Bölüm 2: Bire Çok İlişkiler
Transkript 3:
("çoğa bir" de aynı manadadır).
En ilginç ilişki türü "çoğa bir" ilişkidir.

One-to-Many (Birden Çoğa) Gösterim	
1. Gömme(embedding) a. tekli tarafta b. çoklu tarafta genellikle daha fazla sorgu yapılan tarafta gömülür.	2. Atıf yapma (referencing) a. tekli taraftan atıf b. çoklu taraftan atıf genellikle çoklu taraftan atıf yapılır.

Tek bir koleksiyon kullanmak ve bilgileri onun içine gömmek yerine, iki ayrı koleksiyonla çalışıp bir koleksiyondaki belgelere diğer koleksiyonun belgelerinden atıf yapılabilir.

Ürün kataloğu kullanım örneğimizi kullanarak, çoğa bir ilişkinin tüm bu temsillerine bir göz atalım.

One-to-Many: embed in the "one" side

- "çok"lu taraftaki belgeler gömülür
- şunlar için genel gösterimdir:
  - basit uygulamalar
  - gömülecek belge azdır
- ana objeyi ve N ilişkili belgeleri birlikte işlemek gerekir.
- indeksleme indis üzerinde yapılır.

1.st) İlk gösterim, son belgeleri bir alt belge dizini olarak gömmektir.
review: inceleme, gözden geçirme
items: mamul, ürün, kalem
reference: atıf
array: dizin
document: evrak/belge
collection: dosya, koleksiyon

"items" kataloğumuzda, bir ürünle ilgili en iyi incelemeleri(reviews) ürünün kendisinde tutuyoruz çünkü bu incelemeleri ürün veritabanından aldıktan sonra görüntülemek istiyoruz. Başka bir deyişle, "items" erişiğimiz ana objelerdir.

Gömülü belge sayısının az olduğu basit uygulamalar için bu en yaygın temsildir.

Gömülü olan çoklu taraftaki bölmelere gelince, dizin (array) alanlarındaki değerleri indekslemek için tasarlanmış çok anahtarlı indeksler kullanıyoruz.

2) Çoğa-bir ilişkilerin ikinci gösterimi, birli taraftaki belgeyi çoklu taraftaki onunla ilişkili belgelerin her birine gömmektir.

One-to-Many: embed in the "many" side		

orders [10M]
_id: <objectId>
date: <date>
customer_id: <int>
items [1, 2, 100]
item_id: <string>
quantity: <int>
price: <decimal>
shipping_address
street: <string>
city: <string>
zip: <string>
	{
	date: "2010/06/24",
	address: {
	"300 University",
	"Palo Alto, CA, USA",
	}
	}
	
	{
	date: "2019/06/24",
	address: {
	"100 Forest",
	"Palo Alto, CA, USA",
	}
	}
		• daha seyrek kullanılır
		• çoklu taraf tekli taraftan daha fazla sıklıkta sorgulanıyorsa faydalıdır.
		• gömülü objeler tekrarlanır
		▪ Mükerrerlik dinamik objeler için tercih edilebilir.

Bir adres ve siparişin bu adrese teslimi iyi bir örnek olacaktır.
Zaman içinde, birçok sipariş aynı adrese teslim edildiğinden, adres ve siparişler arasında çoğa bir ilişki vardır. Uygulamanın, belirli bir adrese gönderilen her şeyi anlamaya çalışmaktan ziyade siparişler üzerinde çalışması daha olasıdır. Erişim kalıbının odak noktası adresler değil siparişlerdir.
Ancak, gömülü bilgilerin doğası statiktir, siparişin sevkiyat adresi gibi -- neredeyse hiç değişmez--.
Sonuç olarak, adresi, ilişkinin çoklu tarafında yer alan herbir siparişte ilişkinin bir tarafı olarak saklamak bizim için daha mantıklıdır.

Bu temsil nispeten az kullanılır.
Bu gösterimin belirgin dezavantajı, gömülü nesnenin birçok yerde çoğaltılması gereğidir.

3.rd) Üçüncü temsil iki koleksiyona sahip olmaktır.

One-to-Many: reference in the "one" side	
zips [10000] _id: <objectId> city: <string> loc [2]: <double> pop: <long> state: <string> zip: <string> stores [0, 1, 5]: <string> **stores [1000]** _id: <objectId> storeId: <string> name: <string> address: <string> city: <string> state: <string> zip: <string> coords [2]: <double>	• Atıf dizinleri • Büyük ve sayıca fazla belgelere müsaittir • Ana obje sorgulandığında atıflar listesine erişilir. • Kademeli silme desteklenmeyebilir; bu uygulama tarafından icra edilmelidir.

```
{
   ….
   Stores:
         [ “store1”, “store408”, “store441”],
   ….
}
```

Bunu yapmak için bir referans dizinine ihtiyacımız var.
Posta kodlarımızı içeren "zip" koleksiyonumuzda, dizindeki her öğenin mağaza koleksiyonundaki belgeleri tanımlayan bir mağaza kimliği (storeId) değeri olduğu bir mağaza (stores) dizini oluşturuyoruz.

Kalan varlık bilgilerini gömemezsek veya gömmemize gerek yoksa, mağaza koleksiyonunda ikinci bir sorgu yapmadan tüm referansları, mağazaları tanırız.

Dizindeki girişler yeterince açıklayıcıysa, bazı sorgulardan sıyrılabiliriz.

Ancak bu gösterimde alan referanslarını ve referans belgelerini senkronize tutmamız gerekiyor. Referans belgeyi silersek, bu belgeye yapılan referansı da silmemiz gerekir.

MongoDB şu anda yabancı anahtarları veya kademeli silmeleri desteklememektedir, bu nedenle belge silme sırasında tüm referansların otomatik olarak kaldırılmasının bir yolu yoktur. Uygulama, tüm bu tür veri yönetimini gerçekleştirmelidir.

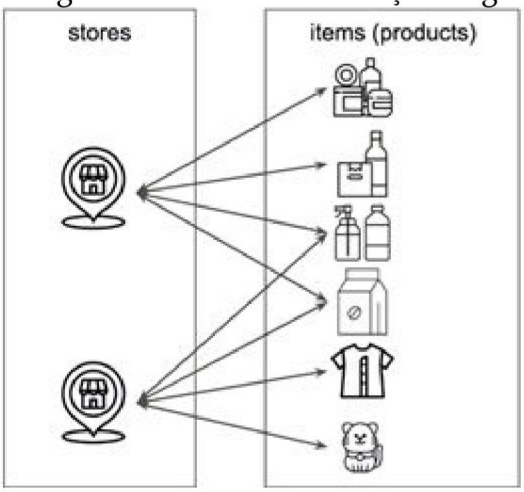
Daha yaygın olarak, referanslar belgelerde bire çok (çoğa bir) ilişkinin çoklu tarafında depolanır. Örneğin, her posta kodunun içine muhtemelen birçok mağazanın eklendiği bir posta kodu koleksiyonumuz olabilir. Mağaza belgelerinin her birine zip adlı tek bir alan ekleyerek, zip koleksiyonundaki belgeye başvurabilirim. Bir mağazayı silersek, referans, kaldırdığımız belgenin içinde olduğu için kaldırılacak ek bir referans yoktur.

"Çoğa bir" ilişkilerin temsilinde bu dersi özetlersek:
• Pek çok seçenek vardır - gömme veya referans verme ve bir ile çok arasındaki tarafı seçin.
• Çok taraflı gömme sırasında çoğaltma meydana gelebilir.
• Ancak, uygun veya hatta tercih edilebilir olabilir.
• Tüm ilgili bilgiler bir arada tutulduğundan, basitlik için veya az sayıda referans belge olduğunda referans yerine yerleştirmeyi tercih edin.
• En çok sorgulanan koleksiyonun yan tarafına gömün.
• En sık sorgulanan belgelerle ilişkili belgelere her zaman ihtiyaç duyulmadığında referans vermeyi tercih edin.

Bölüm 2: Çoğa Çok İlişkiler
Transkript 4:
Store: mağaza
Item: mamul, ürün,kalem

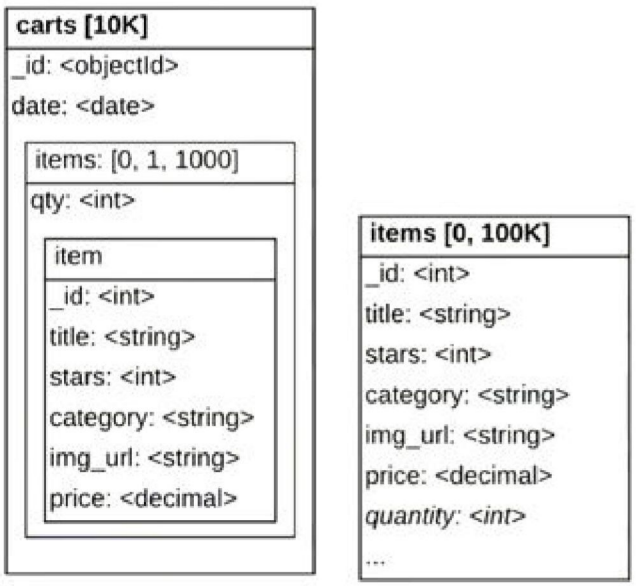
Çoğa çok ilişki, birinci taraftaki belgelerin ikinci taraftaki birçok belgeyle ilişkilendirilmesi ve ikinci taraftaki belgelerin birinci taraftaki birçok belgeyle ilişkilendirilmesi olarak tanımlanır.

Mağazalarımızda (stores) satılan ürünlere(items) baktığımızda, söz konusu mağazanın birçok ürün sattığını ve her bir ürünün birçok mağazada satıldığını görebiliriz.



Genellikle daha az sorgulanan koleksiyonun belgelerini daha çok sorgulanan koleksiyonun belgelerinin içine yerleştiririz.

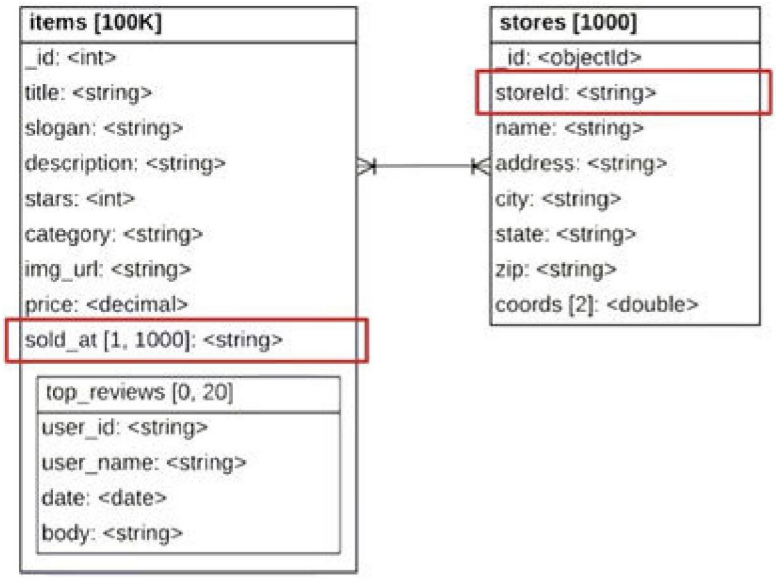
Çoğa Çok: Ana tarafta gömülü



Ana varlık, "item"ları bulmak istediğimiz "cart"tır.
"item"ı "carts"a yerleştiririz çünkü bu bilgileri her zaman birlikte alırız.
"item"belgeleri, "carts" belgelerinden farklı yaşam döngüsüne sahiptir.
Bir girdi kaynağı tutma gerekliliği, yalnızca çoktan çoğa ilişkideki bu özel gösterim için geçerlidir.

Gömmek yerine referansları da kullanabiliriz.

Çoğa Çok: Ana tarafta atıf



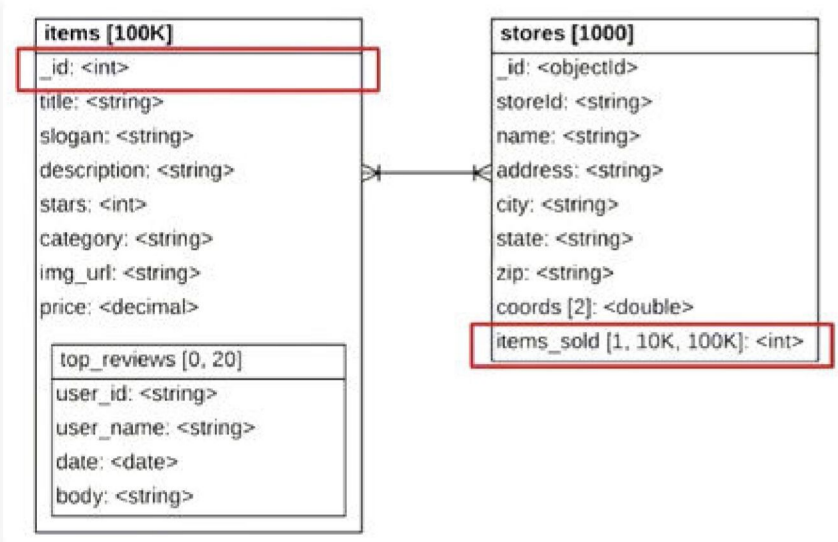
• ana koleksiyonun belgelerine referanslar dizini (array) yerleştirilir.
• halihazırda "ana" koleksiyondaki ilk sorguda referanslara kolayca erişilebilir.

- db.stores.find({_id: {$in: ["storel", "store2|,J ...]}})

Referansları kullanan çoktan çoğa ilişkinin ikinci temsili, başka bir "collection"daki referansı tuttuğumuz yerdir.

Çoğa Çok: İkinci tarafta atıf

- diğer koleksiyonun belgelerine referans dizisi
- daha fazla bilgi almak için ikinci bir sorguya ihtiyacınız var



- Diğer koleksiyonun belgelerine referanslar dizini var
- Daha fazla enformasyon edinmek için ikinci bir sorguya ihtiyaç vardır.
  db.stores.find({items_sold: 10})

Ancak, birkaç farklılık vardır.
Burada, her "store"da, o "store"da satılan "item"lara ilişkin referansların listesini taşıyan "items_sold" adı verilen bir alanı (field) vardır.
"items"ı sorguladığımızda, bir "item"ın hala nerede satıldığını bulamıyoruz. Bu bilgiyi almak için ikinci bir sorguya ihtiyacımız var, ki bu önceki gösterimden başka bir durumdur.
Aşağıdaki gibi bir sorgu, kodu 10 olan item'ı mağazasını döner:

db.stores.find({items_sold: 10})

Bir “item” ekleme ihtimali bir “store” ekleme ihtimalinden çok daha fazladır. Bu halde, “item”ın satıldığı “store” listesini eklemek mi yoksa “item” eklemek için tüm “store”ları tek tek güncellemek mi daha doğrudur? Bu soruyu nasıl evapladığımız, doğru temsili seçmemize yardımcı olacaktır.

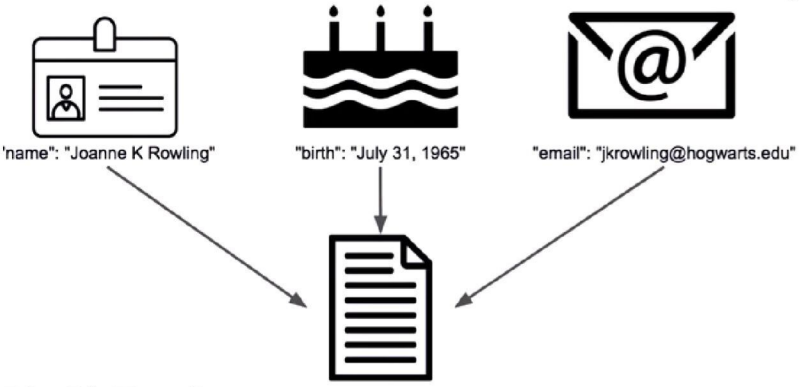
•Gömmeyi (embedding) en çok sorgulanan tarafta tercih edin
•zamana karşı statik olan ve mükerrerlikten (duplication) fayda sağlayabilecek bilgiler için gömmeyi tercih edin
•mükerrerliği halletmek için gömme yerine referans vermeyi tercih edin.

Çoğa Çok İlişkilerin Özeti:
• Basitleştirilmemesi gereken "çoktan çoğa" bir ilişki olduğundan emin olun.
• "Çoktan çoğa" bir ilişki, iki "birden çoğa" ilişkiyle değiştirilebilir, ancak belge modelinde olması zorunlu değildir.
• En çok sorgulanan tarafa yerleştirmeyi tercih edin
• Öncelikli olarak zamana karşı statik olan ve tekrardan fayda sağlayabilecek bilgiler için gömmeyi tercih edin.
• Yinelemeyi yönetmekten kaçınmak için gömme yerine referans vermeyi tercih edin.

Bölüm 2: Bire Bir İlişkiler
Transcript 5:

Bire bir ilişki, ilişkisel bir veritabanında genellikle tek bir tablo ile temsil edilir. Örneğin, bir kişinin adı, doğum tarihi ve e-posta adresi aynı belgede bir arada tutulacaktır. Tüm bu alanların birbirleriyle bire bir ilişkisi vardır. Sistemdeki bir kullanıcının yalnızca bir adı vardır, bir ve yalnızca bir doğum tarihi vardır ve bir ve yalnızca bir e-posta adresiyle ilişkilendirilir(tek eposta adresi olduğunu varsaydığımızda).

Farklı iki varlıktaki enformasyonu birlikte tek bir varlıkta toplamaya embedding(gömme) adını veririz.



Bire Bir Temsil

1. Gömme (embedding):
    a. aynı seviyedeki alanlar (fields)
    b. alt belgelerde gruplandırma
2. Atıf yapma (referencing):
    a. her iki belgede de aynı tanımlayıcı (identifier)
    b. ana tarafta "bir"
    c. ikinci tarafta "bir"

Grup alanları belgede yan yana görünebilir.
Veya mantıksal bilgi grupları oluşturmak için alt belgeler eklemek maksadıyla belge modeli yeteneğini kullanabiliriz. Bu yetenek aynı zamanda belgeyi veya varlığı başka bir belgenin içine gömmemizi sağlar.

Bilgileri gömmeye alternatif olarak, alanları genellikle ayrı koleksiyonlarda birçok belgeye bölebilir ve bir belgeden diğerini referans alabiliriz.
Ürün kataloğu kullanım örneğimizi kullanarak, bire bir ilişkimizin tüm bu temsillerine bir göz atalım.

Bire Bir: gömme, aynı seviyedeki alanlar

- ilişkisel veritabanlarına çok benzer

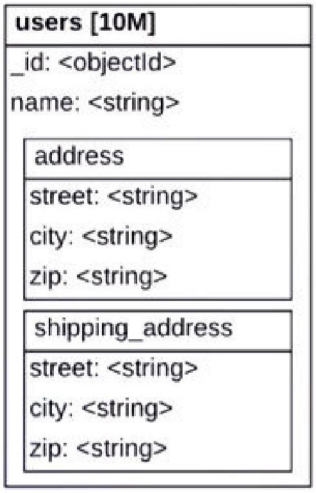
users [10M]
_id: <objectId>
name: <string>
street: <string>
city: <string>
zip: <string>
shipping_street: <string>
shipping_city: <string>
shipping_zip: <string>

Veri parçaları arasında bire bir ilişkiyi temsil etmenin en yaygın yolu, alanları değerleriyle birlikte daha önce açıklandığı gibi belgeye yerleştirmektir.
Kullanıcının koleksiyonunu örnek olarak kullanarak, belirli bir kullanıcıyı, faturalama için kullandığımız adresi ve farklıysa default gönderim adresini takip edebiliriz.

Bir kullanıcının faturalama için yalnızca bir sokak adresi, şehir ve posta kodu ve default teslimat adresi için yalnızca bir sokak adresi, şehir ve posta kodu var.

Bire Bir: alt belgeleri kullanarak gömme

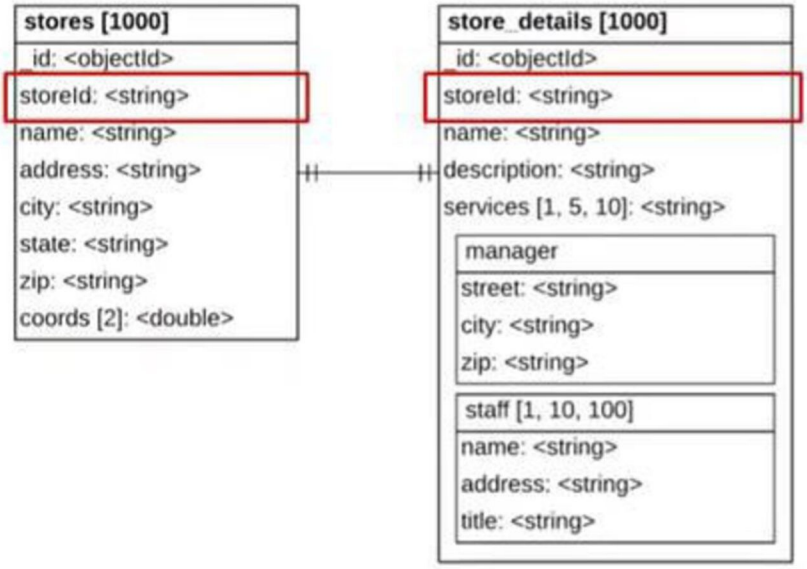
- tercih edilen temsil:
  - o sadeliği korur
  - o belgeler daha net



Belge modelinin gücünü kullanarak, her bir adres bilgisi kümesini bir alt grup halinde yeniden gruplandırabiliriz.
Şimdi hala bire bir ilişkimiz var, ancak bunu bir ve sadece bir fatura adresi olan ve bir ve sadece bir teslimat adresi olan bir kullanıcı olarak tanımlayamayız.
Belge artık daha berrak ve anlaşılması daha kolay.
Bu, bire bir ilişkinin önerilen temsilidir.

Bire Bir: referans (atıf)

Şimdi diyelim ki mağazalarımız gibi ürün kataloğumuzda belirli bir nesne hakkında çok fazla bilgi var.
Bu örnekte, çalışanlarının listesi mağaza belgesinde tutulmaktadır.



Bire Bir: referans (atıf)

- karmaşıklık ekler
- performans iyileştirmeleri şöyle mümkün olabilir:
  - o daha küçük disk erişimi
  - o daha az miktarda RAM gerekli

Mağazaları oluşturduğumuzda çoğu zaman, personel bilgileri umurumuzda olmaz, bu bilgi setini ayırıp ikinci bir koleksiyona yerleştirebiliriz. Sonuç olarak bu, modelimize biraz karmaşıklık katar. Bu yüzden bunu yalnızca şema optimizasyonu nedenleriyle yapmalıyız.

Belirli bir mağazayı sorguladığımızda, bir referansla ifade edilen herhangi bir ilişki için yaptığımız gibi, ilgili belgenin bağlantısını kullanarak mağaza ayrıntıları koleksiyonunu sorgulayarak yönetici ve personel gibi ek bilgilere erişebiliriz.
Bire bir ilişki durumunda, her iki belgede de aynı değeri, burada mağaza kimliğimizi (storId)kullanmak kolaydır.

Ayrıca bu bire bir ilişkinin alt belgeler için bire çok ilişkiye dönüşmesini de önlemek istiyoruz.
Bunu yapabilmek için mağaza kimliği(storeId) alanımızdaki değerlerin her iki koleksiyon için de biricik (unique) olduğundan emin olmamız gerekiyor.

Bu dersi özetlemek gerekirse, bire bir ilişkilerde,

- Basitlik için referans vermek yerine gömmeyi tercih edin.
- Alanları(fields) düzenlemek için alt belgeleri kullanın.
- Optimizasyon amacıyla bir referans kullanın.

Bire Bir İlişkilerin Özeti:

- Basitlik için referans yerine gömmeyi tercih edin
- Alanları düzenlemek için alt belgeleri kullanın
- Optimizasyon amacıyla bir referans kullanın