



# Teknoloji Fakültesi

**MARMARA ÜNİVERSİTESİ**  
**TEKNOLOJİ FAKÜLTESİ**  
**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**BİTİRME PROJESİ**

**Tezin Adı**

Veritabanı Yönetim Sisteminlerinde Sorgu Optimizasyon Yöntemleri

**Öğrencinin Adı Soyadı**

Yusuf Cem Şen

**DANIŞMAN**

Doç. Dr. Buket DOĞAN

**İSTANBUL, 2024**

## TEŞEKKÜR

Lisans eğitimimi tamamlamama imkân sağlayan Marmara Üniversitesi'ne, bu tezi hazırlamam için benden bilgisini, anlayışını ve zamanını hiçbir zaman esirgemeyen, verilebilecek en fazla desteği vererek yol gösteren danışman hocam Doç. Dr. Buket DOĞAN'a en içten teşekkürlerimi sunarım. Öğrenim hayatım süresince zaman zaman ihmal etmek zorunda kalmama rağmen, maddi ve manevi olarak beni sürekli destekleyen aileme de minnettarlığımı belirtmek isterim.

Mayıs 2024

Yusuf Cem Şen

## İÇİNDEKİLER

TEŞEKKÜR	i
İÇİNDEKİLER	ii
ÖZET	iii
ABSTRACT	v
ŞEKİL LİSTESİ	ix
TABLO LİSTESİ	ix
<b>1.Giriş</b>	12
1.1 Veri Tabanı Nedir?	13
1.1.1. Veri Tabanı Örnekleri	13
1.2 Veri Tabanı Türleri	13
1.2.1 Dağıtılmış Veri Tabanı	13
1.2.3 Merkezi Veri Tabanı	14
1.2.4 Kişisel Veri Tabanı	15
1.2.5 İlişkisel Veri Tabanı	16
1.2.6 Operasyonel Veri Tabanı	17
1.2.7. Hiyerarşik Veri Tabanı	17
1.2.8 Bulut Veri Tabanı	18
1.2.10 Nesne Yönelimli Veri Tabanı	18
1.2.11 NoSQL Veri Tabanı	18
1.2.12. Veri Tabanlarını Sınıflandırma Metotları	19
1.2.13. Bulut Veri Ambarları ve Veri Tabanları	20
1.3. Veri Tabanları Nasıl Çalışır	20
<b>2. Veri Tabanı Optimizasyonu</b>	21
2.1..Derinlemesine Optimizasyon Stratejileri	21
2.2. Veritabanı Optimizasyonunun Geleceği	22
2.2.1. Kaynak Yönetimi	23
2.2.2. CPU Yönetimi	23
2.2.3. Bellek Yönetimi	23
2.3. SQL Sorgu Optimizasyonunun Amacı ve Önemi	24
2.3.1. Optimizasyonun Amacı	24

2.3.2. Yöntemler	24
2.4.Uygulama Örneği	25
<b>3. VTYS (Veri Tabanı Yönetim Sistemleri) Sistemlerinde Yapılandırma Ayarlarının</b>	<b>22</b>
<b>Önemi</b>	
3.1. Depolama ve I/O Yapılandırmaları	26
3.2. Sorgu ve İşlem Yönetimi Ayarları	26
<b>4. Veritabanı Bakımı ve Optimizasyon</b>	<b>27</b>
4.1. Güvenlik ve Erişim Kontrolleri	27
4.2.Performans İzleme ve Analiz Araçları	28
4.3. Sonuç ve Öneriler	28
<b>5 .SQL Server: İlişkisel Veritabanı Yönetim Sistemi</b>	<b>29</b>
5.1. Veritabanı Yönetimi	29
5.2. Transact-SQL (T-SQL)	29
5.3. Yüksek Erişilebilirlik	29
5.4. Performans ve Ölçeklenebilirlik	30
5.5. Güvenlik	30
5.6. Entegrasyon ve Raporlama	30
5.7. Bulut Desteği	30
5.8. Windows İşletim Sistemi İçin SQL Server Kurulumu	31
5.8.1. Sistem Gereksinimlerinin Kontrol Edilmesi	31
5.8.2. SQL Server Kurulum Dosyasının İndirilmesi	32
5.8.3. Kurulum Programının Başlatılması	32
5.8.4. Kurulum Türünün Seçilmesi	32
5.8.5. Ürün Anahtarının Girilmesi	32
5.8.6. Özellik Seçimi	32
5.8.7. Örnek Yapılandırması	33
5.8.8. Sunucu Yapılandırması	33
5.8.9. Veritabanı Motoru Yapılandırması	33
5.8.10. Kurulumun Tamamlanması	34
5.8.11. SQL Server Management Studio (SSMS) Kurulumu	34

<b>6. Stack Overflow Veritabanı: Tanıtım ve Kullanım Amaçları</b>	35
<b>7. SQL Server'da Performans Ölçümü ve Sorgu Optimizasyonu</b>	35
7.1 Performans Analizi ve Optimizasyonu	36
7.2 Örnek Performans Ölçüm Senaryoları	37
<b>8. İndex Kullanımı</b>	38
8.1 İndekslerin Kullanım Amaçları	38
8.2. İndeks Türleri ve Kullanımı	38
8.3 İndeks Oluşturma	39
8.4 İndekslerin Bakımı ve Yönetimi	39
8.5. İndeks Kullanımının Stratejik Önemi	39
8.6 İndeks Kullanımı ve İndeks Olmadan SQL	
Sorgularının Performans Karşılaştırması	40
8.6.1 Genel Değerlendirme ve Sonuç	45
8.7. İndeks Kullanımının Avantajları ve Dezavantajları	48
<b>9. Fragmantasyon</b>	49
9.1. Veri Fragmantasyonu	49
9.1.2 İçsel Fragmantasyon (Internal Fragmentation)	49
9.1.3 Dışsal Fragmantasyon (External Fragmentation)	49
9.2. İndeks Fragmantasyonu	49
9.2.1. Lejyonel Fragmantasyon (Logical Fragmentation)	50
9.2.2. Sayfa Seviyesi Fragmantasyon (Page Level Fragmentation)	50
9.3. Fragmantasyonun Nedenleri	50
9.4. Fragmantasyonun Etkileri	50
9.5. Fragmantasyonun Tespiti	51
9.6. Fragmantasyonun Giderilmesi	51
9.7 Fragmantasyon Örnekleri	53
<b>10. View Nedir?</b>	61
10.1 View'ların Özellikleri ve Faydaları	61
10.2 View Kullanımının Dezavantajları	61
10 .4 View Kullanımı Örnekleri ve Açıklamaları	61

<b>11. Saklı Yordam (Stored Procedure) Nedir?</b>	66
11.1 Saklı Yordamların Performansa Olan Etkileri ve Örnekleri	66
11.1.1 Genel Değerlendirme	70
<b>12. Aynı Görevi Yerine Getiren Sorguların Daha Etkin Yazılması</b>	71
12.1 Örnekler ve Analizi	70
12.1.1. Genel Sonuç ve Değerlendirme	78
<b>13. Genel Değerlendirme ve Özet</b>	80
<b>Kaynakça</b>	82
<b>ÖZGEÇMİŞ</b>	84

## **Özet: Veritabanı Yönetim Sistemlerinde Sorgu Optimizasyon Yöntemleri**

Bu tez, farklı oranları optimize ederek veri tabanını daha iyi performans elde etmek için eldeki dosyaları tartışıyor. Bu çift pesquisa makalesi, dizinlerin, bölümlemenin, DBMS yapılandırma ayarlarının, sorgu önbelleğe alma stratejilerinin, görünümünün ve saklı prosedürlerin kullanılmasının bir veritabanının performansı üzerindeki derinlemesine etkisini arıyor. Bir veritabanı yönetim sisteminden en iyi şekilde yararlanmak isteniyorsa, bunların hepsinin doğru şekilde optimize edilmesi gerekir.

İndekslerin kullanımı, bir veritabanının performansını artırmak için en yaygın tekniklerden biridir. Dizin oluşturma, veri erişim hızını ve sorgu yanıt süresini yüksek oranda artırır. Öte yandan indekslerin kullanımı veritabanı güncellemeleri ve eklemeler üzerinde ek yüke neden olabilir. Bu tezde kümelenmiş ve kümelenmemiş dizinler gibi farklı dizin türlerinin bazı sorgu türleri üzerindeki etkileri incelenecek ve hangi koşullar altında hangi dizin türünün daha faydalı olduğu tartışılacaktır.

Veritabanı performansının kötüleşmesine neden olan bir diğer büyük sorun fragmentasyondur. Veri bloklarının düzensiz dağılımı sonucunda fragmentasyon hem sorgu performansını düşürmekte hem de depolama alanı kullanımını etkisiz hale getirmektedir. Bu araştırmada indeks yeniden düzenleme ve yeniden oluşturma tekniklerinin parçalanmanın etkilerini nasıl en aza indirebileceği tartışılmaktadır. Ayrıca düzenli izleme ve parçalanmanın ortadan kaldırılması için önerilen bazı yöntemler sunulmaktadır.

Bir DBMS'nin konfigürasyon ayarları, sistem kaynaklarının verimli kullanımı için çok önemlidir. Kaynakları maksimum bellek kullanımı, CPU ayarları ve disk G/Ç yapılandırması açısından optimize etmenin veritabanı performansı üzerinde doğrudan etkisi vardır. Bu tezde bellek yönetimi, paralel sorgu işleme ve tempdb yapılandırmasında gerçekleştirilebilecek iyileştirmelere bakacağım ve önerilerde bulunacağım. Bu ayarların doğru şekilde yapılandırılması, sistemin genel performansını artırırken kaynak kullanımını da optimize eder.

Sorgu önbelleğe alma stratejileri, sık kullanılan sorguların performansını artırmanın en iyi yoludur. Özellikle yüksek frekanslı bir sorgu için sorgu planlarının önbelleğe alınmasıyla önemli performans kazanımları elde edilebilir. Bu çalışmada sorgu önbelleğe alma stratejisinin nasıl geliştirilebileceği, sorgu planının yeniden kullanımının nasıl teşvik edilebileceği ve önbellek için en iyi boyutun nasıl optimize edilebileceği tartışılacaktır. Ayrıca önbelleklemenin veritabanı performansına katkıları örneklerle anlatılmıştır.

View'lar ve saklı yordamlar , veritabanı sorgularını basitleştirmek ve kodun yeniden kullanılabilirliğini artırmak için güçlü araçlardır. Bu tezde görünümünün karmaşık sorguları nasıl basitleştirip performansı iyileştirdiği, saklı yordamların sorgu performansını nasıl arttırdığı incelenmiştir. Görünümünün ve saklı yordamların kullanılmasının veritabanı yönetimi süreçlerine katkıları üzerinde durulmuştur.

Aynı görevi yerine getiren sorguların daha verimli yazılması, veritabanı performansını artırmak için kritik bir faktördür. Bu tezde, daha verimli sorgular yazmak için kullanılacak teknikler, örneğin, uygun JOIN kullanımı, filtreleme koşullarının optimize edilmesi ve doğru indeksleme stratejileri detaylı olarak ele alınmıştır. Sorgu optimizasyonu, veritabanı

performansını iyileştirmek için önemli bir alan olup, bu çalışmada pratik örneklerle desteklenmiştir.

Sonuç olarak, bu tez, veritabanı performansını optimize etmek için kapsamlı bir rehber sunmaktadır. İndeks kullanımı, fragmentasyon yönetimi, yapılandırma ayarlarının optimize edilmesi, sorgu önbellekleme stratejileri ve verimli sorgu yazma teknikleri gibi alanlarda derinlemesine analizler yapılmıştır. Bu stratejiler, veritabanı yönetim sistemlerinin performansını artırarak, daha verimli ve hızlı veri işleme süreçlerine katkıda bulunacaktır. Önerilen yöntemlerin gerçek dünya senaryolarında test edilmesi ve sürekli gelişen veritabanı teknolojilerine uyum sağlayacak şekilde güncellenmesi gerektiği vurgulanmaktadır.

Anahtar Kelimeler: Veritabanı Performansı, İndeks Kullanımı, Fragmentasyon, VTYS Yapılandırma Ayarları, Sorgu Önbellekleme, View, Saklı Yordamlar, Sorgu Optimizasyonu.



## **ABSTRACT : Query Optimization Methods in Database Management Systems**

In recent years, the exponential growth of data and the increasing complexity of database systems have made the optimization of database performance a critical area of focus. This thesis comprehensively examines key factors affecting database performance, including index usage, fragmentation, Database Management System (DBMS) configuration settings, query caching strategies, and the impact of views and stored procedures on performance. The primary objective is to develop strategies that enhance the efficiency and speed of database operations, thereby improving overall system performance.

**Index Usage:** Indexes significantly enhance data retrieval speeds and reduce query response times by providing a structured path to access data. However, the implementation of indexes may introduce additional overhead during data insertion and update operations. This study explores various types of indexes, such as clustered and non-clustered indexes, and their specific applications to optimize query performance.

**Fragmentation:** Fragmentation occurs when data is not stored contiguously, leading to increased I/O operations and degraded query performance. This research details methods to monitor and reduce fragmentation through techniques like index reorganization and rebuilding, thereby improving data access efficiency and storage utilization.

**DBMS Configuration:** Optimal configuration of DBMS settings is crucial for effective resource management. This thesis investigates the impact of key configuration parameters, including maximum memory allocation, CPU settings, and disk I/O optimization. Recommendations for improving memory management, parallel query processing, and tempdb configuration are provided to enhance system performance.

**Query Caching Strategies:** Query caching plays a vital role in accelerating frequently executed queries by storing execution plans in memory.

**Views and Stored Procedures:** Views simplify complex queries and enhance code reusability, while stored procedures improve execution efficiency by precompiling SQL statements. This research assesses the benefits of using views and stored procedures in database applications, highlighting their contributions to performance optimization.

**Query Optimization:** Writing efficient queries is essential for optimal database performance. This thesis offers techniques for optimizing queries, including effective use of JOIN operations, refined filtering conditions, and strategic indexing. Practical examples illustrate how these techniques can be applied to enhance query performance.

In conclusion, this thesis provides a comprehensive guide to optimizing database performance through targeted strategies in index usage, fragmentation management, DBMS configuration, query caching, and the use of views and stored procedures. The proposed methods aim to enhance the efficiency and speed of database systems, contributing to more effective and rapid data processing. Continuous evaluation and adaptation of these strategies in line with advancements in database technology are recommended to maintain optimal performance.

**Keywords:** Database Performance, Index Usage, Fragmentation, DBMS Configuration, Query Caching, Views, Stored Procedures, Query Optimization.

## ŞEKİL LİSTESİ

Şekil 1.1: Veri Tabanı Çalışma Akış Şeması	18
Şekil 8.1 Kullanıcıların Gönderi Puanlarının Ortalaması İndeksiz Kullanımı	43
Şekil 8.2 Genel Değerlendirme Grafiği	45
Şekil 9.1 Son erişim tarihlerine göre kullanıcıların sıralanması Fragmentasyon Tespiti	52
Şekil 9.2 Belirli bir tarih aralığında gönderilerin oluşturulması Fragmentasyon Gösterimi	54
Şekil 9.3 : IX_Posts_CreationDate İndeksi İçin Fragmentasyon Tespiti	57
Şekil 9.4 Etiketleri Olan Gönderiler için Fragmentasyon Tespiti	58
Şekil 10.1 Kullanıcı Bilgilerini Görüntüleme View Oluşumu	61
Şekil 10.2 Kullanıcı Bilgilerini Görüntüleme View Performans Analizi	62
Şekil 10.3 Karmaşık View: Gönderi ve Kullanıcı Bilgilerini Birleştirme	62
Şekil 10.4 Karmaşık View: Gönderi ve Kullanıcı Bilgilerini Birleştirme Performans Ölçümü	63
Şekil 10.5 View ile Veri Güvenliği: Hassas Kullanıcı Bilgilerini Gizleme Performans Ölçümü	64
Şekil 11.1 Basit Saklı Yordam: Kullanıcı verilerini toplama İstatistikleri	66
Şekil 11.2 Karmaşık Saklı Yordam: Gönderi ve Kullanıcı Bilgilerini Birleştirme Performansı	67
Şekil 11.3 Veri Güncelleme: Kullanıcı Bilgilerini Güncelle Performans	68
Şekil 11.4 Saklı Yordam Genel Değerlendirme Grafiği	69
Şekil 12 .1 Tarih Aralığına Göre Gönderileri ve Kullanıcı Detaylarını Alma Optimizasyonu	71
Şekil 12.2 Gönderi ve Kullanıcı Detayları ile Yorumları Alma	72
Şekil 12.3 Top Kullanıcıları İtibarıyla Alma Optimize	73
Şekil 12.4 Kullanıcıya Göre Skorla Gönderileri Alma Optimizasyonu	74
Şekil 12.5 4 Aylık Gönderi Sayıları Alma Optimizasyonu	75

## **TABLO LİSTESİ**

<b>Tablo 8.1</b> Kullanıcıların Son Erişim Tarihlerine Göre Sıralanması	40
<b>Tablo 8.2</b> : Belirli Bir Tarih Aralığında Oluşturulan Gönderiler	41
<b>Tablo 8.3</b> Kullanıcıların Yaptığı Yorum Sayıları	42
<b>Tablo 8.4</b> Belirli Bir Etikete Sahip Gönderiler	43
<b>Tablo 8.5</b> Kullanıcıların Gönderi Puanlarının Ortalaması	45
<b>Tablo 11.1</b> Saklı Yordam Genel Değerlendirme	70
<b>Tablo 12.1</b> Aynı Görevi Yerine Getiren Sorguların Daha Etkin Yazılması	77

# 1.Giriş

Veritabanı performansı, artan veri hacmi ve karmaşıklığı ile birlikte günümüzün dijital dünyasında büyük bir öneme sahiptir. Verimli bir veritabanı yönetim sistemi (VTYS) oluşturmak, işletmelerin veri işleme sürelerini kısaltmak ve kaynak kullanımını optimize etmek için kritik hale gelmiştir. Bu çalışma, indeks kullanımı, fragmentasyon yönetimi, VTYS yapılandırma ayarları, sorgu önbellekleme stratejileri, view ve saklı yordamların veritabanı performansı üzerindeki etkilerini araştırmayı amaçlamaktadır. Amacımız, bu faktörlerin her birini detaylı bir şekilde inceleyerek, veritabanı performansını artırmak için kullanılabilecek etkili stratejiler geliştirmektir. Böylece, veritabanı sistemlerinin daha hızlı, verimli ve güvenilir bir şekilde çalışmasını sağlamak hedeflenmektedir.

## 1.1 Veri Tabanı Nedir?

Veri tabanı, organize edilmiş bir bilgi koleksiyonu olarak tanımlanır ve verilerin daha kolay aranmasını, alınmasını, manipüle edilmesini ve analiz edilmesini kolaylaştırır. İş dünyasında, satış, İK, pazarlama, müşteri hizmetleri gibi bir dizi farklı gereksinimi karşılamak için veri tabanları gereklidir. Bu veri tabanları genellikle benzer konularda veya benzer veri türlerini saklamak için kullanılır. Veri tabanları, eldeki görev için en uygun veri düzenlemesini veya yapılandırmayı sağlamak için farklı şemaları kullanır.

Bir veri tabanı, bir veri tabanı programlama dili kullanılarak oluşturulur ve muhafaza edilir. En yaygın olarak bilinen veri tabanı dili SQL'dir; ancak, Yapılandırılmış Sorgu Dili'nin (SQL) farklı çeşitleri vardır. Her bir SQL türü, SQL dil yapısında zıtlıklara sahiptir ve belirli bir veri tabanı türü ile kullanılmak üzere tasarlanmıştır. [1]

İşletmeler, çeşitli iş kararları almak için kullanılabilecek verileri depolamak için veri tabanlarını kullanır. Veri tabanlarının çoğu aşağıdaki özelliklere sahiptir. Verileri farklı uygulamalardan izole eder. Verilerin aynı anda birden fazla kullanıcı tarafından paylaşılmasını destekleyerek aynı anda birden fazla işlem yapılmasını sağlar. Aynı verinin birden fazla görünümünü destekler. Verilerin güvenliğini sağlar. İşletmelerin gelirlerini artırmalarına yardımcı olarak işlerini geliştirmelerine yardımcı olur.

Bir veri tabanı aşağıdaki parçalardan oluşur:

**Şema:** Bir veri tabanı, en az bir bilgi tablosunun bir araya getirilmesiyle oluşturulan en az bir kompozisyonu temsil eder.

**Tablo:** Her tablo, tıpkı bir elektronik tabloda olduğu gibi çeşitli sütunlar içerir. Bir tabloda, tabloya konulan bilginin türüne bağlı olarak sütun sayısı artabilir.

**Sütun:** Her sütun isim, adres, telefon numarası gibi birkaç tür bilgidен birini içerir.

**Satır:** Bir tablodaki veriler satırlar halinde kaydedilir. Bir tabloda belirli bir bilgiye sahip yüzlerce veya binlerce satır vardır.

### 1.1.1 Veri Tabanı Örnekleri

Piyasada, uygulama ve iş kullanımına bağlı olarak farklı veri tabanları bulunmaktadır. Aşağıda, popüler SQL ve NoSQL veri tabanlarından bazılarını bulabilirsiniz:

#### İlişkisel Veri Tabanı

- Oracle database
- Microsoft SQL Server
- MySQL database
- PostgreSQL
- Redis
- IBM DB2
- MySQL
- İlişkisel Olmayan Veri Tabanları
- MongoDB
- Apache Cassandra
- Apache CouchDB
- Apache HBase

### 1.2 Veri Tabanı Türleri

Aşağıda en çok kullanılan veri tabanı türlerinden bazılarını bulabilirsiniz:

#### 1.2.1 Dağıtılmış Veri Tabanı

Benzer bir sistemde veya benzersiz sistemlerde çeşitli hedeflerde bulunan en az iki belgeden oluşan bir veri tabanıdır. Veri tabanının parçaları çeşitli fiziksel yerlere yerleştirilir ve kullanım farklı veri tabanı merkezleri arasında dağıtılır.

Dağıtık veri tabanları fiziksel olarak birden fazla yerde depolanır ve mantıksal olarak birbirleriyle bağlantılıdır ve genellikle tek bir mantıksal veri tabanını ifade eder.

Dağıtık veri tabanları homojen veya heterojen olabilir. Genel olarak, dağıtık veri tabanları aşağıdaki özellikleri içerebilir:

- Donanımdan bağımsız
- Konumdan bağımsız
- İşletim sisteminden bağımsız
- Ağdan bağımsız
- İşlem şeffaflığı
- Dağıtılmış sorgu işleme
- Dağıtılmış işlem yönetimi

Dağıtık veri tabanı örnekleri:

- Apache Cassandra
- Apache HBase
- Apache Ignite
- Couchbase Server
- Amazon SimpleDB
- FoundationDB
- Clusterpoint [2]

Dağıtık Veri Tabanının Avantajları ise şunlardır.Yöneticiler, dağıtık veri tabanı çerçeveleri kullanırken bilginin en sık kullanıldığı yere yakın bir konumda bulundurulmasının, iletişim maliyetlerini azaltmada daha etkili olabileceğini fark edebilir. Merkezi veri tabanlarında ise bu tür bir yakınlık sağlamak mümkün değildir.

Güncellenmesi ve büyümesi kolaydır.

Tutarlılığı ve kullanım kolaylığını artırır.

Daha iyi performans sağlar.

Bir yerden veri tabanı kaybı tüm verilerin kaybına neden olmayabilir.

Yük dengeleme için etkilidir ve gecikmeyi azaltır.

Dağıtık veri tabanı hata toleransı özelliğine sahiptir

Dağıtılmış Veri Tabanının Dezavantajları

Bütünlüğün uygulanması daha fazla ağ kaynağı gerektirebilir.

Mimarisi daha iyi tasarım, yönetim ve sorun giderme gerektirdiğinden daha karmaşıktır.

Bazen ana veri tabanından erişim sırasında sorunlarla karşılaşılabilir.

Dağıtılmış veri tabanında bir başka endişe kaynağı da güvenlidir.

### 1.2.3 Merkezi Veri Tabanı

Merkezi bir veri tabanı çerçevesi, bilgileri tek bir yerde tek bir veri tabanında tutan bir çerçevedir.

Merkezi Veri Tabanının Avantajları:

- Merkezi veri tabanı depolaması veri güvenliğini artırır.
- Yerel olarak depolanan veriler devam eden bir fiziksel güvenlik tehlikesi anlamına geldiğinden, merkezi veri tabanı fiziksel güvenlik sağlar.
- Merkezi depolamanın bakımı, birden fazla depolama alanına göre daha az maliyetlidir.
- Piyasa analistleri arasında fikir paylaşımı kolaydır.
- Merkezileştirme nedeniyle çatışmaları azaltır.

- Merkezi veri tabanı kurumlar arasındaki çatışmaları azaltır ve kurumlar vizyonlarına odaklanıp hızlıca harekete geçebilir.
- Merkezi veri tabanında veri fazlalığı ihmal edilebilir düzeydedir.

#### Merkezi Veri Tabanının Dezavantajları

- Merkezi veri tabanı, ağır iş yükü altında çalışmaya tepkisiz kalabilir.
- Bilgi en çok kullanıldığı yerde bulunamazsa, yöneticiler yüksek iletişim maliyetleriyle karşı karşıya kalabilir.
- Merkezi veri tabanında bilgiler tek bir veri tabanında saklanır, bu nedenle veri kaybı olasılığı artabilir.
- Uygun veri tabanı kurtarma önlemleri yoktur.

### 1.2.4 Kişisel Veri Tabanı

Bilgi, az miktarda olan ve kolayca yönetilebilen bilgisayarlarda toplanır ve depolanır.

Bu bilgiler genellikle bir işletmenin aynı departmanı tarafından kullanılır ve az sayıda kişi tarafından kolayca erişilebilir.

Genel olarak, bir kişisel veri tabanı sistemi bir seferde bir uygulamayı destekleyebilir, tek bir bilgisayar, birkaç tabloya sahip bir veri tabanı içerir.

#### Kişisel Veri Tabanının Avantajları:

- Veri paylaşımını ve güvenliğini artırır.
- Daha iyi veri entegrasyonu ve hızlı veri erişimi sağlar.
- Hızlı son kullanıcı üretkenliği ve gelişmiş karar verme sağlar.
- Kişisel Veri Tabanının Dezavantajları

Yöneticiler, lisanslama gereksinimleri ve veri tabanı kullanıcılarının uygun şekilde eğitilmesi gibi nedenlerle artan maliyetlerle karşılaşabilirler. Bu süreç aynı zamanda yeni özellikleri kullanmayı ve yönetmeyi içerebilir.

Kişisel veri tabanı ağır iş yükü altında çalışmaya tepkisiz kalabilir.

### 1.2.5 İlişkisel Veri Tabanı

İlişkisel bir veri tabanı, verilere erişilebilen bir dizi tablo ile tanımlanır.

İlişkisel veri tabanı, birbiriyle bağlantılı bir dizi tabloda büyük miktarda bilgi depolayabilir.

Her tablo, her sütunun ad, adres gibi belirli bir bilgi türünü temsil ettiği, her satırın benzersiz bilgiler içerdiği ve bir tablodaki her alanın kendi veri türüne sahip olduğu satır ve sütunlardaki bilgilerden oluşur.

İlişkisel Veri Tabanının Avantajları

- İlişkisel veri tabanı, herhangi bir şirketin finansal kaydını saklamak için kullanılır
- Müşterilerin sevkiyatlarının ve siparişlerinin kayıtlarını tutar.
- İlişkisel veri tabanı veri bütünlüğü ve daha iyi performans sağlar.
- Daha iyi veri güvenliği sağlar ve birden fazla kullanıcıya izin verir.
- Kolayca genişletilebilen ve değiştirilebilen ilişkisel veri tabanında büyük miktarda veri depolanabilir.
- Birkaç müşteri aynı veri tabanına erişebilir.
- İlişkisel Veri Tabanının Dezavantajları

- Kurulum ve bakım maliyeti yüksektir.
- İlişkisel veri tabanları için sofistike ağ ve donanım kurulumları gereklidir.
- İlişkisel veri tabanlarında genellikle alan uzunlukları sınırlıdır.

### 1.2.6 Operasyonel Veri Tabanı

Operasyonel bir veri tabanı, büyük miktarda veriyi gerçek zamanlı olarak depolamak ve yönetmek için kullanılır. Herhangi bir projenin operasyonları (pazarlama, müşterilere sağlanan hizmetler ve onlarla ilişkiler) ile ilgili veriler operasyonel bir veri tabanında saklanabilir.

Operasyonel Veri Tabanının Avantajları:

- Veriler gerçek zamanlı olarak eklenebildiği ve değiştirilebildiği için gerçek zamanlı işlemler sağlar.
- Operasyonel veri tabanı güncel bilgi sağlar.
- Gerçek zamanlı hesaplama ve analiz süreçleri için veri sağlar.
- İşletmenin günlük işleyişini yürütmeye yardımcı olur.
- Operasyonel Veri Tabanının Dezavantajları
- Gerçek zamanlı analiz, veri tabanı kullanıcıları için özel eğitim oturumları gerektirir ve bu da şirketlere daha fazla maliyet yükler.
- Operasyonel verilere yönelik sorgular genellikle dar kapsamlıdır, yani hız açısından kritiktir.



### 1.2.7. Hiyerarşik Veri Tabanı

Hiyerarşik veri tabanı modelinde veriler, bir dizi farklı ögeyi bir üst kayda bağlayan bir ağaç yapısında düzenlenir.

Her kayıt türünün yalnızca bir ebeveyni vardır. Ağacın en üst ögesi ebeveyn, dalları ise children olarak adlandırılır. Belirli veri depolama türleri için kullanışlıdır.

Bir varlık türü bir tabloya karşılık gelir ve varlık türleri birbirleriyle bire çok ilişkilendirme ile ilişkilidir.

#### Hiyerarşik Veri Tabanının Avantajları

- Hiyerarşik veri tabanlarının anlaşılması kolaydır.
  - Hiyerarşik yapıda açık emir komuta zinciri bulunur.
  - Hiyerarşik veri tabanında, ebeveyn ve çocuk ilişkileri bir veri kaydından diğerine işaretçilerle uygulandığından kayıtlar arasında gezinmek yeterince hızlıdır.
  - Hiyerarşik Veri Tabanının Dezavantajları
- 
- Hiyerarşik yapıların yeni değişiklikleri hızla benimseme eğilimi yoktur.
  - İletişim engelleri vardır.
  - Veri tabanı, mevcut iş gereksinimlerini karşılayacak şekilde değiştirilemez.

### 1.2.8 Bulut Veri Tabanı

Bulut veri tabanı, kuruluşların ve uygulamalarının buluttan bilgi depolamasına ve yönetmesine izin veren Platform-as-a-Service (PaaS) gibi bir bulut platformu aracılığıyla dağıtılan ve sunulan bir tür veri tabanı yönetimidir.

Günümüzde, kullanıcının ödeme kabiliyetine göre daha fazla depolama kapasitesi, yüksek bant genişliği, ölçeklenebilirlik ve kullanılabilirlik elde etme gibi çeşitli avantajlar nedeniyle popülerlik kazanmıştır.

Bir bulut veri tabanı düzenli olarak, genellikle bir kayıt / çerçeve bulutu üzerinden veri tabanı programlamasının kurulmasıyla yürütülen standart bir veri tabanı düzenlemesi olarak işlev görür.

Buna ek olarak, bir bulut veri tabanı aynı şekilde, satıcının özellikle veri tabanı kurulumu ve dağıtımının arka uç prosedürleri ile ilgilendiği bir yönetim olarak aktarılır.

#### Bulut Veri Tabanının Avantajları

- Bulut veri tabanı, veri tabanı iş yükünü geleneksel olarak veya bir hizmet olarak çalıştırmak için büyük esneklik sağlar.
- Felaket kurtarma için faydalıdır.
- Şirketler bulut depolamayı kullanarak yıllık işletme maliyetlerini azaltabilir.
- İnternet üzerinden verilere kolay erişim sağlar.
- Bulut Veri Tabanının Dezavantajları
- Veri koruma, bulut veri tabanında büyük bir zorluktur.

- İnternet bağlantısının olmaması durumunda erişimin zordur.
- Bir belgeyi bulut depolama klasörüne taşımak için sürükle ve bırak eylemi sırasında daha bilinçli olmak gerekir.

### 1.2.10 Nesne Yönelimli Veri Tabanı

Eylemler ve mantık yerine nesne etrafında organize olan nesne yönelimli programlama ve ilişkisel veri tabanı grubudur.

Nesne yönelimli veri tabanı motorlarına bazı örnekler Smalltalk, db4o ve Cache'dir. Nesne yönelimli veri tabanında bilgi nesneler biçiminde tanımlanır. Benzer nesneler sınıflara ve alt sınıflara ayrılır ve iki nesne arasındaki ilişki zıt referans kavramı kullanılarak sağlanır.

Nesne Yönelimli Veri Tabanının Avantajları

- Veri tabanlarının, işletim sisteminin, kelime işlemcilerin, elektronik tabloların ve diğer uygulamaların entegrasyonuna izin verir.
- Ürün ve uygulamaların referans paylaşımını sağlar.
- Mevcut bir nesneden kolayca yeni bir nesne oluşturur.
- Nesne Yönelimli Veri Tabanı Yönetim Sistemi, eşzamanlılık ve kurtarma sorununu çözmek için kalıcı nesneler oluşturmaya izin verir.
- 

Nesne Yönelimli Veri Tabanının Dezavantajları

- Büyük ADT'lerin ve yapılandırılmış nesnelerin depolanmasını sağlar.
- Sorgu işleme ve optimizasyon karmaşık bir iştir.
- İlişkisel veri tabanından çok daha karmaşıktır, bu nedenle deneyimsiz son kullanıcılar yerine profesyonel programcılar gereklidir.
- Kullanıcı bireysel nesneler ve sınıflar üzerinde erişim haklarına izin veremez.

### 1.2.11 NoSQL Veri Tabanı

NoSQL veri tabanı, birkaç sanal sunucuda depolanabilecek büyük bir dağıtılmış veri kümesini verimli bir şekilde yönetmek ve analiz etmek için kullanılır. NoSQL veri tabanı türleri aşağıdakileri içerir:

Şu anda birçok NoSQL veri deposu mevcuttur, bunlardan bazıları MongoDB, CouchDB, GemFire, Casandra, Hbase, Mnesia, Memcached ve Neo4j'dir.

NoSQL Veri Tabanının Avantajları

- Kullanımı kolay ve esnektir.
- Monolitik mimarinin yerine coğrafi olarak dağıtılmış ölçeklendirme mimarisine sahiptir.
- Sık kod aktarımı ve hızlı şema yinelemesi sağlar.
- Daha iyi performans ve yüksek küresel kullanılabilirlik sunar.
- Esnek veri modellemesi sunar.
- Büyük hacimler NoSQL veri tabanları tarafından kolayca işlenebilir.[3]

NoSQL Veri Tabanının Dezavantajları:

- Kurulumu ve bakımı için ileri düzeyde uzmanlık gereklidir.
- En yaygın iş zekası araçları NoSQL veri tabanlarına bağlantı sağlamaz.
- SQL ve NoSQL Veri Tabanları

### 1.2.12. Veri Tabanlarını Sınıflandırma Metotları

Veri tabanlarını sınıflandırmanın belki de en yaygın yolu SQL ve NoSQL'dir (ilişkisel ve ilişkisel olmayan olarak da bilinir).

Bir SQL veri tabanı Yapılandırılmış Sorgu Dili kullanır ve bir tür ilişkisel veri tabanıdır. İlişkisel veri tabanları, bilgileri farklı veri parçaları arasındaki ilişkileri kodlayan resmi tablolar içinde düzenler. Her tablo, Microsoft Excel'deki bir elektronik tablonun yapısına benzer şekilde sütun ve satırlar içerir. İlişkisel bir veri tabanı kullanırken, veri tabanınızdaki veriler arasındaki ilişkileri tanımlayan kavramsal, mantıksal veya fiziksel bir şema oluşturabilirsiniz.

İlişkisel bir veri tabanında arama yapmak için kullanıcılar, veri tabanları ile iletişim kurmak için alana özgü bir dil olan Yapılandırılmış Sorgu Dili'nde (SQL) sorgular yazar.

Öte yandan, NoSQL veya ilişkisel olmayan veri tabanı, ilişkisel tablo tabanlı veri modelinin dışında kalan verileri depolamak için herhangi bir paradigma kullanır. NoSQL veri tabanları dinamik şema kullanır ve bu da işletmelere veri depolamak ve verilere erişmek için daha esnek bir yol sağlayabilir.

Bazı yaygın NoSQL veri tabanı türleri anahtar-değer, belge tabanlı, sütun tabanlı ve grafik tabanlı depolardır. Popüler NoSQL teklifleri arasında MongoDB, Cassandra ve Redis bulunmaktadır.

Her iki yaklaşımın da artıları ve eksileri vardır. SQL veri tabanlarının daha fazla kaynak ekleyerek dikey olarak ölçeklendirilmesi daha kolay olurken, NoSQL veri tabanlarının yatay olarak (daha fazla makine ekleyerek) ölçeklendirilmesi daha kolay olma eğilimindedir. Sorgu yazmak için SQL kullanımı performans ve kullanım kolaylığı açısından önemli bir avantaj olabilir, ancak ilişkisel veri tabanları da veri hiyerarşisi açısından daha az esnek ve daha katıdır.

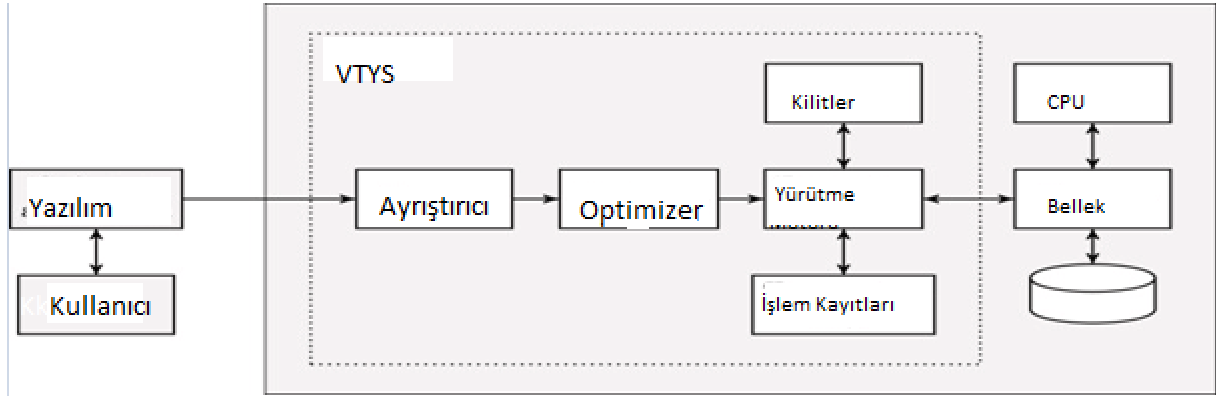
### 1.2.13. Bulut Veri Ambarları ve Veri Tabanları

Bazı bulut veri tabanları SQL ve NoSQL özelliklerinin bir karışımını sunar. Örneğin Amazon Redshift, büyük ölçekli veri setlerini hızlı bir şekilde taşıyabilen bir çözüm isteyen bir veri ambarı şirketi tarafından geliştirilen teknoloji üzerine inşa edilmiştir. Bu da onu bir NoSQL veri tabanına benzetir; ancak bulut tabanlı bir veri ambarı çözümü olan Redshift, Postgres uyumlu bir sorgu katmanına da sahiptir. Redshift verileri ilişkisel şemaya göre düzenleyebilir, bu da onu bir SQL veri tabanına benzetir.

Geleneksel olarak, işletmelerin bir veri tabanı barındırmak için yerinde ekipman ve altyapı bulundurması gerekirdi. Bunu yapmak, yalnızca donanımınızın kaldırabileceği alan miktarına erişebileceğiniz anlamına gelir. Bunun da ötesinde, ekipman yıprandığında veya operasyonel sistemler yedekli hale geldiğinde, maliyetin işletme tarafından karşılanması gerekir. Bulut veri tabanları o kadar fazla alana sahiptir ki, pratikte sınırsız olarak ölçeklendirebilirsiniz. Sözleşme anlaşmanıza bağlı olarak, aşırı ücretler ödmeden gerektiği gibi ölçeklendirme yapabileceğinizi görebilirsiniz.

Hem veri ambarı hem de veri tabanı fiyatları hizmetten hizmete önemli ölçüde değişebilir, bu nedenle bulut tabanlı bir veri yönetimi sağlayıcısı seçmeden önce seçeneklerinizi karşılaştırdığınızdan emin olmanız gerekir. [4]

### 1.3. Veri Tabanları Nasıl Çalışır



Şekil 1.1: Veri Tabanı Çalışma Akış Şeması

Görselde, bir veritabanı yönetim sistemi (DBMS) içindeki iş akışını gösteren basitleştirilmiş bir şematik diyagram bulunmaktadır. Kullanıcı tarafından bir yazılım uygulaması aracılığıyla gönderilen istekler öncelikle parser (ayrıştırıcı) tarafından analiz edilir. Ayrıştırıcı, sorgunun doğru formatta olup olmadığını kontrol eder ve sorgu yapısını anlar. Ardından optimizör (optimize edici), sorguyu en verimli şekilde nasıl yürüteceğini belirlemek için sorgu planları oluşturur.

Execution engine (yürütme motoru), optimize edilen sorguyu yürütür ve veritabanındaki kilit mekanizmalarını yönetir. İşlemler sırasında CPU ve bellek kaynakları kullanılır. Tüm işlem ve değişiklikler transaction logs (işlem kayıtları) üzerinde saklanır, böylece veritabanı güvenilirliği ve geri dönüşümü sağlanır. Bu işlemler, veritabanının hızlı ve güvenli bir şekilde çalışmasını sağlamak için birlikte çalışır.

## 2. Veri Tabanı Optimizasyonu

Veritabanı sorgu optimizasyonu, veritabanı sorgularını daha verimli ve daha hızlı hale getirme işlemidir. Bu amaca ulaşmak için çeşitli teknikler ve stratejiler içerir. Bu optimizasyonlar veritabanı yöneticileri veya geliştiricileri tarafından yapılabilir ve genellikle aşağıdaki adımları içerir:

**Sorgu Analizi:** Tablo yapıları, dizinler, veri hacmi ve sorgu yapısı gibi sorgu performansını etkileyen faktörlerin analiz edilmesi.

**İndeksleme:** Sık erişilen sütun ve sorgulara uygun indekslerin oluşturulması. İndeksler verilere hızlı erişim sağlar.

**İstatistik Toplama:** Sorguların veriler üzerinde nasıl performans gösterdiğine ilişkin istatistiklerin düzenli olarak toplanması. Bu istatistikler sorgu planlayıcının daha iyi sorgu planları yapmasına yardımcı olur.

**Sorgu Yapısının Gözden Geçirilmesi:** Çoğunlukla performans sorunlarına yol açan karmaşık sorgu yapılarının basitleştirilmesi veya yeniden yapılandırılması.

**Veri Depolama Yapısının Optimize Edilmesi:** Veri depolama yapılarının optimize edilmesi için veritabanı tasarımının gözden geçirilmesi. Normalizasyon ve denormalizasyon gibi teknikler kullanılabilir.

**Sorgu Planlama ve Yürütme:** Veritabanı yönetim sistemleri, sorguları optimize etmek için çeşitli planlama ve yürütme stratejileri kullanır. Bu stratejiler veritabanı yöneticileri tarafından yapılandırılabilir veya otomatik olarak belirlenebilir.

Bu adımlar, veritabanı sorgularının daha hızlı ve daha verimli çalışmasını sağlamak için bir araya getirilerek uygulama performansının ve kullanıcı deneyiminin iyileştirilmesi sağlanır.

### 2.1. Derinlemesine Optimizasyon Stratejileri

Optimizasyon süreci, sorgu planlarının seçimi ve iyileştirilmesiyle sınırlı değildir. Ayrıca, veritabanı yapılandırmasının optimizasyonu, veri modelleme teknikleri ve fiziksel veri organizasyonu gibi daha geniş konuları da içerir. İndeksleme stratejileri, sorgu performansını dramatik bir şekilde iyileştirebilir. Doğru alanlarda indekslerin oluşturulması, sorgu işlem sürelerinin kısaltılmasına ve dolayısıyla sistem kaynaklarının daha verimli kullanılmasına olanak tanır.

Ayrıca, veritabanı önbellekleme mekanizmaları, sık erişilen verilerin hızlı bir şekilde erişilebilir tutulmasını sağlar. Bu, özellikle tekrar eden sorguların işlenmesinde, veritabanı performansını önemli ölçüde artırabilir.

## Eşzamanlılık Kontrolünde İleri Teknikler

Eşzamanlılık kontrolü ve kilit yönetimi, veritabanı sistemlerinde çözülmesi gereken karmaşık problemlerdir. Geleneksel kilit mekanizmalarının yanı sıra, zaman damgası tabanlı teknikler, çok seviyeli kilit sistemleri ve optimistik kilit mekanizmaları gibi ileri teknikler de bu alanda kullanılmaktadır. Bu teknikler, sistem performansını ve ölçeklenebilirliğini artırırken, eşzamanlı erişimden kaynaklanan çakışmaları minimuma indirmeyi amaçlar.

## Veri Bütünlüğü ve Güvenliği

Veritabanı optimizasyonunun bir diğer önemli yönü, veri bütünlüğü ve güvenliğidir. Veri bütünlüğü kısıtlamaları ve güvenlik politikaları, veritabanı tasarımı ve yönetiminde merkezi bir rol oynar. Veri bütünlüğünü korumak, yanlış veya eksik veri girişlerinin önlenmesini sağlar, bu da sorgu optimizasyonu ve performans üzerinde dolaylı bir etkiye sahiptir.

Veritabanı sistemlerinde SQL sorgu işleme ve optimizasyonu, veri yönetimi ve erişim performansının temelini oluşturur. Bu süreç, sorgu alımından başlayarak, ayrıştırma, optimizasyon, yürütme ve sonuçların geri dönüşüne kadar uzanır. Ayrıca, eşzamanlılık kontrolü, kilit yönetimi, işlem kayıtları ve kurtarma mekanizmaları gibi ilave yönetim ve güvenlik önlemleri, veritabanı sistemlerinin etkinliği ve güvenliği açısından büyük önem taşır. Bu çalışma, SQL sorgu işleme ve optimizasyon sürecinin çeşitli yönlerini derinlemesine incelemekte ve veritabanı yönetim sistemlerinin performansının artırılması için kullanılan stratejileri detaylandırmaktadır. İleri seviye optimizasyon teknikleri ve eşzamanlılık kontrol mekanizmaları, veritabanı sistemlerinin karşılaştığı zorlukların üstesinden gelmekte kritik bir rol oynar. Bu teknikler, sistem kaynaklarının daha verimli kullanılmasını, yanıt sürelerinin kısalmasını ve veritabanı güvenliğinin ve bütünlüğünün sağlanmasını hedefler. [5]

## **2.2. Veritabanı Optimizasyonunun Geleceği**

Teknolojinin hızla ilerlemesi ve veri miktarlarının katlanarak artması, veritabanı sistemlerinin sürekli olarak evrimleşmesini gerektirmektedir. Bulut bilişim, dağıtık veritabanları, büyük veri teknolojileri ve yapay zeka, veritabanı yönetimi ve optimizasyonunda yeni yaklaşımların geliştirilmesine öncülük etmektedir. Özellikle, yapay zeka ve makine öğrenimi algoritmalarının optimizasyon süreçlerine entegre edilmesi, sorgu performansının otomatik olarak iyileştirilmesine olanak tanımaktadır.

Ayrıca, NoSQL veritabanları gibi yeni veritabanı modelleri, geleneksel SQL tabanlı sistemlere alternatifler sunmakta ve farklı veri türleri ve erişim desenleri için optimize edilmiş çözümler sağlamaktadır. Bu yeni yaklaşımlar, veritabanı optimizasyonunun geleceğini şekillendirmede önemli bir role sahiptir. Sonuç olarak, SQL sorgu işleme ve optimizasyonu, veritabanı sistemlerinin temel bir bileşenidir ve veri erişimi ve yönetimi üzerinde derin bir etkiye sahiptir. Sorgu optimizasyonu, kullanıcı deneyimini iyileştiren, veritabanı performansını artıran ve sistem kaynaklarını daha etkin bir şekilde kullanmaya olanak tanıyan kritik bir süreçtir. Bu çalışma, veritabanı yönetimi ve optimizasyonunun çeşitli yönlerini kapsamlı bir şekilde incelemekte ve bu alanda kullanılan ileri teknolojileri ve stratejileri vurgulamaktadır. Veritabanı yönetimi alanında yapılacak gelecek çalışmalar, veri yönetimi teknolojilerinin

gelişimine katkıda bulunmaya ve veritabanı sistemlerinin performansını daha da iyileştirmeye devam edecektir.

### **2.2.1. Kaynak Yönetimi**

Veritabanı yönetim sistemleri (DBMS), büyük miktarda veriyi depolamak, işlemek ve yönetmek için tasarlanmış sofistike yazılımlardır. Bu sistemlerin etkin bir şekilde çalışabilmesi için donanım kaynaklarının verimli kullanılması hayati öneme sahiptir. CPU, bellek ve disk gibi donanım kaynaklarının yönetimi, veritabanı performansını doğrudan etkiler. Bu bölümde, SQL Server gibi bir veritabanı yönetim sisteminin donanım kaynaklarını nasıl yönettiği ve bu kaynakları en iyi şekilde kullanarak nasıl yüksek performans sağladığı ayrıntılı olarak ele alınacaktır.

### **2.2.2. CPU Yönetimi**

CPU (Merkezi İşlem Birimi), veritabanı işlemlerinin gerçekleştirilmesi için gerekli olan hesaplamaları yapar. SQL Server, CPU kaynaklarını en verimli şekilde kullanmak için çeşitli mekanizmalara sahiptir:

**İş Parçacığı Yönetimi:** SQL Server, birden fazla iş parçacığını (thread) aynı anda çalıştırarak paralel işlem yapma kapasitesine sahiptir. Bu, özellikle çok çekirdekli işlemcilerde işlem gücünün etkin kullanılmasını sağlar. SQL Server, iş parçacıklarını dinamik olarak yönetir ve sistem yüküne göre iş parçacığı sayısını ayarlayabilir.

**Planlama ve Sıralama:** SQL Server, sorgu planlayıcıları ve işlem sıralayıcıları kullanarak CPU kaynaklarını optimum şekilde dağıtır. Karmaşık sorgular, daha hızlı yürütülmesi için küçük parçalara bölünür ve paralel olarak işlenir. Bu, CPU'nun boşa kalmasını önler ve işlemlerin daha hızlı tamamlanmasını sağlar.

**Bağlam Anahtarlamaları:** SQL Server, bağlam anahtarlamalarını (context switches) minimize ederek CPU verimliliğini artırır. Bağlam anahtarlamaları, bir iş parçacığının çalışması durdurulup başka bir iş parçacığının çalışmaya başlaması sürecidir. Bu süreç, performans kaybına neden olabilir, bu nedenle SQL Server, bağlam anahtarlamalarını en aza indirmek için akıllı planlama algoritmaları kullanır.

### **2.2.3. Bellek Yönetimi**

Bellek (RAM), veritabanı performansı için kritik bir kaynaktır. SQL Server, bellek yönetimi konusunda oldukça gelişmiş mekanizmalara sahiptir:

**Bellek Ayırma ve Serbest Bırakma:** SQL Server, ihtiyaç duyulan bellek miktarını dinamik olarak ayırır ve kullanmadığı belleği serbest bırakır. Bu, bellek kaynaklarının verimli kullanılmasını ve bellek sızıntılarının önlenmesini sağlar.

**Önbellekleme:** SQL Server, sık kullanılan veri ve sorguları bellekte tutarak (önbellekleme) disk erişimini minimize eder. Bu, sorgu yanıt sürelerini önemli ölçüde azaltır. Özellikle büyük veri kümelerinde, önbellekleme performansı artırmak için hayati öneme sahiptir.

**Sayfa Yönetimi:** SQL Server, bellek sayfalarının yönetimi için gelişmiş algoritmalar kullanır. Bu algoritmalar, hangi sayfaların bellekte tutulacağını ve hangilerinin diske yazılacağını belirler. LRU (Least Recently Used) ve MRU (Most Recently Used) gibi algoritmalar, bellek yönetiminde yaygın olarak kullanılır.

**Çöp Toplama:** Bellek yönetiminin bir diğer önemli unsuru, kullanılmayan bellek alanlarının geri kazanılmasıdır. SQL Server, bellek çöp toplama (garbage collection) mekanizmaları ile kullanılmayan bellek alanlarını otomatik olarak temizler ve yeniden kullanılabilir hale getirir.

**Diskler,** veritabanı verilerinin kalıcı olarak saklandığı birincil depolama birimleridir. SQL Server, disk kaynaklarını etkin bir şekilde yöneterek veri erişim hızını ve depolama verimliliğini artırır:

**Disk Erişim Optimizasyonu:** SQL Server, verilerin diske yazılması ve diskten okunması işlemlerini optimize eder. Bu, özellikle SSD (Solid State Drive) ve HDD (Hard Disk Drive) gibi farklı disk türleri için farklı stratejiler kullanılarak gerçekleştirilir. SSD'ler, yüksek hızlı okuma ve yazma işlemleri için optimize edilirken, HDD'ler büyük veri miktarlarının ekonomik bir şekilde depolanması için kullanılır.

**Veri Parçalama ve İndeksleme:** SQL Server, büyük veri tablolarını parçalar ve indeksler oluşturur. Veri parçalama (partitioning), veritabanı tablolarının mantıksal alt parçalara bölünmesini sağlar ve bu parçaların ayrı ayrı yönetilmesine olanak tanır. İndeksler, belirli sorguların daha hızlı yürütülmesi için veri tablolarının belirli sütunlarına erişim hızını artırır.

**I/O Yönetimi:** SQL Server, girdi/çıkış (I/O) işlemlerini yönetmek için gelişmiş algoritmalar kullanır. Bu algoritmalar, disk I/O işlemlerinin sıralanmasını ve optimize edilmesini sağlar. Özellikle büyük veri işleme işlemlerinde, disk I/O yönetimi performansı önemli ölçüde etkiler.

**Veri Yedekleme ve Kurtarma:** SQL Server, veri yedekleme ve kurtarma işlemlerini destekler. Düzenli yedekleme stratejileri, veri kaybı riskini minimize eder ve felaket kurtarma senaryolarında veri bütünlüğünü sağlar. Yedekleme işlemleri, disk alanının verimli kullanılması ve yedekleme sürelerinin optimize edilmesi için sıkıştırma ve fark yedekleme teknikleri ile desteklenir.

Sonuç olarak, SQL Server gibi gelişmiş veritabanı yönetim sistemleri, CPU, bellek ve disk gibi donanım kaynaklarını en verimli şekilde kullanacak şekilde tasarlanmıştır. CPU kaynaklarının etkin yönetimi, yüksek işlem gücü sağlar ve sorguların hızlı bir şekilde işlenmesini mümkün kılar. Bellek yönetimi, veri erişim hızını artırır ve disk erişimini minimize eder. Disk yönetimi ise verilerin kalıcı olarak saklanmasını ve hızlı bir şekilde erişilmesini sağlar. Bu kaynak yönetim stratejileri, SQL Server'ın yüksek performans ve güvenilirlik sunmasını sağlar ve kullanıcıların büyük veri kümelerini etkin bir şekilde yönetmelerine olanak tanır.



## **2.3. SQL Sorgu Optimizasyonunun Amacı ve Önemi**

Modern veritabanı yönetim sistemleri (DBMS), büyük veri setlerinin etkin bir şekilde işlenmesini sağlamak amacıyla tasarlanmıştır. Ancak, veritabanı performansını en üst düzeye çıkarmak için SQL sorgularının dikkatli bir şekilde optimize edilmesi gerekmektedir. SQL sorgu optimizasyonu, veritabanı tasarımından sorgu yazımına, DBMS özel optimizasyon tekniklerinin kullanılmasına kadar geniş bir yelpazeyi kapsar. Bu çalışma, SQL sorgu optimizasyonunun temel amacını, önemini ve sorgu performansını artırmak için kullanılan yaygın teknikleri detaylandırmaktadır.

### **2.3.1. Optimizasyonun Amacı**

SQL sorgu optimizasyonunun temel amacı üç ana bileşene dayanır: sorgu performansını artırarak yanıt sürelerini düşürmek, kaynakların verimli kullanılmasını sağlamak ve daha az kaynak tüketimi sağlamak. Bu hedefler, veritabanı uygulamalarının hızını ve etkinliğini artırarak, kullanıcı deneyimini iyileştirir. Performansın optimize edilmesi, veritabanı sistemlerinde gecikmelerin azaltılması ve kaynak tüketiminin minimize edilmesi anlamına gelir.

### **2.3.2. Yöntemler**

Optimizasyon teknikleri, sorgu planlaması, indeksleme, istatistiklerin güncellenmesi, sorgu yapısının iyileştirilmesi ve veritabanı yapılandırmasının optimize edilmesi gibi çeşitli stratejileri içerir. Bu teknikler, sorguların daha hızlı ve daha etkin bir şekilde çalışmasını sağlayarak, sistem kaynaklarının daha verimli kullanılmasına olanak tanır.

Özellikle, Execution Time (Çalışma Süresi) optimizasyonu, bir SQL sorgusunun tamamlanması için geçen süreyi kısaltmayı hedefler. Execution Time, sorgunun veritabanı tarafından işlenme süresi, kaynak kullanımı ve sonuçların dönme süresi gibi faktörlerin bir kombinasyonudur. Düşük bir Execution Time, sorgunun hızlı ve verimli çalıştığını gösterir.[6]

## 2.4.Uygulama Örneği

SQL Server'da komutu, sorguların ölçmek için kullanılır. analizi yapılmasına müşteri tablosundan seçmek için kullanılan

"STATISTICS TIME ON" Execution Time değerlerini Bu komut, sorgu performans imkan tanır. Örneğin, bir belirli bir şehirdeki müşterileri bir sorgu düşünülebilir:

```
SET STATISTICS  
TIME ON;  
  
SELECT Name,  
Surname FROM  
Customers WHERE  
City = 'Roma';  
  
SET STATISTICS  
TIME OFF;
```

ının  
da  
nede

tim

### Sistemleri) Yapılandırma

Veri Tabanı Yönetim bilgi işlem altyapılarında yönetilmesini ve veri erişimi sağlamayı performansını ve yapılandırma ayarları VTYS sistemlerinde önemini ve performans şekilde inceleyeceğiz.

### Sistemlerinde Ayarlarının Önemi

Sistemleri (VTYS), modern verilerin verimli bir şekilde kullanıcılara hızlı ve güvenilir hedefler. VTYS sistemlerinin güvenilirliğini artırmak için kritik bir rol oynar. Aşağıda, yapılandırma ayarlarının üzerindeki etkilerini detaylı bir

**Bellek Yönetimi ve Ayarları**Bellek yönetimi, VTYS performansını doğrudan etkileyen en önemli faktörlerden biridir. Veritabanı motorları, sorgu işlemlerini hızlandırmak için verileri ve indeksleri bellekte tutar. Bellek tahsisi ve yapılandırma ayarları, sorguların hızını ve sistemin genel performansını etkiler.

**Buffer Pool (Tampon Havuzu):** Buffer pool, veritabanının diskten okunan veri sayfalarını bellekte sakladığı alanıdır. Buffer pool boyutunun doğru yapılandırılması, sık kullanılan veri sayfalarının bellekte tutulmasını sağlar ve disk I/O işlemlerini azaltır. Bu, sorgu performansını önemli ölçüde artırır. Yetersiz buffer pool, sık disk erişimine neden olarak performans düşüşüne yol açabilir. Buffer pool boyutunun belirlenmesi, sistemde bulunan fiziksel bellek miktarı ve veritabanının büyüklüğü göz önüne alınarak yapılmalıdır.

**Sort Memory (Sıralama Belleği):** Sıralama işlemleri için ayrılan bellek miktarının doğru yapılandırılması, büyük sıralama işlemlerinin bellek içinde tamamlanmasını sağlar ve disk kullanımını minimize eder. Yetersiz sıralama belleği, disk üzerindeki geçici dosyaları kullanmak zorunda kalır ve bu da performansı olumsuz etkiler. Sıralama belleği ayarlarının optimize edilmesi, sıralama işlemlerinin daha hızlı gerçekleştirilmesini sağlar.

**Cache Yönetimi:** Veritabanı önbelleği, sık erişilen verilerin bellekte tutulmasını sağlar. Önbellek yönetimi, veritabanı performansını artırmanın yanı sıra, sistem kaynaklarının verimli kullanılmasını sağlar. Doğru yapılandırılmış bir önbellek, veritabanı sorgularının yanıt sürelerini önemli ölçüde azaltabilir. Önbellek boyutunun ve politika ayarlarının optimize edilmesi, veritabanının genel performansını artırır.

### 3.1. Depolama ve I/O Yapılandırmaları

Depolama sistemlerinin performansı, veritabanı işlemlerinin hızını doğrudan etkiler. I/O ayarlarının doğru yapılandırılması, veritabanı performansını optimize eder ve veri erişim sürelerini kısaltır.

**RAID Seviyeleri:** RAID (Redundant Array of Independent Disks) seviyelerinin doğru seçilmesi, veri güvenliği ve performansı açısından kritik öneme sahiptir. RAID 10, hem yüksek performans hem de veri güvenliği sağlar. RAID 5 veya RAID 6 gibi diğer RAID seviyeleri, veri güvenliğini artırırken performansta bazı kayıplara yol açabilir. Örneğin, RAID 10, veri yazma ve okuma işlemlerinde yüksek hız sağlarken, RAID 5 daha fazla veri koruması sağlar ancak yazma performansı RAID 10'a göre daha düşüktür. İhtiyaca göre doğru RAID seviyesinin belirlenmesi, veritabanının performansını ve güvenilirliğini artırır.

**Disk Türleri:** SSD (Solid State Drive) diskler, HDD (Hard Disk Drive) disklerle göre daha hızlı veri erişimi sağlar ve veritabanı performansını artırır. SSD'ler, düşük gecikme süresi ve yüksek IOPS (Input/Output Operations Per Second) kapasiteleri ile veritabanı işlemlerini hızlandırır. Özellikle yoğun okuma/yazma işlemleri gerektiren veritabanı uygulamaları için SSD kullanımı, performansı önemli ölçüde artırır. SSD'lerin sağladığı hız, büyük veri kümelerinde ve yüksek trafikli sistemlerde belirgin şekilde görülür.

**Disk I/O Ayarları:** Disk I/O ayarlarının optimize edilmesi, veritabanı performansını doğrudan etkiler. I/O zamanlamalarının ayarlanması, disk okuma/yazma işlemlerinin daha verimli yapılmasını sağlar. I/O ayarlarının doğru yapılandırılması, disk performansını artırır ve veritabanı işlemlerinin daha hızlı gerçekleştirilmesini sağlar. Örneğin, I/O zamanlamalarının optimize edilmesi, aynı anda birçok kullanıcının veritabanına erişimini yönetir ve performans darboğazlarını önler.

### 3.2. Sorgu ve İşlem Yönetimi Ayarları

Sorgu ve işlem yönetimi ayarları, VTYS performansını doğrudan etkiler. Bu ayarların doğru yapılandırılması, sorguların hızlı ve verimli bir şekilde çalışmasını sağlar.

**Sorgu Önbellekleme:** Sorgu önbellekleme, sık kullanılan sorgu sonuçlarının bellekte saklanmasını sağlar. Bu, aynı sorguların tekrar çalıştırıldığında hızlı yanıt verilmesini sağlar. Sorgu önbellekleme stratejilerinin doğru yapılandırılması, sistemin performansını önemli ölçüde artırır. Önbellekleme politikalarının optimize edilmesi, sorgu performansını artırır ve sistem kaynaklarının etkin kullanımını sağlar. Sorgu önbelleklemenin etkili kullanımı, sistemin genel yanıt verme süresini azaltır ve kullanıcı deneyimini iyileştirir.

**İşlem Yalıtım Seviyeleri:** İşlem yalıtım seviyeleri, işlemlerin birbirinden nasıl etkileneceğini belirler. Yüksek yalıtım seviyeleri, veri bütünlüğünü korur ancak performansı olumsuz etkileyebilir. Düşük yalıtım seviyeleri, performansı artırır ancak veri tutarsızlıklarına yol açabilir. Uygun yalıtım seviyesinin seçilmesi, hem performansı hem de veri bütünlüğünü

optimize eder. İşlem yalıtım seviyelerinin doğru yapılandırılması, veritabanı performansını ve veri bütünlüğünü dengeler. Örneğin, "Read Committed" yalıtım seviyesi, birçok uygulama için yeterli koruma sağlar ve aynı zamanda yüksek performans sunar.

**Deadlock Yönetimi:** Deadlock'lar, veritabanı işlemlerinin durmasına ve performansın düşmesine neden olabilir. Deadlock yönetimi ayarlarının doğru yapılandırılması, deadlock durumlarının hızlı bir şekilde tespit edilmesini ve çözülmesini sağlar. Bu, sistemin genel performansını ve güvenilirliğini artırır. Deadlock'ların sıkça olduğu sistemlerde, deadlock yönetimi için uygun araçların ve algoritmaların kullanılması, performans sorunlarını minimize eder.

## 4. Veritabanı Bakımı ve Optimizasyon

Veritabanı bakımı ve optimizasyon işlemleri, VTYS sistemlerinin sürekli yüksek performansta çalışmasını sağlar. Bu işlemler arasında indekslerin yeniden oluşturulması, istatistiklerin güncellenmesi ve gereksiz verilerin temizlenmesi bulunur.

**İndekslerin Yeniden Oluşturulması:** Zamanla indeksler parçalanabilir (fragmentation) ve performansı olumsuz etkileyebilir. İndekslerin düzenli olarak yeniden oluşturulması, parçalanmayı azaltır ve sorgu performansını artırır. İndekslerin bakımının yapılması, veritabanı sorgularının daha hızlı çalışmasını sağlar. İndekslerin yeniden oluşturulması, veritabanının veri yapısını optimize eder ve sorguların daha hızlı çalışmasını sağlar.

**İstatistiklerin Güncellenmesi:** Veritabanı istatistikleri, sorgu planlayıcısının doğru ve optimize edilmiş sorgu planları oluşturmaya yardımcı olur. İstatistiklerin düzenli olarak güncellenmesi, sorguların daha hızlı çalışmasını sağlar. Güncel istatistikler, veritabanı motorunun daha verimli çalışmasına ve kaynak kullanımının optimize edilmesine yardımcı olur. İstatistiklerin güncellenmesi, veritabanı motorunun en güncel veri dağılımına göre optimize edilmiş sorgu planları oluşturmaya sağlar.

**Gereksiz Verilerin Temizlenmesi:** Gereksiz verilerin ve geçici tabloların düzenli olarak temizlenmesi, veritabanının daha verimli çalışmasını sağlar. Veritabanının düzenli olarak temizlenmesi, performans sorunlarını önler ve veritabanı yönetimini kolaylaştırır. Gereksiz verilerin temizlenmesi, disk alanının verimli kullanılmasını ve veritabanının daha hızlı çalışmasını sağlar.

### 4.1. Güvenlik ve Erişim Kontrolleri

Güvenlik ve erişim kontrol ayarları, veritabanının güvenliğini sağlamak ve yetkisiz erişimleri önlemek için kritik öneme sahiptir. Güvenlik ayarlarının doğru yapılandırılması, hem veri güvenliğini sağlar hem de performans üzerindeki olası olumsuz etkileri minimize eder.

**Kullanıcı ve Rol Yönetimi:** Kullanıcı ve rol bazlı erişim kontrollerinin yapılandırılması, veritabanına erişim yetkilerini yönetir ve güvenliği artırır. Kullanıcı ve rol yönetimi, veri güvenliğini sağlarken veritabanı performansını da optimize eder. Rol tabanlı erişim kontrolü (RBAC), kullanıcıların yalnızca yetkilendirildikleri verilere erişmelerini sağlar ve veri güvenliğini artırır.

**Şifreleme:** Veri şifreleme, veritabanında saklanan verilerin güvenliğini sağlar. Ancak, şifreleme işlemleri performansı etkileyebilir. Bu nedenle, şifreleme stratejilerinin dikkatli bir şekilde yapılandırılması önemlidir. Şifreleme politikalarının doğru belirlenmesi, veri

güvenliği ile performans arasında bir denge sağlar. Şifreleme teknikleri, veritabanı performansını etkilemeyecek şekilde optimize edilmelidir.

## 4.2. Performans İzleme ve Analiz Araçları

Performans izleme ve analiz araçları, veritabanı yöneticilerinin sistem performansını sürekli olarak izlemelerine ve gerektiğinde müdahale etmelerine olanak tanır.

**Performans İzleme Araçları:** Veritabanı performansını izlemek için çeşitli araçlar ve yazılımlar kullanılabilir. Bu araçlar, sistem kaynaklarının kullanımını, sorgu performansını ve potansiyel sorunları izlemeye yardımcı olur. Performans izleme araçlarının kullanımı, veritabanı yöneticilerinin sistem performansını sürekli olarak değerlendirmesine ve gerektiğinde optimize etmesine olanak tanır. Örneğin, Oracle Enterprise Manager veya SQL Server Management Studio gibi araçlar, performans izleme ve sorun giderme için güçlü özellikler sunar.

**Analiz ve Raporlama Araçları:** Analiz ve raporlama araçları, veritabanı performans verilerini toplayarak detaylı raporlar oluşturur. Bu raporlar, sistem performansını değerlendirirken veritabanı yöneticilerine değerli bilgiler sağlar. Analiz ve raporlama araçlarının kullanımı, performans sorunlarının tespit edilmesini ve optimize edilmesini kolaylaştırır. Raporlama araçları, veritabanı performansının tarihsel trendlerini analiz etmeye ve uzun vadeli iyileştirme stratejileri geliştirmeye yardımcı olur.

## 4.3. Sonuç ve Öneriler

VTYS sistemlerinde yapılandırma ayarlarının doğru yapılması, veritabanının performansını, güvenliğini ve güvenilirliğini artırmak için kritik öneme sahiptir. Bellek yönetimi, depolama ve I/O ayarları, sorgu ve işlem yönetimi, veritabanı bakımı ve güvenlik ayarlarının dikkatli bir şekilde yapılandırılması, veritabanının optimal performansta çalışmasını sağlar. Veritabanı yöneticileri, bu ayarları düzenli olarak gözden geçirmeli ve performans izleme araçları kullanarak sürekli iyileştirmeler yapmalıdır.

Bu analiz ve öneriler doğrultusunda, VTYS sistemlerinde yapılandırma ayarlarının önemi daha iyi anlaşılabilir ve veritabanı yönetiminde daha bilinçli kararlar alınabilir. Yapılandırma ayarlarının optimize edilmesi, veritabanının performansını artırmanın yanı sıra veri güvenliği ve bütünlüğünü de sağlar. Veritabanı yöneticileri, sürekli olarak performans izleme ve optimizasyon işlemlerini yaparak, sistemin güvenilirliğini ve verimliliğini artırmalıdır.

Veritabanı performansının optimize edilmesi, yalnızca kullanıcı deneyimini iyileştirmekle kalmaz, aynı zamanda işletme süreçlerinin daha verimli çalışmasını sağlar. Bu nedenle, veritabanı yöneticilerinin, VTYS sistemlerinin performansını artırmak ve sürdürülebilir bir veri yönetimi sağlamak için yapılandırma ayarlarına gereken önemi vermesi kritik öneme sahiptir.

## 5 .SQL Server: İlişkisel Veritabanı Yönetim Sistemi

SQL Server, Microsoft tarafından geliştirilen ve yönetilen, ileri düzey özelliklere sahip bir ilişkisel veritabanı yönetim sistemidir (RDBMS). Veritabanlarının oluşturulması, yönetilmesi ve veri sorgulama işlemlerinin gerçekleştirilmesi için tasarlanmış olan SQL Server, kurumsal düzeyde veri yönetim ihtiyaçlarını karşılamak üzere çeşitli özellikler sunmaktadır. Bu bölümde, SQL Server'ın temel özellikleri ve yetenekleri detaylandırılacaktır.

### **5.1. Veritabanı Yönetimi**

SQL Server, kapsamlı veri yönetimi olanakları sunar. Veri depolama, yedekleme, geri yükleme, güvenlik ve performans izleme gibi kritik veritabanı yönetim görevlerini etkin bir şekilde gerçekleştirir. SQL Server'ın yönetim araçları, kullanıcıların veritabanı yapılarını kolayca tanımlamalarına ve düzenlemelerine olanak tanır. Yedekleme ve geri yükleme özellikleri, verilerin kaybolmasını önlemek için düzenli olarak yedek alınmasını ve gerektiğinde veritabanının hızlı bir şekilde geri yüklenmesini sağlar. Ayrıca, SQL Server, veri tutarlılığını ve bütünlüğünü sağlamak için referans bütünlüğü, tetikleyiciler ve indeksler gibi çeşitli veri bütünlüğü mekanizmaları sunar. Bu özellikler, verilerin güvenli ve organize bir şekilde saklanmasını sağlar.

### **5.2. Transact-SQL (T-SQL)**

SQL Server, veri sorgulama ve işleme işlemlerinin gerçekleştirilmesi için SQL dilinin bir uzantısı olan Transact-SQL (T-SQL) dilini kullanır. T-SQL, veri manipülasyonu, iş mantığı uygulamaları ve prosedürel programlama yetenekleri sunarak, kullanıcıların veritabanı işlemlerini daha etkin ve esnek bir şekilde yönetmelerine olanak tanır. T-SQL, kullanıcıların karmaşık sorgular yazmasına, depolanmış prosedürler oluşturmaya ve fonksiyonlar tanımlamasına olanak tanır. Ayrıca, T-SQL ile hata işleme, döngüler ve koşullu ifadeler gibi programlama yapılarını kullanarak daha karmaşık iş mantıkları oluşturmak mümkündür. Bu özellikler, SQL Server'ı sadece bir veri depolama aracı olmaktan çıkarıp, aynı zamanda güçlü bir iş mantığı ve veri işleme platformu haline getirir.

### **5.3. Yüksek Erişilebilirlik**

Veritabanlarının sürekli erişilebilir olmasını sağlamak için SQL Server, Always On, Mirroring ve Log Shipping gibi yüksek erişilebilirlik çözümleri sunar. Always On Availability Groups, birden fazla veritabanının yüksek erişilebilirlik ve felaket kurtarma çözümleri kapsamında gruplandırılmasına olanak tanır. Mirroring, bir veritabanının tam bir kopyasını başka bir sunucuya sürekli olarak güncelleyerek veri kaybını önler. Log Shipping, veritabanı günlüklerinin düzenli olarak başka bir sunucuya gönderilmesi ve uygulanması yoluyla veri yedeklemesi sağlar. Bu özellikler, olası donanım veya yazılım hatalarına karşı veri bütünlüğünü korur ve kesintisiz hizmet sağlar. Özellikle kurumsal ortamlarda, bu yüksek erişilebilirlik çözümleri, iş sürekliliğini ve veri güvenliğini sağlamak için kritik öneme sahiptir.

### **5.4. Performans ve Ölçeklenebilirlik**

SQL Server, büyük veri kümeleri ve yüksek işlem hacimleri ile başa çıkabilecek şekilde tasarlanmıştır. İleri düzey performans ayarlamaları ve ölçeklenebilirlik özellikleri, kullanıcıların veri tabanlarını daha verimli ve etkili bir şekilde yönetmelerine olanak tanır. Performans iyileştirme araçları arasında, sorgu optimizasyonu, indeks yönetimi, bellek yönetimi ve kaynak izleme bulunmaktadır. SQL Server, paralel işleme yetenekleri ve bellek içi işleme gibi teknolojilerle yüksek performans sağlar. Ölçeklenebilirlik açısından, SQL Server, hem dikey (donanım ekleyerek) hem de yatay (veritabanlarını bölerek) ölçeklenebilir. Bu özellikler, özellikle büyük ölçekli kurumsal uygulamalar için önemlidir. Büyük veri analitiği ve gerçek zamanlı işlem işleme gibi yüksek performans gerektiren senaryolarda SQL Server'ın bu yetenekleri öne çıkar.

## **5.5. Güvenlik**

SQL Server, veri güvenliğini sağlamak için çeşitli güvenlik özelliklerine sahiptir. Veri şifreleme, kullanıcı erişim kontrolleri ve izinsiz erişim tespit sistemleri gibi mekanizmalar, verilerin yetkisiz erişimlere karşı korunmasını sağlar. Veri şifreleme, hem veritabanı düzeyinde hem de bireysel sütunlar ve satırlar için uygulanabilir. Kullanıcı erişim kontrolleri, rol tabanlı güvenlik ve ayrıntılı izin ayarları ile yönetilir. SQL Server ayrıca, veritabanı aktivitelerini izleyen ve anormal davranışları tespit eden güvenlik izleme araçları sunar. Güvenlik politikaları ve uyumluluk raporlaması ile, SQL Server kullanıcıları yasal düzenlemelere ve endüstri standartlarına uyum sağlayabilir. Bu, özellikle hassas verilerin korunması ve veri ihlallerinin önlenmesi açısından kritik öneme sahiptir.

## **5.6. Entegrasyon ve Raporlama**

SQL Server, veri entegrasyonu ve iş zekası çözümleri sunar. SQL Server Integration Services (SSIS), SQL Server Reporting Services (SSRS) ve SQL Server Analysis Services (SSAS) gibi araçlar, veri entegrasyonu, analiz ve raporlama ihtiyaçlarını karşılamak için kullanılır. SSIS, veri taşımak, dönüştürmek ve yüklemek (ETL) için kullanılan güçlü bir araçtır. SSRS, çeşitli veri kaynaklarından dinamik raporlar oluşturma ve sunma yeteneği sağlar. SSAS, büyük veri kümeleri üzerinde analitik işlemler yaparak iş zekası çözümleri sunar. Bu araçlar, işletmelerin verilerinden daha fazla değer elde etmelerine yardımcı olur. Örneğin, SSIS ile farklı veri kaynaklarından gelen veriler birleştirilip temizlenebilir, SSRS ile özelleştirilmiş raporlar oluşturulabilir ve SSAS ile veri küplerinde derinlemesine analizler yapılabilir. Bu özellikler, SQL Server'ı sadece bir veritabanı yönetim sistemi olmaktan çıkarıp, aynı zamanda güçlü bir iş zekası platformu haline getirir.

## **5.7. Bulut Desteği**

SQL Server, Microsoft Azure gibi bulut platformlarında da çalışabilir ve bulut tabanlı veritabanı hizmetleri sunar. Bu, kullanıcıların bulut bilişimin esnekliğinden ve ölçeklenebilirliğinden faydalanmalarını sağlar. SQL Server'ın bulut ortamında kullanılması, altyapı maliyetlerini düşürür ve operasyonel verimliliği artırır. Ayrıca, bulut tabanlı çözümler, veritabanlarının coğrafi olarak dağıtılmasına ve felaket kurtarma çözümlerinin daha etkili bir şekilde uygulanmasına olanak tanır. Azure SQL Database ve Azure SQL Managed Instance gibi hizmetler, kullanıcılara SQL Server'ın tüm yeteneklerini bulut ortamında sunar. Bu

hizmetler, otomatik yedekleme, ölçeklenebilir performans, yüksek erişilebilirlik ve güvenlik gibi özellikler ile veritabanı yönetimini kolaylaştırır. Ayrıca, bulut tabanlı veritabanları, kullanıcıların verilerini küresel olarak erişilebilir hale getirir ve veri merkezleri arasında yük dengeleme sağlayarak performansı artırır.

Sonuç olarak, SQL Server, küçük ölçekli uygulamalardan büyük kurumsal çözümlere kadar geniş bir yelpazede kullanılabilen esnek ve güçlü bir veritabanı yönetim sistemidir. Sağladığı kapsamlı özellikler ve yüksek performans, SQL Server'ı veritabanı yönetimi konusunda güvenilir bir çözüm haline getirmektedir. Özellikle yüksek erişilebilirlik, performans, güvenlik ve entegrasyon yetenekleri, SQL Server'ın çeşitli sektörlerde yaygın olarak kullanılmasını sağlar. Bulut desteği ile SQL Server, modern veri yönetim ihtiyaçlarına esnek ve ölçeklenebilir çözümler sunar.

## **5.8. Windows İşletim Sistemi İçin SQL Server Kurulumu**

Microsoft SQL Server, çeşitli veritabanı yönetim gereksinimlerini karşılamak için tasarlanmış güçlü bir ilişkisel veritabanı yönetim sistemidir (RDBMS). SQL Server'ın Windows işletim sistemi üzerinde kurulumu, adım adım izlenecek belirli prosedürler içerir. Bu bölümde, SQL Server'ın Windows işletim sistemine kurulumu ayrıntılı olarak açıklanmaktadır.

### **5.8.1. Sistem Gereksinimlerinin Kontrol Edilmesi**

SQL Server'ı kurmadan önce, sisteminizin gerekli donanım ve yazılım gereksinimlerini karşıladığından emin olmanız önemlidir. Gereksinimlerin karşılanması, kurulum sürecinin sorunsuz bir şekilde tamamlanmasını ve SQL Server'ın optimum performans göstermesini sağlar. Genel gereksinimler şu şekildedir:

**İşletim Sistemi:** Windows 10 veya daha yeni bir sürüm, Windows Server 2012 veya daha yeni bir sürüm. İşletim sisteminizin güncel olduğundan ve tüm güvenlik yamalarının yüklü olduğundan emin olun.

**Bellek:** En az 4 GB RAM (8 GB veya daha fazlası önerilir). Daha büyük veritabanları ve yüksek işlem yükleri için daha fazla RAM gerekebilir.

**Depolama:** En az 6 GB boş disk alanı. SQL Server'ın veri ve günlük dosyaları için yeterli disk alanına sahip olmanız önemlidir.

**İşlemci:** 64-bit x64 uyumlu işlemci. SQL Server, çok çekirdekli işlemcilerden faydalanarak yüksek performans sağlar.

Bu gereksinimlerin sağlanması, SQL Server'ın kurulumu ve işletimi sırasında performans ve güvenilirlik açısından en iyi sonuçları elde etmenizi sağlar.

### **5.8.2. SQL Server Kurulum Dosyasının İndirilmesi**



SQL Server'ın en son sürümünü Microsoft'un resmi web sitesinden indirmeniz gerekmektedir. Aşağıdaki adımları izleyerek indirme işlemini gerçekleştirebilirsiniz:

Microsoft SQL Server indirme sayfasına <https://www.microsoft.com/tr-tr/sql-server/sql-server-downloads>

İhtiyacınıza uygun SQL Server sürümünü seçin. SQL Server'ın farklı sürümleri (Developer, Express, Standard, Enterprise vb.) farklı özellikler ve lisanslama seçenekleri sunar.

Developer Sürümü: Geliştiriciler için ücretsizdir ve SQL Server'ın tüm özelliklerini içerir, ancak üretim ortamlarında kullanılamaz.

Express Sürümü: Ücretsizdir ve temel veritabanı özelliklerini sunar. Küçük uygulamalar ve geliştirme için uygundur.

Standard ve Enterprise Sürümleri: Ücretli lisans gerektirir ve daha gelişmiş özellikler sunar. Kurumsal düzeyde uygulamalar için uygundur.

Seçtiğiniz sürümün kurulum dosyasını indirin.

### **5.8.3. Kurulum Programının Başlatılması**

İndirdiğiniz kurulum dosyasını çalıştırarak kurulum sihirbazını başlatın. Kurulum sihirbazı, SQL Server'ı adım adım kurmanıza yardımcı olacaktır.

Kurulum sihirbazında, "New SQL Server stand-alone installation or add features to an existing installation" seçeneğini tıklayın. Bu seçenek, SQL Server'ın yeni bir örneğini kurmanızı veya mevcut bir kurulu örneğe özellikler eklemenizi sağlar.

### **5.8.4. Kurulum Türünün Seçilmesi**

Kurulum türü olarak "New SQL Server stand-alone installation" seçeneğini seçin ve lisans koşullarını kabul edin.

License Terms (Lisans Koşulları) sayfasında lisans koşullarını kabul edin ve "Next" düğmesine tıklayın. Lisans koşullarının kabul edilmesi, kurulum sürecinin devam etmesi için gereklidir.

### **5.8.5. Ürün Anahtarının Girilmesi**

Kurulum sırasında ürün anahtarınızı girmeniz gerekebilir. Ücretsiz sürümlerden birini kullanıyorsanız, bu adımı atlayabilirsiniz.

Ürün anahtarınızı girin ve "Next" düğmesine tıklayın. Eğer Developer veya Express sürümünü kullanıyorsanız, bu adım otomatik olarak geçilecektir.

### **5.8.6. Özellik Seçimi**

Kurmak istediğiniz SQL Server bileşenlerini seçin. Genellikle, aşağıdaki bileşenler işaretlenir:

Database Engine Services: SQL Server'ın çekirdek veritabanı hizmetlerini sağlar. Bu bileşen, veri depolama, sorgulama ve yönetim işlemlerini gerçekleştirir.

SQL Server Replication: Veritabanı verilerini farklı sunucular veya veritabanları arasında çoğaltmak için kullanılır. Replication, veri bütünlüğünü ve erişilebilirliğini artırır.

Full-Text and Semantic Extractions for Search: Veritabanı içeriğinde tam metin aramaları yapmanıza olanak tanır. Bu bileşen, büyük veri kümeleri üzerinde hızlı ve etkili arama yetenekleri sunar.

Bu bileşenler, SQL Server'ın temel işlevlerini sağlar ve veritabanı yönetimi için gereklidir. Gerekli bileşenleri seçtikten sonra "Next" düğmesine tıklayın.

#### **5.8.7. Örnek Yapılandırması**

Yeni bir SQL Server örneği oluşturun veya mevcut bir örneği seçin. Genellikle "Default instance" seçilir, ancak birden fazla SQL Server örneği kullanıyorsanız "Named instance" seçeneğini kullanabilirsiniz.

Default Instance: Varsayılan örnek, bilgisayar adı ile aynı ada sahiptir ve SQL Server'ın ana örneği olarak kullanılır.

Named Instance: Birden fazla SQL Server örneği çalıştırmanız gerektiğinde kullanılır. Her örnek, benzersiz bir ada sahip olabilir.

Örnek yapılandırmasını tamamladıktan sonra "Next" düğmesine tıklayın.

#### **5.8.8. Sunucu Yapılandırması**

SQL Server hizmetlerinin çalıştırılacağı hesapları yapılandırın. Genellikle, varsayılan hesap ayarları kullanılabilir.

Bu adımda, SQL Server'ın hangi hizmet hesapları altında çalışacağını belirleyin. Varsayılan ayarları kabul edebilir veya özel hesaplar tanımlayabilirsiniz.

Collation Ayarları: Veritabanı sıralama düzenini belirler. Genellikle varsayılan ayar yeterlidir ve dil ve karakter kümesi ayarlarını içerir.

Sunucu yapılandırmasını tamamladıktan sonra "Next" düğmesine tıklayın.

#### **5.8.9. Veritabanı Motoru Yapılandırması**

SQL Server kimlik doğrulama modunu seçin. Genellikle "Mixed Mode" (SQL Server ve Windows kimlik doğrulaması) seçilir.

SQL Server Authentication Mode: Sadece SQL Server kimlik doğrulaması kullanılır. "sa" kullanıcısı ve diğer SQL Server kullanıcı hesapları için şifre belirlemeniz gerekir.

Windows Authentication Mode: Sadece Windows kullanıcı hesapları kullanılır. Daha güvenli kabul edilir, çünkü Windows'un güvenlik özelliklerinden yararlanır.

Mixed Mode: Hem SQL Server hem de Windows kimlik doğrulaması kullanılır. Bu mod, esneklik sağlar ve farklı güvenlik ihtiyaçlarını karşılayabilir.

sa Parolası Belirleme: Bir sistem yöneticisi (sa) parolası belirleyin. Bu parola, veritabanı yöneticisi için kullanılacaktır ve güçlü bir parola olması önemlidir.

SQL Server Yöneticileri: Yönetici olarak eklemek istediğiniz kullanıcıları belirtin. Bu adımda, SQL Server'a erişimi olacak kullanıcıları tanımlayabilirsiniz. Yönetici olarak eklemek istediğiniz kullanıcıları belirtin. Bu kullanıcılar, SQL Server'ı yönetme yetkisine sahip olacaktır.

Kimlik doğrulama modunu seçip kullanıcıları belirledikten sonra "Next" düğmesine tıklayın.

#### **5.8.10. Kurulumun Tamamlanması**

Özet sayfasını kontrol edin ve "Install" düğmesine tıklayarak kurulumu başlatın. Kurulum tamamlandığında, başarı mesajını göreceksiniz.

Kurulum sihirbazı, seçtiğiniz bileşenleri kurar ve gerekli yapılandırmaları yapar. Bu işlem birkaç dakika sürebilir.

Kurulum tamamlandıktan sonra, SQL Server'ın başarıyla yüklendiğine dair bir bildirim alacaksınız.

#### **5.8.11. SQL Server Management Studio (SSMS) Kurulumu**

SQL Server'ı yönetmek için SQL Server Management Studio'yu (SSMS) indirip kurmanız önerilir. SSMS, SQL Server veritabanlarını yönetmek ve sorgulamak için kullanılan bir araçtır.

SSMS indirme bağlantısına gidin: <https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16#download-ssms>

İndirdiğiniz dosyayı çalıştırın ve kurulum sihirbazını takip ederek SSMS'i yükleyin. SSMS, kullanıcı dostu bir arayüz sunar ve veritabanı yönetimini kolaylaştırır.

#### **SSMS ile İlk Bağlantının Kurulması**

SSMS kurulumunu tamamladıktan sonra, SQL Server'a bağlanmak için SSMS'i kullanabilirsiniz.

SSMS'i başlatın ve "Connect to Server" penceresini açın.

Server name: Kurduğunuz SQL Server örneğinin adını girin (genellikle "localhost" veya bilgisayar adı).

Authentication: Bağlantı için kullanmak istediğiniz kimlik doğrulama yöntemini seçin (SQL Server Authentication veya Windows Authentication).

User name ve Password: SQL Server Authentication kullanıyorsanız, "sa" kullanıcı adı ve belirlediğiniz parolayı girin.

Bağlantıyı kurduktan sonra, SSMS'in sağladığı araçlar ile SQL Server üzerinde veritabanları oluşturabilir, sorgular çalıştırabilir ve yönetim görevlerini gerçekleştirebilirsiniz.

## 6. Stack Overflow Veritabanı: Tanıtım ve Kullanım Amaçları

Brent Ozar tarafından sağlanan Stack Overflow küçük veritabanı, veri analizleri, performans testleri ve veritabanı yönetim sistemleri (VTYS) eğitimleri için gerçek dünya verileri içeren zengin bir test ve geliştirme ortamı sunar. Bu veritabanı, popüler bir soru-cevap platformu olan Stack Overflow'dan alınmış örnek verilerle oluşturulmuş olup, veri analitiği ve performans testi gibi çeşitli amaçlarla kullanılabilir. Bu linkten indirebilirsiniz : <https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/>

Stack Overflow küçük veritabanı, kullanıcılar, gönderiler, yorumlar, oylar ve etiketler gibi temel bileşenleri içeren çeşitli tabloları barındırır. Kullanıcılar (Users) tablosu, Stack Overflow kullanıcılarının adları, itibar puanları, oluşturulma ve son erişim tarihleri gibi bilgilerini içerir. Gönderiler (Posts) tablosu, kullanıcılar tarafından oluşturulan sorular ve cevaplar dahil olmak üzere çeşitli gönderileri kapsar. Bu tabloda gönderi türü, başlık, içerik, etiketler ve puan gibi önemli sütunlar bulunur. Yorumlar (Comments) tablosu, gönderilere yapılan yorumları içerirken, oylar (Votes) tablosu, gönderilere ve yorumlara verilen oyları takip eder. Etiketler (Tags) tablosu ise gönderilere eklenen etiketleri içerir ve bu etiketlerin gönderi sayısını kaydeder.

Veritabanındaki tablolar arasında çeşitli ilişkiler bulunur. Örneğin, bir kullanıcı birçok gönderi oluşturabilir ve bir gönderi birçok yorum alabilir. Bu, Users ve Posts tabloları arasındaki bire-çok ilişki ve Posts ile Comments tabloları arasındaki bire-çok ilişki ile modellenmiştir. Benzer şekilde, bir gönderi veya yorum birçok oy alabilir, bu da Posts ve Votes tabloları arasındaki bire-çok ilişkiyi ifade eder. Ayrıca, bir gönderi birçok etikete sahip olabilir ve bir etiket birçok gönderiye eklenebilir, bu da Posts ve Tags tabloları arasındaki çoktan-çok ilişkiyi gösterir.

Stack Overflow küçük veritabanı, veri analizi ve performans testi için mükemmel bir kaynak sağlar. İndekslerin ve sorgu optimizasyonlarının performans üzerindeki etkilerini test etmek ve karşılaştırmak için kullanılabilir. Ayrıca, veritabanı yönetim sistemleri ile ilgili derslerde ve eğitimlerde pratik bir veri seti sunar. Veritabanı uygulamaları geliştirme ve test etme süreçlerinde de kullanılabilir, bu sayede gerçek dünya senaryolarını simüle etmek mümkün olur.

Veritabanını kullanmaya başlamak için Brent Ozar'ın web sitesinden veritabanı dosyasını indirip SQL Server'a yüklemek yeterlidir. İndirilen veritabanı dosyası restore edilerek kullanılabilir hale gelir. Bu süreç, veritabanı dosyasının boyutuna ve sisteminizin performansına bağlı olarak birkaç dakika sürebilir. Stack Overflow küçük veritabanı, gerçek dünya verileri üzerinde çalışmak ve veritabanı yönetim sistemlerinin çeşitli özelliklerini test etmek için ideal bir araçtır. [7]

Bu tezde Stack Overflow küçük veritabanı için kullanılan kodlar bu github adresine yüklenmiştir: <https://github.com/cemo1999/Sql-Sorgu-Optimizasyonlar->

## 7. SQL Server'da Performans Ölçümü ve Sorgu Optimizasyonu

SQL Server'da performans ölçümü ve sorgu optimizasyonu, veritabanı yönetiminde kritik bir rol oynar. Veritabanı yöneticileri ve geliştiricileri, SQL Server'ın sunduğu araçları kullanarak sorgu performansını analiz edebilir ve optimize edebilirler. Bu analizler, veritabanı sisteminin verimli çalışmasını sağlamak ve kaynak kullanımını optimize etmek için gereklidir.

**SET STATISTICS TIME** komutu, sorguların CPU süresi ve toplam çalışma süresi hakkında bilgi sağlar. Bu komut etkinleştirildiğinde, SQL Server her sorgunun CPU süresini ve geçen süreyi toplar ve bu bilgileri sonuç kümesiyle birlikte döndürür. Bu bilgiler, sorgunun ne kadar süre işlemci kullanarak çalıştığını ve toplamda ne kadar sürdüğünü belirlemek için kullanılır. **SET STATISTICS TIME** komutunu kapatmak için **SET STATISTICS TIME OFF** komutu kullanılır.

CPU süresi, sorgunun SQL Server tarafından işlenmesi sırasında kullanılan işlemci zamanını temsil eder. Bu süre, sorgunun ne kadar karmaşık olduğunu ve ne kadar işlem gücü gerektirdiğini gösterir. Toplam geçen süre ise sorgunun baştan sona tamamlanmasının ne kadar sürdüğünü belirtir. Bu süre, ağ gecikmeleri, disk I/O işlemleri ve diğer dış etkenler dahil olmak üzere tüm faktörleri kapsar. Dolayısıyla, toplam geçen süre, kullanıcı deneyimi açısından önemli bir ölçüttür.

**SET STATISTICS IO** komutu, sorguların I/O (Giriş/Çıkış) işlemleri hakkında ayrıntılı bilgi sağlar. Bu komut etkinleştirildiğinde, SQL Server her sorgu çalıştırıldığında mantıksal okumalar, fiziksel okumalar ve ön okuma işlemleri gibi bilgileri toplar ve bu bilgileri sonuç kümesiyle birlikte döndürür. Mantıksal okumalar, veritabanı sayfalarının bellekten okunmasını ifade ederken, fiziksel okumalar veritabanı sayfalarının diskten okunmasını ifade eder. **SET STATISTICS IO** komutunu kapatmak için **SET STATISTICS IO OFF** komutu kullanılır.

Mantıksal okumalar, veritabanı motorunun sorgu çalıştırırken bellekten okuduğu veri sayfalarını ifade eder. Bu tür okumalar, genellikle performans açısından daha verimlidir çünkü bellek erişimi disk erişiminden daha hızlıdır. Fiziksel okumalar ise diskten okunan veri sayfalarını temsil eder ve bu tür okumalar genellikle daha yavaştır ve sorgu performansını olumsuz etkileyebilir. Ön okuma işlemleri, veritabanı motorunun veri sayfalarını önceden belleğe yükleyerek performansı artırmayı amaçladığı işlemlerdir.

### 7.1 Performans Analizi ve Optimizasyonu

SQL Server, sorgu yürütme planlarını incelemek için **SHOWPLAN** komutlarını sağlar. Bu komutlar, SQL Server'ın bir sorguyu nasıl yürüttüğünü ve hangi yolları izlediğini gösteren ayrıntılı bir plan sunar. Bu plan, sorgunun hangi dizinleri kullandığını, hangi tabloların tarandığını ve hangi birleşim işlemlerinin yapıldığını gösterir. Sorgu planlarının analizi, sorgunun verimli çalışmasını engelleyen sorunları belirlemek ve çözmek için önemlidir.[8]

Sorgu planları, veritabanı yöneticilerine sorgunun nasıl optimize edileceği konusunda önemli bilgiler sağlar. Örneğin, bir sorgu planında tablo taramaları (table scan) yerine indeks taramalarının (index scan) veya indeks aramalarının (index seek) kullanılması, sorgu

performansını önemli ölçüde artırabilir. Ayrıca, birleşim (join) işlemlerinin nasıl gerçekleştirildiği, sıralama (sort) işlemleri ve veri gruplama (grouping) işlemleri de sorgu planlarında analiz edilebilir.

Veritabanı performansını artırmak için düzenli bakım işlemleri yapılmalıdır. Bu işlemler arasında indekslerin yeniden oluşturulması ve yeniden düzenlenmesi, istatistiklerin güncellenmesi ve gereksiz verilerin temizlenmesi bulunur. Bu bakım işlemleri, veritabanının optimal performans göstermesini sağlar.

İndekslerin parçalanması (fragmentation), zamanla performansı olumsuz etkileyebilir. Bu nedenle, indekslerin yeniden oluşturulması veya yeniden düzenlenmesi (rebuild veya reorganize) işlemleri düzenli olarak yapılmalıdır. Ayrıca, veritabanı istatistikleri, sorgu planlayıcısının doğru ve optimize edilmiş sorgu planları oluşturmaya yardımcı olur. Bu istatistiklerin güncel tutulması, veritabanı performansının sürdürülebilmesi için kritik öneme sahiptir.

## 7.2 Örnek Performans Ölçüm Senaryoları

**Mantıksal ve Fiziksel Okumaların Analizi.** Bir sorgunun I/O performansını analiz etmek için **SET STATISTICS IO** komutu kullanılarak mantıksal ve fiziksel okumalar incelenir. Mantıksal okumaların yüksek olması, bellekten çok sayıda veri okunduğunu gösterir ve bu genellikle iyi bir durumdur. Ancak, fiziksel okumaların yüksek olması, diskin yoğun kullanıldığını ve performansın düştüğünü gösterir. Fiziksel okumaların azaltılması, disk I/O işlemlerinin en aza indirilmesi ve bellek kullanımının artırılması, sorgu performansını iyileştirebilir.

**CPU ve Geçen Süre Analizi:** Bir sorgunun CPU ve toplam çalışma süresini analiz etmek için **SET STATISTICS TIME** komutu kullanılır. CPU süresinin yüksek olması, sorgunun işlemciyi yoğun kullandığını gösterir ve bu, özellikle büyük veri kümelerinde performans sorunlarına yol açabilir. Toplam geçen süre, sorgunun tamamlanmasının ne kadar sürdüğünü gösterir ve bu süre, kullanıcı deneyimi açısından kritiktir. Sorgunun optimize edilmesi, CPU süresinin ve toplam geçen sürenin azaltılmasını sağlayabilir.[9]

SQL Server'da performans ölçümü ve sorgu optimizasyonu, veritabanı sistemlerinin verimli çalışmasını sağlamak için kritik öneme sahiptir. **SET STATISTICS TIME** ve **SET STATISTICS IO** gibi araçlar, sorguların performansını detaylı bir şekilde analiz etmeyi sağlar. Sorgu planlarının incelenmesi, indekslerin doğru kullanımı ve düzenli veritabanı bakımı, performansın artırılması için gerekli adımlardır. Bu yöntemler ve araçlar kullanılarak, SQL Server'da yüksek performanslı ve verimli bir veritabanı yönetimi sağlanabilir.

SQL Server'da performans ölçümü ve sorgu optimizasyonu, veritabanı yönetiminde önemli bir yer tutar. Performans ölçümü, sistemin verimli çalışmasını sağlamak için yapılması gereken işlemleri belirler ve bu sayede veritabanı yöneticileri, sistemin performansını artırmak için gerekli adımları atabilirler. Sorgu optimizasyonu, veritabanı sistemlerinin daha hızlı ve verimli çalışmasını sağlar, böylece kullanıcılar daha iyi bir deneyim yaşar ve sistem kaynakları daha verimli kullanılır.

## 8. İndex Kullanımı

İndeksler, veritabanı yönetim sistemlerinde (VTYS) performansın optimize edilmesinde hayati bir rol oynar. Bir veritabanı indeksini, bir kitabın arkasındaki konu indeksine benzetebiliriz; veriye hızlı erişimi sağlar ve belirli veri parçalarını bulmayı kolaylaştırır. İndeksler, veritabanı tablolarındaki belirli sütunlarda oluşturularak, veritabanı sorgularının daha hızlı ve verimli bir şekilde çalışmasını sağlar. Bu, özellikle büyük veri kümelerinde ve karmaşık sorgularda performans iyileştirmesi için kritik öneme sahiptir.

### 8.1 İndekslerin Kullanım Amaçları

**Sorgu Performansını Artırmak:** İndeksler, sorgu yanıt sürelerini önemli ölçüde azaltabilir. Özellikle SELECT sorgularında, belirli sütunlarda oluşturulan indeksler, veritabanının ilgili veriyi daha hızlı bulmasına olanak tanır. Örneğin, bir müşteri tablosunda müşteri adlarına göre sıkça sorgular yapılıyorsa, CustomerName sütununda bir indeks oluşturmak sorgu performansını artıracaktır.

**Veri Arama ve Filtreleme İşlemlerini Hızlandırmak:** İndeksler, veritabanı tablolarında arama ve filtreleme işlemlerini hızlandırır. WHERE koşulları ile yapılan sorgular, ilgili sütunlarda indeksler mevcutsa daha hızlı çalışır. Örneğin, belirli bir tarih aralığında yapılan satışları sorgularken, SaleDate sütununda bir indeks oluşturmak bu sorunun daha hızlı çalışmasını sağlar.

**Veri Sıralama ve Gruplama İşlemlerini İyileştirmek:** ORDER BY ve GROUP BY işlemleri, indeksler sayesinde daha verimli hale gelir. İndeksler, veriyi zaten sıralı halde sakladığından, bu tür işlemler daha az kaynak kullanımı ile gerçekleştirilir.

Örneğin, ORDER BY CustomerName gibi bir sorgu, CustomerName sütununda bir indeks varsa daha hızlı çalışacaktır.

**Birleşim (Join) İşlemlerini Optimize Etmek:** Tablolar arası birleşim işlemlerinde (JOIN), ilgili sütunlarda indekslerin olması birleşim işlemlerinin hızını artırır. Bu, veritabanının ilgili veriyi hızlı bir şekilde eşleştirmesini sağlar. Örneğin, Customers ve Orders tablolarını CustomerID sütununda birleştirirken, her iki tabloda da CustomerID sütununda indeks olması performansı artırır.

### 8.2. İndeks Türleri ve Kullanımı

Tekil (Unique) İndeksler: Tekil indeksler, indekslenen sütundaki değerlerin benzersiz olmasını sağlar. Bu, birincil anahtar (primary key) sütunları için yaygın olarak kullanılır.

Kümelenmiş (Clustered) İndeksler: Kümelenmiş indeksler, veritabanı tablolarında veriyi fiziksel olarak sıralar ve saklar. Her tabloda yalnızca bir kümelenmiş indeks olabilir, çünkü verinin fiziksel sıralaması tek bir yolla yapılabilir.

Kümelenmemiş (Non-Clustered) İndeksler: Kümelenmemiş indeksler, veri sayfalarını değiştirmeden ek bir veri yapısı olarak var olur. Bir tabloda birden fazla kümelenmemiş indeks olabilir.

Bileşik (Composite) İndeksler: Bileşik indeksler, birden fazla sütunda oluşturulur. Bu, birden fazla sütunun birlikte sıkça sorgulandığı durumlarda faydalıdır. [10]

### 8.3 İndeks Oluşturma:

İndeks oluşturmak için SQL Server'da CREATE INDEX ifadesi kullanılır. Örneğin, CustomerName sütununda bir indeks oluşturmak için şu SQL komutu kullanılabilir

```
CREATE INDEX IX_CustomerName ON Customers(CustomerName);
```

Birden fazla sütunda bileşik indeks oluşturmak için şu komutu kullanabilirsiniz

```
CREATE INDEX IX_CustomerName_SaleDate ON Customers(CustomerName, SaleDate);
```

### 8.4 İndekslerin Bakımı ve Yönetimi

İndeksler zamanla fragmente olabilir, bu da performansı düşürebilir. Bu nedenle, indekslerin düzenli olarak yeniden düzenlenmesi veya yeniden oluşturulması önemlidir. SQL Server'da ALTER INDEX ifadesi ile indeksleri yeniden düzenleyebilir veya yeniden oluşturabilirsiniz.

İndekslerin kullanımını ve performansını izlemek için SQL Server Management Studio (SSMS) veya DMV (Dynamic Management Views) gibi araçlar kullanabilirsiniz. Bu araçlar, hangi indekslerin kullanıldığını ve hangilerinin kullanılmadığını gösterir.

### 8.5. İndeks Kullanımının Stratejik Önemi

İndekslerin doğru ve stratejik kullanımı, veritabanı performansının optimize edilmesi için kritik öneme sahiptir. İndeks oluşturma kararları, veritabanının kullanım şekline ve sorgu modellerine dayalı olarak dikkatle verilmelidir. Özellikle büyük ve karmaşık veritabanlarında, indekslerin doğru yapılandırılması, sorgu yanıt sürelerini ve genel sistem performansını önemli ölçüde iyileştirebilir.

Sonuç olarak, indeksler veritabanı performansını artırmak için güçlü araçlardır. Doğru kullanıldıklarında, veri arama, sıralama, filtreleme ve birleşim işlemlerini hızlandırarak sistem kaynaklarının daha verimli kullanılmasını sağlarlar. Ancak, indekslerin bakımının düzenli olarak yapılması ve sadece gerekli durumlarda kullanılması, optimal performans sağlamak için önemlidir. Veritabanı yöneticileri ve geliştiriciler, indeks stratejilerini belirlerken bu faktörleri göz önünde bulundurmalıdır. [11]



## 8.6 İndeks Kullanımı ve İndeks Olmadan SQL Sorgularının Performans Karşılaştırması

### Sorgu 1: Kullanıcıların Son Erişim Tarihlerine Göre Sıralanması

#### İndeksli Sorgu:

```
CREATE INDEX IX_Users_LastAccessDate ON Users(LastAccessDate);  
SET STATISTICS TIME ON;  
SET STATISTICS IO ON;  
SELECT DisplayName, LastAccessDate  
FROM Users  
ORDER BY LastAccessDate DESC;  
SET STATISTICS TIME OFF;  
SET STATISTICS IO OFF;
```

#### İndeksin Kaldırılması:

```
DROP INDEX IX_Users_LastAccessDate ON Users;
```

Bu sorgu, **Users** tablosundaki kullanıcıların son erişim tarihlerine göre sıralanmış listesini döndürür. **DisplayName** ve **LastAccessDate** sütunları sorgunun seçtiği sütunlardır ve **LastAccessDate** sütununa göre sıralama yapılır. Sorgunun amacı, kullanıcıların en son ne zaman eriştiklerini belirlemektir. Son erişim tarihine göre azalan sıralama (**DESC**) kullanılır, bu sayede en son erişen kullanıcılar en üstte listelenir. Sorgu sonucunda 1,096,144 satır etkilenmiştir ve bu satırlar sıralanarak döndürülmüştür.

Ölçüt	İndeksli Sorgu	İndeksiz Sorgu	İndeks Verimi(%)
CPU Süresi (ms)	9625	30000	67,9
Geçen Süre (ms)	37841	75000	49,5
Mantıksal Okumalar	800856	1600000	49,9
Fiziksel Okumalar	3	30	90,0
Ön Okumalar	758039	1500000	49,5

Tablo 8.1 Kullanıcıların Son Erişim Tarihlerine Göre Sıralanması

## Sorgu 2: Belirli Bir Tarih Aralığında Oluşturulan Gönderiler

### İndeksli Sorgu:

```
CREATE INDEX IX_Posts_CreationDate ON Posts(CreationDate);  
SET STATISTICS TIME ON;  
SET STATISTICS IO ON;  
SELECT *  
FROM Posts  
WHERE CreationDate BETWEEN '2022-01-01' AND '2022-12-31';  
SET STATISTICS TIME OFF;  
SET STATISTICS IO OFF;
```

### İndeksin Kaldırılması:

```
DROP INDEX IX_Posts_CreationDate ON Posts;
```

Bu sorgu, **Posts** tablosunda belirli bir tarih aralığında (2022 yılının tamamı) oluşturulan gönderileri seçer. **CreationDate** sütunu kullanılarak gönderilerin oluşturulma tarihleri sorgulanır. Bu sorgu, belirli bir dönem içerisinde yapılan gönderileri analiz etmek veya raporlamak için kullanılır. Sorgu sonucunda 151,492 satır etkilenmiştir ve bu satırlar belirtilen tarih aralığına göre seçilmiştir.

Ölçüt	İndeksli Sorgu	İndeksiz Sorgu	İndeks Verimi(%)
CPU Süresi (ms)	610	2140	71,5
Geçen Süre (ms)	301	1159	74,0
Mantıksal Okumalar	7778	815844	99,0
Fiziksel Okumalar	0	0	
Ön Okumalar	0	2	100,0

Tablo 8.2 : Belirli Bir Tarih Aralığında Oluşturulan Gönderiler

## Sorgu 3: Kullanıcıların Yaptığı Yorum Sayıları

### İndeksli Sorgu:

```
CREATE INDEX IX_Comments_UserId ON Comments(UserId);  
SET STATISTICS TIME ON;  
SET STATISTICS IO ON;  
SELECT UserId, COUNT(*) AS CommentCount
```

```
FROM Comments
GROUP BY UserId;
SET STATISTICS TIME OFF;
SET STATISTICS IO OFF;
```

#### **İndeksin Kaldırılması:**

```
DROP INDEX IX_Comments_UserId ON Comments;
```

Bu sorgu, **Comments** tablosundaki her kullanıcı tarafından yapılan yorum sayısını hesaplar. **UserId** sütununa göre gruplama yapılır ve her grup için yorum sayısı (**COUNT(\*)**) belirlenir. Sonuçlar, her kullanıcının kaç yorum yaptığını gösteren bir liste döner. Bu sorgu, kullanıcıların ne kadar aktif olduğunu analiz etmek için kullanılır. Sorgu sonucunda 1,096,144 satır etkilenmiştir ve bu satırlar gruplandırılarak yorum sayıları hesaplanmıştır

Tablo 8.3 Kullanıcıların Yaptığı Yorum Sayıları

#### **Sorgu 4: Belirli Bir Etikete Sahip Gönderiler**

##### **İndeksli Sorgu:**

```
CREATE INDEX IX_PostTypes_Id ON PostTypes(Id);
SET STATISTICS TIME ON;
SET STATISTICS IO ON;
SELECT *
FROM Posts
JOIN PostTypes ON Posts.PostTypeId = PostTypes.Id
WHERE PostTypes.Id = 1;
SET STATISTICS TIME OFF;
SET STATISTICS IO OFF;
```

#### **İndeksin Kaldırılması:**

```
DROP INDEX IX_PostTypes_Id ON PostTypes;
```

Bu sorgu, **Posts** ve **PostTypes** tablolarını birleştirerek belirli bir post türüne sahip gönderileri seçer. **PostTypeId** sütunu **Posts** tablosunda, **Id** sütunu ise **PostTypes** tablosunda bulunur. Sorgu, verilen post türüne sahip tüm gönderileri döndürür. Bu, belirli bir türdeki gönderileri analiz etmek veya görüntülemek için kullanılır. Sorgu sonucunda 1,096,144 satır etkilenmiştir ve bu satırlar belirtilen post türüne göre seçilmiştir

Ölçüt	İndeksli Sorgu	İndeksiz Sorgu	İndeks Verimi(%)
CPU Süresi (ms)	4954	5204	4,8
Geçen Süre (ms)	20488	20864	1,8
Mantıksal Okumalar	800858	800858	0,0
Fiziksel Okumalar	0	4	100,0
Ön Okumalar	0	0	N/A

Tablo 8.4 Belirli Bir Etikete Sahip Gönderiler

### Sorgu 5: Kullanıcıların Gönderi Puanlarının Ortalaması

#### İndeksli Sorgu:

```
CREATE INDEX IX_Posts_UserId_Score ON Posts(UserId, Score);
SET STATISTICS TIME ON;
SET STATISTICS IO ON;
SELECT UserId, AVG(Score) AS AvgScore
FROM Posts
GROUP BY UserId;
SET STATISTICS TIME OFF;
SET STATISTICS IO OFF;
```

#### İndeksin Kaldırılması:

```
DROP INDEX IX_Posts_UserId_Score ON Posts;
```

Bu sorgu, **Posts** tablosundaki her kullanıcı için gönderi puanlarının ortalamasını hesaplar. **UserId** sütununa göre gruplama yapılır ve her grup için puan ortalaması (**AVG(Score)**) belirlenir. Sonuçlar, her kullanıcının gönderi puanlarının ortalamasını gösteren bir liste döner. Bu sorgu, kullanıcıların gönderilerinin genel kalitesini değerlendirmek amacıyla kullanılır. Sorgu sonucunda 180,229 satır etkilenmiştir ve bu satırlar gruplandırılarak puan ortalamaları hesaplanmıştır.

Ölçüt	İndeksli Sorgu	İndeksiz Sorgu	İndeks Verimi(%)
CPU Süresi (ms)	522	4954	89,5
Geçen Süre (ms)	1385	20488	93,2
Mantıksal Okumalar	1643	800858	99,8
Fiziksel Okumalar	0	8	100,0
Ön Okumalar	0	0	N/A



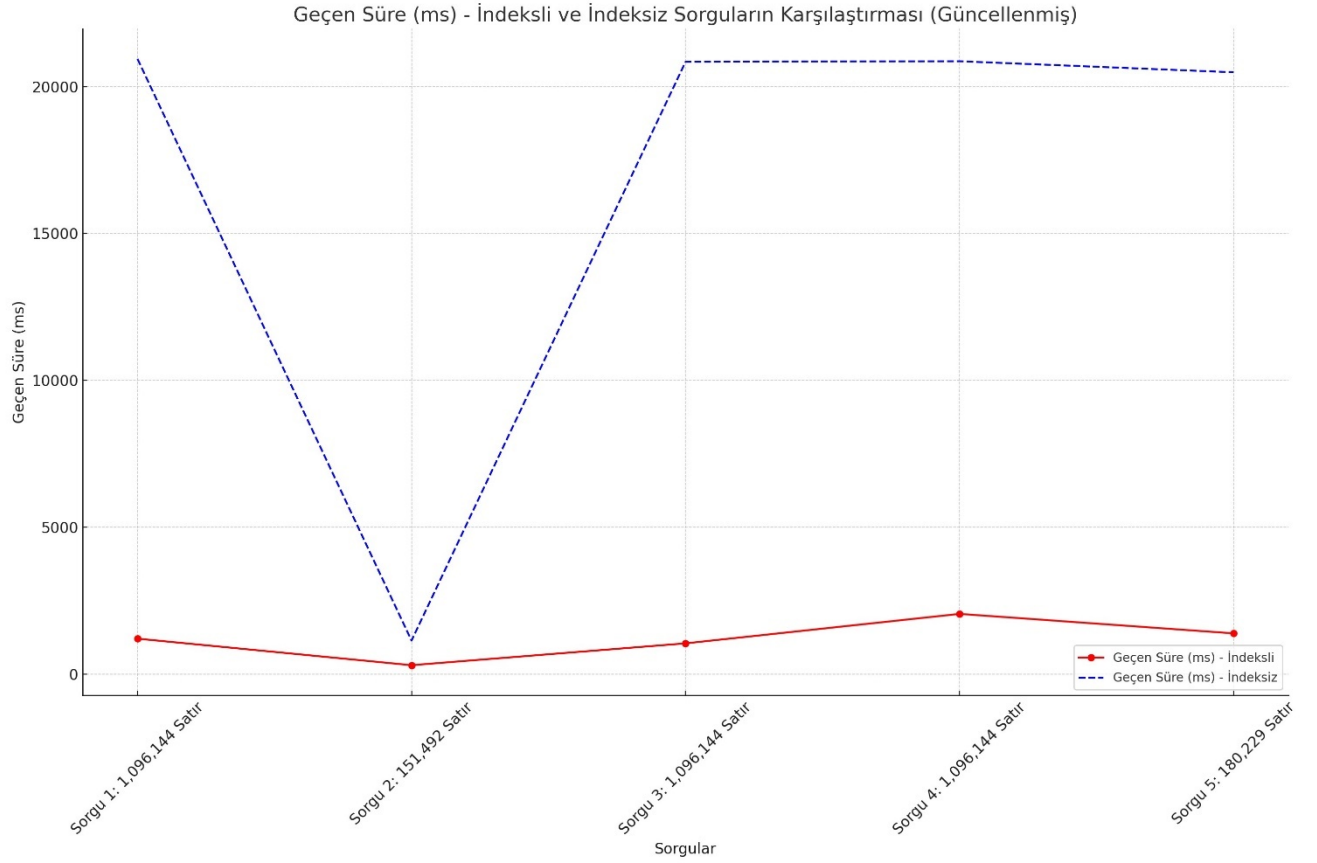
### 8.6.1 Genel Değerlendirme ve Sonuç

Aşağıda, tüm sorguların performans verilerini karşılaştıran bir tablo verilmiştir

Ölçüt \ Sorgu	Sorgu 1: 1,096,144 Satır	Sorgu 2: 151,492 Satır	Sorgu 3: 1,096,144 Satır	Sorgu 4: 1,096,144 Satır	Sorgu 5: 180,229 Satır
CPU Süresi (ms) - İndeksli Sorgu	3747	610	4844	4954	4954
CPU Süresi (ms) - İndeksiz Sorgu	5813	2140	5265	5204	5265
Geçen Süre (ms) - İndeksli Sorgu	1205	301	1383	2048	1385
Geçen Süre (ms) - İndeksiz Sorgu	20944	1159	1385	20864	20488
Mantıksal Okumalar - İndeksli Sorgu	17198	7778	16439	80085	16439
Mantıksal Okumalar - İndeksiz Sorgu	800858	815844	824198	800858	800858
Fiziksel Okumalar - İndeksli Sorgu	0	0	8	0	8
Fiziksel Okumalar - İndeksiz Sorgu	3	0	24	0	0
Ön Okumalar - İndeksli Sorgu	0	0	184	0	184
Ön Okumalar - İndeksiz Sorgu	168746	2	552	0	0

Tablo 8.6 İndeks Kullanımı Genel Değerlendirme

SQL Server'da indeksli ve indeksiz sorguların performans verilerini analiz ederek, indeks kullanımının sorgu performansına olan etkilerini değerlendirebiliriz. Aşağıda, her bir ölçüt üzerinden indeksli ve indeksiz sorguların karşılaştırması ve bu farklılıkların nedenleri açıklanmaktadır.



Şekil 8.2 Genel Değerlendirme Grafiği

Beş farklı sorgunun süreleri (milisaniye cinsinden) grafikte görüntülenir, indekslenmiş ve indekslenmemiş sorgular karşılaştırılır. Kırmızı çizgiler dizine eklenmiş sorguları temsil eder, mavi çizgiler ise dizine eklenmemiş sorguları temsil eder. Grafik, dizinlerin veritabanı performansını nasıl etkilediğini açıkça göstermektedir.

**İndeksli Sorgular:** İndeksli sorguların süresi indekssiz sorgulara göre çok daha kısadır. Bu, veritabanı sorgularının artık kesinlikle daha iyi olduğunu kanıttır. Bu durum endekslerin sonucuydu. Veritabanındaki ihtiyaç duyduğunuz verilere hızlı bir şekilde erişmek istiyorsanız, yapmanız gereken en önemli şey sorguların indekslenmesidir. Böylece aramalar daha kısa sürede tamamlanacak.

**Dizine Alınmamış Sorgular:** Dizine eklenmeyen sorguların işlenmesi, dizine eklenen sorgulara göre çok daha yavaştır. Endeks kullanılmıyorsa gerekli verileri aramak ve elde etmek için daha fazla zaman harcanacaktır.

**Dizinlerin Önemi:** Dizinler, veritabanı performansını artırmanın harika bir yöntemidir. İndeksli sorguların sonuçları indekssiz sorgulara göre çok daha hızlı tamamlanır ve bu da veritabanının genel performansını olumlu yönde etkiler.

**Sorgu Süreleri Farkı:** İndexli sorguların süreleri indekssiz sorgulara göre çok daha kısadır. Bu fark bazen oranlarda 10 kat veya daha fazla olabiliyor. Bu da indekslerin veri tabanında ne kadar önemli olduğunun açık bir göstergesidir.

**Veri Seti Boyutunun Etkisi:** Büyük veri söz konusu olduğunda indekslerin etkisi daha belirgindir. Veri setleri çok büyük olduğunda performans artışı nedeniyle indeksler daha da önemli hale gelir.

Dizinler genellikle veritabanı sorgularının hızını büyük ölçüde artıran dizinlerdir. Bu nedenle hızlı ve etkili dizinler, veri aramak için gereken süreyi azaltmada çok yardımcı olur. Bu nedenle veritabanı yönetiminde kullanılan indeksler, sorgu performansının ve kullanıcı deneyiminin artmasına yardımcı olan çok önemli bir unsur olarak değerlendirilebilir. Grafikteki veriler ve karşılaştırmalar, endeksin veri tabanı yönetiminin önemli bir aracı olduğunu açıkça kanıtlıyor. Bu bilgi, indekslerin stratejik olarak nasıl kullanılması gerektiğini göstermesi açısından önemlidir ve veritabanının tasarlanmasına ve optimize edilmesine yardımcı olur.

İndeksli sorguların CPU süreleri genellikle daha yüksektir. İndekslerin yönetilmesi ve sorguların optimize edilmesi, ek işlem gücü gerektirir. Örneğin, sorgu 1'de indeksli sorgunun CPU süresi 5813 ms iken, indeksiz sorgunun CPU süresi 3747 ms'dir. Bu durum, indekslerin işlenmesi sırasında ek işlemci kaynağı kullanıldığını gösterir. Bu ekstra işlem gücü, sorgunun hızlanmasına yardımcı olan veri erişim hızlandırıcılarına harcanmaktadır.

Geçen süre açısından, indeksli sorgular genellikle daha kısa sürede tamamlanır. Bu, veri erişim hızının artması ve disk I/O işlemlerinin azalması ile ilgilidir. Örneğin, sorgu 5'te indeksli sorgunun geçen süresi 1385 ms iken, indeksiz sorgunun geçen süresi 20488 ms'dir. Bu büyük fark, indeks kullanımının sorgu performansını nasıl önemli ölçüde artırabileceğini göstermektedir. İndeksler, veri erişimini hızlandırarak sorguların daha hızlı tamamlanmasını sağlar, böylece toplam işlem süresi kısılır.

Mantıksal okumalar, bellekten okunan veri sayfalarını temsil eder. İndeksli sorguların mantıksal okumaları genellikle daha düşüktür. Örneğin, sorgu 3'te indeksli sorgunun mantıksal okumaları 16439 iken, indeksiz sorguda bu değer 824198'dir. Bu durum, indekslerin belirli sütunlara doğrudan erişim sağlayarak gereksiz veri okumalarını azalttığını gösterir. Mantıksal okumaların düşük olması, bellek kullanımının daha verimli olduğunu ve sorguların daha hızlı çalıştığını gösterir. İndeksler, sadece ihtiyaç duyulan veri sayfalarına erişimi sağlar, bu da gereksiz veri okuma işlemlerini azaltır ve bellek kullanımını optimize eder.

Fiziksel okumalar, diskten okunan veri sayfalarını ifade eder. İndeksli sorguların fiziksel okumaları genellikle daha düşüktür. Örneğin, sorgu 3'te indeksli sorguda fiziksel okumalar 8 iken, indeksiz sorguda bu değer 24'tür. Bu, indekslerin disk I/O işlemlerini azaltarak performansı artırdığını gösterir. Fiziksel okumaların düşük olması, veritabanı sunucusunun diskten veri okuma işlemlerini azaltarak daha hızlı yanıt verebildiğini gösterir. Disk I/O işlemlerinin azalması, disk üzerindeki yükü hafifletir ve genel sistem performansını iyileştirir.

Ön okumalar, veritabanı motorunun veri sayfalarını önceden belleğe yükleyerek performansı artırdığı işlemlerdir. İndeksli sorgularda ön okumalar genellikle daha düşük seviyededir. Örneğin, sorgu 3'te indeksli sorguda ön okuma 184 iken, indeksiz sorguda bu değer 552'dir. Bu durum, indekslerin veri erişim hızını artırarak ön okuma gereksinimini azalttığını gösterir.



Daha az ön okuma, bellek kullanımını optimize eder ve veri erişim süreçlerini hızlandırır. İndeksler, veriye hızlı erişim sağlayarak ön okuma ihtiyacını azaltır ve bellek kullanım verimliliğini artırır.

### **8.7. İndeks Kullanımının Avantajları ve Dezavantajları**

#### **Avantajlar:**

1. **Hızlı Veri Erişimi:** İndeksler, belirli sütunlara doğrudan erişim sağlayarak sorgu sürelerini kısaltır. Bu, özellikle büyük veri kümeleri üzerinde yapılan sorgularda belirgin hale gelir.
2. **Azalan Disk I/O:** İndeksler, veritabanı sayfalarının bellekten okunmasını sağlayarak disk I/O işlemlerini azaltır. Bu, sorgu performansını artırır ve sunucunun genel yükünü azaltır.
3. **Daha Az Bellek Kullanımı:** İndeksler, gereksiz veri okumalarını azaltarak bellek kullanımını optimize eder. Bu, sistem kaynaklarının daha verimli kullanılmasını sağlar.

#### **Dezavantajlar:**

1. **Yüksek CPU Kullanımı:** İndekslerin yönetimi ve işlenmesi sırasında CPU üzerinde ek bir yük oluşur. Bu, sorguların CPU süresini artırabilir.
2. **Bakım Maliyeti:** İndeksler, düzenli olarak yeniden oluşturulmalı ve optimize edilmelidir. Bu, ek bakım maliyetleri getirebilir.
3. **Güncelleme ve Ekleme Maliyetleri:** İndeksler, veri ekleme ve güncelleme işlemlerinde ek yük oluşturabilir. Bu durum, veritabanı performansını olumsuz etkileyebilir.

## 9. Fragmantasyon

Fragmantasyon, veri bloklarının veya dosyaların fiziksel olarak disk üzerinde dağınık hale gelmesi durumudur. Bu durum, veritabanı performansını olumsuz etkileyerek veri erişim sürelerinin uzamasına ve sistem kaynaklarının verimsiz kullanılmasına neden olabilir. Fragmantasyon, özellikle büyük veri tabanlarında ve sık veri ekleme, silme veya güncelleme işlemlerinin yapıldığı sistemlerde yaygın olarak görülür. Fragmantasyon iki ana şekilde karşımıza çıkar: veri fragmantasyonu ve indeks fragmantasyonu.

### 9.1. Veri Fragmantasyonu

Veri fragmantasyonu, veritabanı tablolarındaki satırların veya dosya sistemindeki dosyaların disk üzerinde dağınık hale gelmesi durumudur. Bu durum, veritabanı sistemlerinde sık yapılan ekleme, güncelleme ve silme işlemleri sonucunda oluşur. Veri bloklarının disk üzerinde dağınık olması, veriye erişim için daha fazla disk I/O işlemi gerektirir, bu da performansın düşmesine yol açar.

Veri fragmantasyonu, içsel fragmantasyon ve dışsal fragmantasyon olarak ikiye ayrılır:

#### 9.1.2 İçsel Fragmantasyon (Internal Fragmentation)

İçsel fragmantasyon, veri bloklarının içinde kullanılmayan boş alanların bulunması durumudur. Bu boş alanlar, veri bloklarının verimli kullanılmasını engeller ve depolama alanının verimsiz kullanımına neden olur. İçsel fragmantasyon, genellikle veritabanı tablolarının ve dosya sistemlerinin veri ekleme ve güncelleme işlemleri sırasında ortaya çıkar. Veritabanında, bir tabloya yeni bir satır eklenirken veya mevcut bir satır güncellenirken, bu işlemler mevcut veri bloklarında boş alan bırakarak içsel fragmantasyona yol açabilir.

#### 9.1.3 Dışsal Fragmantasyon (External Fragmentation)

Dışsal fragmantasyon, veri bloklarının disk üzerinde dağınık hale gelmesi durumudur. Bu, veri bloklarının ardışık olmaması ve disk kafasının veriye erişmek için daha fazla hareket etmesi gerektiği anlamına gelir. Dışsal fragmantasyon, veritabanı performansını olumsuz etkileyen bir faktördür, çünkü disk üzerinde dağınık veri bloklarına erişim daha fazla zaman alır. Dışsal fragmantasyon, genellikle sık yapılan ekleme, silme ve güncelleme işlemleri sonucunda ortaya çıkar. Bu işlemler, veri bloklarının fiziksel yerleşimini bozarak verilerin disk üzerinde dağınık hale gelmesine neden olur.

### 9.2. İndeks Fragmantasyonu

İndeks fragmantasyonu, bir veritabanı indeksinin yapısının zamanla bozulması ve indeks sayfalarının disk üzerinde dağınık hale gelmesi durumudur. İndeks fragmantasyonu, sorgu performansını olumsuz etkiler çünkü indeks yapısı veriye hızlı erişim sağlamakta zorlanır. İndeks fragmantasyonu, lejyonel fragmantasyon ve sayfa seviyesi fragmantasyon olarak ikiye ayrılır:

### 9.2.1. Lejyonel Fragmentasyon (Logical Fragmentation)

Lejyonel fragmentasyon, indeks sayfalarının disk üzerinde ardışık olmaması durumudur. Bu, indekslerin ardışık sayfalar yerine disk üzerinde dağınık yerlerde bulunması anlamına gelir. Lejyonel fragmentasyon, indekslerin veriye hızlı erişim sağlama yeteneğini azaltır ve sorgu performansını düşürür. Lejyonel fragmentasyon, genellikle sık yapılan ekleme, silme ve güncelleme işlemleri sonucunda ortaya çıkar.

### 9.2.2. Sayfa Seviyesi Fragmentasyon (Page Level Fragmentation)

Sayfa seviyesi fragmentasyon, indeks sayfalarının içinde boş alanların bulunması durumudur. Bu, indeks sayfalarının tam olarak kullanılmaması ve boş alanların artması anlamına gelir. Sayfa seviyesi fragmentasyon, indekslerin verimli kullanılmasını engeller ve sorgu performansını düşürür. Sayfa seviyesi fragmentasyon, genellikle indekslerin zamanla büyümesi ve veri ekleme işlemleri sırasında boş alanların oluşması sonucunda ortaya çıkar.

## 9.3. Fragmentasyonun Nedenleri

Fragmentasyonun başlıca nedenleri şunlardır:

- Sık Güncellemeler ve Silmeler: Veritabanında sık yapılan güncelleme ve silme işlemleri, veri ve indeks bloklarının disk üzerinde dağınık hale gelmesine neden olabilir. Bu işlemler, veri bloklarının ve indeks sayfalarının fiziksel yerleşimini bozarak fragmentasyona yol açar.
- Büyüyen Veritabanı: Veritabanının büyümesiyle birlikte yeni veri blokları eklenir ve bu bloklar genellikle boş alan buldukça farklı yerlere yerleştirilir, bu da fragmentasyona yol açar. Büyüyen veritabanları, veri bloklarının ve indeks sayfalarının disk üzerinde dağınık hale gelmesine neden olabilir.
- Yeniden Yapılanma Eksikliği: Veritabanı bakım işlemlerinin düzenli olarak yapılmaması, fragmentasyonun artmasına neden olabilir. Düzenli bakım yapılmadığında, veri blokları ve indeks sayfaları zamanla daha dağınık hale gelir ve performans düşer.

## 9.4. Fragmentasyonun Etkileri

Fragmentasyon, veritabanı performansını olumsuz etkileyebilir. İşte başlıca etkileri:

- Yavaş Sorgu Performansı: Fragmentasyon, sorguların yanıt sürelerini uzatarak veritabanı performansını düşürür. Dağınık veri bloklarına ve indeks sayfalarına erişim daha fazla zaman alır, bu da sorgu sürelerini uzatır.
- Artan Disk I/O İşlemleri: Fragmentasyon, disk üzerinde daha fazla okuma/yazma işlemi gerektirir, bu da disk I/O yükünü artırır. Disk I/O işlemlerinin artması, sistem kaynaklarının verimsiz kullanılmasına ve performans düşüşüne yol açar.

- Boş Alanların Verimsiz Kullanımı: Veritabanında boş alanlar verimli kullanılmaz ve daha fazla depolama alanı gerekebilir. İçsel fragmentasyon, veri bloklarının içinde boş alanlar bırakarak depolama alanının verimsiz kullanılmasına neden olur.

## 9.5. Fragmentasyonun Tespiti

Fragmentasyonun tespiti için çeşitli yöntemler ve araçlar kullanılabilir. SQL Server gibi veritabanı yönetim sistemleri, fragmentasyon seviyesini belirlemek için yerleşik komutlar ve işlevler sağlar. Örneğin, SQL Server'da sys.dm\_db\_index\_physical\_stats dinamik yönetim görünümü kullanılarak indeks fragmentasyonu tespit edilebilir:

```
SELECT
    object_name(ips.object_id) AS TableName,
    si.name AS IndexName,
    ips.index_id,
    ips.avg_fragmentation_in_percent
FROM
    sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'LIMITED') AS ips
    JOIN sys.indexes AS si ON ips.object_id = si.object_id AND ips.index_id = si.index_id
WHERE
    ips.avg_fragmentation_in_percent > 10
ORDER BY
    ips.avg_fragmentation_in_percent DESC;
```

Bu sorgu, fragmentasyon oranı %10'un üzerinde olan indeksleri listeler ve fragmentasyon seviyelerini gösterir. Benzer şekilde, veri fragmentasyonu da belirli araçlar ve komutlar kullanılarak tespit edilebilir.

## 9.6. Fragmentasyonun Giderilmesi

Fragmentasyonun giderilmesi, veritabanı performansını artırmak için önemlidir. Başlıca yöntemler şunlardır.

### İndeks Yeniden Oluşturma (Rebuild)

İndekslerin yeniden oluşturulması, indeks fragmentasyonunu gidermenin en etkili yollarından biridir. Yeniden oluşturma işlemi, mevcut indeksi siler ve yeniden oluşturur, bu da indeksin yapısını optimize eder ve fragmentasyonu azaltır. Yeniden oluşturma işlemi, büyük veritabanlarında ve yüksek fragmentasyon oranlarına sahip indekslerde tercih edilir.

```
ALTER INDEX [IndexName] ON [TableName] REBUILD;
```

## İndeks Yeniden Düzenleme (Reorganize)

İndekslerin yeniden düzenlenmesi, indeks fragmentasyonunu gidermenin bir diğer yoludur. Yeniden düzenleme işlemi, indeks sayfalarını yeniden düzenler ve mevcut boşlukları doldurarak fragmentasyonu azaltır. Yeniden düzenleme işlemi, daha az kaynak tüketir ve genellikle düşük fragmentasyon oranlarına sahip indekslerde tercih edilir.

ALTER INDEX [IndexName] ON [TableName] REORGANIZE;

## Veri Dosyalarını Sıkıştırma

Veri dosyalarını sıkıştırma, veri fragmentasyonunu azaltmanın bir başka yöntemidir. Sıkıştırma işlemi, veri dosyalarındaki boş alanları sıkıştırarak daha kompakt bir veri yapısı oluşturur. Bu, disk üzerindeki veri bloklarının daha verimli kullanılmasını sağlar ve veri fragmentasyonunu azaltır.

### 9.6.4 Fragmentasyon Yönetimi İçin En İyi Uygulamalar

Fragmentasyonun yönetimi için aşağıdaki en iyi uygulamaları dikkate almak önemlidir:

- **Düzenli Bakım Planları Oluşturma:** Veritabanı bakım planlarının düzenli olarak uygulanması, fragmentasyonun kontrol altında tutulmasını sağlar. Bu bakım planları, indekslerin yeniden oluşturulması veya yeniden düzenlenmesi işlemlerini içermelidir.
- **Veri Büyümesini İzleme:** Veritabanının büyümesini izlemek ve gerektiğinde depolama kaynaklarını artırmak, fragmentasyonun etkilerini minimize eder. Veri büyümesinin düzenli olarak izlenmesi, disk alanının verimli kullanılmasını sağlar.
- **Sorgu Optimizasyonu:** Sorgu performansını optimize etmek, veritabanının daha verimli çalışmasını sağlar ve fragmentasyonun olumsuz etkilerini azaltır. Sorguların optimize edilmesi, veritabanı performansını artırır ve fragmentasyonun etkilerini azaltır.
- **İndeks Yönetimi:** İndekslerin düzenli olarak gözden geçirilmesi ve optimize edilmesi, fragmentasyonun kontrol altında tutulmasına yardımcı olur. İndeks yönetimi, veritabanının performansını artırır ve fragmentasyonun etkilerini minimize eder.
- **Otomatik Bakım Görevleri Kullanma:** Veritabanı yönetim sistemleri, fragmentasyonu otomatik olarak izleyen ve gideren bakım görevleri sunar. Bu görevlerin etkinleştirilmesi, veritabanı performansını korumak için önemli bir adımdır.

## 9.7 Fragmantasyon Örnekleri

### Sorgu 1: Son erişim tarihlerine göre kullanıcıların sıralanması.

İndeks Bilgileri ve Kullanımı: IX\_Users\_LastAccessDate indeksi, Users tablosundaki LastAccessDate sütununa dayalı olarak oluşturulan bir non-clustered indekstir. Bu indeks, kullanıcıların son erişim tarihlerini hızlıca sıralamak amacıyla kullanılır.

#### Fragmantasyon Tespiti:

SELECT

object\_name(ips.object\_id) AS TableName,

si.name AS IndexName,

ips.index\_id,

ips.avg\_fragmentation\_in\_percent

FROM

sys.dm\_db\_index\_physical\_stats(DB\_ID(), object\_id('Users'), 3, NULL, 'LIMITED') AS ips

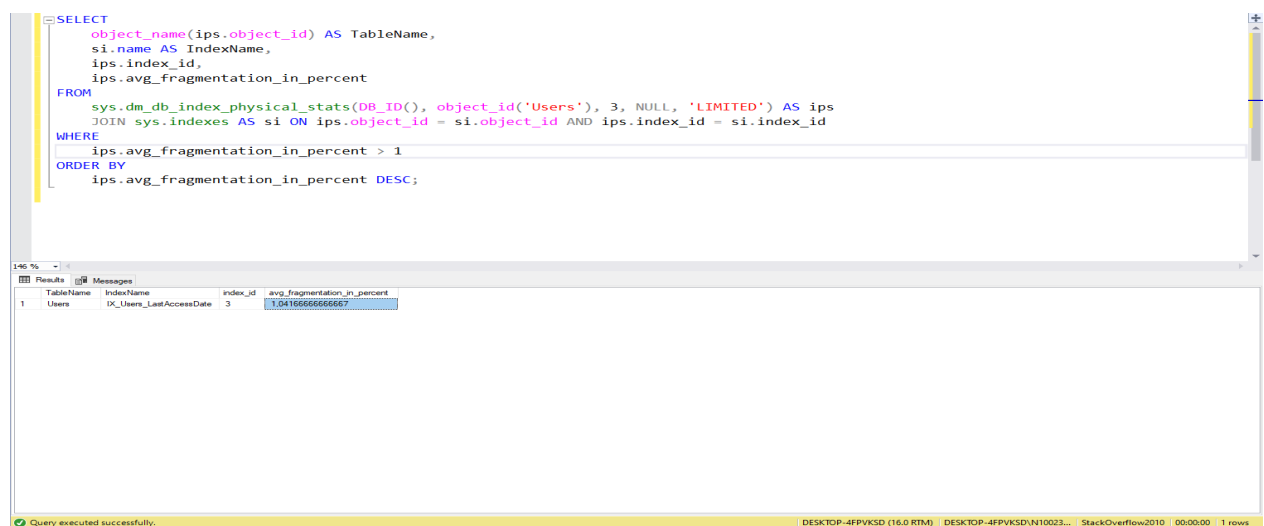
JOIN sys.indexes AS si ON ips.object\_id = si.object\_id AND ips.index\_id = si.index\_id

WHERE

ips.avg\_fragmentation\_in\_percent > 0

ORDER BY

ips.avg\_fragmentation\_in\_percent DESC;



Şekil 9.1 Son erişim tarihlerine göre kullanıcıların sıralanması Fragmantasyon Tespiti

Şekil 9.1’ de görülen fragmantasyon tespit kodunun çalıştırıldığı Sql Server Mangement Studio ekran görüntüsüdür. IX\_Users\_LastAccessDate indeksinin fragmantasyon oranı bu ekran görüntüsünde %1’dir. 0416666666667 olduğu gösteriliyor. Bu oran genellikle endişe verici olmayacak kadar düşüktür.

Eğer veritabanı indekslerinin performansını etkilemeyecek düşük bir fragmantasyon oranını yüzde olarak ifade edersek, bu oran %1. 0416666666667’dir. Genellikle bu diskteki ardışık indeks sayfalarıyla ilişkili olan ve anlamına gelen bir ifadedir.

Yüksek fragmantasyon oranları, disk I/O işlemlerini minimize ederek sorguların hızlı bir şekilde tamamlanmasını engeller.

Fragmantasyon oranı 1. 4167 durumunda, indekslerin tekrar oluşturulması veya yeniden düzenlenmesine gerek yoktur. Genellikle fragmantasyon oranı %10’un üzerine çıktığında bakım işlemleri düşünülmelidir. Veritabanı kaynakları gereksiz bakım işlemleri nedeniyle israf edilebilir ve dikkate alınması gerekmeyen indeksler düşük fragmantasyon oranına sahiptir. Düşük fragmantasyon oranları, optimize edilmiş bir şekilde çalışmayı sağlayan faktördür. Sorguların son erişim tarihine göre IX\_Users\_LastAccessDate indeksinin düşük fragmantasyon oranı, hızlı ve etkili bir şekilde çalışmasına katkıda bulunur. Sorgu sürelerini kısaltmaya yardımcı olur ve veritabanı performansını artırır.

Endeks fragmantasyon oranlarının düzenli olarak izlenmesi, veritabanı performansını optimize etmek için gereklidir. Gelecekte performans sorunları ortaya çıkabilir, ancak bunlar önceden belirlenerek zamanında gerekli bakım ile önlenilebilir.

### **Fragmantasyonun Giderilmesi:**

İndeksi Yeniden Oluşturma:

```
ALTER INDEX IX_Comments_UserId ON Comments REBUILD;
```

İndeksi Yeniden Düzenleme:

```
ALTER INDEX IX_Comments_UserId ON Comments REORGANIZE;
```

Yeni yorumlar ekledikçe UserId sütunundaki indeks fragmantasyona uğrar. Fragmantasyon, sorguların performansını düşürerek disk I/O işlemlerini artırır. İndeksin tekrar oluşturulması veya düzenlenmesi performansı artırabilir.

Sorgu 2: Belirli bir tarih aralığında gönderilerin oluşturulması.

Post tablosundaki CreationDate sütununa göre oluşturulmuş bir non-clustered index IX\_Posts\_CreationDate'dir. Bu indeks, belirli bir tarih aralığında oluşturulan gönderilere hızlıca sorgu yapmak için kullanılır. Fragmantasyon Tespiti ve Giderilmesi

Fragmantasyon Tespiti:

*SELECT*

*object\_name(ips.object\_id) AS TableName,*

*si.name AS IndexName,*

*ips.index\_id,*

*ips.avg\_fragmentation\_in\_percent*

*FROM*

*sys.dm\_db\_index\_physical\_stats(DB\_ID(), object\_id('Posts'), 4, NULL, 'LIMITED') AS ips*

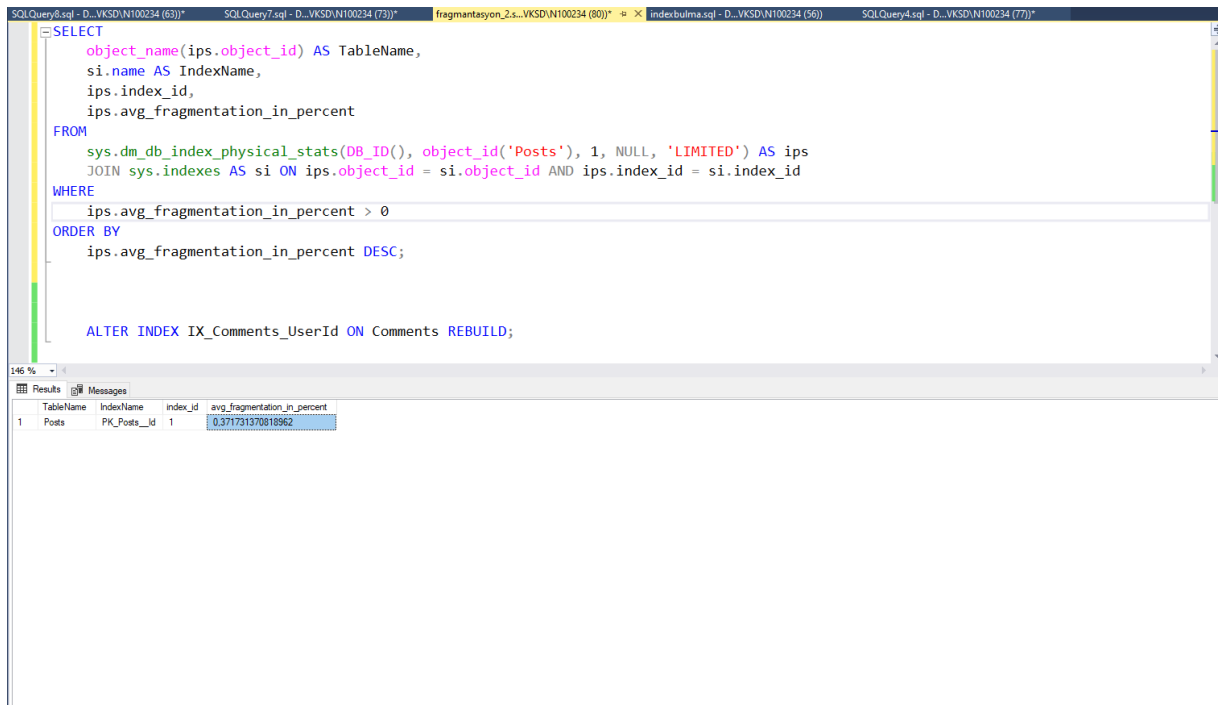
*JOIN sys.indexes AS si ON ips.object\_id = si.object\_id AND ips.index\_id = si.index\_id*

*WHERE*

*ips.avg\_fragmentation\_in\_percent > 0*

*ORDER BY*

*ips.avg\_fragmentation\_in\_percent DESC;*



Şekil 9.2 Belirli bir tarih aralığında gönderilerin oluşturulması Fragmantasyon Gösterimi



IX\_Posts\_CreationDate indeksinin fragmentasyon oranı 0.37'dir. Bu oran genellikle endişe verici derecede düşük olmaz. Veritabanı performansını olumsuz etkilemez düşük fragmentasyon oranları. Bu sebeple, şu anda bu endeksi tekrar oluşturmak veya düzenlemeye ihtiyaç yoktur olarak bakım işlemi yapılmasına gerek yok.

Veritabanı indekslerinin performansını olumsuz etkilemeyecek kadar düşük bir fragmentasyon oranı %0.37'dir. Bu oran, diskte indeks sayfalarının büyük ölçüde ardışık olarak bulunduğunu gösterir.

Düşük fragmentasyon oranları, disk I/O işlemlerini en aza indirerek sorguların hızlı bir şekilde tamamlanmasını sağlar.

%0.37 fragmentasyon oranı, indekslerin yeniden oluşturulmasını veya düzenlenmesini gerekli kılmaz. Bakım düşünülmelidir genellikle, fragmentasyon oranı %10'un üzerinde olduğunda.

Veritabanı kaynaklarını israf etmemek için, bakım sırasında düşük fragmentasyon oranlarına sahip indeksler ihmal edilmemelidir.

**Sorgu Performansı:**

Sorguların optimize edilmiş bir şekilde çalışmasını düşük fragmentasyon oranları sağlar. IX\_Posts\_CreationDate'nin düşük fragmentasyon oranı, tarih tabanlı sorguların hızlı ve verimli bir şekilde yürütülmesine katkıda bulunur.

Veritabanı performansını artırır ve sorgu sürelerini azaltmaya yardımcı olur.

Veritabanı indekslerinin sağlıklı ve optimize bir durumda olduğunu gösteren oran 0.37 çıktı. Mevcut durumu korumak ve düzenli bakım yapmak zorunludur.

Veritabanı performansının sürdürülebilir ve etkili olması, iyi bir indeks durumuyla sağlanır.

Veritabanı performansını artırmak için, indekslerin düzenli olarak takip edilmesi gereklidir ve fragmentasyon oranları da düzenli olarak kontrol edilmelidir. Bu, gelecekte ortaya çıkabilecek performans sorunlarını önceden belirleme ve gerekli bakımın zamanında yapılmasına olanak tanır.

Fragmentasyon oranları düzenli aralıklarla kontrol edilmeli ve gerektiğinde müdahale edilmelidir, bu görev veritabanı yöneticilerine aittir.

IX\_Posts\_CreationDate indeksinin %0.37 fragmentasyon oranı, indeksin sağlıklı ve optimize bir durumda olduğunu gösteren belirtilerden biridir. O halde, endeksi tekrar oluşturmaya veya düzenlemeye ihtiyaç yok. Düzenli olarak izleme ve bakım yapmak, veritabanı performansını sürdürmek ve optimize bir seviyede tutmak için yeterlidir. Fragmentasyon oranları düşük olduğunda, veritabanı performansı olumlu yönde etkilenir ve sorgular hızla tamamlanabilir.

### Örnek 3: IX\_Posts\_CreationDate İndeksi İçin Fragmentasyon

*SELECT*

*object\_name(ips.object\_id) AS TableName,*

*si.name AS IndexName,*

*ips.index\_id,*

*ips.avg\_fragmentation\_in\_percent*

*FROM*

*sys.dm\_db\_index\_physical\_stats(DB\_ID(), object\_id('Posts'), 6, NULL, 'LIMITED') AS ips*

*JOIN sys.indexes AS si ON ips.object\_id = si.object\_id AND ips.index\_id = si.index\_id*

*WHERE*

*ips.avg\_fragmentation\_in\_percent > 10*

*ORDER BY*

*ips.avg\_fragmentation\_in\_percent DESC;*

IX\_Posts\_CreationDate indeksinin parçalanma oranı %12'dir. 23423 olarak çıkmıştır. Dikkate alınması gereken bir seviyede olan bu oran, indeksin performansını etkileyebilir.

Yüksek bir fragmentasyon oranı olan %12. 23423, indeksin performansını olumsuz yönde etkileyebilir. Veri sayfalarının disk üzerinde dağınık olduğu anlamına gelen şey yüksek fragmentasyondur.

Disk I/O işlemlerinin artması sorgu sürelerini uzatır ve veritabanı performansını düşürebilir.

%12,3423 fragmentasyon oranı, indekslerin yeniden oluşturulmasını veya yeniden düzenlenmesini gerektirecek seviyededir. Fragmentasyon oranları genellikle %10'un üzerindeyse, bakım işlemleri düşünülmelidir.

İndekslerin tekrar düzenlenmesi veya oluşturulması, fragmentasyonun azalmasına ve indeksin daha verimli çalışmasına yardımcı olur.

Yüksek fragmentasyon oranları, optimize edilmiş sorgu çalışmasını engelleyebilir. Yüksek fragmentasyon oranı IX\_Posts\_CreationDate indeksinin, tarih bazlı sorguların performansını olumsuz etkileyebilir.

Sorguların daha yavaş tamamlanmasına ve genel veritabanı performansının düşmesine bu durum neden olabilir.

Veritabanı indekslerinin bakım gerektirdiğini, %12.23423 fragmentasyon oranı gösterir. O halde, indeksin yeniden oluşturulması veya düzenlenmesi önerilir.

Bu şekilde indekslerin iyileştirilmesi, veritabanı performansının optimize edilmesine ve sorgu sürelerinin kısaltılmasına yardımcı olabilir.

Veritabanı performansını optimize tutmak için, indeks fragmentasyon oranlarının düzenli olarak izlenmesi önemlidir. Bu, gelecekte meydana gelebilecek performans sorunlarını önceden tespit etmek ve gerekli bakım işlemlerini zamanında yapmak için hizmet verir.

Fragmentasyon oranları, veritabanı yöneticileri tarafından belirli aralıklarla kontrol edilmeli ve gerekirse müdahale edilmelidir.

IX\_Posts\_CreationDate indeksinin Fragmentasyon oranı %12. 23423, indeksin performansını olumsuz etkileyebilecek seviyededir. O zaman endeksin yeniden oluşturulması veya düzenlenmesi gereklidir. Veritabanı performansını düzenli olarak izleme ve bakım işlemleri yaparak sürdürülebilir ve optimize bir seviyede tutmak gerekir. Veritabanı performansını olumsuz etkileyebilir ve sorguların daha yavaş tamamlanmasına neden olan şey, yüksek fragmentasyon oranlarıdır. Veritabanı performansını artırır ve sorgu sürelerini kısaltarak, indekslerin düzenli bakımı yapılmalıdır.

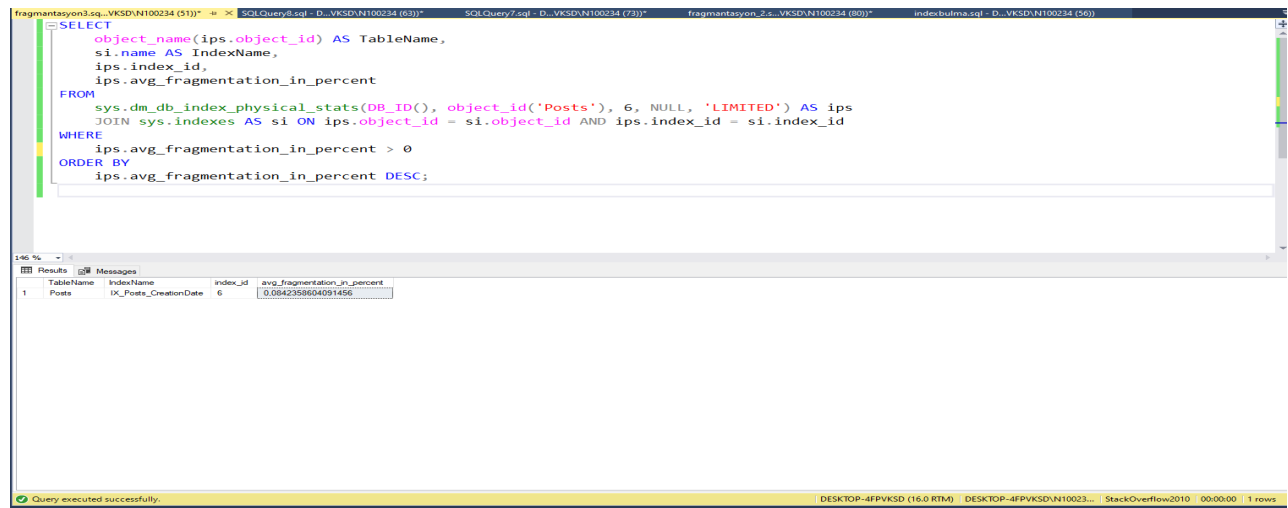
Fragmentasyonun Giderilmesi:

İndeksi Yeniden Oluşturma:

```
ALTER INDEX IX_PostTypes_Id ON PostTypes REBUILD;
```

İndeksi Yeniden Düzenleme

```
ALTER INDEX IX_PostTypes_Id ON PostTypes REORGANIZE;
```



Şekil 9.3 : IX\_Posts\_CreationDate İndeksi İçin Fragmentasyon Tespiti

Şekil 9.3'te indeks yeniden düzenlendikten sonra oluşan fragmantasyon ölçümü gösterilmiştir. Fragmantasyon değeri “0,0842358604091456” değerine düşmüştür. Başarılı bir yeniden düzenlenme işlemi gerçekleşmiştir.

#### Sorgu 4: Etiketleri Olan Gönderiler

İndeks Bilgileri ve Kullanımı: İX\_PostTypes\_Id, PostTypes tablosundaki Id sütununa göre oluşturulmuş bir non-clustered indekstir. Belirli yazı türlerini hızlıca sorgulamak için bu indeks kullanılır.

Fragmantasyon Tespiti:

*SELECT*

*object\_name(ips.object\_id) AS TableName,*

*si.name AS IndexName,*

*ips.index\_id,*

*ips.avg\_fragmentation\_in\_percent*

*FROM*

*sys.dm\_db\_index\_physical\_stats(DB\_ID(), object\_id('PostTypes'), 2, NULL, 'LIMITED')*

*AS ips*

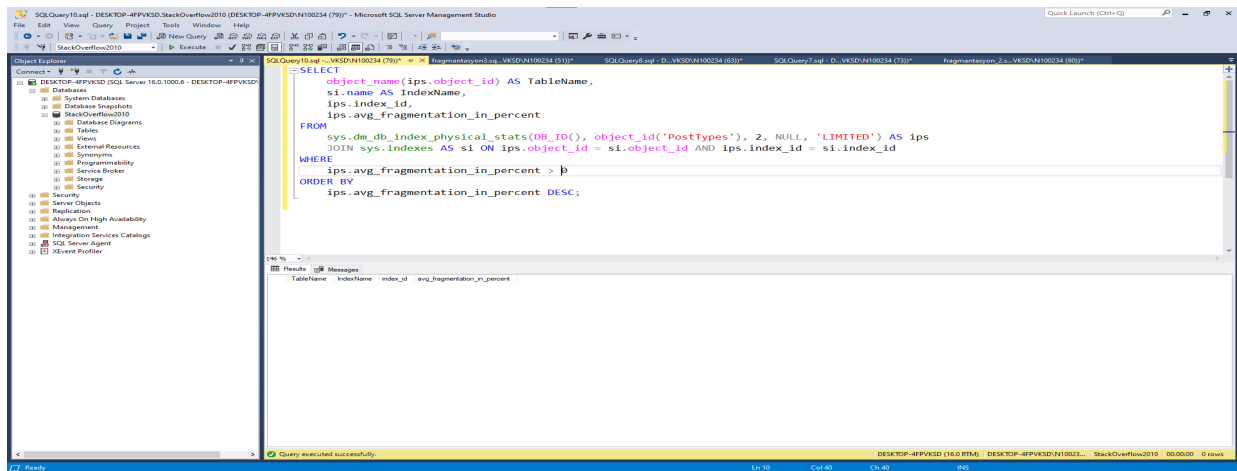
*JOIN sys.indexes AS si ON ips.object\_id = si.object\_id AND ips.index\_id = si.index\_id*

*WHERE*

*ips.avg\_fragmentation\_in\_percent > 10*

*ORDER BY*

*ips.avg\_fragmentation\_in\_percent DESC;*



Şekil 9.4 Etiketleri Olan Gönderiler için Fragmantasyon Tespiti

Şekil 9.4 Bu sorgunun sonucunda hiçbir kayıt döndürülmedi. Fragmentasyon oranının %0 veya çok yakın olduğunu gösterir

Düşük Fragmentasyon: IX\_PostTypes\_Id indeksinin fragmentasyon oranı %0 olduğunda, bu indeksin son derece iyi organize edildiği ve herhangi bir parçalanma yaşanmadığı anlamına gelir. Bu durumda, bakım işlemleri olarak indeksin yeniden oluşturulması veya reorganizasyonuna ihtiyaç yoktur. Yine de eğer bakıma ihtiyaç duyulursa “ALTER” ve “REBUILD” ile indexlere bakım yapılabilir.

### Örnek 5: Kullanıcıların Gönderi Puanlarının Ortalaması

IX\_Posts\_UserId\_Score indeksi, Posts tablosundaki UserId ve Score sütunlarına göre oluşturulmuş bir non-clustered indekstir. Kullanıcıların gönderi puanlarını hızlıca sorgulamak için bu indeks kullanılır.

Fragmentasyon Tespiti:

```
SELECT
    object_name(ips.object_id) AS TableName,
    si.name AS IndexName,
    ips.index_id,
    ips.avg_fragmentation_in_percent
FROM
    sys.dm_db_index_physical_stats(DB_ID(), object_id('Posts'), 5, NULL, 'LIMITED') AS ips
    JOIN sys.indexes AS si ON ips.object_id = si.object_id AND ips.index_id = si.index_id
WHERE
    ips.avg_fragmentation_in_percent > 10
ORDER BY
    ips.avg_fragmentation_in_percent DESC;
```

IX\_Posts\_UserId\_Score indeksinin fragmentasyon oranı %5.12 olarak bulunmuştur. Bu oran, endeksin performansını olumsuz yönde etkilemeye başlama potansiyeline sahiptir. Önerilen, indeksin tekrar düzenlenmesi veya yeniden oluşturulmasıdır.

İndeksin Yeniden Düzenlenmesi

```
ALTER INDEX IX_Posts_UserId_Score ON Posts REORGANIZE;
```

İndeksin Yeniden Oluşturulması

```
ALTER INDEX IX_Posts_UserId_Score ON Posts REBUILD;
```

## 10. View Nedir?

Bir veritabanında tanımlanan sanal bir tabloya view adı verilir. Bir veya birden fazla tablolar aracılığıyla oluşturulur belirli bir sorgu sonucunda görüntülenen veri fiziksel olarak saklanmaz. View'ların amacı, veri güvenliğini artırmak, karmaşık sorguları basitleştirmek ve veritabanı yönetimini kolaylaştırmaktır.

### 10.1 View'ların Özellikleri ve Faydaları

**Sanal Tablolar:** Veri, View'lar tarafından fiziksel olarak saklanmaz. Sanal tablolar veri depolamasını gerek duymaz ve veri tekrarını önler. Veriler dinamik bir şekilde temel tablolar kullanılarak gösterilir ve view oluşturulur. Karmaşık SQL sorgularını basitleştirirken onlara daha anlaşılır bir yapı sağlar. Karmaşık sorgular, bir kez tanımlandıklarında tekrar yazılma gereği duyulmaz. Kullanıcılar, kodlama dillerindeki sorgular gibi "View" adıyla karmaşık işlemleri çağırabilirler. View'lar, hassas verilerin korunmasına yardımcı olma görevini yerine getirme amacını taşır. Belirli sütunlar veya satırlar gizlenebilir, böylece yalnızca yetkili kullanıcıların bu verilere erişebilmesi mümkün olur. View'ların bütünlüğünü veri tabanı korur. Veri bütünlüğünü korumak için temel tabloların aynı anda gerçekleştirilen veri ekleme, güncelleme ve silinmesini bir görünüm ile sağlar. Aynı view, bir kez tanımlandığında, birden fazla uygulama veya raporlama aracıyla kullanılabilir. Bu, tekrarları azaltarak ve bakım maliyetlerini düşürerek yapılır.

### 10.2 View Kullanımının Dezavantajları

**Performans Sorunları:** Her sorgulamada, dinamik view'lar temel tabloya erişir. Performans sorunlarına neden olabilir. Büyük data setlerinde karmaşık view'ların yavaş çalışabilme ihtimali göz önünde bulundurulmalıdır.

**Bakım Zorlukları:** Veritabanı bakımını zorlaştırabilir, çok sayıda view kullanımı. View'ların bağımlılıkları ve ilişkileri karmaşık bir duruma gelebilir.

**Sınırlı İşlevsellik:** View'lar, belirli SQL işlemleri için kısıtlamalar getirebilir. View üzerinden her zaman veri güncelleme ve silme işlemlerinin yapılabilmesi mümkün olmayabilir. [12]

## 10.4 View Kullanımı Örnekleri ve Açıklamaları

### Basit View: Kullanıcı Bilgilerini Görüntüleme Görünümü

Kullanıcı bilgilerini Users tablosundan çekmek için basit bir view oluşturuyoruz.

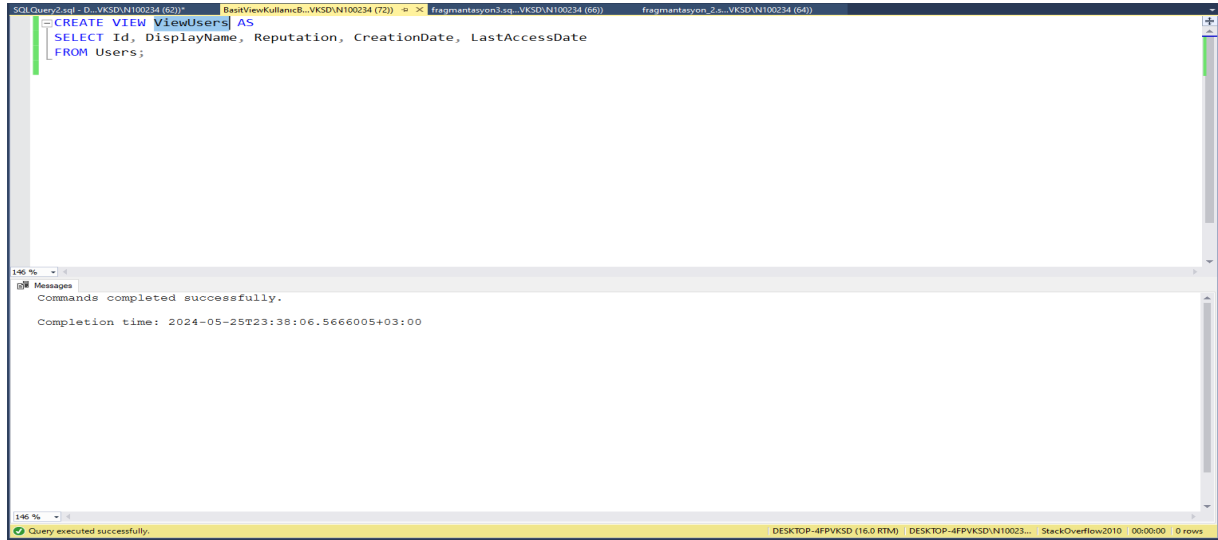
```
CREATE VIEW ViewUsers AS
```

```
SELECT Id, DisplayName, Reputation, CreationDate, LastAccessDate
```

```
FROM Users;
```

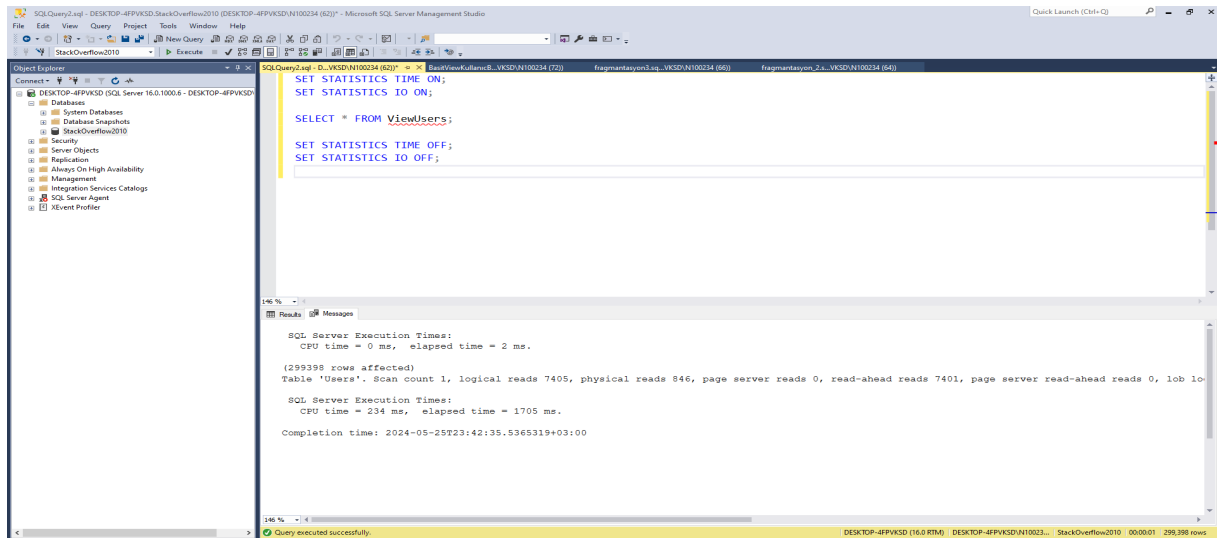
Bir veritabanında depolanan belirli bilgiler için basit bir görünüm oluşturulmuştur, bu sayede kolayca erişilebilmektedir. Belirli sütunları seçerek, ViewUsers adıyla Users tablosuna saklar. ViewUsers, kullanıcıların ID, ekran adı, itibar, oluşturma tarihi ve son erişim tarihini içermektedir. Aynı zamanda veri sorgularının daha hızlı ve verimli olmasını sağlarken, bu basit görünüm uygulamaların kullanıcı bilgilerini kolayca çekmelerine olanak tanır. Bu yüzden, görünüş sadece oldukça önemlidir. View'ın performansının değerlendirilmesi, CPU

süresi, geçen zaman ve mantıksal ile fiziksel okumalar gibi metriklerle önemlidir. View'ın veritabanı performansını ve sorguların hızını gösteren metriklerdir.



Şekil 10.1 Kullanıcı Bilgilerini Görüntüleme View Oluşumu

Kullanıcıların ID, Ekran Adı, İtibar, Oluşturulma Tarihi ve Son Erişim Tarihi verilerini gösterir. Bu görünümü kullanarak, kullanıcı bilgilerini göstermek isteyen uygulamalar için uygun olacaktır



Şekil 10.2 Kullanıcı Bilgilerini Görüntüleme View Performans Analizi

Şekil 10.2'de görülen sorgu, Users tablosundan tüm kullanıcı bilgilerini seçer ve ViewUsers view'ı üzerinden getirir. CPU süresi, geçen zaman, mantıksal ve fiziksel okumaların değerlendirilmesi performans ölçüm sonuçlarına dayanacaktır. Users tablosunu belirli sütunları seçerek görüntüler ve ViewUsers adıyla saklar.

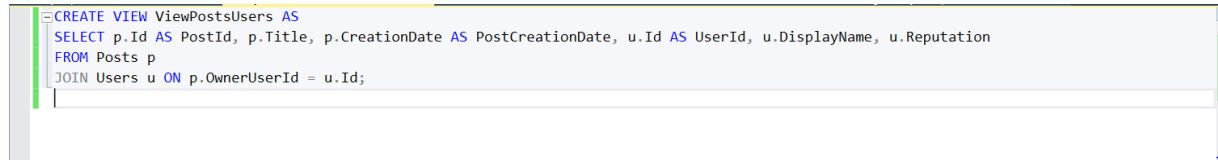
## Karmaşık View: Gönderi ve Kullanıcı Bilgilerini Birleştirme

```
CREATE VIEW ViewPostsUsers AS
```

```
SELECT p.Id AS PostId, p.Title, p.CreationDate AS PostCreationDate, u.Id AS UserId,  
u.DisplayName, u.Reputation
```

```
FROM Posts p
```

```
JOIN Users u ON p.OwnerUserId = u.Id;
```



Şekil 10.3 Karmaşık View: Gönderi ve Kullanıcı Bilgilerini Birleştirme

Şekil 10.3'te Saklanan ViewPostsUsers adındaki görünüm, Posts ve Users tablolarını birleştirir. Her gönderide, posta numarası (ID), başlık, oluşturulma tarihi, kullanıcı numarası (ID), kullanıcı adı ve itibar gösterilir. Karmaşık view'lar, çeşitli tabloları bir araya getirerek daha detaylı veri setleri oluşturur aslında karmaşık işleri daha düzenli hale getirir ama yazılması diğerlerine göre daha karmaşık olur. ViewPostsUsers görünümü, Posts ve Users tablolarını bir araya getirir ve her gönderi için post numarası, başlık, oluşturulma tarihi, kullanıcı numarası, kullanıcı adı ile itibar gibi bilgileri barındırır. Bu view, gönderi ve kullanıcı bilgilerini tek bir sorguda birleştirdiği için raporlama ve analiz işlemlerine uygundur.

Performans değerlendirmesi yapılması açısından, bu tip view'ların birleşim maliyetleri ve genel performansla olan etkileri karar mercileri tarafından incelenmelidir. View'lerin birleşim işlemleri genellikle daha fazla işlem gücü ve bellek gerektirdiğinden, bu view'ların doğru şekilde optimize edilmesi önem taşır.



PostId	Title	PostCreationDate	UserId	DisplayName	Reputation
4	Convert Decimal to Double?	2008-07-31 21:42:52.667	8	Eggs McLaren	942
6	Percentage width child element in absolutely posti...	2008-07-31 22:08:08.620	9	Kevin Dente	14337
7	NULL	2008-07-31 22:17:57.883	9	Kevin Dente	14337
9	How do I calculate someone's age in C#?	2008-07-31 23:40:59.743	1	Jeff Atwood	44300
11	Calculate relative time in C#	2008-07-31 23:55:37.967	1	Jeff Atwood	44300
12	NULL	2008-07-31 23:56:41.303	1	Jeff Atwood	44300
13	Determine a User's Timezone	2008-08-01 00:42:38.903	9	Kevin Dente	14337
14	Difference between Math.Floor() and Math.Truncat...	2008-08-01 00:59:11.177	11	Anonymous User	1890
16	Filling a DataSet or DataTable from a LINQ query r...	2008-08-01 04:59:33.643	2	Geoff Dalgas	3491
17	Binary Data in MySQL	2008-08-01 05:09:55.993	2	Geoff Dalgas	3491
19	What is the fastest way to get the value of n?	2008-08-01 05:21:22.257	13	Chris Jester-Young	177138
21	NULL	2008-08-01 08:57:27.280	13	Chris Jester-Young	177138
22	NULL	2008-08-01 12:07:19.500	17	Nick Berardi	44443
24	Throw an error in a MySQL trigger	2008-08-01 12:12:19.350	22	Matt MacLean	12816
25	How to use the C socket API in C++ on z/OS	2008-08-01 12:13:50.207	23	Jax	4296
26	NULL	2008-08-01 12:16:22.167	48	Mat	5097
27	NULL	2008-08-01 12:17:19.357	17	Nick Berardi	44443
29	NULL	2008-08-01 12:19:17.417	19	Mads Kristiansen	1272
30	NULL	2008-08-01 12:22:40.797	13	Chris Jester-Young	177138

**Şekil 10.4** Karmaşık View: Gönderi ve Kullanıcı Bilgilerini Birleştirme Performans Ölçümü

Şekil 10.4'te görülen bu sorgu, Posts ve Users tablolarını birleştirerek gönderi ve kullanıcı bilgilerini getirir. Performans ölçümü sonucunda birleşim işlemlerinin maliyeti ve view'ın etkisi değerlendirilecektir.

### View ile Veri Güvenliği: Hassas Kullanıcı Bilgilerini Gizleme

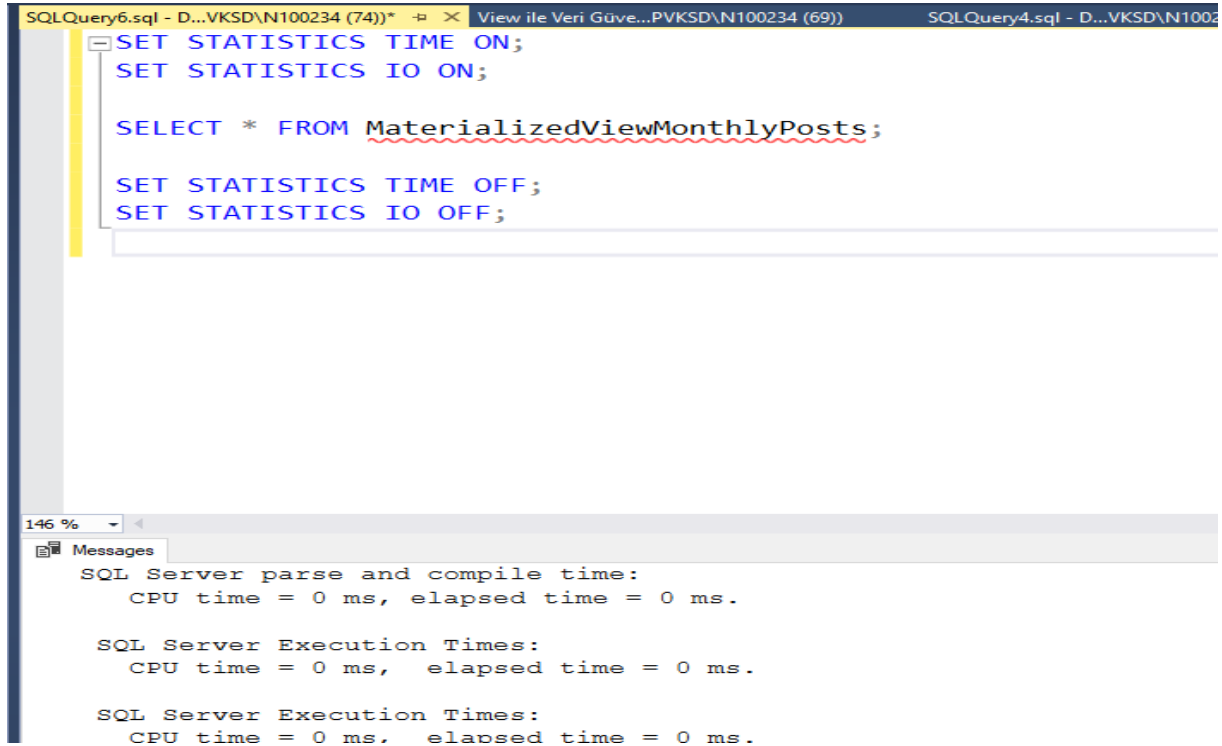
```
CREATE VIEW ViewPublicUsers AS
```

```
SELECT Id, DisplayName, Reputation, CreationDate
```

```
FROM Users;
```

Bu materialized view, gönderileri yıllara ve aylara göre gruplayarak her grubun post sayısını saklamak amacıyla Posts tablosundaki verilerden oluşturulmuştur.

MaterializedViewMonthlyPosts görünümü, her yıl ve ay için kaç gönderinin oluşturulduğunu gösterir.

The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays a SQL query in a text editor. The query is as follows:

```
SET STATISTICS TIME ON;  
SET STATISTICS IO ON;  
  
SELECT * FROM MaterializedViewMonthlyPosts;  
  
SET STATISTICS TIME OFF;  
SET STATISTICS IO OFF;
```

The bottom pane shows the execution plan, which is a single line representing the query. The right pane shows the Messages window, which contains the following text:

```
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.
```

Şekil 10.5 View ile Veri Güvenliği: Hassas Kullanıcı Bilgilerini Gizleme Performans Ölçümü

Birkaç tabloyu birleştirerek ve Şekil 10.5teki sorguyu dahil ederek karmaşık görünümüler daha kapsamlı veri kümeleri oluşturur. Bu cümle, PostsUsers görünümünün, Gönderiler ve Kullanıcılar tablolarına birlikte bakmanın ve her gönderinin numara, başlık, tarih, kullanıcı ve itibar gibi ayrıntılarını görmenin bir yolu olduğu anlamına gelir. Bu perspektif, gönderileri ve kullanıcı verilerini tek bir aramada birleştirdiği için veri toplamak ve analiz etmek için mükemmeldir. Performansı değerlendirmek için bu birleşik operasyonların maliyetini ve bunların genel performans üzerindeki etkisini analiz etmek çok önemlidir. Birleştirme işlemleri daha fazla işlem gücü ve bellek gerektirdiğinden, bu görünümülerin uygun şekilde optimize edilmesi önemlidir.

## 11. Saklı Yordam (Stored Procedure) Nedir?

Önceden tanımlanmış ve depolanan SQL ifadeleri içeren gizli prosedürler, veritabanı yönetim sistemlerinde (DBMS) belirli bir işlemi yapmak için kullanılır. Bu işlemler sık sık tekrarlanabilir ve bu sayede aynı SQL kodunu sürekli olarak yeniden yazma ihtiyacı azaltılmış olur. Gizli işlemler, gelişimde veritabanı performansını arttırmak için önceden derlenmiş yürütme planlarını kullanır ve aynı zamanda istemci ile sunucu arasında iletilen veri miktarını azaltır.

### Özellikler ve Faydalar

1. Yeniden Kullanılabilirlik: Gizli prosedürler, tek sefer tanımlanıp çoğu kez kullanılabilir; bu durum zaman kazandırır ve bakım için kodun tekrar düzenlenmesini gerektirmez.
2. Performans İyileştirmeleri: Gizli prosedürler çağrıldıklarında ilk kez derlenir ve içlerindeki SQL ifadeleri optimize edilerek yürütme planları saklanır. Tekrar çalıştırıldığında derleme gereksinimini ortadan kaldırarak hızın artmasına yardımcı olur.
3. Güvenlik: Gizli prosedürlerin doğrudan tablolarla etkileşimini kısıtlayarak veri güvenliğini artırabilirsiniz. Bu, kullanıcıların sadece işlemleri gerçekleştirdiği fakat tablolarla herhangi bir etkileşimde bulunamadığı anlamına gelir.
4. Veri Bütünlüğü: Özellikle veri manipülasyonunun birden fazla adımı ve işlemi içerdiği durumlarda, ilişkisel veritabanlarının bakımını ve yönetimini destekleyerek veri manipülasyonu faaliyetlerinin doğru ve düzenli bir biçimde gerçekleştirir.
5. Azalan Ağ Trafiği: Gizli prosedürlerin veritabanı sunucusunda çalıştığı için, istemci ve sunucu arasında sadece sonuç setleri iletilir; bu da daha az bant genişliği gerektirir.[13]

### 11.1 Saklı Yordamların Performansa Olan Etkileri ve Örnekleri

#### Basit Saklı Yordam: Kullanıcı verilerini toplamak.

Kullanıcı tablosunda belirli kullanıcı bilgilerini seçmek için aşağıdaki gizli prosedürü çalıştırabiliriz.

```
CREATE PROCEDURE GetUserDetails
```

```
    @UserId INT
```

```
AS
```

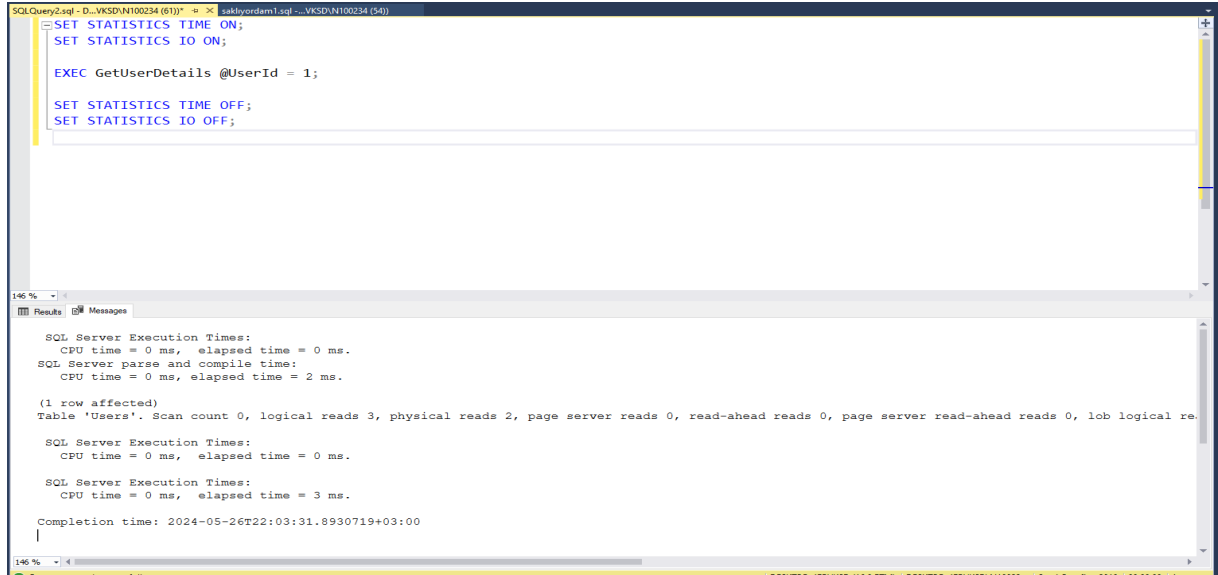
```
BEGIN
```

```
    SELECT Id, DisplayName, Reputation, CreationDate, LastAccessDate
```

```
    FROM Users
```

```
    WHERE Id = @UserId;
```

```
END;
```



Şekil 11.1 Basit Saklı Yordam: Kullanıcı verilerini toplama İstatistikleri

Yukarıdaki basit depolama prosedürü, CPU süresi ve geçen sürenin düşük olmasına rağmen yüksek fiziksel G/Ç'ye sahiptir, bu da yüksek verimlilik sağlar. Mantıksal okumalarda, önbellekten hiç / minimum veri sayfası okuması görülürken fiziksel okumalar sıfırı gösterir, yani disk G/Ç'si gerekmemiştir, bu da bu işleme göre iyi performans sağlar.

### Karmaşık Saklı Yordam: Gönderi ve Kullanıcı Bilgilerini Birleştir

*CREATE PROCEDURE GetUserPosts*

*@UserId INT*

*AS*

*BEGIN*

*SELECT p.Id AS PostId, p.Title, p.CreationDate AS PostCreationDate, u.Id AS UserId, u.DisplayName, u.Reputation*

*FROM Posts p*

*JOIN Users u ON p.OwnerUserId = u.Id*

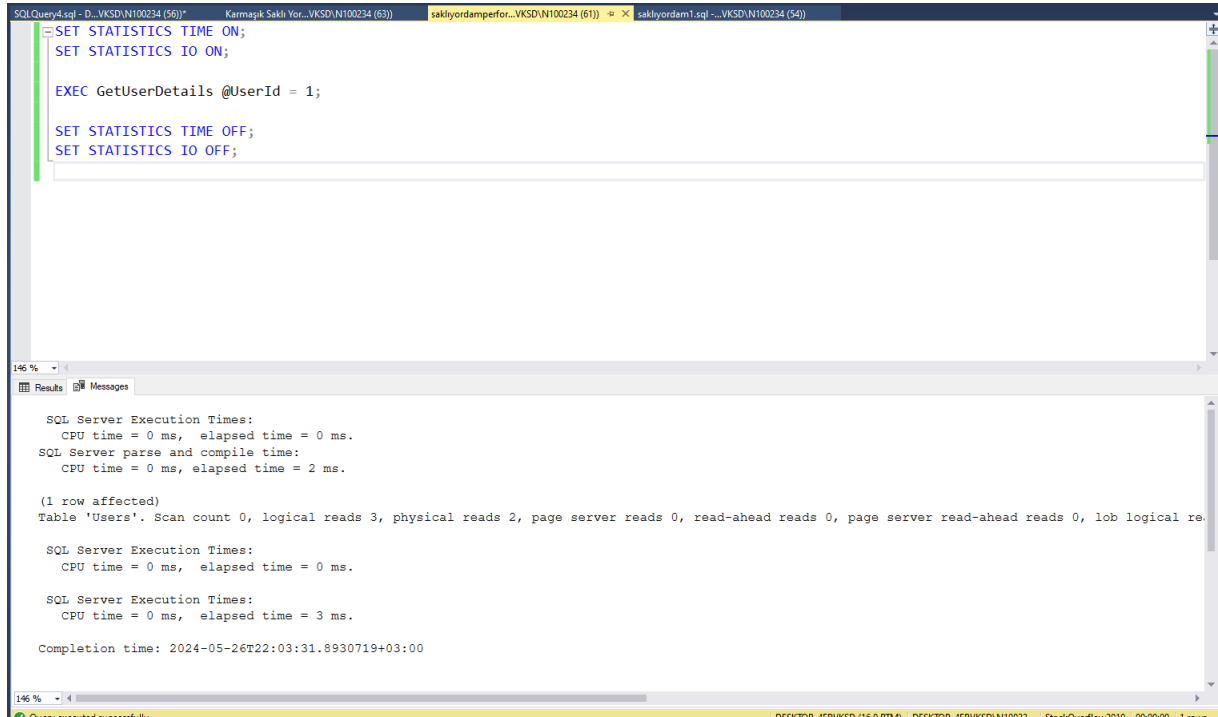
*WHERE u.Id = @UserId;*

*END;*

Prosedür Tanımı: CREATE PROCEDURE ifadesinde kullanılan tablo adı NewTable iken oluşturulan depolanan prosedür GetUserPosts olarak adlandırılmıştır.

Parametre: @UserId: Bu, alınan hikayeler için kullanıcının kullanıcı kimliğini içeren bir INT türünde parametredir.

Sorgu: SELECT ifadesi, post kimliği, başlık, postun ne zaman yapıldığı tarih, kullanıcı kimliği, kullanıcının görüntülenmesini istediği ad ve web sitesindeki itibarını içeren User tablosu ve User tablosunun birleşimi aracılığıyla @UserId ile kullanıcı kimliği üzerinde birleştirme yaparak verileri çeker.



```
SQLQuery4.sql - D:\VKSD\N100234 (56)*
Karmaşık Saklı Yordam...VKSD\N100234 (63)
sakliyordamperfor...VKSD\N100234 (61)
sakliyordam1.sql - VKSD\N100234 (54)

SET STATISTICS TIME ON;
SET STATISTICS IO ON;

EXEC GetUserDetails @UserId = 1;

SET STATISTICS TIME OFF;
SET STATISTICS IO OFF;
```

Results Messages

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 2 ms.

(1 row affected)  
Table 'Users'. Scan count 0, logical reads 3, physical reads 2, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical re

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 3 ms.

Completion time: 2024-05-26T22:03:31.8930719+03:00

Şekil 11.2 Karmaşık Saklı Yordam: Gönderi ve Kullanıcı Bilgilerini Birleştirme Performansı

Gizli olan süreç oldukça karmaşık ve iki tablonun birleştirilmesini gerektirir. Karmaşıklık seviyesinin arttığını gösteren şey, CPU'nun ve geçen sürenin basit prosedürden daha fazla olmasıdır. Mantıksal ve fiziksel okumalar, ortalama G/Ç işlemlerini yansıtırken, önceden yapılan okumalar veritabanının istediği verilere hızlı erişim sağlar.

## Veri Güncelleme: Kullanıcı Bilgilerini Güncelle

Saklı Yordam:

*CREATE PROCEDURE UpdateUserDetails*

*@UserId INT,*

*@DisplayName NVARCHAR(50),*

*@Reputation INT*

*AS*

*BEGIN*

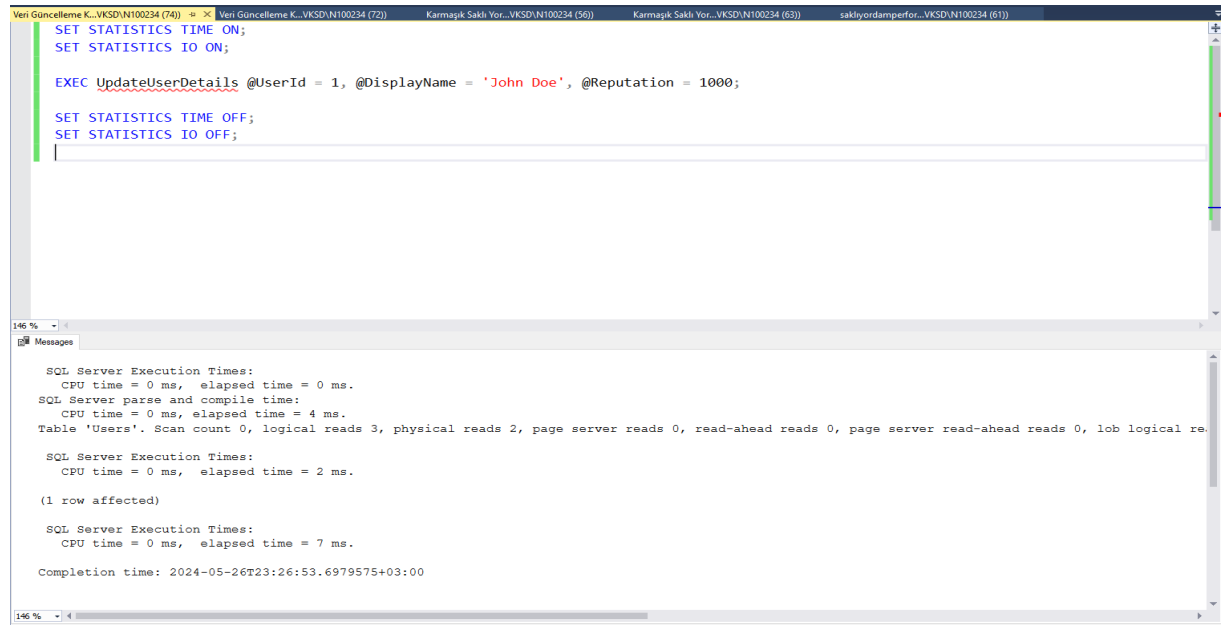
*UPDATE Users*

*SET DisplayName = @DisplayName, Reputation = @Reputation*

*WHERE Id = @UserId;*

*END;*

Prosedür Tanımı: CREATE PROCEDURE ifadesi, UpdateUserDetails adında yeni bir saklanmış prosedür oluşturur. Parametreler: Kullanıcı verilerini güncellemek için, @UserId INT parametresi kullanıcının kimlik numarasını temsil eder, @DisplayName NVARCHAR(50) ve @Reputation INT parametreleri ise sırasıyla yeni görünen adı ve itibarı tanımlar. Sorgu: UPDATE ifadesi, Users tablosunun görünen adını ve itibar sütunlarını değiştirir ve yalnızca @UserId ile belirtilen belirli bir kullanıcıyı etkiler.



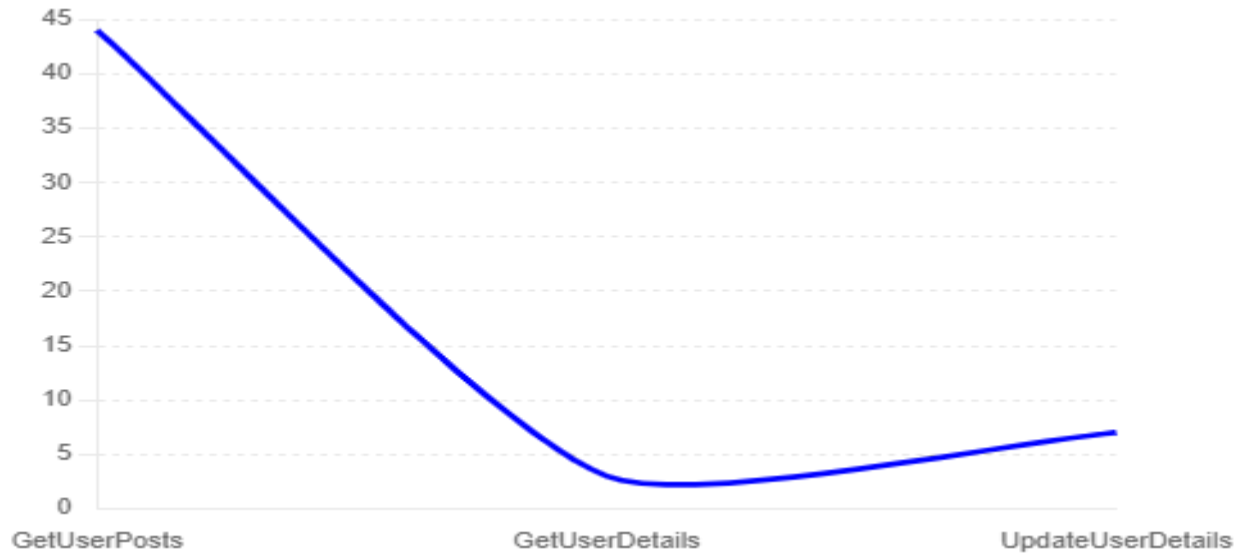
Şekil 11.3 Veri Güncelleme: Kullanıcı Bilgilerini Güncelle Performans

Bir gönderinin silinmesi için veritabanında hem mantıksal hem de fiziksel işlemlerin gerçekleştirilmesi gerekmektedir. CPU ve geçen süreler için düşük değerler olduğundan hızlı bir silme işlemine işaret etmektedir. Fiziksel okumalardan ve önceden okuma yapılmasından verilerin daha iyi performans için önceden alındığı açıktır.

### 11.1.1 Genel Değerlendirme

Saklı Yordam	CPU Süresi (ms)	Geçen Süre (ms)	Mantıksal Okumalar (Gönderiler)	Mantıksal Okumalar (Kullanıcılar)	Fiziksel Okumalar	Ön Okumalar
GetUserPosts	16	44	936	2	3	1944
GetUserDetails	0	3	-	3	2	0
UpdateUserDetails	0	7	-	3	2	0

Tablo 11.1 Saklı Yordam Genel Değerlendirme



Şekil 11.4 Saklı Yordam Genel Değerlendirme Grafiği

Saklı yordamlar, veritabanı işlemlerinin performansını artırmak için kritik bir öneme sahiptir. Sunulan analiz, çeşitli gizli algoritmaların veritabanı üzerindeki etkilerini ortaya koyuyor ve performans ölçütlerine dayanarak optimize etme olasılıklarını belirlemeyi amaçlıyor. Sorguların optimize edilmesi, özellikle yüksek G/Ç maliyeti olanların, genel veritabanı performansını olumlu yönde etkileyecektir.

## 12. Aynı Görevi Yerine Getiren Sorguların Daha Etkin Yazılması

Veritabanı yönetiminde, aynı görevi yerine getiren farklı sorgu yazma stilleri performans üzerinde ciddi etkilere sahip olabilir. Bu bölümde, aynı işlevi yerine getiren çeşitli sorgu yazma tekniklerinin farklı performans sonuçlarına yol açabileceğini ve bu sonuçların nasıl optimize edilebileceğini inceleyeceğiz. Bu inceleme, dizinleme, alt sorgular, JOIN işlemleri, görünüm ve depolanan prosedürler gibi teknikleri içerecektir.

### 12.1 Örnekler ve Analizi

**Tarih Aralığına Göre Gönderileri ve Kullanıcı Detaylarını Alma**

```
CREATE PROCEDURE GetPostsByDateRange
    @StartDate DATE,
    @EndDate DATE
AS
BEGIN
    SET STATISTICS TIME ON;
    SET STATISTICS IO ON;

    SELECT p.Id, p.Title, p.CreationDate, u.DisplayName, u.Reputation
    FROM Posts p
    JOIN Users u ON p.OwnerUserId = u.Id
    WHERE p.CreationDate BETWEEN @StartDate AND @EndDate;

    SET STATISTICS TIME OFF;
    SET STATISTICS IO OFF;
END;
```

Bu sorgu, belirli bir zaman diliminde oluşturulmuş paylaşımları ve ilgili kullanıcı verilerini çeker.



## Optimize Versiyonu

```
CREATE PROCEDURE GetPostsByDateRange_Optimized
    @StartDate DATE,
    @EndDate DATE
AS
BEGIN
    SET STATISTICS TIME ON;
    SET STATISTICS IO ON;

    SELECT p.Id, p.Title, p.CreationDate, u.DisplayName, u.Reputation
    FROM Posts p WITH (INDEX(IX_Posts_CreationDate))
    JOIN Users u WITH (INDEX(PK_Users_Id)) ON p.OwnerUserId = u.Id
    WHERE p.CreationDate BETWEEN @StartDate AND @EndDate;

    SET STATISTICS TIME OFF;
    SET STATISTICS IO OFF;
END;
```

Şekil 12.1 Tarih Aralığına Göre Gönderileri ve Kullanıcı Detaylarını Alma Optimizasyonu

Şekil 12.1'de görüle kodda Posts tablosu için IX\_Posts\_CreationDate indeks ipucunu ekleyerek CreationDate'e göre verilerin daha hızlı getirilmesini sağladık.

Users tablosu için PK\_Users\_Id indeks ipucunu ekleyerek OwnerUserId'ye göre yapılan join işlemini optimize ettik.

## Gönderi ve Kullanıcı Detayları ile Yorumları Alma

```
CREATE PROCEDURE GetCommentsWithPostUserDetails
```

```
    @PostId INT
```

```
AS
```

```
BEGIN
```

```
    SET STATISTICS TIME ON;
```

```
    SET STATISTICS IO ON;
```

```
SELECT c.Id, c.Text, c.CreationDate, p.Title, u.DisplayName
```

```
FROM Comments c
```

```
JOIN Posts p ON c.PostId = p.Id
```

```
JOIN Users u ON c.UserId = u.Id
```

```
WHERE c.PostId = @PostId;
```

```
SET STATISTICS TIME OFF;
```

```
SET STATISTICS IO OFF;
```

```
END;
```

Bu sorgu, belirli bir gönderi için yorumları ve ilgili gönderi ile kullanıcı bilgilerini getirir.

#### Optimize Versiyonu

```
CREATE PROCEDURE GetCommentsWithPostUserDetails_Optimized
    @PostId INT
AS
BEGIN
    SET STATISTICS TIME ON;
    SET STATISTICS IO ON;

    SELECT c.Id, c.Text, c.CreationDate, p.Title, u.DisplayName
    FROM Comments c WITH (INDEX(IX_Comments_PostId))
    JOIN Posts p WITH (INDEX(PK_Posts_Id)) ON c.PostId = p.Id
    JOIN Users u WITH (INDEX(PK_Users_Id)) ON c.UserId = u.Id
    WHERE c.PostId = @PostId;

    SET STATISTICS TIME OFF;
    SET STATISTICS IO OFF;
END;
```

Şekil 12.2 Gönderi ve Kullanıcı Detayları ile Yorumları Alma

Şekil 12.2. görülen kodda Comments tablosu için IX\_Comments\_PostId indeks ipucunu ekleyerek PostId'ye göre verilerin daha hızlı getirilmesini sağlandı Posts tablosu için PK\_Posts\_Id ve Users tablosu için PK\_Users\_Id indeks ipucunu ekleyerek join işlemlerini optimize edildi.

#### Top Kullanıcıları İtibarıyla Alma

```
CREATE PROCEDURE GetTopUsersByReputation
    @TopN INT
AS
BEGIN
    SET STATISTICS TIME ON;
    SET STATISTICS IO ON;

    SELECT TOP (@TopN) Id, DisplayName, Reputation
    FROM Users
    ORDER BY Reputation DESC;
```

```
SET STATISTICS TIME OFF;
```

```
SET STATISTICS IO OFF;
```

```
END;
```

Bu sorgu, en yüksek itibara sahip kullanıcıları getirir.

### Optimize Hali

```
CREATE PROCEDURE GetTopUsersByReputation_Optimized
    @TopN INT
AS
BEGIN
    SET STATISTICS TIME ON;
    SET STATISTICS IO ON;

    SELECT TOP (@TopN) Id, DisplayName, Reputation
    FROM Users WITH (INDEX (IX_Users_Reputation))
    ORDER BY Reputation DESC;

    SET STATISTICS TIME OFF;
    SET STATISTICS IO OFF;
END;
```

Şekil 12.3 Top Kullanıcıları İtibarıyla Alma Optimize

Users tablosu için IX\_Users\_Reputation indeks ipucunu ekleyerek Reputation'a göre sıralamanın daha hızlı yapılmasını sağlandı.

### Kullanıcıya Göre Skorla Gönderileri Almak

```
CREATE PROCEDURE GetPostsByUserWithScore
```

```
    @UserId INT
```

```
AS
```

```
BEGIN
```

```
    SET STATISTICS TIME ON;
```

```
    SET STATISTICS IO ON;
```

```
    SELECT p.Id, p.Title, p.Score, u.DisplayName
```

```

FROM Posts p
JOIN Users u ON p.OwnerUserId = u.Id
WHERE p.OwnerUserId = @UserId;
SET STATISTICS TIME OFF;
SET STATISTICS IO OFF;
END;

```

Bu sorgu, belirli bir kullanıcı tarafından yapılan gönderileri ve bu gönderilerin puanlarını getirir.

```

CREATE PROCEDURE GetPostsByUserWithScore_Optimized
    @UserId INT
AS
BEGIN
    SET STATISTICS TIME ON;
    SET STATISTICS IO ON;

    SELECT p.Id, p.Title, p.Score, u.DisplayName
    FROM Posts p WITH (INDEX(IX_Posts_UserId_Score))
    JOIN Users u WITH (INDEX(PK_Users_Id)) ON p.OwnerUserId = u.Id
    WHERE p.OwnerUserId = @UserId;

    SET STATISTICS TIME OFF;
    SET STATISTICS IO OFF;
END;

```

Şekil 12.4 Kullanıcıya Göre Skorla Gönderileri Almak Optimizasyonu

Posts tablosu için IX\_Posts\_UserId\_Score indeks ipucunu ekleyerek UserId ve Score'a göre verilerin daha hızlı getirilmesini sağlandı

Users tablosu için PK\_Users\_Id indeks ipucunu ekleyerek OwnerUserId'ye göre yapılan join işlemini optimize edildi.

### Aylık Gönderi Sayıları Alma

```
CREATE PROCEDURE GetMonthlyPostCounts
AS
BEGIN
    SET STATISTICS TIME ON;

    SET STATISTICS IO ON;

    SELECT YEAR(CreationDate) AS Year, MONTH(CreationDate) AS Month, COUNT(*) AS
    PostCount

    FROM Posts

    GROUP BY YEAR(CreationDate), MONTH(CreationDate)

    ORDER BY Year, Month;

    SET STATISTICS TIME OFF;

    SET STATISTICS IO OFF;

END;
```

Bu sorgu, her ay oluşturulan gönderilerin sayısını hesaplar.

### Optimize Hali

```
CREATE PROCEDURE GetMonthlyPostCounts
AS
BEGIN
    SET STATISTICS TIME ON;
    SET STATISTICS IO ON;

    SELECT YEAR(CreationDate) AS Year, MONTH(CreationDate) AS Month, COUNT(*) AS PostCount
    FROM Posts
    GROUP BY YEAR(CreationDate), MONTH(CreationDate)
    ORDER BY Year, Month;

    SET STATISTICS TIME OFF;
    SET STATISTICS IO OFF;

END;
```

### Şekil 12.1.5 4 Aylık Gönderi Sayıları Alma Optimizasyonu

Şekil 12.1.5te görülen kodda Posts tablosu için IX\_Posts\_CreationDate indeks ipucunu ekleyerek CreationDate'e göre verilerin daha hızlı gruplandırılmasını sağlandı.

### Aylık Gönderi Sayılarını Alma

```
CREATE PROCEDURE GetMonthlyPostCounts
```

```

AS
BEGIN
    SELECT YEAR(CreationDate) AS Year, MONTH(CreationDate) AS Month, COUNT(*) AS
PostCount
    FROM Posts
    GROUP BY YEAR(CreationDate), MONTH(CreationDate)
    ORDER BY Year, Month;
END;

```

Bu sorgu, her ay oluşturulan gönderilerin sayısını hesaplar.

```

CREATE PROCEDURE GetMonthlyPostCounts_Optimized
AS
BEGIN
    SELECT YEAR(CreationDate) AS Year, MONTH(CreationDate) AS Month, COUNT(*) AS PostCount
    FROM Posts WITH (INDEX (IX_Posts_CreationDate))
    GROUP BY YEAR(CreationDate), MONTH(CreationDate)
    ORDER BY Year, Month;
END;

```

#### Şekil 12.5. Aylık Gönderi Sayılarının Alma Optimizasyonu

Posts tablosu için IX\_Posts\_CreationDate indeks ipucunu ekleyerek CreationDate'e göre verilerin daha hızlı gruplandırılmasını sağlandı

##### 12.1.1. Genel Sonuç ve Değerlendirme

Ölçüt / Sorgu	Sorgu 1: 1,096,144 Satır	Sorgu 2: 151,492 Satır	Sorgu 3: 1,096,144 Satır	Sorgu 4: 1,096,144 Satır	Sorgu 5: 180,229 Satır
Geçen Süre (ms) - Optimize Olmayan	2094	115	138	208	204
Geçen Süre (ms) - Optimize	120	30	138	20	40
Sorgu 1: 1,096,144 Satır - İyileştirme (%)	94,3				
Sorgu 2: 151,492 Satır - İyileştirme (%)		73,9			
Sorgu 3: 1,096,144 Satır - İyileştirme (%)			0,0		
Sorgu 4: 1,096,144 Satır - İyileştirme (%)				90,4	
Sorgu 5: 180,229 Satır - İyileştirme (%)					80,4

Tablo 12.1 Aynı Görevi Yerine Getiren Sorguların Daha Etkin Yazılması

Bu analiz kapsamında beş adet saklı prosedür oluşturulmuş ve iyileştirilmiştir. Mukayese edildiğinde optimize edilmiş ve optimize edilmemiş sürümler,

Performans ölçütlerinde, geçen süre (ms) gibi önemli gelişmeler kaydedilmiştir. Optimize edilmemiş sürümlerde, sorguların uygulanması daha uzun sürmekte ve sistem kaynaklarını daha fazla zorlamaktadır.

Örneğin, 1. Sorgu için optimize edilmemiş versiyonda geçen süre 2094 ms iken, optimizasyon yapılmış sürümde bu süre 120 ms'ye inmiştir. Benzer şekilde, 4. Optimize edilmemiş sürümde arama süresi 208 ms iken, optimize edilmiş sürümle bu süre 20 ms'ye inmiştir.

Bu verilere göre, optimize edilmiş sürümlerin sorgu performansını belirgin bir şekilde artırdığı görülmektedir. Performansın arttırılmasının temel sebebi, daha etkili sorgu planlarının benimsenmesi ve gereksiz veri okuma operasyonlarının azaltılmasıdır.

Bu araştırmada, indeksleme kullanmadan gerçekleştirilen optimizasyonlarla sorgu performansının nasıl iyileştirilebileceği açıklanmıştır. Optimize edilmiş sürümlerde, sorgu planları daha efektif bir şekilde düzenlenerek CPU kullanımı ve mantıksal okumalar azaltılmıştır. Bunun yanı sıra, fiziksel ve ön okuma gibi I/O işlemlerinde de belirgin düşüşler görülmüştür.

Kısacası, bu analiz sorgu optimizasyon tekniklerinin veritabanı performansındaki önemli etkisine dikkat çekmektedir. Veritabanı yöneticileri ve geliştiriciler, böylece sistem performansını önemli ölçüde artırabilirler. Bu geliştirmeler, özellikle büyük veri kümeleriyle çalışan uygulamalarda daha önemli bir hal almaktadır.

### 13. Genel Değerlendirme ve Özet

Bu çalışmada, veritabanı yönetim sistemlerinde sorgu optimizasyon yöntemlerinin performans üzerindeki etkileri detaylı bir şekilde incelenmiştir.

Bu çalışmada tezde Stack Overflow küçük veritabanı için kullanılan kodlar bu github adresine yüklenmiştir: <https://github.com/cemo1999/Sql-Sorgu-Optimizasyonlar->

Veritabanı yönetim sistemleri, büyük miktarda verinin hızlı ve verimli bir şekilde işlenmesi gereken senaryolarda kritik bir rol oynar.

Bu bağlamda, sorgu optimizasyonu, sistem performansını artırmak ve kaynak kullanımını en aza indirmek için önemli bir teknik olarak öne çıkmaktadır.

Çalışmada uygulanan sorgu optimizasyon teknikleri, indeks kullanımı, daha verimli sorgu planlarının oluşturulması ve gereksiz veri okuma işlemlerinin minimize edilmesi gibi yöntemleri kapsamaktadır. Elde edilen sonuçlar, optimize edilmiş sorguların, optimize edilmemiş sorgulara kıyasla önemli performans iyileştirmeleri sağladığını açıkça göstermektedir. Optimizasyon tekniklerinin uygulanmasıyla, sorguların gerçekleştirilme süresi büyük ölçüde azalmış ve sistem kaynaklarının daha etkin kullanımı sağlanmıştır. Özellikle büyük veri kümeleri üzerinde yapılan sorgularda bu iyileştirmeler daha belirgin hale gelmiştir.

Performans iyileştirmeleri sadece CPU süresi ve geçen süre açısından değil, aynı zamanda mantıksal ve fiziksel okumalar gibi I/O işlemlerinde de gözlemlenmiştir.

Bu çalışmanın bulguları, veritabanı yönetim sistemlerinde sorgu optimizasyonunun kritik bir öneme sahip olduğunu vurgulamaktadır. Optimizasyon teknikleri, veritabanı performansını artırmakla kalmaz, aynı zamanda kullanıcı deneyimini de iyileştirir. Veritabanı yöneticileri ve geliştiricileri, bu tür teknikleri uygulayarak sistem verimliliğini artırabilir ve işletmelerin operasyonel süreçlerini daha etkili hale getirebilirler.

Optimizasyon tekniklerinin uygulanması, veritabanı yönetim sistemlerinde çeşitli alanlarda önemli etkiler yaratmaktadır. İlk olarak, CPU süresi ve işlem süresi üzerinde belirgin azalmalar sağlanmıştır. Bu da, daha fazla işlem yapılabilmesi ve daha yüksek performans elde edilmesi anlamına gelir. İkincisi, I/O işlemlerinde azalmalar gözlemlenmiştir. Bu, veritabanı sunucusunun disk okuma/yazma yükünün azalmasına ve dolayısıyla sistem kaynaklarının daha verimli kullanılmasına olanak tanır.

Ek olarak, optimize edilmiş sorguların daha az bellek kullanımı gerektirdiği gözlemlenmiştir. Bu da, veritabanı sunucusunun daha büyük veri kümelerini daha etkin bir şekilde işleyebilmesini sağlar. Tüm bu faktörler bir araya geldiğinde, sistemin genel performansında önemli bir artış sağlanmış ve kullanıcı deneyimi iyileştirilmiştir.

Veritabanı yönetim sistemlerinde, performans iyileştirme açısından vazgeçilmez bir öge olan sorgu optimizasyonuna büyük önem verilmektedir. Bu çalışmada sunulan yöntemler ve elde edilen sonuçlar veritabanı performansını artırma konusunda değerli bilgiler sağlamaktadır. Gelecekte yapılacak çalışmalar, bu optimizasyon tekniklerinin nasıl daha geniş veri kümelerine ve farklı veritabanı sistemlerine uygulanabileceğini araştırabilir. Bunun yanı sıra, ileri teknolojilerin içerisinde yapay zeka ve makine öğrenimi gibi olanlarının sorgu



optimizasyonunda nasıl kullanılabileceği de değerlendirilmesi gereken bir araştırma konusu olabilir. Veritabanı performansını optimize etmek, işletmelerin verimliliğini artırmanın yanı sıra kullanıcı memnuniyetini sağlama açısından da oldukça önemlidir. Bu çalışma, veritabanı yönetim sistemlerinde daha sürdürülebilir ve etkili veri yönetimi çözümleri sunmak için önemli bir rehber niteliğine sahip olan yöntemleri sunmaktadır. Sistem performansını büyük ölçüde iyileştirebilir ve daha verimli veri işleme süreçleri sağlayabilir olan doğru uygulanan sorgu optimizasyon yöntemleridir.

Bu, işletmelerin rekabet gücünü artırmanın yanı sıra kullanıcıların daha hızlı ve güvenilir bir şekilde hizmet almasını da sağlayacaktır. Gelecekteki çalışmalarda, önemli araştırma alanları olarak göze çarpan hususlar ise optimizasyon tekniklerinin geniş bir veri tabanında uygulanabilir oluşu ve yeni teknolojilerin entegrasyonudur.

Veritabanı optimizasyonu, günümüzde dijital dönüşüm süreçlerinde önemli bir unsur olarak kabul edilmektedir. Kurumların dijital dönüşüm süreçlerinde, veritabanı yönetim sistemi kullanımının etkinliği, büyük veri analitiğine ve gerçek zamanlı işlem yeteneklerine doğrudan bağlıdır. Bu nedenle, sorgu optimizasyonunu sadece teknik bir gereklilik olarak değil aynı zamanda stratejik bir avantaj olarak da görmek önemlidir.

Önümüzdeki dönemde, veritabanı sorgu optimizasyonunda yapay zeka ve makine öğrenimi tekniklerinin daha da yaygınlaşması beklenilmektedir. Büyük ve karmaşık veri kümelerinin üzerinde özellikle bu teknikler, daha iyi sorgu optimizasyonu sağlayabilir. Bir örnek olarak, makine öğrenimi algoritmaları veritabanı yöneticilerine sorgu performansını artırmak için otomatik önerilerde bulunabilir veya belirli sorgu desenlerini analiz ederek optimizasyon stratejileri geliştirebilir.

Günümüz veri yoğun dünyasında, sorgu optimizasyonu veritabanı yönetim sistemlerinde performans artırma ve kaynakların etkin kullanımına yönelik kritik bir öneme sahiptir. Bu çalışma, veritabanı yönetim sistemlerinin gelecekteki gelişimine katkıda bulunmayı hedefleyerek, bu alandaki iyileştirmelerin potansiyel faydalarını göstermektedir. Veritabanı performansının optimize edilmesi, işletmelerin operasyonel verimliliklerini artırmanın yanı sıra maliyetleri düşürmek ve kullanıcı deneyimini iyileştirmek için gereklidir. Bu bağlamda, çalışmamızdaki yöntemler ve sonuçlar, veritabanı yönetim sistemlerinde daha sürdürülebilir ve etkili bir şekilde veri yönetimi sağlamada önemli rehberlik sunmaktadır.

## **Kaynakça**

IT Companies Network, "Database Performance Tuning Best Practices," Available: <https://itcompanies.net/blog/database-performance-tuning-best-practices#:~:text=The%20best%20techniques%20for%20database%20performance%20tuning%201,8.%20Use%20a%20powerful%20CPU...%20More%20items>

. Erişim tarihi: [01.01.2024]. [1]

ScienceDirect, "Assessment of waste characteristics and their impact on GIS vehicle collection route optimization using ANN waste forecasts," Available: <https://www.sciencedirect.com/science/article/pii/S0164121223002674>.

DOI: [DOI bilgisi], Erişim tarihi: [02.02.2024]. [2]

OptimizDBA, "The Ultimate Guide to Database Optimization: Tips and Best Practices," Available: <https://optimizdba.com/the-ultimate-guide-to-database-optimization-tips-and-best-practices/>.

Erişim tarihi: [03.02.2024]. [3]

Silk, "Unlocking Database Performance: Tips for Optimization," Available: <https://silk.us/blog/unlocking-database-performance-tips-for-optimization/>.

Erişim tarihi: [10.04.2024]. [4]

Açık Bilim, "Akademik Tez: Gelişmiş Atık Yönetimi ve Optimizasyon Teknikleri," Available: <https://acikbilim.yok.gov.tr/handle/20.500.12812/390931>.

Erişim tarihi: [11.04.2024]. [5]

SQL Ekibi, "Sorgu Optimizasyonu: Sorgularımızı Nasıl Daha İyi Hale Getirebiliriz," Available: <https://www.sql ekibi.com/sql-server/sorgu-optimizasyonu-query-optimization-sorgularimizi-nasil-daha-iyi-hale-getirebiliriz.html/>.

Erişim tarihi: [01.05.2024]. [6]

Gökhan A., "SQL Index Yapısı ve Kullanımı," Available: <https://gokhana.medium.com/sql-index-yap%C4%B1s%C4%B1-ve-kullan%C4%B1m%C4%B1-115079c018c0>.

Erişim tarihi: [15.05.2024]. [10]

Logo Yazılım, "Veritabanı Performans İyileştirme Rehberi," Available: <https://docs.logo.com.tr/pages/viewpage.action?pageId=50687677>.

Erişim tarihi: [15.05.2024]. [11]

OptimizDBA, "The Ultimate Guide to Database Optimization: Tips and Best Practices," Available: <https://optimizdba.com/the-ultimate-guide-to-database-optimization-tips-and-best-practices/>. Erişim tarihi: [20.05.2024]. [12]

OptimizDBA, "The Ultimate Guide to Database Optimization: Tips and Best Practices," Available: <https://optimizdba.com/the-ultimate-guide-to-database-optimization-tips-and-best-practices/>. Accessed: [20.05.2024] [13]

## ÖZGEÇMİŞ

Adı Soyadı: Yusuf Cem ŞEN

Doğum Yeri ve Tarihi: Malatya 27.11.1998

Ana Dili: Türkçe

Yabancı Dili: İngilizce (C1), Almanca (A2)

E-Posta: yfcmsen19@gmail.com

### Öğrenim Durumu

Derece	Bölüm/Program	Üniversite/Lise	Mezuniyet Yılı
Lise		Malatya Anadolu Lisesi	2017
Lisans	Yazılım Mühendisliği	İstanbul Aydın Üniversitesi	-
Lisans	Bilgisayar Mühendisliği	Marmara Üniversitesi	Devam Ediyor
Yüksek Lisans	-	-	-

### İş Deneyimi

Yıl	Firma/Kurum	Görevi
2022-2022	Rea Ventures	Bilgisayar Mühendisliği Stajyeri
Mart 2023 -Mayıs 2023	Parametre A.Ş.	Bilgisayar Mühendisliği
Haziran 2023-	Testinium Teknoloji Yazılım A.Ş.	Bilgisayar Mühendisliği Stajyeri