3. BÖLÜM / CHAPTER 3

VERİ SAKLAMA YÖNTEM VE UYGULAMALARI

DATA STORAGE METHODS AND APPLICATIONS

Emre AKADAL*

*Dr., İstanbul Üniversitesi Enformatik Bölümü, İstanbul, Türkiye

E-mail: emre.akadal@istanbul.edu.tr

DOI: 10.26650/B/ET07.2021.003.03

ÖZ

Bilgiye ulaşmak veriden, veriye ulaşmak ise onu kaydedebilmek ve işleyebilmekten geçmektedir. Veri, kaydedilebilse bile istenilen enformasyonu sunamayacak biçimde olması durumunda niteliğini kaybedecektir. Farklı büyüklükteki organizasyonlar kendi ürettikleri veriyi saklama eğiliminde olurlar. Ancak kaç tanesi bu veriden anlam çıkararak süreçlerini iyileştirebilir? Veriyi kaydetme ve işleme arasındaki sürecin zorluğu biraz da verinin yönetilmesi konusunda yapılan planlarla ilgilidir. Bu bölümde, öncelikle veriyi kaydetmek üzerine tarihsel gelişim ele alınacaktır. Yapılandırılmış veriyi kaydetmenin en yaygın yöntemi olan ilişkisel veritabanları ele alınacak, bu veritabanı türünün avantaj ve öne çıkan özellikleri ayrıntılı olarak ele alınacaktır. Ayrıca veritabanı tasarlamının önemi de vurgulanarak yeni bir veritabanı tasarımı yapma konusunda çeşitli bilgiler sunulacaktır. Sonrasında da karşılaşılabilecek özel durumlar için kullanılan yaklaşımlardan bahsedilecektir. Çok büyük miktardaki veriden çok küçük miktarda olana, yüksek işlem gücü gerektirenlerden dağıtık veritabanlarına çok çeşitli yöntemler sunulacaktır. Büyük ölçekli ve genellikle düzensiz veri yapılarına sahip projeler için ölçeklendirme konusundaki avantajları da göz önünde bulundurularak NoSQL konsepti ele alınacaktır. Bu konsept, doküman tabanlı veri saklamayı mümkün kılması ve JSON gibi platformlar arası veri iletişimi için etkin bir biçim kullanıyor olması sebebiyle öne çıkmaktadır. Nesnelerin interneti alanında sıkça karşılaşılan, kısıtlı donanıma sahip becerikli cihazlar içerisinde kullanılmak üzere hafif (lightweight) veritabanlarından da bahsedilecek ve örnekendirilecektir. Bitcoin gibi devrimsel bir gelişmenin ardında yatan teknoloji olan ve kriptoloji temelleri sayesinde güvenli yanılla öne çıkan blokzincir teknolojisi de ayrıntılı olarak açıklanacaktır. Tüm bunların yanında yüksek performans sağlayabilmek için güçlendirilmiş donanım desteğinin nasıl kullanıldığından da bahsedilerek bölüm sonuçlandırılacaktır. Bölüm sayesinde veritabanı kurmadan önce veriyi tanımanın önemi ve yöntem seçimi için verinin hangi özelliklerinin göz önünde bulundurulması gerektiği vurgulanacaktır.

Anahtar Kelimeler: Veri, veritabanı, ilişkisel veritabanı, nosql veritabanları, hafif veritabanları, gpu, blokzincir

ABSTRACT

Although data are required to achieve information, we must have the ability to store and process data to gain some of them. Data could lose their qualification when they are not presented in a suitable format. Organizations tend to store data generated in their inner processes in various sizes. But how many of them can really improve their processes using the data? Also, data management plays a key role in data storage and processing. This chapter compound a few subsections. First, we inspect the historical improvement on data storage. Then, we will discuss relational databases, the most common method of storing structured data, and the advantages and highlights of this database type in detail. Additionally, we will present various information about realizing a new database design, further emphasizing the importance of database design. Afterward, we will handle the approaches for specific situations of some practical problems. We will introduce various methods in terms of size, required computation power, etc. Importantly, we will also discuss the NoSQL concept, which is used in projects with large-scale and often irregular data structures while also considering its advantages in scaling. This concept stands out as it enables document-based data storage and uses an efficient format for cross-platform data communication such as JSON. Lightweight databases will also be mentioned and exemplified to be used in resourceful devices with limited hardware, which are frequently encountered in the field of Internet of Things. Blockchain technology, which is the technology behind a revolutionary development like Bitcoin and stands out with its secure side thanks to its cryptology fundamentals, will also be explained in detail. In addition to all these concepts, we will conclude the section by mentioning how to use reinforced hardware support to provide high performance. Finally, we will emphasize the importance of identifying data and determining the features of the data to choose the correct method.

Keywords: Data, database, relational database, NoSQL database, lightweight database, GPU, blockchain

1. Giriş

Enformasyon ve bilgi bugünün çeşitli ölçekteki organizasyonları için hayati öneme sahiptir. Çoğu zaman organizasyon için önemli birer faydaya dönüştürülebilen bu olgular, ancak toplanan verinin işlenmesiyle elde edilebilmektedir. Rowley (2007)'in bilgelik hiyerarşisinde görüldüğü üzere veri; enformasyon ve bilgi üretme sürecinde temel yapı taşı görevi görmektedir. Ayrıca miktarı da büyük önem taşır.

Veriden anlam çıkarabilmek için doğru kaynaktan, doğru miktarda edinmek ve doğru yöntemlerle işlemek gerekmektedir. Ancak bunların yanında verinin kaydedilebilmesi; özellikle ihtiyaç anında ihtiyaç duyulan miktarı hızlıca elde edebilmek de çok önemlidir. Veriyi kaydetmek, dikkate alınacak tek özellik verinin hacmi olduğu sürece kolay bir işlem sayılabilir. Bir telefon numarasını kaydetmek için en kolay yol nedir? Geleneksel yöntem ile bir kağıda yazmak yeterli olabilir. Dijital bir yöntem olarak da benzer şekilde bir not alma uygulaması iş görecektir. Bir telefon numarası kaydetmek için veritabanı yapısı kurmaya ihtiyaç duyulmayacaktır. Ancak binlerce telefon numarasından bahsediyorsak biraz daha düzenli bir yapıya ihtiyaç duyulur. Bu noktada “düzenli” kelimesini irdelemek gerekir. Elbette ki “güzel bir el yazısı ile not almak” kastedilmemektedir. Burada “düzenli”, bir ihtiyaç halinde aradığımız bilgiye hızlıca ulaşabilmek anlamındadır. Binlerce kişiye ait telefon numarasını kaydettikten

sonra, ihtiyaç anında ilgili kişinin telefon numarasını bulamadıktan sonra eldeki verinin değeri hakkında ne söylenebilir? Bir kişinin birden fazla telefon numarasının olabileceğini düşünelim. Kayıtlarımıza baktığımızda, aradığımız kişinin tüm numaralarını aynı anda görmek istiyorsak verileri kaydetme biçimimizi bu duruma göre özelleştirmemiz de gerekecektir. Veriyi ilgili kaynaktan almak ve istiflemek dışında bunu ne şekilde gerçekleştirdiğimiz de enformasyon ve bilgi üretme sürecinde hayati öneme sahiptir.

Atomik anlamda veriyi kaydetmek temel bir mantığa dayanmaktadır. Bunu örneklendirmek için yine geleneksel açıdan ele alalım ve bu sefer not almaktan daha ilkel bir örnek seçelim. Camların ıslak olması bizim için bir veridir ve bu veri, bize yağmur yağdığı ya da yağmıyor olduğuyla ilgili enformasyon sağlayabilir. Buradaki veri, ikili (*binary*) tiptedir. Kodlama yapılmak istenildiğinde 0 ve 1 kullanılabilir. 1, ıslaklığı; 0 ise ıslak olmama durumu için kullanılabilir. Böylece pencere camları, ıslaklık verisini ikili veri tipinde saklamış olmaktadır. Dijital ortamda da veri saklama işlemi benzer şekilde gerçekleşmektedir. Elektronik veri saklama işleminde elektrik olup olmaması 1 ve 0 şeklinde kodlanırken, fiziksel veri saklamada bazı kalıcı işaretlemeler yine 1 ve 0 anlamına gelmektedir. Yalnızca tek bir 1 ya da 0 durumunu kaydetme işlemi, elektronik ortamda 1 bitlik kapasite olarak adlandırılmaktadır. Bir anlam ifade eden yanyana 8-bit ile 1 bayt (*byte*)'lık kapasite elde edilmiş olur. Kilobayt ise 1024 bayta karşılık gelmektedir. Sonraki her bir kapasite ölçüsü de sırasıyla bir öncekinin 1024 katıdır: kilobayt, megabayt, gigabayt, terabayt, petabayt, eksabayt, zettabayt, yottabayt.

Veri kaydetme kapasitesindeki artış, enformasyon artışını garanti etmeyebilir. Daha önce verilen örnekteki gibi, binlerce telefon numarasının yapılandırılmadan, yalnızca not alınarak kaydedilmesi; istenilen bilgiye erişimi kısıtlayacaktır. Bu mikro örnek, makro düzeyde de benzer olaylara sebep olabilmektedir. Yüzlerce nakliye aracı olan bir firma, tüm araçlarının her bir andaki hareketini kaydediyor olsun. Kayıtlar, bir süre geçtikten sonra oldukça büyüyecek; bunun ötesinde veri üretimi devam ettiği için büyüme durmayacaktır. Eğer örnekteki firma bu veriyi gerçek zamanlı işleyerek enformasyon elde edemezse bilgi de oluşmayacak, dolayısıyla veriye dayalı karar alınabilmesi de mümkün olmayacaktır. Bu durum da ayrıntılı bir veriye sahip ancak bu veriden anlam çıkaramayan bir organizasyon yönetimine yol açacaktır. Örnekler, daha fazla verinin idaresi için daha fazla saklama alanının tek başına yeterli olamayacağını göstermektedir. Büyük veri kavramı da bu noktada karşımıza çıkmaktadır.

Dünya çapında verinin yüksek hızlı artışı altında çok büyük veri kümeleri “büyük veri” terimiyle açıklanmaktadır. Alışlagelmiş veri kümeleriyle karşılaştırıldığında büyük veri, genellikle gerçek zamanlı analizde sıklıkla ihtiyaç duyulan çok büyük miktarda yapılandırılmamış veri içermektedir (Chen, Mao, & Liu, 2014). Sosyal ağlardaki kullanıcı tarafından oluşturulan

ya da sensörlerden toplanan büyük miktarda veriler büyük veriye iki basit örnektir (Gessert, Wingerath, Friedrich, & Ritter, 2017).

Veri toplamak için gerekli donanımsal altyapıyı sağladıktan sonra, hedef bu veriden anlam çıkarmak olduğunda kurgulanacak veri yönetim sistemi hayati öneme sahip olacaktır. Herhangi bir kişisel bilgisayarda rahatlıkla işlenebilecek boyuttaki veriler için hazırlanacak bir veri yapısında oluşacak hata, belki yalnızca birkaç saniyelik gecikmeye sebep olabilir. Ancak zettabayt seviyesine ulaşan veri miktarlarında en ufak bir hata, veriyi işlemeyi imkansız hale getirebilir. Günümüz veri kümelerinin büyüklükleri bile veri yöneticilerini zorlayacak seviyeye ulaştığı bir gerçektir. Diğer bir yandan işletme analitiği için endüstride toplanan ve analiz edilen veri kümelerinin boyutları da hızla artmaktadır (Thusoo et al., 2010). Tüm bunlar, bir sistem için veri yapısı kurgulamada alınacak kararların önemini göstermektedir.

Gelecek başlıklar altındaki bölümler, çeşitli veritabanlarının genel özelliklerini sunmakta; veritabanı yapısı kurma tercihinde hareket edilmesi gereken yön hakkında bilgi vermektedir.

2. İlişkisel Veritabanları

Verinin yapılandırılmış şekilde kaydedilmesi süreci, veritabanlarının ortaya çıkması ile başlamış; özel durumlar için üretilen çözümler ile bugünkü halini almıştır. Verilere kolaylıkla erişilmesini ve organize edilmesini sağlayan yapılar olan veritabanlarının gelişimi; Henry (1969) tarafından hiyerarşik, Bachman (1969) tarafından ağ, Codd (1970) tarafından ilişkisel veritabanlarının önerilmesiyle büyük ivme kazanmıştır (Çağiltay & Tokdemir, 2010). İlişkisel veritabanları güncel veri yönetimi problemlerinin büyük çoğunluğuna çözüm getirebilecek niteliktedir. Öyle ki güncel uygulamaların birçoğu ilişkisel veritabanları kullanmakta ve tüm işlevlerini bu veritabanı türü ile yönetebilmektedirler.

İlişkisel veritabanı sayesinde tüm veri, birbirleriyle bağlantılı veri kümelerine bölünmektedir. Her bir küme, aralarında fonksiyonel bağımlılık olan niteliklerin bir araya gelmesiyle oluşmaktadır. Fonksiyonel bağımlılık; bir nitelik kümesinin, tek bir nitelik ya da bir nitelik kümesi tarafından temsil edilmesiyle ortaya çıkmaktadır. Diğer bir deyişle x_i ve y_i adlı iki nitelik kümesinde x 'in herhangi bir değeri y 'nin yalnızca tek bir değerine denk gelmesi durumunda y , x 'e fonksiyonel bağımlıdır ifadesi kullanılabilir (Riordan, 2005). Şekil 1, örnek bir veritabanı tasarımı sunmaktadır.

Şekil 1. Örnek Bir İlişkisel Veritabanı Tasarımı

Şekil 1, kaz ayağı türünde örnek bir varlık-ilişki diyagramı (*entity-relationship diagram*) sunmaktadır. Bu tasarım, kişilerin eğitim kurumlarına kayıtlarının geçmiş ve güncel kayıtlarını tutabilecek niteliktedir. Kavram bütünlüğünü ve bilgi seviyesini eşitlemek için burada diyagram bileşenlerinden kısaca bahsetmek gerekir. Diyagram, Kisi, Kayit ve EgitimKurumu adlı 3 varlık (tablo) içermektedir. Her varlık 2 boyutlu birer veri tablosu olarak düşünülebilir. Kisi varlığı id (*identification*), ad ve soyad adlı 3 nitelik (sütun); Kayit varlığı id, kisi_id, egitimKurumu_id, kayitTarihi ve mezuniyetTarihi niteliklerini; EgitimKurumu varlığı ise id, ad ve tur niteliklerini içermektedir. Kisi ve Kayit tabloları “1’e çok”; Kayit ve EgitimKurumu tabloları “çoğa 1” türünde ilişkiler ile bağlanmıştır. PK (*primary key*) kısaltması birincil anahtarı, FK (*foreign key*) kısaltması ise ikincil anahtarı ifade etmektedir. Birincil anahtarlar, bir varlık içerisinde tüm nitelikleri temsil eden bir nitelik oluşturmaya yarayan sistematik çözümlerdir. Genelde sıralı sayı değerler alır ve benzersiz olurlar. Bu sayede varlık içerisindeki her bir kayıt benzersiz bir değerle temsil edilmiş olur. Diğer bir deyişle fonksiyonel bağımlılık sağlanır. İkincil anahtarlar ise varlıklar arasındaki bağlantının sağlanabilmesi için bir varlık-taki birincil anahtarın başka bir varlık içerisinde nitelik olarak yer almasıdır.

Artık Şekil 1 ile verilen veri yapısını irdeleyebiliriz. Elbette ki bu yapının sağladığını MS Excel ya da benzeri bir yazılımla da tek bir dosya ile gerçekleştirebiliriz. Tablo 1’de bunun bir örneği verilmiştir.

Tablo 1. Veritabanı Tasarımının Klasik Veri Yapısıyla Gösterimi

ad	soyad	kayitTarihi	mezuniyetTarihi	kurumAdi	kurumTuru
...	...	...	...	...	...

Daha pratik gibi gözüксе de verinin bu biçimdeki bir yapıda tutulması beraberinde bazı problemler getirmektedir. Bu problemleri ve sebeplerini biraz tartışmak gerekir.

2.1. Avantajlar

Tablo 1'deki veri yapısı, çok fazla sayıda gereksiz tekrara yol açmaktadır. Bir kişiye ait birden fazla kayıt, ad ve soyad alanlarındaki verilerin tekrarlanmasına; benzer şekilde bir kurumun birden fazla öğrenciye sahip olması durumunda da kurum adı ve türü değeri kendini tekrar etmek zorundadır.

Şekil 1 ile verilen yapıda, her kişi ve kurum birer kez, kişilerin kurumlara kaydı ise ayrı bir tabloda bir kez kaydedildiği için Tablo 1'e göre çok daha az hacme sahip olabilecektir. Böylece daha büyük veri yapıları ilişkisel veritabanları ile daha kolay saklanabilmektedir.

Her veri kaydının bir kez tutulması ve bir birincil anahtarla işaret ediliyor olması, kullandığımız kolaylığın yanı sıra güncellemede de büyük kolaylık getirmektedir. Dosya temelli bir yaklaşım ile tutulan veride güncellenmek istenen bir bilgi; tüm dosyada aranmalı ve bulunan her kayıt tek tek değiştirilmelidir. Bu noktada eksik ya da fazla değişiklik veri tutarsızlığına yol açabilir. Örnek vermek gerekirse, kurum adı değişikliği gerçekleştirilmek istenildiğinde Tablo 1 ile verilen yapıda değiştirilmek istenilen kurum adı aranmalı ve bulunanlar güncellenmelidir. Ancak aynı ada sahip iki kurum olması durumunda doğrulayıcı bir kontrol noktası daha gerekli olacaktır. İlişkisel veritabanlarında her kayıt bir kez tutulduğu için bu tarz bir değişikliğin tek bir yerde gerçekleştirilmesi tüm sistemin güncellenmesi anlamına gelmektedir.

İlişkisel veritabanları, kullanılacak yöntem ya da verinin içeriği ile ilgili değildir. Basit bir web sitesi, karmaşık bir analiz ya da makine öğrenmesi uygulaması için ilişkisel veritabanı kullanılabilir. Önemli olan ele alınacak veri kümesi için uygun veritabanı tasarımını gerçekleştirebilmektir. Uygun veritabanı yapısı geliştirme ortamına kolayca entegre edilebilir. Hemen hemen tüm araçlar, ilişkisel veritabanlarına bağlanmak ve onları sorgulamak için faydalı özellikler içermektedirler.

Elde bir veri dosyası varsa bu dosyayı açabilir, güncelleyebilir ve kapatabilirsiniz. Bu dosyaya iki kişinin erişimi varsa bu kişiler sırasıyla işlemlerini gerçekleştirebilir. Ancak ilgili dosya üzerinde yüzlerce kişinin aynı anda okuma/güncelleme yapması gerekiyorsa burada özel bir yöntem gerekir. Dosya tabanlı yaklaşım ile bunu gerçekleştirmek son derece zor iken ilişkisel veritabanı yönetim sistemleri bunu kolaylaştıran algoritmalara sahiptir.

Dosya temelli yapılar, genelde istemciler üzerinde bulundurulurlar. Ortak bir sunucu üzerinde oldukları durumda dahi okunmaları ve güncellenmeleri oldukça zahmetlidir. Buna karşın ilişkisel veritabanları, bir sunucu üzerinde yapılandırılmaya oldukça elverişlidir. Bu sayede merkezi bir veri yapısı elde edilmiş olur. SQL (*Structured Query Language – Yapılandırılmış*

Sorgu Dili) adı verilen, metnin devamında ayrıntılarından bahsedilecek olan sorgulama dili sayesinde çok sayıda kişi aynı anda ilişkisel veritabanı üzerinde işlem gerçekleştirebilir. Veritabanı yönetim sistemi, aynı anda gelecek olan sorgular için bile kontrol mekanizmalarına sahip olduğu için verinin yönetimi çok daha kolaylaşmış olur.

Bir veri kümesi; birden fazla platformda aynı anda kullanılmak isteniyor olabilir. Kullandığımız birçok internet tabanlı sistem hem web hem de mobil sürümlere sahiptir. Sistemlerin web sürümlerinin veritabanına bağlanma yöntemi ile mobil uygulamaların veritabanlarına bağlanma yöntemleri büyük farklılıklar içermektedir. İlişkisel veritabanları, çoğu platform üzerinden çağrılabilmelerine olanak sağlayan sürücü yazılımlara sahiptirler. Bu sayede bir sunucu üzerinde yapılandırılan bir veritabanı; internete bağlı neredeyse tüm platformlar üzerinden sorgulanabilir niteliktedir.

Sayılan tüm bu özellikler, ilişkisel veritabanları için birer avantaj olarak görülmektedir (Sumathi & Esakkirajan, 2007; Zeng, Cao, Zhu, & Author, 2010). Ayrıca geleneksel ilişkisel veritabanı yönetim sistemleri güçlü stabilite altındaki yapılandırılmış verileri saklamak ve sorgulamak için güçlü mekanizmalar sağlar. Bunun yanında yüksek güven seviyesi, kararlılık ve sürekli geliştirmeye gelen destek sayesinde sorgu sonuçlarının sağlıklı olacağını garanti eder (Gessert et al., 2017).

2.2. Sorgulama

İlişkisel veritabanlarının yönetiminde sorgular kullanıldığından daha önce bahsedilmişti. SQL olarak adlandırılan yöntem, birçok varlıktan oluşan bir veritabanında, ihtiyaca göre bağlantılar kurma ve istenilen filtrelemeler neticesinde sorgu sonucunun elde edilmesini sağlayabilmektedir. Basit bir örnek olması açısından, tüm eğitim kurumlarının listesini alabileceğimiz SQL kodunu inceleyelim.

```
SELECT    *
FROM      Kurum
```

Burada SELECT anahtar kelimesi, bir okuma sorgulaması yaptığımızı göstermektedir. FROM ise veritabanındaki hangi varlığın sorgulandığını belirtir. Burada açıktır ki yalnızca Kurum varlığı ele alınmaktadır. SELECT kelimesinin yanına, seçilen varlık ya da varlıklardan hangi niteliklerin getirileceğini belirtmektedir. Bu değeri * olarak seçmek, tüm alanların elde edilmesini sağlayacaktır. Sonuç olarak id, ad ve tr başlıklarına sahip 3 sütunlu bir veri yanıtı elde edilir.

SQL sayesinde Şekil 1 ile verilen yapının Tablo 1 ile verilen yapı olarak elde edilmesi mümkündür. Bunu sağlayacak sorgu kodu aşağıdaki gibidir.

```

SELECT      Kisi.ad as ad,
            Kisi.soyad as soyad,
            Kayit.kayitTarihi as kayitTarihi,
            Kayit.mezuniyetTarihi as mezuniyetTarihi,
            EgitimKurumu.ad as egitimKurumu,
            EgitimKurumu.tur as kurumTuru
FROM        Kisi, Kayit, Kurum
WHERE       Kayit.kisi_id = Kisi.id
AND         Kayit.egitimKurumu_id = EgitimKurumu.id

```

Verilen sorgu, Kisi, Kayit ve Kurum varlıklarını kullanarak; Kisi ve Kayit varlıklarını Kisi varlığının birincil anahtarı, Kayit ve EgitimKurumu varlıklarını ise EgitimKurumu varlığının birincil anahtarı üzerinden bağlayarak SELECT anahtar kelimesi yanındaki tüm nitelikleri içerecek biçimde elde edilmesine olanak sağlamaktadır. Burada * ifadesi kullanılmamış, bunun yerine seçilen nitelikler birer birer yazılmıştır. Bunun iki sebebi bulunmaktadır. Birincisi, tablolarda bulunan birincil ve ikincil anahtarların sorgu sonucunda ekrana bastırılması istenilmemektedir. İkinci sebebi ise Şekil 1 ile Tablo 1 arasındaki nitelik adı farklılıklarıdır. Örneğin Şekil 1’de Kurum varlığındaki ad niteliği, Tablo 1’de egitimKurumu olarak karşımıza çıkmaktadır. Bu güncelleme, SQL sorgusunda as anahtar kelimesiyle gerçekleştirilebilmektedir. Ancak bu değişikliğin asıl veri yapısı üzerinde bir güncellemeye sebep olmadığı; yalnızca ekrana bastırılacak sonuçta bu değişikliğin yansıtılacağı önemli bir ayrıntıdır.

Okuma işlemleri için SELECT, güncelleme işlemleri için UPDATE, silme işlemleri için DELETE, yeni kayıt girme işlemleri için ise INSERT INTO komutu kullanılır. Veritabanının ayrıntılı yönetimi için birçok komut seçeneği mevcuttur ancak SQL işlemlerinin birçoğu bu temel komutlar ile gerçekleştirilmektedir.

2.3. Normalizasyon

Halihazırda mevcut bir veri kümesinin ilişkisel veri yapısı biçimine kavuşturulması, veritabanı tasarımcılığının görevi dahilindedir. İlişkisel veri yapısı oluşturma süreci, eldeki veri kümesinin incelenmesi ve fonksiyonel bağımlılıkların belirlenmesi ile başlayacaktır. Devamında veritabanı tasarımcısı, ilişkisel veritabanı kurallarına uygun bir yapı elde eder. Ancak elde edilen yapının en etkin yapı olup olmadığı test edilmelidir. Bu test, normal form adı verilen kurallar ile gerçekleştirilir. Literatürde bulunan 7 normal form;

- 1. Normal Form
- 2. Normal Form
- 3. Normal Form

- Boyce-Codd Normal Formu
- 4. Normal Form
- 5. Normal Form
- Alan/Anahtar (6.) Normal Form

olarak sıralanabilir (Fagin, 1981; Hoffer, Ramesh, & Topi, 2016). Normal formlar, ilişkisel veritabanı yapısında olması ya da olmaması gereken durumları içermektedir. Veritabanı tasarımcısı, etkin bir veritabanı inşa edebilmek için normal formlara uygun bir tasarım elde etmeye çalışmaktadır. Yine de bu süreç öznel kabul edilir ve tasarımcının ulaştığı sonucun en iyi çözüm olup olmadığı garanti edilememektedir (Akadal, 2017).

3. Özel Çözümler

İlişkisel veritabanları birçok uygulama için çözüm getirebilir. Güncel birçok veritabanı yönetim sistemi çok fazla sayıda varlık, kayıt ve sorgu sayısı ile başa çıkabilecek niteliktedir. Bu sebeple her yeni projede öncelikle bir ilişkisel veritabanı kullanma fikri değerlendirilmektedir. Ancak bazı durumlarda, özelleştirilmiş veritabanı çözümleri daha performanslı sonuçlar elde edilmesinde fayda sağlayabilir. Bu başlık altında sık karşılaşılan özel veritabanı çözümlerinden bahsedilecektir.

3.4. NoSQL Veritabanları

İnternetin ve bulut bilişimin gelişmesiyle birlikte büyük verinin okuması ve yazılmasında ilişkisel veritabanlarının yetersiz kalması sebebiyle daha performanslı veritabanı çözümlerine ihtiyaç duyulmuştur (Gessert et al., 2017; Han, Haihong, Le, & Du, 2011). İlişkisel veritabanlarında veri yapısı başlangıçta belirlenmekte, bundan sonraki tüm süreçler ve sorgular bu veritabanı yapısına uygun gerçekleştirilmektedir. Bu sebeple ilişkisel veritabanları oldukça katı bir yapıda olarak görülür (Stonebraker, 2010). Katı bir yapı büyük veri problemleri için, özellikle de yapılandırılmamış ya da eksik verinin çok olduğu durumlarda dezavantaj yaratmaktadır. Büyük veri işleme problemlerini aşmak için çeşitli veritabanı yaklaşımları ortaya çıkmıştır (Han, Haihong, et al., 2011). “Not Only SQL” ifadesinin bir kısaltması olan NoSQL adlı bu veritabanları, geleneksel ilişkisel veritabanlarına göre büyük farklılıklar göstermektedir (Han, Haihong, et al., 2011). Ayrıca “Not Relational SQL” anlamına da gelebileceği, üzerinde henüz uzlaşma sağlanamamış bir konudur (Cattell, 2010). Bu veritabanı türü, yatay ölçekleme ve ilişkisel veritabanlarından daha yüksek geçerliliğe sahip olması sebebiyle büyük veri süreçleriyle daha kolay başa çıkabilmektedir (Gessert et al., 2017).

NoSQL veritabanları;

- Düşük gecikmeli, yüksek tutarlılığa sahip okuma ve yazma
- Etkili büyük veri saklama ve erişim gereklilikleri
- Yüksek yatay ölçeklenebilirlik ve erişilebilirlik ile çok sayıda sunucuda çalışma
- Basit sorgulamalar sayesinde düşük yönetsel ve operasyonel maliyet
- Kaynakların etkin kullanımı
- Dinamik veri yapısı

açılarından avantajlı çözümlerdir (Cattell, 2010; Fan, 2010; Han, Song, & Song, 2011). Bunların yanında ilişkisel veritabanlarına göre daha zayıf tutarlılık göstermesi ise bir dezavantajdır (Cattell, 2010).

NoSQL veri modelleri anahtar-değer (key-value), doküman deposu (document stores), geniş sütun deposu (wide-column stores) ve sütun yönelimli (column-oriented) olarak sıralanmaktadır (Gessert et al., 2017; Han, Haihong, et al., 2011).

Anahtar-değer modelinde her bir değer bir anahtara bağlanır. Böylece çok basit bir yapı kurulmuş olur. Bu yapıdaki sorgu hızı ilişkisel veritabanlarından daha yüksektir. Çok büyük veri depolarını büyük tutarlılık oranlarıyla destekler (Han, Haihong, et al., 2011). Sütun yönelimli veritabanında kayıtlar satırlar yerine sütunlar olarak tutulmaktadır. Bu yöntem genellikle veri yığını oluşturma uygulamalarında daha avantajlıdır (Han, Haihong, et al., 2011). Doküman deposu veritabanı yapısı, anahtar-değer yapısı ile oldukça benzerdir. Değer yerine doküman gelerek anahtarlarla dokümanların temsil edilmesi sağlanır. Dokümanlar genellikle JSON ya da XML formatındadır (Han, Haihong, et al., 2011).

Anahtar-değer veritabanları için Redis, Tokyo Cabinet-Tokyo Tyrant, Flare, Project Wol-demort, Riak, Scalaris, Memcached, Membrain, Membase, Dynamo; sütun yönelimli veritabanlar için Cassandra, Hypertable; doküman deposu veritabanları için MongoDB, CouchDB, SimpleDB, Terrastore örnek olarak verilebilir (Cattell, 2010; Han, Haihong, et al., 2011; Stonebraker, 2010).

3.5. Hafif (*Lightweight*) Veritabanları

Bir önceki başlıkta ilişkisel veritabanlarının çözüm üretmediği ya da düşük performanslı kaldığı büyük hacimdeki veriler için kullanılan NoSQL veritabanlarından söz edildi. Bu alt başlık ise doğrunun diğer ucuyla ilgili. İlişkisel veritabanları, merkezi yönetim için bir sunucu

yapılandırmasına ihtiyaç duyar. Ancak bazı durumlarda bu donanım konfigürasyonuna sahip olmayabiliriz ya da ilişkisel veritabanı yapacağımız iş için fazla büyük bir yapı olabilir. Bölümün başlarında sunduğumuz telefon rehberi örneği buna bir örnektir. Bir diğer örnek mobil cihazımızdaki uygulamalardır. Kendi içerisinde ilişkisel yapıda veri saklaması gereken mobil uygulamalar da hafif (*lightweight*) veritabanlarına ihtiyaç duyarlar. Bu veritabanları en az kaynak tüketimiyle en iyi performansı sunmaya odaklanırlar. Dolayısıyla yetenekleri kısıtlıdır ancak çoğu durumda sahip oldukları yetenekler ihtiyacı karşılamaya yetecek düzeydedir.

Güncel durum incelendiğinde önde gelen hafif veritabanlarına verilebilecek örnekler SQLite, NeDB, LiteDB, RxDB, LevelDB, OrientDB, MonetDB, FileDB, WhiteDB ve ObjectDB olarak sıralanabilir (Slant, 2020). Birçok alternatifi bulunsa da SQLite en sık kullanılan hafif veritabanı türüdür. Öyle ki tüm veritabanları sıralaması içerisinde Nisan 2020’de dokuzuncu, hafif veritabanları içinde birincidir (Şekil 2).

Şekil 2. Nisan 2020 Veritabanları Kullanım Sıralaması

3.6. Blok Zincir

Nakamoto (2019) tarafından öne sürülen ve en yeni veritabanı teknolojilerinden biri olan Blok Zincir (*blockchain*), dijital işlem kayıtlarının ya da bu kayıtların yer aldığı dosyaların (*ledger*) saklanması sağlayan dağıtık veritabanıdır (Crosby, Nachiappan, Pattanayak, Verma, & Kalyanaraman, 2016). Bu tanımdaki en önemli unsur “dağıtık” olmasıdır. Klasik veritabanı yönetimi anlayışında, tüm veritabanının bir ya da eşlenik çalışan birden fazla sunucuda

barındırılması gerekmektedir. Bu yapı, her bireyin sahip olduğu erişim izni çerçevesinde veritabanında işlem yapmasına olanak sağlamakla birlikte; veri bütünlüğü ve tutarlılığı da ön plana koymaktadır. Ancak burada sözü edilmeyen ayrıntı, sunucuların yönetimini elinde bulunduran kişi ya da otoritenin tüm veriyi yönetme yetkisine sahip olduğudur. Bu özellik, geleneksel veritabanı yönetiminin “merkezi” olduğunu göstermektedir. Blok Zincir teknolojisiyle ise veritabanının yönetimi kişi ya da kurumdan alınarak konsensüse verilmektedir. Böylece oluşturulan bir veritabanı, kimin oluşturduğundan bağımsız olarak konsensüsün aldığı kararlara göre varlığını sürdürür. Bir Blok Zincir’in kapatma tuşu yoktur. Ancak topluluk desteklemeyi bıraktığında kendiliğinden yok olacaktır.

Blok Zincir’in çalışma mantığını anlayabilmek için öncelikle özetleme fonksiyonlarına (*hash function*) bir göz atmak gerekir. Özetleme fonksiyonları, herhangi bir uzunluktaki metin, sayı ya da bunların kombinasyonlarını sabit uzunluktaki bir sayıya dönüştürür (Velde, 2013). Özetleme işlemi tek taraflı bir işlemidir. Her bir girdi için bir özet (*hash*) karşılığı elde edilebilir ancak bir özetin hangi girdiye denk geldiğini bulmak neredeyse imkansızdır.

Şekil 3. Örnek Bir Veri Özetleme İşlemi

Şekil 3’te, “42” içeriği bir özetleme fonksiyonu olan SHA-256 kullanılarak özetlenmiş ve özet değeri olarak “73475cb40a568e8da8a045ced110137e159f890ac4da883b6b17dc651b3a8049” elde edilmiştir. Bu ifade, 16 tabanında yazılmış bir sayıdır. SHA-256 algoritması verilen girdiye karşılık 256 bitlik bir sayı üretir. Sayı, daha kısa bir gösterimle; yani 16 sayı tabanında yazılarak 64 karakter kullanılarak gösterilebilmektedir. Bir özetleme fonksiyonu için girdi, “42” gibi kısa bir ifade olabileceği gibi büyük boyutlara sahip dosyalar da olabilir. 1 terabayt büyüklüğündeki bir dosyanın da SHA-256 ile özetlenmesi neticesinde 64 karakterli ve 16 sayı tabanında bir sayı elde edilecektir.

Blok Zincir, işlem kayıtlarını (ya da bunların bir özetini) içeren ve blok adını alan veri kümelerinden oluşmaktadır. Blokların boyutlarının üst limiti, Blok Zinciri oluşturulurken tanımlanır. Bir blok, boyutları açısından üst limite ulaştığında ya da tanımlanan bekleme süresi sona erdiğinde yeni bir blok oluşturulur. Yeni blok, önceki bloğun tamamının bir özetini içermektedir. Bu koşul blokların birbirine bağlanmasını sağlar. Verilen örnekteki gibi ardarda bağlanmış 10 blok olduğunu düşündüğümüzde; örneğin 2. blokta yer alan tek bir kayıt bile

değiştirilirse 2. bloğun özeti değişecektir. 2. bloğun özetini içeren 3. blok da bu durumda değişmiş olacak, benzer olay sonrasındaki tüm bloklarda yaşanacaktır. İkinci blokta yapılan bir değişiklik sonraki blokların tamamının hatalı olmasına yol açacaktır. Bu da Blok Zincir yapısındaki zincirin kırılması anlamına gelmektedir.

Blok Zincir konsensüsü zincirin tamamını doğrulamakla görevlidir. İstemcilerin elindeki en uzun zincir, geçerli zincir olarak kabul edilir ve tüm ağa dağıtılır. Bu sayede kötü niyetli bir güncellemede, tüm zinciri uygun şekilde güncellemek zaman alacağı ve bu zaman geçene kadar gerçek zincire yeni bloklar ekleneceği için zincir korunmuş olmaktadır. Blokları oluşturmanın zaman alıcı bir eylem olması teknik ve metnin ana konusu dışında olduğu için ayrıntılı olarak ele alınmamıştır. Daha fazla ayrıntı, İş İspatı (*Proof of Work – PoW*) algoritması incelenerek edinilebilir.

Blok Zincir, veri güvenliğinin ön planda olduğu bir çok alana yeni bir bakış açısı getirmiştir. Herhangi bir otorite ya da yönetimin yaptırım gücünün olmadığı, tüm ağa açık şekilde izlenebilen ve ortak biçimde yönetilebilen bir veri yapısı başta finans dünyası olmak üzere birçok alanda heyecan uyandırmaktadır. Blok Zincir teknolojisinin en ünlü uygulaması olan Bitcoin, arzı sınırlı olan bu dijital paranın transfer kayıtlarını tutan bir Blok Zincire sahiptir. Bu sayede Bitcoin, uluslararası dolaşımda özgürce yer alabilmektedir. Çünkü aslında dolaşımda olan kripto paranın kendisi değil; para transferlerini, dolayısıyla da hesap bakiyelerini içeren Blok Zincirin tüm dünyada erişime açık olmasıdır. Bunun yanında Blok Zincir teknolojisinin gerçek dünyada fiziksel alışveriş (Notheisen, Cholewa, & Shanmugam, 2017), nesnelerin interneti (Bahga & Madiseti, 2016), kimlik doğrulama (Lin et al., 2018) ve e-devlet uygulamaları (Hou, 2017) gibi birçok alana uygulandığı da görülmektedir.

3.7. Donanım Destekli Veritabanları

Veritabanı sistemleri genellikle yüksek işlem gücü gerektiren komutlardan öte, çok sayıda komutun bir sıraya alınarak en kısa zamanda işlenerek yanıt verilmesi odağında çalışmaktadırlar. Ancak bazı durumlarda veritabanı üzerinde işlem yapılması gerekebilir. PL/SQL ve T-SQL gibi modern yaklaşımlar, karmaşık alt yordamlar oluşturulmasına müsaade etmektedir. Alt yordamlar veritabanını çok daha becerikli yapmakla birlikte çok sayıda komut söz konusu olduğunda ciddi işlem yükü getirebilirler. Bu durumda veritabanının, işlem gücü anlamında da desteklenmesi gerekecektir. Birkaç yaklaşım, veritabanının donanım desteği ile hızlandırılması konusunda öneri ve testler gerçekleştirmiştir.

Donanım destekli hızlandırmada öncelikli olarak grafik kartları (*Graphincs Processing Unit – GPU*)’ndan bahsetmek gerekir. GPU, paralelleştirmeye açık yapısı ile son yıllarda

sıklıkla kullanılan güçlü bir işlem donanımdır. Veritabanı işlemlerinde de işlem gücünü artırma odaklı olarak kullanılmaktadır. CoGaDB, GPUDB, GPUQP, GPUDx, MapD, Ocelot, OmniDB, Virginian, multiQx-GPU ve hippogriffDB, GPU hızlandırması kullanan veritabanları için örneklerdir (Arefyeva, Broneske, Campero, Pinnecke, & Saake, 2018; Breß, Heimel, Siegmund, Bellatreche, & Saake, 2014; Fang, Mulder, Hidders, Lee, & Hofstee, 2020).

Sabit disklere erişmek ve içerisindeki veri ile işlem yapmak zaman maliyetine neden olmaktadır. Her ne kadar Katı Hal Sürücülerıyla (*Solid State Disk – SSD*) performans oldukça artmış olsa da söz konusu bir veritabanının yönetimi olduğunda ihtiyaç duyulan performans çok daha yüksek olacaktır. Çok büyük miktarda verinin gerçek zamanlı analiz edilmesindeki zorlukların aşılabilmesi için verinin sürekli Rastgele Erişimli Hafıza (*Random Access Memory – RAM*)’da tutulması gerekmektedir (Zhang, Chen, Ooi, Tan, & Zhang, 2015). RAM üzerinde çalıştırılarak hızlandırılmış bazı veritabanları şunlardır: HyPer, H-Store, Hekaton MS SQL Server, Oracle Timesten In-Memory Database, Silo, solidDB, SAP HANA, nuodb, memsql.

Veritabanı hızlandırması için kullanılan bir diğer yöntem Sahada Programlanabilir Kapı Dizisi (Field-Programmable Gate Array – FPGA) adlı donanımdır. FPGA, kullanıcı tarafından hazırlanacak bir program ile kullanılabilen bütünleşmiş devrelerdir (İçer, 2016). FPGA’lar, üretildikten sonra hem donanım hem de yazılımında isteğe bağlı güncellemeler yapılarak çeşitli amaçlarla kullanılabilirler. Ayrıca paralel işlem yapmaya uygun olması, performansını önemli ölçüde yükseltmektedir. Bu teknoloji, bazı dönemlerde performans artışı için veritabanlarıyla kullanılmış, bazı dönemlerde ise taşıdığı dezavantajlar sebebiyle geride kalmıştır. Ancak hala bu konuda iyileştirme çalışmaları gerçekleştirilmektedir (Fang et al., 2020).

3.8. Veri Ambarı

İşletmeler için veri, maden niteliğindedir. Özellikle veri akışının hızlı olduğu durumda veriyi değerlendirme işlemi de büyük bir hız ve doğrulukla gerçekleştirilmelidir. Ayrıca işletmeler genellikle birden fazla bileşene sahiptirler. Muhasebe, satış ve insan kaynakları bu bileşenlere örnek olarak sayılabilir. Tahmin edileceği üzere her bir bileşen bir veri kaynağı olmakla birlikte üretilen veri biçimi bileşenden bileşene değişiklik gösterecektir. İşletmenin, tüm bileşenlerden gelen veriyi topladığı yapıya veri ambarı (*Data Warehouse*), ambarı besleyen her bir bileşenin veri yapısına ise veri deposu (*Data Mart*) adı verilmektedir. Veri ambarları veriyi ilişkisel veritabanına göre farklı bir yapıda tutmaktadır. Bundan bahsedebilmek için öncelikle iki kavrama daha değinmek gerekir.

Çevrimiçi sistemlerde öncelikli görev kayıt ve sorgu işlemedir. Bu sistemler çevrimiçi kayıt işleme (*On-Line Transaction Processing – OLTP*) sistemleri olarak adlandırılırlar. Veri ambarı

sistemleri, veri analizi ve karar verme görevinde çalışan bilgi çalışanlarına ya da kullanıcılara hizmet vermektedirler. Bu tarz sistemler, farklı kullanıcı ihtiyaçlarını karşılayabilmek için çeşitli biçimlerdeki veriyi organize eder ve sunabilirler. Bu sistemler çevrimiçi analitik işleme (*On-Line Analytical Processing* – OLAP) sistemleri olarak adlandırılırlar (Satyanarayana Reddy, Srinivasu, Poorna, Rao, & Rikkula, 2010). OLTP, maksimum performans ile verileri saklama amacı taşıdığı için veritabanı ilkelerine bağlı, en yüksek performansı gösterecek yapıda gerçekleştirilir. Ancak OLAP daha çok verilerini sürekli anlamlandırmak isteyen organizasyonların, bazı veritabanı kurallarını ihlal ederek veriyi daha hızlı işlenebilir ve raporlanabilir hale getirmesine verilen ad olarak da görülebilir. Bölüm boyunca tanıtılan yöntemler performans, dolayısıyla OLTP yöntemi ile ilgiliyken, veri ambarları genellikle OLAP yöntemini kullanan yapılardır. Her ne kadar veritabanı ilkelerine karşı durulan noktalar olsa da OLAP sayesinde işletmeler veri ambarlarından çok daha hızlı ve faydalı raporlar elde edebilmektedirler.

4. Sonuç

Geçmişin ve şimdinin bir kaydının elde bulunması istatistik ve veri madenciliği yöntemleri sayesinde mevcut durumu daha net görmeye; bunun yanında gelecek hakkında gerçekçi yorum ve tahminler yapabilmeye olanak sağlamaktadır. Verileri kaydetmenin birçok yolu olduğu gibi herhangi birini rasgele seçmek gerçek amaca ulaşmaya engel olabilir.

Kitabın bu bölümünde, yaygın olarak kullanılan ilişkisel veritabanlarının ortaya çıkışı ve çalışma prensipleri ele alınmış; sonrasında da özel durumlar için hangi tür veritabanlarından faydalanılabileceği vurgulanmıştır. Büyük veri patlaması, ultra düşük gecikmeye sahip sistemler ve gerçek zamanlı veri analizini desteklemek için sistemler geliştirme araştırmalarına sebep olmuştur (Zhang et al., 2015). Nitekim veri çok büyüdükçe ya da aksine çok küçüldükçe seçilecek yöntemin türü değişmelidir. Klasik bir ilişkisel veritabanı çok büyük veri için olduğu kadar çok küçük bir alanda çalışırken de performans kaybına yol açabilecektir. Bunun yanında ortalama büyüklüğe sahip ama yüksek hıza ihtiyaç duyulan durumlarda donanımsal hızlandırma yapma yöntemleri de mevcuttur. İhtiyaç duyulan hızın türüne göre çeşitli donanım hızlandırma yöntemleri metin içerisinde ele alınmıştır. Eğer bir veritabanına erişen herkese veritabanının bütünlüğü ve değiştirilmemiş olması konusunda güvence verilmek isteniyorsa çözüm Blok Zincir olabilir. Veritabanı yönetimini merkeziyetten uzaklaştırarak dağıtık bir hale getiren bu yöntem sayesinde, verinin sahibi dahi veri üzerinde herhangi bir manipülasyon yetkisine sahip olamayacaktır.

Bu noktaya kadar sayılan yöntemler, hızlı ve stabil veri kaydını sağlamak üzere OLTP ile ele alınmaktadır. Ancak organizasyonlar gerçek zamanlı analizlere ihtiyaç duyuyor olabilirler.

Birden fazla veri deposundan gelerek veri ambarında toplanan veriler, raporlamaya uygun biçimde tutulan OLAP yapısında bulundurulurlar.

Güncel istatistiklere bakıldığında Nisan 2020 itibariyle ilişkisel veritabanlarının hala ciddi bir liderliğe sahip olduğu görülmektedir¹. Ancak bunun yanında özel durumlar için geliştirilen yöntemler de listede önemli sıralarda bulunmaktadırlar.

Görülen odur ki, bilgi hiyerarşisindeki veriden enformasyona geçiş süreci ancak doğru yöntemlerin seçilmesiyle sağlıklı şekilde gerçekleşebilmektedir. Enformasyonun, dolayısıyla bilginin niteliği, veriyi amaca uygun ele alma, kaydetme ve geri çağırabilme yeteneğimizle doğrudan ilişkilidir.

Kaynakça / References

- Akadal, E. (2017). *Ham Verilerin Genetik Algoritmalarla İlişkisel Veritabanlarına Dönüştürülmesi ve Bir Uygulama*. Istanbul University.
- Arefyeva, I., Broneske, D., Campero, G., Pinnecke, M., & Saake, G. (2018). Memory management strategies in CPU/GPU database systems: A survey. *Communications in Computer and Information Science*, 928, 128–142. https://doi.org/10.1007/978-3-319-99987-6_10
- Bachman, C. W. (1969). Data structure diagrams. *ACM SIGMIS Database*, 1(2), 4–10. <https://doi.org/10.1145/1017466.1017467>
- Bahga, A., & Madiseti, V. K. (2016). Blockchain Platform for Industrial Internet of Things. *Journal of Software Engineering and Applications*, 09(10), 533–546. <https://doi.org/10.4236/jsea.2016.910036>
- Breß, S., Heime, M., Siegmund, N., Bellatreche, L., & Saake, G. (2014). GPU-Accelerated Database Systems: Survey and Open Challenges. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8920, 1–35. https://doi.org/10.1007/978-3-662-45761-0_1
- Çağıltay, N., & Tokdemir, G. (2010). *Veritabanı Sistemleri Dersi*.
- Cattell, R. (2010). Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4), 12–27. <https://doi.org/10.1145/1978915.1978919>
- Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile Networks and Applications*, 19(2), 171–209. <https://doi.org/10.1007/s11036-013-0489-0>
- Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6), 377–387. <https://doi.org/10.1145/362384.362685>
- Crosby, M., Nachiappan, Pattanayak, P., Verma, S., & Kalyanaraman, V. (2016). BlockChain Technology: Beyond Bitcoin. In *j2-capital.com*. Retrieved from <https://j2-capital.com/wp-content/uploads/2017/11/AIR-2016-Blockchain.pdf>
- Fagin, R. (1981). A Normal Form for Relational Databases That Is Based on Domains and keys. *ACM Transactions on Database Systems (TODS)*, 6(3), 387–415. <https://doi.org/10.1145/319587.319592>
- Fan, K. (2010). Suvey on nosql. *Programmer*, 6, 76–78.
- Fang, J., Mulder, Y. T. B., Hidders, J., Lee, J., & Hofstee, H. P. (2020). In-memory database acceleration on FPGAs: a survey. *Vldb Journal*, 29(1), 33–59. <https://doi.org/10.1007/s00778-019-00581-w>
- Gessert, F., Wingerath, W., Friedrich, S., & Ritter, N. (2017). NoSQL database systems: a survey and decision guidance. *Computer Science - Research and Development*, 32(3–4), 353–365. <https://doi.org/10.1007/s00450-016-0334-3>

1 URL: <https://db-engines.com/en/ranking> Erişim Tarihi: 20.04.2020

- Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database. *Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011*, 363–366. <https://doi.org/10.1109/ICPCA.2011.6106531>
- Han, J., Song, M., & Song, J. (2011). A novel solution of distributed memory NoSQL database for cloud computing. *Proceedings - 2011 10th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2011*, 351–355. <https://doi.org/10.1109/ICIS.2011.61>
- Henry, W. R. (1969). Hierarchical structure for data management. In *ieeexplore.ieee.org*. Retrieved from <https://ieeexplore.ieee.org/abstract/document/5388345/>
- Hoffer, J., Ramesh, V., & Topi, H. (2016). *Modern Database Management*. Retrieved from <https://sites.google.com/site/books183pdf274/PDF-Modern-Database-Management-12th-Edition.pdf>
- Hou, H. (2017). The application of blockchain technology in E-government in China. *2017 26th International Conference on Computer Communications and Networks, ICCCN 2017*. <https://doi.org/10.1109/ICCN.2017.8038519>
- İçer, Y. (2016). *Temel Kenar Algılama Algoritmalarının FPGA Üzerinde Gerçeklenmesi*. Fırat Üniversitesi.
- Lin, Q., Yan, H., Huang, Z., Chen, W., Shen, J., & Tang, Y. (2018). An ID-Based Linearly Homomorphic Signature Scheme and Its Application in Blockchain. *IEEE Access*, 6, 20632–20640. <https://doi.org/10.1109/ACCESS.2018.2809426>
- Nakamoto, S. (2019). Bitcoin: A Peer-to-Peer Electronic Cash System. *SSRN Electronic Journal*. <https://doi.org/10.2139/ssrn.3440802>
- Notheisen, B., Cholewa, J. B., & Shanmugam, A. P. (2017). Trading Real-World Assets on Blockchain: An Application of Trust-Free Transaction Systems in the Market for Lemons. *Business and Information Systems Engineering*, 59(6), 425–440. <https://doi.org/10.1007/s12599-017-0499-8>
- Riordan, R. (2005). *Designing effective database systems*. Retrieved from <https://dl.acm.org/citation.cfm?id=1050930>
- Rowley, J. (2007). The wisdom hierarchy: representations of the DIKW hierarchy. *Journal of Information Science*, 33(2), 163–180. <https://doi.org/10.1177/0165551506070706>
- Satyanarayana Reddy, G., Srinivasu, R., Poorna, M., Rao, C., & Rikkula, S. R. (2010). Data Warehousing, Data Mining, Olap and Oltp Technologies Are Essential Elements To Support Decision-Making Process in Industries. *International Journal on Computer Science and Engineering*, 02(09), 2865–2873. Retrieved from <http://pwp.starnetinc.com/larryg/articles.html>
- Slant. (2020). 10 Best lightweight databases as of 2020 - Slant. Retrieved April 14, 2020, from <https://www.slant.co/topics/69/~best-lightweight-databases>
- Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4), 10–11. <https://doi.org/10.1145/1721654.1721659>
- Sumathi, S., & Esakkirajan, S. (2007). *Fundamentals of Relational Database Management Systems (Studies in Computational Intelligence)*.
- Thusoo, A., Sarma, J. Sen, Jain, N., Shao, Z., Chakka, P., Zhang, N., ... Murthy, R. (2010). Hive - A petabyte scale data warehouse using hadoop. *Proceedings - International Conference on Data Engineering*, 996–1005. <https://doi.org/10.1109/ICDE.2010.5447738>
- Velde, F. R. (2013). *Bitcoin: A primer*. Retrieved from www.chicagofed.org.
- Zeng, Q., Cao, Q., Zhu, X., & Author, C. (2010). A Complex XML Schema to Map the XML Documents of Distance Education Technical Specifications into Relational Database Xin-hua A Complex XML Schema to Map the XML Documents of Distance Education Technical Specifications into Relational Database. *CiteSeer*. <https://doi.org/10.4156/jdcta.vol4>
- Zhang, H., Chen, G., Ooi, B. C., Tan, K. L., & Zhang, M. (2015). In-Memory Big Data Management and Processing: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(7), 1920–1948. <https://doi.org/10.1109/TKDE.2015.2427795>