



No-SQL MongoDB

Haluk Bilgin

JAVA – Backend Developer



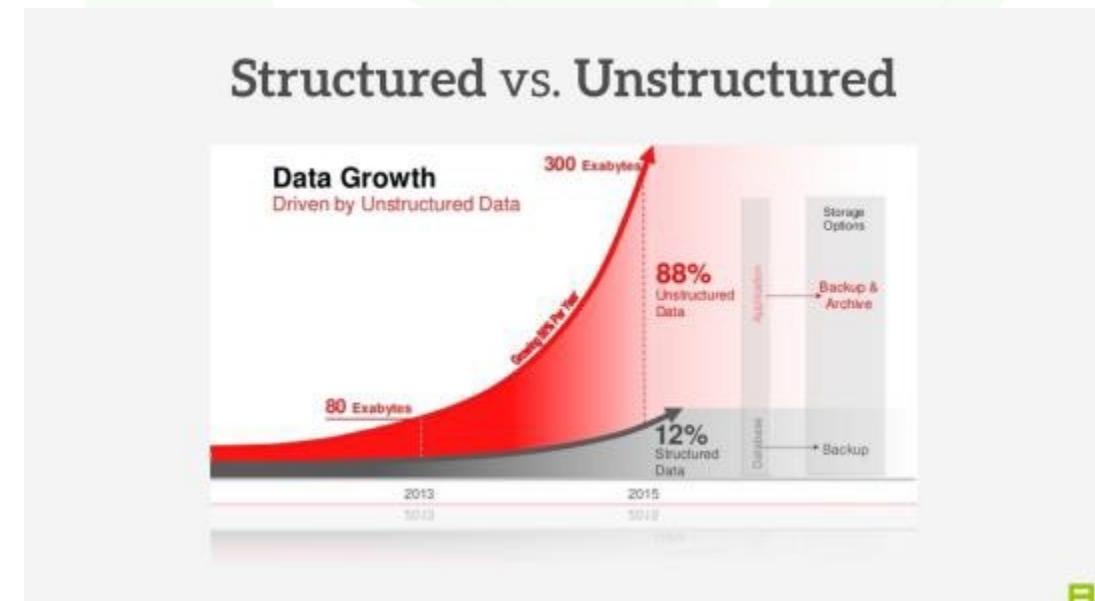
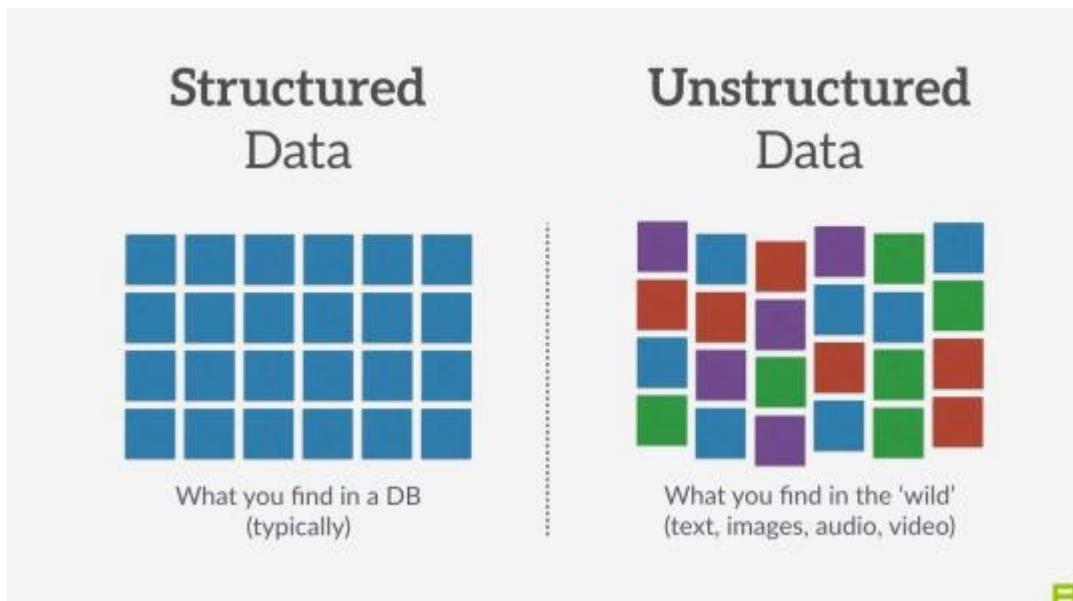
Temel Kavramlar ve Kısaltmaları

- RDBMS : Relational Database Management System
- ACID : Atomicity – Consistency – Isolation – Durability
- CAP : Consistency, Availability, Partition Tolerance
- SQL : Structured Query Language
- UnQL : Unstructured Data Query Language
- DB : Database
- JSON : JavaScript Object Notation



Yapılandırılmış Veri ... Yapılandırılmamış Veri

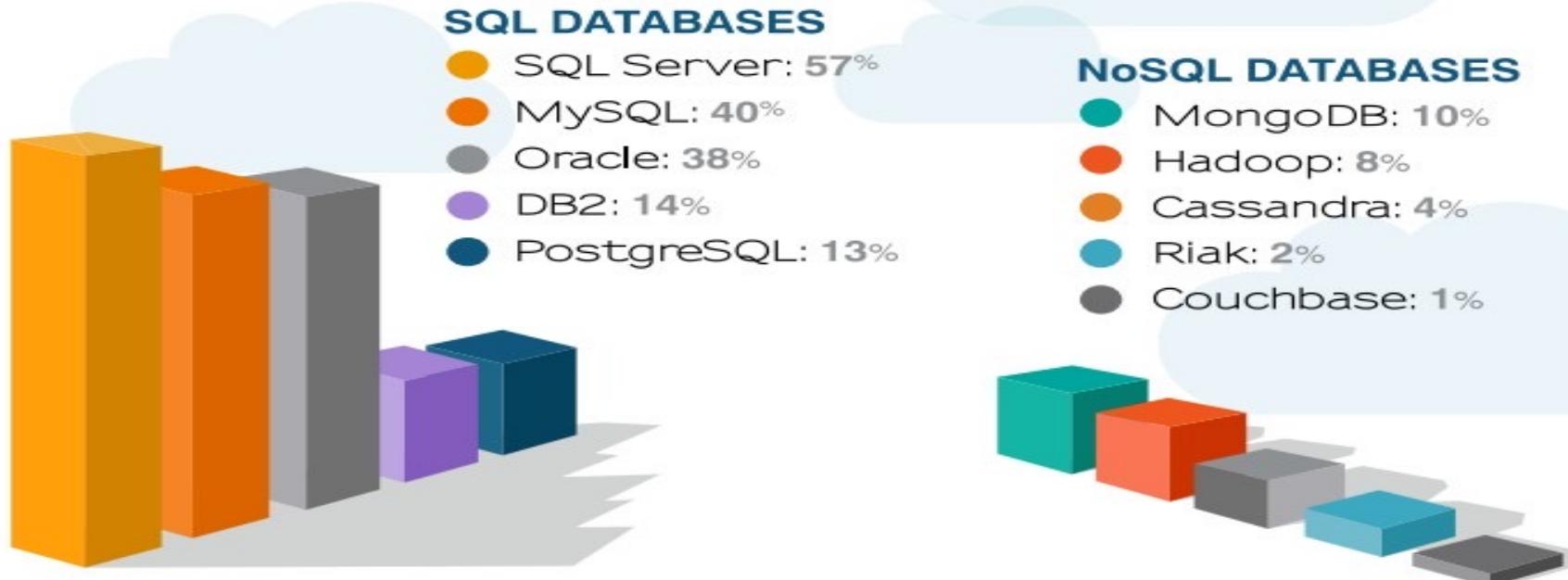
- Büyük veri, yapılandırılmış veya yapılandırılmamış veri şeklinde olabilir.
- Yapılandırılmış verilerde her satırdaki sütun sayısı sabitken yapılandırılmamış verilerde her bir satırda farklı sayıda sütun olabilir.



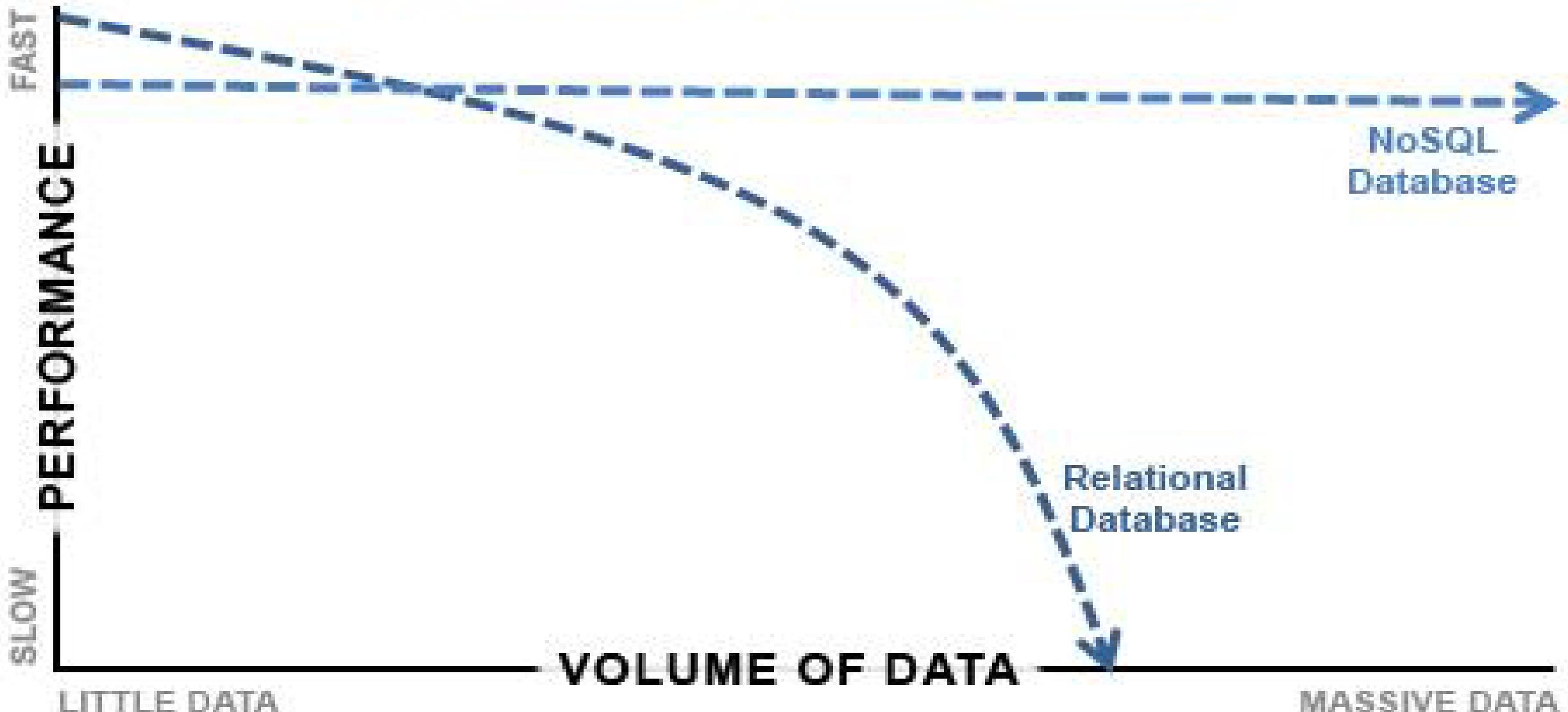


NoSQL -SQL

- NoSQL ilişkisel (RDBMS) (Relational Database Management System) veri tabanlarına alternatif olarak çıkan, nesneyi dikkate alan SQL'deki gibi belirli kolonlara ihtiyaç duymayan veri depolama yaklaşımıdır.
- NoSQL dendğinde bilinen Hadoop, MongoDB, ElasticSearch vb. gibi birçok marka bulunmaktadır.
- Bu ders kapsamında, yaygın kullanımı nedeniyle NoSQL veri tabanı olarak MongoDB tercih edilmiştir.



Scalability of NoSQL Database vs Traditional Relational Database



Büyük veri, tek bir sunucu üzerinden yönetilemeyen, çok sayıda kullanıcının eş zamanlı olarak işlem yaptığı verilerdir.

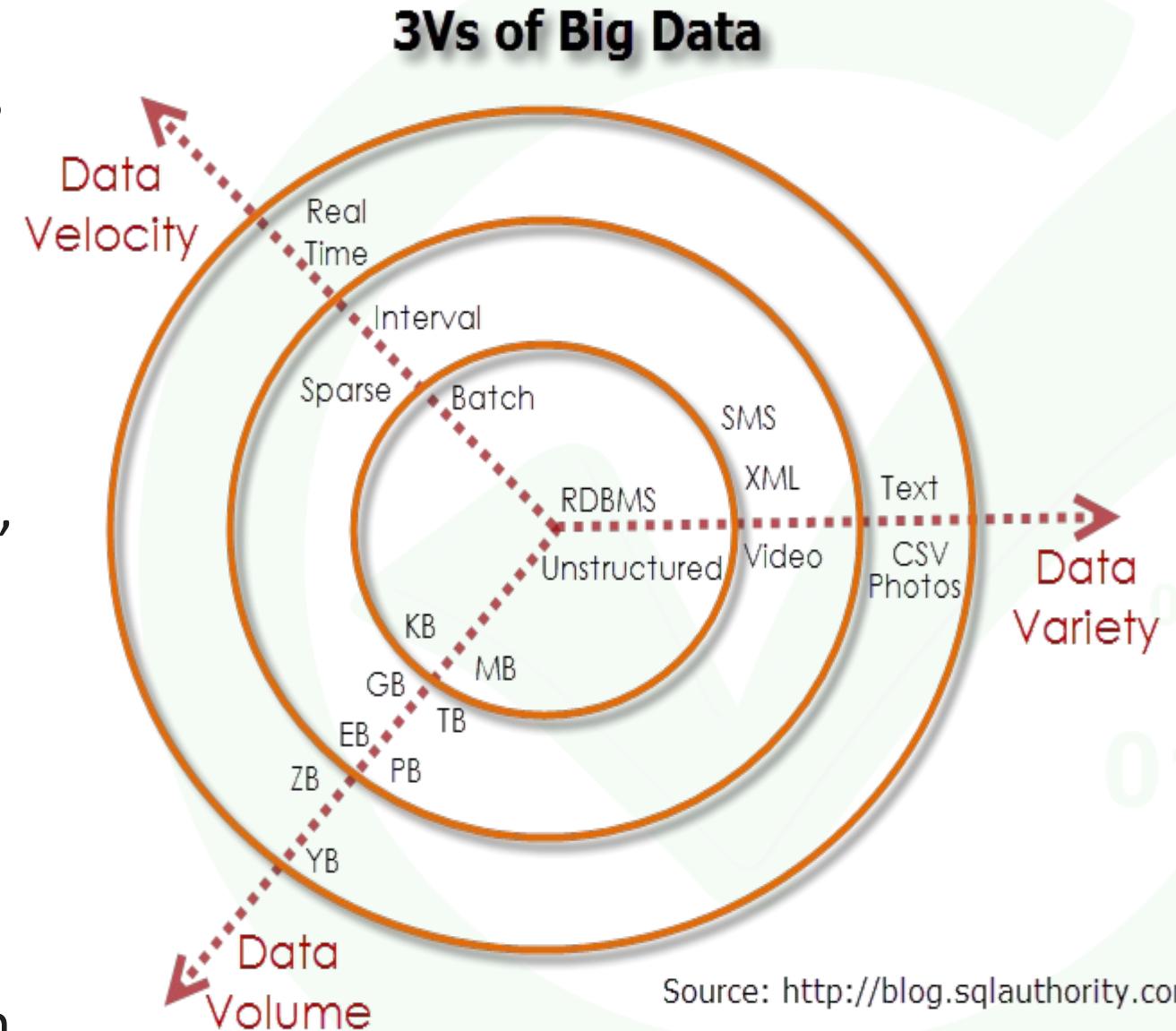
Bir veriye, büyük veri diyebilmek için üç parametreye bakılır:

Volume: Verinin **hacimsel** büyüklüğü

Velocity: Veri akışının **hızı** (sensör verileri, finansal işlemler, uygulama kullanım bilgileri -log kayıtları- gibi)

Variety: Verinin farklı **formatlarda** gelebilmesi

NoSQL veritabanları, bu üç özelliğe sahip verilerin işlenmesini sağlayan dağıtık, ölçeklendirilebilir ve yüksek erişime (high availability) sahip veritabanlarıdır.



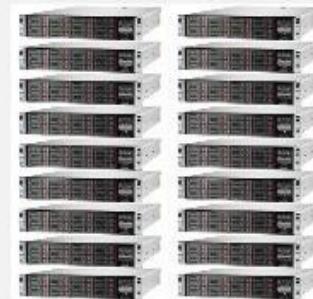
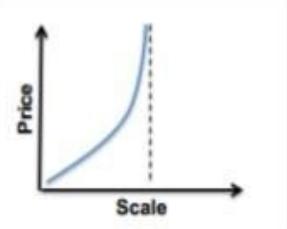
Source: <http://blog.sqlauthority.com>

NoSQL yapıları ilişkisel veri tabanlarının performans sorunlarının yanında lisans ve işletme maliyetlerinin de karşısına çıkmıştır.

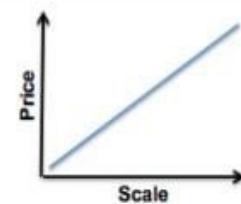
Scale-out vs Scale-up



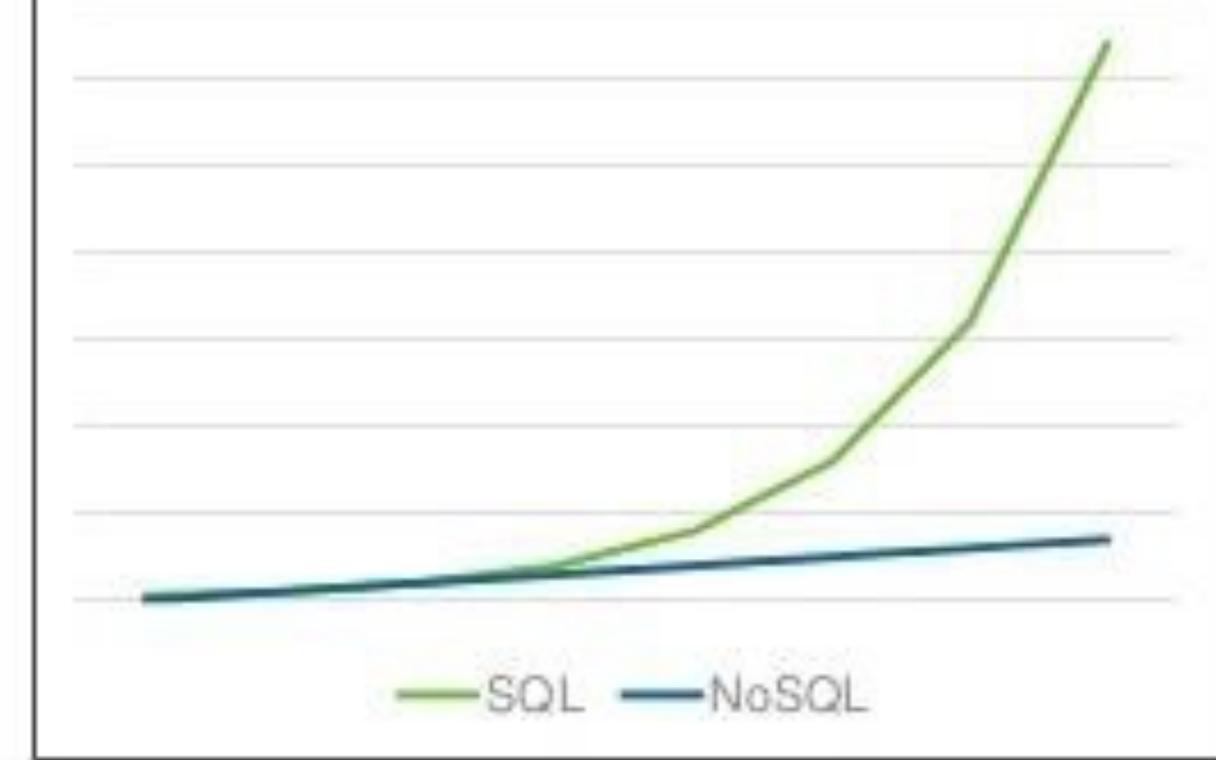
Then: Scale Up



Now: Scale Out



Cost of scaling



Scale-up



Scale-out



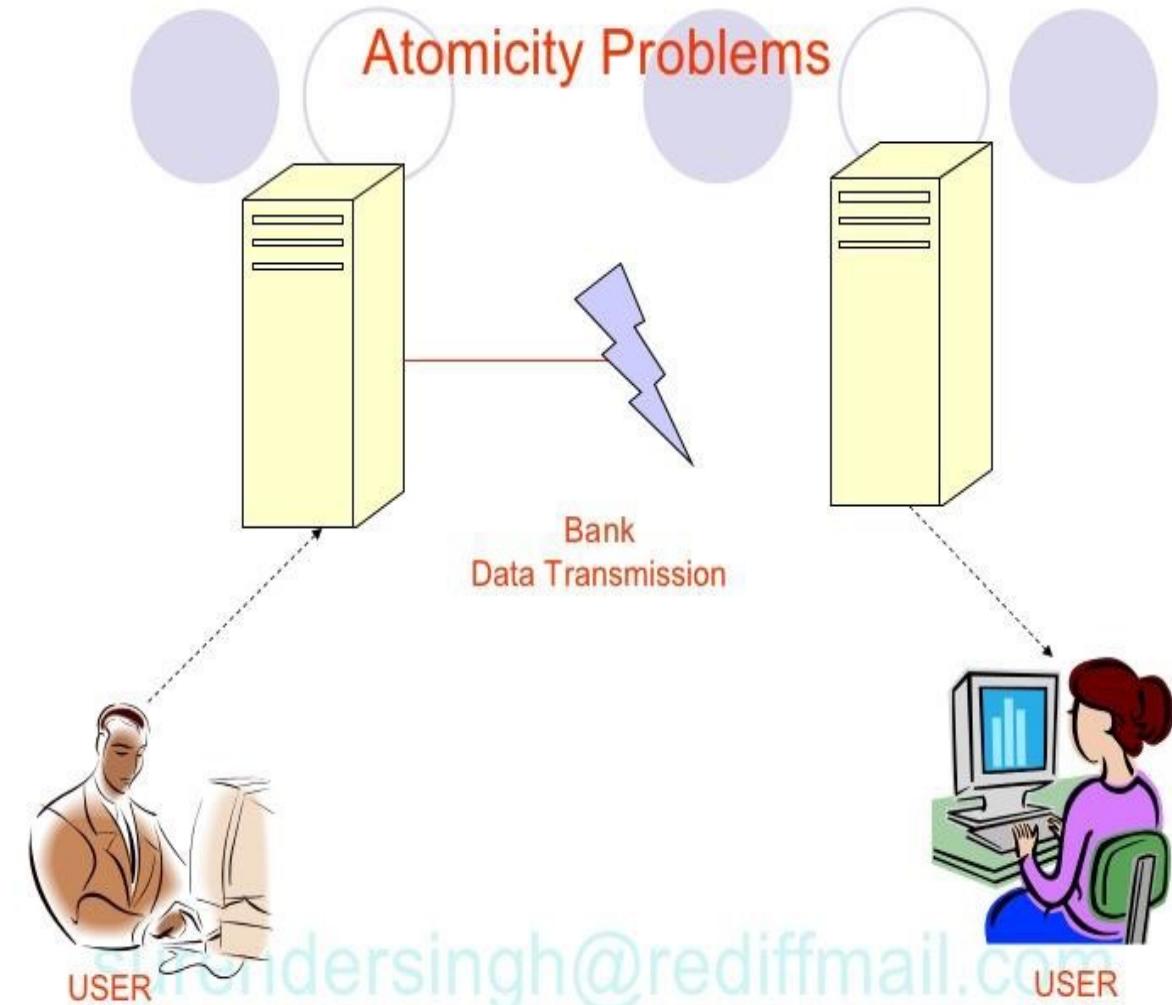
- NoSQL ACID garantisi vermeyen dağıtık veri depolarını dikkate alarak yola çıkmıştır.
- Bir transaction oluşabilmesi için ACID ilkelerine uyması gereklidir.
- ACID, Atomicity –Consistency – Isolation – Durability
- (bölgünmezlik– tutarlılık - izolasyon - dayanıklılık) anımlarına gelen kelimelerin baş harfleridir
- ACID, IBM DB2, MySQL, Microsoft SQL Server, PostgreSQL, Oracle İVTYS, Informix gibi klasik ilişkisel veritabanı sistemlerinde sağlanan temel özelliklerdir.

Atomicity (Bölünmezlik)

Transaction'ı oluşturan tüm işlem parçaları hataya düşmemelidir ve her bir işlemin başarılı olarak sonuçlanması gerekmektedir. Aksi halde başlatılan işlem geri alınır. "Ya hep Ya hiç" mantığına dayanır.

Atomicity fiziksel olaylar dahil olmak üzere elektrik kesintisi, çökme, hata benzeri durumlarda işlem gerçekleşmesini garanti eder.

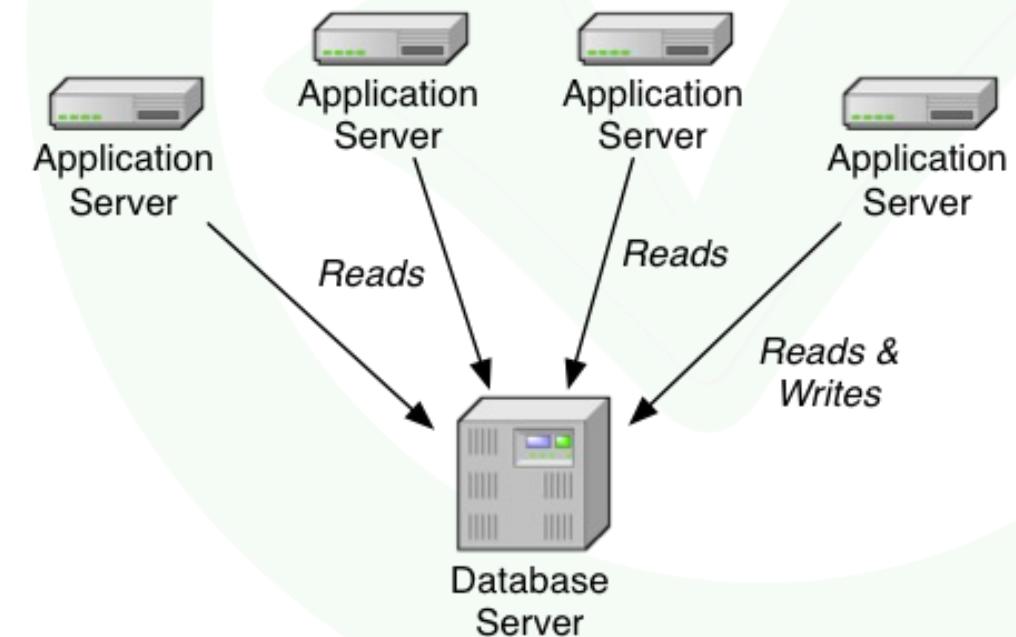
Dışardan bakıldığındá (atom) bölünmez olmak ilkesi göze çarpar.



Consistency (Tutarlılık)

Her kullanıcı verilerin tutarlı halini görürler. Buna kullanıcının kendi yaptığı değişiklikler ve diğer kullanıcıların yaptığı değişiklikler de dahildir.

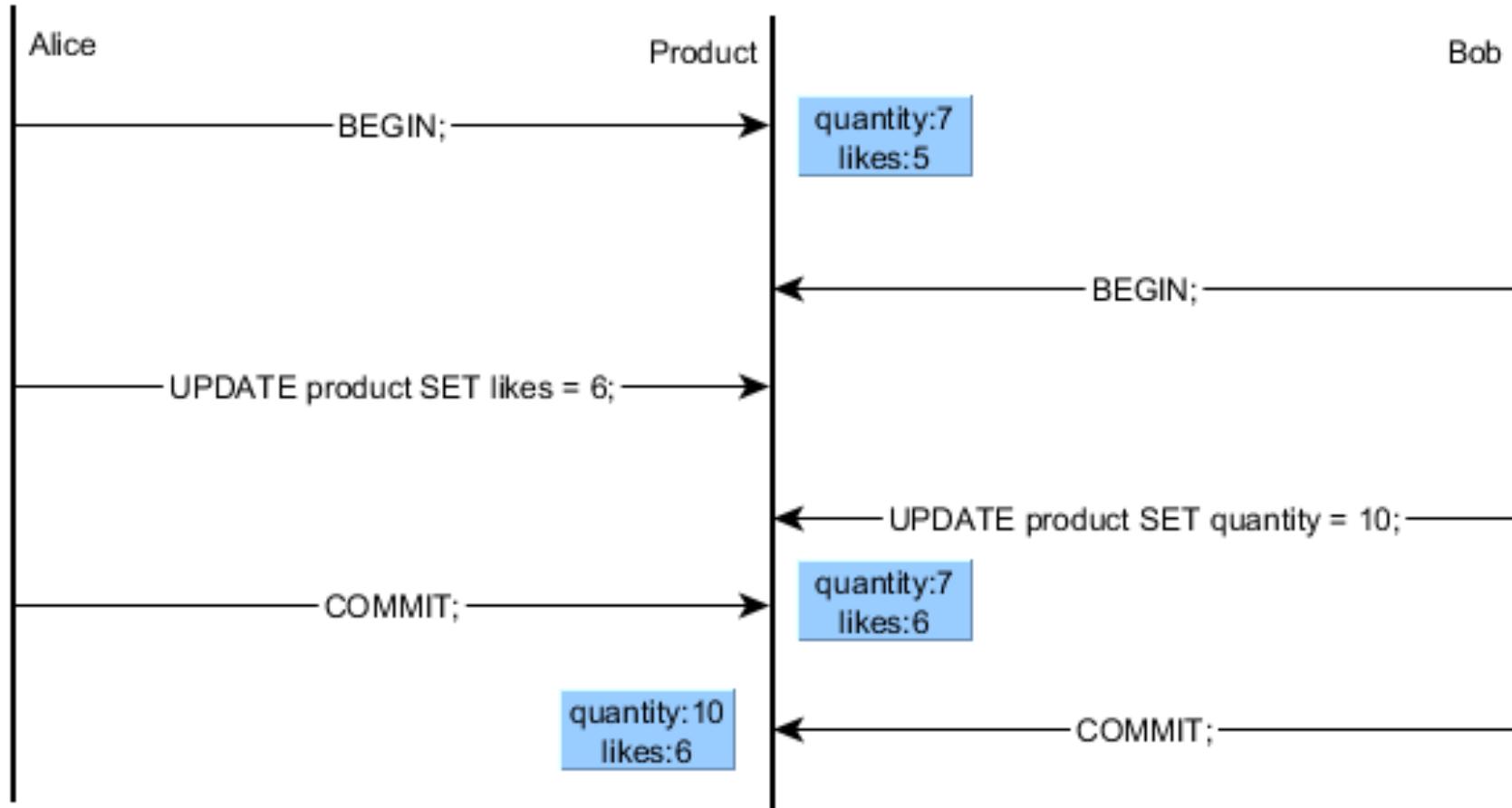
Bir kullanıcı bir tablodaki veriye eriştiği zaman ve o veriyi "update" ederken, diğer hiçbir kullanıcı o veriyi update veya delete edemez. Çünkü ilk kullanıcı tablonun update etmeye çalıştığı kolonu üzerine kilitlemiştir.



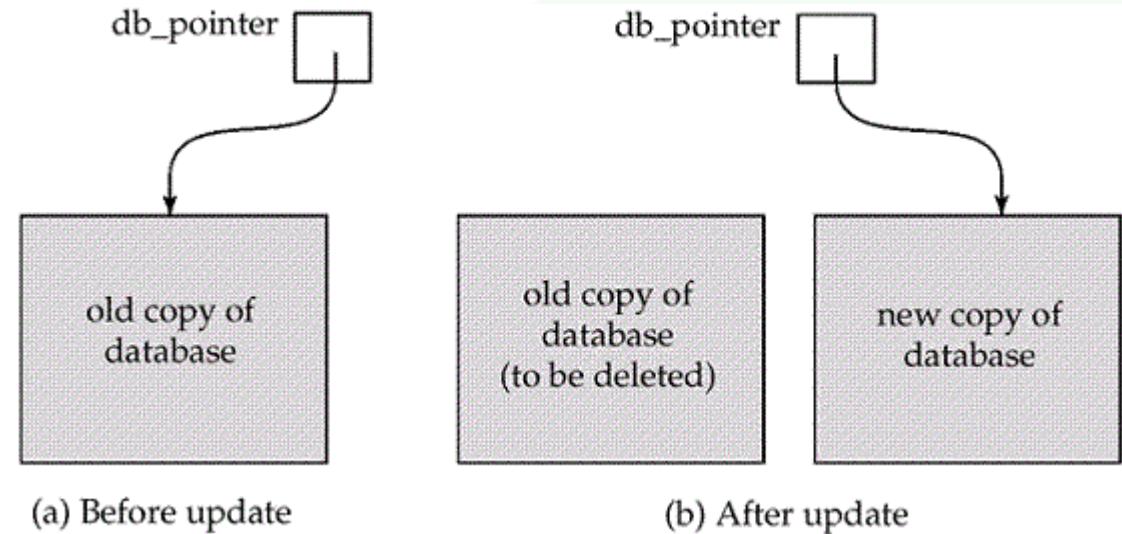
Isolation (İzolasyon)

Veritabanında oluşan transactionlar için eş zamanlı çalışma sürecinde birbirlerini etkilemeden çalışma esasıdır.

Ek olarak kuyruk yönetimi ile bu çalışma süreci kontrol altına alınmaktadır.



Durability(dayanıklılık)



- **Durability(Rollback)** : Atomcityi içerisinde bahsettiğimiz transaction işlemlerini konu almaktadır.
- Elektrik kesintisi, çökme ve ya hata gibi durumlarda oluşabilecek durumlar veri tabanındaki tüm verileri kaybetmemize sebep olabilir.
- SQL Transaction işlemlerinin hızlı olabilmesi için Caching modeli kullanılarak RAM 'den faydalankmaktadır.
- Elektrik kesintisinde ise RAM'de bulunan işlemelere erişme imkânı kalmayacağı gibi hata durumlarında RAM'e erişim yönetimi imkânsız olacaktır.
- Durability, hata durumlarına karşı RAM yönetimi yerine temp file ya da log yönetimi esas alınır.

CAP (Consistency, Availability, Partition Tolerance) Tutarlılık, Müsaitlik ve Parçalanma Payı

- CAP teorisine göre herhangi dağıtık bir sistem aynı anda Tutarlılık (Consistency), Müsaitlik (Availability) ve Parçalanma payı (Partition tolerance) kavramlarının **hepsini birden asla sağlayamaz**. Yani dağıtık bir sistemin her bir parçasından aynı veriye erişebilme, aynı anda bütün isteklere cevap verebilme, kaybedilen paketlere rağmen verinin bütünü kaybetmemeye özelliğine aynı anda sahip olamaz. Bunlardan mutlaka birinde problem çıkacaktır. NoSQL sistemlerde ise bu kavramlar esnetilerek yatayda büyümeye ile performans kazancı amaçlanmıştır. Örneğin veriyi bölgerek, belirli sayıdaki kopyalarını da dağıtık sistemin farklı parçalarına göndererek tutarlılık sağlanabilir. Bu sayede paket kaybı yaşansa da verinin tamamı kaybedilmez, aynı zamanda veri bölündüğü için her bir parçaaya düşen yük dengelenmiş olur.



NoSQL nedir ?

Consistency(Tutarlılık): Dağıtık bir sistemde bir sunucuya x değeri olarak 5 yazdırın. Aynı soruyu ağınızdaki başka sunucuya yaptıınız ve cevap olarak 5 değerini aldınız. Bu tutarlılıktır.

Availability(Erişebilirlik): Dağıtık her zaman eriştiniz. Hem x değerini yazabildiniz, hemde okuyabildiniz.

Partition Tolerance (Bölünebilme Toleransı): Ağınızdaki sunucuların arasındaki bağlantı gittiğinde de çalışmaya devam edebilme.

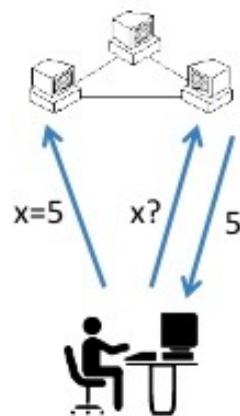
CAP Teoremi ne diyor Aynı anda 3ünü birden yapamazsınız diyor. Neden ? Bu durumu analiz edelim.



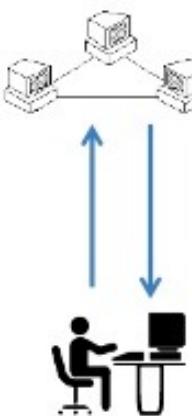
CAP Theorem

CAP Theorem

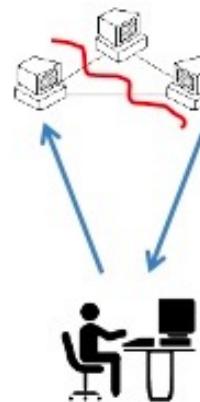
Consistency



Availability



Partition tolerance





(CA) İse Neden (P) olamıyor

Sunucularınızın arasındaki ağ bağlantısı gitmiş. Siz gittiniz birinci sunucuya yazdınız, yukarıdaki resimde görüldüğü gibi diğer sunuculara verinin güncel halini gönderemedi. 2nci, 3ncü sunucuya veriyi sorduğunuzda verinin olmadığını veya 1inci sunucu ile aynı olmadığını göreceksiniz. Yani sistem CA olduğu zaman P(Partition Tolerance) olamıyor. Veriniz parçalanmaya toleranslı değil.

Verinin doğruluğunun kesin olmasını istediğiniz. Müşteri işlemleri, finansal işlemler vb.. bu çok önemlidir. İlişkisel veritabanları ve transactional işlemler buna uygun veriler tutar.



(AP) İse Neden (C) olamıyor

2 sunucunuz(server) olduğunu düşünelim X değeri 2 sunucuda da bulunuyor. Sunucular arası ağınız çöktü. Sunuculardan birinde X değerini güncellediniz. Sorguladığınızda iki sunucu ayrı değer vereceği için bu durumda C(Consistent) olamıyor. Bir sunucu X değeri için 5 , bir diğeri 4 değerini dönebilir.

Sosyal medyadan twitter veya facebook akan verilerde bunlar kullanılabilir. Like sayısını düşünün o anda her kişinin doğru şekilde görmesi/görmemesi önemli değildir. Burada bu veriler parçalanarak tutulabilir. NoSQL veritabanlarından Cassandra, Dynamo gibi sistemler buna uygundur. Eventual Consistency yöntemi ile arkaplanda Node'lardaki veriler eşleştirilir. Bu eşleştirmeden önce yapılan bir sorgularda veri değerlerinde tutarsızlıklar olabilir.

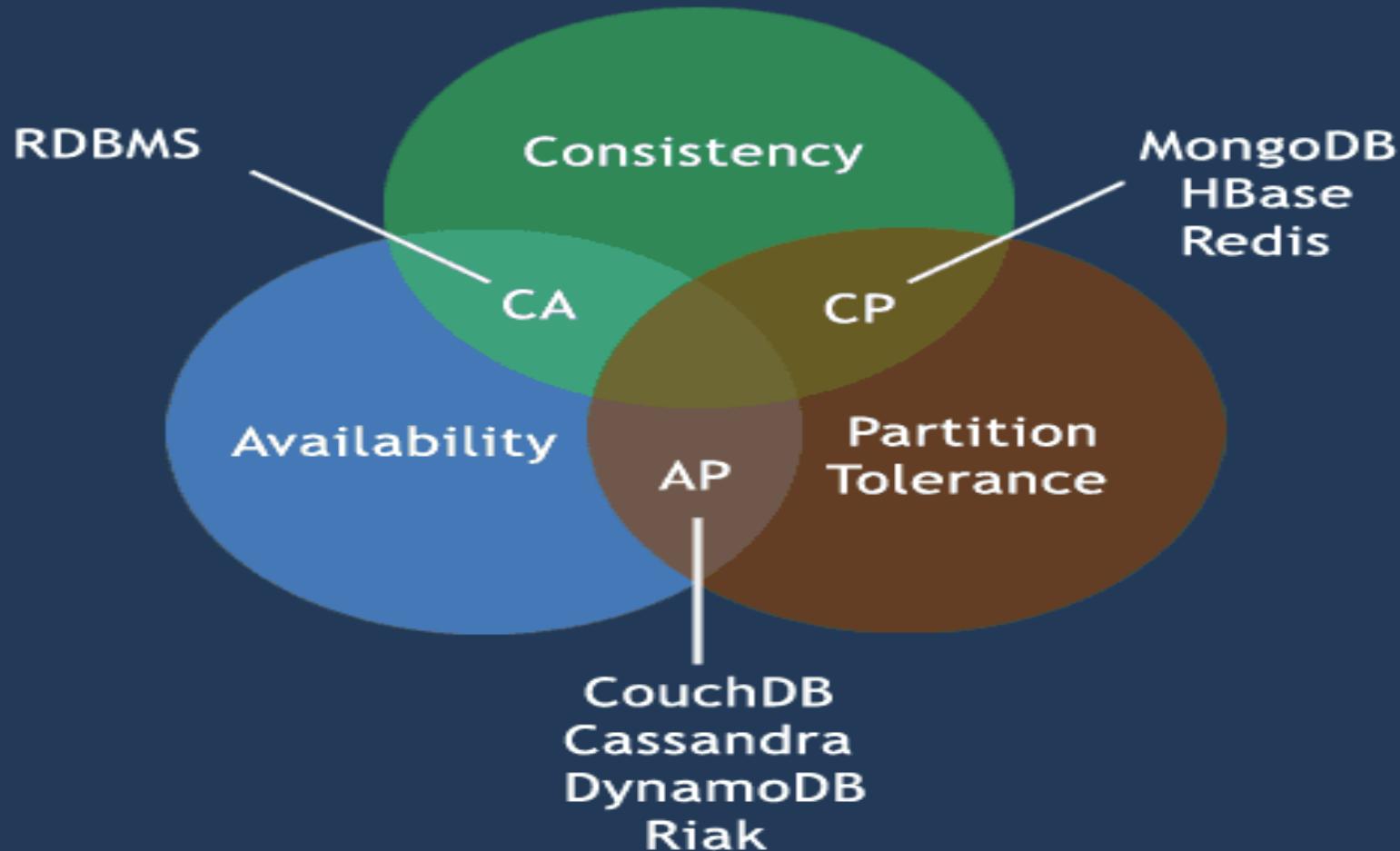


(CP) İse Neden (A) olamıyor

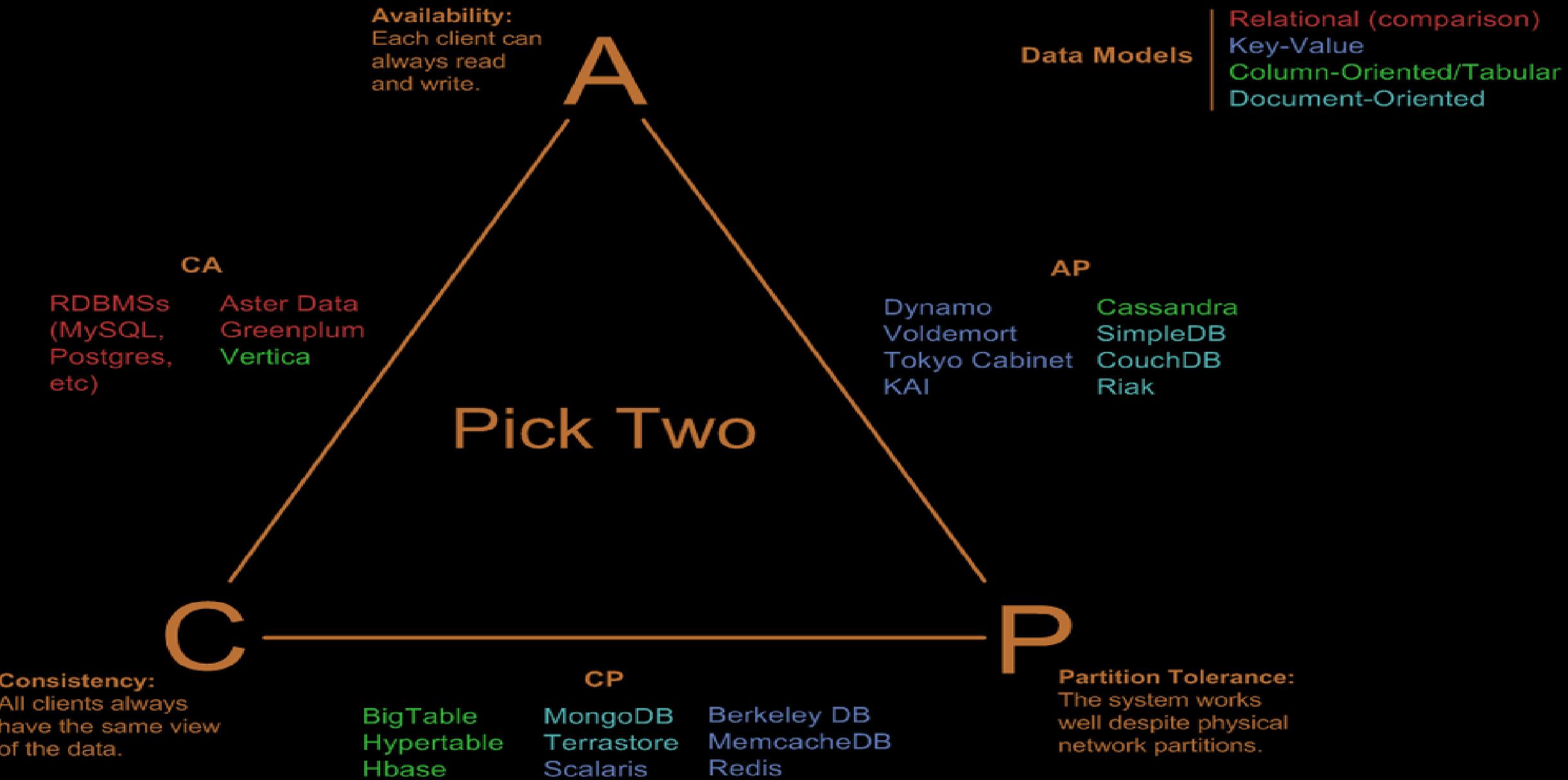
2 sunucunuz(server) olduğunu düşünelim X değeri 2 sunucuda da bulunuyor. Sunucular arası ağınız çöktü. Sunuculardan birinde X değerini güncellediniz. Sorguladığınızda iki sunucu ayrı değer vereceği için bu durumda C(Consistent) olamıyor. Bir sunucu X değeri için 5 , bir diğeri 4 değerini dönebilir.

Sosyal medyadan twitter veya facebook akan verilerde bunlar kullanılabilir. Like sayısını düşünün o anda her kişinin doğru şekilde görmesi/görmemesi önemli değildir. Burada bu veriler parçalanarak tutulabilir. NoSQL veritabanlarından Cassandra, Dynamo gibi sistemler buna uygundur. Eventual Consistency yöntemi ile arkaplanda Node'lardaki veriler eşleştirilir. Bu eşleştirmeden önce yapılan bir sorgularda veri değerlerinde tutarsızlıklar olabilir.

CAP Theorem



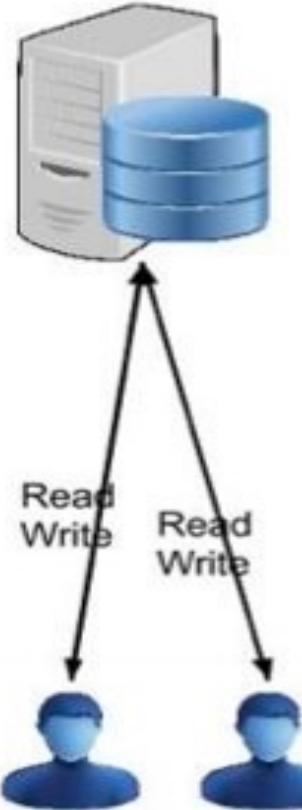
Visual Guide to NoSQL Systems



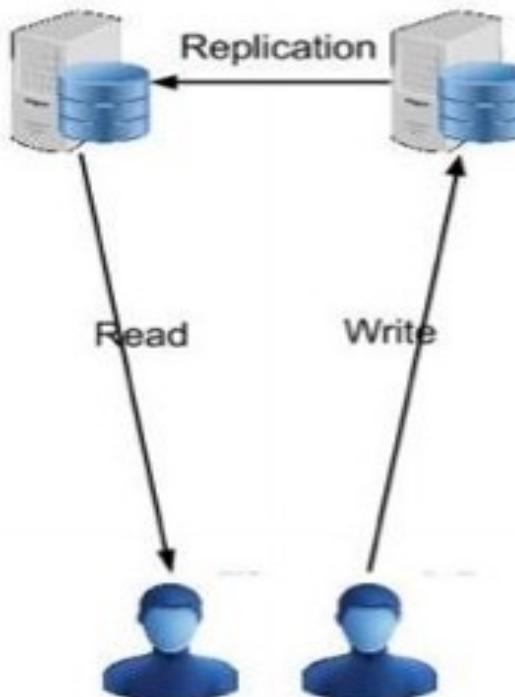


NoSQL nedir ?

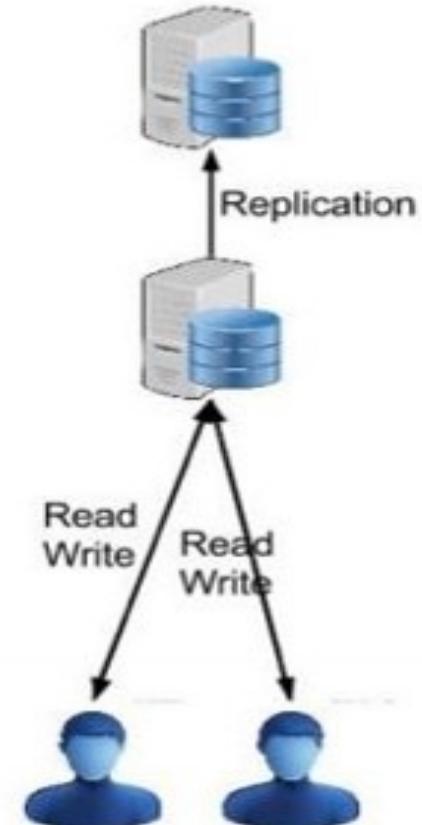
**Consistent
Available
Partition Tolerant**



**Consistent
Available
Partition Tolerant**



**Consistent
Available
Partition Tolerant**



- Remembrance Inc! — Asla unutmayın!

- Not defterinizdeki müşteri sayfasına bilgi kaydedilir.
- **Sorun:** Müşteriler çoğalır, telefonda **bekleme** süresi artar. Memnuniyetsizlik. Hasta olunca o gün iş yapamıyorsunuz.
- **Çözüm:** Eşinizi de işe alıyorsunuz. Bir hat daha alıyorsunuz. Tek numarayı iki hatta yönlendiriyorsunuz. 2 farklı not defteri ile kayıt işi sürüyor.
- **Sorun:** Birinizdeki kayıt diğerinde yok! Dağıtık sisteminiz **tutarlı** değil!
- **Çözüm:** Yeni bilgi girildikçe karşılıklı güncelleme yapılacak.
- **Sorun:** Birisi işe gelemezse?
- **Çözüm:** Tutarlı ve ulaşılabilir olmak için o gün olmayana e-posta atıp, işe başlamadan yeni bilgileri güncellemesi istenecek.
- **Sorun:** Biriniz diğerini güncellemediği durumda ne olur?
- Sistem tutarlılık ve **uygunluk** açısından çalışıyor olsa da **bölünebilme toleransı** bakımından zayıf kalıyor.

- **Tutarlılık:** Kayıt ekleyen müşterilen, ne kadar çabuk geri ararlarsa arasınlar, her zaman her koşulda en güncel kayda ulaşacaklardır.
Uygunluk: Remembrance Inc! siz ve/veya eşimiz işte olduğu sürece her zaman çağrı kabul etmeye müsait olacaktır.
Bölünebilme/parçalanma toleransı: Remembrance Inc! sizin ve eşinizin arasında iletişim kopukluğu olsa bile çalışmaya devam edecektir.
- Yerel kayıtlar değişince arka planda çalışan bir işlem tarafından diğerleri de güncellenir. (**eventual consistency**) Tek sorun arada sırada tutarlılığı kaybediyor olmanızdır. Örneğin müşteri önce eşinize kayıt yaptırip sizin kayıtlarınız güncellenmeden evvel tekrar aradığında size ulaşırsa bilgileri bulunamaz.

NoSQL

anti-İVTYS değildir,
anahtar-değer depolarının,
belge veritabanlarının
Graph veritabanlarının
kullanımının altını çizmesine
rağmen ilişkisel olmayandır.

Document Database	Graph Databases
  	 
Wide Column Stores	Key-Value Databases
   	   

- NoSQL sistemlerini genel olarak 4 grupta toplayabiliriz:
- **Döküman (Document) tabanlı:** Bu sistemlerde bir **kayıt döküman** olarak isimlendirilir. Dökümanlar genelde **JSON** formatında tutulur. Bu dökümanların içerisinde sınırsız alan oluşturulabilir. Farklı formatlarda büyük verilerin saklandığı uygulamalar için uygundur. MongoDB, CouchDB, HBase, Cassandra ve Amazon SimpleDB bunlara örnektir.
- **Anahtar / Değer (Key / Value) tabanlı:** Bu sistemlerde anahtara karşılık gelen tek bir bilgi bulunur. Yani **kolon kavramı yoktur**. Azure Table Storage, MemcacheDB ve Berkeley DB bunlara örnektir.
- **Grafik (Graph) tabanlı:** Diğerlerinden farklı olarak verilerin arasındaki ilişkiyi de tutan, Graph theory modelindeki sistemlerdir. BPM (Business Process Management) çözümleri ve sosyal ağ uygulamaları için uygundur. Neo4J, FlockDB bunlara örnektir.
- **Geniş Sütun tabanlı:** **Çok büyük verilerin tutulduğu dağıtık sistemler** için uygundur. Veriler anahtar & değer ikilisi şeklinde saklanır fakat anahtar birden fazla kolona işaret etmektedir. Örnek sistemler: Cassandra, HBase

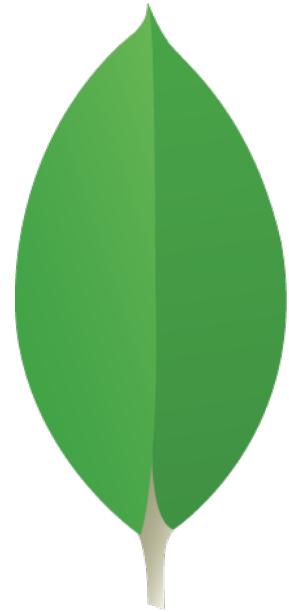
Sorgu Modeline Göre NoSQL Veritabanları

- İlişkisel veritabanları karmaşık sorgu ihtiyaçlarına NoSQL sistemlerden daha iyi cevap vermektedir.
- **Doküman tabanlı** veritabanları verideki **tüm alanlar üzerinden sorgulama** imkânı sunarlar. Ayrıca farklı indeks seçenekleriyle (text, sparse, TTL, ...) sorgu performansını artırmak da mümkündür.
- **Çizge tabanlı** veritabanları veriler arasındaki ilişkiler ve bu ilişkilerin özellikleri üzerinden sorgulama imkânı sunar. Diğer sorgu modellerinde yüksek performans sunamadıkları için genel kullanımında tercih edilmezler.
- **Anahtar – değer** tabanlı veri tabanları sadece **anahtar** üzerinden erişim sunarlar. Bazı ürünlerin ikincil indeks desteği olsa da bu destek sınırlıdır.
- **Kolon** tabanlı veritabanları, anahtar – değer tabanlı veritabanları gibi veriye anahtar üzerinden erişim sağlarlar. Doküman tabanlı sistemler gibi **verinin tüm alanlarından sorgulama yapmak mümkün değildir**. Bu grupta yer alan Cassandra veritabanı geniş ikincil indeks ve özel veri tipleri (set, map, list, tuple, ...) desteği ile çok farklı sorgu modellerine destek verebilmektedir.

Performans için ne yapmalıyız?

- Sorgu iyileştirmeleri, yeni indeksler oluşturmak...
- Sunucu güçlendirme (dikey büyümeye)
- Yatay bölünmesi (Sharding)
 - ❖ Bağlantı (join) ve Birleştirme (aggregation) yapılamaz
 - ❖ Verilerin taşınması
 - ❖ Şema değişiklikleri

Yatay bölünmenin sorunlu olması nedeniyle NoSQL sistemler ön plana çıkmıştır.



mongoDB®

MongoDB=Hu**MONGO**us DB

- **10gen** firmasınca 2007 yılında başlanan ve 2009 yılında **AGPL** lisansıyla açık kaynak projesine dönüştürülen MongoDB en popüler NoSQL yapılarından birisidir.
- MongoDB yüksek performans, yüksek kullanılabilirlik ve otomatik ölçeklendirme sağlayan bir açık kaynak belge veri tabanıdır
- MongoDB yapı olarak dokümanları dikkate alır.
- MongoDB dokümanları JSON nesneleri benzemektedir(**BSON**), alanların değerleri diğer dokümanları ya da dizileri içerebilir.
- NoSQL veri yapılarının geneli KEY ile sorgulamaya izin vermektedir. Fakat MongoDB QUERY desteği ile veriye ulaşılmasını sağlamaktadır.
- Sorgu oluşturulan alanlara performans artımını oluşturmacı adına secondary indekse destek vermektedir.

BSON

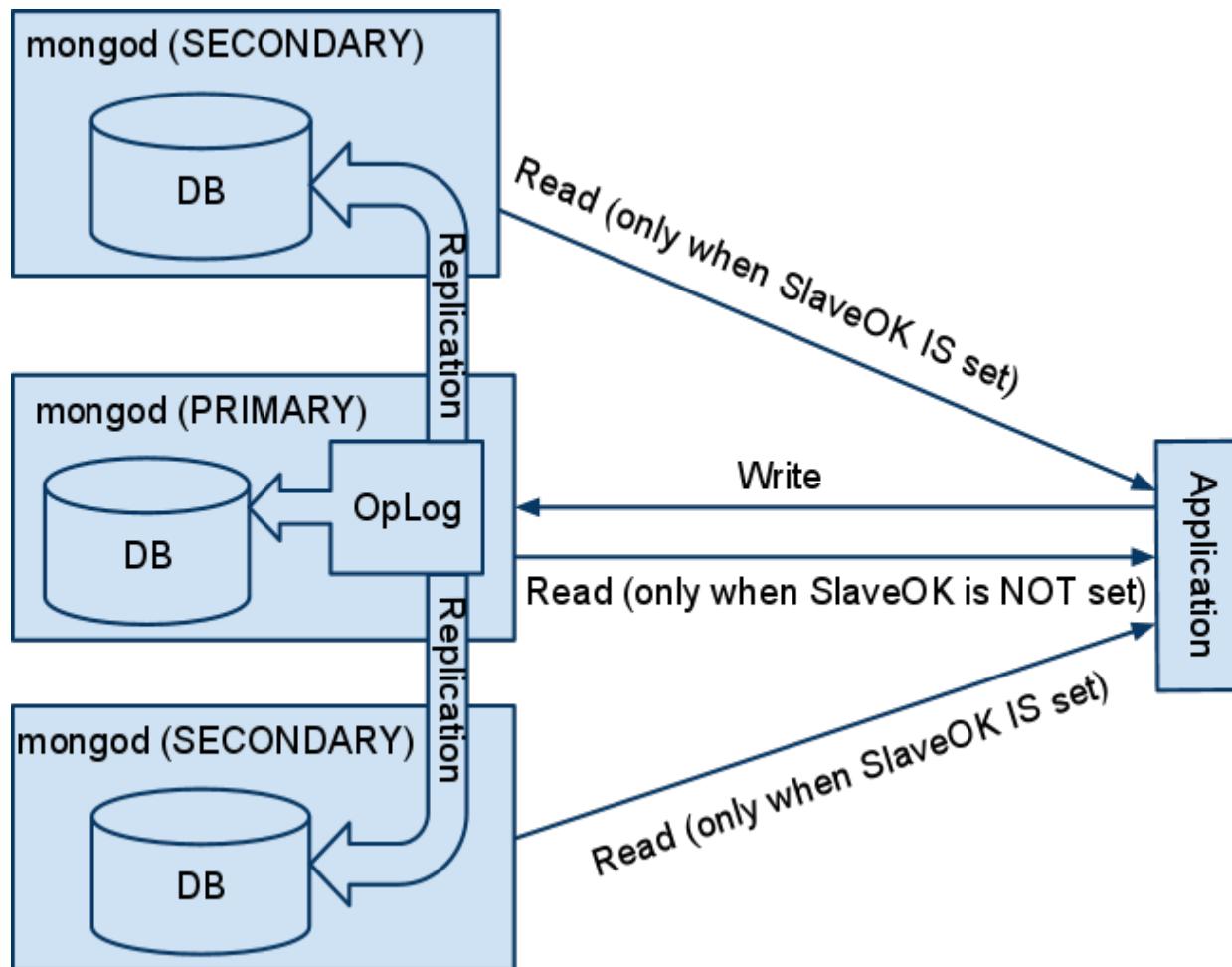
- Binary JSON = BSON
- {"hello":"world"}
- Bson:

```
\x16\x00\x00\x00          // total document size
\x02                      // 0x02 = type String
hello\x00                  // field name
\x06\x00\x00\x00world\x00    // field value (size of value, value, null terminator)
\x00                      // 0x00 = type EOO ('end of object')
```

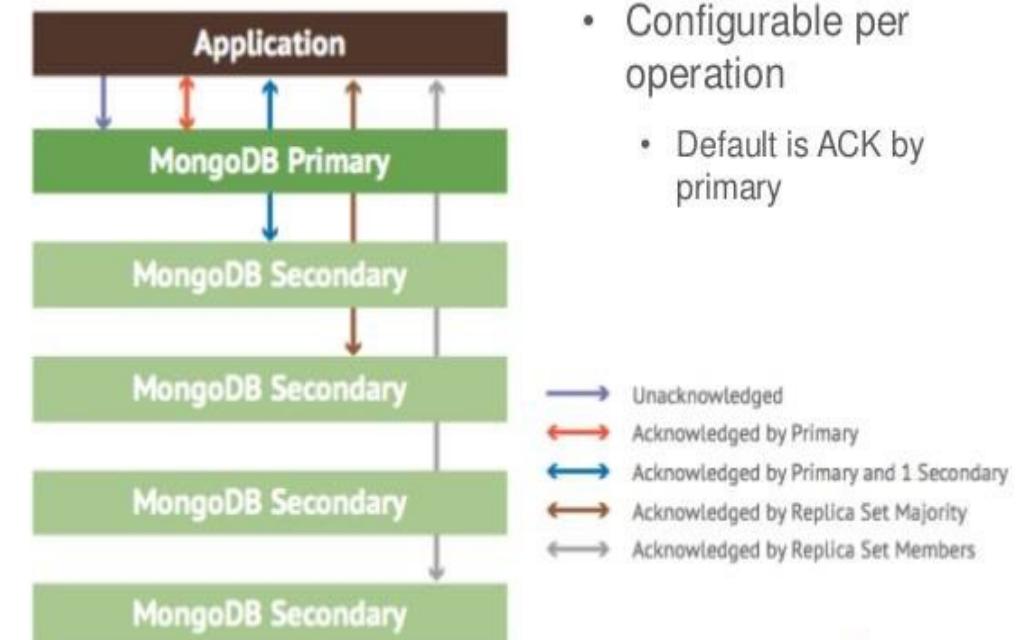
BSON, veri kümelerinin boyutunu da sakladığından, çeşitli algoritmalar ile arama süresi azaltılabilmektedir. Dolayısıyla gezinme çok daha kolay gerçekleşmektedir.

x16 = 22 = 0001 0110
xFF = 255 = 1111 1111

Master-Slave Replication destekleri ile Master sunucu yazma işlemi yaparken Slave sunucu okuma işlemi yapmaktadır. Master sunucu herhangi bir sebepler cevap veremez durumda olursa Slave'lerden biri belirlenen koşullarda Master görevini üstlenmektedir.



Data Durability – Write Concerns



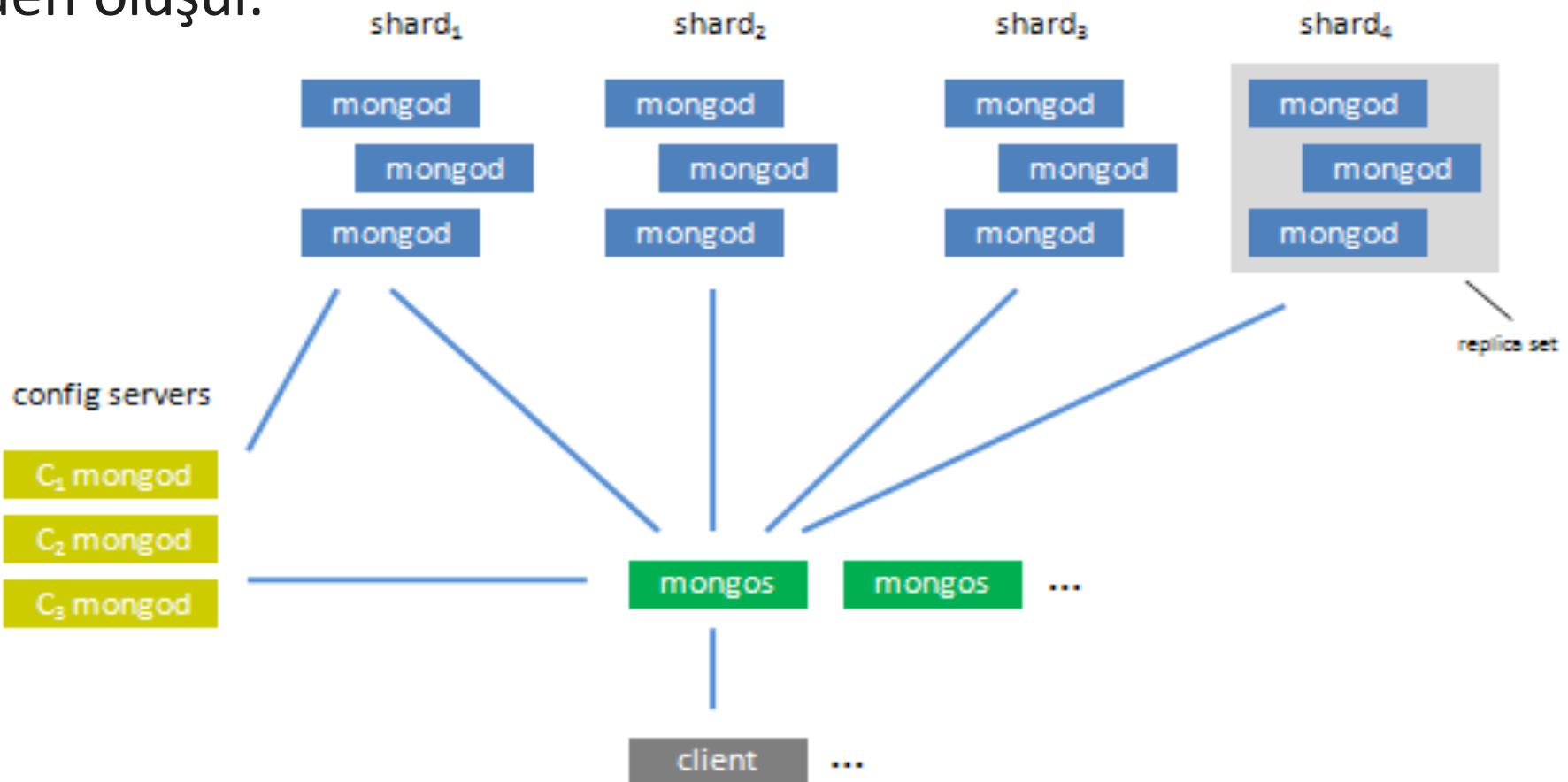
MongoDB'in en önemli özelliklerinden biri ise **Sharding**'dir. Replication yapısı ile beraber kullanılan bir sistemdir. Büyük veri ve yoğun şekilde veri yazma işlemi kullanılabıkça ve tek Mongod işleme yetişemiyorsa, sistem ölçek olarak yetmeyorsa “ram ve ya disk gibi” devreye Shard Mimarisi girer.

Shard üç ana bölümden oluşur.

Shards

Mongos

Config Server



Shards

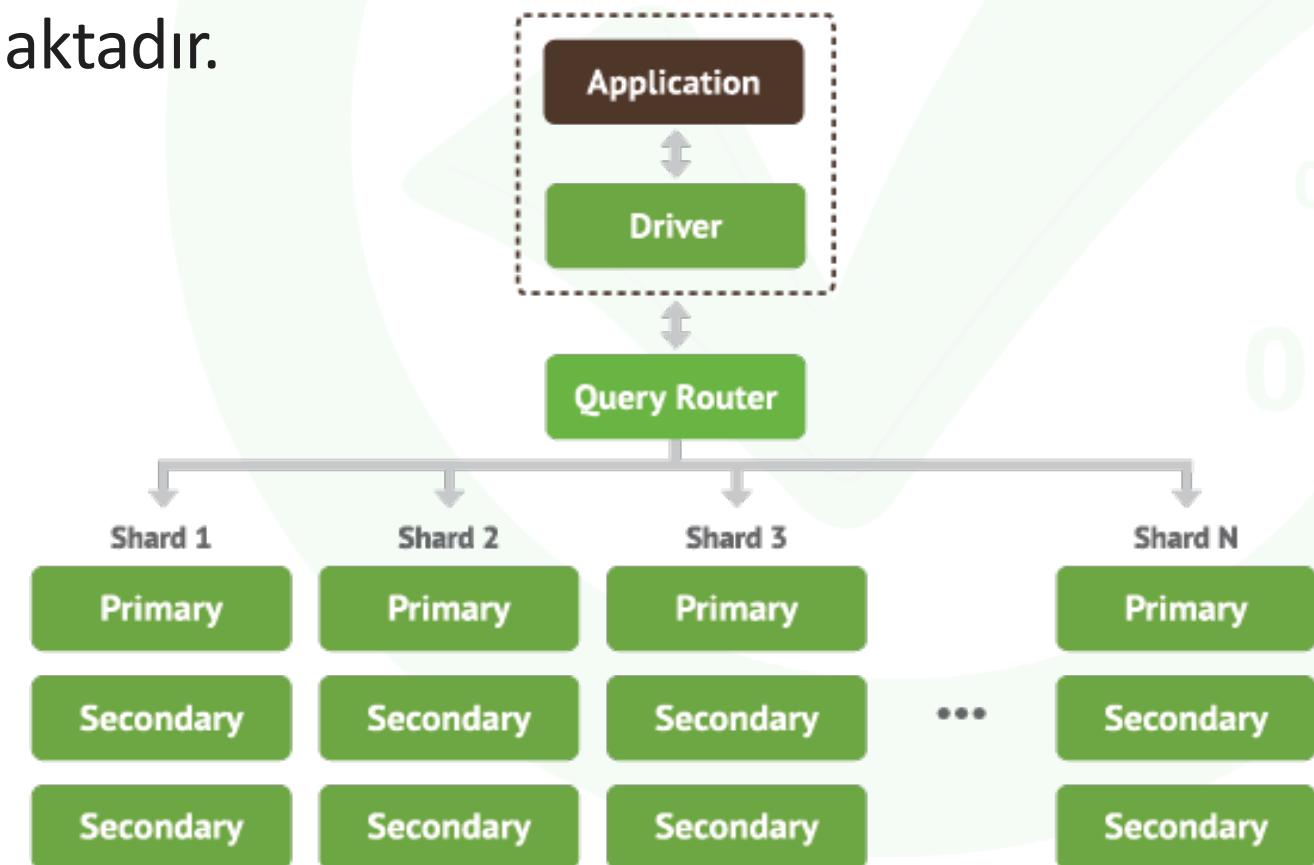
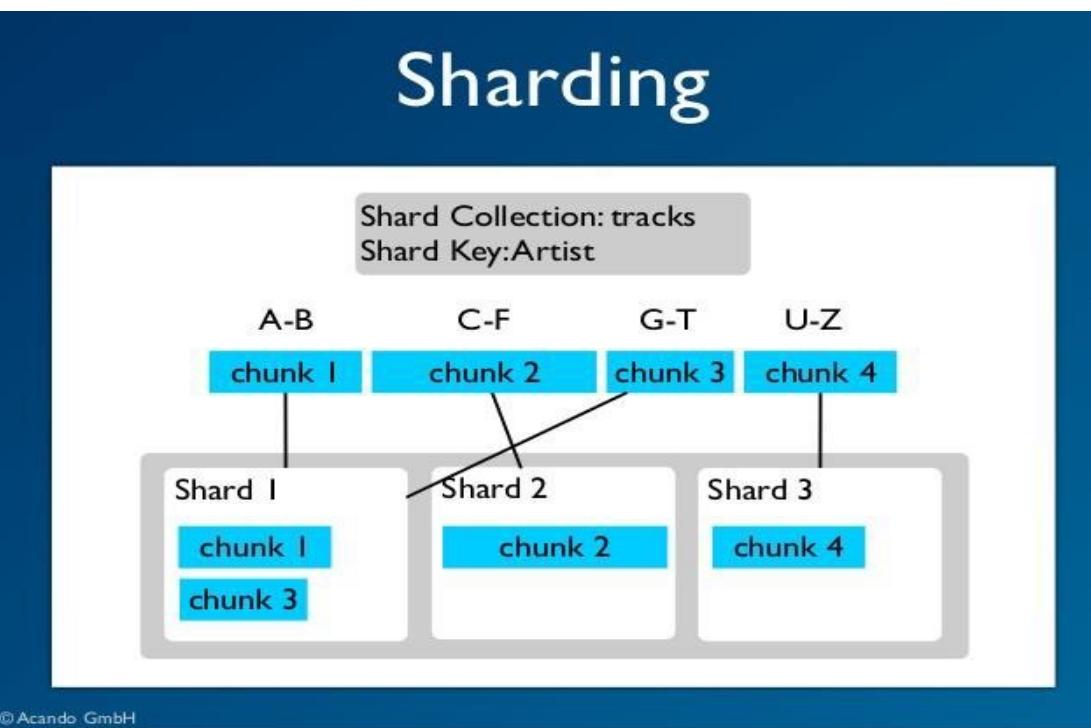
Verinin Mongod işlemlerine bölündüğü alandır.

Aynı aynı Mongod'ların yönetilmesini Mongos sağlar.

Her bir veri belirli Chunk'lara bölünür.

Aynı verinin bölünmüş Chunk'ları farklı Shard'larda barınır.

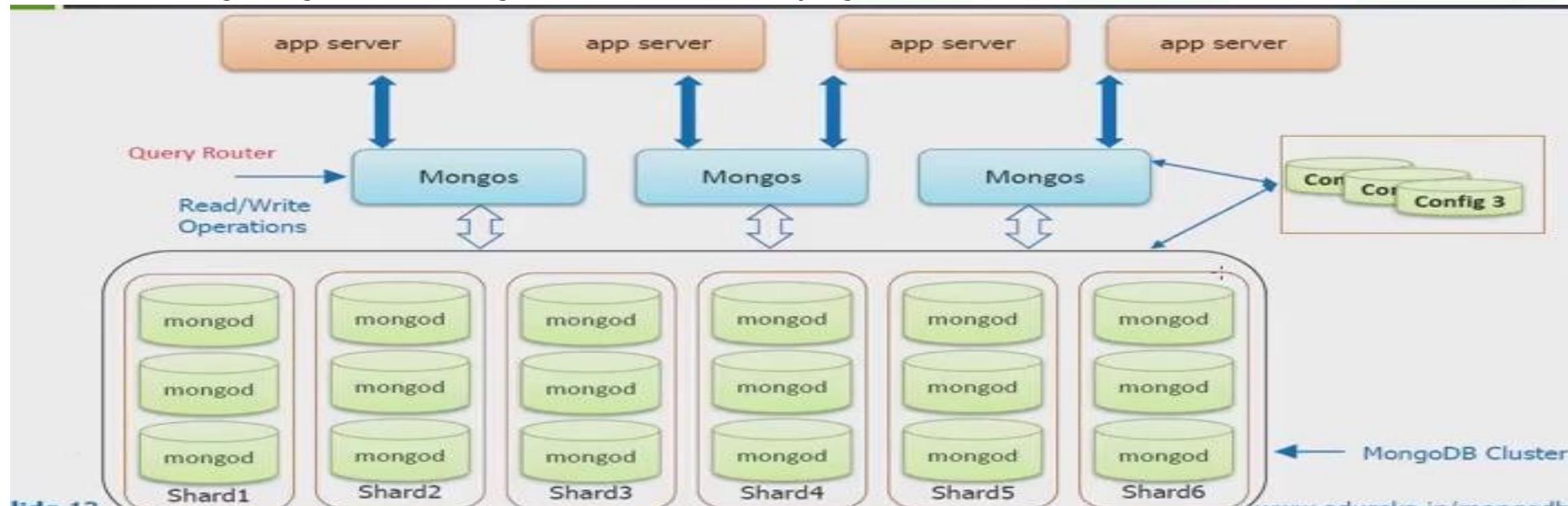
Her bir Shard Replica Set olarak çalışmaktadır.



Mongos “Query Router”

İstekler direkt olarak Shard'larla bağ kuramazlar. Çünkü hangi veri hangi Shard'da bilemezler. Bu işi üzerine Mongos alır. Bu sebepler Query Router denmektedir.

İstek oluştugunda Mongos devreye girerek ilgili Shard'lara görev yönlendirmesi yapar ve dönen sonuçları birleştirerek oluşan isteğin cevabını iletir. Mongos veri tutmaz sadece işlem yönetimini üstlenir. İşleyişle ilgili süreci takip edebilmesi için işlem süreç verilerine ihtiyaç tutar.



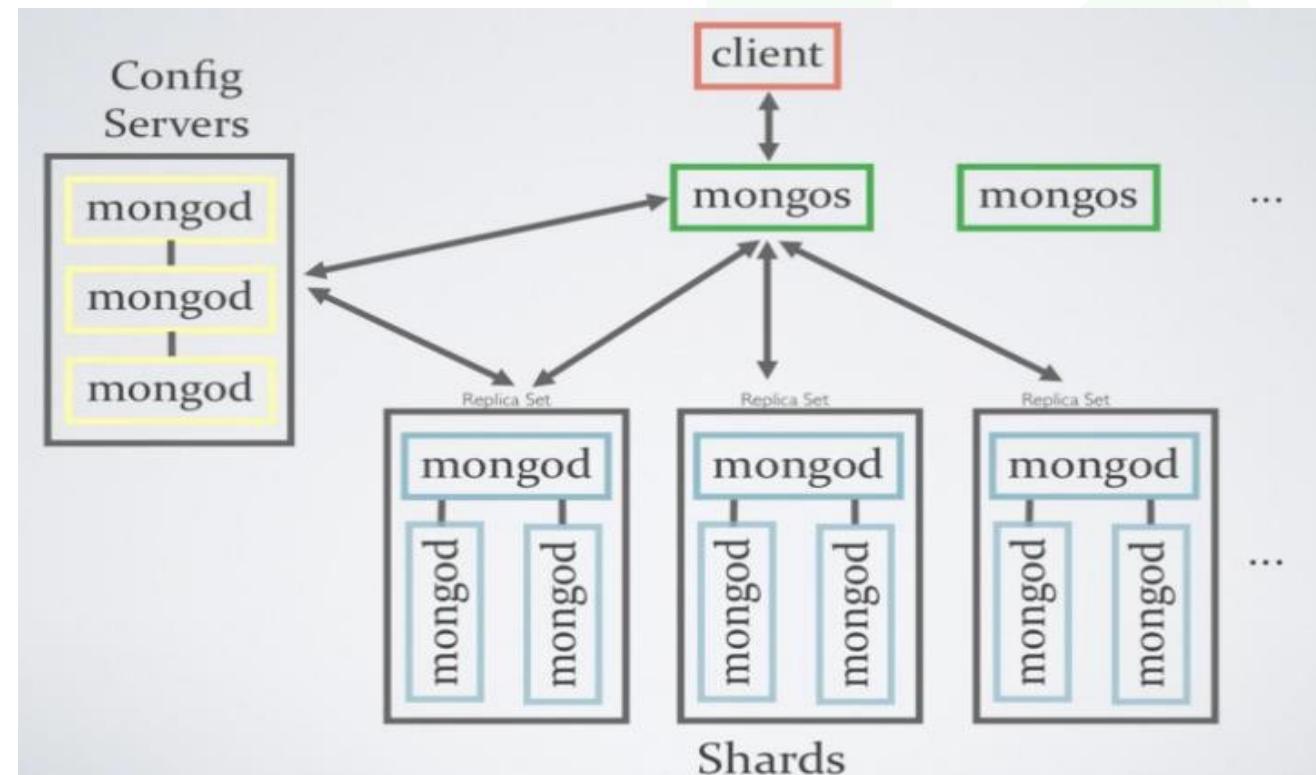
Config Server

Mongos'un görev sürecindeki işleme durumlarının veri olarak tutulduğu Mongod yapılandırma örneğidir.

Kısaca Cluster'ın tüm bilgisinin barındıran veri alanıdır.

Her 200MB 'lık veri için 1KB'lık Config Server alanı ayırılması önerilir.

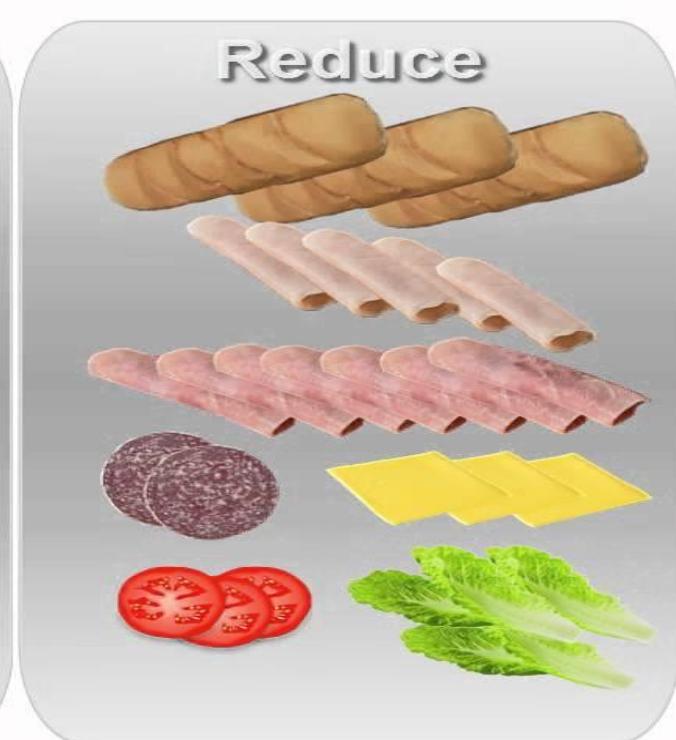
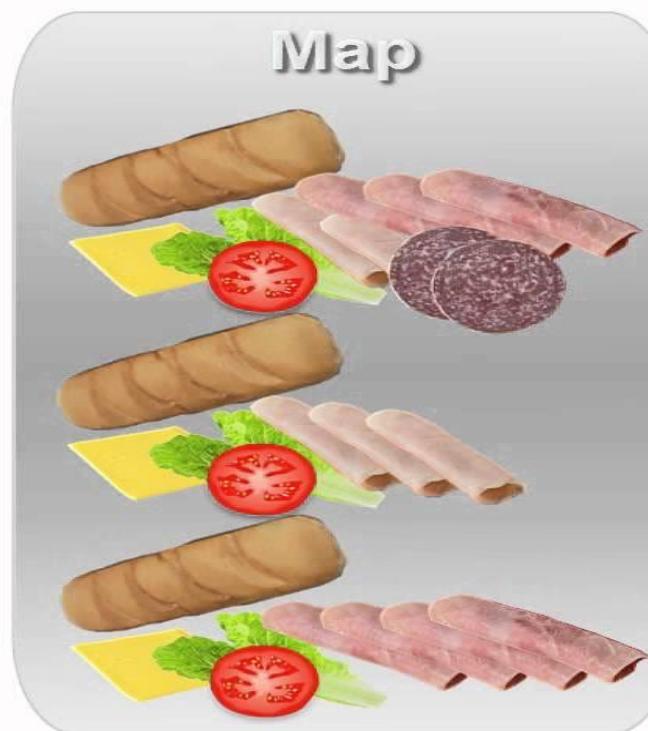
Mongos'a gelen isteklerle ilgili hangi Shard 'da hangi Chunk barınır bilgisini bununla beraber serverların durum bilgisini barındırırlar.



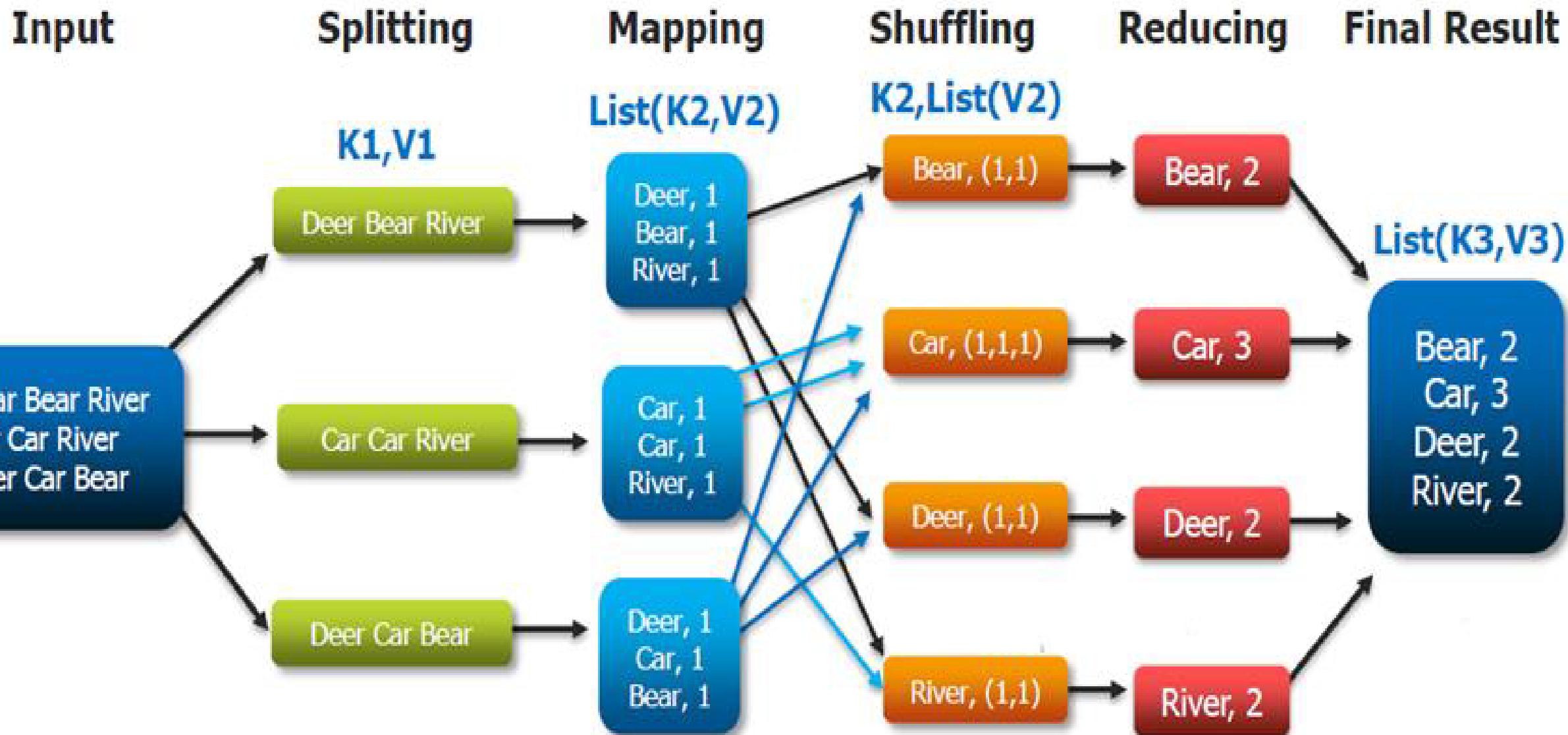
- MongoDB aynı zamanda **MapReduce** desteği sağlamaktadır. Oluşturacağınız analizleri görebilmenizi kolaylaştıracak bir sistemdir
- MapReduce ilişkisel veritabanı sistemleri ile (RDBMS) karşılaştırırsak, Map select ifadesine ve where kıtaslarını belirlemeye, Reduce ise having, sum, count, average gibi hesaplamalar yapmamıza benzemektedir.



IBM Cloudant®

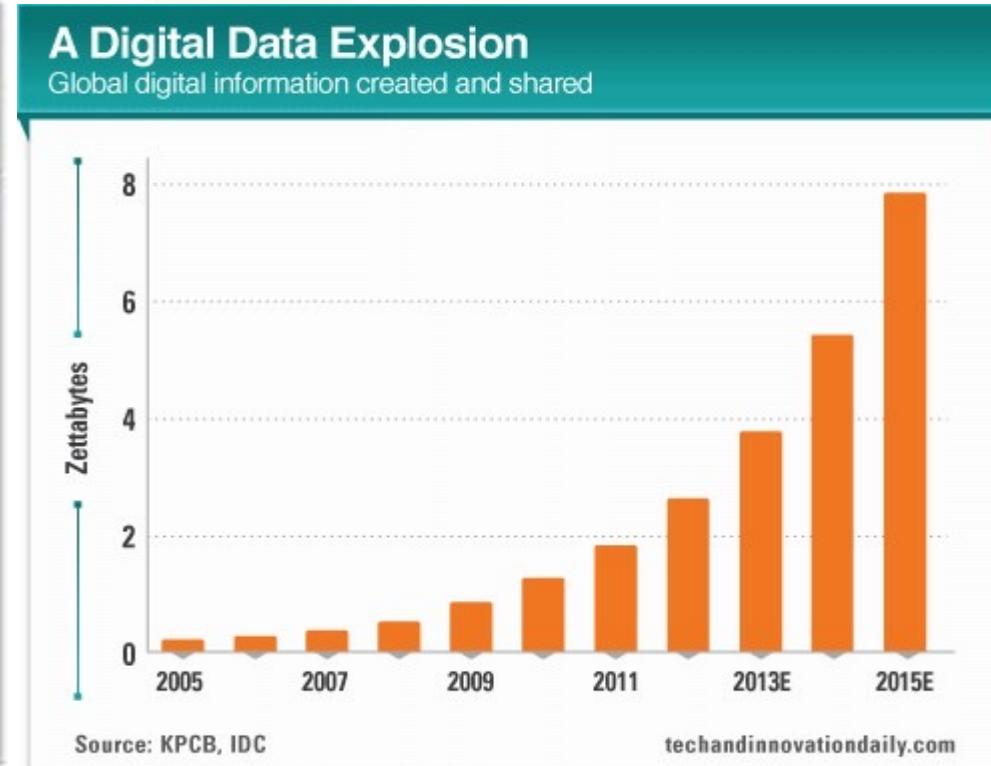


The Overall MapReduce Word Count Process



- Klasik veritabanı sistemleri ile Petabyte mertebesindeki verilerin işlenmesi ancak milyon dolar seviyesindeki donanım ve yazılım ile mümkün iken, MapReduce bu soruna çok ciddi bir alternatif durumundadır.

Bayt Birimleri						
Yaygın önek			Binari önek			
Ad	Sembol	Ondalık İkililik	Ad	Sembol	İkililik	
kilobayt	KB	10^3	2^{10}	kibibayt	KiB	2^{10}
megabayt	MB	10^6	2^{20}	mebibayt	MiB	2^{20}
gigabayt	GB	10^9	2^{30}	gibibyte	GiB	2^{30}
terabayt	TB	10^{12}	2^{40}	tebibayt	TiB	2^{40}
petabayt	PB	10^{15}	2^{50}	pebibayt	PiB	2^{50}
eksabayt	EB	10^{18}	2^{60}	eksbibayt	EiB	2^{60}
zettabayt	ZB	10^{21}	2^{70}	zebibayt	ZiB	2^{70}
yottabayt	YB	10^{24}	2^{80}	yobibayt	YiB	2^{80}



RDBMS vs MongoDB

RDBMS vs MongoDB		
	RDBMS	MongoDB
Veri Yapısı	Sabit	Düzensiz
Join	Dahil	İç içe yapı
Transaction	Dahil	Atomic operasyonlar
Ölçeklenebilme	Dikey	Yatay
SQL Dili	Dahil	API 'ler ile
Primary Key	Var	Var
Foreign Key Constraint	Var	Referans

MongoDB verilerini **Collection** içerisinde Document olarak tutar. Veri yapılarını dikkate alırsak SQL veri tabanları ile karşılaştırırsak Collection tabloya, Document ise satırlara karşılık gelir.

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document
column	field
Index	index

RDBMS'lerde veriler tablolarda, tanımlı sütunlarda satır satır bulunurken, NoSQL sistemler sabit tablo tanımlarına bağlılığından farklıdır. Yani daha sonradan özel bir işlem yapmadan yeni kolonlar eklenebildiği gibi, kayıtlar arasında kolon farklılıklarını olabilir.

Örneğin 1. satırda a ve b kolonları varken 2. satırda bambaşka kolonlar bulunabilir.

Bu esnek yapı bize **denormalizasyon** kolaylığı sağlar.

RDBMS'lerde veriler ilişkilerine göre ayrı tablolara düzenli bir şekilde yerleştirilerek normalize edilirken, verilere erişim için birleştirme (**join**) kullanılır.

Dağıtık sistemlerde birleştirme operasyonu sıkıntılara sebep olur. Çünkü birleştirilmek istenen veriler farklı parçalarda olabilir. Bu yüzden dağıtık sistemlerde birleştirme işlemi elle yapılmalıdır. Yani önce a verisi çekilir, sonra bununla ilişkili b verisi çekilip programatik olarak birleştirilme yapılır. Normal sistemlerde bu işlemler performans kaybına sebep olurken, dağıtık sistemlerde bu performans kaybı sistemin genel performansı yanında önemsizdir. İşte bu sebeple dağıtık sistemlerde birleştirme operasyonu yapmak yerine veri denormalize tutularak verilere erişim kolaylaştırılır.

Denormalize edilmiş veriler içerisinde veriler kendini tekrar edecktir ancak bu da dağıtık sistemlerin çalışma performansını artırdığı için önemli değildir. NoSQL sistemler de bu sebeple sabit tablo tanımı içermez, verilerin denormalize tutulmasını kolaylaştırır.

Soyadi	Adi	Yas
AYDIN	Onur Ekin	16
AYDIN	Efe Çınar	9
AYDIN	Gürkan	42

MongoDB'de her dokümanın benzersiz kimliklikleri olmak zorundadır. Benzersiz Kimlik alanı varsayılan olarak bu alan adı **_id** tanımlanmıştır. Eklenen document'a bir tekil anahtar atanmamışsa MongoDB ObjectId tipinde otomatik olarak benzersiz bir kimlik atar.

```
{
  "_id": ObjectId("124ega5c2b7d284dad101e4bc9"),
  "Soyadi": "AYDIN",
  "Adi": "Gürkan",
  "Yas": 42
},
{
  "_id": ObjectId("124efa8d2b7d284dad101e4bc8"),
  "Last Name": " AYDIN ",
  "First Name": "Onur Ekin",
  "Age": 16,
  "Adres": "Bulgurlu Mh. Karlıdere Cd.",
  "Sehir": "İstanbul"
},
{
  "_id": ObjectId("124efa8d2b7d284dad101e4bc9"),
  "Last Name": " AYDIN",
  "First Name": "Efe Çınar",
  "Age": 9,
  "Adres": "Bulgurlu Mh. Karlıdere Cd.",
  "Okul": "Faik Reşit Unat",
  "Sehir": "İstanbul"
}
```

ObjectId – 12 byte hexadecimal bir değerdir

507f191e-810c19-729d-e860ea

a 4-byte value representing the seconds since the Unix epoch,

a 3-byte machine identifier,

a 2-byte process id, and

a 3-byte counter, starting with a random value.

12 byte = 96 bit = 2^{96} = 79,228,162,514,264,337,593,543,950,336 adet farklı kayıt numarası tutulabiliyor.

- NoSQL sistemler temel olarak verilere **tekil anahtarlar** üzerinden erişir. Her NoSQL sisteminde **ikincil indeks (secondary index)** yeteneği de bulunmayabilir. Bu yüzden verilere erişim SQL sorguları ile yapıldığı gibi kolay yapılamaz.
- Uygulamanın özelliğine göre birincil anahtarlar belirli bir mantığa göre verilir ve bu sayede veriye erişmeden önce zaten bu anahtar biliniyor ya da oluşturulabiliyor olur.
- Örneğin **zamana göre artan ön ek (prefix), kaydın anahtarı ile birleştirilerek** tek seferde ilgili kaydın belirli bir zaman aralığındaki kayıtlarına erişmek mümkündür.
- Bu sayede herhangi bir sorguya gerek olmaksızın tamamen **anahtarlar** üzerinden verilere **hızlıca erişim sağlanmış** olur.



MONGODB

MongoDB veritabanı gibi sistemler CP uygundur. Default'ta *strongly consistent*' dır.

MongoDB NoSql veri tabanı sistemlerinin Doküman tabanlı sistemlerinden biridir. Veriler JSON\BSON formatında saklanır.

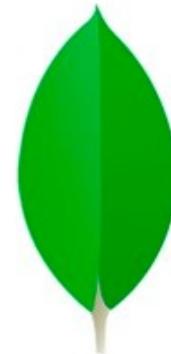
```
{  
    name: "sue",  
    age: 26,  
    status: "A",  
    groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

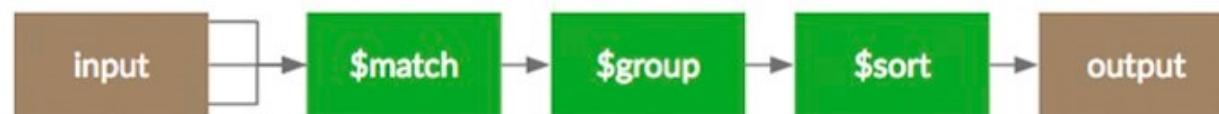


MongoDB Altyapısı

Aggregation: Dağınık halde bulunan verileri toplayıp gruplandırmak ve bunlar üzerinden gerekli işlemleri yapmak.



mongoDB
Aggregation Framework





MONGODB ARTILARI

Map-Reduce desteği: Böl gönder, topla gönder. Burada kullanıcının sisteme yüklediği veriler mapperlar ile parçalara bölünür ve gerekli alanlara dağıtilır, bu sayede işlemler daha hızlı yapılır ve sistemin her alanına aktarılan yük dahada azaltılmış olur, sistem tarafından gerekli işlemler yapıldıktan sonra bu veriler Reducer'lar ile tekrar bir araya getirilir ve kullanıcıya aktarılır.

Text Search: MongoDB içerisinde bir metin arama desteği bulunmaktadır. Bunun için bir string ifadeyi \$text fonksiyonunu kullanarak arayabilirsiniz. Bana sorarsanız mongodb'nin en güzel özelliklerinden biri de budur istediğiniz zaman hiç bir zorluk yaşamadan bir metin arayabilir ve bulabilirsiniz bu işlemi RDBMS sistemlerde MsSQLde yapmak için Full text search özelliğini kullanmak gerek bazen bunun için taklalar atmanız bile gerekebiliyor.

```
db.stores.find( { $text: { $search: "java coffee shop" } } )
```



MONGODB ARTILARI

- **Data Models:** MongoDB'nin veri tutma biçimi Sqlden farklı bir halde bulunmaktadır. Bir dizi içerisinde string ve integer ifade bulunabilir. Bu özellikle RDBMS sistemlerden fark olarak düşünebileceğimiz bir özellik.

```
{  
  _id: <ObjectId1>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```

The diagram illustrates an example of an MongoDB document structure. The document is enclosed in a blue-bordered box. It contains several fields: '_id', 'username', 'contact', 'access', and two curly braces at the end. Two arrows point from the text 'Embedded sub-document' to the sub-documents within the 'contact' and 'access' fields. The 'contact' field contains 'phone' and 'email' fields, while the 'access' field contains 'level' and 'group' fields.

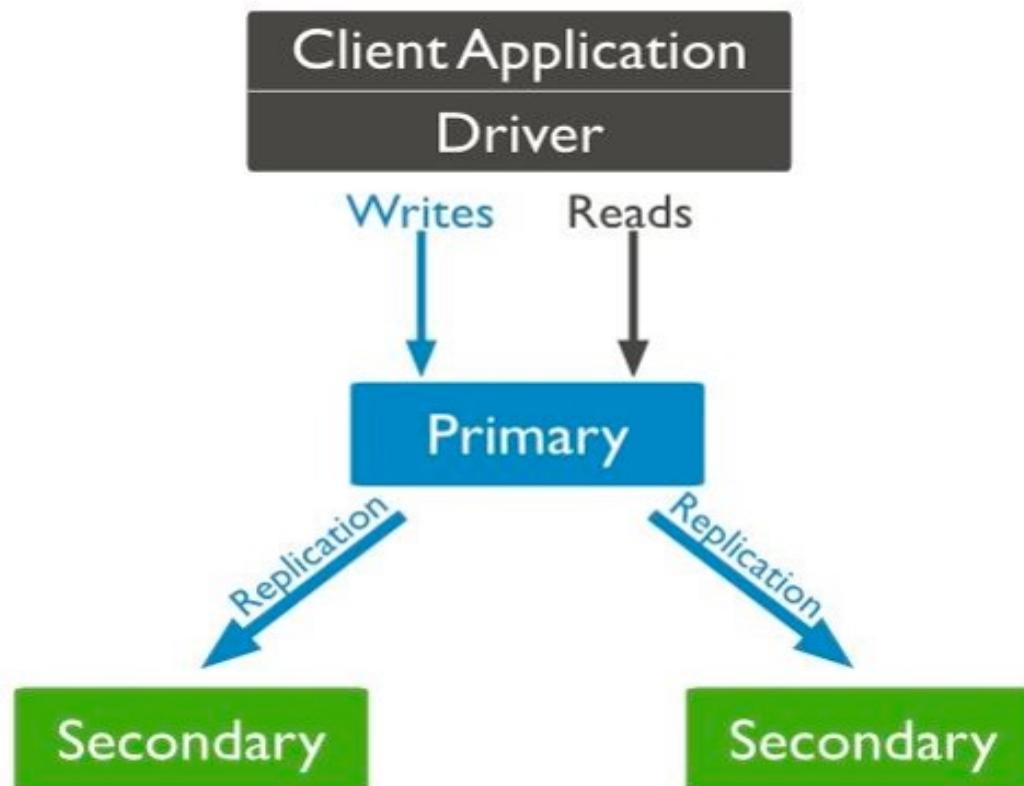
Embedded sub-document

Embedded sub-document



MONGODB ARTILARI

- **Replication:** MongoDB'nin herhangi bir server hatası durumunda kendini güvenceye alması. RDBMS sistemlerdeki Disaster Recovery'ler benzeri olarak da düşünülebilir. Ana sunucunun herhangi bir sorunla karşılaşması tehlikesine karşın yedek sunucular ile verilerin yedeklenmesi ve ana sunucuda sorun gerçekleştiğinde yedek sunucunun ana sunucu görevini üstlenip veri kayıplarını engellemesi için gerekli bir özelliktir.





MONGODB ARTILARI

- **Master-Slave Replication desteği:** Yazma ve okuma işlemlerini ayrı sunuculara yönlendirebilme. Replication özelliğinin bir alt özelliği olarak da düşünülebilir. Yine aynı şekilde ana sunucu ve yedek sunucular oluşturulabilir. Burada ek olarak yazma ve okuma işlemleri ayrı sunuculardan yapılabilir böylelikle sunucular üzerindeki trafik ve yük azaltılacağından performans açısından önemli bir artış daha sağlanabilir.



MONGODB ARTILARI

- **Sharding desteği**: Büyük ölçekli verilerin sunucular arasında paylaşılması özelliği. MongoDB de performansın en öncelikli konusu budur. bu nedenle bazen veriler çok büyük boyutlara ulaştığında tek bir sunucu artık bizim için yetersiz bir hal alabilir bu nedenle yeni sunucular ile yatay büyümeye giderek veriler bu sunuculara dağıtilır ve yük azaltılmış olur.

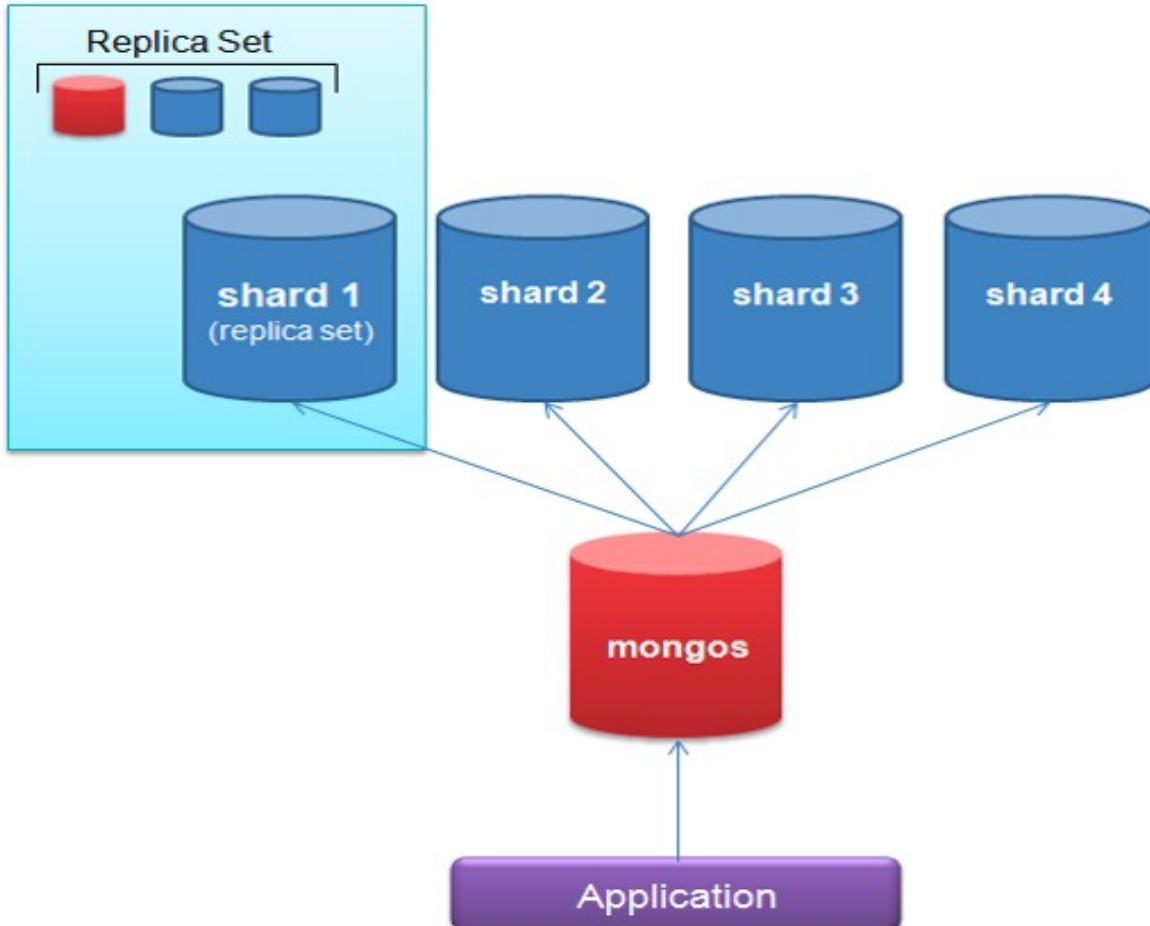


MONGODB ARTILARI

- **Dikey büyümeye**: Sunucu özelliklerini artırma. Burada elimizde bulunan mevcut sunucunun özellikleri(CPU, Memory, Disk vb.) artırılır. Bu bi açıdan her konuda çok maliyetli olabilir ve bir sunucunun özelliklerininin ne kadar artırabilirsiniz ki ? eğer küçük veriler üzerinden çalışıyorsanız belki orta ölçekli bir sunucu işinizi görebilir fakat çok büyük verilere sahip sistemler düşünüldüğünde dikey büyümeye aşırı bir uğraş ve maliyet gerektirebilir.
- **Yatay büyümeye**: Sunucu artırma. Mesela techpro nun verilerinin bulunduğu tek bilgisayar olursa öğrenciler yüklenliğinde sıkıntı olur, bir bilgisayar daha almak lazım. Bu işlemde ise mevcut sunucunuz üzerinde herhangi bir değişiklik ve özellik artırımı yapmakla uğraşmadan aynı özelliklerde ya da ihtiyaç doğrultusunda farklı özelliklerde bir sunucuyu daha sunucunuza bağlayarak ihtiyacınızı giderebilirsınız.



MONGODB ARTILARI





MONGODB ARTILARI

- 1 shard 1 veya daha fazla aralıktan sorumludur

['a', 'f')

Shard 1

['a', 'f)
['f', 'j')

Shard 2

['f', 'j')
['j', 'o')

Shard 3

['o', '{')

Shard 4

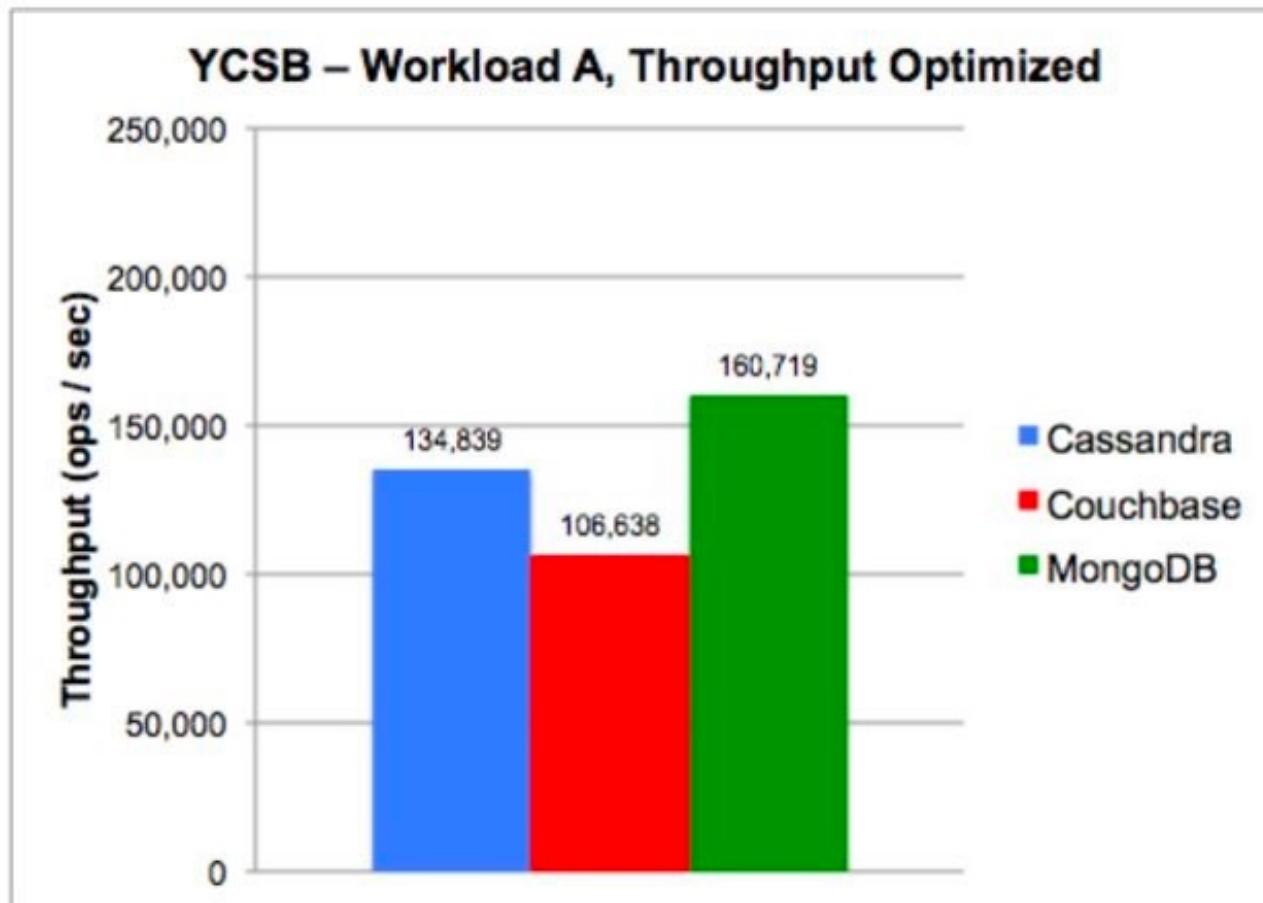
- MongoDB'nin kullandığı yöntem



MONGODB ARTILARI

- Aşağıdaki görselde üç Database'de yapılan %50 okuma ve %50 yazma işlemi baz alınmış ve saniyede yapılan işlem sayısı verilmiştir.

Workload A (50% read, 50% update)





MONGODB ARTILARI

- Buradaki görselde ise bu işlemlerin yapılış sırasında performans süreklilığı gözlemlenmiştir. Bakıldığı zaman iki veri tabanında işlem uzadıkça bir süre sonra işlemi gerçekleştirmeye süresi artmakte ve performansta yavaşlama olmaktadır fakat MongoDB sürekli aynı performans ile çalışmaya devam etmektedir.

YCSB (Latencies) – Workload A, Throughput Optimized		
	99th (Read)	99th (Update)
Cassandra	4ms	3ms
Couchbase	<1ms	3ms
MongoDB	1ms	1ms



MONGODB ARTILARI

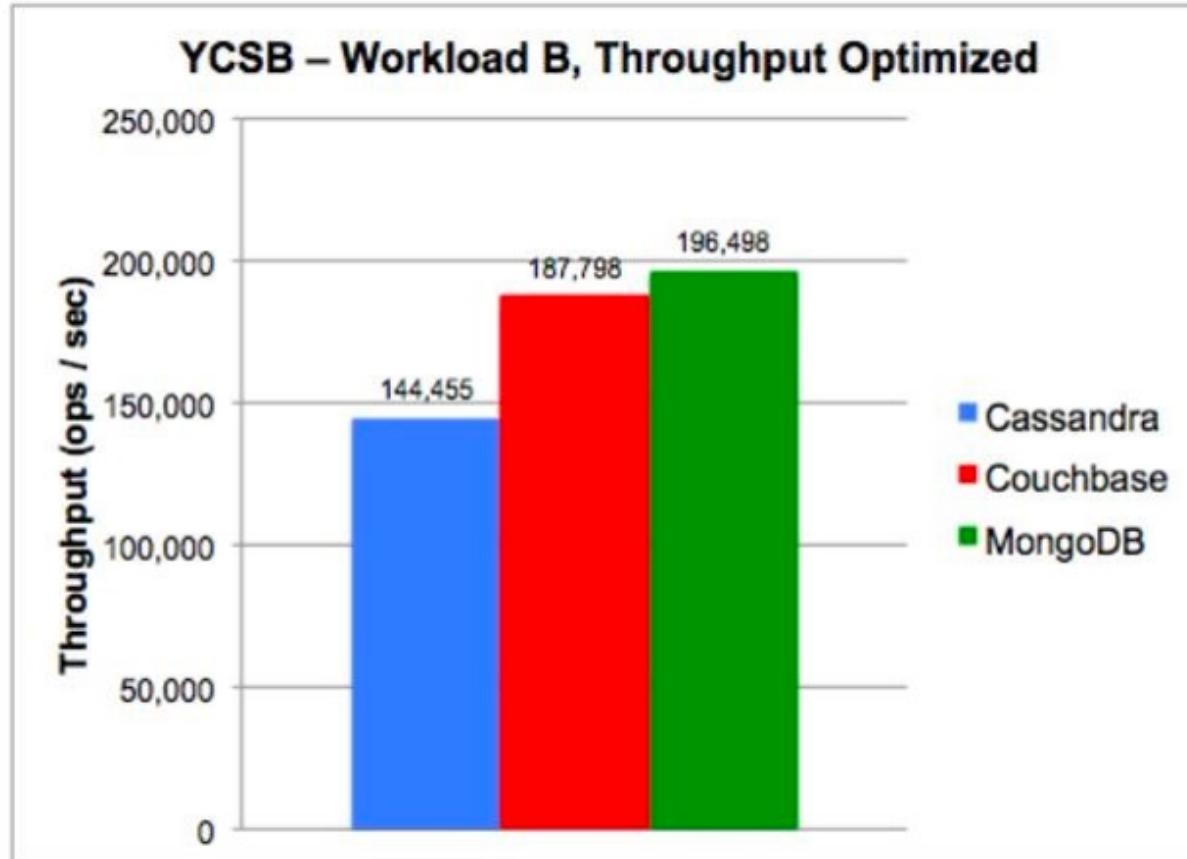
- Yine bir işlemde %95 okuma ve %5 yazma işleminde MongoDB'nin saniye bazında daha fazla işlem yaptığı görebiliriz.
- Aynı şekilde performans sürekliliğini de hiç hiç kaybetmemektedir.

YCSB (Latencies) – Workload B, Throughput Optimized		
	99th (Read)	99th (Update)
Cassandra	1ms	1ms
Couchbase	2ms	5ms
MongoDB	1ms	1ms



MONGODB ARTILARI

Workload B (95% read, 5% update)





Sql'de ve MongoDB'de tanımlama farklılıkları

SQL	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field
Index	Index



MONGODB ARTILARI

MongoDB vs MySQL

MySQL term	Mongo term
Database	Database
table	Collection
index	Index
row	BSON document
column	BSON field
join	embedding and linking
primary key	_id field
group by	aggregation



Sorgulama

MySQL	MongoDB
select * from arkadas_listesi;	db.arkadas_listesi.find({});
select isim from arkadas_listesi;	db.arkadas_listesi.find({}, {isim:1, _id:0});
select * from arkadas_listesi where isim = "Doğan";	db.arkadas_listesi.find({isim:"Doğan"});
insert into arkadas_listesi(isim,soyisim,telefon) values('Doğan','Aydın',7845759);	db.arkadas_listesi.save({isim:'Doğan', soyisim:'Aydın', telefon:7845759});
update arkadas_listesi set isim="Haktan" where isim="Doğan"	db.arkadas_listesi.update({isim:"Doğan"}, {\$set:{isim:"Haktan"}});
delete from arkadas_listesi;	db.arkadas_listesi.remove({});
delete from arkadas_listesi where isim="Doğan";	db.arkadas_listesi.remove({isim:"Doğan"});
select * from arkadas_listesi where isim in ("Doğan","Can");	db.arkadas_listesi.find({isim:{\$in:["Doğan","Can"]}});
select * from arkadas_listesi where isim = "Doğan", limit 4,12;	db.arkadas_listesi.find({isim:"Doğan"}).skip(4).limit(12);
select * from arkadas_listesi where yas > 20;	db.arkadas_listesi.find({yas:{\$gt:20}});
select * from arkadas_listesi where yas < 20;	db.arkadas_listesi.find({yas:{\$lt:20}});



Sorgulama

SQL	MONGO
WHERE	\$match
GROUP BY	\$group
HAVING	\$match
SELECT	\$project
ORDER BY	\$sort
LIMIT	\$limit
SUM()	\$sum
COUNT()	\$count, \$sum
JOIN	\$lookup
MERGE	\$merge (MongoDB 4.2 ve sonrası)

Yeni bir tablo nasıl oluşturulur?

SQL	MongoDB
<pre>CREATE TABLE people (id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30), age Number, status char(1), PRIMARY KEY (id))</pre>	<pre>db.createCollection("people") db.people.insertOne({ user_id: "abc123", age: 55, status: "A" })</pre>

Yeni bir değer nasıl eklenir?

SQL	MongoDB
<pre>INSERT INTO people(user_id, age, status) VALUES ("bcd001", 45, "A")</pre>	<pre>db.people.insertOne({ user_id: "bcd001", age: 45, status: "A" })</pre>

Arama nasıl yapılır?

SQL	MongoDB
<code>SELECT * FROM people</code>	<code>db.people.find()</code>

Özelleştirilmiş arama nasıl yapılır?

SQL	MongoDB
<pre>SELECT id, user_id, status FROM people</pre>	<pre>db.people.find({}, { user_id: 1, status: 1 })</pre>
<pre>SELECT user_id, status FROM people</pre>	<pre>db.people.find({}, { user_id: 1, status: 1, _id: 0 })</pre>
<pre>SELECT * FROM people WHERE status = "A"</pre>	<pre>db.people.find({ status: "A" })</pre>

Güncelleme nasıl yapılır?

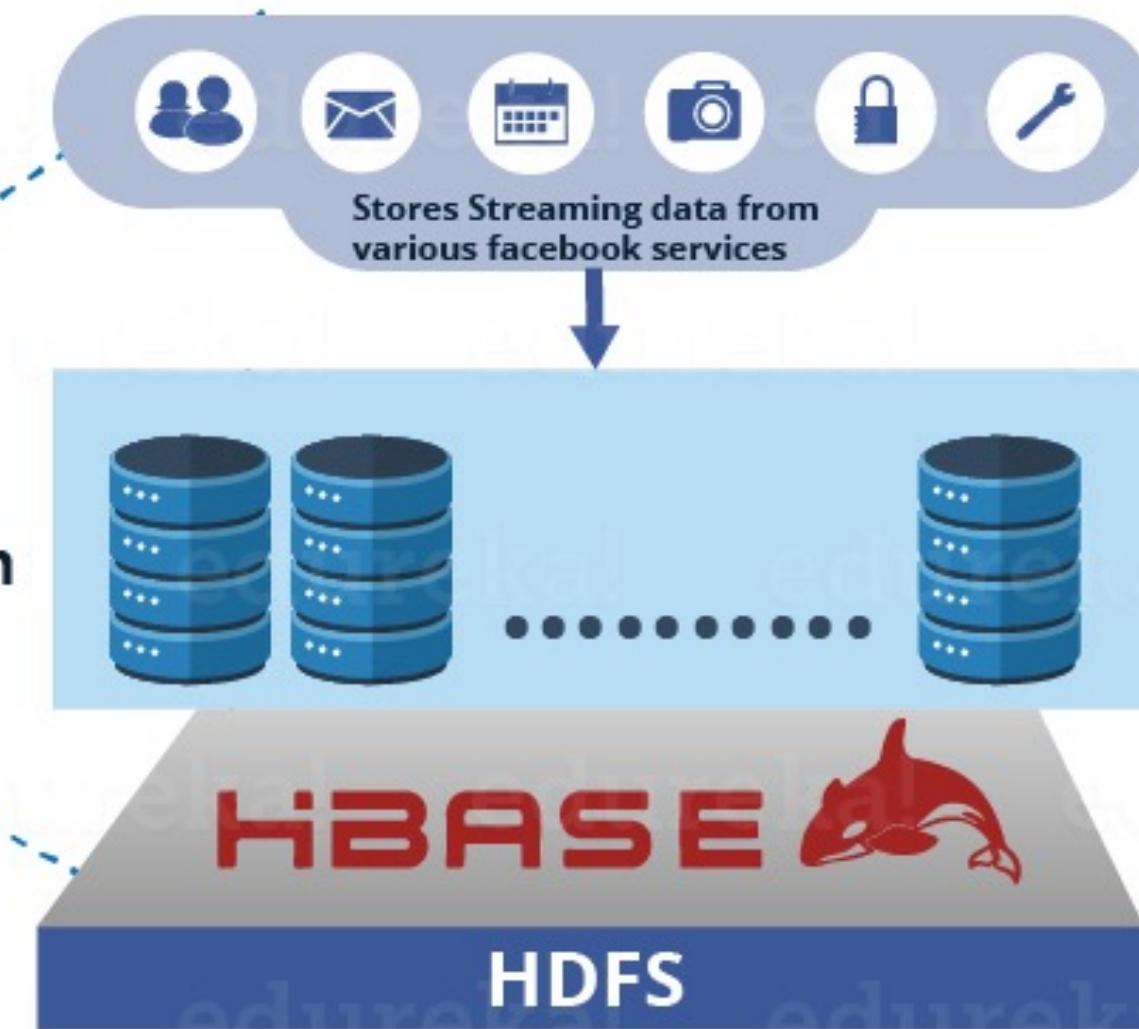
SQL	MongoDB
UPDATE people SET status = "C" WHERE age > 25	db.people.updateMany({ age: { \$gt: 25 } }, { \$set: { status: "C" } })
UPDATE people SET age = age + 3 WHERE status = "A"	db.people.updateMany({ status: "A" }, { \$inc: { age: 3 } })

Silme nasıl yapılır?

SQL	MongoDB
DELETE FROM people WHERE status = "D"	db.people.deleteMany({ status: "D" })
DELETE FROM people	db.people.deleteMany({})

edureka!

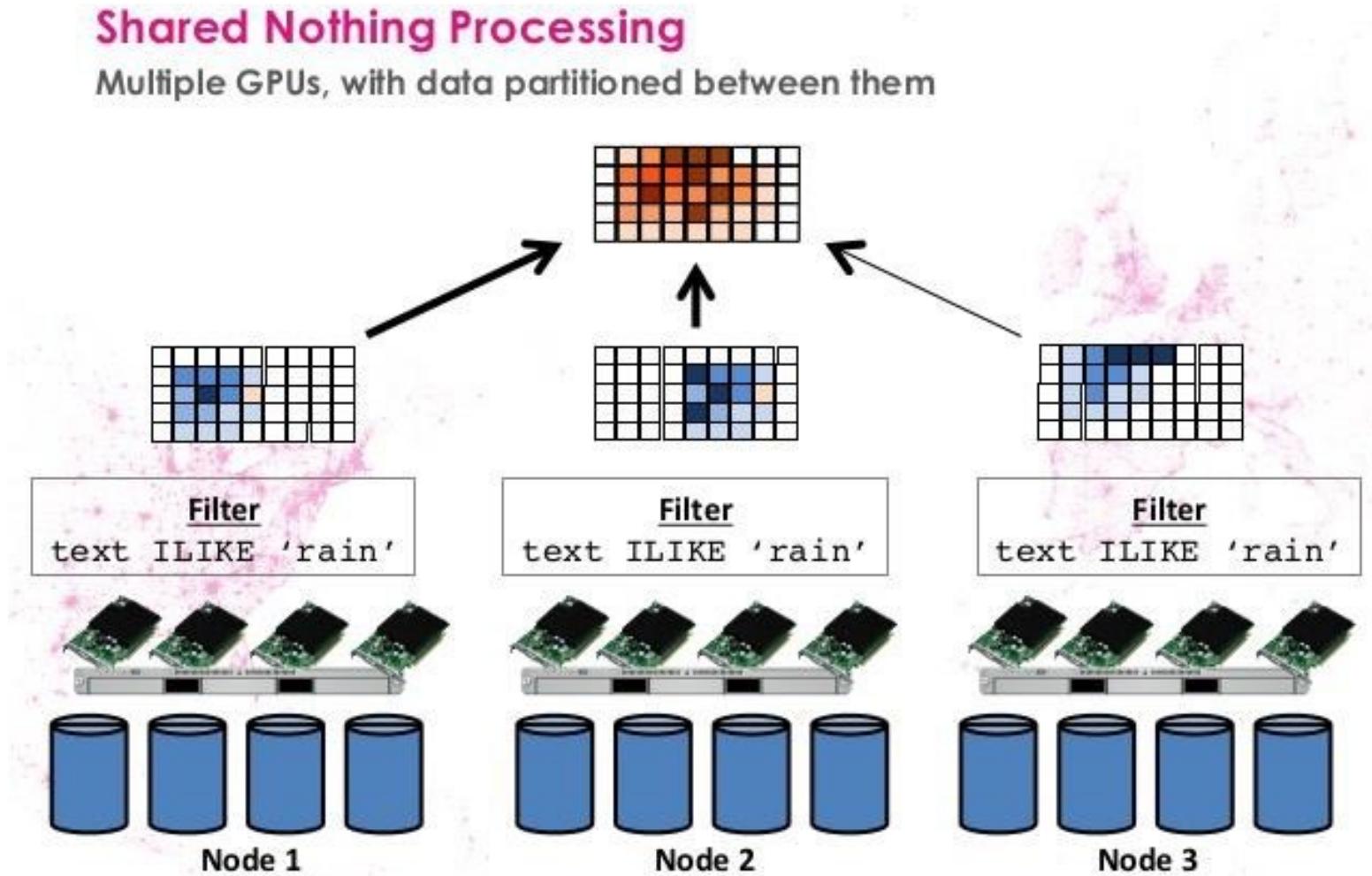
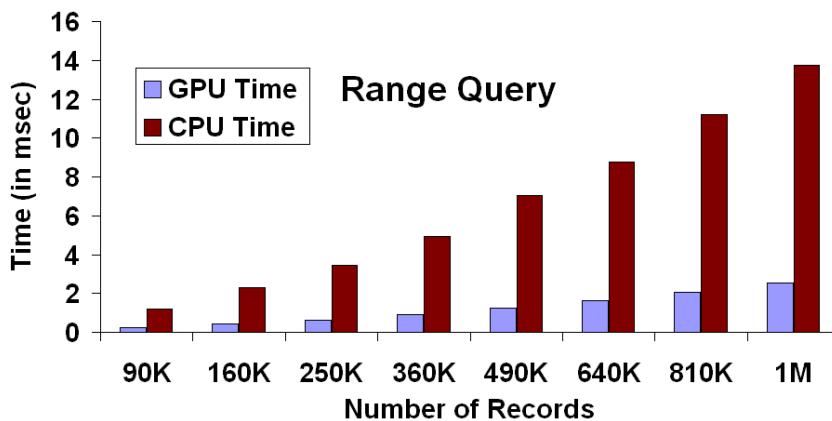
Serving
Requests



Solution

Ek Bilgi: GPU üzerinde çalışan veri tabanları

- MapD
- Kinetica
- BlazingDB
- Blazegraph
- PG-Strom



Genel Tekrar

	SQL	NoSQL
Type	Relational	Non-Relational
Data	Structured Data stored in Tables	Un-structured stored in JSON files but the graph database does supports relationship
Schema	Static	Dynamic
Scalability	Vertical	Horizontal
Language	Structured Query Language	Un-structured Query Language
Joins	Helpful to design complex queries	No joins, Don't have the powerful interface to prepare complex query
OLTP	Recommended and best suited for OLTP systems	Less likely to be considered for OLTP system
Support	Great support	community dependent, they are expanding the support model
Integrated Caching	Supports In-line memory(SQL2014 and SQL 2016)	Supports integrated caching
flexible	rigid schema bound to relationship	Non-rigid schema and flexible
Transaction	ACID	CAP theorem
Auto elasticity	Requires downtime in most cases	Automatic, No outage required

Kaynaklar

- «No-SQL VERİ TABANLARI ÜZERİNDE BİR METİN MADENCİLİĞİ UYGULAMASI» Gürkan AYDIN'ın Yüksek Lisans Tezi
- <http://devveri.com/nosql-nedir>
- <https://medium.com/turkce/cap-teoremi-nedir-49a23e6d2e10>
- <http://ksat.me/a-plain-english-introduction-to-cap-theorem/>
- <https://kodcu.com/2013/04/nosql-kavrami-ve-mongodb/>
- <https://kodcu.com/2016/03/nosql-veritabanlari/>
- <https://kodcu.com/2016/03/hangi-nosql-veritabani/>