Şekil 14.9 Satır Merkezli ve Sütun Merkezli Depolamanın Karşılaştırılması

CUSTOMER ilişkisel tablosu

Cus_Code	Cus_LName	Cus_FName	Cus_City	Cus_State
10010	Ramas	Alfred	Nashville	TN
10011	Dunne	Leona	Miami	FL
10012	Smith	Kathy	Boston	MA
10013	Olowski	Paul	Nashville	TN
10014	Orlando	Myron		
10015	O'Brian	Amy	Miami	FL
10016	Kahverengi	James		
10017	Williams	George	Mobil	AL
10018	Farriss	Anne	Opp	AL
10019	Smith	Olette	Nashville	TN

Satır merkezli depolama

Blok 1

10010,Ramas,Alfred,Nashville,TN
10011,Dunne,Leona,Miami,FL

Blok 2

10012,Smith,Kathy,Boston,MA
10013,Olowski,Paul,Nashville,TN

Blok 3

10014,Orlando,Myron,NULL,NULL
10015,O'Brian,Amy,Miami,FL

Blok 4

10016,Brown,James,NULL,NULL
10017,Williams,George,Mobile,AL

Blok 5

10018,Farriss,Anne,OPP,AL
10019,Smith,Olette,Nashville,TN

Sütun merkezli depolama

Blok 1

10010,10011,10012,10013,10014
10015,10016,10017,10018,10019

Blok 2

Ramas,Dunne,Smith,Olowski,Orlando
O'Brian,Brown,Williams,Farriss,Smith

Blok 3

Alfred,Leona,Kathy,Paul,Myron
Amy,James,George,Anne,Olette

Blok 4

Nashville,Miami,Boston,Nashville,NULL
Miami,NULL,Mobile,Opp,Nashville

Blok 5

TN,FL,MA,TN,NULL,
FL,NULL,AL,AL,TN

Sütun odaklı veya sütunlu bir veritabanı, verileri satır sütuna göre bloklar halinde depolar. Tek bir müşterinin verileri birkaç bloğa yayılacaktır, ancak tek bir sütundaki tüm veriler yalnızca birkaç blokta yer alacaktır. Şekil 14.9'da, müşterilere ait tüm şehir verileri, tıpkı tüm eyalet verilerinin birlikte depolanması gibi, birlikte depolanacaktır. Bu durumda, her müşteri için şehir ve eyaleti almak yalnızca iki veri bloğuna erişmeyi gerektirebilir. Bu tür sütun merkezli depolama, birçok raporlama sistemi ve veri ambarında olduğu gibi, öncelikle az sayıda sütun ancak çok sayıda satır üzerinde sorgu çalıştırmak için kullanılan veritabanları için çok iyi çalışır.

Şekil 14.9 yalnızca birkaç satır ve veri bloğunu gösterse de, tablo boyutu yüz binlerce veri bloğunda milyonlarca ya da milyarlarca satıra ulaştığında kazanımların önemli olacağını hayal etmek kolaydır. Aynı zamanda, ekleme, güncelleme ve silme faaliyetleri çok yoğun disk kullanımı gerektireceğinden, sütun merkezli depolama işlemlerin işlenmesi çok verimsiz olacaktır. Sütun merkezli depolamanın ilişkisel veritabanı teknolojisi içinde elde edilebileceğini, yani hala yapılandırılmış veri gerektirdiğini ve sorgular için SQL'i destekleme avantajına sahip olduğunu belirtmek gerekir.

Sütun ailesi veritabanı olarak da adlandırılan sütun *odaklı veritabanı* teriminin diğer kullanımı, sütun merkezli depolama kavramını ilişkisel modelin sınırlarının ötesine taşıyan bir NoSQL veritabanı türünü tanımlamaktır. NoSQL veritabanları olarak bu ürünler, verilerin önceden tanımlanmış yapılara uymasını gerektirmez ve sorgular için SQL'i desteklemez. Bu veritabanı modeli Google'ın BigTable ürünü ile ortaya çıkmıştır. Diğer sütun odaklı veritabanı ürünleri arasında daha önce açıklanan Hbase, Hypertable ve Cassandra bulunmaktadır. Cassandra Facebook'ta bir proje olarak başladı, ancak Facebook bunu açık kaynak topluluğuna yayınladı ve bu topluluk Cassandra'yı en popüler sütun odaklı veritabanlarından biri haline getirmeye devam etti. **Sütun ailesi** veritabanı, verileri anahtar-değer çiftleri halinde düzenleyen ve anahtarları değer bileşenindeki bir dizi sütunla eşleştiren bir NoSQL veritabanıdır. Sütun ailesi veritabanları ilişkisel veritabanlarıyla aynı çoğunu kullansa da, terimler tam aynı anlama gelmez. Neyse ki, sütun ailesi veritabanları olarak basittir ve ilişkisel modeli anlamanızın sütun ailesi modelini anlamanıza yardımcı olabileceği kadar ilişkisel modele yakındır. Sütun, ilişkisel veritabanındaki bir veri hücresine benzeyen bir anahtar-değer çiftidir. Anahtar, sütunun adıdır ve değer bileşeni de bu sütunda depolanan verilerdir. Bu nedenle, "cus_lname: Ramas" bir sütundur; *cus_lname* sütunun adıdır ve *Ramas* sütundaki veri değeridir. Benzer şekilde, "cus_city: Nashville" başka bir sütundur; *cus_city* sütun adı ve *Nashville* veri değeridir.

sütun ailesi veritabanı
Verileri aşağıdaki şekilde düzenleyen NoSQL veritabanı modeli
değer bileşeninin bulunduğu anahtar-değer çiftleri
satıra göre değişen bir dizi sütundan oluşur.

Not

Sütun ailesi veritabanları (henüz) standart SQL'i desteklemese de Cassandra geliştiricileri bir Cassandra sorgu dili (CQL) oluşturmuştur. Bu dil birçok açıdan SQL'e benzer ve Cassandra'yı benimsemek için en ikna edici nedenlerden biridir.

Daha fazla sütun eklendikçe, cus_fname, cus_lname ve cus_initial gibi bazı sütunların doğal gruplar oluşturduğu ve bunların mantıksal olarak bir müşterinin adını oluşturacak şekilde bir araya geldiği anlaşılır. Benzer şekilde, cus_street, cus_city, cus_state ve cus_zip mantıksal olarak bir araya gelerek bir müşterinin adresini oluşturur. Bu gruplamalar süper sütunlar oluşturmak için kullanılır. **Süper sütun**, mantıksal olarak ilişkili bir sütun grubudur. Bölüm 4'te varlık ilişki modelindeki basit ve bileşik nitelikler hakkındaki tartışmayı hatırlayın. Birçok durumda, süper sütunlar bileşik olarak ve süper sütunu oluşturan sütunlar da basit öznitelikler olarak düşünülebilir. Tüm basit özniteliklerin bir bileşik özniteliğe ait olması gerekmediği gibi, tüm sütunların da bir süper sütuna ait olması gerekmez. Bu benzetme birçok bağlamda faydalı olsa da mükemmel değildir. Sütunları, uygulama işleme nedenleriyle mantıksal olarak birbirine ait olan ancak bileşik özniteliğin ilişkisel fikrine uymayan bir süper sütun halinde gruplamak mümkündür.

süper sütun
Bir sütun ailesi veritabanında, bir grup diğer ilgili bir sütun.

Ortamdaki nesneleri tanımlamak için satır anahtarları oluşturulur. Bu nesneleri tanımlayan tüm sütunlar veya süper sütunlar bir sütun **ailesi** oluşturmak için bir araya getirilir; bu nedenle, bir sütun ailesi kavramsal olarak ilişkisel modeldeki bir tabloya benzer. Bir sütun ailesi kavram olarak ilişkisel bir tabloya benzese de, Şekil 14.10 yapısal olarak çok farklı olduğunu göstermektedir. Şekil 14.10'da sütun ailesindeki her bir satır anahtarının farklı sütunlara sahip olabileceğine dikkat edin.

sütun ailesi
Bir sütun ailesi veritabanında, bir satır koleksiyonuyla ilgili bir sütun veya süper sütun koleksiyonu.

Şekil 14.10 Sütun Aile Veritabanı

Sütun Aile Adı	MÜŞTERİLER
Anahtar	Rowkey 1
Sütunlar	Şehir: Nashville Fname: Alfred Lname: Ramas Eyalet: TN
Anahtar	Rowkey 2
Sütunlar	Denge: 345.86 Fname: Kathy Lname: Smith
Anahtar	Rowkey 3
Sütunlar	Şirket: Yerel Pazarlar A.Ş. Lname: Dunne

Not

Bir sütun ailesi sütunlardan veya süper sütunlardan oluşabilir, ancak her ikisini de içeremez.

14-3d Çizge Veritabanları

grafik veritabanı
İlişki açısından zengin ortamlardaki verileri bir düğümler ve kenarlar koleksiyonu olarak depolayan grafik teorisine dayalı bir NoSQL veritabanı modeli.

Çizge veritabanı, ilişki açısından zengin ortamlar hakkında veri depolamak için çizge teorisine dayanan bir NoSQL ver. Çizge teorisi, düğüm adı verilen nesneler arasındaki ilişkileri veya *kenarları* modelleyen bir matematik ve bilgisayar bilimi alanıdır. İlişkiler hakkındaki verilerin modellenmesi ve depolanması çizge veritabanlarının odak noktasıdır. Çizge teorisi, geçmişi yüzlerce yıl öncesine dayanan köklü bir çalışma alanıdır. Sonuç olarak, çizge teorisine dayalı bir veritabanı modeli oluşturmak, çizge veritabanlarının hızla karmaşıklık kazanmasına yardımcı olan algoritmalar ve uygulamalar için hemen zengin bir kaynak sağlar. Son on yıldaki veri patlamasının büyük bir kısmı ilişki açısından zengin veriler içerdiğinden, çizge veritabanları iş ortamında önemli bir ilgi görmeye hazırdır.

Çizge veritabanlarına olan ilgi sosyal ağlar alanında ortaya çıkmıştır. Sosyal ağlar, hemen akla gelen tipik Facebook, Twitter ve Instagram'ın ötesinde geniş bir uygulama yelpazesini içerir. Arkadaşlık siteleri, bilgi yönetimi, lojistik ve yönlendirme, ana veri yönetimi ve kimlik ve erişim yönetimi, nesneler arasındaki karmaşık ilişkilerin izlenmesine büyük ölçüde dayanan alanlardır. Elbette ilişkisel veritabanları da ilişkileri destekler. İlişkisel modelin en büyük ilerlemelerinden biri, ilişkilerin sürdürülmesinin kolay olmasıdır. Bir müşteri ile bir müşteri temsilcisi arasındaki ilişkiyi ilişkisel modelde uygulamak, ortak bir öznitelik oluşturmak için bir yabancı anahtar eklemek kadar kolaydır ve müşteri ve müşteri temsilcisi satırları, ortak özniteliklerde aynı değere sahip olarak ilişkilendirilir. Eğer müşteri

değişirse, yabancı anahtardaki değerin değiştirilmesi verilerin bütünlüğünü korumak için satırlar arasındaki ilişkiyi değiştirecektir. İlişkisel model tüm bunları çok iyi yapar. Ancak, müşterilerin web sitemizdeki aracıları "beğenebilmesi" için bir "beğen" seçeneği istersek ne olur? Bu durumda, bu ikinci ilişkiyi desteklemek üzere yeni bir yabancı anahtar eklemek için veritabanında yapısal bir değişiklik yapılması gerekecektir. Daha sonra, ya şirket web sitesindeki müşterilerin birbirlerini "arkadaş" edinmelerine izin vermek isterse, böylece bir müşteri arkadaşlarının hangi acenteleri ya da arkadaşlarının arkadaşlarını sevdiğini görebilirse? Sosyal ağ verilerinde, bireyler arasında izlenmesi gereken düzinelerce farklı ilişki olabilir ve genellikle ilişkiler birçok katman derinliğinde izlenir (örneğin, arkadaşlar, arkadaşların arkadaşları ve arkadaşların arkadaşlarının arkadaşları). Bu da ilişkilerin verinin kendisi kadar önemli hale geldiği bir durumla sonuçlanır. Bu, grafik veritabanlarının parladığı alandır.

Çizge veritabanlarının temel bileşenleri Şekil 14.11'de gösterildiği gibi düğümler, kenarlar ve özelliklerdir. Bir düğüm, ilişkisel bir varlık örneği fikrine karşılık gelir. **Düğüm**, hakkında veri tutmak istediğimiz bir şeyin belirli bir örneğidir. Şekil 14.11'deki her düğüm (daire) tek bir aracıyı temsil eder. Özellikler, nitelikler gibidir; düğüm hakkında saklamamız gereken verilerdir. Tüm aracı düğümleri ad ve soyad gibi özelliklere sahip olabilir, ancak tüm düğümlerin aynı özelliklere sahip olması gerekmez. **Kenar**, düğümler arasındaki bir ilişkidir. Kenarlar (Şekil 14.11'de oklar olarak gösterilmiştir) tek yönlü olabileceği gibi çift yönlü de olabilir.

Örneğin, Şekil 14.11'de *arkadaş* ilişkileri çift yönlüdür, ancak *beğeni* ilişkileri değildir. Kenarların da **özelliklere** sahip olabileceğini unutmayın. Şekil 14.11'de, müşteri Alfred Ramas'ın temsilci Alex Alby'yi *beğendiği* tarih çizge veritabanına kaydedilmiştir. Çizge veritabanındaki bir sorguya **çaprazlama** denir. *Veritabanını sorgulamak* yerine, doğru terminoloji *çizge üzerinde gezinmek* olacaktır. Çizge veritabanları, en kısa yol ve bağlantılılık derecesi gibi düğümler arasındaki ilişkilere geçişlerde mükemmeldir.

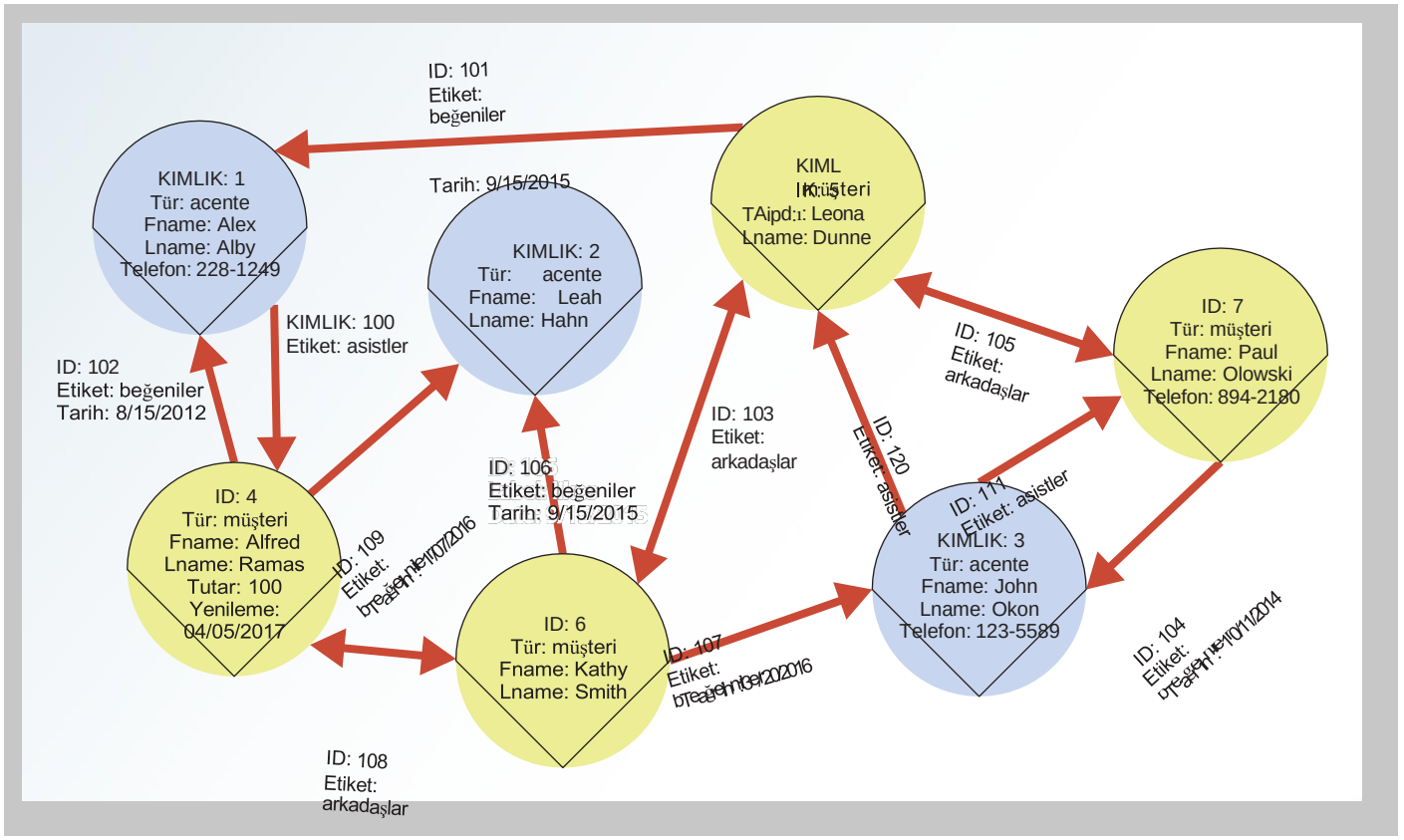
düğüm
Bir çizge veritabanında, tek bir varlık örneğinin temsili.

kenar
Bir çizge veritabanında, bir çizge veritabanının temsili
düğümler arasındaki ilişki.

Özellikler
Bir grafik veritabanında, bir düğümün veya kenarın kullanıcıların ilgisini çeken nitelikleri veya özellikleri.

çapraz geçiş
Grafik veritabanındaki sorgu.

Şekil 14.11 Grafik Veritabanı Gösterimi



Grafik veritabanları, verileri önceden tanımlanmış yapılara uymaya zorlamadığı, SQL'i desteklemediği ve en azından ilişki yoğun veriler için işleme hızı sağlamak üzere optimize edildiği için diğer NoSQL veritabanlarıyla bazı özellikleri paylaşmaktadır. Ancak, diğer temel özellikler çizge veritabanları için geçerli değildir. Grafik veritabanları, toplam farkındalıktaki farklılıklar nedeniyle kümelere çok iyi ölçeklenmez.

14-3 e Toplam Farkındalık

agregat farkındalığı Verilerin kullanılma şekline olarak verileri merkezi bir varlık etrafında düzenleyen bir veri modeli.

Anahtar-değer, belge ve sütun ailesi veritabanları **toplu farkındalıklıdır**. Toplu farkındalık, verilerin merkezi bir konu veya varlık etrafında toplandığı veya bir araya getirildiği anlamına gelir. Örneğin, bir blog web sitesi verileri tek tek blog gönderileri etrafında düzenleyebilir. Her bir blog gönderisiyle ilgili tüm veriler, blog gönderisi (başlık, içerik ve gönderildiği tarih), gönderiyi gönderen kişi (kullanıcı adı ve ekran adı) ve gönderiye yapılan tüm yorumlar yorum içeriği ve yorumcunun kullanıcı adı ve ekran adı) hakkındaki verileri içerebilecek tek bir normalize edilmemiş koleksiyonda toplanır. Normalleştirilmiş, ilişkisel bir veritabanında aynı veriler için USER, BLOGPOST ve COMMENT tabloları gerekebilir. Toplamları oluşturmak için en iyi merkezi varlığın belirlenmesi, çoğu NoSQL veritabanının tasarımındaki en önemli görevlerden biridir ve uygulamanın verileri nasıl kullanacağına göre belirlenir.

Toplu farkındalıklı veritabanı modelleri, her bir veri parçasını nispeten bağımsız hale getirerek kümeleme verimliliği elde eder. Bu sayede bir anahtar-değer çifti, VTYS'nin kümedeki farklı bir düğümde bulunabilecek başka bir anahtar-değer çiftiyle ilişkilendirmesine gerek kalmadan kümedeki bir düğümde depolanabilir. Bir veri işlemine dahil olan düğüm sayısı arttıkça, koordinasyon ve kaynakların merkezi kontrolüne duyulan ihtiyaç da artar. NoSQL veritabanlarının bu kadar etkili bir şekilde ölçeklenmesini sağlayan şey, genellikle *parça* olarak adlandırılan bağımsız veri parçalarını kümedeki düğümler arasında ayırmaktır.

toplu cahil Verilerin öngörülen kullanımına dayalı olarak verileri merkezi bir varlık etrafında bir veri modeli.

İlişkisel veritabanları gibi grafik veritabanları da **toplamdan habersizdir**. Toplu bilgisiz modeller, verileri merkezi bir varlığa dayalı koleksiyonlar halinde organize etmez. Her bir konuyla ilgili veriler ayrı ayrı depolanır ve gerektiğinde tek tek veri parçalarını bir araya getirmek için birleştirmeler kullanılır. Bu nedenle Aggre- gate ignorant veritabanları, uygulamaların veri öğelerini çok çeşitli şekillerde birleştirmesine izin verme konusunda daha esnek olma eğilimindedir. Grafik veritabanları, bağımsız veri parçalarında değil, yüksek oranda ilişkili verilerde uzmanlaşmıştır. Sonuç olarak, grafik veritabanları, ilişkisel veritabanlarına benzer şekilde, merkezi veya hafif kümelenmiş ortamlarda en iyi performansı gösterme eğilimindedir.

14-4 NewSQL Veritabanları

İlişkisel veritabanları kurumsal verilerin temel dayanağıdır ve NoSQL veritabanları iş kolu işlemlerini desteklemek için bunların yerini almaya çalışmaz. İş dünyasının günlük operasyonlarını destekleyen bu işlemler, Bölüm 10'da tartışıldığı gibi ACID uyumlu işlemlere ve birimi kontrolüne dayanırNoSQL veritabanları (ilişki açısından zengin belirli alanlara odaklanan grafik veritabanları hariç) kullanıcı tarafından üretilen ve makine tarafından üretilen verilerin büyük kümeler üzerinde dağıtılmasıyla ilgilenir. NewSQL veritabanları RDBMS ve NoSQL arasındaki boşluğu doldurmaya çalışır. **NewSQL** veritabanları, yüksek oranda dağıtılmış bir altyapı üzerinden ACID uyumlu işlemler sağlamaya çalışır. NewSQL veritabanları, Büyük Veri sorunlarını ele almak için veri yönetimi arenasında ortaya çıkan en son teknolojilerdir. Yeni bir veri yönetimi ürünleri kategorisi olarak NewSQL veritabanları henüz bir başarı geçmişine sahip değildir ve nispeten az sayıda kuruluş tarafından benimsenmiştir.

NewSQL Oldukça dağıtık bir altyapıda ACID uyumlu işlemler sağlamaya çalışan bir veritabanı modeli.

ClusterixDB ve NuoDB gibi NewSQL ürünleri, ilişkisel veritabanları ve NoSQL veritabanlarının özelliklerini bir araya getiren hibrit ürünler olarak sıfırdan tasarlanmıştır.

RDBMS'ler gibi, NewSQL veritabanları da destekler:

- Birincil arayüz olarak SQL
- ACID uyumlu işlemler

NoSQL'e benzer şekilde, NewSQL veritabanları da destekler:

- Yüksek oranda dağıtılmış kümeler
- Anahtar-değer veya sütun yönelimli veri depoları

Beklendiği gibi, hiçbir teknoloji hem RDBMS hem de NoSQL'in avantajlarını mükemmel bir şekilde sağlayamaz, bu nedenle NewSQL'in dezavantajları vardır. (Bölüm 12'de ele alınan CAP teoremi hala geçerlidir!) Temel olarak, dezavantajlar NewSQL'in bellek içi depolamayı yoğun bir şekilde kullanması etrafında toplanmaktadır. Eleştirmenler bunun ACID'nin "dayanıklılık" bileşenini tehlikeye atabileceğine işaret etmektedir. Ayrıca, bellekte tutulabilecek veri miktarının pratik sınırları olduğundan, büyük veri kümelerini işleme yeteneği bellek içi yapılara olan bağımlılıktan etkilenebilir. Teorik olarak NewSQL veritabanlarının önemli ölçüde ölçeklenebilmesi gerekse de, pratikte birkaç düzine veri düğümünün ötesine ölçeklenmek için çok az şey yapılmıştır. Bu, geleneksel RDBMS dağıtımına göre belirgin bir gelişme olsa da, NoSQL veritabanları tarafından kullanılan yüzlerce düğümden çok uzaktır.

Birkaç NoSQL veritabanı ürünü, belirli iş ihtiyaçlarına çözümler sunarak niş pazarlarda başarı elde etmiştir. Aşağıdaki bölümler, yaygın olarak kullanılan iki NoSQL veritabanı olan MongoDB ve Neo4j'ye kısa bir giriş sunmaktadır. Bu iki veritabanı, geleneksel ilişkisel veritabanları tarafından henüz eşleştirilmemiş bir dizi işlevsellik sağlar. Bu veritabanlarının daha ayrıntılı uygulamalı örneklerini sırasıyla Ek P ve Ek Q'da bulabilirsiniz.

14-5 MongoDB Kullanarak Belge Veritabanları ile Çalışma

Bu bölüm size popüler bir belge veritabanı olan MongoDB'yi tanıtmaktadır. Şu anda mevcut olan NoSQL veritabanları arasında MongoDB, veritabanı pazarına girme konusunda en başarılı olanlardan biri olmuştur. Bu nedenle, MongoDB ile çalışmanın temellerini öğrenmek veritabanı uzmanları için oldukça faydalı olabilir.

Not

MongoDB, MongoDB, Inc. şirketinin bir ürünüdür. Bu kitapta, açık kaynak kodlu olan ve MongoDB, Inc. tarafından ücretsiz olarak sunulan Community Server v.5.0.6 sürümünü kullanmaktayız. Yeni sürümler düzenli olarak yayınlanmaktadır. MongoDB'nin bu sürümü Windows, MacOS ve Linux için MongoDB web sitesinden edinilebilir. Kullanıcıların hem Topluluk Sunucusunu hem de Mongo kabuğunu (mongosh) yüklemeleri önerilir. Mongo kabuğu, sunucuya gönderilmek üzere sorguların yazıldığı ön uç arayüzüdür.

MongoDB ismi *humongous* kelimesinden gelmektedir, çünkü geliştiricileri yeni ürünlerini son derece büyük veri setlerini destekleyecek şekilde tasarlamışlardır. Şunlar için tasarlanmıştır:

- Yüksek kullanılabilirlik
- Yüksek ölçeklenebilirlik
- Yüksek performans

Bir belge veritabanı olarak MongoDB, şemasızdır ve toplama farkındadır. Şemasız olmanın, tüm belgelerin aynı yapıya uyması gerekmediği ve belgelerin yapısının önceden bildirilmesi gerekmediği anlamına geldiğini hatırlayın. Toplu farkındalık, belgelerin merkezi bir varlıkla ilgili tüm ilgili verileri aynı belge içinde kapsadığı anlamına gelir. Veriler belgelerde saklanır, benzer türdeki belgeler koleksiyonlarda saklanır ve ilgili koleksiyonlar bir veritabanında saklanır.

Çevrimiçi İçerik

MongoDB'yi kullanarak belge ekleme, güncelleme, silme ve alma dahil olmak üzere genişletilmiş bir dizi uygulamalı alıştırma Ek P'de bulunabilir, Çalışma MongoDB ile, www.cengage.com .adresinde mevcuttur

Kullanıcılar için belgeler JSON dosyaları olarak görünür, bu da onları okumayı ve çeşitli programlama dillerinde manipüle etmeyi . JavaScript Nesne Gösteriminin (JSON), verileri mantıksal bir nesne olarak temsil eden bir veri değişim formatı olduğunu hatırlayın. Nesneler, anahtar-değer çiftleri içeren küme parantezleri {} içine alınır. Tek bir JSON nesnesi virgülle ayrılmış birçok *anahtar:değer* çifti . Bir kitaptaki verileri depolamak için basit bir JSON belgesi aşağıdaki gibi görünebilir:

{_id: 101, başlık: 'Veritabanı Sistemleri'}

Bu belge iki *anahtar:değer* çifti içerir:

- *_id*, ilişkili değeri 101 olan bir anahtardır
- *title*, ilişkili değeri 'Veritabanı Sistemleri' bir anahtardır

Değer bileşeni, belirli bir anahtar için uygun olabilecek birden fazla değere sahip olabilir. Önceki örnekte, yazarlar için bir *anahtar:değer* çifti eklemek 'Coronel' ve 'Morris' değerlerine sahip olabilir. Tek bir anahtar birden fazla değere sahip olduğunda, bir dizi kullanılır. JSON'daki diziler köşeli parantez [] içine yerleştirilir. Örneğin, yukarıdaki belge şu şekilde genişletilebilir:

{_id: 101, başlık: 'Veritabanı Sistemleri', yazar: ['Coronel', 'Morris']}

JSON belgelerinin insanlar tarafından okunması amaçlandığında, okunabilirliği artırmak için genellikle her bir *anahtar:değer* çifti ayrı bir satırda gösterilir, örneğin

```
{
_id: 101,
başlık: 'Veritabanı Sistemleri',
yazar: ['Coronel', 'Morris']
}
```

MongoDB veritabanları belge koleksiyonlarından oluşur. Her MongoDB sunucusu birçok veritabanı barındırabilir. MongoDB sunucusuna bağlanıldığında, ilk görev hangi veritabanı nesnesi ile çalışmak istediğinizi belirtmektir. Sunucuda bulunan veritabanlarının bir listesi şu komutla alınabilir:

dbs'yi göster

MongoDB'deki tüm veri işleme komutları belirli bir veritabanına yönlendirilmelidir. MongoDB'de yeni bir veritabanı oluşturmak *use* komutunu vermek kadar kolaydır.

use fact

use komutu, sunucuya takip eden komutların hedefinin hangi veritabanı olduğunu bildirir. Belirtilen isimde bir veritabanı varsa, sonraki komutlar için bu veritabanı kullanılır. Bu isimde bir veritabanı yoksa, otomatik olarak bir tane oluşturulur.

14-5 a MongoDB'de Belgeleri İçe Aktarma

Çevrimiçi İçerik

Olgu veritabanı için belgeler, doğrudan MongoDB'ye aktarılabilen bir JSON koleksiyonu olarak mevcuttur. Dosya Ch14_Fact.json olarak adlandırılmıştır ve www.cengage.com adresinde mevcuttur

Bir MongoDB veritabanının bir belge koleksiyonu olduğunu unutmayın. Örnek bir MongoDB sorgusunu göstermek için kullanacağımız belge koleksiyonu, Yapılandırılmış Sorgu Diline (SQL) Giriş başlıklı Bölüm 7'de kullanılan Ch07_FACT veritabanından uyarlanan fact veritabanı ve kullanıcı koleksiyonuna dayanmaktadır. Bilgisayar Teknolojisine Ücretsiz Erişim (FACT), Tiny College'da Bilgisayar Bilgi Sistemleri bölümü tarafından işletilen küçük bir kütüphanedir. Modelin burada kullanılan kısmı, merkezi varlık olarak *kullanıcı* ile belgelerden oluşmaktadır. Belgeler aşağıdaki yapıya sahiptir:

```
{_id: <sistem tarafından oluşturulan ObjectID,
display: <kullanıcılara gösterileceği şekliyle kullanıcının tam adı>,
fname: <kullanıcının tüm küçük harflerle yazılmış ilk adı>,
```

lname: <patronun tüm küçük harflerle soyadı>, type:
<either "faculty" or "student">,
age: <kullanıcının yaşı, yalnızca kullanıcı öğrenciyse yıl olarak>,
checkouts: <kullanıcının ödeme geçmişi için bir dizi nesne>

[id: <bu ödeme nesnesi için atanmış bir numara>, yıl: <bu ödemenin gerçekleştiği yıl>, ay: <bu ödemenin gerçekleştiği ay>,
gün: <bu ödemenin gerçekleştiği ayın günü>, kitap: <bu ödeme için kitabın kitap numarası>,
başlık: <kitabın başlığı>,
pubyear: <kitabın yayınlandığı yıl>, subject:
<kitabın konusu>]

}

Kullanıcı belge koleksiyonunun her kullanıcı ve kullanıcının ödünç tüm kitaplar hakkında bilgi içerdiğine dikkat edin. Ayrıca, checkouts alt belgesinin her kullanıcı altında bir nesne dizisi olduğuna dikkat edin. Son olarak, kullanıcının adının iki kez saklandığına dikkat edin; bir kez adı ve soyadı büyük harflerle birlikte, bir kez de adı soyadı tüm küçük harflerle ayrı anahtar:değer çiftleri halinde. Bunun nedeni, MongoDB'deki tüm aramaların varsayılan olarak büyük/küçük harfe duyarlı olması ve fakülte adının iki kez saklanmasının aramaları kolaylaştırmasıdır.

Not

Veritabanı, Ch14_Fact.json dosyası kullanılarak bir işletim sistemi komut isteminde aşağıdaki komut kullanılarak oluşturulabilir (komutun MongoDB kabuğunun içinde değil, işletim sistemindeki bir komut isteminde kullanılmak için olduğunu unutmayın).

mongoimport --db fact --collection patron --type json --file Ch14_Fact.json

Mongoimport MongoDB ile birlikte yüklenen ve verileri bir MongoDB veritabanına aktarmak için kullanılan çalıştırılabilir bir programdır. Önceki komut, içe aktarılan dokümanların "fact" veritabanına (eğer yoksa, oluşturulacaktır) ve "patron" koleksiyonuna (eğer , oluşturulacaktır) yerleştirilmesi gerektiğini belirtir. Mongoimport, CSV dosyaları ve JSON dosyaları gibi farklı dosya türleriyle çalışabilir. Tür parametresi, içe aktarılan belgelerin zaten JSON biçiminde olduğunu belirtir. Dosya parametresi içe aktarılacak dosyanın adını belirtir. Ch14_Fact.json dosyasının kopyası komut isteminizin geçerli dizininde değilse, dosya konumu için uygun bir yol sağlamanız gerekecektir.

14-5 b find() Kullanan MongoDB Sorgusu Örneği

Kullanıcı koleksiyonu içe aktarıldıktan sonra MongoDB veritabanını sorgulamaya hazırsınız demektir. Koleksiyonları manipüle etmek için MongoDB veritabanı metotlar kullanır. **Yöntemler**, nesneleri manipüle etmek için programlanmış işlevlerdir. Bu tür yöntemlere örnek olarak createCollection(), getName(), insert(), update(), find() ve benzerleri verilebilir. **find()** yöntemi, bir koleksiyondan sağlanan kısıtlamalarla eşleşen nesneleri alır. find() yönteminin iki parametresi vardır: find({<query>},{<projection>})

<query> parametresi, koleksiyon nesnelerini almak için kriterleri belirtir. Sorgu <projection> parametresi isteğe bağlıdır ve hangi anahtar:değer çiftlerinin döndürüleceğini belirtir. Projeksiyon nesnesindeki her bir anahtarın değeri ya 0 (döndürme) ya da 1'dir (döndür).

yöntem
Bir nesne içinde, aynı nesne içindeki verileri işlemek için kullanılan programlanmış bir işlev.

find()
Bir koleksiyondan belge almak için bir MongoDB yöntemi.

Şekil 14.12 MongoDB Belge Sorgusu Örneği

```
fact> db.patron.find({$or:[
... {$and:[{lname:"barry"},{type: "faculty"}]},
... {$and:[{lname:"hays"},{age:{$lt:30}}]}
... ]},
... {display: 1, age: 1, type: 1})
[
  {
    _id: ObjectId("6202b97b9a1655a319337307"),
    display: 'Cory Barry',
    type: 'faculty'
  },
  {
    _id: ObjectId("6202b97b9a1655a31933731f"),
    display: 'Jose Hayes',
    type: 'student',
    age: 20
  }
]
fact>
```

Örneğin, Şekil 14.12'de soyadı "barry" olan ve fakülte öğrencisi olan ya da soyadı "hays" olan ve 30 yaşın altında olan kullanıcılar için _id'yi alan, adı ve yaşı görüntüleyen kod gösterilmektedir:

```
db.patron.find({$or: [
    {$ve: [{lname: "barry"}, {type: "faculty"}]},
    {$ve: [{lname: "hays"}, {age: {$lt: 30}}]}
    ]},
{görüntü: 1, yaş: 1, tip: 1})
```

MongoDB, birçok kuruluş tarafından benimsenen güçlü bir belge veritabanıdır. Başlangıçta web tabanlı işlemleri desteklemek için tasarlanmıştır ve bu nedenle, belgelerinin yapısı ve sorgu dili için büyük ölçüde JavaScript'ten yararlanır.

Not

Burada size bir MongoDB koleksiyonunun temel kavramlarını ve find() yöntemini kullanarak nasıl sorgulanacağını tanıttık, ancak belge veritabanlarında kariyer yapmakla ilgileniyorsanız öğrenmeniz gereken çok şey var. Ek P, MongoDB ile Çalışma, bu güçlü belge veritabanının nasıl kullanılacağına dair daha kapsamlı bir öğretici içerir.

14-6 Neo4j Kullanarak Grafik Veritabanları ile Çalışma

Çevrimiçi İçerik

Neo4j kullanan genişletilmiş bir dizi uygulamalı alıştırma şu adreste bulunabilir Ek Q, Neo4j ile Çalışma, www. cengage.com adresinde mevcuttur.

Neo4j henüz MongoDB kadar yaygın olarak benimsenmemiş olsa da, LinkedIn ve Walmart gibi binlerce benimseyeniyle en hızlı büyüyen NoSQL veritabanlarından biri olmuştur. Neo4j bir grafik veritabanıdır. İlişkisel veritabanları gibi, çizge veritabanları da varlıklar ve ilişkilere benzer kavramlarla çalışır. Ancak ilişkisel veritabanlarında odak noktası öncelikle varlıklardır. Çizge veritabanlarında ise odak noktası ilişkilerdir.

Graph veritabanları, varlıklar arasında karmaşık ilişkilerin olduğu ortamlarda kullanılır. Bu nedenle Graph veritabanları, verileri arasındaki karşılıklı bağımlılığa büyük ölçüde bağımlıdır, bu nedenle NoSQL veritabanı türleri arasında en az ölçeklenebilir olanlardır. LinkedIn gibi insanları bağlayan bir sosyal ağ örneğini düşünün. Bir kişi birçok başka kişiyle arkadaş olabilir ve bu kişilerin her biri de birçok kişiyle arkadaş olabilir. İlişkisel bir model açısından, bunu çoktan çoğa tekil ilişkiye sahip bir kişi varlığı olarak temsil edebiliriz. Uygulamada, ilişki için bir köprü oluşturacak ve iki varlıklı bir çözüm elde edeceğiz. Kişi tablosunda 10.000 kişi (satır) olduğunu ve bu kişilerin her birinin ortalama 30 arkadaşı olduğunu, böylece köprü tablosunda 300.000 satır olduğunu düşünün. Bir kişiyi ve arkadaşlarının adlarını almak için yapılacak bir sorgu iki birleştirme gerektirecektir: biri kişiyi köprüdeki arkadaşlarına bağlamak için, diğeri de bu arkadaşların adlarını kişi varlığından almak için. İlişkisel bir veritabanı bu sorguyu hızlı bir şekilde gerçekleştirebilir. Sorun, bu doğrudan arkadaş ilişkisinin ötesine baktığımızda ortaya çıkar. Ya arkadaşların arkadaşları hakkında bilgi edinmek istiyorsak? O zaman köprü tablosunu kendisine bağlayan başka bir birleştirme dahil edilmesi gerekecektir. 300.000 satırlık bir tabloyu kendi içinde birleştirmek önemsiz bir iş değildir (VTYS motorunun birleştirmeyi oluşturmak için uğraştığı Kartezyen çarpımında 90 milyar satır vardır). İlişkisel veritabanı bu hacmi kaldırabilir, ancak yavaşlamaya başlıyor. Şimdi arkadaşların arkadaşlarının arkadaşları için sorgulama yapın. Bu, köprü tablosunun başka bir kopyasının birleştirilmesini gerektirir, böylece sorgu, 2.7×10^{16} satırlı bir Kartezyen çarpımı üretir! Gördüğünüz gibi, "altı derece ayrılık" türünde problemler üzerinde çalıştığımız zaman, ilişkisel veritabanı teknolojisi buna ayak uyduramamaktadır. İlişkisel bir veritabanında çalıştırılması saatler alabilecek ilişkiler hakkındaki bu tür yüksek derecede birbirine bağlı sorgular, grafik veri tabanlarının en güçlü yanıdır. Graf veri tabanları bu sorguları saniyeler içinde tamamlayabilir. Aslında, grafik veritabanlarını benimseyenler kullanımlarını açıklarken sık sık "dakikalar ila milisaniyeler" ifadesiyle karşılaşırsınız.

Not

Neo4j, Neo4j, Inc. şirketinin bir ürünüdür. Neo4j'nin birden fazla sürümü mevcuttur. Bu kitapta, açık kaynak kodlu olan ve Neo4j, Inc. tarafından ücretsiz olarak sunulan Community Server v.3.2.2 sürümünü kullanıyoruz. Yeni sürümler düzenli olarak yayınlanmaktadır. Neo4j'nin bu sürümü Windows (64-bit ve 32-bit), MacOS ve Linux için Neo4j web sitesinden edinilebilir.

Neo4j çeşitli arayüz seçenekleri sunar. Başlangıçta Java programlama düşünülerek tasarlanmış ve bir Java API aracılığıyla etkileşim için optimize edilmiştir. Daha sonraki sürümlerde MongoDB kabuğuna benzer bir Neo4j komut kabuğu, web sitesi etkileşimi için bir REST API ve sezgisel etkileşimli oturumlar için grafiksel, tarayıcı tabanlı bir arayüz seçenekleri eklenmiştir. Bu bölümde, web tarayıcısı arayüzünü kullanacaksınız.

14-6 a Neo4j'de Düğüm Oluşturma

Not

Bir Neo4j örneği aynı anda yalnızca bir etkin veritabanına sahip olabilir. Ancak, veritabanı için veri yolu Neo4j sunucusu başlatılmadan önce yapılandırmada değiştirilebilir. Veri yolu boş bir dizini gösterecek şekilde değiştirilirse, Neo4j başlangıçta bu dizinde gerekli tüm dosyaları otomatik olarak oluşturur. Her veritabanını ayrı bir klasörde tutarak ve sunucuyu başlatmadan önce veri yolunu değiştirerek, uygulama için birden fazla veritabanı .

Bölümün başlarında öğrendiğiniz gibi, çizge veritabanları düğümler ve kenarlardan oluşur. Kabaca konuşmak gerekirse, bir çizge veritabanındaki düğümler, ilişkisel bir veri tabanındaki varlık örneklerine karşılık gelir. Neo4j'de etiket, ilişkisel modeldeki tablo kavramına en yakın şeydir.

Etiket, bir düğüm koleksiyonunu aynı türden veya aynı gruba ait olarak ilişkilendirmek için kullanılan bir etikettir. Varlık örneklerinin, o örneğin özelliklerini tanımlayan öznitelik değerlerine sahip olması gibi, bir düğüm de o düğümün özelliklerini tanımlayan özelliklere sahiptir. İlişkisel modelin aksine, çizge veritabanları şemasızdır, bu nedenle aynı etikete sahip düğümlerin aynı özellik kümesine sahip olması gerekmez. Aslında, mantıksal olarak birden fazla gruba ait olmaları halinde düğümler birden fazla etikete sahip olabilir.

Üyelerin bölgedeki restoranların yorumlarını paylaştığı bir yemek eleştirmenleri kulübü örneği düşünün. Her kulüp üyesi bir düğüm olarak temsil edilecektir. Her bir restoran ise bir düğüm olarak temsil edilecektir. Hem üyeler hem de restoranlar düğüm olmasına rağmen, üyeler bir tür veya tip düğüm iken restoranlar başka bir tür veya tip düğümdür. Düğüm türlerini hem kodda hem de kullanıcıların ve programcıların zihninde ayırt etmeye yardımcı olmak için etiketler kullanabilirsiniz. Üyelere ait düğümler Üye etiketi, restoranlara ait düğümler ise Restoran etiketi alabilir. Bu, kodda düğüm türleri arasında ayrım yapmayı daha kolay hale getirir.

Cypher
Neo4j'de bir grafik veritabanını sorgulamak için kullanılan bildirimsel bir sorgu dili.

Neo4j'deki etkileşimli, bildirimsel sorgu dili **Cypher** olarak adlandırılır. Cypher, sözdizimi çok farklı olsa da SQL gibi bildirimseldir. Bununla birlikte, zorunlu bir dil yerine bildirimsel bir dil olan Cypher'ı öğrenmek çok kolaydır ve temel veritabanı işlemlerini gerçekleştirmek için birkaç basit komut kullanılabilir.

Düğümler ve ilişkiler CREATE komutu kullanılarak oluşturulur. Aşağıdaki kod bir üye düğüm oluşturur:

```
CREATE (:Member {mid: 1, fname: "Phillip", lname: "Stallings"})
```

Not

Neo4j, her düğüm ve ilişki için <id> adında dahili bir kimlik alanı oluşturur; ancak bu alan, depolama algoritmaları için veritabanı içinde dahili kullanım içindir. Benzersiz bir anahtar olarak kullanılması amaçlanmamıştır ve kullanılmamalıdır.

Önceki komut, Member etiketine sahip bir düğüm oluşturur. Bu düğüme 1 değerine sahip *mid*, "Phillip" değerine sahip *fname* ve "Stallings" değerine sahip *lname* özellikleri verilmiştir. *mid* özelliği, üyeleri tanımlamak için bir üye kimliği alanı olarak kullanılmaktadır. Zaten Member adında bir etiket yoksa, düğümle aynı anda oluşturulur.

14-6 b MATCH ve WHERE ile Düğüm Verilerinin Alınması

Tek bir üye düğümü almak için basit bir komut vererek başlayalım. MATCH (m)
RETURN(m)

Bu komut grafik veritabanındaki tüm düğümleri alır. Bu durumda, tek düğüm Phillip Stallings içindir, bu nedenle görüntülenecek tek düğüm budur. Veritabanında çok sayıda düğüm olsaydı, aşağıdaki komut Phillip Stallings'i getirirdi:

Çevrimiçi İçerik
Aşağıdaki bölümde kullanılan Ch14_FCC.txt dosyası www.cengage. com adresinde mevcuttur. Dosyanın içeriği kopyalanıp Neo4j editör çubuğuna yapıştırılmalı ve arayüzdeki oynat düğmesi kullanılarak çalıştırılmalıdır.

```
MATCH (m {fname: "Phillip"}), (x {lname: "Stallings"})
RETURN m
```

Bu durumda, özellikler ve değerler düğümün içine gömülmüştür. Alternatif olarak, WHERE cümlesinin kullanılması, eşitlik dışındaki diğer karşılaştırma operatörlerinin kullanılması gibi daha karmaşık kriterlere olanak tanır. Önceki komut WHERE cümlesi kullanılarak aşağıdaki gibi yeniden yazılabilir:

```
MAÇ (m)
WHERE m.fname 5 "Phillip" AND m.lname 5 "Stallings"
RETURN m
```

Aşağıdaki bölümde Neo4j yemek eleştirmenleri veritabanını, çevrimiçi olarak erişebileceğiniz Ch14_FCC.txt dosyasını kullanarak önceden yüklediğiniz varsayılmaktadır. Bu dosya 78 ek üye, 43 işletme sahibi, 67 restoran ve 8 mutfak oluşturan tek ve büyük bir komut içerir. Tarayıcı arayüzünü kullanıyorsanız kodu tek bir komut olarak vermeniz gereklidir. Etkileşimli kullanım için tasarlandığından, birden fazla komut içeren komut dosyalarını desteklemez. Komut, size yabancı gelebilecek birçok ifade içerir. Bu tür komutlar hakkında daha fazla bilgi edinmek için lütfen Ek Q, Neo4j ile Çalışma bölümüne bakın.

14-6c İlişki Verilerini MATCH ve WHERE ile Alma

Düğümleri almanın ötesinde, düğümler arasındaki ilişkilere dayalı olarak veri almak mümkündür. Daha önce de belirtildiği gibi, ilişkilere odaklanmak çizge veritabanlarının birincil gücüdür. Örneğin, aşağıdaki komut "Tofu for You" restoranını inceleyen ve restorana lezzet konusunda "4" puan veren her üyeyi getirir.

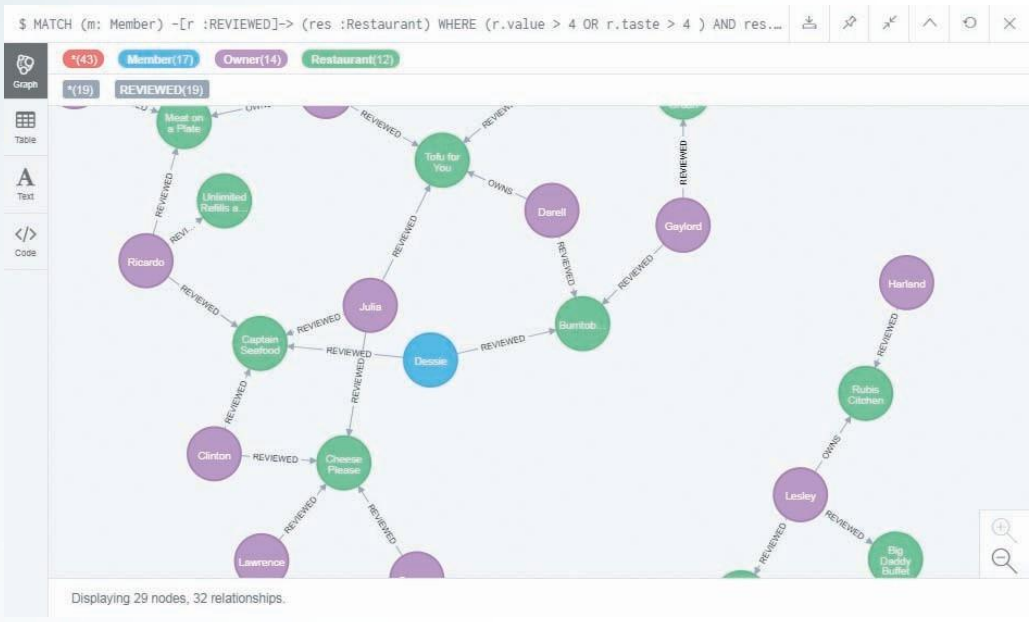
```
MATCH (m :Member) -[r :REVIEWED {taste: 4}]-> (res :Restaurant {name: "Tofu for You"})
RETURN m, r, res
```

Bir ilişkiye dayalı olarak veri alınırken, ilişkinin yönüne ilişkin kriterler ve ilişkinin herhangi bir veri özelliği sorguda belirtilebilir. Bu örnekte, iki düğüm (*m* ve *res*) ve bunları birleştiren bir ilişki (*r*) vardır. Bu durumda, üye olan tüm düğümleri, "Tofu for You" adlı bir düğümü ve REVIEWED olarak etiketlenen ve "4" değerine eşit "taste" adlı bir özelliğe sahip tüm ilişkileri eşleştiriyoruz.

Aşağıdaki komutta gösterildiği gibi WHERE cümlesini kullanarak karşılaştırmalar ve mantıksal operatörler ekleyebilir ve sonuçları Şekil 14.13'te gösterebilirsiniz:

```
MATCH (m :Member) -[r :REVIEWED]-> (res :Restaurant)
WHERE r.value> 4 OR r.taste> 4 ) AND res.state 5 "KY"
RETURN m, r, res
```

Şekil 14.13 MATCH/WHERE/RETURN Kullanarak Neo4j Sorgusu



Komut, Kentucky'deki herhangi bir restoranı inceleyen ve restorana "değer" veya "lezzet" açısından "4 "ten fazla puan veren tüm üyeleri alır. WHERE cümlesinin kullanılmasının, greater than ve mantıksal operatör gibi eşitsizliklerin kullanılmasına izin verdiğine dikkat edin.

Not

Bu bölüm Neo4j'ye çok kısa bir giriş niteliğindedir, ancak çizge veritabanlarında kariyer yapmakla ilgileniyorsanız öğrenmeniz gereken çok şey var. Ek Q, Neo4j ile Çalışma, bu güçlü çizge veritabanının nasıl kullanılacağına dair daha kapsamlı bir öğretici içerir.

Bölüm 13'te, karar destek verilerini modellemek ve depolamak için veri ambarları ve yıldız şemaları hakkında bilgi edindiniz. Bu bölümde, kuruluşların yapılandırılmamış formatlarda topladıkları geniş veri depolarını ve bu verileri kullanıcıların kullanımına sunan teknolojileri keşfederek buna eklemeler yaptınız. Bölüm 13'te ele alınan veri analitiği, tüm bu veri kaynaklarından -NoSQL veritabanları, Hadoop veri depoları ve veri ambarları- bilgi çıkarmak ve tüm kurumsal kullanıcılara karar desteği sağlamak için kullanılır. İlişkisel veritabanları çoğu ticari işlem için hala baskın olsa ve öngörülebilir gelecekte de öyle olmaya devam edecek olsa da, Büyük Veri'nin büyümesine uyum sağlanmalıdır. Kuruluşlar için mevcut olan muazzam miktardaki yapılandırılmamış verinin görmezden kadar büyük bir değeri vardır. Veritabanı uzmanları, her iş için doğru aracın kullanıldığından emin olmak için veri yönetimine yönelik bu yeni yaklaşımlar hakkında bilgi sahibi olmalıdır.

Özet

- Büyük Veri, ilişkisel modelin uyum sağlamakta zorlanacağı kadar hacimli, hızlı ve/veya çeşitli verilerle karakterize edilir. Hacim, depolanması gereken veri miktarını ifade eder. Hız, hem verilerin depolamaya girme hem de işlenmeleri gereken hızı ifade eder. Çeşitlilik, depolanan verilerin yapısındaki tek biçimlilik eksikliğini ifade eder. Büyük Veri'nin bir sonucu olarak, kuruluşlar ilişkisel veritabanlarına ek olarak teknolojileri de içeren çeşitli veri depolama çözümleri kullanmak zorunda kalmaktadır; bu durum çoklu kalıcılık olarak adlandırılmaktadır.
- Hacim, hız ve çeşitlilik topluca Büyük Verinin 3 Vs'si olarak adlandırılır. Ancak, veri yöneticilerinin hassas olması gereken Büyük Verinin tek özelliği bunlar değildir. Veri yönetimi endüstrisi tarafından önerilen ek özellikler arasında değişkenlik, doğruluk, değer ve görselleştirme yer almaktadır. Değişkenlik, verilerin anlamında zaman içinde meydana gelebilecek değişikliklerdir. Doğruluk, verilerin güvenilirliğidir. Değer, verilerin faydalı olup olmadığı ile ilgilidir. Son olarak, görselleştirme, verilerin aşağıdaki gibi olması gerekliliğidir karar vericiler için anlaşılabilir hale getirecek şekilde sunulabilmelidir. Bu ek unsurların çoğu Büyük Veri'ye özgü değildir. İlişkisel veritabanlarındaki veriler için de endişeler vardır.
- Hadoop çerçevesi, Büyük Verinin fiziksel depolanması için hızla bir standart olarak ortaya çıkmıştır. Çerçevenin temel bileşenleri arasında Hadoop Dağıtılmış Dosya Sistemi (HDFS) ve MapReduce bulunmaktadır. HDFS, verileri çok büyük bir emtia sunucu kümesi üzerinde güvenilir bir şekilde dağıtmaya yönelik ortak bir teknolojidir. MapReduce, veri işlemeyi dağıtılmış veriler arasında dağıtmaya yönelik tamamlayıcı bir süreçtir. MapReduce için anahtar kavramlardan biri, verileri hesaplamalara taşımak yerine hesaplamaları verilere taşımaktır. MapReduce, alt görevleri işlenecek verileri tutan küme sunucularına dağıtan *map* ve map sonuçlarını tek bir sonuç kümesinde birleştiren *reduce* işlevlerini birleştirerek çalışır. Hadoop çerçevesi ayrıca karmaşık bir Büyük Veri işleme sistemi üretmek için birlikte çalışan Hive, Pig ve Flume ek araç ve teknolojilerden oluşan bir ekosistemi de desteklemektedir.

- NoSQL, veri yönetimine yönelik çeşitli doğrusal olmayan veritabanı yaklaşımlarından herhangi birini ifade etmek için kullanılan geniş bir terimdir. Çoğu NoSQL veritabanı dört kategoriden birine girer: anahtar değer veritabanları, belge veritabanları, sütun yönelimli veritabanları veya grafik veritabanları. NoSQL şemsiyesi altındaki ürünlerin geniş çeşitliliği nedeniyle, bu kategoriler her şeyi kapsayıcı olmayabilir ve birçok ürün birden fazla kategoriye girebilir.
- Anahtar-değer veritabanları verileri anahtar-değer çiftleri halinde depolar. Bir anahtar-değer çiftinde, anahtarın değeri DBMS tarafından bilinmelidir, ancak değer bileşenindeki veriler herhangi bir türde olabilir ve DBMS içindeki verilerin anlamını anlamak için hiçbir girişimde bulunmaz. Bu tür veritabanları, veriler tamamen bağımsız olduğunda çok hızlıdır ve uygulama programlarının verilerin anlamını anlamasına güvenilebilir.
- Belge veritabanları da verileri anahtar-değer çiftleri halinde depolar, ancak değer bileşenindeki veriler kodlanmış bir belgedir. Belge, XML veya JSON gibi etiketler kullanılarak kodlanmalıdır. DBMS, belgelerdeki etiketlerin farkındadır ve bu da etiketler üzerinde sorgulama yapılmasını mümkün kılar. Belge veritabanları, belgelerin kendi kendine yeten ve nispeten birbirinden bağımsız olmasını bekler.
- Sütun ailesi veritabanları olarak da adlandırılan sütun yönelimli veritabanları, verileri, değer bileşeninin kendileri de anahtar-değer çiftleri olan bir dizi sütundan oluştuğu anahtar-değer çiftleri halinde düzenler. Sütunlar, ilişkisel modeldeki bir bileşik özniteliğin basit özniteliklerden oluşmasına benzer şekilde süper sütunlar halinde gruplandırılabilir. Benzer türdeki tüm nesneler şu şekilde tanımlanır satırlar, bir satır anahtarı verilir ve bir sütun ailesi içine yerleştirilir. Bir sütun ailesi içindeki satırların aynı yapıya sahip olması gerekmez, yani aynı sütunlara sahip olmaları gerekmez.
- Çizge veritabanları çizge teorisine dayanır ve verileri düğümler, kenarlar ve özellikler aracılığıyla temsil eder. Bir düğüm, ilişkisel modeldeki bir varlığın örneğine benzer. Kenarlar, düğümler arasındaki ilişkilerdir. Hem düğümler hem de kenarlar, ilgili düğümü veya kenarı tanımlayan nitelikler özelliklere sahip olabilir. Grafik veri tabanları, sosyal medya verileri gibi birbiriyle oldukça ilişkili verilerin izlenmesinde mükemmeldir. Düğümler arasındaki çok sayıda ilişki nedeniyle, bir çizge veri tabanını bir kümeye yüksek oranda dağıtılmış bir şekilde dağıtmak zordur.
- NewSQL veritabanları hem RDBMS (ACID uyumlu işlemler sağlayan) hem de NoSQL veritabanlarının (oldukça dağıtık bir altyapı kullanan) özelliklerini entegre etmeye çalışır.
- MongoDB, belgeleri JSON formatında saklayan bir belge veritabanıdır. Belgeler, MongoDB Sorgu Dili adı verilen JavaScript benzeri bir dil kullanılarak oluşturulabilir, güncellenebilir, silinebilir ve sorgulanabilir. Veri alımı öncelikle find() yöntemi aracılığıyla yapılır.
- Neo4j, verileri düğümler ve ilişkiler olarak depolayan ve her ikisi de bunları tanımlamak için özellikler içerebilen bir grafik veritabanıdır. Neo4j veritabanları, SQL ile birçok benzerliği paylaşan ancak yine de birçok yönden önemli ölçüde farklı olan bildirimsel bir dil olan Cypher kullanılarak sorgulanır. Veri alımı, öncelikle desen eşleştirme gerçekleştirmek için MATCH komutu aracılığıyla yapılır.

Anahtar Terimler

aggregate aware aggregate
ignorant algoritması
toplu¨işleme blok
raporu
BSON (İkili JSON) kovası
koleksiyon sütun
ailesi
sütun ailesi veritabanı
sütun merkezli depolama
Cypher
belge veritabanı
kenarı
geri bildirim döngüsü
işleme find()
grafik veritabanı
Hadoop Dağıtılmış Dosya Sistemi (HDFS)
kalp atışı iş
takipçisi
JSON (JavaScript Object Notation)
anahtar-değer (KV) veritabanı
harita
eşleyici
MapReduce
yöntemi
NewSQL
düğüm
NoSQL
polyglot kalıcılık özellikleri
redüktörü
azaltın
satır merkezli depolama
ölçeklendirme
duygu analizini
ölçeklendirme
akış işleme
yapılandırılmış
veri süper sütun
görev izleyici
çaprazlama
yapılandırılmamış
veri değeri
değişkenlik
çeşitlilik hız
doğruluk
görselleştirm
e hacim

İnceleme Soruları

1. Büyük Veri nedir? Kısa bir tanım veriniz.
2. Büyük Verinin geleneksel 3 Vs'si nelerdir? Her birini kısaca tanımlayın.
3. Google ve Amazon gibi şirketlerin neden Büyük Veri sorununu ele alan ilk şirketler arasında yer aldığını açıklayın.
4. *Ölçek büyütme* ile *ölçek büyütme* arasındaki farkı açıklayın.
   Ölçeklendirme.
5. Akış işleme nedir ve neden bazen gereklidir?
6. Akış işlemenin geri bildirim döngüsü işlemeden farkı nedir?
7. Doğruluk, değer ve görselleştirmenin neden ilişkisel veritabanlarının yanı sıra Büyük Veri için de geçerli olduğunu açıklayınız.
8. Çoklu kalıcılık nedir ve neden yeni bir yaklaşım olarak kabul edilir?
9. Hadoop Dağıtılmış Dosya Sistemi yaklaşımı tarafından yapılan temel varsayımlar nelerdir?
10. HDFS'de bir isim düğümü ile bir veri düğümü arasındaki fark nedir?
11. MapReduce işlemedeki temel adımları açıklar.
12. HDFS ve MapReduce'un birbirini nasıl kısaca açıklayınız.
13. NoSQL veritabanlarının dört temel kategorisi nelerdir?
14. Bir anahtar-değer veritabanı ile bir belge veritabanının değer bileşenleri nasıl farklıdır?
15. Satır merkezli ve sütun merkezli veri depolama arasındaki farkı kısaca açıklayınız.
16. Sütun ailesi veritabanında bir sütun ile bir süper sütun arasındaki fark nedir?
17. Grafik veritabanlarının neden ölçek büyütme ile mücadele etme eğiliminde olduğunu açıklayın.
18. Bir veritabanının toplu farkındalığa sahip olmasının ne anlama geldiğini açıklayın.