Süleyman Demirel Üniversitesi
Fen Bilimleri Enstitüsü Dergisi
Cilt 22, Sayı 2, 887-895, 2018

Süleyman Demirel University
Journal of Natural and Applied Sciences
Volume 22, Issue 2, 887-895, 2018

DOI: 10.19113/sdufbed.02548

Veritabanı Tasarımının Yazılım Performansına Etkisi: Normalizasyona karşı Denormalizasyon

Erdinç UZUN[1], Halil Nusret BULUŞ[1], Cihat ERDOĞAN*[1]

[1]Namık Kemal Üniversitesi, Çorlu Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 59860, Tekirdağ

(Alınış / Received: 25.05.2017, Kabul / Accepted: 29.11.2017, Online Yayınlanma / Published Online: 16.02.2018)

Anahtar Kelimeler
Veritabanı tasarımı,
Normalizasyon,
Denormalizasyon,
Yazılım performansı,
İndeksleme,
CAP teoremi

Özet: Yazılım performansını etkileyen en önemli faktörlerden biri veritabanı tasarımında yapılabilecek iyileştirmelerdir. Veritabanı tasarımında sıklıkla ilişkisel veritabanı teorisi olan normalizasyon işlemi kullanılır. Fakat veri miktarı arttıkça normalizasyon işleminden kaynaklı performans sorunları ortaya çıkmaya başlar. Performans sorunlarını ortadan kaldırmak için teorisi oluşmamış denormalizasyon işlemi kullanılır. Bu çalışmada, bir anket uygulamasında performans arttırıcı bir veritabanı tasarımı tanıtılmış ve bu veritabanı tasarımının MySQL, PostgreSQL ve Oracle olmak üzere üç farklı ilişkisel veritabanı yönetim sistemindeki performans artışı incelenmiştir. Ayrıca, günümüzün popüler veritabanı sistemlerinden NoSQL'e ne zaman geçilmesi gerektiği CAP teoremi üzerinden anlatılıp, normalizasyon ve denormalizasyon işlemlerinin bu teoremdeki yeri belirtilmiş olacaktır.

Impact of Database Design on Software Performance: Normalization vs. Denormalization

Keywords
Database design,
Normalization,
Denormalization,
Software performance,
Indexing,
CAP theorem

Abstract: One of the most important factors affecting software performance is the improvements that can be made in database design. The normalization process, which is based on the relational database theory, is often used in database design. However, as the amount of data increases, performance problems arise due to the normalization process. In order to overcome the performance problems, denormalization without theoretical process is utilized. In this study, a performance enhancement database design is introduced in a survey application and the performance improvements of three different relational database management systems including MySQL, PostgreSQL and Oracle are examined. In addition, it is explained through CAP theory when to pass to NSQL, one of today's popular database systems, and the place of normalization and denormalization processes in this theory.

1. Giriş

Veritabanı tasarımı yazılımın performansını etkileyen en önemli kıstaslardan biridir. İlişkisel veritabanı tasarım teorisi olan normalizasyon işlemleri 1970'li yıllarda Dr. Edgar F. Codd tarafından önerilmiştir [1]. Günümüzde birçok uygulamanın veritabanı tasarımı bu işlemleri kullanır. Fakat 1990'lı yıllarda veri miktarının artması ile birlikte sorgu performansında düşüşler olmaya başlamış ve bunun için de normalizasyon kurallarının dışına çıkılarak performans çözümleri aranmaya başlanmıştır [2]. Bu çalışmada, normalizasyonun doğurduğu kusurlar ve bu kusurlardan kurtulmak için denormalizasyon işleminin performans getirisi fakültemizde iki senedir kullanılan web tabanlı bir anket yazılımının veritabanı tasarımı üzerinden anlatılacaktır.

Normalizasyon, veri tekrarını azaltıp veri bütünlüğünü iyileştirmek için bir ilişkisel veritabanındaki özellikleri ve tabloları düzenleme işlemleridir. Normalizasyon işlemlerinin kuralları 1970 yıllarda oluşturulmuştur. En temel normalizasyon kuralı 1 NF (Normal Form) atomik ilişkilerin çözümlenmesi, başka bir deyişle satır ve sütunlara karar verilmesidir. 2 NF kısmi fonksiyonel bağımlılıkların çözümlenmesi iken 3 NF geçişli fonksiyonel bağımlılıkların çözümlenmesi yani tasarımın birden fazla tabloya ayrılıp ilişkilerin belirlenmesi işlemidir [3]. 3NF işlemi tamamlanmış

* İlgili yazar: cerdogan@nku.edu.tr

bir veritabanı tasarımı, normal duruma gelmiş yani iyi bir tasarım olarak kabul edilir. 1974 yılında Edgar Codd ve Raymond F. Boyce tarafından her belirleyicinin bir anahtar olmasından kaynaklı problemin çözümü için BCNF (Boyce-Codd Normal Form) sunulmuştur. Bunun yanı sıra, 4NF ve 5NF kuralları var olsa da uygulama tarafında fazla tercih edilmemektedir [4]. Bu çalışmada, öncelikle 3NF durumuna gelmiş bir veritabanı tasarımının performans sorunları incelenip normalizasyon kuralları dışına çıkılarak performans için çözüm üretilmesi amaçlanmaktadır. Ayrıca, bu çözümün performans getirileri de incelenecektir.

Veritabanındaki kayıt sayısı arttıkça verinin içinde arama yapılması ve verinin içinden bilgi çıkarılması gibi konularda performans kaybı kaçınılmazdır. İndeksleme [5] ile verinin aranmasını hızlandırmak ve OLAP (OnLine Analytical Processing) [6] ile hızlı bilgi çıkarımı yapabilmek mümkündür. İndeksleme işlemini popüler veritabanı yönetim sistemlerinde gerçekleştirmek son derece basit bir işlemdir. Sadece indeksleme yapılması düşünülen alanı veya alanları seçip veritabanı yönetim sisteminin sunduğu indeksleme yöntemlerinden biri ile indekslemek yeterlidir. İndeksleme arama işlemini hızlandırsa da depolama maliyeti açısından bir olumsuzluk olarak görülmektedir. Diğer taraftan OLAP ise verideki önemli özellikleri belirleyip hızlı bir şekilde bilgi çıkarımına olanak sağlayan ek yazılımlardır. Popüler veritabanı yönetim sistemlerine ek yazılımlar ile OLAP desteği sunulmaktadır. Bu çalışmada, 3 farklı veritabanı yönetim sistemi için indekslemenin getirilerinin yanı sıra, veritabanı tasarımında yapılacak denormalizasyon işlemlerinin getirileri de karşılaştırılacaktır. Ayrıca, OLAP'a gereksinimi ortadan kaldıracak denormalizasyon işleminin veritabanı tasarımındaki performans getirileri de incelecektir.

Denormalizasyon işlemi bilgi çıkarımını daha hızlı şekilde yapmak için veritabanı tasarımına ek özellikler ekleme veya veritabanı tasarımındaki mevcut özellikleri birleştirme olarak tanımlanabilir. Denormalizasyon işlemi kesin bir teorisi olan ama kuralları normalizasyon kadar açık olmayan bir tekniktir. Bu teknikte, sadece veritabanı tasarım tarafına değil aynı zamanda yazılım tarafına da hâkim olan bir uzman geliştirici olmak gerekir. Bu uzman geliştirici işlem karmaşıklığını göz önüne alarak normalizasyon kurallarının dışına çıkabilmelidir. Bu kuralların dışına çıkılması ile problemin içeriğine bağlı olarak çok farklı tasarımlar ortaya çıkarabilir. Bu çalışmada, anket uygulaması için geliştirdiğimiz veritabanı tasarımı anlatılacaktır.

Veri sayısı arttıkça büyük veri kavramına bir yaklaşım söz konudur. Büyük veriyi yönetmenin en temel tekniklerinden biri dağıtık sistemleri kullanmaktır. Günümüzün problemlerinden biri de bu dağıtık sistemlere ne zaman geçilmesi gerektiği sorusudur. Bu çalışmada, anket uygulamasının CAP teoremindeki yeri anlatılacaktır. Ayrıca, normalizasyon ve denormalizasyon işlemlerinin bu teoremdeki yeri literatürde incelediğimiz kadarıyla ilk defa ortaya konulacaktır.

2. İlgili Çalışmalar

Bu Performans için denormalizasyon işleminin temelleri ilk olarak Finkelstein ve ark. tarafından tanıtılmıştır [3]. Bu çalışmada, problemi iyi bilen bir uzman ve iyi bir veritabanı tasarımcısının yaptığı denormalizasyon işlemleri ile veritabanı tarafında performans artışı sağlanabileceği belirtilmiştir. Cerpa ve birçok araştırmacı, denormalizasyon işleminin uzman bir tasarımcı tarafından yapılması gerektiğinin altını çizmiştir [8, 9, 10]. Coleman, denormalizasyon işlemini yaparken istenen performansa ulaşmak için veri miktarının, hangi veritabanı yönetim sisteminin kullanıldığının, işletim sisteminin ve hatta donanımın dikkate alınması gerektiğini belirtmiştir [11].

Denormalizasyon işleminin temel ilkesi "Önce normalize et, eğer performans problemi var ise denormalize işlemi uygula" şeklindedir. Denormalizasyon işlemini, veri miktarının artması sonucu sorgu sürelerinin yavaşlamaya başlamasına karşı üretilen çözümler olarak düşünebiliriz [12]. Örneğin bu çözümler:

- Birden fazla tablodan veri çekme sorgu sürelerinin yavaşlaması durumunda alt tablolardaki verinin üst tablolara taşınması işlemi, kısacası JOIN işlemini azaltma işlemi
- SQL fonksiyonları (SUM, AVG, COUNT, MIN, MAX vs) oluşturulmuş sorguların yavaş kalması durumunda hesaplama sonuçlarının ayrı bir özellik olarak bir tabloya yazılması işlemi
- Birden fazla satırı ve/veya sütunu birleştirip tek sütunluk özelliğe çevirme işlemi

şeklinde gruplandırılabilir. Bu çalışmada, bu üç çözüm göz önüne alınarak yavaşlayan anket sorgu sürelerine karşı bir denormalizasyon işlemi yapılacaktır. Fakat, bu çözümün tek denormalizasyon çözümü olarak düşünmemek gerekir. Aynı problem için birden fazla denormalizasyon çözümü üretilebilir. Hatta problem çok büyüdüğünde veri miktarı çok arttığında NoSQL tarzı çözümleri [13, 14] ve CAP teoremini [15, 16] dikkate almak gerekir. Fakat bu çözümlere geçmeden önce klasik ilişkisel veritabanı çözümlerinin hepsini denemek gerekir. Örneğin sorgu hızlandırmak için indeksleme işlemi kullanılabilir. İndeksleme işlemi, sorguda arama kriterinde kullanılan özelliklerin indekslenmesidir. MySQL, PostgreSQL, Oracle, MS SQL Server gibi birçok popüler veritabanı yönetim sistemi farklı indeksleme türlerine destek vermektedir [4]. Diğer bir hızlandırma çözümü ise OLAP çözümleridir. OLAP çözümlerinde belirli özellikler seçilir ve bu

özelliklerden çok boyutlu bir veri modeli oluşturulur. Çok boyutlu veri modelindeki özellikler üzerine karmaşık analitik sorgu cevapları çok hızlı bir şekilde elde edilebilir [5]. İndeksleme basit sorguları hızlandırmak için kullanılırken OLAP karmaşık analitik sorguları hızlandırmak için kullanılır. Her iki yöntemin de en büyük dezavantajı ek alana ihtiyaç duymasıdır. Aralarındaki iki temel fark vardır. Birinci fark, indeksleme her kayıt ekleme işlemi sırasında kendini günceller ancak OLAP sadece veriler girildikten sonra ek bir yazılımla kendini günceller. İkinci fark ise, OLAP yazılım araçlarına ihtiyaç duyarken indeksleme gelişmiş veritabanı yönetim sistemleri için standart haline gelmiştir. Bu çalışmada, indekslemenin avantajları ve dezavantajları incelenecektir.

Denormalizasyon işlemi büyük veri kavramında da karşımıza çıkar. Lee ve Zheng [17] ile Ho ve ark. [18] ilişkisel veri modelinden NoSQL modele geçişte denormalizasyon işleminin nasıl yapılması gerektiğinin üzerinde durmuşlardır. Ordonez ve ark. gelişmiş E-R modellerinde veri madenciliği için veri seti yapmada denormalizasyon işleminin önemi üzerinde durmuşlardır [19]. Bu çalışmada denormalizasyon getirilerinin farklı ilişkisel veritabanı sistemlerinde etkileri incelenecek ve CAP teoremindeki denormalizasyon işleminin yeri belirlenmeye çalışılacaktır.

3. Problem ve Çözümler

Problem, Fakültemiz Dekanlığı tarafından istenen bir web tabanlı anket uygulamasının test aşamasında ortaya çıkmıştır. İlk olarak uygulamada, klasik veritabanı tasarım süreçlerinde olduğu gibi normalizasyon işlemi yapılmış ve SQL fonksiyonları ile genel sonuçların öğretim üyesine gösterilmesi amaçlanmıştır. Ancak, test aşamasında öğretim üyesinin anket sonuçlarına geç cevap alabileceği problemi ile karşılaşılmıştır. Bu noktada, öğretim üyesinin sonuçları daha hızlı alabilmesi için denormalizasyon işlemine gidilmiş ve performans iyileştirilmeye çalışılmıştır. Bu bölüm yapılan hem normalizasyon hem de denormalizasyon işlemlerini kapsamaktadır.

3.1. Klasik çözüm

Yapılan görüşmeler sonunda normalize edilmemiş veri sözlüğü:

Öğrenci_No, Öğrenci_Adı, Öğrenci_Soyadı, Ders_Kodu, Ders_Adi, Öğretim_Üyesi_No, Öğretim_Üyesi_Adi, Öğretim_Üyesi_Soyadi, Bölüm_No, Bölüm_Adi, Dönem_No, Dönem_Adi, Soru_No, Soru, Cevap

şeklindedir.

Fakültemiz, her öğrencinin kendi dersleri için anket doldurmasını istemiştir. Ancak, anket dolduran öğrencinin hangi anketi doldurduğunun bilgisinin tutulmaması istenmiştir. Diğer taraftan, anketi bir kez dolduran öğrencinin o ders için ikinci bir anket dolduramaması istenmiştir. 20 civarında soru olacağı ve her bölümün ortak, ortak olmayan soruları olabileceği söylenmiştir. Bu bilgilerin ışığı altında normalizasyon işlemi sonrası

Öğrenci(Öğrenci_No, Adı, Soyadı, …)
Bölüm(Bölüm_No, Bölüm Adı, …)
Öğretim_Üyesi(Öğretim_Üyesi_No, Adı, Soyadı, *Bölüm_No*, …)
Dönem(Dönem_No, Dönem Adı)
Ders(Ders_ID, DersKodu, Ders_Adı, *Bölüm_No*, Dönem_No, *Öğretim_Üyesi_No* …)
Anket_Kontrol(Öğrenci_No, Ders_ID, Dönem_No)
Soru(Soru_No, Soru, Bölüm_No)
Anket_Cevap (Ders_ID, Donem_No, Soru_No, Cevap)

tablolarına ulaşılmıştır. Birincil anahtarlar altı çizili, ikincil anahtarlar italik bir şekilde verilmiştir. Öğrenci, Bölüm, Öğretim_Üyesi ve Dönem tabloları sadece birincil anahtar ve açıklayıcı özellikler içermektedir. Ders tablosundaki Ders_ID her dönem güncellenen bir alandır. Bu veri üniversitenin öğrenci bilgi sisteminden elde edilmektedir. Şu anda bir dersi bir öğretim üyesi verdiği için Öğretim_Üyesi_No aynı tabloda tutulmuştur. Anket_Kontrol sayesinde belirli bir döneme ait ders için belirli bir öğrencinin anket doldurup doldurmadığı bilgisi tutulmaktadır. Bu bilgi sayesinde öğrenci aynı ders ve dönem için ikinci bir anket gönderememektedir. Soru tablosunda Sorular ve bölüm bilgisi tutulmaktadır. Bu tablodan belirli bir bölüme ait bilgiler Bölüm_No üzerinden kolayca çekilir. Doldurulan anket sonuçları Anket_Cevap tablosunda tutulmaktadır. Burada bir dersin bir dönemine ait anketteki sorulara karşı verilen cevapları tutulmaktadır. Bu tabloda kesinlikle öğrenciye ait bir bilgi tutulmamaktadır. Her anket cevabı için bir satırlık kayıt ekleme işlemi yapılır. Eğer anket 20 sorudan oluşuyorsa bir derse ait anket için bir öğrencinin 20 satırlık verisi tutulur. Anket_Cevap tablosu kullanılarak bir dönemdeki bir ders için SQL fonksiyonları ile sorular gruplanarak Cevap özelliği için veriler elde edilebilir. Sonraki bölümlerde (bölüm 3.1.1 ve 3.1.2) bu işlem anlatılmıştır.

3.1.1. Anket kayıt ekleme

Öğrenci web uygulaması üzerinden anketi doldurduktan sonra eğer tüm soruları işaretlediyse anket bilgileri sunucu tarafına JSON (JavaScript Object Notation) formatında gönderir. JSON formatında gelen anket cevapları ayrıştırılır ve veritabanına ekleme işlemi yapacak SQL hazırlanır. Örneğin:

```
INSERT INTO Anket_Cevap (Ders_ID, Donem_No, Soru_No, Cevap) VALUES (2, 1, 1, 4), (2, 1, 2, 3), (2, 1, 3, 5), …
```

Burada Ders_ID, Donem_No bilgileri form üzerinden alınırken Soru_No ve o soruya verilen Cevap JSON üzerinden elde edilir. VALUES bölümünde her soru için bir satır eklendiği görülmektedir. Soru sayısı kadar değer SQL ifadesinde görüldüğü gibi topluca veritabanına eklenir. Daha hızlı ekleme için toplu ekleme tercih edilmelidir. Örneğin 20 anket sorusu cevabı için 20 INSERT sorgusu çağırmak yerine VALUES bölümüne 20 kayıt girişi yapmak performans açısından önemli bir kazanç sağlar.

Daha önceden belirlenen bir tarihe ulaşıldığında sisteme kayıt girişi bitirilir. Kayıt işlemi bittikten sonra belirli bir tarihte öğretim üyesinin sonuçları görmesine imkân verilir. Bu nokta da SQL fonksiyonları ile eklenmiş kayıtlar üzerinden öğretim üyesi ve derse ait değerlendirmeler elde edilebilir.

3.1.2. Anket değerlendirme

Eklenmiş kayıtlar üzerinden öğretim üyesinin belirli bir dönemdeki dersine ait kayıtlar temel bir WHERE kalıbı ile kolayca elde edilebilir. SQL fonksiyonları ile de bu kayıtlar işlenip öğretim üyesine anketin genel sonuçları gösterilebilir.

```
SELECT s.Soru, AVG(Cevap), MIN(Cevap), MAX(Cevap) , STD(Cevap) FROM Anket_Cevap a
INNER JOIN Soru s ON a.Soru_No = s.Soru_No
WHERE a.Ders_ID=@X AND a.Donem_No=@Y
GROUP BY a.Soru_No
```

Bir öğretim üyesi dersine tıkladığında şifrelenmiş istek sunucu tarafına ulaşır. Şifrelenmiş metin içinden Ders_ID ve Donem_No bilgileri deşifre edilip @X ve @Y parametreleri doldurulur. Şifreleme sayesinde başka öğretim üyelerinin farklı derslere ulaşmamaları sağlanmıştır. Ayrıca, bu kısımda COOKIE (çerez) içinde öğretim üyesine ait dersler şifreli olarak taşınıp dersin öğretim üyesine ait olup olmadığı kontrol edilir. Bu iki kontrol sayesinde güvenlik sağlanmıştır. JOIN işlemi ile soru metinlerine ulaşılmıştır. Bir dönemdeki ders için aynı soru numarasına ait kayıtlar GROUP BY ile gruplanmış ve AVG, MIN, MAX ve STD ile bu grupların sırasıyla cevapların ortalaması, minimum değeri, maksimum değeri ve standart sapması hesaplanmıştır. Standart sapma hesabı veritabanı yönetim sistemine göre değişiklikler gösterebilir. Bu çalışmada deney kısmında kullanılan standart sapma fonksiyonları MySQL için STD iken PostgreSQL ve Oracle için STDDEV'dir.

Ders_ID ve Donem_No üzerinden yapılan arama işlemi kayıt sayısı arttıkça yavaşlamaya başlayabilir. Bu durumda düşünülebilecek en temel çözüm indekslemedir. Çalışmamızda, bu iki alana ait indeksleme yapılmış indeksli olmayan ve indeksli durumun avantaj ve dezavantajları dördüncü bölümde incelenmiştir. İndekslemede çok farklı algoritmalar kullanılabilir. Bu çalışmada, her veritabanı yönetim sisteminin en temel indeksleme işlemi kullanılmıştır.

3.2. Denormalizasyon tabanlı çözüm

Anket sayısı arttıkça sorgu sürelerinin yavaşlaması ve sonuçta anket değerlendirme işleminin yavaşlaması kaçınılmazdır. Denormalizasyon işlemi ile bu problem aşılabilir ancak denormalizasyon işlemi normalizasyon işleminde olduğu gibi kesin bir çözüme sahip değildir. Denormalizasyon işleminin kesin bir çözümü olmadığı için problemdeki performans ihtiyacına bağlı olarak birden fazla çözüm üretilebilir. Bu çalışmada fakültemizde iki yıldır kullanılan denormalizasyon tabanlı çözümümüz anlatılacaktır.

Klasik çözümde her anket cevabı Anket_Cevap tablosunda tutulmakta ve tablo üzerinden SQL fonksiyonları ile çıkarımlar yapılmaktadır. Anket_Cevap tablosu bir anket için yüzlerce satır kayıt içerebilir. Örneğin bir dersin bir dönemine ait 200 öğrencinin 20 sorudan oluşan bir anketi doldurması halinde 4000 satırlık bir veri eklenir. Bunun da bir bölümdeki 40 ders için yapıldığını düşünecek olursak 160.000 satırlık bir kayıt içinde arama yapılması gerekmektedir. Diğer taraftan 50 bölüm için bu veritabanını kullanırsak 8.000.000 kayıtlık veri oluşur. Ek olarak her dönem bu veritabanının kullanılacağını düşünürsek bu kadar büyük bir veri içinde arama işleminin yavaşlaması kaçınılmazdır. Geliştirilen denormalizasyon çözümünde bir dersin bir dönemine ait kayıtların tek satırda toplanması amaçlanmıştır. Başka bir deyişle örnekteki 4000 satırlık veri yerine tek satırlık veri ile arama problemine çözüm aranmıştır. Anket_Cevap tablosu yerine Anket_Cevap_DeNor adında üç özellikten oluşan tablo kullanılmıştır.

Anket_Cevap_DeNor (<u>Ders_ID</u>, <u>Donem_No</u>, Veri)

Burada Veri üzerinde birden fazla öğrencinin anket sonuç verisinin saklanması amaçlanmıştır. Öğrenci kayıt girişi yaptıkça bu satır güncellenecek ve öğretim üyesi bu tablo üzerinden bir sorgulama ile kayıtlara ulaşabilecektir. Veri özelliği uygulamaya özgü bir biçimde tasarlanmıştır.

```
Soru_1=4-20,2-28,5-18,3-18,1-16&Soru_2=1-22,5-19,4-21,2-22,3-16&Soru_3=4-20,3-19,2-17,1-14,5-30&Soru_4=2-23,1-22,5-19,4-15,3-21&…
```

Veri özelliğinde Soru_1, Soru_2, … , Soru_N şeklinde her sorunun kendine ait bir alanı vardır ve bu alanlar & işareti ile birbirinden ayrılmıştır. Her soru içinde CevapNo – cevap işaretleme sayısı bilgisi alınmış ve farklı cevaplar virgül ile birbirinden ayrılmıştır.

Kaba Kod 1. Sunucu tarafında veri güncelleme işlemi

01 İstemci tarafından Gelen şifreli veriden Ders_ID ve Donem_No kayıtlarını deşifre ederek ve ayrıştırma yaparak bul.
02 İstemci tarafından JSON formatında gelen Anket cevaplarını ayrıştır ve diziye kaydet. (Dizi_Soru ve Dizi_Cevap)
03 Ders_ID ve Donem_No'ya ait kayıt olup olmadığını sorgula
04 Eğer kayıt yoksa Dizi_Soru ve Dizi_Cevap dizilerini düzenleme ve INSERT işlemi ile Anket_Cevap_DeNor tablosuna ekle.
05 Eğer kayıt var ise Veri özelliğini ayrıştır ve Hashtable'a ekle. Hashtable'ın anahtar bölümünde Soru_No (Soru_1, Soru_2, vs.) ve değer bölümünde sorulara verilen cevaplar (4-20,2-28,5-18,3-18,1-16) olsun. Dizi_Soru ve Dizi_Cevap dizilerindeki Soru numarasını bul ve uygun cevabı 1 arttır. Tüm sorular kontrol edildikten sonra UPDATE ile Anket_Cevap_DeNor tablosunu güncelle.

Örneğin Soru_1 için 20 öğrenci 4. şıkkı, 28 öğrenci 2. şıkkı, 18 öğrenci 5. şıkkı, 18 öğrenci 3. şıkkı ve 16 öğrenci 1. şıkkı seçmiştir. Normalize bir tabloda yüzlerce satırda tutulan bu verilerin, tasarlanan bu yapı sayesinde sadece bir satırda tutulması sağlanmıştır.

3.2.1. Veri özelliğin güncellenmesi

Web uygulaması üzerinden anket doldurulduktan sonra şifreli Ders_ID ve Donem_No bilgileriyle birlikte anket cevaplarını içeren veriler JSON formatında sunucuya gönderilir. (Kaba Kod 1)

Soru_1=3&Soru_2=4&Soru_3=2&Soru_4=3&…

Burada, soru numarası ve soruya verilen işaretlemenin cevap numarası ayrıştırılır ve dizi şeklinde kaydedilir. Ders_ID ve Donem_No bilgileri üzerinden ankete ait bir kayıt olup olmadığı bir SQL sorgusu ile sorgulanır. Eğer kayıt yok ise Veri özelliğine uygun olarak bir düzenleme işlemi yapılır ve veritabanına kaydedilir. Örneğin bir önceki gelen anket sonucuna göre veri özelliği

Soru_1=3-1&Soru_2=4-1&Soru_3=2-1&Soru_4=3-1&…

şeklinde olur. Aslında mevcut gelen sonucun cevaplarına sadece "-1" şeklinde bir ekleme yapılmıştır. Anket_Cevap_DeNor tablosunda kayıt var ise güncelleme işlemi yapılır. Örneğin başlangıçta

Soru_1=4-20,2-28,5-18,3-18,1-16&Soru_2=1-22,5-19,4-21,2-22,3-16&Soru_3=4-20,3-19,2-17,1-14,5-30&Soru_4=2-23,1-22,5-19,4-15,3-21&…

şeklinde iken Dizi_Soru ve Dizi_Cevap verilerine göre güncelleme yapılır. Örneğin Soru_1=3 & Soru_2=4 gibi bir veri için soru 1'in 3 değeri ve soru 2'nin 4 değeri 1 arttırma şeklinde tüm sorular için arttırma yapılır. Sonuç olarak veri

Soru_1=4-20,2-28,5-18,**3-19**,1-16&Soru_2=1-22,5-19,**4-22**,2-22,3-16&Soru_3=4-20,3-19,**2-18**,1-14,5-30&Soru_4=2-23,1-22,5-19,4-15,**3-22**&…

şeklinde yeni bir durum kazanır ve bu yeni durumun güncellemesi yapılır. Burada, soru aramasını hızlandırmak için Hashtable kullanılmıştır. Hashtable olmaması durumunda dizi şeklinde bir arama işlemi için O(N)'lik bir maliyet söz konusu iken HashTable sayesinde O(1) gibi çok hızlı bir arama imkânına kavuşulmuştur. Hashtable veri yapısı anahtar ve değer kısımlarından oluşur. Anahtar kısmı arama işleminde kullanılırken değer kısmında anahtara ait veri tutulur. Burada, Soru Numaraları anahtar olarak kullanılırken Veri Cevap Sonuçları değer olarak tutulmuştur. Öğretim üyesi değerlendirme sonuçlarını görmek istediğinde minimum değer, maksimum değer, ortalama değer ve standart sapma gibi değerler bu veri üzerinden üretilecektir.

3.2.2. Değerlendirme işlemi

Öğretim üyesi değerlendirme işlemine tıkladığında şifreli olarak gelen veri sunucu tarafında ayrıştırılır ve Ders_ID / Dönem_No bilgileri elde edilir. Anket_Cevap_DeNor tablosundan belirli bir dönemdeki ders için arama işlemi yapılır. Veri özelliği ayrıştırılır ve tüm sorular için minimum değer, maksimum değer, ortalama ve standart sapma değerleri hesaplatılır. Örneğin Soru_1 için minimum değer ve maksimum değer aşağıdaki gibi kolayca bulur.

Soru_1=4-20,2-28,5-18,3-19,1-16
Minimum: 1, Maksimum: 5

Ortalama değer ve standart sapma için aşağıdaki formüller kullanılır.

$$Ort = \frac{\sum_{i=0}^{n} c_i * cs_i}{\sum_{i=0}^{n} cs_i} \quad (1)$$

$$Std.Sapma = \sqrt{\frac{\sum_{i=0}^{n} (c_i - ort)^2 * cs_i}{(n-1) * \sum_{i=0}^{n} cs_i}} \quad (2)$$

Bu formüllerde, c ile cevap, cs ile cevap sayısı ve ort ile ortalama ifade edilmiştir. Soru_1 için ortalama ve standart sapma sırasıyla 2.96 ve 0.675 şeklindedir. Normalize tabloda SQL fonksiyonları ile yapılan bu işlemler burada kodlama yaparak hesaplanmak zorunda kalınmıştır. Ancak, yüzlerce satırı işleme yerine bu yöntem sayesinde sadece bir satırın işlenmesi uygulamamıza performans getirisi

Tablo 1. Veritabanı Yönetim Sistemi Yazılım Versiyonları

PostgreSQL 9.6.1 on x86_64-pc-mingw64 64-bit
MYSQL 5.7.17 64-bit
Oracle Database 11g Express Edition 11.2.0.2.0 64-bit

Tablo 2. .Net Veritabanı Bağlantı Kütüphaneleri

PostgreSQL	Npgsql - . Net Data Provider for PostgreSQL – 3.1.9.0
MySQL	MySql.Data Connector 6.9.9.0
Oracle	Oracle Data Provider for .Net - 4.112.2.0

sağlayacaktır. Bu getiri bir sonraki bölümde incelenecektir.

4. Performans Testleri

Anket uygulamasında klasik çözüm ve denormalizasyon tabanlı çözüm için bir veri seti hazırlanmıştır. Bu veri setinde 32 bölüm, her bölüme ait 30 ders, her derse ait 100 öğrenci ve her anket için 20 soru olduğu varsayılmıştır. Anket cevapları 1 ile 5 arasında rasgele bir seçim şeklindedir. Klasik çözümde indekslemenin performans kazancını anlamak için indeksleme yapılmamış durum ile karşılaştırılmıştır. Performans testleri MySQL, PostgreSQL ve Oracle olmak üzere üç farklı veritabanı yönetim sistemi üzerinde yapılmıştır (Tablo 1). Test kodları .Net Framework 4.5 üzerinde C# dili ile yazılmıştır. Veritabanı yönetim sistemlerine bağlanmak için Tablo 2'deki bağlantı kütüphaneleri kullanılmıştır. Yapılan testlerde kütüphanelerin yaratılma süreleri ve ilk yüklenme süreleri dikkate alınmamış, kütüphaneler yüklendikten sonraki süreler dikkate alınmıştır. Ayrıca, her işlem için 3 kez tekrar yapılıp bu işlemlerin ortalamaları alınmıştır.

Her üç veritabanı yönetim sistemi yazılımı ve test yazılımları aynı bilgisayar üzerine kurulmuştur. Bu test bilgisayarı Windows 10 Pro 64 bit işletim sistemine sahip Intel Core i5-6500 3.20 Ghz işlemcili 8 GB Bellekli ve 128 GB SSD diske sahiptir. Zaman ölçüm işlemlerinde performans ölçümleri için .NET sınıfı bulunan ve önerilen Stopwatch kütüphanesi kullanılmıştır.

4.1. Anket sonuçlarını kaydetme

Web uygulamamızda anket açıldıktan sonra öğrenciler kendi dersleri için anketleri doldurmaya başlayabilir. Anket kaydetme işleminde 20 soru için 20 cevap rastgele üretilmiş ve MySQL, PostgreSQL ve Oracle olmak üzere üç farklı veritabanı yönetim sisteminin her biri için üç farklı tabloya kaydetme işlemi yapılmıştır. Üç farklı tablodan: birincisine sadece normalizasyon işlemi uygulanmış (T1), ikincisinde normalizasyon işlemine ek olarak arama yapılan alanlarda indeksleme işlemi yapılmış (T2) ve üçüncüsü denormalizasyon tabanlı çözüm sonucunda elde edilmiştir (T3). Bu sayede farklı veritabanı yönetim sistemlerindeki farklı özellikteki tablolara

Tablo 3. Tablolara sonuçları kaydetme ortalama süreleri ve standart sapmaları

	T1		T2		T3	
Veritabanı	ms	Std	ms	Std	ms	Std
MySQL	1.945	0.538	2.080	0.582	2.667	0.504
PostgreSQL	0.666	0.062	0.852	0.080	1.134	0.096
Oracle	2.836	0.177	3.148	0.193	1.955	0.171

Tablo 4. Tablolardan sonuçları elde etme ortalama süreleri ve standart sapmaları

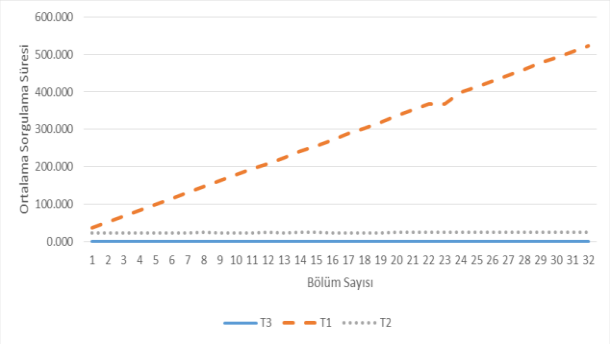
	T1		T2		T3	
Veritabanı	ms	Std	ms	Std	ms	Std
MySQL	280.3	144.7	25.17	0.893	0.674	0.109
PostgreSQL	74.3	40.1	4.459	0.443	0.530	0.071
Oracle	71.4	34.3	7.790	0.346	1.195	0.275

öğrenciler tarafından doldurulan anket verilerinin kaç ms'de kaydedildiği araştırılmıştır.

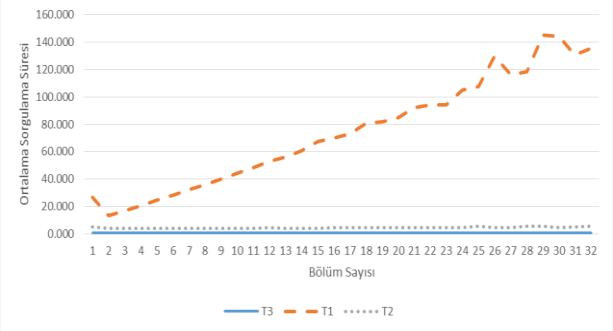
Üç veritabanından en hızlı kayıt ekleme sonuçlarının PostgreSQL üzerinden alındığı görülmektedir. Bu noktada üç veritabanı içinde geçerli ayarlar kullanılmış ve herhangi bir performans arttırıcı ek ayar yapılmamıştır. Ayrıca, testler .Net (C#) ve bir bağlantı kütüphanesi üzerinde yazıldığı için bu durumda performansı etkilemiş olabilir. Bu sebepten her veritabanı yönetim sistemi kendi içinde değerlendirilmiştir. İndeksleme işleminin üç veritabanı yönetim sisteminde de eklemeyi yavaşlattığı görülmektedir. Bunun sebebi, indekslemenin ek dosya oluşturması ve her kayıt ekleme işleminde bu dosyayı da düzenlemesidir. Denormalizasyon çözümünde bir dönem ve ders için bir arama işlemi yapılmakta ve kayıt yoksa ekleme var ise de güncelleme işlemi yapılmakta idi (Kaba kod 1.). Başka bir deyişle, bir SELECT sorgusu ile arama yapılmakta ve INSERT/UPDATE sorguları ile de kayıt düzenlenmektedir. Hatta INSERT bir dönemdeki ders için bir kez yapılmakta UPDATE ise ilk INSERT işleminden sonra bu dönemdeki ders için hep yapılmaktadır. Örneğin, 100 öğrencinin bir dönemdeki ders için ankete katıldığını düşünürsek ilk öğrencinin kaydı insert ile alınırken diğer öğrencilerin kaydı UPDATE ile alınmaktadır. Birden fazla sorgu çalışacağı için bu durumun daha yavaş çalışacağı düşünülmektedir ki MySQL ve PostgreSQL'de durum böyledir. Ancak, Oracle'ın güncelleme işlemini çok kısa sürede yaptığı ve bu sebeple denormalizasyon işleminin beklenildiği gibi yavaşlamadığı görülmektedir.

4.2. Anket sonuçlarını sorgulama

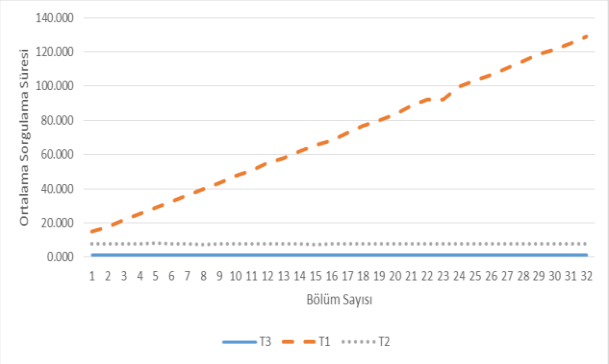
Web uygulamamızda belirli bir tarihte öğrencilerin anket doldurabilmesi durdurulur. Öğretim üyesi belli bir tarihte sonuçları görmeye başlar. Bu bölümde öğretim üyesinin 32 bölümlü, her bölümün 30 dersi olan ve 100 öğrencinin 20 soru için kayıt doldurduğu bir sistemde sonuçların kaç ms'de hazırlandığı araştırılmıştır (Tablo 4). Ayrıca, veri miktarı arttıkça



Şekil 1. MySQL sorgulama sonuçları



Şekil 2. PostgreSQL sorgulama sonuçları



Şekil 3. Oracle sorgulama sonuçları

klasik çözümün sorunları ve denormalizasyon çözümünün getirileri incelenmiştir.

Sadece normalizasyon işlemi uygulanmış bir tabloda en hızlı cevap Oracle veritabanı yönetim sistemi olmuştur. Normalize tabloya indeksleme uygulandığında PostgreSQL'in 4.449 ms gibi kısa bir sürede cevap verdiği görülmüştür. Ayrıca, üç veritabanı yönetim sisteminin de beklenildiği gibi indeksleme sayesinde sorgu sonuçlarının elde edilmesinin hızlandığı görülmektedir. Denormalizasyon çözümünden elde edilen tabloyu sorgulama işleminde her üç veritabanının da çok iyi sonuçlar verdiği görülmüştür. Denormalizayon işlemini indekslenmemiş normalizasyon işlemi ile kıyasladığımızda MySQL, PostgreSQL ve Oracle sonuçlarında sırasıyla yaklaşık 416, 150, 60 katlık iyileşmeler olmuştur. Denormalizayon işlemini indekslenmiş normalizasyon işlemi ile kıyasladığımızda MySQL, PostgreSQL ve Oracle sonuçlarında sırasıyla yaklaşık 37, 8, 7 katlık iyileşmeler olmuştur. Sonuç olarak iyi yapılmış bir

Tablo 5. Tabloların kapladığı alan (MB)

	T1		T2		T3	
Veritabanı	Dos.	İnd.	Dos.	İnd.	Dos.	İnd.
MySQL	83.6	-	88.6	115.8	1.5	-
PostgreSQL	110	41	110	247	1.64	0.055
Oracle	30	17	30	95	0.438	0.063

denormalizasyon çözümünün sorgu performansını çok fazla arttırabileceği görülmektedir.

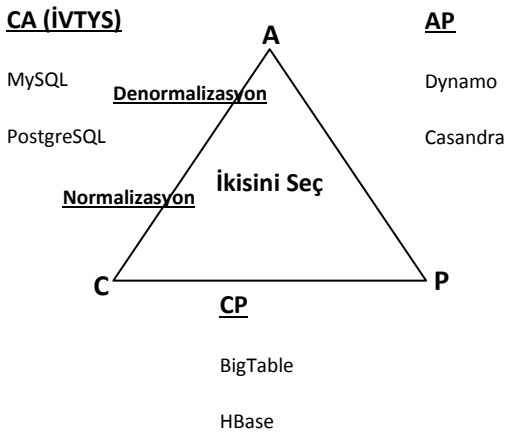
Veri miktarının artması sonuçların elde edilmesini yavaşlatan bir faktördür. Şekil 1, Şekil 2 ve Şekil 3 MySQL, PostgreSQL ve Oracle için veri miktarı artışından üç tablonun nasıl etkilendiğini göstermektedir.

Şekil 1, Şekil 2 ve Şekil 3'te bir bölüme ait 30 ders 100 öğrenci anketi ve her ankette 20 soru sisteme eklendikten sonra 30 ders sonuçları için yapılan sorgulamaların ortalama süreleri alınmıştır. Bu sonuçlara göre sadece normalizasyon işlemi uygulanmış (T1) tablonun kayıt sayısının artmasından etkilendiği görülmektedir. MySQL'de birinci bölüme yapılan anketten sonra yapılan sorgulama 38.066 ms sürerken 32 bölümün eklenmesinden sonra sorgulama süresi 522.779 ms'e kadar çıkmıştır. PostgreSQL 26.847 ms'den 135.532 ms'ye ve Oracle da ise 14.932 ms'den 128.912 ms'ye kadar artışların olduğu görülmektedir. İndeksleme yapılmış tabloda (T2) sorgulamanın hızlandığı ve veri eklendikçe sorgulama süresinin çok değişmediği görülmektedir. MySQL'de 24.550 ms'den 25.501 ms'ye, PostgreSQL'de 4.911 ms'den 5.328 ms'ye ve Oracle'da 7.691 ms'den 7.894 ms'ye bir artış söz konusudur. Görüldüğü gibi indeksleme işlemi anket sonuçlarının kaydedilmesinde ufak bir zaman kaybı oluştursa da sorgulama işleminde sorgulama performansı açısından getirisi büyüktür. Denormalizasyon çözümünden elde edilen tablonun (T3) ise tüm veritabanı yönetim sistemlerinde en iyi sorgulama sonucunu verdiği görülmektedir. Sorgulama sonuçları 1 ms civarındadır.

4.2. Tabloların dosya boyutları

İndeksleme işlemi her ne kadar sorgulama işlemini hızlandırsa da ek depolama maliyeti ortaya çıkarır. Bu bölümde her durum için oluşan dosya boyutlarına yer verilmiştir (Tablo 5).

Birincil indeks MySQL'de direkt tablo içinde tutulmakta ve ayrıca bir indeks olarak belirtilmemektedir. Diğer taraftan PostgreSQL ve Oracle birincil indeksi ayrıca vermektedir. Bu sebepten sonuçlar Dosya Boyutu (Dos.) ve İndeksi (İnd.) şeklinde verilmiştir. Tablo 5'teki sonuçlara göre en küçük boyuta sahip dosya Oracle veritabanı yönetim sistemi olmuştur. Normalizasyon işlemi oluşturulan tabloya indekslerin eklenmesi (T2) ile dosya boyutlarının çok arttığı görülmektedir. MySQL'de indekslenmemiş normalize durum (T1) 83.6 MB alana sahip iken indeks uygulandığında (T2)



Şekil 4. CAP teoremi

2.44 katlık bir artış ile 204.4 MB alana ulaşmıştır. PostgreSQL ve Oracle sırasıyla 2.17 ve 2.66 katlık indekslemeden kaynaklı dosya boyutu artışı söz konusudur. Görüldüğü gibi indeksleme işlemi bu problem için yaklaşık 2.42'lik bir dosya boyutu artışına sebebiyet vermiştir. İndeksleme her ne kadar sorgulama sürecini hızlandırsa da gereksiz indekslemenin depolama maliyetini arttıracağı unutulmamalıdır.

Denormalizasyon işlemi sonucu elde edilen tabloların boyutları ise MySQL, PostgreSQL ve Oracle için sırasıyla 1.5, 1.695 ve 0.5 MB'tır. Tüm veri için bir satır tutmak yerine (T1 ve T2'de olduğu gibi) verileri tek satırda (T3'te olduğu gibi) depolamanın maliyeti düşürdüğü görülmektedir. Ancak, anket uygulamasındaki isteklere bağlı bir tasarım yapıldığı unutulmamalıdır. Örneğin, hangi öğrencinin hangi anketi doldurduğuna ve hangi soruya ne cevap verdiğine dair bilginin tutulmaması istenmiştir. Eğer istenseydi daha farklı bir denormalizasyon çözümü gerekecekti ve bu dosya maliyetini arttırabilirdi.

4.3. Büyük veri ve CAP teoremi

Anket uygulamasının testini yaparken 32 bölümün 30 dersinin, derse ait 100 öğrencinin ve her bölüm ait ankette de 20 sorunun olduğu varsayılmıştır. Testlerde, en fazla 1,920,000 satırlık bir veri oluşturulmuştur. İleride uygulamamızın ülkemizde hatta tüm dünya üzerinde kullanılmaya başlandığını düşündüğümüzde çok daha büyük bir veri ile karşılaşılacaktır. Bu durumda NoSQL gibi veritabanı düşünülecek başka bir deyişle dağıtık mimari çözümleri düşünülecektir. Bu noktada CAP teoremi üzerinden bir çözüm düşünülebilir (Şekil 4).

CAP teoremi, dağıtık bir sistemde tutarlılık (Consistency), ulaşılabilirlik (Availability) ve bölünebilme toleransı (Partition tolerance) koşullarından sadece ikisine sahip olunabileceğini belirler. CA kısmı (tutarlılık - erişilebilirlik) ilişkisel veritabanı yönetim sistemlerini temsil ederken diğer kısımlar NoSQL veritabanlarını temsil eder. Günümüzün en büyük problemlerinden biri ne zaman NoSQL çözümlere geçileceğidir. Yaptığımız çalışma göstermiştir ki CA kısmında C veya A tarafına yaklaşan çözümler üretilebilir. Eğer normalizasyon işlemini ön plana çıkarırsak C kısmına yaklaşırken denormalizasyon çözümlerinde ise A kısmına yaklaşıldığı görülmektedir. Bir performans sorunu karşısında çözüm olarak problemi birden fazla bilgisayara bölme yerine denormalizasyon çözümü ile tek sunucuda da bir çözüme erişilebileceği unutulmamalıdır.

5. Tartışma ve Sonuç

Günümüzde büyük veri kavramı ile birlikte veriyi birden fazla bilgisayara dağıtmak performans sağlayıcı bir çözüm olmuştur. Ancak, yaptığımız çalışma göstermiştir ki iyi bir veritabanı tasarımı ile veriyi birden fazla bilgisayara dağıtmadan klasik bir ilişkisel veritabanı yönetim sistemi üzerinde de hızlı bir sistem tasarlamak mümkündür. Bu noktada, problem iyi şekilde ortaya konulmalı ve klasik çözümlerin dışına çıkıldığında oluşacak getiriler ve götürüler iyi incelenmelidir. Anket için ürettiğimiz çözümde anket kaydetme kısmında çok az bir yavaşlama olurken anket sonuçlarını işlemede büyük bir performans kazancı elde edilmiştir.

Denormalizasyon işlemi NoSQL veritabanı çözümlerine geçilmeden önce başvurulması gereken bir konu niteliğindedir. Denormalizasyon işleminin kesin bir çözümü olmaması, daha fazla kodlama ve uzmanlık gerektirmesi açısından bir sorun gibi gözükse de ilişkisel veritabanı tasarımında performans artırımında önemli rol oynar. Yaptığımız çalışmada dağıtık sistemin performans getirileri yerine tek bilgisayarda hızlı çözümlere ulaşmanın mümkün olduğu gösterilmiştir. Bu çalışma iyi bir tasarım sayesinde depolama maliyetlerinin de azaltılabildiğini göstermektedir.

İleriki çalışmalarımızda, NoSQL tabanlı çözümlerin getirileri ve götürüleri incelenip hangi durumda hangi veritabanı çözümünün kullanılması gerektiği araştırılacaktır. Ayrıca, iyi veritabanı tasarımlarının NoSQL tarafındaki etkileri de incelenecektir.

Teşekkür

Geliştirilen Anket Uygulaması Akıllı Ders Yönetim Sistemine (ADYS - adys.nku.edu.tr) entegre edilmiş ve Namık Kemal Üniversitesi Çorlu Mühendislik Fakültesi tarafından 2 yıldır kullanılmaktadır. ADYS, 18.12.2013 tarihinden bu yana NKUBAP.00.17.AR.13.15 protokol nolu 'Akıllı Ders Yönetim Sistemi ile Programlama Ödevleri için İntihal Tespiti Uygulaması' başlıklı projemiz kapsamında NKU-BAP tarafından desteklenmiştir. Tüm desteklerinden dolayı üniversitemize teşekkür ederiz. Çalışmadaki yazılım kodlarına ve veritabanı tablo yaratma SQL'lerine aşağıdaki web adresinden ulaşılabilir:

http://bilgmuh.nku.edu.tr/erdincuzun/post/denormalizasyon-bir-anket-uygulamasi

Kaynakça

[1] Codd, E. F. 1970. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 13 (6): 377–387.

[2] Chen, P. 1976. The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems 1 (1): 9–36.

[3] Codd, E.F. 1971. Further Normalization of the Data Base Relational Model. IBM Research Report RJ909.

[4] Harrington, Jan L. 2009. Relational Database Design and Implementation: Clearly Explained. Elsevier-Morgan Kaufmann, Chapter 6, pp. 105-126.

[5] Powell, G. 2006. Chapter 8: Building Fast-Performing Database Models. Beginning Database Design ISBN 978-0-7645-7490-0, Wrox Publishing.

[6] Mailvaganam, H. 2007. Introduction to OLAP - Slice, Dice and Drill!. Data Warehousing Review - DWreview.com. (Erişim Tarihi: 08.05.2017)

[7] Finkelstein, S., Schkolnick, M. ve Tiberio, P. 1988. Physical Database Design For Relational Databases. ACM Transactions on Database Systems, vol. 13, no. 1, pp. 91-128.

[8] Cerpa, N. 1995. Pre-physical data base design heuristics. Information Management, vol. 28, no. 6, pp. 351-359.

[9] Hanus, M. 1994. To normalize or denormalize, that is the question. Proceedings of 19th International Conference for the Management and Performance Evaluation of Enterprise Computing Systems, San Diego, CA, 1994, pp. 416– 423.

[10] Rodgers, U. 1989. Denormalization: why, what, and how?. Database Programming & Design, (12) 46–53.

[11] Coleman, G. 1989. Normalizing not only way. Computerworld,(12) 63– 64.

[12] Nizam, A. "Veritabanı Tasarımı İlişkisel Veri Modeli ve Uygulamaları". Bölüm 9: Denormalizasyon, Papatya Yayıncılık, 159 – 178.

[13] NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable. http://nosql-database.org (Erişim Tarihi: 08.05.2017)

[14] Leavitt, N. 2010. Will NoSQL Databases Live Up to Their Promise?. IEEE Computer, 43 (2), 12-14.

[15] Eric A. B. 2000. Towards robust distributed systems. In Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing (PODC '00). ACM, New York, NY, USA.

[16] Seth, G. ve Nancy, L. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, 33 (2), 51–59.

[17] Lee, C.-H. ve Zheng, Y.L. 2015. SQL-to-NoSQL Schema Denormalization and Migration: A Study on Content Management Systems". IEEE International Conference on SMC, pp. 2022 – 2026.

[18] Ho, L.-Y., Hsieh, M.-J., Wu, J.-J. ve Liu, P. 2015. Data Partition Optimization for Column-Family NoSQL databases. IEEE International Conference on Smart City/SocialCom/SustainCom together with DataCom 2015, pp. 668-675.

[19] Ordonez, C., Maabout, S., Matusevich, D. S. ve Cabrera, W. 2014. Extending ER models to capture database transformations to build data sets for data mining. in Data and Knowledge Engineering, 89, 38-54.