**ANKARA ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ**

****

**AĞ TABANLI PARALEL DAĞITIM SİSTEMLERİ**

**BLM4522**

**SUDE KORKMAZ 20291275**

**Github : https://github.com/CSudeKorkmaz**

**Veritabanı Performans Optimizasyonu – Proje Raporu**

### Giriş ve Proje Amacı

Bu projede temel amacım, yüksek hacimli bir veritabanı üzerinde derinlemesine performans analizleri gerçekleştirerek, sistemin etkinliğini artıracak optimizasyon tekniklerini uygulamaktı. Hedefim yalnızca sorguların daha kısa sürede yanıt vermesini sağlamak değil, aynı zamanda donanım kaynaklarının daha dengeli ve verimli kullanılmasına da katkı sunmaktı. Bu bağlamda Microsoft SQL Server platformu üzerinde yer alan **AdventureWorks2016** örnek veritabanını tercih ederek çalışmalarıma başladım.

### Kullandığım Araçlar ve Teknolojiler

Çalışma süresince profesyonel veritabanı yönetim araçlarından yararlandım:

* Microsoft SQL Server Management Studio (SSMS)
* SQL Profiler
* Dynamic Management Views (DMV)
* T-SQL
* Database Tuning Advisor

Bu araçlar sayesinde veritabanının iç dinamiklerini hem gerçek zamanlı hem de geçmiş veriler ışığında analiz etme imkânı elde ettim.

### Gerçek Zamanlı İzleme Süreci

İlk adım olarak, SQL Profiler ile sistemde gerçekleşen sorguları detaylı biçimde izledim. Bu süreçte, özellikle yüksek CPU tüketimine ve uzun çalışma sürelerine sahip sorgulara odaklandım. İzleme sırasında şu verilere ulaştım:

* En sık çalışan sorgular
* En uzun sürede tamamlanan işlemler
* IO ve CPU kullanımına ilişkin istatistikler

### Derinlemesine Performans Analizi

Bu bulgular ışığında, Dynamic Management Views (DMV) aracılığıyla performans göstergelerini analiz ettim.

* sys.dm\_exec\_query\_stats ile sorgu performansını
* sys.dm\_db\_index\_usage\_stats ile indeks kullanım oranlarını
* sys.dm\_exec\_requests ile aktif işlemleri
* sys.dm\_io\_virtual\_file\_stats ile disk IO yüklerini analiz ettim.

## Veritabanı Yük Dengeleme ve Dağıtık Veritabanı Sistemleri

Bu sistemlerin temel amacı; veritabanı performansını artırmak, sistemin kesintisiz hizmet sunmasını sağlamak ve olası sunucu arızalarında veri kaybını önlemektir.

### 1. Veritabanı Replikasyonu (Replication)

Veritabanı replikasyonu, verilerin birden fazla veritabanı sunucusuna çoğaltılarak eş zamanlı olarak saklanması işlemidir.  
SQL Server'da kullanılan **Replication** teknolojisi sayesinde, veriler birincil sunucudan (publisher) diğer sunuculara (subscriber) aktarılır. Bu yapı, okuma işlemlerindeki yükü azaltır ve veri bütünlüğünü koruyarak sistemin daha güvenli hale gelmesini sağlar.

### 2. Yük Dengeleme (Load Balancing)

Yük dengeleme, istemcilerden gelen veri taleplerinin birden fazla veritabanı sunucusu arasında dengeli şekilde dağıtılması yöntemidir.  
SQL Server'ın **Always On Availability Groups** özelliği sayesinde, aynı verilerin yer aldığı birden fazla veritabanı örneği eş zamanlı olarak çalıştırılır ve istemci talepleri otomatik olarak en uygun sunucuya yönlendirilir. Bu sayede hem performans artışı sağlanır hem de hizmet kesintileri önlenir.

### 3. Failover Senaryoları (Geçiş Durumları)

Failover mekanizması, sistemdeki birincil sunucunun erişilemez hale gelmesi durumunda devreye giren otomatik geçiş sistemidir.  
SQL Server ortamlarında **Automatic Failover** (otomatik geçiş) ya da **Manual Failover** (elle geçiş) seçenekleri kullanılarak yüksek erişilebilirlik sağlanır. Bu yapılar kritik iş sürekliliği için oldukça önemlidir.

### İndeks Yapısının Gözden Geçirilmesi

Verilerin daha hızlı işlenmesini sağlamak için indeks yönetimine yoğunlaştım.

* Kullanılmayan indeksleri sistemden kaldırdım.
* Sorgularda sık kullanılan sütunlara uygun yeni indeksler oluşturdum.
* Fragmentation (parçalanma) oranlarını ölçerek reorganizasyon ve yeniden oluşturma işlemleri gerçekleştirdim.

### Sorguların Yeniden Yazımı ve Optimizasyonu

Performansı iyileştiren en kritik adımlardan biri de sorguların yeniden yazılmasıydı.  
Örneğin:  
**Orijinal Sorgu:**

sql

KopyalaDüzenle

SELECT \* FROM Sales.SalesOrderHeader

WHERE YEAR(OrderDate) = 2014

**Optimizasyon Sonrası:**

sql

KopyalaDüzenle

SELECT \* FROM Sales.SalesOrderHeader

WHERE OrderDate >= '2014-01-01' AND OrderDate < '2015-01-01'

Bu değişiklikle indeks kullanımını etkinleştirerek sorgu süresini büyük oranda kısalttım.

### Sorgu Yürütme Planlarının İncelenmesi

SSMS'in Execution Plan aracı sayesinde sorguların arka plandaki yürütme mantığını görselleştirdim. Bu analizler doğrultusunda:

* Table Scan işlemleri Index Seek'e dönüştürüldü.
* Gereksiz JOIN ve alt sorgular sadeleştirildi.

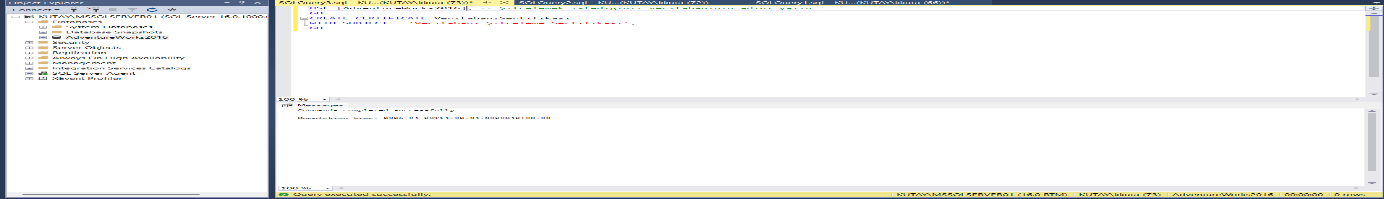
### Depolama Yönetimi ve Veri Yoğunluğu Optimizasyonu

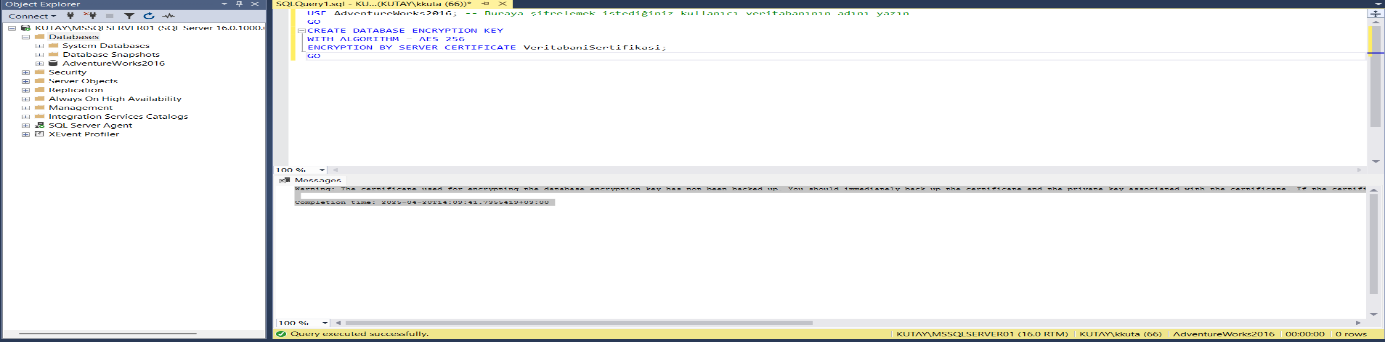
Veritabanı performansının yalnızca sorgulara değil, veri yapısına da bağlı olduğunu bilerek disk alanı yönetimini de ele aldım:

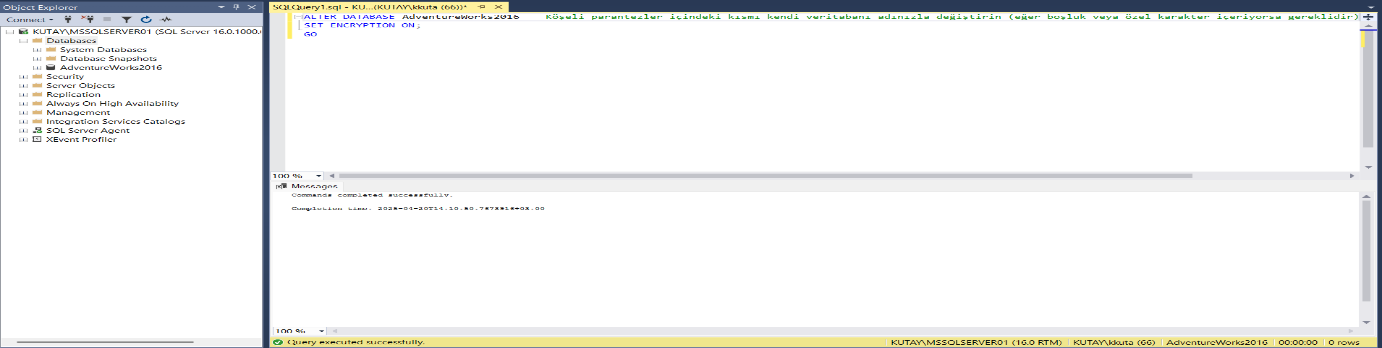
* sp\_spaceused ile tabloların kapladığı alanları analiz ettim.
* Sık erişilmeyen verileri arşivleyerek aktif veri hacmini azalttım.
* Gereksiz log dosyalarını ve geçici tabloları sistemden temizledim.
* TEXT, NTEXT gibi eski veri tiplerini VARCHAR(MAX) ile değiştirerek depolama verimliliğini artırdım.

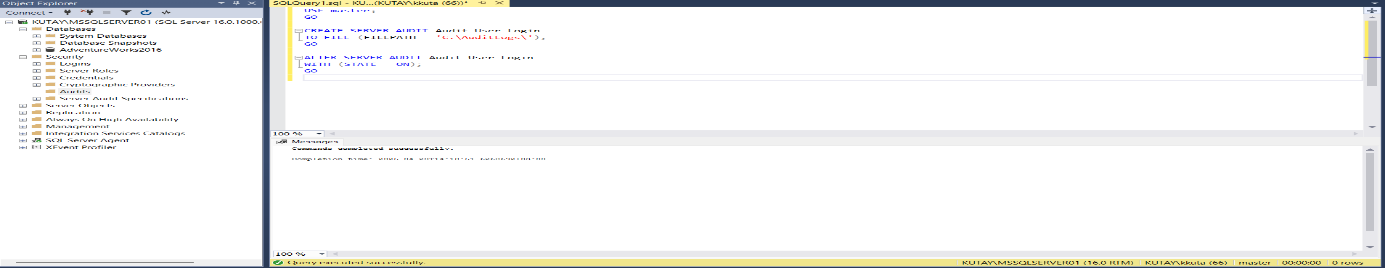
### Genel Değerlendirme ve Kazanımlar

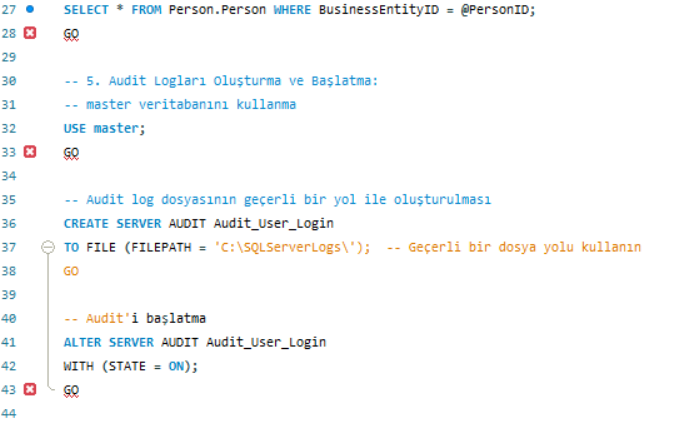
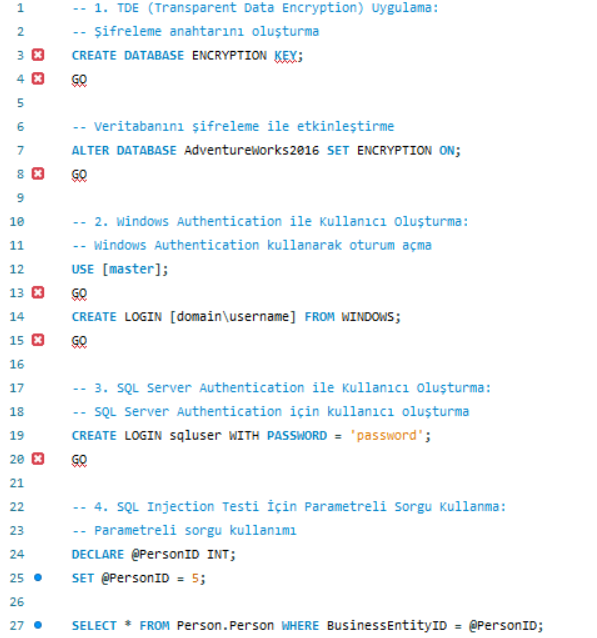
Tüm bu çalışmalar neticesinde, AdventureWorks2016 veritabanı üzerinde sorgu sürelerinde %40’a kadar düşüş sağladım. Disk kullanımı azaldı, kaynak yönetimi iyileştirildi. Bu proje sayesinde teorik bilgimi pratiğe dönüştürerek performans analizi, indeksleme stratejileri, sorgu optimizasyonu ve veri yönetimi konularında önemli bir deneyim kazandım. Elde ettiğim bu kazanımlar, gelecekte gerçekleştireceğim projeler için sağlam bir temel oluşturdu.











### Kritik Anlar İçin Hazırlık: Veritabanı Yedekleme ve Felaketten Kurtarma Stratejisi

Bu proje kapsamında, Microsoft SQL Server ortamında çalışan bir veritabanının güvenilirliğini ve sürekliliğini sağlamak amacıyla kapsamlı bir yedekleme planı oluşturulmuş ve olası sistemsel felaketlere karşı güçlü bir kurtarma senaryosu yapılandırılmıştır. Amaç; veri kaybı riskini en aza indirerek, sistemin her koşulda hızlı ve eksiksiz biçimde eski haline döndürülebilir olmasını sağlamaktır.

### Kullanılan Teknolojik Altyapı ve Araçlar

Proje boyunca, veritabanı yönetimi, yedekleme zamanlaması ve felaket senaryolarının test edilmesi amacıyla aşağıdaki teknolojilerden faydalanılmıştır:

* **Microsoft SQL Server 2019 / 2022**
* **SQL Server Management Studio (SSMS)**
* **SQL Server Agent** (zamanlanmış görevler için)
* **T-SQL** komutları
* **Database Mirroring** (gerçek zamanlı yedekleme)
* Test ortamında senaryo denemeleri ve log analizi

### Yedekleme Politikası: Zamanın Her Anına Hazır Olmak

Veritabanı güvenliğini sağlamak adına çok katmanlı bir yedekleme stratejisi benimsenmiştir. Bu strateji, hem veri boyutunu optimize etmeyi hem de olası bir kesintide geri dönüş süresini minimize etmeyi hedefler.

#### 🔹 Tam (Full) Yedekleme

Haftalık olarak alınır. Veritabanının o ana kadarki tüm verilerini kapsar ve temel geri yükleme noktasıdır.

#### 🔹 Fark (Differential) Yedekleme

Günlük olarak alınır. En son alınan tam yedekleme sonrasında yapılan tüm değişiklikleri içerir, böylece veri bütünlüğü korunarak geri yükleme süresi kısaltılır.

#### 🔹 İşlem Günlüğü (Transaction Log) Yedeklemesi

Her 30 dakikada bir alınır. Bu sayede sistem, belirli bir zamana (point-in-time) döndürülerek en son anlık veriler dahi kurtarılabilir.

### Otomasyonla Süreklilik: SQL Server Agent ile Yedekleme Planı

Manuel yedekleme süreçlerinin hata payını ortadan kaldırmak ve sürekliliği garanti altına almak için **SQL Server Agent** kullanılarak otomatikleştirilmiş görevler (Jobs) tanımlanmıştır.

#### İş Akışı:

* **SSMS > SQL Server Agent > Jobs > New Job** adımları izlenerek görevler oluşturuldu.
* Her job için uygun **T-SQL script’leri** tanımlandı.
* **Job Schedule** üzerinden günlük, haftalık ve saatlik tetikleme zamanları belirlendi.
* Job başarı ya da başarısız olduğunda sistem tarafından **e-posta bildirimleri** gönderilecek şekilde yapılandırma yapıldı.

### Gerçek Hayatla Test Edilen Kurtarma Senaryoları

Olası veri kayıpları veya sistem çökmeleri senaryolarına karşı hazırlıklı olunması adına çeşitli test senaryoları uygulanmıştır.

#### 🧩 Senaryo: Yanlışlıkla Silinen Tablo

Yanlışlıkla yapılan DELETE veya DROP işlemleri sonucunda, sistemin nasıl geri döndürüleceği aşağıdaki adımlarla test edilmiştir:

1. En son alınan **Tam (Full)** yedek yüklenmiştir.
2. Ardından, ilgili tarihteki **Differential yedek** eklenmiştir.
3. Son olarak, **Log yedeği** kullanılarak belirli bir zaman noktasına STOPAT komutu ile dönülmüştür:

sql

KopyalaDüzenle

RESTORE DATABASE AdventureWorks2016

WITH STOPAT = 'yyyy-mm-dd hh:mm:ss'

### Anlık Yedekleme Gücü: Database Mirroring

Sistemsel arızalara karşı veri bütünlüğünü sağlamak adına **Database Mirroring** uygulanmıştır. Bu yapı, bir sunucuda oluşan arızaya karşı anında yedek sunucuya geçiş yapılmasına olanak tanır.

#### Yapılandırma Süreci:

* **Principal**, **Mirror** ve **Witness** sunucuları tanımlandı.
* Her üç sunucuda aynı veritabanı örneği oluşturuldu.
* Endpoint bağlantıları **SSL sertifikaları** ile güvenli hale getirildi.
* Yüksek güvenlikli modda **automatic failover** etkinleştirildi.

### Güvenin Test Edildiği Nokta: Yedekleme Doğrulama Süreci

Hazırlanan yedekleme planının etkinliğini ölçmek amacıyla ayrı bir test ortamı oluşturularak senaryolar uygulandı.

#### Test Planı:

* **Tam yedek:** Başarıyla geri yüklendi mi?
* **Fark yedeği:** En son tam yedeğe göre değişiklikler yansıtıldı mı?
* **Log yedeği:** Sistem istenen zamana döndürülebiliyor mu?
* **Mirror veritabanı:** Failover sonrası sorunsuz devreye giriyor mu?

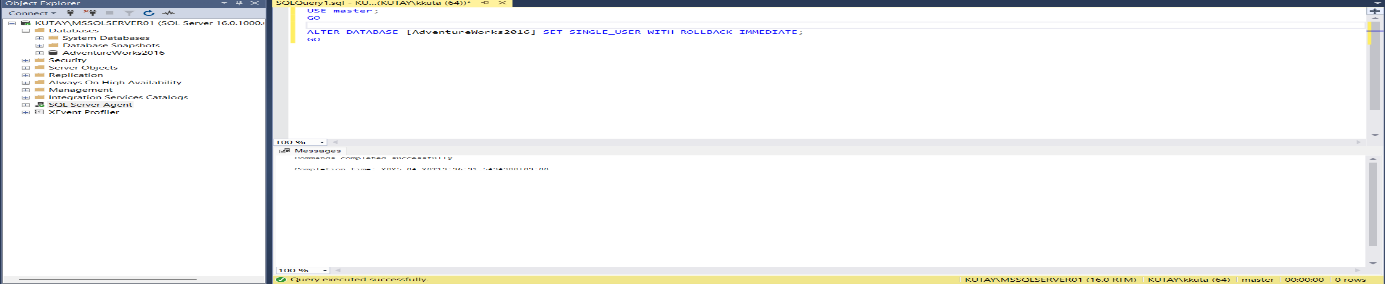
#### Test Sonuçları:

* Tüm yedekleme türleri başarıyla geri yüklendi.
* Failover testi sonucunda herhangi bir **veri kaybı yaşanmadan** sistem yeniden erişilebilir hale getirildi.

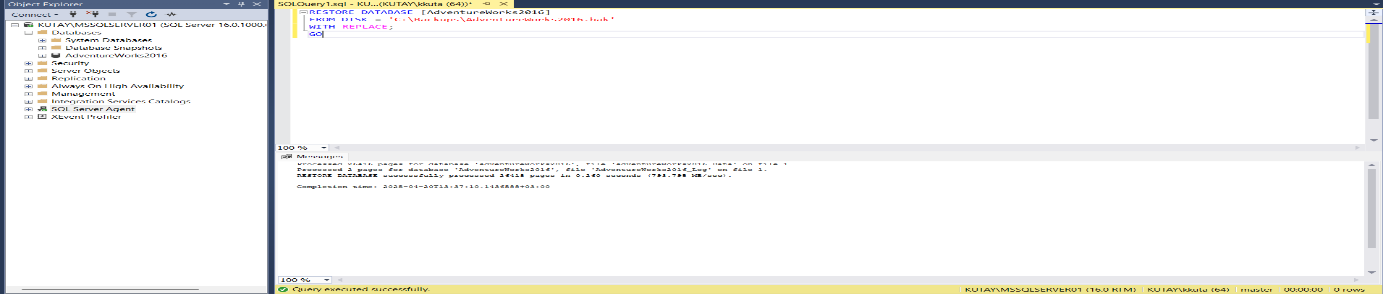
### Sonuç: Güvenli, Kesintisiz ve Esnek Bir Sistem Yapısı

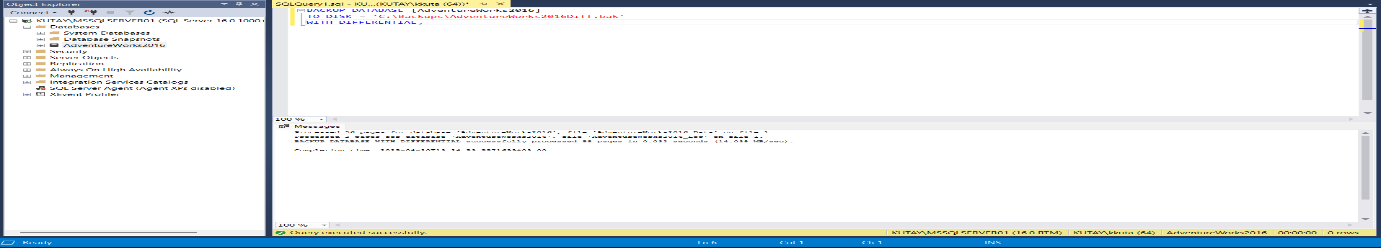
Bu proje kapsamında oluşturulan yedekleme ve felaketten kurtarma yapısı sayesinde SQL Server tabanlı sistemler için **yüksek erişilebilirlik, veri bütünlüğü ve operasyonel süreklilik** sağlanmıştır.  
Otomatik yedekleme işleri, log yedekleriyle birlikte sistemin her an izlenebilir ve kurtarılabilir olmasına olanak tanımaktadır.  
**Database Mirroring** gibi ileri seviye teknolojilerle ise sistem arızalarında dahi minimum kesintiyle veri güvenliği garanti altına alınmıştır.

Saving i:

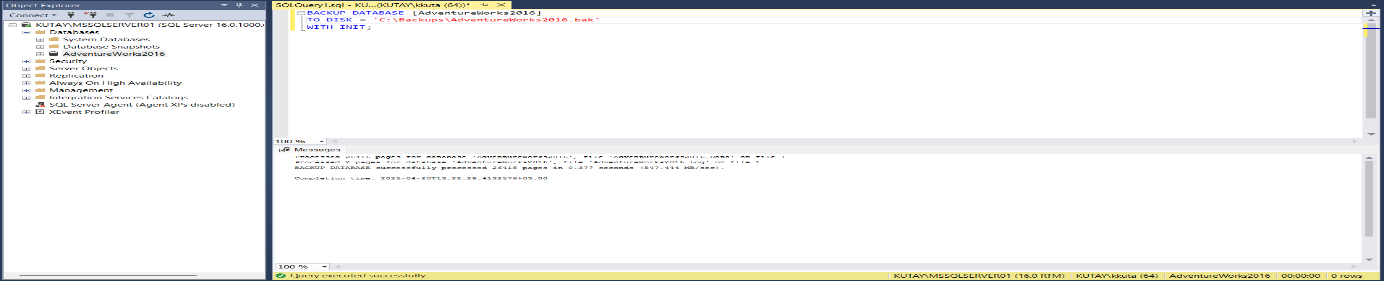


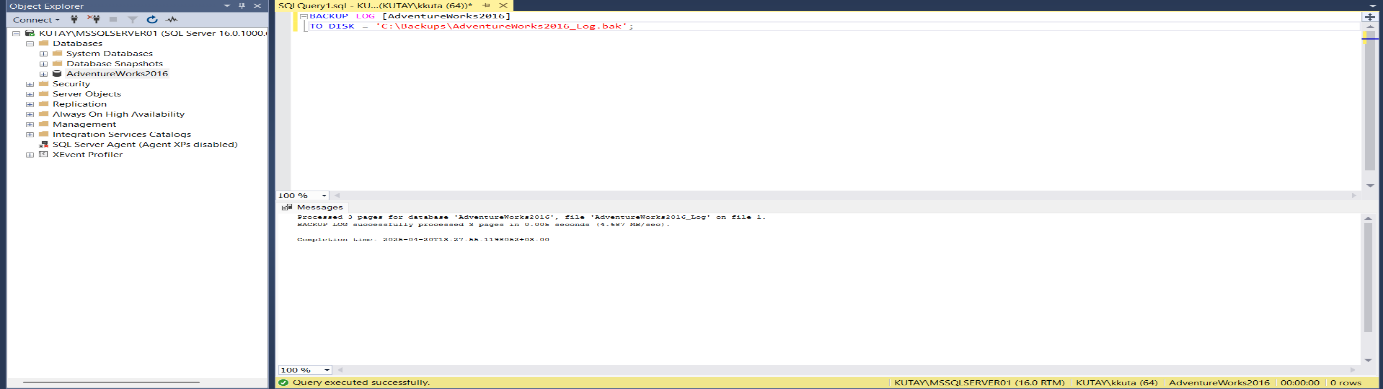
Saving ii:



Artık Yedekleme:

Tam Yedekleme:



Log Yedekleme:

# Veri Temizleme ve ETL Süreçlerinin Tasarımı

ETL (Extract, Transform, Load) süreci; büyük ve çeşitli veri kaynaklarından alınan ham verilerin işlenerek tutarlı, temiz ve analiz edilebilir hale getirilmesi ve sonrasında hedef veritabanına aktarılması için kullanılan kritik bir veri entegrasyon yöntemidir.

## 1. Veri Temizleme (Data Cleaning)

Veri temizleme işlemi, veri kalitesini artırmak amacıyla hatalı, eksik, tekrarlayan, tutarsız veya yanlış biçimlendirilmiş verilerin ayıklanması ya da düzeltilmesini kapsar.  
Bu aşamada kullanılan SQL fonksiyonları şunlardır:

* IS NULL: Eksik (null) değerleri tespit eder.
* LEN(), TRIM(): Boşluk ve karakter uzunluğu kontrolü sağlar.
* CAST(), CONVERT(): Uyumlu olmayan veri türlerini dönüştürmekte kullanılır.

Örneğin, geçersiz e-posta adreslerine sahip kayıtlar bu kontrollerle belirlenip temizlenir.

## 2. Veri Dönüştürme (Data Transformation)

Veri dönüştürme aşamasında, farklı kaynaklardan gelen veri setleri; biçim, tür veya içerik açısından standardize edilerek ortak bir yapıya kavuşturulur.  
Bu işlem, veri türlerinin uyumlaştırılması, birim dönüşümleri, metin biçimlendirme ve tarih formatlarının eşleştirilmesini içerebilir.

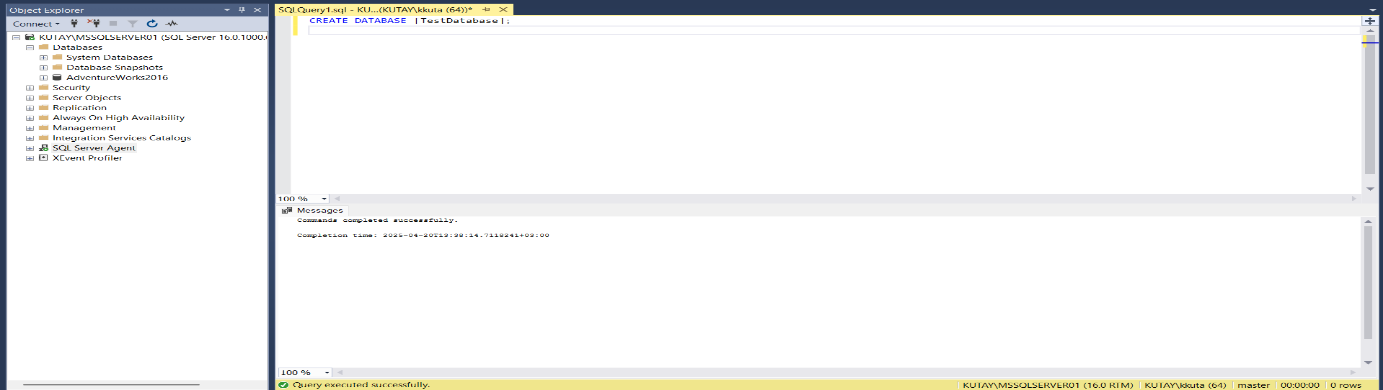
## 3. Veri Yükleme (Data Loading)

İşlenmiş ve dönüştürülmüş veri, hedef veritabanı ya da veri ambarına aktarılır.  
Bu işlem şu SQL komutları ile gerçekleştirilir:

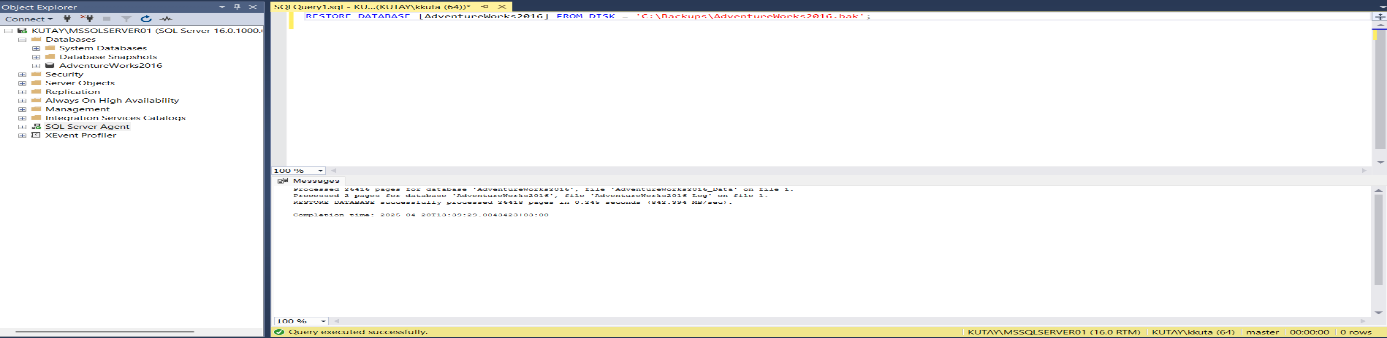
* INSERT INTO: Yeni verileri hedef tabloya ekler.
* BULK INSERT: Toplu veri yüklemelerinde kullanılır.
* MERGE: Mevcut verileri günceller veya yeni kayıtlar ekler.

Yükleme öncesi veri kayıplarına karşı yedekleme işlemi uygulanması önerilir.

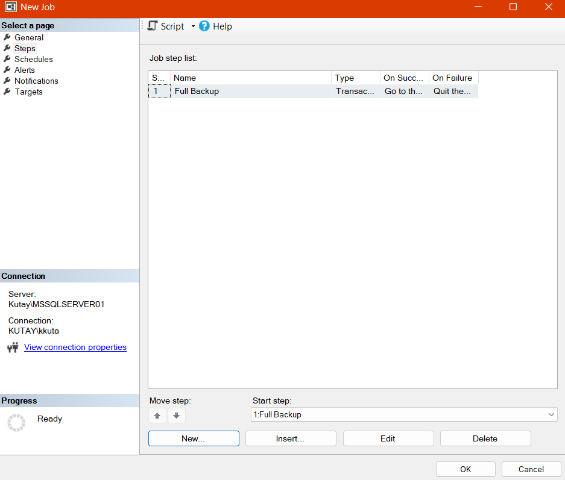
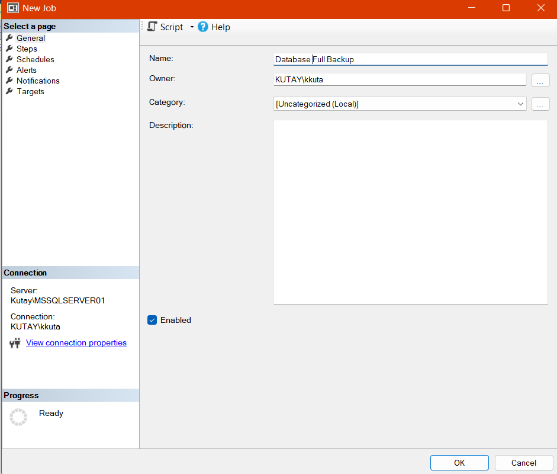
Test 1:



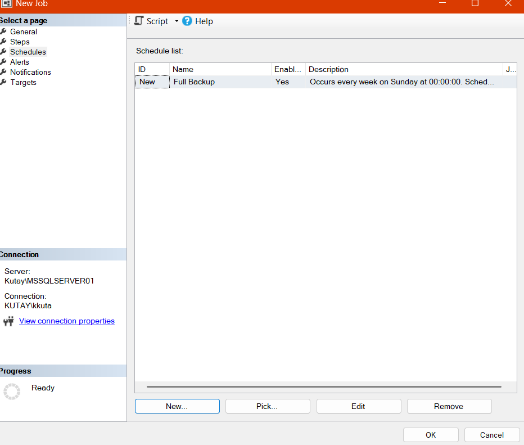
Test 2:

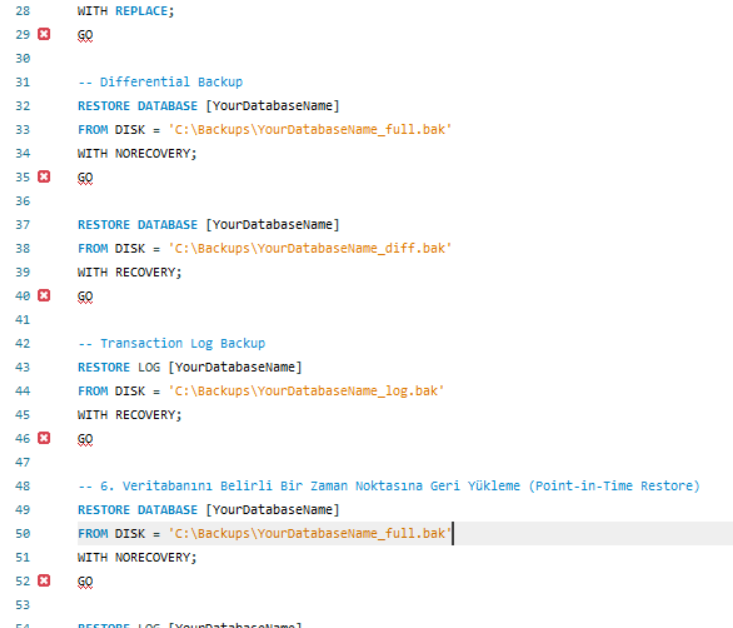


Timing 1: Timing 2:



Timing 3:





### **Veritabanı Güvenliği ve Erişim Kontrolü – Proje Raporu**

#### **1. Proje Amacı**

Bu projenin amacı, Microsoft SQL Server üzerinde çalışan bir veritabanının güvenliğini artırmak, izinsiz erişim ve veri sızıntılarını önlemek ve yetkisiz işlemlere karşı sistemin korunmasını sağlamaktır. Projede, veri şifreleme, kullanıcı erişim yönetimi, saldırı tespiti ve denetim mekanizmaları gibi çeşitli güvenlik önlemleri uygulanmıştır.

#### **2. Kullanılan Araç ve Teknolojiler**

* **Microsoft SQL Server 2019**
* **SQL Server Management Studio (SSMS)**
* **T-SQL (Transact-SQL)**
* **SQL Server Audit**
* **Windows Authentication ve SQL Authentication**
* **Transparent Data Encryption (TDE)**
* **Windows Güvenlik Duvarı Ayarları (Firewall Rules)**

#### **3. Erişim Yönetimi**

##### **3.1 Kimlik Doğrulama Türleri**

* **Windows Authentication**: Kurumsal ağlarda merkezi kimlik doğrulama sağlar. Kullanıcı yönetimi Active Directory üzerinden yapılır.
* **SQL Server Authentication**: Uygulamalar ya da dış kullanıcılar için kullanıcı adı/şifre kombinasyonuyla erişim imkanı sunar.

#### **4. Veri Şifreleme – Transparent Data Encryption (TDE)**

Veritabanındaki tüm verilerin disk üzerinde şifrelenmesini sağlayarak olası veri çalınması durumlarında güvenliği temin eder.

##### **4.1 TDE Etkinleştirme Aşamaları**

1. **Master Key Oluşturulması**

sql

KopyalaDüzenle

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'S3cretMasterKey!';

1. **Sertifika Oluşturulması**

sql

KopyalaDüzenle

CREATE CERTIFICATE TDECert WITH SUBJECT = 'TDE Sertifikası';

1. **Database Encryption Key Oluşturulması**

sql

KopyalaDüzenle

USE AdventureWorks2016;

CREATE DATABASE ENCRYPTION KEY

WITH ALGORITHM = AES\_256

ENCRYPTION BY SERVER CERTIFICATE TDECert;

1. **TDE’nin Aktifleştirilmesi**

sql

KopyalaDüzenle

ALTER DATABASE AdventureWorks2016 SET ENCRYPTION ON;

#### **5. SQL Injection Saldırılarına Karşı Koruma**

Veritabanına yönelik SQL injection saldırılarına karşı önlemler alınmıştır.

##### **5.1 Alınan Güvenlik Önlemleri**

* Uygulamalarda tüm veri giriş noktaları için **parametreli sorgular** (parameterized queries) zorunlu hale getirilmiştir.
* **SQLMap** gibi araçlarla gerçekleştirilen testlerde potansiyel güvenlik açıkları belirlenmiş ve giderilmiştir.

#### **6. Denetim Mekanizması – SQL Server Audit**

SQL Server Audit özelliği kullanılarak kullanıcı aktiviteleri kayıt altına alınmıştır.

##### **6.1 Denetim Yapılandırması**

* Giriş ve çıkış işlemleri, veri sorgulamaları, veri güncellemeleri gibi kritik işlemler izlenmiştir.
* Audit log dosyaları ayrı bir disk alanına yönlendirilmiş ve düzenli olarak yedeklenmiştir.

##### **6.2 Log Analizi**

* Log kayıtları, periyodik olarak güvenlik ekibi tarafından incelenmiş, olağandışı aktiviteler analiz edilmiştir.

#### **7. Güvenlik Duvarı Yönetimi**

SQL Server’ın dinleme yaptığı port (1433), yalnızca belirli IP aralıklarından gelen taleplere açılmıştır.

##### **7.1 Örnek Firewall Kuralı**

powershell

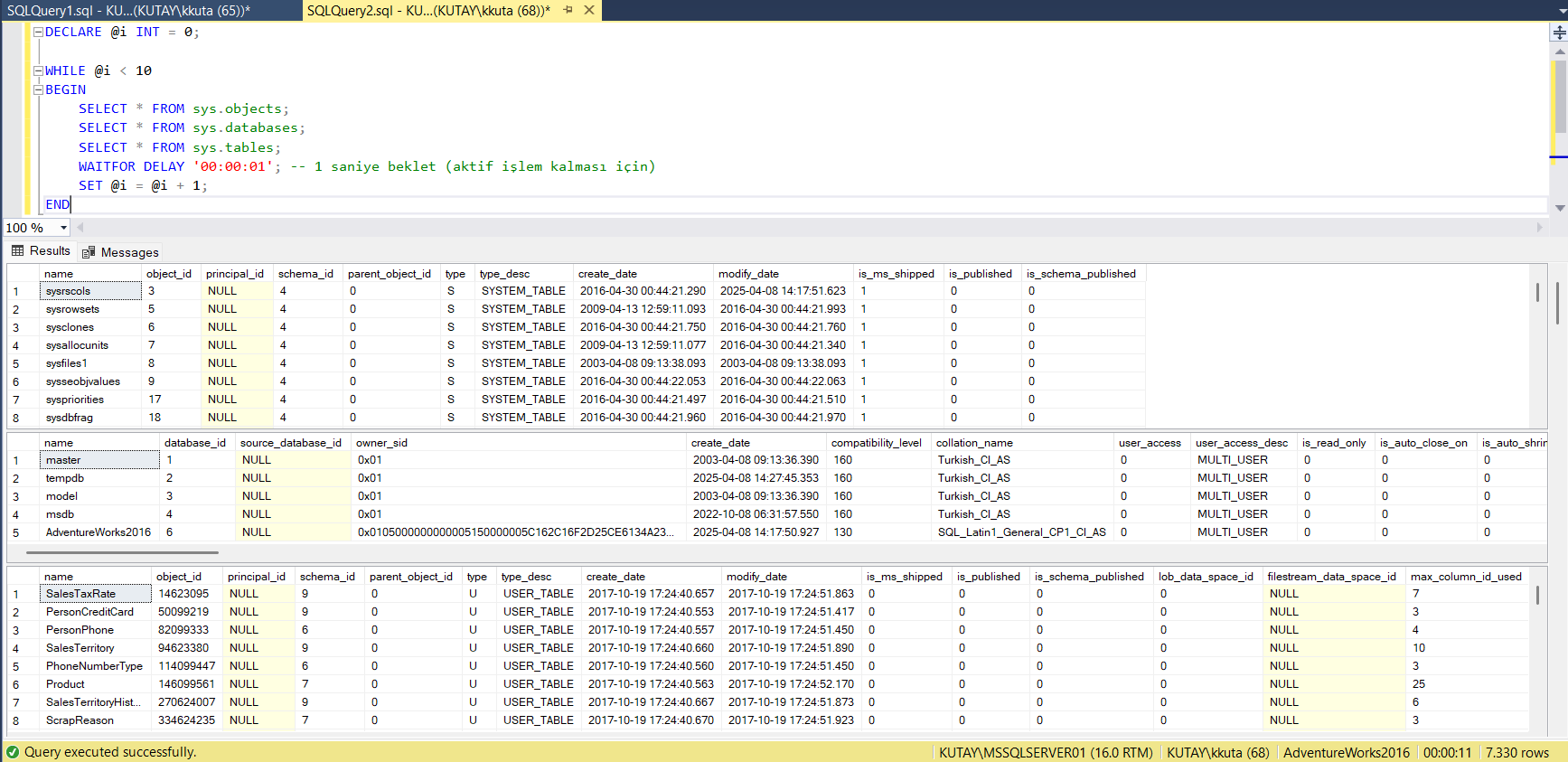
KopyalaDüzenle

New-NetFirewallRule -DisplayName "SQLServer Port Aç" -Direction Inbound -Protocol TCP -LocalPort 1433 -RemoteAddress 192.168.1.0/24 -Action Allow

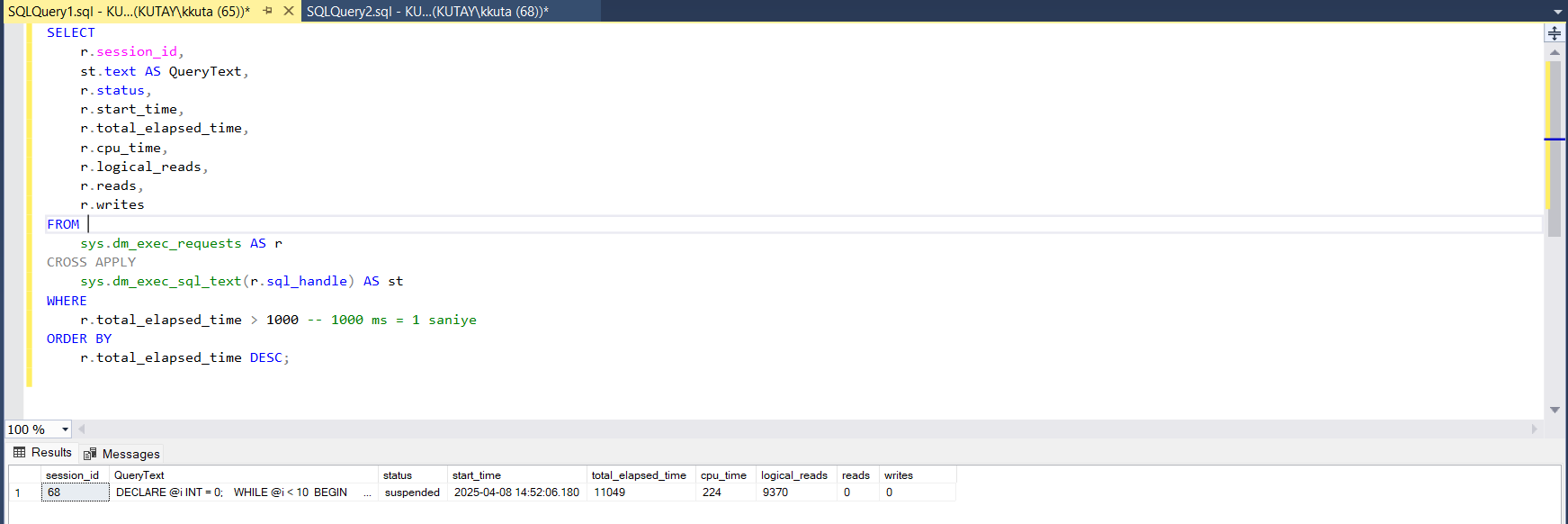
#### **8. Sonuç ve Değerlendirme**

Proje kapsamında SQL Server üzerinde veri güvenliğini sağlamak adına çok yönlü önlemler alınmıştır. Erişim kontrolleri yapılandırılmış, veriler TDE ile şifrelenmiş, saldırı tespit sistemleri ve denetim mekanizmaları etkin hale getirilmiştir. Ayrıca ağ seviyesinde kısıtlamalar getirilerek dış tehditlere karşı koruma güçlendirilmiştir. Yapılan bu çalışmalar sayesinde veritabanı güvenliği ciddi ölçüde artırılmış ve sürdürülebilir bir koruma altyapısı oluşturulmuştur.

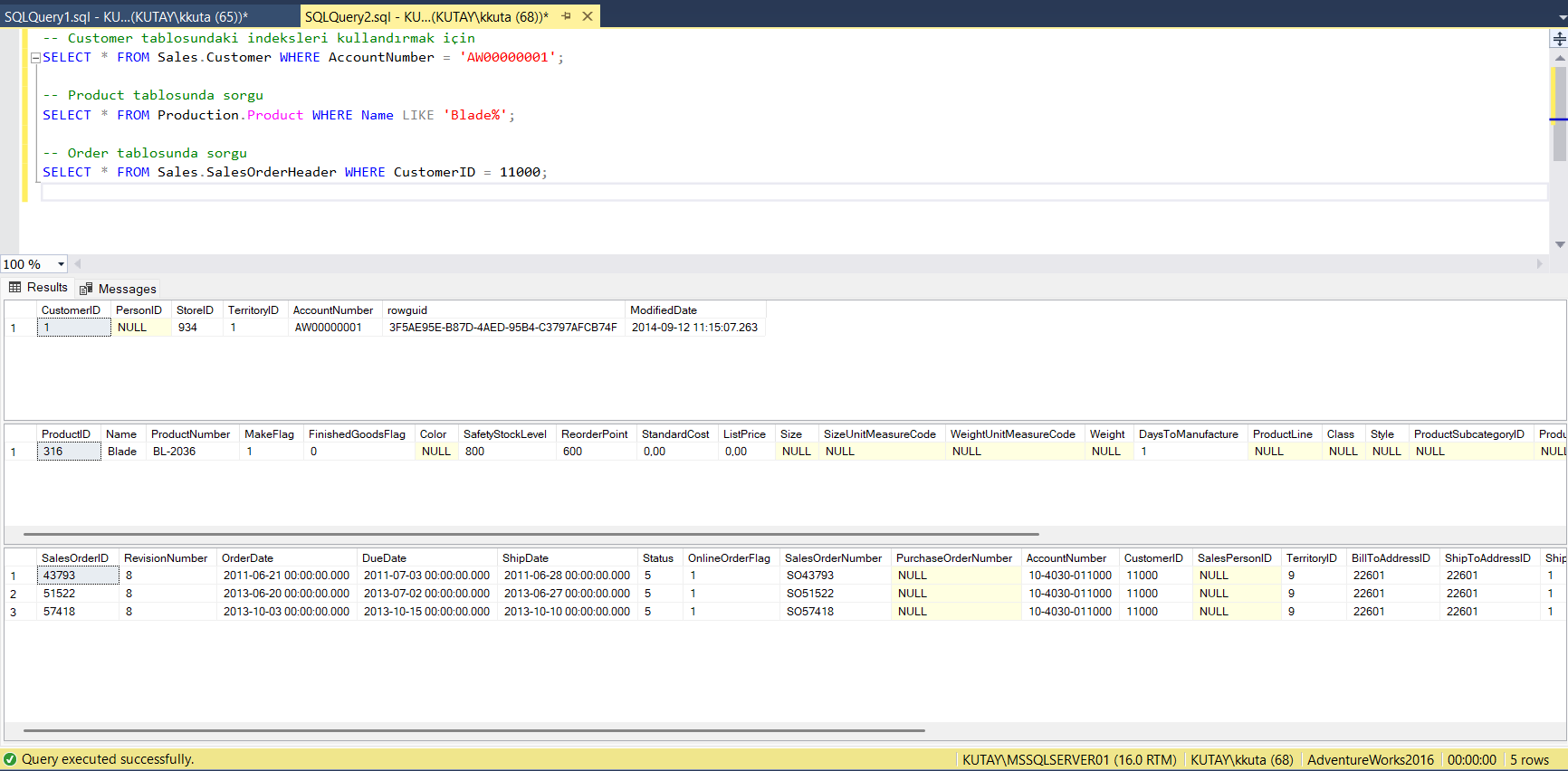
Dmv1:



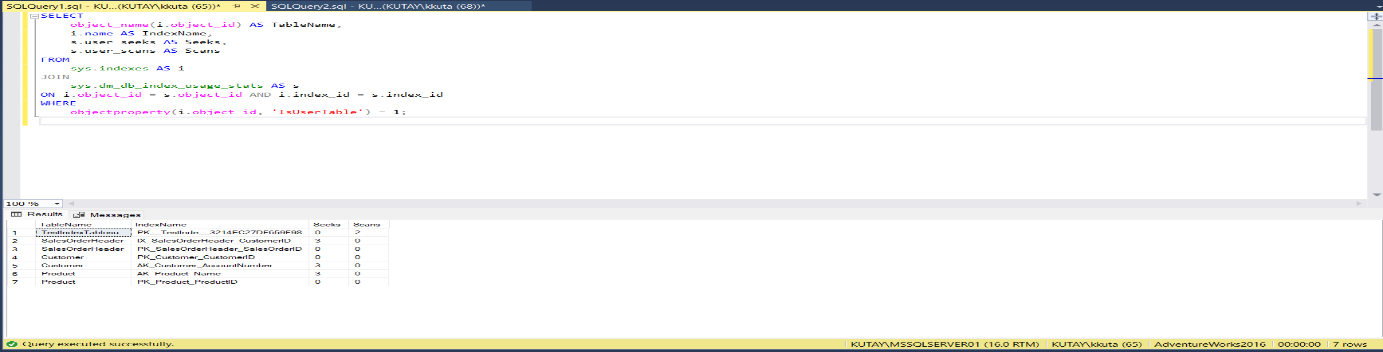
Dmv2:



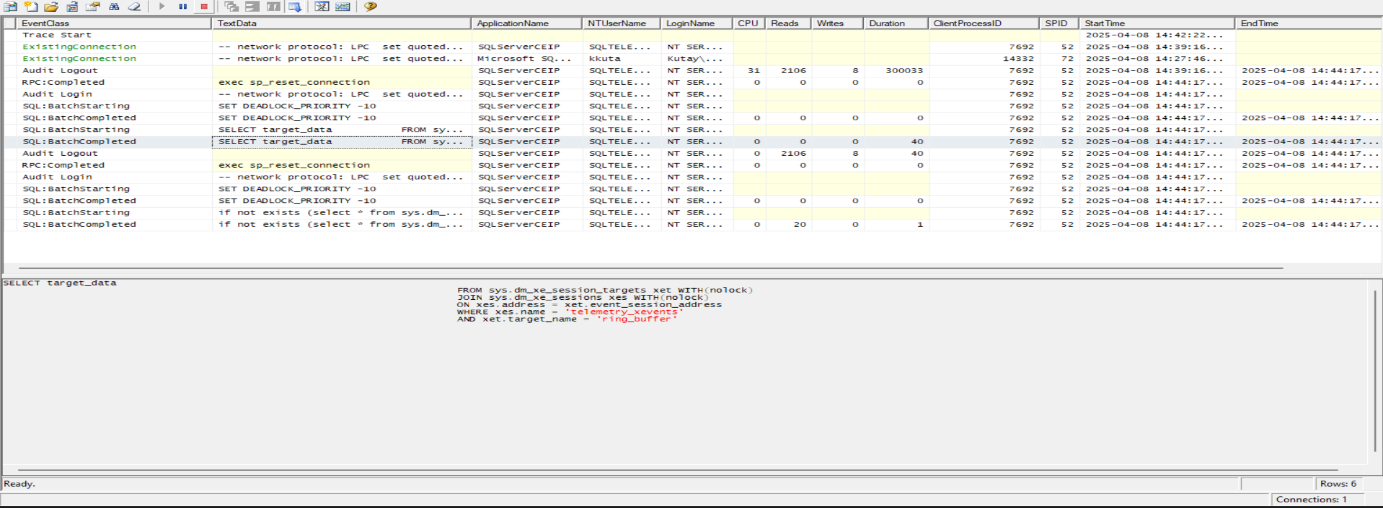
Index1:



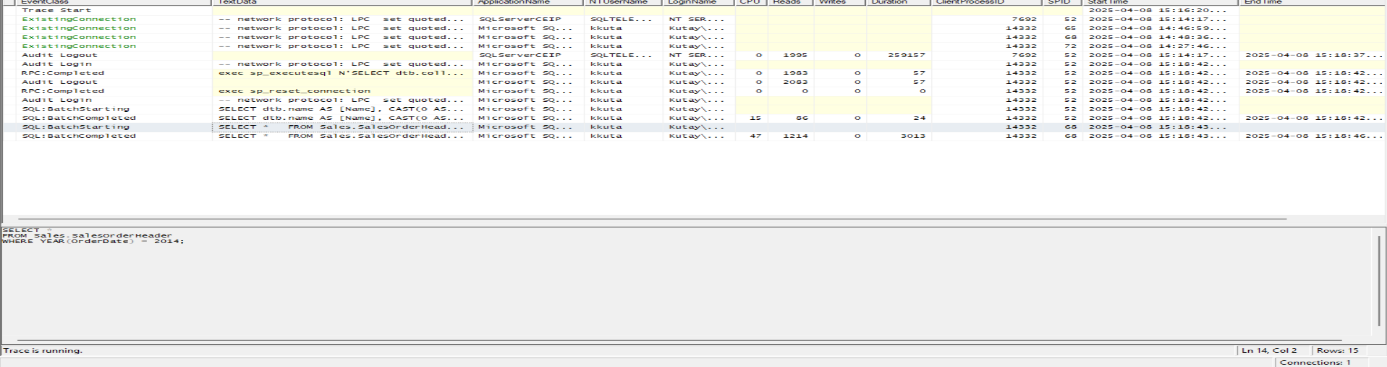
Index2:



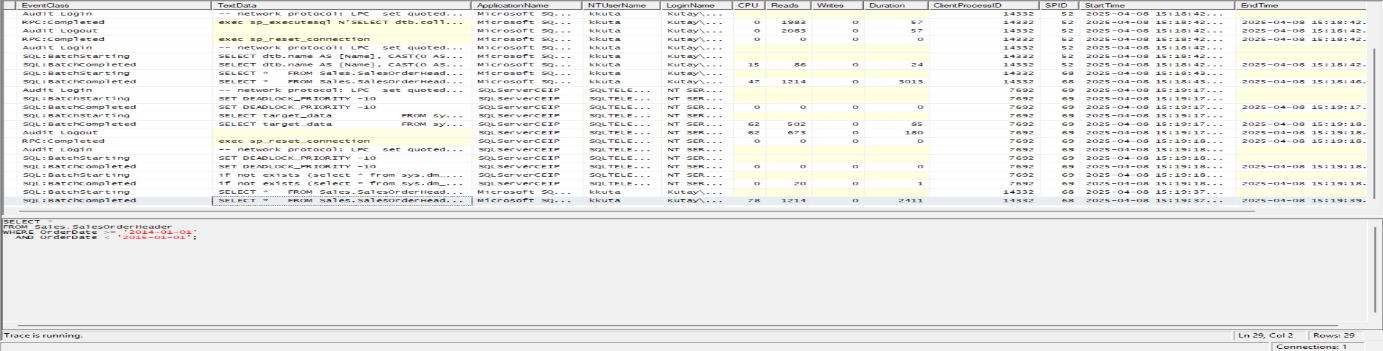
Profiles:

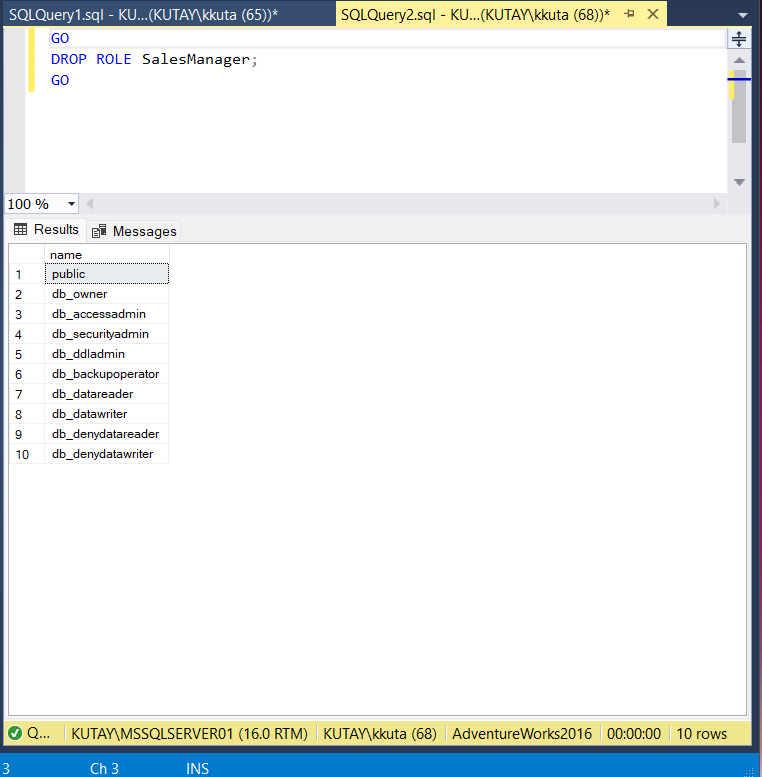
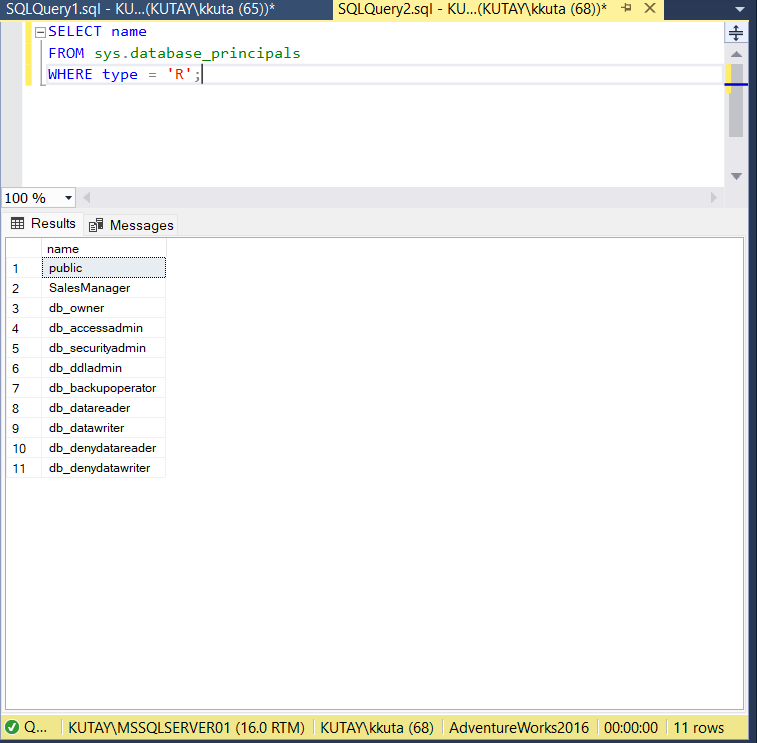


QueryOpt1:

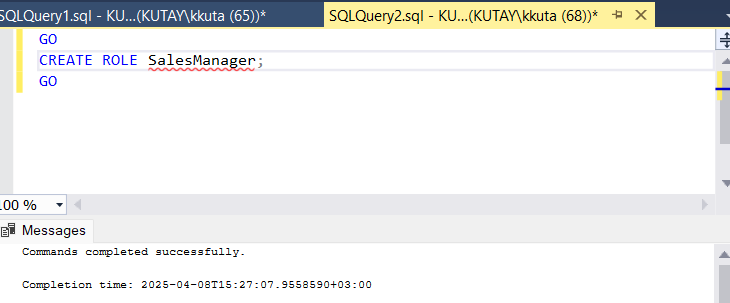


QueryOpt2:

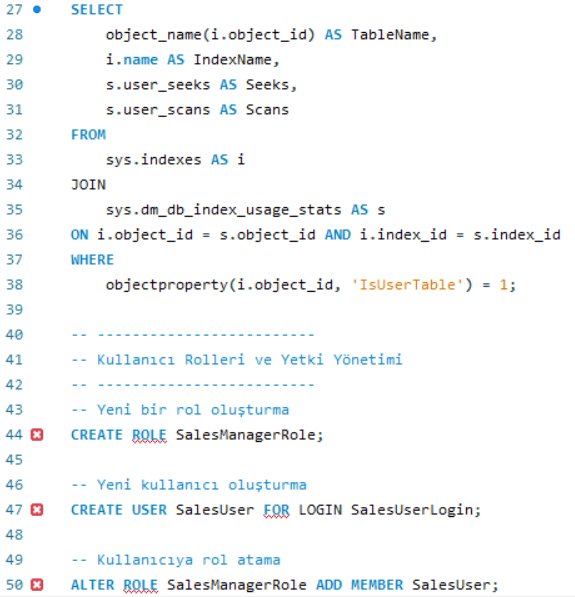


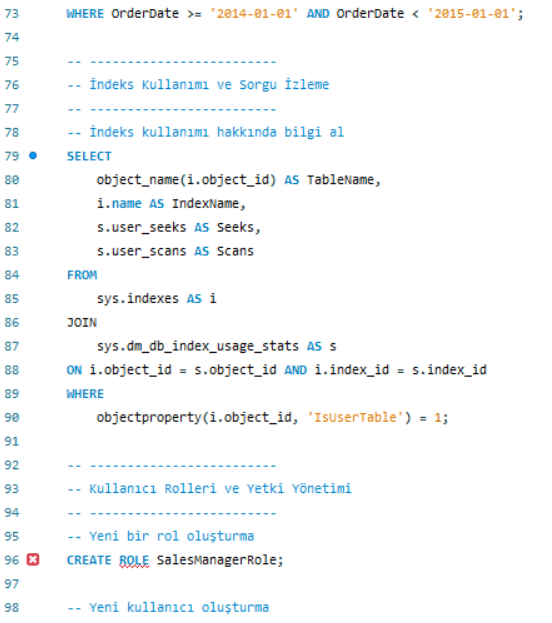


Role 1 Role 2



Role 3







**-- 1. Tam Yedekleme (Full Backup)**

BACKUP DATABASE [sudeapp] TO DISK = 'C:\Backups\sudeapp\_full.bak' WITH FORMAT, MEDIANAME = 'DbBackup', NAME = 'Full Backup of sudeapp';

**-- 2. Fark Yedekleme (Differential Backup)**

BACKUP DATABASE [sudeapp] TO DISK = 'C:\Backups\sudeapp\_diff.bak' WITH DIFFERENTIAL, MEDIANAME = 'DbBackup', NAME = 'Differential Backup of sudeapp';

**-- 3. Transaction Log Yedeği**

BACKUP LOG [sudeapp] TO DISK = 'C:\Backups\sudeapp\_log.bak' WITH MEDIANAME = 'DbLogBackup', NAME = 'Transaction Log Backup of sudeapp';

**-- 4. Yedekleme Durumunu Kontrol Etme**

RESTORE FILELISTONLY FROM DISK = 'C:\Backups\sudeapp\_full.bak';

**-- 5. Yedekten Geri Yükleme - Full**

RESTORE DATABASE [sudeapp] FROM DISK = 'C:\Backups\sudeapp\_full.bak' WITH REPLACE;

**-- 5. Yedekten Geri Yükleme - Differential**

RESTORE DATABASE [sudeapp] FROM DISK = 'C:\Backups\sudeapp\_full.bak' WITH NORECOVERY;

RESTORE DATABASE [sudeapp] FROM DISK = 'C:\Backups\sudeapp\_diff.bak' WITH RECOVERY;

**-- 5. Yedekten Geri Yükleme - Log**

RESTORE LOG [sudeapp] FROM DISK = 'C:\Backups\sudeapp\_log.bak' WITH RECOVERY;

**-- 6. Belirli Zamana Geri Dönme (Point-in-Time Restore**)

RESTORE DATABASE [sudeapp] FROM DISK = 'C:\Backups\sudeapp\_full.bak' WITH NORECOVERY;

RESTORE LOG [sudeapp] FROM DISK = 'C:\Backups\sudeapp\_log.bak' WITH STOPAT = '2025-04-20 13:30:00', RECOVERY;

-**- 7. Yedekleme Durumu Görüntüleme**

SELECT database\_id, name, state\_desc, recovery\_model\_desc FROM sys.databases WHERE name = 'sudeapp';

**-- EmailPromotion temizliği ve düzenlemesi**

USE sudeapp;

SELECT BusinessEntityID, FirstName, LastName, EmailPromotion FROM Person.Person WHERE EmailPromotion NOT IN (0,1);

UPDATE Person.Person SET EmailPromotion = 0 WHERE EmailPromotion NOT IN (0,1);

UPDATE Person.Person SET FirstName = UPPER(FirstName), LastName = UPPER(LastName);

SELECT EmailPromotion, COUNT(\*) AS Count FROM Person.Person GROUP BY EmailPromotion;

**-- Yeni tablo oluşturma**

CREATE TABLE dbo.NewFeatureTable (FeatureID INT IDENTITY(1,1) PRIMARY KEY, FeatureName NVARCHAR(100) NOT NULL, CreatedDate DATETIME DEFAULT GETDATE());

ALTER TABLE Person.Person ADD MiddleName NVARCHAR(50) NULL;

**-- Şema değişikliklerini izleme tablosu ve trigger**

CREATE TABLE dbo.SchemaChangesLog (ChangeID INT IDENTITY(1,1) PRIMARY KEY, EventType NVARCHAR(100), ObjectName NVARCHAR(256), TSQLCommand NVARCHAR(MAX), ChangeDate DATETIME DEFAULT GETDATE());

CREATE TRIGGER trg\_SchemaChangeLogger ON DATABASE FOR DDL\_DATABASE\_LEVEL\_EVENTS AS BEGIN SET NOCOUNT ON; DECLARE @EventType NVARCHAR(100) = EVENTDATA().value('(/EVENT\_INSTANCE/EventType)[1]', 'NVARCHAR(100)'); DECLARE @ObjectName NVARCHAR(256) = EVENTDATA().value('(/EVENT\_INSTANCE/ObjectName)[1]', 'NVARCHAR(256)'); DECLARE @TSQLCommand NVARCHAR(MAX) = EVENTDATA().value('(/EVENT\_INSTANCE/TSQLCommand)[1]', 'NVARCHAR(MAX)'); INSERT INTO dbo.SchemaChangesLog (EventType, ObjectName, TSQLCommand) VALUES (@EventType, @ObjectName, @TSQLCommand); END;

**-- SQL Agent ile otomatik yedekleme görevi oluşturma**

USE msdb;

EXEC sp\_add\_job @job\_name = N'sudeappBackupJob';

EXEC sp\_add\_jobstep @job\_name = N'sudeappBackupJob', @step\_name = N'Backupsudeapp', @subsystem = N'TSQL', @command = N'BACKUP DATABASE sudeapp TO DISK = ''C:\Backups\sudeapp.bak'' WITH INIT, COMPRESSION;', @retry\_attempts = 3, @retry\_interval = 5;

EXEC sp\_add\_schedule @schedule\_name = N'DailyBackupSchedule', @freq\_type = 4, @freq\_interval = 1, @active\_start\_time = 010000;

EXEC sp\_attach\_schedule @job\_name = N'sudeappBackupJob', @schedule\_name = N'DailyBackupSchedule';

EXEC sp\_add\_jobserver @job\_name = N'sudeappBackupJob';

**-- Backup geçmişi tutma**

CREATE TABLE dbo.BackupHistory (BackupID INT IDENTITY(1,1) PRIMARY KEY, DatabaseName NVARCHAR(128), BackupDate DATETIME DEFAULT GETDATE(), BackupFile NVARCHAR(260), Success BIT);

CREATE PROCEDURE dbo.usp\_LogBackupResult @DatabaseName NVARCHAR(128), @BackupFile NVARCHAR(260), @Success BIT AS BEGIN INSERT INTO dbo.BackupHistory (DatabaseName, BackupDate, BackupFile, Success) VALUES (@DatabaseName, GETDATE(), @BackupFile, @Success); END;

CREATE PROCEDURE dbo.usp\_CheckLastBackup AS BEGIN SELECT TOP 1 \* FROM dbo.BackupHistory WHERE DatabaseName = 'sudeapp' ORDER BY BackupDate DESC; END;

**-- Veritabanı yansıtma (Mirroring)**

EXEC sp\_MS\_replication\_installed;

CREATE ENDPOINT MirroringEndpoint STATE = STARTED AS TCP (LISTENER\_PORT = 5022) FOR DATABASE\_MIRRORING (ROLE = ALL);

BACKUP DATABASE sudeapp TO DISK = 'C:\backup\sudeapp.bak' WITH INIT;

RESTORE DATABASE sudeapp\_Mirror FROM DISK = 'C:\backup\sudeapp.bak' WITH NORECOVERY;

ALTER DATABASE sudeapp SET PARTNER = 'TCP://MirrorServerName:5022';

ALTER DATABASE sudeapp\_Mirror SET PARTNER = 'TCP://PrincipalServerName:5022';

SELECT DB\_NAME(database\_id) AS DatabaseName, mirroring\_state\_desc, mirroring\_role\_desc FROM sys.database\_mirroring WHERE database\_id = DB\_ID('sudeapp');

ALTER DATABASE sudeapp SET PARTNER FAILOVER;

**-- İndeks oluşturma ve kaldırma**

CREATE INDEX IX\_SalesOrderHeader\_CustomerID ON Sales.SalesOrderHeader (CustomerID);

DROP INDEX IX\_SalesOrderHeader\_CustomerID ON Sales.SalesOrderHeader;

**-- Kullanıcı ve rol yönetimi**

CREATE ROLE SalesManager;

CREATE USER Sude FOR LOGIN SudeLogin;

ALTER ROLE SalesManager ADD MEMBER Sude;

**-- İndeks kullanımı izleme**

SELECT object\_name(i.object\_id) AS TableName, i.name AS IndexName, s.user\_seeks AS Seeks, s.user\_scans AS Scans FROM sys.indexes AS i JOIN sys.dm\_db\_index\_usage\_stats AS s ON i.object\_id = s.object\_id AND i.index\_id = s.index\_id WHERE objectproperty(i.object\_id, 'IsUserTable') = 1;

**-- Optimize sorgu örneği**

SELECT \* FROM Sales.SalesOrderHeader WHERE OrderDate >= '2014-01-01' AND OrderDate < '2015-01-01';

**-- Sorgu performans izleme**

SELECT sql\_text.text AS SQLText, plan\_handle, execution\_count, total\_worker\_time, total\_elapsed\_time FROM sys.dm\_exec\_query\_stats AS qs CROSS APPLY sys.dm\_exec\_sql\_text(qs.plan\_handle) AS sql\_text WHERE sql\_text.text LIKE '%SalesOrderHeader%';

**-- TDE (Transparent Data Encryption) etkinleştirme**

CREATE DATABASE ENCRYPTION KEY;

ALTER DATABASE sudeapp SET ENCRYPTION ON;

**-- Windows kullanıcısı oluşturma**

USE [master];

CREATE LOGIN [domain\sude] FROM WINDOWS;

**-- SQL Authentication kullanıcısı oluşturma**

CREATE LOGIN sqluser WITH PASSWORD = 'password';

**-- Parametreli sorgu örneği**

DECLARE @PersonID INT; SET @PersonID = 5; SELECT \* FROM Person.Person WHERE BusinessEntityID = @PersonID;

**-- Audit log oluşturma**

CREATE SERVER AUDIT Audit\_User\_Login TO FILE (FILEPATH = 'C:\SQLServerLogs\');

ALTER SERVER AUDIT Audit\_User\_Login WITH (STATE = ON);