სამაგისტრო ნაშრომი
თბილისი, 2019

დიდი მონაცემები, მათი დამუშავების და ანალიზის სისწრაფეზე ორიენტირებული მეთოდები SQL vs NoSQL

ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

ირაკლი არევაძე

ინფორმაციული ტექნოლოგიები

ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი

ხელმძღვანელი

პროფ. მაია არჩვაძე

სარჩევი

ანოტაცია 2
შესავალი 2
რელაციური მონაცემთა ბაზები 3
ძირითადი თეორემები, რომელსაც ემყარება რელაციური მოდელი ACID 6
CAP თეორემა, SQL და NoSQL სისტემები 7
არარელაციური მონაცემთა ბაზები 11
ძირითადი დებულებები, რომელთაც ემყარება არარელაციური მოდელი BASE 17

ანოტაცია

ნაშრომი შეეხება დღესდღეობით ერთ-ერთ ყველაზე პოპულარულ თემას - დიდი მონაცემები, მათი დამუშავება და ანალიზი.

გამომდინარე იქიდან, რომ დღევანდელ მსოფლიოში, საბანკო-საფინანსო, სავაჭრო თუ სოციალურ სფეროებში, უამრავმა ინფორმაციამ მოიყარა თავი და მათი დამუშავება და ანალიზი, რეალურ დროში, გარკვეულ სირთულეებთანაა დაკავშრებული, ჩნდება კითხვა: რა მეთოდები და ტექნოლოგიები არსებობს დიდ მონაცემებთან სამუშაოდ, როგორია მათი სიმძლავრე და წარმადობა, რა ხარჯებთან არის დაკავშრებული ამა თუ იმ ტექნოლოგიის გამოყენება და გვაძლევს თუ არა იგი სასურველ შედეგს.

ამ ნაშრომში მიმოვიხილავთ დღესდღეობით ყველაზე აქტუალურ ტექნოლოგიებს დიდ მონაცემებთან სამუშაოდ. განვიხილავთ თითოეული მათგანის უპირატესობებს და ნაკლოვანებებს, უფრო ზუსტად კი ერთმანეთს შევადარებთ რელაციურ და არარელაციურ მონაცემთა ბაზებს. ვისაუბრებთ დიდ მონაცემებთან მათი მუშაობის დადებით და უარყოფით მხარეებზე. შევადარებთ მათ მოქნილობას და სისწრაფეს, კერძოდ კი აქცენტს გავაკეთებთ რელაციური და არარელაციური მონაცემთა ბაზების დღესდღეობით ყველაზე გამოყენებად წარმომადგენლებზე Microsoft Sql Servers-სა და MongoDB-ზე.

შესავალი

დღეს, თანამედროვე საინფორმაციო ტექნოლოგიების ერაში, მუდმივი განხილვა მიმდინარეობს, იმისა თუ როგორ შეიძლება დავამუშაოთ დიდი მონაცემები ისე, რომ ამას „წლები“ არ დასჭირდეს და ანალიტიკურად დამუშავებული მონაცემები რეალურ დროში მივიღოთ.
გამომდინარე იქიდან, რომ საქმე სხვადასხვა ტიპის მილიონობით ჩანაწერს ეხება, დამუშავების მეთოდებიც განსხვავებულია. მეთოდები ერთმანეთისგან იმის მიხედვით განსხვავდება თუ რა გვინდა მივიღოთ საბოლოოდ. მონაცემების დაცულობა, სტრუქტურის შენარჩუნება და კლასიფიცირებული, მკაცრად განსაზღვრული ინტერფეისი თუ უბრალოდ სისწრაფე და მოქნილობა.

უპირველესად უნდა აღინიშნოს, რომ მონაცემების საცავებად, მათი ორგანიზებისთვის, წაკითხვისა და განახლებისთბვის დღეს ყველაზე გამოყენებადი სისტემები სწორედ მონაცემთა ბაზის მართვის სისტემებია, რომელიც თავის მხრივ ორ კლასად იყოფა: რელაციურ, რომელიც SQL მონაცემთა ბაზების სახელითაა ცნობილი და არარელაციური ე.წ No SQL მონაცემთა ბაზები.

ამ ნაშრომში ჩვენ მიმოვიხილავთ რელაციური და არარელაციური მონაცემთა ბაზების გამოყენების პლიუსებს და მინუსებს დიდ მონაცემებთან.

შევადარებთ ორ ყველაზე გამოყენებად მონაცემთა ბაზის სისტემას (რელაციურ) Microsoft Sql Server-ს და (არარელაციურ) MongoDB-ს რეალურ ისტორიულ მონაცემებზე ტესტირებით. თითოეულ მათგანს შევამოწმებთ სხვადასხვა ტიპის ბრძანებებზე და სხვადასხვა რაოდენობის მონაცემებზე.

რელაციური მონაცემთა ბაზები

რელაციური მონაცემთა ბაზების შექმნას სათავე Edgan F. Codd-მა ჩაუყარა 1970 წელს, როდესაც მან IBM-ში მუშაობის დროს გამოსცა ნაშრომი სახელწოდებით "A Relational model of data for large shared data banks". სწორედ ამ ნაშრომის მიხედვით IBM-მა 1974 წელს შექმნა პირველი რეალური პროდუქტი სახელწოდებით "System R", რომელიც რელაციურ მოდელს ეფუძნებოდა და იყენებდა ენას სახელწოდებით "SEQUEL". ხოლო პირველი კომერციული რელაციური მონაცემთა ბაზა 1979 წელს შეიქმნა კომპანია "Relational Software Inc." ის მიერ და მას სახელად "Oracle" ეწოდა. მოგვიანებით კი ამ კომპანიამ სახელი შეიცვალა და "Oracle Systems Corporation" დაირქვა.

რელაციურ მონაცემთა ბაზებში ინფორმაცია ინახება ცხრილებში, რომლის სტრუქტურა წინასწარ, მკაცრად არის განსაზღვრული იმ სქემის, მიხედვით რომელ სქემაშიც იქმნება ესა თუ ის ცხრილი. აქვე უნდა აღვნიშნოთ, რომ რელაციურ მონაცემთა ბაზებში ცხრილი სქემის გარეშე არ არსებობს. ცხრილის შექმნისას განისაზღვრება მისი ყველა მახასიათებელი, როგორებიცაა სვეტების სახელები და ტიპები, რომელიც თითეული ჩანაწერისთვის ველებს წარმოადგენს, ხოლო სტრიქონი ველების ერთობლიობით ერთ ობიექტს ქმნის. მონაცემთა ტიპები მკაცრად განსაზღვრულია, რაც იმას ნიშნავს რომ ტექსტის ტიპის ველში მხოლოდ ტექსტის შენახვაა შესაძლებელი, თარიღის ტიპის ველში მხოლოდ თარიღის და ა.შ. თითოეულ ცხრილში ჩანაწერს შეიძლება ჰქონდეს ველების მხოლოდ ის რაოდენობა რაც წინასწარ განისაზღვრა სქემაში. ყოველ ცხრილს აქვს თავისი უნიკალური იდენტიფიკატორი, რომელიც ცხრილში ჩანაწერების უნიკალურობას განსაზღვრავს. თითოეული ცხრილი შეიძლება ერთმანეთთან იყოს დაკავშირებული სწორდე ამ იდენტიფიკატორის მეშვეობით. რელაციურ მონაცემთა ბაზებში არსებობს ცხრილებს შორის კავშრების 3 განსხვავებული მეთოდი. თითეული ცხილი კი შესაძლოა ბევრ სხვა ცხრილს გარკვეული კანონზომერებით უკავშრდებოდეს.
დღესდღეობით ყველაზე გავრცელებული რელაციური მონაცემთა ბაზის მართვის სისტემებია: Microsoft Sql Server, MySql, Oracle და ა.შ.

შემდეგ მაგალითში განვიხილავთ, მონაცემების წარმოდგენას რელაციურ მონაცემთა ბაზის სტრუქტურაში. კერძოდ კი არტისტების და ჟანრების ცხრილებს ერთმანეთთან დავაკავშირებთ და კავშირის ცხრილს ალბომს დავარქმევთ, რომელიც არტისტების და ჟანრების ცხრილს "Many to Many"- კავშირით დააკავშირებს, სადაც თითეულ არტისტ (ობიექტს) წარმოადგენს ArtistId , ხოლო თითეულ ჩანაწერს ჟანრებიდან წარმოადგენს GenreId რომელიც თავის მხრივ წარმოადგენს Primary key - ს მთავარ ცხრილებში (Artist და Genre).

Artist

ArtistId	ArtistName
1	AC/DC
2	Slim Dusty
3	The Wiggles
4	…

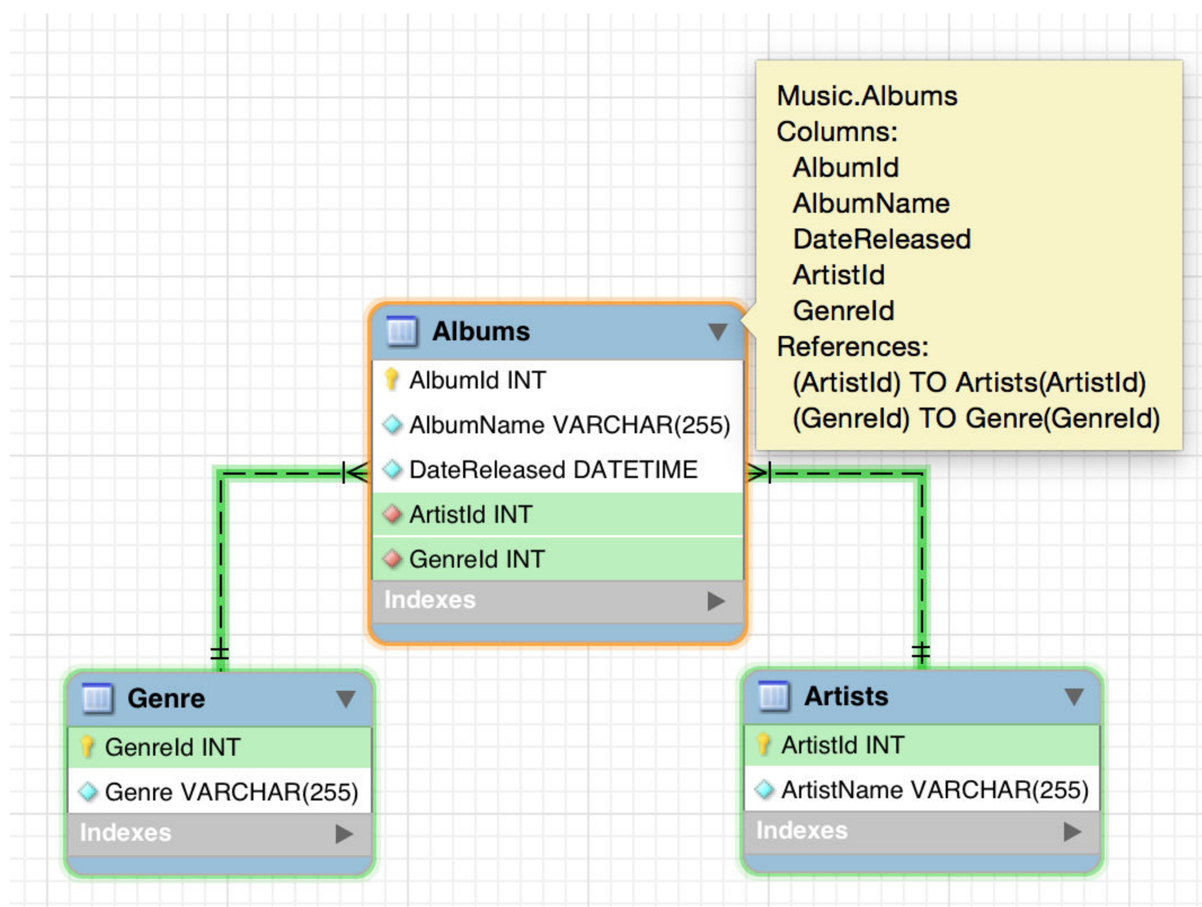
Genre

GenreId	Genre
1	Country
2	Blues
3	Hard Rock
4	…

Album

AlbumId	AlbumName	DateReleased	ArtistId	GenreId
1	Back in Black	1980	1	3
2	Powerage	1978	1	3
3	High Voltage	1976	1	3

Database Diagram Design



ცხრილებს შორის კავშირების გრაფიკული გამოსახულება

ძირითადი თეორემები, რომელსაც ემყარება რელაციური მოდელი ACID

ტექნოლოგიების განვითარებასთან ერთად, გაჩნდა საჭიროება იმისა, რომ მონაცემთა ბაზის მართვის სისტემებს ქონოდათ მექანიზმი, რომელიც ბაზის ტრანზაქციულობას გააკონტროლებდა. გამომდინარე იქიდან, რომ ერთ მონაცემთა ბაზასთან ბევრ იუზერს უწევდა მუშაობა, არ უნდა მომხდარიყო პარარელური პროცესების ერთამეთში არევა, რაც მონაცემების დაზიანებას გამოიწვევდა. 1970 წელს Jim Gray-ს მიერ ჩამოყალიბებული თეორემები გახდა საფუძვლი მექანიზმისა, სახელად “ACID” რომელიც თავის თავში 4 თეორემას აერთიანებს და უზრუნველყოფს რელაციური მონაცემთა ბაზების ტრანზაქციულობას.

აბრევიატურა “ACID” შემდეგნაირად იშიფრება - Atomic, Consistent, Isolated, Durable.

ტრანზაქცია წარმოადგენს ერთი ან რამდენიმე ოპერაციის ერთობლიობას.

- **Atomic**: ამ თვისების თანახმად ბაზაში მხოლოდ მაშინ მოხდება მონაცემების ცვლილება თუ ყველა ოპერაცია წარმატებით შესრულდება. ხოლო იმ შემთხვევაში თუ რომელიმე მათგანი წარუმატებლად დასრულდება ბაზაში მონაცემები უცვლელი დარჩება.

- **Consistent:** თითოეულმა ტრანზაქციამ უნდა გაითვალისწინოს წესები, რომლებიც ბაზაზეა გაწერილი. არცერთ ტრანზაქციას არ შეუძლია იმოქმედოს ბაზის წესების საპირისპიროდ. ყველა ინფორმაცია, რომელიც უნდა ჩაიწეროს მონაცემთა ბაზაში უნდა ეთანხმებოდეს ბაზის წესებს (სქემას, მონაცემთა ტიპებს და ა.შ).

- **Isolated:** თვისება უზრუნველყოფს ბაზაში პარარელურად გაშვებულ და დაუსრულებელ ტრანზაქციებს შორის დიფერენცირებას.
მაგალითად: ერთ დაუსრულებელ ტრანზაქციას არ შეუძლია გავლენა მოახდინოს მეორე დაუსრულებელ ტრანზაქციაზე.

- **Durable:** მას შემდეგ რაც ტრანზაქცია საბოლოოდ შესრულდება და დადასტურდება, მისი ცვლილება შეუძლებელია, რადგან არ მოხდეს სხვა დაუსრულებელ ტრანზაქციებთან კონფლიქტი.

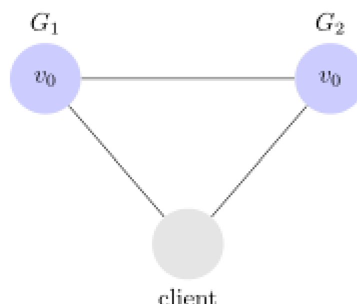
CAP თეორემა, SQL და NoSQL სისტემები

CAP თეორემა იგივე Eric Brewer ის თეორემა, წარმოადგენს საფუძველს ყველა განაწილებადი სისტემისთვის, მათ შორის მონაცემთა ბაზებისთვისაც.
CAP აკრონიმი შემდეგნაირად იშიფრება:
Consistency (თანმიმდევრულობა, მთლიანობა), **Availability** (ხელმისაწვდომობა), **Partition tolerance** (განაწილებადობასთან ადაპტირებულობა).
ეს ის სამი თვისებაა, რომლიდანაც მხოლოდ 2 ის მიღწევაა შესაძლებელი განაწილებად სისტემებში. იმის მიხედვით თუ როგორი სისტემა გვინდა მივიღოთ საბოლოოდ, უნდა ავირჩიოთ ან მთლიანობა და განაწილებადობა CP ან ხელმისაწვდომობა და განაწილებადობა AP. თუ სისტემა მხოლოდ ერთ გარემოზე (ერთ მონაცემთა სერვერზე) მუშაობს მაშინ CAP თეორემა აზრს კარგავს ასეთი სისტემებისთვის.
რელაციური მონაცემთა ბაზები სწორედ ასეთ სისტემებს მიეკუთვნებიან. მონაცემების გადანაწილება სხვადასხვა სერვერებზე, აქ რელაციურ მოდელში, თითქმის შეუძლებელია, ამიტომ მონაცემების ზრდასთან ერთად ერთადერთი რაც შეგვიძლია გავაკეთოთ სერვერის ტექნიკური მახასიათებლების გაუმჯობესებაა. სწორედ ამიტომ რელაციური მოდელი ეფუძნება არა **CAP** არამედ **ACID** თეორემას.
მაგ: ოპერატიული მეხსიერების გაზრდა, უკეთესი პროცესორის ან/და მყარი დისკის ჩადგმა და ა.შ. ხოლო, რაც შეეხება **არარელაციურ** მოდელს, აქ მონაცემების სხვადასხვა სერვერებზე გადანაწილება შესაძლებელია, რაც არარელაციური მონაცემთა ბაზების დისტრიბუციულობას ნიშნავს, შესაბამისად ის ასეთი სისტემები CAP თეორემას ეფუძნებიან.

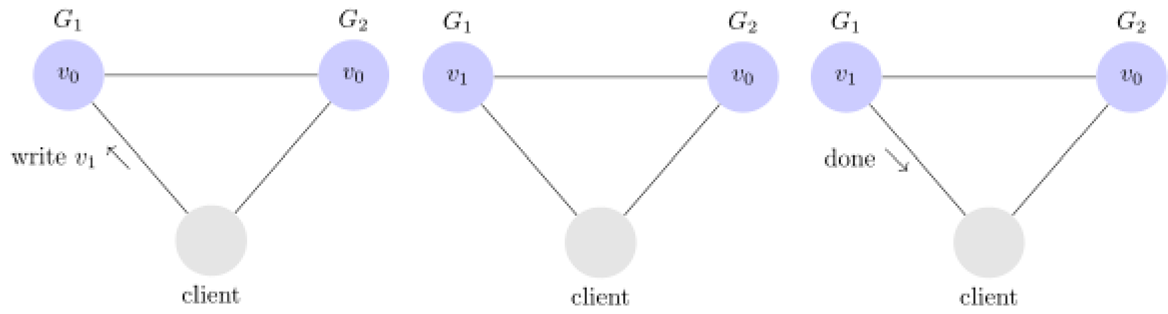
დავუბრუნდეთ Brewer - ის თეორემას და განვსაზღვროთ CAP - აკრონიმის მნიშვნელობა დისტრიბუციულ სისტემებში.
თავდაპირველად, პატარა მაგალითის მეშვეობით განვსაზღვროთ თუ რას ნიშნავს **დისტრიბუციული სისტემა**.

<u>დისტრიბუციული სისტემის მაგალითი:</u>

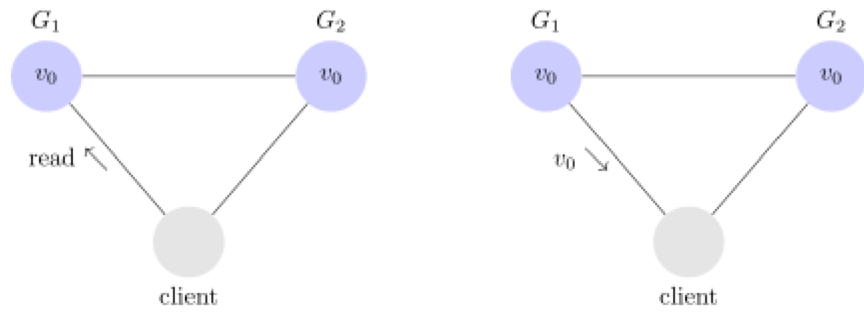
ჩვენი სისტემა შედგება ორი სერვერისგან G1 და G2. ორივე სერვერზე ინახება ერთიდაიგივე მონაცემი - v, რომლის მნიშვნელობაც მიმდინარე მომენტში არის v0, ორივე სერვერზე. G1 და G2 სერვერებს შეუძლიათ ერთმანეთთან კომუნიკაცია, ხოლო კლიენტს შეუძლია ორივე მათგანთან.



კლიენტს შეუძლია განახორციელოს ჩაწერის ან წაკითხვის ოპერაცია ნებისმიერ სერვერზე. ხოლო, როცა სერვერი მიიღებს მოთხოვნას კლიენტისგან მან უნდა დაამუშაოს ის და დაუბრუნოს პასუხი. ასე მაგალითად:



ჩაწერის ოპერაცია - კლიენტმა მოითხოვა ჩაწერის ოპერაცია G1 სერვერზე. მას შემდეგ რაც G1- ზე ჩაიწერა v1, სერვერმა დაუბრუნა შეტყობინება კლიენტს წარმატებული ჩაწერის შესახებ.



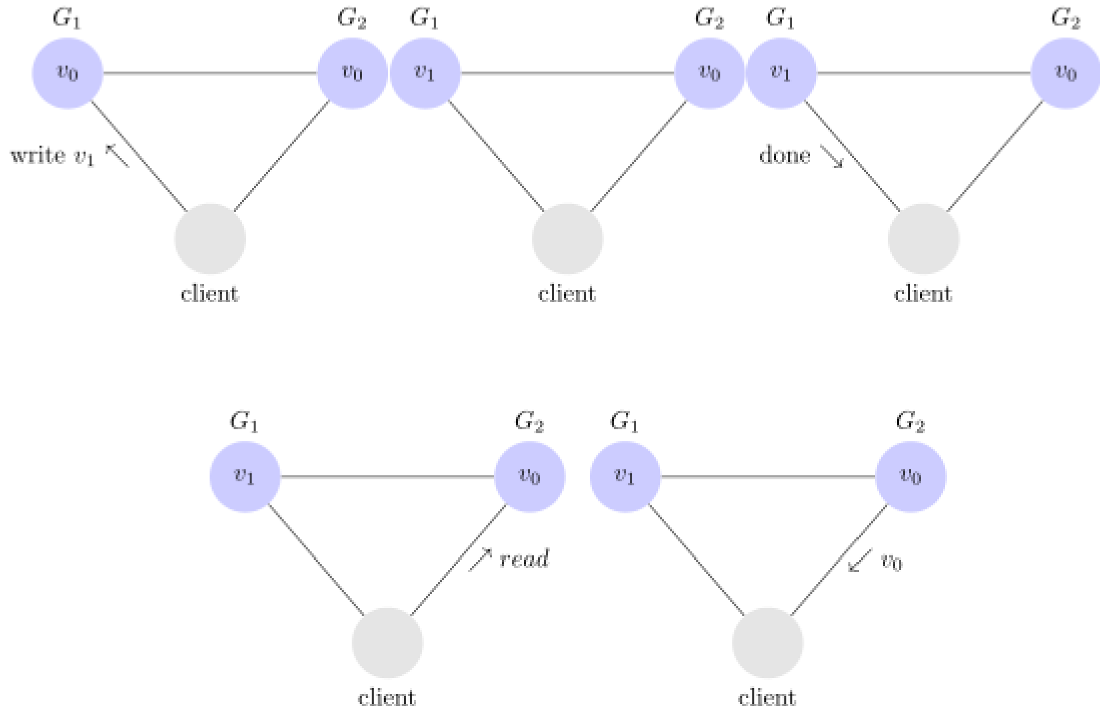
წაკითხვის ოპერაცოია - კლიენტმა წაიკითხა G1 სერვერიდან v ცვლადის მნიშვნელობა, სერვერმა დაუბრუნა v0 – v ცვლადის მიმდინარე მნიშვნელობა G1 სერვერზე.

ახლა კი ზემოთ მოყვანილი დისტრიბუციული სისტემის მაგალითზე განვსაზღვროთ თუ რას ნიშნავს - **Consistency, Availability** და **Partition tolerance** თვისებები CAP თეორემაში.

- **Consistency**: თვისებაა, რომელიც უზრუნველყოფს დაყოფილი სისტემის თანმიმდევრულობას და ერთიანობას.

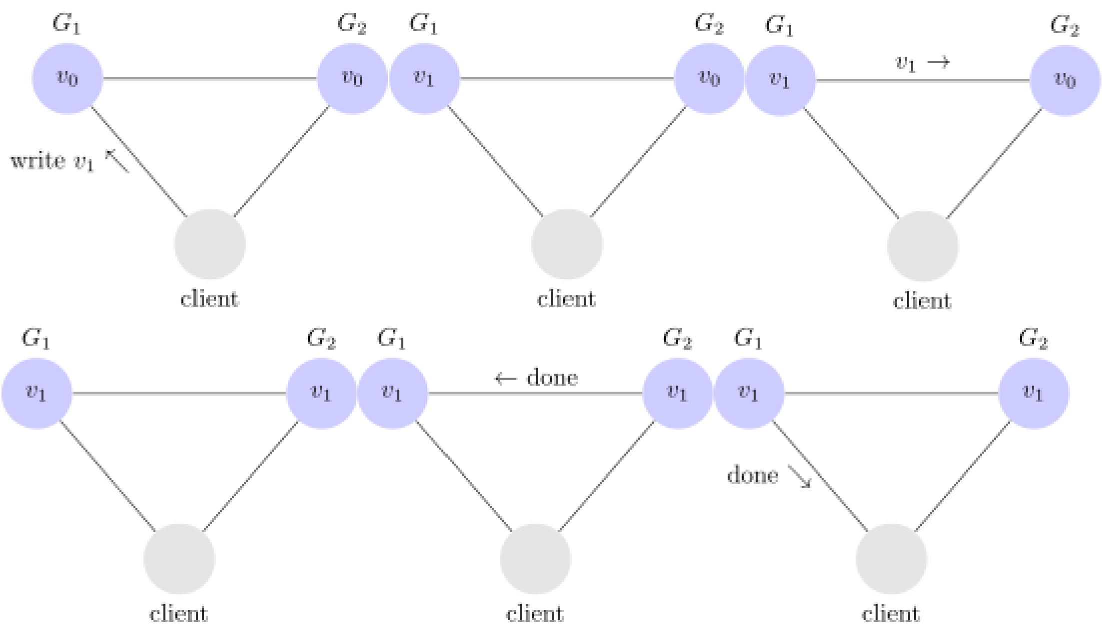
ასეთ სისტემაში მას შემდეგ, რაც კლიენტი შეცვლის/ჩაწერს ახალ მნიშვნელობას v ცვლადში და სერვერიც დადებით პასუხს დაუბრუნებს, კლიენტმა უნდა მიიღოს შეცვლილი მნიშვნელობა ყველა სხვა სერვერიდანაც.

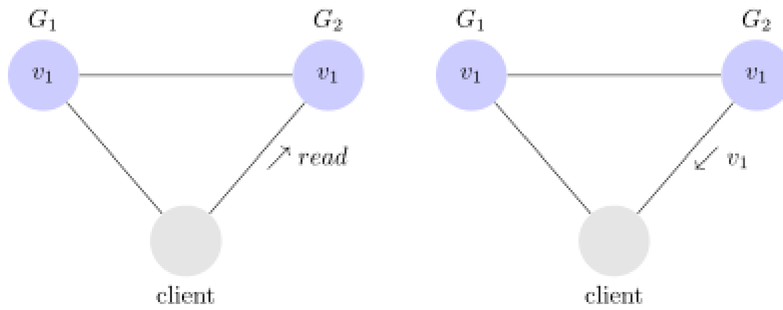
არათანმიმდევრული სისტემის მაგალითი:



კლიენტმა G1 სერვერზე ჩაწერა v ცვლადის ახალი მნიშვნელობა v1, მაგრამ არ მოხდა ცვლადის განახლება G2 სერვერზე. აქედან გამომდინარე, როცა კლიენტმა მოითხოვა v ცვლადის მნიშვნელობა G2 სერვერიდან მან მიიღო ძველი მნიშვნელობა. ასეთ სისტემას არათანმიმდევრული სისტემა ეწოდება.

თანმიმდევრული სისტემის მაგალითი:





ამ შემთხვევაში G1 სერვერი G2 სერვერთან კომუნიკაციის შემდეგ უბრუნებს პასუხს მომხმარებელს. კომუნიკაცია კი ნიშნავს G2 სერვერზე არსებული ცვლადის განახლებას. შესაბამისად, როდესაც მომხმარებელი G2 სერვერიდან წაიკითხავს მონაცემს მას განახლებული მნიშვნელობა დაუბრუნდება. ასეთ სისტემას თანმიმდევრული, ერთიანი სისტემა ეწოდება.

- **Availability:** თვისების თანახმად სისტემა ყოველთვის უნდა პასუხობდეს კლიენტს, მიუხედავად იმისა მოხდება თუ არა რაიმე ტიპის შეცდომა სისტემაში. ჩვენი მაგალითიდან გამომდინარე, თუ კლიენტი მოითხოვს ახალი მონაცემის ჩაწერას რომელიმე „მუშა“ სერვერზე, სერვერი ვალდებულია ყოველთვის დაუბრუნოს პასუხი მომხმარებელს.

- **Partition tolerance:** თვისების თანახმად სისტემამ უნდა იმუშაოს სწორად, მიუხედავად G1 და G2 სერვერებს შორის კავშირის ხარვეზებისა. ხარვეზები შეიძლება იყოს, ინტერნეტ კავშირის არ არსებობა კონკრეტულ მომენტში, მონაცემების გადაცემის შენელება და ა.შ

არარელაციური მონაცემთა ბაზები

ტერმინი „არარელაციური მონაცემთა ბაზა“ პირველად 1998 წელს გაჟღერდა, როცა “Carlo Strozzi”-მ თავისი რელაციური ბაზა გამოუშვა, რომელიც SQL ენას არ იყენებდა. ხოლო უკვე შემდგომ 2009 წელს, მაშინ როცა რამოდენიმე არარელაციური მონაცემთა ბაზა უკვე არსებობდა - Mapreduce, BigTable (Google), Dynamo (Amazon), Hadoop, Casandra და MongoDb; Eric Evans-მა და Johan Oskarsson-მა საბოლოოდ დაუმკვიდრეს სახელი არარელაციურ მონაცემთა ბაზებს და დღეს ისინი NoSQL (Not Only Sql) მონაცემთა ბაზების სახელითაა ცნობილი.

არარელაციური მონაცემთა ბაზების შექმნის საფუძველი გახდა ინტერნეტის აქტუალურობის ზრდა, კერძოდ კი ე.წ არასტრუქტურირებული ვებ ინფორმაციის შენახვის და ანალიზის პრობლემა.

განსხვავებით რელაციური მოდელისგან აქ არარელაციურ მონაცემთა ბაზებში არ არსებობს სქემა. მონაცემები კი შემდეგი ოთხი მეთოდის მიედვით ინახება:

- **Key-Value storage**
- **Document storage**
- **Wide Column storage**
- **Graph database**

- **Key-Value storage:** ასეთი ტიპის მონაცემთა ბაზაში ინახება გასაღებებისა და გასაღებზე „მიმაგრებული“ მონაცემების წყვილები; სადაც გასაღები შესაძლებელია იყოს ნებისმიერი ტიპის, გასაღებები უნიკალურია და არ მეორდება მასზე მიმაგრებული მონაცემები კი შეიძლება იყოს ნებისმიერი ტიპის, მათ შორის გასაღები-მონაცემის ტიპისაც კი, (გარდა გამონაკლისი მართვის სისტემებისა). ასეთი ტიპის სტრუქტურის უპირატესობა სიმარტივეა, თუმცა მისი გამოყენება არაეფექტურია იმ შემთხვევაში თუ მხოლოდ მონაცემების გაფილტვრა, წაკითხვა და განახლება გვინდა, რადგან ასეთი თიპის ბაზებში მონაცემების ფილტრაცია მხოლოდ გასაღებით არის შესაძლებელი.

Key	Value
artist:1:name	AC/DC
artist:1:genre	Hard Rock
artist:2:name	Slim Dusty
artist:2:genre	Country

Key-Value storage სტრუქტურის მაგალითი

- **Document storage:** ამ არქიტექტურის მქონდე მონაცემთა ბაზებში მონაცემები, ისევე როგორც **Key-Value storage** ტიპის ბაზებში, გასაღები-მონაცემები პრინციპით ინახება იმ განსხვავებით, რომ აქ მონაცემები სტუქტურირებული ან ნახევრადსტუქტურირებული სახით შემდეგ დოკუმენტის ტიპის ფორმატებში შეიძლება შეინახოს: **Xml, Json, Yaml, Bson.** განსხვავებით წინა მაგალითისგან დოკუმენტური ტიპის მონაცემთა ბაზებში შესაძლებელია როგორც გასაღებით ასევე მონაცემებით ფილტრაცია.
მაგ: „მომეცი ყველა ჯგუფის სახელი სადაც, სახელი შეიცავს "D" სიმბოლოს.

```
<artist>
 <artistname>AC/DC</<artistname>
 <albums>
   <album>
     <albumname>Back in Black</albumname>
     <datereleased>1980</datereleased>
     <genre>Hard Rock</genre>
   </album>
   <album>
     <albumname>Powerage</albumname>
     <datereleased>1978</datereleased>
     <genre>Hard Rock</genre>
   </album>
   <album>
     <albumname>High Voltage</albumname>
     <datereleased>1976</datereleased>
     <genre>Hard Rock</genre>
   </album>
 </albums>
</artist>
```

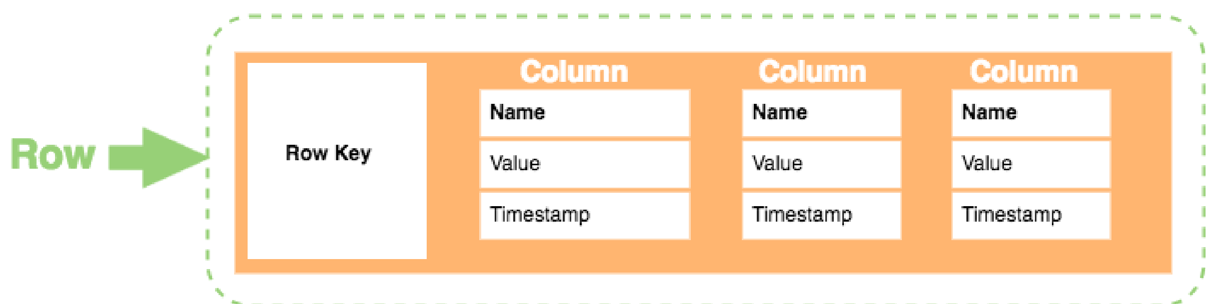
XML ფორმატის დოკუმენტის მაგალითი

```
{
   '_id' : 1,
   'artistName' : { 'AC/DC ' },
   'albums' : [
      {
         'albumname' : 'Back in Black',
         'datereleased' : 1980,
         'genre' : 'Hard Rock'
      }, {
         'albumname' : 'Powerage',
         'datereleased' : 1978,
         'genre' : 'Hard Rock'
      }, {
         'albumname' : 'Powerslave',
         'datereleased' : 1976,
         'genre' : 'Hard Rock'
      }
   ]
}
```

JSON ფორმატის დოკუმენტის მაგალითი

წინა მაგალითისგან განსხვავებით აქ ერთ-ერთ ველად ჩავამატეთ _id, რომელიც ზოგიერთ მართვის სისტემაში სავალდებულო არ არის და ზოგიერთი ჩანაწერს ავტომატურად უგენერირებს უნიკალურ იდენტიფიკატორს თუ მას ხელით არ გავუწერთ.

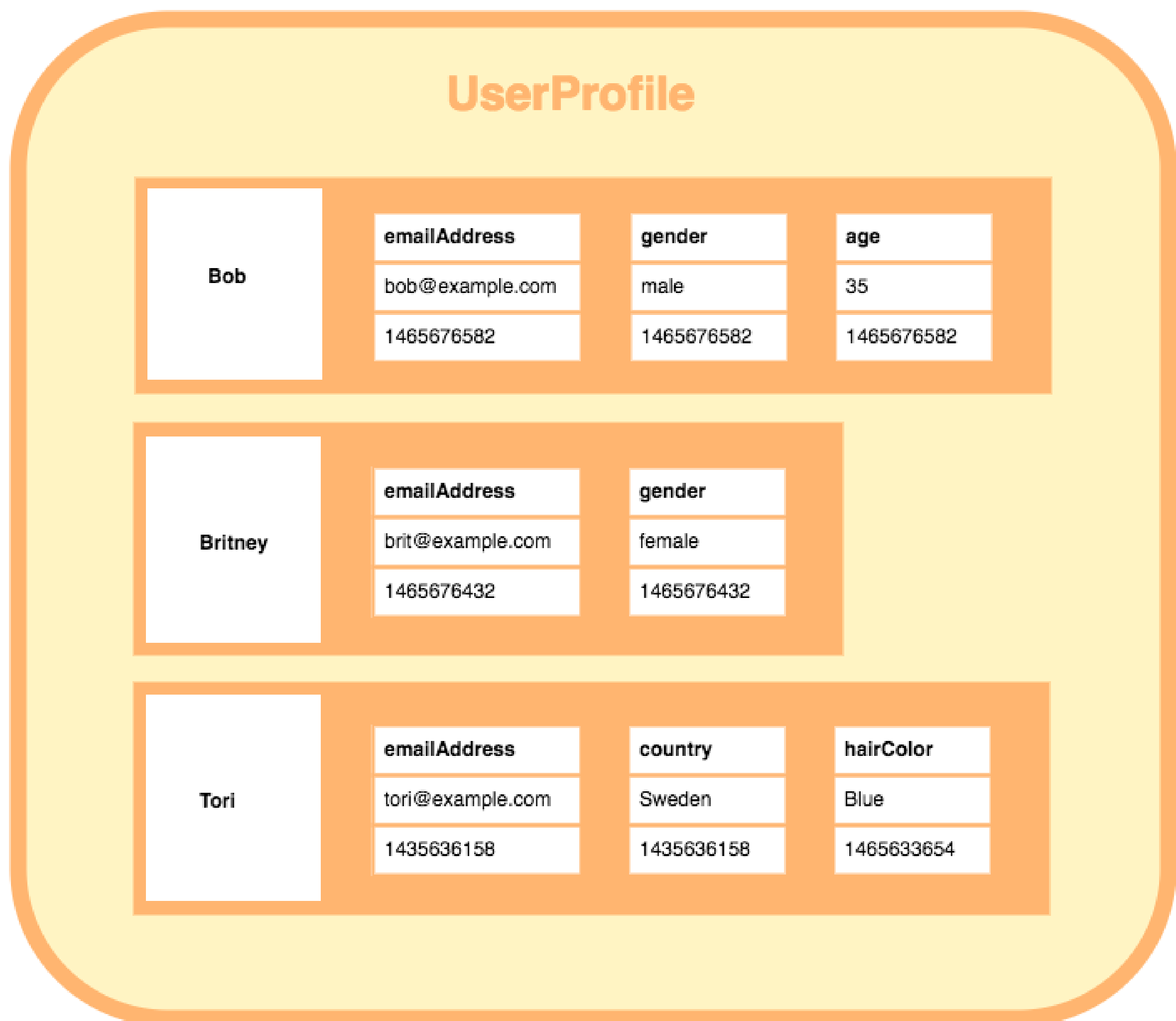
- **Wide Column storage:** ამ ტიპის სტრუქტურაში მონაცემების შენახვის მეთოდი წააგავს რელაციური მონაცემთა ბაზების ცხრილურ სისტემას იმ განსვავებით, რომ აქ თითოეულ ჩანაწერს შეიძლება ჰქონდეს განსხვავებული რაოდენობის და განსხვავებული ტიპის სვეტები.



- **Row key** - წარმოადგენს თითეული ჩანაწერის უნიკალურ იდენტიფიკატორს.
- **Column** - თითოეული სვეტი შეიცავს სვეტის სახელს, მის მნიშვნელობას და დროით მნიშვლეობას.

- **Name** - სვეტის სახელი.
- **Value** - მნიშვნელობა თითოეული ველისთვის/სვეტისთვის. მნიშვნელობა შეიძლება იყოს როგორც პრიმიტიული ტიპის (int, float, string, bool …) ასევე ჩადგმული ობიექტის ტიპიც.
- **Timestamp** - ჩანაწერის გაკეთების დროითი მნიშვნელობა, რომელიც შეიძლება გამოყენებულ იქნას ყველაზე ბოლო ან ყველაზე ადრე გაკეთებული ჩანაწერების გასარჩევად და ა.შ.

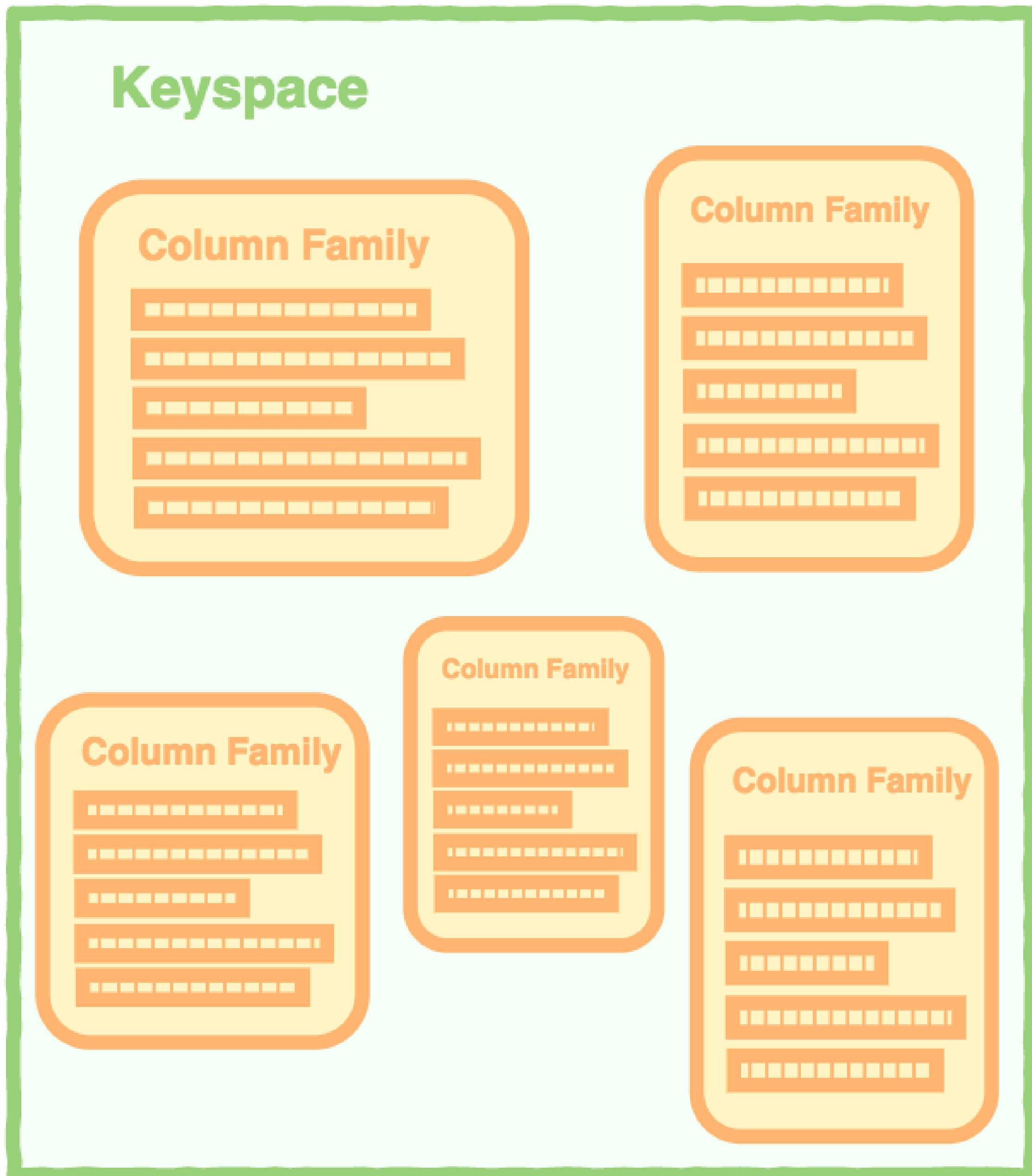
როგორც უკვე ვთქვით **Wide Column storage** ტიპის მონაცემთა ბაზები წააგავს რელაციური ტიპის მონაცემთა ბაზებს. აქ ჩანაწერების ერთობლიობას Column Family - სვეტების ჯგუფები გამოხატავს, რომელიც დაახლოებით იგივეა რაც რელაციურ მოდელში ცხრილები.



UserProfile - წარმოადგენს Collumn Family -ს რომელიც თავის თავში შეიცავს სამ ჩანაწერს, თითოეულ ჩანაწერს აქვს მისთვის დამახასიათებელი მონაცემთა ველების, სვეტების რაოდენობა და ტიპები. თითოეულ მათგანს აქვს უნიკალური

იდენთიფიკატორი, რომელიც ამ შემთხვევაში მომხმარებლის სახელებით გამოიხატება.

თითოეული გაერთიანება იგივე - Column Family თავის მხრივ ერთიანდება ჯგუფებში, რომელთაც Keyspace - ეწოდება და ის დაახლოებით იგივე როლს ასრულებს ამ ტიპის არარელაციურ მონაცემთა ბაზებში, რასაც რელაციურ მოდელში ასრულებს სქემა, იმ განსხვავებით, რომ აქ სქემა დინამიურია.

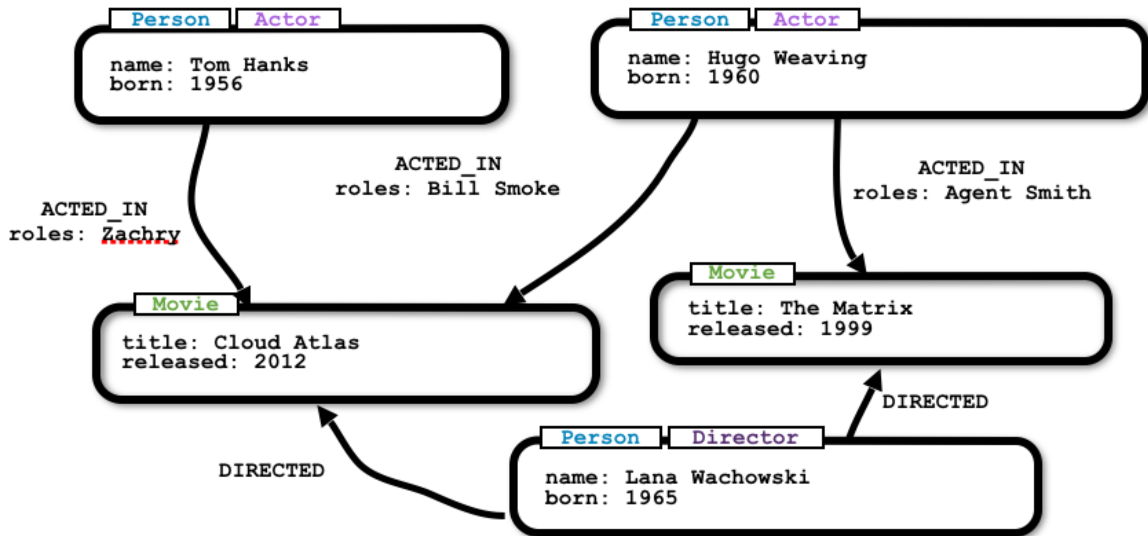


- **Graph database:** გრაფ მონაცემთა ბაზებში მონაცემთა შენახვა გრაფების მსგავს სტრუქტურებში ხდება და ის გრაფთა თეორიას ეფუძნება. აქ გრაფის წვეროები

ობიექტებს წარმოადგენენ ხოლო მათი შემაერთებელი ხაზები - კავშირებს ობიექტებს შრის.

გრაფ სტუქტურის მაგალითი

ზემოთ სურათზე მოცემულ მაგალითში წრეებით ანუ გრაფის წვეროებით აღნიშნულია ობიექტები, შესაბამისად ლურჯით მსახიობები და მწვანით ფილმები მათ შორის კავშირები კი სახელებითაა წარმოდგენილი, ასე მაგალითად: ტომ ჰენქსი მონაწილეობდა ფილმში „That Thing You Do" რომელსაც "ACTED_IN" - კავშირი გვიჩვენებს, ხოლო ამავე ფილმში ტომ ჰენქსი იყო რეჟისორი, რომელსაც "DIRECTED" - კავშირის სახელი გვიჩვენებს ფილმსა და მსახიობს შორის.



თითოეული ობიექტი, გრაფ მონაცემთა ბაზებში დაჯგუფებულია, რაც გრაფის წვეროების „დალეიბლებით" გამოიხატება. ზემოთ მოცემულ სურათზე Tom Hanks(Person, Actor), Hugo Weaving(Person, Actor) და Lana Wachowski (Person,Director) ობიექტებს აქვთ ლეიბლები, ნიშნები რომელიც განსაზღვრავს თუ რომელ ჯგუფში ერთიანდება ესა თუ ის ობიექტი. მაგ: Cloud Atlas და The Matrix ფილმები ერთიანდებიან Movie ჯგუფში, რაც რელაციურ მოდელში შეგვიძლია ცხრილის სახელს შევადაროთ. გრაფ მონაცემთა ბაზებში სქემა დინამიურია და ობიექტების სტრუქტურის და ტიპების განსაზღვრა წინასწარ არ გვიწევს. დაჯგუფებული ობიექტები კი ამარტივებს მონაცემებთან მუშაობას.

ძირითადი დებულებები, რომელთაც ემყარება არარელაციური მოდელი BASE

არარელაციური მონაცემთა ბაზის მართვის სისტმები, გამომგდინარე თავიანთი დისტრიბუციული ბუნებიდან ეფუძნებიან **CAP** თეორემას. სწორედ Eric Brewer ის თეორემის საფუძველზე ჩამოყალიბდა მოდელი **BASE** (Basic Availability, Soft State, Eventual Consistency), რომელიც **ACID** მოდელის ალტერნატივას წარმოადგენს და არარელაციურ მონაცემთა ბაზის მართვის სისტემებში გამოიყენება. **BASE** მოდელი **ACID** სგან განსხვავებით მსუბუქი და ნაკლებად მოთხოვნადია სისტემის მიმართ. სწორედ ამ თვისებების გამოყენებით არარელაციური მონაცემთა ბაზები აღწევენ მაღალ წარმადობას.

- **Basic Availability**: ნიშნავს, რომ სისტემამ ყოველთვის უნდა დააბრუნოს პასუხი მოთხოვნაზე, იმ შემთხვევაშიც კი როცა სისტემის გარკვეული ნაწილები არ მუშაობს. ეს თვისება იგივეა რაც **CAP** თეორემაში Availability (ერთანობა).

- **Soft State:** ნიშნავს, რომ სისტემაში შეიძლება გამუდმებით მიდიოდეს ცვლილებები მონაცემებში, მიუხედავად იმისა კონკრეტულ მომენტში ცდილობს თუ არა ვინმე ამ მონაცემების ცვლილებას. რაც იმას ნიშნავს, რომ აქ თანმიმდევრულობა (Consistency) უფრო პროგრამისტის საქმეა ვიდრე სისტემის.

- **Eventual Consistency**: **ACID** მოდელისგან განსხვავებით, სადაც ყოველი ჩანაწერის ასახვა, და ბაზის მთლიანობის აღდგენა დაუყოვნებლივ ხდება ყოველი ტრანზაქციის ფარგლებში, აქ მთლიანობის აღდგენა “დროში გაწელილი პროცესია , რომელიც აუცილებლად დადგება, თუმცა როდის არ ვიცით. როგორც წესი ბაზის მთლიანობის აღდგენა, როგორც პროცესი, ყოველი ტრანზაქციის ფარგლებში არ ხდება ამას სისტემა დრო და დრო უზრუნველყოფს.