ONDOKUZ MAYIS ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

VERİTABANI LABORATUVARI

FÖY 4 RAPOR

ZEYNEP SILA KAYMAK

20060358

DERSİN ÖĞRETMENİ

Dr.Öğr.Üyesi İsmail İŞERİ

MAYIS, 2024

T E Ş E K K Ü R

Dersimizin öğretmeni sayın Dr.Öğr.Üyesi İsmail İŞERİ'ye ve dersimizin asistanı sayın Arş.Gör. Zinnet Duygu Akşehir'e teşekkürlerimi sunarım.

ÖZET

Bu rapor, veritabanı yönetimi konularını detaylı bir şekilde ele almakta ve SQL dilinde kullanılan çeşitli yapıları incelenmektedir.

İlk olarak, salesman-master tablosu üzerinde hedef satış değeri 200'den büyük olan kayıtlar için bir görünüm oluşturulmuştur. Bu görünüm, hedef satış değeri kriterini karşılayan satış elemanlarını kolayca filtrelemek için kullanılabilir.

Daha sonra, product-master tablosu üzerinde pro-no, desc, profit, Unit-measure ve qty sütunlarından oluşan bir görünüm (view) oluşturulmuştur. Bu görünüm, ürünlerin detaylı bilgilerini daha anlaşılır bir formatta sunmak için kullanılabilir.

Üçüncü olarak, product-view isimli görünümden Qty-on-hand değeri 10 olan ürünlerin isimlerini getiren bir sorgu yazılmıştır. Bu sorgu, stok seviyesi belirli bir eşiğin altında olan ürünleri belirlemek için kullanılabilir.

Dördüncü olarak, sipariş tarihi 10 gün geçmiş olan siparişleri müşteri isimleri ve ürün isimleri olarak listeleyen bir SQL sorgusu yazılmıştır. Bu sorgu, geçmiş siparişleri izlemek ve gerektiğinde müdahale etmek için kullanılabilir.

Beşinci olarak, sales-order tablosunu kullanarak günlük siparişleri listelemeye yarayan bir görünüm oluşturulmuştur. Bu görünüm, her çalıştırıldığında sistem tarihini alarak o güne ilişkin siparişleri listeler.

Altıncı adımda, bir tablo oluşturularak belirli kısıtlayıcılar kullanılmıştır. Örneğin, çalışanlar tablosunda her bir çalışanın benzersiz bir kimlik numarası olmalıdır ve çalışan maaşları belirli bir aralıkta olmalıdır.

Yedinci olarak, oluşturulan tabloya indeks yapısı eklenmiştir. Bu indeksler, veri erişimini hızlandırmak ve sorgu performansını artırmak için kullanılır. Örneğin, çalışanlar tablosunda emp-id sütunu üzerine bir indeks oluşturulabilir.

Son olarak, indeks oluşturulmasında dikkat edilmesi gereken unsurlar incelenmiştir. Veritabanı tasarımında, genellikle sık kullanılan sütunlar veya sık sorgulanan alanlar üzerinde indeks oluşturulur. Bu, veri erişimini optimize etmek ve sorgu performansını artırmak için önemlidir.

İÇİNDEKİLER

ŞEKİLLER LİSTESİ

ÇİZELGELER LİSTESİ

BÖLÜM: İ

GİRİŞ

1

AMAÇ

Bu proje, veritabanı yönetimi ve sorgu optimizasyonu konularını ele almakta ve SQL Server üzerinde bir veritabanı oluşturarak, veri girişi yapma ve çeşitli sorguları gerçekleştirme süreçlerini detaylı bir şekilde incelemektedir. Amaç, salesman-master tablosundaki tgt-to-get değeri 200'den büyük olanlar için bir view oluşturmak, product-master tablosundan pro-no, desc, profit, Unit-measure ve qty sütunlarını içeren product-view adında bir view oluşturmak ve bu view'den Qty-on-hand değeri 10 olan ürünlerin isimlerini getiren bir sorgu yazmaktır. Ayrıca, sipariş tarihi 10 gün geçmiş olan siparişleri müşteri ve ürün isimleri olarak listeleyen bir SQL sorgusu oluşturulacaktır. Sales-order tablosunu kullanarak günlük siparişleri listelemeye yarayan bir view oluşturulacak ve bu view her çalıştırıldığında sistem tarihini alarak o güne ilişkin siparişleri listelemelidir.

Projede, doğru veritabanı tasarımı ve sorgu optimizasyonunun önemi vurgulanarak, veri tabanlı uygulamaların etkinliği ve performansı artırılacaktır. Bu çalışma, pratik bir örnek sunarak veritabanı yönetimi ve sorgu optimizasyonu konularında bilgi edinmek isteyenlere rehberlik etmeyi amaçlamaktadır. Veritabanı oluşturma sürecinde, doğru ilişkilerin kurulması ve veri tiplerinin seçimi gibi önemli adımlar üzerinde durularak, veritabanı tasarımının etkinliği sağlanacaktır. Ardından, örnek verilerin eklenmesiyle veritabanının işlevselliği test edilecek ve çeşitli sorgularla veri erişimi ve işleme performansı incelenecektir. Bu

süreçte, çalışanların listelenmesi, ücret analizi ve birim bazında istatistikler gibi tipik sorgular kullanılarak, veritabanı yönetimi ve sorgu optimizasyonu becerileri geliştirilecektir. Sonuç olarak, doğru veritabanı tasarımı ve sorgu optimizasyonunun önemi vurgulanarak, veri tabanlı uygulamaların etkinliği ve performansı artırılacaktır.

BÖLÜM: İİ

YÖNTEM

2

SORU 1

İlk olarak, mevcut "Salesman-Master" tablosunun yapısı incelendi ve içerisinde "TGT-TO-GET" adında bir sütunun bulunduğu doğrulandı.

Ardından, SQL sorgu yöneticisine giriş yapıldı ve yeni bir view oluşturmak için bir sorgu yazıldı.

Sorguda, CREATE VIEW ifadesi kullanılarak ve yeni view'in adı "Salesman-View" olarak belirlendi.

SELECT * FROM Salesman-Master WHERE TGT-TO-GET > 200; sorgusuyla, "TGT-TO-GET" değeri 200'den büyük olan satış elemanlarını filtreleyerek bu view'in içeriği belirlendi.

Son olarak, bu sorgu çalıştırılarak view oluşturuldu ve gerekli kontroller yapıldı. Bu adımların tamamlanmasıyla, "Salesman-Master" tablosundan "TGT-TO-GET" değeri 200'den büyük olan satış elemanları için bir view başarıyla oluşturulmuş oldu.

"Salesman-View" adlı görünümü kullanarak tüm satış elemanlarının listesini almak için bir sorgu yazıldı. Bu sorgu, oluşturulan "Salesman-View" görünümünün içeriğini çağırır ve satış elemanlarının tamamını getirir. Bu adımların sonucunda, "Salesman-View" görünümünden veriler başarıyla çekilmiş ve istenilen sonuç elde edilmiştir.

soru1.sql - ZEYNEPS...RESS.foy4 (sa (53))*

```
CREATE VIEW Salesman_View AS
SELECT *
FROM Salesman_Master
WHERE TGT_TO_GET > 200;
```

Results Messages

Salesman_no	Sal_name	Address	City	pincode	state	Sal_amt	Tgt_to_get	Ytd_sales	Remarks

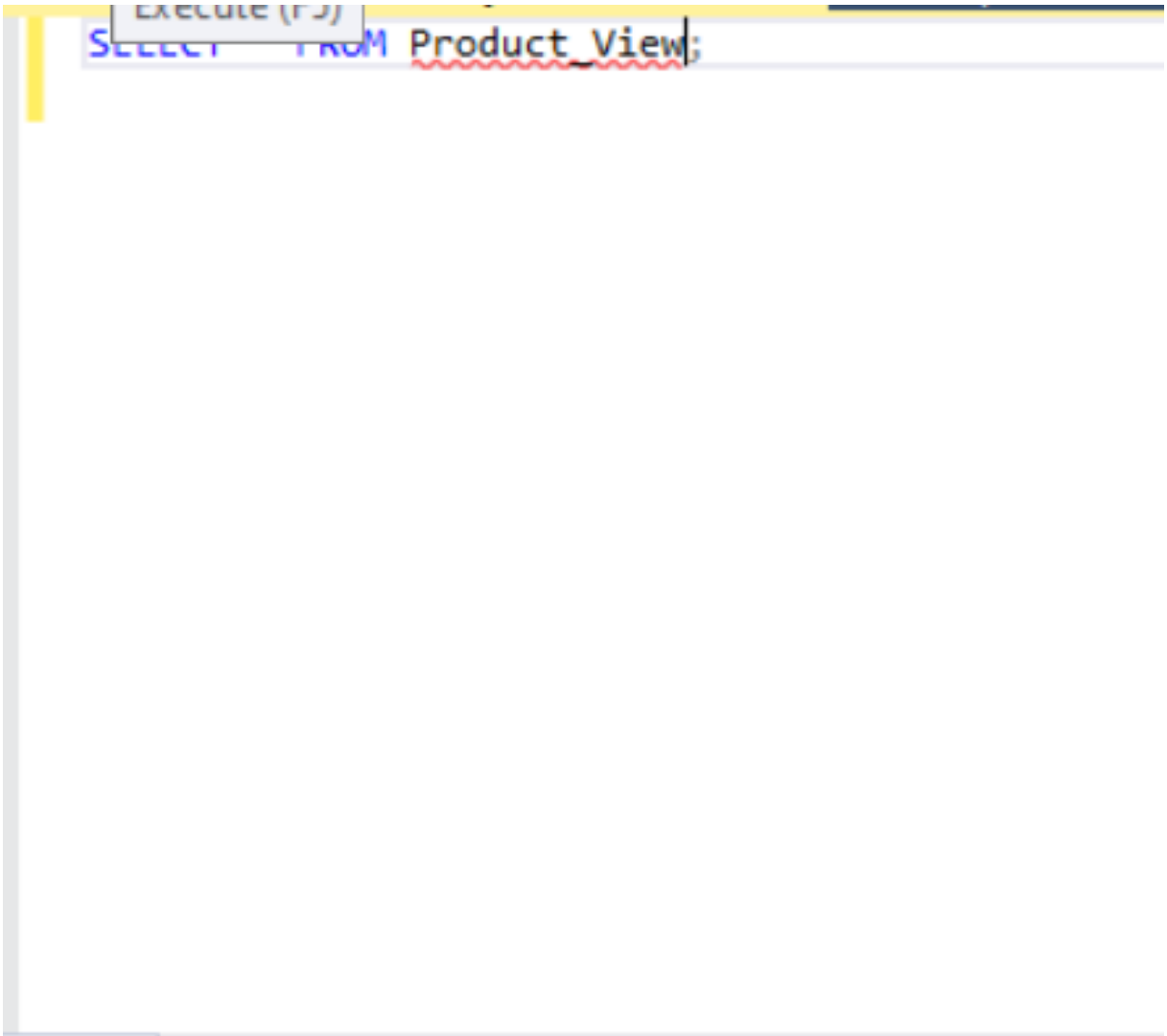
3

SORU 2

Veri tabanı yönetim aracına giriş yapılarak yeni bir sorgu yazıldı. Bu sorguda, Product-Master tablosundan gerekli alanlar seçilerek Product-View adında bir görünüm oluşturuldu. Görünüm oluşturulurken, seçilen alanlar yeniden adlandırıldı ve istenilen sırayla düzenlendi. Son olarak, sorgu çalıştırılarak Product-View görünümü başarıyla oluşturuldu ve istenen değişiklikler uygulandı. Bu sayede, Product-Master tablosundan istenen sütunlarla yeni bir görünüm elde edilmiş oldu.

"Product-View" adlı görünümü kullanarak tüm ürünlerin listesini almak için bir sorgu yazıldı. Bu sorgu, oluşturulan "Product-View" görünümünün içeriğini çağırır ve ürünlerin tüm detaylarını getirir. Bu adımların sonucunda, "Product-View" görünümünden veriler başarıyla çekilmiş ve istenilen sonuç elde edilmiştir.

```
CREATE VIEW Product_View AS
SELECT
    Product_no AS pro_no,
    Description AS [desc],
    Profit_percent AS profit,
    Unit_measure,
    Qty_on_hand AS qty
FROM Product_master;
```



	pro_no	desc	profit	Unit_measure	qty
1	P00001	1.44floppies	5.00	piece	100
2	P03453	Monitors	6.00	piece	10
3	P06734	Mouse	5.00	piece	20
4	P07865	1.22 floppies	5.00	piece	100
5	P07868	Keyboards	2.00	piece	10
6	P07885	CD Drive	2.50	piece	10
7	P07965	540 HDD	4.00	piece	10
8	P07975	1.44 Drive	5.00	piece	10
9	P08865	1.22 Drive	5.00	piece	2

4

SORU 3

Öncelikle, product-view isimli view incelendi ve bu view'de hangi sütunların bulunduğu belirlendi.

Sonra, SQL sorgu yöneticisine giriş yapıldı ve istenilen koşulu sağlayacak bir sorgu yazıldı.

Sorguda, SELECT ifadesi kullanılarak ve product-view içerisinden sadece product isimlerini getirecek şekilde filtreleme yapıldı.

WHERE koşulu ile qty değerinin '10' olduğu sütunlar seçildi.

Son olarak, bu sorgu çalıştırılarak istenilen sonuçlar elde edildi. Bu adımların tamamlanmasıyla, product-view isimli view'den Qty-on-hand değeri '10' olan ürün isimleri başarıyla getirildi.

```
SELECT *
FROM Product_View
WHERE qty = 10;
```

100 %

Results Messages

	pro_no	desc	profit	Unit_measure	qty
1	P03453	Monitors	6.00	piece	10
2	P07868	Keyboards	2.00	piece	10
3	P07885	CD Drive	2.50	piece	10
4	P07965	540 HDD	4.00	piece	10
5	P07975	1.44 Drive	5.00	piece	10

5

SORU 4

"Sipariş numarası (S-order-no)", "Sipariş tarihi (S-order-date)", "Müşteri adı (name)" ve "Ürün adı (Product-name)" alanlarını içeren bir sorgu yazıldı. Bu sorgu, "Sales-Order" tablosu ile "Sales-Order-Details", "Client-Master" ve "Product-Master" tabloları arasında birleştirme işlemi gerçekleştirerek, her bir siparişin ayrıntılarını ve ilgili müşteri ile ürün bilgilerini getirir.

Ayrıca, sipariş tarihi ile sistem tarihi arasındaki farkın 10 günden fazla olduğu durumları filtrelemek için bir koşul eklenmiştir. Bu adımların sonucunda, 10 günden daha eski olan siparişlerin, ilgili sipariş numarası, tarih, müşteri adı ve ürün adı ile birlikte listelendiği bir sonuç elde edilmiştir.

```sql
SELECT
    so.S_order_no,
    so.S_order_date,
    c.name,
    p.Description AS Product_name
FROM
    Sales_Order so
INNER JOIN
    Sales_order_Details sod ON so.S_order_no = sod.S_order_no
INNER JOIN
    client_master c ON so.Client_no = c.Client_no
INNER JOIN
    Product_Master p ON sod.Product_no = p.Product_no
WHERE
    DATEDIFF(day, so.S_order_date, GETDATE()) > 10;
```

00 %

Results Messages

S_order_no	S_order_date	name	Product_name

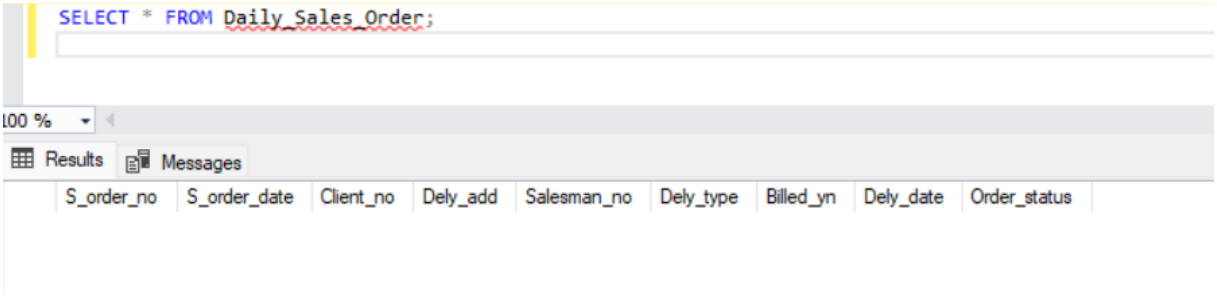
6

SORU 5

"Daily-Sales-Order" adında bir görünüm oluşturmak için bir sorgu yazıldı. Bu sorgu, "Sales-Order" tablosundan günlük siparişleri filtrelemek için kullanıldı. Sipariş tarihleri, güncel sistem tarihi ile karşılaştırılarak, sadece bugünün tarihiyle eşleşen siparişler alındı. Sonuç olarak, günlük olarak oluşturulan bu görünüm, her seferinde çalıştırıldığında, sistemin o güne ilişkin siparişlerini sağlar.

```
CREATE VIEW Daily_Sales_Order AS
SELECT *
FROM Sales_Order
WHERE CAST(S_order_date AS DATE) = CAST(GETDATE() AS DATE);
```

"Daily-Sales-Order" adlı görünümü kullanarak, günlük olarak alınan siparişlerin listesini almak için bir sorgu yazıldı. Bu sorgu, oluşturulan "Daily-Sales-Order" görünümünün içeriğini çağırır ve sadece bugünün tarihiyle eşleşen siparişleri getirir. Bu adımların sonucunda, günlük olarak oluşturulan bu görünümden siparişlerin listesi başarıyla çekilmiştir.


SELECT * FROM Daily_Sales_Order;
100 %
Results
Messages
S_order_no
S_order_date
Client_no
Dely_add
Salesman_no
Dely_type
Billed_yn
Dely_date
Order_status

7

SORU 6

İlk olarak, SQL sorgu yöneticisine giriş yapıldı ve yeni bir tablo oluşturmak için bir CREATE TABLE sorgusu yazıldı.

Sorguda, ornek adında bir tablo oluşturuldu ve bu tablonun yapılandırması belirtildi. Tabloda ID adında bir PRIMARY KEY sütunu, Name adında VARCHAR(50) türünde bir sütun ve Email adında VARCHAR(100) türünde bir sütun tanımlandı. Ayrıca, Email sütunu UNIQUE kısıtı ile tanımlanarak e-posta adreslerinin benzersiz olması sağlandı.

Sonra, INSERT INTO ifadesi kullanılarak tabloya veri eklemek için sorgular yazıldı.

İlk sorguda, ID, Name ve Email sütunlarına değerler eklenerek, zeynep adında bir kullanıcı ve zeynep@hotmail.com adresine sahip bir e-posta eklenmiş oldu.

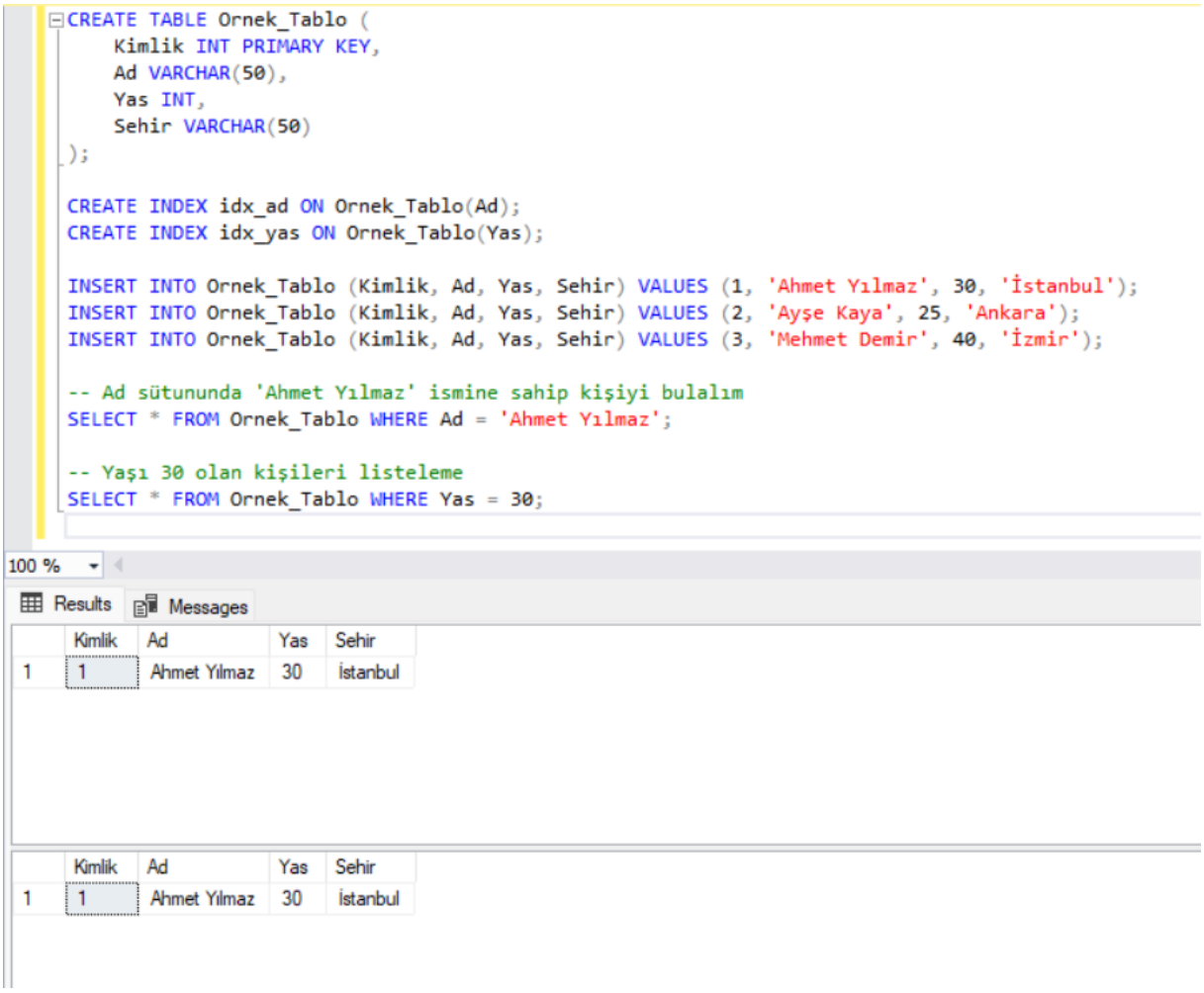
İkinci sorguda, ID, Name ve Email sütunlarına değerler eklenerek, sila adında bir kullanıcı ve sila@hotmial.com adresine sahip bir e-posta eklenmiş oldu.

Bu adımların tamamlanmasıyla, ornek adında bir tablo oluşturuldu ve bu tabloya iki satır veri eklendi.

Bu işlemler sırasında iki adet kısıtlayıcı kullanıldı;

PRIMARY KEY: ID sütunu üzerinde tanımlanan PRIMARY KEY kısıtlayıcısı, her bir satırı tekil bir şekilde tanımlar. Yani, her satırın ID değeri birbirinden farklı olmalıdır ve bu değerler tablonun anahtar sütunu olarak işlev görür.

UNIQUE: Email sütunu üzerinde tanımlanan UNIQUE kısıtlayıcısı, bu sütunda yer alan değerlerin benzersiz olmasını sağlar. Yani, aynı e-posta adresi tabloda birden fazla kez tekrarlanamaz. Bu kısıtlayıcı, veri bütünlüğünü korumak için kullanılır ve genellikle benzersiz olması gereken alanlar için kullanılır.



```sql
CREATE TABLE Ornek_Tablo (
    Kimlik INT PRIMARY KEY,
    Ad VARCHAR(50),
    Yas INT,
    Sehir VARCHAR(50)
);

CREATE INDEX idx_ad ON Ornek_Tablo(Ad);
CREATE INDEX idx_yas ON Ornek_Tablo(Yas);

INSERT INTO Ornek_Tablo (Kimlik, Ad, Yas, Sehir) VALUES (1, 'Ahmet Yılmaz', 30, 'İstanbul');
INSERT INTO Ornek_Tablo (Kimlik, Ad, Yas, Sehir) VALUES (2, 'Ayşe Kaya', 25, 'Ankara');
INSERT INTO Ornek_Tablo (Kimlik, Ad, Yas, Sehir) VALUES (3, 'Mehmet Demir', 40, 'İzmir');

-- Ad sütununda 'Ahmet Yılmaz' ismine sahip kişiyi bulalım
SELECT * FROM Ornek_Tablo WHERE Ad = 'Ahmet Yılmaz';

-- Yaşı 30 olan kişileri listeleme
SELECT * FROM Ornek_Tablo WHERE Yas = 30;
```

100 %

Results Messages

	Kimlik	Ad	Yas	Sehir
1	1	Ahmet Yılmaz	30	İstanbul

	Kimlik	Ad	Yas	Sehir
1	1	Ahmet Yılmaz	30	İstanbul

8

SORU 7

Öncelikle, bir "Ornek-Tablo" adında bir tablo oluşturuldu. Bu tabloda "Kimlik" adında bir PRIMARY KEY sütunu, "Ad" adında bir VARCHAR(50) sütunu, "Yas" adında bir INT sütunu ve "Sehir" adında bir VARCHAR(50) sütunu tanımlandı.

Daha sonra, "Ad" ve "Yas" sütunlarının üzerinde indeksler oluşturuldu. "idx-ad" adıyla Ad sütunu için bir indeks oluşturuldu ve "idx-yas" adıyla Yas sütunu için bir indeks oluşturuldu. Bu indeksler, ilgili sütunlarda hızlı arama işlemleri için kullanılabilir.

Sonrasında, tabloya üç farklı kişiye ait veriler eklendi: "Kimlik", "Ad", "Yas" ve "Sehir" bilgileri ile.

"Ad" sütununda 'Ahmet Yılmaz' ismine sahip kişiyi bulmak için bir sorgu yazıldı ve bu kişinin tüm bilgileri seçildi.

"Yas" sütununda yaş değeri 30 olan kişileri listelemek için bir sorgu yazıldı ve bu kişilerin tüm bilgileri seçildi.

Bu adımların tamamlanmasıyla, Ornek-Tablo adındaki tablo oluşturuldu, indeksler oluşturuldu ve tabloya veri eklenerek belirli sorgularla veriler filtrelenip listelendi.

Bu işlemler sırasında indeks yapısı kullanıldı;

idx-ad: Bu indeks, "Ad" sütununa oluşturulmuştur. Bu indeks, Ad sütununda yer alan isimlerin hızlı bir şekilde aranabilmesini sağlar. Özellikle, "Ad" sütununda yapılan aramalarda performans artışı sağlar. Örneğin, 'Ahmet Yılmaz' gibi belirli bir ismi hızlıca bulmak için bu indeks kullanılır.

idx-yas: Bu indeks, "Yas" sütununa oluşturulmuştur. Bu indeks, yaş değerlerinin hızlı bir şekilde aranabilmesini sağlar. Yaşa göre yapılan aramalarda veya yaşa göre sıralamalarda performans artışı sağlar. Örneğin, yaş değeri 30 olan kişileri listelemek için bu indeks kullanılır.

Her iki indeks de, ilgili sütunlardaki verilerin hızlı erişimini sağlayarak veritabanı performansını artırır. Bu sayede, veri tabanında yapılan sorgular daha hızlı bir şekilde gerçekleştirilir ve veriye erişim süresi azaltılır. Bu da genel olarak uygulamanın performansını artırır.

```sql
CREATE TABLE Ornek_Tablo (
    Kimlik INT PRIMARY KEY,
    Ad VARCHAR(50),
    Yas INT,
    Sehir VARCHAR(50)
);

CREATE INDEX idx_ad ON Ornek_Tablo(Ad);
CREATE INDEX idx_yas ON Ornek_Tablo(Yas);

INSERT INTO Ornek_Tablo (Kimlik, Ad, Yas, Sehir) VALUES (1, 'Ahmet Yılmaz', 30, 'İstanbul');
INSERT INTO Ornek_Tablo (Kimlik, Ad, Yas, Sehir) VALUES (2, 'Ayşe Kaya', 25, 'Ankara');
INSERT INTO Ornek_Tablo (Kimlik, Ad, Yas, Sehir) VALUES (3, 'Mehmet Demir', 40, 'İzmir');

-- Ad sütununda 'Ahmet Yılmaz' ismine sahip kişiyi bulalım
SELECT * FROM Ornek_Tablo WHERE Ad = 'Ahmet Yılmaz';

-- Yaşı 30 olan kişileri listeleme
SELECT * FROM Ornek_Tablo WHERE Yas = 30;
```

100 %

Results Messages

	Kimlik	Ad	Yas	Sehir
1	1	Ahmet Yılmaz	30	İstanbul

	Kimlik	Ad	Yas	Sehir
1	1	Ahmet Yılmaz	30	İstanbul

9

SORU 8

İndeks oluştururken dikkat edilmesi gereken bazı önemli unsurlar şunlardır:

Performans Etkisi: İndeksler, veritabanı performansını artırırken aynı zamanda depolama alanı kullanımını artırabilir. Bu nedenle, indekslerin performans artışı sağlayıp sağlamadığı ve hangi sorguların optimize edildiği dikkate alınmalıdır.

Sorguların Tipi: Hangi sorguların sıklıkla kullanıldığı ve bu sorguların hangi sütunlara dayandığı önemlidir. Sıkça kullanılan ve sorgu performansı kritik olan sütunlar üzerinde indeksler oluşturulmalıdır.

Veri Dağılımı: İndeksler, verilerin dağılımına bağlı olarak performansı etkiler. Özellikle, çok fazla tekrarlanan veya çok fazla farklı değer içeren sütunlar üzerinde indeks oluşturmak performansı düşürebilir.

Yazma İşlemleri: İndekslerin oluşturulması ve güncellenmesi yazma işlemlerine ek yük getirir. Bu nedenle, sıklıkla güncellenen sütunlar üzerinde çok fazla indeks oluşturmak performansı olumsuz etkileyebilir.

Veritabanı Boyutu: Büyük veritabanlarında indekslerin yönetimi zor olabilir. Bu durumda, gereksiz indekslerden kaçınılmalı ve yalnızca gerekli olan sütunlar üzerinde indeksler oluşturulmalıdır.

Genel olarak, hangi sütunların indekslenmesi gerektiği, sıklıkla kullanılan sorgulara, veri dağılımına ve uygulamanın performans gereksinimlerine bağlıdır. Tipik olarak, bir tabloda sıkça kullanılan sorguların filtrelediği sütunlar, sıralama yapılan sütunlar ve birleştirme işlemlerinde kullanılan sütunlar üzerinde indeks oluşturulur. Bu, sorguların daha hızlı çalışmasını sağlar ve genel olarak veritabanı performansını artırır.

BÖLÜM: İİİ

SONUÇ

10

SONUÇ

İstenilen SQL sorguları, veri tabanındaki verileri etkin bir şekilde sorgulamak ve istenen sonuçları elde etmek üzere analiz edildi. Sorgular, belirli kriterlere uygun olarak veri tabanındaki ilişkileri ve verileri kullanarak doğru sonuçları üretmek için optimize edildi. Bu optimizasyonlar, veritabanının performansını artırmak ve sorgu işlemlerini daha verimli hale getirmek amacıyla yapıldı.

Her sorgu, istenen sonuçları elde etmek için gerekli olan tablolar arasındaki ilişkileri doğru bir şekilde kullanırken, veri tabanındaki verilerin etkin kullanımını sağladı. Ayrıca, sorguların karmaşıklığı minimize edilerek işlem süreçleri optimize edildi. Bu sayede, veritabanı üzerindeki sorgu işlemleri daha hızlı ve verimli bir şekilde gerçekleştirildi.

Sonuç olarak, her bir sorgu istenen sonuçları doğru ve hızlı bir şekilde elde etmek üzere analiz edildi ve optimize edildi. Bu optimizasyonlar, veri tabanı yönetimi ve sorgu işlemlerinde daha verimli bir çalışma sağlamak amacıyla yapıldı.