoxList, asp:Button, asp:Label და ა.შ., რომლებიც ასახულია Registration.aspx ფაილის 4.8 ლისტინგში. გავხსნათ Registration.aspx და შევიტანოთ შემდეგი კოდი (ან გამოვიყენოთ წიგნიდან მისი შესაბამისი პროტოტიპი):

<!— ლისტინგი_4.8 ———— –>

```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeBehind="Registration.aspx.cs"Inherits="WebAppRegister8
    .Registrati on" %>
<html >
  <head>
   <title>რეგისტრაციის ფორმა</title>
    <style type="text/css">
      .auto-style1 {
         width: 253px;     }
    </style>
```

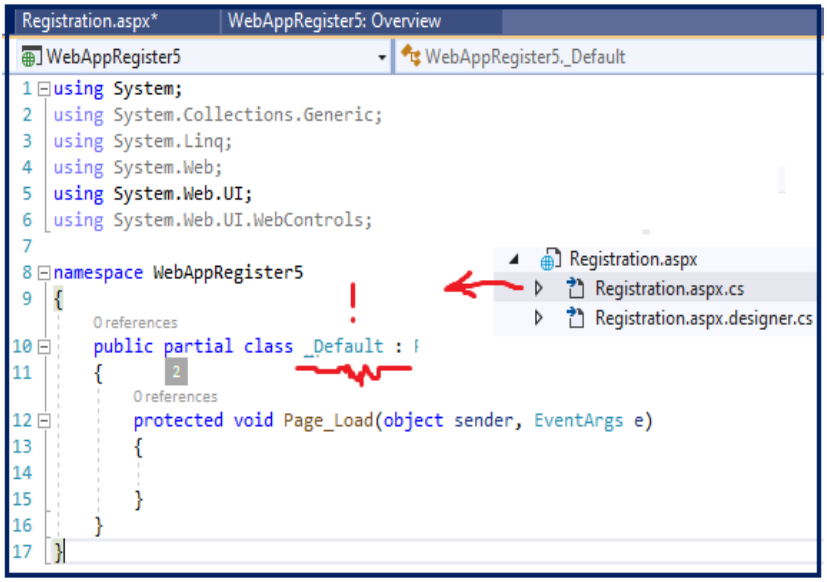
```
 </head>
<body>
 <form method="post" runat="server" id="registration">
  შეიტანეთ მონაცემები:
  <table border="1">
   <tr>
    <td>სახელი:</td>
    <td class="auto-style1">
     <asp:TextBox id="FirstName" runat="server"></asp:TextBox></td>
   </tr>
   <tr>
    <td>გვარი:</td>
    <td class="auto-style1">
     <asp:TextBox id="LastName" runat="server"></asp:TextBox></td>
   </tr>
   <tr>
    <td>სქესი:</td>
    <td class="auto-style1">
      <asp:RadioButtonList id="Sex"
                    runat="server" RepeatDirection ="Horizontal">
       <asp:ListItem Value="მდედრობითი"></asp:ListItem>
       <asp:ListItem Value="მამრობითი"></asp:ListItem>
      </asp:RadioButtonList></td>
   </tr>
   <tr>
    <td>ქალაქი</td>
    <td class="auto-style1">
      <asp:DropDownList id="City" runat="server">
        <asp:ListItem Value="თბილისი"></asp:ListItem>
        <asp:ListItem Value="ქუთაისი"></asp:ListItem>
        <asp:ListItem Value="რუსთავი"></asp:ListItem>
        <asp:ListItem Value="გორი"></asp:ListItem>
```

```
            <asp:ListItem Value="ბათუმი"></asp:ListItem>
            <asp:ListItem Value="თელავი"></asp:ListItem>
          </asp:DropDownList>
        </td>
       </tr>
       <tr>
        <td>ინტერესების სფერო:</td>
        <td class="auto-style1">
        <asp:CheckBoxList id="Interests" runat="server">
          <asp:ListItem Value="საინფორმაციო ტექნოლოგიები">
          </asp:ListItem>
         <asp:ListItem Value="სამართალმცოდნეობა"></asp:ListItem>
         <asp:ListItem Value="ეკონომიკა და მენეჯმენტი">
        </asp:ListItem>
         <asp:ListItem Value="სამშენებლო სფერო"></asp:ListItem>
        </asp:CheckBoxList></td>

       </tr>
       </table>
          <asp:Button id="Register" runat="server" Text="რეგისტრაცია"
                OnClick="Register _Click">
          </asp:Button>
        </br>
        <asp:Label id="Message" runat="server"></asp:Label>
      </form>
     </body>
    </html>
```

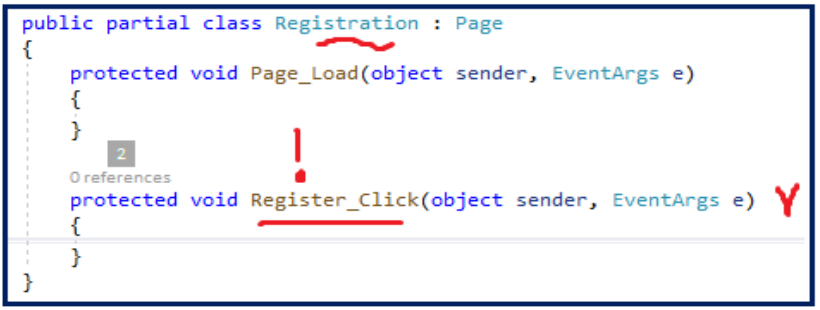
– Web-გვერდზე „რეგისტრაცია“ Button-ის დაკლიკვით უნდა გადავიდეთ C# კოდში... მაგრამ ეს ვერ მოხერხდება, რადგანაც .aspx და .aspx.cs ფაილებს შორის კავშირი არაა გასწორებული...

კერძოდ Registration.aspx.cs გადასვლით (ნახ.4.49-ა,ბ) ჩანს, რომ მე-10 სტრიქონში დარჩენილია საწყისი კლასის სახელი:

_Default : Page.



ნახ.4.49-ა. _Default -ის შეცვლა Registration-ით



ნახ.4.49-ბ. შეცვლის შედეგი

Web-გვერდის ჩატვირთვის და მონაცემების შევსების შემდეგ „რეგისტრაცია“ Button-ის დაჭერისას გამოიძახება OnClick მოვლენაზე მიბმული მეთოდი Register_Click. ის აღიწერება C# კოდში, რომლის 4.9 ლისტინგი მოცემულია ქვემოთ.

გავხსნათ Registration.aspx.cs ფაილი და შევიტანოთ შემდეგი კოდი:

```
// ── ლისტინგი_4.9 ──────────
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
  namespace WebAppRegister5
  {
     public partial class Registration : Page
      {
       protected void Page_Load(object sender, EventArgs e)
        {
        }
       protected void Register_Click(object sender, EventArgs e)
        {
         System.Text.StringBuilder sb = new System.Text .StringBuilder();
              sb.Append("თქვენი გადაცემული მონაცემები:<br>");
              sb.AppendFormat("სახელი: {0}<br>", FirstName.Text);
              sb.AppendFormat("გვარი: {0}<br>", LastName.Text);
              sb.AppendFormat("სქესი: {0}<br>", Sex.SelectedValue);
              sb.AppendFormat("ქალაქი: {0}<br>", City.SelectedValue);
```

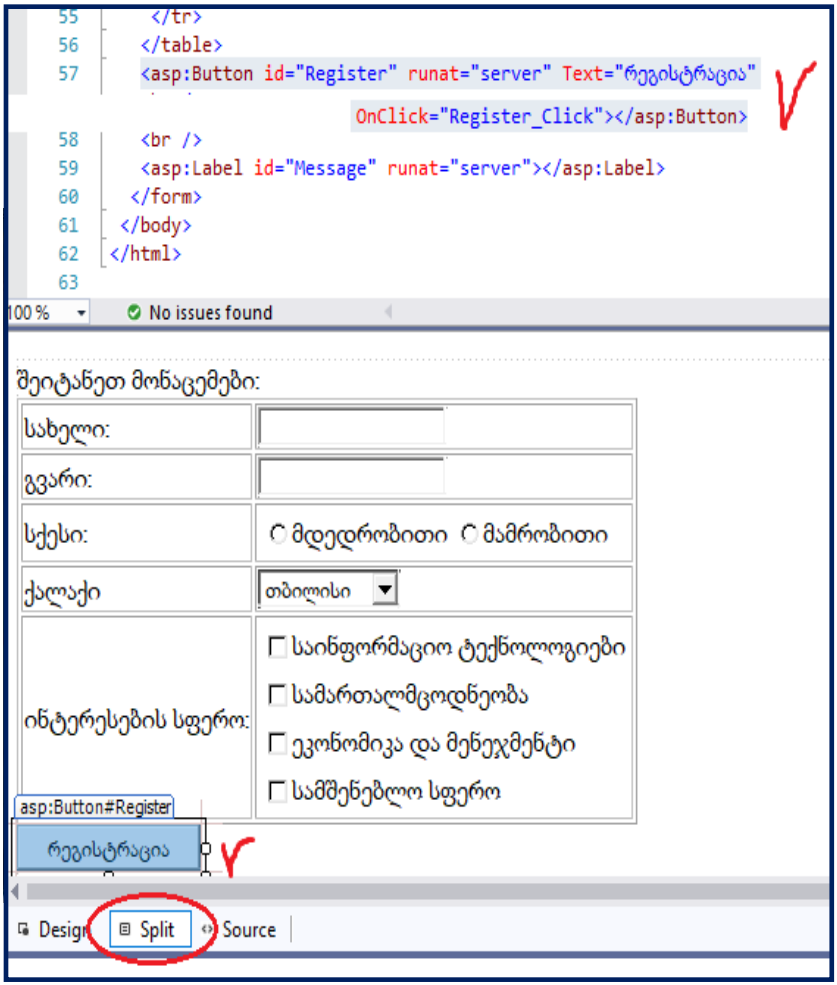
```
            sb.Append("ინტერესები: ");
        foreach (ListItem item in Interests.Items)
            {
               if (item.Selected)
                  sb.AppendFormat("{0}, ", item.Value);
            }
            sb.Append("<br>გმადლობთ რეგისტრაციისთვის");
            Message.Text = sb.ToString();
        }
      }
    }
```

ამის შემდეგ Registration.aspx დიზაინის „რეგისტრაცია“ ღილაკი ამუშავდება. Split რეჟიმში ჩანს ფორმის დიზაინიც და შესაბამისი .aspx პროგრამის ტექსტის ფრაგმენტიც (ნახ.4.50). აქ გამუქებულია <asp:Button id …>.

– „რეგისტრაცია“ ღილაკის ამოქმედებით სისტემა გადაგვიყვანს C# -ის კოდში, Registracion.aspx.cs-ის „Register_Click“ - მოვლენაზე;

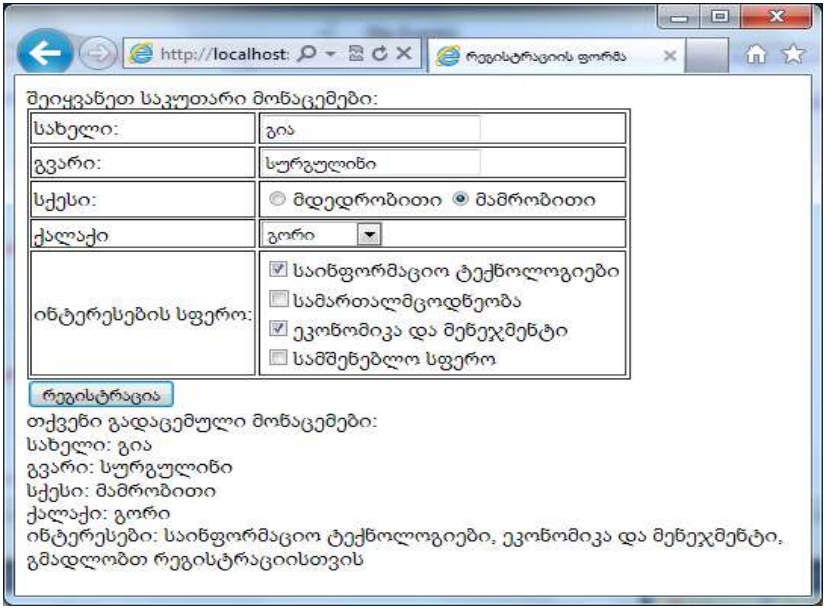
– ავამუშავოთ პროგრამა შესრულებაზე;

– Web-გვერდი, მომხმარებლის მიერ შეტანილი მონაცემებით, „რეგისტრაცია“ ღილაკზე დაკლიკვის შემდეგ, შეასრულებს C# კოდის Register_Click -ში ჩაწერილ ოპერაციებს.



ნახ.4.50

სინტაქსური და სისტემური შეცდომების არარსებობის შემთხვევაში შედეგებს ექნება 4.51 ნახაზზე მოცემული სახე.



ნახ.4.51. სწორი შედეგით დასრულება

4.5.2. სერვერის, კლიენტების, ფორმების და როლების აუთენტიფიკაცია

ვებ-აპლიკაციებში ხშირად საჭიროა მომხმარებელთა იდენტიფიკაცია და მათთვის სხვადასხვა რესურსებზე წვდომის უფლების განსაზღვრა. ASP.NET-ში არსებობს *აუთენტიფიკაციის* (ან ავტორიზაციის) რამდენიმე მეთოდი: Windows, Forms, Passport, Password და სხვ. ეს მოთოდები განისაზღვრება კონფიგურაციის ფაილში authentication ტეგის mode ატრიბუტის საშუალებით [68, 93, 113].

```
<configuration>
    <system.web>
        <authentication />
```

```
    </system.web>
</configuration>
```

მეთოდები:

```
<authentication mode="Windows" />
<authentication mode="Forms" />
<authentication mode="Passport" />
<authentication mode="None" />
```

Web–აპლიკაციაზე მომხმარებლის წვდომის უფლებას განსაზღვრავს authentication ელემენტი, ხოლო აპლიკაციის გარკვეულ ნაწილებზე განისაზღვრება authorization ელემეტით: deny და allow ქვეელემენტების საშუალებით:

***** – *განსაზღვრავს ყველა მომხმარებელს,*

? – *ანონიმურ მომხმარებლებს.*

მაგალითად, ანონიმურ მომხმარებელთათვის საიტზე წვდომის უფლების გასათიშად ვებ-საიტის კონფიგურაციის ფაილში ჩაიწერება:

```
<configuration>
  <system.web>
  <authorization>
     <deny users="?" />
  </authorization>
 </system.web>
</configuration>
```

შესაძებელია ცალკეულ მომხმარებელზე და მომხმარებელთა ჯგუფებზე წვდომის უფლების მინიჭება. შემდეგი ჩანაწერი ნიშნავს, რომ წვდომის უფლება აქვთ someone და Admins ჯგუფში შემავალ მომხმარებლებს:

```
<authorization>
```

```
   <allow users="someone" />
   <allow roles="Admins" />
   <deny users="*" />
</authorization>
```

შესაძლებელია აგრეთვე ცალკეულ ვებ-გვერდებზე წვდომის უფლებების დაყენებაც:

```
<location path="webpage.aspx">
 <authorization>
   <allow roles="managers" />
   <deny users="?" />
 </authorization>
</location>
```

წინა ფრაგმენტი გვიჩვენებს, რომ webpage.aspx წვდომის უფლება აქვთ მხოლოდ managers ჯგუფის მომხმარებლებს.

Forms მეთოდით მომხმარებელთა ავტორიზაციის დროს საჭიროა მომხმარებლის სარეგისტრაციო გვერდის მითითება:

```
<configuration>
  <system.web>
  <authentication mode="Forms">
   <forms name=".ASPXCOOKIE" loginUrl="login.aspx" protection="All"
                    timeout="30" path="/"></forms>
  </authentication>
 </system.web>
</configuration>
```

ვებ საიტზე მომხმარებლების ავტორიზაციის მაგალითი:

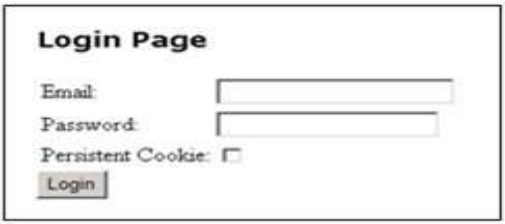
```
// - - - - - ლისტინგი 4.10 - - - Web.config  ფაილი - - - - -
<configuration>
  <system.web>
     <authentication mode="Forms">
```

```
      <forms name=".ASPXUSERDEMO" loginUrl="login.aspx"
protection="All" timeout="60" />
    </authentication>
    <authorization>
      <deny users="?" />
    </authorization>
   <globalization requestEncoding="UTF-8" responseEncoding="UTF-8"
/>
  </system.web>
</configuration>
      // ფაილი Default.aspx:
<%@ Import Namespace="System.Web.Security " %>
<html>
 <script language="C#" runat="server">
   void Page_Load(Object Src, EventArgs E )
    { Welcome.Text = "Hello, " + User.Identity.Name;
    }
   void Signout_Click(Object sender, EventArgs E)
    { FormsAuthentication.SignOut();
      Response.Redirect("login.aspx");
    }
 </script>
 <body>
  <h3><font face="Verdana">Using Cookie Authentication</font></h3>
  <form runat="server" ID="Form1">
   <h3><asp:label id="Welcome" runat="server" /></h3>
   <asp:button text="Signout" OnClick="Signout_Click" runat="server"
ID="Button1" NAME="Button1" />
  </form>
 </body>
```

```
</html>

// ფაილი Login.aspx:
<%@ Import Namespace="System.Web.Security " %>
<html>
 <script language="C#" runat="server">

    void Login_Click(Object sender, EventArgs E) {
//authenticate user: this samples accepts only one user with a
//name of someone@www.contoso.com and a password of 'password'
if((UserEmail.Value=="someone")&&(UserPass.Value=="password"))
 {
  FormsAuthentication.RedirectFromLoginPage(UserEmail.Value,
                    PersistCookie.Checked);
 }
else
 { Msg.Text = "Invalid Credentials: Please try again"; }
 }
 </script>
 <body>
   <form runat="server" ID="Form1">
  <h3><font face="Verdana">Login Page</font></h3>
  <table>
   <tr>
     <td>Email:</td>
     <td><input id="UserEmail" type="text" runat="server"
          NAME="UserEmail" /></td>
     <td
     </td>
   </tr>
```

```
   <tr>
    <td>Password:</td>
    <td><input id="UserPass" type="password" runat="server"
          NAME="UserPass" /></td>
    <td>
    </td>
   </tr>
   <tr>
    <td>Persistent Cookie:</td>
    <td><ASP:CheckBox id="PersistCookie" runat="server" />
    </td>
    <td></td>
   </tr>
  </table>
  <asp:button text="Login" OnClick="Login_Click"
        runat="server" ID="Button1" NAME="Button1" />
  <asp:Label id="Msg" ForeColor="red" Font-Name="Verdana"
        Font-Size="10" runat="server" />
 </form>
</body>
</html>
```

შედეგი ნაჩვენებია 4.52 ნახაზზე



ნახ.4.52

4.6. უნივერსიტეტის IT ინფრასტრუქტურა: მეთოდოლოგია და უსაფრთხოების სტანდარტები

წინამდებარე პარაგრაფში გადმოცემულია საგანმანათლებლო დაწესებულების (მაგალითად, უნივერსიტეტის), როგორც ორგანიზაციული მართვის სისტემის სრულყოფის საკითხები ინფორმაციული უსაფრთხოების საერთაშორისო სტანდარტების (BSI – British Standards Institution. ბრიტანეთის სტანდარტების ინსტიტუტი) გათვალისწინების და ინფორმაციული ტექნოლოგიების ინფრასტრუქტურის ბიბლიოთეკის (ITIL – Information Technology Infrastructure Library) მეთოდოლოგიის საფუძველზე [130, 131].

წარმოდგენი;ლია უნივერსიტეტებში IT-ინფრასტრუქტურისა და მენეჯმენტის პროცესების ანალიზი, კლასიფიცირებულია მათი ძირითადი ფუნქციური ამოცანები [132]. განსაკუთრებით გამახვილებულია ყურადღება პროცესების სწორად დაგეგმვისა და მათი განხორციელების მონიტორინგის ამოცანებზე, რაც მნიშვნელოვანია როგორც ორგანიზაციის IT დეპარტამენტისა და აუდიტისთვის, ასევე მომხმარებლებისთვის (სტუდენტების, აკადემიური პერსონალის და ა.შ.). შედეგად შესაძლებელი ხდება სათანადო სერვისების სწრაფად და სწორად მიწოდება IT-დეპარტამენტის მიერ, კერძოდ, როგორც ინფრასტრუქტურული მოთხოვნების მართვა, ასევე ინციდენტებზე დროული რეაგირება და მისი სწორად წარმართვა [133-136].

ამგვარად, ჩვენი კვლევის მიზანი, მისი ინოვაციურობა და მონოგრაფიის ორიგინალობა მდგომარეობას იმაში, რომ მოხდეს საგანმანათლებლო დაწესებულების პროცესების სრულყოფა მსოფლიოში მიღებული საუკეთესო პრაქტიკების შესაბამისად ციფრული ტექნოლოგიების და ხელოვნური ინტელექტის მეთოდების საფუძველზე. IT-ის მიზანი ბიზნესობიექტის მიზნის მიღწევის მხარდაჭერაა, რათა შეიქმნას ეფექტიანი სასწავლო გარემო [130].

- ***BSI და ინფორმაციული უსაფრთხოება***

BSI (British Standards Institution). ბრიტანეთის სტანდარტების ინსტიტუტი შეიქმნა 1901 წელს როგორც ინჟინრების კომიტეტი სტანდარტების განსაზღვრის საკითხებზე [131]. ამჟამად იგი სტანდარტების საერთაშორისო კომიტეტის (ISO) წევრია. მისი ფუნქციებია [137]:

- მენეჯმენტის სისტემებზე სერვისები და გადაწყვეტები;
- სერვისები შეფასებებზე და სერტიფიკაციაზე;
- პროდუქციის სერტიფიკაცია;
- მენეჯმენტის სისტემების სწავლება;
- სტანდარტები და გამოცემები;

და სხვ.

BSI-სტანდარტი პასუხობს შემდეგ კითხვებს:

- რა არის მართვის საინფორმაციო უსაფრთხოების წარმატების ფაქტორები ?
- როგორ შეიძლება უსაფრთხოების პროცესის მართვა და მონიტორინგი საპასუხისმგებლო მენეჯმენტით ?
- როგორ ხდება უსაფრთხოების მიზნებისა და შესაბამისი უსაფრთხოების სტრატეგიის განვითარება ?
- როგორ შეირჩევა უსაფრთხოების ზომები და როგორ იქმნება უსაფრთხოების კონცეფცია (პოლიტიკა) ?
- როგორ შეიძლება უსაფრთხოების ერთხელ მიღწეული დონის შენარჩუნება და სრულყოფა ?

IT-უსაფრთხოება ახორციელებს, პირველ რიგში, ელექტრონულად შენახული ინფორმაციის დაცვას და ასევე ზრუნავს მონაცემთა დამუშავებაზე. მისი კლასიკური საბაზო ფასეულობებია კონფიდენციალობა, მთლიანობა და წვდომა, აგრეთვე საიმედოობა და შეუფერხებლობა.

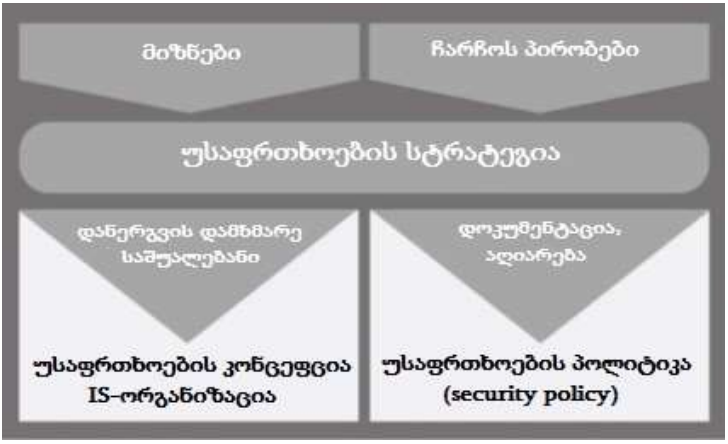
განვიხილოთ რამდენიმე ძირითადი ISO-სტაქნდარტი [131]:

- *ISO 2700x* – ესაა ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემების (ISMS) სტანდარტი. მოიცავს პრინციპებს, კონცეფციებს, ტერმინებს და განსაზღვრებებს (ნახ.4.53);

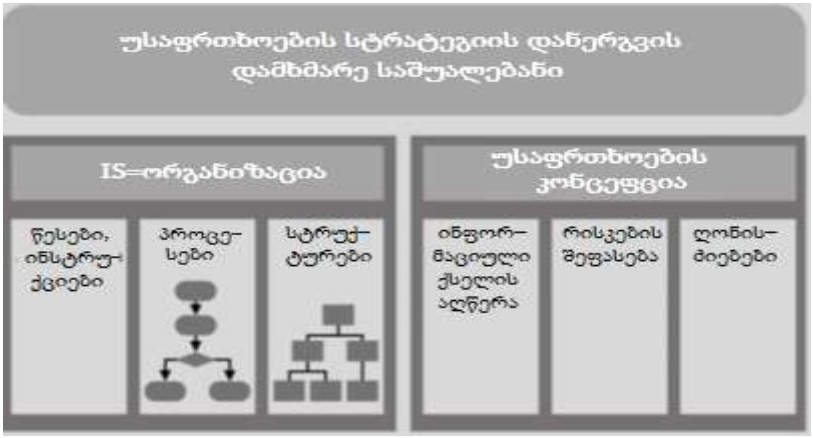
ინფორმაციული უსაფრთხოების ორგანიზაცია და უსაფრთხოების კონცეფცია არის მენეჯმენტის ინსტრუმენტი მისი უსაფრთხოების სტრატეგიის დასანერგად.

ნახ. 4.53. ISMS–ის შედგენილობა

4.54 და 4.55 ნახაზებზე კარგადაა ასახული ეს დამოკიდებულებები [131].

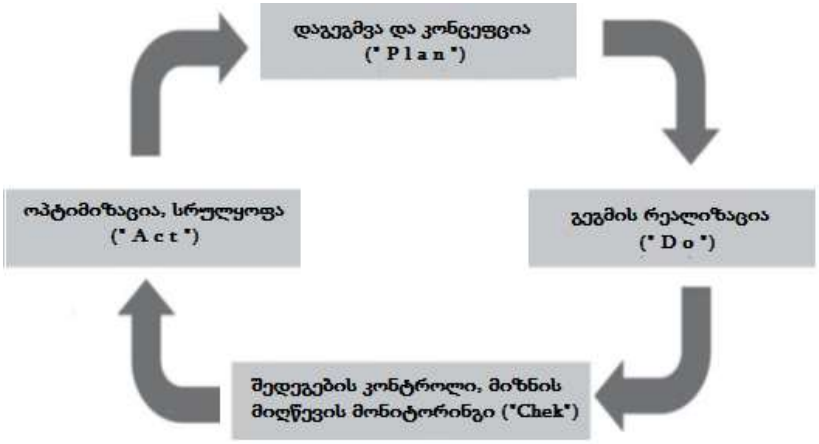


ნახ. 4.54. ინფორმაციული უსაფრთხოების სტრატეგია, ISMS–ის მთავარი კომპონენტი



ნახ.4.55. უსაფრთხოების სტრატეგიის დანერგვა უსაფრთხოების კონცეფციის და ინფორმაციული უსაფრთხოების ორგანიზაციის დახმარებით

უსაფრთხოების პროცესის დინამიკა აღიწერება PDCA მოდელით (4 ფაზით, ნახ. 4.56) და სასიცოცხლო ციკლით (ნახ.4.57).



ნახ.4.56. სასიცოცხლო ციკლი Deming-ის მიხედვით (PDCA-მოდელი)

ნახ.4.57. უსაფრთხოების კონცეფციის სასიცოცხლო ციკლის მიმოხილვა

რისკების შეფასებისას უნდა იყოს იდენტიფიცირებული: დასაცავი ინფორმაცია, ბიზნესპროცესები და მათი შესაბამისი ყველა საფრთხე; სუსტი ადგილები, რომლებშიც შეუძლია ზემოქმედება საფრთხეებს; შესაძლო დაზიანებები, გამოწვეული კონფიდენციალობის, მთლიანობისა და წვდომის დაკარგვის გამო.

ამასთანავე უნდა იქნას გაანალიზებული სავარაუდო ზეგავლენები ბიზნესზე ან შესასრულებელ ამოცანებზე, გამოწვეული უსაფრთხოების ინციდენტებით და უნდა შეფასდეს რისკი, რომელიც ამ ინციდენტებით იწვევს დაზიანებას.

- *ISO 27001* – ინფორმაციული უსაფრთხოების მართვის სისტემის მოთხოვნების სპეციფიკაცია. იგი იძლევა 10-გვერდიან ზოგად რეკომენდაციებს, მათ შორის შესავლის (დანერგვის), ექსპლუატაციის და დოკუმენტირებული ინფორმაციის უსაფრთხოების მართვის სისტემის სრულყოფისთვის, რისკების გათვალისწინებით;
- *ISO 27002 არის* ინფორმაციული უსაფრთხოების მენეჯმენტის საპროცესო კოდექსი, რომლის მიზანია ინფორმაციული უსაფრთხოების მენეჯმენტის ჩარჩოს განსაზღვრა. გადმოცემულია აუცილებელი ბიჯები (ეტაპები) ფუნქციონირებადი უსაფრთხოების მენეჯმენტის ასაგებად და ორგანიზაციაში მის დანერგვას. იგი აკმაყოფილებს ISO სტანდარტის 27001-ის მოთხოვნებს;
- *ISO 27005* სტანდარტი ეხება ინფორმაციული უსაფრთხოების რისკების მენეჯმენტს. იგი შეიცავს ძირითად რეკომენდაციებს რისკების მართვის შესახებ ინფორმაციული უსაფრთხოებისთვის;
- *ISO 27006* სტანდარტით განისაზღვრება მოთხოვნები სერტიფიცირების აკრედიტაციული ორგანოების მიმართ ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემებში", განსაზღვრავს აკრედიტაციის მოთხოვნებს სერტიფიცირების ორგანოებისთვის ISMS-ში და განიხილება ამ სერტიფიცირების პროცესების თავისებურებანი.

არსებობს ასევე ISO 2700x სხვა სტანდარტებიც, რომელთა დეტალური გაცნობა შესაძლებელია ინტერნეტულ წყაროებში [137].

➢ *სხვა სტანდარტები – COBIT და ITIL*

- *COBIT* (Control Objectives for Information and related Technology – კონტროლის მიზნები ინფორმაციული და დაკავშირებული ტექნოლოგიებისთვის) აღწერს რისკების კონტროლის მეთოდს, რომელიც ხორციელდება IT–დანერგვის საშუალებით კრიტიკული ბიზნესპროცესების შესრულების მხარდასაჭერად [131, 138]. COBIT–დოკუმენტები გაიცემა საინფორმაციო სისტემების აუდიტის და

კონტროლის ასოციაციის (ISACA– Information Systems Audit and Control Association) IT მართვის ინსტიტუტის (ITGI – IT Governance Institute) მიერ. COBIT–ის დამუშავების დროს ავტორები ორიენტირებულნი იყვნენ უსაფრთხოების მენეჯმენტის არსებულ სტანდარტებზე, როგორიცაა ISO 27002.

- *ITIL* (IT Infrastructure Library – IT ინფრასტრუქტურის ბიბლიოთეკა) არის IT სერვისმენეჯმენტის რამდენიმე წიგნის კოლექცია [131, 139]. იგი შემუშავებულ იქნა გაერთიანებული სამეფოს სახელმწიფო კომერციის მთავრობის მიერ (OGC). ITIL განიხილავს IT–სერვისების მენეჯმენტს IT–მომსახურების თალსაზრისით. IT–მომსახურება შეიძლება იყოს როგორც შიგა IT–დეპარტამენტის ან გარე სერვისის პროვაიდერის. საერთო მიზანია IT მომსახურების და ხარჯების ეფექტიანობის ოპტიმიზაცია და ხარისხის სრულყოფა.

დიდი ბრიტანეთის გაერთიანებული სამეფოს მთავრობის ცენტრალურმა კომპიუტერულმა და ტელეკომუნიკაციების სააგენტომ (CCTA – Central Computer and Telecommunications Agency) 1980-იან წლებში შეიმუშავა რეკომენდაციების ნაკრები IT მენეჯმენტის პრაქტიკის სტანდარტიზებისთვის სამთავრობო ფუნქციებში. იგი სრულყოფილი v3 ვერსიის სახით იქნა წარმოდგენილი 21-საუკუნის დასაწყისში, ხოლო 20-ანი წლებისთვის გამოვიდა მისი ახალი, გაფართოებული v4 ვერსია [140 - 143].

კვლევის პროცესში გამოყენებულია შემდეგი მეთოდები:

1) *ფოკუს ჯგუფები,* სადაც დისკუსიის შედეგად, გამოჩნდა საჭიროება და გამოკვლეულ იქნა მიზეზ-შედეგობრივი კავშირები;

2) *ინტერვიუ,* სადაც გამოყენებულ იქნა წინასწარ შერჩეული კითხვები რომლებზე პასუხებიც მკაფიოდ ასახავს პრობლემის მნიშვნელობას.

კვლევის გაანალიზებისას გამოიკვეთა. რომ საგანმანათლებლო დაწესებულებაში, კერძოდ უნივერსიტეტში, IT ინფრასტრუქტურის

სტრუქტურირებული მართვა მნიშვნელოვანია უნივერსიტეტის სრულფასოვანი მუშაობისათვის. ხშირად არის შემთხვევები როდესაც ინფრასტრუქტურასთან დაკავშირებული პრობლემები ან მოთხოვნები და მათზე დაგვიანებული რეაგირება აფერხებს სასწავლო თუ სხვა პროცესებს.

მოთხოვნებისა და ინციდენტების დასაფიქსირებლად არ არსებობს არხი, რომლითაც შეეძლებათ მომხმარებლებს სარგებლობაც, არ ხდება საკითხების შენახვა და რეპორტინგის წარმოება, რაც დაგვეხმარება სამომავლოდ ინციდენტების პრევენციაში.

მაგალითად, IT-თანამშრომლებს, რომლებმაც რეაგირება უნდა მოახდინონ პრობლემურ საკითხებზე, არ აქვთ თავმოყრილი ინციდენტებისა და მოთხოვნების ნუსხა, რომელიც შეიძლება პრიორიტეტის მიხედვით შეასრულონ. ასევე არ აქვთ ინფორმაცია საკითხზე რომელიც დააფიქსირეს, მათი სავარაუდო მოგვარების შესახებ ინფორმაცია და სხვ.

IT-ინფრასტრუქტურასთან დაკავშირებული საკითხების შესახებ ინფორმაცია უნდა ინახებოდეს ჩანაწერების შესაბამის ფაილში. იდეალურ შემთხვევაში, ამ ინსტრუმენტმა ასევე უნდა უზრუნველყოს ბმულები შესაბამის მოთხოვნებთან, ინციდენტებთან და სხვ., რათა მოხდეს სწრაფი და ეფექტური დიაგნოსტიკა და აღდგენა. თანამედროვე IT სერვისების მართვის ინსტრუმენტებს შეუძლია უზრუნველყოს ინციდენტების ავტომატური დაკავშირება სხვა ინციდენტებსა და პრობლემებზე, რათა გაანალიზოს მათი პრევენციის საკითხი.

IT Service Management (ITSM) ინსტრუმენტები საშუალებას აძლევს უნივერსიტეტის IT-ოპერაციულ სამსახურებს, კერძოდ, ინფრასტრუქტურის და ოპერაციების (I&O) მენეჯერებს, მოახდინონ სისტემის მხარდაჭერა.

- *JSM (Jira Service Management*) და *Insight (Asset Management)* ერთად, ერთ-ერთი საუკეთესო ტექნოლოგიაა IT ინფრასტრუქტურის მართვის სერვის დესკის პორტალის რეალიზების ამოცანის გადასაჭრელად, იგი Atlassian-კორპორაციის პროდუქტია [133-135].

Atlassian Corporation ავსტრალიური პროგრამული კომპანიაა, რომელიც ავითარებს პროდუქტებს პროგრამული უზრუნველყოფის შემქმნელებისთვის, პროექტის მენეჯერებისთვის და სოფთის დეველოპერების გუნდებისათვის (კომპანია მდებარეობს დელავერში, გლობალური სათაო ოფისი სიდნეისა და აშშ-ს შტაბბინა სან-ფრანცისკოში).

Insight საშუალებას გვაძლევს აღვწეროთ სრულად ტექნიკური სერვისების კატალოგი და მოვახდინოთ მისი დაკავშირება JSM-თან, ვაწარმოოთ რეპორტინგი და სხვა მნიშვნელოვანი სტანდარტული საკითხების ავტომატიზაცია.

JSM შესაძლებლობას იძლევა: სამუშაო ნაკადების (workflows) მარშრუტიზაციის, რესურსების დაგეგმვის სისტემების და ცოდნის ბაზის ინტეგრაციისათვის; მონიტორინგის დაფებისა (Dashboards) და რეპორტინგისათვის და სხვ.

ამგვარად, IT-ინფრასტრუქტურის მართვის სრულყოფა საგანმანთლებო დაწესებულებაში უმნიშვნელოვანეს როლს თამაშობს. განსაზღვრული პროცესები, რომლის მიხედვითაც მოხდება ინფრასტრუქტურის მართვა გააუმჯობესებს აკადემიური პერსონალისა და სტუდენტების კმაყოფილებას. შეფერხების გარეშე წარიმართება სწავლა/სწავლების პროცესები.

კვლევის კითხვებზე პასუხები გვიჩვენებს, რომ მნიშვნელოვანია პროცესების პროგრამული რობოტიზაცია, რაც შედეგად მოგვცემს განმეორებადი პროცედურების ავტომატიზაციას და მოხდება ინციდენტების პრევენცია.

და ბოლოს, მნიშვნელოვანია ამ მიმართულებით ხელოვნური ინტელექტის და მანქანური დასწავლის მეთოდების გამოყენება.

თავი 5
მანქანური დასწავლის მეთოდების დანერგვის ამოცანა სასწავლო პროცესში

მანქანური დასწავლა (Machine Learning – ML) არის ხელოვნური ინტელექტის (Artificial intelligence – AI) ერთ-ერთი მევნიერულ-პრაქტიკული მიმართულება. მათ შორის არსებობს გარკვეული განსხვავება [115]. კონტექსტურად, ხელოვნური ინტელექტი გულისხმობს კომპიუტერულ ზოგად უნარს, მიბაძოს ადამიანის აზროვნებას და შეასრულოს ამოცანები რეალურ გარემოში. ხოლო მანქანური დასწავლა ეხება ტექნოლოგიებსა და ალგორითმებს, რომლებიც საშუალებას აძლევს სისტემებს ამოიცნოს არსებული კანონზომიერებანი, მიიღოს გადაწყვეტილებები და სრულყოს საკუთარი თავი გამოცდილებისა და მონაცემების საფუძველზე.

პროგრამისტ-დეველოპერები კომპიუტერებს საშუალებას აძლევენ გააანალიზონ მონაცემები და გადაჭრან დასმული ამოცანები. ისინი ქმნიან ხელოვნური ინტელექტის სისტემებს - ისეთი ინსტრუმენტების გამოყენებით, როგორიცაა:

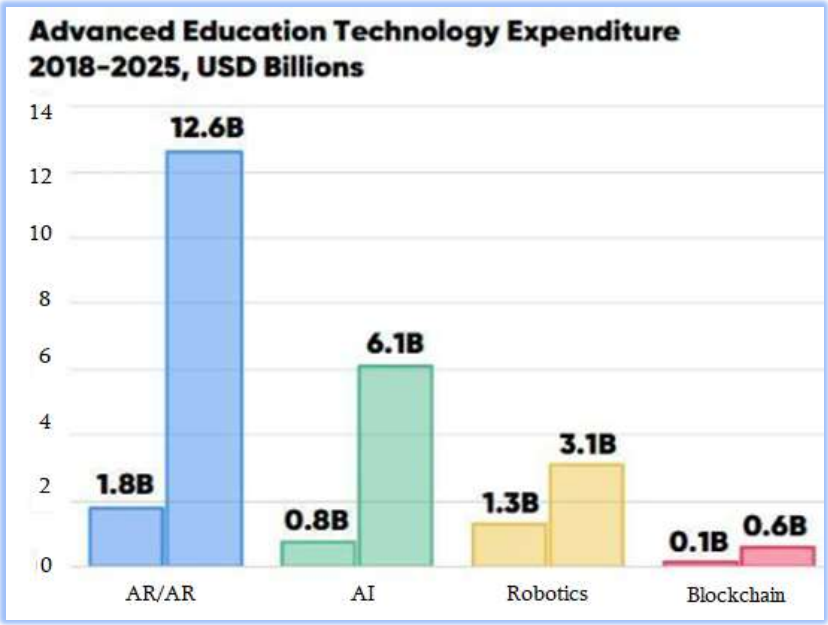
- მანქანური დასწავლა;
- ღრმა სწავლება;
- ნეირონული ქსელები;
- კომპიუტერული ხედვა;
- ექსპერტული სისტემები'
- ბუნებრივი ენის დამუშავება.

XXI საუკუნის დასაწყისიდან საგრძნობლად გაფართოვდა ხელოვნური ინტელექტისა და მანქანური დასწავლის სისტემების გამოყენება ინდუსტრიის თითქმის ყველა სფეროში, სოფლის მეურნეობის, ჯანმრთელობის დაცვისა და, განსაკუთრებით, საგანმანათლებლო დაწესებულებებში [116-121].

5.1. ხელოვნური ინტელექტის (AI) მნიშვნელობა თანამედროვე საგანმანათლებლო სფეროში

MarketWatch–ის ანგარიშმა აჩვენა, რომ განათლებაში მანქანური დასწავლა ინვესტიციების მოზიდვის ერთ – ერთ წამყვან ინდუსტრიად რჩება [122], 2030 წლისთვის მთავარ მოთამაშედ აშშ და ჩინეთი იქნებიან. მსხვილი კომპანიები, როგორიცაა Google და IBM, მონაწილეობენ საგანმანათლებლო დაწესებულებების შექმნაში. განათლება უფრო პროგრესული და ინოვაციურია.

EdWeek ასევე იუწყება, რომ AI განათლება ინვესტორების-თვის მთავარ საყურადღებო ობიექტად დარჩება, მეორე ადგილზე, Augmented Reality (AR) და Virtual Reality (VR) ტექნოლოგიების შემდეგ [114].



ნახ. 5.1. მოწინავე განათლების ტექნოლოგიის დანახარჯები

5.2. მანქანური დასწავლის (ML) მოდელებისა და მეთოდების გამოყენების საერთაშორისო გამოცდილება

ხელოვნური ინტელექტი და მანქანური დასწავლა სასწაულებს ახდენს სასკოლო განათლებაში. მაგალითად, Wall Street Journal-მა (WSJ) თავის ერთ-ერთ ბოლო მოხსენებაში ხაზგასმით აღნიშნა, თუ რა გავლენას ახდენს AI და ML ჩინეთის საგანმანათლებლო სფეროში.

ჩინეთის მიღწევები ამ სფეროში წარმოუდგენელლად დიდა [123]. მაგალითად, კლასებში არის რობოტები, რომლებიც აკვირდებიან მოსწავლის ჯანმრთელობას, მოსწავლეები ატარებენ ტრეკერებს, რათა ისინი უსაფრთხოდ იყვნენ სკოლის ტერიტორიაზე ყოფნის დროს, ყველაზე მნიშვნელოვანი მიღწევაა თავსაბურავები, რომლებიც განსაზღვრავს სტუდენტის კონცენტრაციის დონეს.

WSJ– ს მიერ გამოკითხული პედაგოგები ერთხმად უჭერენ მხარს ამ ინოვაციებს და აცხადებენ, რომ სკოლაში AI–ს შემოტანა მოსწავლეებს უფრო გულმოდგინედ აქცევს და აუმჯობესებს მათ აკადემიურ მოსწრებას. მანქანური დასწავლის მეთოდების გამოყენების დონეც, როგორც ხელოვნური ინტელექტის აქტუალური ნაწილისა, ბოლო რამდენიმე წლის განმავლობაში სწრაფად გაიზარდა.

2019 წელს მკვლევარებმა წამოიწყეს მანქანური დასწავლის საფუძველზე შექმნილი ალგორითმის შექმნის ტენდენცია, რომელიც სტუდენტებს დაეხმარება განსაზღვრონ მათი მომავალი კარიერული გზა და კოლეჯის პარამეტრები.

კალიფორნიის ლიბერალური ხელოვნების კოლეჯმა Occidental College-მა დაიწყო კვლევა იმ მოდელის შესაქმნელად,

რომელიც ითვალისწინებს თითოეული სტუდენტის ინტერესს კოლეჯის გადაწყვეტილებაში. მათი ინტერესებისა და შედეგებიდან გამომდინარე, სისტემა ეხმარება ამ ფაქტორების შესაბამისი კოლეჯის შერჩევაში [124].

საშუალო სკოლის მოსწავლეებს ხშირად აქვთ ზეწოლა, რომ აირჩიონ სწორი კოლეჯი და შემდეგ სპეციალობა. „ჩვენმა ბოლოდროინდელმა კვლევამ აჩვენა, რომ აშშ-ს სტუდენტთა მხოლოდ 10-15%-მა იცის, რომელი გზა უნდა გაიაროს სკოლის დამთავრების შემდეგ", - ამბობს ნ. უაიტი, SupremeDissertations და IsAccurate ერთობლივ პროექტზე მომუშავე მკვლევარი.

მანქანური დასწავლის პროგნოზირების ამ მოდელის საბოლოო მიზანია დაეხმაროს სტუდენტებს ობიექტურად გააანალიზონ თავიანთი უნარები, შესაძლებლობები და ინტერესები. მათ საფუძველზე, აირჩიონ სწორი გზა სწავლის დასრულების შემდეგ.

განათლების სფეროს პროფესიონალები მიუთითებენ სკოლებში შეფასების მიკერძოებულ სისტემაზე. რადგან არ არსებობს შეფასების ერთიანი სისტემა, მასწავლებლები ზოგადად იყენებენ შეფასების საკუთარ გზას, რომელიც ხშირად მიკერძოებულია.

ამიტომ განათლების პროფესიონალები ელიან, რომ AI და ML მნიშვნელოვან წვლილს შეიტანს უფრო ზუსტი შეფასების სისტემის შექმნაში. ჩინეთში, დაახლოებით 60,000 სკოლა ტესტირებას უტარებს Paper-grading სისტემას, რომელიც ავტომატურად აფასებს სტუდენტების ესეებს.

Paper-grading სისტემის უპირატესობა არის ის, რომ იგი ზუსტად აფასებს ესეს სტილს, სტრუქტურას და თხრობას სხვა ფაქტორების გათვალისწინების გარეშე, რომლებსაც მასწავლებლები ჩვეულებრივ ითვალისწინებენ, მაგალითად, აკადემიური მიღწევები ან გაკვეთლებზე დასწრება [18, 114].

5.3. Python ენის გამოყენება AI და ML-სთვის

ხელოვნური ინტელექტი (AI) და მანქანური დასწავლა (ML) აქტიურად გამოიყენება დღეს და ასევე აქვს დიდი პერსპექტივა ახლო მომავალშიც სხვადასხვა ინდუსტრიის მრავალ ობიექტზე გამოყენებისათვის. სწორედ ამიტომ, მსხვილი კორპორაციები ინვესტიციებს ახორციელებენ ამ სფეროებში და შესაბამისად იზრდება მოთხოვნა ისეთი პროფილის ექსპერტებზე, როგორიცაა ბიზნეს-ანალიტიკოსი, სისტემის არქიტექტორო, დესკტოპ-, ვებ- და მობილური პროგრამისტ-დეველოპერები, ტესტერები, სისტემის უსაფრთხოების მენეჯერები და სხვ.

ამ თვალსაზრისით, განსაკუთრებით მნიშვნელოვანია ინფორმატიკის დიდაქტიკის როლი, რომელიც შეძლებს თანამედროვე ციფრული ტექნოლოგიების ბაზაზე მოამზადოს როგორც Backend-სპეციალისტები, ისე Frontend-მომხმარებლები. მსოფლიოს ყველა ქვეყნის უნივერსიტეტები დიდი გამოწვევის წინაშე დგას და ესაა სწორედ „ინფორმაციული საზოგადოების“ ფორმირება !

პროგრამული ინჟინერია, თავისი თეორიული და პრაქტიკული მნიშვნელობის გადასაწყვეტი ამოცანებით ახალ ორბიტაზე გავიდა, სადაც ხელოვნური ინტელექტის სისტემების შექმნა და მათი ინტენსიური გამოყენება დომინირებადი და შეუქცევადი პროცესი გახდა. განსაკუთრებული როლი ამ პროცესებში მულტიპარადიგმული ტიპის კლასიკურმა და ხელოვნურ ინტელექტზე მორგებულმა დაპროგრამების ენებმა შეასრულეს და კვლავაც აგრძელებენ განვითარებას.

IBM-ის თანამშრომელი ჟან-ფრანსუა პუჟემ, რომელიც ხელოვნური ინტელექტის და მანქანური დასწავლის სფეროში მოღვაწეობს. გამოთქვა აზრი, რომ პროგრამირების ენა Python არის ყველაზე პოპულარული ამ სფეროში [125-128]. ასეთი დასკვნა ემყარება დასაქმების საიტის Indeed.com ანალიზის შედეგებს [129].

Python, როგორც საუკეთესო პროგრამირების ენა AI და ML-სთვის – არის შემდეგი ძირითადი მახასიათებლების გამო [144]:

1) ვრცელი ბიბლიოთეკები;
2) დამწყებთათვის მეგობრული;
3) მოქნილი, გაფართოებადი და მასშტაბირებადი;
4) პლატფორმის დამოუკიდებლობა;
5) ვიზუალიზაციის კარგი საშუალება;
6) საზოგადოების მხარდაჭერა;
7) IoT შესაძლებლობები;
8) ჩაშენებადი;
9) მზარდი პოპულარობა.

Python ენას აქვს თავისი ნაკლოვანებებიც, რომელთაგან შეიძლება გამოვყოთ: დიზაინის პრობლემები, ნელი სწრაფქმედება (ინტერპრეტატორია), მეხსიერების მაღალი მოხმარება, რთულია მრავალნაკადურობა, დაბალი უსაფრთხოება და სხვ. მაგრამ ეს საკითხები სულაც არ უშლის ხელს ამ ენას, რომ იყოს პოპულარული.

ინტერნეტში მრავლად მოიპოვება Python ენაზე შექმნილი აპლიკაციების აღწერის სამეცნიერო წიგნები და სტატიები სხვადასხვა სფეროში, განსაკუთრებით ხელოვნური ინტელექტის და მანქანური დასწავლის მიმართულებით.

მაგალითად, ML-ის რეგრესიის, კლასიფიკაციის, კლასტერიზაციის ალგორითმების რეალიზაციისათვის გამოიყენება Python ენაში scikit-Learn ბიბლიოთეკა. SVM (Support Vector Machine) ალგორითმისათვისაც ხელმისაწვდომია იგი [146].

მომდევნო პარაგრაფებში განვიხილავთ ჩვენ მიერ აგებული მოდელების, მეთოდების და აპლიკაციების შედეგებს, რომლებიც, ამჯერად, ორიენტირებულია საგანამანათლებლო სფეროში (სკოლა, უნივერსიტეტი) გამოსაყენებლად.

5.4. ML-ის გამოყენება გამოცდების შედეგების პროგნოზირებისთვის

სასწავლო დაწესებულებაში ერთ-ერთი ყველაზე მნიშვნელოვანი პროცესი მოსწავლეთა გამოცდების შედეგების განხილვის და დამუშავების ნაწილია, რომელიც ზედმეტად შრომატევადია შესაბამისი პასუხისმგებელი პირისთვის.

გამოვყავით ორი საკითხი:

პირველი – კონკრეტულ საგანში ჩაბარებული გამოცდების შედეგების მიხედვით *მომავალი გამოცდის შედეგის პროგნოზირება,* რათა მასწავლებლებმა შეძლონ დროული რეაგირება;

მეორე საკითხია ის, რომ აღმოვაჩინოთ გადაწერის შესაძლო მცდელობები.

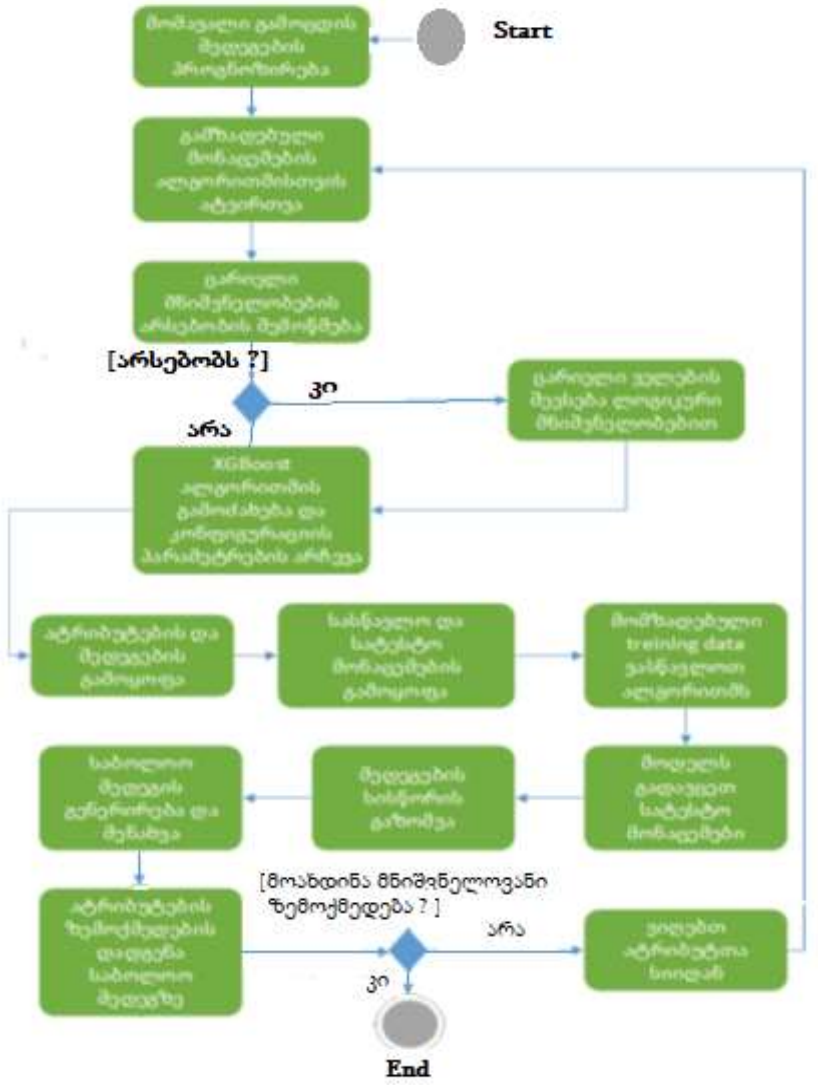
ჩვენი კონცეფციაა ამ მიზნისთვის გამოვიყენოთ *მანქანური დასწავლის ალგორითმი,* რათა იგი დაგვეხმაროს კონკრეტულ საგანში მოსწავლეთა მომავალი სავარაუდო შედეგების დადგენაში (პროგნოზირებაში).

თუ გვეცოდინება კონკრეტულ საგანში, მაგალითად, ქართულში, თითოეული მოსწავლის წინა ორ ან მეტ გამოცდაში მიღებული შედეგები და ასევე მოსწავლის შესახებ სხვდასხვა მახასიათებლები, მაშინ მანქანური დასწავლის ალგორითმის დახმარებით შევძლებთ დასაშვები (ჩვენთვის მისაღები) სიზუსტით დავადგინოთ *მომდევნო* გამოცდის სავარაუდო შედეგები.

ეს წარმოადგენს რეგრესიის ამოცანას.

შემდეგ პარაგრაფში განვიხილავთ აღნიშნული ამოცანის დასმის და გადაწყვეტის საკითხებს.

5.2 ნახაზზე მოცემულია მომავალი გამოცდის შედეგების პროგნოზირების პროცესის UML-აქტიურობის დიაგრამა.



ნახ. 5.2. მომავალი გამოცდის შედეგების პროგნოზირების Activity დიაგრამა

5.5. რეგრესიის ამოცანის დასმა და ამოხსნა

რეგრესიული ანალიზი მოიცავს მანქანური სწავლების მეთოდების ერთობლიობას, რომლის საშუალებითაც შესაძლებელია განვსაზღვროთ უწყვეტი შედეგის ცვლადი (y) ერთი ან მეტი პროგნოზირების ცვლადის (x) მნიშვნელობის საფუძველზე [19].

ამგვარად, რეგრესიული მოდელის მიზანია მათემატიკური „განტოლების" შექმნა, რომელიც განსაზღვრავს y-ს როგორც x ცვლადის ფუნქციის მნიშვნელობას. შემდეგ ეს განტოლება შეიძლება გამოყენებულ იქნას y-ის მნიშვნელობის პროგნოზირებისთვის x ცვლადის ახალი მნიშვნელობებით.

5.5.1 მონაცემები

მაგალითად, მოსწავლეზე გვაქვს შემდეგი სახის სავარაუდო ინფორმაციული მონაცემები:

• გვარი სახელი • სქესი • ასაკი • ოჯახის წევრების რაოდენობა • მშობლების ერთად ყოფნის სტატუსი • დედის განათლების დონე • მამის განათლების დონე • დედის მუშაობის სტატუსი • მამის მუშაობის სტატუსი • სკოლამდე მისასვლელი დრო • კვირის განმავლობაში მეცადინეობის ხანგრძლივობა • დამატებითი საგანმანათლებლო დახმარება • ოჯახის წევრები ეხმარებიან თუ არა მეცადინეობისას	• დამატებითი ფასიანი კლასები • კლასგარეშე • საბავშვო ბაღში დადიოდა თუ არა • უმაღლესი განათლების მიღების სურვილი • ინტერნეტთან წვდომის შესაძლებლობა • შეყვარებული • ოჯახური მდგომარება • ჯანმრთელობის მდგომარეობა • გაცდენების რაოდენობა • ქულა 1 • ქულა 2 • მოსალოდნელი ქულა

ჩვენი რეგრესიის ამოცანის ამოხსნის გზა:

პირველი ეტაპზე საჭიროა აიტვირთოს ალგორითმისათვის გამზადებული მონაცემები (ცხრ. 5.1).

მოსწავლის შესახებ მონაცემები **ცხრ.5.1**

	gender	age	moreThen3Memb	apart	momEdu	fatherEdu	momWork	fatherWork	timeMoreThan30	studyTimeMoreThan5	...	nursery	higherEdu	inte
0	0	18	1	1	1	1	0	1	0	0	...	1	1	
1	0	17	1	0	0	0	0	1	0	0	...	0	1	
2	0	15	0	0	0	0	0	1	0	0	...	1	1	
3	0	15	1	0	1	0	1	1	0	1	...	1	1	
4	0	16	1	0	0	0	1	1	0	0	...	1	1	

5 rows × 24 columns

შემდეგ ეტაპზე მოწმდება გადაცემულ მონაცემებში ცარიელი მნიშვნელობების არსებობა:

```
In [6]: #We dont have missing values
        Data.info()
```

ცარიელი მნიშვნელობების შემოწმების შედეგები ჩანს 5.2 ცხრილში.

აღნიშნულ მონაცემებში, სიმარტივის მიზნით, ცარიელი ველები არ გვაქვს, თუმცა არსებობის შემთხვევაში საჭიროა სიცარიელე შეივსოს ლოგიკური მნიშვნელობით (მაგალითად 0-ით, ან მსგავსი მოსწავლეების საშუალოთი და ა.შ). ეს დამოკიდებულია ველის შინაარსზე.

მონაცემებში ცარიელი მნიშვნელობის ძებნის შედეგი ცხრ.5.2

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   gender                649 non-null    int64
 1   age                   649 non-null    int64
 2   moreThen3Memb         649 non-null    int64
 3   apart                 649 non-null    int64
 4   momEdu                649 non-null    int64
 5   fatherEdu             649 non-null    int64
 6   momWork               649 non-null    int64
 7   fatherWork            649 non-null    int64
 8   timeMoreThan30        649 non-null    int64
 9   studyTimeMoreThan5    649 non-null    int64
 10  extraEduSupport       649 non-null    int64
 11  parentHelps           649 non-null    int64
 12  extraPaiedClasses     649 non-null    int64
 13  activities            649 non-null    int64
 14  nursery               649 non-null    int64
 15  higherEdu             649 non-null    int64
 16  internet              649 non-null    int64
 17  InARelationship       649 non-null    int64
 18  familyRelationships   649 non-null    int64
 19  health                649 non-null    int64
 20  absences              649 non-null    int64
 21  score1                649 non-null    int64
 22  score2                649 non-null    int64
 23  score                 649 non-null    int64
dtypes: int64(24)
memory usage: 121.8 KB
```

5.5.2. XGBoost ალგორითმი

გამოვიყენებთ პროგრამირების Python ენას და ამ ამოცანის გადასაჭრელად - XGBoost ალგორითმს [7, 86]. ეს არის ერთ-ერთი ეფექტური და მარტივი ალგორითმი, რომელიც აგებს გადაწყვეტილებათა მიმღებ ხეს (რაოდენობა კონტროლდება პარამეტრით, რომელსაც ჩვენ ვუთითებთ) და ყველა მათგანის პასუხების გათვალისწინებით გამოყავს საბოლოო პროგნოზი.

არსებობს მრავალი ტიპის ალგორითმი და რეალურ აპლიკაციაზე მუშაობის დროს საჭირო იქნება სხვადასხვა მათგანის გატესტვა, თუმცა ამჟამად შედეგებს წარმოვადგენთ XGBRegressor-ის გამოყენებით.

ეს ალგორითმი გადაწყვეტილებას იღებს შექმნილ ხეზე (ხეებზე) დაყრდნობით, რომელიც იყენებს გრადიენტული გაძლიერების გარემოს [7].

Boosting-ის დროს ახალი მოდელების დამატება ხდება, რათა გასწორდეს წინა მოდელების მიერ დაშვებული უზუსტობები. მოდელები ემატება მანამ, სანამ შეცდომების რაოდენობა შესამჩნევად არ იკლებს. პოპულარული მაგალითია AdaBoost ალგორითმი, რომელიც წონის მონაცემთა წერტილებს, რომელთა პროგნოზირებაც რთულია [145].

ეს მიდგომა უზრუნველყოფს როგორც რეგრესიის, ისე კლასიფიკაციის ამოცანების გადაწყვეტას. XGBoost ალგორითმი განისაზღვრება შემდეგნაირად:

```
Model = XGBRegressor(objective = 'reg:linear', max_depth =6,
n_estimators=61, min_child_weight=1, colsample_bytree=0.99,
subsample=0.85, learning_rate=0.1, early_stopping_rounds=10)
```

შესაბამის ალგორითმს შესაძლებელია გადავცეთ კონფიგურაციის პარამეტრები (Hyperparameter), ან გამოვიყენოთ მისი default კონფიგურაცია. ასეთი ძირითადი პარამეტრებია: n_estimators – ალგორითმის მუშაობის პროცესში შედგენილი ხეების რაოდენობა, learning_rate – სწავლის სიზუსტე და ა.შ.

პარამეტრების მნიშვნელობის მიხედვით განსხვავდება ალგორითმის მიერ დაბრუნებული შედეგები. შესაბამისად, რეალურ აპლიკაციაზე და რეალურ მონაცემებზე მუშაობის დროს საჭირო იქნება პარამეტრების ოპტიმალური მნიშვნელობების დადგენა, რისთვისაც არსებობს დამხმარე ბიბლიოთეკები. ისინი გვეხმარება პარამეტრებისთვის ისეთი მნიშვნელობების პოვნაში, რომლის დროსაც ალგორითმის საერთო უზუსტობა, ანუ შეცდომების რაოდენობა იქნება მცირე.

ძირითადი პარამეტრების მნიშვნელობები შესაძლოა შერჩეულ იქნეს ჩვენს მიერ. სხვადასხვა მნიშვნელობების ტესტირების შედეგებს ვნახულობთ და ვადგენთ სასურველ რიცხვს, სადაც ალგორითმის სიზუსტე და მუშაობის დრო მისაღებია.

მაგალითად, რაც მეტ გადაწყვეტილების მიმღებ ხეს ააგებს ჩვენი მოდელი, მით მეტი დრო დასჭირდება საბოლოო შედეგების მიღებას. დრო ერთ-ერთი მნიშვნელოვანი რესურსია, თუ მოგვიწევს დიდი რაოდენობის მონაცემთა დამუშავება.

5.5.3 ატრიბუტები და შედეგები

ჩვენი მომზადებული მონაცემებიდან საჭიროა გამოვყოთ თუ რომელი ველებია ატრიბუტები და რომელი შესაბამისი ატრიბუტების საფუძველზე მიღებული, შედეგი:

X=Data.iloc[:,:23] - ატრიბუტები

Y=Data.iloc[:,23:] - საბოლოო შედეგის ველი

შესაბამისი ჩანაწერის მიხედვით გადავცემთ ინფორმაციას, რომ პირველი 23 ველი წარმოადგენს ატრიბუტებს, ხოლო 24-ე ველი წარმოადგენს მიღებულ შედეგს.

შემდეგ საჭიროა ჩვენ მიერ მომზადებული treining data შევასწავლოთ ჩვენ ალგორითმს (ანუ გადავცეთ treining data) შემდეგი ბრძანების მეშვეობით (fit):

```
Model.fit(
   X_train,
   Y_train,
   eval_metric="rmse",
   eval_set=[(X_test, Y_test)],
   verbose=True, )
```

5.5.4. სასწავლო და სატესტო მონაცემები

შემდეგი ძირითადი ეტაპი მოდელის სიზუსტის გატესტვაა, ანუ მოდელს გადავცემთ სატესტო მონაცემებს. ეს მონაცემები ჩვენს მიერაა მომზადებული. მონაცემების გარკვეული ნაწილი (მაგალითად 20%), რომელიც მოდელისთვის სწავლების პროცესში არ გადაგვიცია. treining data-ს გადაცემის შემდეგ შესაძლებელია მივუთითოთ გადაცემული მონაცემების რა ნაწილი გამოიყენოს სასწავლად, ხოლო რა ნაწილი სატესტოდ. მაგალითად, გადაცემული მონაცემების 80% გამოიყენოს სასწავლად, ხოლო დანარჩენი 20% - სატესტოდ): X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=7)

ალგორითმი გამოიყენებს training data-დან შემენილ ცოდნას და შემუშავებული ლოგიკის მიხედვით დაადგენს სატესტო მონაცემებისთვის საბოლოო შედეგს (საბოლოო ქულა საგანში), შემდეგი ბრძანებით (predict): Test_Predictions=Model.predict(X_test). ვინაიდან, ჩვენ ვიცით ჩვენი სატესტო მონაცემების სწორი შედეგი (რეალური საბოლოო ქულა საგანში) შეგვიძლია შევადაროთ იგი მოდელის მიერ ნავარაუდებ მნიშვნელობას და შევამოწმოთ რამდენად სწორად შეძლო მოდელმა ვარაუდი.

მოდელის სისწორე შეიძლება გაიზომოს განსხვავებულად. მარტივი ახსნა იქნება, რომ ვთქვათ რამდენად ახლოს იყო მოდელის პროგნოზი რეალურ მონაცემთან. ანუ თითოეულ პროგნოზსა და რეალურ შედეგს შორის სხვაობების (დადებით მაჩვენებელში, მანძილი რიცხვებს შორის, მოდული) საშუალო. დაახლოებით ეს არის მოდელის ცდომილება თითოეული მისი პროგნოზისთვის.

5.5.5. საშუალო კვადრატული შეცდომა (RMSE)

რეგრესიის შედეგების სისწორის გასაზომად განსხვავებული ფორმულებით მიღებული მაჩვენებლები გამოიყენება და ჩვენ თავად უნდა განვსაზღვროთ, რომელ მათგანს დავაკვირდეთ და რა არის ჩვენთვის მოდელის მისაღები სიზუსტე. ანუ დასაშვები ცდომილება, რომლის არსებობაც მოდელსა და ჩვენს ამოცანას უსარგებლოს არ გახდის.

გამოყენებადი KPI-ები რეგრესიის შედეგის სიზუსტის ასახვისთვის: MAE (Mean Absolute Error); MSE (Mean_Squared_Error); RMSE; R2 Score.

RMSE-ის დათვლა სატესტო მონაცემებისთვის:

rmse = np.sqrt(mean_squared_error(Y_test, Test_Predictions))

RMSE არის დარჩენილი ნაშთების ვარიანტების კვადრატული ფესვი. ის მიუთითებს მოდელის აბსოლუტურ შესაბამისობას მონაცემებთან – რამდენად ახლოსაა დაფიქსირებული მონაცემების წერტილები მოდელის პროგნოზულ მნიშვნელობებთან. RMSE-ის მცირე მნიშვნელობები მიუთითებს უკეთეს შედეგზე.

ქვემოთ მოცემულია თუ როგორ მცირდება RMSE, ხეების რაოდენობის დამატებასთან ერთად, მსგავსი დაკვირვებით შეგვიძლია ვიპოვოთ ხეების ოპტიმალური რაოდენობა და ასევე დავადგენთ სისწორის კოეფიციენტის მნიშვნელობას. უკეთესია თუ ეს რიცხვი მცირეა.

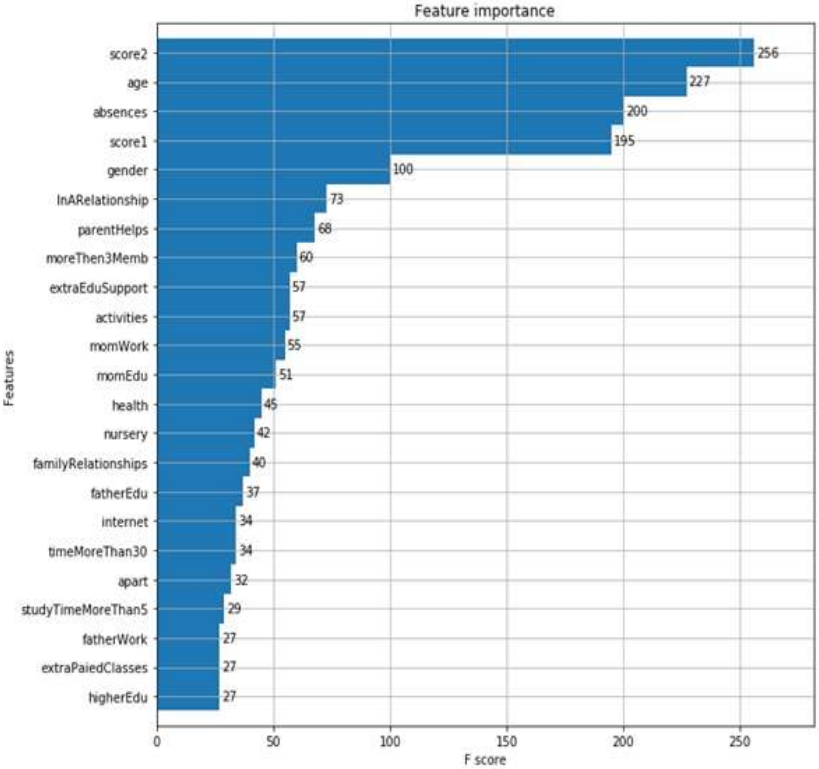
RMSE-ის შედეგი **ცხრ.5.3**

```
[0]     validation_0-rmse:1.65882
[1]     validation_0-rmse:1.57451
[2]     validation_0-rmse:1.51092
[3]     validation_0-rmse:1.45811
[4]     validation_0-rmse:1.40350
[5]     validation_0-rmse:1.36761
[6]     validation_0-rmse:1.33369
[7]     validation_0-rmse:1.30840
[8]     validation_0-rmse:1.29717
[9]     validation_0-rmse:1.28100
[10]    validation_0-rmse:1.26251
```

- MAE: mean_absolute_error (Y_test, Test_Predictions) (0.86)
- MSE: mean_squared_error (Y_test, Test_Predictions) (1.68)
- R2 score: r2_score(Y_test, Test_Predictions) (0.84) საუკეთესო მნიშვნელობა 1 არის.

5.5.6. ატრიბუტების ზემოქმედება

5.3 ნახაზზე მოცემულია კონკრეტული ატრიბუტების ზემოქმედების დიაგრამა შედეგზე. ანუ რომელმა ჩვენს მიერ მოპოვებულმა ინფორმაციამ ითამაშა მნიშვნელოვანი როლი ალგორითმის მიერ ხეების აგებისას.



ნახ. 5.3. ატრიბუტების ზემოქმედების დიაგრამა შედეგზე

საბოლოოდ, მოდელს ეცოდინება, თუ როგორი შედეგები ექნებათ მოსწავლეებს, იმ ფაქტორების გათვალისწინებით, რომლებიც მოქმედებს მათ მომავალ შედეგებზე.

ახალი მონაცემების გადაცემის შემდეგ კი უკვე შესაბამისი მოდელი შეძლებს კონკრეტული მოსწავლისთვის მომავალი სავარაუდო ქულის პროგნოზირებას.

5.6. COMPARE_SENTENCES და STR_SIMILARITY-ის დახმარებით გადაწერის მცდელობების გამოვლენა

მას შემდეგ რაც მივიღებთ შედეგებს, რეალურად ჩატარდება შესაბამისი გამოცდა და გვეცოდინება შედეგები, მოხდება მანქანური დასწავლის მიერ მიღებული და რეალური შედეგების *დადარება.*

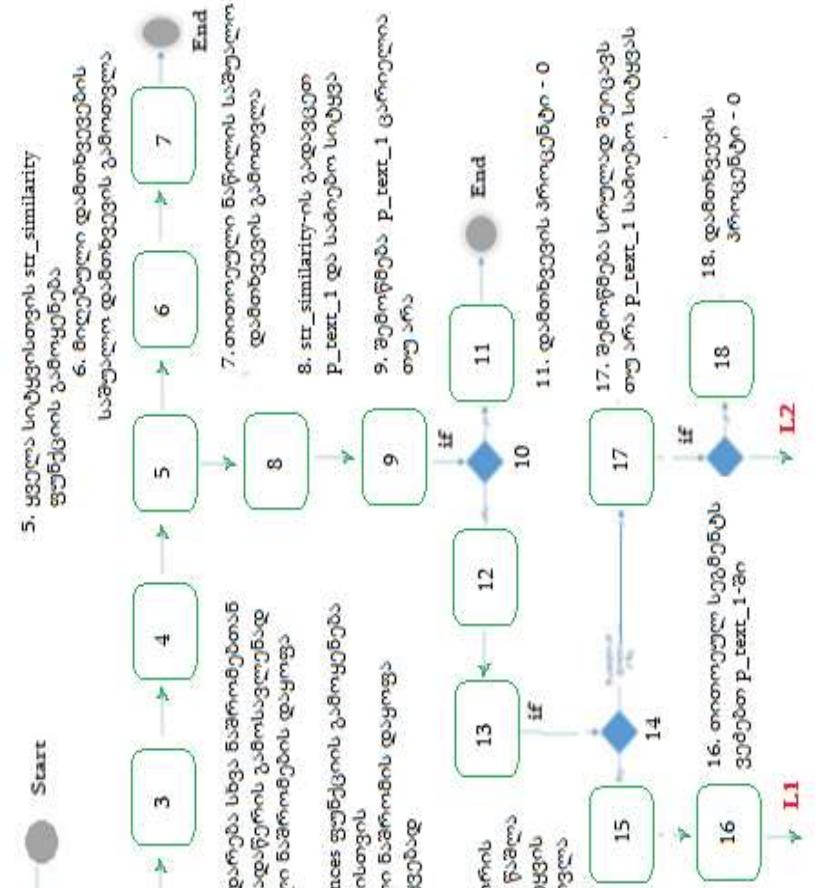
თუ რეალური შედეგი პროგნოზირების შედეგზე კონკრეტული პროცენტით (თავად ორგანიზაცია წყვეს პროცენტის მნიშვნელობას) მეტია, მაშინ დასაშვები ხდება გადაწერის მცდელობა და გენერირდება შესაბამისი მოსწავლეების სია.

5.6.1. COMPARE_SENTENCES

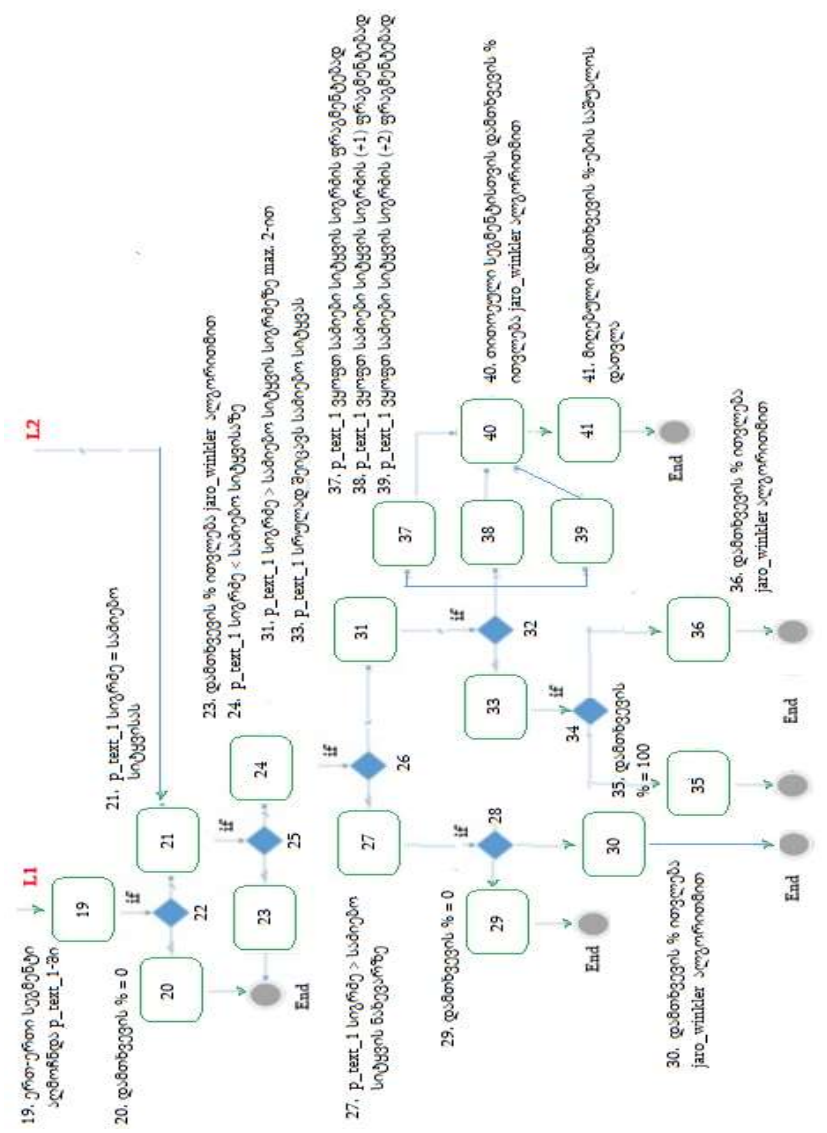
სისტემაში შესაძლებელია საგამოცდო ნაშრომის შედარება სხვა მოსწავლეების ნაშრომებთან.

Compare Sentences ახორციელებს წინადადებების შედარებას.

5.4 ნახაზზე მოცემულია ამ ბიზნესპროცესის UML-აქტიურობის დიაგრამა, რომელიც დეტალურად აღწერს მის ყველა ქმედებას (ოპერაციას) და ბიზნესწესებს.



ნახ. 5.4. COMPARE_SENTENCES და STR_SIMILARITY ფუნქციების აქტიურობის დიაგრამა (დასაწყისი)



ნახ. 5.4-ის გაგრძელება

პირველ რიგში ხდება ნაშრომების დაყოფა ფრაგმენტებად და ფრაგმენტულად მათი დამუშავება, რომლებიც გადაეცემა compare_sentences ფუნქციას:

```
create or replace function compare_sentences(p_text1 in varchar2,
p_text2 in varchar2)
  RETURN number is
  v_new_guid              varchar2(100);
  v_match_percent         number(3, 2);
  v_match_percent_loop    number(3, 2);
begin
  v_new_guid := sys_guid();
  v_match_percent := 0;

  if (p_text1 is null or p_text2 is null) then
    v_match_percent := 0;
  end if;

  if (p_text1 is not null) then

    insert into compare_sentences_library (key_text, inserted_date,
batch_guid_id)
    select regexp_substr(p_text2,'[^ ]+', 1, level), sysdate, v_new_guid from
dual
    connect by regexp_substr(p_text2, '[^ ]+', 1, level) is not null;
    commit;

     for i in (select ID,key_text
               from compare_sentences_library
              where batch_guid_id = v_new_guid) loop
```

```
    v_match_percent_loop := str_similarity (p_text1, i.key_text);

    update compare_sentences_library set match_percent =
v_match_percent_loop where ID=i.ID;
    commit;
    end loop;


   select avg(match_percent) into v_match_percent from
compare_sentences_library where batch_guid_id = v_new_guid;

   delete from compare_sentences_library where batch_guid_id =
v_new_guid;

 end if;
 return v_match_percent;
end;
```

ფუნქცია იყენებს ცხრილს, რომელშიც ახდენს მეორე გადაცემული ტექსტის დაყოფის შედეგად მიღებული სიტყვების ჩაწერას და ანიჭებს თითოეულ ჩანაწერს საერთო უნიკალურ კოდს, რათა მარტივად მოხდეს გარჩევა ჩაწერილი პორციების. შემდეგ ხდება ჩვენი ფუნქციის გამოძახება, str_similarity - ის, რომელსაც გადაეცემა ზემოხსენებული ცხრილიდან თითოეული ჩანაწერი და compare_sentences-სთვის გადაცემული პირველი ტექსტი. ის ცდილობს დაადგინოს შესაბამისი სიტყვა-გასაღების ტექსტთან მსგავსების პროცენტი. შემდეგ მიღებული შედეგების საშუალოთი ადგენს შესაბამისი ორი ტექსტის დამთხვევის პროცენტს.

5.6.2. STR_SIMILARITY და მისი შედარება სხვა მსგავს ალგორითმებთან

როგორც უკვე აღვნიშნეთ str_similarity არის დადარების ალგორითმი.

განვიხილოთ str_similarity ალგორითმი და მისი განსხვავება სხვა არსებულ ალგორითმებთან:

არსებობს რამდენიმე ყველაზე ცნობილი დადარების ალგორითმი, განვილიხოთ რამდენიმე მათგანი :

5.6.2.1. Soundex ალგორითმი

Soundex-ის ალგორითმი აგენერირებს ოთხნიშნა კოდს სიტყვის წარმოთქმის საფუძველზე. შესაბამისი ალგორითმი შეიძლება გამოყენებულ იქნას ორი სიტყვის შესადარებლად, რათა დადგინდეს მათი მსგავსება. ეს შეიძლება ძალიან გამოსადეგი იყოს მაშინ, როცა გვჭირდება ინფორმაციის მოძიება თუნდაც მონაცემთა ბაზაში, ან ტექსტურ ფაილში, განსაკუთებით თუ ვეძებთ, მაგალითად, კონკრეტულ სახელს, რომელიც, როგორც წესი, შეცდომით არის ხოლმე ჩაწერილი.

შესაბამისი ალგორითმი მუშაობს შემდეგნაირად :

1) პირველი სიმბოლო დაგენერირებული კოდის არის გადაცემული ტექსტის პირველი ასო;

2) შემდეგ ის შლის გადაცემულ ტექსტში a, e, h, i, o, u, w, y ასოებს;

3) შემდეგ კონკრეტულ ასოს შეუსაბამებს კონკრეტულ ციფრს:

b, f, p, w = 1

c, g, j, k, q, s, x, z = 2

d, t = 3

l = 4

m, n = 5

r = 6

4) შემდეგ ფუნქცია ეძებს ერთნაირი წონის ასოებს ტექსტში და შლის პირველის გარდა ყველას;

5) საბოლოოდ აბრუნებს პირველ 4 ბაიტს, შევსებულს 0-ებით, იმ შემთხვევაში თუ მიღებული კოდი 4 ბაიტზე ნაკლებია.

select soundex('Technical'); from dual

შედეგი : T252

select soundex('Thechnical'); from dual;

შედეგი : T252

select soundex('Technical'); from dual

შედეგი : T252

select soundex('Technital'); from dual;

შედეგი : T253

მაგრამ შესაბამისი ალგორითმი ჩვენთვის არაოპტიმალურია, რადგან ჩვენ შემთხვევაში *დამთხვევის პროცენტია მნიშვნელოვანი და არა ორობითი პასუხი*, სადაც ან ორი სიტყვა ერთნაირად ჟღერს ან არა. მითუმეტეს, რომ ქვევით მოყვანილი მაგალითი Soundex ალგორითმისთვის არ არის მსგავსი, ჩვენთვის კი თითქმის იდენტური უნდა იყოს, ხოლო მათი მსგვასება უნდა გამოხატული იყოს კონკრეტულ პროცენტებში.

select soundex('Georgia') from dual;

შედეგი : G620

select soundex('Georgian') from dual;

შედეგი : G625

ამავდროულად შესაბამისი ალგორითმი ვერ მუშაობს წინადადებასთან ერთიანად:

select soundex('Georgian Technical University'); from dual

შედეგი : G625

```
select soundex('Georgia') from dual;
```
შედეგი : G620
```
select soundex('Georgian Technical University'); from dual
```
შედეგი : G625
```
select soundex('Gheorgian') from dual;
```
შედეგი : G620
```
select soundex('Georgian Technical University'); from dual
```
შედეგი : G625
```
select soundex('Thechnical'); from dual;
```
შედეგი : T252

5.6.2.2. Jaro similarity ალგორითმი

Jaro similarity არის ორ სტრიქონს შორის მსგავსების საზომი. მიღებული მნიშვნელობა 0-დან 1-მდეა, სადაც 1 ნიშნავს რომ ტექსტები იდენტურია, ხოლო 0 ნიშნავს რომ არ არსებობს არანაირი მსგავსება სტრიქონებს შორის.

$$Jaro\ similarity = \begin{cases} 0, \text{ if m=0} \\ \frac{1}{3}\left(\frac{m}{|s1|} + \frac{m}{|s2|} + \frac{m-t}{m}\right), \text{ for m!=0} \end{cases}$$

სადაც :

- m სიმბოლოების დამთხვევების რაოდენობაა;
- t ტრანსპოზიციაა, სადაც ტრანსპოზიცია არის დამთხვეული სიმბოლოების რაოდენობის ნახევარი ორივე ტექსტში (განსხვავებული თანმიმდევრობით);
- |s1| და |s2| არის s1 და s2 ტექსტების სიგრძეები.

5.6.2.3. jaro_winkler ალგორითმი

jaro_winkler-ის მსგავსების ალგორითმი არის სტრიქონების მსგავსების საზომი, რომელიც ზომავს რედაქტირების მანძილს ორ სტრიქონს შორის. jaro-winkler ჰგავს jaro-ს. ორივე განასხვავებს, როდესაც ორი სტრიქონის პრეფიქსი ემთხვევა. Jaro_winkler-ის მსგავსება იყენებს პრეფიქსის მასშტაბს "p", რომელიც იძლევა უფრო ზუსტ პასუხს, როდესაც სტრიქონებს აქვს საერთო პრეფიქსი განსაზღვრული მაქსიმალური სიგრძით.

$$Sw = Sj + P * L * (1 - Sj)$$

სადაც :

- Sj არის jaro - ს შედეგი;
- Sw არის jaro- winkler -ის შედეგი;
- P მაშტაბურობის ფაქტორი (0.1 default);
- L არის დამთხვეული პრეფიქსის სიგრძე მაქსიმუმ 4 სიმბოლომდე.

მაგ.: 2.1

```
select UTL_MATCH.jaro_winkler('Georgian','Georgian') from dual;
```

შედეგი : 1

მაგ.: 2.2

```
select UTL_MATCH.jaro_winkler('GEORGIAN','Georgian') from dual;
```

შედეგი : 0.47

მაგ.: 2.3

```
select UTL_MATCH.jaro_winkler('it is unbearables','unbearable') from
dual;
```

შედეგი : 0.79

მაგ.: 2.4

```
select UTL_MATCH.jaro_winkler('it isunbearables','unbearable') from dual;
```

შედეგი: 0.80

მაგ.: 2.5

```
select UTL_MATCH.jaro_winkler('it is Jeorgian University','Georgian') from dual;
```

შედეგი: 0.57

შესაბამისი ალგორითმი ვერ ახდენს იმ შედეგის მოცემას რაც ჩვენ გვჭირდება, რადგან თუნდაც არის "Key Sensitive" და არამხოლოდ. სწორედ ამიტომ შევქმენით ჩვენი ორიგინალური ალგორითმი str_similarity.

Str_similarity ახდენს მსგავსების პროცენტის დაბრუნებას. ის იყენებს jaro_winkler-ის ალგორითმს, მაგრამ მხოლოდ იმ შემთხვევაში, როდესაც შესაბამისი ალგორითმის დაბრუნებული შედეგი მისაღებია ჩვენი ამოცანისთვის.

5.6.2.4. str_similarity ალგორითმის კოდი

```
create or replace function str_similarity(p_text_1 in varchar2, p_key  in varchar2)
 RETURN number is
 v_text            varchar2(2000);
 v_key             varchar2(2000);
 v_segment         varchar2(2000);
 v_key_segment     varchar2(2000);
 v_text_count      number(10);
 v_key_text_count  number(10);
```

```
  v_diff                number(10);
  v_match_percent       number(3, 2);
  v_segment_match_percent  number(3, 2);
  v_max_loop            number(10);
  v_max_loop2           number(10);
  v_half_match          number(1);
begin
  v_match_percent       := 0;
  v_text            := UPPER(regexp_replace(p_text_1,
                                '[[:space:]]*',
                                ''));
  v_key := UPPER(regexp_replace(p_key,
                                '[[:space:]]*',
                                ''));


  if (p_text_1 is null) then
    v_match_percent := 0;
  end if;

  if (p_text_1 is not null) then


      v_text_count     := length(v_text);
      v_key_text_count     := length(v_key);
      v_diff           := v_text_count - v_key_text_count;
      v_match_percent := 0;
      v_half_match       := 0;

      v_max_loop2 := ROUND(v_key_text_count / 2 - 1);
```

```
if (v_key_text_count > 2) then
  for i in 1 .. v_max_loop2 loop
    v_key_segment := SUBSTR(v_key,
                            i,
                            ROUND(v_key_text_count / 2 + 1));

    if (INSTR(v_text, v_key_segment) > 0) then
      v_half_match := 1;
      EXIT;
    end if;
  end loop;
else
  if (INSTR(v_text, v_key) > 0) then
    v_half_match := 1;
  end if;
end if;

 if (v_half_match = 1) then

  if (v_diff = 0)
   then
    v_match_percent := UTL_MATCH.jaro_winkler(v_text,
                                              v_key);
  end if;
  if (v_diff < 0 and v_text_count > v_key_text_count / 2)
   then
    v_match_percent := UTL_MATCH.jaro_winkler(v_text,
                                              v_key);
  end if;
```

```
if (v_diff < 0 and v_text_count < v_key_text_count / 2)
 then
  v_match_percent := 0;
end if;
if (v_diff > 0)
 then
  if (v_diff <= 2)
   then
    if (INSTR(v_text, v_key) > 0) then
     v_match_percent := 1;
    else
     v_match_percent := UTL_MATCH.jaro_winkler(v_text,
                                        v_key);
    end if;
   else

    for extra_symbols_count in 0 .. 2
     loop
      v_max_loop := (v_text_count -
                (v_key_text_count + extra_symbols_count)) + 1;

      for i in 1 .. v_max_loop loop
       v_segment := SUBSTR(v_text,
                    i,
                    v_key_text_count + extra_symbols_count);


       v_segment_match_percent                                    :=
UTL_MATCH.jaro_winkler(v_segment,
                                          v_key);
```

```
            if (v_segment_match_percent > v_match_percent) then
              v_match_percent := v_segment_match_percent;
            end if;
          end loop;

        end loop;
      end if;
    end if;

  else
    v_match_percent := 0;
  end if;

end if;
return v_match_percent;

end;
```

5.6.2.5. str_similarity-ის ბიჯების აღწერა

str_similarity საბოლოო შედაგის მისაღებად გადის შემდეგ ბიჯებს:

str_similarity-ს გადაეცემა ორი პარამეტრი ტექსტი text1 და საძიებო სიტყვა-გასაღები.

თუ text1 არის null, მაშინ დამთხვევის პროცენტი ავტომატურად არის 0.

1.1. თუ გასაღების სიგრძე მეტია 2-ზე და გასაღების გარკვეულ სეგმენტს მაინც არ შეიცავს text1, მაშინ პროცენტი ავტომატურად ხდება 0-ის ტოლი. გასაღების სეგმენტის სიგრძეს ვითვლით გასაღების სიგრძის ნახევარს +1,

ხოლო შესაბამის სეგმენტს კი თითო წევრის გადანაცვლებით ვიღებთ. ანუ, მაგალითად, თუ გასაღებია smsoff, მაშინ მიღებული სეგმენტები იქნება smso,msof,soff. თუ ამ სამი სეგმენტიდან ვერცერთის პოვნა ვერ მოხერხდა text1-ში, მაშინ პროცენტი იქნება 0.

თუ 1.1.-ის გავლა წარმატებით მოხდა, ანუ მოხდა რომელიმე სეგმენტის დამთხვევა, მაშინ:

1.2. თუ გადმოცემული ტექსტის სიგრძე ტოლია გასაღების სიგრძის, მაშინ ვითვლით დამთხვევის პროცენტს (jaro_winkler);

1.3. თუ გადმოცემული ტექსტის სიგრძე გასაღების სიგრძეზე ნაკლებია, მაგრამ მის ნახევარზე მეტია, მაშინ ვითვლით დამთხვევის პროცენტს (jaro_winkler);

1.4. თუ გადმოცემული ტექსტის სიგრძე გასაღების სიგრძის ნახევარზე ნაკლებია, მაშინ პროცენტი ტოლია 0 - ის;

1.5. თუ გადმოცემული ტექსტის სიგრძე გასაღების სიგრძეზე მეტია მაქსიმუმ დასაშვები 2 სიმბოლოთი და ის სრულად შეიცავს გასაღებს, მაშინ პროცენტი ტოლია 100-ის;

1.6. თუ გადმოცემული ტექსტის სიგრძე გასაღების სიგრძეზე მეტია მაქსიმუმ დასაშვები 2 სიმბოლოთი და ის სრულად არ შეიცავს გასაღებს, მაშინ ვითვლით დამთხვევის პროცენტს (jaro_winkler);

1.7. თუ გადმოცემული ტექსტის სიგრძე გასაღების სიგრძეზე მეტია ვიდრე დასაშვები 2 სიმბოლოთი, მაშინ შესაბამის ტექსტს ვყოფთ სეგმენტებად (პირველ რიგში ვიღებთ გასაღების სიგრძის მქონე სეგმენტებს წანაცვლებით, შემდეგ ვიღებთ გასაღების სიგრძეს +1 სიგრძის მქონე სეგმენტებს წანაცვლებით, ხოლო ბოლოს ვიღებთ გასაღების სიგრძეს +2 სიგრძის მქონე სეგმენტებს წანაცვლებით) და თითოეული სეგმენტისათვის გამოთვლილი პროცენტებიდან ვიღებთ მაქსიმუმს.

5.6.3. შედარების ანალიზის შედეგები

გავაანალიზოთ ალგორითმების შედარების შედეგები და შევიმუშაოთ რეკომენდაციები მათი სრულფასოვანი და ეფექტიანი გამოყენების შესახებ.

ვადარებთ ჩვენი ალგორითმისა და soundex-ის მიერ დაბრუნებულ პასუხებს ერთიდაიმავე შემავალი ტექსტის შემთხვევაში:

```
select soundex('Georgia') from dual; -- G620
select soundex('Georgian') from dual;  -- G625
select str_similarity('Georgia','Georgian') from dual; -- 0.97
```

soundex-ის მიხედვით 'Georgia' და 'Georgian' სიტყვები ჟღერადობით არ არიან მსგავსები, ჩვენი ალგორითმის მიხედვით კი დამთხვევა არის 97%.

შევადაროთ ჩვენი ალგორითმისა და Jaro_Winkler-ის მიერ დაბრუნებული პასუხები ერთსა და იმავე შემავალი ტექსტის შემთხვევაში [20]:

(გამოვიყენოთ მაგ.: 5.1)

```
select UTL_MATCH.jaro_winkler('Georgian','Georgian') from dual; -- 1
select UTL_MATCH.jaro_winkler('GEORGIAN','Georgian') from dual; --
0.47
select str_ similarity ('Georgian','Georgian') from dual; -- 1
select str_ similarity ('GEORGIAN','Georgian') from dual; -- 1
```

(გამოვიყენოთ მაგ.: 5.2)

```
select UTL_MATCH.jaro_winkler('GEORGIAN','Georgian') from dual; --
0.47
select str_ similarity ('GEORGIAN','Georgian') from dual; -- 1
```

(გამოვიყენოთ მაგ.: 5.3)

```
select UTL_MATCH.jaro_winkler('it is unbearables','unbearable') from dual; --0.79
select str_ similarity ('it is unbearables','unbearable') from dual; -- 1
```

(გამოვიყენოთ მაგ.: 5.4)

```
select UTL_MATCH.jaro_winkler('it isunbearables','unbearable') from dual; --0.80
select str_ similarity ('it isunbearables','unbearable') from dual; -- 1
```

(გამოვიყენოთ მაგ.: 5.5)

```
select UTL_MATCH.jaro_winkler('it is Jeorgian University','Georgian') from dual; -- 0.57
select str_ similarity ('it is Jeorgian University','Georgian') from dual; -- 0.92
```

[20].

დასკვნა

მონოგრაფიაში გამოკვლეულ იქნა ინფორმატიკის (კომპიუტინგის) დიდაქტიკის მეცნიერული მიმართულების საკითხები ინფორმაციული საზოგადოების ფორმირების პროცესის მხარდაჭერის თვალსაზრისით. განხორციელდა სასწავლო დაწესებულებათა (*სდ*: სკოლა, უნივერსიტეტი, პროფესიული და ა.შ.) ბიზნესპროცესების ობიექტ-ორიენტირებული ანალიზი, მათი კლასიფიკაცია, ობიექტ-ორიენტირებული პროექტირება და იმპლემენტაცია, დანერგილი სისტემების შესწავლა, პრობლემების აღმოჩენა და შემდეგ მათი გადაჭრის გზების მოძიება. ყურადღება მახვილდება ისეთ პროცესებზე, რომლებიც მასწავლებლებს გაუმარტივებს სასწავლო პროცესის მართვას და ხელს შეუწყობს მოსწავლეებს/სტუდენტებს განვითარებასა და წინსვლაში.

შეიქმნა პლატფორმა, რომელიც უზრუნველყოფს მოსწავლეთა/სტუდენტთა დასწრების კონტროლის, მოსწავლეთა/სტუდენტთა შეფასებების ანალიზისა და მათი დამუშავების, მოსწავლეთა/სტუდენტთა მიღწევების ანალიზის, სწავლის საფასურის გადახდების ისტორიის მონიტორინგის და პროგნოზირების, ბიბლიოთეკაში მიმდინარე პროცესების ავტომატიზაციას და მეტად გამჭირვალობას. კერძოდ ნაშრომში:

• განხორციელდა საგანმანათლებლო დაწესებულებებში მიმდინარე პროცესების კლასიფიკაცია და განისაზღვრა საავტომატიზაციო ამოცანები და პროცესები, მათი შემდგომი სრულყოფისა და გამარტივებისთვის;

• აგებულ იქნა საავტომატიზაციო პროცესების Agile (მოქნილი) და UML, (უნიფიცირებული), სტატიკურ/დინამიკური მოდელები;

• განისაზღვრა ინფორმატიკის და კომპიუტერული მევნიერების კლასიკური და ახალი ციფრული ტექნოლოგიების აკადემიური კურსების ინტერდისციპლინური სწავლების მეთოდოლოგია ობიექტ-, პროცეს- და სერვის-ორიენტირებული მიდგომებით;

• შეიქმნა მოქნილი არქიტექტურის სისტემა სერვისებით, დომენზე ორიენტირუებული დიზაინით, რომელიც მთლიანად მორგებულია საგანმანათლებლო სისტემის პროცესებზე;

• შემუშავდა საპრობლემო სფეროს პორტალის აგების CASE ტექნოლოგია დაპროგრამების ავტომატიზაციით, ღრუბლოვანი მონაცემთა საცავით და უსაფრთხოების BSI / ITIL სტანდარტებით;

• ***სდ***-თვის შემუშავდა თვითდასწავლადი ავტომატიზაბული სისტემა ML-ის ალგორითმების გამოყენებით. შეიქმნა ტექსტების დადარების ალგორითმი, რომელიც გამოიყენება კონკრეტული ამოცანების გადასაჭრელად და ის რიგ შემთხვევებში იძლევა უფრო ზუსტ, მეტად ოპტიმალურ შედეგს, ვიდრე არსებული, ტრადიციული ალგორითმები. შედეგები ილუსტრირებულია კონკრეტული მაგალითებით.

გამოყენებული ლიტერატურა:

1. ჩოგოვაძე გ., ფრანგიშვილი ა., ჯაგოდნიშვილი თ., სურგულაძე გ. საინფორმაციო სისტემებიდან ინფორმაციული საზოგადოებისაკენ. სტუ, შრ.კრ. „მას“, N 1(23). თბ., 2017. გვ.7-16

2. Resolution adopted by the General Assembly on 27 March 2006. A/60/L.50. World Summit on the Information Society. Intern.resource: http://wikivisually.com/wiki/World_Information_Society_Day (10.05.23)

3. World Summit on the Information Society Forum 2023. Internet resource: https://www.itu.int/net4/wsis/forum/2023/en

4. მიმდინარე პროგრამები. განათლების და მეცნიერების სამინისტრო. ინტერნეტ რესურსი: https://www.mes.gov.ge/content.php?id=536&lang=geo, (25.06.2023)

5. საქართველოს განათლებისა და მეცნიერების სამინისტრო. განათლებისა და მეცნიერების სისტემის განვითარების სტრატეგიული მიმართულებები. ინტერნეტ რესურსი: https://www.mes.gov.ge/uploads/strategia..pdf, (25.06.2023)

6. პაპავაძე ს. სასწავლო დაწესებულებებში მიმდინარე პროცესების ავტომატიზებული მართვა. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N2(29), 2019. გვ.174-178

7. პაპავაძე ს., სურგულაძე გ., ნარეშელაშვილი გ. სასწავლო დაწესებულების მენეჯმენტის პროცესების სრულყოფა ინფორმაციული ტექნოლოგიების გამოყენებით. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(32), 2021. გვ. 206-211

8. პაპავაძე ს. მანქანური დასწავლის მეთოდების გამოყენება სასწავლო დაწესებულებებში. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(30), 2020. გვ. 124-130.

9. Booch G., Jacobson I., rambaugh J. Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara, 1996

10. გოგიჩაიშვილი გ., ბოლხი გ., სურგულაძე გ., პეტრიაშვილი ლ. მართვის ავტომატიზებული სისტემების ობიექტორიენტირებული დაპროექტების და მოდელირების ინსტრუმენტები (MsVisio, WinPepsy, PetNet, CPN). სტუ.. თბილისი, „ტექნიკური უნივერსიტეტი“, 2013, 232 გვ. https://gtu.ge/book/ims/Gogichai-Surgul.pdf

11. Петкович Д. Microsoft SQL Server 2012. Руководство для начинающих: Пер. с англ. - СПб.: БХВ-Петербург. 2013

12. სურგულაძე გ., კივილაძე გ. შესავალი NoSQL მონაცემთა ბაზებში. ISBN978-9941-0-9642-6. სტუ. „IT-კონსალტინგ ცენტრი“. თბ., 2017. -152 გვ. https://gtu.ge/book/NoSQL_Surgul.pdf

13. სამხარაძე რ., გაჩეჩილაძე ლ. SQL სერვერი. სტუ. თბ., 2016. -450 გვ. https://gtu.ge/book/SQL_Serveri.pdf

14. Tactical Domain-Driven Design with Angular and Monorepos. 2019. Tactical Domain-Driven Design with Angular and Monorepos? - ANGULARarchitects, (25.06.2022).

15. Design a DDD-oriented Microservice. 2021. Internet resource: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice, (25.06.2021).

16. Gorodinski L. Services in Domain-Driven Design (DDD). 2012. http://gorodinski.com/blog/2012/04/14/services-in-domain-driven-design -ddd/

17. Rajesh Rajagopalan. Domain-Driven Microservices Design from a practitioner’s view. 2020. Domain-Driven Microservices Design from a practitioner's view (Part 1) | PeerIslands, (20.06.2022)

18. Ibtehal Talal Nafea. Machine Learning In Educational Technology. 2018. https://www.intechopen.com/books/machine-learning-advanced-techniques-and-emerging-applications/machine-learning-in-educational-technology, (20.06.2022)

19. ჩოგოვაძე გ., სურგულაძე გ., გულიტაშვილი მ., დოლიძე ს. პროგრამული აპლიკაციების ხარისხის მართვა: ტესტირება და ოპტიმიზაცია. სტუ. „IT-კონსალტინგ ცენტრი“. თბ., 2020. -363 გვ. https://gtu.ge/book/Surgu_SoftwareQuality.pdf

20. პაპავაძე ს. სასწავლო დაწესებულებებში გამოცდების შედეგების განხილვა და დამუშავება. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(32), 2021. გვ. 161-167

21. სურგულაძე გ. ინფორმატიკის ინოვაციური დიდაქტიკა: 100 წიგნი - სტუ-ს 100 წლის იუბილესთვის (ისტორია და რეალობა). საქართველოს ტექნიკური უნივერსიტეტის 100 და იმს ფაკულტეტის 65 წლისთავისადმი მიძღვნილი საერთაშორისო სამეცნიერო - პრაქტიკული კონფერენციის „ინოვაციები და თანამედროვე გამოწვევები“. შრ.კრ., 19-20 ნოემბერი, 2022. გვ. 11-16

22. https://en.wikipedia.org/wiki/Information_society

23. Mansell, Robin. The information society. Critical concepts in sociology. Routledge, London, UK. 2009. https://eprints.lse.ac.uk/23743/

24. ჩოგოვაძე გ., ფრანგიშვილი ა., სურგულაძე გ., პეტრიაშვილი ლ., ამილახვარი ნ. „ინფორმატიკის“ – მულტიდისციპლინური მეცნიერების თანამედროვე გამოწვევები და სტუ-ის სადოქტორო პროგრამის სტრატეგიული მიზნები. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(30), 2020. გვ. 5-10

25. Informatics education at school in Europe. European Commission / EACEA / Eurydice, 2022. Eurydice report. Luxembourg: Publications Office of the European Union. By Mariya Gabriel. Commissioner responsible for Innovation, Research, Culture, Education and Youth

26. ჩოგოვაძე გ., სურგულაძე გ., თოფურია ნ., ხარიტონაშვილი მ. ინფორმაციული საზოგადოება და ინტერდისციპლინური სწავლება ციფრული ტექნოლოგიების ბაზაზე. ISBN 978-9941-8-3338-0. მონოგრაფია. სტუ. „IT-კონსალტინგ.სამეცნ. ცენტრი”, თბ., 2021. –360 გვ. https://gtu.ge/book/Surgu_ InfoSociety-21%20new.pdf

27. Осмоловская И.М. Дидактика: от классики к современности: монография. ISBN 978-5-4469-1706-8, –М.; СПб., Ред. „Нестор-История", 2020. –248 с.

28. Flipped Classrooms - Active Learning. Harvard University. The Derek Bok Center for Teaching and Learning. Internet resource: https://bokcenter.harvard.edu/flipped-classrooms (10.09.22)

29. სურგულაძე გ. ინტერდისციპლინური სწავლების კონცეფცია მართვის საინფორმაციო სისტემების სპეციალობაზე UML/2-ის ფონზე. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(6), 2009. გვ. 11-15

30. ზუმბურიძე ო., ძიძიგური გ., ლომინაძე თ., ჟიჟილაშვილი ნ. საინჟინრო განათლება ევროპაში: მომზადება პროფესიული საინჟინრო სტანდარტებისთვის - CDIO ინიციატივა. ევროკომისიის საგრანტო კონკურსის TEMPUS-ის პროექტი. სტუ-ს შრ.კრ. „მას", N 1(17), 2014, გვ. 7-10.

31. ზუმბურიძე ო., ძიძიგური გ., ლომინაძე თ., ჟიჟილაშვილი ნ. საინჟინრო განათლების პროგრამების მიმოხილვა წამყვან ევროპულ უნივერსიტეტებში: დიდი ბრიტანეთის (UNIVERSITY OF LEEDS), შვედეთის (KUNGLIGA TEKNIꟾKA HOGꟾKOLAN) და იტალიის (POLITECNICO DI TORINO) მაგალითები. TEMPUS-ის პროექტი. სტუ-ს შრ.კრ. „მას", N 1(17), 2014, გვ. 11-16

32. ჩოგოვაძე გ., ფრანგიშვილი ა., კივილაძე გ., სურგულაძე გ., ნარეშელაშვილი გ. ინფორმაციული საზოგადოება, მონაცემთა მენეჯმენტის ახალი ტექნოლოგიები და ექსტრემალური სიტუაციების მართვის სისტემები. სტუ-ს შრ.კრ. „მას", N 1(25), 2018, გვ. 7-16

33. სურგულაძე გ., კივილაძე გ., კახელი ბ. Big Data ტიპის მონაცემების კვლევითი ცენტრის ფორმირება. სტუ-ს შრ.კრ. „მას“.N2 (26), თბ., 2018, გვ. 189-192

34. Enterprise Architect. https://sparxsystems.com/

35. Visual Paradigm. https://en.wikipedia.org/wiki/Visual_Paradigm

36. Halpin T. ORM2, Graphical Notation, Neumont University. 2005. http://www.orm.net/ pdf/ORM2_-TechReport1.pdf.

37. Meier R. Professional Android™ 4 Application Development. John Wiley & Sons, Inc., Indianapolis, Indiana Published simultaneously in Canada. 2012. https://staff.emu.edu.tr/ruhsanonder/en/Documents-/professional_android_4_application_development%20-%20Copy.pdf

38. სურგულაძე გ., ოხანაშვილი მ., სურგულაძე გ. მარკეტინგის ბიზნეს-პროცესების უნიფიცირებული და იმიტაციური მოდელირება. მონოგრ., სტუ. „ტექნიკური უნივ.", თბ., 2009. 170 გვ.

39. სურგულაძე გ., ქრისტესიაშვილი ხ., სურგულაძე გ. საწარმოო რესურსების მენეჯმენტის ბიზნეს-პროცესების მოდელირება და კვლევა. მონოგრ., სტუ. „ტექნიკური უნივერსიტეტი". თბ., 2015. –216 გვ.

40. ჩოგოვაძე გ., ფრანგიშვილი ა., სურგულაძე გ. მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. თბილისი, „ტექნიკური უნივერსიტეტი", 2017, 1001 გვ.

41. OMG. Unified Modeling Language. Version 2.5.1 (OMG UML), 2017, 796 p. https://www.omg.org/spec/UML/2.5.1/pdf

42. What is Unified Modeling Language (UML)? (visual-paradigm.com) https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/, (25.02.2023)

43. Beck K., Fowler M., Martin R.C., Mellor S. et al. (2001). Manifesto for Agile Software Development. https://agilemanifesto.org/

44. What is Agile Software Development? Internet resource: http://www.inflectra.com/Metho-dologies/AgileDevelopment .aspx#Scrum

45. Boruta I. Agile Methodology. Mother of dragons or all agile methodologies. w-Blog. 2017. Internet resource: https://worksection. com/blog/-agile.html

46. Resources Scrum. 2019 Internet resource: https://www.scrum.¬org/resources

47. Anderson, David J. Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press. 2017

48. სურგულაძე გ. კომპიუტერული პროგრამირების მეთოდები და მეთოდოლოგიები (SP, OOP, VP, Agile, UML). სტუ. „IT-კონსალტინგ ცენტრი“. თბ., 2019. -200 გვ

49. სურგულაძე გ. მართვის ავტომატიზებული სისტემების მიმართულებით ინტერდისციპლინარული სწავლების კონცეფცია UML/2 ტექნოლოგიის ფონზე. სტუ-ს შრ.კრ. „მას", N 1(25), 2009, გვ. 11-15

50. რეისიგი ვ. , სურგულაძე გ., გულუა დ. ვიზუალური ობიექტ-ორიენტირებული დაპროგრამების მეთოდები (BorlandC++Builder, PetriNet). ISBN 99928-943-9-3, სახელმძღ., სტუ, „ტექნიკური უნივერსიტეტი". თბ., 2002, -200 გვ.

51. ბოტჰე კ., სურგულაძე გია, დოლიძე თ., შონია ო., , სურგულაძე გიორგი. თანამედროვე პროგრამული პლატფორმები და ენები (WindowsNT, Unix, Linux, C++, Java, XML). ISBN 99940-14-11-0. სახელმ., სტუ, „ტექნიკური უნივერსიტეტი". თბ., 2003, -250 გვ.

52. Bothe K., Surguladze G. Objektorientierte Modellierung und Programmierung mit der UML. ISBN 99940-56-01-8. GTU, Tbilissi - Berlin. 60 S.

53. გოგიჩაიშვილი გ., სურგულაძე გ., შონია ო. დაპროგრამების მეთოდები: სტრუქტურული და ობიექტ-ორიენტირებული მიდგომების საფუძვლები C და C++ ენების ბაზაზე. სახელმძღვ. სტუ, „ტექნიკური უნივერსიტეტი", თბ., 1997. -275 გვ.

54. სურგულაძე გ., პეტრიაშვილი ლ. მონაცემთა საცავის დაპროექტება და აგება ინტერნეტული ბიზნესისათვის. ISBN 978-9941-8-0623-0, მონოგრაფია. სტუ-ს „IT-კონსალტინგ სამეცნიერო ცენტრი". თბ., 2022, - 200 გვ.

55. Meyer M., Winter, R. Organization of Data Warehousing in Large Service Companies: A Matrix Approach Based on Data Ownership and Competence Centers, Journal of Data Warehousing, 6 (4), 2018

56. Codd E.F., Codd S.B., Salley C.T. Providing OLAP (On-Line Analytical Processing) to User Analyst: An IT Mandate." Available from Arbor Software's. 1993.

57. Inmon, W.H. Building the Data Warehouse (Third Edition), New York: John Wiley & Sons, 2002

58. ჩოგოვაძე გ., სურგულაძე გ., შონია ო. მონაცემთა და ცოდნის ბაზების აგების საფუძვლები. სახელმძღვ. „განათლება", თბ., 1996. - 375 გვ.

59. სურგულაძე გ., შონია ო., ყვავაძე ლ. მონაცემთა განაწილებული ბაზების მართვის სისტემები (Ms SQL Server, Access, InterBase, JDBC, Oracle). სახელმძღვ., ISBN 99940-35-18-5. სტუ, თბ., 2004. -230 გვ.

60. Types of Databases. Internet resource: https://www.javatpoint. com/types-of-databases (20.04.23)

61. What Are The Various Types of Databases? Internet resource: https://www.simplilearn.com/tutorials/dbms-tutorial/what-are-various-types-of-databases (12.06.23)

62. Чоговадзе Г., Качибая В., Сургуладзе Г. Теория реляционных зависимостей и проектирование логической схемы баз данных. Моногр., ISBN 5-511-00072-8 1988, Тбил. Госуд. Унив. им. Ив. Джавахишвили. -230 ст.

63. სურგულაძე გ., თოფურია ნ., ბერულავა ა. პროგრამული პროდუქტების დეველოპმენტი. ISBN 978-9941-8- 3810-1, დამხმარე სახელმძ., სტუ-ს „IT-კონსალტინგ ცენტრი". თბ., 2022, - 250 გვ.

64. სურგულაძე გ., გულუა დ., კახელი ბ. პროგრამული აპლიკაციების აგება ვირტუალიზაციის პირობებში. ISBN 978-9941-8-0627-4. სტუ. „IT-კონსალტინგ ცენტრი". თბ., 2019. -159 გვ.

65. მეიერ-ვეგენერი კ., სურგულაძე გ., ბასილაძე გ. საინფორმაციო სისტემების აგება მულტიმედიური მონაცემთა ბაზებით. ISBN 978-9941-20-468-5. სტუ. „ტექნიკ.უნივერს.“, თბ., 2014 -345 გვ.

66. Surguladze G., Turqia E., Topuria N., Gavardashvili A. Construction of the Multimedia Databases and Users Interfaces for Ecological System of Black Sea with Orm/Erm. Nova Science Publishers. © Copyright, 3rd Quarter. Chapter 45. 2017, USA. https://novapublishers .com/shop/information-and-computer-technology-modeling-and-control -proceedings-of -the-international-scientific-conference-devoted-to-the-85th-anniversary-of-academician-i-v-prangishvili/

67. Halpin T. Object Role Modeling: An Overview.Internet rersource: https://courses.washington.edu/css475/orm.pdf

68. სურგულაძე გ., პეტრიაშვილი ლ. აპლიკაციების დაპროგრამება და მონაცემთა მენეჯმენტი. ISBN 978-9941-8- 3810-1, სტუ-ს „IT-კონსალტინგ სამეცნიერო ცენტრი". თბ., 2022, - 135 გვ.

69. Neo4j. Int.resource: https://en.wikipedia.org/wiki/Neo4j (1.5.23)

70. OrientDB. https://en.wikipedia.org/wiki/OrientDB (10.5.23)

71. TinkerPop. Documentation. Internet resource: https://hackolade.com/help/TinkerPop.html (28.07.23)

72. Apache TinkerPop - The Gremlin Console. Internet resource: https://tinkerpop.apache.org/docs/current/tutorials/the-gremlin-console /#toy-graphs (28.07.23)

73. Redis. Internet resource: https://en.wikipedia.org/wiki/Redis (28.07.23)

74. Advantages And Disadvantages of Redis. Internet resource: https://prosconslab.com/articles/32/13-advantages-and-disadvantages-of-redis (31.07.23)

75. Computer Aided Software Engineering (CASE). Internet resource: https://www.geeksforgeeks.org/computer-aided-software-engineering-case/ (28.07.23)

76. ვედეკინდი ჰ., სურგულაძე გ., თოფურია ნ. განაწილებული ოფის-სისტემების მონაცემთა ბაზების დაპროექტება და რეალიზაცია UML-ტექნოლოგიით. ISBN 99940-57-17-0. სტუ. „ტექნიკ. უნივერს.“, თბ., 2006 -237 გვ.

77. სურგულაძე გ., თურქია ე. პროგრამული სისტემების მენეჯმენტის საფუძვლები. ISBN 978-9941-20-651-1. სტუ. „ტექნიკ. უნივერსიტეტი“, თბ., 2016. -350 გვ.

78. Crispin L., Gregory J. (2009). Agile Testing: a Practical Guide for Testers and Agile Teams. Addison-Wesley. – 579 p.

79. სურგულაძე გ., ხატიაშვილი ხ. მართვის ინფორმაციული სისტემების აგება სუფთა არქიტექტურით ASP.NET Core გარემოშიჟისტუ-ს შრ.კრებ..: „მას"-N2(34), თბ., 2022. გვ. 55-62. DOI: https://doi.org/ 10.36073/1512-3979

80. Patel A. (2021). Clean Architecture with .NET and .NET Core - Overview. https://medium .com/dotnet-hub/clean-architecture-with-dotnet-and-dotnet-core-aspnetcore-overview-introduction-getting - started-ec922e53bb97 (20.12.22)

81. Karabulut A. Understanding Clean Architecture and Domain-Driven Design (DDD). Internet resource: https://medium.com/bimar-teknoloji/understanding-clean-architecture-and-domain-driven-design-ddd-24e89caabc40

82. Мартнн Р. Чистая архитектура. Искусство разработки программного обеспечения. СПб.: Питер, 2018. -352 с.: ил. - (Серия «Библиотека программиста»)

83. Martin R.C. (Uncle Bob). The Clean Code Blog. The Clean Architecture. 2012. Internet resource: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (10.08.23)

84. Clean Architecture Layers – What they are and the Benefits. Transparity Ballard Chalmersa. Ssoftware Development company, UK.

https://ballardchalmers.com/resources/clean-architecture-layers-what-they-are-and-the-benefits/ (12.05.23)

85. Nitesh Malviya. Domain-Driven Design and Microservices. Published in The Startup, 2020. Intern. res.: https://medium.com/swlh/domain-driven-design-and-microservices-c62255790c3b (20.06.23)

86. პაპავაძე ს. დომენზე ორიენტირებული დიზაინის გამოყენება სასწავლო დაწესებულების მართვის სისტემაში. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 2(31), 2020. გვ. 32-37

87. Domain-Driven Design. Internet resource: https://www.dremio.com/wiki/domain-driven-design/ (1.08.23)

88. Evans E. Domain-Driven Design Tackling Complexity in the Heart of Software. Addison Wesley. ISBN 978-0321125217. 2004. 560 p. Internet resource: https://www.amazon.com/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215

89. Creating a Model for an Existing Database in Entity Framework Core. Internet resource: https://www.entityframeworktutorial.net/efcore/create-model-for-existing-database-in-ef-core.aspx#google_vignette (10.07.23)

90. Anderson R. Overview and Creating the Project. e-Tutorial. 2022. Internet resource: https://learn.microsoft.com/en-us/aspnet/web-api/overview/older-versions/using-web-api-1-with-entity-framework-5/using-web-api-with-entity-framework-part-1 (10.07.23)

91. Zang A. .NET Basics: ORM (Object Relational Mapping). Internet resource: https://www.telerik.com/blogs/dotnet-basics-orm-object-relational-mapping (10.07.23)

92. Kanjilal J. How to implement the Repository design pattern in C#. 2020. https://www.infoworld.com/article/3107186/how-to-implement-the-repository-design-pattern-in-c.html, (25.06.2021)

93. სურგულაძე გ., ბულია ი., თურქია ე. Web-აპლიკაციების დამუშავება მონაცემთა ბაზების საფუძველზე (ADO.NET, ASP.NET,

C#). თბილისი, სტუ, „ტექნიკური უნივერსიტეტი“, 2014, 189 გვ. http://gtu.ge/book/gia_sueguladze/SurgEka_ASP_NET.pdf

94. ბახტაძე თ. ობიექტზე ორიენტირებული პროგრამირების ენა C#. თბილისი, სტუ, „ტექნიკური უნივერსიტეტი“, 2006, 130 გვ. https://gtu.ge/book/Tengiz_bakhtadze_C_Sharpi.pdf

95. სურგულაძე გ. ვიზუალური დაპროგრამება C#_2010 ენის ბაზაზე. სტუ. თბ., სახელმძღ., 2011. -445 გვ., ბიბლ.ინდ. 681.03.06(02) 69. https://gtu.ge/book/GiaSurg_C_2010.pdf

96. Cohen N. What Is Performance Testing vs. Load Testing vs. Stress Testing? 2022. https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing (31.07.23)

97. Codd, E. F. A relational model of data for large shared data banks. (PDF). Communications of the ACM, v.13, N 6: 1970, pp.377-387. https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

98. Codd E.F. Further normalization of the database relational model. In Data Base Systems, Courant Computer Science Symposia 6. Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 65-98

99. Мартин Дж. Организация БД в вычислительных системах. Пер. с англ., -М., Мир, 1978. 472 ст.

100. Chen P. The Entity-Relationship Model: Toward a Unified View of data. ACM Trans.Database System. 1976, v.1, N 1., pp.9-36

101. Wang C., Wedekind H. (1975). Segment Synthesis in Logocal Data Base Design. IBM J. RSD 19, N1, January, pp.71-77.

102. Wedekind H., Surguladze G. (1996). Technology of Designing of Distributed Systems on the Basis of Objectoriented Programming. ISSN 021-7164, GTU, Tbilissi. pp.96-100, (in Georgian)

103. სურგულაძე გ. კორპორაციის ავტომატიზებული სამუშაო ადგილების ქსელის აგების ტექნოლოგია (პირველი ქართული ERP სისტემა). ISBN 978-9941-8-5109-4, მონოგრაფია. სტუ-ს „IT-კონსალტინგ სამეცნიერო ცენტრი". თბ., 2023, - 331 გვ.

104. Сургуладзе Г., Об одном подходе управления данными в реляционных системах банка данных. Труды ГПИ, №8 (190). Тбилиси, 1976.

105. Сургуладзе Г., Чачанидзе Г. Некоторые вопросы выбора оптимальных структур информационных данных при проектировании БД. Труды ГПИ, №8 (190). Тбилиси, 1976.

106. Chogovadze G., Surguladze G., Chachanidze G. (1977). Designing of Relational Databases and Issues of Communication in Them. Abstracts of the 1-st All-Union School-Seminar "Intellectual Information Banks". – Moscow. MEPhI-GPI. Sukhumi, 23.09-2.10.1977, pp.55-58 (in Russian)

107. Manage Indexes. MongoDb Compass Documentation. Inter.res., https://www.mongo¬db.com/docs/compass/current/indexes/ (15.05.22)

108. SurguladzeG., Kachibaia V., Kortua T. (1983). On the Choice of Acceptable normal forms for Logical Relational Databases. Transact. of GTU, “Techn. Cybernetics”, N 10(267), Tb.,pp.47-51, (in russ.)

109. ცერცვაძე მ,, კაიშაური თ., სურგულაძე გ. კოლექციათა-შორისი კავშირების დაპროექტება და ანალიზი NoSQL-ის MongoDB ბაზაში (ისტორია და რეალობა). სტუ-ს შრ.კრ., „მას", No 1(33), ტ.2, 2022 გვ.154-163

110. Database. Deploy a multi-cloud database. Internet resource: https://www.mongodb.com/atlas/database

111. Google Cloud Platform (GCP). Internet resource: https://www.mongodb.com/docs/atlas/reference/google-gcp/

112. ჩოგოვაძე გ., სურგულაძე გ., თოფურია ნ., ხარიტონაშვილი მ. ინფორმაციული საზოგადოება და ინტერდისციპლინური სწავლება ციფრული ტექნოლოგიების ბაზაზე. ISBN 978-9941-8-3338-0. მონოგრაფია. სტუ. „IT-კონსალტინგის სამეცნიერო ცენტრი”, თბ., 2021. -360 გვ.

113. სურგულაძე გ., ბულია ი. კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. მონოგრ., ISBN 978-9941-20-165-3. სტუ. „ტექნიკური უნივერსიტეტი“, თბ., 2012. – 324 გვ.

114. Machine Learning for Education: Benefits and Obstacles to Consider in 2020. ODSC - Open Data Science. 2019. https://medium.com/@ODSC/machine-learning-for-education-benefits-and-obstacles-to-consider-in-2020-4e8008dbd732 (16.07.23)

115. Artificial Intelligence (AI) vs. Machine Learning. Internet resource: https://ai.engineering.columbia.edu/ai-vs-machine-learning/ (30.06.2023)

116. ODSC. Machine Learning for Education: Benefits and Obstacles. Machine Learning for Education: Benefits and Obstacles to Consider in 2020. https://medium.com/@ODSC/machine-learning-for-education-benefits-and-obstacles-to-consider-in-2020-4e8008dbd732, უკანასკნელად იქნა გადამოწმებული - 25.06.2021.

117. Schmelzer R. AI Applications In Education. forbes.com. 2019. https://www.forbes.com/sites/cognitiveworld/2019/07/12/ai-applications-in-education/?sh=5d17fb7262a3 (25.06.2021).

118. Plitnichenko L. 5 Main Roles Of Artificial Intelligence In Education. 2020. elearningindustry.com https://elearningindustry.com/5-main-roles-artificial-intelligence-in-education, (25.06.2021)

119. Daria R. AI and Education or How to Create an Advanced Artificial Intelligence Program. 2021. https://www.cleveroad.com/blog/ai-in-education-or-what-advantages-of-artificial-intelligence-in-education-you-can-gain. (25.06.2021)

120. *McGuinness W.* Benefits and the Limitations of Machine Learning in Education. 2018. https://www.gettingsmart.com/2018/02/the-benefits-and-the-limitations-of-machine-learning-in-education (25.6.2021)

121. Pedamkar P.. Uses of Machine Learning. https://www.educba.com/uses-of-machine-learning/, უკანასკნელად იქნა გადამოწმებული - 25.06.2021.

122. New Oriental Education & Technology Group Inc. ADR. Internet resource: https://www.marketwatch.com/investing/stock/edu/company-profile

123. Why China Is Using A.I. in Its Classrooms. 2019. Internet resource: https://www.wsj.com/podcasts/whats-news/why-china-is-using-ai-in-its-classrooms/9e1d74c0-5a5a-4ac4-9c1b-48f1078e66d7

124. Occidental college, California. Computer Science. Machine Learning. Internet resource: https://oxy.smartcatalogiq.com/en/2023-2024/catalog/course-descriptions/comp-computer-science/ (5.07.23)

125. Brownlee J. Your First Machine Learning Project in Python Step-By-Step. 2020. https://machinelearningmastery .com/machine-learning-in-python-step-by-step/ (25.06.2021)

126. Commonly used Machine Learning Algorithms (with Python and R Codes). https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ (25.06.2021)

127. Luashchuk A. Why I Think Python is Perfect for Machine Learning and Artificial Intelligence. 2019. https://towardsdatascience. com/8-reasons-why-python-is-good-for-artificial-intelligence-and-machine-learning-4a23f6bed2e6 (25.06.2021)

128. Ryabtsev A. Reasons Why Python is Good for AI and ML. 2022. Internet resource: https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/#utm_source=medium&utm_medium=towardsdatascience.com&utm_campaign=8%20reasons%20why%20python%20is%20good%20for%20ml%20and%20ai&utm_content=why%20Python%20is%20good%20for%20ML%20and%20AI, (12.07.23)

129. White J., Bottorff C., Watts R. Indeed Review 2023: Features, Pros & Cons. 2023. Internet resource: https://www.forbes.com/advisor/business/software/indeed-review-for-employers/

130. მაჩალაძე ო., სურგულაძე გ. IT ინფრასტრუქტურის მართვა საგანმანათლებლო დაწესებულებისთვის. სტუ-ს შრ.კრ., „მას", No 1(33), ტ.2, 2022 გვ.113-117

131. სურგულაძე გ., ურუშაძე ბ. საინფორმაციო სისტემების მენეჯმენტის საერთაშორისო გამოცდილება (BSI, ITIL, COBIT). ISBN 978-9941-20-458-6. სტუ. „ტექნიკ.უნივერს.“, თბ., 2014 -345 გვ.

132. Information Technology Infrastructure. IBM. 2022. Internet resource: https://www.ibm.com/topics/infrastructure (25.08.22)

133. Incident management for high-velocity teams. Atlassian. 2022. Internet resource: https://www.atlassian.com/itsm/incident-management (28.08.22)

134. Jira Service Management (JSM). Atlassian. 2022. Internet resource: https://support.atlassian.com/jira-service-management-cloud/docs/what-is-jira-service-management/ (31.08.22)

135. ITSM for high-velocity teams. Atlassian. 2021. Internet resource: https://www.atlassian.com/itsm (2.10.22)

136. Service Desk – Benefits. Itarian. 2020. Internet resource: https://www.itarian.com/service-desk/service-desk-definition.php (5.10.22)

137. BSI-Standard 100-2: IT-Grundschutz-Vorgehensweise. (BSI) Godesberger Allee 185-189, 53175 Bonn, 2008/2013

138. COBIT: An ISACA Framework. Effective IT Goverance. 2019. Internet resource: https://www.isaca.org/resources/cobit (15.05.23)

139. Maya G. ITIL Enterprise Architecture. 2021. Internet resource: https://www.itsm-docs.com/blogs/itil-concepts/enterprise-architecture (20.07.21)

140. ITIL v3. CCTA (Central Computer and Telecommunications Agency). UK. 2023. Internet resource: https://en.wikibooks. org/wiki/ITIL_v3_ Information_Technology_Infrastructure_Library)/Introduction

141. Kelley K. What is ITIL: Essential Guide to ITIL V4 Processes and Framework. 2023. Internet resource: https://www.simplilearn.com/tutorials/itil-tutorial/what-is-itil

142. ITIL. Office of Government Commerce (UK). 13 July 2009. Archived from the original on 9 September 2009. Retrieved 19 August 2009

143. ITIL 4. IT Process Wiki. Archived from the original on 14 February 2022. Retrieved 11 February 2022

144. Pros and Cons of Python Programming Language. 2022. Internet resource: https://www.pixelcrayons.com/blog/python-pros-and-cons/ (15.12.22)

145. Raman. Boosting in Machine Learning | Boosting and AdaBoost. 2023. Internet resource: https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/

146. Willems K. Python Machine Learning: Scikit-Learn Tutorial. 2019. Internet resource: https://www.datacamp.com/community/tutorials/machine-learning-python

გადაეცა წარმოებას 15.07 2023. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 16,25. ტირაჟი 50 ეგზ.

Verba volant, scripta manent

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრი~,
თბილისი, მ.კოსტავას 77

ISBN 978-9941-8-5443-9