ლილი პეტრიაშვილი, გია სურგულაძე

მონაცემთა მენეჯმენტის თანამედროვე ტექნოლოგიები

(Oracle, MySQL, MongoDB, Hadoop)

„სტუ-ს IT-კონსალტინგის ცენტრი“

საქართველოს ტექნიკური უნივერსიტეტი

ლია პეტრიაშვილი, გია სურგულაძე

მონაცემთა მენეჯმენტის თანამედროვე ტექნოლოგიები

(Oracle, MySQL, MongoDB, Hadoop)

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

თბილისი
2017

უაკ 004.65

განხილულია მართვის საინფორმაციო სისტემებში განსაკუთრებით დიდი მონაცემების დამუშავების და შენახვის თანამედროვე მეთოდები და ინსტრუმენტული საშუალებები. მონაცემთა კლასიკური რელაციური ბაზების: Oracle და MySQL -ის გვერდით წარმოდგენილია NoSQL ოჯახის ერთ-ერთი პოპულარული წევრი - MongoDB. მოცემულია მათი დაპროექტების საწყისები, ამ პროცესების ავტომატიზაციის თეორიულ-პრაქტიკული ინსტრუმენტული საშუალებები ობიექტ-როლუური მოდელირების საფუძველზე. შედარებულია რელაციური და არარელა-ციური ბაზების ეფექტურად გამოყენების მახასიათებლები მართვის საინფორმაციო სისტემებში. წიგნში განსაკუთრებული ყურადღება გამახვილებულია Rel_DB და NoSQL ბაზების სტრუქტურებზე და კომპონენტებზე, მომხმარებლის სამუშაო გარემოზე და დიდ მონაცემთა (Big Data) ადმინისტრირების პრობლემებსა და ტექნოლოგიებზე (Hadoop).

დამხმარე სახელმძღვანელო განკუთვნილია ინფორმატიკისა და მართვის საინფორმაციო სისტემების სპეციალობის სტუდენტებისა და მაგისტრანტებისთვის, ასევე, აღნიშნული საკითხებით დაინტერესებული მკითხველისთვის.

რეცენზენტები: - პროფ. ე. თურქია (სტუ)
- ასოც.პროფ. დ. გულუა (თსუ)

რედკოლეგია:

ა. ფრანგიშვილი (თავმჯდომარე), მ. ახობაძე, გ. გოგიჩაიშვილი, ზ. ბოსიკაშვილი, ე. თურქია, ნ. ლომინაძე, ჰ. მელაძე, თ. ობგაძე, რ. სამხარაძე, გ. სურგულაძე (რეადაქტორი), გ. ჩაჩანიძე, ზ. წვერაიძე

ISBN 978-9941-27-176-2

შინაარსი

შესავალი

მონაცემთა რელაციური ბაზების თეორია სათავეს 70-იანი წლებიდან იღებს, როდესაც გამოჩნდა ედგარ კოდის სტატიები მონაცემთა რელაციური მოდელების შესახებ [1,2]. მანამდე კი აშშ-ის და ევროპის განვითარებული ქვეყნების უნივერსიტეტებში ფართოდ გამოიყენებოდა და წარმოებაში ინერგებოდა არარელაციური ტიპის ბაზები, ე.წ. იერარქიული და ქსელური ბაზები [3,4,5]. მათზე საკმაო ლიტერატურაა შექმნილი, ამიტომ ჩვენ აქ მათ არ განვიხილავთ.

80-იანი წლებიდან დაიწყო სწრაფი განვითარება და 21-ე საუკუნიდან ინფორმაციული ტექნოლოგიების ბაზარზე უკვე გამეფდა რელაციური მონაცემთა ბაზის სისტემები: Oracle, Ms SQL Server, MySQL, Ms Access, Sybase, DB/2, PostgreSQL და სხვ. [5,6]. ასეთ სისტემებს თამამად შეიძლება ვუწოდოთ კლასიკური მონაცემთა ბაზები, რომელთა როლი და გამოყენების არეალი დღესაც დიდია.

რაც შეეხება სიახლეს ამ სფეროში და ჩვენი წიგნის ძირითად მიზანს, ესაა „ძველი“ ახალი ტიპის, ანუ არარელაციური მონაცემთა ბაზების, ე.წ. NoSQL ტიპის ბაზების საფუძვლების გაცნობა, მათი შედარება რელაციურ მონაცემთა ბაზებთან. ანალიზის საფუძველზე კი გარკვეული დასკვნის გაკეთება შეიძლება, თუ როდის უნდა გამოვიყენოთ რელაციური და როდის NoSQL ტიპის ბაზები, საინფორმაციო სისტემების ეფექტიანი მუშაობის ორგანიზების თვალსაზრისით.

წიგნის 1-ელ თავში განხილულია მონაცემთა კლასიკური და ახალი ბაზების თეორიული ასპექტები. მათი ძირითადი დადებითი და უარყოფითი მხარეები პრაგმატული თვალსაზრისით. მახასიათებლების შედარება. SQL-ენის ოჯახის წარმომადგენლად აქ განიხილება Oracle და MySQL ბაზები.

NoSQL ბაზების იდეოლოგია განხილულია დოკუმენტ-ორიენტირებული MongoDB ბაზის მაგალითზე. აქვე მოკლედ არის განხილული ახალი ჰიბრიდული ტიპის, NewSQL მონაცემთა ბაზები, დიდ მონაცემთა (BigData) ახალი ტექნოლოგია Hadoop-ის სახით და მისი აქტუალობა დღეისათვის [15].

მე-2 თავი მთლიანად ეხება NoSQL ტიპის ბაზების კონცეფციის საფუძვლების გაცნობას. განხილულია მათი ფუნქციონირების ძირითადი ასპექტები და პრინციპები. კერძოდ, ნაშრომში დეტალურად არის წარმოდგენილი მონაცემთა ბაზების ვერტიკალური და ჰორიზონტალური ორგანიზაცია, მონაცემთა შენახვის ტიპები: „გასაღები-მნიშვნელობა“, „დოკუმენტ-ბაზირებული“, „სვეტებზე ბაზირებული“ და „გრაფზე ბაზირებული“.

განსაკუთრებული ყურადღება აქვს დათმობილი მონაცემთა მთლიანობის (Data integrity) და ტრანზაქციათა იზოლირების დონეების გაცნობას. მნიშვნელოვანია აგრეთვე ACID პრინციპებისა და CAP სამკუთხედის თეორემის განხილვა.

მე-3 თავი ეხება მონაცემთა საცავების ტექნოლოგიას. განხილულია მისი სტრუქტურა, ფუნქციური ბლოკების დანისნულება. განსაკუთრებით გამახვილებულია ყურადღება მონაცემთა ონლაინ დამუშავების OLAP ტექნოლოგიაზე. შემდეგ კი, მე-4 თავში წარმოდგენილია Data Mining - მონაცემთა ინტელექ-ტუალური ანალიზის ინსტრუმენტი. ორივე მათგანი განხილულია Oracle მონაცემთა ბაზების სისტემის კონტექსტში.

მე-5 თავში გადმოცემულია MongoDB ბაზის სამუშაო გარემო, გრაფიკული ინტერფეისი RoboMongo პლატფორმის სახით. აღწერილია მონაცემთა ბაზის აგებისა და მასთან მუშაობის კონკრეტული მაგალითები.

I თავი
მონაცემთა ბაზების დაპროექტების კლასიკური და ახალი ტექნოლოგიები

მონაცემთა ბაზების მართვის სისტემა (მბმს, Database Management System, DBMS) – არის პროგრამული უზრუნველყოფა (Software), რომლის დანიშნულებაცაა ამ ბაზის მონაცემთა გამოყენება და მოდიფიკაცია მრავალმომხმარებლურ რეჟიმში.

წინამდებარე თავში განვიხილავთ მონაცემთა რელაციურ ბაზას, როგორც კლასიკური ტიპის ბაზას, რომელზეც მოთხოვნილება საკმაოდ მაღალია. ობიექტურ ბაზებს, ობიექტ-რელაციურ ბაზებს და არარელაციურ NoSQL ბაზებს. ამ უკანასკნელის თვალსაზრისით ჩვენ შევეხებით დოკუმენტ-ორიენტირებულ და გრაფულ-ორიენტირებულ მონაცემთა ბაზებს.

1.1. რელაციური მონაცემთა ბაზა და SQL ენა

რამდენიმე ათეული წელია, რაც მართვის ავტომატიზებულ სისტემებში ინფორმაციის შენახვისა და გადამუშავების ყველაზე ეფექტური და ფართოდ გამოყენებადი მონაცემთა ბაზები ეფუძნება ედგარ კოდის რელაციურ მოდელს, Oracle, Ms SQL Server, SyBase, MsAccess, MySQL და სხვ. [1-3]. ასეთი ბაზებია დანერგილი დღეს სახელმწიფო და კერძო სტრუქტურების უმრავლეს ორგანიზაციასა და დიდ კორპორაციაში.

მონაცემთა რელაციურ მოდელს საფუძვლად უდევს სიმრავლური მიმართების მათემატიკური ცნება. ამავე დროს, მიმართება მოცემულ მოდელში შეიძლება იყოს წარმოდგენილი ცხრილის სახით, სადაც ცხრილის სვეტები მიმართების თვისებებია ან ატრიბუტები. ავიღოთ n სიმრავლეთა $A_1, A_2, \ldots, An$ ერთობლიობა. R მიმართებას ამ სიმრავლეებზე ეწოდება დეკარტული ნამრავლის

ქვესიმრავლე, რომელიც ($a_1, a_2, \ldots, a_n$) სახისაა, სადაც $a_i \in A_i$, $i=(1,n)$. ამავე დროს, მიმართების მათემატიკური განსაზღვრებიდან შეგვიძლია ჩავწეროთ:

$$R \subseteq A_1 \times A_2 \times \ldots \times A_n.$$

სიმრავლეთა $A_1, A_2, \ldots, An$ ერთობლიობა არის R მიმართების განსაზღვრის არე, ხოლო A_i სიმრავლეს უწოდებენ დომენს. R მიმართების ელემენტებს უწოდებენ კორტეჟებს ან ამონაკრებებს.

რელაციური მიმართების ცხრილი ორგანზომილებიანია, რომლის სტრიქონები შეესაბამება ატრიბუტების მნიშვნელობების კორტეჟს, ხოლო სვეტები - ატრიბუტებს. აქვე აღვნიშნოთ, რომ ატრიბუტი განსაზღვრულია დომენის სიმრავლეზე. დავუშვათ, C_i არის ატრიბუტი, მაშინ $C_1 \subseteq A_1$, $C_2 \subseteq A_2, \ldots, C_n \subseteq n_1$. R მიმართების სიმძლავრე (კარდინალური რიცხვი) განისაზღვრება კორტეჟების რაოდენობით.

1.1 ნაზაზზე მოცემულია რელაციური მოდელის მაგალითი ცხრილური ფორმით.

აკადემიური–ჯგუფი

ჯგუფის ნომერი	სპეციალობა	სპეციალობის შიფრი	სტუდენტების რაოდენობა	სექტორი
108435	მას	2202	14	ქართული
608536	კაქს	2201	16	ქართული
108739	ამტს	2101	11	რუსული
108638	სსტ	1906	12	ინგლისური
608534	მას	2202	15	ქართული

ნახ.1.1. მონაცემთა რელაციური ცხრილი

რელაციური მიმართების სახელია **აკადემიური_ჯგუფი.** თუ მას აღვნიშნავთ R-ით, მაშინ გვექნება:

$$R \subseteq A_1 \times A_2 \times A_3 \times A_4 \times A_5.$$

სადაც სადაც A_1 დომენი არის ჯგუფის ნომერი, A_2 დომენი - სპეციალობა, A_3 - სპეციალობის შიფრი, A_4 - სტუდენტების რაოდენობა, A_5 სექტორი.

დომენები შეიცავს შემდეგ მნიშვნელობებს:

A_1 = {108435, 608536, 108739, 108638, 608534};
A_2= {მას, კქს, ამს, სსტ};
A_3= {2202, 2201, 2101, 1906};
A_4= {14, 16, 11, 12, 15};
A_5= {ქართული, ინგლისური, რუსული}.
და ა/შ.

მონაცემთა ბაზების თეორიაში განიხილავენ მონაცემთა მოდელირების სამ - კონცეპტუალურ, ლოგიკურ და ფიზიკურ დონეებს. აგრეთვე მონაცემთა სტრუქტურების ოპტიმიზაციის საკითხებს ნორმალურ ფორმათა თეორიის საფუძველზე [4].

მონაცემთა რელაციურ ბაზებზე და მათ მოდელებზე საკმაო ლიტერატურული წყაროებია ქართულ ენაზეც [2-6], ამიტომ აქ ჩვენ მათ დეტალურად არ განვიხილავთ, ვინაიდან წიგნის მიზანი უფრო ახალი, არარელაციური NoSQL ბაზის გაცნობაა.

ამჯერად ჩვენ განვიხილავთ მონაცემთა რელაციური ბაზების ოჯახის მთავარ ენას SQL-ს, მის ძირითად სტრუქტურასა და ოპერაციათა სინტაქსს.

შემდეგ, საილუსტრაციო მაგალითის სახით განვიხილავთ მონაცემთა რელაციური ბაზების მართვის ერთ-ერთ პოპულარულ სისტემას - MySQL -ს.

განვიხილოთ SQL (Structured Query Language) მოთხოვნების სტრუქტურირებადი ენა, რომელსაც იყენებს თითქმის ყველა რელაციური მონაცემთა ბაზის მართვის სისტემა.

აქ უნდა ვახსენოთ ასევე ორი ენა - მონაცემთა აღწერის ენა (DDL – Data Definition Language) და მონაცემთა მანიპულირების ენა

(DML – Data Manipulation Language). პირველი გამოიყენება მონაცემთა ბაზის ობიექტების გამოცხადების (შექმნის, მოდიფიკაციის) დროს, ხოლო მეორე - ამ ობიექტების დამუშავებისას [5].

SQL ენა – მოთხოვნების სტანდარტული ენაა მონაცემთა რელაციურ ბაზებში. იგი შეიქმნა ე. კოდის რელაციური ალგებრის ენის საფუძველზე. SQL ენის დანიშნულებაა ბაზის ცხრილებზე (Tables) ოპერაციების ჩატარება: შექმნა, წაშლა, მოდიფიკაცია, მონაცემთა ამორჩევა, ცვლილება, დამატება განსაზღვრული პირობებით. ცხრილებთან მუშაობისას იქმნება წარმოდგენები (Views), ანუ ესეც კომბინირებული ცხრილია არსებულების საფუძველზე, რომლებშიც ატრიბუტების (სვეტების) და კორტეჟების (სტრიქონების, ჩანაწერების) შედგენილობა დამოკიდებულია შემოსულ მოთხოვნაზე.

როგორც ცნობილია, ყველა რელაციურ ბაზის სისტემას გააჩნია მოთხოვნების აგების როგორც სკრიპტული ფორმა, ასევე გრაფიკული ინსტრუმენტული საშუალება. იგი მნიშვნელოვნად ამარტივებს მომხმარებელთა მუშაობას ბაზასთან, მოთხოვნების ფორმირების პროცესის თვალსაზრისით.

SQL ენის ოპერატორები პირობითად შეიძლება დავყოთ სამ ძირითად ჯგუფად:

- SELECT ოპერატორი;
- მონაცემთა მანიპულირების ოპერატორები;
- მონაცემთა აღწერის ოპერატორები.

➢ SELECT ოპერატორის სინტაქსი ასეთია:

SELECT [ALL| DISTINCT] <ველების სია>|*
FROM <ცხრილების სია>
[WHERE <პირობის-პრედიკატი ამორჩევისათვის ან შეერთებისათვის>]
[GROUP BY <შედეგის ველების სია>]
[HAVING <პირობის-პრედიკატი ჯგუფისთვის>]
[ORDER BY <ველების სია შედეგის მოწესრიგებისთვის>];

სადაც:

ALL *- ყველა სტრიქონია (დასაშვებია დუბლირება);*

DISTINCT *- უნიკალური სტრიქონები (არაა დუბლირება);*

WHERE **–** *შედარების პრედიკატები:* **(=, <>, >, >=, <, <=)**

GROUP BY **–** *მოიცემა ველების სია დაჯგუფების მიზნით;*

HAVING *- ახდენს ჩანაწერების ფილტრაციას ჯგუფში;*

ORDER BY *- ალაგებს შედეგს ველების სიის მიხედვით*

და ა.შ.

- მონაცემთა მანიპულირების ენაზე (DML) მოთხოვნები იწერება ბაზაში კორტეჟების დასამატებლად, წასაშლელად და შესაცვლელად. მათი სინტაქსი ასეთია:

INSERT INTO *ცხრილის_სახელი [(<სვეტების სია>)]*
VALUES *(<მნიშვნელობათა სია>)*

DELETE FROM *<ცხრილის_სახელი>* ***[WHERE*** *<ამორჩევის_პირობა>]*

თუ ამორჩევის პირობა არაა მითითებული, მაშინ ცხრილში წაიშლება ყველა ჩანაწერი.

მონაცემთა განახლების ოპერაცია *UPDATE* შედგება სამი ნაწილისგან:

- ***UPDATE*** *წინადადება,* მიუთითებს განსაახლებელ ცხრილს;
- ***SET*** *წინადადება,* იძლევა განსაახლებელ მონაცემებს;
- ***WHERE*** *არააუცილებელი კრიტერიუმი,* ზღუდავს ჩანაწერთა რაოდენობას, რომელზეც უნდა იმოქმედოს განახლების მოთხოვნამ.

- მონაცემთა სქემის აღწერის ენა (SDL, Schema Definition Language) ან (DDL) - არის SQL-ის ინსტრუქციები, რომლებიც უზრუნველყოფს მონაცემთა ბაზის სტრუქტურის ელემენტების შექმნას და მოდიფიცირებას.

- ცხრილის შექმნის ოპერატორს აქვს შემდეგი სინტაქსი:

CREATE TABLE <ცხრილის სახელი>
(<სვეტის სახელი> <მონაცემთა ტიპი> [NOT NULL]
[,<სვეტის სახელი> <მონაცემთა ტიპი> [NOT NULL]]…)

- ცხრილის წაშლის ინსტრუქცია:

DROP TABLE <ცხრილის სახელი>

- ცხრილის სტრუქტურის მოდიფიკაცია (ველების დამატება, წაშლა, მათი ტიპების შეცვლა):

ALTER TABLE <ცხრილის სახელი>
MODIFY | ADD| DROP <ველის სახელი> [<მონაცემთა ტიპი>]

- ცხრილის შექმნის შემდეგ შესაძლებელია ინდექსების შექმნა CONSTRAINT წინადადებით:

CREATE [UNIQUE] INDEX <ინდექსის სახელი>
ON <ცხრილის სახელი>
(<სვეტის სახელი> [ASC | DESC]
[, <სვეტის სახელი> [ASC | DESC]…)

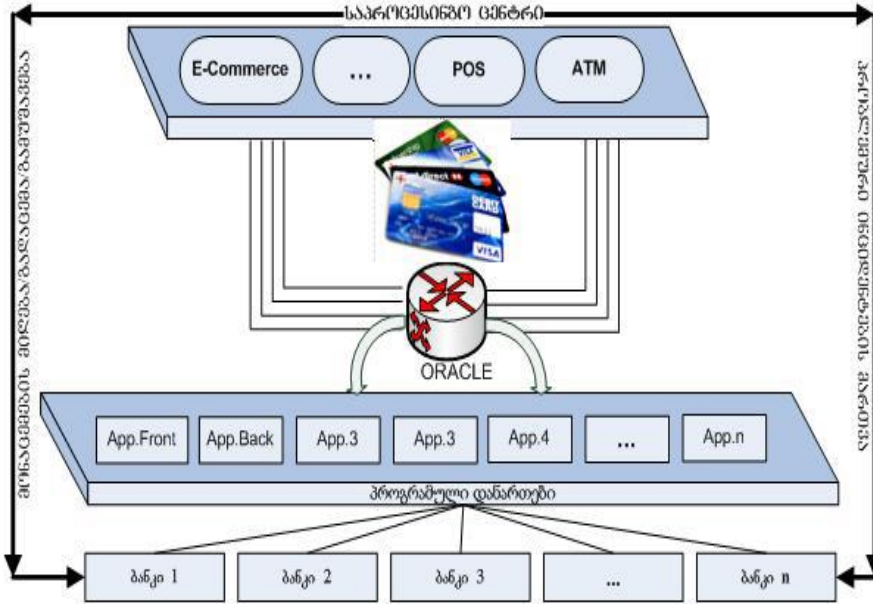
ინდექსების შექმნა ცხრილის ერთი ან არამდენიმე ველისთვის უზრუნველყოფს მოთხოვნის შესაბამისი საძიებო ოპერაციების დაჩქარებას. ინდექსების წაშლა ხდება ინსტრუქციით:

DROP INDEX <ინდექსის სახელი> ON<ცხრილის სახელი>

1.2. Oracle მონაცემთა ბაზის მართვის სისტემა

1.2.1. ORACLE სისტემის არქიტექტურა

თანამედროვე საინფორმაციო ტექნოლოგიებში ფართოდაა გავრცელებული მონაცემთა ბაზები და მათთან მომუშავე პროგრამები. პრაქტიკულად, უმეტესი პროგრამული დანართები (Application) მონაცემთა ბაზებთან სამუშაოდ იქმნება (ნახ.1.2). აქედან გამომდინარე მომხმარებელს ან პროგრამისტს, თავის პროფესიულ საქმიანობაში, ადრე თუ გვიან, მოუხდება მონაცემთა ბაზის გამოყენება, ან ბაზასთან სამუშაო რაიმე დანართის დაპროექტება [12].



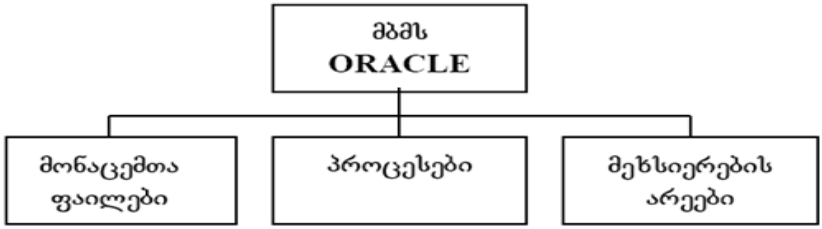
ნახ.1.2.

იმისათვის, რომ შეიქმნას რეალური მონაცემთა ბაზა კონკრეტული ამოცანების შესასრულებლად, საჭიროა მისი

დაპროექტება. პროექტირების პროცესში მიიღება გადაწყვეტილებები ბაზაში შესანახი მონაცემებისა და მათი ორგანიზაციის შესახებ. შემდეგ კი შერჩეულ უნდა იქნას პროექტის სარეალიზაციო რომელიმე მონაცემთა ბაზის მართვის სისტემა ან დაპროგრამების რომელიმე ენა [10,11].

დღეისათვის ერთ–ერთი ყველაზე მძლავრი და მოქნილი მონაცემთა ბაზების მართვის სისტემაა Oracle, რომლის ეფექტურობა მით უფრო მაღალია რაც მეტი კომპიუტერია მის ქსელში გაერთიანებული (1000 ან მეტი). განვიხილოთ მბ–ორაკლეს ძირითადი ელემენტები:

ეგზემპლიარი - ესაა პროცესებისა და კომპიუტერის მეხსიერების არეები, რომლებიც უზრუნველყოფს მონაცემებთან მიმართვას. 1.3 ნახაზზე მოცემულია ამ მონაცემთა ბაზის სისტემის ძირითადი კომპონენტები.



ნახ.1.3. Oracle-ს სტრუქტურა

მონაცემთა ფაილებში ინახება საწყისი და გაანგარიშებადი რელაციურად დაკავშირებული) მონაცემები, რომლებიც მოთავსებულია ცხრილებში, ინდექსურ ფაილებსა და წარმოდგენებში.

როგორც ყველა თანამედროვე მონაცემთა ბაზების სისტემა, **Oracle** უზრუნველყოფს მონაცემთა ბაზაში ე.წ. შენახვადი

პროცედურების (Stored_Procedures) განთავსებას, რომლებიც მაკროსების და ქვეპროგრამების სახითაა წარმოდგენილი.

პროცესები ოთხი კატეგორიით განიხილება:

- მომხმარებელთა მოთხოვნების დამუშავების პროცესები;
- მონაცემთა ჩაწერის პროცესები ფაილებში;
- ტრანზაქციების ჟურნალში რეგისტრაციის პროცესები;
- მონაცემთა ბაზის მუშაობის კონტროლის პროცესები.

მომხმარებელთა პროგრამული დანართები მაგნიტური დისკოებიდან ამოარჩევს და მეხსიერების არეებში განალაგებს მისთვის საჭირო მონაცემთა და პროგრამულ ფრაგმენტებს, ასევე გარდაქმნის და ჩაწერს მათ უკან, დისკზე. რაცლე-ს პროცესები უზრუნველყოფს დისკოებთან მეხსიერების არეების მიმაგრებას და მათ მართვას.

Oracle–ს პროცესები იყოფა ძირითად და არააუცილებელ პროცესებად. პირველი მოიცავს ფუნქციებს, რომლებიც უშუალოდ მონაცემთა ბაზის მუშაობისთვისაა აუცილებელი. მეორე კი – ისეთ ფუნქციებს, რომლებიც დამხმარე ხასიათისაა, მაგალითად, მწარმოებლურობის გაზრდა, არქივირება, მონაცემთა აღდგენა, პარალელური მოთხოვნების შესრულება და ა.შ.

Oracle-ს ეგზემპლიართან მიმართებაში განიხილავენ მის ოთხ ძირითად პროცესს:

1) სისტემის მონიტორი (SMON). მისი დანიშნულებაა Oracle - ეგზემპლიარის მზადყოფნის მდგომარეობის უზრუნველყოფა. იგი გააქტიურდება სამ შემთხვევაში:

- Oracle ეგზემპლიარის ამუშავებისას აღმოჩნდება, რომ მისი წინა სეანსი არ იყო სწორად დასრულებული (shutdown -მსგავსად) და დარჩენილია დაუმთავრებელი ტრანზაქციები ან ძველი მონაცემებისგან გაუწმენდავი მეხსიერების არეები;

- ეგზემპლიარის მუშაობის პროცესში SMON პერიოდულად ამოწმებს, ხომ არაა დარჩენილი დროებითი, არასაჭირო სეგმენტები.

მათი წაშლით თავისუფლდება ოპერატიული მეხსიერება ახალი მოთხოვნებისათვის. SMON უზრუნველყოფს აგრეთვე ფრაგმენტაციის ამოცანის გადაწყვეტასაც. პარალელური სერვერის გარემოში კომპიუტერის ან ეგზემპლიარის მუშაობის ავარიულად დასრულების შემდეგ საჭიროა ამ ეგზემპლარის აღდგენა. SMON-ს აქვს ასეთი საშუალება, იგი სპეციალურ მონაცემთა ბაზის ფაილში ინახავს ეგზემპლარის მუშაობის პროცესის ისტორიას;

- სისტემის მონიტორი გააქტიურდება, თუ მას მიმართავს სხვა პროცესები. მაგალითად, მონაცემთა ბაზაში ინფორმაციის ჩაწერის პროგრამას სჭირდება მეხსეიერების სეგმენტები და მან ეს უნდა მიიღოს SMON-ის დახმარებით.

2) პროცესების მონიტორი (PMON). იგი ემსახურება სისტემის გაწმენდას პროცესების დამთავრების შემდეგ. კერძოდ მას შეუძლია:

- პროცესის იდენტიფიკატორის წაშლა;
- აქტიური ტრანზაქციების ცხრილიდან შესაბამისი სტრიქონების წაშლა;
- პროცესთან დაკავშირებულ ბლოკირებათა მოხსნა;
- პროცესთან დაკავშირებული კეშ-ელემენტების მოხსნა.

3) მონაცემთა ბაზის ჩაწერის პროცესი (DBWR). ეს ინსტრუმენტი ემსახურება კეშ-მეხსიერების ბუფერში მოდიფიცირებული ჩანაწერების გადაგზავნას ვინჩესტერზე შესაბამის მონაცემთა ფაილში. ბუფერის გათავისუფლება ხელს უწყობს ახალი მოთხოვნებისათვის მეხსიერების გამოყოფას. სისტემაში განსაზღვრული უნდა იყოს წინასწარ, თუ რომელი მონაცემთა ფაილები იქნეს დიდხანს დატოვებული ბუფერში (მაგალითად, ისეთი მონაცემები, რომელთა გამოყენების სიხშირეც მაღალია) და რომელი გადაიგზავნოს დისკზე (ერთჯერადი ან იშვიათად გამოყენების ფაილები). სწორედ DBWR -ის ფუნქციაა განსაზღვროს, თუ რომელი ფაილები გადაწეროს ბუფერიდან დისკზე.

4) ჟურნალის ჩაწერის პროცესი (LGWR). მონაცემთა ბაზის მიერ ხდება შესრულებული ტრანზაქციების შესახებ ინფორმაციის დაფიქსირება განახლების ჟურნალის ბუფერში. LGWR-ს საშუალებით ეს მონაცემები ჩაიწერება დისკზე განახლების ჟურნალის ფაილის სახით. ასეთი ჟურნალის ფაილი საჭიროა მონაცემთა ფაილის აღსადგენად მისი სრული დაზიანების შემთხვევაში. ამგვარად, ეს ინსტრუმენტი უზრუნველყოფს სისტემის საიმედოობას.

1.2.2. Oracle მონაცემთა ბაზის ობიექტები

მონაცემთა ბაზის ობიექტებია: ცხრილები, ინდექსები, წარმოდგენები, სინონიმები, მიმდევრობითობები, დანაყოფები, კლასტერები,ფუნქციები, ტრიგერები, პროცედურები, პაკეტები, მომხმარებელთა მონაცემთა ტიპები, ცხრილური სივრცეები და შეზღუდვები.

1.2.2.1. ცხრილები (Tables)

მონაცემთა ბაზაში ინფორმაციის შენახვის ძირითადი კომპონენტი არის ცხრილი. იგი შედგება რეალური ობიექტის (არსის) ატრიბუტების (ველების) და სტრიქონებისაგან (ჩანაწერებისაგან). ატრიბუტთა მნიშვნელობის (მონაცემები) ტიპები Oracle-ბაზაში მრავალფეროვანია, რიცხვითი, სტრიქონული, თარიღის, ლოგიკური, აუდიო-ვიზუალური და სხვ. [10].

მაგალითად:

- char(n) - სიმბოლური ტიპის მუდმივი სიგრძით (n);
- varchar2(n) - სიმბოლური ტიპის ცვლადი სიგრძით (n);
- number(n,m) - რიცხვითი ტიპის ცვლადი სიგრძით;
- date - თარიღისა და დროის ტიპი;
- raw(n) - ნებისმიერი ტიპის ცვლადი სიგრძით (2000 ბაიტამდე);
- rowid - ორობითი ტიპის (6 ბაიტი სტრიქონზე);

ახლა განვიხილოთ ცხრილების შექმნისა და მასთან მუშაობის რამდენიმე კონკრეტული მაგალითი. დავუშვათ, რომ გვაქვს სტუდენტების მონაცემთა ბაზა უნივერსიტეტში. მისთვის დამახასიათებელი ატრიბუტებისა და ცხრილის აღწერას ექნება შემდეგი სახე:

```
create table Students
      (St_ID number(4) not null,
       F_Name varchar2(15),
       L_Name varchar2(20),
       Birth_Date date,
       Faculty varchar2(40),
       Course number(1),
       Salary number(5,2),
       Adress varchar2(30),
       Hobbies varchar2(50) )
storage (
  initial 20K next 10K
  pctfree 15 pctused 50
  minextents  1 maxextents 100
  pctincrease 0 ) ;
```

ამ მაგალითში გარდა ობიექტის „სტუდენტი" ატრიბუტების აღწერისა: დასახელებებით, ტიპებითა და სიგრძეებით, მოთავსებულია ექსტენტის პარამეტრები.

ექსტენტი არის ცხრილის მონაცემებისთვის წინასწარ გამოყოფილი შესანახი ადგილი. იგი ბაიტებში იზომება. ფრაზა storage შეიცავს 5 პარამეტრს მოცემული ცხრილისათვის. თუ ის არაა მოცემული, მაშინ სისტემა თვითონ იღებს ამ მნიშვნელობებს გამოუცხადებლად.

ჩვენ შემთხვევაში Student-ცხრილის საწყისი ექსტენტი 20 კილობაიტია, ხოლო ყოველი შემდეგი ექსტენტი 10 კილობაიტი. minextents და maxextents არის ექსტენტების მინიმალური და მაქსიმალური მნიშვნელობები. pctincrease ყოველი მომდევნო (next) ექსტენტის ზრდის კოეფიციენტია (პროცენტულად). მონაცემების ჩასმა ბლოკში ხორციელდება მანამ ბლოკის შევსების პროცენტი არ აღემატება pctfree-ს მნიშვნელობას, ხოლო შემდგომი ჩასმა მონაცემთა ბლოკში ნებადართული იქნება მაშინ, როცა მისი თავისუფალი სივრცის პროცენტი მიაღწევს pctused-ს მნიშვნელობას.

1.2.2.2. ინდექსები (Indexes)

ინდექსი არის მოწესრიგებულ მონაცემთა სია, რომელიც ცხრილის სახით რამდენიმე სვეტითაა წარმოდგენილი.

მონაცემთა ბაზის მომხმარებელი გარკვეული კრიტერიუმით ეძებს მონაცემთა ჩანაწერებს. ეს კრიტერიუმი შეიძლება შედგებოდეს ერთი ან მეტი სვეტისაგან (ან მათი კონკრეტული მნიშვნელობებისგან).

ძირითადი იდეაა - ძებნის პროცესის დროის შემცირება ინდექსის საფუძველზე. ანუ წინასწარ მოწესრიგებული ინდექსის ცხრილში გადაისინჯება არა ყველა ჩანაწერი, არამედ მხოლოდ მითითებული კრიტერიუმისა. თუ ასეთი ჩანაწერი ნაპოვნია, მაშინ აქედან ხდება მიმართვა უშუალოდ შესაბამის მონაცემთა ცხრილის კონკრეტულ ჩანაწერზე. მისი სტრუქტურა ასეთია:

CREATE [UNIQUE] INDEX *index_name* ON
table_name(*column_name*[, *column_name*...])
TABLESPACE *tab_space;*

წარმოგიდგენთ უკვე შექმნილ ინდქსს:

```
CREATE INDEX customers_last_name_idx ON
customers(last_name);
```

customers_last_name_idx ინდექსი დადებული აქვს customers ცხრილის last_name-ს.

შესაძლებელია ასევე ერთდროულად ორი ველის დაინდექსება:

```
CREATE INDEX customers_first_last_name_idx ON
customers(first_name, last_name);
```

ამჟამად ინდექსი დაედო customers ცხრილის ორ ველს first_name, last_name-ს.

1.2.2.3. მიმდევრობები (Sequences)

Sequences არის მონაცემთა ბაზის ერთი რომელიმე მონაცემის მნიშვნელობის ზრდა ავტომატურად. მისი შექმნის დროს საჭიროა მიეთითოს თუ რა მნიშვნელობამდე უნდა გაიზარდოს აღნიშნული ველი და როგორი დიაპაზონით. მისი სტრუქტურა შემდეგნაირია:

```
CREATE SEQUENCE sequence_name
[START WITH start_num]
[INCREMENT BY increment_num]
[ { MAXVALUE maximum_num | NOMAXVALUE } ]
[ { MINVALUE minimum_num | NOMINVALUE } ]
[ { CYCLE | NOCYCLE } ]
[ { CACHE cache_num | NOCACHE } ]
[ { ORDER | NOORDER } ];
```

მოცემულ Sequences-ში 10-მდე გაიზრდება მნიშვნელობა 5 დიაპაზონით.

```
CREATE SEQUENCE SEQUENCE_1
```

```
START WITH 10 INCREMENT BY 5
MINVALUE 10 MAXVALUE 20
CYCLE CACHE 2 ORDER;
```

1.2.2.4. წარმოდგენები (Views)

წარმოდგენა არის სვეტებისა და სტრიქონებისაგან შემდგარი ვირტუალური ცხრილი, რომლებიც დინამიკურად ამოირჩევა ერთი ან მეტი ცხრილიდან და/ან წარმოდგენიდან.

ფიზიკურად წარმოდგენას SELECT მოთხოვნის სახე აქვს, რომლის საფუძველზეც სრულდება მონაცემების ამორჩევა. წარმოდგენა ხშირად გამოიყენება ისეთი სვეტების დასამალად რომლების კონფიდენციალურ ინფორმაციას შეიცავს.

წარმოდგენაში შეიძლება ისეთი სვეტების გამოჩენა, რომელთა ნახვაც ნებადართულია მომხმარებლისათვის. თუ წარმოგდენაში ჩართული არ არის ცხრილის რომელიმე სვეტი, მაშინ ცხრილზე დადებულია ვერტიკალური ფილტრი. თუ მოთხოვნა შეიცავს სტრიქონების ამორჩევის პირობებს, მაშინ ცხრილზე დადებულია ჰორიზონტალური ფილტრი.

წარმოდგენა შეიძლება შეიცავდეს ბმული ცხრილების სვეტებს. მაგალითად წარმოდგენა შეიძლება შეიცავდეს customer ცხრილიდან თანამშრომლების გვარებს, და department ცხრილიდან თუ რომელ დეპარტამენრში მუშაობს იგი.

წარმოდგენასთან მიმართვის დროს მოწმდება იმ ობიექტების არსებობა რომლების საჭიროა წარმოდგენის განმსაზღვრელი SELECT მოთხოვნის შესასრულებლად. თუ მოთხოვნაში მითითებული რომელიმე ცხრილი წაშლილია მაშინ წარმოდგენა ვერ იმუშავებს და გაიცემა შეტყობინება შეცდომის შესახებ. თუ წაშლილი ცხრილის ნაცვლად შევქმნით იგივე სახელის და სტრუქტურის მქონე ცხრილს, მაშინ წარმოდგენა იმუშავებს.

თუ ახალ ცხრილს განსხვავებული სტრუქტურა აქვს მაშინ წარმოდგენა უნდა წავშალოთ და ახალი შევქმნათ. მოცემულია შემდეგი წარმოდგენა რომელსაც ინფორმაცია მოაქვს customer ცხრილიდან.

```
CREATEORREPLACEVIEWcustomer_VIEW AS
SELECT a.cust_id, a.first_name, a.last_name
FROMcustomer  a;
```

მოცემულ წარმოდგენას კი ინფორმაცია მოაქვს 2 ცხრილიდან:

```
CREATEORREPLACEVIEWcustomer_VIEW_2AS
SELECT a.cust_id, a.first_name, a.last_name,
       t.dep_id, t.dasaxeleba
FROMcustomer  a,t.deepartment
Where a.dep_id = t.dep_id;
```

1.2.2.5. სინონიმები (Synonyms)

სინონიმი არის მომხმარებლის მიერ ცხრილის ან წარმოდგენისათვის ფსევდონიმის შერჩევა, რომლითაც ის მომავალში გამოიყენებს მათ. შეიძლება ცხრილი ეკუთვნოდეს სხვა მომხმარებელს, მაშინ მასთან მიმართვისას უნდა მოხდეს მომხმარებლის_სახელის და ცხრილის_სახელის მითითება.

არსებობს ორი სახის: *კერძო* და *საერთო გამოყენების* სინონიმები. Oracle გამოიყენებს მონაცემთა ბაზის ადმინის-ტრატორის შემდეგ პრივილეგიებს:

- CREATE SYNONYM;
- CREATE PUBLIC SYNONYM;
- DROP PUBLIC SYNONYM;

- სინონიმის შექმნა:

  Create public synonym cust_temp for cust_temp;

- სინონიმის წაშლა:

  Drop public synonym cust_temp;

1.2.2.6. კურსორები (Cursors)

მოთხოვნის შესრულების შედეგად, კლიენტის პროგრამას შეუძლია დაუბრუნოს ასობით ათასი სტრიქონი. კლიენტის პროგრამები ყოველთვის ვერ ახერხებს ასეთი მოცულობის მონაცემებთან მუშაობას, რადგან მათ შესანახად დიდი ზომის მეხსიერებაა საჭირო.

ამ პრობლემების გადასაწყვეტად გამოიყენება კურსორები. *კურსორი კლიენტის პროგრამას საშუალებას აძლევს იმუშაოს არა ასობით ან ათასობით სტრიქონთან, არამედ ერთ სტრიქონთან ან სტრიქონების მცირე ბლოკთან.* ძირითადად კურსორი ჯდება მოთხოვნილი ინფორმაციის ამოსაღებად.

კურსორები უნდა გამოვიყენოთ მხოლოდ აუცილებლობის შემთხვევაში, რადგან, ჯერ ერთი, ისინი საშუალებას არ გვაძლევს მონაცემების დამუშავების ოპერაციები შევასრულოთ მონაცემთა მთელ ნაკრებზე, და მეორეც, კურსორების საშუალებით მონაცემთა დამუშავების სიჩქარე გაცილებით დაბალია სერვერის სტანდარტულ საშუალებებთან შედარებით.

მაგალითად:

```
CURSOR cv_product_cursor IS
  SELECT product_id, name, price
  FROM products
  ORDER BY product_id;
```

1.2.3. ოპერატორები, ფუნქციები და პროცედურები

1.2.3.1. ლოგიკური პირობის ოპერატორები

IF,THEN, ELSE, ELSIF, END IF ოპერატორები არის ლოგიკის ოპერატორები. ისინი გამოიყენება პირობის შესრულების ან არ შესრულების დროს. მათი სტრუქტურა შემდეგნაირია:

```
IF condition1 THEN
  statements1
    ELSIF condition2 THEN
        statements2
      ELSE
          statements3
END IF;
```

მაგალითად:

```
update customer
set a.name  = “giorgi'
where a.cust_id = 5;
if (sql%rowcount = 1)
     then commit;
  else
      rollback;
   raise_application_error(-20000, skriptshi Secdomaa');
 end if;
```

ამ შემთხვევაში თუ ერთზე მეტი ჩანაწერი დააფდეითდა (Update), ამოაგდებს შეცდომას „სკრიპტში შეცდომაა“ და აღარ შეასრულებს პირობას, დაროლბექდება (rollback).

1.2.3.2. LOOP ოპერატორი

LOOP ოპერატორი გამოიყენება რომელიმე ველის დასათვლელად. შესაძლებელია როგორც ციკლი ისე განვიხილოთ. ქვემოთ მოცემულ მაგალითში counter-ი 0-დან დაიწყებს დათლას 1 დიაპაზონით და გაჩერდება, როდესაც მიაღწევს 5-ის მნიშვნელობას.

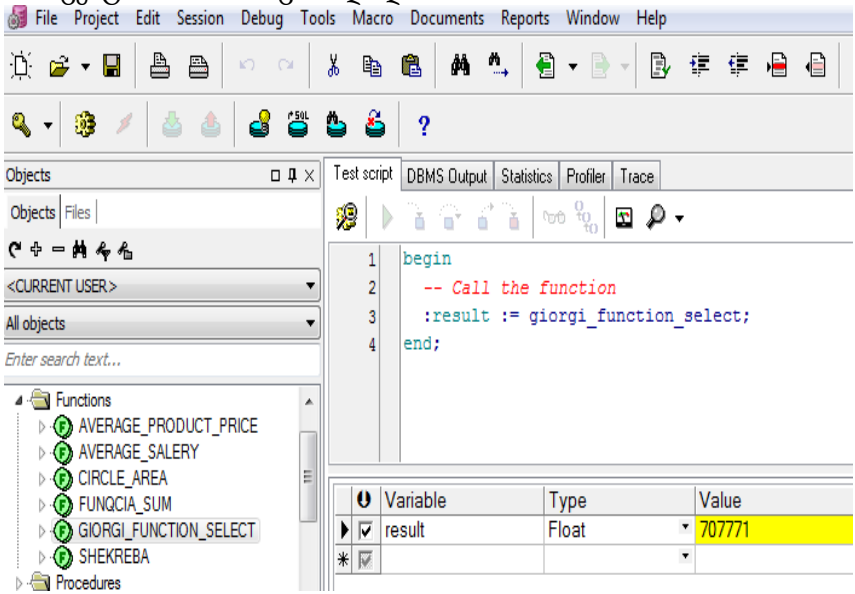
```
counter := 0;
LOOP
  counter := counter + 1;
   EXIT WHEN counter = 5;
END LOOP;
```

1.2.3.3. ფუნქციები (Functions)

ფუნქციები წარმოადგენენ მონაცემთა ბაზის დამოუკიდებელ ობიექტებს და მოთავსებული არიან შესაბამის მონაცემთა ბაზაში. ამ ფუნქციებს შეიძლება ჰქონდეთ ერთი ან მეტი პარამეტრი ან არც ერთი. განვიხილოთ მაგალითი:

```
createorreplacefunction giorgi_function_select
returnnumberas
  V_giorgi_function_select number;
begin
selectsum(salary) + sum(employee_id)
into V_giorgi_function_select
from employees;
return V_giorgi_function_select;
end giorgi_function_select;
```

აღნიშნული ფუნქცია დააბრუნებს დაჯამებულ მნიშვნელობას ორი ცხრილიდან. მისი გამოძახებისას დაბრუნდება შედეგი რომელიც მოცემულია ქვემოთ (ნახ.1.4). ამ ფუნქციას პარამეტრის გადაცემა არ სჭირდება რაგდან ავტომატურად იღებს მონაცემტა ბაზის ორი ცხრილიდან.



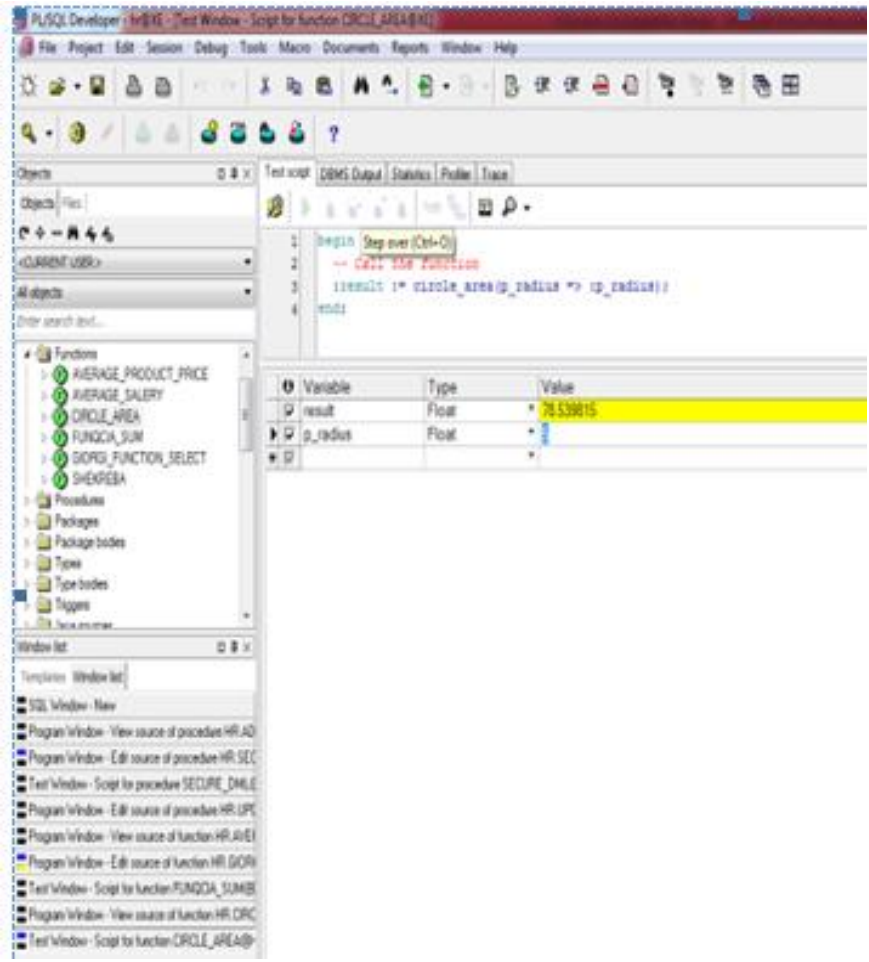
ნახ.1.4

მოცემული ფუნქცია კი ასრულებს წრის ფართობის გამოთვლას. გამოძახებისას საჭიროა გადაეცეს წრის რადიუსის მნიშვნელობა:

```
CREATEORREPLACEFUNCTION circle_area (
p_radiusINNUMBER
) RETURNNUMBERAS
v_piNUMBER := 3.1415926;
v_areaNUMBER;
BEGIN
```

```
    v_area := v_pi * POWER(p_radius, 2);
RETURN v_area;
END circle_area;
```

შედეგი იხილეთ ქვემოთ (ნახ.1.5): პასუხია 78.539815.



ნახ.1.5

1.2.3.4. პროცედურები (Procedures)

პროცედურა არის Oracle-ს ბრძანებების სახელდებული ნაკრები, რომელიც უშუალოდ სერვერზე ინახება და წარმოადგენს მონაცემთა ბაზის დამოუკიდებელ ობიექტს. თუ სერვერზე ხშირად გვიწევს ერთი და იგივე ოპერაციების შესრულება, მაშინ უმჯობესია მათი გაფორმება პროცედურის სახით.

პროცედურების გამოძახება შესაძლებელია კლიენტის პროგრამის, სხვა პროცედურის, ჯობის ან ტრიგერის მიერ.

პროცედურის კოდის შეცვლა შეუძლია მხოლოდ მის მფლობელს ან მონაცემთა ბაზის db_owner ფიქსირებული როლის წევრს. საჭიროების შემთხვევაში პროცედურის ფლობის უფლება შეგვიძლია ერთი მომხმარებლიდან მეორეს გადავცეთ.

პროცედურის გამოყენებას შემდეგი უპირატესობები აქვს:

პროცედურის შესრულების წინ სერვერი მისთვის ადგენს შესრულების გეგმას.

Execution plan - ასრულებს მის ოპტიმიზებას და კომპილირებას. ამრიგად, პროცედურის პირველი გამოძახებისას სერვერი დროს და რესურსებს ხარჯავს არა მარტო პროცედურის ბრძანებების შესრულებაზე, არამედ შესასრულებლად მათ მომზადებაზე, რაც საკმაო რესურსებს მოითხოვს.

მწარმოებლურობის ასამაღლებლად სერვერი ასრულებს პროცედურის შესრულების გეგმისა და კომპილირებული კოდის ქეშირებას. იგივე პროცედურის განმეორებით გამოძახების შემთხვევაში სერვერი მაშინვე დაიწყებს ბრძანებების შესრულებას.

ამრიგად, პროცედურები იძლევა ოპერაციების შესრულების სიჩქარის გაზრდის შესაძლებლობას, რადგან მათი განმეორებით გამოყენების შემთხვევაში, ისინი უკვე ჩატვირთულია ოპერატიულ მეხსიერებაში, სადაც მათი მოძებნა გაცილებით სწრაფად ხდება.

პროცედურის აგების სტრუქტურა შემდეგნაირია:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type[, ...])]
{IS | AS}
BEGIN
  procedure_body
END procedure_name;
```

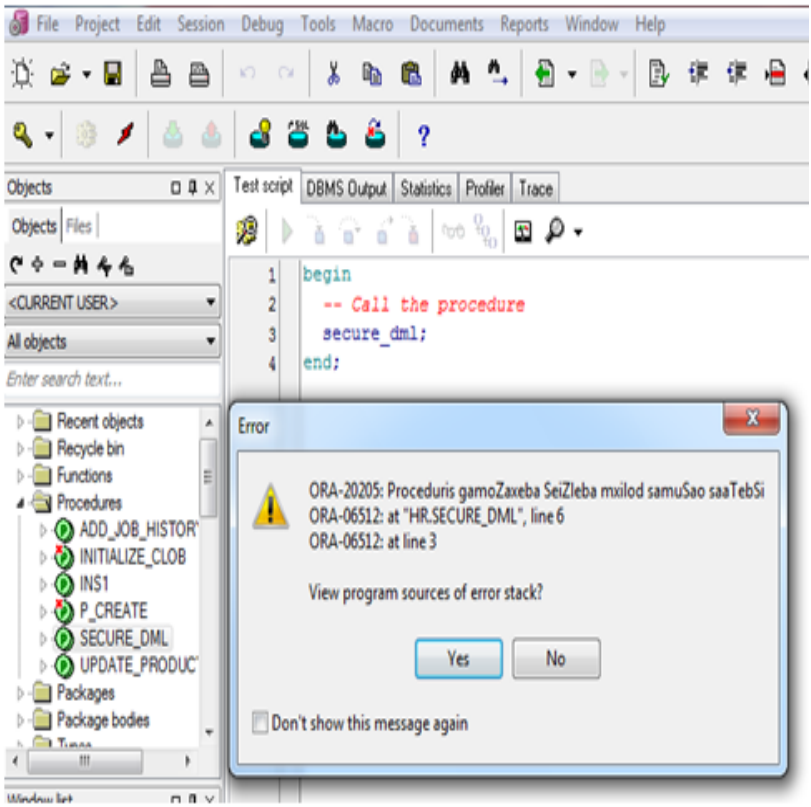
განვიხილოთ მაგალითი:

```
CREATE OR REPLACE PROCEDURE secure_dml
IS
BEGIN
IF TO_CHAR (SYSDATE, 'HH24:MI')
NOTBETWEEN'10:00'AND'18:00'
OR TO_CHAR (SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
      RAISE_APPLICATION_ERROR (-20205,
            'პროცედურის გამოძახება ხდება მხოლოდ
             სამუშაო საათებში');
ENDIF;
END secure_dml;
```

მოცემული პროცედურა მონაცემთა ბაზის მომხმარებელს საშუალებას მისცემს პროცედურა გამოიძახოს მხოლოდ მისთვის განკუთვნილ სამუშაო საათებში. თუ მომხამარებელმა ამ დროს გადააცილა მაშინ სისტემა დაუბრუნებს შეცდომას (ნახ.1.6):

„პროცედურის გამოძახება ხდება მხოლოდ სამუშაო საათებში“ [12].

წიგნის N1 დანართში მოცემულია Oracle ბაზის ობიექტების ტესტირების მაგალითები პროცედურებით და ტრიგერებით [14].



ნახ.1.6

1.2.3.5. ტრიგერები (Triggers)

ტრიგერი არის შენახვადი პროცედურის სპეციალური ტიპი, რომელიც სერვერის მიერ ავტომატურად გაიშვება ამა თუ იმ ცხრილზე მოქმედებების შესრულების დროს.

ტრიგერები სხვადასხვა დანიშნულებით გამოიყენება. მათი საშუალებით შესაძლებელია ბმულ ცხრილებში კასკადური ცვლილებების შესრულება, მონაცემების შემოწმების რთული ალგორითმების რეალიზება და ა.შ.

თითოეული ტრიგერი დაკავშირებულია კონკრეტულ ცხრილთან. მაგალითად, ცხრილში მონაცემების შეცვლის წინ სერვერი ავტომატურად გაუშვებს ტრიგერს, და თუ მისი შესრულება წარმატებით დამთავრდა, მაშინ შესრულდება ცხრილში მონაცემების ცვლილება. ტრიგერის მიერ შესრულებული მოქმედებები განიხილება როგორც ერთი ტრანზაქცია.

ტრიგერების გამოყენება შეიძლება, აგრეთვე, ჩვენი შეხედულების მიხედვით. ტრიგერები არ უნდა გამოვიყენოთ მარტივი შემოწმებების შესასრულებლად. ან იმ მოქმედებების შესასრულებლად, რომლებიც შეიძლება შესრულდეს შენახვადი პროცედურებით (Stored procedure). ტრიგერების გამოყენება არ არის სასურველი, რადგან ისინი ბლოკავენ რესურსს მუშაობის დამთავრებამდე და სხვა მომხმარებლებს არ ეძლევათ ამ რესურსთან მიმართვის შესაძლებლობა.

ტრიგერები განსხვავდება იმ ბრძანებების ტიპის მიხედვით, რომლებზეც ისინი რეაგირებენ. არსებობს ტრიგერების სამი ტიპი:

- INSERT TRIGGER. ამ ტიპის ტრიგერები გაიშვება INSERT ბრძანებით მონაცემების ჩასმის მცდელობის დროს;
- UPDATE TRIGGER. ამ ტიპის ტრიგერები გაიშვება UPDATE ბრძანებით მონაცემების შეცვლის მცდელობის დროს;
- DELETE TRIGGER. ამ ტიპის ტრიგერები გაიშვება DELETE ბრძანებით მონაცემების წაშლის მცდელობის დროს.

წიგნის N2 დანართში ნაჩვენებია ტრიგერის შექმნის სქემა.

1.2.4.1. პაკეტები (Packages)

შენახვადი პროცედურები არის პროგრამების (პროცედურების) მონაცემთა ბაზაში შენახვის ხერხი. ჩვენ შემთხვევაში PL/SQL ენაზე დაწერილი სცენარები (პროგრამები) შეგვიძლია შევინახოთ Oracle-ს ბაზაში. ასეთი შენახვადი პროცედურების ერთობლიობა ქმნის პაკეტს.

პაკეტები შეიძლება დავაყენოთ ჯობზე (დავალებებში) და გაეშვას მაშინ როდესაც მას მიაკითხავენ. მისი შექმნის სტრუქტურა შემდეგნაირია:

```
CREATE [OR REPLACE] PACKAGE package_name
{IS | AS}
  package_specification
END package_name;
```

განვიხილოთ ერთ-ერთი პაკეტი:

```
CREATE OR REPLACE PACKAGE product_package AS
  TYPE t_ref_cursor IS REF CURSOR;
  FUNCTION get_products_ref_cursor RETURN t_ref_cursor;
  PROCEDURE update_product_price (
p_product_id IN products.product_id%TYPE,
p_factor IN NUMBER
  );
END product_package;
```

ქვემოთ მოცემულ მაგალითში წერია PACKAGE BODY თავისი ყველა ელემენტით.

```
CREATE OR REPLACE PACKAGE BODY product_package AS
 FUNCTION get_products_ref_cursor
 RETURN t_ref_cursor IS
  products_ref_cursor t_ref_cursor;
 BEGIN
  -- get the REF CURSOR
  OPEN products_ref_cursor FOR
   SELECT product_id, name, price
   FROM products;
  -- return the REF CURSOR
```

```
    RETURN products_ref_cursor;
  END get_products_ref_cursor;

  PROCEDURE update_product_price (
p_product_id IN products.product_id%TYPE,
p_factor IN NUMBER
  ) AS
v_product_count INTEGER;
  BEGIN
-- count the number of products with the
-- supplied product_id (should be 1 if the product exists)
    SELECT COUNT(*)
    INTO v_product_count
    FROM products
    WHERE product_id = p_product_id;
    -- if the product exists (v_product_count = 1) then
    -- update that product's price
    IF v_product_count = 1 THEN
      UPDATE products
      SET price = price * p_factor
      WHERE product_id = p_product_id;
      COMMIT;
    END IF;
  EXCEPTION
    WHEN OTHERS THEN
      -- perform a rollback when an exception occurs
      ROLLBACK;
  END update_product_price;
          END product_package;
```

1.2.4.2. კლასტერები (Clusters)

კლასტერი არის ცხრილთა ჯგუფი, რომელთა შენახვა მიზანშეწონილია ერთად, ვინაიდან აქვთ ზოგიერთი ერთნაირი სვეტი და ხშირად საჭიროა მათი გაერთიანება მომხმარებელთა მოთხოვნებში.

Oracle -პროგრამული პაკეტის ახალი ვერსიები თვითონ წყვეტს ამ პრობლემას და მომხმარებელს ნაკლებად უხდება მასზე ფიქრი. ამიტომ აქ მეტს აღარ განვიხილავთ.

1.2.4.3. როლები და პრივილეგიები

Oracle-ში გამოიყენება *ე.წ. „პრივილეგიების" ცნება, რაც ზოგადად მონაცემთა ბაზასთან მიმართვის კონტროლისთვისაა გათვალისწინებული.* მაგალითად, პრივილეგიები - სისტემური, ბაზის ადმინისტრატორის, მომხმარებლების, ობიექტების და სხვ.

ამჯერად შევეხებით ცხრილის - როგორც მბ-ობიექტის პრივილეგიებს.

- ALTER: მომხმარებლის მიერ ცხრილის სტრუქტურის შეცვლა (ველების დამატება, პარამეტრების შეცვლა და ა.შ.);
- DELETE: მომხმარებლის მიერ ცხრილის სტრიქონების წაშლა (თვით ცხრილი არ იშლება);
- INDEX: მომხმარებლის მიერ ცხრილის მონაცემებთან მიმართვა ინდექსური ფაილების შესაქმნელად;
- INSERT: მომხმარებლის მიერ ცხრილში ახალი სტრიქონების ჩამატება;
- REFERENCES: მომხმარებლის მიერ ცხრილთაშორის კავშირების აგება;
- SELECT: მომხმარებლის მიერ ცხრილიდან ინფორმაციის ამორჩევა (წაკითხვა);

- UPDATE: მომხმარებლის მიერ ცხრილის ჩანაწერების კორექტირება. ვერ ქმნის ახალ სტრიქონს და ვერ შლის ძველ სტრიქონს.

არსებობს აგრეთვე Oracle-ს კომპიუტერულ სისტემასთან მომუშავეთა პრივილეგიების სცენარის ცნება, რომელიც მათი როლებისა და უფლებების საფუძველზე დგება. მოხმარებლებს მიეცემათ როლები და უფლებები მონაცემთა ბაზის ადმინის-ტრატორისა და ობიექტების მფლობელთა მიერ. ამის საფუძველზე განისაზღვრება მონაცემებთან მომხმარებელთა საკმარისი წვდომის შესაძლებლობანი. პრივილეგიების მისაცემად ადმინისტრატორი გამოიყენებს შემდეგ ფუნქციას:

grant *privilege* **to** *user/role;*

ერთნაირი პრივილეგიების მომხმარებელთა სახელები ან როლები, ან თვით რამდენიმე პრივილეგიის დასახელება შეიძლება " , "-ით ჩაიწეროს ერთ **grant**-ში. მაგალითად,

grant *create table, create index, create view* **to** *gio,levan,nino;*

პრივილეგიებისათვის ობიექტებზე გამოიყენება შემდეგი ფორმატი:

grant *select, delete* **on** *Students* **to** *gio, tazo;*

პრივილეგიების მოსახსნელად ან შესაცვლელად გამოიყენება კონსტრუქცია:

revoke *privilege* **from** *user/role;*

თუ სისტემა ღიაა და ყველა მომხმარებელს ეძლევა ყველაფრის უფლება, მაშინ შეიძლება ასეთი ფრაზის მიწოდება:

grant *create session* **to** *public;*

როლი არის მონაცემთა ბაზის ადმინისტრატორის მარტივი და მძლავრი ფუნქცია. მაგალითად, როცა მონაცემთა ბაზის

მომხმარებელთა რიცხვი მატულობს (ანუ ბევრია სტუდენტი გიო, დიტო, თაზო და ა.შ.), მაშინ მბა განსაზღვრავს როლებს „სტუდენტი", „ლექტორი", „ბუღალტერი" და მასში ჩამოთვლის ყველა მომხმარებელს. ასე იქმნება სწორედ პრივილეგიების სქემა, ანუ რომელ როლს რა პრივილეგიები მიეცემა.

ამგვარად, როლის შესაქმნელად გამოიყენება კონსტრუქცია:

create role ***role_name;***

grant*role***to** ***user;***

მბა-ს შეუძლია დაადგინოს კონკრეტული როლების სია, რომლებიც მინიჭებულია გამოუცხადებლად:

alter user ***user_id*** **default** ***rolelist_of_roles;***

თუ სისტემაში მოხდა ცვლილებები და შეიცვალა როლები, მაშინ მბა-ს შეუძლია შექმნას ახალი და გააუქმოს ძველი როლები.როლის გაუქმება ხდება კონსტრუქციით:

drop role ***role_name;***

როლი არის უფლებელის ნაკრები, რომელიც შეიძლება დაუკავშირო ერთ კონკრეტულ იუზერს, იუზერების ნაკრებს ან თვითონ სხვა როლს. იგი საკმაოდ მოხერხებულია ბაზის ადმინისტრატორისათვის, რადგან არ ჭირდება ყოველ ბაზის მომხმარებელს მისცეს ხელით სხადასხვა უფლებელი. User-ს ჩასვავს ერთ რომელიმე როლში და ამით სრულდება მისი უფლებების გაცემა. სინტაქსი ასეთია:

```
CONNECT system/manager
GRANT CREATE ROLE TO store;
GRANT CREATE USER TO store WITH ADMIN OPTION;
```

უფლებების გაცემა როლზე ხდება შემდეგნაირად:

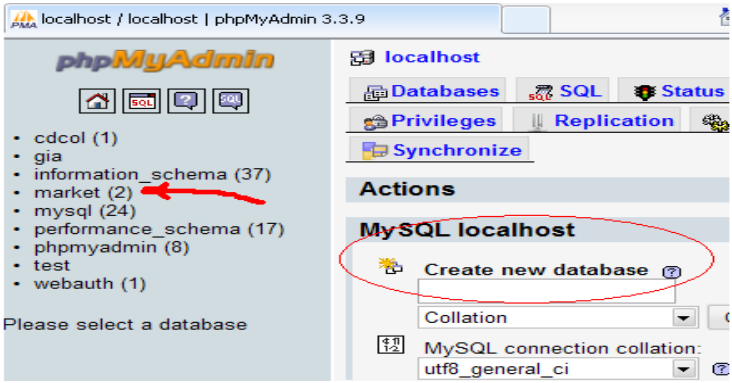
```
GRANT SELECT, INSERT, UPDATE, DELETE
  ON product_types TO product_manager;
GRANT SELECT, INSERT, UPDATE, DELETE
```

```
    ON products TO product_manager;
GRANT SELECT, INSERT, UPDATE, DELETE
    ON salary_grades TO hr_manager;
GRANT SELECT, INSERT, UPDATE, DELETE
    ON employees TO hr_manager;
GRANT CREATE USER TO hr_manager;
GRANT product_manager, hr_manager TO overall_manager;
```

1.3. MySQL მონაცემთა ბაზის მართვის სისტემა

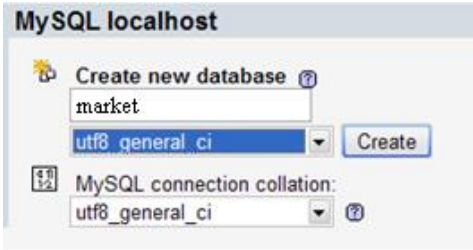
განვიხილოთ MySQL მონაცემთა ბაზების მართვის სისტემა. იგი, როგორც განაწილებული რელაციური ბაზა ერთ-ერთი აქტუალური და ფართოდ გამოყენებადი კლიენტ-სერვერული პაკეტია დღეისათვის, განსაკუთრებით php ვებ-ტექნოლოგიებში.

ვგულისხმობთ, რომ MySQL სისტემა დაინსტალირებულია კომპიუტერზე.როგორც წესი, MySQL სისტემის გამოძახება ხდება რომელიმე ბრაუზერში, მაგალითად, Internet Explorer-ში url-მისამართად უნდა მიეთითოს: http://localhost/phpmyadmin/ სტრიქონი. 1.7 ნახაზზე გამოტანილია შედეგი.



ნახ.1.7

აქ ავირჩევთ ჩვენთვის საჭირო ბაზას, მაგალითად, market, რომელსაც აქვს 2 ცხრილი. თუ საჭიროა ახალი ბაზის შექმნა, მაშინ გამოვიყენებთ ველს: create new database. აქ ჩავწერთ ახალი ბაზის სახელს, Collation ველ-ში ვირეჩევთ სტრიქონს utf8_general_ci, რაც უნიკოდის გამოყენების საშუალებას იძლევა (ნახ 1.8).



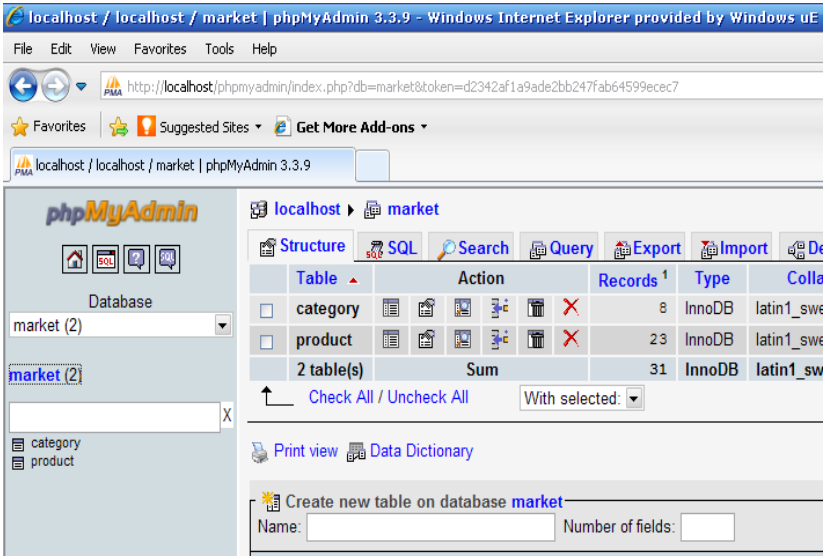
ნახ.1.8

შემდეგ უნდა შევქმნათ ბაზის ცხრილები, რისთვისაც ველში "Create new table on database": market - ჩავწერთ ცხრილის სახელს, მაგალითად, product ან category და დავიწყებთ ცხრილის ველების (სვეტების) შექმნას (მაგალითად, ნახ.1.9).

Field	Type	Length/Values[1]
pr_ID	INT	4
Name	VARCHAR	30
prize	DECIMAL	
cat_ID	INT	2

ნახ.1.9

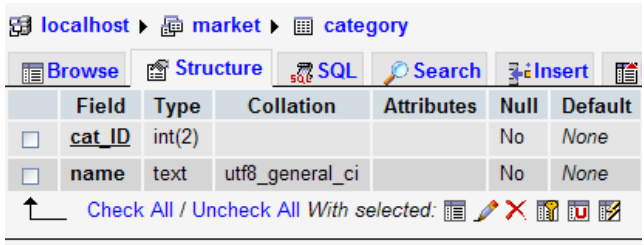
1.10 ნახაზზე ნაჩვენებია ცხრილები: category და product.



ნახ.1.10

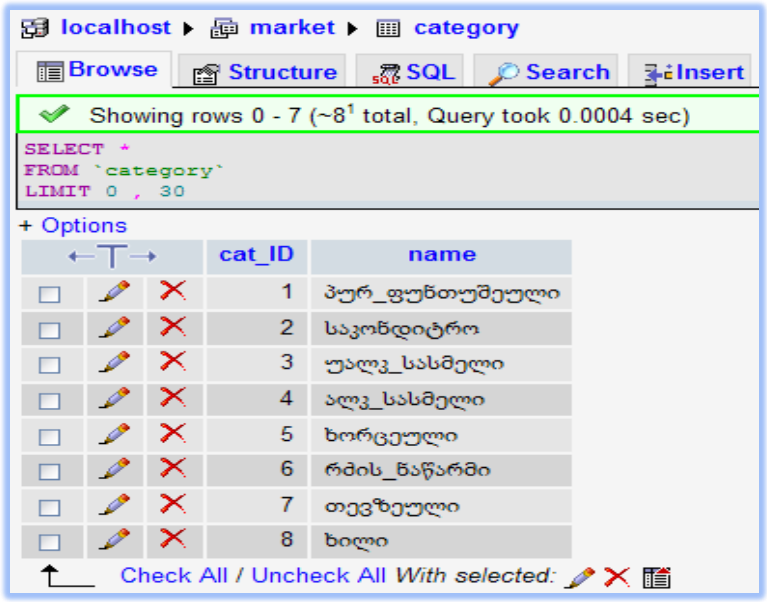
ავირჩიოთ category ცხრილი და მენიუში Structure.

1.11 ნახაზზე ნაჩვენებია შედეგი.



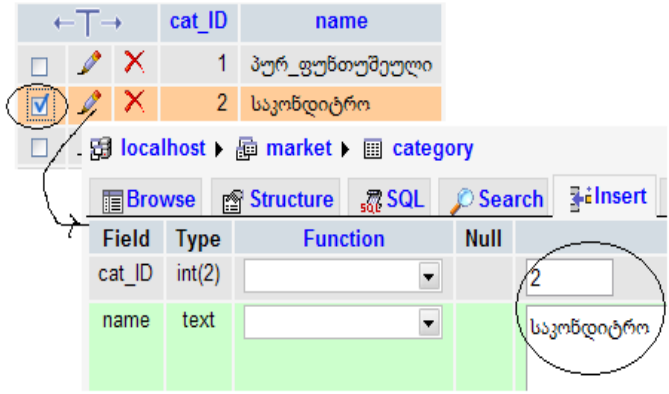
ნახ.1.11

cat_ID გასაღებური ველი (პირველადი ინდექსი) აქ გახაზულია. ახლა უნდა შევავსოთ სტრიქონები კონკრეტული მნიშვნელობებით. ამისათვის ავამოქმედებთ Browse გადამრთველს და გამოჩნდება 1.12 ნახაზზე მოცემული ფანჯარა.



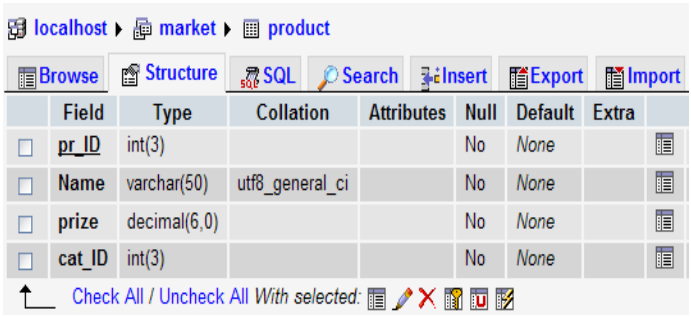
ნახ.1.12

„ჩეკბოქსის" ჩართვით და ფანქრის არჩევით შევალთ სტრიქონის რედაქტირების რეჟიმში და შევასრულებთ ჩვენთვის საჭირო ფუნქციას (ნახ.1.13).

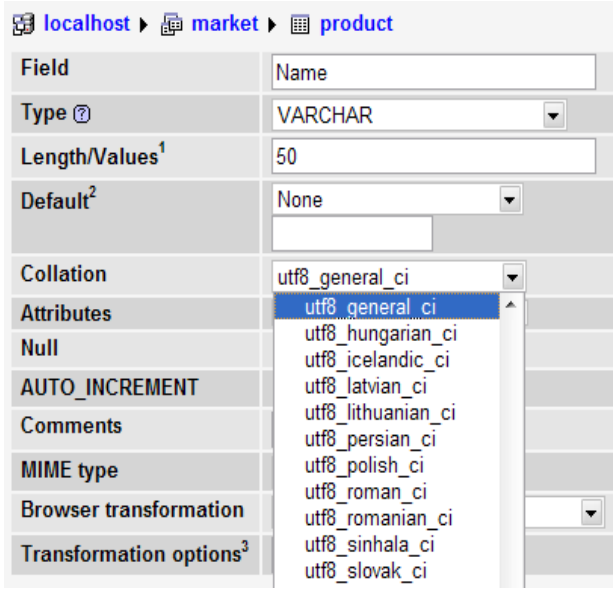


ნახ.1.13

ახლა გადავიდეთ product ცხრილზე. მისი ველების განსაზღვრით და სტრიქონების შეტანით შესაძლოა ქართული უნიკოდების მაგივრად '????? ????' სტრიქონის მიღება. ასეთი ველის სტრუქტურაში Collation სვეტში უნდა ჩავსვათ utf8_general_ci (ნახ.1.14-ა,ბ).

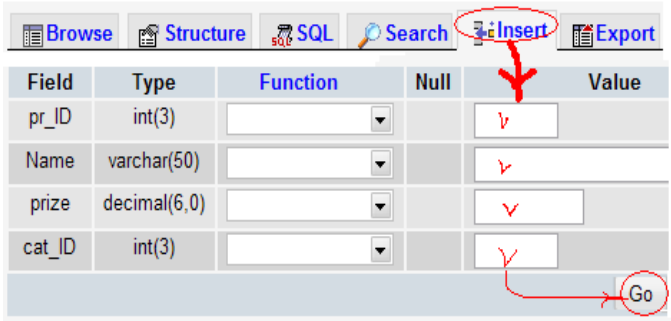


ნახ.1.14-ა



ნახ.1.14-ბ

შევავსოთ პროდუქციის ცხრილი მნიშვნელობებით Browse და Insert გადამრთველების გამოყენებით (ნახ.1.15).



ნახ.1.15

შევსებული ცხრილის ფრაგმენტი მოცემულია 1.16 ნახაზზე.

	pr_ID	Name	prize	cat_ID
	1	არაჟანი	4	6
	2	პური	1	1
	3	ძეხვი	11	5
	4	ხაჭო	3	6
	5	ფუნთუშა ქიშმიშით	1	1
	6	კოკა_კოლა	2	3
	7	ფანტა	2	3
	8	ღვინო ქიწმმარაული	18	4
	9	ღვინო ხვანჭკარა	23	4
	10	ტირამუსი	7	2
	11	საქონლის ხორცი	18	5
	12	კონიაკი "ვარციხე"	20	4
	13	არაყი "გომი"	13	4
	14	ასატრიწა	25	7
	15	ვაშლი "გოლდენი"	3	8
	16	კონსერვი "შპროტი"	5	7
	17	ფორთოხალი	3	8
	18	მინერალური "ბორჯომი"	2	3
	19	მინერალური "ნაბეღლავი"	1	3
	20	ორცხობილა "ნუშის"	4	2
	21	ნაყინი "ვანილის"	6	2
	22	მინერალური "ლიკანი"	2	3
	23	ღორის სამწვადე	23	5

Check All / Uncheck All *With selected:*

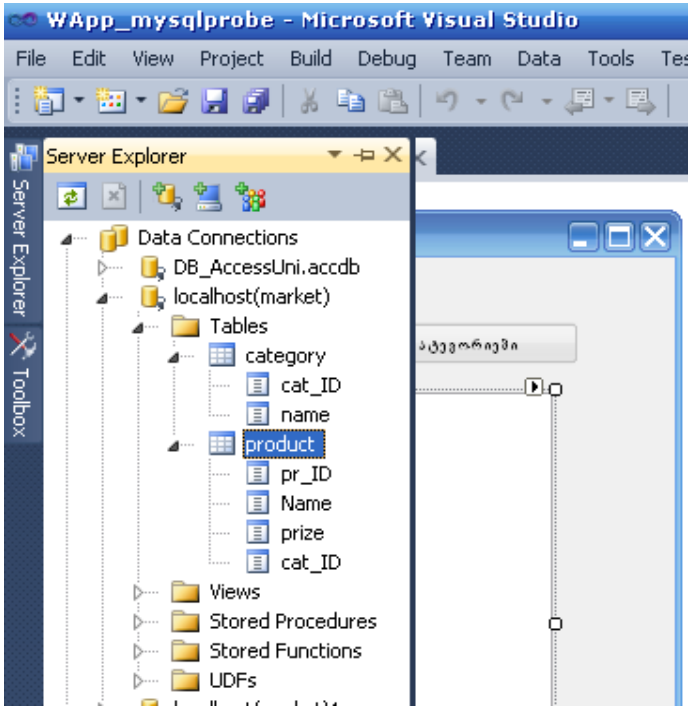
ნახ.1.16

ამგვარად, MySQL ბაზის ორი ცხრილი category და product მზადაა გამოსაყენებლად. დავხუროთ ეს ბაზა.

1.4. MySQL ბაზის დამუშავება C# ენის გამოყენებით

ახლა ავამუშავოთ Ms Visual Studio.NET პაკეტი და C# ენის WindowsForm რეჟიმში Form1-ზე გამოვიტანოთ MySQL ბაზის მონაცემები, სხვადასხვა ჭრილში, მოთხოვნების შესაბამისად.

1.17 ნახაზზე ნაჩვენებია Server Explorer-ის ფანჯარა, სადაც localhost(market) მიერთებულ ბაზაში ჩანს Tables: category და product თავიანთი ველებით (ატრიბუტებით). მთავარი მომენტია C# კოდიდან MySQL-ის market ბაზის მიერთება სამუშაოდ.



ნახ.1.17

პროგრამის კოდში უნდა მოთავსდეს MySQL ბაზის დასაკავშირებელი რამდენიმე სტრიქონი, რომელიც ქვემოთაა მოცემული.

```
MySql.Data.MySqlClient.MySqlConnection msqlConnection = null;
msqlConnection = new  MySql.Data.MySqlClient. MySqlConnection
      ("server=localhost;"+
       "User Id=user@localhost; database=market;
                                        Persist Security Info=True");
```

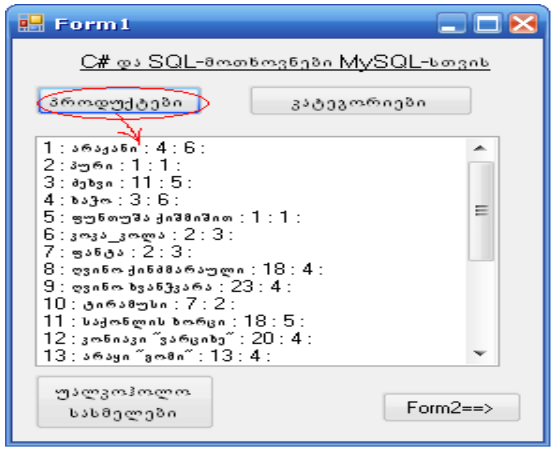
აქ განსაზღვრულია კლიენტის ბაზის მისაერთებლად აუცილებელი მონაცემები, როგორიცაა localhost-სერვერი, მომხმარებლის User-იდენტიფიკატორი და ბაზის სახელი market.

პროგრამის მთლიან კოდს შემდეგ განვიხილავთ. ახლა Form1-ფორმაზე მოთავსებული რამდენიმე ღილაკი განვიხილოთ.

ღილაკით "**პროდუქტები**" ListBox1-ში გამოიტანება ყველა პროდუქტის სია, იდენტიფიკატორით, დასახელებით, ფასით და კატეგორიის ნომრით. მისი SQL-მოთხოვნის "ამორჩევის" კოდს ექნება ასეთი სახე:

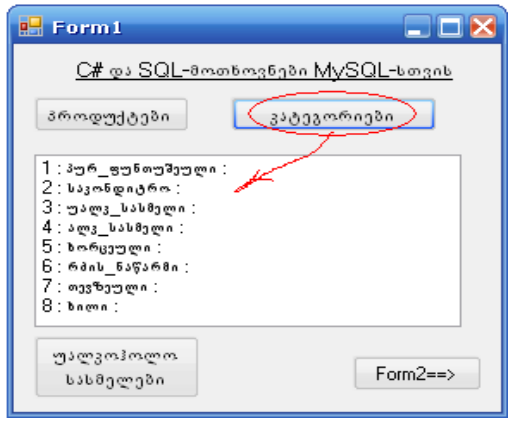
```
private void button1_Click(object sender, EventArgs e)
 {
    Amorcheva("select * from product order by pr_ID, Name",
                                  "pr_ID", "Name", "prize", "cat_ID");
 }
```

შედეგები ასახულია 1.18 ნახაზზე მოცემულ ფანჯარაში.



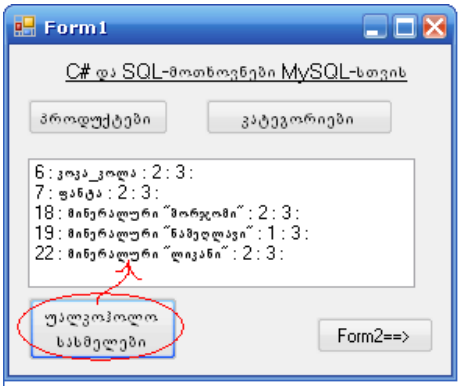
ნახ.1.18

ღილაკით "**კატეგორიები**" ListBox1-ში გამოიტანება ბაზაში არსებული პროდუქციის ყველა კატეგორიის დასახელება. როგორც 1.19 ნახაზიდან ჩანს, სახეზეა 8 კატეგორია.



ნახ.1.19

ღილაკით "**უალკოჰოლო სასმელები**" ListBox1-ში გამოიტანება იმ პროდუქტების სია, რომელთა ველში "კატეგორიის იდენტიფიკატორი" შეესაბამება უალკოფოლო სასმელებს, ანუ ჩვენ შემთხვევაში `cat_ID=3`. შედეგი ასახულია 1.20 ნახაზზე.



ნახ.1.20

C# პროგრამის კოდი მოცემულია 1.1 ლისტინგში.

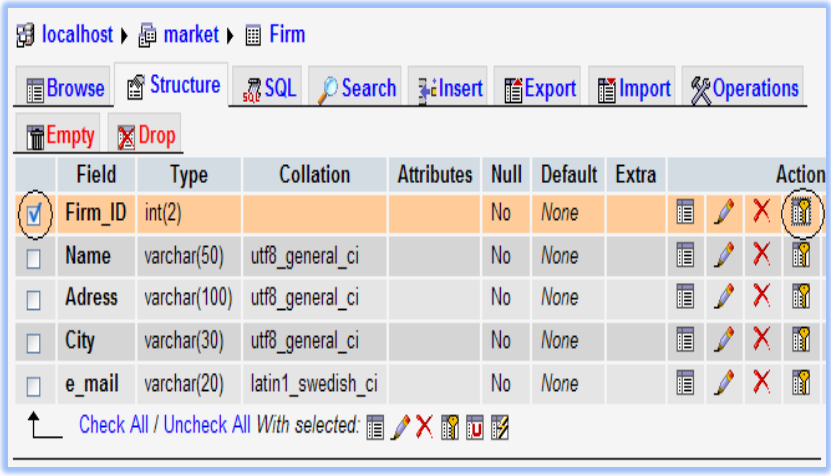
```
// ლისტინგი_1.1 ----------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WApp_mysqlprobe
{
 public partial class Form1 : Form
 {
   public Form1() { InitializeComponent(); }
   private void Amorcheva(string sqlbefehl,
                                  params string[] velebi)
    {
      int i;
```

```
    string striqoni;
MySql.Data.MySqlClient.MySqlConnection msqlConnection = null;
    msqlConnection = new
MySql.Data.MySqlClient.MySqlConnection("server=localhost;"+
       "User Id=user@localhost;database=market;Persist
                                        Security Info=True");
MySql.Data.MySqlClient.MySqlCommand msqlCommand = new
                        MySql.Data.MySqlClient.MySqlCommand();
   msqlCommand.Connection = msqlConnection;
    msqlCommand.CommandText = sqlbefehl;
 try
   {
     msqlConnection.Open();
     MySql.Data.MySqlClient.MySqlDataReader msqlReader =
                                  msqlCommand.ExecuteReader();
     listBox1.Items.Clear();
     while (msqlReader.Read())
      {
       striqoni = "";
       for (i = 0; i < velebi.Length; i++)
          striqoni += msqlReader[velebi[i]] + " : ";

       listBox1.Items.Add(striqoni);
      }
    }
  catch (Exception ex)
    {
       MessageBox.Show(ex.Message);
    }
  finally
      {
        msqlConnection.Close();
      }
 }
 private void button1_Click(object sender, EventArgs e)
  {
    Amorcheva("select * from product order by pr_ID, Name",
                         "pr_ID", "Name", "prize", "cat_ID");
  }
 private void button2_Click(object sender, EventArgs e)
  {
    Amorcheva("select * from category order by cat_ID",
                                             "cat_ID", "Name");
```

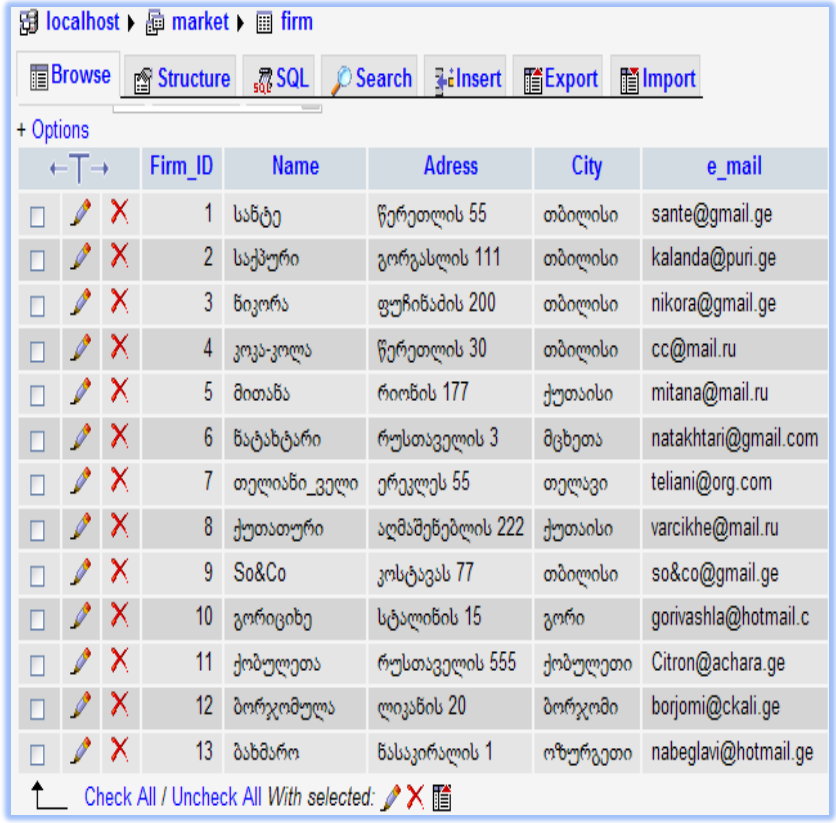
```
  }
private void button3_Click(object sender, EventArgs e)
  {
    Amorcheva("select * from product where cat_ID=3 order by
           pr_ID", "pr_ID", "Name", "prize", "cat_ID");
  }
 }
}
```

ამოცანა: განხილულ ბაზას დავამატოთ ახალი ცხრილი პროდუქციის მწარმოებელი ფირმებისათვის, სახელით Firm, რომელსაც ექნება ხუთი ველი: იდენტიფიკატორი, დასახელება, მისამართი, ქალაქი და ელ-ფოსტა (ნახ.1.21).



ნახ.1.21

Browse გადამრთველით გამოვიტანოთ ფირმების ცხრილი და შევავსოთ იგი (ნახ.1.22).



ნახ.1.22

product ცხრილში საჭიროა დავამატოთ ველი Firm_ID, რომლითაც ის დაუკავშირდება ამ პროდუქციის მწარმოებელი ფირმის სტრიქონს. შევიტანოთ product ცხრილის Firm_ID ველის სვეტში შესაბამისი ფირმების იდენტიფიკატორები (ნახ.1.23).

	pr_ID	Name	prize	cat_ID	Firm_ID
☐	1	არაჟანი	4	6	1
☐	2	პური	1	1	2
☐	3	ძეხვი	11	5	3
☐	4	ხაჭო	3	6	1
☐	5	ფუნთუშა ქიშმიშით	1	1	3
☐	6	კოკა_კოლა	2	3	4
☐	7	ფანტა	2	3	6
☐	8	ღვინო ქინძმარაული	18	4	7
☐	9	ღვინო ხვანჭკარა	23	4	7
☐	10	ტირამუსი	7	2	5
☐	11	საქონლის ხორცი	18	5	5
☑	12	კონიაკი "ვარციხე"	20	4	8
☐	13	არაყი "გომი"	13	4	8
☐	14	ასატრინა	25	7	9
☐	15	ვაშლი "გოლდენი"	3	8	10
☐	16	კონსერვი "შპროტი"	5	7	9
☐	17	ფორთოხალი	3	8	11
☐	18	მინერალური "ბორჯომი"	2	3	12
☐	19	მინერალური "წაბეღლავი"	1	3	13
☐	20	ორცხობილა "ნუშის"	4	2	5
☐	21	ნაყინი "ვანილის"	6	2	1
☐	22	მინერალური "ლიკანი"	2	3	12
☐	23	ღორის სამწვადე	23	5	5

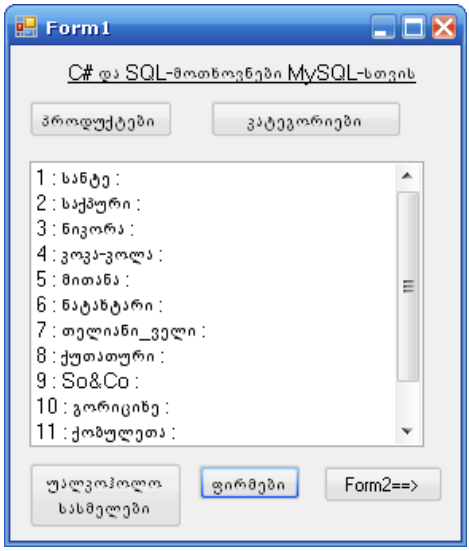
Check All / Uncheck All *With selected:*

ნახ.1.23

ღილაკით "**ფირმები**" ListBox1-ში გამოვიტანოთ პროდუქციის მწარმოებელი ფირმების სია. C#კოდის ფრაგმენტს ექნება შემდეგი სახე:

```
private void button5_Click(object sender, EventArgs e)
   {
     Amorcheva("select * from Firm order by Firm_ID",
                                        "Firm_ID", "Name");
   }
```

შედეგები მოცემულია 1.24 ნახაზზე.



ნახ.1.24

1.5. MariaDB ახალი ალტერნატიული MySQL ბაზა

მონაცემთა ბაზა MariaDB არის MySQL-ის ახალი ალტერნატიული free-ვარიანტი, რომელიც შექმნა მაიკლ ვიდენიუსმა 2009 წელს. იგი იყო ავტორი MySQL-ისაც (1995), რომელიც Oracle კორპორაციამ შეიძინა 2008 წელს და გახადა იგი კომერციული პროდუქტი [16,17]. (მ. ვიდენიუსის უფროსი ქალიშვილია მაია -MySQL, ხოლო უმცროსი - მარია).

MariaDB თავსებადია MySQL-თან, უზრუნველყოფს შესაბამისობას API-სთან და MySQL-ის ბრძანებებთან. მასში დამატებულია მონაცემთა საცავის (storage engine) ქვესისტემა XtraDB, რომელიც ცვლის MySQL-ის InnoDB-ს [18]. ეს საცავი მაღალმწარმოებლურია InnoDB-სთან შედარებით, აქვს მეხსიერების მართვის და ინფორმაციის ნაკადების შეტანა-გამოტანის უფრო სრულყოფილი მექანიზმები, რეალიზებულია ტრანზაქციების ACID მოთხოვნები (Atomicity, Consistency, Isolation, Durability) და აქვს MVCC (MultiVersion Concurrency Control - მონაცემთა ბაზასთან წვდომის პარალელური უზრუნველყოფა) არქიტექტურა.

იმისათვის, რომ დავაყენოთ MariaDB მონაცემთა ბაზა, უნდა შესრულდეს შემდეგი პირობები:

1) გამართული უნდა იყოს ლინუქსის ოპერაციული სისტემა ფიზიკურ ან ვირტუალურ მანქანაზე (2 CORE , 2 GB RAM , 20 GB HDD). რაც შეეხება Linux-ის დისტრიბუციას, არჩევანი დიდია (მაგალითად, CentOS-ს, რომელიც დიდი პოპულარობით სარგებლობს) [19];

2) გამართულ ოპერაციულ სისტემას წვდომა უნდა ჰქონდეს ინტერნეტში (ბაზის დაყენების დროს).

მას შემდეგ რაც ოპერაციული სისტემა გამართულია და ინტერნეტში წვდომაც გვაქვს, შეგვიძლია დავიწყოთ მონაცემთა ბაზის დაყენება (აღნიშნული ინსტრუქცია გათვლილია „CentOS 6 64-bit“-სთვის) [16].

ოპერაციულ სისტემაში შევდივართ „root“ მომხმარებლით და ტერმინალში ვწერთ შემდეგ ბრძანებებს:

1) touch /etc/yum.repos.d/MariaDB.repo რაც შექმნის MariaDB.repo ფაილს /etc/yum.remos.d/ დირექტორიაში;

2) vi/etc/yum.repos.d/MariaDB.repo vi ედიტორით გავხსნათ MariaDB.repo ფაილი კლავიატურაზე „i“ ღილაკის გამოყენებით გადავიდეთ „insert“ რეჟიმში და ფაილში ჩავწეროთ შემდეგი :

- [mariadb]
- name = MariaDB
- baseurl = http://yum.mariadb.org/5.5/centos6-amd64
- gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
- gpgcheck=1

კლავიატურაზე „Esc“ ღილაკის გამოყენებით გადავიდეთ ბრძანების რეჟიმში ვწერთ „wq“ და ვაწვებით „Enter“ ღილაკს, რის შემდეგადაც ჩვენი შეყვანილი ინფორმაცია შეინახება „MariaDB.repo“ ფაილში.

3) yum -y install MariaDB MariaDB-server (დაიწყება ბაზის ინსტალაცია).

4) /etc/init.d/mysql start (მონაცემთა ბაზის გაშვება)

5) როდესაც მონაცემთა ბაზა გაეშვება „mysql“ ბრძანებით შეგვიძლია შევიდეთ მონაცემთა ბაზის ტერმინალში სადაც გაუშვებს mysql ის ბრძანებებს:

- show databases;
- quit;

6) მას შემდეგ, რაც მონაცემთა ბაზას წარმატებით დავუკავშირდით და ყველაფერმა იმუშავა, აუცილებელია

უსაფრთხოების პარამეტრების გამართვა, რისთვისაც ვუშვებთ შემდეგ ბრძანებას და დაკვირვებით გავივლით შემდგომ ეტაპებს:

mysql_secure_installation

```
/usr/bin/mysql_secure_installation: line 379: find_mysql_client:
command not found
NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED
FOR ALL MariaDB
SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP
CAREFULLY!
In order to log into MariaDB to secure it, we'll need the current
password for the root user.  If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.
Enter current password for root (enter for none):
OK, successfully used password, moving on...
Setting the root password ensures that nobody can log into the MariaDB
root user without the proper authorisation.
Set root password? [Y/n] Y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
... Success!
Remove anonymous users? [Y/n] y
... Success!
Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the
network.
Disallow root login remotely? [Y/n] y
... Success!
By default, MariaDB comes with a database named 'test' that anyone
can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.
Remove test database and access to it? [Y/n] y
- Dropping test database...
... Success!
```

- Removing privileges on test database...
... Success!
Reloading the privilege tables will ensure that all changes made so far will take effect immediately.
Reload privilege tables now? [Y/n] y
... Success!
Cleaning up...
All done! If you've completed all of the above steps, your MariaDB installation should now be secure.
Thanks for using MariaDB!

7) /etc/init.d/mysql restart (კონფიგურაციის გავლის შემდეგ აუცილებელია მონაცემთა ბაზის გადატვირთვა).

8) chkconfig mysql on (სისტემის ჩატვირთვისას მონაცემთა ბაზა ავტომატურად რომ გაეშვას)

mysql -u root –p (მონაცემთა ბაზის ტერმინალში შესასვლელად -u პარამეტრით გადავცემთ მომხმარებლის სახელს, ხოლო -p პარამეტრით პაროლს).

1.6. MongoDB: დოკუმენტ-ორიენტირებული NoSQL ბაზა

დოკუმენტ-ორიენტირებული მონაცემთა ბაზა (Document-Oriented Database) არის მონაცემთა ბაზების მართვის სისტემა (მბმს), რომელიც გამოიყენება დოკუმენტების (მონაცემთა იერარქიული სტრუქტურების) შესანახად და რეალიზებულია NoSQL მიდგომის საშუალებით [13,16,21].

დოკუმენტ-ორიენტირებული მბმს-ას საფუძვლად უდევს დოკუმენტების საცავი (document Store), რომელსაც აქვს ხის სტრუქტურა. ხის სტრუქტურა იწყება ფესვური კვანძით და შეიძლება შეიცავდეს რამდენიმე შიგა კვანძს და ფოთლების კვანძს.

ფოთლების კვანძი შეიცავს მონაცემებს, რომლებიც დოკუმენტის დამატების დროს შეიტანება ინდექსებში, რაც უზრუნველყოფს, რთული სტრუქტურების შემთხვევაშიც კი, მოიძებნოს გზა საჭირო მონაცემებისკენ [22].

მოთხოვნის საფუძველზე API (Application Programming Interface) ახორციელებს დოკუმენტების და მათი ნაწილების ძებნას.

დოკუმენტები შეიძლება იყოს ორგანიზებული (დაჯგუფებული) კოლექციებში. ეფექტური ინდექსირების მიზნით სასურველია კოლექციებში მსგავსი სტრუქტურების დოკუმენტების გაერთიანება.

დოკუმენტ-ორიენტირებული მონაცემთა ბაზები გამოიყენება შინაარსის მართვის სისტემებში (CMS -Content Management System), საგამომცემლო საქმეში, დოკუმენტების საძიებო სისტემებში და სხვ. ასეთი ბაზების მართვის სისტემების მაგალითებია: MongoDB, CouchDB, Couchbase, MarkLogic, eXist, IBM Lotus Notes და სხვ.[16].

დოკუმენტ-ორიენტირებული მონაცემთა ბაზების 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.

დოკუმენტი მონაცემთა ბაზაში მისამართდება უნიკალური გასაღების საშუალებით. ხშირად ეს გასაღები მარტივი სტრიქონია, რომელიც შეიძლება იყოს URI (Unified Resource Identifier) ან გზა (path) დოკუმენტამდე. ასეთი გასაღების ან მისი ინდექსის საშუალებით მოიძებნება დოკუმენტი ბაზაში და შესაძლებელია მისი სწრაფად ამოღება.

დოკუმენტური ბაზის დამახასიათებელია სიტყვა-გასაღების (მნიშვნელობის-გასაღების) მარტივად განსაზღვრა მოთხოვნილი დოკუმენტების მოსაძებნად. მონაცემთა ბაზას აქვს სპეციალური API ანუ მოთხოვნების ენა, რომელიც უზრუნველყოფს დოკუმენტების მიღებას მათი შინაარსის (content) მიხედვით.

API არის აპლიკაციების დაპროგრამების ინტერფეისი. იგი შეიცავს მზა კლასების, პროცედურების, ფუნქციების, სტრუქტურებისა და კონსტანტების ერთობლიობას, რომელსაც წარმოადგენს დანართი (ბიბლიოთეკა ან სერვისი) ან ოპერაციული სისტემა. იგი გამოიყენება პროგრამისტების მიერ აპლიკაციის შექმნისას.

მომდევნო თავებში ჩვენ დეტალურად შევეხებით NoSQL მონაცემთა ბაზების საკითხებს და კონკრეტულად, MongoDB პაკეტის შესაძლებლობებს.

MongoDB არის დოკუმენტზე ორიენტირებული NoSQL მონაცემთა ბაზა [24]. მისი ინსტალიაციის მიზნით კომპიუტერზე აუცილებელია იგივე წინაპირობები და ეტაპები, რაც MariaDB-სთვის:

1) სისტემაში შევდივართ „root“ მომხმარებლით
2) touch /etc/yum.repos.d/mongodb.repo
3) vi /etc/yum.repos.d/mongodb.repo vi ედიტორის საშუალებით mongodb.repo ფაილში ჩავწრეოთ შემდეგი :

- [mongodb]
- name=MongoDB Repository
- baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
- gpgcheck=0
- enabled=1

4) yum –y install mongo-10gen mongo-10gen-server (მონაცემთა ბაზის დაყენება)
5) service mongod start (მონაცემთა ბაზის სერვისის გაშვება)

6) service mongod status (მონაცემთა ბაზის სტატუსის შემოწმება)

7) mongostat (მონაცემთა ბაზაში მიმდინარე პროცესების ნახვა)

ამით MongoDB-ს დაყენება დამთავრებულია. იმისათვის, რომ მონაცემთა ბაზაში მუშაობა შევძლოთ, ოპერაციული სისტემის ტერმინალზე უნდა ავკრიფოთ mongo და შევიდეთ მონაცემთა ბაზის ტერმინალში, სადაც უშუალოდ mongo-ს ბრძანებების გაშვებას შევძლებთ. mongo (მონაცემთა ბაზის კლიენტი) ბრძანების გაშვებისას ოპერაციული სისტემა ავტომატურად მიმართავს „localhost:27017“ და ცდის ბაზასთან დაკავშირებას.

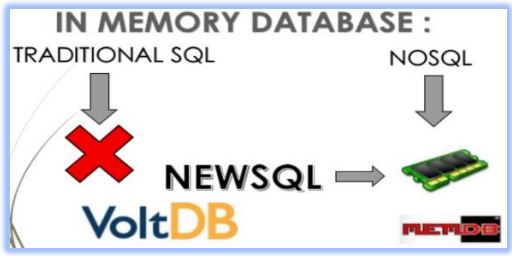
მომდევნო თავებში დეტალურად გავეცნობით ამ ბაზის ფუნქციონირების პრინციპებს და მუშაობის შესაძლებლობებს.

1.7. NewSQL მონაცემთა ბაზები

NewSQL არის თანამედროვე რელაციური ბაზების მართვის სისტემების კლასი, რომელიც გაფართოებული ფუნქციონალობის საფუძველზე უზრუნველყოფს NoSQL ბაზების მსგავს მწარმოებლურობას ტრანზაქციების ოპერატიული დამუშავებისათვის [16,25]. ამასთანავე იგი ინარჩუნებს ACID პრინციპებს.

განსაკუთრებით საყურადღებოა აქ მონაცემთა შენახვის პრინციპულად ახალი პლატფორმების შექმნა, რომლებიც ორიენტირებულია განაწილებული არქიტექტურის და მრავალნაკადურ სისტემებზე.

ასეთი მიდგომის ერთ-ერთი პოპულარული მონაცემთა ბაზაა MemSQL (რელაციური ბაზა ოპერატიული მეხსიერებით [In-Memory Storage] Linux ოპერაციული სისტემისთვის) [26] (ნახ.1.25). იგი იყენებს SQL ენას. კოდის გენერაცია სრულდება C++ ენაზე. ანუ MemSQL სერვერზე გაგზავნილი მოთხოვნები გარდაიქმნება C++ -ზე და კომპილირდება GCC-ს დახმარებით.



ნახ.1.25

MemSQL თავსებადია MySQL-თან. აპლიკაციები შეიძლება შეერთდეს MemSQL სისტემასთან ODBC/JDBC სტანდარტებით, ასევე დრაივერებით და MySQL-ის მომხმარებლებით.
გარდა ზემოაღნიშნულისა, ლიტერატურულ წყაროებში განიხილავენ NewSQL-ის ტიპის შემდეგ ბაზებს: NuoDB, VoltDB, OrientDB, Clustrix, ScaleDB, dbShards და სხვ. [27] (ნახ.1.26).

ნახ.1.26

1.27 ნახაზზე მოცემულია მონაცემთა ტრადიციული SQL ბაზების, NoSQL და NewSQL ბაზების შედარება ოთხი ძირითადი თვისებით (Properties) [26].

როგორც ვხედავთ, NewSQL მონაცემთა ბაზა აერთიანებს ტრადიციული (რელაციური) და NoSQL (არარელაციური) ბაზების საუკეთესო თვისებებს, ამიტომაც იგი ძალზე პერსპექტიულია სამომავლო პროექტებისათვის.

ნახ.1.27

ერთ-ერთი საინტერესო გადაწყვეტა ამ თვალსაზრისით არის MySQL და NoSQL ბაზების ინტეგრაციის საკითხი, რომელსაც მომდევნო პარაგრაფში განვიხილავთ.

1.8. MySQL და NoSQL ბაზების შედარება და მათი პრინციპების ინტეგრაციის კონცეფცია

რელაციური და არარელაციური მოდელები საკმაოდ განსხვავდება ერთმანეთისგან. რელაციურ მოდელისთვის არსებობს სქემა და მონაცემები განთავსებულია მის დაკავშირებულ ცხრილებში (Tables). ცხრილი შეიცავს სტრიქონებს (Rows) და სვეტებს (Columns). ცხრილები ერთმანეთს უკავშირდება პირველადი (PrimaryKey) და მეორადი (ForeignKey) გასაღებების საშუალებით. როდესაც მოთხოვნის საფუძველზე იძებნება რაიმე ინფორმაცია, შესაბამისი ჩანაწერები მიიღება რამდენიმე ცხრილის შეერთების (join), პროექციის, შეზღუდვის, უნიკალურობის (სიმრავლის) და/ან სხვა ოპერაციების მიმდევრობით (ან პარალელური) შესრულების საფუძველზე. მსგავსია ჩაწერის და მოდიფიკაციის პროცედურებიც, რომლებიც უნდა მოხდეს

რამდენიმე ცხრილში ერთდროულად (ბაზის მთლიანობის შესანარჩუნებლად) [2-4]..

NoSQL ბაზებს, რელაციურთან შედარებით, აქვს სრულიად განსხვავებული მოდელი. მაგალითად: დოკუმენტზე ორიენტირებული NoSQL ბაზები მონაცემებს ინახავს JSON (JavaScript Object Notation) ფორმატში [20]. თითოეული JSON დოკუმენტი შეიძლება განვიხილოთ როგორც ობიექტი, რომელსაც მიიღებს აპლიკაცია. ის შეიძლება შეიცავდეს რელაციური მოდელის რამდენიმე ცხრილის გადაბმით მიღებულ ინფორმაციას ერთ დოკუმენტში/ობიექტში, რაც უზრუნველყოფს ჩაწერა/წაკითხვის ოპერაციების წარმადობის გაუმჯობესებას.

მაგალითად, MongoDB არის open-source მონაცემთა ბაზის სისტემა. იგი ინახავს მონაცმებს JSON ტიპის დოკუმენტებში, რომელთა სტრუქტურაც შეიძლება იცვლებოდეს [33]. ინფორმაცია ინახება ერთად. MongoDB იყენებს დინამიურ სქემებს, რაც ნიშნავს, რომ შესაძლებელია ჩანაწერების შექმნა სტრუქტურის (ველების, მნიშვნელობათა ტიპების) წინასწარი განსაზღვრის გარეშე. შემდგომ შეიძლება ჩანაწერების სტრუქტურის (ანუ დოკუმენტების) მარტივად შეცვლა ახალი ველის დამატებით ან არსებულის წაშლით.

ეს მოდელი გვაძლევს საშუალებას წარმოვადგინოთ იერარქიული კავშირები და სხვა უფრო რთული სტრუქტურები შედარებით მარტივად. დოკუმენტებს კოლექციაში არ სჭირდება იდენტური ველები და მონაცემთა დენორმალიზაცია არის აქ ჩვეულებრივი მოვლენა. MongoDB ბაზის სისტემა შეიქმნა მაღალი წვდომადობისა და მასშტაბირების რეალიზაციის მიზნით, ფლობს რეპლიკაციას და ავტომატურ სეგმენტაციას (auto-sharding).

MySQL და MongoDB ბაზებში მრავალი საერთო ტერმინი და ცნებაა (ცხრ.1.1) [32].

ტერმინების შედარება ბაზებში ცხრ.1.1

MySQL	MongoDB
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents, linking

Collection არის mongo დოკუმენტების ჯგუფი. იგი RDBMS ცხრილების ეკვივალენტია.

Document – ცხრილის ძირითადი ნაწილია. დოკუმენტები არის დინამიური. MongoDB და SQL ბაზები გვთავაზობს ფუნქციების მდიდარ კომპლექსს (ცხრ.1.2) [32].

ფუნქციები და შესაძლებლობები ცხრ.1.2

დასახელება	MySQL	MongoDB
Rich Data Model	No	Yes
Dynamic Schema	No	Yes
Typed Data	Yes	Yes
Data Locality	No	Yes
Field Updates	Yes	Yes
Easy for Programmers	No	Yes
Complex Transactions	Yes	No
Auditing	Yes	Yes
Auto-Sharding	No	Yes

MongoDB ბაზას აქვს მაღალფუნქციონალური მოთხოვნების ენა (query language), რომელსაც შეუძლია აგრეთვე ტექსტებთან და სივრცით მონაცემებთან (გეოსისტემები) მუშაობა. აღნიშნული

ფუნქციების გამოყენება შესაძლებელია მონაცემთა მრავალი სახის ტიპებთან, ვიდრე რელაციურ ბაზებში.

ქვემოთ მოყვანილია რამდენიმე მარტივი მოთხოვნის ფორმირების მაგალითი MySQL და MongoDB ბაზებთან მუშაობის დროს:

- „შევიტანოთ products მონაცემთა ბაზაში (იხ. ნახ.1.9) ახალი პროდუქტის შესახებ ინფორმაცია (სტრიქონი), რომელშიც გვაქვს ოთხი ველი: პროდუქტის იდენტიფიკატორი (pr_ID), დასახელება (Name), ფასი (price) და პროდუქტის კატეგორიის იდენტიფი-კატორი (cat_ID)“.
  - MySQL-ში:

```
INSERT INTO products (pr_ID, Name, price, cat_ID)
VALUES (101, 'არაჟანი', 5.80, 6)
```

  - MongoDB-ში:

```
db.products.insert ({
    pr_ID: 101,
    Name: 'არაჟანი',
    price: 5.80,
    cat_ID: 6
})
```

- „ვნახოთ (ავირჩიოთ) ყველა პროდუქტის მონაცემები“:
  - MySQL-ში:

```
SELECT * FROM products
```

  - MongoDB-ში:

```
db.products.find()
```

- „შევცვალოთ (გავზარდოთ) მე-6 კატეგორიის („რძის ნაწარმი“) პროდუქტების ფასი 20 % -ით“:

- MySQL-ში:

```
UPDATE products SET price = price*0.2
WHERE cat_ID = 6
```

- MongoDB-ში:

```
db.products.update(
   { cat_ID: 6 },
   { $set: { price: price*0.2 } }
   { multi: true }
)
```

➢ „წავშალოთ products ბაზაში ალკოჰოლური სასმელების მონაცემები (ალკ_სასმელის იდენტიფიკატორია cat_ID = 4)“.

- MySQL-ში:

```
DELETE FROM products
WHERE cat_ID = 4
```

- MongoDB-ში:

```
db.products.remove (
   ( cat_ID:  4  }
)
```

და ა.შ.

MongoDB ორგანიზაციებს აძლევს საშუალებას უფრო სწრაფად შექმნას აპლიკაციები, დაამუშავოს განსხვავებულ ტიპთა დიდი მოცულობის ჩანაწერები, აპლიკაციათა მასშტაბირების მართვა განახორციელოს უფრო ეფექტურად. ამავდროულად, მონაცემთა ბაზის დამუშავება (დეველოპმენტი) და განახლება საკმაოდ გამარტივებულია.

მონაცემთა ობიექტ-რელაციური მოდელის (სქემის) ცვლილება, მაგალითად, MySQL-ში, მოითხოვს განახლების შედარებით დიდ

დროს, როდესაც MongoDB-ს მონაცემთა მოქნილი მოდელის გამო ასეთი პროცედურები სწრაფად ხორციელდება.

მომდევნო თავში ჩვენ უფრო დეტალურად განვიხილავთ MongoDB ბაზის სისტემაში მუშაობის საკითხებს.

მონაცემთა რელაციური ბაზების მწარმოებლურობის (ეფექტიანობის) ამაღლების ერთ-ერთი აქტუალური მიმართულება არის NoSQL ბაზების პრინციპების ჩანერგვა რელაციურ სისტემებში. ამის კონკრეტული მაგალითია Oracle კორპორაციის მიერ C-ენაზე დაწერილი კროსპლატფორმული პროდუქტი InnoDB, რომელიც გამოიყენება MySQL მონაცემთა ბაზების მართვის სისტემაში (5.5 ვერსიიდან) [18].

InnoDB არის მონაცემთა საცავი (database engine, storage engine), ბაზების მართვის სისტემის პროგრამული კომპონენტი, რომელიც გამოიყენება მონაცემთა ბაზის შენახვის, განახლების და ინფორმაციის ძებნის (create, update, delete, read) მექანიზმების სამართავად.

ამგვარად, Oracle-მ მოახერხა MySQL-ის (v.5.6) InnoDB-ში NoSQL-ის შესაძლებლობების დამატებით 9-ჯერ აემაღლებინა ტრანზაქციათა დამუშავების ეფექტურობა [28].

აქ რეალიზებულია NoSQL-ის ინტერფეისები MySQL და MySQL Cluster მონაცემთა ბაზებთან. ისინი სრულიად უვლის გვერდს SQL შრეს, მის სინტაქსურ ანალიზს და ოპტიმიზაციას. მონაცემები შეიძლება უშუალოდ ჩაიწეროს MySQL-ის ცხრილებში 9-ჯერ უფრო სწრაფად, ACID პრინციპების დაცვით.

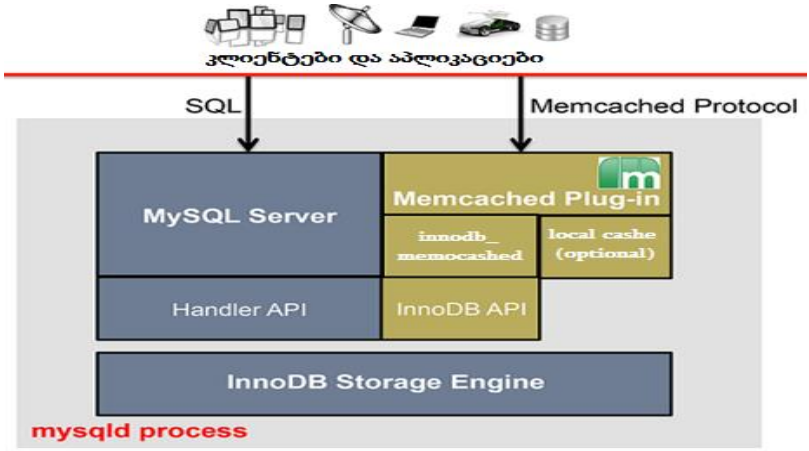
როგორაა NoSQL რეალიზებული MySQL-ში ?

MySQL 5.6 უზრუნველყოფს მარტივ, პირდაპირ *გასაღებ-მნიშვნელობა* ურთიერთქმედებას InnoDB-ს მონაცემებთან ცნობილი API-ის კეშ-მეხსიერების დახმარებით.

კეშ-მეხსიერება (Memcached) წარმოადგენს საერთო დანიშნულების განაწილებული მეხსიერების კეშირების სისტემას [40]. იგი

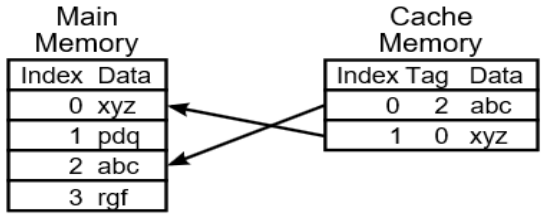
ხშირად გამოიყენება მონაცემთა ბაზების მქონე დინამიკური ვებ-საიტების დასაჩქარებლად მონაცემებისა და ობიექტების კეშირების დახმარებით ოპერატიულ მეხსიერებაში. აქ მნიშვნელოვნად მცირდება, მაგალითად, აპლიკაციის ინტერფეისით ინფორმაციის წაკითხვა მონაცემთა გარე წყაროებიდან. API შესაძლებლობას იძლევა აგრეთვე Memcached-ისა და კლიენტების სტანდარტული ბიბლიოთეკით (C++, Java, Python, PHP და სხვა ენებისათვის) კეშირებულ იქნას მონაცემები ხელმისაწვდომი სერვერების ოპერატიულ მეხსიერებაში და შემდგომ მრავალჯერ იქნას გამოყენებული.

სისტემის რეალიზაცია მოცემულია 1.28 ნახაზზე.



ნახ.1.28

MySQL-ის გაფართოებულ ვერსიაში NoSQL შესაძლებლობებით კლიენტების და აპლიკაციების მოთხოვნები შედის სისტემაში SQL ენაზე (პირდაპირ MySQL Server-ზე) ან Memcached პროტოკოლით (InnoDB-ს გავლით). მეორე შემთხვევაში აპლიკაციებისთვის გარანტირებულია მაღალი სისწრაფის კითხვა-ჩაწერის ოპერაციების შესრულება, ვინაიდან მონაცემები ინახება კეშ-მეხსიერებაში (ნახ.1.29).



ნახ.1.29

კეშ-მეხსიერება (Cache memory) არის აპარატურული ან პროგრამული ნაწილის კომპონენტი, რომელიც იქმნება მონაცემთა დროებით შესანახად, მათი სწრაფად, მრავალჯერადი გამოყენების მიზნით. აქ ინახება წინასწარ გათვლილი ან ამორჩეული მონაცემთა ერთობლიობა, რომელთა ხშირად გამოყენების ალბათობა მაღალია. ძირითად მეხსიერებასთან შედარებით კეშ-მეხსიერება არაა დიდი ზომის.

1.9. Hadoop – „დიდ მონაცემთა“ ახალი ტექნოლოგია

ჩვენ ვცხოვრობთ ინფორმაციის ეპოქაში. 2013 წლისთვის ციფრული სამყაროს ზომა 4.4 ზეტაბაიტი იყო, 2020 წლისთვის კი ნავარაუდებია ინფორმაციის მოცულობის ათმაგი ზრდა, 44 ზეტაბაიტამდე (44 მილიარდი ტერაბაიტი) [16,29].

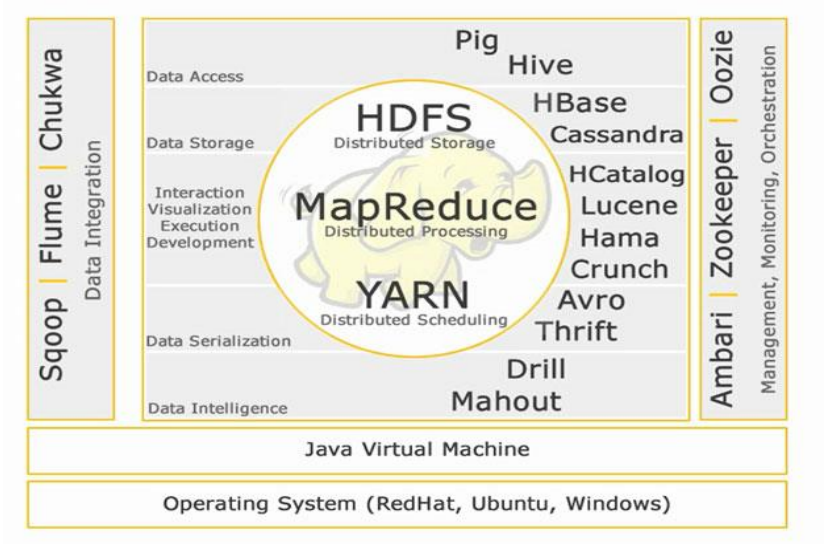
ამ მოცულობის ინფორმაციიდან გარკვეული ნაწილი ისეთ კორპორაციებზე მოდის, როგორიცაა ნიუ იორკის საფონდო ბირჟა – 4-5 ტერაბაიტი დღეში, Facebook.com – ჯამში 240 მილიარდი ფოტო, თვეში 7 პეტაბაიტი ზრდის მახასიათებლით [30], Ancestry.com – 10 პეტაბაიტი მოცულობის გენეალოგიის ბაზა [31].

ეს ადამიანის შექმნილი მონაცემებია, მაგრამ ბოლო ათწლეულში ტენდენცია შეიცვალა, დღეს უკვე ინფორმაციის უდიდეს ნაწილს ადამიანების ნაცვლად კომპიუტერული ტექნიკა აგენერირებს.

ბოლო წლებში აქტუალური გახდა ტერმინი IOT (Internet Of Things), რაც თავის თავში მოიცავს ყველა იმ აპარატს და კომპიუტერულ ტექნიკას, რაც მიმდინარე დროში დიდი რაოდენობით მონაცემებს აგენერირებს. მაგალითად, მანქანის GPS სისტემები, სხვადასხვა სენსორები და ყველა ის ტექნიკა, რაც ძირითადი ფუნქციონირების პარალელურად წარმოშობს დიდი რაოდენობის დამხმარე ინფორმაციას (metadata).

ამ რაოდენობის მონაცემების დამუშავებას სრულიად განსხვავებული სისტემა სჭირდება არა მხოლოდ რელაციური და არარელაციური ბაზების დონეზე, არამედ იმ სერვერული არქიტექტურის დონეზე, სადაც ვაყენებთ მონაცემთა ბაზებს.

დღეისათვის საუკეთესო გამოსავალი დააპროექტა Apache Software Foundation-მა, სახელით Hadoop. Hadoop არის უფასო, ჯავაზე დაფუძნებული პლატფორმა, რომელიც შექმნილია დიდი ზომის მონაცემთა ნაკადის დასამუშავებლად (ნახ.1.30) [27].



ნახ.1.30

Hadoop ეკოსისტემაში სხვადასხვა პროდუქტებია გაერთიანებული, ბირთვად კი სამი ძირითადი კომპონენტი აქვს:

- HDFS (Hadoop Distributed File System) - განაწილებული ფაილური სისტემა მონაცემების შესანახად;
- Map Reduse - მთავარი კომპონენტი განაწილებული გამოთვლების ჩასატარებლად;
- YARN (Yet Another Resource Negotiator) - განაწილებული გარემოს მართვა.

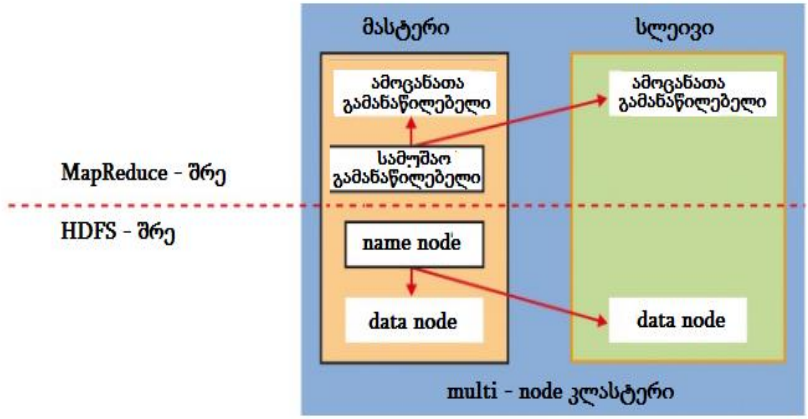
Hadoop-ის განვითარება დაიწყო 2002 წლიდან ამერიკელი მეცნიერის, სტენფორდის უნივერსიტეტის პროფესორის, დაგ გატინგის მიერ, მისი ძირითადი თეორიული საფუძველი კი ჩამოყალიბდა 2003 წელს Google-ს თანამშრომლების მიერ პარალელულ გამოთვლებზე დაწერილი სტატიის გამოქვეყნების შემდეგ და დღემდე ინარჩუნებს უზარმაზარ პოტენციალს, მთელ მსოფლიოში ვითარდება და გამოიყენება Hadoop-კლასტერი [38].

სისტემის სახელი და ლოგო განსაზღვრა, მისი ფუძემდებლის ორი წლის შვილის ყვითელი ფერის სათამაშო სპილომ, რომელსაც ერქვა „ჰადუპი“.

Hadoop - ის ფუნქციონირებას საფუძვლად უდევს ორი ძირითადი პრინციპი (ნახ.1.31):

1) HDFS - განაწილებული ფაილური სისტემა, როგორც კლასტერულ - მონაცემთა სისტემა, სადაც მონაცემები სხვადსხვა წყაროებიდან ერთიანდება ერთ საერთო კომპიუტერულ სისტემაში;

2) MapReduce უზრუნველყოფს მონაცემთა დამუშავებისათვის საჭირო ალგორითმების ფორმულირებასა და იმპლემენტაციას. ალგორითმების დახმარებით დიდი მონაცემები ნაწევრდება მცირე ზომებად, რაც აჩქარებს მათ დამუშავებასა და პარალერულ თვლას.



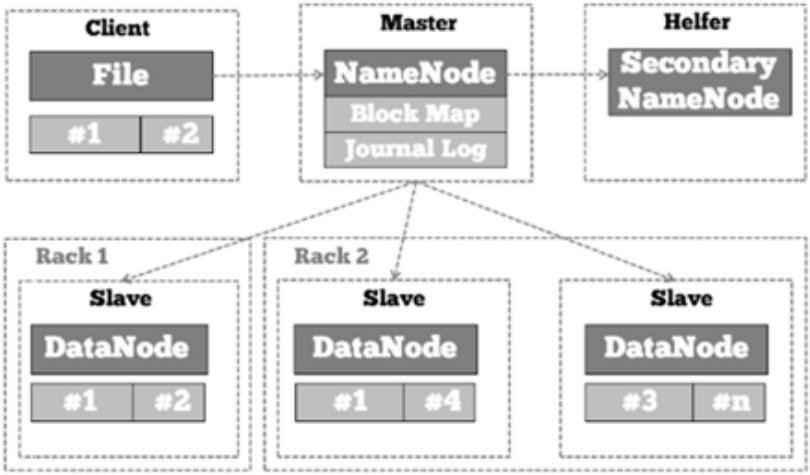
ნახ. 1.31. Hadoop - ის ფუნქციონირება

მასტერი არის Hadoop სისტემის ძირითადი ბირთვი: NameNode- სთვის წინასწარ ცნობილია თუ სად განთავსდეს data node - ში მონაცემები, ხოლო სამუშაო გამანაწილებელი (JobTracker) ამოცანებს ანაწილებს ამოცანათა სერვერში (TaskTracker)

Hadoop-კლასტერი ფუნქციონირებს მასტერ-სლეივი (Master-Slave) პრინციპით. მასტერის დანიშნულებაა გააკონტროლოს “Name node”. აღნიშნული კვანძი ფაილ სისტემიდან მართავს ერთიან მეტამონაცემებს. მონაცემები თავს იყრის "DataNode"-ში რომლის დანიშნულებასაც სლეივი განსაზღვრავს. გამოტოვებული მონაცემების თავიდან ასაცილებლად Hadoop ახდენს ფაილების რეპლიკაციას. ამისათვის სისტემა ფაილებს თავს უყრის ერთ მონაცემთა ბლოკში, რომელსაც აქვს განსაზღვრული სიდიდე. Name node (მასტერი) უზრუნველყოფს აღნიშნული ბლოკის დაყოფას და ყოველი ბლოკი თავიდან გადის რეპლიკაციას (მონაცემთა დუბლირება რამდენიმე სერვერზეა, რაც ზრდის ინფორმაციის

მოპოვების ეფექტურობას). რომელიმე კვანძის გამოტოვების შემთხვევაში ზემოთ აღნიშნული პროცესის საფუძველზე ინფორმაცია არ იკარგება.

სტანდარტული კონფიგურაციის საფუძველზე HDFS-ში განთავსებული მონაცემთა ბლოკები ერთიანდება სამ კლასტერში: File, NameNode, DataNode (ნახ.1.32).



ნახ.1.32. HDFS - ის არქიტექტურა

სისტემის შემოწმების მიზნით პერიოდულად DataNode-დან ხდება მონაცემთა შესახებ შეტყობინებების გაგზავნა ცენტრალურ NameNode-ში, თუ სლეივიდან მოხდა მონაცემის განახლების შეყოვნება განსაზღვრულზე მეტი პერიოდის განმავლობაში, მაშინ მასტერი უკავშირდება სხვა მონაცემთა ბლოკს და პოულობს კოპირებულ მონაცემს. სისტემაში ყოველთვის არის ე.წ. შუალედური Name Node ("SecondaryNameNode"), რომელიც ხელს უწყობს მონაცემთა განახლების პროცესს.

Hadoop-ში მონაცემთა დამუშავების პროცესს უზრუნველყობს MapReduce სისტემა. კომპანია Google-მ შექმნა სპეციალური ალგორითმი კლასტერში მონაცემთა პარალელური თვლისათვის. მონაცემები დამუშავებისას გადანაწილებულია რამდენიმე Map-პროცესზე. Map - ფაზაში Map-პროცესები ითვლება პარალერულად და მიიღება შუალედური შედეგები. ამის შემდეგ Reduce-ფაზა, Reduce-პროცესის მსვლელობისას აგროვებს შუალედურ შედეგებს და ადგენს საბოლოო შედეგს.

Hadoop-ში გამოთვლითი პროცესები აღიწერება, როგორც „სამუშაო“. სამუშაო გამანაწილებელი (JobTracker) ასოცირდება როგორც მასტერი, იგი მართავს და ანაწილებს სამუშაოს კლასტერში. სამუშაოს შესრულების ფუნქციას, ფაქტობრივად თავის თავზე იღებს სლეივ სისტემიდან "TaskTracker"-ი. ყოველი კლასტერის კვანძში არის ერთი ასეთი ამოცანათა გამანაწილებელი (TaskTracker) ინსტალირებული. წესისამებრ, Hadoop-ში ყოველთვის წყვილად მუშაობს DataNode, ისევე როგორც ამოცანათა გამანაწილებელი (TaskTracker).

Hadoop - ასევე მოიაზრება როგორც Hadoop ეკოსისტემა, რომელიც უზრუნველყოფილია მონაცემთა „თავისუფალი არჩევანის ბიბლიოთეკით“ და სხვადასხვა კომბინაციების საშუალებით შესაძლებელი ხდება დიდი მონაცემთა (Bid-Date) დამუშავება.

Hadoop სისტემის გამოყენებით დიდ წარმატებასა და განვითარებას მიაღწია Facebook, Twitter და eBay, თუმცა პირველი იყო Yahoo, სადაც მოხდა სისტემის გამოყენება და იმპლემენტაცია.

დღემდე კომპანიები Cloudera და Hortonworks ახორციელებენ Hadoop - ის დანერგვასა და სრულყოფას.

1.33 ნახაზზე მოცემულია Hadoop სისტემის მნიშვნელოვანი კომპონენტები და მისი როლი ეკოსისტემებში.

ნახ.1.33. Hadoop-ის ბირთვი და ეკოსისტემა

- **Hadoop -ის ბირთვი**

Hadoop-ის ბირთვად შესაძლებელია ჩავთვალოთ ორი ძირითადი კომპონენტი:

1. განაწილებული ფაილური სისტემა - Hadoop Distributed File System (HDFS), ხდება მონაცემთა შენახვა

2. MapReduce - მართავს მონაცემთა ანალიზის, დამუშავების და გამოთვლების პროცესებს. აქ ხდება დეცენტრალიზებული მართვა მონაცემთა დამუშავებასა და გადამუშავების პროცესში

ორივე კომპონენტი ასრულებს Hadoop-ის ძირითად ფუნქციას, ეს არის მონაცემთა შენახვა და პროგრამული დამუშავება. იმ შემთხვევაშიც კი თუ ეს ორი კომპონენტი ერთმანეთთან კავშირშია, შეგვიძლია ჩავთვალოთ როგორც დამოუკიდებელი კომპონენტები და ასევე შესაძლებელია სხვა მონაცემთა სისტემის HDFS გამოყენებული იქნას MapReduce თან. რეალობაში თითქმის ყოველთვის ეს ორი ცენტრალური ელემენტი გმოიყენება კომბინირებულად.

განაწილებული ფაილური სისტემა - **Hadoop Distributed File System (HDFS).**

HDFS არის ჯავა ენაზე ბაზირებული განაწილებული მონაცემთა სისტემა, სადაც უზრუნველყოფილია პერსისტენტულ მონაცემთა საიმედო შენახვა. ამ შემთხვევაში გამოიყენება პარადიგმა „ჩაწერე ერთხელ - წაიკითხე მრავალჯერ“ („Write once, read many“). ჩაწერილი მონაცემების მოდიფიცირება თითქმის შეუძლებელია, მხოლოდ შესაძლებელია მონაცემების ჩამატება. მონაცემების შენახვა ხდება განაწილებულ ბლოკებში (ხშირ შემთხვევაში მათი ზომაა - 64/128 MB), ისინი ნაწილდება Hadoop კლასტერის კვანძებში. ყოველი ბლოკი კოპირდება სამჯერ სხვადასხვა სერვერზე კლასტერების სახით, ეს პროცესი უზრუნველყოფს მონაცემთა მაღალ საიმედოობას და გამორიცხავს რაიმე სახის შეცდომებს. მონაცემთა მართვა უზრუნველყოფილია HDFS ავტომატურად, მომხმარებელს აქვს შესაძლელობა წვდომა ჰქონდეს ვირტუალურ მონაცემთა სისტემასთან.

NameNode არის ცენტრალური მასტერ - კომპონენტი, რომელიც მართვას მეტამონაცემებს და ინახავს DataNode-ში (ნახ.1.32). ამასთან ის გეგმავს და განსაზღვრავს ბლოკში მონაცემთა შესანახ

ადილს და ე.წ. ჟურნალ-ფაილიში ახდენს აქტიურ ოპერაციათა რეგისტრაციას. აღნიშნულის საფუძველზე დროის ნებისმიერ მომენტში შესაძლებელია განისაზღვროს მონაცემთა შენახვის შესახებ ინფორმაცია.

DataNodes არის სისტემის არქიტექტურის სლეივ - კომპონენტი, რომლსაც აქვს მხოლოდ მონაცემთა შენახვაის ფუნქცია. მისი მეშვეობით ხდება კლასტერებში მონაცემთა სკალირებული წარმოდგენა.

Secondary NameNode არ არის NameNode-ს სარეზერვო საშუალება, არამედ მას აქვს მხოლოდ დამხმარე ელემენტის ფუნქცია, რომელიც პერიოდულად ახდენს მონაცემთა განახლებას Map-სა და ჟურნალში და გადასცემს NameNode-ს, რომლის მეშვეობით NameNode - თავისუფლდება და კლასტერი იწყებს შევსებას. თავიდან მხოლოდ ერთი მონაცემი ჩაიწერება ფაილურ სისტემაში, სადაც პირველ რიგში NameNode - დან კლიენტი სვავს კითხვას - არის თუ არა მინაცემის შენახვის საშუალება, დადებითი პასუხის მიღების შემთხვევაში NameNode - დეტალებს ატყობინებს DataNode-ს მონაცემთა შენახვის შესაძლებლობის შესახებ. საბოლოო კომუნიკაცია კლიენტსა და DataNode-ს შორის მთავრდება იმით, რომ მონაცემები იწერება DataNode ბლოკებში.

აღნიშნულის გათვალისწინებით HDFS უქრუნველყოფს ბლოკების ავტომატურ რეპლიკაციას. სტანდარტული რეპლიკაცია არის 3-ის ტოლი და ალგორითმი უზრუნველყოფს განსხვავებულ NameNode-ში მონაცემთა მეორედ და მესამედ კოპირებას. მონაცემთა წაკითხვისას HDFS ახდენს კომუნიკაციას კლიენტასა და NameNode-ს შორის, თუ კლიენტი უფლებამოსილია წაიკითხოს მონაცემი. ეს არის შემთხვევა, როდესაც NameNode-ში წარმოდგენილია ბლოკების ჩამონათვალი, რომელიც გადაეცემა DataNode-ს და პირიქით, ეს პროცესი ხელს უწყობს ჩაწერის და წაკითხვის

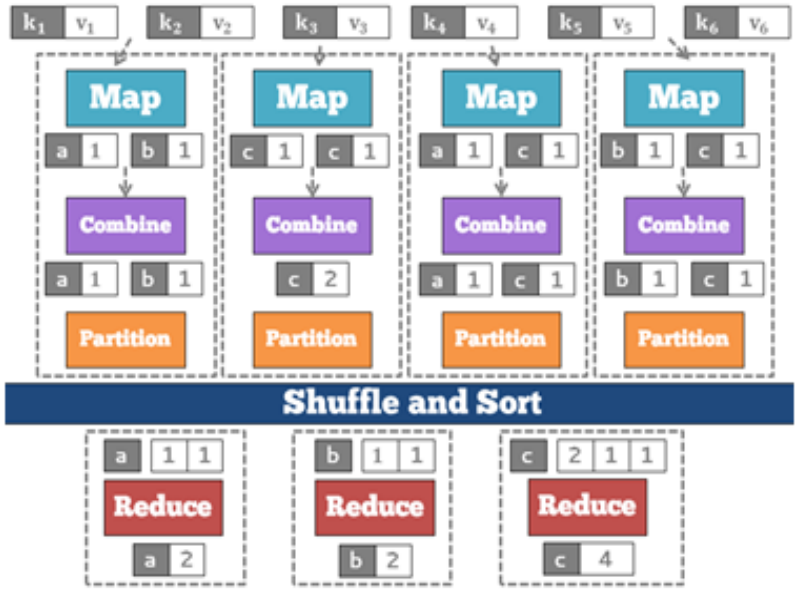
პარალერულად შესრულებას, რაც ამაღლებს მონაცემთა წარმოდგენის სიჩქარეს.

- **MapReduce**

MapReduce არის ჯავაზე ბაზირებული ფრეიმვორკი, რომელიც ანაწილებს და პარალერულედ ამუშავებს ჰადუპის ფაილურ სისტემაში შენახულ დიდი მოცულობის სტრუქტურულ და არასტრუქტურულ მონაცემებს. იგი უზრუნველყოფს განაწილებული სისტემებიდან მონაცემთა ერთ კლასტერში საიმედოდ და შეცდომების გარეშე წარმოდგენას. მას ასევე შეიძლება ვუწოდოთ ფაილური სისტემის მართვის ბიბლიოთეკა, რომელიც მხარს უჭერს Hadoop-ს და ქმნის სცენარებს აუცილებელი ინფრასტრუქტურით და მართავს განაწილებულ მონაცემებს. ასევე უზრუნველყოფს პარარელურ გამოთვლებს [39].

MapReduce-ს სამუშაო მეთოდი და სამუშაოს შესრულების ფაზები ყველაზე კარგად შესაძალებელია წარმოდგენილი იქნას სიტყვების Hello World ის მაგალითზე. ამოცანა მდგომარეობს მოცემული ტექსტიდან განისაზღვროს რამდენად ხშირად მეორდება სიტყვები. ამოცანის გადაჭრის ალგორითმი მოცემულია ნახ.1.34-ზე.

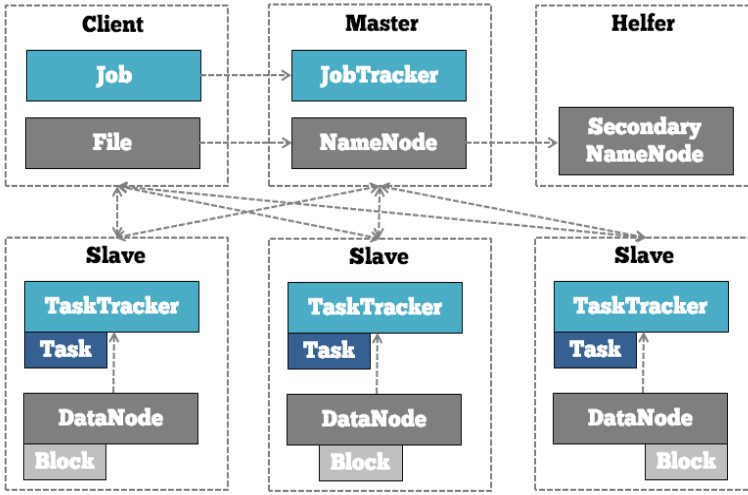
MapReduce, როგორც განაწილებული სისტემა ბაზირებულია გასაღებურ მნიშვნელობათა წყვილზე (Key-Value-Paaren). მაგ. როგორც დავალებათა ერთეულს, დოკუმენტს, რომელიც შეიცავს სიტყვებს ენიჭება ერთი გასაღები, ხოლო მეორე გასაღები ენიჭება უშუალოდ ტექსტს შეფასებისთვის. მოცემული გასაღებურ მნიშვნელობათა წყვილი ქმნის ერთიან პროცესს, რომელიც მონაცემებს ანაწილებს კლასტერის კვანძებში და მომდევნო ფაზები ნაწილდება პარალელურად:



ნახ.1.34. Hadoop - ის MapReduce

- **Map-Phase** - ტექსტი ნაწილდება ცალკეულ სიტყვებად და ყოველი სიტყვა ითვლება და განისაზღვრება ცალკე. ამიტომ ყოველ სიტყვას ენიჭება ერთი გასაღებური მნიშვნელობათა წყვილი და ფასდება 1 ემიტირებულ გასაღებად;
- **Combine-Phase** არის ოფცია, რომელიც გამოიმუშავებს ლოკალურ Mini-Reducer-ის სახეობას. აქ გასაღებურ მნიშვნელობათა წყვილს აქვს იგივე გასაღები, რაც ქონდა ლოკალურად და ენიჭება მნიშვნელობა Mapper #2;
- **Reduce-Phase**-ში ყველა ბლოკი შეფასებულია თანაბრად, მინიჭებული აქვს ერთნაირი გასაღები ერთნაირი რედუსერი, რაც უზრუნველყოფს იმას რომ ბლოკებში მონაცემთა დათვლა მოხდება ერთად. მაგალითად, სიმბოლო a და b განმეორდა 2-ჯერ, ხოლო c სამჯერ.

1.35 ნახაზზე მოცემულია MapReduce და HDFS კომბინირებული არქიტექტურის სქემა.



ნახ.1.35. MapReduce და HDFS კომბინირებული არქიტექტურა

- ***JobTracker*** გამოიყენება Hadoop-კლასტერში როგორც მასტერ-კომპონენტი, ყველა ქმედების და რესურსის მართვისათვის. მას ზუსტად იგივე დანიშნულება აქვს MapReduce-ში, რაც NameNode - ს განაწილებულ ფაილურ სისტემაში (HDFS);
- ***TaskTracker*** არის სლეივის კომპონენტი, რომელიც ყოველი სერვერის კლასტერში აღწევს და, ფაქტობრივად, ივსება MapRedus-Job -იდან, სისტემაში მას აქვს ერთგვარი მეთვალყურის ფუნქცია;
- ***ამოცანები*** იმართება ლოკალურად *TaskTracker* -იდან, ყოველი კლასტერული კვანძი იმართება და პასუხობს იმ მნიშვნელოვან ჯავა ვირტუალურ მანქანას (JVM), რომელშც როგორც პროცესი Mapper-იან Reducer-ი გამოიყენება.

MapReduce-ს მუშაობის პრინციპის განსაზღვრისათვის თავიდან ჯავა ენაზე პროგრამირდება MapReduce-ს ლოგიკა. პროგრამული კოდი აფასოებს შესასრულებელ სამუშაოს სპეციფიური კონფიგურაციით, ანიჭებს მას JAR-მნიშვნელობას და აგზავნის NameNode-ში. მონაცემთა შევსებისას პერიოდულად იგზავნება შესაბამისი ინფორმაცია *JobTracker*-ში.

შესაძლებელია, რომ ამოცანა იყოს არასაკმარისად კოორდინირებული. სწორად შევსებული და ფორმულირებული ამოცანა დამოკიდებულია იმ ფაზებზე, რომელიც სრულდება განსხვავებული ღონისძიებების ჩატარებისას.

საბოლოოდ გარანტირებულად ხდება დავალების შესრულება და შედეგები ინახება ფაილურ სისტემაში, რომელიც შეიძლება გამოყენებული იქნას ან გადაიგზავნის შემდგომი დამუშავებისათვის.

- **Hadoop-ის სხვა მნიშვნელოვანი კომპონენტები:**

იმ ძირითადი კომპონენტების გვერდით, რომლებიც უზრუნველყოფს მონაცემთა შენახვას და დამუშავებას, არის ისეთი კომპონენტებიც, რომლებიც Hadoop სისტემას ხდის მოქნილს და მძლავრს - როგორც Hadoop ეკოსისტემას.

Apache Pig - ჯავა ენაზე ბაზირებული პროგრამული პლატფორმა, რომელსაც იყენებენ მონაცემთა მასივების ანალზის მიზნით. იგი შედგება აბსტრაქტული სკრიპტების ენისგან (*Pig Latin*), რომელიც Hadoop-ში უზრუნველყოფს მონაცემთა მანიპულირების ოპერაციების შესრულებას და ასევე აბსტრაქტულად ახდენს მონაცემთა ნაკადების აღწერას MapReduce -ში. მას ასევე უწოდებენ მონაცემთა დინამიკური მართვის ფრეიმვორკს [49].

Apache Hive - აბსტრაქტული თვალსაზრისით არის ინსტრუმენტი, რომელიც MapReduce - ფრეიმვორკს უზრუნველყოფს მონაცემთა საცავებით, რაც ხელს უწყობს Hadoop-ში მონაცემთა წარმოდგენას. Hive არის SQL-ის მსგავსი ენა (HiveQL), რომლიც შესაძლებელს ხდის მონაცემთა აგრეგაციას, ანალიზს და შემდგომ მათ განაწილებულ ფაილურ სისტემაში გადაგზავნას. ენა HiveQL კვლავ MapReduce-ში ახდენს მონაცემთა ტრანსფორმირებას და შევსებას [50].

Apache Hcatalog. Hcatalog-არის ინსტრუმენტი, რომელიც აკავშირებს ერთმანეთისგან აბსოლუტურად განსხვავებულ პროცესებს. მას აქვს ცენტრალური ცხრილის სახე და მეტამონაცემთა მართვის სერვისი, რომელიც აღწერს Hadoop-ში წარმოდგენილ მონაცემებს. ერთხელ აღწერილი მონაცემები, შესაძლებელია გამოყენებული იქნას როგორც Pig, ისე Hive. იგი გაცილებით მოსახერხებელს ხდის სამუშაოთა ჯაჭვის შესრულებას MapReduce - ში. მაგალითად, Pig ასრულებს მონაცემთა იმპორტირების პროცესს, ხოლო Hive მონაცემთა დარეგულირებას და დამუშავებას [51].

Apache HBase არის არარელაციური სკალირებული მონაცემთა ბაზა, რომელიც მსგავსია NoSql ტიპის მონაცემთა ბაზების სისტემისა. Apache Hbase ბაზირებულია დიდ ცხრილებზე (BigTable) და მონაცემთა შენახვა ხდება სვეტოვანი ორიენტაციით. Hbase-ი ეყრდნობა განაწილებულ ფაილურ სისტემას (HDFS), ამიტომ მონაცემთა შენახვა ხდება შეცდომებით, რადგან მონაცემები კომბინირებულია დიდ განაწილებულ მონაცემთა მასივებთან. ამის

გარდა Hbase-ი აფართოებს Hadoop-სისტემის ტრანზაქციების მენეჯმენტს, მონაცემთა განახლების, წაშლისა და დამატების ოპერაციების შესრულებისას [52].

Apache ZooKeeper არის პროგრამული ინტერფეისის აპლიკაცია, რომელიც უზრუნველყოფს განაწილებული პროცესების სინქრონიზაციას. Hadoop-კლასტერის მთელი განაწილებული პროცესი უნდა იყოს კოორდინირებული. განაწილებული პროცესებისას ZooKeeper-ი ააქტიურებს, ინახავს და ანაწილებს მნიშვნელოვან საკონფიგურაციო ინფორმაციას, იგი ამცირებს მონაცემთა დამუშავებაზე გამოყოფილ დროს, რის შედეგადაც იზოგება ელ-დენის დანახარჯები. ამდენად აღნიშნული ინსტრუმენტი არის ეკოეფექტური. ამასთან ZooKeeper არ არის შემოსაზღვრული მხოლოდ Hadoop-ით, მისი გამოყენება აქტიურად ხდება აგრეთვე Twitter-ში [53].

Apache Ambari არის ინსტრუმენტი, რომელიც უზრუნველყოფს Hadoop-ის უსაფრთხო ინსტალაციას, კლასტერების ადმინისტრირებასა და მონიტორინგს, რომელიც ათასობით კვანძს შეიცავს. დიდი ხნის განმავლობაში იყო ერთ-ერთი დომენი კომპანია კლაუდერაში. Apache Ambari ინსტრუმენტის გამოყენებით შესაძლებელია ინტუიციური ვებ ინტერფეისის წარმოდგენა [54].

Apache Sqoop. როგორ ფუნქციონირებს Hadoop, თუ მარტივად არ ხდება მონაცემთა ინტეგრირება ? და თუ დღემდე ფირმების უმეტესობა მონაცემებს ინახავს მონაცემთა რელაციურ ბაზებში, ამ

შემთხვევაში ყველაზე კარგი გამოსავალია Apache Sqoop-ის გამოყენება. იგი არის როგორც შუამავალი ექსპორტირებულ და იმპორტირებულ მონაცემებს შორის. Apache Sqoop ინსტრუმენტია, რომელიც ეფექტურად ცვლის მონაცემებს რელაციურ მონაცემთა ბაზებსა და Hadoop-ს შორის [55].

Apache Flume, განსხვავებით რელაციური მონაცემებისა, ხშირად ჩნდება მოთხოვნა და სურვილი, რომ Hadoodp-ში მოხდეს არალერაციური მონაცემების შენახვა და დამუშავება. სწორედ ასეთ შემთხვევებში გამოიყენება Apache Flume. იგი აგროვებს შემთხვევით და განაწილებულ მონაცემებს, ახდენს მათ აგრეგირებას და გადააქვს Log-Daten-ში, სადაც ხდება მათი ტრანსფორმაცია [56].

Apache Mahout არის ბიბლიოთეკა, სკალირებუ-ლი მანქანური სწავლების ალგორითმი, რომელიც იმპლემენტირებულია ApacheHadoop-ში. Mahout-ის საბაზისო სისტემაა მძლავრი რეკომენდაციათა ერთობლიობა. მაგალითად, ონლაინ გაყიდვებისას აღნიშნული სისტემის გამოყენებით კომპანია ამაზონმა წინასწარ იცის, თუ რა სურს მის კლიენტებს [57].

II თავი
NoSQL მონაცემთა ბაზების აგების ძირითადი პრინციპები

2.1. მონაცემების ასახვა და მათი დამუშავება

რელაციურ მოდელში მონაცემები აისახება (Data Representation,) ერთი სტრუქტურით (ERM – Entity-Relationship Model), აპლიკაციაში კი გამოიყენება სრულიად განსხვავებული სტრუქტურით. ამგვარად, მონაცემთა სტრუქტურა, რომელიც წარმოდგენილია ბაზაში, აბსოლუტურად განსხვავდება ოპერატიულ მეხსიერებაში მისი შესაბამისი სტრუქტურისაგან, რომელსაც შემდგომ იყენებს აპლიკაცია (დანართი).

პროგრამული აპლიკაცია მონაცემთა დამუშავების პროცესში საკმაოდ დიდ დროს და რესურსს უთმობს შემდეგ ოპერაციებს:

- მონაცემთა წაკითხვა რელაციური ბაზის სხვადასხვა ცხრილებიდან (Tables);
- ცხრილების გაერთიანება (Join) აპლიკაციისათვის საჭირო ობიექტში;
- მიღებული ობიექტის დამუშავება (პროექცია, შეზღუდვა და სხვა რელაციური ოპერაციები) და მისი გამოყენება;
- საშედეგო ობიექტის დაშლა და ცვლილებების განთავსება შესაბამის ცხრილებში (ბაზის მთლიანობის შენარჩუნებით).

რაც შეეხება NoSQL ტიპის დოკუმენტურ მონაცემთა ბაზას, როგორიცაა მაგალითად, MongoDB, იგი მონაცემებს ინახავს JSON (JavaScript Object Notation) ფორმატში [20]. უფრო ზუსტად, BSON-ში, რომელიც არის JSON -ის ბინარული წარმოდგენა. დოკუმენტების მონაცემები MongoDB-ში აისახება წყვილით: „ველი-მნიშვნელობა“. მაგალითად, ველის დასახელება, მოთავსებული ორმაგ ბრჭყალებში, შემდეგ „ : “ და ბოლოს, მნიშვნელობა ორმაგ ბრჭყალებში. მნიშვნელობა შეიძლება იყოს ისევ დოკუმენტი,

მასივები და დოკუმენტების მასივი. თითოეული წყვილი გამოიყოფა მძიმით. დოკუმენტები თავსდება ფიგურულ ფრჩხილებში “{ }“, მასივები კი - კვადრატულ ფრჩხილებში “[ ]“. ქვემოთ მოცემულია დოკუმენტის სტრუქტურის მქონე მონაცემთა ბაზა, რომელიც MongoDB – თვისაა დაწერილი,

```
“category”:
  {
     “cat_ID” : “<integer val>”,
     “name” : “<string val>”
  }
“products”:
  {
     “pr_ID” : “<integer val>”,
     “Name” : “<string val>”,
     “price”  :  “<number val>”,
     “cat_ID” : [“<integer val>”]
  }
```

განვიხილოთ მოთხოვნების ფორმირება ჩვენ ბაზებთან, MySQL და MongoDB შემთხვევაში:

1) „ვიპოვოთ ბაზაში არსებული თითოეული კატეგორიის დასახელების ყველა პროდუქტის დასახელება“.

- MySQL-ში:

```
SELECT MAX(price) FROM products;  // მაქსიმუმი
SELECT MIN(price) FROM products;  // მინიმუმი
```

MongoDB-ში:

```
db.products.find().sort({price:-1}).limit(1)     //     მაქსიმუმი
db.products.find().sort({price:+1}).limit(1)     //     მინიმუმი
```

2) „ვიპოვოთ პროდუქტის საერთო რაოდენობა კონკრეტული კატეგორიისთვის (cat_ID = 4)“.

- MySQL-ში:

```
SELECT  count ( pr_ID)
```

```
FROM    products p, category c
WHERE  c.cat_ID = p.cat_ID and c.cat = 4;
```

- MongoDB-ში:

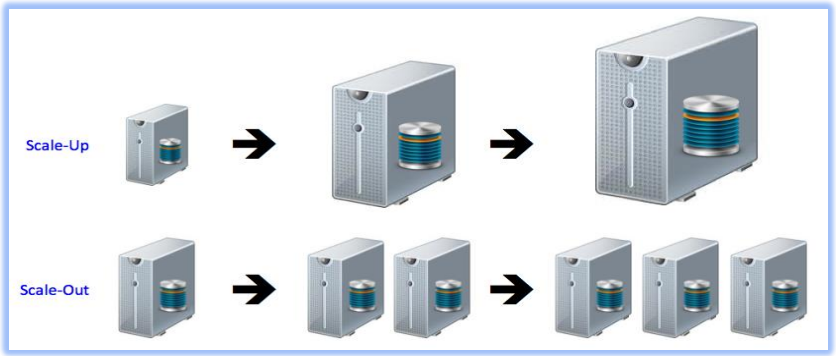
```
db.products.count ( {cat_ID = 4 } )
```

2.2. მონაცემეთა შენახვის ტიპები

ინფორმაციული და კომუნიკაციური ტექნოლოგიების გამოყენების სფეროში საგრძნობლად გაიზარდა დიდი მოცულობის მონაცემებისა და პარალელური მოთხოვნების დამუშავების რაოდენობა, რაც ხშირად საჭიროს ხდის არსებული სისტემების არქიტექტურისა და კონფიგურაციების სრულიად შეცვლას.

ტრადიციულ რელაციურ ბაზებში, თუ იზრდება მოთხოვნების დატვირთვა, ვაძლიერებთ სერვერს (vertical scale). გარკვეული ზღვრის შემდეგ სერვერის განვითარება საკმაოდ ძვირი ჯდება, თანაც ნაკლებად ეფექტურია.

სპეციალისტები მივიდნენ იმ დასკვნამდე, რომ მცირე რაოდენობის მძლავრ სერვერებს სჯობს დიდი რაოდენობის იაფფასიანი სერვერები (horizontal scale), რომლებიც იმუშავებს როგორც ერთი კლასტერი (ნახ.2.1).



ნახ.2.1

NoSQL ბაზებში მონაცემების შენახვის 4 ძირითადი ტიპი არსებობს:

1. **Key-Value Store** – აქვს დიდი ჰეშ ცხრილი გასაღებებისა და მათი მნიშვნელობებისათვის. მონაცემებზე წვდომა გვაქვს პირველადი გასაღების (Primary Key) გამოყენებით. (მაგალითად, Riak, Amazon S3 [Dynamo], Redis…);
2. **Document-based Store** - ინახავს იარლიყიანი ელემენტე-ბისგან (tagged elements) შემდგარ დოკუმენტებს. ამ ტიპის მონაცემთა ბაზებს JOIN ოპერატორის მხარდაჭერა არ აქვთ. ცხრილების გადაბმის/გაერთიანების ლოგიკა აპლიკაციის მხარეს არის დასაწერი. სამაგიეროდ, ობიექტზე ორიენტირებული აპლიკაციისთვის ძალიან მარტივია მთელი საჭირო ინფორმაციის ერთი დოკუმენტიდან ამოღება. (მაგალითად, MongoDB, CouchDB…);
3. **Column-based Store** - მეხსიერების თითოეული ბლოკი ინახავს მხოლოდ ერთი სვეტის ინფორმაციას, რაც ბევრად ამარტივებს აგრეგატული ფუნქციების (MIN, SUM, AVG, COUNT…) შესრულებას. ამ ტიპის მონაცემთა ბაზები კვლავ ცხრილურ სტრუქტურას იყენებენ, მაგრამ JOIN ოპერატორის მხარდაჭერა არ აქვთ. ცხრილების გადაბმის/გაერთიანების ლოგიკა აპლიკაციის მხარეს არის დასაწერი. (მაგალითად, HBase, Cassandra);
4. **Graph-based** ქსელური ტიპის მონაცემთა ბაზაა, რომელიც იყენებს წიბოებსა და კვანძებს მონაცემების შესანახად და წარმოსადგენად. მაგალითად, Neo4.

დეტალურად განვიხილოთ მონაცემთა შენახვის თითოეული ტიპი.

2.4.1. Key-Value Store

Key-Value ყველაზე მარტივი სტრუქტურაა: გვაქვს უნიკა-ლური პარამეტრი key და შესაბამისი მნიშვნელობა - value (ცხრ.2.1).

ცხრ.2.1

გავრცელებული სისტემები	Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB, Amazon SimpleDB, Riak
ტიპური გამოყენება	Content caching (განაწილებულ სერვერებზე დიდი ოდენობის ინფორმაციის დამუშავებაზე ორიენტირება), ლოგ–სერვერები...
მოდელი	Key-Value წყვილების კოლექცია
უპირატესობა	სწრაფი ძებნა
სისუსტე	შენახულ მონაცემებს არ აქვს სქემა

სიმარტივისთვის რომ წარმოვიდგინოთ, შესაბამის ლოგიკას რელაციურ ბაზაში ორი სვეტით ავაწყობდით, – Primary key და მასთან დაკავშირებული სვეტი მონაცემებისთვის. განსხვავება ის იქნებოდა, რომ რელაციურ ბაზაში წინასწარ უნდა მიგვეთითებინა მონაცემების სვეტის ტიპი, NoSQL ბაზებში კი ეს მნიშვნელობა შეიძლება იყის რიცხვი, ტექსტი, სურათი და ამასთან ერთად, არ არის აუცილებელი, რომ ყველა სტრიქონში ერთსა და იმავე ტიპის ინფორმაციას ვინახავდეთ.

მონაცემების ამგვარი შენახვა გვაძლევს იმის საშუალებას, რომ ერთი ცხრილის სხვადასხვა ჩანაწერები კლასტერის სხვადასხვა კვანძებზე მოვათავსოთ. როგორც წესი, ეს გადანაწილება გასაღების მნიშვნელობის მიხედვით ხდება.

Key-Value ტიპის ბაზებში რთულია ატომარობისა და კონსისტენტურობის დაცვა – სანამ ერთი მომხმარებელი ჩანაწერს ანახლებს, სხვა მომხმარებელმა შეიძლება წასაკითხად მიაკითხოს იმავე ჩანაწერს. ამგვარ შემთხვევებში გვაქვს ორი ვარიანტი – მივაწოდოთ მომხმარებელს ჩანაწერის ბოლო განახლებული ვერსია, ან მივაწოდოთ ყველა არსებული ვერსია და თავად მივცეთ იმის საშუალება, რომ გაარკვიოს, რომელი ვერსიის გამოყენება ურჩევნია.

ამგვარად, აპლიკაციის მხარეს მეტნაკლებად შესაძლებელი ხდება კონსისტენტურობის დაცვაც, მაგრამ თუ დასმული ამოცანისთვის კრიტიკულია ბაზის ტრანზაქციულობა, მაშინ key-value strorage საიმედო გადაწყვეტილება ვერ იქნება.

როგორც წესი, key-value მეთოდი იყენებს ჰეშ ცხრილებს, რომლებშიც ინფორმაცია ლოგიკურ გაერთიანებებად (bucket) არის დაყოფილი. რეალურად, გასაღები არის ჩვენს გასაღებს + Bucket მნიშვნელობების კონკატენაციის ჰეში – hash (Bucket+ Key)

CAP თეორემას თუ მივუბრუნდებით, ცხადი ხდება, რომ key-value მეთოდი იდეალურია Availability და Partition მხარდასაჭერად. მაგრამ საგრძნობლად მოიკოჭლებს Consistency-ის კუთხით.

Key-value storage-ს საკმაოდ კარგი გამოყენებაა სესიის ინფორმაციის შენახვა, როგორც key - სესიის იდენტიფიკატორი და value - სესიასთან დაკავშირებული ინფორმაციის ნაკრები.

ასევე, საინტერესო მაგალითია მომხმარებლის პირადი ინფორმაციის შენახვა, სადაც key გასაღები არის მომხმარებლის უნიკალური იდენტიფიკატორი, value მნიშვნელობა კი მთელი ის ინფორმაცია, რაც ახასიათებს მომხმარებელს.

2.2 ცხრილში Key-value პრინციპით შეტანილია საქართველოს ბანკის ფილიალები დასახლებების მიხედვით.

ცხრ..2.2

Key	**Value**
“ლანჩხუთი”	{“ჟორდანიას ქუჩა #101"
“ქობულეთი”	{“ნინოშვილის ქუჩა #1"}
“თბილისი”	{“ვეკუას ქუჩა #1თბილისი","პუშკინის ქუჩა #3თბილისი","კოსტავას ქუჩა #24თბილისი","თაბუკაშვილის ქუჩა #38თბილისი"}

უნდა გავითვალისწინოთ, რომ თუ გასაღებად ტექსტურ მნიშვნელობებს ავირჩევთ, დროთა განმავლობაში გაგვიჭირდება უნიკალურობის დაცვა.

Key-value ტიპის ბაზა მონაცემების დასამუშავებლად მომხმარებელს შემდეგ ფუნქციებს სთავაზობს:

- Get(key), აბრუნებს პარამეტრად მიღებული key გასაღების შესაბამის value მნიშვნელობას;
- Put(key, value), აკავშირებს მნიშვნელობას გასაღებთან;
- Multi-get(key1, key2, .., keyN), აბრუნებს მიღებული გასაღებების შესაბამისი მნიშვნელობების მიმდევრობას;
- Delete(key), მონაცემთა ბაზიდან ამოშლის შესაბამის ჩანაწერს.

გასაღები-მნიშვნელობა მოდელს თუ რელაციურ მოდელს შევადარებთ, შემდეგ შესაბამისობებს მივიღებთ:

- Table -> bucket
- Row -> key-value
- Rowid -> key

2.4.2. Document-based Store

მონაცემების წარმოდგენის დოკუმენტზე ორიენტირებული ტიპები, key-value ტიპის მსგავსად, ინფორმაციას ინახავს. განსხვავება იმაშია, რომ დოკუმენტებად შენახულ ინფორმაციას გარკვეული სტრუქტურა და წარმოდგენის განსხვავებული სახე გააჩნია (ცხრ.2.3).

ცხრ.2.3

გავრცელებული სისტემები	CouchDB, MongoDb
ტიპური გამოყენება	ლოგ–სერვერები, ვებ აპლიკაციები (Key-value–ს დახვეწილი ვერსია)

მოდელი	Key-Value წყვილების კოლექციათა კოლექციები (ჩადგმული კოლექციები)
უპირატესობა	ფუნქციონირებს არასრული ინფორმაციის შემთხვევაშიც
ნაკლოვანება	ტრანზაქციების წარმადობა; არ აქვს სტანდარტული სინტაქსი მოთხოვნების ჩამოსაყალიბებლად.

Document-based მონაცემთა ბაზები ინფორმაციას List<Object> სიის სახით ინახავს. ამგვარ სიებში შესაძლებელია ჩაიწეროს მრავალი სხვადასხვა სახის მონაცემი.

მაგალითად, Document-based მონაცემთა ბაზები შეიძლება შევადაროთ ფოლდერს, რომელშიც Ms Word-ის მრავალი ფაილია მოთავსებული, თითოეულ ფაილში კი სხვადასხვა სტრუქტურა და შიგთავსია. თითოეული ფოლდერი ჩვენთვის დოკუმენტების კოლექცია იქნება (collection).

დოკუმენტებზე ორიენტირებულ მოდელს თუ რელაციურ მოდელს შევადარებთ, შემდეგ შესაბამისობებს მივიღებთ:

Table -> collection
Row -> BSON document
Column -> BSON Field
Rowed -> _id
Index -> Index
Join -> ჩადგმული დოკუმენტი (Embedded Document)
Partition -> Shard
Partition Key -> Shard Key

2.4.3. Column-based Store

Column-based Store პრინციპი შემუშავებულ იქნა მრავალ მანქანაზე გადანაწილებული დიდი რაოდენობის მონაცემების დასამუშავებლად (ცხრ.2.4).

ცხრ.2.4

გავრცელებული სისტემები	Cassandra, HBase, Riak
ტიპური გამოყენება	განაწილებული ფაილური სისტემები Hadoop Distributed File System (HDFS)
მოდელი	სვეტი –> სვეტების გაერთიანება
უპირატესობა	სწრაფი ძიება, განაწილებული გარემოს საუკეთესო მხარდაჭერა
ნაკლოვანება	დაბალი დონის API

2.5. მონაცემთა მთლიანობა

მონაცემთა ბაზების მართვის სისტემები უზრუნველყოფს ინსტრუმენტებს მონაცემთა მთლიანობის (Data Integrity) შესანარჩუნებლად. საბაზისო წესების მნიშვნელოვან სიმრავლეს, რომელიც უზრუნველყოფს ბაზის მონაცემთა მთლიანობას და არაწინააღმდეგობრივობას, კავშირების (მინიშნებების) დონეზე, მთლიანობის შეზღუდვები ეწოდება (Referential Integrity Constraints) [2,3].

მონაცემთა თანმიმდევრულობისა და მთლიანობის დამცავი პრინციპები რეალიზებულია შემდეგი ჩაშენებული წესებით:

1. ახალი სტრიქონის დამატება მშობელ-ცხრილში ყოველთვის ნებადართულია;
2. ახალი სტრიქონის დამატება შვილ-ცხრილში ნებადართულია მხოლოდ იმ შემთხვევაში თუ აქ შესაბამისი გარე გასაღები არსებობს მშობელ-ცხრილისკენ;

3. მშობელი-ცხრილიდან სტრიქონის ამოშლა ნებადართულია მხოლოდ იმ შემთხვევაში თუ მას არ გააჩნია შვილი სტრიქონები შვილ-ცხრილში;

4. შვილი-ცხრილიდან სტრიქონის ამოშლა ყოველთვის არის ნებადართული;

5. პირველადი გასაღების განახლება მშობელ-ცხრილში ნებადართულია მხოლოდ მაშინ, თუ არ არსებობს შვილი სტრიქონები;

6. შვილი სტრიქონის გარე გასაღების (Foerign key) განახლება ნებადართულია იმ შემთხვევაში, თუ ამ გარე გასაღების შესაბამისი პირველადი გასაღების (Primary key) ახალი მნიშვნელობა არსებობს მშობელ-ცხრილში.

ხშირ შემთხვევებში ბიზნეს-წესების უზრუნველსაყოფად საჭირო ხდება დამატებითი შეზღუდვების შემოტანა [23].

მონაცემთა ბაზების მართვის სისტემები იძლევა კიდევ ერთ შესაძლებლობას მონაცემთა მთლიანობის უზრუნველსაყოფად. ამ შესაძლებლობას ტრანზაქცია (transaction) ეწოდება.

ტრანზაქცია არის მონაცემთა ბაზის ერთმანეთთან დაკავშირებული ცვლილებების დაჯგუფების სპეციალური მექანიზმი, რომელიც გამოიყენება იმ შემთხვევაში, როდესაც უნდა განხორციელდეს ყველა ცვლილება, ან საერთოდ არცერთი ცვლილება არ უნდა შესრულდეს.

მონაცემთა მართვის სისტემა ნებას აძლევს მომხმარებელს (ან პროგრამისტს) განსაზღვროს ტრანზაქციის საზღვრები. მონაცემთა ბაზის ყველა ცვლილება, რომელიც გათვალისწინებულია ტრანზაქციის საზღვრების შიგნით ან უნდა განხორციელდეს წარმატებულად, ან ტრანზაქცია მთლიანად უნდა დაბრუნდეს უკან. როდესაც ხდება ტრანზაქციის უკან დაბრუნება, ყველა სვეტის მნიშვნელობები უბრუნდება იმ მნიშვნელობებს, რომლებიც მათ ჰქონდათ ტრანზაქციის დაწყებამდე.

ტრანზაქციების განხორციელება ეფუძნება მონაცემთა წინასწარ რეგისტრაციას. ტრანზაქციის დასაწყისში მონაცემთა ბაზებში შესატანი ახალი მნიშვნელობები (ანუ ცვლილებები) და ბაზაში არსებული საწყისი მნიშვნელობები (ანუ ის მნიშვნელობები, რომლებიც უნდა შეიცვალოს ახალი მნიშვნელობებით) ჩაიწერება სარეგისტრაციო ჟურნალში და არა მონაცემთა ბაზაში. როდესაც ტრანზაქცია მთლიანად წარმატებულად შესრულდება, მაშინ ის ფიქსირდება (ანუ მონაცემთა მართვის სისტემა აფიქსირებს წარმატებულ ტრანზაქციას). ამ ფაზაში ის ცვლილებები, რომლებიც იყო ჩაწერილი სარეგისტრაციო ჟურნალში, გადადის მონაცემთა ბაზაში და შესაბამისი ცვლილებები ხილვადი ხდება სხვა მომხმარებლებისათვის. მეორეს მხრივ, თუ ტრანზაქციის რომელიმე ნაწილი ვერ განხორციელდა (ანუ მისი განხორციელება ჩავარდა) ნებისმიერი მიზეზის გამო, მაშინ ცვლილებები ბრუნდება უკან, ანუ არცერთი ცვლილება, ჩაწერილი სარეგისტრაციო ჟურნალში, არ გადადის მონაცემთა ბაზაში.

წინასწარი რეგისტრაცია ასევე სასარგებლოა მონაცემთა ბაზის აღდგენისას ავარიული სიტუაციის შემდეგ. სარეგისტრაციო ჟურნალი (log) მოიცავს მონაცემთა ბაზაში განხორციელებულ ყველა ცვლილებას, იმ ინფორმაციის ჩათვლით დაფიქსირდა თუ უკან დაბრუნდა თითოეული ტრანზაქცია. მონაცემთა ბაზის აღსადგენად ადმინისტრატორს შეუძლია აღადგინოს მონაცემთა ბაზის წინამორბედი *სარეზერვო ასლი* და გაიმეოროს დაფიქსირებული ტრანზაქციები. ამას ეწოდება მონაცემთა ბაზის აღდგენა დაფიქსირებული ტრანზაქციების გამეორებით (roll forward recovery).

ზოგიერთი მონაცემთა ბაზის მართვის სისტემა იყენებს წინასწარ რეგისტრაციას (ცვლილებების წინსწრებით რეგისტრაციას), მაგრამ ამასთან ერთად ახორციელებს მონაცემთა ბაზის

ფაქტობრივ ცვლილებას ტრანზაქციის ფორმალურ დაფიქსირებამდე. ასეთ სისტემებში ავარიული სიტუაციის შემდეგ აღდგენა შესაძლებელია განხორციელდეს მონაცემთა ბაზების მართვის სისტემის გადატვირთვით და სარეგისტრაციო ჟურნალში არსებული ყველა იმ ტრანზაქციების გაუქმებით, რომლებიც არ დაფიქსირდა. ასეთ მიდგომას დაუფიქსირებელი ტრანზაქციების გაუქმებით აღდგენას უწოდებენ (rollback recovery).

2.6. ტრანზაქციის იზოლირების დონეები

როდესაც მონაცემთა ბაზას ერთდროულად მიმართავს მრავალი მომხმარებელი, არსებობს იმის შესაძლებლობა, რომ ერთი პიროვნების მიერ შესრულებული ცვლილებები ზემოქმედებას მოახდენს სხვა პიროვნების მუშაობაზე [16]. მაგალითად წარმოვიდგინოთ, რომ ორი პიროვნება ერთდროულად იმყოფება ავიაბილეთების შეკვეთათა ცხელი ხაზის სისტემაში (on-line flight reservation system), ორივე ხედავს, რომ ფანჯარასთან მდებარე ადგილი მე-18 რიგში ჯერ კიდევ თავისუფალია, და ორივე უკვეთავს ამ ადგილს თითქმის ერთდროულად. შესაბამისი კონტროლის გარეშე ორივე პიროვნება დარწმუნებული იქნება, რომ მათი ადგილი შეკვეთილია, მაგრამ ერთერთი მათგანი იმედგაცრუებული დარჩება. მოყვანილი მაგალითი წარმოადგენს დამთხვევათა პრობლემების ერთ-ერთ სახეს, რომელსაც დაკარგულ განახლებათა პრობლემა ეწოდება (the lost update problem).

გარდა აღნიშნული პრობლემისა არსებობს სხვა პოტენციალური პრობლემებიც. მაგალითად, შეცდომითი წაკითხვა (მცდარი მონაცემების წაკითხვა - **dirty reads**) ხდება მაშინ, როდესაც ერთი ტრანზაქცია კითხულობს მეორე დასაფიქსირებელი ტრანზაქციით შეცვლილ მონაცემებს და ეს მეორე ტრანზაქცია მოგვიანებით უკან ბრუნდება ყველა შესაბამისი ცვლილებების გაუქმებით.

მაგალითად, სესია1–მა დაიწყო ტრანზაქცია და შეცვალა მონაცემები. სესია2–მა წაიკითხა უკვე შეცვლილი მონაცემები, რომელიც სესია1–ს ჯერ დამახსოვრებული (commited) არ ჰქონდა. რეალურად, სესია2–მა წაიკითხა ინფორმაცია, რომელიც რაიმე პრობლემის შემთხვევაში შეიძლება უკან დაბრუნდეს (roll back). ამგვარ შემთხვევაში სესია2–ს ექნება დამახინჯებული მონაცემები (dirty data).

განსხვავებული სახის პრობლემაა არაგანმეორებადი წაკითხვა (**nonrepeatable read**). ეს პრობლემა წამოიჭრება მაშინ, როცა ტრანზაქცია რამდენიმეჯერ კითხულობს ერთსა და იმავე მონაცემებს. პირველი ტრანზაქციის შესრულების დროს თუ მეორე ტრანზაქციამ მონაცემები შეცვალა, მაშინ პირველი ტრანზაქცია მონაცემების განმეორებით წაკითხვისას სხვა მნიშვნელობებს მიიღებს.

მაგალითად, სესია1–მა დაიწყო ტრანზაქცია და მიიღო გარკვეული სტრიქონი. სესია2–მა განაახლა ზუსტად იგივე სტრიქონი. სესია1–მა ხელმეორედ გაუშვა წინა ტრანზაქცია, მაგრამ შედეგი უკვე განსხვავებული მიიღო.

მსგავს პრობლემას წარმოადგენს მოჩვენებითი წაკითხვა (phantom read). ამ პრობლემას ადგილი აქვს მაშინ, როცა პირველი ტრანზაქცია ცხრილიდან ირჩევს სტრიქონებს, მეორე ტრანზაქცია კი ამავე ცხრილში სვავს სტრიქონებს პირველი ტრანზაქციის მუშაობის დამთავრებამდე. შედეგად, პირველი ტრანზაქციის მიერ წასაკითხი სტრიქონები არაკორექტული იქნება. თუ თავისი მუშაობის დროს ერთი ტრანზაქცია კითხულობს ჩანაწერების რომელიმე სიმრავლეს ორჯერ, მაშინ მან შესაძლებელია წაიკითხოს ახალი ჩანაწერები მეორე წაკითხვისას. ეს შესაძლებელია მოხდეს იმ შემთხვევაში, თუ პარალელურად მონაცემთა ბაზაში სხვა ტრანზაქციას ახალი ჩანაწერები შეაქვს.

ამ ტიპის ყველა პრობლემის გადაჭრა ხორციელდება ჩაკეტვის მექანიზმებით (locking mechanisms), რომლებიც უზრუნველყოფს მომხმარებელთა და ტრანზაქციების ერთმანეთისაგან იზოლირებას.

ადრე პროგრამისტებს საჭირო დაცვის განსახორციელებლად ჩაკეტვების მართვის საკუთარი სისტემების შემოტანა სჭირდებოდათ, მაგრამ დღეს ჩაკეტვების მართვას მთლიანად მონაცემთა ბაზების მართვის სიტემა ახორციელებს.

დაკარგული განახლებების პრობლემას თანამედროვე მონაცემთა ბაზების მართვის სიტემები წაკითხვისა და ჩაწერის ჩაკეტვების მართვით ახორციელებს. სხვა შესაძლო პრობლემების გადასაწყვეტად ჩვენ უბრალოდ განვსაზღვრავთ საჭირო იზოლაციის დონეს იმ ოთხი სტანდარტული დონიდან, რომლებსაც ადგენს 1992 წლის SQL სტანდარტი. დაცვის ოთხი სხვადასხვა სტანდარტის არსებობა დაკავშირებულია იმ ფაქტთან, რომ რაც უფრო ძლიერია დაცვა (მაღალია იზოლაციის დონე) მით უფრო მცირეა მოქმედების სისწრაფე [34].

ტრანზაქციათა იზოლაციის დონეებია:

- Read uncommitted დონე არ უზრუნველყოფს დაცვას იმ დამთხვევათა პრობლემებისაგან, რომლებიც განვიხილეთ, მაგრამ უზრუნველყოფს მოქმედების მაქსიმალურ სისწრაფეს, რადგან იზოლაციის ამ დონეში არ ხდება მოთხოვნილი მონაცემების ბლოკირება (lock). ცხადია, ამ დონის გამოყენებისას მზად უნდა ვიყოთ, რომ თუ ერთი პროგრამიდან შეტანილი ცვლილებები უკან დაბრუნდება, update-სა და rollback-ს შორის შუალედში ყველა სხვა მოთხოვნას dirty data-ს ანუ დამახინჯებულ მონაცემებს გავატანთ. ამ დონეს გამოვიყენებთ ისეთი მოთხოვნებისთვის, რომელთა სისწრეც არ არის სასიცოცხლოდ აუცილებელი, ან თუ ვიცით, რომ სანამ ჩვენ ვკითხულობთ, მესამე მხარე ცვლილებას არ შეიტანს;
- Read committed დონე გარანტირებულად არიდებს თავს შეცდომითი წაკითხვების პრობლემას, რადგანაც იზოლაციის ამ

დონეზე ნებადართულია მხოლოდ დაფიქსირებული ტრანზაქციებით შეცვლილი მონაცემების წაკითხვა;

- Repeatable read დონე იცილებს ასევე არაგანმეორებადი წაკითხვების პრობლემასაც;
- Serializable დონე უზრუნველყოფს ერთდროული ტრანზაქციების სრულ განცალკევებას, რაც ხორციელდება ტრანზაქციაში მონაწილე ყველა სტრიქონის ჩაკეტვით. ფანტომების წარმოქმნის პრობლემა წყდება გასაღების დიაპაზონის დაბლოკვით. იზოლირების ამ დონის დაყენების შემთხვევაში მცირდება პარალელიზმის ხარისხი და იზრდება სისტემაზე დატვირთვა.

ჩვეულებრივ ყველაზე უფრო ხშირად გამოიყენებენ read committed იზოლაციის დონეს [35].

ცხრ.2.5

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
READ UNCOMMITTED	შესაძლებელია	შესაძლებელია	შესაძლებელია
READ COMMITTED	--	შესაძლებელია	შესაძლებელია
REPEATABLE READ	--	--	შესაძლებელია
SERIALIZABLE	--	--	--

იზოლაციის ეს დონეები შემუშავებულია 1992 წლის SQL სტანდარტით, მაგრამ რეალურად, სხვადასხვა მონაცემთა ბაზებში ტრანზაქციის იზოლირების დონეებს სხვადასხვა სახე აქვს. მაგალითად, Oracle მონაცემების წაკითხვისას lock-ს საერთოდ არ იყენებს და dirty data–ს პრობლემა latch-ებით აქვს გადაწყვეტილი.

Lock-ის შემთხვევაში რიგში ვდგებით, რომ მივიღოთ მონაცემებზე წვდომა, latch–ის დროს კი პერიოდულად ვაკითხავთ მონაცემებს და ვამოწმებთ, თუ არის თავისუფალი.

Oracle-ის ტრანზაქციის იზოლირების დონეებია [35]:

- Read Committed: ტრანზაქციის იზოლაციის ეს დონე არის default (დუმილით) მნიშვნელობა;
- Serializable: ტრანზაქციის იზოლირების ამ დონეზე არ გვაქვს დამახინჯებული მონაცემები (dirty data);
- Read Only: მხოლოდ ის მონაცემები არის ხელმისაწვდომი, რომლებიც ტრანზაქციის დაწყების მომენტში უკვე დასრულებული (commited) იყო.

DB2-ის ტრანზაქციის იზოლირების დონეებია [36]:

- Uncommitted Read: lock-ები არ გვაქვს, სამაგიეროდ მზად უნდა ვიყოთ დამახინჯებული ინფორმაციისთვის;
- Cursor Stability: გვაქვს მცირე ხანგრძლივობის ბევრი lock. ეს დონე არის DB2-სთვის default მნიშვნელობა. კითხვის დროს, ამ შემთხვევაში, ბლოკი ედება მხოლოდ მიმდინარე სტრიქონს. თუ B და C აპლიკაციები ერთსა და იმავე სტრიქონს კითხულობს, გამოიყენება shared lock, რაც იმის საშუალებას იძლევა, რომ ერთი და იმავე მონაცემის წასაღებად მისული აპლიკაციები ერთმანეთს არ დაელოდოს;
- Read Stability: გვაქვს დიდი ხანგრძლივობის ცოტა lock. წინა დონისგან განსხვავებით, RS იზოლაციის დონე ერთდროულად ბლოკავს ყველა მოთხოვნილ სტრიქონს;
- Repeatable Read: გვაქვს დიდი ხანგრძლივობის მქონე ბევრი lock. RR ყველაზე მკაცრი დონეა და მოთხოვნილ სტრიქონებთან ერთად ბლოკავს შედეგთან დაკავშირებულ ყველა ჩანაწერსაც.

Oracle, SQL Server და DB2-ში იზოლაციის დონე შეიძლება მიეთითოს კონკრეტული სესიის ან ტრანზაქციისთვის (DB2-ს დამატებით აქვს აპლიკაციის დონეც).

NoSQL სისტემებისთვის, კონკრეტულად კი MongoDB-სთვის ტრანზაქციების იზოლირების მსგავსი მეთოდები არ გვაქვს.

ნაშრომის შემდგომ თავებში დეტალურად განვიხილავთ იმ მეთოდებს, რომლებსაც NoSQL სისტემები იყენებს სტანდარტული პრობლემების ასაცილებლად.

2.7. ACID პრინციპები და CAP სამკუთხედის თეორემა

1970-იანი წლების ბოლოს ACID (Atomicity, Consistency, Isolation, Durability) ტესტის სახელით ჩამოყალიბდა ის კრიტერიუმები, რაც უმთავრესია ტრანზაქციის საიმედოობისთვის [6,16]:

- Atomicity (ატომარობა) – ტრანზაქციის პროცესი ან მთლიანად სრულდება (უწყვეტად) ან საერთოდ არ სრულდება. თუ ტრანზაქციის პროცესი ნაწილობრივ შესრულდა, მაშინ ატომარობა დარღვეულია;
- Consistency (მთლიანობა) – ტრანზაქციის პროცესი სრულდება მაშინ, როცა ბაზა მთლიანია. ტრანზაქციის პროცესის დასრულებისას ბაზა ისევ მთლიანობის მდგომარეობაში რჩება (მთლიანობა ნიშნავს, რომ თითოეული სტრიქონი და მნიშვნელობა უნდა შეესაბამებოდეს რეალურ სიტუაციას და უნდა სრულდებოდეს ყველა შეზღუდვა. მაგალითად, თუ წერია შეკვეთები და არ წერია საქონელი, მთლიანობის პრინციპი დარღვეულია);
- Isolation (იზოლირება) – ტრანზაქციის თითოეული პროცესი უნდა იყოს იზოლირებული, რაც ნიშნავს, რომ ყველა ტრანზაქციის პროცესი უნდა ხორციელდებოდეს ერთმანეთისაგან დამოუკიდებლად. ახალმა პროცესმა არ უნდა შეუშალოს ხელი უკვე დაწყებული ტრანზაქციის პროცესის დასრულებას. ეს პრინციპი განსაკუთრებით მნიშვნელოვანია, როცა ბაზასთან მუშაობს რამდენიმე მომხმარებელი;
- Durability (მდგრადობა) – გულისხმობს ტრანზაქციის პროცესის შესრულებას სისტემური შეფერხებებისაგან დამოუკიდებლად.

მონაცემთა ბაზების მართვის სისტემა ისე უნდა იყოს დაპროექტებული, ახალი ტრანზაქციის შესრულებისას შეფერხებების არსებობის შემთხვევაში, შესაძლებელი გახდეს ბოლოს, სრულად შესრულებული ტრანზაქციის პროცესის შემდეგ არსებული მდგომარეობის აღდგენა.

რელაციურ ბაზაში ნებისმიერი ოპერაცია ხორციელდება პრინციპით: *ყველაფერი ან არაფერი.*

არარელაციურ ბაზებში ACID გარანტია არ გვაქვს, მაგრამ შეგვიძლია სხვადასხვა ლოგიკით მივაღწიოთ ტრანზაქციების საიმედოობას. ამგვარი გვერდის ავლის ერთ–ერთი საუკეთესო მაგალითი ორფაზიანი განხორციელებაა (Two Phase Commits).

მართალია ACID გარანტია არ გვაქვს, მაგრამ სამაგიეროდ გვაქვს საშუალება, ავირჩიოთ CAP თვისებები, მაგალითად, 100%–იანი წვდომა.

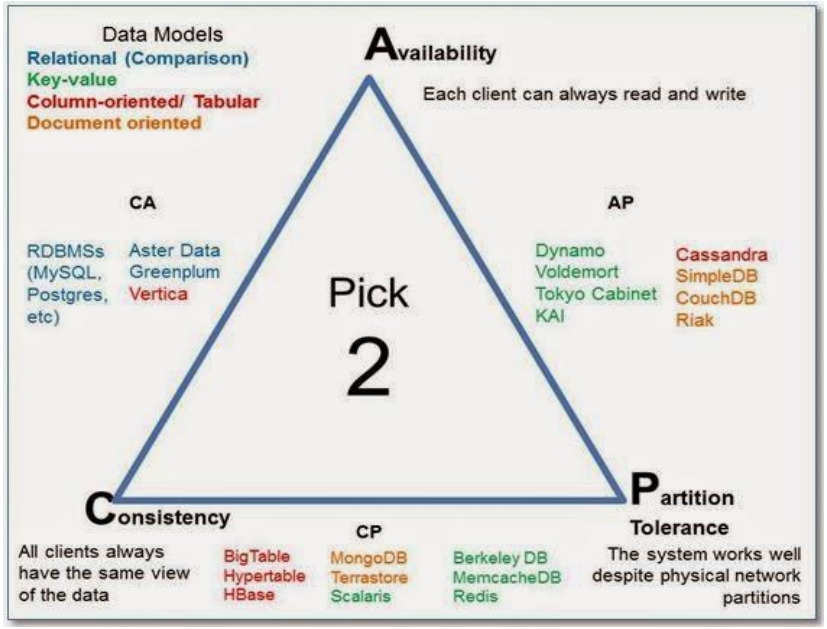
მონაცემთა ბაზებთან მუშაობისას შესაძლოა გვქონდეს “network partition”-ები, რომლებიც გულსიხმობს კლასტერში კვანძებს (node) შორის კომუნიკაციის დაკარგვას და შესაბამისად ინფორმაციის არასინქრონულობას [6,7] (ნახ.2.2).

CAP თეორემა (Consistency, Availability, Partition_tolerance) ან ბრიუერის (*Brewer Eric*) თეორემა არის ევრისტიკული მტკიცება იმის შესახებ, რომ განაწილებული გამოთვლების ნებისმიერ რეალიზაციაში შესაძლებელია სამი (C,A,P) თვისებიდან მხოლოდ ორის უზრუნველყოფა (ნახ.2.2) [8].

Consistency (მთლიანობა) – განაწილებული კომპიუტერული სისტემის ყველა კვანძში (node – კლასტერში) მონაცემები შეთანხმებულია (არაწინააღმდეგობრივია დროის მოცემულ მომენტში);

Availability (წვდომადობა) – ნებისმიერი მოთხოვნა განაწილებულ სისტემასთან სრულდება კორექტული პასუხით (თუ რომელიმე კვანძი გათიშულია, მას ცვლის სხვა);

Partition Tolerance (განაწილების მდგრადობა) – კლასტერი განაგრძობს ფუნქციონალობას მაშინაც კი როდესაც კვანძებს შორის არის კომუნიკაციის პრობლემა.

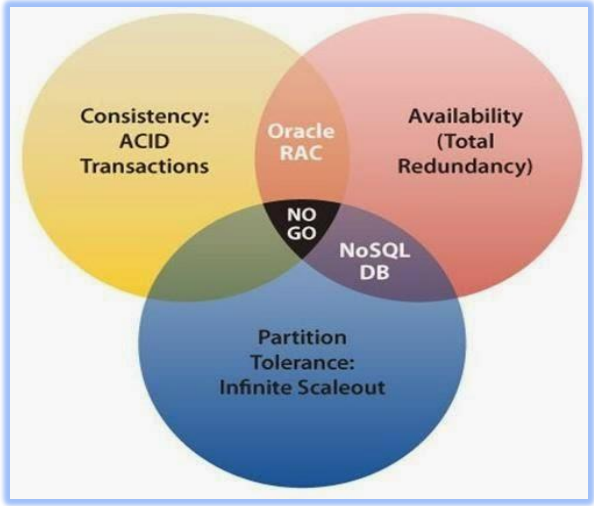


ნახ.2.2

მოვიყვანოთ დეტალური მაგალითები CAP სამკუთხედზე (ნახ.2.3). როდესაც კომუნიკაციის პრობლემაა, არჩევანი გვაქვს:

1. ან დავუშვათ, რომ არასინქრონირებული მონაცემები იყოს ბაზაში (დავივიწყოთ Consistency);

2. ჩავთვალოთ რომ კლასტერი დაზიანდა, ანუ დავივიწყოთ Availability.



ნახ.2.3

CAP თეორემის თვალსაზრისით, ამ სამიდან ერთდროულად მხოლოდ რომელიმე ორი თვისება შეიძლება განხორციელდეს. განაწილებული სისტემები იყოფა სამ კლასად:

CA (მთლიანობა და წვდომადობა) – ერთი და იგივე მონაცემებია ყველა კლასტერში. თუ კომუნიკაციის რაიმე პრობლემა დაფიქსირდა კვანძებს შორის (partition), მონაცემები იქნება არასინქრონირებული მანამ, სანამ პრობლემა არ გადაიჭრება, ანუ კლასტერებში განსხვავებული მონაცემები იარსებებს;

CP (მთლიანობა და განაწილების მდგრადობა) – მონაცემები არაწინააღმდეგობრივია კლასტერებში. მაგალითად, დაფიქსირდა კომუნიკაციის პრობლემა. ამ დროს არასინქრონული რომ არ იყოს მონაცემები, შესაბამისი კვანძი გახდება გაუქმებული (ანუ წაკითხვა/ჩაწერა იქნება გაჩერებული შესაბამის node-ში). აქ თავს ვიცავთ იმისგან, რომ ბაზაში სხვადასხვა მონაცემები არ გვეწეროს;

AP (წვდომადობა და განაწილების მდგრადობა) – არაა გარანტირებული მონაცემთა მთლიანობა. მაგალითად, დაფიქსირდა პრობლემა კლასტერებს შორის. აქ დაზიანებული node-ები არაა გაუქმებული. ისინი დამოუკიდებლად განაგრძობს მუშაობას. როდესაც კომუნიკაციის პრობლემა აღდგება, დაიწყება მონაცემების სინქრონიზაცია, თუმცა გარანტია არ გვაქვს რომ ყველა მონაცემი იგივე იქნება კვანძებს შორის.

კალიფორნიის უნივერსიტეტის პროფესორის ერიკ ბრიუერის CAP თეორემა, რომელიც მან 2000 წელს შემოგვთავაზა, პოპულარული გახდა განაწილებული კომპიუტერული სისტემების სფეროში [8]. NoSQL ბაზების კონცეფცია, რომლითაც იქმნება განაწილებული არატრანზაქციული მონაცემთა ბაზების სისტემები, ხშირად იყენებს სწორედ ამ პრინციპს მონაცემთა მთლიანობის (consistency) დარღვევის გარდაუვალობის დასაბუთების მიზნით. უმრავლესი NoSQL სისტემები არ იძლევა მონაცემთა ბაზის მთლიანობის გარანტიას. ამიტომაც AP-სისტემების აგების ამოცანა მდგომარეობს მონაცემთა მთლიანობის გარკვეული, პრაქტიკულად მიზანშეწონილი დონის უზრუნველყოფის განხორციელებაში. ასეთი AP-სისტემები ლიტერატურაში მოიხსენიება ტერმინით „სუსტი მთლიანობის“ (weak consistent) სისტემები [2].

2.8. განაწილებული გარემო: replication და sharding

➢ ***რეპლიკაცია*** *არის პროცესი, რომლის დროსაც მონაცემები სინქრონიზირდება ერთი, მთავარი სერვერიდან რამდენიმე სათადარიგო სერვერზე.* რეპლიკაციის დროს მთავარ (primary) სერვერზე ჩაწერისა და კითხვის ოპერაციების განხორციელება შეგვიძლია, სათადარიგო სერვერებიდან კი მხოლოდ ვკითხულობთ. რეპლიკაციას რამდენიმე ძირითადი დადებითი მხარე გააჩნია:

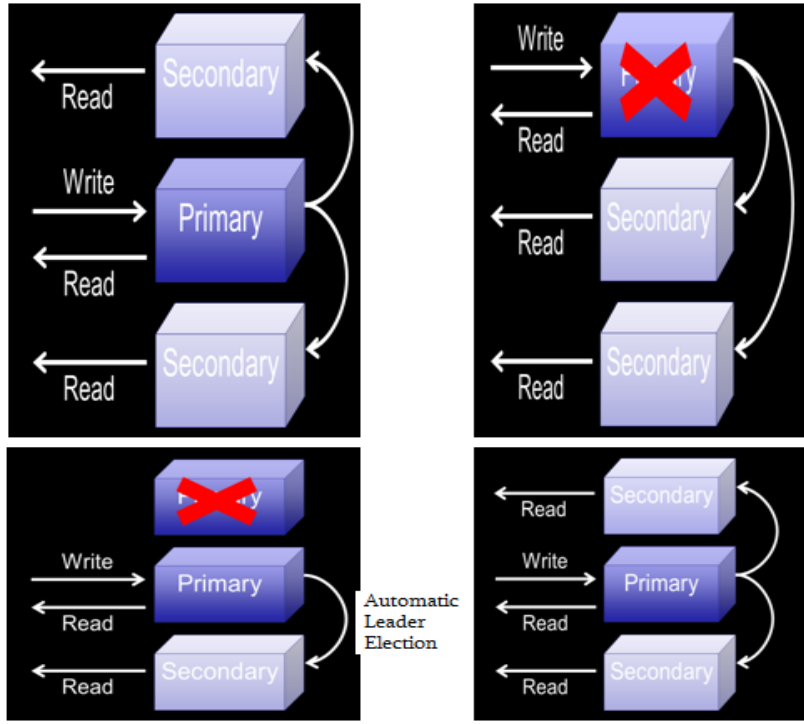
- მონაცემების კითხვის გაუმჯობესება – რამდენ კოპიო სერვერსაც დავაკონფიგურირებთ, იმდენი დამოუკიდებელი წყაროდან შეგვიძლია ინფორმაციის პარალელურ რეჟიმში წაკითხვა;
- Disaster Recovery – პრობლემის ან გეგმიური სამუშაოების დროს შეგვიძლია რომელიმე სათადარიგო (standby) სერვერი ვაქციოთ ძირითად (primary) სერვერად;
- აღარ არის საჭირო backup-ებისა და ინდექსების რეორგანიზაციის გამო მონაცემთა ბაზის გაჩერება ან შენელება;
- რეპლიკა სეტები აპლიკაციის დონეზე არ ჩანან.

მთავარ სერვერთან კავშირის დაკარგვის ან გათიშვის შემთხვევაში სათადარიგო სერვერები აწყობს არჩევნებს და ირჩევს ახალ primary სერვერს.

რეპლიკა სეტების არჩევნებში მონაწილეობს წინასწარ განსაზღვრული სერვერები – არბიტრები. არბიტრი არ შეიცავს მომხმარებელთა ინფორმაციას, მისი ერთადერთი დანიშნულება არჩევნებში ხმის მიცემაა.

თუ Replica set-ში გვაქვს ლუწი რაოდენობის კვანძი, მაშინ არბიტრის დამატებით ვიღებთ კენტი რაოდენობის წევრებს, რითაც ვაღწევთ ხმათა უმრავლესობას არჩევნებში.

ყველაზე ხშირად რეპლიკაციას სამი რეპლიკა სეტისგან აწყობენ ხოლმე (ნახ.2.4).



ნახ.2.4

რეპლიკაციის აწყობა დიდ სირთულეებთან არ არის დაკავშირებული, მთავარ სერვერზე ვქმნით რეპლიკაციის გარემოს:

mongod --port "PORT" --dbpath "ბაზის მისამართი" --replSet "რეპლიკა სეტის ინსტანსის სახელი";

ამის შემდეგ ვუკავშირდებით უკვე შექმნილ გარემოს და ვუშვებთ რეპლიკაციის პროცესს : rs.initiate().

რეპლიკა სეტის კონფიგურაციის შემოწმება შეგვიძლია ბრძანებებით rs.conf() და rs.status().

ახალი წევრების დასამატებლად ვუშვებთ

rs.add(HOST_NAME:PORT)

სტრუქტურის ბრძანებას.

თუ არაფერს არ შევცვლით, კითხვისათვის კლიენტები ძირითად სერვერს მიმართავენ, მაგრამ აპლიკაციიდან შეგვიძლია ავირჩიოთ „Read Preference“ და კითხვა განვახორციელოთ მეორადიდან.

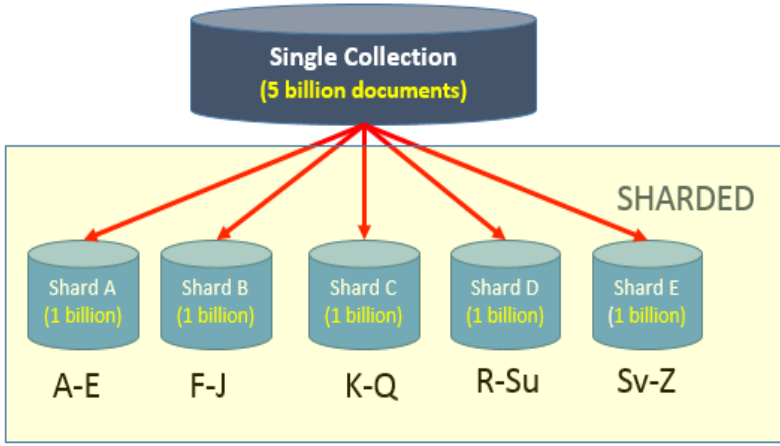
Read Preference პარამეტრი საგრძნობლად აუმჯობესებს წარმადობას, მაგრამ ცხადია, პრობლემა ხდება კონზისტენტურობა,– სანამ ძირითად(primary) სერვერზე განხორციელებული ცვლილება სათადარიგო(secondary) ბაზამდე მიაღწევს, კლიენტები ძველ ინფორმაციას კითხულობენ. ეს პრობლემა გადაჭრილია majority პარამეტრის შემოღებით. ამ პარამეტრით ვუთუოთებთ, მინიმუმ რამდენ კვანძზე უნდა ჩაიწეროს ინფორმაცია, რომ ტრანზაქცია წარმატებულად ჩაითვალოს. ძირითად სერვერზე ინფორმაციის რამდენიმე ვერსია გვექნება, მაგრამ მომხმარებლებს მივაწვდით მხოლოდ იმ ვერსიას, რომელიც დასრულეულია, ანუ სათადარიგო სერვერების საჭირო რაოდენობაზე უკვე ჩაწერილია.

- ***Sharding*** *ტექნოლოგია მონაცემთა ბაზის დანაწევრების (database partitioning) ერთერთი სახეა.* Shard ტერმინი ერთი მთლიანის პატარა ნაწილს ნიშნავს,– შარდინგის მეშვეობით დიდი ზომის უმართავ ბაზას ვყოფთ პატარა, სწრაფ და ადვილად მართვად ნაწილებად.

ტექნიკურად არც შარდინგის აწყობაა რთული, მთავარი სირთულე shard key გასაღების სწორად არჩევაა. ამ გასაღების მიხედვით ნაწილდება დოკუმენტები შესაბამის Shard-ებზე.

მაგალითისთვის შეგვიძლია მოვიყვანოთ 5 ბილიონი ჩანაწერის მქონე კოლექცია, რომელიც გადანაწილებულია 5 Shard-ზე, გასაღებად კი აღებულია ტექსტური ველი. გასაღების ამგვარი არჩევის შემთხვევაში მარტივად შეგვიძლია ახალი Shard-ის

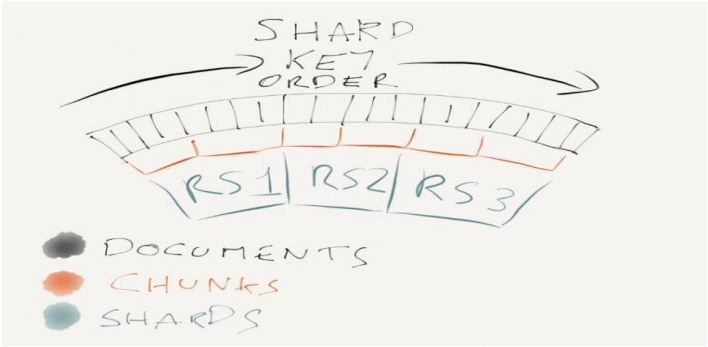
დამატება რომელიმე ძველ Shard-ზე ინფორმაციის გადანაწილებით (ნახ.2.5).



ნახ.2.5

ჩანკების (chunk) დასაყოფად შეგვიძლია გამოვიყენოთ Range based partitioning ან hash based partitioning.

დოკუმენტები ერთიანდება ფიქსირებული ზომის chunk-ებში (default ზომა – 64 მბ). თავად chunk-ები კი shard-ებში ერთიანდება (ნახ.2.6).



ნახ.2.6

მონაცემების გაზრდის შემდეგ ახალი shard სერვერის დამატებას რომ დავაპირებთ, ერთი ან რამდენიმე არსებული shard-იდან ახალ სერვერზე უნდა გადმოვიტანოთ ახალი დაყოფის შესაბამისი chunk-ები.

mongoDB chunk-ების ბალანსირების პროცესს ავტომატურად, კლიენტებისგან დამოუკიდებლად ასრულებს.

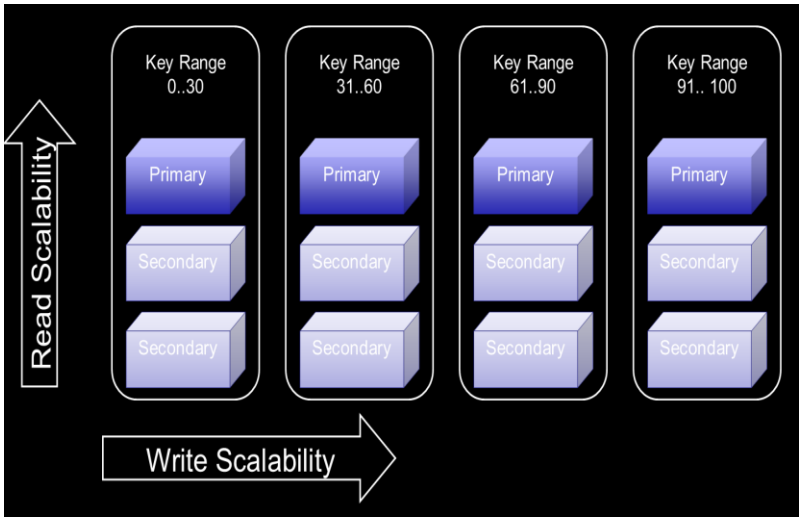
ასევე, მომხმარებლისგან დამოუკიდებლად წყდება, თუ რომელი shard-ებია პასუხისმგებელი კლიენტის მოთხოვნაზე. ამისთვის არსებობს მოთხოვნების მარშრუტიზატორი (query router), რომელიც კლიენტისგან იღებს მოთხოვნას, წინასწარ შენახული metadata (მეტა ინფორმაციის) მიხედვით არჩევს შესაბამის shard სერვერებს და კლიენტს საბოლოოდ აწყობის შედეგს უბრუნებს.

- **განაწილებული გარემო – replication + sharding**

საერთოდ, მონაცემთა ბაზების წარმადობა დამოკიდებულია სამ ძირითად მახასიათებელზე : პროცესორი, ოპერატიული მეხსიერება და მყარი დისკო.

შარდინგისა და რეპლიკაციის დროს კი მოთხოვნის ზრდასთან ერთად შეგვიძლია ასობით და ათასობით ახალი

მანქანის დამატება და თითოეული ახალი მანქანა გაგვიუმჯობესებს სამივე ძირითად მახასიათებელს (ნახ.2.7).



ნახ..2.7

ხშირად, ადმინისტრატორები მონაცემთა ბაზებს საჭიროზე მეტ ოპერატიულსა და პროცესორებს უყოფენ, ამ დროს bottleneck (ბოთლის შევიწროებული ყელი) პრობლემას ვაწყდებით დისკების სიმცირეში. სერვერი დიდ დროს კარგავს დისკიდან ინფორმაციის ამოკითხვაზე.

ეს პრობლემა განაწილებულ გარემოში მუშაობისას მინიმუმამდეა დაყვანილი :

- გვჭირდება სწრაფი კითხვა? – ვამატებთ ახალ რეპლიკა სეტს, ანუ ბაზის ახალ კოპიოს;
- გვჭირდება სწრაფი ჩაწერა/წაკითხვა? – შარდინგ ტექნოლოგიით უფრო მცირე ნაწილებად ვყოფთ მონაცემთა ბაზას, შესაბამისად, უფრო მეტი დისკი წერს და კითხულობს ერთდროულად.

III თავი
მონაცემთა საცავები

3.1. OLAP - მონაცემთა დამუშავების on-line ტექნოლოგია და მისი აგების პრინციპები

ორგანიზაციული სისტემების მართვის პრობლემების და ამოცანების გადასაწყვეტად სწრაფი განვითარება ჰპოა ახალმა კომპიუტერულმა და ინფორმაციულმა ტექნოლოგიებმა, რომელთა შორის ერთერთი აქტუალური და მნიშვნელოვანი მიმართულებაა მონაცემთა საცავების (data warehouse) აგება. ბოლო პერიოდში იგი ითვლება, განსაკუთრებით პერსპექტიულ და დინამიკურ მიმართულებად საინფორმაცო სისტემების დაპროექტებაში, იგი უკავშირდება მრავალდონიან განაწილებულ საინფორმაციო სისტემის შექმნას. მონაცემთა საცავი განიხილება როგორც რომელიმე კონკრეტული ორგანიზაციის ან დიდი საწარმოსთვის განკუთვნილი სპეციალური სუპერ-ბაზა, სადაც მიმდინარე ოპერატიული სამუშაოს შესრულებისას თავს იყრის ქრონოლოგიურ ინფორმაციათა მთელი სპექტრი, რომელთა დანიშნულებაცაა მომხმარებლისთვის ინტერნეტ გვერდებზე მიზნობრივად განლაგებული ტექსტური, გრაფიკული და აუდიო-ვიზუალური საინფორმაციო ბლოკების მიწოდება.

მონაცემთა საცავების დაპროექტებისა და მისი ფუნქციონირებისათვის, მეთოდური თვალსაზრისით საყურადღებოა მონაცემთა რელაციური მოდელების „მამის“, ედგარ კოდის მიერ ჩამოყალიბებული პრინციპები [37].

ესაა ის 12 წესი, რომელსაც უნდა აკმაყოფილებდეს ნებისმიერი განაწილებული სისტემა მონაცემთა საცავით, რათა ჩატარდეს საინფორმაციო ბლოკების სრულფასოვანი ოპერატიული ანალიზური სამუშაოები.

1. მონაცემთა მრავალგანზომილებიანი კონცეპტუალური წარმოდგენა. 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 დროის ნებისმიერ მონაკვეთში, ანალიზური ინსტრუმენტის საშუალებით დაუკავშირდეს სერვერს და მიიღოს მისთვის სასურველი ინფორმაცია;

3. მიღწევადობა. OLAP – ის მომხმარებელ ანალიზატორს უნდა ჰქონდეს ანალიზის ჩატარების საშუალება, რომელიც ემყარება საერთო კონცეპტუალურ სქემას, რომელშიც განთავსებულია რელაციური მონაცემთა ბაზა სწარმოთა 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. ანგარიშთა დამუშავებისას მუდმივი წარმადობა. თუ ანალიტიკოსის მიერ ჩატარებული გაზომვათა რაოდენობა ან მონაცემთა ბაზების რიცხვი მნიშვნელოვნად იზრდება, მომხმარებელ ანალიზატორისთვის ეს პროცესი უნდა დარჩეს შეუმჩნეველი და არ უნდა აისახებოდეს საწარმოო პროცესების წარმადობის შემცირებაზე;

5. კლიენტ სერვერის _ არქიტექტურა. მონაცემთა დიდი ნაწილი, რომელიც მოითხოვს ოპერატიულ ანალიზურ გადამუშავებას, უნდა მუშაობდეს კლიენტ-სერვერულ რეჟიმში. ამ თვალსაზრისით აუცილებელია, რომ ანალიზური ინსტრუმენტის სერვერული კომპონენტები იყოს `ინტელექტუალური~, რადგან განსხვავებულ კლიენტებს შეეძლოთ დაუკავშირდნენ სერვერს და გამოიყენონ პროგრამიული პაკეტები. `ინტელექტუალურ~ სერვერს უნდა შეეძლოს მონაცემთა ბაზის შეუთავსებადი ლოგიკური და ფიზიკური სქემის ასახვა და გაერთიანება. ეს უზრუნველყოფს გამჭვირვალეობას და საშუალებას იძლევა აიგოს საერთო კონცეპტუალური, ლოგიკური და ფიზიკური სქემები;

6. მრავალგანზომილება. გაზომვის ყოველი მცდელობის დროს გამოყენებული უნდა იყოს მიუკერძოებელი სტრუქტურა და ოპერაციული შესაძლებლობა. დამატებითი ოპერაციული შესაძლებლობა უნდა მიეცეს ერთ-ერთ რომელიმე ცდას და თუ ეს გაზომვა სიმეტრიული იქნება სხვა გაზომვის შედეგების, მაშინ ცალკე აღებული ფუნქცია შეიძლება წარმოვადგინოთ ნებისმიერი გაზომვის სახით;

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

8. მრავალმომხმარებლობა. ხშირ შემთხვევაში მომხმარებელი ანალიტიკოსი დასმულ მოთხოვნებს აყენებს ერთ ანალიზურ მოდელთან ან ქმნის განსხვავებულ მოდელს ერთი სახის მონაცემებიდან. OLAP ინსტრუმენტი კი საშუალებას გვაძლევს მივიღოთ უსაფრთხო, სრულყოფილი და ზუსტი ანალიზური შედეგები;

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. თუ რაიმე სირთულე წარმოიქმნება, უნდა შევჩერდეთ ინდივიდუალურ

ინფორმაციულ მოთხოვნაზე და შეირჩეს მხოლოდ მომლოდინე მოთხოვნა;

12. შეუზღუდავი ზომები და აგრეგაციათა რაოდენობა. გამოკვლევებმა აჩვენეს, რომ აუცილებელი გაზომვა ერთდროულად შეიძლება ჩატარდეს 19-ჯერ. აქედან გამომდინარე შეიძლება ვთქვათ, რომ ანალიზური ინსტრუმენტი საშუალებას გვაძლევს ერთდროულად ვაწარმოოთ 15-დან 20-მდე გაზომვა, ამასთან თითოეული გაზომვის მცდელობა არ არის შემოსაზღვრული დადგენილი რიცხვით. ეს პირობები შეიძლება ჩავთვალოთ, ოპერატიული ანალიზური დამუშავების თეორიულ ბაზისად. როგორც უკვე ავღნიშნეთ OLAP-ში ძევს მონაცემთა დამუშავების მრავალგანზომილებიანი სტრუქტურის იდეა. როდესაც ვსაუბრობთ OLAP–ზე, უნდა ვიგულისხმოთ, რომ ეს არის მონაცემთა ლოგიკური სტრუქტურის მრავალგანზომილებიანი ანალიზური ინსტრუმენტი.

3.2. OLAP - მრავალგანზომილებიან მონაცემთა დამუშავება

მრავალგანზომილებიან მონაცემთა საცავში თავმოყრილი აგრეგატული მონაცემები სხვადასხვა დონეზე ერთმანეთისგან განსხვავდება. მაგალითად, ყიდვა-გაყიდვის მოცულობა ერთ დღეში, თვეში, წელიწადში, საქონლის კატეგორია და ა.შ.

აგრეგატულ მონცემთა შენახვის მიზანია მოთხოვნათა დაკმაყოფილებაზე დახარჯული დროის შემცირება, რამდენადაც უმეტეს შემთხვევაში ანალიზისა და პროგნოზის სფეროს წარმოადგენს არა დეტალური ინფორმაცია, არამედ გაერთიანებული, საბოლო სახემდე მიყვანილი დაჯგუფებული მონაცემები.

ამიტომ მრავალგანზომილებიან მონაცემთა ბაზის შექმნის დროს, ყოველთვის გასათვალისწინებელია, რომ შეტანილ იქნეს რამდენიმე აგრაგატული მონაცემი. მაგრამ ყველა მონაცემის აგრაგატული სახით შენახვა გაუმართლებელია, რადგან მონაცემთა

მოცულობაში ახალი განზომილების დამატების შემთხვევაში კუბი, სადაც ამ მონაცემის ჩამატება მოხდება, დაიწყებს ზრდას ექსპონენციალურად. ამ პროცესს უწოდებენ მონაცემთა მოცულობის „ფეთქებად ზომას~, ამიტომ აგრეგატულ მონაცემთა მოცულობის ზომის ხარისხი დამოკიდებულია კუბების რაოდენობაზე და იერარქიის სხვადასხვა დონეზე მიმდინარე გაზომვებზე.

„ფეთქებადი ზომის~ პრობლემის გადასაჭრელად გამოიყენება სხვადასხვა სახის მონაცემთა შენახვის სქემები: MOLAP, ROLAP, JOLAP და HOLAP.

MOLAP – (მრავალგანზომილებიანი OLAP) პროდუქტს საფუძვლად უდევს მონაცემთა არარელაციური სტრუქტურა, რომელიც უზრუნველყოფს მონაცემთა მრავალგანზომილებიან შენახვას;

ROLAP – (რელაციური OLAP) ასეთ ინსტრუმენტში მრავალგანზომილებიანი სტრუქტურა რეალიზდება რელაციური ცხრილებით, ხოლო მონაცემები ანალიზის პროცესში შესაბამისად შეირჩევა მონაცემთა რელაციური ბაზებიდან ანალიზური ინსტრუმენტით;

JOLAP – ინსტრუმენტის გამოყენებისას თითოეული მოთხოვნის მოდელირება შესაძლებელია, როგორც ერთიან ობიექტთა ჯგუფი, რის განხორციელებაშიც ეხმარება გრაფიკულ ინტერფეისში ოპერაციათა თანამიმდევრული სტრუქტურა და მოთხოვნათა მოდიფიკაციის განსაზღვრა. JOLAP (Java Specification Requet-69) გმოიყენება სხვადასხვა სპეციფიკაციაში მაგალითად, როგორიცაა: ჯგუფი OMG მეტამოდელისთვის Common Warehouse Metamodel (CWM), ობიექტური მოდელი Meta Object Facility (MOF), XML Metadata Interchange (XMI) და აგრეთვე Java Metadata Interface (JMI,JSR-40). ამ ინტერფეისის არქიტექტურა ერთდროულად J2EE და J2SE;

მრავალგანზომილებიანი OLAP უზრუნველყოფს მაღალ მწარმოებლურობას, მაგრამ სტრუქტურები არ შეიძლება გამოვიყენოთ დიდი მოცულობის მონაცემთა დამუშავებისთვის, რადგანაც დიდი გაბარიტები მოითხოვს დიდ აპარატულ რესურსებს. ამასთან ერთად ჰიპერკუბის დატვირთვა შეიძლება იყოს ძალიან მაღალი, რაც შემდგომში თავს იჩენს ინფორმაციის მიწოდების სისწრაფეზე.

შეიძლება ვთქვათ პირიქითაც, რელაციური OLAP უზრუნველყოფს შენახული მონაცემების დიდი მასივების დამუშავებას, ასე რომ შეიძლება უზრუნველვყოთ უფრო ეკონომიური შენახვა, მაგრამ ამასთან ერთად, მნიშვნელოვან როლს ითამაშებს მრავალგანზომილებიან სამუშაოს შესრულების სიჩქარეზე.

მსგავსს მსჯელობას მივყავართ ახალი კლასის გამოყოფაზე, რომელიც არის HOLAP ანალიზური ინსტრუმენტი. ეს არის ჰიბრიდული მონაცემთა ოპერატიულ-ანალიზური დამუშავების ინსტრუმენტი. ასეთი კლასის ინსტრუმენტი საშუალებას იძლევა შეთანხმდეს ორივე მიდგომა – რელაციური და მრავალგანზომილებიანი. ხელმისაწვდომი გახდება როგორც მრავალგანზომილებიანი მონაცემთა ბაზები, ასევე რელაციური მონაცემები.

ახლა განვიხილოთ OLAP კუბში მონაცემთა აგრეგაციის საკითხი. როგორც ცნობილია, მონაცემთა ფორმალიზების აგრეგაციის მექანიზმის სახით ფართოდ გამოიყენება გრაფების თეორია. მონაცემთა მრავალგანზომილებიანი მოდელის ძირითადი პრინციპებია:

მაჩვენებელიანი – სიდიდე (იგი ჩვეულებრივი რიცხვია), რომელიც გამოიყენება საგნობრივი ანალიზისათვის. მაგალითად, რომელიმე პროდუქტის ყიდვა-გაყიდვის მოცულობა, ან ამ პროცესიდან მიღებული შემოსავალი. ერთი OLAP-კუბი შეიძლება შეიცავდეს ერთ ან რამდენიმე მაჩვენებელს.

განზომილება (dimension) – ერთი ან რამდენიმე სახის ობიექტთა სიმრავლეა, რომელიც ორგანიზებულია იერარქიული სტრუქტურით და უზრუნველყოფს რიცხვითი მაჩვენებლის საინფორმაციო კონტექსტს. გაზომვა ვიზუალურად უნდა აისახებოდეს მრავალგანზომილებიანი კუბის გვერდებზე.

ობიექტები – მათი ერთობლიობა წარმოქმნის განზომილებას, ანუ ეგრეთწოდებულ განზომილებათა წევრებს (members). განზომილებათა წევრები ვიზუალურად აისახება, როგორც ცალკეული წერტილები ან უბნები, რომელიც მოთავსებულია ჰიპერკუბის ღერძებზე. მაგალითად, დროითი განზომილება: დღე, თვე, კვარტალი, წელი: 26 მაისი 2002 წელი, მე-2 კვარტალი 2002 წელი და 2002 წელი. განზომილებაში მოცემული ობიექტები შეიძლება იყოს სხვადასხვა სახის, მაგალიტად, „მწარმოებელი“, „პროდუქტის საფირმო ნიშანი“, „წელი“, „კვარტალი“. ეს ობიექტები ორგანიზებული უნდა იყოს იერარქიული სტრუქტურით. ერთი სახის ობიექტებს შეესაბამება, იერარქიის მხოლოდ ერთი დონე.

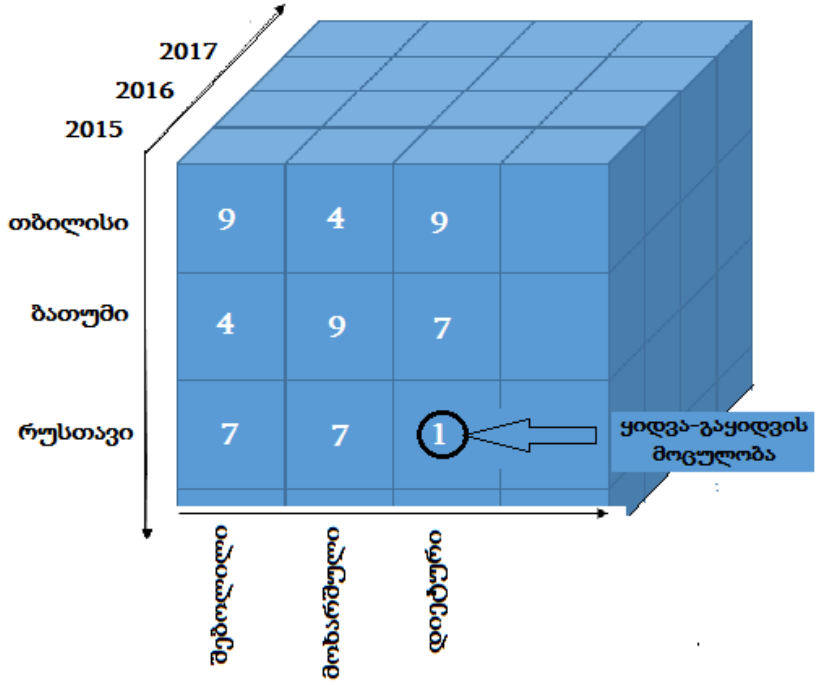
უჯრედი (cell) – კუბის ატომური სტრუქტურაა, რომელიც შეესაბამება რომელიმე მაჩვენებლის კონკრეტულ მნიშვნელობას. უჯრედები ვიზუალურად მოთავსებულია კუბის შიგნით და მათზე ასახულია მაჩვენებლების შესაბამისი მნიშვნელობები.

განზომილება თამაშობს ინდექსების როლს, იგი გამოიყენება ჰიპერკუბის უჯრედში მაჩვენებლის მნიშვნელობის იდენტიფიკაცი-ისათვის. სხვადასხვა განზომილებით მიიღებული კომბინაციები თამაშობს კოორდინატების როლს, რომელიც განსაზღვრავს კონკრეტული მაჩვენებლის მნიშვნელობას.

ყველა განზომილების შედეგად მიღებული კომბინაციებით შესაძლებელია რამდენიმე უჯრედის მნიშვნელობის განსაზღვრა. ამიტომ უჯრედის ერთმნიშვნელოვანი იდენტიფიკაციისათვის უცილებელია მივუთითოთ განზომილების ყველა კომბინაცია და მაჩვენებელი. ამავე დროს დაცული უნდა იყოს განზომილების

იერარქიული დონეები, რათა აგრეგაციისა და მაჩვენებელთა მნიშვნელობის დეტალიზაციისათვის ამას დიდი მნიშვნელობა ენიჭება.

არსებობს ***ბალანსირებული (balanced) იერარქია,*** რომელშიც დონეთა რიცხვი განისაზღვრება მისი სტრუქტურით და მუდმივია ამავე დროს, იერარქიული ხის თითოეული ტოტი შეიცავს ობიექტებს ყოველი დონიდან. მაგალითად, ძეხვის ნებისმიერ მწარმოებელს შეუძლია აწარმოოს რამდენიმე სახის ძეხვი სხვადასხვა საფირმო ნიშნით (ნახ.3.2).



ნახ.3.2. მრავალგანზომილებიანი კუბი

ამ შემთხვევაში შეგვიძლია ვისაუბროთ სამდონიან იერარქიაზე, რომლის პირველ დონეზე იქნება „მწარმოებელი“, მეორეზე „სახეობა“, ხოლო მესამეზე „საფირმო ნიშანი“.

ბალანსირებული ირერარქიის ფორმირებისათვის აუცილებელია არსებობდეს კავშირი „ერთი-მრავალთან“. მისი ყოველი დონე შეიძლება წარმოვადგინოთ როგორც ცალკეული განზომილება, მაგრამ ეს განზომილება დამოკიდებული იქნება სხვა რომელიმე განზომილებაზე, ამიტომ საჭირო გახდება კუბის გაკვეთა.

დაუბალანსებელი (unbalanced) იერარქია, რომელშიც დონეთა რიცხვი შესაძლებელია შევცვალოთ, ხოლო იერარქიული ხის თითოეული ტოტი შეიძლება შეიცავდეს ობიექტებს არა ყოველი დონიდან, არამედ მხოლოდ პირველიდან. აუცივლებელია აღინიშნოს, რომ დაუბალანსებელ იერარქიაში არსებული ყველა ობიექტი არის ერთი სახის. მაგალითად, იერარქია „ხელმძღვანელი–დამფასოვებელი“ ყველა ობიექტი ერთიანდება სახეში: „თანამშრომელი“.

არათანაბარი იერარქია, რომელშიც დონეთა რიცხვი განისაზღვრება მისი სტრუქტურით და მუდმივია.

ბალანსირებულისგან განსხვავდება მხოლოდ იმით, რომ იერარქიული ხის რომელიმე ტოტი შეიძლება არ შეიცავდეს რომელიმე დონის ობიექტებს. ასეთი სახის იერარქია შეიცავს ისეთ წევრებს, როგორიცაა ე.წ. ლოგიკური „მშობელი“ და იგი უშუალოდ არ მდებარეობს რომელიმე დონეზე.

ტიპური მაგალითია გრაფიკული იერარქია, რომელსაც აქვს შემდეგი დონეები: „ქვეყანა“, „შტატი“ და „ქალაქი“. თუ მონაცემთა ნაკრებში მოცემული გვექნება ქვეყნები, რომელთაც არ გააჩნია „შტატი“, მაშინ იგი მოთავსდება დონეებს შორის „ქვეყანა“ და „ქალაქი“. ჩვენ განვიხილავთ მხოლოდ ბალანსირებულ იერარქიას, რადგან დაუბალანსებელი და არათანაბარი პრაქტიკაში თითქმის არ გამოიყენება.

აგრეგატებს უწოდებენ განსაზღვრული პირობის შესაბამისად მახასიათებლის საწყის მნიშვნელობათა აგრეგირებას. აქ იგულისხმება მცირე რაოდენობის აგრეგატების მნიშვნელობათა ფორმირების ნებისმიერი პროცედურა, რომელიც ეყრდნობა დიდი რაოდენობის საწყის მნიშვნელობებს.

ნაწილობრივი და სრული აგრეგაცია. კუბის აგრეგაციის ხარისხი გამოითვლება ფორმულით:

$$\alpha = \frac{a}{b}$$

სადაც a–არის აგრეგირებულ მახასიათებლის მნიშვნელობის რეალური რაოდენობა, b - კუბის საწყისი მონაცემების აგრეგატული მნიშვნელობის მაქსიმალური რაოდენობა [40].

3.3. Oracle და OLAP ტექნოლოგია

როგორც ზემთ აღვნიშნეთ, OLAP კუბი აღწერს მონაცემებს და შესაძლებელს ხდის მათ მარტივად მოპოვებას, ამასთან მონაცემები შესაძლებელია ერთ ან რამდენიმე კუბის ღერძებზე წარმოვადგინოთ და ჩავუტაროთ მრავალგანზომილებიანი ანალიზი.

ბოლო პერიოდში OLAP ტექნოლოგიის დანერგვა და გამოყენება ღებულობს ფართო მასშტაბებს. ერთ-ერთ ევოლუციურ ნაბიჯად შეიძლება ჩაითვალოს Oracle 12g ვერსია, სადაც ორგანიზებული და მატერიალიზებულია OLAP კუბი, რომელიც ეხმარება მონაცემთა საცავებში მოთხოვნების დაყენებასა და მათ ფორმულირებში, ასევე მომსახურების პროცესში, მაგალითად, როგორიცაა მონაცემთა აქტუალიზაცია, განახლება, დამუშავება და ა.შ. [41].

Oracle OLAP არის მრავალგანზომილებიანი ანალიზური მექანიზმი, რომელიც აგებულია Oracle12g მონაცემთა ბაზის

საფუძველზე. იგი უზრუნველყოფს რთულ გამოთვლებს, სადაც გამოიყენება მარტივი SQL მოთხოვნის ენა და ხორციელდება მონაცემთა ცენტრალიზებული მართვა. Oracle OLAP-ის არქიტექურას აქვს 3.3 ნახაზზე მოცემული სახე.

ნახ.3.3. Oracle OLAP -ის არქიტექურა

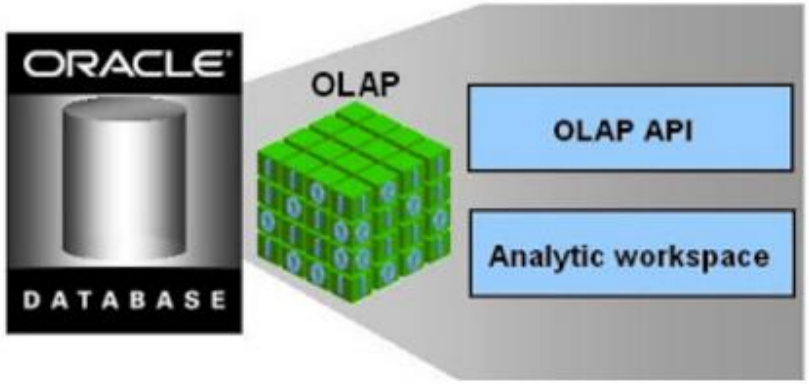
Oracle OLAP სისტემა ერთიანი ფუნქციონალური პლატფორმაა მონაცემთა საცავის და ბიზნესანალიზური სისტემის ასაგებად. ერთიანი პლატფორმა ემყარება საიმედო და იაფ Grid ინფრასტრუქტურას, რომელიც ამავე დროს არის უნიკალური, ინტეგრირებული პლატფორმა ანალიზისთვის.

Oracle OLAP მომხმარებელს აძლევს საშუალებას გამოიყენოს ანალიზური საშუალებები, მაგალითად, ფინანსურ მოდელებში, პროგნოზირებაში, დროითი მწკრივის გამოთვლაში, რეგრესიასა და

ა.შ. ასობით ანალიზური ფუნქცია შესაძლებელია მარტივად გაერთიანდეს სამომხმარებლო ფუნქციაში, რომელიც გადაჭრის ნებისმიერი სახის ანალიზურ მოთხოვნას.

OLAP Oracle კუბი ბაზირებულია „ვარსკვლავის“ სქემაზე, განზომილებები ფორმულირებულია ფაქტების ცხრილის გარშემო.

Oracle OLAP კომპონენტები შედგება ორი ძირითადი პარამეტრისგან: OLAP API და ანალიზური სამუშაო გარემო (AW) (ნახ.3.4).



ნახ.3.4. Oracle OLAP - კომპონენტები

- **OLAP API ფუნქციონალი**

OLAP API - არის Java ენაზე ბაზირებული აპლიკაცია, პროგრამული ინტერფეისი, რომელიც უზრუნველყოფს ონლაინ რეჟიმში მონაცემებზე წვდომას და მათ ანალიზურ დამუშავებას, ხოლო OLAP API ფუნქციონალი ახდენს ონლაინ ანალიზური პროცესების იმპლემენტაციას და სამუშაო გარემოს მოწყობას.

მრავალგანზომილებიან მონაცემებთან წვდომა შესაძლებელია არა მხოლოდ Java დანართის საშუალებით, არამედ აპლიკაციითაც, რომელიც ორიენტირებულია SQL-ენაზე.

იმ შემთხვევაში თუ მონაცემები ფიზიკურად მდებარეობს რელაციურ ცხრილებში, მაშინ მათთან შესაძლებელია ნებისმიერ სტრუქტურირებულ მონაცემთა მოთხოვის ენის გამოყენება. ანალიზურ სივრცეში არის შესაძლებლობა შეიქმნას რელაციურ წარმოდგენათა გარემო (View), სადაც ობიექტებს ანალიზურ სივრცესთან ექნება წვდომა. ასეთი წარმოდგენა ავტომატურად გენერირდება სპეციალური შენახვის პროცედურების დახმარებით, სადაც გამოიყენება ობიექტ-ორიენტირებული მონაცემთა ბაზები Oracle და ცხრილური ფუნქციების ტექნოლოგიები [5].

Oracle Express server მრავალი წლის განმავლობაში იყო OLAP ტექნოლოგიის ძირითადი სერვერი, თავის უპრეცედენტო ფუნქციონალური შესაძლებლობების გამო, მაგრამ Oracle კორპორაციამ წარმოადგინა ახალი ვერსია Oracle OLAP, სადაც Oracle Express-ის ყველაზე მძლავრი მხარეებია და ჩააშენა მასში მონაცემთა ბაზის ბირთვი [43].

Oracle OLAP სრულად ინტეგრირებულია რელაციურ მონაცემთა ბაზასთან, სადაც ყველა მონაცემი და მეტამონაცემი იმართება და ინახება Oracle9i-ს საშუალებით, იგი გამოირჩევა სისტემის მონაცემთა დიდი მასშტაბების უზრუნველყოფით, მართვისას არის კონტროლირებადი და უსაფრთხო და ასევე არის ხელმისაწვდომი კომერციული თვალსაზრისით.

რევოლუციური თავისებურება, რაც Oracle და OLAP -ის ინტეგრირებაში აისახება არის ის, რომ შესაძლებელი გახდა მრავალგანზომილებიან მონაცემებში სტანდარტულ SQL ენაზე დაიწეროს მოთხოვნები, მონაცემთა შენახვის და ანალიზური დამუშავებისათვის.

სხვა შესაძლებლობებთან ერთად, Oracle OLAP-ს აქვს OLAP კატალოგი მეტამონაცემთა დონეზე, რომელშიც აღიწერება როგორც რელაციური, ასევე მრავალგანზომილებიანი მონაცემები, სადაც

გამოიყენება ე.წ. მრავალგანზომილებიანი კუბები, Java და OLAP API.

აღნიშნული ინსტრუმენტებით შესაძლებელი ხდება აიგოს დანართები, რომელთანაც OLAP ქმნის საერთო ინტერფეისს.

ამგვარად, შეიძლება ითქვას, რომ Oracle OLAP არის მრავალგანზომილებიანი მონაცემთა ბაზა, სადაც მონაცემთა ბაზა Oracle პარალელურად მუშაობს OLAP-ის ფუნქციონალურ შესაძლებლობებთან.

IV თავი
Data Mining - მონაცემთა ინტელექტუალური ანალიზი

4.1. მონაცემთა მაინინგი (Data Mining)

ტრადიციული მათემატიკური სტატისტიკა დიდი ხნის განმავლობაში ინარჩუნებდა ერთ-ერთ მნიშვნელოვან ადგილს, როგორც მონაცემთა ანალიზის ინსტრუმენტი, მაგრამ თანამედროვე გამოწვევების წინაშე აღნიშნული მიდგომა არაეფექტური აღმოჩნდა, რადგან მათემატიკური ანალიზის საფუძველზე შეუძლებელი იყო ჰიპოთეზათა წინასწარი ფორმულირება, მონაცემთა ოპერატიული ანალიზი, მრავალგანზომილებიანი ანალიზი და ა.შ. მეოცე საუკუნის 90-იანი წლებიდან აქტიურად ინერგება და გამოიყენება მონაცემთა ანალიზის OLAP და Data Mining ტექნოლოგიები.

მონაცეთა მაინინგი (Data Mining) არის ე.წ. ინტელექტუალური მოდელის ტიპი, რომელიც ერთის მხრივ განისაზღვრება როგორც გენერირებული ჰიპოთეზათა სახეობა, ხოლო მეორეს მხრივ მნიშვნელოვანი გადაწყვეტილების მიღება ანალიზის პროცესისას. ხშირ შემთხვევაში მონაცემთა ანალიზის მარტივი მოდელი წარმოდგენილია ფორმით, რომელიც პასუხობს „თუ - მაშინ“ ლოგიკას. მონაცემთა მაინინგში გამოიყენება ე.წ. „გადაწყვეტილების ხე“, სადაც არის ხშირად გამოყენებადი სპეციალური წესების ერთობლიობათა ფორმა. კომპლექსური პროცესი დამყარებულია პრედიკტების ლოგიკაზე ან გამოიყენება სპეციალური ნეირონული ქსელები როგორიცაა 1980 წელს ფინელი პროფესორის ტ. კოჰონენის მიერ შემოთავაზებული კოჰონენის ქსელი [44].

მონაცემთა სტრუქტურული წარმოდგენა მრავალგანზომილებიან ნეირონულ ქსელებში, მათი ფარული შრეების სტრუქტურიდან გამომდინარე, დაკავშირებულია ძალიან რთულ პროცესებთან, კოჰონენის ქსელი კი უზრუნველყოფს მონაცემთა

დამუშავების პროცესში ქსელს შესასვლელზე მათ თანმიმდევრულად მიეწოდებას, რის შემდეგაც ქსელი იწყებს გადაწყობას შემავალი მონაცემების კანონზომიერების მიხედვით და არა გამომავალი მონაცემების სტანდარტული შაბლონების შესაბამისად.

მონაცემთა მიწოდების დროს, ქსელის შესასვლელზე თანმიმდევრული გადაცემისას განისაზღვრება ე.წ. „დომინანტი ნეირონი“, რომელიც შემდეგში გამოიყენება მეზობელ ნეირონებში „წონების“ განაწილების ცენტრად.

აღნიშნული მიდგომის გამოყენება ერთის მხრივ შესაძლებელის ხდის საწყის და საბოლოო მონაცემთა კანონზომიერებების ძებნასა და ანალიზს და ამავდროულად კოჰონენის მეთოდი უზრუნველყოფს მრავალგანზომილებიანი შემავალი სივრცის გამოსახვას ორ განზომილებიან გამავალ სივრცედ. ხდება მონაცემთა შეკუმშვა, რაც საბოლოოდ აჩქარებს ალგორითმის შესრულებას.

ნეირონული ქსელის გამოყენების პროცესში აუცილებელ პირობაა „დომინანტი ნეირონის“ განსაზღვრა, რომლის მიხედვითაც ხდება ნეირონული ქსელის კოეფიციენტების კორექტირება.

კოჰონენის ქსელი, მრავალშრიანი ნეირონული ქსელისგან განსხვავებით, შეიცავს მხოლოდ ორ შრეს: შემავალს და გამომავალს, რაც ამარტივებს და ეფექტურს ხდის აღნიშნული ქსელის გამოყენებას. ქსელში არსებული მონაცემების ძიების პროცესში არსებული კომპლექსური მოდელებიდან უნდა შეირჩეს ისეთი მოდელი, რომელიც შეძლებს ევრისტიკული მეთოდების გამოყენებას, რადგან იგი აჩქარებს მისაღები შედეგის დადგომის პროცესს.

Data Mining უზრუნველყოფს მონაცემთა საცავთან დაკავშირებას და კონკრეტულ კომპანიის მონაცემემებთან წვდომას. საჭირო მონაცემები მომხმარებელს მიეწოდება ცხრილების სახით.

Data Mining (discovery-driven data mining) ტექნოლოგია ეფუძნება შაბლონების კონცეფციას, რომელიც აისახება მონაცემთა მრავალასპექტიან ფრაგმენტებთან ურთიერთკავშირში. ეს შაბლონები წარმოდგენილია კანონზომიერად, მონაცემთა ქვეჯგუფებად და მომხმარებელს კომპაქტურად მიეწოდება ადამიანისათვის გასაგები ფორმით. შაბლონთა ძიება მიმდინარეობს მეთოდების გამოყენებით, რომელიც აპრიორულად არ არის შეზღუდული.

DataMining-ის სტრუქტურა მოცემულია 4.1 ნახაზზე [45].

ნახ.4.1

Data Mining კონცეფცია მოიაზრება როგორც მეთოდების და ალგორითმების გამოყენების შესაძლებლობა, როდესაც მონაცემთა ბაზასა და სამიზნე ობიექტს შორის ავტომატურად ხდება ემპირიულ მონაცემთა ექსტრაქცია. მას ასევე შესაძლებელია ვუწოდოთ ინტელექტუალური ანალიზის ინსტრუმენტი, რომელიც მნიშვნელოვან მონაცემებს აგენერირებს, როგორც „ცოდნა“ - ს. მონაცემთა მაინინგის მიზანია მომხმარებელს დაეხმაროს გადაწყვეტილების მიღებაში საინტერესოსა და საჭიროს შორის.

Data Mining-ში გამოიყენება ხელოვნური ინტელექტის და სტატისტიკის ინტეგრირებული მთოდები. იგი არის კომპლექსური პროცესი, როდესაც კლასიფიცირდება ცოდნის ბაზები.

Data Mining-ის ძირითადი ამოცანაა განსაზღვროს მონაცემთა:

- კლასიფიკაცია;
- სეგმენტაცია;
- პროგნოზირება;
- დამოკიდებულებათა ანალიზი;
- გადახრათა ანალიზი.

მონაცემთა კლასიფიკაციისას ყალიბდება კლასები ერთგვაროვან მონაცემთა ობიექტების სახით.

სეგმენტაციისას მონაცემთა ობიექტები ლაგდება ჯგუფებად ერთი მახასიათებელი ნიშანთვისების შესაბამისად. ამავდროულად ერთ ჯგუფში შემავალი ყველა ობიექტი უნდა იყოს ჰომოგენური.

პროგნოზირებისას წინასწარ ხდება უცნობი ნიშანთვისების განსაზღვრა და ყალიბდება როგორც „ცოდნა“.

დამოკიდებულება აღწერს ურთიერთმიმართებას ერთი ნიშანთვისების მქონე ობიექტებს ან განსხვავებულ ობიექტებს შორის.

გადახრათა ანალიზისას სრულად ხდება ისეთი ობიექტების იდენტიფიკაცია, რომელიც არ შეესაბამება დამოკიდებულ

ობიექტებს. ამის გათვალისწინებით დგინდება ობიექტებს შორის ახალი დამოკიდებულება.

თანამედროვე მოთხოვნათა სპეციფიკაცია უნდა პასუხობდეს შემდეგ გამოწვევებს:

- მონაცემებს აქვს უსაზღვრო მოცულობა;
- მონაცემები არის სხვადასხვაგვარი (რაოდენობრივად, ხარისხობრივად, სინტაქსურად);
- შედეგი უნდა იყოს კონკრეტული და გასაგები;
- პირველადი მონაცემების დამუშავების ინსტრუმენტი უნდა იყოს მარტივი.

4.2. Data Mining და Big Data

ხშირ შემთხვევაში ტერმინს Big Data და Data Mining ერთ კონტექსტში გამოიყენებენ. აქ მნიშვნელოვანია ის გარემოება, რომ ორივე თვალსაზრისი ერთმანეთისგან განუყოფელია. Big Data მოიცავს განსაკუთრებით დიდი მოცულობის მონაცემებს, სადაც ტრადიციული მეთოდები და ინსტრუმენტები არაეფექტურია მოცემულ დროის ერთეულში მათ დასამუშავებლად. Data Mining ხშირად გამოიყენება დიდი მოცულობის მონაცემთა ბაზებში, ისევე როგორც Big Data-ში.

Data Mining ანალიზისას აღწერს შემავალ საწყის მონაცემებს მათი რელევანტური დამოკიდებულების შესაბამისად. ამავე დროს შესაძლებელია გამოყენებულ იქნას პატარა მონაცემთა ბაზებიც. Big Data-ს შემთხვევაში მიეწოდება დიდი მოცულობის მონაცემები და გარდაიქმნება ტექნიკურ პლატფორმად რომელიც არის ეფექტური საშუალება მონაცემთა ამოცნობისა და ახალი ჯგუფის გამოვლენისათვის, სადაც ხდება ჰიპოთეზათა გენერირება, რომელიც ვერიფიკაციის შედეგად იხვეწება და ზუსტდება [45].

მონაცემთა მაინინგი იყენებს სტატისტიკურ და ხელოვნური ინტელექტის ალგორითმებს, რაც აძლევს იმის საშუალებას, რომ Big-

Data ტექნოლოგიის სფეროში მიღწეული იქნას კოლოსალური წარმატება. შესაძლებელი ხდება როგორც სტრუქტურირებული, ისე არასტრუქტურირებული მონაცემების დამუშავება, როდესაც Big-Data და Data Mining გამოიყენება ერთად და მოიპოვება ცოდნა იმის შესახებ, თუ რომელი მონაცემებია სტატისტიკური თვალსაზრისით დაჯგუფებადი, ან რომელი კატეგორიის ჯგუფია ჯერ კიდევ ამოუცნობი და აღუწერელი, და ამავე დროს განისაზღვრება ის წესები და ურთიერთდამოკიდებულებები, რომელიც საანალიზო ელემენტების სივრცეშია მოცემული.

4.3. Data Mining -ის გამოყენების სფეროები

მონაცემთა მაინინგს მომოვალში აქვს ძალიან დიდი გამოყენების პოტენციალი, თუმცა დღესაც გამოიყენება მრავალი მიმართულებით მაგ. საბანკო სექტორში, სადაზღვევო სისტემებსი, ტელეკომუნიკაციაში მედიცინაში და ა.შ. [44].

- **Data Mining საბანკო საქმიანობაში**

Data Mining- ის მოწინავე ტექნოლოგიების გამოყენება საბანკო სექტორში, იგი ხელს უწყობს შემდეგი ამოცანების შესრულებას:

- *საკრედიტო გარათების თაღლითური მოხმარების გამოვლენა.* წინა შესრულებული ტრანზაქციების ანალიზის შედეგად, რაც შემდგომში აღმოჩნდა თაღლითური, ბანკს აძლევს შესაძლებლობას გამოავლინოს მსგავსი თაღლითობის გარკვეული სტერეოტიპი;
- *მომხმარებელთა (კლიენტთა) სეგმენტაცია.* მომხმარებლის სხვადასხვა კატეგორიებად დაყოფა, ხელს უწყობს ბანკების მარკეტინგული პოლიტიკის მიზანმიმართულ და ეფექტურ განვითარებას, რაც გულისხმობს სხვადასხვა სახის სხვადასხვა

მომხმარებლთა ჯგუფებისათვის შესაბამისი სერვისების შეთავაზებას;

- მომხმარებ*ლის ცვალებადობის პროგნოზირება.* Data Mining ხელს უწყობს ბანკებს, საკუთარი მომხმარებლის ფასეულობათა განსაზღვრაში და შესაბამისი კატეგორიის მოდელის პროგნოზირების ჩამოყალიბებაში, რის საფუძველზეც დგინდება მომხმარებელზე ორიენტირებული განსაზღვრული წესების ჩამონათვალი თითოეული კატეგორიის მომსახურებაში.

➢ Data Mining ტელეკომუნიკაციებში

ტელეკომუნიკაციების სფეროში Data Mining-ი ხელს უწყობს კომპანიებს უფრო აქტიურად წარადგინონ მარკეტინგის და ფასების საკუთარი პროგრამები, რათა შეინარჩუნონ არსებული მომხმარებელი და მოიზიდონ ახალი. ტიპიურ ღონისძიებათა შორის, აღსანიშნავია შემდეგი:

- გამოწვევების დეტალური მახასიათებლების შესახებ ჩანაწერების ანალიზი: ამგვარი ანალიზის ჩატარება - ემსახურება მსგავსი სტერეოტიპების მქონე მომხმარებელთა კატეგორიების გამოვლენას, მათი მომსახურებისას სასურველი ფასების და მომსახურების პაკეტის შემუშავებას;

- მომხმარებელთა ლოიალობის გამოვლენა: შესაძლებელია Data Mining-ის გამოყენება იმ მომხმარებლის დახასიათების ამოცნობის მიზნით, რომელმაც ერთხელ მაინც ისარგებლა მოცემული კომპანიის მომსახურებით და დიდი ალბათობით შეინარჩუნებს ერთგულებას კომპანიის მიმართ. შესაბამისად, მარკეტინგისთვის გამოყოფილი თანხების გამოყენება მიზანშეწონილია გარანტირებული დაბრუნების სფეროში.

➢ Data Mining დაზღვევაში

წლების განმავლობაში სადაზღვევო კომპანიები ახორციელებენ მონაცემთა ბაზის დიდი მოცულობის შევსება. ამ სფეროში

Data Mining-ს გააჩნია მოქმედებისა და მეთოდების გამოყენებისათვის დიდი შესაძლებლობები.

- თაღლითობის გამოვლენა: სადაზღვევო კომპანიებს შესწევთ უნარი შეამცირონ თაღლითობის ზღვარი, გარკვეული სტერეოტიპების შესაბამისი გადახდილი სადაზღვევო ანაზღაურების განცხადებებში მოძიების მეთოდით, რაც ექიმებს, იურისტებსა და განმცხადებელს შორის ურთიერთობის დამახასიათებელ ფაქტორზე იქნება დაფუძნებული.

- რისკის შეფასების ანალიზი. გადახდილ განაცხადებთან დაკავშირებული რიგი ფაქტორების გამოვლენის შედეგად, დამზღვეველს შეუძლია შეამციროს საკუთარი ზარალი ვალდებულებებზე. ცნობილია შემთხვევა, როდესაც აშშ-ში მსხვილმა სადაზღვევო კომპანიამ გამოავლინა, რომ ქორწინებაში მყოფ დაავადებულ პირებზე გადახდილი თანხები, ორმაგად აღემატებოდა თანხებს, რაც გადახდილი იყო მარტოხელა პირებზე. კომპანიამ მოახდინა დაუყონებლივი რეაგირება აღნიშნულ ფაქტზე, მან შეცვალა საკუთარი ზოგადი პოლიტიკა დაოჯახებულ მომხმარებლეზე შეღავათიანი სესხების გაცემის წესებში და დაზოგა მილიონობით თანხები.

შესაძლოა DataMining-ის გამოყენება სხვა სფეროებშიც, მაგალითად:

- საავტომობილო მრეწველობის განვითარება. ავტომობილების აწყობისას აუცილებელია თითოეული მომხმარებლის მოთხოვნის გათვალისწინება, აქედან გამომდინარე საჭიროა გარკვეული, მოთხოვნადი მახასიათებლების პროგნოზირება და ცოდნა იმისა თუ რაზეა მომხმარებელი ორიენტირებული;

- გარანტიების პოლიტიკა. მწარმოებელმა უნდა ივარაუდოს მომხმარებელთა ის რაოდენობა, რომელმაც შესაძლოა წარადგინოს საგარანტიო მოთხოვნები და მოთხოვნების საშუალო ღირებულება;

- ავიახაზებით ხშირად მოგზაური მომხმარებლების წახალისება. შესაძლებელია რომ ავიაკომპანიებმა გამოავლინონ მომხმარებელთა ჯგუფი, რომელიც ხშირად ისარგებლებს აღნიშნული ავიაკომპანიის მომსახურებით.

4.4. სპეციალური პროგრამები

- **მედიცინა**

არსებობს სამედიცინო დიაგნოსტიკის რამდენიმე საექსპერტო სისტემა. ის ძირითადად ეყრდნობა წესებს, რომლებიც აღწერს სხვადასხვა დაავადებების და სხვადასხვა სიმპტომების ჯგუფს. ასეთი წესების დახმარებით, შესაძლებელია დადგინდეს არა მარტო პაციენტის დაავადება, არამედ მისი მკურნალობის მეთოდიკა.

აღნიშნული წესების მიხედვით, ასევე შესაძლებელი ხდება მედიკამენტოზული საშუალებების შერჩევა, მათი ჩვენების განსაზღვრა - უკუჩვენება და სამკურნალო პროცედურებზე ორიენტირება, გაცილებით ეფექტური მკურნალობის პირობების შემუშავება.

Data Mining-ის ტექნოლოგიის გამოყენება უზრუნველყოფს სამედიცინო მონაცემებში შაბლონების შექმნას, რაც წარმოადგენს აღნიშნული წესების საფუძველს და შესაძლებელს ხდის ონლაინ რეჟიმში პაციენტის დიაგნოსტირებას და დანიშნულების მიცემას.

- **ტექსტური მაინინგი**

Data Mining-ის განსაკუთრებული ფორმაა ტექსტური მაინინგი. იგი მოიცავს ისეთ ძირითად მეთოდებს, რომელიც ემსახურება არასტრუქტურირებულ ტექსტურ მონაცემთა დამუშავებას.

ტექსტური მაინინგის დახმარებით შესაძლებელია ცოდნის ექსტრაგირება ტექსტურ მონაცემებში. მომხმარებელს ეძლევა საშუალება ავტომატურად მიიღოს ძირითადი ტექტური მონაცემები

მათი დეტალურად დანაწევრების გარეშე. მაგალითად, ტექსტური მაინინგი შესაძლებელს ხდის დიდი რაოდენობის სტატიებიდან ამოარჩიოს მხოლოდ ის მნიშვნელოვანი ინფორმაცია, რომელიც საჭიროა რომელიმე კონკრეტული პროექტის რეალიზებისათვის.

ასევე ცნობილია, რომ უმეტეს შემთხვევაში თითქმის ყველა კომპანიაში ინფორმაცია წარმოდგენილია ტექსტური ფორმით, ტექსტური მაინინგი უზრუნველყოფს კომპანიის შესახებ მხოლოდ მნიშვნელოვანი ინფორმაციის ამოღებას და Data Mining-ში წარმოდგენას

4.5. ალგორითმები მონაცემთა მაინინგში

Data Mining ანალიზის პროცესში იყენებს ალგორითმებს, განსხვავებული ტიპის კატეგორიისა და სტრუქტურის მონაცემებისათვის. ალგორითმი შეიცავს კლასიფიქტორებს, რომელთაც აქვს განსხვავებული სტრუქტურა.

კლასიფიკატორი ალგორითმში წინასწარ საზღვრავს მონაცემებს კლასიფიკაციის მიხედვით, თუ სად განთავსდეს და რომელ კლასს ან კატეგორიას მიეკუთვნოს ახალი მონაცემი.

არსებობს რამდენიმე ცნობილი ალგორითმი, რომელთა გამოყენებით შესაძლებელი ხდება მონაცემთა ვიზუალიზაცია, დამუშავება და პრიორიტეტების განსაზღვრა. განვიხილოთ რამდენიმე ალგორითმი.

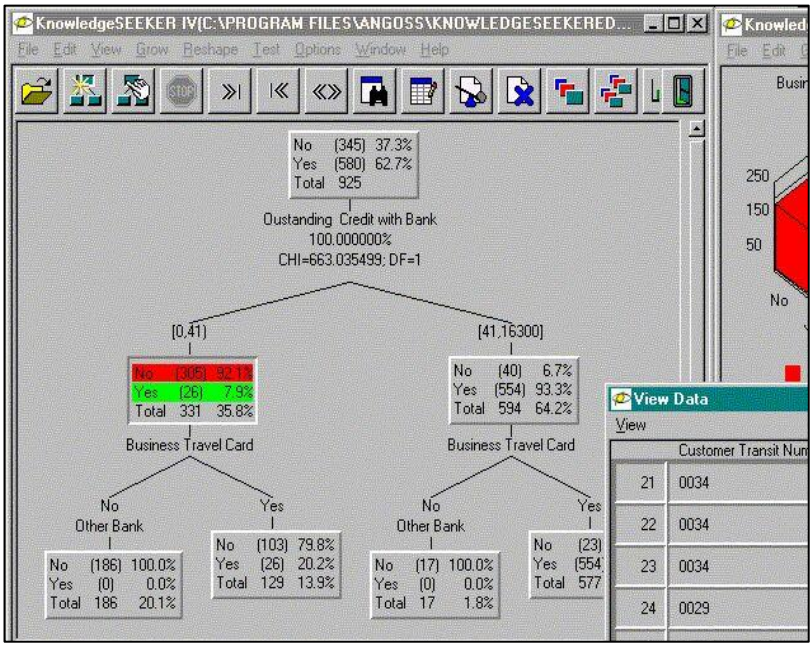
- **ალგორითმი C4.5**

ალგორითმი **C4.5,** აგებულია კლასიფიკატორზე, რომელსაც აქვს გადაწყვეტილებათა მიღების ხის ფუნქცია [46].

გადაწყვეტილებათა მიღების ხე **(decision trees)** მონაცემთა მაინინგში არის ერთ-ერთი ყველაზე განსაკუთრებული მიდგომა, სადაც ხის იერარქიულ სტრუქტურა კლასიფიცირებულია „თუ-მაშინ“ (if-then) ლოგიკით. გადაწყვეტილების მიღებისას ხდება

განსაზღვრა, თუ რომელ კლასს მიეკუთვნის რომელიმე ობიექტი ან არსებული სიტუაცია, ამავე დროს ხდება პასუხის გაეცემა კითხვებზე, რომელიც ხის კვანძებზეა განთავსებული და ღებულობს დადებით ან უარყოფით პასუხს.

საკითხი დგას შემდეგნაირად, მაგალითად, „A პარამეტრის მნიშვნელობა მეტია x ”-ზე“, თუ პასუხი დადებითია, მაშინ გადასასვლელი იხსნება ხის მარჯვენა კვანძის მომდევნო დონეზე, ხოლო უარყოფითი პასუხის შემთხვევაში მარცხენა მხარეს. შემდეგ ისევ დგება მომდევნო კვანძის შესაბამისი კითხვა და ა.შ. ხის სტრუქტურა საკმაოდ მარტივი და თვალსაჩინოა 4.2 ნახაზზე წარმოდგენილია გადაწყვეტილებათა მიღების ხის სტრუქტურის ფრაგმენტი.



ნახ.4.2 გადაწყვეტილებათა მიღების ხის სტრუქტურა

გადაწყვეტილებათა ხე შესაძლებელია არ იყოს ერთადერთი საუკეთესო გამოსავალი მონაცემთა წარმოდგენისას, მაგრამ თანმიმდევრულად ხდება იმ ძირითადი მახასიათებლების წარმოდგენა და შეფასება, რომელიც უშუალოდ განსაზღვრავს მონაცემის შესაბამისობას ამა თუ ინ კლასისი მახასიათებელ ძირითად პარამეტრებთან.

დღეს ეს მეთოდი საკმაოდ გავრცელებულია და იყენებენ ისეთ დიდ სისტემებში როგორიცაა: See5/C5.0 (RuleQuest, ავსტრალია), Clementine (Integral Solutions, დიდი ბრიტანეთი), SIPINA (University of Lyon, საფრანგეთი), IDIS (Information Discovery, ამერიკა), KnowledgeSeeker (ANGOSS, კანადა) და ა.შ.

როგორც დასაწყისში აღვნიშნეთ, ალგორითმი *C4.5* მონაცემთა წარმოდგენისას იყენებს გადაწყვეტილებათა მიღების ხეს, სადაც მონაცემები წინასწარ უნდა იყოს კლასიფიცირებული.

კლასიფიკატორი არის ინსტრუმენტი, რომელიც გამოიყენება მონაცემთა მაინინგში. იგი შედგება კლასიფიციცირებული მონაცემებისაგან, რომლის საფუძველზეც ხდება წინასწარ განსაზღვრა, თუ რომელ კლასს უნდა მიეკუთვნოს ახალი მონაცემი.

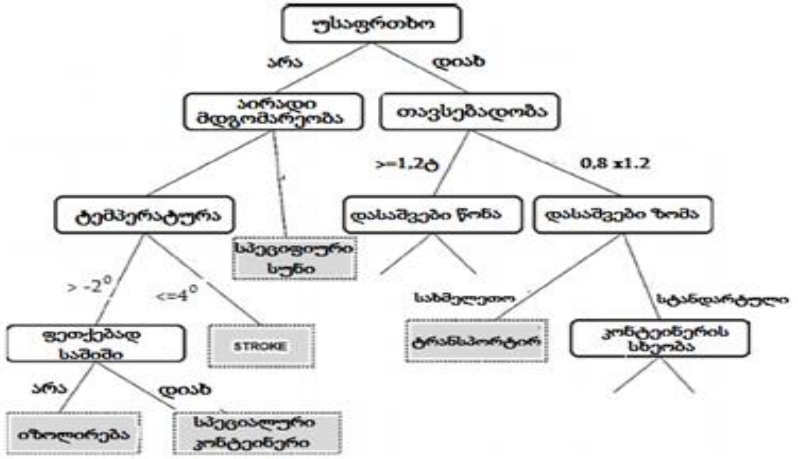
მარტივი მაგალითის საფუძველზე ეს პროცესი შეიძლება ასე ავხსნათ: მაგალითად, სატრანსპორტო გადაზიდვებში განსხვავებული ტვირთების შესახებ გვაქვს მონაცემთა ჯგუფები, რომელიც მოიცავს რამდენიმე ძირითად ატრიბუტს.

ატრიბუტების საშუალებით ხდება ტვირთის აღწერა, მაგალითად, ლოკაციის ადგილი, ტრანსპორტირების საბოლოო ადგილი, ტვირთის სახეობა, ღირებულება, მოცულობა, წონა, ზომა, კონტეინერთან თავსებადობა და ა.შ. ჩვენ წინასწარ ვიცით განსხვავებული ტვირთების შესახებ ძირითადი მახასიათებელი პარამეტრები, რომელთაც ატრიბუტების სახით ვაერთიანებთ კლასებად, აღნიშნული ატრიბუტების საფუძველძე წინასწარ გვინდა განვსაზღვროთ - ტვირთი საშიშია თუ არა. ტვირთი

შესაძლებელია მოხვდეს ორი კლასიდან ერთში, საშიში ან უსაფრთხო. ალგორითმი შეტყონინებას უგზავნის თითოეული ტვირთის შესაბამის კლასს.

ტვირთის განმსაზღვრელი ატრიბუტების და შესაბამისი კლასების საფუძველზე ალგორითმი C4.5 აგებს გადაწყვეტილებათა ხეს, რომელის დახმარებით წინასწარ ატრიბუტების საფუძველზე მოხდება ტვირთის შესაბამისი კლასისი განსაზღვრა.

ტვირთის კლასიფიკაციის განსაზღვრა გადაწყვეტილებათა ხის მეთოდის გამოყენებით ხდება სპეციალური ბლოკ-სქემის საშუალებით, მაგალითად: თავსებადობა; მდგომარეობა; იზოლირება; სპეციფიური სუნი; განსაზღვრულ ტემპერატურაზე ტრანსპორტირება; სპეციალური კონტეინერი; დაზიანების საფრთხე; აალებადი; მაკოროზირებელი და ა. შ. ბლოკ-სქემა ყოველი წერტილიდან აგზავნის შეკითხვას ამა თუ იმ ატრიბუტის შესახებ. მოცემული ატრიბუტების გათვალისწინებით განისაზღვრება, თუ რომელ კლასს განეკუთვნოს მოცემული პროდუქტი. 4.3 ნახაზზე მოცემულია გადაწყვეტილებათა მიღების ხე.



ნახ.4.3. გადაწყვეტილებათა მიღების ხე

C4.5 ალგორითმში გადაწყვეტილებათა მიღების ხის გამოყენებით მიღებული უპირატესობები:

- ალგორიათმი C4.5 იყენებს შემომავალ ინფორმაციულ ნაკადს, რის საფუძველზევ იგება გადაწყვეტილებათა მიღების ხე.
- ალგორითმს C4.5, აქვს ერთ არხიანი დატოტვილი გადაწყვეტილებათა მიღების ხის, ფორმა, სადაცდანაწევრებული ტოტები აუმჯობესებს მოდელის მუშაობას.
- ალგორითმი C4.5-ს აქვს შესაძლებლობა იმუშაოს როგორც დისკრეტულ ასევე უწყვეტ მნიშვნელობებთან. ალგორითმი ქნის შეზღუდვის დიაპაზონს, სადაც ადგენს მონაცემთა საზღვრებს და უწყვეტი მონაცემები გადაყავს დისკრეტულში.
- გამოტოვებული მონაცემები, რომელიც არ კლასიფიცირდება, გამოიმუშავებენ თავის საკუთარ შესაძლებლობებს, რათა დეტალიზების საშუალებით დაუახლოვდნენ რომელიმე კლასტერს.

➢ *მეთოდი „k-საშუალო“*

„k-საშუალო“ მეთოდი ქმნის ობიექტთა ჯგუფს, ისე რომ ჯგუფის მახასიათებელი წევრები არის ერთგვაროვანი. ეს არის კლასტერული ანალიზის ცნობილი მეთოდი, რომელიც გამოიყენება მონაცემთა ჯგუფების კვლევისას.

კლასტერული ანალიზი – არის ალგორითმების ერთობლიობა, რომელიც ახდენს ჯგუფების ფორმირებას, ისე რომ ჯგუფის წევრები მეტნაკლებად ერთმანეთის მსგავსია და მკვეთრად განსხვავდება ჯგუფის არაწევრი ელემენტებისგან. კლასტერი და ჯგუფი კლასტერულ ანალიზში არის სინონიმები.

კლასტერული ანალიზის ჩატარებისას მაგ. როდესაც განიხილება ტვირთის შესაბამისი მახასიათებელი ნიშნ-თვისებები, კლასტერულ ანალიზში უწოდებენ დაკვირვებას. წინასწარ ყოველთვის ცნობილია გარკვეული სახის ინფორმაცია ტვირვის შესახებ, მაგ. ტიპი, სახეობა, მოცულობა, ზომა, წონა და ა.შ. მას პირობითად ვუწოდებთ ვექტორს, რომელიც აღწერს ტვირთს.

ვექტორი შესაძლებელია წარმოვიდგინო როგორც ჩამონათვალში მოცემული რიცხვები, რომელიც კოორდინატებად ინტერპრეტირდება მრავალგანზომილებიან სივრცეში. მაგალითად, ტიპი ერთ განზომილებაში, მოცულობა მეორეში და ა.შ., მაგრამ იმისთვის რომ ერთმანეთთან დავაჯგუფოთ კონკრეტული ტვირთის შესაბამისი ტიპი, ზომა და წონა მოცემული ვექტორის დახმარებით, გამოიყენება „k-საშუალოს“ მეთოდი. მისი გამოყენებით, განსხვავებული ტიპის მონაცემების იტერაციის შედეგად მიიღება განსხვავებულ ვარიანტთა სიმრავლე, რაც წარმოდგენილია შემდეგ პუნქტებად, რომელთა ვიზუალიზაცია მოცემულია 4.4 ნახაზზე.

k-საშუალო მეთოდი მრავალგანზომილებიან სივრციდან ირჩევს წერტილებს, რომელიც შემდგომში იტერაციის შედეგად წარმოდგენილი იქნება, როგორც ახალი k კლასტერი. ამ წერტილებს ეწოდება დატვირთვის ცენტრები.

1. ყოველი ტვირთის შესაბამისი ატრიბუტები მიახლოებული უნდა იყოს მოცემული სივრცის დატვირთვის წერტილებთან, რის შედეგადაც მათ გარშემო ჩამოყალიბდება ახალი კლასტერი;

2. მიღებულ k-კლასტერში ყოველთვის აღმოჩნდება ახალი ტვირთის სახეობა, დამახასიათებელი ატრიბუტების შესაბამისად;

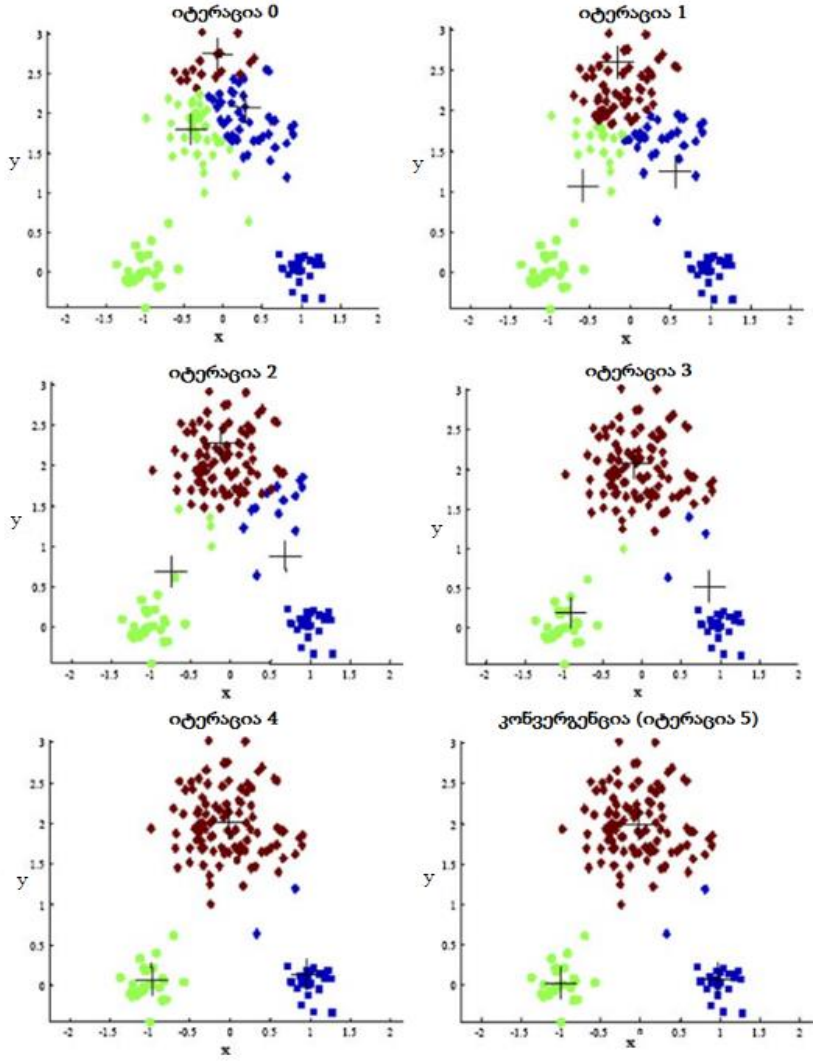
3. k-საშუალო მეთოდის გათვალისწინები კლასტერების წევრების ცენტრი მდებარეობს k-კლასტერში;

4. მიღებული ცენტრი კლასტერის დატვირთვის ცენტრია;

5. შესაძლებელია დატვირთვის ცენტრის შეცვლა, ტვირთი შესაბამისი მახასიათებლების გათვალისწინებით ანალიზის პროცესში შესაძლებელია აღმოჩნდეს დატვირთვის სხვა ცენტრში, ანუ შეუძლია შეიცვალოს ადგილი და გადაინაცვლოს სხვა კლასტერში;

6. ბიჯი 2-6 მეორდება მანამ სანამ დატვირთვის ცენტრი არ შეწყვეტს ცვლილებას და მიიღებს სტაბილურ სახეს, მას უწოდებენ კონვერგენციას (დაახლოებას).

სქემატურად დატვირთვის ცენტრის განსაზღვრა და ახალი k - კლასტერის ჩამოყალიბების პროცესი მოცემულია 4.3 ნახაზზე.



ნახ.4.4. ახალი კლასტერის ჩამოყალიბების პროცესი

აღნიშნული ალგორითმის გამოყენების უპირატესობა მის სწრაფქმედებაში, ეფექტურობასა და სიმარტივეშია. k-საშუალო მეთოდი გამოიყენება დიდ მონაცემთა ჯგუფის დანაწევრებისთვისაც, რის შემდეგაც ტარდება შედარებით დიდი მოცულობის კლასტერების ანალიზი და ქვეკლასტერებად წარმოდგენა. ეს მეთოდი ასევე გამოიყენება კლასტერების რაოდენობის შესამცირებლად.

აღნიშნული ალგორითმის სუსტ მხარედ შესაძლებელია ჩაითვალის ის, რომ იგი არ გამოიყენება დისკრეტულ მონაცემებში. გამოყენების ძირითად სფეროებია: Apache Mahout; Julia; R; SciPy; Weka; MATLAB; SAS. [ ].

- **აპრიორების ალგორითმი**

აპრიორების ალგორითმი ეძებს ასოციაციურ წესებს და კანონზომიერებებს მონაცემთა ბაზაში, სადაც არის დიდი რაოდენობის ტრანზაქციები. ასოციაციურ წესებში მოიაზრება გარკვეული სახის ტექნიკური საშუალება, რომელიც გამოიყენება მონაცემთა მაინინგში, რათა მონაცემთა ბაზაში ცვლადებს შორის განისაზღვროს დამოკიდებულება [47].

აპრიორულ ალგორითმზე განვიხილოთ მარტივი მაგალითი, რომელიც შეეხება სუპერმარკეტის მონაცემთა ბაზის ტრანზაქციებს. მონაცემთა ბაზა შესაძლებელია განვიხილოთ, როგორც დიდი მოცულობის ცხრილი, სადაც ყოველი სტრიქონი არის ტრანზაქციების ნომერი, ხოლო სვეტები შეიცავს ცალკეული გაყიდვების შესახებ ინფორმაციას (ნახ.4.5).

Transaction ID	არაჟანი	რძე	კარაქი	ყველი	ვაშლი
1	X	X	X		
2	X	X			X
3	X		X	X	

ნახ.4.5

აპრიორების ალგორითმის გამოყენებით შესაძლებელია განისაზღვროს ის პროდუქტები, რომელსაც მომხმარებელი ყიდულობს ერთად, ასე რომ განისაზღვრება მათ შორის ასოციაციური დამოკიდებულება, რაც მოგვცემს საშუალებას განვსაზღვროთ ის პროდუქტები, რომელიც ხშირ შემთხვევაში იყიდება ერთად.

ძირითადი მარკეტინგული ამოცანაა, რაც შეიძლება მეტი პროდუქტი შეიძინოს მომხმარებელმა, აღნიშნული მეთოდის გამოყენება მარკეტინგის მენეჯერს უადვილებს მომხმარებლისთვის შეარჩიოს ის პროდუქტი ან პროდუქტები რაზეც იოლად მიიღებს შეძენის გადაწყვეტილებას.

ერთმანეთზე დამოკიდებულ პროდუქტებს ეწოდება ჯგუფი. ერთი ჯგუფის პროდუქტები, მაგალითად, არაჟანი, რძე, კარაქი ხშირად შეგვინიშნავს, რომ მარკეტში თაროებზე მოთავსებულია ერთმანეთის გვერდით. ორი რომელიმე სახის პროდუქტს, რომელზეც მომხმარებელი უმეტეს შემთხვევაში აკეთებს არჩევანს უწოდებენ ორელემენტიან ჯგუფს. როდესაც მონაცემთა ბაზა საკმაოდ დიდია, უფრო რთულია პროდუქტებს შორის დამოკიდებულების დანახვა, განსაკუთრებით მაშინ როდესაც ხდება სამელემენტიანი, ოთხი ან უფრო რთული ჯგუფის განსაზღვრა, სწორედ ასეთ შემთხვევებში გამოიყენება აპრიორების ალგორითმი. სანამ უშუალოდ ალგორითმს განვსაზღვრავთ უნდა ჩამოვაყალიბოთ სამი ძირითადი პარამეტრი:

1. პირველ ეტაპზე უნდა განისაზღვროს ჯგუფის **ზომა**, გვინდა განვსაზღვროთ ორელემენტიანი, სამელემენტიანი თუ სხვა რომელიმე;

2. მეორე ეტაპზე განისაზღვრება **მხარდაჭერა** - ტრანზაქციების საერთო რაოდენობიდან იმ ტრანზაქციების გამოყოფა და რაოდენობის განსაზღვრა, რომელიც შედის ჯგუფში. ჯგუფი

რომელიც უზრუნველყოფილია მხარდაჭერით, ის ითვლება ყველაზე ხშირად განმეორებად ჯგუფად;

3. მესამე ეტაპზე განისაზღვრება **საიმედოობა,** ალბათობა იმისა, რომ კონკრეტული პროდუქტი მოხვდება კალათაში სხვა პროდუქტთან ერთად. მაგალითად, არაჟანს აქვს 65% ალბათობა იმისა რომ მოხვდება რძესთან ერთად კალათაში.

აპრიორების ალგორითმი შედგება სამი ძირითადი ბიჯისგან:

1. **გაერთიანება.** მონაცემთა ბაზაში უნდა განისაზღვროს ცალკეული პროდუქტის განმეორების სიხშირე;
2. **განცალკევება.** ის ჯგუფები, რომელთაც აქვთ მხარდაჭერა და საიმედოობის საკმარისი მაჩვენებელი გადადის მომდევნო იტერაციაზე ორ კომპონენტიან ჯგუფში;
3. **განმეორება.** პირველი ორი ბიჯი მეორდება ჯგუფში შემავალი ყოველი სიდიდისთვის მანამ, სანამ არ მიიღება ადრე განსაზღვრული ზომა.

- **EM-ალგორითმი**

მონაცემთა მაინინგში გამოიყენება **მოლოდინის მაქსიმიზაციის** კლასტერული ალგორითმი (expectation-maximization - EM), რომლის დანიშნულებაცაა „ცოდნის“ აღმოჩენა.

მათემატიკურ სტატისტიკაში EM-ალგორითმი ითვლება იტერაციულად და გამოიყენება სტატისტიკური მოდელის მაქსიმალურად მსგავსი პარამეტრების გამოთვლისას, როდესაც ცვლადები დაფარულია. სტატისტიკური მოდელები არის ისეთი მოდელები, როდესაც აღწერილია უკვე ცნობილი, დადასტურებული მონაცემები, მაგალითად, გამოცდაზე მიღებული შეფასება შესაძლებელია წარმოვადგინოთ როგორც ნორმალური განაწილება, ამიტომ მოსალოდნელია, რომ შეფასებები დაგენერირდეს შესაბამისად ნორმალურ განაწილების მოდელში.

განაწილება წარმოადგენს ყველა იმ ალბათურ შედეგს, რომელიც შეიძლება მიღებული იქნას გამოცდის შედეგად.

მაგალითად, გამოცდაზე მიღებული შეფასება შეესაბამება ნორმალურ განაწილებას, სადაც არის ალბათობა იმისა, რომ წარმოდგენილი იქნება გამოცდაზე მიღებული ყველა სავარაუდო შედეგი. ე.ი. განაწილება გვეხმარება იმაში, გავიგოთ გამოცდაზე გასულმა რამდენმა ადამიანმა მიიღო ესა თუ ის შეფასება.

მოდელის მნიშვნელოვანი მახასიათებელია პარამეტრი, რომელიც აღწერს მოდელის ძირითად ნაწილს, განაწილებას.

მაგალითად, საშუალო განაწილება აღიწერება საშუალო არითმეტიკულით და დისპერსიით. ზოგადად ნორმალური განაწილებისათვის აუცილებელია ორი პარამეტრის განსაზღვრა:

1. საშუალო არითმეტიკული;
2. დისპერსია.

იმ შემთხვევაში, როდესაც არ ვიცით ყველა შეფასების საშუალო არითმეტიკული ან დისპერია, შესაძლებელია შეფასდეს მხოლოდ ერთი მაგალითის მონაცემები.

EM-ალგორითმის ერთ-ერთი მნიშვნელოვანი მახასიათებელია, ხდომილება.

ხდომილება – არის ალბათობა იმისა თუ რამდენად მოხდა გადახრა ნორმალური განაწილებიდან. ანუ ალბათობა იმისა, ნორმალური განაწილების მრუდი რამდენად სწორად აღწერს გამოცდაზე მიღებული შედეგების საშუალო არითმეტიკულს და დისპერსის.

მონაცემთა მაინინგსა და კლასტერიზაციაში მნიშვნელოვანია შეფასდეს კლასები მასში გაერთიანებული მონაცემების შესაბამისად, როგორც გამოტოვებული მონაცემები, რადგან თავიდან უცნობია კლასისი სახეობა, ამიტომ გამოტოვებული მონაცემების იტერაცია საკმაოდ მნიშვნელოვანი პროცესია კლასტერიზაციის ამოცანაში EM-ალგორითმის გამოყენებისას.

EM-ალგორითმი არის იტერაციული ალგორითმი, რომელიც გამოიყენება მაქსიმალურად თანხვედრი პარამეტრების მქონე

სავარაუდო მოდელის შესაფასებლად. როდესაც მოდელში არის რამდენიმე არაცნობადი პარამეტრი, ფასდება მათი მაქსიმალური თანხვედრა სხვა მოდელთან, რის შედეგადაც EM - ალგორითმი ქმნის ახალ მოდელს. ახალ მონაცემთა ერთობლიობა აისახება როგორც კლასი. კლასტერიზაციის პროცესში კი სრულდება სამ ბიჯიანი იტერაციული პროცესი:

1. E - ბიჯი: აღნიშნულ ბიჯზე მოდელის ძირითადი პარამეტრების საფუძველზე გამოითვლება ალბათობა იმისა, ეკუთვნის თუ არა ეს მონაცემები მოცემულ კლასტერს;
2. M - ბიჯი: ხდება მოდელის პარამეტრების განახლება შესაბამის კლასტერულ განაწილებაში, რომელიც ჩატარდა E ბიჯზე;
3. პირველი ორი ბიჯი მეორდება მანამ, სანამ მოდელის პარამეტრები და კლასტერული განაწილება არ გათანაბრდება.

EM-ალგორითმის მთავარი ფასეულობაა გამოყენების სიმარტივე, ასევე მას შეუძლია მოახდინოს არა მარტო მოდელის პარამეტრების ოპტიმიზაცია, არამეტ შეუძლია განსაზღვროს რამდენად ღირებულია გამოტოვებული მონაცემები მოდელისათვის. EM - შესაძლებელია ჩავთვალოთ, როგორც საუკეთესო მეთოდი კლასტერიზაციისა და პარამეტრებზე დამოკიდებული მოდელის შექმნისათვის.

- **PageRank ალგორითმი**

PageRank – არის ალგორითმი ე.წ. რანჟირებული ბმა. იგი განსაზღვრავს ობიექტის მნიშვნელობას და კავშირს ქსელში არსებულ სხვა ობიექტებთან. აღნიშნულ ალგორითმს ეფექტურად იყენებენ საძიებო სისტემები, მაგალითად, Google, Wikipeida და ა.შ.

რანჟირებული ბმა არის ქსელური ანალიზის ტიპი, რომელიც განსაზღვრავს ობიექტებს შორის (წაიკითხე, დააკავშირე) ასოციაციურ კავშირს.

ინტერნეტის თითქმის ყველა ვებ-გვერდი კავშირშია ერთმანეთთან. მაგალითად, თუ გავააქტიურებთ ვებ-გვერდს gtu.ge, მივიღებთ ბმას http://eqe.ge/ -ზე, სადაც განთავსებულია „განათლების ხარისხის განვითარების ეროვნული ცენტრი“, ეს ნიშნავს, რომ გვერდი gtu.ge რელევანტურად თვლის გვერდს eqe.ge - ს. ამავე დროს აღსანიშნავია ის გარემოება, რომ ნებისმიერი ვებ გვერდი, სადაც ხდება მიმართვა gtu.ge - დან, ამაღლებს gtu.ge -ს რელევანტურობას. მეთოდი PageRank ვებ-გვერდებს ანიჭებს 0-დან 10-მდე პრიორიტეტს.

4.1. ცხრილში მოცემულია კომპანია Google-ს მიერ გამოქვეყნებული რეიტინგი.

ცხრ.4.1

Website	PageRank
twitter.com	10
facebook.com	9
reddit.com	8
stackoverflow.com	7
tumblr.com	6
crucial.com	5
programmingzen.com	4
dearblogger.org	3

ცხრილში მაქსიმალური მნიშვნელობა (10 ქულა) ენიჭება ვებ-გვერდს, რომელიც ყველაზე პოპულარული და რელევანტურია.

ალგორითმი PageRank სპეციალურადაა შექმნილი გლობალური ქსელისათვის, რომელსაც ადამიანების დამოკიდებულება გადაყავს ციფრებში.

PageRank ალგორითმი არის სუპერეფექტური საშუალება, რომელიც ახდენს ბმების რანჟირებას. გასათვალისწინებელია ის

გარემოება, რომ დასაკავშირებელი ობიექტები არაა აუცილებელი იყოს ვებ-გვერდები [48].

PageRank ალგორითმის ინოვაციური გამოყენების სფეროებია:

1. *ეკოლოგია,* სადაც აღნიშნული ალგორითმის გამოყენებით განისაზღვრება ეკოსისტემების სასიცოცხლო ციკლი;

2. ტვიტერმა PageRank ალგორითმის გამოყენებით შეიმუშავა WTF (Who-to-Follow) – პერსონალიზებული სარეკომენდაციო ვარიანტები, სადაც ჩამონათვალში წარმოდგენილია იმ ადამიანთა სია, რომელთაც სჭირდებათ სხვადასხვა სახის შეთავაზებები და რეკომენდაციები;

3. PageRank ალგორითმი აქტიურად გამოყენება ჰონკონგის პოლიტექნიკური უნივერსიტეტის პროფესორის ბინ ჟენის (Bin Jiang) მიერ ტოპოლოგიურ ჩანაწერებში, სადაც წინასწარ ხდება ადამიანების აქტიური მოძრაობის განსაზღვრა.

PageRank ალგორითმის მთავარი ღირებულება არის საიმედოობა, მიუხედავად იმ სირთულისა, რომელიც უკავშირდება რელევანტური ბმის პროცესს.

გრაფიკული ან სქემური მონაცემების შესაბამისი პარამეტრების, პრიორიტეტებისა და რელევანტურობის განსაზღვრისათვის ასევე ყველაზე ეფექტური საშუალებაა PageRank -ის გამოყენება.

საფირმო ნიშანი PageRank ეკუთვნის Google კომპანიას. ალგორითმი PageRank შეიქმნა და დაპატენტდა სტენფორდის უნივერსიტეტში.

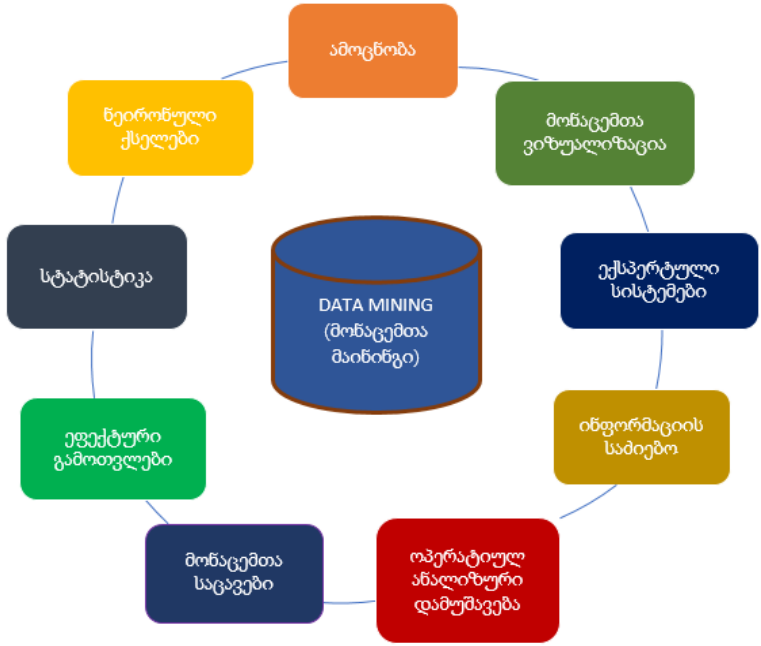
PageRank ალგორითმი ასევე რეალიზებულია შემდეგ პროგრამულ პაკეტებში:

1. C++ OpenSource PageRank;
2. Python PageRank;
3. ქსელური ანალიზის პაკეტი - igraph

ზოგადად შესაძლებელია ითქვას, რომ მონაცემთა მაინინგი არის მულტიდისციპლინარულ დარგი, სადაც მონაცემთა ბაზების

შეფასება ხდება გამოყენებითი სტატისტიკის მეშვეობით, ხოლო ამოცნობის თვალსაზრისით გამოიყენება ხელოვნური იტელექტის მეთოდები, მონაცემთა ბაზების თეორია და ა.შ.

4.6. ნახაზზე მოცემულია ის სისტემები და ტექნოლოგიები, რომელიც უშუალო კავშირშია მონაცემთა მაინინგთან.



ნახ.4.6. Data Mining - მულტიდისციპლინარული დარგი

წარმოდგენილი სისტემებიდან ბევრი ინტეგრირდება სხვა მეთოდებთან და ტექნოლოგიებთან, მაგრამ თითოეული მათგანი შეიცავს ე.წ. გასაღებურ კომპონენტს, რაც მნიშვნელოვანს და ეფექტურს ხდის მათში Data Mining-ის გამოყენებას.

V თავი
მონაცემთა ბაზა MongoDB

5.1. მონაცემთა ბაზის აგება MongoDB გარემოში

MongoDB-ში ბაზას ცხადად არ ვქმნით, ვირჩევთ მიმდინარე ბაზას (რომელიც ახალი ბაზის შექმნის დროს რეალურად არც არსებობს) და არჩეულ ბაზაში უბრალოდ ვამატებთ ახალ ჩანაწერებს. თვითონ მონაცემთა ბაზა პირველივე მონაცემის ჩაწერის პარალელურად, ავტომატურად იქმნება.

განვიხილოთ საცდელი ბაზის შექმნის მაგალითი MongoDB სისტემისთვის [16,58]. კონკრეტული ოპერაციებისა და მაგალითების განხილვამდე შევქმნათ moviesDB მონაცემთა ბაზა დანართის მიხედვით.

moviesDB მონაცემთა ბაზაში მხოლოდ ერთი კოლექცია გვაქვს - movies, თუმცა, ცხადია, რეალურ სისტემაში ამ კოლექციის გარდა შესაძლოა, გვქონოდა music, games, users, actors … სხვადასხვა კოლექციები. ამ კოლექციაში თავმოყრილია მთელი ის ინფორმაცია, რაც ფილმის შესახებ ინახება.

ექსპერიმენტული ნაწილი: MongoDB ბაზის ასაგებად განიხილება ფილმების (moviesDB) ბაზა, რომელშიც მხოლოდ ერთი კოლექციაა - movies [16,58]. ამ კოლექციაში თავმოყრილია მთელი ის ინფორმაცია, რაც ფილმის გარშემო ინახება:

- "_id" - უნიკალური იდენტიფიკატორი (ჩადგმული დოკუმენტით)
- "title" - ფილმის სათაური
- "year" - ფილმის გამოშვების წელი
- "rated" - ფილმის შეფასება

- "released" - საიტზე ფილმის დამატების თარიღი (ჩადგმული დოკუმენტით)
- "runtime" - ფილმის ხანგრძლივობა წუთებში
- "countries" - მწარმოებელი ქვეყნების სია
- "genres" - ფილმის ჟანრების სია
- "director" - ფილმის რეჟისორი
- "writers" - სცენარისტები
- "actors" - მსახიობების სია
- "plot" - ფილმის მოკლე შინაარსი
- "poster" - ფილმის პოსტერის ბმული
- "imdb" - ჩადგმული დოკუმენტი IMDB კინოკრიტიკოსების რეიტინგის შესანახად
- "tomato" - ჩადგმული დოკუმენტი Rotten Tomatoes მაყურებლების რეიტინგის შესანახად
- "metacritic" – Metacritic რეიტინგის შესანახად
- "awards" - ჩადგმული დოკუმენტი ჯილდოების შესანახად
- "type" - სრულმეტრაჟიანი/მოკლემეტრაჟიანი/სერიალი...
- "reviews" - ჩადგმული დოკუმენტების მასივი საიტის მომხმარებლების კომენტარების შესანახად.

```
use movieDB
db.movies.insert([
{
        "_id": {
                "oid": "5692a15524de1e0ce2dfcfa3"
        },
        "title": "Toy Story 4",
        "year": 2011,
        "rated": "G",
        "released": {
```

```
            "date": "2010-06-18T04:00:00.000Z"
        },
        "runtime": 206,
        "countries": [
            "USA"
        ],
        "genres": [
            "Animation",
            "Adventure",
            "Comedy"
        ],
        "director": "Lee Unkrich",
        "writers": [
            "John Lasseter",
            "Andrew Stanton",
            "Lee Unkrich",
            "Michael Arndt"
        ],
        "actors": [
            "Tom Hanks",
            "Tim Allen",
            "Joan Cusack",
            "Ned Beatty"
        ],
        "plot": "ყველა ბავშვს სჯერა, რომ როდესაც ის თავის სათამაშოებს მარტო ტოვებს, ისინი ცოცხლდება და თავის საქმეებს აკეთებს. ეს ანიმაციური ფილმი მათ თავიანთ აზრებში კიდევ ერთხელ დაარწმუნებს.",
        "poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
```

```
"imdb": {
        "id": "tt0435761",
        "rating": 8.4,
        "votes": 500084
},
"tomato": {
        "meter": 99,
        "image": "certified",
        "rating": 8.9,
        "reviews": 287,
        "fresh": 283,
        "consensus": "Deftly blending comedy, adventure, and honest emotion, Toy Story 3 is a rare second sequel that really works.",
        "userMeter": 89,
        "userRating": 4.3,
        "userReviews": 602138
},
"metacritic": 92,
"awards": {
      "wins": 56,
      "nominations": 86,
   "text": "Won 2 Oscars. Another 56 wins \u0026 86 nominations."
},
"type": "movie",
"reviews": [
        {
                "date": {
                        "date": "2017-02-13T04:00:00.000Z"
                },
                "name": "parvesh",
```

```
                "rating": 8.9,
                "comment": "My first review for Toy Story 3, hoping it will execute while trying for the very first time."
                },
                {
                        "date": {
                                "date": "2017-02-13T04:00:00.000Z"
                        },
                        "name": "Prabhash",
                        "rating": 8.9,
                        "comment": "My first review for Toy Story 3, hoping it will execute while trying for the very first time."
                },
                {
                        "date": {
                                "date": "2017-02-11T04:00:00.000Z"
                        },
                        "name": "praveen",
                        "rating": 6.7,
                        "comment": "My first review for Toy Story 3, hoping it will execute while trying for the very first time."
                }
        ]
},{
        "_id": {
                "oid": "589cbda9c0b9fec62febf274"
        },
        "title": "Deadpool",
        "year": 2016,
        "rated": "R",
```

```
"released": {
        "date": "2016-06-18T04:00:00.000Z"
},
"runtime": 108,
"countries": [
        "USA"
],
"genres": [
        "Comics character",
        "Adventure",
        "Action"
],
"director": "Tim Miller",
"writers": [
        "Rhett Reese",
        "Paul Wernick"
],
"actors": [
        "Ryan Reynolds",
        "Morena Baccarin",
        "Ed Skrein",
        "T.J. Miller",
        "Gina Carano",
        "Leslie Uggams",
        "Stefan Kapičić",
        "Brianna Hildebrand"
],
"plot": "ყოფილი სპეცდანიშნულების რაზმის წევრი მონაწილეობას იღებს ექსპერიმენტში, რომელიც მას
```

შესაძლებლობას აძლევს მიიღოს წარმოუდგენელი ძალა, სისწრაფე და უნარი სწრაფი განკურნებისა. ",

"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
"imdb": {
"id": "tt1431045",
"rating": 8.1,
"votes": 585141
},
"tomato": {
"meter": 99,
"image": "certified",
"rating": 6.9,
"reviews": 287,
"fresh": 241,
"consensus": "Fast, funny, and gleefully profane, the fourth-wall-busting Deadpool.",
"userMeter": 90,
"userRating": 4.3,
"userReviews": 181719
},
"metacritic": 92,
"awards": {
"wins": 5,
"nominations": 12,
"text": "wo Golden Globe Award nominations for Best Motion Picture – Musical or Comedy and Best Actor – Motion Picture Musical or Comedy."
},
"type": "movie"

```
},{
        "_id": {
                "oid": "589cc22cc0b9fec62febf275"
        },
        "title": "BATMAN V SUPERMAN: DAWN OF JUSTICE",
        "year": 2016,
        "rated": "PG-13",
        "released": {
                "date": "2016-03-19T04:00:00.000Z"
        },
        "runtime": 151,
        "countries": [
                "USA"
        ],
        "genres": [
                "Action",
                "Adventure",
                "Sci-Fi"
        ],
        "director": "Lee Unkrich",
        "writers": [
                "Chris Terrio",
                "David S. Goyer"
        ],
        "actors": [
                "Amy Adams",
                "Henry Cavill",
                "Ben Affleck"
        ],
```

"plot": "იმის შიშით, რომ მეტროპოლისის სუპერგმირის ქმედებები ისევ დარჩება უკონტროლოდ, გოთემ-სითის რაინდი გამოუცხადებს სუპერმენს ომს. სანამ ისინი არჩევენ საქმეებს და აყენებენ კაცობრიობას არჩევანის წინაშე: თუ რომელი სუპერგმირი უფრო სჭირდებათ მათ, ჩნდება ახალი, გაცილებით საშიში საფრთხე",

"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",

"imdb": {

"id": "tt2975590",

"rating": 6.7,

"votes": 3206

},

"tomato": {

"meter": 27,

"image": "certified",

"rating": 4.9,

"reviews": 353,

"fresh": 97,

"consensus": "Batman v Superman: Dawn of Justice smothers a potentially powerful story -- and some of Americas most iconic superheroes -- in a grim whirlwind of effects-driven action.",

"userMeter": 64,

"userRating": 3.6,

"userReviews": 225954

},

"metacritic": 44,

"awards": {

"wins": 6,

"nominations": 26,

```
                "text": "Actor of the Year, Most Original Poster, Best Body of Work"
        },
        "type": "movie"
},{
        "_id": {
                "oid": "589cc417c0b9fec62febf276"
        },
        "title": "doctor strange",
        "year": 2016,
        "rated": "PG-13",
        "released": {
                "date": "2016-11-04T04:00:00.000Z"
        },
        "runtime": 115,
        "countries": [
                "USA"
        ],
        "genres": [
                "Sci-Fi",
                "Fantasy",
                "Adventure",
                "Action"
        ],
        "director": "Scott Derrickson",
        "writers": [
                "Jon Spaihts",
                "Scott Derrickson"
        ],
        "actors": [
```

```
            "Benedict Cumberbatch",
            "Chiwetel Ejiofor",
            "Rachel McAdams"
        ],
        "plot": "მას შემდეგ, რაც მისი კარიერა განადგურდა, ბრწყინვალე მაგრამ ქედმაღალი ექიმი ახალ გამოწვევას იღებს ცხოვრებაში, როდესაც ჯადოქარი მას მფარველობაში აიყვანს და წვრთნის, რათა სამყარო დაიცვას ბოროტებისაგან. ",
        "poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
        "imdb": {
            "id": "tt1211837",
            "rating": 7.8,
            "votes": 191181
        },
        "tomato": {
            "meter": 27,
            "image": "certified",
            "rating": 4.9,
            "reviews": 353,
            "fresh": 97,
            "consensus": "Batman v Superman: Dawn of Justice smothers a potentially powerful story -- and some of Americas most iconic superheroes -- in a grim whirlwind of effects-driven action.",
            "userMeter": 64,
            "userRating": 3.6,
            "userReviews": 225954
        },
        "metacritic": 44,
        "awards": {
```

```
                "wins": 6,
                "nominations": 38,
                "text": "Oscar, Best Visual Effects"
        },
        "type": "movie"
},{
        "_id": {
                "oid": "589cc696c0b9fec62febf277"
        },
        "title": "kung fu panda 3",
        "year": 2016,
        "rated": "PG",
        "released": {
                "date": "2016-01-29T04:00:00.000Z"
        },
        "runtime": 95,
        "countries": [
                "USA"
        ],
        "genres": [
                " Animation",
                "Action",
                "Adventure",
                "Comedy",
                "Family"
        ],
        "director": " Alessandro Carloni",
        "writers": [
                " Jonathan Aibel",
                "Glenn Berger"
```

],
"actors": [
" Jack Black",
"Dustin Hoffman",
"Bryan Cranston"
],
"plot": ”კუნგ ფუ პანდა 3” ისევ დიდ თავგადასავალს გვპირდება. ამჯერად პო და მისი ისევ გამოჩენილი ღვიძლი მამა ლი მიემგზავრებიან საიდუმლო პანდების ადგილსამყოფელში, სადაც ისინი ბევრ წინააღმდეგობას აწყდებიან. მათ მთავარ საშიშროებას ცბიერი და ზებუნებრივით დაჯილდოვებული კაი წარმოადგენს. პანდა პოს რთული მისია აკისრია – მან სხვა პანდებს საბრძოლო ხელოვნება უნდა ასწავლოს. არც თუ ისე მარტივი დავალება აქვთ ფუმფულა პანდებს",
"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
"imdb": {
"id": "tt2267968",
"rating": 7.2,
"votes": 83809
},
"tomato": {
"meter": 87,
"image": "certified",
"rating": 6.8,
"reviews": 153,
"fresh": 133,
"consensus": "Kung Fu Panda 3 boasts the requisite visual splendor, but like its rotund protagonist, this sequels narrative is also surprisingly nimble, adding up to animated fun for the whole family.",

```
                "userMeter": 79,
                "userRating": 3.9,
                "userReviews": 98794
        },
        "metacritic": 44,
        "awards": {
                "wins": 0,
                "nominations": 6,
                "text": "Best Animated Feature, Most Wanted Pet"
        },
        "type": "movie"
},{
        "_id": {
                "oid": "589cc846c0b9fec62febf278"
        },
        "title": "zootopia",
        "year": 2016,
        "rated": "PG",
        "released": {
                "date": "2016-04-04T04:00:00.000Z"
        },
        "runtime": 108,
        "countries": [
                "USA"
        ],
        "genres": [
                "Animation",
                "Adventure",
                "Comedy",
                "Crime",
```

```
            "Family",
            "Mystery"
        ],
        "director": "Byron Howard",
        "writers": [
            "Byron Howard",
            "Rich Moore"
        ],
        "actors": [
            "Ginnifer Goodwin",
            "Jason Bateman",
            "Idris Elba"
        ],
        "plot": " მოგესალმებით ცხოველთა ქალაქში - თანამედროვე ქალაქი, დასახლებული სხვადასხვა ცხოველებით, დიდი სპილოები პატარა თაგვები. ქალაქი იყოფა რაიონებად და აქ ასევე არის ელიტარული უბანი საჰარის ტერიტორიაზე და ასევე არასტუმართმოყვარე ტუნდრათაუნი. ქალაქში არის ახალი პოლიციელი, მხიარული კურდღელი ჯუდი ჰოპსი, რომელიც სამუშაოს პირველი დღიდანვე აცნობიერებს თუ რა ძნელია იყო პატარა და ფუმფულა დიდ და ძლიერ პოლიციელებს შორის. ჯუდი ცდილობს პირველი შესაძლებლობისთანავე წარმოაჩინოს საკუთარი თავი იმის მიუხედავად, რომ მისი მეწყვილე არის მელია ნიკ უაიდლი. მათ ერთად მოუწევთ რთული საქმის გამოძიება, რის გამოაშკარავებამაც შეიძლება საფრთხე შეუქმნას მთელი ქალაქის მაცხოვრებლებს. ",
        "poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
        "imdb": {
            "id": "tt2948356",
```

```
                "rating": 8.1,
                "votes": 262258
        },
        "tomato": {
                "meter": 98,
                "image": "certified",
                "rating": 8.1,
                "reviews": 241,
                "fresh": 236,
                "consensus": "Kung Fu Panda 3 boasts the requisite visual splendor, but like its rotund protagonist, this sequels narrative is also surprisingly nimble, adding up to animated fun for the whole family.",
                "userMeter": 92,
                "userRating": 4.4,
                "userReviews": 95658
        },
        "metacritic": 44,
        "awards": {
                "wins": 26,
                "nominations": 52,
                "text": "Best Animated Feature Film of the Year, Best Motion Picture - Animated"
        },
        "type": "movie"
},{
        "_id": {
                "oid": "589d733a296ba85b1bc3bee6"
        },
        "title": "John Carter",
        "year": 2012,
```

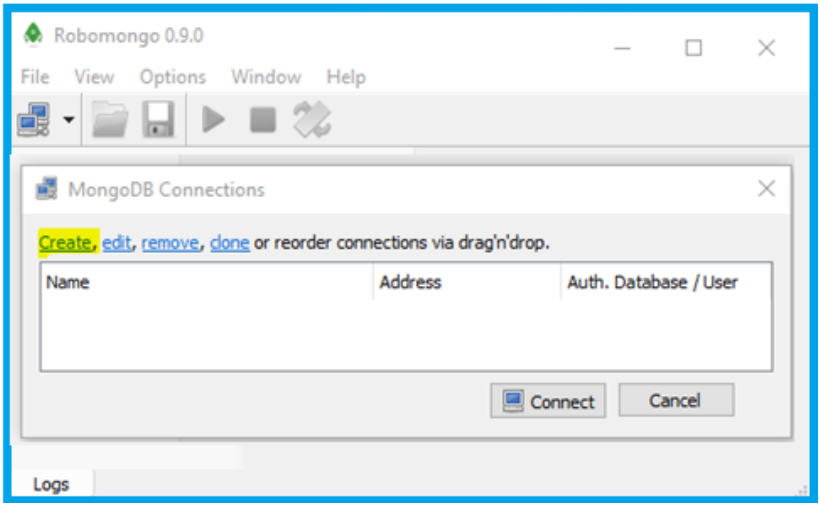
"rated": "PG-13",
"released": {
"date": "2012-03-09T04:00:00.000Z"
},
"runtime": 132,
"countries": [
"USA"
],
"genres": [
"Action",
"Adventure",
"Sci-Fi"
],
"director": "Andrew Stanton",
"writers": [
"John Lasseter",
"Andrew Stanton",
"Lee Unkrich",
"Michael Arndt"
],
"actors": [
"Andrew Stanton",
"Mark Andrews"
],
"plot": "ამერიკის სამოქალაქო ომის ვეტერანი ჯონ კარტერი, მარსზე აღმოჩნდება, სადაც ტყვედ ხვდება მტრულად განწყობილ ოთხმეტრიან აბორიგენებთან. კარტერი არამარტო თავად უნდა გადარჩეს, არამედ პრინცესა დეა ტორისიც უნდა გადაარჩინოს. ",
"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",

```
        "imdb": {
                "id": "tt0401729",
                "rating": 6.6,
                "votes": 217518
        },
        "tomato": {
                "meter": 51,
                "image": "certified",
                "rating": 5.7,
                "reviews": 219,
                "fresh": 111,
                "consensus": "While John Carter looks terrific and delivers its share of pulpy thrills, it also suffers from uneven pacing and occasionally incomprehensible plotting and characterization.",
                "userMeter": 60,
                "userRating": 3.5,
                "userReviews": 113966
        },
        "metacritic": 92,
        "awards": {
                "wins": 2,
                "nominations": 7,
                "text": "Top Box Office Films, Best Original Score for a Fantasy/Science Fiction/Horror Film"
        },
        "type": "movie"
}
])
```

5.2. RoboMongo პლატფორმა

RoboMongo პლატფორმა არის ათზე მეტი ალტერნატიული ინსტრუმენტიდან ერთ-ერთი ყველაზე გავრცელებული GUI (Graphical User Interface) გრაფიკული ინტერფეისი MongoDB მონაცემთა ბაზასთან სამუშაოდ [59].

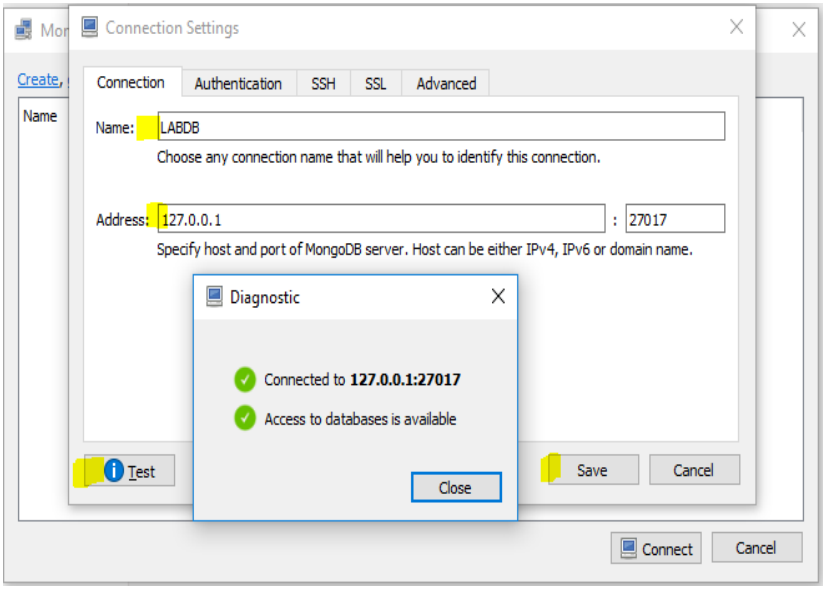
RoboMongo პლატფორმა მოხერხებულობითა და სიმარტივით გამოირჩევა. თავდაპირველად ვამატებთ ახალ კავშირს MongoDB სისტემასთან (ნახ. 5.1).



ნახ. 5.1

დიალოგის ფანჯარაში (ნახ. 5.2) შევავსებთ საჭირო ველებს:

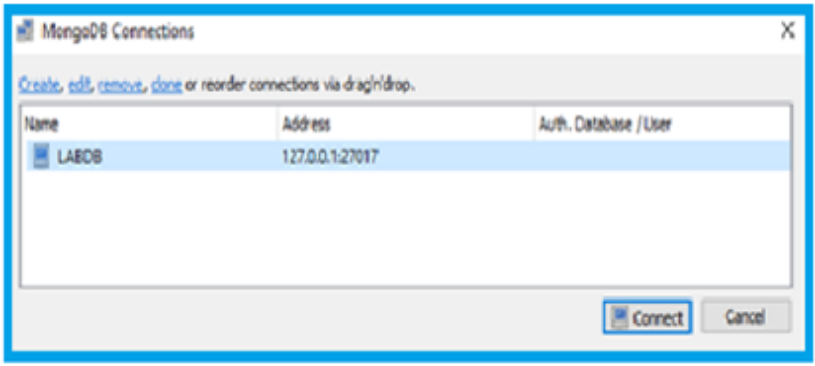
- Name - კავშირის სახელი (შეგვიძლია ნებისმიერი სახელის დარქმევა);
- Address - სერვერის IP მისამართი და პორტი, რომელზეც ელოდება კლიენტებისგან მოსულ მოთხოვნებს. ჩვენ შემთხვევაში ჩავწერეთ 127.0.0.1 - რადგან სერვერი ჩვენივე კომპიუტერზე გვაქვს დაყენებული. პორტი კი უცვლელად დავტოვეთ (default – 27017).



ნახ. 5.2

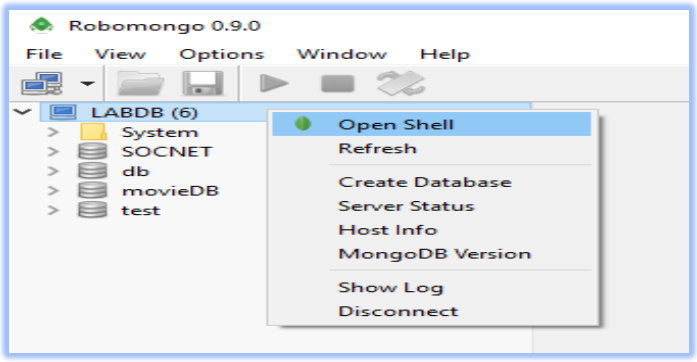
კავშირის შექმნამდე (Save) შეგვიძლია შევამოწმოთ მისი ვალიდურობა Test ღილაკზე დაჭერით.

შემდეგ, 5.3 ნახაზზე ვხსნით ახლად შექმნილ კავშირს.



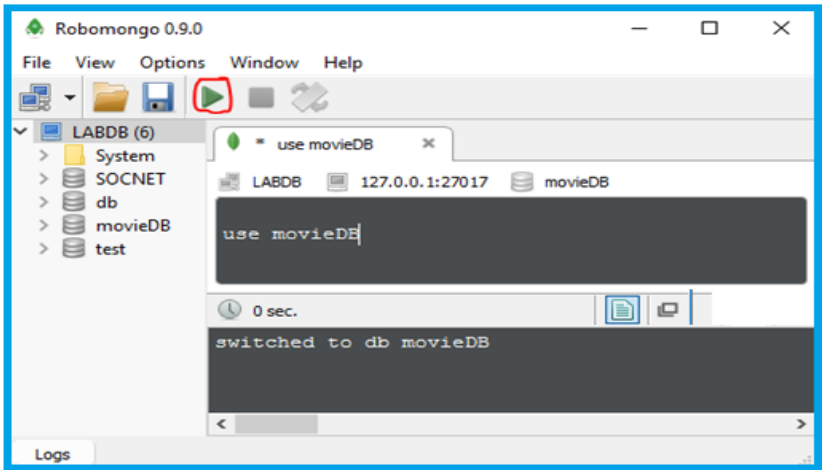
ნახ. 5.3

5.4 ნახაზზე ჩანს უკვე დაკავშირებულ სერვერთან ბრძანებების რეჟიმში გადასვლის მეთოდი.



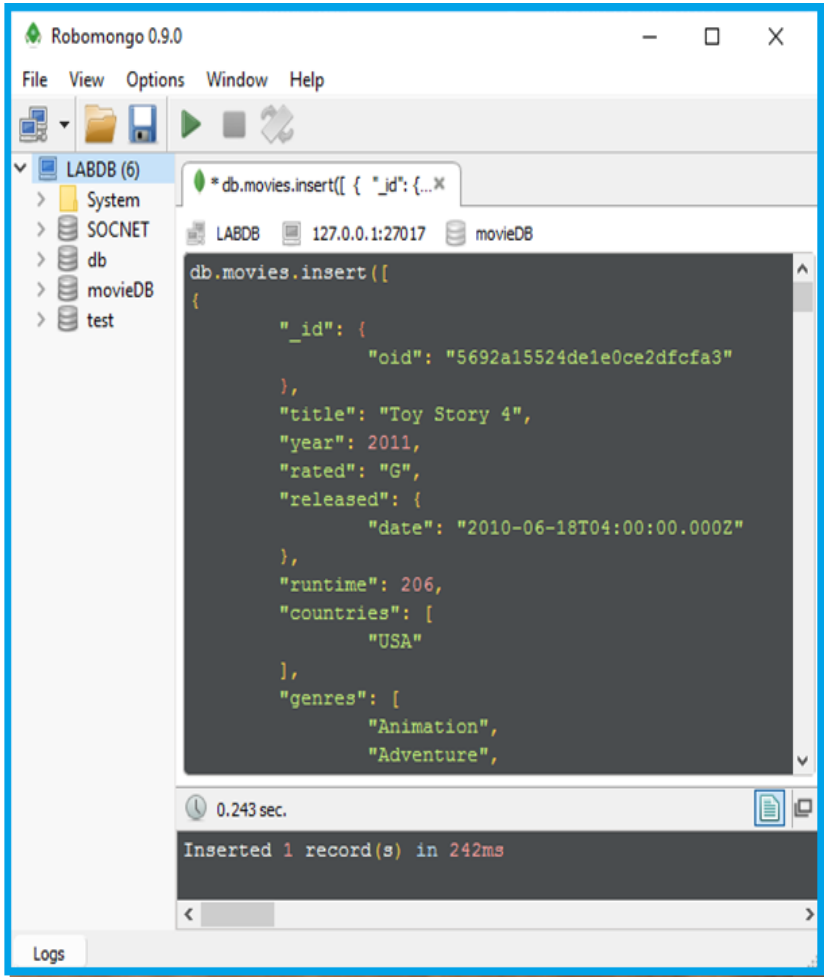
ნახ. 5.4

შემდეგ 5.5 ნახაზზე მოცემულია პირველი ბრძანების - ბაზასთან დაკავშირების მაგალითი



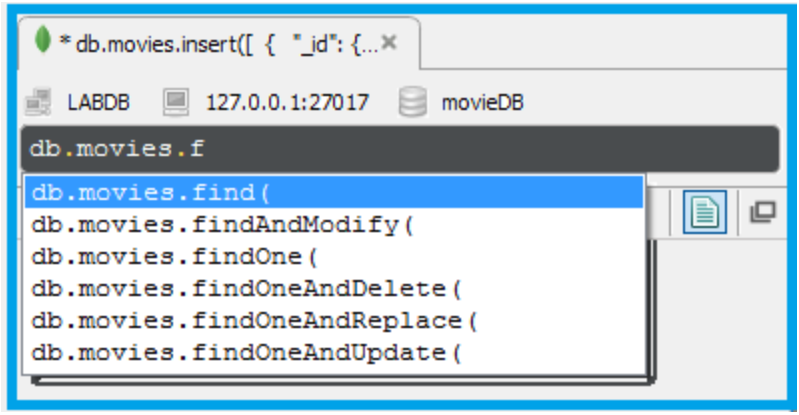
ნახ. 5.5

შემდეგ ეტაპზე დავამატოთ 1-ელ დანართში ნაჩვენები მონაცემები movies კოლექციაში (ნახ.5.6).



ნახ. 5.6

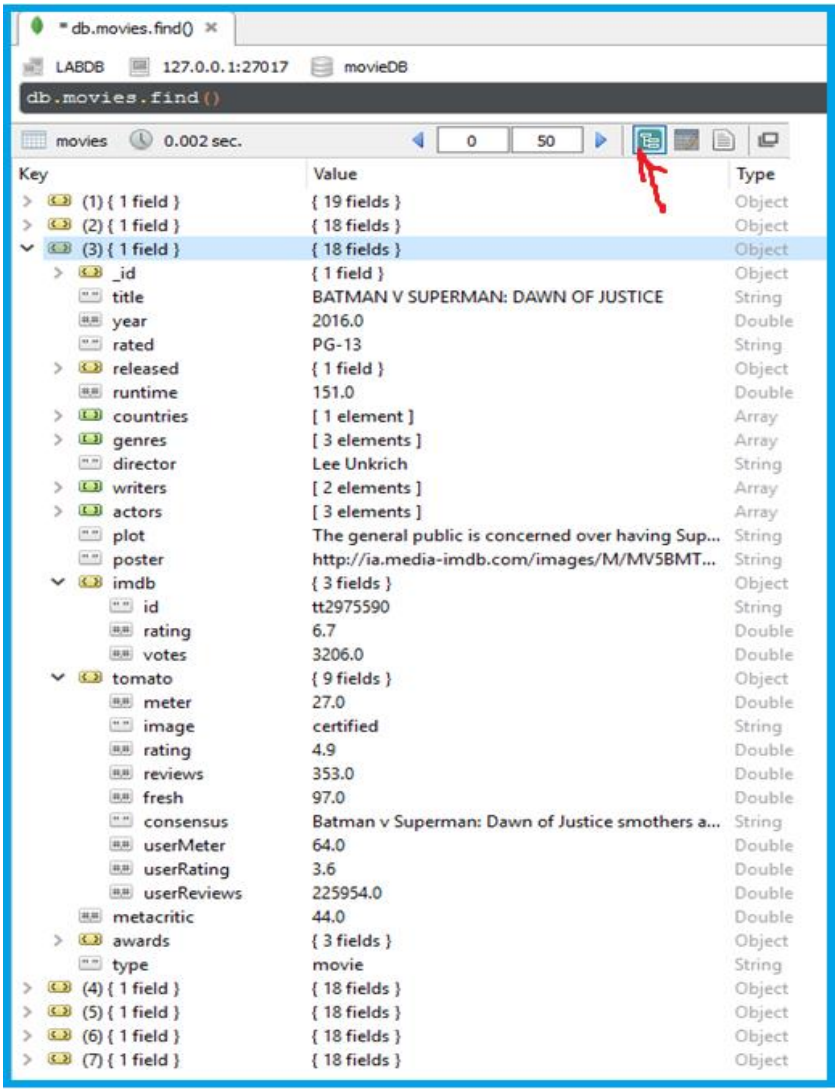
შემდეგ, 5.7 ნახაზზე ჩანს RoboMongo გარემოს intellisense შესაძლებლობა - შემოგვთავაზოს შესაძლო ვარიანტები კოდის წერის დროს.



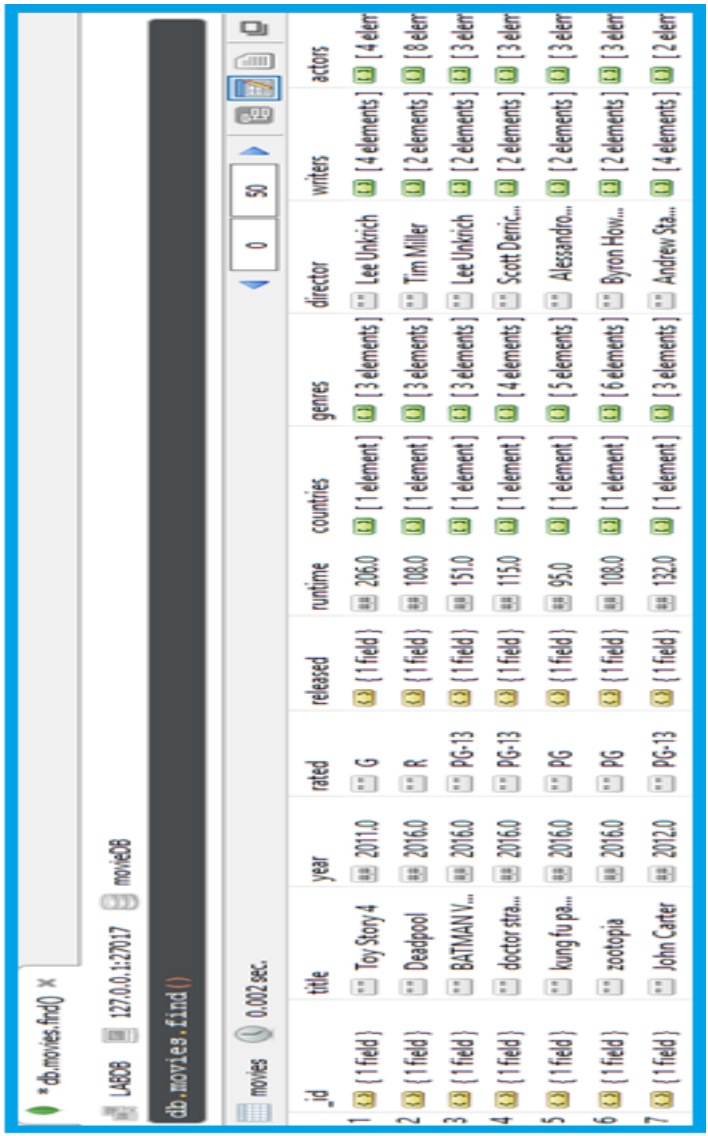
ნახ. 5.7

db.movies.find() ბრძანება გამოიტანს მიმდინარე ბაზის (movieDB) movies კოლექციის ყველა ჩანაწერს. შედეგის გამოტანა შესაძლებელია სამი სხვადასხვა ფორმატით:

- View results in tree mode - ხისებრი სტრუქტურა (ნახ. 5.8);
- View results in table mode - ცხრილის სტრუქტურა (ნახ. 5.9);
- View results in text mode – JSON სტრუქტურა (ნახ. 5.10).



ნახ.5.8. ხისებრი სტრუქტურა



ნახ. 5.9. ცხრილის სტრუქტურა

ნახ. 5.10. JSON სტრუქტურა

5.3. მანიპულაციური მაგალითები MongoDB ბაზაში

5.3.1. ინფორმაციის ამოღების მაგალითები

მას შემდეგ, რაც წარმატებით შევქმენით და შევავსეთ movieDB მონაცემთა ბაზა დანართში მოცემული საცდელი მონაცემებით, მოვიყვანოთ სხვადასხვა ოპერაციების მაგალითები ამ სისტემის გამოყენებით.

- მიმდინარე ბაზის არჩევა
  use movieDB
- ძიება ფილმის სათაურის მიხედვით
  db.movies.find({"title":'Deadpool'})
- ძიება ფილმის გამოშვების თარიღის მიხედვით
  db.movies.find({ year: { $gt: 2010 } })
  db.movies.find({ year: { $gt: 2010, $lt: 2012 } })

სადაც $gt და $lt შედარების ოპერატორებია (Greater Then, Less, Then)

- ფილმის ძიება IMDB რეიტინგის მიხედვით (ამ მაგალითში ჩანს, როგორ უნდა მივწვდეთ ჩადგმული დოკუმენტის ველებს)

```
db.movies.find(
 {
        "imdb.rating": {$gt:8}
 }
)
```

- იმ ფილმების ძიება სადაც ფავორიტი მსახიობებიდან ერთ-ერთი მაინც თამაშობს

```
db.movies.find(
  {   actors: { $in: [ "Tom Hanks", "Gina Carano" ] }
  }
)
```

- ძიება უნიკალური ნომრის მიხედვით

```
db.movies.find(
 {
   "_id.oid" : "5692a15524de1e0ce2dfcfa3"
 }
)
```

- ძიება ჩადგმული დოკუმენტების მასივში:

შემდეგი ფრაგმენტი დაგვიბრუნებს ყველა იმ ფილმს, რომელზეც დატოვებულია კონკრეტული მომხმარებლის კომენტარი

```
db.movies.find(
  {
    reviews: {
          $elemMatch: {
             name: "parvesh"
          }
    }
  }
)
```

- სორტირება

```
db.movies.find().sort( { year: 1 } )
```

- რაოდენობის შეზღუდვა

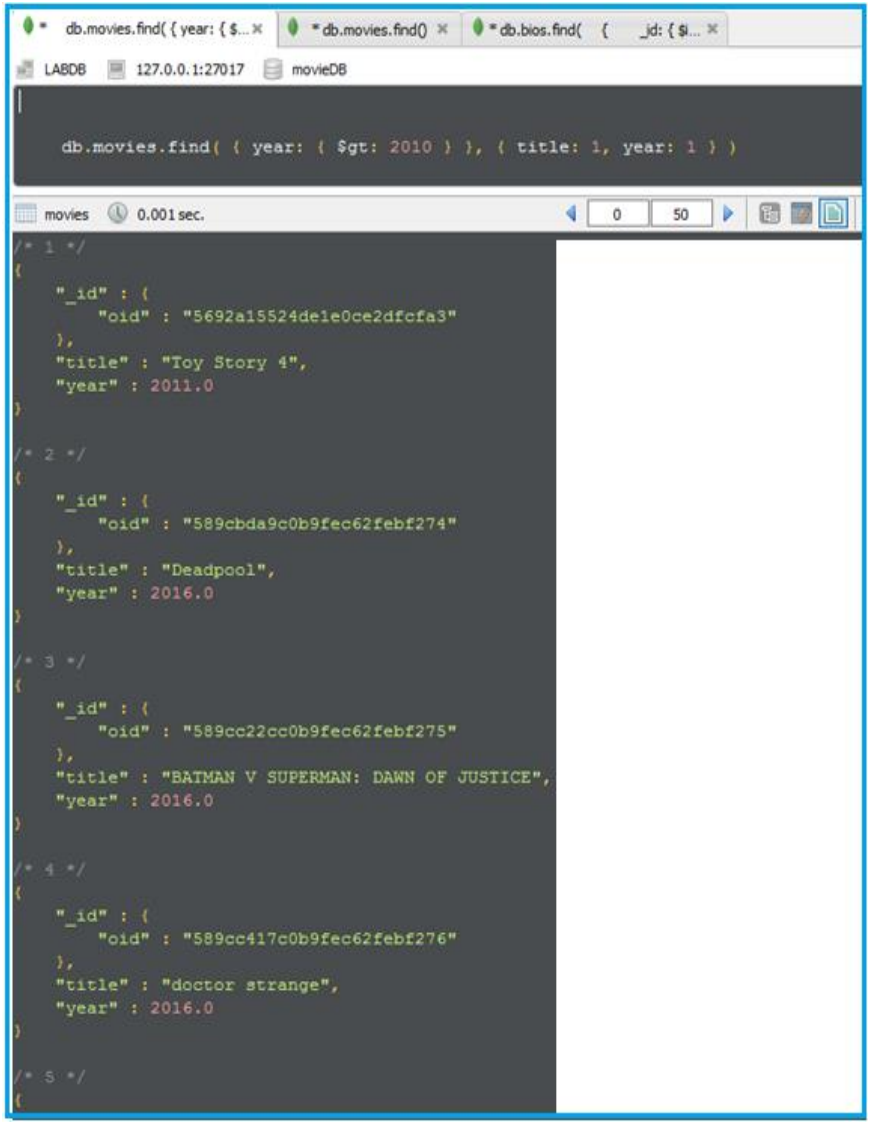
```
db.movies.find().sort( { year: 1 } ).limit(1)
```

ამ ბრძანებით შეგვიძლია ამოვიღოთ ინფორმაცია ყველაზე ძველ ფილმზე

- სასურველი ველების პროექცია

```
db.movies.find( { year: { $gt: 2010 } }, { title: 1, year: 1 } )
                                { title: 1, year: 1 }
```

ამ სახის ფორმატით შევძლებთ მხოლოდ სასურველი ველების დაბრუნებას (ნახ.5.11)


db.movies.find({ year: { $...
db.movies.find()
db.bios.find({ _id: { $i...
LABDB
127.0.0.1:27017
movieDB
db.movies.find({ year: { $gt: 2010 } }, { title: 1, year: 1 })
movies
0.001 sec.
0
50
/* 1 */
{
"_id" : {
"oid" : "5692a15524de1e0ce2dfcfa3"
},
"title" : "Toy Story 4",
"year" : 2011.0
}
/* 2 */
{
"_id" : {
"oid" : "589cbda9c0b9fec62febf274"
},
"title" : "Deadpool",
"year" : 2016.0
}
/* 3 */
{
"_id" : {
"oid" : "589cc22cc0b9fec62febf275"
},
"title" : "BATMAN V SUPERMAN: DAWN OF JUSTICE",
"year" : 2016.0
}
/* 4 */
{
"_id" : {
"oid" : "589cc417c0b9fec62febf276"
},
"title" : "doctor strange",
"year" : 2016.0
}
/* 5 */
{

ნახ. 5.11

ანალოგიურად, შეგვიძლია ამოვიღოთ მხოლოდ სასურველი ველები ჩადგმული დოკუმენტების ძებნითაც:

```
db.movies.find( { "imdb.rating": {$gt:8}}, { title: 1, "imdb.rating": 1} )
{ title: 0, "imdb.rating": 0}
```

ამ შემთხვევაში მივიღებდით ყველა ველს, ამ ორის გარდა:

```
/* 1 */
{
   "_id" : {
      "oid" : "5692a15524de1e0ce2dfcfa3"
   },
   "title" : "Toy Story 4",
   "imdb" : {
      "rating" : 8.4
   }
}
/* 2 */
{
   "_id" : {
      "oid" : "589cbda9c0b9fec62febf274"
   },
   "title" : "Deadpool",
   "imdb" : {
      "rating" : 8.1
   }
}
/* 3 */
{
   "_id" : {
      "oid" : "589cc846c0b9fec62febf278"
   },
   "title" : "zootopia",
   "imdb" : {
      "rating" : 8.1
   }
}
```

5.3.2. სხვადასხვა ოპერაციების მაგალითები წიგნების კოლექციისათვის

- ერთმანეთში ჩადგმული დოკუმენტები

```
var book = {
        title: 'MongoDB: The Definitive Guide',
        authors: [
            { lastName: 'Chodorow', firstName: 'Kristina' },
            { lastName: 'Dirolf', firstName: 'Michael' }
        ],
        tags: ['NoSQL', 'Database', 'BigData', 'Programming'],
        pages: 195,
        published: 2010
};
```

- **Insert**

```
db.books.save(book);
// primary key '_id'
// generated by client driver
// e.g. 4fba97070f318c1e73763350
book._id;
```

- **Update**

```
db.books.update({title: /Good Parts/},
                        {$inc: {pages: 3}});
db.books.update({title: /in Action/},
            {$set: {publisher: 'Manning'}},
            false, true);
db.books.update({},
      {$addToSet: {tags: 'Programming'}},
```

```
    false, true);
```

- **Delete:**

```
db.books.remove({_id: mybook._id});  // წაშლა კონკრეტული
                                     // იდენტიფიკატორის  მიხედვით
db.books.remove({tags: 'Cooking'});
db.books.remove();            // კოლექციის გასუფთავება
```

- **მარტივი მოთხოვნები**

```
db.books.find();  // შედეგში აისახება ყველა წიგნისგან შემდგარი სია
db.books.count();  // შედეგად დაბრუნდება დოკუმენტების საერთო
                   // რაოდენობა
db.books.find().count();  // შედეგად დაბრუნდება დოკუმენტების
                          // საერთო რაოდენობა
 db.books.findOne({  // უნიკალური იდენტიფიკატორის მიხედვით ძებნა
        _id: ObjectId("4fba97190f318c1e73763353")
});
   // ძიება სხვადასხვა პარამეტრების მიხედვით
db.books.find({ title: 'JavaScript Patterns' });
db.books.find({ title: /^MongoDB/ });
db.books.find({ title: /^MongoDB/,  pages: {$gt: 200} });
// ძიება ჩადგმულ დოკუმენტებიან სტრუქტურაში
db.books.find({
        'authors.lastName': 'Katz'});
db.books.find({
        'authors.lastName':
           {$in: ['Katz', 'McCaw']} });
 db.books.find({
        $or: [
           {'authors.lastName': 'Katz'},
           {'authors.lastName': 'McCaw'}
        ]});
```

```
// პროექცია, სორტირება, ზღვარი
db.books.
        find({/* all */},
        {title: 1, pages: 1}).
        sort({title: 1}).
        limit(4);
```

- **ბრძანებები:**

```
db.runCommand({count: 'books',
        query: {published: 2012}});
db.runCommand({distinct: 'books', key:'tags'});
db.runCommand({group: {
        ns: 'books',
        key: { published: true },
        $reduce: function (obj, prev) {
                prev.pages += obj.pages;
        },
        initial: { pages: 0 }
}});

db.runCommand({ dropDatabase: 1 });
db.runCommand({ getLastError: 1 });
db.runCommand({ serverStatus: 1 });
db.runCommand({ shutdown: 1 });
```

- **Indexes**

```
db.books.ensureIndex({"title": 1}, {unique: true});
db.books.ensureIndex({"authors.lastName": 1});
db.books.ensureIndex({"tags": 1});
db.books.getIndexes();
db.books.dropIndex('title_1');
```

- **Import / Export**

```
mongoexport -d test -c books > mongo.books.txt
mongo test --eval "db.books.remove()"
 mongoimport -d test -c books --file books.txt
mongoexport -d test -c books --jsonArray > books.json
mongoimport -d test -c books --jsonArray < books.json
```

5.4. ოთხი რეპლიკა სეტის აწყობა შარდინგ ტექნოლოგიით

განვიხილოთ რეპლიკა სეტის კომპლექსური მაგალითი [16]. ამოცანის არსი მდგომარეობს ოთხი რეპლიკა სეტის აწყობაში, რომლებიც Sharding ტექნოლოგიით ინაწილებს მონაცემებს. სასწავლო რეჟიმში ყველა სერვისი ერთ კომპიუტერზე იყო გაშვებული და განსხვავებულ პორტებზე მუშაობდა. რეალურ შემთხვევაში Mongo-ს სერვისები იქნება გაშვებული სხვადასხვა სერვერზე [60].

A რეპლიკა სეტი:

```
mkdir  a0    192.168.0.10
mkdir  a1    192.168.0.11
mkdir  a2    192.168.0.12
```

B რეპლიკა სეტი:

```
mkdir  b0    192.168.1.10
mkdir  b1    192.168.1.11
mkdir  b2    192.168.1.12
```

C რეპლიკა სეტი:

```
mkdir  c0    192.168.2.10
mkdir  c1    192.168.2.11
mkdir  c2    192.168.2.12
```

D რეპლიკა სეტი:

```
mkdir  d0    192.168.3.10
```

```
mkdir  d1    192.168.3.11
mkdir  d2    192.168.3.12

// mkdir-ში ვგულისხმობთ შესაბამის სერვერებზე
// რეპლიკა სეტის, საქაღალდეების, datafile-ებისა
// მთლიანად მონაცემთა ბაზისთვის საჭირო დირექტორიების შექმნას

mkdir  cfg0    192.168.9.10    // კონფიგ–სერვერების datafile-ები
mkdir  cfg1    192.168.9.11
mkdir  cfg2    192.168.9.12
# config servers    კონფიგ–სერვერების შექმნა
mongod  --configsvr  --dbpath cfg0  --port 26050  --fork  --logpath log.cfg0
--logappend
mongod  --configsvr  --dbpath cfg1  --port 26051  --fork  --logpath log.cfg1
--logappend
mongod  --configsvr  --dbpath cfg2  --port 26052  --fork  --logpath log.cfg2
--logappend
```

რეპლიკა სეტის მოსაწყობად შესაბამის A,B,C,D[0,1,2] სერვერებზე სათითაოდ ვუშვებთ შესაბამის სერვისებს:

```
# “shard servers” (mongod data servers)
# note: don’t use  small files  such a small oplogsize in production;
# “shard servers” (mongod data servers)
# note: don’t use  small files  such a small oplogsize in production;
mongod --shardsvr --replSet a --dbpath a0 --logpath log.a0 --port 27000 -
-fork --logappend
 --smallfiles --oplogSize 50
mongod --shardsvr --replSet a --dbpath a1 --logpath log.a1 --port 27001 -
-fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet a --dbpath a2 --logpath log.a2 --port 27002 -
-fork --logappend
--smallfiles --oplogSize 50
```

```
mongod --shardsvr --replSet b --dbpath b0 --logpath log.b0 --port 27100 --fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet b --dbpath b1 --logpath log.b1 --port 27101 --fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet b --dbpath b2 --logpath log.b2 --port 27102 --fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet c --dbpath c0 --logpath log.c0 --port 27200 --fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet c --dbpath c1 --logpath log.c1 --port 27201 --fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet c --dbpath c0 --logpath log.c0 --port 27200 --fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet c --dbpath c1 --logpath log.c1 --port 27201 --fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet c --dbpath c2 --logpath log.c2 --port 27202 --fork --logappend
 --smallfiles --oplogSize 50
mongod --shardsvr --replSet d --dbpath d0 --logpath log.d0 --port 27300 --fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet d --dbpath d1 --logpath log.d1 --port 27301 --fork --logappend
--smallfiles --oplogSize 50
mongod --shardsvr --replSet d --dbpath d2 --logpath log.d2 --port 27302 --fork --logappend
--smallfiles --oplogSize 50
```

პარამეტრები --smallfiles და --oplogSize 50 მხოლოდ სასწავლო/სატესტო რეჟიმისთვის არის რეკომენდებული:

```
# mongos processes Default port 27017
mongos --configdb mongo.db:26050,mongo.db:2651,mongo.db:26052 --fork --logappend --logpath
log.mongos0
mongos --configdb mongo.db:26050,mongo.db:2651,mongo.db:26052 --fork --logappend --logpath
log.mongos1 --port 26061
mongos --configdb mongo.db:26050,mongo.db:2651,mongo.db:26052 --fork --logappend --logpath
log.mongos2 --port 26062
mongos --configdb mongo.db:26050,mongo.db:2651,mongo.db:26052 --fork --logappend --logpath
log.mongos3 --port 26063
```

MongoDB–ს აქვს მრავალსერვერიანი სისტემებისთვის რეპლიკა სეტისა და შარდინგის სკრიპტების დაგენერირების შესაძლებლობაც.

რეპლიკა სეტი:

```
mongo --nodb
var rst = new ReplSetTest({name:"testSet",
nodes:{node1:{smallfiles:"",oplogsize:40,nopreallоc:null},node2:{smallfiles
:"",oplogSize:40,nopreallo
c:null}, arb:{smallfiles:"",oplogSize:40, nopreallоc:null}}});
rst.startSet();
```

შარდინგი:

```
mongo --nodb

> config = { d0 : { smallfiles : "", nopreallоc : "", nopreallocj : ""}, d1 : {
smallfiles : "", nopreallоc : "",
nopreallocj : "" }, d2 : { smallfiles : "", nopreallоc : "", nopreallocj : ""}};
> cluster = new ShardingTest( { shards : config } );
```

ზემოთ მოყვანილი სკრიპტები ერთმანეთთან აკავშირებს სერვერებს.

რეალურად, რეპლიკა სეტს კი შემდეგი ბრძანებებით ვააქტიურებთ და ვამოწმებთ:

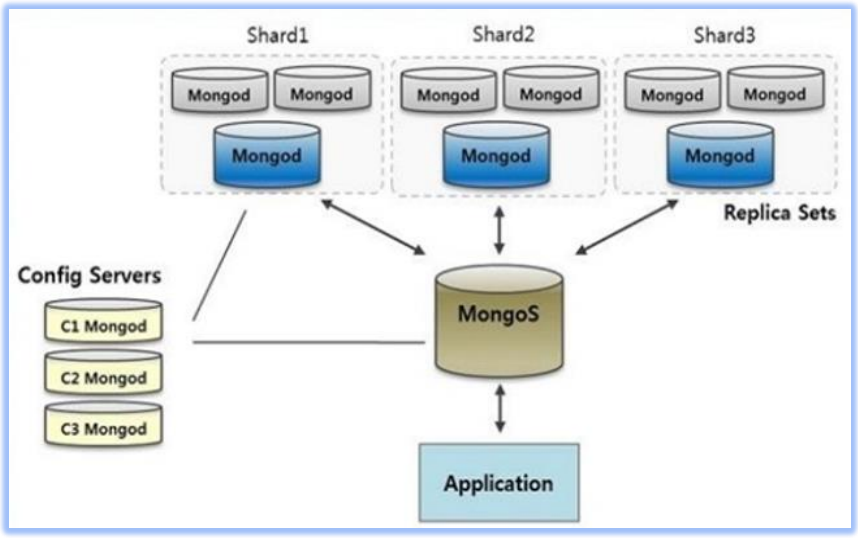
```
mongo --port 27000
rs.status()
rs.initiate()
rs.status()
rs.add("Mongo.db:27001")
rs.add("Mongo.db:27002")
rs.conf()
connect mongos
mongo
sh.addshard("a/Mongo.db:27000")
sh.status()
```

- **ინფორმაციის მიღება განაწილებული გარემოდან:**

განვიხილოთ მაგალითი, თუ როგორ მოქმედებს Sharding-ი MongoDB-ში: მომხმარებელი ითხოვს კონკრეტულ ინფორმაციას.

პირველ რიგში, მოთხოვნა შემოვა Mongos მოთხოვნების გამანაწილებელ და მთავარ კონტროლერ სერვისთან, რომელიც კონფიგურაციის სერვერიდან მიიღებს ინფორმაციას, თუ რომელ Shard-ს უნდა მიმართოს ამ კონკრეტული მოთხოვნის შესასრულებლად და გადასცემს მოთხოვნას შესაბამის Shard-ს, მისგან მიღბულ ინფორმაციას კი დაუბრუნებს კლიენტს (ნახ.5.12). ამ ოპერაციის შესასრულებლად სხვა Shard-ების რესურსი არ გამოიყენება.

შესაბამისად, ვიღებთ ოპერაციების გადანაწილებას ჰორიზონტალურად, რაც, თავის მხრივ, უზრუნველყოფს სწრაფქმედებას.


Shard1
Shard2
Shard3
Mongod
Mongod
Mongod
Mongod
Mongod
Mongod
Mongod
Mongod
Mongod
Replica Sets
Config Servers
C1 Mongod
C2 Mongod
C3 Mongod
MongoS
Application

ნახ. 5.12

ლიტერატურა:

1. 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.

2. ჩოგოვაძე გ., სურგულაძე გ., ქაჩიბაია ვ. მონაცემთა ბაზების მართვის სისტემები. სტუ. თბილისი. 1988

3. Чоговадзе Г., Сургуладзе Г., Качибая В. Теория реляционных зависимостей и проектирование логической схемы баз данных. Монография. „Мецниереба“, Тбилиси, 1988

4. ჩოგოვაძე გ., სურგულაძე გ., შონია ო. მონაცემთა და ცოდნის ბაზების აგების საფუძლები. სტუ. თბილისი. 1996

5. მეიერ-ვეგენერი კ., სურგულაძე გ., ბასილაძე გ. საინფორმაციო სისტემების აგება მულტიმედიური მონაცემთა ბაზებით. სტუ. თბილისი. 2014

6. Wedekind H., Surguladze G. Technology of Designing of Distributed Systems on the Basis of Objectoriented Programming. ISSN 021-7164, GTU, Tbilissi. 1996. pp.96-100.

7. ქოროღლიშვილი ვ. NoSQL'n CAP. Just Development. Computer sciences. 2014. http://vakhokor.blogspot.com/2014/10/nosql-n-cap.html

8. Гранков М.В., Жуков А.И. Системы управления Базами данных. Донской гос.техн. Университет. Ростов-на-Дону. 2013

9. Кузнецов С. Объектно-ориентированные базы данных - основные концепции, организация и управление: краткий обзор. http://citforum.ru/database/articles/art_24.shtml

10. სურგულაძე გ., შონია ო., ყვავაძე ლ. მონაცემთა განაწილებული ბაზების მართვის სისტემები (MsSQL Server, Access, InterBase, JDBC, Oracle) // სახელმძღ., სტუ. თბ., 2004

11. Lecluse Ch., Richard Ph., Velez F. O2, an Object-Oriented Data Model. Proc.ACM SIGMOD Int.Conf.Manag.Data. Chicago, Ill, USA, June 1-3, 1988, ACM SIGMOD Record.- 17, N 3.- 1988.- 424-433

12. თურქია ე., ჩერქეზიშვილი გ., კილასონია თ., ნეფარიძე მ. საპროცესინგო სისტემებში პრობლემების გადაწყვეტის გზები // სტუ, შრ.კრ. "მას"- N 1(12), 2012, გვ.117–120

13. NoSQL. https://en.wikipedia.org/wiki/NoSQL

14. სურგულაძე გ., ჩერქეზიშვილი გ. საპროცესინგო სისტემებში პრობლემების მართვის ავტომატიზაცია. სტუ-ს 90 წლისადმი მიძღვნილი საერთაშორისო სამეცნ. კონფერენცია შრ.კრ. „21-ე საუკუნის მეცნიერებისა და ტექნოლოგიების ძირითადი პარადიგმები". სტუ, თბილისი, სექტ., 2012. გვ.30-33.

15. ჩოგოვაძე გ., ფრანგიშვილი ა., სურგულაძე გ. მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. სტუ. „ტექნიკური უნივერსიტეტი", თბილისი. 2017. -1001 გვ.

16. სურგულაძე გ., კივილაძე გ. შესავალი NoSQL მონაცემთა ბაზებში (MongoDB). სტუ, „IT კონსალტინგის ცენტრი", თვ., 2017, -152 გვ.

17. MariaDB. https://en.wikipedia.org/wiki/MariaDB

18. InnoDB - Storage Engine. https://en.wikipedia.org/wiki/InnoDB

19. Nemeth E., Snyder G., Hein T.R., Whaley B. UNIX and LINUX System Administration Handbook fourth edition. Prentice Hall. 2010

20. NoSQL For Dummies®. Published by: John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com Copyright © 2015, New Jersey

21. Document-Oriented Database. https://en.wikipedia.org/wiki/-Document-oriented_database

22. Document-oriented Database. Clusterpoint. Retrieved on 2015-10-08. https://www.clusterpoint.com/

23. სურგულაძე გ., ჰ.ვედეკინდი, ნ.თოფურია. განაწილებული ოფის-სისტემების მონაცემთა ბაზების დაპროექტება/რეალიზაცია UML-ტექნოლოგიით. სტუ. თბ., 2006, -237 გვ..

24. MongoDB manual – http://docs.mongodb.org/manual/

25. Stonebraker M. New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps. June 2011. http://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext

26. Choudhary S. NewSQL: The Best of Both "OldSQL" and "NoSQL"http://www.slideshare.net/sushantbchoudhary/newsql-the-best-o. 2014

27. Glushkov I. NewSQL overview. http://www.slideshare.net/-IvanGlushkov/newsql-overview. 2015.

28. Morgan A., Lord M. NoSQL with MySQL 2014. http://www.dr-dobbs.com/database/nosql-with-mysql/240167115

29. The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things - http://bit.ly/digital_universe.

30. Miller R. Facebook Builds Exabyte Data Centers for Cold Storage. http://bit.ly/facebook_exabyte

31. Ancestry.com. Company Facts. www.ancestry.com/corporate/-about-ancestry/company-facts

32. https://www.mongodb.com/compare/mongodb-mysql

33. სურგულაძე გ., კივილაძე გ., კახელი ბ. NoSQL მონაცემთა ბაზები: განვითარების პერსპექტივები და პრობლემები მართვის საინფორმაციო სისტემებში. სტუ, შრ.კრ. „მას“, N 2(22). გვ.230-239.

34. მიდოდაშვილი პ., შესავალი: მონაცემთა ბაზები და მათი მართვის სისტემები; http://www.pacoge.net/Physics/Sesavali_monacemTa%20bazebi%20da%20maTi%20marTvis%20sistemebi.pdf

35. Tom Kyte, Oracle Technology Network, http://www.oracle.-com/technetwork/issue-archive/2010/10-jan/o65asktom-082389.html

36. IBM Knowledge Center, http://www.ibm.com/support/knowledgecenter/SSEPGG_10.1.0/com.ibm.db2.luw.admin.perf.doc/doc/c0004121.html

37. Dave Voorhis.. Codd's Twelve Rules. Posted on September 17, 2015. https://computing.derby.ac.uk/c/codds-twelve-rules/

38. Rahm E., Kaufmann M. Verteiltes und paralleles datenmanagement: von verteilten datenbanken zu Big Data und Cloud. 2015. German

39. https://blog.codecentric.de/2013/08/einfuhrung-in-hadoop-die-wichtigsten-komponenten-von-hadoop-teil-3-von-5/

40. Jablonski St., Architektur von Workflow-ManagementSystemen // Informatik forschung und Entwicklung, Germany, 1997.

41. http://www.oracle.com/webfolder/technetwork/de/community/dbadmin/tipps/olap_cubes_mit_awm/index.html#DB

42. http://www.oracle.com/webfolder/technetwork/de/community/faktenblatt/development.pdf

43. http://oracleolap.blogspot.com/2007/12/olap-workshop-2-understanding-olap.html

44. Schweiggert F. Anwendungen des Data Mining in der Praxis. Universität Ulm, Germany 2015

45. http://wirtschaftslexikon.gabler.de/Definition/datamining.html

46. Zehn algorithm in data mining. Technische Universität Ilmenau. Germany. 2016

47. Runkler Th.A. Data Mining: Methoden und Algorithmen intelligenter Datenanalyse. Germany. 2014

48. https://entwickler.de/online/datenbanken/data-mining-typische -verfahren-und-praxisbeispiele-115010.html

49. https://en.wikipedia.org/wiki/Pig_(programming_tool)

50. https://en.wikipedia.org/wiki/Apache_Hive

51. https://hortonworks.com/hadoop-tutorial/how-to-use-hcatalog-basic-pig-hive-commands/

52. https://en.wikipedia.org/wiki/Apache_HBase

53. http://www.waytoeasylearn.com/2016/04/apache-zookeeper-tutorial.html

54. https://cwiki.apache.org/confluence/display/AMBARI/Ambari

55. http://sqoop.apache.org/

56. https://www.cloudera.com/documentation/other/tutorial/CDH5/topics/ht_flume_to_hdfs.html

57. https://en.wikipedia.org/wiki/Apache_Mahout

58. Parvesh Tandon, RestFul API using node.js and mongodb. https://github.com/mistertandon/node-express-hbs

59. AlternativeTo – Crowdsourced software recommendations http://alternativeto.net/software/robomongo/

60. სურგულაძე გ., კივილაძე გ., მაღალი წვდომადობის მქონე მონაცემთა საცავის დაპროექტება არარელაციური მონაცემთა ბაზების გამოყენებით. https://www.slideshare.net/Giorgiiviladze/nosql-50798944

დანართი N1

Oracle-ს ობიექტების ტესტირება

განვიხილოთ Oracle მონაცემთა ბაზის პროცედურებისა და ტრიგერების ფუნქციონირების მაგალითები ბაზის ობიექტების ტესტირების მიზნით.

1) პროცედურა-ტრიგერის ერთობლივი მუშაობის ტესტირება. განიხილება მაგალითი, როდესაც ტრიგერი ინფორმაციას გადაიტანს product_check მონაცემთა ბაზის ცხრილში ინფორმაციის update შესრულების შემდეგ:

```
CREATE or REPLACE TRIGGER After_Update_Stat_product
AFTER
DELETEON product
BEGIN
INSERTINTO product_check
Values('After update', sysdate);
End;
```

2) მონაცემთა ბაზაში ტრიგერის არსებობის შესახებ ინფორმაცია შეიძლება მივიღოთ შემდეგი სკრიპტით:

```
SELECT * FROM user_triggers WHERE trigger_name =
      'Before_Update_Stat_product';
```

3) ქვემოთ მოცემული ტრიგერი გვიჩვენებს თუ როდის მოხდა მონაცემთა ბაზაში `job_id` ველის ცვლილება. ტრიგერი აინსერტებს მონაცემებს პროცედურაში add_job_history, ხოლო პროცედურა კი `job_history` ცხრილში:

```
CREATE OR REPLACE TRIGGER update_job_history
AFTERUPDATEOF job_id ON employees
FOREACHROW
BEGIN
  add_job_history(:old.employee_id,
:old.hire_date, sysdate,
:old.job_id, :old.department_id);
END;
```

4) ქვემოთ მოცემულია პროცედურა, რომელშიც ტრიგერი ამუშავებს insert -ს:

```
CREATE OR REPLACE PROCEDURE add_job_history
  (  p_emp_id          job_history.employee_id%type
   , p_start_date      job_history.start_date%type
   , p_end_date        job_history.end_date%type
   , p_job_id          job_history.job_id%type
   , p_department_id
                  job_history.department_id%type
   )
IS
BEGIN
  INSERT INTO job_history (employee_id,
     start_date, end_date, job_id, department_id)
  VALUES(p_emp_id, p_start_date, p_end_date,
                     p_job_id, p_department_id);
END add_job_history;
```

დ1 ნახაზზე ნაჩვენებია ცხრილი, რომელსაც მოცემული ტრიგერი აკონტრილებს: update_job_history, ხოლო დ2 ნახაზზე ცხრილი, სადაც ტრიგერიდან შემოსულ ინფორმაციას იყენებს პროცედურა insert-ისთვის.

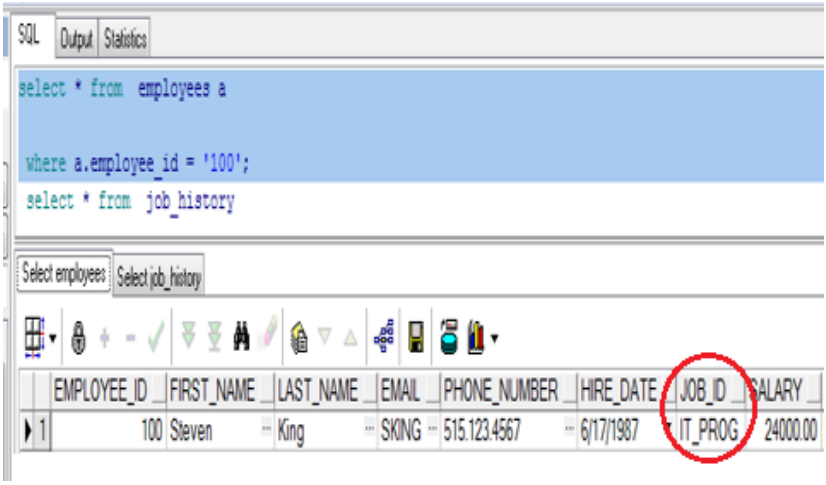
ნახ. დ1. Oracle-ს ობიექტების ტესტირება (ტრიგერისთვის)

ნახ. დ2. Oracle-ს ობიექტების ტესტირება (პროცედურისთვის)

5) ჩავატაროთ ტესტირება ტრიგერის და პროცედურის ფუნქციონირების შესამოწმებლად. update-ს გამოყენებით შევცვალოთ **job_id** ველის მნიშვნელობა თანამშრომლისთვის, რომელსაც **employee_id** = 100.

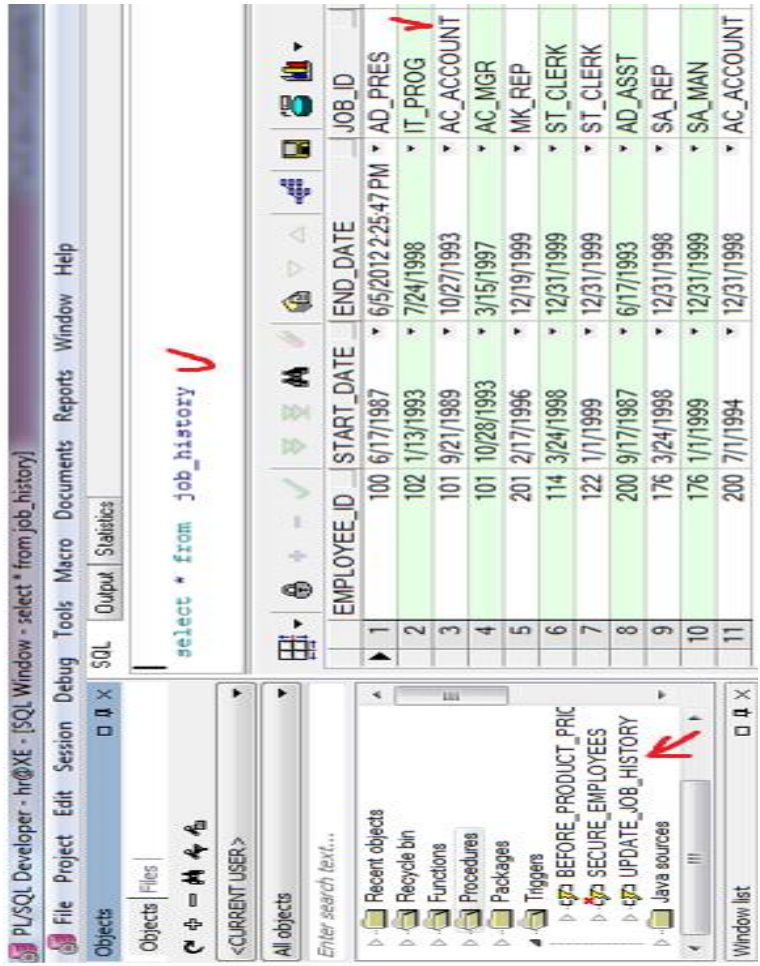
```
update employees a
set a.job_id = 'Manager'
where a.employee_id = '100';
```

როგორც დ3 ნახაზიდან ჩანს მდგომარეობა employees ცხრილში შეიცვალა და job_id ამჟამად გახდა „IT_PROG“



ნახ. დ3.

ახლა გადავამოწმოთ job_history ცხრილი (ნახ. დ4).



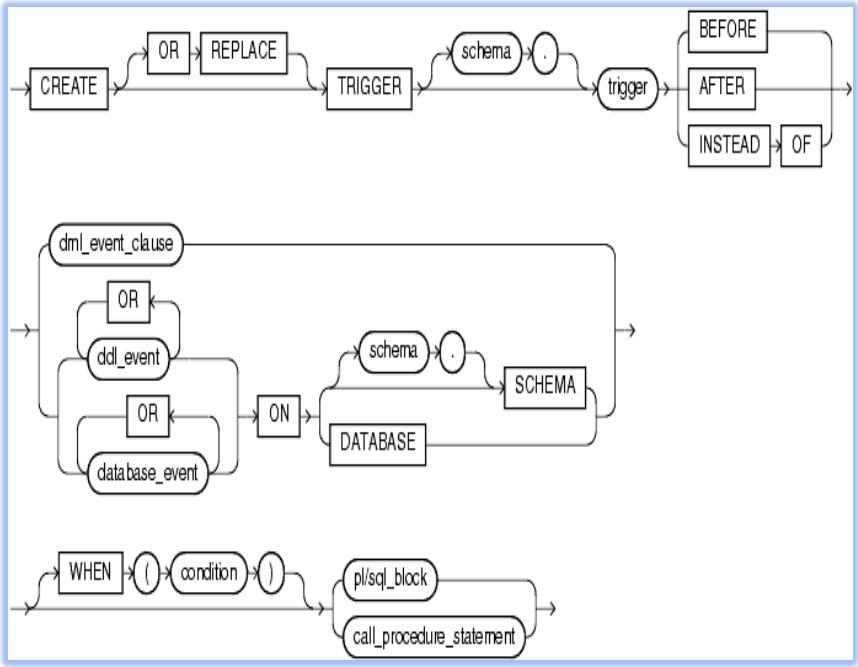
ნახ. დ4. Oracle-ს ობიექტების ტესტირება
job_History ცხრილი (ტრიგერისთვის)

ცხრილში ველის ძველი მნიშვნელობა აქაც შეიცვალა. ამგვარად, პროცედურამაც სწორად იმუშავა და ტრიგერმაც.

დანართი N2

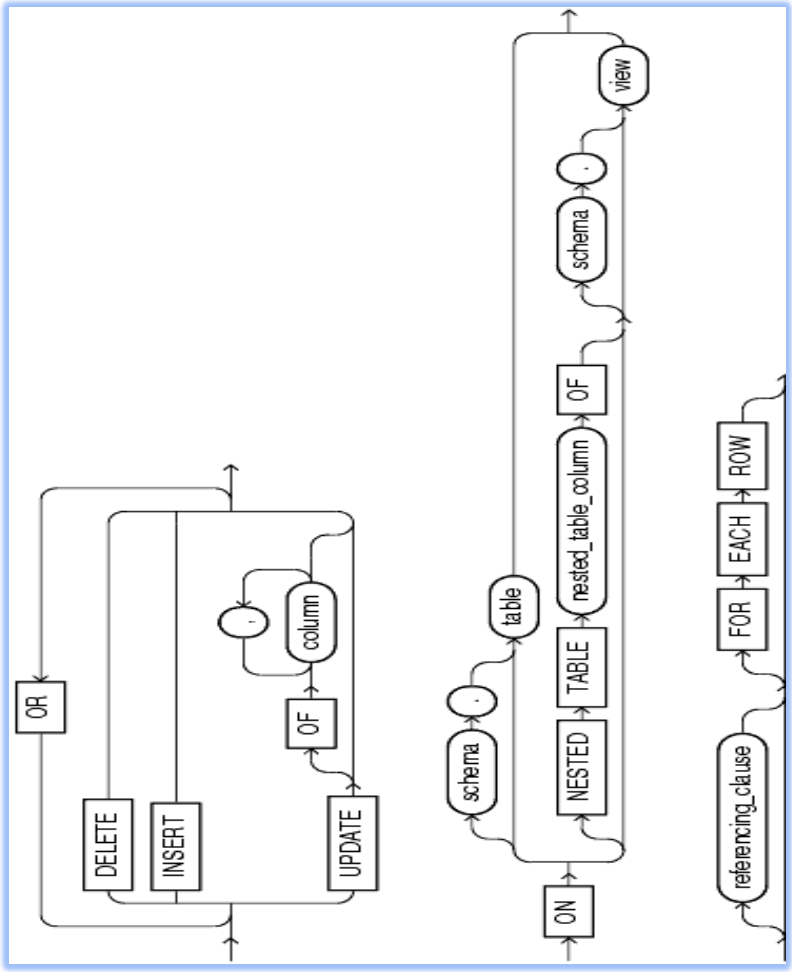
Oracle ბაზაში ტრიგერების შექმნის სქემა

create_trigger::=



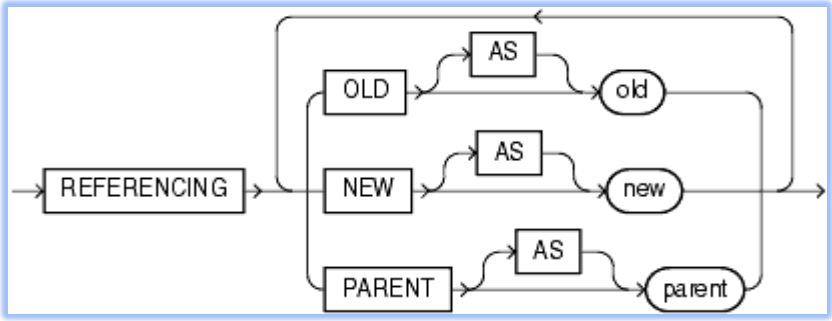
ნახ. დ5-1) ტრიგერის შექმნის სქემა

DML_-ის პუნქტი



ნახ. დ5-2) ტრიგერის შექმნის სქემა

referencing პუნქტი ::=



ნახ. დ5-3) ტრიგერის შექმნის სქემა

გადაეცა წარმოებას 20.03.2017 წ. ხელმოწერილია დასაბეჭდად 30.03.2017 წ. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 9. ტირაჟი 100 ეგზ.

სტუ-ს „IT კონსალტინგის ცენტრი“

(თბილისი, მ.კოსტავას 77)