CSD DERS NOTLARI – MONGO DB

1. Büyük Veri Nedir?

Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it...

Dan Ariely (Professor at MIT)

Büyük verinin herkes tarafından kabul edilmiş olan tam bir tanımı yoktur. Büyük verinin karakteristik özelliklerini Doud Laney 3V ile açıklamıştır. 3V'li betimleme en çok kullanılan betimlemedir:

Volume (Miktar): Büyük veride işlenecek veri miktarı çok fazladır.
Velocity (Hız): Kaynaklardan veriler çok büyük hızlarda gelmektedir.
Variety (Değişkenlik): Verinin niteliği çok değişik olabilmektedir. (Örneğin resim, yazı, ses, görüntü vs.)

3V'ye daha sonra bir 3V daha eklenmiş (veracity, validity, volatily), 6V haline getirilmiştir.

1) Miktar (Volume)

Büyük verinin en önemli özelliği yüksek volümlü olmasıdır. Eskiden büyük miktarda verilerle karşılaşmak çok zordu. Büyük miktarda veriler operatörler tarafından günlerce, aylarca hatta yıllarca elle tek tek giriliyordu. Ancak artık verilerin elde ediliş biçimleri değişmiştir. Şöyle ki: Veriler -sosyal medya uygulamalarında olduğu gibi- milyonlarca kişi tarafından oluşturulabilmektedir. Veriler çeşitli aygıtlar tarafından otomatik biçimde elde edilebilmektedir. (Örneğin kameralar, turnike geçiş sistemleri, MR cihazları, GPS aygıtları, Arama motorları, Bilet reservasyon siteleri vs.) Verinin elde ediliş biçiminin değişmesi miktarın büyümesinde önemli rol oynamıştır. Zaten «büyük veri» kavramı bu gelişmelerden sonra ortaya çıkmıştır. Eskiden büyük verilerle sık karşılaşılmıyordu. Zaten onu analiz edebilecek donanımsal yazılımsal alt yapı da yeterli değildi. Büyük verileri analiz etmek için gerçekten bugünkü gibi gelişmiş bilgisayar donanımlarına ve yazılımlarına gerek vardır.

2) Oluşma Hızı (Velocity)

Büyük verinin diğer bir karakteristik özelliği bunların çok hızlı bir biçimde oluşup duruma göre kaybolmasıdır. Örneğin, 2012 yılı itibarıyla:

- Youtube'a bir dakikada 48 saatlik video yükleniyor.
- Dakikada 571 yeni web sayfası oluşturuluyor.
- Facebook'a her gün 100 terabyte bilgi aktarılıyor. (Bugün kişisel bilgisayarlarımızdaki en büyük diskler 2 terabyte)
- Tweeter'da her gün 175 milyon tweet atılıyor.
- Facebook'ta her ay 30 milyar içerik paylaşılıyor.
- 2014 yılı itibarıyla WhatsApp'ta bir günde 64 milyar mesaj atılıyor.

Bir günde GSM operatörlerindeki mesajları, MR cihazlarında yapılan tetkikleri, GPS'lerden atılan konum bilgilerini, turnikelerden geçenlerin sayılarını, bilet rezervasyonu yapanların oluşturdukları verileri, otoyollara giriş yapan araçları, Google'da aranan içerikleri düşünün… Bu veriler çok hızlı oluşuyor. Bazıları hemen güncelliğini yitiriyor…

3) Çeşitlilik (Variety)

Büyük veriyi oluşturan verilerin formatları çok değişkenlik göstermektedir. Örneğin bu veriler:

- Yazı biçiminde (text) bulunuyor olabilir.
- Resim biçiminde bulunuyor olabilir (bmp, jpeg, gif, pif vs.)
- Video biçiminde bulunuyor olabilir (mpeg4, avi, flv vs.)
- Kompozit biçimde bulunyor olabilir (pdf, doc vss.)
- Binary biçimde bulunuyor olabilir.

Büyük verinin bu çeşitliliği onun ilişkisel veritabanlarında saklanmasını zorlaştırabilir ve işlenmesini karmaşık hale getirebilir. Bu nedenle büyük verilerin işlenmesinde yoğun olarak başka veritabanı modelleri kullanılmaktadır. Büyük verilerin depolanmasında en çok tercih edilen veritabanı modeli NoSQL'dir. En çok kullanılan NoSql tarzı veritabanı yazılımlarından biri MongoDB'dir. MongoDB kursumuzun konuları içerisindedir.

4) Doğruluk (Veracity)

Büyük veriyi oluşturan veriler yanlı (biased) olabilir, bozulmuş (abnormal) olabilir, gürültü (noise) içeriyor olabilir. Çok miktarda verinin hızlı bir biçimde toplanması sırasında verilerin bir kısmının sağlıklı olmayabileceği veri işlemesi sırasında akılda tutulmalıdır.

5) Geçerlilik (Validity)

Büyük veriyi oluşturan verilerin geçerli ve tam olup olmadığı duruma göre değişebilmektedir.

6) Kalıcılık (Volatility)

Büyük veriyi oluşturan verilerin büyük kısmı kalıcı değildir. İlgili sistem tarafından oluşturulur. O andaki gereksinimi gerçek zamanlı olarak karşılar ve önemini yitirir. Önemini yitiren veriler tamamen çöp haline gelmez. Onlar analiz edilerek çeşitli korelasyonlar, bağıntılar ve içgörüler elde edilebilir. Geçmiş veri geleceğe ışık tutabilir.

2. Büyük Veri Analizi (Big Data Analytics) Nedir?

Büyük verinin analizi bilgi parçalarının içindeki ve arasındaki ilişkileri görmek ve anlamakla ilgilidir. Büyük veri analizinde nedensellik yerine korelasyonlar aranır. Yani neden yerine ne yakalanmaya çalışılmaktadır. Örneğin ilk büyük veri uygulamalarından biri olan Farecast, uçak biletininin en ucuza alınabileceği zamanı kestirmeye çalışıyordu. Bu sistem uçak bilet fiyatlarının neden ve nasıl dalgalandığını açıklamaya çalışmıyordu. Verileri konuşturarak doğrudan kestirimde bulunmayı amaçlıyordu.

Büyük veri incelemesinin en önemli noktalarından biri örnekleme (sampling) yerine tüm verinin (yani ana kütlenin hepsinin) dikkate alınmasıdır. Örnekleme ana kütlenin çok büyük olmasından dolayı onu inceleyecek olanak olmaması nedeniyle başvurulan bir yöntemdir. Artık pek çok durum için örneklemeye gerek kalmamıştır. Artık tüm veri analiz edilebilir.

3. Büyük Veri Analizinde Kullanılan Araçlar ve Platformlar
Büyük veri analizinde pek çok araç ve platform kullanılmaktadır. Burada önemli birkaçı üzerinde duracağız.

Hadoop

Hadoop büyük veri alanında en çok kullanılan platformlardan biridir. Açık kaynak kodludur (dolayısıyla bedavadır). Hadoop büyük veriyi cluster'lara ayırarak dağıtık biçimde (yani birden fazla bilgisayarda) saklayablmektedir. Hadoop dağıtık (distributed) çalışan bir sistemdir ve çok güçlü bir hesaplama gücü vardır. Hadoop esnek bir sistemdir. Hata toleransı yüksektir. Örneğin cluster'ları oluşturan bilgisayarlardan birinde bir sorun çıksa bile sistem çalışmaya devam eder. Hadoop ölçeklenebilir (scalable) bir sistemdir. Ölçeklendirilebilir demek verinin boyutu arttıkça ek cluster'lar oluşturularak sistemde bir değişiklik yapmadan sistemin devamının sağlanması demektir.

R Programlama Dili

R istatistiksel ve matematiksel veri analizi için tasarlanmış «domain specific» bir programlama dilidir. R açık kaynak kodlu (dolayısıyla bedava) bir dil ve ortamdır. R büyük veri analizinde çok kullanılan araçlardan biridir. R ile çok büyük veriler üzerinde arama, karşılaştırma gibi işlemler hızlı bir biçimde yapılabilmektedir. R'ın onlarca paketi vardır. Bu paketler sayesinde her türlü analizler, istatistiksel hipotez testleri, yapısal eşitlik modellemesi vs. yapılabilmektedir.

Python Programlama Dili

Son zamanlarda özellikle R gibi ortamların yerini python programlama dili almıştır. Python programlama dili ile kullanılabilen pek çok istatistiksel ve büyük veri üzerinde işlemler yapılabilen kütüphaneler bulunmaktadır. Ayrıca "yapay zeka" ve "makine öğrenmesi" gibi konular için de tasarlanmış çok fazla kütüphane bulunmaktadır.

Google Big Query

Big Query ile petabyte'lar mertebesinde veriler üzerinde çok hızlı analizler yapılabilmektedir. Platform Google'a aittir ve belli bir ücret ödenerek kullanılabilmektedir. Sistemi kullanmak için bir veritabanı yöneticisine gereksinim yoktur. Sistemi kısa bir eğitimle kullanmak mümkündür.

Amazon Web Services

Amazon firması tarafından geliştirilmiş çok yönlü bir platformdur. Amazon Web Services ücretlidir. Arka planda Hadoop'u da kullanmaktadır. Yüksek hızlı, ölçeklenebilir bir analiz platformu sunmaktadır. Makine öğrenmesi (machine learning) için de araçları bulunmaktadır.

MongoDB Veritabanı Yönetim Sistemi

Büyük verilerin tutulması için kullanılan "NoSql" yaklaşımlı veritabanı yönetim sistemidir. MongoDB temelinde Javascript ile geliştirme yapılabilen bir uygulamadır.

mongod programı ile mongodb server başlatılabilir:

mongod programının --dbpath komut satırı argümanı ile herhangi bir dizinde mongodb başlatılabilir.

mongod - -dbpath=.

Hiç bir port numarası ve ip numarası verilmezse localhost (127.0.0.1) ve 27017 olarak başlatılır. mongod programına --port seçeneği ile port numarası belirlenebilir. mongod programında --bind_ip_all seçeneği ile o an çalışılan sistemdeki tüm network kartlarından yani ip numaralarından erişim gerçekleşek şekilde konfigürasyon belirlenebilir.

mongod --dbpath="." --port 27018 --bind_ip_all

Herhangi bir ip adresi bind edilmezse o sistem dışından erişim yapılamaz. Yani localhost ya da diğer adıyla 127.0.0.1 numaralı ip bind edilmiş olur.

mongo programı ile istenilen bir mongodb server'a erişilebilir. mongo tek başına kullanıldığında localhost:27017 erişimini arar. mongo programında --port ve --host seçenekleri ile port numarası ve/veya ip numarası verilebilir:

mongo - -host 192.168.2.238 - - port 40000

Server üzerindeki tüm veritabanları aşağıdaki gibi görüntülenebilir:

db.adminCommand("listDatabases")

Bu işlem console üzerinde show dbs yazılarak da yapılabilir. “show dbs” db.adminCommand("listDatabases") kadar detaylı bilgi vermez.

mongodb server üzerinde hiçbir veritabanı belirlemesi yapılmazsa test isimli bir veritabanı üzerinde çalışılmış olur. İstenilen bir veritabanı için use komutu kullanılabilir. use komutu kullanıldığında veritabanı yaratılmamış olsa bile hemen yaratılmaz. Veritabanı üzerinde ilk kez işlem yapıldığında yaratılır. O an çalışılan veritabanı console üzerinde db yazılarak görüntülenebilir

Bir veritabanı üzerinde adeta bir tabloya karşılık gelen collection'lar bulunmaktadır. Bir collection içerisinde BSON (Binary JSON) formatında bilgi tutulur. Dolayısıya collection' a bilgiler JSON formatında eklenmelidir.

MongoDB Veri Türleri

Mongo Shell Veri Türlerine ilişkin sınıflar

MongoDB Veritabanı İşlemleri (CRUD)

Veritabanı üzerinde işlem yapan fonksiyonlar collection’ların elemanlarıdır.

Veri eklemek için insertOne fonksiyonu kullanılabilir. insert fonksiyonu ile bir JSON data verilmelidir:

db.sensors.insertOne({"name":"test", "data":23.4, "date":"21-03-2019 09:58:45.789"})

Her eklenen veri için bir ObjectId üretilir. Bu ObjectId bir collection içerisinde her veri için tekil (unique) bir bilgidir. Üretilen ObjectId _id isimli bir elemana atanır. _id elemanı manuel olarak da verilebilir. Bu elemanın bir collection içerisinde tekil (unique) olması gerekmektedir.

Anahtar Notlar: insert fonksiyonu "deprecated" olduğundan yerine insertOne fonksiyonu kullanılmalıdır.

Sorgulama İşlemleri

MongoDB üzerinde sorgulama işlemleri için find fonksiyonu kullanılabilir. find fonksiyonu arama işlemi için de JSON formatı istemektedir. Örneğin sensors collection'ı içerisindeki tüm veriler aşağıdaki gibi elde edilebilir:

```
db.sensors.find({})
```

ya da

```
db.sensors.find()
```

pretty fonksiyonu ile elde edilen JSON formatı daha düzgün görüntülenebilmektedir:

```
db.sensors.find().pretty()
```

find fonksiyonuna JSON formatı ile koşullar yazılabilir. Örneğin:

```
db.sensors.find({name:"test"})
```

find fonksiyonuna verilen JSON formatlarında kullanılabilecek bazı operatörler vardır. Operatör isminden önce $ ile kullanılmalıdır. Örneğin data bilgisi 30 değerinden büyük olan verileri getiren sorgu:

```
db.sensors.find({data:{$gt:30}}
```

Birden fazla operatör find içerisinde aynı alan için kullanılabilir:

```
db.sensors.find({data:{$gt:20, $lt:40}})
```

in operatörü ile select-in sorgulamasına benzer bir işlem yapılabilir:

```
db.sensors.find({name:{$in:["test", "mest"]}, data:{$gt:30}})
```

or operatörü ile "veya" mantıksal işlemine ilişkin sorgular yazılabilir:

```
db.sensors.find({$or:[{data:56}, {data:24}, {data:{$gt:20}}]})
```

and operatörü ile "ve" mantıksal işlemine ilişkin sorgular yazılabilir:

```
db.sensors.find({$and:[{data:56}, {name:"test"}]})
```

Bilindiği gibi ilişkisel veritabanı yönetim sistemlerinde ilişki kavramı primary key ve foreign key kullanılarak yapılabilir. MongoDB'de primary key kavramı vardır. Bu da _id elemanıdır. Ancak foreign key kavramı yoktur. İlişki oluşturmak istenirse collection içerisindeki elemana veriler bir dizi biçiminde eklenebilir. Aşağıdaki örnekte *mantıksal* "one to many" ilişkisi oluşturulmuştur.

```
db.sensors.insertOne({name:"test", data:[{val:23, date:"12-03-2019 00:00:00.0"}, {val:26, date:"12-03-2019 12:00:00.0"}, {val:34, date:"13-03-2019 12:00:00.0"}]})
```

Aşağıdaki gibi eklenen JSON bilgide içiçe sorgular yapılabilmektedir:

```
db.sensors.insertOne({name:"jest", data:{val:234, description:"size", unit:"cm"}})
```

“data” içerisindeki “val” değeri 45 den büyük olan tüm sensörleri getiren sorgu:

```
db.sensors.find({"data.val": {$gt:45}})
```

Örneğin, birimi cm olan tüm sensörlere ilişkin bilgileri getiren sorgu:

```
db.sensors.find({"data.unit": "cm"})
```

Yukarıdaki örneklerin herbirinde JSON formatına ilişkin tüm verilerin tüm alanları gelmektedir.

İsterse programcı dilediği alanları alabilecek şekilde (projection) sorgulama yapabilir:

```
db.sensors.find({}, {data:1, name:1})
```

Burada gelmesi istenen elemanlar için 1 konmalıdır. Ancak _id elemanının default değeri 1 dir. Bu elemanın gelmesi istenmiyorsa, ”gelmesin” bilgisi yani sıfır eklenmelidir:

```
db.sensors.find({},{data:1, name:1, _id:0})
```

Belirli elemanların gelmemesi ancak diğerlerinin gelmesi isteniyorsa bu durumda gelmesi istenmeyen elemanlar için sıfır yazılmalıdır. Ancak _id değerinin default değeri 1 olmasına

karşın yazılmaması problem oluşturmaz:

```
db.sensors.find({},{date:0, _id:0})
```

_id dışındaki elemanlar için bir projeksiyonda hem bir hem de sıfır değerinde olan elemanlar yazılamaz:

```
db.sensors.find({},{date:0, data:1, _id:0}) //error
```

Güncelleme için updateOne fonksiyonu kullanılabilir:

```
db.sensors.updateOne({name:"test"}, {$set:{data:566, name:"jest"}})
```

Burada test isimli bulunan ilk sensorün data bilgisini ve isim bilgisini değiştiren JSON verilmiştir.

upadateMany isimli fonksiyon çağrılarak birden fazla veri üzerinde güncelleme yapılabilmektedir.

```
db.sensors.updateMany({data:{ $lt:35}}, {$set: {data:2367}})
```

Burada updateMany ile koşula uyan tüm veriler güncellenir.

Bir verinin BSON formatını değiştirmek için replaceOne isimli fonksiyon kullanılabilir:

```
db.sensors.replaceOne({name:"jest"}, {name:"mest", device_data:345})
```

Anahtar Notlar: *Tüm tekli (One) ve çoklu (many) yazma işlemleri atomiktir.*

updateOne, updateMany ve replaceOne fonksiyonlarına üçüncü argüman olarak verilen JSON formatında upsert:true geçilirse koşula uyan hiç eleman yoksa yenisi yaratılır:

```
db.sensors.updateOne({name:"hest"}, {$set:{data:666, name:"rest"}}, {upsert:true})
```

upsert flag default olarak false değerindedir.

delete işlemi için deleteOne ve deleteMany fonksiyonları kullanılabilir. deleteOne fonksiyonu

update işleminde olduğu gibi koşula uyan ilk veriyi siler. deleteMany fonksiyonu koşula uyan

tüm verileri siler:

```
db.sensors.deleteOne({name:"mest"})

db.sensors.deleteMany({name:"mest"})
```

delete işlemi de atomik olarak yapılmaktadır.

findOneAndDelete fonksiyonu ile sıralama kriteri belirlenerek istenilen koşula uygun ilk

veri silinebilir:

```
db.sensors.findOneAndDelete({name:"test"}, {sort: {data:1}})
```

Yukarıdaki fonksiyon data alanına göre artan sırada sıralanmış test isimli sensörlerden ilkini silmektedir. Sort işleminde sıralanacak alan değeri için 1 yazılması artan (ascending) sırada, -1 yazılması ise azalan (descending) sırada olmasını sağlar.

MongoDB Sorgulama ve Projeksiyon Operatörleri

MongoDB'de bir çok operator bulunmaktadır. Bu operatörler resmi dökümanlara göre çeşitli kategorilere ayrılmıştır:

Eleman Sorgulama Operatörleri

Bu operatörler bir elemanın varlığı ve türüne ilişkin sorgulama yapmak için kullanılmaktadır:

$exists operatörü bir alanın var olup olmadığını test eden operatördür. Örneğin:

```
db.sensors.find( { name:"test", unit: { $exists: true } } )
```

ya da örneğin:

```
db.sensors.find( { name:"test", unit: { $exists: false } } )
```

$type operatörü bir türe göre sorgulama yapmak için kullanılabilir:

```
db.sensors.find( {unit: { $type: "string" } } )
```

Evaluation Query Operators

$expr operatörü bir ifadeye göre sorgulama işlemi yapmakta kullanılabilir. Aşağıdaki örnekte income değeri expense değerinden büyük olan sorgular getirilmektedir:

```
db.budgets.find( { $expr: { $gt: [ "$income" , "$expense" ] } } )
```

$mod operatörü mod alma işlemi için kullanılmaktadır:

```
db.budgets.find( { "income": { $mod: [ 3, 0 ] } } )
```

Dizi Sorgulama Operatörleri

Dizi üzerinde sorgulama yapmaya yönelik opetörlerdir.

Bu operatörler ileride ele alınacaktır.

Güncelleme Operatörleri

Bu operatörler üç gruba ayrılır: Alan güncelleme operatörleri, dizi güncelleme operatörleri, bitsel güncelleme operatörleri:

Alan güncelleme operatörleri bir verinin alanları üzerinde güncelleme yapmak için kullanılmaktadır:

$inc operatörü verilen bir alana ilişkin değeri artırma işlemini yapar:

```
db.products.updateOne( { name: "Mangoes" }, { $inc: { stock: -2 } })
```

$inc operatörü verilen alanlar veride bulunamazsa yeni alan olarak değeri veriye ekler.

$min operatörü verilen alana ilişkin veri ve operator ile verilen veriyi karşılaştırır. Küçük olan ile günceller:

```
db.scores.updateOne( { _id: 1 }, { $min: { lowScore: 150 } } )
```

Bu operator de eğer min ile verilen alan(lar) yoksa yeni alan olarak ekleme yapar.

$max operatörü verilen alana ilişkin veri ve operator ile verilen veriyi karşılaştırır. Büyük olan ile günceller:

```
db.products.updateMany({}, {$max: {unit_price:1000}})
```

$mul operatörü verilen bir alana ilişkin değer ile operatöre ilişkin değeri çarpar:

```
db.products.updateMany({}, {$mul: {unit_price:2}})
```

$mul operatörü verilen alanlar veride bulunamazsa yeni alan olarak değeri veriye ekler.

$rename operatörü bir alanın ismini değiştirmekte kullanılır

```
db.products.updateOne({name:"test"}, {$rename:{"prices.price":"prices.unit_price"}})
```

$set operatörü bir alana ilişkin verinin güncellenmesinde kullanılır.

```
db.products.updateOne({name:"test"}, {$set:{unit_price:500}})
```

Bu operator de eğer set ile verilen alan(lar) yoksa yeni alan olarak ekleme yapar.

$unset operatörü bir alanı silmek için kullanılabilir.

```
db.products.updateOne({name:"test"}, {$unset: {price: ""}})
```

Dizi Güncelleme Operatörleri

Dizi güncelleme operatörleri dizi elemanları üzerinde işlem yapmakta kullanılır.

$ operatörü bir dizi içerisindeki bir elemanın güncellenmesi için aşağıdaki gibi kullanılabilir:

```
db.players.updateOne({name:"insomnia",  score:300}, {$set: {"score.$":500}})
```

Burada score dizisi içerisindeki 300 olan veri 500 ile değiştirilmiştir.

$pop operatörü dizinin içerisinde 1 veya -1 değerine göre dizinin sonundan ya da başından eleman silmek için kullanılabilir:

```
db.players.updateOne({name:"insomnia"}, {$pop:{score:-1}}) //başındaki elemanı at

db.players.updateOne({name:"insomnia"}, {$pop:{score:1}})  //sonundaki elemanı at
```

$pull operatörü verilen koşula uyan elemanları diziden silmek için kullanılabilir.

```
db.players.updateOne({name:"insomnia"}, {$pull: {score:{$lte:80}}})
```

$push operatörü dizinin sonuna eleman eklemek için kullanılabilir:

```
db.players.updateOne({name:"insomnia"}, {$push: {score:45}})
```

Aşağıdaki örnekte dizinin elemanı olarak başka bir dizi eklenmiştir.

```
db.players.updateOne({name:"insomnia"}, {$push: {score:[34, 56]}})
```

$each operatörü ile bir diziye eklenecek elemanlar bir dizi biçiminde verilebilir

```
db.players.updateOne({name:"insomnia"}, {$push: {score: {$each:[100, 123, 45]}}})
```

$pullAll operatörü verilen tüm elemanları silmek için kullanılır:

```
db.players.updateOne({name:"insomnia"}, {$pullAll:{score:[100, 233, 45]}})
```

Bu operatörde pull operatörü gibi bir koşul değil, elemanlar verilir.

$position operatörü ile herhangi bir konumdaki elemana ekleme yapılıp diğerleri kaydırılabilir. Burada position sıfırdan başlar:

```
db.players.updateOne({name:"insomnia"}, {$push: {score:{$each:[34, 67, 89], $position:1}}})
```

position operatörü kullanımı kaydırma işlemi yaptığından göreceli yavaştır. Hızın önemli olduğu uygulamalarda dikkatli olunmalıdır. $position operatörü için negatif index numaraları dizinin eleman sayısı ile toplandığında elde edilen indeks numarasıdır:

```
db.players.updateOne({name:"insomnia"}, {$push: {score:{$each:[34, 67, 89], $position:-1}}})
```

$sort operatörü 1 veya -1 değerine göre diziyi artan sırada veya azalan sırada olacak şekilde dizer:

```
db.players.updateOne({name:"insomnia"}, {$push: {score :{$each:[23, 45], $sort:1}}})
```

Kütle (bulk) Yazma İşlemleri

MongoDB'de kütle yazma işlemleri yapılabilmektedir. Kütle yazma işlemleri aynı anda birden fazla işlemi gerçekleştirmektir. Yani, ekleme silme ve güncelleme işlemleri aynı fonksiyon ile yapılabilmektedir. bulkWrite fonksiyonu ile hangi fonksiyonla işlem yapılacağı yine JSON formatında verilebilmektedir:

```
db.users.bulkWrite([
{
        insertOne:
        {
                "document":
                {"username":"kaan", "password":"1234567"}
        }
},
{updateOne: {"filter": {"char":"Brisbane"}, "update": {$set: {class:"xxx"}}}}])
```

Text Arama işlemleri

Bu işlemler için text operatörü kullanılır. text operatörü index olmadan geçersizdir. Index yaratmak için createIndex fonksiyonu çağrılmalıdır. text operatorü için gereken index "text" ismiyle verilmelidir. Bir collection içerisinde "text" indeksinden bir tane yaratılabilir. Indeks yaratılma noktasında birden fazla alan için verilebilir:

```
db.sensors.createIndex({name:"text", description:"text"})
```

Burada artık name ve description alanları için text arama yapılabilir:

```
db.sensors.find( { $text: { $search: "Hava nem" } } )
```

Burada indekslenen alanlar içerisinde Hava veya nem geçen tüm veriler getirilecektir. Yine find fonksiyonunun özellikleri ayrı parametrelerle kullanılabilmektedir. İçerisinde boşluk (whitespace) geçen bir yazı aranıyorsa iki tırnak içerisine alınmalıdır:

```
db.users.find( { $text: { $search: "\"Oğuz Karan\"" } } )
```

Burada içerisinde Oğuz veya Karan geçen değil sadece Oğuz Karan geçen veriler getirilecektir. İki tırnak karakteri aşağıdaki gibi de kullanılabilir:

```
db.users.find( { $text: { $search: '"Oğuz Karan"' } } )
```

Arama işleminde bulunursa elde edilmesini istemediğimiz kelimeleri başına "-" işareti koyarak belirleyebiliriz:

```
db.users.find( { $text: { $search: "Oğuz -Karan" } } )
```

Verilen yazılarda AND işlemi için iki tırnak içerisine alınarak yapılabilir. Aşağıdaki örnekte veri içerisinde hem Oğuz hem de calculus geçen veriler elde edilebilir:

```
db.users.find( { $text: { $search: '"Oğuz" "calculus"'} }  )
```

Bu örnekte iki tırnak içerisine alındığından "mantıksal ve" işlemi yapılır. İki tırnak içerisine alınması zorunlu olan ancak veya işlemi yapılması durumunda aşağıdaki syntax uygulanmalıdır:

```
db.users.find( { $text: { $search: "('Oğuz')  ('calculus')"} } )
```

Burda parantez içerisinde bulunan yazılarda boşluk olabilmektedir:

```
db.users.find( { $text: { $search: "('Oğuz Karan')  ('calculus')"} } )
```

Mongo Shell Üzerinde Script Yazımı

Mongo shell üzerinde JS ile veriler üzerinde işlem yapan veya yönetimsel işlemleri içeren programlar yazılabilmektedir. Aslında Mongo shell ile veriler üzerinde veya yönetimsel işlemler yapmadan da JS kodu yazılabilmektedir:

```
for (let i = 0; i < 10; ++i) {
        print(`${parseInt(Math.random() * 100)}`)
}
```

Mongo nesnesi kullanılarak veritabanına bağlantı sağlanabilmektedir:

```
connection = new Mongo() //mongo
```

Elde edilen referans ile db referansı getDB fonksiyonu ile elde edilebilir:

```
db = connection.getDB("sensorsdb") //use sensorsdb
```

Bu işlem ile o an kullanılan shell için aktif veritabanı da değiştirilmiş olur. Örneğin:

```
connection = new Mongo()
db = connection.getDB("sensorsdb")
db.sensors.insertOne({name:"test", data:-1})
```

Anahtar Notlar: *Mongo shell üzerinde bir JS dosyası çalştırabilmek için load isimli aşağı seviyeli (native) fonksiyon kullanılabilir. Bu konu ileride ele alınacaktır. Ancak bir shell üzerinde çalışmak yerine mongo uygulaması ile de bir js dosyası çalıştırılabilir:*

```
mongo app.js
```

mongo - -eval seçeneği ile komut satırından bir JS kodu çalıştırılabilir:

```
mongo --eval "db.sensors.insertOne({name:'test', data:-1})"
```

Bu şekilde iki tırnak veya tek tırnak içerisindeki JS kodları mongo üzerinde çalıştırılabilir.

MongoDB Cursor Kullanımı

Mongo shell üzerinde cursor kullanılabilir. find fonksiyonu aslında bir cursor döndürür. Cursor nesnesinin hasNext isimli metodu ile verinin varolup olmadığı test edilir, next metodu ile de verinin kendisi elde edilir. Elde edilen veri BSON formatında olduğundan doğrudan işlem yapılmaması gerekir. Bunun için JSON formatına dönüştürebilecek fonksiyonlar kullanılabilir. Örneğin printjson fonksiyonu ile bir BSON veri JSON formatı olarak ekrana basılabilir. Örneğin:

```
connection = new Mongo()
db = connection.getDB("sensorsdb")
db.sensors.insertOne({name:"ttt", data:34})
cursor = db.sensors.find()
while (cursor.hasNext())
        printjson(cursor.next())
```

Cursor nesnesinin toArray fonksiyonu ile cursor içerisindeki bilgiler diziye dönüştürülebilir:

```
connection = new Mongo() //mongo
db = connection.getDB("sensorsdb") //use sensorsdb
db.sensors.insertOne({name:"ttt", data:34})
cursor = db.sensors.find({}, {_id:0})
sensors = cursor.toArray()
//printjson(sensors)
print("***********")
for (let i = 0; i < sensors.length; ++i) {
        let sensor = sensors[i]

        print(`Name:${sensor.name}, Data:${sensor.data}`)
}
```

Ya da örneğin:

```
connection = new Mongo() //mongo
db = connection.getDB("sensorsdb") //use sensorsdb

db.sensors.insertOne({name:"ttt", data:34})
cursor = db.sensors.find({}, {_id:0})
sensors = cursor.toArray()
for (let i in sensors) {
        let sensor = sensors[i]
```

```
        print(`Name:${sensor.name}, Data:${sensor.data}`)
}
```

Ya da örneğin:

```
connection = new Mongo() //mongo
db = connection.getDB("sensorsdb") //use sensorsdb

db.sensors.insertOne({name:"ttt", data:34})

cursor = db.sensors.find({}, {_id:0})

sensors = cursor.toArray()

for (let sensor of sensors) {
        print(`Name:${sensor.name}, Data:${sensor.data}`)
}
```

Ya da fonksiyonel programlama tekniği de uygulanarak yukarıdaki algoritma aşağıdaki gibi yazılabilir:

```
connection = new Mongo() //mongo
db = connection.getDB("sensorsdb") //use sensorsdb
db.sensors.insertOne({name:"ttt", data:34})
cursor = db.sensors.find({}, {_id:0})
sensors = cursor.toArray().forEach(sensor => print(`Name:${sensor.name}, Data:${sensor.data}`) )
```

Aşağıdaki örnekte bir veritabanından bilgiler ayrık (distinct) olarak okunmuş ve başka bir veritabanına rasgele data üretilerek aktarılmıştır:

```
function connect(dbname)
{
    let connection = new Mongo()
```

```
    let db = connection.getDB(dbname)

    return db
}

function getNameOfAllSensors(db)
{
    return db.sensors.distinct("name")
}

db = connect("sensorsdb")
names = getNameOfAllSensors(db)

db = connect("mysensorsdb")

n = 10

while (n--) {
    let sname = names[parseInt(_rand() * names.length)]
    let sdata = _rand() * 100
    db.sensors.insertOne({name:sname, data:sdata})
}

printjson(db.sensors.find({}, {_id:0}).toArray())
```

Sınıf Çalışması: Hava durumu bilgilerini içeren bir veritabanını MongoDB ile oluşturunuz. Veritabanında aşağıdaki gibi bir collection bulunacaktır:

- place
- city
- country

- latitude
- longitude
- datetime
- degree
- status

- degree bilgileri _rand() fonksiyonu ile üretilsin, lat ve lng bilgileri yine _rand() fonksiyonu ile örneğin Türkiye içerisinde üretilebilir. date bilgisi ve diğerleri aşağıdaki gibi üretilebilir:

```
db.weatherinfo.insertMany([{place:"şişli", city:"istanbul", country:"tr", lat:_rand() * 23, lng:_rand() * 44, datetime:new Date(), degree:12, status:"rainy"}, {place:"florya", city:"istanbul", country:"tr", lat:23.56, lng:44.00, datetime:new Date(), degree:11, status:"rainy"}])
```

- Belirlenen lat ve lng bilgilerine göre hava durumunu derece ve durumu ile place bilgisi gelecek şekilde sorgulayınız

```
db.weatherinfo.find({lat:23.56, lng:44.00}, {_id:0, degree:1, status:1})
```

- Belirlenen bir dereceden daha küçük olan hava durumlarını getiren sorguyu yazınız

```
db.weatherinfo.find({degree:{$lt:12}}, {_id:0, status:1, place:1})
```

Önemli MongoDB Shell Fonksiyonları

MongoDB shell üzerinde birçok fonksiyon bulunmaktadır. Bu fonksiyonlar çeşitli kategorilere ayrılmıştır:

Aşağı Seviyeli Fonksiyonlar (Native Functions)

Bu fonksiyonlar genel olarak Mongo shell dışında kalan işlemler için kullanılmaktadır. Yani örneğin, işletim sistemine özgü fonksiyonlar da bu grubun içerisindedir.

load fonksiyonu

Bu fonksiyon parametresi ile aldığı yol bilgisine ilişkin dosya içerisindeki JS kodlarını o anki shell üzerinde çalıştırır:

```
load("app.js")
```

_rand Fonksiyonu

Bu fonksiyon [0, 1) aralığında rasgele sayı üretmek için kullanılabilir:

```
for (let i = 0; i < 5; ++i)
    print(`${_rand()}`)
```

_rand fonskiyonu JS'nin Math nesnesinin random fonksiyonu ile aynıdır. Aşağıdaki örneği inceleyiniz:

```
sensorNames = ["test", "mest", "huminity", "temperature"]

connection = new Mongo()
db = connection.getDB("sensorstestdb")

for (let i = 0; i < 10; ++i) {
        let sname = sensorNames[parseInt(_rand() * sensorNames.length)]
        let value = _rand() * 100

        db.sensors.insertOne({name:sname, data:value})
}

cursor = db.sensors.find()

while (cursor.hasNext())
        printjson(cursor.next())
```

Cursor yerine aşağıdaki bir döngü de yazılabilir:

```
for (let val of db.sensors.find({}, {_id:0}).toArray())
        printjson(val)
```

version fonksiyonu

O an çalışılan mongodb version bilgisini döndürmektedir:

```
function connect(dbname)
{
        let connection = new Mongo()
        let db = connection.getDB(dbname)

        return db
}
```

db = connect("testdb")

n = 10

while (n--)

db.testdata.insertOne({name:"test", data:_rand(), version:version()})

pwd fonksiyonu

MongoDB client uygulamanın çalışma dizini bilgisini döndürür:

_isWindows Fonksiyonu

MongoDB shell'in çalıştığı işletim sistemi Windows ise bu fonksiyon true döndürür.:

```
function connect(dbname)
{
    let connection = new Mongo()
    let db = connection.getDB(dbname)

    return db
}
db = connect("testdb")

db.testdata.deleteMany({})
n = 10
while (n--) {
    let curOs = _isWindows() ? "Windows" : "Other"

    db.testdata.insertOne({name:"test", data:_rand(), os:curOs, version:version()})
}
```

mkdir Fonksiyonu

Parametresi ile aldığı yol ifadesine ilişkin dizini (directory) yaratmak için kullanılır. Yol ifadesinde bulunan ara dizinler yoksa yaratılır. Bu fonksiyon işletim sistemleri için default erişimle dizinleri yaratır:

mongo --eval "mkdir('./test/mest/jest')"

cat Fonksiyonu

Bir dosyanın içeriğini döndürür. Fonksiyonun ikinci parametresi dosyanın text veya binary modda açılıp açılmayacağına ilişkin flag değeridir. Default değeri false'dur. true değeri binary anlamına gelir. Windows sistemlerinde kullanımı anlamlıdır. Diğer sistemler için true ya da false olmasının bir önemi yoktur.

Anahtar Notlar: *Windows işletim sisteminde bir dosya text ve binary olarak iki gruba ayrılır. Text dosyalarda bir sonraki satırın başına geçmek "carriage return \r" ve "line feed \n" karakterlerinin birlikte olmasıyla gerçekleşir. Ayrıca Windows ortamında tect dosyaların içerisinde Ctrl+Z karaktesi dosyanın sonuna gelindiğini gösterir. Yani bir dosya text modda okunurken Ctrl+Z karakteri görüldüğünde dosya okunmaya devam etmez. Windows dışı işletim sistemlerinde text ve binary dosyalar arasında fark yoktur*

cd Fonksiyonu

Çalışma dizinini (current working directory) değiştirmek için kullanılabilir.

getMemInfo Fonksiyonu

Fonksiyon o an kullanılan shell için bellekte ne kadar yer kapladığını bir JSON veri olarak döndürür.

getHostName ve hostname Fonksiyonları

Fonksiyon shell' in çalıştırıldığı işletim sistemine ilişkin host bilgisini döndürür.

ls Fonksiyonu

Dizin içerisindeki dosyaların isimlerinden oluşan bir dizi döndürür. Aşağıdaki örnekte ls fonksiyonun döndürdüğü dosya isimlerinden oluşan dizinin elemanları ekrana yazdırılmıştır:

```
var list = ls()

for (let name of list)
    print(name)
```

listFiles fonksiyonu

Fonksiyon çalışma dizini içerisinde bulunan dosya ve dizinlere ilişkin detaylı bilgi içeren bir dizi döndürmektedir. Dizinin elemanları BSON formatında elde edilir:

```
for (let fileInfo of listFiles()) {
    if (fileInfo.isDirectory)
        print(`${fileInfo.baseName} <DIR>`)
    else
        print(`${fileInfo.baseName} ${fileInfo.size} bytes`)
}
```

sleep Fonksiyonu

Parametresi ile aldığı milisaniye kadar JS akışını bloke eder:

```
for (let fileInfo of listFiles()) {
    if (fileInfo.isDirectory)
        print(`${fileInfo.baseName} <DIR>`)
    else
        print(`${fileInfo.baseName}`)

    sleep(1000)
}
```

Anahtar Notlar: *Mongo shell üzerinde JS' de standart olarak bulunan setInterval ve setTimeout fonksiyonları desteklenmez. Bu durumda sleep fonksiyonu kullanılmalıdır.*

removeFile Fonksiyonu

Parametresi ile aldığı yol ifadesine ilişkin dosyayı siler.

quit Fonksiyonu

Shell' den çıkış için çağrılan fonksiyondur.

Collection Metotları

Collection'lar üzerinde işlem yapmaya yarayan metotlar bu gruptadır. Bu metotların bir kısmını daha önce görmüştük. Burada geri kalan önemli bazı metotları göreceğiz.

count Metodu

İlgili collection içerisindeki veri sayısını döndüren metottur:

```
db.sensors.count()
```

count metodu ayrıca cursor ile de kullanılabilmektedir:

```
db.sensors.find({name:"xxx"}).count()
```

save Metodu

Bu metot data varsa güncellemek yoksa insert etmek için kullanılabilir:

```
db.sensors.save({name:"xxx", data:1234}
```

çağrısı ile eğer veri yoksa yeni bir veri olarak eklenir.

Bilindiği gibi MongoDB de _id alanı tekil bir bilgidir ve aynı collection içerisinde aynı _id değerinde birden fazla veri olamaz. _id değeri ekleme yapan metotlarda da manuel olarak verilebilir:

```
db.sensors.insertOne({_id:1, name:"xxx", data:78})
```

Bu durumda artık _id için ObjectId üretilmez.

Güncelleme işlemi için _id alanı kullanılabilir:

```
db.sensors.save({_id:1, name:"jest", data:90})
```

Burada _id değeri 1 olan bir veri varsa güncellenecektir. Yoksa yeniden yaratılacaktır.

Burada save fonksiyonu eğer _id ye ilişkin veri varsa o verinin o anki alanlarından farklı alanlar verilmişse replace işlemi yapar:

```
db.sensors.save({_id:1, sensor_name:"jest", data:90})
```

Burada _id değeri 1 olan verinin içerisinde save ile verilen alanlardan en az biri bile yoksa güncelleme işlemi yapılır.

dataSize fonksiyonu

Bu fonksiyon bu collection' a ilişkin verinin kapladığı alanı byte cinsinden verir.

```
db.sensors.dataSize()
```

Aşağıdaki örnekte collection 500 byte' ın üstüne çıkmışsa başka bir veritabanına kopyalayan ve orjinal veritabanından verileri silen küçük bir script yazılmıştır:

```
function connect(dbname)
{
    let connection = new Mongo() //mongo
    let db = connection.getDB(dbname) //use sensorsdb

    return db
}

db = connect("sensorsdb")
dataSize = db.sensors.dataSize()

if (dataSize > 500) {
    let bakDb = connect("sensorsdbbak")
    let cursor = db.sensors.find({})

    while (cursor.hasNext())
        bakDb.sensors.insertOne(cursor.next())

    db.sensors.deleteMany({})
}
```

totalSize fonksiyonu

Bu fonksiyon bir collection' a ilişkin verinin ve index değerleri gibi bir takım diğer verilerin toplam kapladığı alanı byte cinsinden verir.

```
db.sensors.totalSize()
```

Aşağıdaki örnekte periodic olarak bir collection'ın total size bilgisine bakılıp, belli bir büyüklüğe ulaştığında yedeklenmektedir:

```
function connect(dbname)
{
    let connection = new Mongo() //mongo
    let db = connection.getDB(dbname) //use sensorsdb

    return db
}

db = connect("sensors")

while (1) {
    dataSize = db.sensors.totalSize()
    if (dataSize > 45000) {
        let bakDb = connect("sensorsdbbak")
        let cursor = db.sensors.find({})
```

```
        while (cursor.hasNext())
            bakDb.sensors.insertOne(cursor.next())
        db.sensors.deleteMany({})
    }
}
```

Veritabanına ilişkin Metotlar

Bu metotlar ilgili veritabanına yönelik genel bazı işlemleri yapmakta kullanılmaktadır.

cloneCollection metodu

Bu metot bir collection'ı bulunduğu server dan başka server a aynı veritabanı olacak biçiminde kopyalar:

use sensorsdbbak

db.cloneCollection("localhost:27017", "sensors")

Burada hedef server üzerinde sensors isimli bir collection'ın varolmaması gerekir. Aksi durumda klonlama gerçekleşmez. Bu metot kendi veritabanı üzerine kopyalama yapamaz. Bu metot 4.2 sürümünden sonra deprecated olmuştur. Bu metot yerine mongodb ile birlikte gelen çelitli yardımcı programlar kullanılabilir. Bunun için dökümanlara bakılmalıdır. Şüphesiz bir script yazarak programlama yöntemi ile de yapılabilir. Aşağıdaki örnekte cloneCollection metodunun yaptığı işe ilişkin bir script yazılmıştır:

getCollectionNames metodu

Bu metot o anki veritabanı üzerindeki tüm collection isimlerini bir dizi olarak döndürür:

db.getCollectionNames()

Aşağıdaki örnekte tüm collection'ları listeleyen bir script yazılmıştır:

```
function connect(dbname)
{
    let connection = new Mongo() //mongo
    let db = connection.getDB(dbname) //use sensorsdb

    return db
}

db = connect("sensorsdb")

for (let collection of db.getCollectionNames())
    print(collection)
```

getCollection metodu

Parametresi ile aldığı collection ismine ilişkin collection nesnesini döndürür.

Aşağıdaki örnekte bir veritabanına ilişkin tüm collection'ların size'ları başka bir veritabanına kopyalanmaktadır:

```
function connect(dbname)
```

```
{
    let connection = new Mongo() //mongo
    let db = connection.getDB(dbname) //use sensorsdb
    return db
}

db = connect("sensorsdb")

collections = db.getCollectionNames()

sizeDb = connect("sensorssizedb")

for (let collName of collections) {
    let collection = db.getCollection(collName)
    let collSize = collection.dataSize()
    sizeDb.sizes.insertOne({name:collName, size:collSize, date:new Date()})
}
```

Yukarıdaki örnek başka server' a kaydedilecek şekilde de yapılabilir:

```
function connect(url, dbname)
{
    let connection = url == "" ? new Mongo() : new Mongo(url)
    let db = connection.getDB(dbname)

    return db
}

function connectTo(host, port, dbname)
{
    let url = host + ":" + port

    return connect(url, dbname)
}

db = connect("","sensorsdb")

collections = db.getCollectionNames()

sizeDb = connectTo("localhost", 27018, "sensorssizedb")

for (let collName of collections) {
    let collection = db.getCollection(collName)
    let collSize = collection.dataSize()
    sizeDb.sizes.insertOne({name:collName, size:collSize, date:new Date()})
}
```

getName metodu

Bu metot ile o anki aktif veritabanı ismi elde edilebilir:

```
db.getName()
```

MongoDB Kullanıcı işlemleri

MongoDB üzerinde yetkilendirme yapılabilmektedir. Bu şlem için veritabanı fonksiyonları da bulunmaktadır. Ayrıca mongo shell üzerinde de bu işlem yapılabilmektedir. MongoDB üzerinde admin denilen bir veritabanı bulunur. Mongo server - -auth seçeneği ile başlatılmazsa herhangi bir yetkilendirme kullanılmaz. Yetkilendirme için client ilgili bilgileri bağlanırken verebilir. - -auth seçeneği ile başlatılması durumunda admin olan kişi için mongo server doğrudan da erişilebilirdir. Öyleyse bir kullanıcı çok basit olarak şu şekilde oluşturulabilir:

- mongod programı ile server başlatılır. Burada başlama kısmının ilk durumda - -auth başlatılmaması uygundur
- mongod ile başlatılan server' a bağlantı sağlanır.
- Admin veritabanı için createUser fonksiyonu çağrılır:

  db.createUser({user:"oguz", pwd:"csd1993", roles: [{role:"read", db:"sensorsdb"}]})

  Bu şekilde yaratılan kullanıcı veritabanı üzerinde yalnızca okuma işlemi yapabilir. Hem okuma hem yazma için "readWrite" rolü kullanılabilir. Ayrıca başka roller de belirlenebilir. Bu işlem sırasında eklenecek kullanıcı bilgileri için uygun veritabanı belirlenmelidir.

- Mongo server bu işlemlerden sonra kapatılarak uygun yerde - -auth seçeneği ile başlatılmalıdır.
- Client olarak erişim önceden "user name" ve "password" verilerek yapılabilir. Ya da önce bağlanılır sonradan yetkilendirme kullanılır. mongo programı ile username ve password bağlantısı şu şekilde yapılabilir:

  mongo --host 192.168.2.104 -u "oguz" -p "csd1993" --authenticationDatabase "admin"

  burada authenticationDatabase seçeneği ile verilen veritabanı bu kullanıcının tutulduğu veritabanını belirtir.

  Veritabanına bağlandıktan sonra yetki alımı auth isimli fonksiyon çağrılarak da yapılabilir:

  use admin

  db.auth("oguz", "csd1993")

  Bu işlemden sonra yetkisi bulunduğu veritabanına geçilerek işlemler yapılabilir.

- Admin rolü yaratabilmek için createUser metodu aşağıdaki gibi çağrılabilir:

  db.createUser({user:"myadmin", pwd:"csd1993", roles: [{role:"userAdminAnyDatabase", db:"admin"}]})

  Burada yaratılan myadmin kullanıcısı kullanıcı yaratma hakkına sahiptir.

Transaction işlemleri

MongoDB üzerinde transaction işlemleri için çeşitli fonksiyonlar kullanılmaktadır:

```
function connect(dbname)
{
    let connection = new Mongo() //mongo
    let db = connection.getDB(dbname) //use sensorsdb
    return db
```

```
}

db = connect("sensorsdb")

session = db.getMongo().startSession({readReference: {mode:"primary"}})

collection = session.getDatabase("sensorsdb").sensors
session.startTransaction({readConcern: {level:"snapshot"}},  {writeConcern: {w:"majority"}})

try {
    collection.insertOne({_id:85, name:"test", data:34})
    collection.insertOne({_id:67, name:"test", data:34})
    session.commitTransaction()
}
catch (err)  {
    session.abortTransaction() //roll back
    session.endSession()
    throw err
}

session.endSession()
```

Bu uygulama bir iskelet gibi düşünülebilir.

Anahtar Notlar: *Transaction işlemleri için MongoDB'nin bazı kuralları vardır. Örneğin, bir mongo server -- replSet ile başlatılmalıdır ve mongo client program da rs.initiate isimli fonksiyon çağrılarak transaction işlemi yapılabilir.*

mongo --replSet "rs0" - -dbpath=.

Transaction işlemler mümkün olduğunca çabuk bir biçimde sonlandırılmalıdır.