MongoDB Kullanım Dokümanı

Bilgilendirme Bölümü

Bu bölümde MongoDB öğrenmeye başlamadan önce gerekli olan temel bilgileri, MongoDB nedir? Ve NoSQL ne anlama gelir gibi bazı soruların cevabını vereceğiz. Ayrıca MongoDB’nin içerisinde bulunan depolama yapısında, SQL veritabanına karşılık gelen elementlerin ne olduğunu anlatacağız.

NoSQL nedir?

Geçtiğimiz yıllar içerisinde saklanan verilerin boyutlarının çok yüksek boyutlara gelmesi ve sürekli olarak artmaya devam etmesi sonucunda SQL veritabanları yerine ortaya atılmış yeni bir fikir olduğunu söyleyebiliriz. SQL ve NoSQL arasındaki en büyük fark, SQL veritabanlarında veriler tablo ve sütunlarda saklanırken, NoSQL veritabanlarında veriler JSON tipinde bir yapıda tutulmaktadır.

Peki verileri neden bir tablo yerine JSON tipi bir yapıda tutmanın bizim için faydaları nelerdir? Bunlardan ilki performanstır. Okuma ve yazma işlemleri SQL veritabanlarına oranla çok daha hızlı gerçekleşir. Diğer bir faydası ise NoSQL veritabanları yatay olarak genişletilebilir olduklarından çok fazla sayıda sunucu bir arada çalışarak çok yüksek boyutlardaki veriler üzerinde işlem yapabilir.

MongoDB nedir?

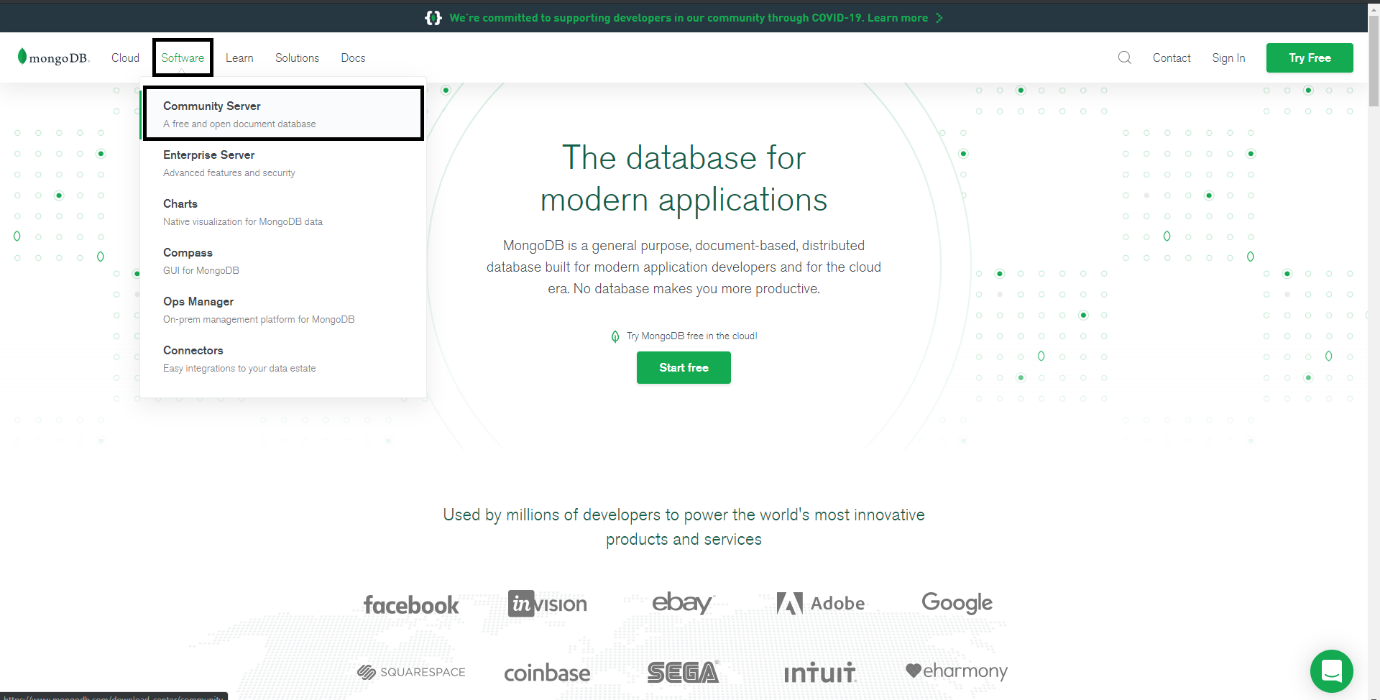
Son yıllarda popülerliğini oldukça arttıran MongoDB, 2009 yılında [MongoDB Inc..](https://tr.wikipedia.org/w/index.php?title=Vikipedi:MongoDB_Inc&action=edit&redlink=1) tarafından geliştirilen ölçeklenebilir, doküman tabanlı, NoSQL bir veritabanı uygulamasıdır. C++ ile geliştirilmiştir ve açık kaynaklıdır. İlk kararlı sürümü 8 Aralık 2015 yılında piyasaya sürülmüştür ve hala geliştirme süreci aktif olarak devam etmektedir. SQL veritabanlarında kullanılan tablolar yerine koleksiyonlar, satırlar yerine Dokümanlar, ve sütunlar yerine field(Türkçe tam karşılık bulamadığımdan böyle bıraktım) yapıları bulunur.

Kurulum

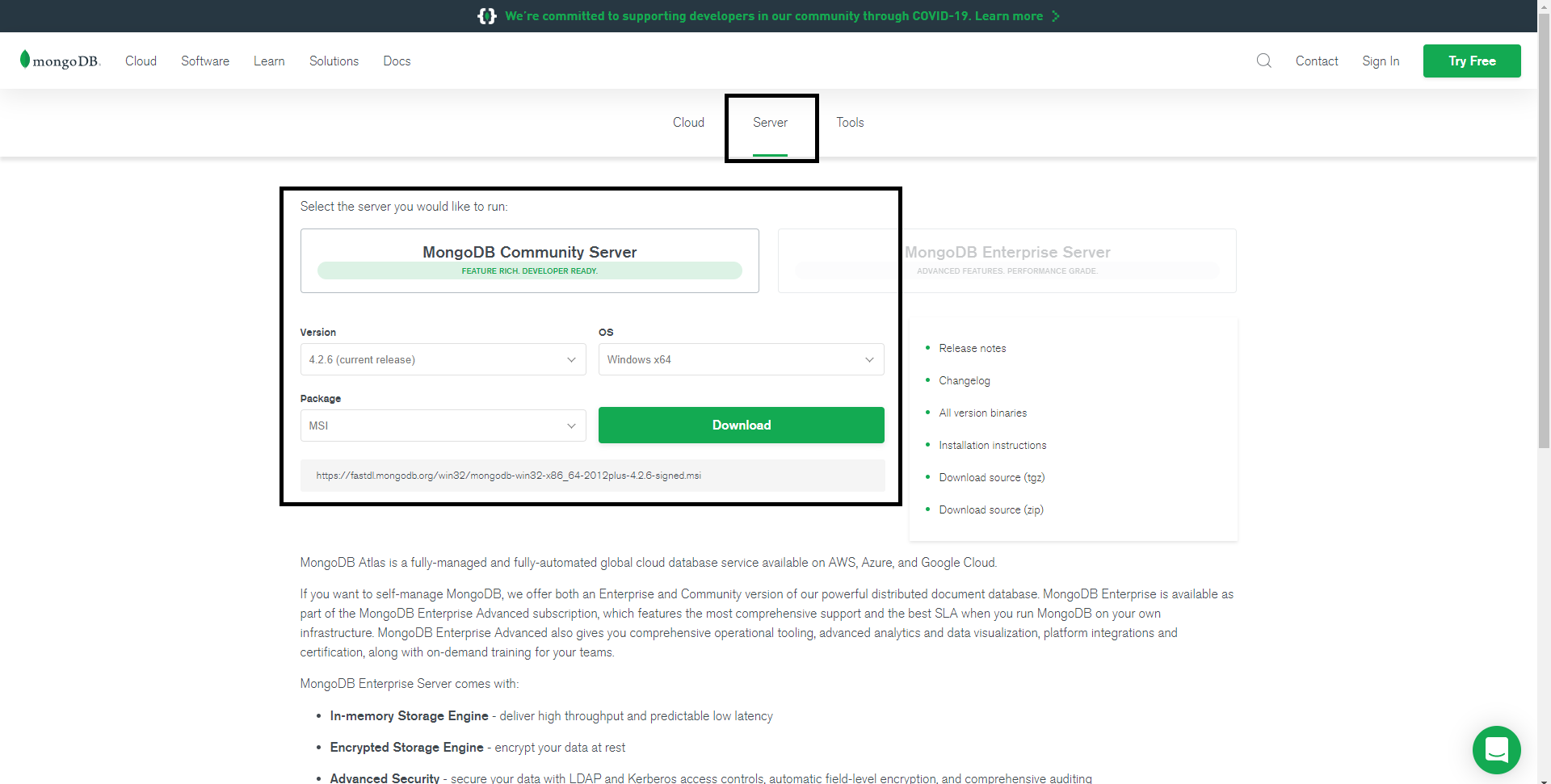
Burada anlatacağımız kurulum MongoDB Community edition’dur. Ücretsiz sürümü oldukça kısıtlı bir depolama alanı ve sınırlı performans ile geliyor olsa da veritabanını keşfetmemize ve temel kavramlarını anlamamıza rahatlıkla yetecek kaynağı sağlamaktadır. Kurulum, diğer veritabanı uygulamalarına kıyasla çok daha basittir.

Öncelikle uygulamayı indirebileceğiniz web sitesine ulaşmak için linki kopyalayıp tarayıcınızın adres çubuğuna yapıştırın veya linkin üzerine ctrl+LMB yaparak siteye ulaşın.

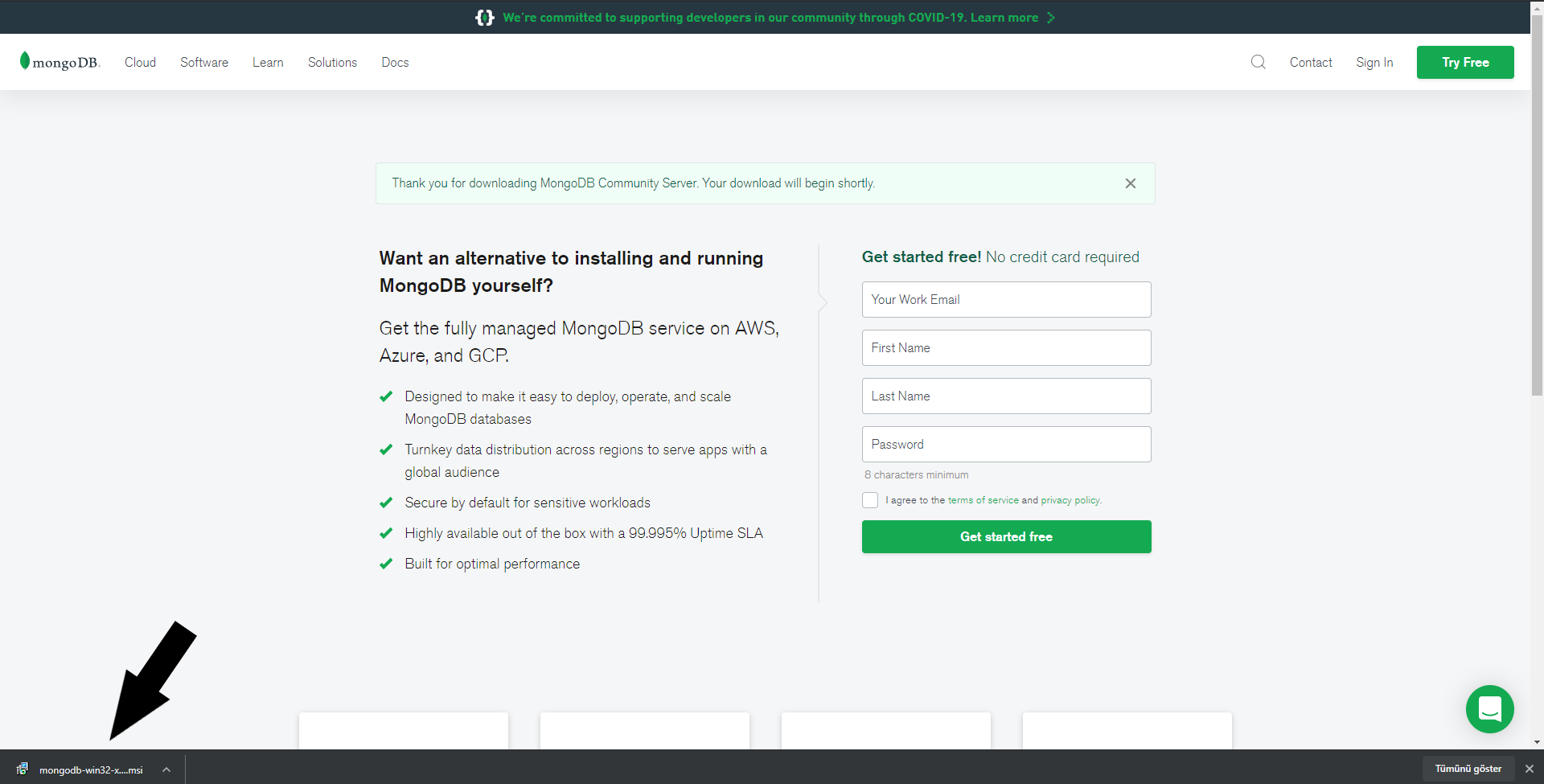
<https://www.mongodb.com/>



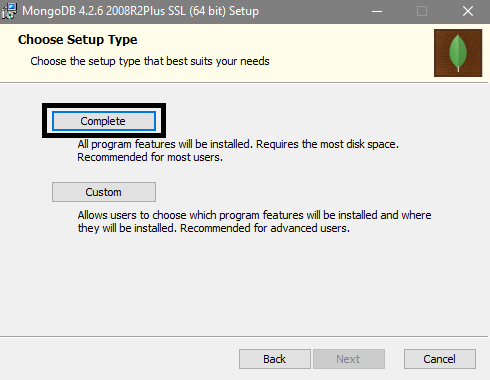
Daha sonra Software kategorisi altında bulunan Community server başlığına tıklayın. (ilk resim)



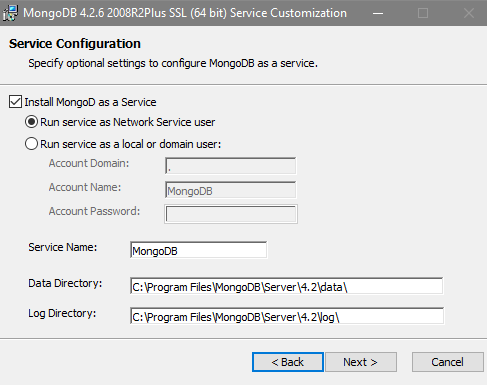
Karşınıza gelen ekranda Serverin seçili olduğundan emin olun ve kendi bilgisayarınıza uygun şekilde kutuları doldurup Download tuşuna basın.



İndirdiğiniz dosyayı açın.



Özel bir Kurulum yapmak istiyorsanız Custom seçeneğini seçebilir ve kendi ayarlarınızı kullanabilirsiniz. Complete tuşunu tıklayarak hızlı bir kurulum yapacağız.



Herhangi bir ayarda değişiklik yapmanız gerekmez. Next tuşuna basıp kuruluma başlatıyoruz.

Kurulum tamamlandı.

MongoDB veri modeli

MongoDB’deki veriler esnek bir yapıya sahiptir. Aynı alanlara sahip farklı belgeler aynı tür verileri içermek zorunda değildir. Farklı türde veriler içerebilirler.

MongoDB iki tür veri modeline sahiptir. İlk veri modeli türü gömülü veri modeli, İkinci veri modeli ise normalleştirilmiş veri modelidir. Bu veri modelleri, gereksinimlerinize göre dokümanlarınızı hazırlamanıza olanak sağlar.

**Gömülü veri modeli –** Bu veri modelinde birden fazla veriyi tek bir doküman içerisinde saklayabilirsiniz. Örneğin bir çalışanınız var ve çalışanınızın ismi, soyismi ve adresi gibi bilgileri kaydetmek istiyorsunuz fakat her bir veri kendi içinde daha fazla veri barındırıyor. Örneğin şu şekilde bir doküman oluşturabilirsiniz;

{

Personel:{

İsim:”Ömer Orçun”,

Soyisim:”Tükenmez”,

Yas:20,

},

İletişim\_bilgileri:{

e-mail:”calisan@hotmail.com”,

telefon:”9876543210”

},

Adres:{

il:”Sakarya”,

ilçe:”adapazarı”

}

}

Görüldüğü üzere tek bir doküman içerisinde birden fazla veriyi sakladık.

**Normalleştirilmiş veri modeli –** Bu veri modelinde asıl belgenin alt belgelerine ulaşmak için belgelere referanslar verilir. Örnek olarak yukarıda yazmış olduğumuz dokümanı bu veri modelini kullanarak yazalım.

**Çalışan numarası:**

{

\_id: <11Numara>,

calisan\_id: "1"

}

**Çalışanın bilgileri:**

{

\_id: <12Numara>,

hangiCalisan: "11Numara",

isim:"Ömer Orçun",

soyisim:"Tükenmez",

yas:20

}

**İletişim bilgileri:**

{

\_id: <13Numara>,

HangiCalisan: "11Numara",

e-mail: "calisan@hotmail.com",

telefon: "987654321"

}

**Adres bilgileri:**

{

\_id: <14Numara>,

HangiCalisan: “11Numara",

il: "Sakarya",

ilçe: "Adapazarı"

}

Her bir bilgi kümesi ayrı bir belgede tutuluyor ve bu ayrı belgelerin hepsi asıl belgenin sahip olduğu ID değerini kendi içerisinde barındırıyor. Bütün alt dosyalar asıl dosyayı işaret ediyor.

**Şema oluştururken dikkat edilmesi gereken unsurlar:**

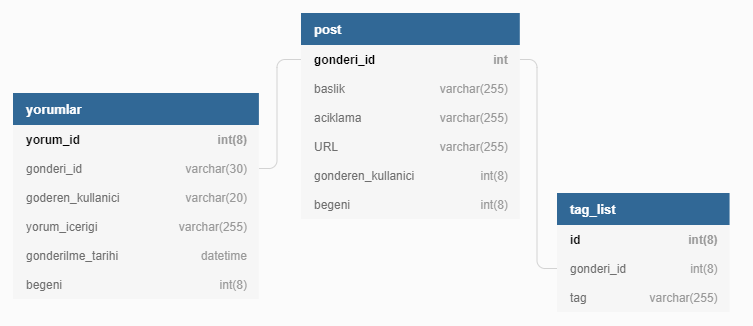
* Şemaları kullanıcıların gereksinimlerini tam olarak karşılayacak şekilde hazırlayın.
* Birlikte sıklıkla kullanılacak olan verilerin hepsini tek bir dokümanda toplamaya çalışın.
* Disk alanından feragat ederek performanstan kazanç sağlamak için verileri çoğaltın. Disk alanı, performanstan daha az değerli olduğundan bu durum avantajımıza olur.
* Birleştirme işlemlerini yazma esnasında yapmak, okuma esnasında yapmaktan daha avantajlı olur.
* En çok karşılaşılan durumları belirleyerek, bu durumla karşılaşıldığında daha hızlı işlem gerçekleştirilmesi için şemalarınızı optimize edin.
* Şemalarınızda Aggregation işlemini sıkça uygulayın.

**Örnek;**

Bir müşteri tarafından bir blog sitesi veritabanı şeması isteniyor ve gereksinimleri şu şekilde belirtiliyor.

* Her gönderi eşi olmayan bir başlığa sahiptir, ve her gönderinin açıklaması ve URL’si vardır.
* Her gönderi 1 veya daha fazla tag sahibi olabilir.
* Her gönderinin kim tarafından gönderildiği, adı ve toplam beğeni sayısı vardır.
* Her gönderinin yorumları vardır ve bu yorumların gönderen kişisi beğenileri ve ne zaman gönderildikleri verisi saklanır.
* Her gönderinin 0 veya daha fazla yorumu olabilir.

Örnek şemanın çizilmiş şekli;



Bu şemayı MongoDB ile oluşturmak istersek aşağıdaki gibi bir yol izlememiz gerekir;

{

\_id: gönderi\_id

baslik: gönderinin basligi,

aciklama: gönderinin aciklamasi,

gönderen: gönderiyi yollayan kisi,

url: gönderinin URL’si,

etiketler: [

TAG1,

TAG2,

TAG3

],

begeni: begeni sayisi,

yorumlar: [

{

kullanici: yorumu yapan kullanici,

mesaj: mesajin icerigi,

gönderilme\_tarih: tarih,

begeni: begeni sayisi

},

{

kullanici: yorumu yapan kullanici,

mesaj: mesajin icerigi,

gönderilme\_tarih: tarih,

begeni: begeni sayisi

}

]

}

Minimum kaynak kullanılarak oluşturulabilecek şema bu şekildedir.

MongoDB Kullanımı

Yapacağımız işlemler MongoDB Shell üzerinden gerçekleştirilecektir. MongoDB Shell’i açmak için, eğer ayarlarda değişiklik yapmadıysanız C:\Program Files\MongoDB\Server\4.2\bin dosyasında Mongo.exe dosyasıdır.

MongoDB, sahip olduğunuz veritabanlarının ismini görüntüemek için “show dbs” komutunu kullanır. Bu komut ile sahip olduğunuz bütün veritabanları ekranda gösterilecektir.

**Use komutu**

MongoDB yeni bir veritabanı oluşturmak veya var olan bir veritabanını seçmek için “use <veritabani\_ismi>” komutunu kullanır. Komutta belirttiğiniz veritabanı daha önceden oluşturulmuşsa bu veritabanı seçilir, ve bundan sonra belirtilen veritabanı üzerinde işlem yapılır. Eğer belirtmiş olduğunuz isimde bir veritabanı daha önceden oluşturulmamışsa, otomatik olarak bu veritabanı oluşturulur ve işlemlere bu veritabanı üzerinden devam edilir.

use deneme

Şuanda daha önceden deneme adında bir veritabanı oluşturmadığımız için yeni bir veritabanı oluşturuldu ve onun üzerinde işlem yapabiliriz. Seçmiş olduğumuz veritabanını görmek için

db

Bu komutu yazdıktan sonra ekranımıza “deneme” yazısı gelecektir. “db” komutu seçili olan veritabanını gösterir. Eğer tüm veritabanlarının bir listesini görmek istersek kullanacağımız komut

show dbs

Bu komutu kullandığınızda az önce oluşturmuş olduğunuz veritabanı listede yer almayacaktır. Bunun sebebi oluşturduğumuz veritabanının içinde herhangi bir veri bulunmamasıdır. Öyleyse oluşturduğumuz veritabanına bir veri ekleyelim.

db.books.insert({name:”Beyaz Diş”})

Şimdi Show dbs Komutunu kullandığımızda sonuç:

>show dbs

local 0.78125GB

deneme 0.23012GB

test 0.23012GB

MongoDB veritabanının varsayılan veritabanı “test” veritabanıdır. Eğer hiçbir veritabanı oluşturmadıysanız koleksiyonlarınız burada yer alacaktır.

**dropDatabase() komutu**

Daha önce oluşturduğumuz bir veritabanını silmek istersek bunu “db.dropDatabase()” komutunu kullanırız.

db.dropDatabase()

örnek olarak oluşturduğumuz deneme veritabanını silmek istersek öncelikle deneme veritabanını “use deneme” komutunu kullanarak seçmemiz gerekir. Daha sonra db.dropDatabase() komutunu kullanarak seçili olan veritabanını silmiş oluruz. Eğer hiçbir veritabanı seçili değilse varsayılan olarak gelen test veritabanı silinir.

**createCollection() komutu**

MongoDB, koleksiyon oluşturmak için “db.createCollection()” komutunu kullanır. Parantez içerisinde oluşturulmak istenen koleksiyonun ismi ve isteğe bağlı olarak özellikleri belirtilir.

db.createCollection(isim, ozellik)

Koleksiyonunuza isim vermek için “isim” yazan yere koleksiyonun ismini ve “ozellik” yazan yere dokümanın sınırlamalarını ve doküman tipini belirtebiliriz. Kullanabileceğiniz örnek parametreler aşağıda verilmiştir.

|  |  |  |
| --- | --- | --- |
| **Field** | **Tip** | **Tanım** |
| capped | Boolean | İsteğe bağlı bir seçenektir. Bu parametreyi kulanarak koleksiyonunuzda bulunan maksimum doküman sayısını sınırlandırabilirsiniz. Maksimum doküman sayısına ulaşıldığında, yeni bir doküman eklemek için en eski doküman otomatik olarak silinir. Bu ayarı True yaparsanız, ayrıca dokümanın alabileceği maksimum doküman sayısını da ayrıca belirtmeniz gerekir. |
| autoIndexId | Boolean | İsteğe bağlı bir seçenektir Eğer True yaparsanız, her bir dokümana otomatik olarak bir index değeri ekleyecektir. Varsayılan değeri False olarak gelir. |
| size | number | İsteğe bağlı bir seçenektir Koleksiyonunuza Byte cinsinde n bi maksimum boyut vermenizi sağlar. True değeri aldığında alabileceği maksimum değeri ayrıca belirtmeniz gerekir. |
| max | number | (Optional) Sınırlandırılmış bir koleksiyonda izin verilen maksimum doküman sayısını belirtir. |

Bildiklerimizi örnek üzerinde gösterelim. İlk önce sıradan bir koleksiyon oluşturalım.

db.createCollection(“Merhaba”)

Oluşturduğunuz koleksiyonu görmek isterseniz aşağıdaki komutu kullanabilirsiniz.

show collections

Şimdi ise yeni bir koleksiyon oluşturalım ve oluşturacağımız koleksiyonu sınırlandırarak yeni özellikler verelim.

db.createCollection("CappedCollection", { capped : true, autoIndexID : true, size :3229110, max : 9870 } )

Herhangi bir koleksiyona doküman eklediğimizde, eğer olmayan bir koleksiyona ekleme yapmaya çalışıyorsak, MongoDB otomatik olarak koleksiyonu oluşturur ve ekleme işlemini gerçekleştirir.

**drop() komutu**

MongoDB, herhangi bir seçtiğimiz koleksiyonu silmek için”drop()” komutunu kullanır. Komutun örnek kullanımı şu şekildedir;

db.kolekssiyon\_ismi.drop()

örnek bir kullanım gerçekleştirelim. Öncelikle kullandığımız veritabanı içerisinde bir koleksiyon oluşturalım. Ve daha sonra “show collections” komutuyla oluşturduğumuz koleksiyonu görelim.

use deneme

db.createCollection(“silmeIslemi”)

show collections

deneme veritabanına ait “silmeIslemi” isimli bir koleksiyon oluşturduk. Şimdi “drop()” komutunu örnekteki gibi kullanarak silelim. İşlem bittikten sonra tekrar “show collections” komutunu kullanarak oluşturmuş olduğumuz koleksiyonun artık olmadığını görebiliriz.

db.silmeIslemi.drop()

show collections

**Koleksiyonlara doküman ekleme ve veri tipleri**

MongoDB veritabanında doküman ekleme işlemlerine başlamak için öncelikle bazı temel veri tiplerini bilmeliyiz. Eklemek istediğimiz dokümanlar hangi veri tiplerinde olabilir bunları bilmeden ekleme işlemini istediğimiz gibi yapamayız. MongoDB veritabanında depolama yapabileceğiniz veri tipleri şunlardır;

* **String –** En çok kullanılan veri tipidir. MongoDB ile işlem yaparken **UTF-8** karakter seti kullanılması **zorunludur**.
* **Integer –** Sayısal ifadeleri kapsayan veri tipidir.
* **Boolean –** Boolean veri tipi doğru veya yanlış (True/False) değerlerinden birini alır.
* **Double –** Tamsayı olmayan, buçuklu sayıları kapsayan veri tipi.
* **Min/Max Keys–** Bu veri tipi, bir değeri en düşük ve en yüksek BSON elemanlarıyla karşılaştırmak için kullanılır.
* **Arrays –** Herhangi bir dizi veya listeyi tek bir değişken içerisinde depolamak için kullanılır.
* **Timestamp –** Tarih bilgisini içinde tutan veri tipi.
* **Object –** Bu veri tipi gömülü dokümanlar için kullanılır.
* **Null –** Boş değer.
* **Date –** Tarih değişkenini gün ay yıl şeklinde içinde tutabilir.
* **Object ID –** Dokümanların kendine özel eşsiz seri numarası.
* **Binary –** ikili sayı sisteminde kodlanmış şekilde saklanan verileri tutar.

**insert komutu**

MongoDB koleksiyonunuza veri eklemek için “Insert()” komutunu kullanmalısınız. Örnek kullanım şudur;

db.koleksiyon\_ismi.insert(doküman)

örnek olarak “deneme” veritabanı altında bulunan “books” koleksiyonuna bir kitap ekleyelim.

use deneme

db.books.insert({

ad: “kitapAdi”,

yazar: “yazarAdi”,

sayfaSayisi: 51

})

Ayrıca “insert()” komutu kullanılarak bir dizi dokümanı aynı anda ekleyebiliriz. Örnek kullanım;

db.books.insert([

{

Ad: “kitap1”,

Yazar: “yazar1”,

sayfaSayisi: 1

begeni\_yorum\_sayisi: [61, 93]

},

{

Ad: “Kitap2”,

Yazar: “yazar2”,

sayfaSayisi: 2,

begeni\_yorum\_sayisi: [129, 35]

}

])

**insertOne() komutu**

Eğer yalnızca bir dokümanı koleksiyona eklemek istiyorsanız bu metodu kullanabilirsiniz. Kullanım şekli;

db.koleksiyon\_ismi.insertOne(doküman)

Aşağıda “insertOne()” komutu kullanılarak “books” koleksiyonuna tek bir doküman eklenmiştir. Örnek kullanım;

db.books.insertOne({

ad: “kitap3”,

yazar: “yazar3”,

sayfaSayisi: 3,

begeni\_yorum\_sayisi: [45, 145]

})

**insertMany() komutu**

Bu komut kullanılarak birden fazla doküman liste şeklinde verilerek aynı anda ekleme işlemi gerçekleştirilebilir. Verilen örnekte 3 adet veri tek seferde koleksiyona eklenmiştir.

db.books.insertMany([

{

Ad: “kitap4”,

Yazar: “yazar5”,

sayfaSayisi: 5

begeni\_yorum\_sayisi: [76, 193]

},

{

Ad: “Kitap5”,

Yazar: “yazar5”,

sayfaSayisi: 5,

begeni\_yorum\_sayisi: [293, 3]

},

{

Ad: “Kitap6”,

Yazar: “yazar6”,

sayfaSayisi: 6,

begeni\_yorum\_sayisi: [21, 42]

}

])

**find() komutu**

MongoDB koleksiyonlarınızda arama yapmak için kullanacağınız komuttur. Birçok parametre alabilir ve aramalar bu parametreler ile koşullandırılabilir. En basit kullanımı aşağıdaki gibidir;

db.koleksiyon\_ismi.find()

Eğer hiçbir parametre girilmeden “find()” komutu kullanılırsa koleksiyon altında bulunan tüm dokümanlar listelenecektir. Şuana kadar “books” koleksiyonuna birçok ekleme yapmıştık. Şimdi bu yaptığmıız eklemeleri görmeyi deneyelim.

db.books.find()

bu komutu yazdığımızda karşımıza şuana kadar eklemiş olduğumuz tüm veriler listelenir.

**pretty() komutu**

“find()” metodu kullanılarak karşımıza getirilen sonuçların göze daha hoş gelecek biçimde listelenmesini sağlar. İsteğe bağlı olarak kullanabileceğiniz bir komuttur. En basit yazılış biçimi şudur;

db.koleksiyon\_ismi.find().pretty()

örnek olarak daha önce arama yaptığımız “books” koleksiyonu üzerindeki arama sonuçlarımızda kullanarak normal listeleme şekli ile “pretty()” komutunun kullanılmış şekliyle listeleme sonucunu karşılaştıralım. Yazacağımız komut;

db.books.find().pretty()

**findOne() komutu**

kullanıldığında şartlara uygun bulunan ilk doküman dışında herhangi bir doküman listelenmez. Sadece tek bir doküman kullanıcıya gösterilir. Yazım şekli şudur;

db.koleksiyon\_ismi.findOne()

örnek bir kullanım için “books” koleksiyonu içerisinde bulunan ilk dokümanı listeleyelim. Örnek kullanım;

db.books.findOne()

**MongoDB üzerinde arama yapılırken kullanılabilecek parametreler**

MongoDB ile her zaman bütün dokümanları görmek istemeyiz. Bunun yerine belli özelliklere sahip özel dokümanları arayıp bulmak isteriz. Bunun için “find()” komutu ile beraber kullanılabilecek parametrelerin listesi sonraki tabloda açıklanmıştır.

|  |  |  |
| --- | --- | --- |
| **Operation** | **Syntax** | **Example** |
| Eşitlik | {<key>:{$eg;<value>}} | db.books.find({"by":"tutorials point"}).pretty() |
| Küçüktür | {<key>:{$lt:<value>}} | db.books.find({"sayfaSayisi":{$lt:1}}).pretty() |
| Küçük Eşittir | {<key>:{$lte:<value>}} | db.books.find({" sayfaSayisi ":{$lte:2}}).pretty() |
| Büyüktür | {<key>:{$gt:<value>}} | db.books.find({" sayfaSayisi ":{$gt:2}}).pretty() |
| Büyük Eşittir | {<key>:{$gte:<value>}} | db.books.find({" sayfaSayisi ":{$gte:2}}).pretty() |
| Eşit Değildir | {<key>:{$ne:<value>}} | db.books.find({" sayfaSayisi ":{$ne:2}}).pretty() |
| Listede İstenen Değerler | {<key>:{$in:[<value1>, <value2>,……<valueN>]}} | db.books.find({"name":{$in:[“kitap1", "kitap3", "kitap6"]}}).pretty() |
| Listede İstenmeyen Değerler | {<key>:{$nin:<value>}} | db.books.find({"name":{$nin:["kitap1", "kitap4"]}}).pretty() |

**MongoDB AND operatörü**

MongoDB koleksiyonlarımız içerisinde arama yaparken bazen birden fazla koşulla arama yapmamız gerekebilir. Bu durumda iki koşulu birbirine “and” operatörü ile bağlayarak istediğimiz aramayı gerçekleştirebiliriz. Örnek yazım şekli şu şekildedir;

db.books.find({ $and :[ {<key1>:<value1>}, { <key2>:<value2>} ] })

Örnek olarak hem adı kitap1 olan, hemde sayfa sayısı 1 olan öğeyi bulalım. Örnek kod şudur;

db.books.find({ $and :[ {“ad”:”kitap1”}, { “sayfaSayisi”:1} ] })

Örnek komutu çalıştırdığımızda karşımıza yalnızca hem adı kitap1 olan, hemde sayfa sayısı 1 olan kitaplar listelenecektir.

**MongoDB OR operatörü**

Yazım kuralları açısından and operatörüne oldukça benzemektedir. Belirtilen 2 veya daha fazla koşuldan en az bir koşulu sağlayan verilerin listelenmesi için kullanılır. Örnek yazılış;

db.koleksiyon\_ismi.find({ $or :[ {key1:value1}, {key2:value2} ] })

örnek olarak “books” koleksiyonunda bulunan verilerden, adı kitap1 olan veya sayfa sayısı 2 olan verileri listeleyelim. Örnek kod;

db.books.find({ $or :[ {“ad”:”kitap1”}, { “sayfaSayisi”:2} ] })

**AND ve OR operatörünü beraber kullanmak**

AND ve OR operatörlerinin birlikte kullanılmasıyla yapacağımız aramaların daah etkili olmasını sağlayabilir ve tam olarak istediğimiz verilere erişebiliriz. Örnek bir yazım aşağıda verilmiştir.

db.koleksiyon\_ismi.find({key1: {$gt:value1}, $or: [{key2: value2},

{key3: value3}]})

**NOT operatörü**

Arama sonucunun içermesini istemediğimiz verileri NOT operatörüyle belirterek aramada karşımıza gelmemesini sağlayabiliriz. Örnek yazım;

db.koleksiyon\_ismi.find({$not :[{key1:value1}, {key2:value2}]})

Bunun için bir örnek yapalım. Öncelikle not adında bir koleksiyon oluşturarak içerisine bazı veriler koyalım.

db.not.insertMany(

[

{

isim: "Radhika",

soyisim: "Sharma",

yas: "26",

e\_mail: "radhika\_sharma.123@gmail.com",

telefon: "9000012345"

},

{

isim: "Rachel",

soyisim: "Christopher",

yas: "27",

e\_mail: "Rachel\_Christopher.123@gmail.com",

telefon: "9000054321"

},

{

isim: "Fathima",

soyisim: "Sheik",

yas: "24",

e\_mail: "Fathima\_Sheik.123@gmail.com",

telefon: "9000054321"

}

]

)

Daha sonra ismi Rachel olmayan ve yaşı 24 olmayan kişilerin sonucu listeleyen kodu yazalım.

db.not.find({$not :[{isim:”Rachel”}, {yas:24}]})

**Update() komutu**

Bu komut kullanılarak seçilen dokümandaki veriler güncellenebilir. Yazım Örneği;

db.koleksiyon\_ismi.update(seçim\_kriterleri, güncel\_veri)

örnek olarak “not” koleksiyonu içerisinde bulunan, ismi Rachel olan verilerin isim kısmını Amanda olarak değiştirelim;

db.not.update( {“isim”:”Rachel”}, {$set:{“isim”:”Amanda”}})

İşlem gerçekleştirildikten sonra daha önce öğrendiğimiz find() komutunu kullanarak güncel verileri listeleyebilirsiniz. Varsayılan olarak MongoDB, bulmuş olduğu ilk doküman üzerinde update işlemini gerçekleştirir ve işleme devam etmez. Yani yalnızca tek bir doküman üzerinde değişiklik yapılır. Eğer birden fazla doküman üzerinde değişiklik yapmak istersek “multi” parametresine True değerini vermemiz gerekir. Örnek;

db.not.update( {“isim”:”Rachel”}, {$set:{“isim”:”Amanda”}}, {multi:true})

**findOneAndUpdate() metodu**

Bu metot seçilen dokümandaki verilerin güncellenmesi için kullanılır. Örnek yazım;

db.koleksiyon\_ismi.findOneAndUpdate(seçim\_kriterleri, güncel\_veri)

örnek olarak yine “not” koleksiyonu içerisinde bulunan, daha önceden ismini amanda olarak değiştirdiğimiz verinin ismini tekrar Rachel olarak güncelleyelim ve yaşını 30 yapalım. Örnek;

db.not.findOneAndUpdate( {“isim”:”Amanda”}, {$set:{“isim”:”Rachel”}, {yas:30} })

**updateOne() metodu**

filtre kullanarak bulduğunuz tek bir dokümanı güncellemenizi sağlayan metot. Örnek yazım;

db.koleksiyon\_ismi.updateOne(<filter>, <update>)

Örnek olarak ismi Rachel olan veriyi bulup, yaşını 25 olarak ayarlayalım.

db.not.updateOne( {“isim”:”Rachel”}, {$set:{“yas”:25}})

**updateMany() komutu**

Belirlenen koşullara uyan bütün verileri güncellemenize olanak sağlayan komuttur. Örnek yazım;

db.koleksiyon\_ismi.updateMany(<filter>, <update>)

örnek olarak yaşı 25’ten küçük olan herkesin yaşını 26 olarak ayarlayalım. Örnek;

db.not.updateMany( {yas:{$lt: “25”}}, {$set:{“yas”:”26”}})

örnekten sonra find komutunu tekrar kullanarak yaptığınız tüm değişiklikleri listeleyebilirsiniz.

**Remove() komutu**

MongoDB veritabanında, herhangi bir koleksiyonda bulunan bir dokümanı, koleksiyondan silmek için remove() komutunu kullanmamız gerekir. Remove() komutu 2 farklı parametre alabilir. Bunlar sırasıyla;  
**Deletion Criteria –** Silme işlemini gerçekleştireceğimiz dosyaların uyması gereken kriterler. Kriterlere uyan tüm dokümanlar silinecektir.

**justOne –** Yalnızca tek bir dokümanın silinmesini istiyorsak bu parametreyi true yada 1 olarak ayarlarız.

Komutla ilgili örnek yazım kuralı şu şekildedir;

db.koleksiyon\_ismi.remove(silme\_kriterleri)

örnek olarak not koleksiyonuna bir veri ekleyelim ve eklediğimiz bu veriyi daha sonra silelim.

db.not.insert({“isim”:”Ahmet”})

db.not.find()

db.not.remove({“isim”:”Ahmet”})

db.not.find()

Yalnızca tek bir dokümanı silmek istersek komutu şu şekilde kullanmalıyız.,

db.koleksiyon\_ismi.remove(silme\_kriterleri,1)

tüm dokümanları silmek için kullanabileceğimiz komut ise şudur;

db.koleksiyon\_ismi.remove()

**limit() komutu**

Arama sonucunda listelenen verilerin sayısını sınırlandırmak için kullanacağımız komut limit() komutudur. Parametre olarak sadece sayı alabilir ve parametre olarak verilen sayı adedinde sonucun ekranda listelenmesini sağlar.

db.koleksiyon\_ismi.find().limit(sayi)

örnek olarak not koleksiyonundaki 20 yaşından büyük kişilerin verilerini listeleyelim. Fakat en fazla 2 adet veri ekranda listelensin. Eğer komuta herhangi bir parametre girilmezse tüm sonuçlar ekranda gösterilir.

db.not.find({$gt:{yas:”20”}}).limit(2)

**Skip() komutu**

Limit() komutuna benzer şekilde parametre olarak yalnızca sayı alabilir. Arama sonucunda bulunan sonuçlardan baştan başlayarak, verilen parametre kadar değerin atlanarak listelenmeye başlanmasını sağlayan komuttur. Daha açıklayıcı olmak gerekirse, find() komutunu kullandık ve limit komutuyla beraber en fazla 4 sonucun listelenmesini istedik. Ve arama sonucunda listelenebilir 6 değer bulundu. Eğer skip(1) komutunu kullanırsak ilk baştan 1 adet veriyi yok sayacaktır ve sonraki 4 veri listelenecektir. Eğer hiçbir parametre verilmezse varsayılan değer 0’dır. Yazım şekli;

db.koleksiyon\_ismi.find().limit().skip(sayi)

Örnek olarak yaşı 20’den büyük olan kişilerin verilerinden en üstteki 1 adet veri hariç 2 tanesini listeleyelim.

db.not.find({$gt:{yas:”20”}}).limit(2).skip(1)

**sort() komutu**

MongoDB'de belgeleri sıralamak için sort () komutunu kullanmanız gerekir. Sıralama düzenini belirtmek için 1 ve -1 kullanılır. 1 artan düzen için kullanılırken -1 azalan düzen için kullanılır.

db.koleksiyon\_ismi.find.sort({key:1})

koleksiyon içerisindeki verilerin isim kısmına göre alfabetik sıranın tersine şekilde listelememizi sağlayan komut;

db.not.find().sort({“isim”:-1})

**createIndex() komutu**

dizin oluşturmak, yani her bir dokümana sıra numarası vermek arama işlemlerinin daha hızlı gerçekleştirilmesini sağlar. Dizinler olmadan yapılan aramalarda her bir doküman tek tek aranır ve arama kriterleriyle eşleşme olup olmadığına bakılır. Bu tarama işlemi her seferinde tüm dosyalar üzerinde tarama yaptığından oldukça verimsizdir. Verilere bir sıra numarası vermek, yani bir dizin haline getirmek işimizi kolaylaştırır. 1 veya -1 değerlerini parametre olarak alabilir. Yazım kuralları için örnek;

db.koleksiyon\_ismi.createIndex({key:1})

Bu komutla beraber kullanabileceğiniz diğer ayarların listesi aşağıdaki tabloda verilmiştir;

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| background | Boolean | Bir dizin oluşturmanın diğer veritabanı etkinliklerini engellememesi için dizini arka planda oluşturur. Arka planda oluşturmak için true değerini girin. Varsayılan değer false değeridir. |
| unique | Boolean | Koleksiyonun, dizin anahtarının veya anahtarlarının dizinde varolan bir değerle eşleştiği belgelerin eklenmesini kabul etmemesi için benzersiz bir dizin oluşturur. Benzersiz bir dizin oluşturmak için true değerini belirtin. Varsayılan değer false değeridir. |
| name | string | Dizinin adı. Belirtilmezse, MongoDB, dizinlenmiş alanların adlarını ve sıralama düzenini birleştirerek bir dizin adı oluşturur. |
| sparse | Boolean | Değer true ise, dizin yalnızca belirtilen alana sahip belgelere başvurur. Bu dizinler daha az yer kaplar, ancak bazı durumlarda (özellikle farklı türlerde) farklı davranır. Varsayılan değer false değeridir. |
| expireAfterSeconds | integer | MongoDB'nin bu koleksiyondaki belgeleri ne kadar süreyle tutacağını kontrol etmek için TTL olarak saniye cinsinden bir değer belirtir. |
| weights | document | Ağırlık, 1 ila 99,999 arasında değişen bir sayıdır ve alanın puan olarak diğer endekslenmiş alanlara göre önemini belirtir. |
| default\_language | string | Bir metin dizini için, hangi dilin kullanıldığını belirtmek için kullanılır. Varsayılan değer İngilizcedir. |
| language\_override | string | Metin dizini için, belgedeki alanın adını, varsayılan dili geçersiz kılan dili belirtin. Varsayılan değer dildir. |

**dropIndex() komutu**

MongoDB'nin dropIndex () komutunu kullanarak belirli bir dizini silebilirsiniz. Artan sıra için 1 değeri, azalan sıra için -1 değerini kullanmalısınız. Örnek yazım;

db.koleksiyon\_ismi.dropIndex({key:1})

**DropIndexes() komutu**

Bu komutla belirtilen bir koleksiyondaki birden çok dizini silebilirsiniz.

db.koleksiyon\_ismi.dropIndexes()

**getIndexes() komutu**

Bu komut, koleksiyondaki tüm dizinlerin listesini ekrana yazdırır. Örnek bir yazım şekli aşağıda verilmiştir;

db.koleksiyon\_ismi.getIndexes()

örneğin testValues adında bir koleksiyon oluşturalım ve bu oluşturduğumuz koleksiyona veriler ekleyelim. Daha sonra createIndex() komutunu kullanalım ve getIndex() komutuyla elde ettiğimiz sonucu görelim. Yazmamız gereken komutlar;

db.testValues.insertMany(

[

{

isim: "Radhika",

soyisim: "Sharma",

yas: "26",

e\_mail: "radhika\_sharma.123@gmail.com",

telefon: "9000012345"

},

{

isim: "Rachel",

soyisim: "Christopher",

yas: "27",

e\_mail: "Rachel\_Christopher.123@gmail.com",

telefon: "9000054321"

},

{

isim: "Fathima",

soyisim: "Sheik",

yas: "24",

e\_mail: "Fathima\_Sheik.123@gmail.com",

telefon: "9000054321"

}

]

)

db.testValues.createIndex({"deneme":1,"deneme2":-1})

db.testValues.getIndexes()

**aggregate() komutu**

Toplama işlemleri veri kayıtlarını işler ve hesaplanan sonuçları döndürür. Toplama işlemleri, birden çok dokümandaki değerleri birlikte gruplandırır ve tek bir sonuç döndürmek için gruplandırılmış veriler üzerinde çeşitli işlemler gerçekleştirebilir. SQL sayımı (\*) ve group by ile birlikte MongoDB toplaması eşdeğerdir.

MongoDB'deki toplama için aggregate () yöntemini kullanabiliriz. Örnek yazım;

db.koleksiyon\_ismi.aggregate(operatör)

Örnek yapmak için koleksiyonumuza bu örnek verileri ekleyelim.

{

\_id: ObjectId(7df78ad8902c)

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by\_user: 'tutorials point',

url: 'http://www.tutorialspoint.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 100

},

{

\_id: ObjectId(7df78ad8902d)

title: 'NoSQL Overview',

description: 'No sql database is very fast',

by\_user: 'tutorials point',

url: 'http://www.tutorialspoint.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 10

},

{

\_id: ObjectId(7df78ad8902e)

title: 'Neo4j Overview',

description: 'Neo4j is no sql database',

by\_user: 'Neo4j',

url: 'http://www.neo4j.com',

tags: ['neo4j', 'database', 'NoSQL'],

likes: 750

}

Şimdi, tüm koleksiyonu kapsayacak şekilde, eğer her kullanıcı başına kaç tutorials yazdığını görmek için yazmamız gereken komut;

db.koleksiyon\_ismi.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

aşağıda verilen tabloda aggregete() komutuyla beraber kullanılabileceğiniz operatörler açıklamalarıyla beraber listelenmiştir.

|  |  |  |
| --- | --- | --- |
| **operatör** | **açıklama** | **örnek** |
| $sum | Koleksiyondaki tüm belgelerden tanımlanan değeri toplar. | db.test.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : "$likes"}}}]) |
| $avg | Koleksiyondaki tüm belgelerden verilen tüm değerlerin ortalamasını hesaplar. | db.test.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |
| $min | Koleksiyondaki tüm belgelerdeki karşılık gelen değerlerin minimum değerini alır. | db.test.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}]) |
| $max | Koleksiyondaki tüm belgelerdeki karşılık gelen değerlerin maximum değerini alır. | db.test.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}]) |
| $push | Ortaya çıkan belgedeki bir diziye değer ekler. | db.test.aggregate([{$group : {\_id : "$by\_user", url : {$push: "$url"}}}]) |
| $addToSet | Değeri, sonuçta ortaya çıkan belgedeki bir diziye ekler, ancak kopya oluşturmaz. | db.test.aggregate([{$group : {\_id : "$by\_user", url : {$addToSet : "$url"}}}]) |
| $first | İlk belgeyi gruplama işlemine göre kaynak belgelerden alır. Genellikle sort() komutuyla beraber kullanılması daha mantıklı olan bir seçenektir. | db.test.aggregate([{$group : {\_id : "$by\_user", first\_url : {$first : "$url"}}}]) |
| $last | Gruplamaya göre kaynak belgeden son belgeyi alır. Genellikle sort() komutuyla beraber kullanılması daha mantıklı olan bir seçenektir. | db.test.aggregate([{$group : {\_id : "$by\_user", last\_url : {$last : "$url"}}}]) |

MongoDB, içerdiği komutların sonuna farklı komutlar ekleyerek bir işlemin sonucunun başka bir komuta parametre olarak verilmesi, ve o komuttan çıkan verinin başak bir komuta parametre olarak verilmesi sonucu tek bir sorgu ile birden fazla işlemin yapılması ve nihai sonucun JSON formatında belgelerde saklanmasını sağlar. Diğer komutların çıktılarını parametre olarak alabilen komutlar aşağıda listelenmiştir;

* **$project** − Bir koleksiyondaki belirli alanları seçmek için kullanılır.
* **$match** − Bu bir filtreleme işlemidir. Bir sonraki aşamaya girdi olarak verilen belge miktarını azaltabilir.
* **$group** – özelliğin ayrıntıları yukarıda okuduğunuz gibidir. Gruplandırma işlemini gerçekleştirir.
* **$sort** – Dokümanları listeler.
* **$skip** – Belirtilen miktarda dokümanı işlem sırasında atlamak için kullanılabilir. Daha önce açıklanmıştır.
* **$limit** − Bu, bakılacak belge sayısını geçerli konumlardan başlayarak belirli bir sayı ile sınırlar.
* **$unwind** −. Bu komut dizi kullanan dokümanların çözümlenmesi için kullanılır. Dizi kullanmış olduğunuzda, veriler geçici olarak birleştirilmiştir ve tekrar aynı işlem tekrarlanarak veriler ayrılacaktır.. Bir sonraki komuta gönderilecek belge sayısını arttırır.

**MongoDB’nin PHP ile kullanımı**

MongoDB'yi PHP ile kullanmak için MongoDB PHP sürücüsünün kurulumunu yapmamız gerekli. <https://s3.amazonaws.com/drivers.mongodb.org/php/index.html> Linki tıklayarak kendinize uygun sürümü seçin ve indirmeye başlayın. Şimdi arşivi açın ve php\_mongo.dll dosyasını PHP uzantısı dizininize (varsayılan olarak "ext") koyun ve php.ini dosyanıza aşağıdaki satırı ekleyin

* extension = php\_mongo.dll

Bu işlemi yaptıktan sonra MongoDB PHP bağlantısını yapmaya hazır olmuş oluyoruz. Şimdi PHP Bağlantısını kuralım.

<?php

// MongoDB’ye bağlanıyoruz.

$m = new MongoClient();

echo "Bağlantı Başarılı";

// MongoDB’ye başarılı bir şekilde bağlandık, şimdi Veritabanı bağlantısını kurmamız gerek.

$db = $m->testVT;

echo "testValues veritabanı seçildi.";

?>

**PHP ile koleksiyon oluşturmak**

Aşağıda MongoDB’de PHP ile koleksiyon oluşturmanın nasıl olduğu gösterilmiştir.

<?php

// MongoDB’ye bağlanıyoruz

$m = new MongoClient();

echo "Bağlantı başarılı";

// Veritabanını seçiyoruz

$db = $m->testVT;

echo "testVT veritabanı seçildi";

$collection = $db->createCollection("testKoleksiyon");

echo "Koleksiyon Başarıyla oluşturuldu!";

?>

**PHP ile doküman eklemek**

Daha önce MongoDB ile işlem yaparken, doküman eklemek için insert() komutunun kullanıldığını öğrenmiştik. Bu işlemin PHP ile nasıl gerçekleştirildiği aşağıdaki örnek kodda gösterilmiştir.

<?php

// MongoDB bağlantısı kuruluyor.

$m = new MongoClient();

echo "MongoDB bağlantısı başarıyla tamamlandı.";

// Veritabanı seçiyoruz.

$db = $m->testVT;

echo "testVT veritabanı seçildi";

$collection = $db->testKoleksiyon;

echo "testKoleksiyon koleksiyonu başarıyla seçildi.";

$document = array(

"isim" => "Ömer Orçun",

"soyIsim" => "Tükenmez",

"yas" => 20,

"okul" => "KLU",

"bolum" => "yazilim muhendisligi"

);

$collection->insert($document);

echo “Doküman başarıyla koleksiyona eklendi.";

?>

**PHP ile find() komutu kullanımı**

Veritabanına veri yolladığımız gibi bazı verileri de almak isteriz. MongoDB’den PHP ile veri çekmeyi aşağıdaki kod dizininde gösterdik.

<?php

// MongoDB bağlantısı gerçekleşiyor

$m = new MongoClient();

echo " MongoDB bağlantısı başarıyla tamamlandı.";

// Veritabanı seçiyoruz.

$db = $m->testVT;

echo " testVT veritabanı seçildi.";

$collection = $db->testKoleksiyon;

echo " testKoleksiyon koleksiyonu başarıyla seçildi.";

$cursor = $collection->find();

// belgeler üzerinde gezen imleç

foreach ($cursor as $document) {

echo $document["isim"] . "\n";

}

?>

**PHP ile update() komutu kullanımı**

Dokümanları güncellemek için update() komutunu kullanırız. Aşağıdaki örnekte daha önceden eklenmiş bir verinin PHP ile nasıl güncelleneceğini gösterdik.

<?php

// MongoDB bağlantısı gerçekleşiyor

$m = new MongoClient();

echo " MongoDB bağlantısı başarıyla tamamlandı.";

// Veritabanı seçiyoruz.

$db = $m->testVT;

echo " testVT veritabanı seçildi.";

$collection = $db->testKoleksiyon;

echo " testKoleksiyon koleksiyonu başarıyla seçildi.";

// Dokümandaki verileri güncelliyoruz

$collection->update(array("yas"=>"20"),

array('$set'=>array("yas"=>”21")));

echo "Doküman başarıyla güncellendi.";

?>

Güncellediğimiz verileri görmek istersek kodumuza bu satırları ekleyebiliriz.

<?php

// Dokümanı görüntülüyoruz

$cursor = $collection->find();

// Dokümanlar üzerindeki verilerde gezen imleç

echo "Güncellenen doküman";

foreach ($cursor as $document) {

echo $document["yas"] . "\n";

}?>

**PHP ile koleksiyondan doküman silmek**

PHP ile MongoDB veritabanından bir dokümanı silmek için yazmamız gereken komutlar sırasıyla aşağıda ki örnekte verilmiştir.

<?php

// MongoDB bağlantısı gerçekleşiyor

$m = new MongoClient();

echo " MongoDB bağlantısı başarıyla tamamlandı.";

// Veritabanı seçiyoruz.

$db = $m->testVT;

echo " testVT veritabanı seçildi.";

$collection = $db->testKoleksiyon;

echo " testKoleksiyon koleksiyonu başarıyla seçildi.";

// Dokümanı silme işlemini gerçekleştirdiğimiz yer

$collection->remove(array("isim"=>"Ömer Orçun"),false);

echo "Doküman başarıyla silindi.";

?>