სამაგისტრო ნაშრომი

განსაკუთრებით დიდი ზომის ინფორმაციის დაცვა/დამუშავება მონაცემთა ბაზებში

სტუდენტი: ირაკლი ქობესაშვილი

ხელმძღვანელი: ვახტანგ კვანტალიანი

ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

2015 წელი

სარჩევი

➢ შესავალი

- ინფორმაციის ზრდის პერსპექტივები და ტემპები
- Big Users
- Big Data
- The Internet Of Things
- როდის ხდება "Big Data" რეალურად პრობლემა ?

➢ რელაციური მოდელის შესაძლებლობები და შეზღუდვები.

- ისტორია
- რელაციური მოდელის მიმოხილვა
- ACID კონცეპტი
- ყველაზე პოპულარული რელაციური მონაცემთა მართვის სისტემის მიმოხილვა - Oracle
- რელაციური სისტემების შესაძლებლობები დიდი ზომის მონაცემების დამუშავების დროს. მასშტაბურობა (scalability) და რელაციური სისტემის მიდგომა და პრობლემები ორაკლის მაგალითზე.

➢ არარელაციური მოდელის შესაძლებლობები და შეზღუდვები

- ისტორია
- არარელაციური მოდელის მიმოხილვა
- ყველაზე პოპულარული "NO SQL" ბაზის მიმოხილვა - MongoDB
- (MongoDB) - რეპლიკაცია
- (MongoDB) - შარდინგი (sharding)

➢ პრაქტიკული ამოცანა (Oracle vs MongoDB)

ანოტაცია

სამაგისტრო ნაშრომი შეეხება დღესდღეობით ისეთ აქტუალურ პრობლემას, როგორიცაა განსაკუთრებით დიდი მონაცემების შენახვა და დამუშავება.თუ როგორი რთულია და რიგ შემთხვევაში შეუძლებელია რელაციური მიდგომით ესეთი მოცულობის ინფორმაციის მართვა. ჩვენ განვიხილავთ სხვადასხვა მიდგომებს და გადაწყვეტილებებს არსებული პრობლემის აღმოსაფხვრელად.

ინფორმაციის განსაკუთრებით ზრდა იწვევს ბევრ პრობლემას მისი შენახვისა და დამუშავების დროს, ამისათვის დადგა მოთხოვნილება შექმნილიყო მონაცემთა შენახვისა და დამუშავების ახალი მიდგომა, რომელიც შეძლებდა გამკლავებოდა არსებულ მოთხოვნებს.

1998 წელს კარლო სტროზიმ შემოიღო ტერმინი NOSQL, ის გვთავაზობდა რელაციურ მიდგომაზე უარის თქმას, ამ იდეას შეიძლება ვუწოდოთ არა რელაციური(NoRel).

NOSQL ტექნოლოგიის "პიონერები" იყვნენ ისეთი გიგანტი ინტერნეტ კომპანიები როგორებიც არიან: Google, Facebook, Amazon და LinkedIn. მათ გამოიყენეს მონაცემთა შენახვა/დამუშავების NOSQL ტექნოლოგია, რათა გადაეჭრათ რელაციური მიდგომის შეზღუდვებისგან წარმოქმნილი პრობლემები. დღეს უკვე თანამედროვე სისტემები აქტიურად იყენებენ NOSQL მიდგომის მეგა-ტენდენციებს როგორიც არის: Big Users, Big Data, The Internet of Things და Cloud Computing.

შესავალი

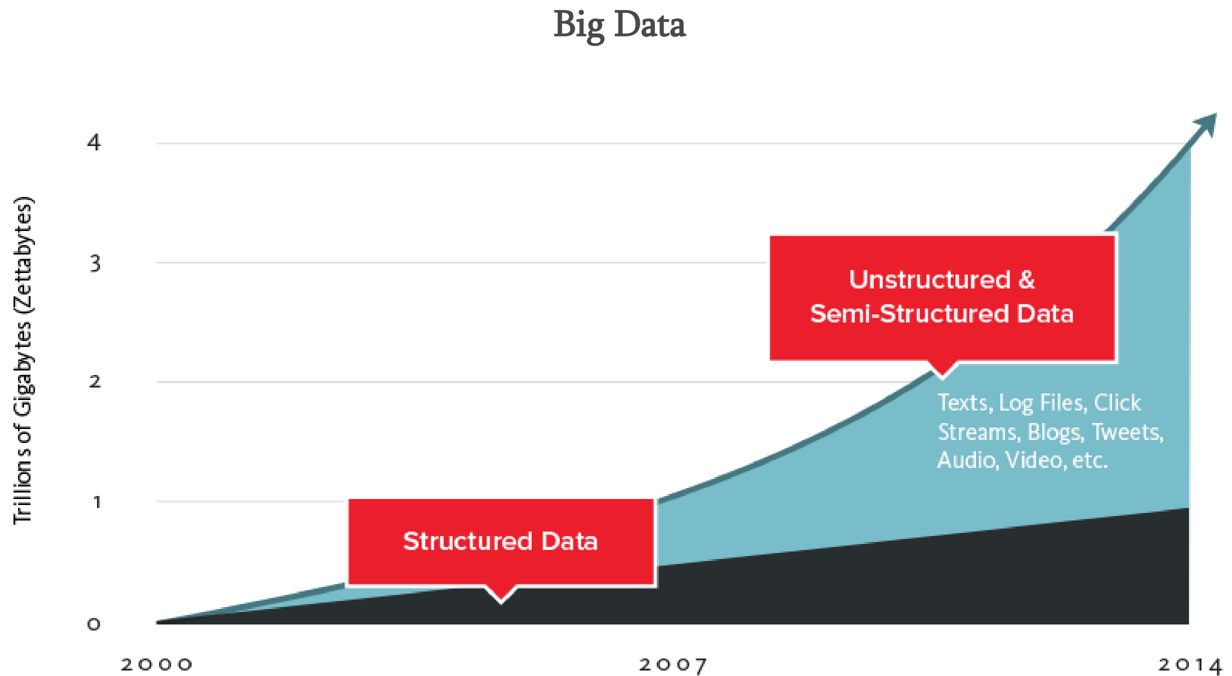
XXI საუკუნეში ელექტრო-გამომთვლელი მანქანა იგივე კომპიუტერი ადამიანის სამუშაო გარემოს განუყოფელ ნაწილად იქცა, ის გვხვდება მოღვაწეობის თითქმის ყველა სფეროში, მის გარეშე წარმოუდგენელი იქნებოდა ნებისმიერი კომპანიის არსებობა. ბიზნეს პროცესის გამართულად მუშაობისთვის, კონტროლისა და მენეჯმენტისთვის არსებობს შესაბამისი პროგრამული უზრუნველყოფა , რომლებიც მონაცემების შენახვა/დამუშავებისთვის იყენებენ სხვადასხვა ტიპის მონაცემთა ბაზებს.

Big Users

არც თუ დიდი ხნის წინ დღეში 1000 მომხმარებელი/ვიზიტორი ითვლებოდა ბევრად , ხოლო 10 000 კი ექსტრემალურ შემთხვევად.დღესდღეისობით ,დაახლოებით 3 მილიარდი მომხარებელი იყენებს ინტერნეტს და დრო რომელსაც ატარებენ არის დაახლოებით 35 მილიარდი საათი თვეში , ეს რიცხვი უფრო და უფრო იზრდება დროთა განმავლობაში.

ასევე არ არის მოულოდნელი ფაქტი , რომ მობილური ინტერნეტისა და აპლიკაციების განვითარებამ, ლომის წვლილი შეიტანა არა რელაციური მონაცემთა ბაზების საჭიროებასა და განვითარებაში. ვინაიდან 24 საათის განმავლობაში

მობილურ აპლიკაციებს და ინტერნეტს იყენებს მილიონობით ადამიანი წელიწადში 365 დღის განმავლობაში.

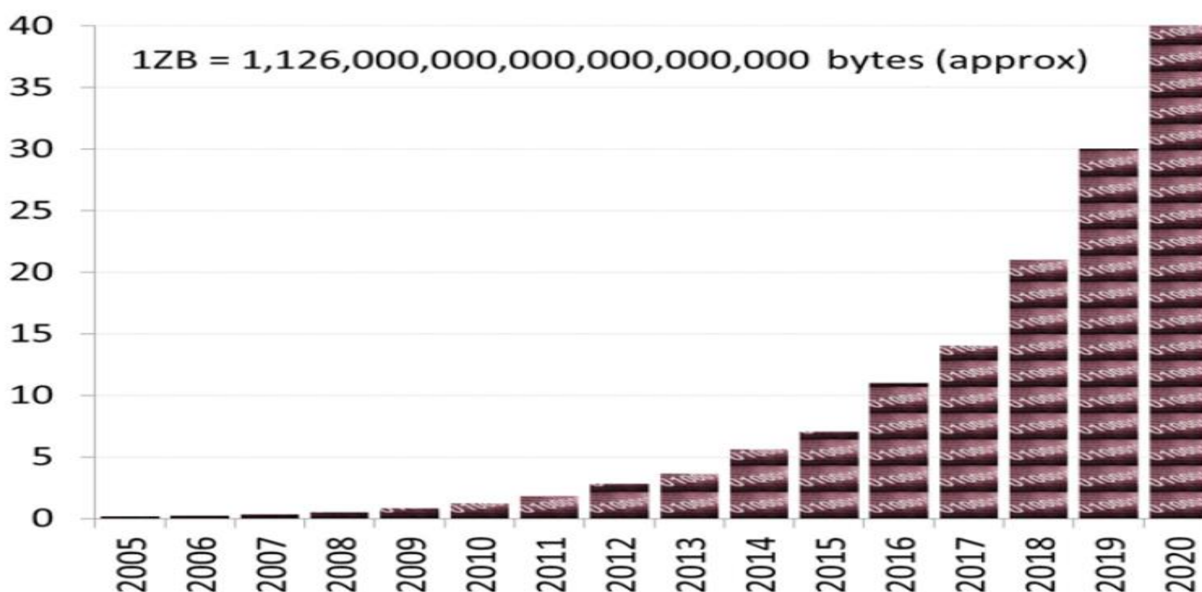


მონაცემები იზრდება სწრაფად, ასევე იცვლება მონაცემთა ტიპები, რადგან დეველოპერებს სჭირდებათ უფრო და უფრო მეტი არასტრუქტურირებული ან ნახევრად სტრუქტურირებული მონაცემების შენახვა და დამუშავება ბიზნეს ამოცანიდან გამომდინარე. არა სტრუქტურირებული ან ნახევრად სტრუქტურირებული მონაცემები, წარმოადგენს ინფორმაციას რომელიც არ არის ან ნაწილობრივ არის სტრუქტურული რელაციური მოდელისთვის.

მისი თვისებებია :

1) არაფიქსირებული სქემა
2) სტრუქტურა ბუნდოვანია და მოუწესრიგებელი
3) ჩადგმული და ჰეტეროგენული
   მაგ: ვებ გვერდები,XML, data integration (კომბინირებულიინფორმაცია მიღებული სხვადასხვა წყაროებიდან)

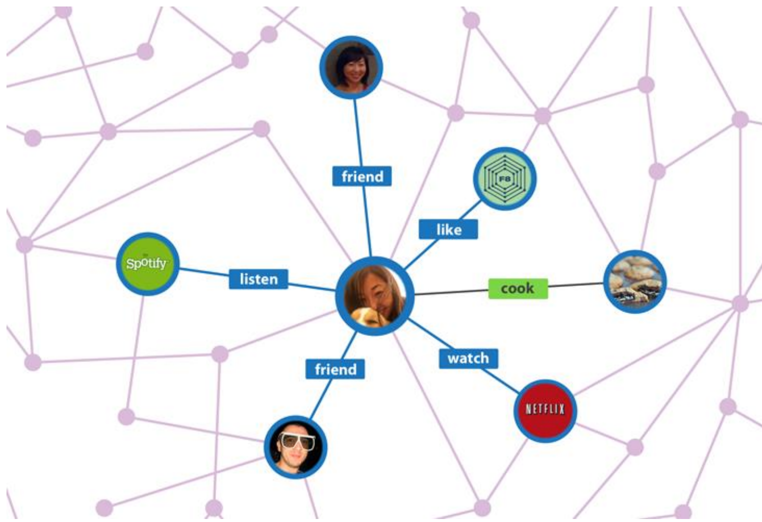
ინტერნეტის მოხმარების სწრაფი ზრდა, მობილური და სოციალური აპლიკაციები, ასევე მანქანა-მანქანა(Machine to Machine) კავშირი მივყავართ Big Data რევოლუციამდე.



IDC(International Data Corporation) კვლევის მიხედვით 2013 წელს მსოფლიოს მასშტაბით ციფრული მონაცემების ზომა იყო 4.4 ზეტაბაიტი - 4.4 ტრილიონი გიგაბაიტი და სავარაუდოდ 2020 წელს ეს ციფრი გახდება 10-ჯერ მეტი ანუ 44 ზეტაბაიტი.

ინფორმაციის მოპოვება და მასთან წვდომა გახდა ძალიან მარტივი მესამე მხარის მეშვეობით მაგალითად Facebook, google+ შეგვიძლია წამოვიღოთ ინფორმაცია და დავამუშავოთ ჩვენს აპლიკაციებში. პერსონალური ინფორმაცია, გეოგრაფიული მდებარეობა, სენსორების მიერ დაგენერირებული მონაცემები, მანქანების ლოგ ინფორმაციები და სოციალური გრაფები არის მცირე ჩამონათვალი რისი გამოყენების და დამუშავების მოთხოვნილება დგება.

სოციალური გრაფები იყენებენ გრაფულ მონაცემთა ბაზებს, რომლებიც დაფუძნებულნი არიან გრაფთა თეორიაზე, მაგალითად შეგვიძლია ავიღოთ facebook-ის მომხმარებლებს შორის კავშირები და დიაგრამის სახით გამოვსახოთ.



(დაიგრამა რომელიც ახდენს ილუსტრაციას ადამიანებს შორის კავშირებზე, მათ ურთიერთიერთქმედებას ადგილთან,მოვლენებთან მიმართებაში)

The Internet Of Things

მანქანის მიერ დაგენერირებული მონაცემების მოცულობა არის ძირითადი მიზეზი განსაკუთრებით დიდი ინფორმაციის დაგროვების.რომელიც იზრდება ერთმანეთისგან დაშორებული მოწყობილობების რიცხვის მატებასთან ერთად.

დღეს,დაახლოებით 20 მილიარდი მოწყობილობა არის დაკავშირებული ერთმანეთთან ინტერნეტის საშუალებით. ეს მოწყობილობები იღებენ მონაცემებს გარემოზე, ადგილმდებარეობაზე, მოძრაობაზე, ტემპერატურაზე, ამინდზე და სხვა. მაგალითად ტაბლეტის გამოყენებით სახლში მისვლამდე შეგვიძლია დავუკავშირდეთ კონდიციონერს და დავაყენოთ სასურველ ტემპერატურაზე, ან ტემპერატურის სენსორი დაუკავშირდეს კონდიციონერს და დაარეგულიროს ტემპერატურა. ამისათვის საჭიროა რომ ორივე მოწყობილობა იყოს ჩართული ინტერნეტში რათა მოხდეს ერთმანეთთან დაკავშირება.

თუმცა ტელემეტრიული მონაცემები რომელიც უწყვეტად გენერირდება პრობლემას უქმნის რელაციურ ბაზას რადგან მას სჭირდება ფიქსირებული სქემა და სტრუქტურირებული მონაცემი.

იმისათვის რომ ვუპასუხოთ ამ გამოწვევებს,ინოვაციური სისტემები ეყრდნობიან NOSQL ტექნოლოგიებს, ამ მოწყობილობებისგან დაგენერირებული ინფორმაციის შესანახად, მონაცემებთან სწრაფი წვდომა, რაც უზრუნველყოფს მილიონობით ერთმანეთთან დაკავშირებული მოწყობილობების გამართულად მუშაობას.

Cloud Computing

დღესდღეისობით ,ახალი აპლიკაციების უმრავლესობა განთავსებულია public, private, და hybrid "Cloud"-ში, იყენებენ სამ დონიან ინტერნეტ არქიტექტურას.

სამ დონიან არქიტექტურაში,აპლიკაციებზე წვდომა ხდება ინტერნეტ ბრაუზერით ან მობილურით,რომლებსაც თავის მხრივ აქვთ ინტერნეტთან კავშირი."Cloud"-ში ხდება შემოსული მოთხოვნების ნაკადის ბალანსირება რათა მოხდეს კონკურენტული მოთხოვნების სწრაფი დამუშავება და შედეგის დაბრუნება.

"Scale-out" არქიტექტურა კარგად მუშაობს აპლიკაციის მხარეს.მაგალითად ყოველ დამატებით 10 000 ახალ კონკურენტულ მომხმარებელზე მარტივად ამატებ ახალ სერვერს სადაც განათავსებ იგივე აპლიკაციას,მათ შორის გადანაწილდება მოთხოვნები. რელცაიური ბაზა პოპულარობით სარგებლობს,მიუხედავად იმისა რომ დატვირთვა მაინც ცენტრალიზებულია.

"NOSQL" ბაზები არის შექმნილი იმისათვის რომ მონაცემი იყოს განაწილებული, ამიტომაც ის ყველაზე კარგად ერგება სამ დონიან არქიტექტურას.

რელაციური მოდელის შესაძლებლობები და შეზღუდვები

ისტორია

1960 წლიდან 1970 წლამდე კოდი მუშაობდა თავის თეორიაზე მონაცემების დალაგების შესახებ,მათემატიკური სიმრავლის თეორიის საფუძველზე.მას სურდა შეენახა მონაცემი ერთმანეთთან დაკავშირებულ ცხრილებში, სხვადასხვა ფორმატით . ეს იყო რევოლუციური მიდგომა, საბოლოოდ 1970 წელს ედგარ კოდმა გამოაქვეყნა სტატია „Relational Model of Data for Large Shared Data Banks “. კოდის ეს იდეა გამოიყენეს ადგილობრივმა ორგანიზაციებმა როგორებიცაა Oracle, Ingre , Informix ,Sybase.

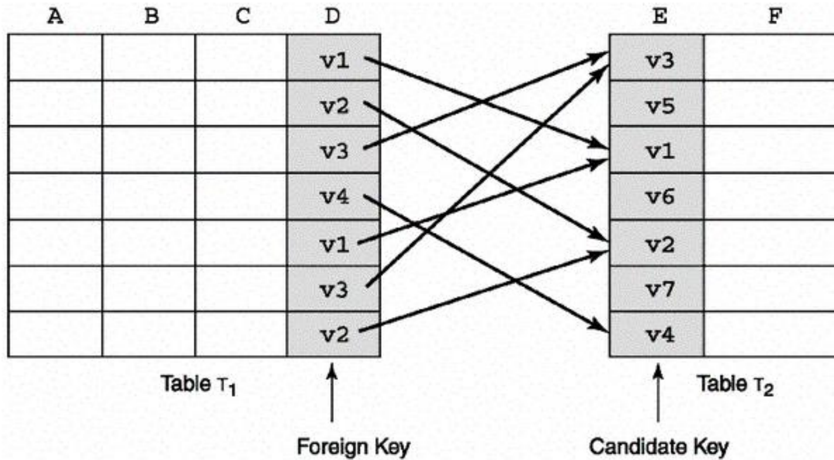
რელაციური მოდელის მიმოხილვა

რელაციური მოდელის ძირეული ელემენტია „რელაცია“ და მსათან დაკავშირებული ცნებები : სტრიქონი - კორტეჟი(row),სვეტი - ატრიბუტი.

Relation
SSAN is a key
Column
Tuple

SSAN	Name	Date of Birth			
999-9	Doug	7/52			

კორტეჟი (Row)- ელემენტების დალაგებული სია.რელაციურ მოდელში ყოველ კორტეჟს უნდა ჰქონდეს თავისი უნიკალური მაინდეტიფიცირებელი.ამ შემთხვევაში SSAN (social security account number) არის მაინდეტიფიცირებელი. რელაციურ მოდელს ასევე აქვს გარე გასაღები რომელიც გამოიყენება სხვა ცხრილთან კონკრეტული კავშირისათვის.



რელაციური მონაცემთა ბაზებისათვის არსებობს რამდენიმე ნორმალური ფორმა.

არსებობს რამდენიმე ნორმალური ფორმა.

1) პირველი ნორმალური ფორმა წარმოადგეს მიმართების ატომურობას, ყველა ატრიბუტი უნდა იყოს ერთმნიშვნელიანი.
   მაგ : თუ person ცხრილის privatenumber ატრიბუტს , მიმართების თითოეულ კორტეჟს უნდა ჰქონდეს ამ ატრიბუტის მხოლოდ ერთი მნიშვნელობა.ხოლო თუ გვჭირდება სხვადასხვა მნიშვნელობა მაშინ ცალკე მიმართება უნდა გაკეთდეს და იქ უნდა ჩაიწეროს ატრიბუტების მუშაობა.

2) მეორე ნორმალური ფორმა მოითხოვს, რომ თითოეული არაგასაღებრივი ატრიბუტი უნდა იყოს ფუნქციონალურად დამოკიდებული მთლიან გასაღებზე. სხვა სიტყვებით რომ ვთქვათ, თითოეული არაგასაღებრივი ატრიბუტის მნიშვნელობა უნდა იძლეოდეს ფაქტს მთლიანი ეგზემპლარის შესახებ, და არა ფაქტს გასაღების ნაწილის შესახებ. თუ მიმართების გასაღები წარმოდგენილია ერთადერთი ატრიბუტის მნიშვნელობებით მაშინ მიმართება ავტომატურად იმყოფება მეორე ნორმალურ ფორმაში.

3) მესამე ნორმალური ფორმა მოითხოვს, რომ მიმართებას არ უნდა გააჩნდეს ტრანზიტული დამოკიდებულებები (transitive dependencies).

სხვა სიტყვებით შეიძლება ითქვას, რომ ყოველი არაგასაღებრივი ატრიბუტი უნდა იძლეოდეს ფაქტს მთლიანი ეგზემპლარის შესახებ და არა რომელიმე სხვა არაგასაღებრივ ატრიბუტზე.

4) ბოის-კოდის ნორმალური ფორმა (Boyce-Codd normal form = BCNF) წარმოადგენს მესამე ნორმალური ფორმის რაფინირებას (დახვეწას). მიმართება იმყოფება ბოის-კოდის ნორმალურ ფორმაში, თუ მიმართების ყოველი დეტერმინანტი წარმოადგენს მიმართების პოტენციურ გასაღებს (შესაძლო გასაღებს). ნებისმიერ პოტენციურ გასაღებს შეუძლია შეასრულოს მიმართების პირველადი გასაღების როლი.

არსებობენ უფრო მაღალი დონის ნორმალური ფორმებიც: მეოთხე, მეხუთე და მეექვსე ნორმალური ფორმები. ყოველდღიურ პრაქტიკულ მოღვაწეობაში მაღალი დონის ნორმალური ფორმის გამოყენების აუცილებლობა საკმაოდ იშვიათია.

ACID კონცეპტი

იმისათვის რომ ბაზა იყოს რელაციური უნდა აკმაყოფილებდეს ACID (Atomicity, Consistency, Isolation, Durability) კონცეპტს.

1) Atomicity (ატომურობა)

   „ან ყველაფერ, ან არაფერი“ გულისმხომბს რომ თუ ტრანზაქციის რომელიმე ნაწილი ვერ/არ შესრულდება ბაზის მდგომარეობა დარჩება შეუცვლელი.

2) Consistency (კონსისტენტურობა)
   კონსისტენტურობის თვისება გულისხმობს რომ როდესაც გაეშვა query-მოთხოვნა ბაზაზე იმ მდგომარეობით უნდა მოგვცეს ინფორმაცია, ანუ თუ მე გავუშვი select და ამის შემდეგ სხვა რომელიმე მომხმარებელმა შეცვალა/წაშალა/ჩაამატა რაიმე ინფორმაცია იმ ცხრილზე რომლიდანაც

მე ვიღებ ინფორმაციას, ჩემთვის უნდა გამოჩნდეს ის ძველი მდგომარეობა რომელიც იყო select გაშვებისას.

3) Isolation (იზოლირებულობა)
ტრანზაქციები რომლებიც გაშვებულია და ჯერ არ არის დაკომიტებული(დასრულებული) რჩება იზოლირებული ერთმანეთისგან. იზოლირებულობა არ განსაზღვრავს რომელი ტრანზაქცია გაეშვება პირველი ,ის უზრუნველყოფს რომ ტრანზაქციები ერთმანეთზე არ მოახდენენ ზემოქმედებას.

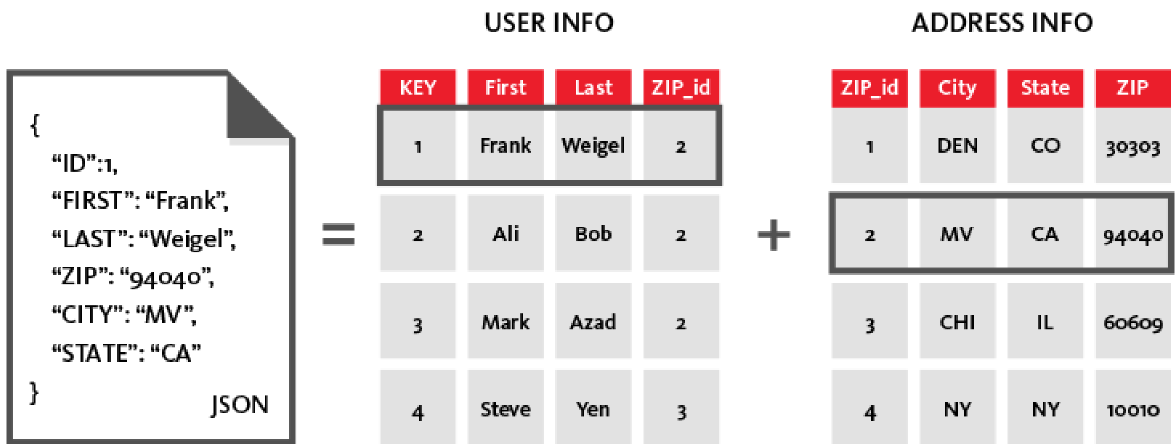
4) Durability (მუდმივობა)
იძლევა გარანტიას რომ თუ ტრანზაქცია დაკომიტტდება შეცვლილი ან ჩაწერილი ინფორმაცია არ დაიკარგება მონაცემთა ბაზიდან.

არარელაციური მოდელის შესაძლებლობები და შეზღუდვები

რელაციური და არა რელაციური მოდელები ძალიან განსხვავდებიან ერთმანეთისგან, რელაციური მოდელი იღებს მონაცემებს და ყოფს მათ ბევრ ერთმანეთთან დაკავშირებულ ცხრილებში, რომელიც შეიცავს სტრიქონებს და სვეტებს, ცხრილები ერთმანეთს უკავშირდება პირველადი და მეორადი გასაღებების მეშვეობით.

როდესაც ვეძებთ რაიმე ინფორმაციას, შესაბამისი ჩანაწერები შეგროვდება რამდენიმე ცხრილიდან შეერთდება და მივიღებთ სასურველ შედეგს, მსგავსია ჩაწერის ოპერაციაც, ჩაწერა უნდა მოხდეს რამდენიმე ცხრილში ერთდროულად.

არა რელაციურ(NOSQL) ბაზებს რელაციურ ბაზებთან შედარებით აქვთ სრულიად განსხვავებული მოდელი. მაგალითად: დოკუმენტ ორიენტირებული NOSQL ბაზები მონაცემებს ინახავენ JSON ფორმატში. თითოეული JSON დოკუმენტი შიეძლება განვიხილოთ როგორც ობიექტი, რომელსაც მიიღებს აპლიკაცია, ის შეიძლება შეიცავდეს რელაციური მოდელის 20 ცხრილის გადაბმით მიღებულ ინფორმაციას ერთ დოკუმენტში/ობიექტში, რაც უზრუნველყოფს ჩაწერის და წაკითხვის ოპერაციების წარმადობის გაუმჯობესებას.



სურათზე ნაჩვენებია არა რელაციური მოდელის ინფორმაციის შენახვა(მარცხნივ) და რელაციური მოდელის ინფორმაციის შენახვა(მარჯვნივ), NOSQL მიდგომა მონაცემებს ინახავს გასაღები-მნიშვნელობა წყვილებით და მიიღება ერთი დოკუმენტი. ცხრილები USER_INFO და ADDRESS_INFO ერთმანეთთან გადაბმულია პირველადი და მეორადი გასაღებებით, ADDRESS_INFO-ს პირველადი გასაღები zip_id დაკავშირებულია USER_INFO-ს მეორად გასაღებთან zip_id. ინფორმაციის ამოღება ისეთ მომხმარებელზე, რომლის ZIP არის 94040 რელაციურ მოდელის მქონე ბაზიდან მოხდება შემდეგნაირად:

```
SELECT u.first, u.last, a.city, a.state, a.zip
  FROM USER_INFO u, ADDRESS_INFO a
 WHERE a.zip = 94040
   AND u.zip_id = a.zip_id;
```

ხოლო არა რელაციურიდან იგივე ინფორმაციის ამოსაღებად გვჭირდება შემდეგი:

db.users.find({ ZIP:94040 }) ; --- MongoDB სინტაქსი

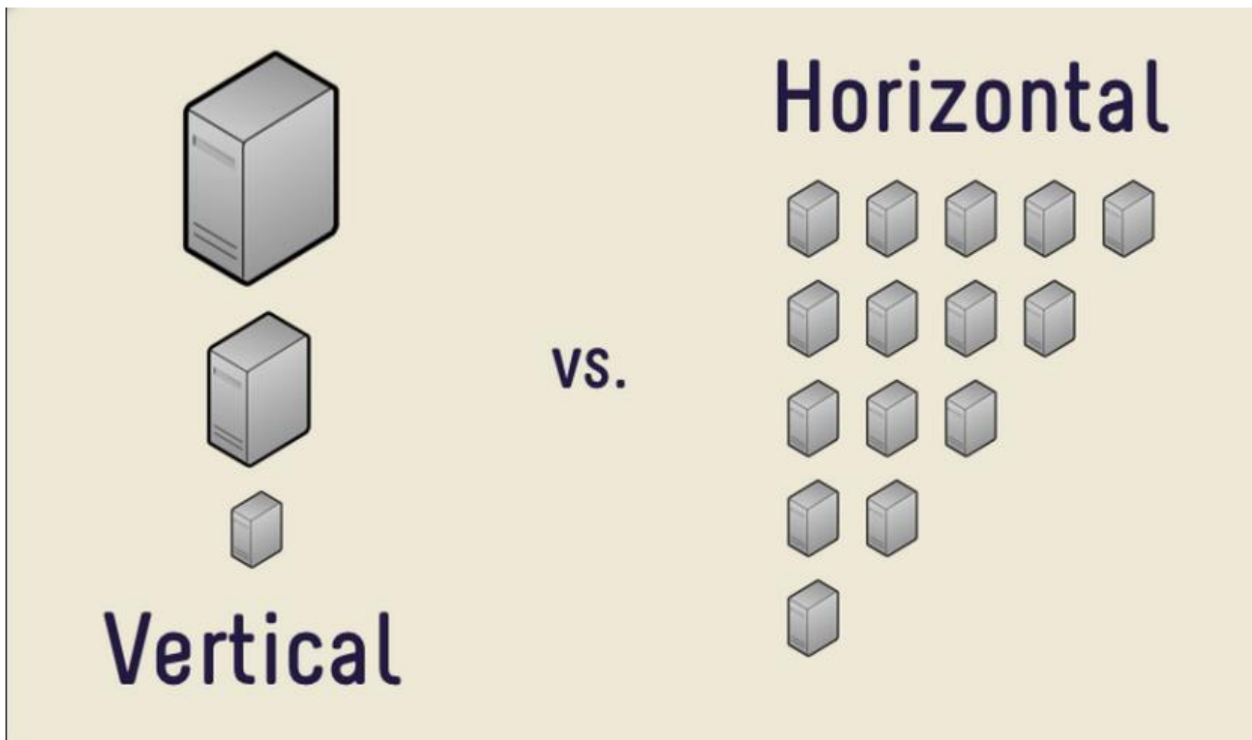
ცხრილების გადაბმის თავიდან არიდებით კი ვიღებთ ჩაწერისა და წაკითხვის წარმადობის გაუმჯობესებას.

ერთ-ერთი უმთავრესი განსხვავება არის ის რომ რელაციური ტექნოლოგია ეყრდნობა მკაცრ სქემას, ხოლო NOSQL ტექნოლოგიაში სქემა არ არსებობს. რელაციური მოდელი მოითხოვს სქემის განსაზღვრას სანამ რაიმეს ჩავწერთ ბაზაში, სქემის შეცვლა მას შემდეგ რაც უკვე მონაცემები ჩაწერილია ბაზაში წარმოადგენს პრობლემას როდესაც გაქვს Big Data, სადაც აპლიკაციის დეველოპერებს მუდმივად სჭირდებათ მონაცემებთან წვდომა, აპლიკაციის დახვეწისთვის. არა რელაციური მოდელის შემთხვევაში სადაც არ არსებობს სქემა, დეველოპერს აქვთ სრული თავისუფლება დაამატონ ველები JSON დოკუმენტში და შეცვალონ არსებული ველის ფორმატი ნებისმიერ დროს აპლიკაციის მონაცემთა ბაზის მუშაობის შეფერხების გარეშე.

არა რელაციური მონაცემთა ბაზები მხარს უჭერენ როგორც ვერტიკალურ ასევე ჰორიზონტალურ მასშტაბურობას (scale), რაც პრაქტიკულად შეუძლებელია რელაციური მონაცემთა ბაზების შემთხვევაში, ისინი იყენებენ ვერტიკალური მასშტაბურობას.

ვერტიკალური მასშტაბურობა გულისხმობს, სერვერის მონაცემების გაზრდას ან ახალ უფრო მძლავრ სერვერზე ბაზის გადატანას, მაგრამ ამას აქვს დიდი მინუსი, რაც უფრო გაიზრდება ბაზა მით უფრო მეტი რესურსი დაჭირდება, შესაბამისად გვიწევს ერთი სერვერის მონაცემთა გაზრდა დაუსრულებლად.

NOSQL ტექნოლოგიაში შეგვიძლია გამოვიყენოთ ჰორიზონტალური მასშტაბურობა, რაც გულისხმობს, რომ ერთი ძლიერი სერვერის ნაცვლად გვაქვს რამდენიმე სერვერი და ოპერაციები სრულდება თითოეულ მათგანზე პარალელურად, როდესაც საკმარისი აღარ იქნება მიმდინარე რესურსი ვამატებთ იგივე მონაცემების და არა უფრო მძლავრ ახალ სერვერს, შესაბამისად ეს სერვერიც ერთვება მთლიან არქიტექტურაში და პარალელურად სრულდება მათზე დაკისრებული სამუშაოები.

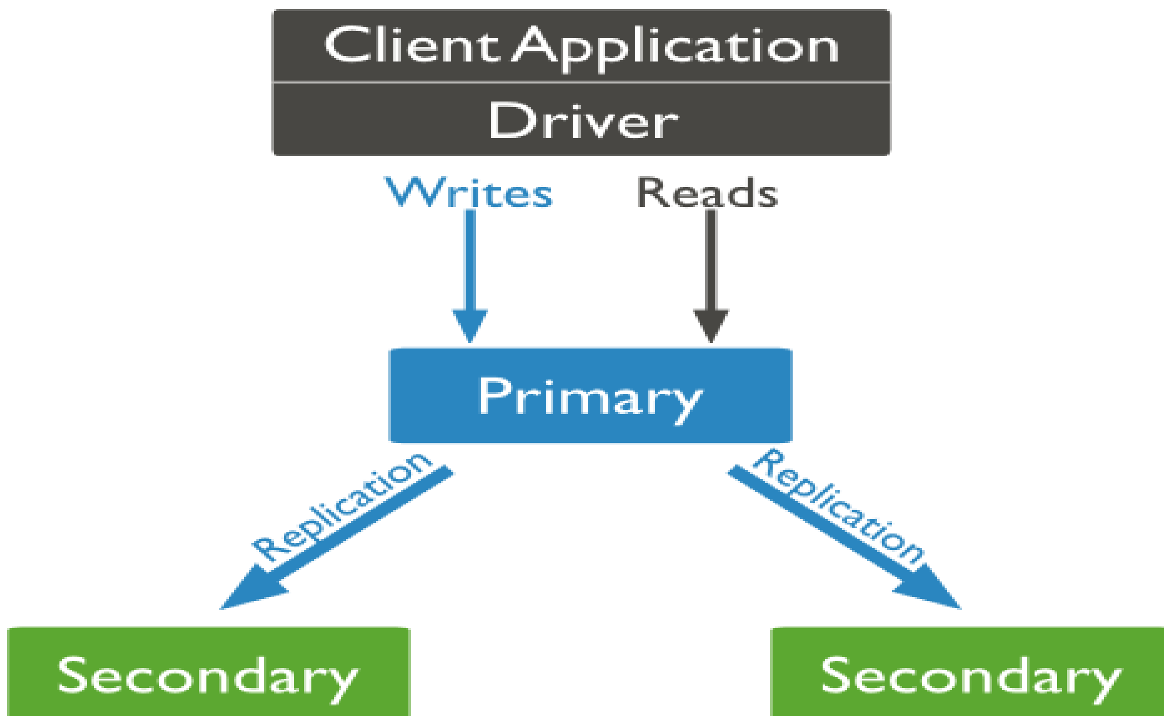


MongoDB - რეპლიკაცია

რეპლიკაცია“ არის მონაცემთა სინქრონიზაცია რამდენიმე სერვერს შორის. ის უზრუნველყოფს redundancy-ის და ამაღლებს მონაცემთა წვდომადობას. მისი მეშვეობით შეგვიძლია დავიცვათ მონაცემთა ბაზა მონაცემების დაკარგვისგან.

Replica set არის mongod „ინსტანსების“ ჯგუფი. Replica set-ში გვაქვს შემდეგი ტიპის წევრები: პირველადი, მეორადი და არბიტრი.

გვაქვს მხოლოდ ერთი პირველადი ბაზა, რომლის მეშვეობითაც ხდება ჩაწერის და წაკითხვის ოპერაციები, ხოლო მეორადი ბაზები იღებენ მონაცემებს პირველადისგან, რათა მოხდეს მონაცემთა სინქრონიზაცია.



არბიტრი არ შეიცავს მომხმარებელთა ინფორმაციას, ის არსებობს იმიტომ რომ მისცეს ხმა არჩევნებში.

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

არბიტრი ყოველთვის იქნება არბიტრი, ხოლო პირველადი შეიძლება გახდეს მეორადი და პირიქით.

ავტომატურად კლიენტები მონაცემთა კითხვისათვის მიმართავენ პირველადს, კლიენტის დონეზე შეგვიძლია მივუთითოთ „Read Preference“ და კითხვა განვახორციელოთ მეორადიდან, რათა ავიცილოთ პირველადის ზედმეტი დატვირთვა.

Read Preference Mode	აღწერა
primary	ავტომატური მნიშვნელობა, კითხვა ხდება პირველადიდან,
primaryPreferred	უმეტეს შემთხვევაში კითხვა ხდება პირველადიდან, მაგრამ თუ ის არ არის წვდომადი კითხვა მოხდება მეორადიდან.
secondary	კითხვა ხდება მეორადიდან.
secondaryPreferred	უმეტეს შემთხვევაში კითხვა ხდება მეორადიდან, მაგრამ თუ ის არ არის წვდომადი კითხვა მოხდება პირველადიდან.
nearest	კითხვა ხდება Replica set-ის იმ წევრიდან რომლისგანაც მინიმალური (latency)შეყოვნება იქნება, მიუხედავად მისი ტიპისა

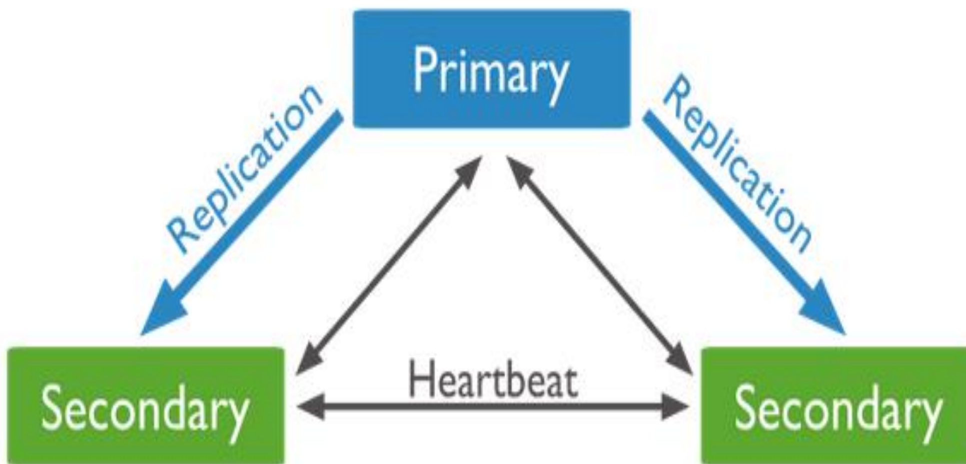
როდესაც Replica set-ში პირველადი არ უკავშირდება სხვა წევრებს 10 წამზე მეტი დროით, მაშინ mongodb შეეცდება ჩაატაროს არჩევნები რათა სხვა წევრი გახდეს პირველადი.

მეორადი ბაზა შეიძლება განვსაზღვროთ პრიორიტეტით 0, მაშინ ის არასოდეს გახდება პირველადი, დანარჩენი ფუნქციონალი აქვს იგივე რაც ჩვეულებრივ მეორადს.

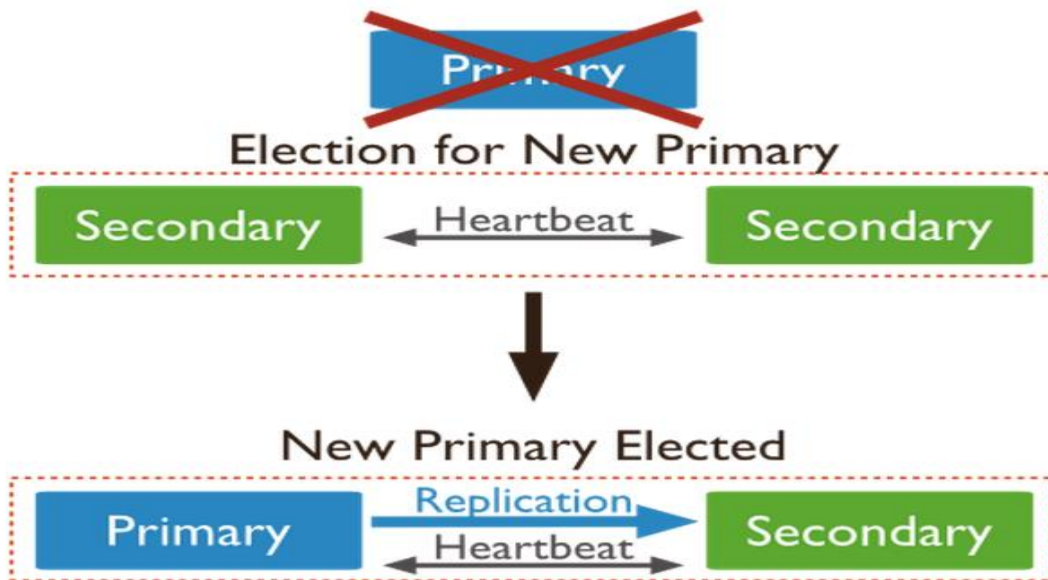
მეორადი აგრეთვე შეიძლება განვსაზღვროთ როგორც (hidden) უხილავი, ის უხილავი იქნება კლიენტის აპლიკაციისთვის, აგრეთვე იქნება პრიორიტეტით 0 და არასოდეს გახდება პირველადი.

თუ პირველადი არის მიუწვდომელი, მაშინ Replica set ჩაატარებს არჩევნებს და რომელიმე მეორადი გახდება პირველადი.

გვაქვს Replica set სამი წევრით, ერთი არის პირველადი, ხოლო ორი ჩვეულებრივი მეორადი. კენტი რიცხვი იმიტომ გვაქვს, რომ ხმათა უმრავლესობას მივაღწიოთ არჩევნებში.



გაითიშა პირველადი, ჩატარდება არჩევნები და აირჩევა ახალი პირველადი, მივიღებთ შემდეგ სიტუაციას:



შარდინგი MongoDB-ში

შარდინგი არის მონაცემების სხვადასხვა მანქანებზე შენახვის მეთოდი.

მონაცემთა ბაზების ზომა ყოველდღიურად იზრდება, რაც იწვევს სხვადასხვა პრობლემებს, თუ ამისათვის არ ვართ მომზადებულები.

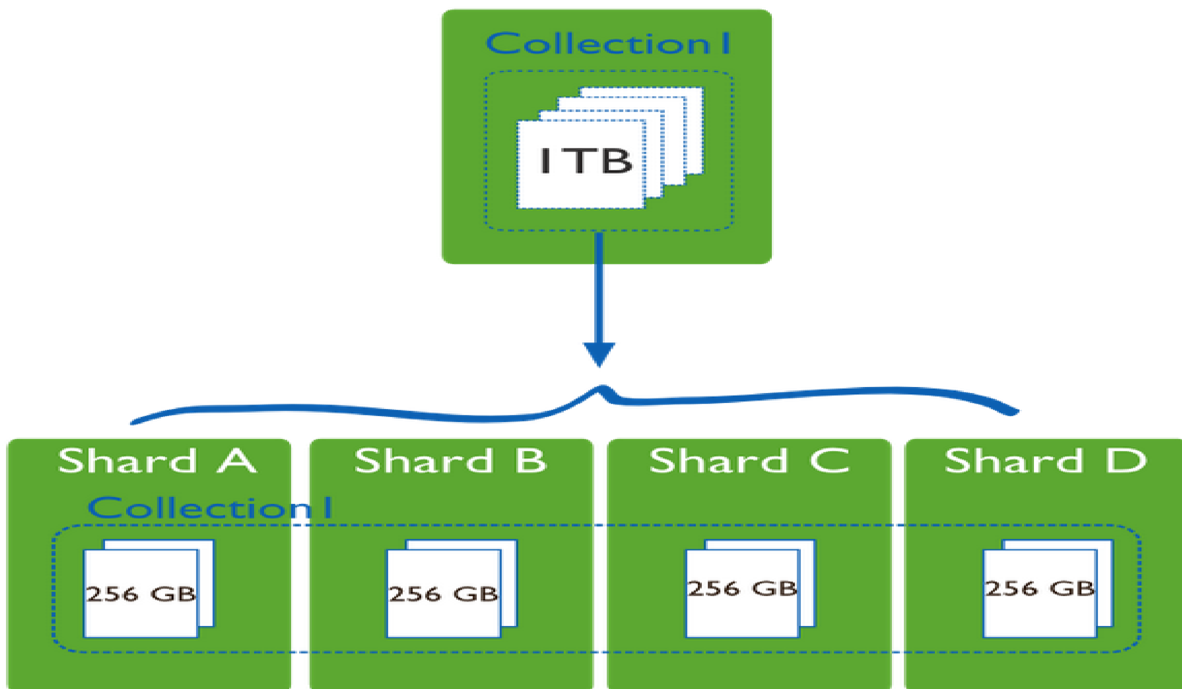
მონაცემთა ბაზებში გამოიყენება ვერტიკალური მასშტაბურობა(scale) და/ან ჰორიზონტალური.

ვერტიკალური მასშტაბურობა გულისხმობს, იმ სერვერის მონაცემების გაზრდას ან ახალ უფრო მძლავრ სერვერზე ბაზის გადატანას, მაგრამ ამას აქვს დიდი მინუსი, რაც უფრო გაიზრდება ბაზა მით უფრო მეტი რესურსი დაჭირდება, შესაბამისად გვიწევს ერთი სერვერის მონაცემთა გაზრდა დაუსრულებლად.

MongoDB-ში შეგვიძლია გამოვიყენოთ ჰორიზონტალურ მასშტაბურობა, რაც გულისხმობს, რომ ერთი ძლიერი სერვერის ნაცვლად გვაქვს რამდენიმე სერვერი და ოპერაციები სრულდება თითოეულ მათგანზე პარალელურად.

შარდინგის მეშვეობით მონაცემები ნაწილდება რამდენიმე სერვერზე(შარდზე).

თითოეული შარდი არის დამოუკიდებელი ბაზა და ერთობლიობაში ქმნიან ერთ მთლიან ბაზას.



შარდინგი ამცირებს ოპერაციების რაოდენობას, მაგალითად თუ გვინდა ჩავწეროთ რაიმე ინფორმაცია insert ბრძანებით, აპლიკაციას სჭირდება დაუკავშირდეს მხოლოდ იმ შარდს, რომელიც არის პასუხისმგებელი ამ ოპერაციაზე.

შარდინგი ამცირებს მონაცემთა რაოდენობას, რომელიც ინახება სერვერზე(შარდზე), მაგალითად თუ გვაქვს ბაზა 1 TB და გვაქვს ოთხი შარდი, როგორც ჩვენს სურათზე, მაშინ თითოეული შარდი შეინახავს 256 GB, თუ გვექნება 40 შარდი, მაშინ თითოეული შეინახავს 25 GB.

შარდის კლასტერი შეიცავს შემდეგ კომპონენტებს:

 შარდები.

 კონფიგურაციის სერვერებს - config servers, რომელიც ინახავს metadata-ს შარდების შესახებ.

"ქუერი როუტერი" – mongos არის მარშრუტიზატორი, რომელიც იღებს კლიენტის მოთხოვნებს, უგზავნის შარდს და შემდეგ შედეგს აწვდის ისევ კლიენტს.

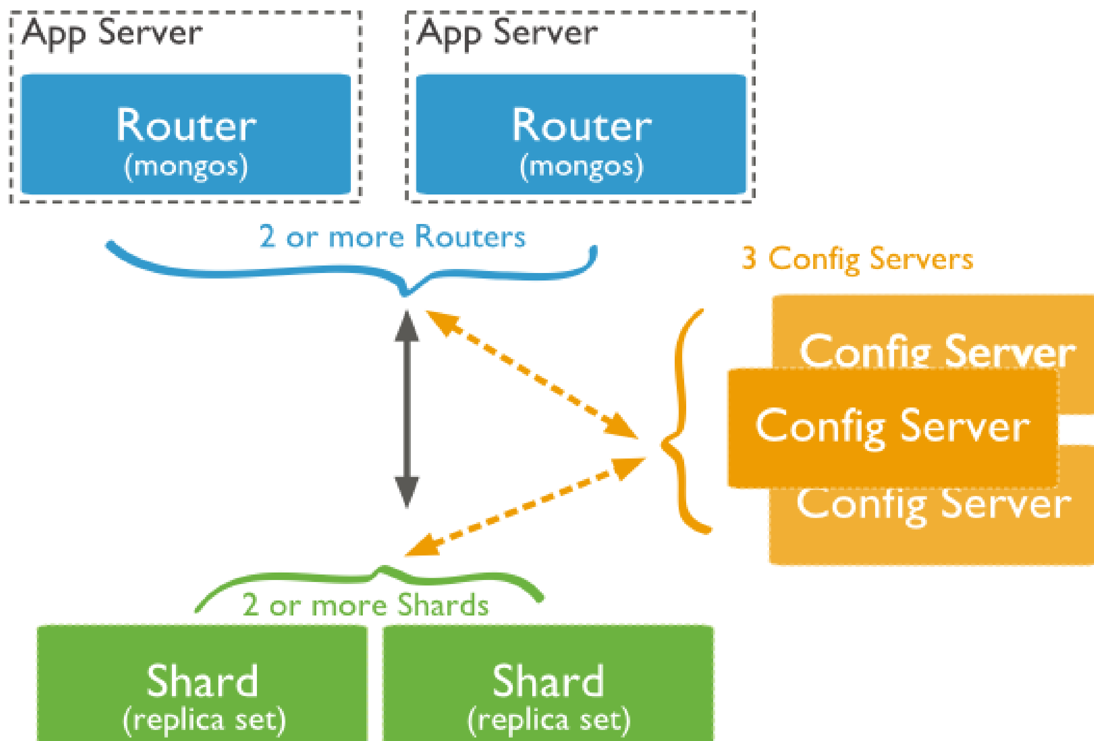


Diagram of a sample sharded cluster for production purposes. Contains exactly 3 config servers, 2 or more `mongos` query routers, and at least 2 shards. The shards are replica sets.

მონაცემების განაწილება ხდება შარდის გასაღებით(shard key).

შარდის გასაღები არის ინდექსირებული ველი ან ინდექსირებული ველთა ჯგუფი, რომელიც არსებობს კოლექციის ყველა დოკუმენტში.

MongoDB დაყოფს შარდის გასაღებს „ჩანკებად"(chunks) და გაანაწილებს ჩანკებს შარდებზე.

შარდის გასაღების ჩანკებად დასაყოფად MongoDB იყენებს Range based partitioning ან hash based partitioning.

Range based partitioning-ის დროს MongoDB დაყოფს მონაცემებს მწკრივებად, შარდის გასაღების მეშვეობით და წარმოადგენს მას როგორც ჩანკებს.

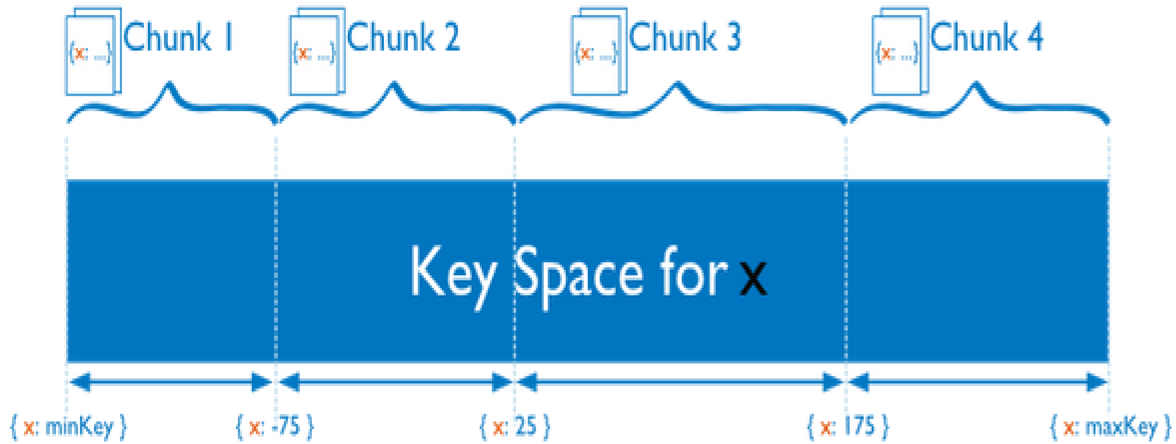


Diagram of the shard key value space segmented into smaller ranges or chunks.

Hash based partitioning-ის დროს MongoDB გამოთვლის ველის მნიშვნელობის ჰეშს და გამოიყენებს მას ჩანკების შესაქმნელად.

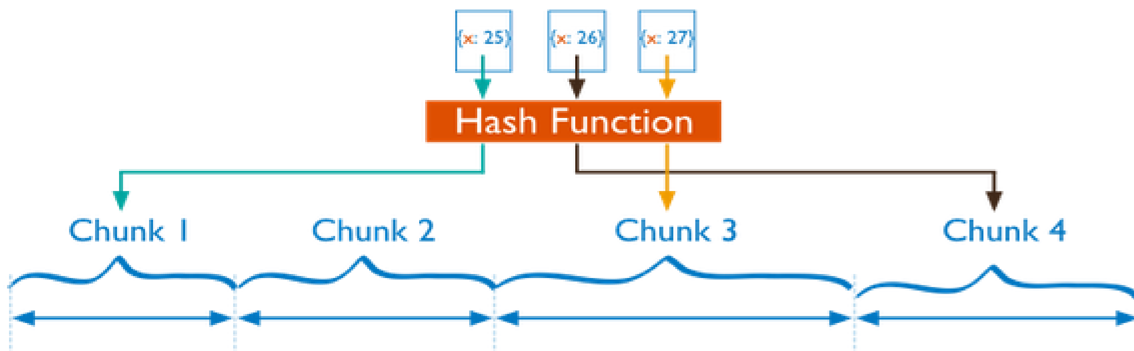
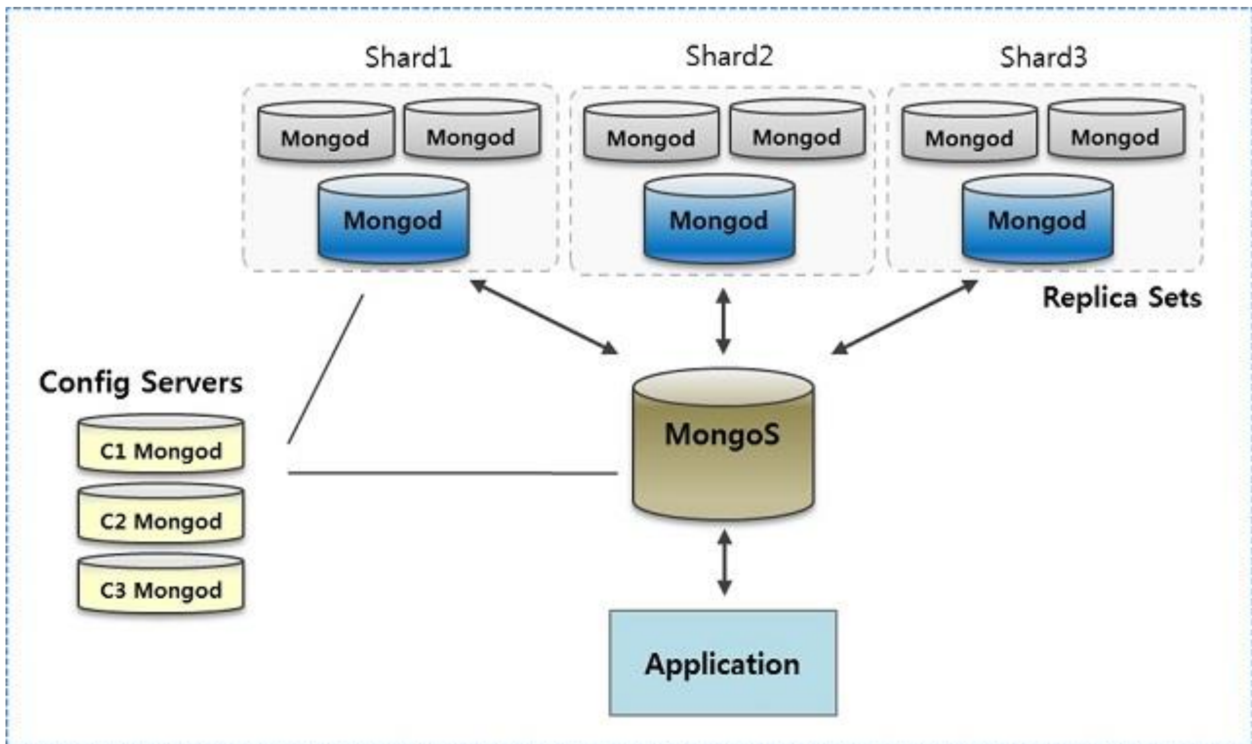


Diagram of the hashed based segmentation.

select ბრძანების მუშაობის მაგალითი:



განვიხილოთ პატარა მაგალითი, თუ როგორ მოქმედებს შარდინგი, mongoDB-ში: მომხმარებელმა მოითხოვა რაღაც ინფორმაცია select ბრძანებით, პირველ რიგში მოთხოვნა შემოვა "ქუერი როუტერთან" – mongos, რომელიც კონფიგურაციის სერვერიდან მიიღებს ინფორმაციას, თუ რომელ შარდს უნდა მიმართოს ამ კონკრეტული მოთხოვნის შესასრულებლად და გადასცემს მოთხოვნას შესაბამის შარდს, ხოლო მისგან მიღებულ ინფორმაციას დაუბრუნებს კლიენტს, ამ დროს სხვა შარდები არ გამოიყენება ამ ოპერაციის შესასრულებლად, შესაბამისად ვიღებთ ოპერაციების გადანაწილებას, ჰორიზონტალურად, რაც თავის მხრივ სწრაფქმედებას უზრუნველყოფს.

პრაქტიკული ამოცანა (Oracle vs MongoDB)

ამოცანა:

1. გვაქვს 150-200 GB დღიური მონაცემები ძირითადად web browsing-ის ლოგები.
2. ინფორმაცია მოწოდებულია ტექსტური ფაილების მეშვეობით.
3. დღეში არის დაახლოებით 40 000 - 50 000 ფაილი.
4. ჩვენი ტესტისთვის გამოვიყენეთ მხოლოდ 1 საათის ინფორმაცია.

მიზანი:

მოვათავსოთ ეს ინფორმაცია მონაცემთა ბაზაში უმოკლეს დროში.

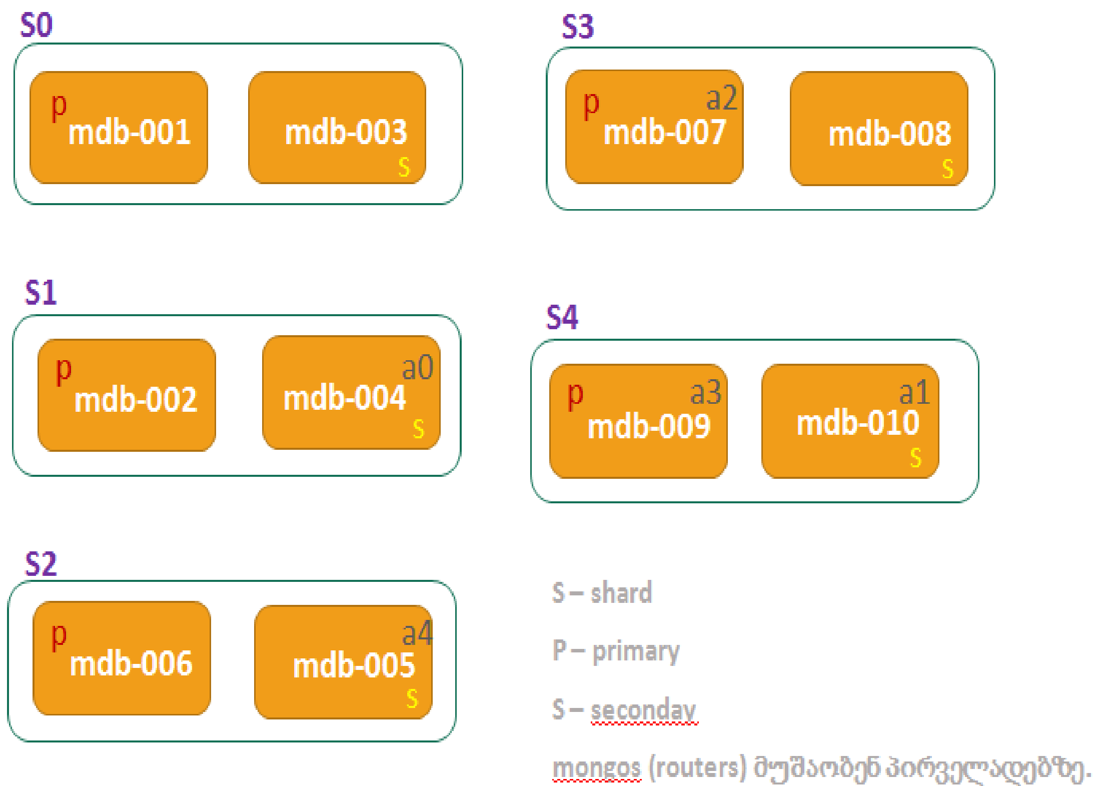
ჩვენი ამოცანის ფარგლებში აწყობილი კლასტერი

Clustering factor – 2 (one *Secondary* in replica set)

OS – Oracle Linux (red hat) 6.5
Avg Memory – 4 GB
CPU – 4 Cores, AMD Opteron 2220, Intel Xeon
Network – 1Gbit
HDD – 100-300 72k, GB local.

10 machines – DL380 g3, g4, Sun x4200, DL580 G5 (VM)

ჩვენი კლასტერის კომპონენტები



შესრულებული სამუშაოები

- გადავანაწილეთ ტექსტური ფაილები შარდებზე.
- გარდავქმენით ფაილები json ფორმატში
- მონაცემების ჩატვირთა ბაზაში განხორციელდა ორი გზით

1. mongoimport - MongoDB-ის ოფიციალური ხელსაწყო
2. Phyton script -(შედეგი უფრო სწრაფი ჩატვირთვა) bulk load მეშვეობით.

- წუთი ავიღეთ shard key-დ, მონაცემები შარდებზე გადანაწილდა ველი - წუთის მეშვეობით 0-დან 59 ჩათვლით.
- შევქმენით ინდექსი მას შემდეგ რაც მონაცემები ჩაიტვირთა რათა ჩატვირთვის სისწრაფე არ შეგვენელებინა.

შედეგები

- ერთი საათის მონაცემები ბაზაში იტვირთება 7-10 წუთის განმავლობაში.
- დაახლოებით 3-4 საათი დასჭირდება ერთი დღის ინფორმაციის ჩატვირთვას.
- თუ დავამატებთ სხვა სერვერებს და ავირჩევთ უკეთეს shard key-ს წარმადობა გაიზრდება საგრძნობლად.
- შედარებისთვის :

მივიღეთ 4-ჯერ მეტი სისწრაფე ვიდრე ძალიან ძვირი Oracle rdbms + DI ძალიან ძვირ

სერვერებზე.

ეს ტესტი ჩატარებულია MongoDB 2.6-ზე, მის შემდეგ გამოვიდა 3.0 ვერსია რომელშიც

ბევრი მნიშვნელოვანი ასპექტი არის გაუმჯობესებული და კიდევ უფრო მეტ წარმადობას

მივიღებთ.

გამოყენებული პითონის სკრიპტი :

```
import csv

import sys

import re

import glob

import socket


from joblib import Parallel, delayed

from time import time

from pymongo import MongoClient

from multiprocessing import Pool

from multiprocessing.dummy import Pool as ThreadPool
```

```
connection = MongoClient ('localhost',27017);
db=connection.impdb

def csv_reader((file_obj,)):
    with open(file_obj, "r") as f_obj:
       bulk = db.big.initialize_unordered_bulk_op()
       reader = csv.reader(f_obj, delimiter='\t')
       for row in reader:
         data = {"url": row[20], "dur": row[0], "date": row[2], "hh": int(row[2][11:-6])}
         bulk.insert(data)
       #db.big.insert(data)
       try:
        res=bulk.execute()
       #print res
       except:
        e = sys.exc_info()[0]
       # print "<p>Error: %s</p>" % e
       f_obj.close()

file_list = glob.glob('/home/mongo/load' + '/*.cdr')
currtime = time()
po = Pool(4)
res = po.map_async(csv_reader,((csv_path,) for csv_path in file_list))
po.close()
po.join()
```

```
print socket.gethostname() + ':time elapsed mins:', (time() - currtime)/60
```

აწყობილი სისტემის სტატუსი :

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
	"_id" : 1,
	"minCompatibleVersion" : 5,
	"currentVersion" : 6,
	"clusterId" : ObjectId("53e9c1271e3d349ea9da97fe")
}
  shards:
	{  "_id" : "s0",  "host" : "s0/mdb-001:31100,mdb-003:31101" }
	{  "_id" : "s1",  "host" : "s1/mdb-002:31100,mdb-011:31103" }
	{  "_id" : "s2",  "host" : "s2/virtual-mdb-005:31101,virtual-mdb-006:31100" }
	{  "_id" : "s3",  "host" : "s3/virtual-mdb-007:31100,virtual-mdb-008:31101" }
	{  "_id" : "s4",  "host" : "s4/mdb-009:31100,mdb-010:31101" }
  balancer:
	Currently enabled:  no
	Currently running:  no
	Failed balancer rounds in last 5 attempts:  0
	Migration Results for the last 24 hours:
		No recent migrations
```

```
databases:
    {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
    {  "_id" : "impdb",  "partitioned" : true,  "primary" : "s1" }
        impdb.big
            shard key: { "hh" : 1 }
            chunks:
                s0    1
                s1    3
                s2    1
                s3    1
                s4    1
            { "hh" : { "$minKey" : 1 } } -->> { "hh" : 0 } on : s1 Timestamp(5, 1)
            { "hh" : 0 } -->> { "hh" : 4 } on : s0 Timestamp(5, 0)
            { "hh" : 4 } -->> { "hh" : 8 } on : s1 Timestamp(1, 5)
            { "hh" : 8 } -->> { "hh" : 12 } on : s3 Timestamp(3, 0)
            { "hh" : 12 } -->> { "hh" : 16 } on : s2 Timestamp(2, 0)
            { "hh" : 16 } -->> { "hh" : 20 } on : s1 Timestamp(1, 11)
            { "hh" : 20 } -->> { "hh" : { "$maxKey" : 1 } } on : s4 Timestamp(4, 0)
    {  "_id" : "test",  "partitioned" : false,  "primary" : "s1" }

mongos>
```

დასკვნა

როდესაც საქმე გვაქვს ისეთი დიდი ინფორმაციის დამუშავებასთან, როგორიც არის მოწყობილობების მიერ დაგენერირებული ლოგები, მობილური ინტერნეტით სარგებლობისას მიღებული web მისამართები, გეოგრაფიული მდებარეობის მონაცემები და სხვა, სადაც ძირითადად გვხვდება არა სტრუქტურირებული მონაცემები, ინფორმაციის დაცვა/დამუშავებისთვის უმჯობესია გამოვიყენოთ არა რომელიმე რელაციური მოდელის მქონდე მონაცემთა ბაზა, არამედ არა რელაციური, NOSQL მოდელის მქონე მონაცემთა ბაზა.

გამოყენებული ლიტერატურა და ბმულები:

OCA-OCP Oracle Database 11g All-in-One Exam Guide (Exam 1Z0-051, 1Z0-052, and 1Z0-053)
http://docs.mongodb.org/manual/
http://www.couchbase.com/nosql-resources/what-is-no-sql
http://en.wikipedia.org/wiki/NoSQL