DENORMALİZASYON İLE SORGU İŞLEME SÜRESİNİ İYİLEŞTİRME: AKADEMİK VERİ YÖNETİM SİSTEMİ UYGULAMASI

FATMA DEMET DURAK

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi
Danışman: Doç. Dr. Erdinç UZUN
2022

T.C.
TEKİRDAĞ NAMIK KEMAL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DENORMALİZASYON İLE SORGU İŞLEME SÜRESİNİ İYİLEŞTİRME: AKADEMİK VERİ YÖNETİM SİSTEMİ UYGULAMASI

FATMA DEMET DURAK

ORCID: 0000-0001-7807-1958

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
YÜKSEK LİSANS TEZİ
Danışman: Doç. Dr. Erdinç UZUN

HAZİRAN-2022

ÖZET

DENORMALİZASYON İLE SORGU İŞLEME SÜRESİNİ İYİLEŞTİRME: AKADEMİK VERİ YÖNETİM SİSTEMİ UYGULAMASI

Fatma Demet DURAK

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Danışman: Doç. Dr. Erdinç UZUN

Günümüz teknoloji çağında yazılım performansı kullanıcılar için büyük önem taşımaktadır. Veri tabanı tasarımı yazılım performansını iyileştirmek için yapılacak en etkili yöntemlerden biridir. Genellikle Veri tabanı tasarımında kullanılan normalizasyon işlemi veri sayısı arttıkça performans kaybına neden olur. Bu sorunu ortadan kaldırmak amacı ile denormalizasyon devreye girer. Bu çalışmada, akademik verilerin tutulduğu bir veri tabanında sorgu süresini iyileştirmek amaçlı çalışmalar yapılmıştır. Çalışmada Normalizasyon, İndeksleme, NoSQL ve Denormalizasyon olmak üzere dört farklı veri tabanı tasarım yöntemi incelenmiştir.

Anahtar Kelimeler: Veri Tabanı Tasarımı, Normalizasyon, İndeksleme, No SQL, MongoDB, Denormalizasyon.

ABSTRACT

IMPROVING QUERY PROCESSING TIME WITH DENORMALIZATION: ACADEMIC DATA MANAGEMENT SYSTEM APPLICATION

Fatma Demet DURAK

Department of Computer Engineering

MSc. Thesis

Supervisor: Assoc. Prof. Erdinç UZUN

In today's technology age, software performance is of great importance for users. Database design is one of the most effective ways to improve software performance. Normalization process, which is generally used in database design, causes performance loss as the number of data increases. In order to eliminate this problem, denormalization comes into play. In this study, studies were carried out to improve the query time in a database where academic data is kept. In the study, four different database design methods, namely Normalization, Indexing, NoSQL and Denormalization, were examined.

Keywords: Database Design, Normalization, Indexing, No SQL, MongoDB, Denormalization.

İÇİNDEKİLER

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

PROGRAM KODLARI DİZİNİ

KISALTMALAR DİZİNİ

1NF	First Normal Form
2NF	Second Normal Form
3NF	Third Normal From
BCNF	Boyce-Codd Normal Form
BRIN	Block Range Index
B-Tree	Balanced Search Tree
CAP	Consistency, Availability, Partition Tolerance
GB	Gigabyte
GIST	Generalized Search Tree
KB	Kilobyte
MB	Megabyte
NoSQL	Not Only SQL
PHP	Hypertext Preprocessor
RDMS	Relational Database Management System
SP-GIST	Space Partitioned Generalized Search Tree
UML	Unified Modelling Language

TEŞEKKÜR

Yüksek lisans eğitim sürem boyunca bana danışmanlık ederek gösterdiği büyük emek, sabır ve desteklerinden dolayı saygıdeğer danışmanım Doç. Dr. Erdinç UZUN'a sonsuz teşekkürlerimi sunarım.

Eğitim hayatım boyunca bana her zaman destek olan aileme; bana karşı her zaman sabırlı ve anlayışlı davranan sevgili babam İlhan DURAK'a, her an kendime güvenmemi sağlayan annem Nursen DURAK'a, ömrüm boyunca her anımda yanımda olan ağabeyim Mehmet DURAK'a sonsuz teşekkür ederim.

Tez yazım sürecimde varlığı ile bana her zaman güç veren Şenol AKYAZI'ya sonsuz teşekkürler.

Fatma Demet DURAK

Yazılım Mühendisi

1. GİRİŞ

Teknoloji her geçen gün gelişmekte ve bununla birlikte kayıt altında tutulan veri sayısı hızla artmaktadır. Veri sayısı arttıkça yazılımlardaki sorgu süreleri uzamaya başlamakta ve kullanılan sistemlerde yavaşlamalar meydana gelmektedir. Özellikle bu durum, web uygulamalarında sayfanın geç yüklenmesine ve hatta sunucu istem süresinin aşılarak sayfanın yüklememesine sebebiyet verebilmektedir. Bu tez, akademik değerlendirme sistemlerinde sorgu sürelerinin farklı tasarımlarda ve ortamlarda araştırılması üzerinedir.

Bir yazılım uygulaması verileri okumak, işlemek ve yazmak için zaman harcar. Günümüzde web uygulamaların artması ve kullanıcının veri üretme sürecine dahil olmasıyla veri üzerine yapılan işlemlerin sayısı artmıştır. Örneğin, Facebook'ta bir yorum yaptığınızda bunun veri tabanına yazılması ve diğer kişilerin bunu görmesi başlı başına bir problemdir. Verimli bir web uygulaması geliştirmede veri okuma, işleme ve yazma işlemlerini dikkate alarak verilerin düzgün şekilde düzenlenmesi ve saklanması önemlidir. Bu tür bir uygulamada, sistem kaynaklarının iyi kullanılması ve kullanıcı isteklerinin istenilen sürede cevaplanması gerekir.

Web uygulaması geliştiricilerinin İlişkisel Veri tabanı Yönetim Sistemleri (Relational Database Management Systems – RDMS) ve NoSQL veri tabanı sistemleri olmak üzere iki ana veri deposundan birini veya ikisini de tercih ederler. Bu veri depolarına ait birçok uygulama geliştirilmiştir. Örneğin, RDMS tarafında PostgreSQL, MySQL, Oracle, Microsoft SQL Server ve benzeri birçok çözüm vardır. Özellikle açık kaynak olan PostgreSQL ve MySQL gibi ilişkisel veri tabanları birçok web uygulamasında kullanılmaktadır. Ancak, ilişkisel veri tabanlarını herhangi bir sayıda makineye ölçeklemek zordur ve genellikle manuel süreçler içerir [1]–[3]. NoSQL veri tabanı sistemleri [4], RDMS'e göre mükemmel ölçeklenebilirlik ve esneklik sağlar. Ancak, karmaşık sorgular ve veri tutarlılığı konusunda bazı desteklerden yoksundurlar [5]. Bu iki veri depolama yönteminin kendilerine ait avantajları olsa da probleme göre iki yöntemin dışında ara yöntemler geliştirmek gerekebilir. Bu tez, RDMS kuralları dışına çıkıp sorgu sürelerini iyileştirmeyi amaçlar.

Bir yazılımda veri üzerine yapılan işlemler için geliştirme araçları ve çözümleri ortaya çıkmıştır. Bu sayede, yazılım geliştirme süreleri azaltılmıştır. Başka bir deyişle programcının verimliliği arttırmayı amaçlayan araçlar sayesinde geliştirme süreleri önemli derecede hızlanmıştır. Örneğin veri tabanı kütüphaneleri, UML (Unified Modelling Language) [6] ve

ORM (Object Relational Mapping) [7] gibi araçlar günümüzde programcıların işini kolaylaştırmaktadır. Bu araçların yanı sıra gelişmiş veri tabanları da karmaşık sorgular ve güçlü veri tutarlılığı gibi üst düzey veri tabanı işlevleri sayesinde uyguma üretkenliğini büyük ölçüde arttırır. Bu iyileşmelere rağmen bir web uygulamasının yanıt süreleri uzayabilir. Bu durumda, problem ve veri sayısı dikkate alınarak veri tasarımı ve düzenlemesi performans iyileştirilmesi açısından tekrar ele alınmalıdır.

Bir web uygulamasında kısa yanıt sürelerini korumak çok önemlidir. Örneğin, bir e-Ticaret sitesinde kullanıcının sistemde çok beklemesi farklı e-Ticaret sitelerine yönelmesine sebebiyet verir. İnsan-bilgisayar etkileşimi üzerine yapılan çalışmalarda, bir kullanıcının düşünce akışının kesintisiz kalması için bir saniyenin sınır olarak kabul edildiği zorlu hedefler önerilmektedir [8], [9]. Bir web uygulamasında yanıt süresi hem uygulama kodunu yürütme hem de veri tabanı tasarımını kapsamaktadır. Bu noktada, yanıt süresini iyileştirme için kaynakların arttırılması çözümlerden biri olsa da maliyet yaratabilir. Bu sebepten uygulama kodu ve veri tabanı tasarımı üzerine iyileşmeler düşünülebilir. Bu tez, veri tabanı tasarımı tarafına odaklamaktadır.

Web uygulamalarındaki kullanıcı sayısının artması ile hem uygulama hem de veri tabanı tasarımları ve kaynaklar tekrar ele alınmalı ve tekrar düzenlenmelidir. Günümüzde, bulut sistemlerinin ortaya çıkması ile ölçeklenebilirlik, hız, maliyet, performans ve güvenirlik gibi kavramlar tekrar ele alınmıştır. Web uygulamalarında son zamanlarda birçok yenilik ortaya çıkmıştır.

- NoSQL veri tabanları [10]: belirli veri modelleri destekleyen birçok farklı NoSQL çözümü üretilmiştir. NoSQL veri tabanları yazılım geliştirme kolaylığı yanı sıra performans katkıları ile çok fazla web uygulama çözümünde kullanılmaktadır.

- Konteyner (Container) [11]: Sanallaştırma için işletim sistemi yerine yalnızca uygulamayı, kütüphanelerini ve bağımlılıklarını içerir. Konteyner, hızlı ve taşınabilir olması sebebiyle yazılım dünyası tarafından kabul görmüştür.

- Kubernetes [12]: konteynerlerin hızlı şekilde devreye alınması, ölçeklenmesi ve izlenmesini otomatikleştiren açık kaynak çözümdür. Özellikle bir web uygulamasında kullanıcı sayısı arttırdığında bu yapı sayesinde konteyner sayısı

arttırılabilir ve kullanıcı sayısı azaldığında konteyner sayısı azaltılabilir. Bir anlamda kaynaklar çok daha verimli kullanılmış olur.

Bu yeniliklere rağmen hem yazılım hem de veri tabanı tarafında yapılacak birçok iyileşme vardır. Bu tez, veri tabanı tarafında yapılabilecek iyileştirmeler üzerinedir. Bu tezde, üniversitemizde ve diğer üniversitelerde akademik değerlendirme sistemleri üzerinde farklı veri tabanı tasarım teknikleri kullanıp sorgu süreleri incelenecektir. Web ortamında tasarlanacak test ortamında PHP dili, PostgreSQL (İlişkisel veri tabanı) ve MongoDB (NoSQL sistemi) kullanılacaktır.

Bu bölümün devamındaki alt bölümlerde literatür özeti ve çalışmanın kapsamı sunulacaktır. Bölüm 2, giriş bölümünde tanıtılan kavramlar hakkında daha geniş bilgi sunar. Bölüm 3, problemimiz için farklı veri tabanı tasarımı çözümlerini kapsar. Bölüm 4, deney ortamının kurulması ve farklı tasarımlardan elde edilen sonuçların incelenmesine ayrılmıştır. Son bölüm, çalışmanın katkıları ve sonuçları verildikten sonra gelecek çalışmaları kısaca tanıtır.

1.1 Literatür Özeti

Veri tabanı kavramı, 1960'ların ortalarında yaygın olarak kullanılan manyetik disklerin ortaya çıkması ile depolama ihtiyaçları karşılamak için ortaya çıkmıştır. Veri depolama konusunda verinin nasıl bir hiyerarşide tutulması gerektiğini 1970 yılında Edgar F. Codd adında IBM'de çalışan bilgisayar bilimci tarafından önerilmiştir. İlişkisel model (Relational model) [13], olarak adlandırılan bu öneri her bir varlık türü için tablo kümelerine dayanır. Edgar Normalizasyon (Normalization) olarak ta bilinen bu konunun ilk tanıtımını IBM raporlarında sunmuştur [14]. Bu raporda, birincil normal form (1NF) ve 1971 yılında ikinci ve üçüncü (2NF ve 3NF) tanıtılmıştır. Sonraki yıllarda, 4NF, BCNF ve 5NF gibi normalizasyon formları da önerilmiştir [15]. İlişkisel modelin geniş çapta yayılmasına neden olan veri tabanı uygulamaların ortaya çıkması 1980'in ortalarını buldu. 1990 sonrası tüm büyük ölçekli veri tutma ve işleme uygulamalarına ilişkisel model egemen oldu ve günümüzde halen bu model hakimiyetini sürdürmektedir. İlişkisel veri tabanı yönetim sistemleri (RDMS) veya Veri tabanı yönetim sistemleri (Database Management Systems – DBMS) olarak adlandırılan bu uygulamaların IBM Db2, Oracle, Microsoft SQL Server gibi ticari uygulamalarının yanı sıra MySQL ve PostgreSQL gibi açık kaynak kodlu uygulamaları da vardır. Tezimizde, günümüzün popüler DBMS'lerinden biri olan PostgreSQL seçilmiştir.

Veri tabanı Tasarımı (Database Design), verilerin bir veri tabanı modeline göre organize edilmesidir. Hoffer ve ark. [16] ve Sanders'in [17] belirttiği gibi kavramsal, mantıksal ve fiziksel tasarımın birleşimidir. Veri tabanı tasarımcısı, hangi verilerin saklanması gerektiğini ve veri öğelerinin birbiriyle nasıl ilişkili olduğunu belirler. Bu verilerle uygun veri modelini belirleyip bir veri tabanı uygulamasında fiziksel tasarıma geçer [18]. Bu tasarım işleminde daha çok ER modeli [19] (Entity-Relationship Model) kullanılır. ER modeli, veri tabanının verimli bir şekilde tasarlanmasına yardımcı olan mantıksal ve görsel bir tasarımdır. Bu modelde çizilen diyagramlar sayesinde problem daha iyi analiz edilip gereksinimler daha hızlı tespit edilebilir. Ayrıca, normalizasyon işlemi [20] veri tabanı tasarımının en önemli aşamalarından biridir. Normalizasyon genel amaçlı sorgulamalara uygun bir yapı elde edilir. Ayrıca, iyi bir normalizasyon sayesinde veri bütünlüğünü bozulmasına yol açabilecek ekleme, güncelleme ve silme anormalliklerinden arınmış bir veri tabanı elde edilmiş olacaktır.

İnternetteki veri miktarının artması ile birlikte bu verinin ölçeklendirilmesi problemi karşımıza çıkmıştır. Bu konuda çalışan araştırmacılar, tablo ilişkilerini kullanmayan ve verinin depolanmasını daha esnek hale getiren mekanizmalar üzerinde çalışmıştır. Bu konuya 21. yüzyılın başlarında "NoSQL" ismi verilmiştir. Aslında, bu tür veri tabanları 1960'ların sonlarından beri var olmuştur, ancak ilişkisel modelin birçok ihtiyacı karşılamasıyla birlikte uzun yıllar sadece bu model üzerine uygulamalar geliştirilmiştir. Web 2.0 ile birlikte ihtiyaçlar değişmeye başlamış ve özellikle ölçeklenebilirlik önem kazanmıştır [20]. NoSQL veri tabanları, büyük veri ve gerçek zamanlı web uygulamalarında RDMS'in yerini alıp giderek daha fazla kullanılmaktadır [21]. Bu yaklaşımın motivasyonu ilişkisel veri tabanlarının "yatay" ölçekleme yani birden fazla bilgisayarda dağıtık olarak kullanılmasının zor olmasıdır [22]. Lee ve Zheng [23] ile Ho ve ark. [24] ilişkisel modelden NoSQL modele geçişte denormalizasyon işleminde dikkat edilmesi gereken konuları açıklamışlardır.

Veri tabanı tasarımda her ne kadar normalizasyon ve NoSQL tarafı gibi iki taraf oluşmuş olsa da ihtiyaçlar doğrultusunda daha farklı tasarımlar yapılabilir. İnternet ortamındaki kayıt sayısının sürekli artar bir durumda olmasından dolayı verinin sorgulanması sorunu ortaya çıkar. Kayıt sayısı arttıkça aranan veri veya verilere ulaşma gibi konularda zaman performansı olumsuz etkilenir. Zaman performansını iyileştirme yöntemlerinden biri de Powell'ın [25] altını çizdiği gibi indekslemedir. Veri tabanında indeksleme, veri sorgulama işlemini hızını arttıran bir yapısıdır. Bu hız elde edilirken indeks yapısını oluşturmak için ek depolama maliyeti oluşmayacağı unutulmamalıdır.

Günümüzde veri miktarının artmasıyla indeksleme işleminin de yetersiz kalacağı durumlar olmuştur. Ayrıca, indeksleme işleminin oluşturacağı ek depolama maliyeti incelenmelidir. Veri miktarı arttıkça performans açısından çıkabilecek olumsuzluklara karşı tasarım tekrar kontrol edilmelidir [26]. Veri miktarının artması sebebiyle normalizasyon kurallarının dışında performans arttıracak çözümler aranmaya başlanmıştır. Bu çözümlerden biri de denormalizasyon (Denormalization) stratejisidir. Bu strateji, performans arttırmak için önceden normalize edilmiş veri tabanını kullanır. Denormalizasyon, bir veri tabanının okuma performansını ve bazı durumlarda yazma performansını kaybetme pahasına, verilerin yedek özet kopyalarını ekleyerek veya verileri gruplayarak performansı iyileştirme stratejisidir [27], [28]. Araştırmacıların farklı denormalizasyon türlerini önerirken buluştuğu ortak nokta bu işlemin dikkatli bir şekilde yapılması gerektiğidir [29], [30]. Performans arttırmak amacı ile denormalizasyon yapılırken veri miktarı, hangi veri tabanı yönetim sisteminin kullanıldığı, işletim sistemi ve donanım dikkate alınması gereken unsurlardır. Denormalizasyon stratejisini daha sistematik hale getirmeye çalışan çalışmalar [31], [32] olsa da denormalizasyon yavaşlayan sorgulara karşı bulunan kesin kuralları olmayan çözümlerdir [33], [34]. İhtiyaç duyulduğunda ilişkisel veri tabanlarına nasıl denormalizasyon yapılacağı dair öneriler vardır [35], [36].

Uzun ve ark. [38], denormalizasyon işlemi sayesinde bir anket uygulamasında sorgu performansını iyileştirmişlerdir. Sorgu süresi 38 kat hız kazanırken verileri depolanması konusunda da 130 kat iyileşme sağlamışlardır. Diğer çalışmalarında [38], MySQL, PostgreSQL ve Oracle olmak üzere üç farklı veri tabanında denormalizasyonun etkisini incelemişlerdir. Bunun yanı sıra indeksleme ve NoSQL'e ne zaman geçilmesi gerektiğini CAP teorimi üzerinden anlatmışlardır. Taşyürek [39], [40], mekânsal verilerin sıklıkla güncellendiği coğrafi bilgi sistemlerinde arama işleminde denormalizasyon kullanıp performans iyileşmesi elde etmiştir.

1.2 Çalışmanın Amacı ve Kapsamı

Üniversitemizin, Akademik Değerlendirme Sistemi'nde (https://aves.nku.edu.tr/) "İstatistik" bölümü tüm yayınlardan istatiksel bilgileri sunar. Örneğin "birimlerin yıllara göre yaptığı yayınların istatistikleri" için tüm verinin değerlendirilmesi ve özet bilginin oluşturulması gerekir. Bu bilginin oluşturulması klasik yöntemler ile zaman alıcı bir süreçtir. Bu çalışmada klasik süreçler ve güncel süreçler denenip bir denormalizasyon çözümü önerilecektir.

Yazılım performanslarının hızını etkileyen en önemli etken veri tabanı tasarımıdır. Edgar F. Codd tarafından 1970'li yıllarda sunulan "Normalizasyon" bir ilişkisel veri tabanı tasarım teorisidir. Normalizasyon, tabloları ve sütunları ayrıştırmayı amaçlar. Veri sayısı arttıkça sorgu süreleri yavaşlamasın diye "İndeksleme" önerilir. İndeksleme disk erişimini en aza indirerek veri tabanını optimize etmenin en temel yoludur. Veri sayısının daha da artması ile birlikte "Denormalizasyon" teknikleri ile performans arttırılması amaçlanmıştır. Bu teknikler, probleme ve sorgulara göre tasarımı getirir. Ayrıca, son zamanlarda veri sayısının daha çok artması ile birlikte NoSQL veri tabanı sistemleri popüler olmuştur. NoSQL, verileri dağıtık sistemlerde tutmak için geliştirilmiştir. Bu tezde, bu tasarım ve çözümlerin her biri ayrı ayrı incelenecektir. Bu çalışmanın katkıları:

- Web uygulamalarında veri tabanı tasarımının yazılım performansı üzerinde etkilerinin incelenmesi
- Normalizasyon ve indeksleme çözümlerinin araştırılması
- NoSQL veri tabanlarından MongoDB üzerine bir sistem oluşturulup çözümün incelenmesi
- Denormalizasyon çözümü üretilip çözümün artı ve eksilerinin incelenmesi şeklinde özetlenebilir.

2. KAVRAMLAR

Bu bölüm, tezimizdeki problemin çözüm yolları hakkında temel kavramlara ayrılmıştır. Alt bölümlerde normalizasyon, veri tabanlarında indeksleme, NoSQL veri tabanları ve MongoDB anlatıldıktan sonra denormalizasyon kavramı ile bölüm sonlandırılacaktır.

2.1 Normalizasyon

Normalize edilmemiş bir veri tabanında veri sayısı arttıkça veri tekrarı artmaya başlar ve bu durum depolama maliyetlerini arttırır. Ayrıca, bu tekrarlayan veride bir düzeltme işlemi yapıldığında tüm satırların güncellenmesi çok zaman alıcı bir işlem olacaktır. Normalizasyon, bu zaman alıcı süreci iyileştirmek için veri tabanı tablolarındaki gereksiz veri tekrarlarını ortadan kaldırmak ve tekrarlayan veri sayısını azaltmak için yapılan tasarım işlemleridir.

Normalizasyon yapılırken uyulması gereken kurallar bütününe normal form adı verilir. İlk olarak birincil normal form (1NF) 1970 yıllarda Edgar F. Codd tarafından önerilmiştir. 1971 yılında ise yine Edgar F. Codd tarafından ikinci ve üçüncü normal formlar (2NF ve 3NF) tanıtılmıştır [14]. 1974 yılında ise Boyce-Codd Normal Form (BCNF) Raymond F. Boyce ve Edgar F. Codd tarafından tanıtılmıştır. Zamanla ortaya çıkan dördüncü normal form (4NF) ve beşinci normal form (5NF) çok tercih edilmez. Yaygın olarak 1NF, 2NF ve 3NF kullanılmaktadır. Gereksiz veri performansı engelledikçe ve veri tutarlılığı arttıkça normal form seviyesi artar.

2.1.1 Birincil Normal Form

Bir veri tabanının birinci normal form düzeyinde olması için gerekli 3 kural vardır.

- Tabloda tekrarlayan satırlar bulunmaması.
- Her hücrede yalnızca bir değer ile ilgili bilgi bulunması.
- Her satırın eşsiz bir anahtar ile tanımlanması.

İlişkisel veri tabanı yönetim sistemlerinde veri tabanında bir tablo oluşturulurken her satırın kendine ait bir değeri olur. Bu değere birincil anahtar (primary key) denir. Bu değerin eşsiz bir veri olması ve boş (Null) olmaması gerekir.

2.1.2 İkinci Normal Form

Bir veri tabanının 2NF seviyesinde olması için belirli kurallar vardır.

- Tablo 1NF seviyesinde olmak zorundadır.
- Tabloda bir birincil anahtar olmalı ve diğer sütunlar birincil anahtara bağlı olmalıdır.

2.1.3 Üçüncü Normal Form

Üçüncü Normal Form kuralları ise;

- Tablo 2NF seviyesinde olmak zorundadır.
- Anahtar olmayan sütunlar anahtar olan sütunlara tam bağımlı olmalı ancak anahtar olmayan sütunlarla da bir bağı olmamalıdır.

Bu üç durumu tamamlamış bir veri tabanı tasarım süreci sonunda gereksiz veri tekrarı olmayan ve veri bütünlüğü sağlanmış bir veri tabanı elde edilmiş olur. Veri tekrarı azaldığı için kullanılan depolama alanından da tasarruf sağlanmıştır. Ayrıca, güncelleme ve silmede ortaya çıkabilecek problemler normalizasyon sayesinde büyük ölçüde azaltılmış olur.

2.2 İndeksleme

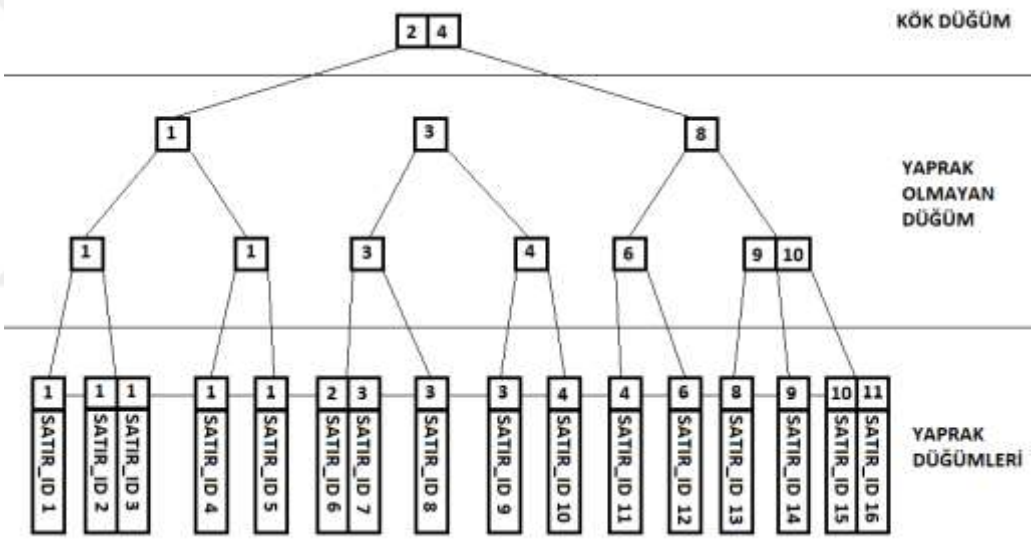
İndekslenmemiş veri tabanı tablosunda veya tablolarında bir veri aranırken tüm tablo gözden geçirilmek zorunda kalabilir. Tablo Tarama (Table Scan) olarak ifade edilen bu durum sorguyu oldukça yavaşlatır. İndeksler veri tabanı tablolarının gerekli alan veya alanları üzerinde tanımlanmış veriye az işlemle daha hızlı ulaşmayı sağlayan nesnelerdir. İndekslenmiş bir tabloda, tablonun tamamının okunmasındansa oluşturulan indeks anahtarı (index key) sayesinde daha az veri okunarak sorgu sonucuna daha kısa sürede ulaşılabilir.

İndeksleme konusunda birçok faklı indeksleme yöntemi literatürde önerilmiştir [25]. Tezimizde PostgreSQL veri tabanı kullanılacaktır. Bu veri tabanında ön plana çıkan indeks türleri:

- B-tree indeks
- Hash indeks
- GiST indeks

- SP-GiST indeks
- BRIN indeks

B-Tree İndeks: En yaygın olarak kullanılan indeks türüdür. Bilgisayar biliminde, bir B-tree, sıralanmış verileri koruyan ve O(log N) zamanda aramalara, sıralı erişime, eklemelere ve silmelere izin veren kendi kendini dengeleyen (self-balancing) bir ağaç veri yapısıdır. B-tree, ikili arama ağacını genişleterek ikiden fazla çocuk düğümü oluşturulması izin verir. B-Tree, veri tabanları ve dosya sistemleri gibi nispeten büyük veri bloklarına uygun bir veri yapısı olduğu için sıklıkla kullanılmaktadır [41]. Birçok veri tabanında olduğu gibi PostgreSQL'de de indeks türü belirtilmediğinde geçerli indeks türü olarak kabul edilir. Şekil 2.1, PostgreSQL B-Tree indeksinin veri yapısını anlamak için bir B-Tree indeksinin basitleştirilmiş bir versiyonunu göstermektedir.



Şekil 2.1 B-Tree algoritması

Şekil 2.1'de görülebileceği gibi, ağacın altındaki bağlantılı liste (linked list), anahtar (key) olarak bilinen gerçek indeks değerini ve ayrıca tablo satırına referans veren SATIR_ID'yi saklar. Anahtar, indeks yapısı içindeki tablo satırını bulmak için kullanılırken, SATIR_ID, tablo satırını indeks dışında bulmak için kullanılır. Liste öğelerine yaprak düğümleri (leaf nodes) denir. Bağlantılı listenin üstünde, ağacın ilk seviyesi başlar. Dengeli arama ağacı, giriş anahtarlarının ve alt işaretçilerin her birinde depolar. Yaprak düğümlerden belirli SATIR_ID'lere erişmek için, bu ağaç kökten başlayarak yaprak düğüme kadar geçilir, yol üzerindeki anahtarlar karşılaştırılarak yol seçilir. Her ağaç düğümüne yaprak olmayan

düğüm veya dahili düğüm denir. B-Tree algoritmasının temel amacı verileri ağaç yapısında tutup, sorgulama süresini hızlandırmaktır.

Hash indeks: Eşitlik sorgulamak için etkilidir.

Generalized Search Tree (GIST): en yakın komşu (nearest-neighbor) veya desen eşleştirme (pattern matching) gibi aramalar için yararlı olan daha karmaşık bir indeks yapısıdır. GiST indeksi tek bir indeks türü değil, içinde birçok farklı indeks oluşturma stratejisinin uygulanabileceği bir altyapıdır.

Space Partitioned GiST (SP-GiST): GiST ile benzer şekilde, bu indeks uygulaması, quadtrees, k-d trees ve radix trees gibi alanla bölümlenmiş ağaçları destekler.

Block Range Index (BRIN): her tablo bloğu aralığı için özet bilgileri depolar. GiST ve SP-GiST gibi, BRIN de birçok farklı indeksleme stratejisini destekler. Ayrıca, bir BRIN indeksinin kullanılabileceği belirli operatörler indeksleme stratejisine bağlı olarak değişir.

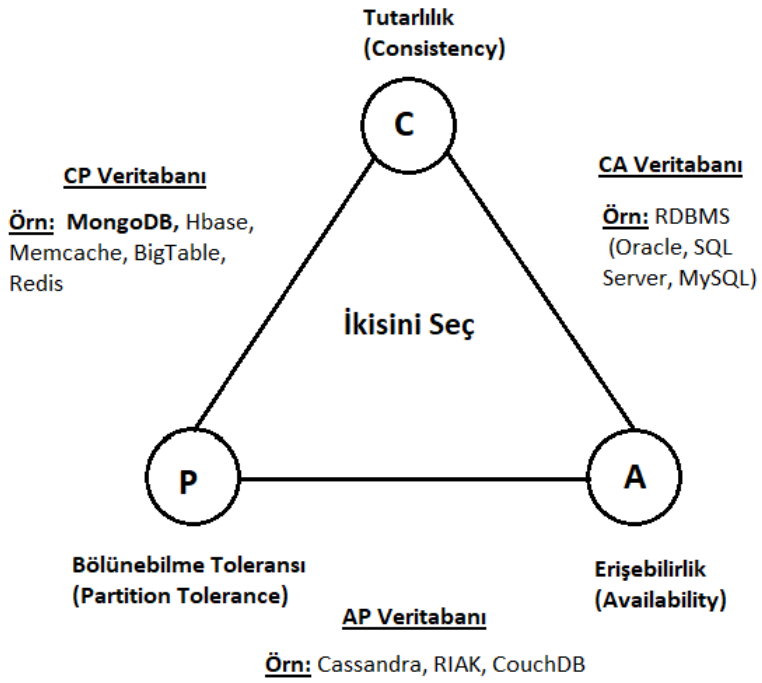
Deneyler bölümünde PostgreSQL tarafından geçerli değer olarak tercih edilen ve literatürde sıklıkla kullanılan B-Tree ile indeksleme yapılacaktır.

2.3 NoSQL

Yıllardır kullanılan Oracle, MySQL, MsSQL ve PostgreSQL gibi ilişkisel veri tabanları yerine NoSQL veri tabanları birçok farklı çözümde karşımıza çıkmaya başlamıştır. İlişkisel veri tabanlarının en temel özelliği ACID (atomicity, consistency, isolation, durability) kurallarıdır [42]. Bu kurallar, veri tabanı sistemini hatalara, elektrik kesintilerine, ortaya çıkabilecek diğer aksiliklere rağmen veri geçerliliğini garanti etmeye amaçlar. NoSQL veri tabanları ise ACID özelliğini dikkate almaz. NoSQL veri tabanı belirli kurallara sahip faklı modeller içerir ve bu kurallara göre özellikleri arasında farklılıklar vardır. Bu farklıkları açıklamak için en uygun teorem CAP (Consistency, Availabity, Partition tolerance)'tır. CAP [43], [44], Eric Brewer tarafından önerilmiş dağıtılmış veri tabanlarında üç garantiden yalnızca ikisinin sağlanabileceğini belirtir.

- Tutarlılık (Consistency): Her okuma en güncel yazılan veriyi alır veya hata verir.
- Erişebilirlik (Availability): Son yazılan değeri garanti etmeden her istek hatasız bir veri gönderir.

- Bölümleme toleransı (Partition tolerance): Ağda birden fazla bilgisayarda bulunan veri gecikmeler veya düşmeler olmasına rağmen sistem çalışmaya devam eder.



Şekil 2.2 CAP Teoremi

Dağıtılmış veri tabanlarında network ağında gecikmeler veya kopmalar olması, olasılığı yüksek olan bir durumdur. Bu sebeple CAP teoreminin üç özelliği bir arada bulunmaz olarak kabul edilir. Şekil 2.2 incelendiğinde iki özelliğin seçilmesi gerektiği görülmektedir.

CP Veri Tabanı: Tutarlılık ve bölünme toleransını sağlayan CP veri tabanında, iki düğüm arasında bölünme gerçekleştiğinde işlem iptal edilir. Verilerini BSON belgeleri olarak saklayan MongoDB bir CP veri tabanıdır. Ağ bölünmeleri olduğunda erişebilirliği yok sayarak tutarlılığı korur.

CA Veri Tabanı: Tutarlılık ve erişebilirlik sağlayan CA veri tabanında, düğümler arasında bir bölünme olduğunda bölünme toleransı sağlayamaz. Farklı sunucularda bulunan veri, her bir sunucuda aynı değilse sistem hata verir ve veri dönüşü yapmaz.

AP Veri Tabanı: Erişilebilirlik ve bölünme toleransı sağlayan AP veri tabanında, farklı sunucularda bulunan veri, her bir sunucuda farklı değerdedir. Bu durumda sistem tutarlı olamaz.

Örneğin dağıtık mimaride okuma ve yazma işlemi yapmak isteyen bir istemci olduğunu kabul edelim. Birbirine bağlı iki sunucuda var olan bir kayıt için güncelleme işlemi yapılırken, istemci bunu ilk sunucuya iletmiş olsun ancak ilk sunucu ikinci sunucuya iletirken network ağında bir sorun olduğunda güncelleme yalnızca ilk sunucuda gerçekleşir. İlk sunucudan işlem sonucu istemciye doğru olarak iletilir ancak veri başka bir sunucudan çekilmek istendiğinde dönecek olan veri güncel olmayacaktır. Bu durumda sistem yalnızca erişilebilir ve bölümleme toleransına sahip olur. Yani sistem CAP teoreminin yalnızca AP özelliklerini taşır.

Bunun yerine ilk sunucu güncel veriyi ikinci sunucuya iletemediğinde işlemi iptal edip, istemciye işlemin başarısız olduğunu ilettiğinde sistem CAP teoreminin CP özelliklerine sahip olmuş olur. Yani sistem tutarlılık ve bölümleme toleransına sahip olur.

Kullanılan sistemde düğümler arasında iletişim sorunu varken karar verilmesi gereken iki durum oluşur. İşlem iptal edilir ve sistemin tutarlılığı sağlanmış olur ya da düğümlerde farklı veriler olabilme ihtimaline karşı erişilir olması kabul edilir.

Büyük verilerde, NoSQL'in yüksek ölçeklenebilirlik ve yüksek erişilebilirlik özelliklerine sahip olması en büyük avantajıdır. NoSQL veri tabanları 4 farklı türe ayırabiliriz. Bu 4 tür üzerine birçok NoSQL uygulaması ortaya çıkmıştır.

2.3.1 Anahtar Değer Tabanlı

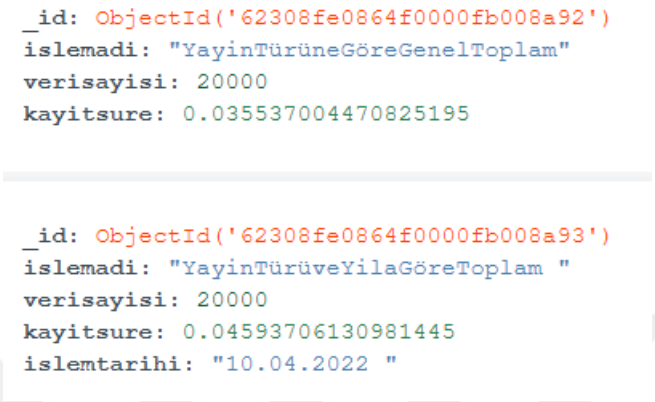
Veri, veri tabanında tutulmak amacı ile kaydedilirken edilirken bir anahtar ve anahtara karşılık gelen bir değer ile kullanılır. Sorgulama yapılacağı zamanda veriye anahtar değeri ile ulaşılır. Aerospike, Berkeley DB, Redis, Dynamo, ZooKeeper gibi veri tabanları anahtar/değer tabanlı kullanım sağlarlar.

2.3.2 Sütun Tabanlı

Veri miktarı çok büyükse sütun tabanlı veri tabanı tercih edilir. Sütun tabanlı veri tabanı büyük hacimli verileri çok hızlı şekilde sorgulayabilir. Anahtar/değer veri tabanlarındaki gibi anahtar bulunur ancak farklı olarak anahtar bir değer yerine sütun gruplarına atanır. Cassandra, Google BigTable, Scylla, HBase gibi veri tabanları sütun tabanlı kullanım sağlarlar.

2.3.3 Belge Tabanlı

Kaydedilen verinin doküman olarak adlandırıldığı belge tabanlı veri tabanlarında genellikle JSON formatında kayıt yapılır. Şekil 2.3'de belge tabanlı NoSQL veri tabanı örneği verilmiştir. MongoDB, CouchDB, BaseX gibi veri tabanları belge tabanlı kullanım sağlarlar.



Şekil 2.3 Belge tabanlı NoSQL veri tabanında saklanan JSON veri örneği

2.3.4 Grafik Tabanlı

Birbirine bağlı verilerin düğüm ve uçları esas alarak karmaşık ilişkileri basite indirger ve verilerin aralarındaki ilişkiyi de tutar. AllegroGraph, MarkLogic, FlockDB, Neo4j gibi veri tabanları grafik tabanlı kullanım sağlarlar.

NoSQL veri tabanları üzerine birçok çözüm çıkmıştır. Seçim yaparken avantajlar ve dezavantajlar incelenmeli ve buna göre en uygun veri tabanı seçilmelidir.

2.3.5 NoSQL Avantajları

- İlişkisel veri tabanlarına göre okuma ve yazma performansı daha hızlıdır.
- İşlem tabanlı olan ilişkisel veri tabanlarının aksine yatay yönde genişlerler ve bu durum kullanıcıya performans kazancı olarak geri döner.
- Sabit tablo tanımlarına gerek yoktur, kullanımı ilişkisel veri tabanlarına göre daha esnektir.
- Genelde açık kaynak kodlu ve ücretsiz oldukları için düşük bir maliyet ile kullanılırlar.

2.3.6 NoSQL Dezavantajları

- İlişkisel veri yapısı yoktur.

- İlişkisel veri tabanlarında yapılmış uygulamaların NoSQL veri tabanlarına taşınması zahmetlidir.
- İşlem (Transaction) tabanlı olmadığı için veri kayıpları olabilir. Bu yüzden belirli muhasebe, bankacılık ve finans sektörlerinde tercih edilmez.
- Veri güvenliği ilişkisel veri tabanları kadar gelişmiş değildir.
- İlişkisel veri tabanları kadar dokümana ve desteğe sahip değildir.

Tezimizde web uygulamaları JSON formatında haberleştiği için belge tabanlı MongoDB deneyler için seçilmiştir.

2.4 MongoDB

Açık kaynaklı kodlu ve bir NoSQL veri tabanı olan MongoDB 2009 yılında C++ programlama dili ile geliştirilmiştir. İlişkisel olmayan yani genel amaçlı bir veri tabanı yönetim sistemi olan MongoDB'nin kullanımı ücretsizdir. MongoDB büyük verilerin olduğu gerçek zamanlı uygulamalarda tercih edilir.

MongoDB veri ekleme açısından incelendiğinde, performans olarak çoğu ilişkisel veri tabanı uygulamalarına göre hızlıdır. Ancak kural ve standartlara bağlı olmaması sebebiyle önemli verilerin olduğu, güvenlik gerektiren sistemlerde kullanılması tercih edilmez.

İlişkisel veri tabanlarında kullanılan table ifadesi MongoDB'de collection olarak kullanılmaktadır. Bunun yanı sıra row yapısı document, column yapısı ise field olarak ifade edilir. Verileri depolamak için ilişkisel veri tabanlarındaki tablo ve satırlar yerine JSON belgeleri kullanılır. Verilerin JSON yapısında saklanmasının en büyük avantajı gelen verilerde belge yapısı değişse bile, veriler kendi kendilerini tanımlayabildiğinden dolayı yeni veri kaydedilirken bir sorun ortaya çıkmaz. Şemasız veri tabanı türünde olan MongoDB'de veriler eklenmeden önce sütun sayısı ya da türü tanımlanmaz. Belge veri modeli esnek olan MongoDB'de bir dokümana yeni bir alan eklenecekse, eklenecek alan sadece ilgili dokümana eklenir ve diğer dokümanlar bu durumdan etkilenmezler.

Belge tabanlı veri tabanı sistemi olan MongoDB'nin yatay ölçeklenebilen bir format olması (scalable) en önemli özelliklerinden biridir. Otomatik ölçeklenen veri tabanı yüksek performans ve kullanılabilirlik sağlar. Çoğaltma (replication) işlemi ile verilerin birden fazla kopyası oluşturulup saklanabildiğinden dolayı yüksek kullanılabilirlik sağlar ve böylece veri kaybı engellenmiş olur. Veri setlerini saklamak için kullanılan dahili bellek ve veriler

üzerinde oluşturulabilen indeksler sayesinde verilere ulaşım hızlıdır. MongoDB diğer programla dilleri ile entegre edilebildiğinden uygulayıcı açısından büyük kolaylık sağlamaktadır.

Tüm bu olumlu özelliklerinin yanı sıra bazı dezavantajlara da sahiptir. Her belge için ayrı bir anahtar adı depolanması gerektiğinden dolayı MongoDB daha fazla bellek alanına ihtiyaç duyar. Bunun yanı sıra ilişkisel veri tabanlarının aksine birleştirme işlemi yapılması gerektiğinde birden fazla sorgu yazmak gerekebilir.

2.5 Denormalizasyon

Veri tabanı tasarımı kavramsal, mantıksal ve fiziksel tasarım aşamalarını içerir. Denormalizasyon işlemleri veri tabanı performansını arttırmak için yapılan çalışmalardır. Normalizasyon işleminin tam tersi olup, kesin kurallara bağlı olmayan denormalizasyon işlemi, veri sorgulama ve okuma hızını arttırmak için kullanılır. İyi bir şekilde tasarlanmış denormalizasyon sorgulama ve okuma söz konusu olduğunda performans kazancı sağlasa da ekleme ve güncelleme işlemlerini yavaşlatmaktadır.

Tabloları normalleştirdiğimizde birden fazla tablo elde ederiz. Böylece bir veriye ulaşmak istediğimizde birleştirme işlemi uygulamak zorunda kalırız. Bu durum veri sorgulama ve okuma hızını yavaşlatır. Normalizasyonun dezavantajı olan bu durumu ortadan kaldırmak için denormalizasyon işlemine başvurulur.

Çalışmamızda yayın verilerinin tutulduğu tabloda yayın adı, yayın türü, akademisyen ID, bölüm ID ve yıl bilgileri bulunmaktadır. Bu verilerden yayın türleri, akademisyen bilgileri veya bölüm bilgileri ile ilgili işlem yapılmak istenildiğinde yayın türleri, bölüm ve akademisyen tabloları ile birleştirme işlemi yapılması gerekmektedir. Az veri miktarlarında olumsuz sonuçlar ortaya çıkmayabilir. Ancak veri miktarı arttıkça işlem sonuçlarına ulaşmak güçleşir hatta bazen sonuç alınamayabilir. Örneğin herhangi bir bölümde, herhangi bir yayın türünden kaç tane yayın olduğunu hesaplamak istediğimizde yayın verilerinin tutulduğu tablo ile bölüm ve yayın türleri tablolarının birleştirilmesi gerekir. Birleştirme işlemini uyguladığımızda 3.000.000 veride bu sonucu almak oldukça fazla zaman alacaktır. Bu olumsuz durumu ortadan kaldırmak için sorgu çeşitlerine göre yeni tablolar oluşturulmuştur. Her yeni veri eklendiğinde, verinin ait olduğu satırda bulunan sayı değeri, tetikleyiciler sayesinde 1 arttırılır. Böylece yapılmak istenen sorgu yeni tablo üzerinden yapılır ve bu

işlemin sonucunda çok hızlı bir şekilde ulaşılır. Tetikleme işlemleri ve tabloların Denormalize edilmeleri ileriki bölümlerde anlatılacaktır.

2.6 Tetikleyiciler (Triggers)

Denormalizasyon yapılırken yeni tablolar kullanılır. Bir ekleme, düzeltme veya silme olduğunda otomatik olarak diğer tablo veya alanlar üzerinde işlem yapmak tetikleyici fonksiyonlar ile gerçekleştirilir.

Tetikleyiciler ilişkisel veri tabanı yönetim sistemlerinde (RDMS) ana tabloda ekleme, silme veya güncelleme işlemlerinden herhangi biri yapılmadan önce veya sonra çalışan, bağlı olduğu diğer tablolar için daha önce kodlanmış işlemleri yerine getiren prosedürlerdir.

3. VERİTABANI TASARIMLARI

Problem, üniversitemizin akademik değerlendirme sistemin geliştirilmesi farklı veri tabanı tasarımlarının performans kazanç ve kayıplarını incelenmesi olarak belirlenmiştir. Bu bölümde, "Kavramlar" bölümünde anlatılan konuların uygulaması anlatılacaktır.

3.1 İlk Tasarım ve Normalizasyon

Bir akademik yayında yayın ismi, yayın türü, yayını yapan akademisyenler, yayın tarihi temel alanlardır. Bu alanlardan normalizasyon işlemi yapıldığında aşağıdaki gibi tablolar ortaya çıkar.

- Yayınlar (Yayın ID, Yayın İsmi, Yayın Tür ID, Tarih)
- Yayın Türleri (Yayın Tür ID, Yayın Türü)
- Akademisyenler (Personel ID, Adı, Soyadı, Bölüm ID)
- Bölüm (Bölüm ID, Bölüm Adı, Fakülte/MYO ID)
- Fakülte/MYO (Fakülte/MYO ID, Fakülte/MYO Adı)
- Akademisyen Yayınları (Personel ID, Yayın ID)

Yayınlar tablosu ile Yayın Türleri, Akademisyenler ile Bölüm ve Bölüm ile Fakülte/MYO arasında bire çok bir ilişki vardır. "ID" uzantılı alanlar birincil (altı çizili) ve yabancı anahtarlarda ilişki kurmak için kullanılmıştır. Diğer taraftan Yayınlar ile Akademisyenler arasında çoğa çok ilişki olduğu için ara bir tabloya (Akademisyen Yayınları) ihtiyaç duyulmuştur.

3.2 PostgreSQL İndeksleme

Bir indeksleme işlemi tablodaki bir veya birden fazla alan üzerine uygulanabilir. Uygulama sırasında hızlandırılmak istenen sorgunun WHERE ve GROUP BY kısımlarındaki alanlar dikkate alınır. Program Kodu 3.1'de GROUP BY örneği verilmiştir.

```
SELECT COUNT(*) FROM "Yayınlar" GROUP BY "Yayın Tür ID",
"Tarih";
```

Program Kodu 3.1 İndeksleme sorgu örneği

Program Kodu 3.2'de verilen sorgu için "Yayın Tür ID" ve "Tarih" alanları üzerine bir iyi indeks oluşturulmasında fayda vardır. Bu indeksi PostgreSQL oluşturmak için

```
CREATE INDEX "indeks ismi" ON Yayınlar ("Yayın Tür ID ",
"Tarih");
```

Program Kodu 3.2 PostgreSQL'de İndeks oluşturma

Program Kodu 3.2'de görüldüğü gibi bir sorgu yazılabilir. Bu şekilde diğer indekslerde oluşturulmuştur.

3.3 MongoDB Çözümü

Web uygulamaları haberleşirken genelde JSON formatını kullanır. İstemci tarafından JSON formatında gelen bir veriyi MongoDB veri tabanı kaydetmek ve sorgulamak kolaydır. Örneğin bir yayın oluşturmak için Program Kodu 3.3'te kullanılması gereken kod gösterilmiştir.

```
for($i=1; $i <= 10000; $i++)
{
        $yayinturu=rand(1, 9);
        $hocaid=rand(1,5000);
        $bolumid=rand(1,2000);
        $yil=rand(2000,2020);
        $doc = array(
                'Yayin_ismi' =>'a',
                'Yayin_Türü'=>$yayinturu,
                'Akademisyen_ID'=>$akademisyenid ,
                'Bölüm_ID'=>$bolumid,
                'Yıl'=>$yil
                        );
$insertOneResult = $yayinlarmongodb->insertOne($doc);
}
```

Program Kodu 3.3 MongoDB veri ekleme kodu

3.4 Denormalizasyon Çözümü

Denormalizasyon, veri tabanı performansını arttırmak için veri tabanı tasarımında normalizasyon kurallarının dışına çıkılmasıdır. Denormalizasyon için yeni tablo ve yeni alanlara oluşturmaya ihtiyaç vardır.

- Yayınlar (Yayın ID, Yayın İsmi, Yayın Tür ID, Personel ID, Tarih, Yıl)

Normalizasyon çözümünde akademisyen yayınları tablosuna başvurmak yani JOIN işlemini gerekir. Bu çözümde birden çok akademisyenin olduğu yayınlar tekrarlamak zorundadır. Normalizasyon kurallarının dışına çıkılmış olur. Normalizasyon kuralları veri tekrarını önlemek için iken burada performansı arttırmak normalizasyon kuralları dışına çıkılmıştır. Diğer taraftan "Tarih" alanının yanına "Yıl" alanı eklenmiştir. "Tarih" alanı içinden "Yıl" verisini çıkarmak için bir fonksiyona ihtiyaç vardır. Program Kodu 3.4'te görüldüğü gibi date_part fonksiyonu üzerinden yıl verisi elde edilebilir.

```
date_part("year", "Tarih")
```

Program Kodu 3.4 Yıl verisi çıkarma

Ancak, tüm satırlarda aynı fonksiyonun çalışması performansı etkiler. Yine normalizasyon kuralları dışına çıkılmıştır. Ulaşılmak istenen veri hem "Tarih" hem "Yıl" olmasına rağmen "Yıl" verisine fonksiyon kullanmadığımız içi daha hızlı şekilde erişebiliriz.

Bu alanlar oluşturulurken SQL sorgularının çok iyi incelenmesi ve bu analizlere göre ek alan veya tabloların oluşturulması gerekir. Bu çalışmada, ileri bölümlerde anlatacağımız SQL sorguları incelenmiş ve aşağıdaki tabloların oluşturulması karar verilmiştir.

- Yayınlar (Yayın ID, Yayın İsmi, Yayın Tür ID, **Personel ID**, Tarih, **Yıl**)
- Yayın Türleri (Yayın Tür ID, Yayın Türü, **Sayı**)
- **Yayın Türleri Yıl (Yayın Tür ID, Yıl, Sayı)**
- **Yayın Türleri Akademisyen (Yayın Tür ID, Personel ID, Sayı)**
- **Yayın Türleri Bölüm (Yayın Tür ID, Bölüm ID, Sayı)**
- **Yayın Türleri Bölüm Yıl (Yayın Tür ID, Bölüm ID, Yıl, Sayı)**

Her ekleme işleminden sonra yukarıdaki tablolar güncellenir. Bu tablolar, uzun sürmesini beklediğimiz sorgulara uygulanır. Örneğin, yıllara göre yayın türlerini gösteren bir sorgu için tüm kayıtlar sorgulanır ve hesaplamalar yapılıp çıktı oluşturulur. Burada kayıt sayısına bağlı olarak sorgu süresi uzayabilir. Bunun için her ekleme işleminden sonra oluşturulan "Yayın Türleri Yıl" tablosu "Yayın Tür ID" ve "Yıl" verileri kontrol edilip "Sayı" verisi güncellenir. Eğer "Yayın Tür ID" ve "Yıl" verileri içeren satır yoksa bu veriler için "Sayı" değeri 1 olarak yeni bir satır açılır.

4. DENEYLER

Bu bölüm tezimizdeki problemler için geliştirilen çözüm yollarının gerçekleştirilme adımlarının anlatılması için ayrılmıştır.

Deneylerde normalizasyon, indeksleme, NoSQL ve denormalizasyon çözümleri tek tek denenmiş ve hepsinin sonuçları hesaplanmıştır. Çözüm yolları arasındaki farkların hesaplanması amacı ile her bir çözüm yolunda yayınlar için oluşturulan tablolara yayın sayısı 10 binden başlayıp her yeni döngüde 10 bin arttırarak 3 milyon veri eklemesi yapılmıştır. Ekleme süreleri ve oluşan veriler için sorgu sürelerinin incelemesi bu bölümde yapılacaktır. Testlerimizde ASUS-N56VZ laptop kullanılmıştır. Kullanılan bilgisayar Intel(R) Core(TM)i7-3630QM CPU @ 2.40GHz, 64 bit işletim sistemi, 8GB RAM özelliklerine sahiptir. Windows 10 Enterprise kullanılmıştır.

4.1 Deney Ortamının Hazırlanması

Ekleme işleminde yayın türü olarak dokuz tür belirlenmiştir. Yıl aralığı olarak ise 2000-2022 yılları arası baz alınmıştır. 5 bin akademisyen ve 2 bin bölüm olduğu varsayılmıştır. Yayın ismine göre sorgulama yapılmayacağından dolayı yayın ismi standart tutulmuştur. 4 farklı çözüm yolu için sonuçlar incelenecektir. Bu çözüm yolları:

- Çözüm 1: Normalizasyon çözümü
- Çözüm 2: İndeksleme çözümü
- Çözüm 3: MongoDB çözümü
- Çözüm 4: Denormalizasyon çözümü

Yapılan çalışmalarda her bir çözüm yolu için ayrı ayrı tablolar oluşturulurken, sorgulanmak istenen veriler aynıdır. Sorgulanmak istenen veriler için belirlenen sorgular,

- Sorgu 1: Yayın Türüne Göre Genel Toplam
- Sorgu 2: Yayın Türü ve Yıla Göre Genel Toplam
- Sorgu 3: Yayın Türü ve Akademisyene Göre Genel Toplam
- Sorgu 4: Yayın Türü ve Bölüme Göre Genel Toplam
- Sorgu 5: Yayın Türü 1 Olan Bölüme ve Yıla Göre Genel Toplam olarak belirlenmiştir.

Sorgu 1'de yayın verilerinin türlerine göre sayılarının hesaplanması, Sorgu 2'de yayın verilerinin tür ve yıllara göre sayılarının hesaplanması, Sorgu 3'te yayın verilerinin yayın türleri ve akademisyenlere göre verilerin hesaplanması, Sorgu 4'te yayın verilerinin yayın türleri ve bölümlere göre sayıların hesaplanması, Sorgu 5'te yayın verilerinden yalnızca yayın türü 1 olan verilerin, bölüm ve yıllara göre sayılarının hesaplanması amaçlanmıştır.

4.2 Ekleme Testleri

Veri kaydetme süreleri ile her bir yöntem için Sorgu 1, Sorgu 2, Sorgu 3, Sorgu 4 ve Sorgu 5 olarak ayrı ayrı süre hesapları yapılmış olup süre kayıt tablolarına eklenmiştir. Sorgu sürelerinin hesaplanması için yapılacak olan her bir sorgu işleminin başında ve sonunda PHP'nin kendi fonksiyonu olan mikrosaniye için doğru verileri döndüren "microtime" fonksiyonu kullanılmıştır. "microtime" fonksiyonun kullanımının kodu Program Kodu 4.1'de belirtilmiştir.

```
$sure_baslangici = microtime(true);
insertYayinlar(10000);
$sure = microtime(true) - $sure_baslangici;
$kayit_sayisi = kayitSayisiTablosu();
sureKaydet("insertYayin", $kayit_sayisi, $sure);
```

Program Kodu 4.1 "microtime" fonksiyonunun kullanım kodu

microtime fonksiyonu, "true" parametresi ile verildiğinde float döndürür. insertYayinlar fonksiyonu belirtilen miktarda kayıt ekleme işlemi yapar. Süre başlangıç ve süre bitiş olarak tanımlanan değerler arasındaki süre, sorgunun başladığı ve bittiği an arasındaki fark olarak ortaya çıkmıştır. Ortaya çıkan sürelerin kaydedilmesi için ise sureKaydet fonksiyonu oluşturulmuştur. Çıkan sonuç ile birlikte işlenen veri sayısı ve işlem adı ilgili çözüm yolunun süre kayıt tablosuna eklenmiştir.Program Kodu 4.2. sayesinde oluşan veriler deneyler sonrasında incelenecektir.

```
function sureKaydet($islemadi, $sayac, $sure)
{
   $surekayitekle= pg_fetch_assoc(pg_query($sqlbaglanti,
   "INSERT INTO Tablo_Adı (islemadi, verisayisi, kayitsure)
    VALUES('".$islemadi."','".$sayac."','".$sure."')"));
}
```

Program Kodu 4.2 Süre kaydetme fonksiyonu kodu

pg_query fonksyionu, belirtilen veri tabanı bağlantısında sorguyu çalıştırır. pg_fetch_assoc fonksyionu, getirilen satıra (kayıtlara) karşılık gelen bir ilişkisel dizi döndürür.

4.2.1 Normalizasyon ve İndeksleme içi Yayın Verilerini Ekleme

Belirlenen sayı aralıklarında yayın türü, bölüm, akademisyen ve yıl değerlerinin rastgele belirlenebilmesi amacı ile PHP'nin rastgele sayı üretme fonksiyonu olan rand() fonksiyonu kullanılmıştır. Her ekleme döngüsünde eklenen veri sayısı 10.000 adet arttırılarak toplamda normalize edilmiş ve indeksleme işlemi uygulanmış yayınlar tablolarına test için belirlenen 3.000.000 adet veri eklenmiştir. Program Kodu 4.3'te normalize edilmiş bir tablo ve indeksleme işlemi kullanılmış bir tabloya yayın verilerinin eklenmesi için kullanılan kod gösterilmiştir.

```
function insertYayinlar($veri_miktari)
{
  for($i=1; $i <= $veri_miktari; $i++)
  {
   $yayınismi="Yayın İsmi";
   $yayın_türü=rand(1,9);
   $akademisyenid=rand(1,5000);
   $bölümid=rand(1,2000);
   $yıl=rand(2000,2020);
   $kayıtekle = pg_fetch_assoc(pg_query($sqlbaglanti,
   "INSERT INTO yayınlar_tablo_adı (yayınismi, yayıntürü,
akademisyenid, bölümid,
yil)VALUES('".$yayinismi."',".$yayıntürü.",".$akademisyenid.",".$böl
ümid.",".$yil.")"));
  }
}
```

Program Kodu 4.3 Normalizasyon ve İndeksleme işlemlerinde yayın ekleme kodu

Belirtilen miktarda yayın ekleme işlemini yapan insertYayinlar fonksiyonunun içerisinde kullanılan rand() fonksiyonu PHP'nin belirtilen değerler arasında rastgele tam sayı üretme fonksiyonudur. Eklenen verilerde yayın türü, akademisyen ID, bölüm ID ve yıl değerlerini rastgele atamak için kullanılmıştır.

4.2.2 MongoDB ve Yayın Verilerini Ekleme

Program Kodu 4.4'te verilen yayın eklemek için kullanılan kod incelendiğinde her bir döngüde 10.000 veri eklendiği görülmektedir. Yayın türü, akademisyen ID, bölüm ID ve yıl değerleri için rand() fonksiyonu kullanılmış olup belirlenen aralıklarda değerler atanmıştır. Her bir veri ekleme döngüsünün sonunda, veri sayısının sorgu hızlarına etkilerini görebilmek amacı ile kayıtlı olan tüm veriler için sorgulama işlemleri yapılmıştır. Çıkan sonuçlar sorgu sürelerini kayıt altında tutmak için oluşturulan tabloya kaydedilmiştir.

```
for($j=1;$j<=100;$j++){
        $sure_baslangici = microtime(true);
        $veri_miktari=10000*$j;
        for($i=1; $i <= 10000; $i++)
        {
                $yayinismi="a";
                $yayinturu=rand(1, 9);
                $akademisyenid=rand(1,5000);
                $bolumid=rand(1,2000);
                $yil=rand(2000,2020);
                $doc = array
                (
                 'yayinismi' =>'Yayın İsmi',
                 'yayinturu'=>$yayinturu,
                 'akademisyenid'=>$akademisyenid ,
                 'bolumid'=>$bolumid,
                 'yil'=>$yil
                );
$insertOneResult = $yayinlarmongodb->insertOne($doc);
        }
}
```

Program Kodu 4.4 MongoDB yayın ekleme kodu

Array() olarak tanımlanan diziler bir değişkende birden çok veriyi tutan yapılardır. Bir koleksiyonda anahtar değer çiftleri olarak saklanan veriler benzer türlere sahiptir. MongoDB'de veriler koleksiyonlarda dizi şeklinde saklanır. Bu yüzden Program Kodu 4.4'te gözüktüğü gibi array () komutu ile ekleme işlemi yapılmıştır. insertOne() ise MongoDB'de tek veri ekleme metodudur.

4.2.3 Denormalizasyon ve Yayın Verilerini Ekleme

Program Kodu 4.5'te Denormalize işlemleri uygulanmış bir veri tabanına veri eklemek için oluşturulan fonksiyon gösterilmiştir. Fonksiyon incelendiğinde diğer işlemlerde yapıldığı gibi yayın türü, akademisyen ID, bölüm ID ve yıl değerleri için rand() fonksiyonu kullanılmış olup belirlenen aralıklarda değerler atanmıştır. Her bir veri ekleme döngüsünün sonunda, veri sayısının sorgu hızlarına etkilerini görebilmek amacı ile kayıtlı olan tüm veriler için sorgulama işlemleri yapılmıştır. Çıkan sonuçlar sorgu sürelerini kayıt altında tutmak için oluşturulan tabloya kaydedilmiştir.

```
function insertYayinlar($veri_miktari)
{
  for($i=1; $i <= $veri_miktari; $i++)
  {
   $yayinismi="Yayın İsmi";
   $yayinturu=rand(1,9);
   $akademisyenid=rand(1,5000);
   $bolumid=rand(1,2000);
   $yil=rand(2000,2020);
   $kayitekle = pg_fetch_assoc(pg_query($sqlbaglanti,
   "INSERT INTO yayinlardenormalizasyon (yayinismi, yayinturu,
akademisyenid, bolumid, yil)
VALUES('".$yayinismi."',".$yayinturu.",".$akademisyenid.",".$bolumid
.",".$yil.")"));
  }
}
```

Program Kodu 4.5 Denormalizasyon yayın ekleme kodu

Sorgu 4 için Yayın_Türleri_Bölüm tablosu oluşturulmuştur. Program Kodu 4.6'da belirtilen tetikleyici sayesinde Yayınlar tablosuna yeni bir veri eklendiği zaman aynı bölüm ve

yayın türüne ait veri satırı varsa o satırdaki sayı değeri 1 artar. Aynı değerlere ait veri satırı yoksa yeni bir satır eklenir ve sayı değeri 1 olarak atanır. Veri silme işlemi yapıldığında ise verinin ait olduğu satırdaki sayı değeri 1 azalır.

Sorgu 4 için yapılan işlemler diğer sorgulama işlemleri içinde uygulandığında, Sorgu 1 için oluşturulan Yayın_Türleri tablosunda yayın türü değeri, Sorgu 2 için oluşturulan Yayın_Türleri_ Akademisyen tablosunda yayın türü ve akademisyen değeri, Sorgu 3 için oluşturulan Yayın_Türleri_Bölüm tablosunda yayın türleri ve bölüm değeri ve Sorgu 5 için oluşturulan Yayın_Türleri_Bölüm_Yıl tablosunda yayın türü 1 olan yayınların bölüm ve yıl değerine göre ait oldukları satırdaki sayı değerleri yapılan işlemlere göre artar, eklenir veya azalır.

```
BEGIN
IF TG_OP = 'INSERT' THEN
    IF      NEW."Bölüm_ID"=(SELECT      "Bölüm_ID"      FROM
"Yayın_Türleri_Bölüm"  WHERE  "Bölüm_ID"=NEW."Bölüm_ID"  AND
"Yayın_Tür_ID" =NEW."Yayın_Tür_ID")
    THEN
 UPDATE  "Yayın_Türleri_Bölüm"  SET  Sayı=  (Sayı+1)  WHERE
"Yayın_Tür_ID"=NEW."Yayın_Tür_ID"                          AND
"Bölüm_ID"=NEW."Bölüm_ID";
   ELSE
        INSERT  INTO  "Yayın_Türleri_Bölüm"  ("Yayın_Tür_ID",
"Bölüm_ID",                                                Sayı)
VALUES(NEW."Yayın_Tür_ID",NEW."Bölüm_ID",1);
    END IF;
 RETURN NEW;
END IF;
IF TG_OP = 'DELETE' THEN
    UPDATE  "Yayın_Türleri_Bölüm"  SET  Sayı=  (Sayı-1)  WHERE
"Yayın_Tür_ID"=OLD."Yayın_Tür_ID"                          AND
"Bölüm_ID"=OLD."Bölüm_ID";
   RETURN OLD;
END IF;
END;
```

Program Kodu 4.6 Denormalizasyon tetikleyici kodu

Program Kodu 4.6'da BEGIN komutu ile başlayan, END komutuyla biten tetikleyici kodumuzda TG_OP ile ekleme (INSERT), güncelleme (UPDATE) ya da silme (DELETE) işlemlerinden hangisinin yapılacağı belirtilmiştir. Yayınlar tablosuna veri girişi yapıldığında eğer tetikleyici için oluşturulan tabloda eklenen veri değerlerine ait bir satır varsa, satırda bulunan sayı değeri UPDATE komutu ile 1 arttırılır. Eğer tetikleyici için oluşturulan tabloda eklenen veri değerine ait satır yoksa INSERT komutu ile yeni bir satır eklenir ve sayı değeri olarak 1 atanır. Yayınlar tablosundan herhangi bir silme işlemi yapıldığında ise silinen değerin ait olduğu satırdaki sayı değeri UPDATE komutu ile 1 azaltılır.

4.2.4 Yayın verilerini ekleme sonuçları

Çizelge 4.1 ve Çizelge 4.2'de gösterildiği gibi kayıtlı olan veri sayısına göre her 10.000 adet veri girişi yapıldığında, toplam veriler için Çözüm 1, Çözüm 2, Çözüm 3 ve Çözüm 4 işlemleri sonucunda alınan süre değerleri karşılaştırılmıştır. Bu karşılaştırmalar sonucunda ortaya çıkan değerler incelendiğinde yayın ekleme sürelerinde, veri sayısı az olduğunda olumsuz bir fark görülmese de veri sayısı arttıkça yayın ekleme süreleri açısından karşılaştırma yapıldığında en olumsuz performans denormalizasyon işleminde alınmıştır.

Çizelge 4.1 Veri ekleme sonuçları (Mikrosaniye)

Veri sayısı	Normalizasyon	İndeksleme	MongoDB	Denormalizasyon
10.000	5,53240	5,40623	2,31351	24,11738
100.000	4,94518	5,27324	2,23552	246,69085
250.000	5,29008	5,16888	2,35630	452,53377
500.000	5,04006	5,37520	2,26418	682,26990
1.000.000	5,17066	5,46595	2,28490	868,14671
2.000.000	5,52351	5,54577	2,27217	1051,08329
3.000.000	5,15748	5,42643	2,22907	1000,73677

Çizelge 4.2 Tüm verilerin çözüm yollarına göre ortalama ekleme değerleri

Çözüm Yolu	Ortalama Ekleme Değeri
Normalizasyon	0,00051± 0,00001
İndeksleme	0,00054± 0,00001
MongoDB	0,00023± 0,00001
Denormalizasyon	0,08999± 0,00001

MongoDB veri ekleme işlemi çok hızlıdır. Uygulama tarafında sorgulamalardan ziyade veri tutmanın önemli olduğu uygulamalarda MongoDB tercih edilebilir. İndeksleme işlemi beklenildiği gibi ekleme işlemini yavaşlatmıştır. İndekslemenin ek dosyaya ihtiyaç duyduğu ve ek işlemler sebebiyle sistemi yavaşlatması beklenen bir durumdur. Ancak, sistemi çokta yavaşlatmadığı görülmektedir. Denormalizasyon işleminde bir kayıt eklendiğinde arka planda birçok tabloda güncellemeler yapılmaktadır. Bu güncellemelerin diğer yöntemlere göre büyük bir zaman kaybı oluşturmaktadır. Denormalizasyon süresi MongoDB'ye kayıt eklemeye göre yaklaşık 391 kat daha yavaştır. Bir sonraki bölümde sorgu performanslarına etkisi araştırılacaktır.

4.3 Sorgu Testleri

4.3.1 Normalizasyon ve İndeksleme Sorguları

Verilerin yayın türlerine göre yayın sayısının sorgulanması, normalizasyon ve indeksleme çözümlerinde Program kodu 4.7'de GROUP BY ifadesiyle yayın türlerine göre gruplanan yayınların sayıları COUNT metodu ile hesaplanmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tablolara eklenmiştir.

```
SELECT "Yayın_Türü", COUNT(*) FROM "Tablo_Adı" GROUP BY "Yayın_Türü"
```

Program Kodu 4.7 Sorgu 1- Normalizasyon ve İndeksleme

Verilerin yayın türü ve yıla göre yayın sayısının sorgulanması, normalizasyon ve indeksleme çözümlerinde Program Kodu 4.8'de GROUP BY ifadesiyle yayın türleri ve yıllara göre gruplanan yayınların sayıları COUNT metodu ile hesaplanmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tablolara eklenmiştir.

```
SELECT "Yayın_Türü", "Yıl", COUNT(*) FROM "Tablo_Adı" GROUP BY
"Yayın_Türü", "Yıl"
```

Program Kodu 4.8 Sorgu 2- Normalizasyon ve İndeksleme

Verilerin yayın türü ve akademisyene göre yayın sayısının sorgulanması, normalizasyon ve indeksleme çözümlerinde Program kodu 4.9'da GROUP BY ifadesiyle yayın türleri ve akademisyenlere göre gruplanan yayınların sayıları COUNT metodu ile hesaplanmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tablolara eklenmiştir.

```
SELECT "Yayın_Türü", "Akademsiyen_ID",COUNT(*) FROM "Tablo_Adı"
GROUP BY "Yayın_Türü", "Akademsiyen_ID"
```

Program Kodu 4.9 Sorgu 3- Normalizasyon ve İndeksleme

Verilerin yayın türü ve bölüme göre yayın sayısının sorgulanması, normalizasyon ve indeksleme çözümlerinde Program Kodu 4.10'da GROUP BY ifadesiyle yayın türleri ve bölümlere göre gruplanan yayınların sayıları COUNT metodu ile hesaplanmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tablolara eklenmiştir.

```
SELECT "Yayın_Türü", "Bölüm_ID", COUNT(*) FROM "Tablo_Adı"
GROUP BY "Yayın_Türü", "Bölüm_ID"
```

Program Kodu 4.10 Sorgu 4- Normalizasyon ve İndeksleme

Verilerin yayın türü 1 olan, yıl ve bölüm değerlerine göre yayın sayısının sorgulanması, normalizasyon ve indeksleme çözümlerinde Program Kodu 4.11'de WHERE ifadesi ile yalnızca yayın türü 1 olan yayınlar seçilirken, GROUP BY ifadesiyle yıl ve bölüm değerlerine göre gruplanan yayınların sayıları COUNT metodu ile hesaplanmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tablolara eklenmiştir.

```
SELECT "Yıl", "Bölüm_ID", COUNT(*) FROM "Tablo_Adı" WHERE
"Yayın_Türü"=1 GROUP BY "Yıl", "Bölüm_ID"
```

Program Kodu 4.11 Sorgu 5- Normalizasyon ve İndeksleme

4.3.2 MongoDB Sorguları

Verilerin yayın türlerine göre yayın sayısının sorgulanması, MongoDB çözümünde Program kodu 4.12'de gösterildiği şekilde yapılmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
$yayin = $yayinlarmongodb->aggregate(array(
    array('$group'      =>      array('_id'      =>
'$yayinturu', 'count' => array('$sum' => 1))),
    array('$sort' => array('count' => -1)),
));
```

Program Kodu 4.12 Sorgu 1 - MongoDB

SQL'de GROUP BY ve COUNT komutları ile yapılan işlemler için MongoDB'de aggregate() fonksiyonu kullanılır. Aggregate() fonksiyonu dokümanları $group ile belirlenen kriterlere göre gruplandırır. Program Kodu 4.13'te $group komutu ile yayınlar, yayın türlerine göre gruplanmıştır. Belirlenen grupların sayı değerleri ise yine belge değerini 1 olarak döndüren $sum komutu ile count değerini 1 arttırarak hesaplanmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
$yayin = $yayinlarmongodb->aggregate(array(
    array('$group'         =>         array('_id'
=>array('$yayinturu','$yil'),'count'          =>
array('$sum' => 1))),
    array('$sort' => array('count' => -1)),
));
```

Program Kodu 4.13 Sorgu 2- MongoDB

Verilerin yayın türleri ve akademisyenlere göre yayın sayısının sorgulanması, MongoDB çözümünde Program kodu 4.14'te gösterildiği gibi $group komutu ile yayın türleri ve akademisyenlere göre gruplandırılmıştır. Belirlenen grupların sayı değerleri ise yine belge değerini 1 olarak döndüren $sum komutu ile count değerini 1 arttırarak hesaplanmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
$yayin = $yayinlarmongodb->aggregate(array(
    array('$group'        =>        array('_id'
=>array('$yayinturu','$akademisyenid'),'count'
=> array('$sum' => 1))),
    array('$sort' => array('count' => -1)),
));
```

Program Kodu 4.14 Sorgu 3 - MongoDB

Verilerin yayın türleri ve bölümlere göre yayın sayısının sorgulanması, MongoDB çözümünde Program kodu 4.15'te gösterildiği gibi $group komutu ile yayın türleri ve bölümlere göre gruplandırılmıştır. Belirlenen grupların sayı değerleri ise yine belge değerini 1 olarak döndüren $sum komutu ile count değerini 1 arttırarak hesaplanmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
$yayin = $yayinlarmongodb->aggregate(array(
    array('$group'        =>        array('_id'
=>array('$yayinturu','$bolumid'),'count'      =>
array('$sum' => 1))),
    array('$sort' => array('count' => -1)),
));
```

Program Kodu 4.15 Sorgu 4 - MongoDB

Verilerin yayın türü 1 olan, bölümler ve yıllara göre yayın sayısının sorgulanması, MongoDB çözümünde Program kodu 4.16'da gösterildiği gibi $match komutu ile yalnızca yayın türleri 1 olan yayınlar belirlenirken, $group komutu ile tabloda bulunan veriler bölümlere ve yıllara göre gruplandırılmıştır. Belirlenen grupların sayı değerleri ise yine belge değerini 1 olarak döndüren $sum komutu ile count değerini 1 arttırarak hesaplanmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
$yayin = $yayinlarmongodb->aggregate(array(
       array('$match' => array('yayinturu'=>1)),
    array('$group'                    =>                    array('_id'
=>array('yayinturu'=>'$yayinturu','bolumid'=>'$bolumid','yil'=>'$yil
'),'count' => array('$sum' => 1))),
    array('$sort' => array('count' => -1)),
));
```

Program Kodu 4.16 Sorgu 5 – MongoDB

4.3.3 Denormalizasyon Sorguları

Verilerin yayın türlerine göre yayın sayısının sorgulanması, Denormalizasyon çözümünde Program kodu 4.17'de gösterildiği Yayın_Türleri tablosundan yapılmıştır. Yalnızca yayın türü ile sayı değerinin bulunduğu ve tetikleyici kod ile güncellenen tablodaki sayı değerine ulaşılma süresi Sorgu 1 'in sonucu olarak ortaya çıkar. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
SELECT "Yayın_Türü", "Sayı" FROM "Yayın_Türleri"
```

Program Kodu 4.17 Sorgu 1 - Denormalizasyon

Verilerin yayın türleri ve yıllara göre yayın sayısının sorgulanması, Denormalizasyon çözümünde Program kodu 4.18'de gösterildiği gibi Yayın_Tur_Yıl tablosundan yapılmıştır. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
SELECT "Yayın_Türü", "Yıl", "Sayı" FROM "Yayın_Tür_Yıl"
```

Program Kodu 4.18 Sorgu 2 – Denormalizasyon

Verilerin yayın türleri ve akademisyenlere göre yayın sayısının sorgulanması, Denormalizasyon çözümünde Program kodu 4.19'da gösterildiği gibi Yayın_Tür_Akademisyen tablosundan yapılmıştır. Yayın türü, akademisyen ID ile birlikte sayı değerinin bulunduğu ve tetikleyici kod ile güncellenen tablodaki sayı değerine ulaşılma

süresi Sorgu 3'ün sonucu olarak ortaya çıkar. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
SELECT "Yayın_Türü", "Akademisyen_ID", "Sayı" FROM
"Yayın_Tür_Akademisyen"
```

Program Kodu 4.19 Sorgu 3 - Denormalizasyon

Verilerin yayın türleri ve bölümlere göre yayın sayısının sorgulanması, Denormalizasyon çözümünde Program kodu 4.20'de gösterildiği şekilde Yayın_Tür_Bölüm tablosundan yapılmıştır. Yayın türü ve bölüm ID değerleri ile birlikte sayı değerinin de bulunduğu, tetikleyici kod ile güncellenen tablodaki sayı değerine ulaşılma süresi Sorgu 4'ün sonucu olarak ortaya çıkar. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
SELECT "Yayın_Türü", "Bölüm_ID", "Sayı" FROM
"Yayın_Tür_Bölüm"
```

Program Kodu 4.20 Sorgu 4- Denormalizasyon

Verilerin yayın türü 1 olan, bölümler ve yıllara göre yayın sayısının sorgulanması, Denormalizasyon çözümünde Program kodu 4.21'de gösterildiği Yayın_Tür_Bölüm_Yıl tablosundan yapılmıştır. Where komutu ile yalnızca yayın türü 1 olan yayınlar belirlenirken, tablodaki yıl ve bölüm ID değerleri ile birlikte sayı değerinin de bulunduğu, tetikleyici kod ile güncellenen tablodaki sayı değerine ulaşılma süresi Sorgu 5'in sonucu olarak ortaya çıkar. Sorgu işlemlerinin gerçekleştiği süreler, süre kayıtlarının tutulması için oluşturulan tabloya eklenmiştir.

```
SELECT "Yıl", "Bölüm_ID", "Sayı" FROM "Yayın_Tür_Bölüm_Yıl"
WHERE "Yayın_Türü"=1
```

Program Kodu 4.21 Sorgu 5- Denormalizasyon

4.3.4 Kayıt Sayısının Sorgu Sürelerine Etkisi

Kayıt sayılarının sorgu sürelerini etkilerinin karşılaştırılması için Çizelge 4.3, Çizelge 4.4, Çizelge 4.5, Çizelge 4.6, Çizelge 4.7 oluşturulmuştur.

Sorgu 1 için oluşturulan Çizelge 4.3 incelendiğinde veri sayısının artmasıyla sorgulama süreleri Normalizasyon işlemleri sonucunda yaklaşık olarak 240 kat arttığı, İndeksleme işlemlerinde ise bu değerin yaklaşık 255 kat olduğu gözlemlenmiştir. MongoDB işlemlerinde 59 kat artsa da genel olarak sorgulama işlemi MongoDB'de diğer yöntemlere göre daha yavaş olduğu için en uzun süreyi MongoDB işlemleri sonucunda almaktayız. Denormalizasyon işlemleri sonucunda ise bu değer yaklaşık olarak 1,8 kat artarak sonuca en hızlı şekilde ulaşmamızı sağladığı görülmüştür.

Çizelge 4.3 Sorgu 1 (Mikrosaniye)

Veri Sayısı Aralığı	Normalizasyon	İndeksleme	MongoDB	Denormalizasyon
0 - 10.000	0,0031791	0,0028661	0,0724849	0,0002069
90.000 - 100.000	0,0250058	0,0451958	0,1479809	0,0002491
240.000 - 250.000	0,0630219	0,0585461	0,3869950	0,0002491
490.000 - 500.000	0,1588540	0,1189971	0,7463481	0,0002861
990.000 - 1.000.000	0,2519660	0,2399771	1,4347701	0,0003199
1.990.000 - 2.000.000	0,5023959	0,4833400	2,8935630	0,0002808
2.990.000 - 3.000.000	0,7645020	0,7337210	4,2856581	0,0003879

Sorgu 2 için oluşturulan Çizelge 4.4 incelendiğinde veri sayısının artmasıyla sorgulama süreleri Normalizasyon işlemlerinde yaklaşık olarak 179 kat arttığı, İndeksleme işlemlerinde ise bu değerin yaklaşık 269 kat olduğu gözlemlenmiştir. MongoDB işlemlerinde ise bu farkın 256 kat olduğu hesaplanmıştır. Denormalizasyon işlemleri sonucunda ise bu değer yaklaşık olarak 1,7 kat artarak sonuca en hızlı şekilde ulaşmamızı sağladığı görülmüştür.

Çizelge 4.4 Sorgu 2 (Mikrosaniye)

Veri Sayısı Aralığı	Normalizasyon	İndeksleme	MongoDB	Denormalizasyon
0 - 10.000	0,0052452	0,0033719	0,0221371	0,0003499
90.000 - 100.000	0,0317549	0,0444018	0,2010450	0,0004060
240.000 - 250.000	0,0785980	0,0728709	0,4728450	0,0005769
490.000 - 500.000	0,1543850	0,1486499	0,9548380	0,0004050
990.000 - 1.000.000	0,3131568	0,2937328	1,9184138	0,0004329
1.990.000 - 2.000.000	0,6227610	0,5981650	3,8171830	0,0005300
2.990.000 - 3.000.000	0,9433190	0,9088180	5,6707139	0,0006198

Sorgu 3 için oluşturulan Çizelge 4.5 incelendiğinde veri sayısının artmasıyla sorgulama süreleri Normalizasyon işlemlerinde yaklaşık olarak 71 kat arttığı, İndeksleme işlemlerinde ise bu değerin yaklaşık 67 olduğu gözlemlenmiştir. MongoDB işlemlerinde ise bu farkın 200 kat olduğu hesaplanmıştır. Denormalizasyon işlemleri sonucunda ise bu değer yaklaşık olarak 4,3 kat artarak yine sonuca en hızlı şekilde ulaşmamızı sağladığı görülmüştür.

Çizelge 4.5 Sorgu 3 (Mikrosaniye)

Veri Sayısı Aralığı	Normalizasyon	İndeksleme	MongoDB	Denormalizasyon
0 - 10.000	0,0230138	0,02032685	0,0364120	0,0083980
90.000 - 100.000	0,0815989	0,1831941	0,3260338	0,0344901
240.000 - 250.000	0,1791179	0,1500589	0,6723620	0,0404489
490.000 - 500.000	0,2933838	0,2685689	1,2997689	0,0431199
990.000 - 1.000.000	0,5496089	0,5315039	2,4764158	0,0339379
1.990.000 - 2.000.000	1,0059950	0,9445400	4,9374041	0,0509400
2.990.000 - 3.000.000	1,6376211	1,3764929	7,3153729	0,0364379

Sorgu 4 için oluşturulan Çizelge 4.6 incelendiğinde veri sayısının artmasıyla sorgulama süreleri Normalizasyon işlemlerinde yaklaşık olarak 70 kat arttığı, İndeksleme işlemlerinde ise bu değerin yaklaşık 75 olduğu gözlemlenmiştir. MongoDB işlemlerinde ise bu farkın 190 kat olduğu hesaplanmıştır. Denormalizasyon işlemleri sonucunda ise bu değer yaklaşık olarak 2,4 kat artarak yine sonuca en hızlı şekilde ulaşmamızı sağladığı görülmüştür.

Çizelge 4.6 Sorgu 4 (Mikrosaniye)

Veri Sayısı Aralığı	Normalizasyon	İndeksleme	MongoDB	Denormalizasyon
0 - 10.000	0,0181539	0,0150499	0,0391151	0,0076079
90.000 - 100.000	0,0553288	0,1834850	0,3368890	0,0144441
240.000 - 250.000	0,1260530	0,1524279	0,6503541	0,0155909
490.000 - 500.000	0,3054599	0,2184979	1,2450649	0,0215318
990.000 - 1.000.000	0,4258859	0,4903249	2,4230771	0,0144939
1.990.000 - 2.000.000	0,7926671	0,7609272	4,9115710	0,0186498
2.990.000 - 3.000.000	1,2567019	1,1335639	7,4671831	0,0184991

Sorgu 5 için oluşturulan Çizelge 4.7 incelendiğinde ise veri sayısının artmasıyla sorgulama süreleri Normalizasyon işlemlerinde yaklaşık olarak 123 kat arttığı, İndeksleme işlemlerinde ise bu değerin yaklaşık 179 olduğu gözlemlenmiştir. MongoDB işlemlerinde ise bu farkın 168 kat olduğu hesaplanmıştır. Denormalizasyon işlemleri sonucunda ise bu değer yaklaşık olarak 36 kat artarak sonuca en hızlı şekilde ulaşmamızı sağladığı görülmüştür.

Çizelge 4.7 Sorgu 5 (Mikrosaniye)

Veri Sayısı Aralığı	Normalizasyon	İndeksleme	MongoDB	Denormalizasyon
0 - 10.000	0,0048630	0,0021970	0,0190639	0,0021791
90.000 - 100.000	0,0243551	0,0169110	0,1406009	0,0249168
240.000 - 250.000	0,0638909	0,0431628	0,3080110	0,0353369
490.000 - 500.000	0,1261041	0,0740308	0,5926301	0,0580220
990.000 - 1.000.000	0,2033491	0,1176741	1,1643409	0,0672340
1.990.000 - 2.000.000	0,3722870	0,2449309	2,1794829	0,0745830
2.990.000 - 3.000.000	0,5995311	0,3934528	3,2161028	0,0796060

Ekleme işleminde MongoDB hızlı olsa da sorgulama açısından yavaş kaldığı görülmektedir. İndeksleme işlemi kayıt eklemeyi yavaşlatmasına rağmen sorgu sürelerini iyileştirdiği görülmektedir. Denormalizasyon yönteminin ise sorgulamaları çok hızlandırdığı görülmektedir.

4.3.5 Çözümlerin Son Durumlarının Karşılaştırılması

Yapılan sorgulamalar sonucu ortaya çıkan süre kayıtlarının her çözüm yolu için ortalamaları ayrı ayrı hesaplandığında, sonuçların incelenmesi için Çizelge 4.8 ve Şekil 4.1 oluşturulmuştur.

Çizelge 4.8 incelendiğinde Normalizasyon yönteminde veri sayısının arttıkça tüm sorgu çeşitlerinde sorgu sürelerinin arttığı gözlemlenmiştir.

Çizelge 4.8 Normalizasyon sorgu süreleri (Mikrosaniye)

Veri Sayıları Aralığı	Sorgu 1	Sorgu 2	Sorgu 3	Sorgu 4	Sorgu 5
0 - 10.000	0,0031797	0,0052452	0,0230138	0,0181539	0,0048630
90.000 - 100.000	0,1588540	0,1543850	0,2933838	0,3054599	0,1261041
240.000 - 250.000	0,2519659	0,3131568	0,5496089	0,4258859	0,2033491
490.000 - 500.000	0,3714020	0,4634451	0,7741439	0,6021091	0,2884988
990.000 - 1.000.000	0,5023958	0,6227610	1,0059950	0,7926671	0,9722870
1.990.000 - 2.000.000	0,7645020	0,9433190	1,6376211	1,2567019	0,5995311

Çizelge 4.9 incelendiğinde İndeksleme yönteminde veri sayısının arttıkça tüm sorgu çeşitlerinde sorgu sürelerinin arttığı gözlemlenmiştir.

Çizelge 4.9 İndeksleme sorgu süreleri (Mikrosaniye)

Veri Sayıları Aralığı	Sorgu 1	Sorgu 2	Sorgu 3	Sorgu 4	Sorgu 5
0 - 10.000	0,0028660	0,0033719	0,0203268	0,0150499	0,0021970
90.000 - 100.000	0,1189970	0,1486499	0,268568	0,2184979	0,0740308
240.000 - 250.000	0,2399771	0,2937328	0,5315039	0,4903249	0,1176741
490.000 - 500.000	0,3544650	0,5054380	0,7459690	0,5830421	0,1913340
990.000 - 1.000.000	0,4833400	0,5981650	0,9445400	0,7609272	0,2119309
1.990.000 - 2.000.000	0,7337210	0,9088180	1,3764929	1,1335639	0,3934528

Çizelge 4.10 incelendiğinde MongoDB yönteminde veri sayısının arttıkça tüm sorgu çeşitlerinde sorgu sürelerinin arttığı gözlemlenmiştir.

Çizelge 4.10 MongoDB sorgu süreleri (Mikrosaniye)

Veri Sayıları Aralığı	Sorgu 1	Sorgu 2	Sorgu 3	Sorgu 4	Sorgu 5
0 - 10.000	0,0724849	0,0221371	0,0364120	0,0391151	0,0190639
90.000 - 100.000	0,7463481	0,9548380	1,2997689	1,2450649	0,5926301
240.000 - 250.000	1,4347701	1,9184138	2,4764158	2,4230771	1,1643409
490.000 - 500.000	2,1590270	3,5574390	3,6414349	3,6054041	1,7040040
990.000 - 1.000.000	2,8935630	3,8171830	4,9374041	4,9115710	2,1794829
1.990.000 - 2.000.000	4,2856581	5,6707139	7,3153729	7,4671831	3,2161028

Çizelge 4.11 incelendiğinde Denormalizasyon yönteminde veri sayısının artmasının sorgu sürelerine etki etmediği gözlemlenmiştir.

Çizelge 4.11 Denormalizasyon sorgu süreleri (Mikrosaniye)

Veri Sayıları Aralığı	Sorgu 1	Sorgu 2	Sorgu 3	Sorgu 4	Sorgu 5
0 - 10.000	0,0002069	0,0003499	0,0083980	0,0076079	0,0021791
90.000 - 100.000	0,0002861	0,0004050	0,0431199	0,0215318	0,0580220
240.000 - 250.000	0,0003199	0,0004329	0,0339379	0,0144939	0,0672340
490.000 - 500.000	0,0003490	0,0006899	0,0422070	0,0219750	0,0748360
990.000 - 1.000.000	0,0002808	0,0005300	0,0509400	0,0186498	0,0745830
1.990.000 - 2.000.000	0,0003879	0,0006198	0,0364379	0,0184991	0,0796060

Çizelge 4.12'de belirtilen ortalama sorgu süreleri karşılaştırıldığında Normalizasyon ve İndeksleme değerleri için sonuçların çok yakın olduğu görülmektedir. Ekleme işlemlerinde en hızlı sonucu veren MongoDB'nin, sorgulama işlemlerinde sonuca en yavaş şekilde ulaştığı gözlemlenmiştir. Denormalizasyon yöntemi ile yapılan sorguların süreleri ise en iyi ortalama değerleri vermiştir.

Çizelge 4.12 Sorgu süreleri ortalamaları (Mikrosaniye)

Sorgular	Normalizasyon	İndeksleme	MongoDB	Denormalizasyon
Sorgu 1	0,39400± 0,00001	0,37496± 0,00001	2,20061± 0,00001	0,00029± 0,00001
Sorgu 2	0,48083± 0,00001	0,45908± 0,00001	2,96141± 0,00001	0,00047± 0,00001
Sorgu 3	0,78629± 0,00001	0,73148± 0,00001	3,77114± 0,00001	0,03819± 0,00001
Sorgu 4	0,63714± 0,00001	0,57887± 0,00001	3,66711± 0,00001	0,01719± 0,00001
Sorgu 5	0,30024± 0,00001	0,19236± 0,00001	1,66230± 0,00001	0,07230± 0,00001

4.3.6 Dosya Boyutlarının Karşılaştırılması

Bu bölümde tüm çözüm yolları için, 3.000.000 veride ortaya çıkan dosya ve indeks boyutlarına yer verilmiştir.

Çizelge 4.13 incelendiğinde en küçük dosya boyutuna sahip çözüm yolunun, verileri tek satırda tutan MongoDB olduğu gözlemlenmektedir. Normalizasyon ve denormalizasyon işlemlerinde tablo boyutları aynı kalırken, normalize edilmiş bir tabloya göre indeksleme işlemi uygulanmış tabloda dosya boyutunun arttığı görülmektedir. İndeksleme işlemlerinin sorgu sürelerini hızlandırmasına rağmen dosya boyutunun 3.22 kat artması depolama maliyetini arttıracaktır. Bu durum olumsuz olarak değerlendirilmiştir.

Denormalizasyon değerlendirmeleri sonucu oluşturulan tabloların boyutları ise Çizelge 4.14'te verildiği gibi 0.04 MB, 1.120 MB, 2.752 MB, 21.504 MB olmuştur.

Çizelge 4.13. 3.000.000 veriye sahip tabloların boyutları

Çözüm Yolu	Dosya Boyutları
Normalizasyon	149 MB
İndeksleme	480 MB
MongoDB	78.80 MB
Denormalizasyon	149 MB

Çizelge 4.14. Denormalize tabloların boyutları

Tablo Adı	Dosya Boyutu
Yayın_Tür	0.040 MB
Yayın_Tür_Bölüm	1.120 MB
Yayın_Tür_Akademisyen	2.752 MB
Yayın_Tür_Bölüm_Yıl	21.504 MB

Çizelge 4.15'te görüldüğü gibi indeksleme işlemi ve MongoDB için ek depolama maliyeti ortaya çıkar. İndeksleme ek dosya maliyeti oluştursa da performans üzerinde olumlu etkisi vardır.

Çizelge 4.15. 3.000.000 veriye sahip tablolarda indeks boyutu

Çözüm Yolu	İndeks Boyutu
Normalizasyon	0
İndeksleme	330,50 MB
MongoDB	35,05 MB
Denormalizasyon	0

5. SONUÇLAR

Web uygulamalarının sayısı hızla artması ve birçok uygulamanın web ortamına taşınması ile birlikte bu uygulamaların daha kısa sürede cevap vermesi bir problem olmuştur. Bu tezde, veri tabanı tarafında tasarımın önemi ve denormalizasyon sayesinde sorgu performansındaki iyileşmeler gösterilmiştir. Web uygulaması olarak üniversitemizin akademik veri sistemi üzerine testler yapılmıştır.

Normalizasyon işlemi literatürde en çok bilinen veri tabanı tasarım yöntemidir. Geliştirilen ilişkisel veri tabanı yönetim sistemleri bu yöntemi dikkate alıp geliştirilmişlerdir. Veri sayısı attıkça farklı tablolardan verileri toplamak ve hesaplama fonksiyonlarını çalıştırmak zaman alıcı süreçler haline gelmiştir. İndeksleme sayesinde bu zaman alıcı süreç hızlandırılabilir. Diğer taraftan veriyi farklı bilgisayarlara dağıtma ve daha hızlı şekilde yönetme için NoSQL veri tabanları ortaya çıkmıştır. Özellikle web ortamında MongoDB JSON verileri depolayabildiği için tercih edilmektedir. Bu tezde, MongoDB'nin sorgu fonksiyonları da test edilmiştir. Ekleme konusunda çok başarılı olmasına rağmen sorgu cevap süresi konusunda yavaş kaldığı görülmüştür.

Denormalizasyon yöntemi, normalizasyonun amacı olan veri tekrarını önlemeyi amaçlamaz. Denormalizasyon, ek veriler kullanıp sorgu performansı iyileştirmeyi odaklanır. Bu yüzden sorgulara göre uygun eklemeler ve kodlar yazılmalıdır. Başka bir deyişle denormalizasyon yönetimi zorlaştırır ve daha fazla kod yazmamıza sebebiyet verir. Bu tezde tetikleyiciler kullanılmış ve ek alanlar tetikleyici kodları ile düzenlenmiştir. Diğer yöntemlere göre ekleme olayının yavaşlamasına rağmen sorguların hızlandığı görülmektedir. Trigger'ların yavaş kalması durumda literatürde alternatif çözümler oluşmaktadır. Bu çözümlerden biri olan OLAP (Online analytical processing) veri eklendikten sonra değil, belli periyotlarda veriyi analiz edip daha kolay sorgulanabilecek veriler üretir. Bu yeni üretilen veriler üzerinden çok daha hızlı sorgulamalar yapılabilir. Bu çözümde, sorgulamaların güncelliği periyotlara bağlı olacaktır. Periyot saatlik veya günlük tutulursa sorgulanacak verinin oluşturulmasıyla sık sık sistem kaynakları kullanılacaktır. Periyot aylık yapılırsa sorgulama güncelliği için bir aylık bekleme söz konusu olacaktır. Sorgulama sonuçlarını hızlandırmak için diğer bir çözüm bellek tabanlı veri tabanlarının kullanılması olabilir. Günümüzde REDIS ve Apache Ignite gibi uygulamalar ön plana çıkmıştır. Bu uygulamalar fiziksel disk yerine belleği kullandığı için çok hızlı sorgulamalara destek verir. Ancak, veri sayısına bağlı olarak sistem kaynaklarını fazlasıyla kullanan uygulamalardır. Son çözüm,

tetikleyiciler yerine web uygulaması tarafında Kafka ve RabbitMQ gibi veri işlem hatları düşünülebilir. Bu hatlar sayesinde bir kuyruk modeli oluşturulup tetikleyiciler gibi çalışmanın sonlanması beklenmez ve ekleme yapıldığında değişiklikler Kafka veya RabbitMQ üzerine gönderilir. Burada veri sayısına bağlı olarak tabloların güncellenmesi zaman alabilir ancak kaydetme işlemi çok daha hızlı sonuçlanacaktır. Bu tezdeki denormalizasyon uygulamasında sorgulanacak verinin güncel olmasına ve sistem kaynaklarını az kullanmasına dikkat edilmiştir.

Bir web uygulamasında veri sayısı, kod yazma maliyeti, eldeki donanım kaynakları, web sayfalarında istenen cevap süreleri, sistemin aynı anda kaç kişi tarafından kullanılacağı gibi kriterler göz önüne alınıp bir tasarım yapılmalıdır. Bu tezde, her ne kadar veri tabanı tasarımı üzerinde durulmuş olsa da web tarafında yapılacak performans arttırıcı tasarımlar ile web uygulamaları hızlandırılabilir. Bir web uygulaması geliştirilirken tüm faktörler göz önüne alınıp sistemin bir çalışma tasarımı yapılmalıdır.

Sonuç olarak, hızla artmakta olan veri sayıları ile oluşan büyük verilerin istatistik değerlerinin yayınlandığı web sayfasında en hızlı nasıl sonuç alınabilir diye yapılan çalışmalarımızda iyi bir veri tabanı tasarımının hızlı bir sistem yarattığı gözlemlenmiştir. Uygulanan tüm yöntemlerin yayın ekleme süreleri karşılaştırıldığında kayıt sayısı az olduğunda çok fark gözükmese de kayıt sayısı arttıkça yalnızca yayın ekleme açısından en olumsuz performans denormalizasyon işleminden alınmıştır. Tüm çözüm yollarında sorgu sürelerinin ortalamaları karşılaştırıldığında ise, yayın eklemenin aksine en iyi performans sonucu denormalizasyon işleminden alınmıştır. Yapılan deney sonuçları ele alındığında istatistik verilerinin yayınlandığı bir site olan sistemimizde uygulamanın yapıldığı sistemde aynı anda çok sayıda veri ekleme işlemi yapılmayacağından dolayı kayıt ekleme süresi fazla olmayacaktır. Günümüzde kullanıcı için web sayfalarında hızın önemli olmasından dolayı denormalizasyon işlemi akademik veri sistemleri için en iyi sonuçlar verdiği görülmüştür.

Gelecek çalışmalar olarak üç konu üzerinde durmayı düşünüyoruz. Birinci konu, Kafka veya RabbitMQ veri işlem hatlarında ekleme, düzenleme ve silme performanslarını incelemedir. İkinci konu, farklı NoSQL veri tabanlarının test edilmesi ve tasarım içinde kullanılmasıdır. Üçüncü düşünülen konu ise, farklı problemler için web ortamını dikkate alan daha geniş bir tasarım yapma üzerinedir.

KAYNAKLAR

[1]R. Cattell, "Scalable SQL and NoSQL Data Stores," *SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011, doi: 10.1145/1978915.1978919.

[2]M. Rys, "Scalable SQL: How Do Large-Scale Sites and Applications Remain SQL-Based?," *Queue*, vol. 9, no. 4, pp. 30–37, Apr. 2011, doi: 10.1145/1966989.1971597.

[3]F. and P. J. and O. R. Vilaça Ricardo and Cruz, "An Effective Scalable SQL Engine for NoSQL Databases," in *Distributed Applications and Interoperable Systems*, 2013, pp. 155–168.

[4]N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," *Computer (Long Beach Calif)*, vol. 43, no. 2, pp. 12–14, 2010, doi: 10.1109/MC.2010.58.

[5]W. Vogels, "Eventually consistent," *Commun ACM*, vol. 52, no. 1, pp. 40–44, Jan. 2009, doi: 10.1145/1435417.1435432.

[6]E. J. Nailburg and R. A. Maksimchuk, *UML for Database Design*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[7]C. Babu and G. Gunasingh, "DESH: Database evaluation system with hibernate ORM framework," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2016, pp. 2549–2556. doi: 10.1109/ICACCI.2016.7732441.

[8]R. B. Miller, "Response Time in Man-Computer Conversational Transactions," in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, 1968, pp. 267–277. doi: 10.1145/1476589.1476628.

[9]S. K. Card, G. G. Robertson, and J. D. Mackinlay, "The Information Visualizer, an Information Workspace," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1991, pp. 181–186. doi: 10.1145/108844.108874.

[10] A. Davoudian, L. Chen, and M. Liu, "A Survey on NoSQL Stores," *ACM Comput. Surv.*, vol. 51, no. 2, Apr. 2018, doi: 10.1145/3158661.

[11] C. Anderson, "Docker [Software engineering]," *IEEE Software*, vol. 32, no. 3, pp. 102-c3, 2015, doi: 10.1109/MS.2015.62.

[12] K. Hightower, B. Burns, and J. Beda, *Kubernetes: Up and Running Dive into the Future of Infrastructure*, 1st ed. O'Reilly Media, Inc., 2017.

[13] E. F. Codd, "A relational model of data for large shared data banks," *Commun ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970, doi: 10.1145/362384.362685.

[14] E. F. Codd, "Further Normalization of the Data Base Relational Model.," May 1971.

[15] W. Kent, "A simple guide to five normal forms in relational database theory," *Commun ACM*, vol. 26, no. 2, pp. 120–125, Feb. 1983, doi: 10.1145/358024.358054.

[16] J. A. Hoffer, R. Venkataraman, and H. Topi, *Modern Database Management*, 10th ed. USA: Prentice Hall Press, 2010.

[17] G. L. Sanders, *Data Modeling*. International Thomson Publishing, 1995.

[18] T. J. Teorey *et al.*, *Database Design: Know It All*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[19] P. P.-S. Chen, "The entity-relationship model—toward a unified view of data," *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9–36, Mar. 1976, doi: 10.1145/320434.320440.

[20] C. Mohan, "History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla," in *Proceedings of the 16th International Conference on Extending Database Technology - EDBT '13*, 2013, pp. 11–16. doi: 10.1145/2452376.2452378.

[21] Paul Andlinger, "RDBMS dominate the database market, but NoSQL systems are catching up," *https://db-engines.com/en/blog_post/23*, Nov. 21, 2013.

[22] P. J. Sadalage and M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*, 1st ed. Addison-Wesley Professional, 2012.

[23] C.-H. Lee and Y.-L. Zheng, "SQL-to-NoSQL Schema Denormalization and Migration: A Study on Content Management Systems," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 2015, pp. 2022–2026. doi: 10.1109/SMC.2015.353.

[24] L.-Y. Ho, M.-J. Hsieh, J.-J. Wu, and P. Liu, "Data Partition Optimization for Column-Family NoSQL Databases," in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, Dec. 2015, pp. 668–675. doi: 10.1109/SmartCity.2015.146.

[25] G. Powell, *Beginning Database Design and Implementation (Chapter 8: Building Fast-Performing Database Models)*. GBR: Wrox Press Ltd., 2005.

[26] S. S. Lightstone, "Physical Database Design for Relational Databases," in *Encyclopedia of Database Systems*, M. T. LIU LING and ÖZSU, Ed. Boston, MA: Springer US, 2009, pp. 2108–2114. doi: 10.1007/978-0-387-39940-9_644.

[27] S. K. Shin and G. L. Sanders, "Denormalization Strategies for Data Retrieval from Data Warehouses," *Decis. Support Syst.*, vol. 42, no. 1, pp. 267–282, Oct. 2006, doi: 10.1016/j.dss.2004.12.004.

[28] G. L. Sanders and S. Shin, "Denormalization effects on performance of RDBMS," in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 2001, pp. 9 pp.-. doi: 10.1109/HICSS.2001.926306.

[29] G. Coleman, "Normalizing not only way," *Computerworld*, vol. 12, no. 1989, pp. 63–64, 1989.

[30] U. Rodgers, "Denormalization: why, what, and how," *Database Programming and Design*, vol. 2, no. 12, pp. 46–53, 1989.

[31] A. Balmin and Y. Papakonstantinou, “Storing and querying XML data using denormalized relational databases,” *The VLDB Journal*, vol. 14, no. 1, pp. 30–49, 2005.

[32] Y. Pinto, “A framework for systematic database de-normalization,” 2009.

[33] C. J. Date and C. J. Date, “Denormalization,” *Database Design and Relational Theory: Normal Forms and All That Jazz*, pp. 161–182, 2019.

[34] Ali Nizam, “Veri tabanı Tasarımı İlişkisel Veri Modeli ve Uygulamaları (Bölüm 9. Denormalizasyon),” in *Papatya Yayıncılık*, 2011, pp. 159–178.

[35] G. Karnitis and G. Arnicans, “Migration of Relational Database to Document-Oriented Database: Structure Denormalization and Data Transformation,” in *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*, 2015, pp. 113–118. doi: 10.1109/CICSyN.2015.30.

[36] T. Vajk, P. Fehér, K. Fekete, and H. Charaf, “Denormalizing data into schema-free databases,” in *2013 IEEE 4th International Conference on Cognitive Infocommunications (CogInfoCom)*, 2013, pp. 747–752. doi: 10.1109/CogInfoCom.2013.6719198.

[37] E. UZUN, H. N. BULUŞ, and C. ERDOĞAN, “Veri tabanı Tasarımının Yazılım Performansına Etkisi: Normalizasyona karşı Denormalizasyon,” *Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, vol. 22, no. 2, p. 887, Feb. 2018, doi: 10.19113/sdufbed.02548.

[38] E. Uzun, H. N. Buluş, C. Erdoğan, and H. Kaya, “İlişkisel Veri tabanlarında Denormalizasyon Etkisi: Bir Anket Uygulaması - Denormalization Effects on Relational Databases: A Survey Application,” 2016. [Online]. Available: https://erdincuzun.com/wp-content/uploads/download/ubmk_69_paper.pdf

[39] M. Taşyürek, “Regenerating Large Volume Vector Layers with a Denormalization-Based Method,” in *2021 6th International Conference on Computer Science and Engineering (UBMK)*, 2021, pp. 124–128. doi: 10.1109/UBMK52708.2021.9558893.

[40] M. Taşyürek, “Mekânsal verilerin sıklıkla güncellendiği coğrafi bilgi sistemleri arama işleminde denormalizasyon yöntemi,” *Avrupa Bilim ve Teknoloji Dergisi*, no. 24, pp. 18–23, 2021.

[41] D. Comer, “Ubiquitous B-Tree,” *ACM Computing Surveys*, vol. 11, no. 2, pp. 121–137, Jun. 1979, doi: 10.1145/356770.356776.

[42] T. Haerder and A. Reuter, “Principles of transaction-oriented database recovery,” *ACM Computing Surveys*, vol. 15, no. 4, pp. 287–317, Dec. 1983, doi: 10.1145/289.291.

[43] S. Gilbert and N. Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, Jun. 2002, doi: 10.1145/564585.564601.

[44] E. Brewer, “CAP twelve years later: How the ‘rules’ have changed,” *Computer (Long Beach Calif)*, vol. 45, no. 2, pp. 23–29, Feb. 2012, doi: 10.1109/MC.2012.37.