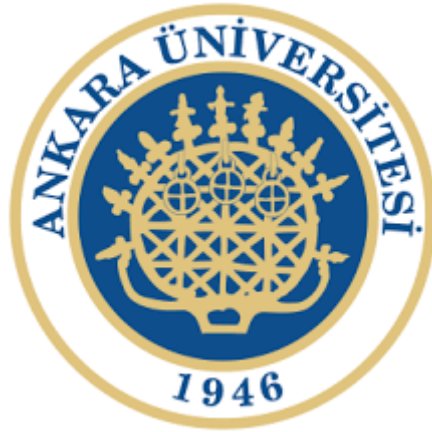


Ankara Üniversitesi Bilgisayar Mühendisliği
Ağ Tabanlı Paralel Dağıtım Sistemleri(BLM 4522)
Proje Ödevi



Doruk Ulaş Akbıyık-19291093

Erdem Demirbaş-20290242

<https://github.com/dorukku/MSSQL>

https://drive.google.com/drive/folders/1q5JewtDab_4IkU8PVEJsK_1MVAYNavVD?usp=sharing

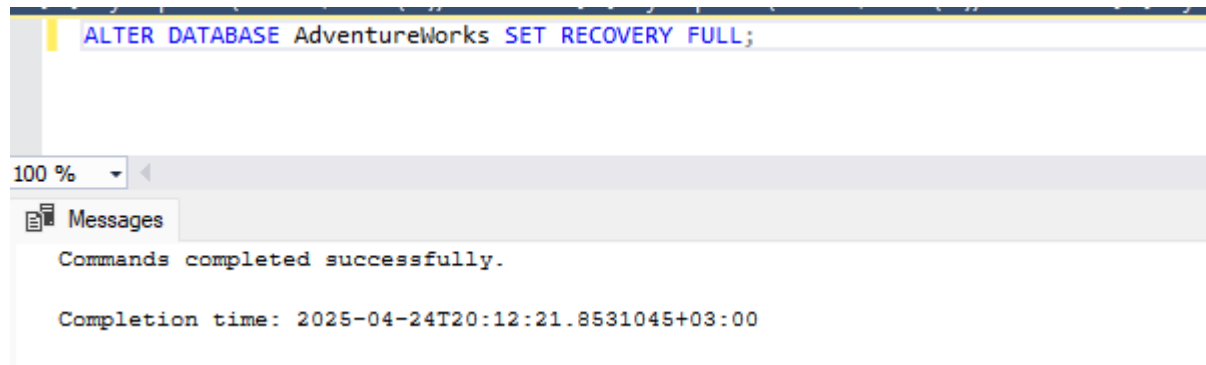
Proje 2: Veritabanı Yedekleme Ve Felaketten Kurtarma Planı

Giriş

Bu proje, SQL Server üzerinde çalışan bir veritabanının yedeklenmesi, otomatikleştirilmesi ve felaket senaryolarına karşı kurtarılmasını konu alır. Projede AdventureWorks2019 veritabanı örnek olarak kullanılmıştır.

1. Adım: Recovery Model Ayarının Yapılması

Projenin başında, AdventureWorks2019 veritabanının yedekleme işlemlerini tam anlamıyla yapabilmek için **kurtarma modelini (recovery model)** değiştirdim.



```
ALTER DATABASE AdventureWorks SET RECOVERY FULL;
```

100 %

Messages

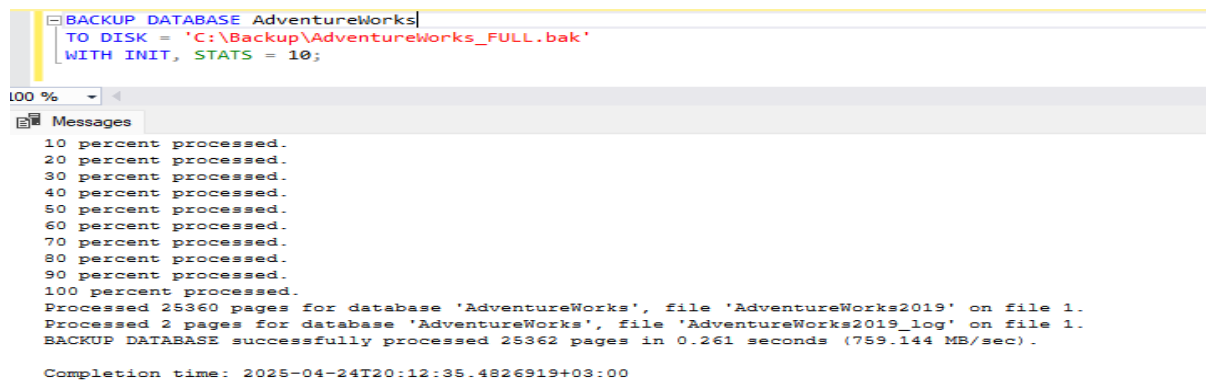
Commands completed successfully.

Completion time: 2025-04-24T20:12:21.8531045+03:00

2. Adım: Tam Yedekleme İşlemi

Veritabanının tam yedeğini aldım.

Bu, veritabanının o andaki tüm haliyle saklandığı bir yedek türüdür.
Felaket senaryosu durumunda önce bunu geri yüklemem gerekecek.



```
BACKUP DATABASE AdventureWorks  
TO DISK = 'C:\Backup\AdventureWorks_FULL.bak'  
WITH INIT, STATS = 10;
```

100 %

Messages

10 percent processed.
20 percent processed.
30 percent processed.
40 percent processed.
50 percent processed.
60 percent processed.
70 percent processed.
80 percent processed.
90 percent processed.
100 percent processed.
Processed 25360 pages for database 'AdventureWorks', file 'AdventureWorks2019' on file 1.
Processed 2 pages for database 'AdventureWorks', file 'AdventureWorks2019_log' on file 1.
BACKUP DATABASE successfully processed 25362 pages in 0.261 seconds (759.144 MB/sec).

Completion time: 2025-04-24T20:12:35.4826919+03:00

3. Differential Yedeği Alma

Full yedekten sonra veritabanında değişiklik yapılırsa, sadece değişen verileri içeren fark yedeği aldım.

Bu işlem, full yedeğe ek olarak kullanılır.

```
BACKUP DATABASE AdventureWorks
TO DISK = 'C:\Backup\AdventureWorks_DIFF.bak'
WITH DIFFERENTIAL, INIT, STATS = 10;
```

100 %

Messages

57 percent processed.
76 percent processed.
95 percent processed.
100 percent processed.
Processed 56 pages for database 'AdventureWorks', file 'AdventureWorks2019' on file 1.
Processed 2 pages for database 'AdventureWorks', file 'AdventureWorks2019_log' on file 1.
BACKUP DATABASE WITH DIFFERENTIAL successfully processed 58 pages in 0.023 seconds (19.531 MB/sec).

Completion time: 2025-04-24T20:12:46.6766665+03:00

4. Log Yedeği Alma

Log yedeği, veritabanında yapılan işlemleri adım adım saklar.

Özellikle zamana göre restore yapmak için önemlidir.

Veritabanında değişiklik yaptıktan sonra log yedeğini aldım.

```
BACKUP LOG AdventureWorks
TO DISK = 'C:\Backup\AdventureWorks_LOG1.trn'
WITH INIT, STATS = 10;
```

100 %

Messages

100 percent processed.
Processed 8 pages for database 'AdventureWorks', file 'AdventureWorks2019_log' on file 1.
BACKUP LOG successfully processed 8 pages in 0.009 seconds (6.944 MB/sec).

Completion time: 2025-04-24T20:12:59.3980613+03:00

5. SQL Server Agent ile Otomatik Yedekleme

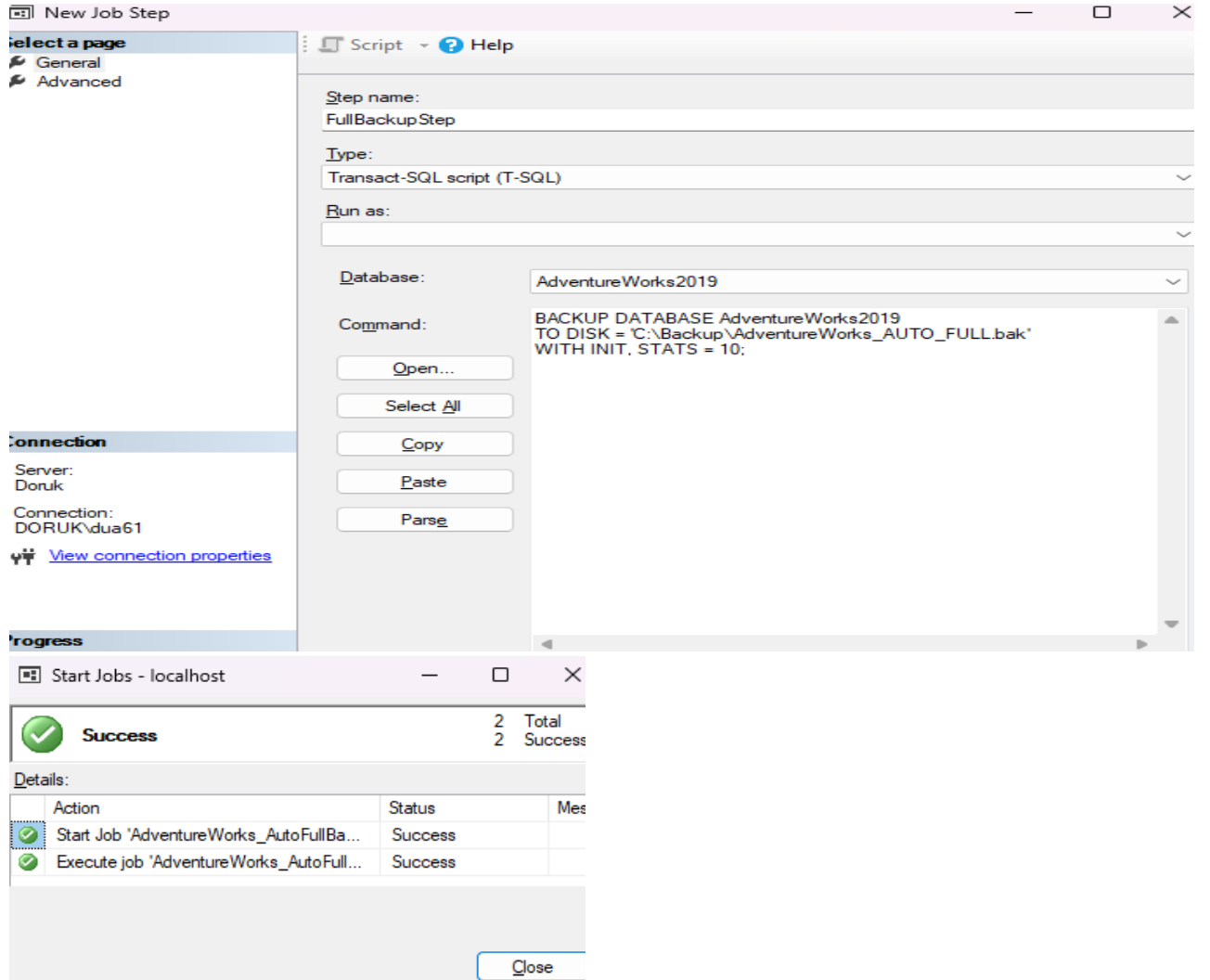
Yedeklerin her zaman manuel alınması zor olacağı için SQL Server Agent kullanarak bir job oluşturdum.

Bu job her gün saat 03:00'te full yedek alacak şekilde ayarlandı.

Job içinde kullandığım komut yine aynı şekilde full backup alıyor.

Job'ın adı: AdventureWorks_AutoFullBackup

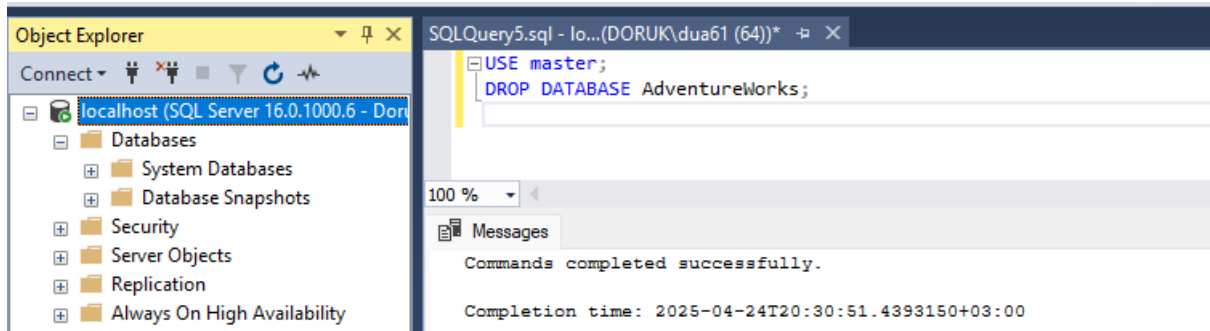
Otomatik çalışması için zamanlama da ekledim. SSMS üzerinden ekran görüntüsü aldım.



6. Felaket Senaryosu - Veritabanı Silme

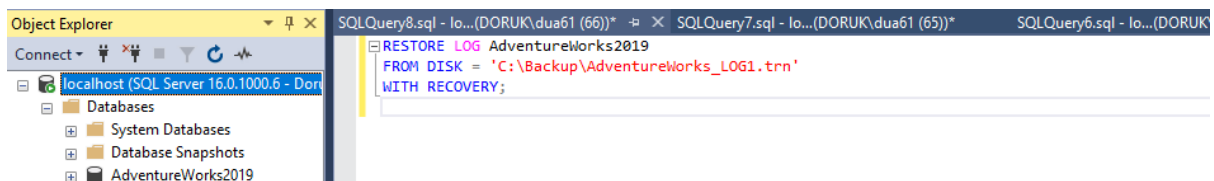
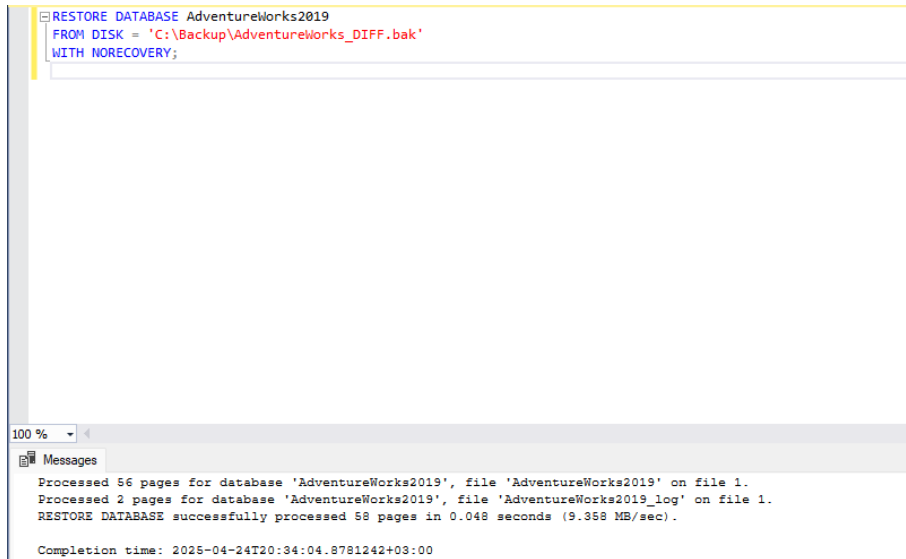
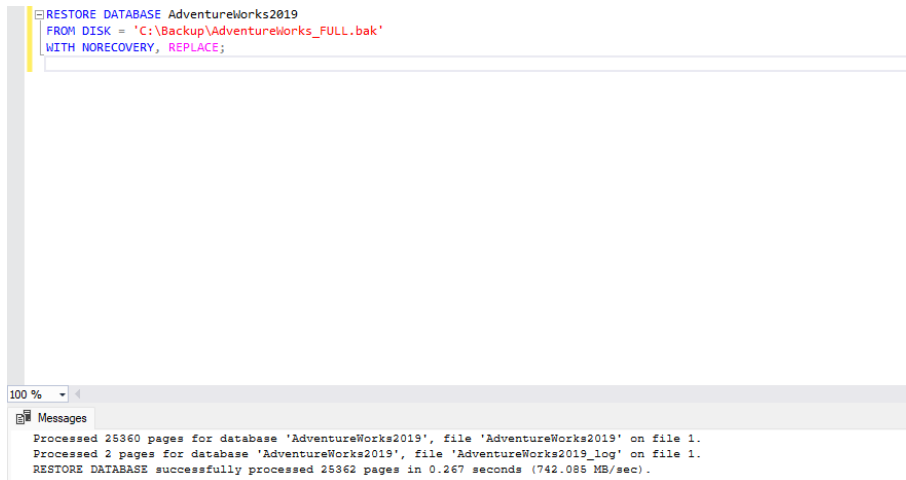
Gerçek hayattaki bir veri kaybını simüle etmek için veritabanını bilerek sildim.

Bu işlemten sonra veritabanını tamamen geri getirmeye çalıştım.



7. Geri Yükleme (Restore) Adımları

Veritabanını önce full yedek ile, sonra diff ve log yedekleriyle sırasıyla geri yükledim.



8. Yedeğin Doğruluğunu Test Etme

Geri yükleme sonrası bu verinin geri geldiğini kontrol ettim.

USE Adventureworks2019;
GO

SELECT TOP 5 * FROM Person.Person;

100 %
Results Messages

	BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	EmailPromotion	AdditionalContactInfo	Demographics	rowguid	ModifiedDate
1	1	EM	0	NULL	Ken	J	Sánchez	NULL	0	NULL	<IndividualSurvey xmlns="http://schemas.microsoft.com/SQL/DM/IndividualSurvey" ...	92C4279F-1207-48A3-8448-4636514EB7E2	2009-01-07 00:00:00.000
2	2	EM	0	NULL	Teri	Lee	Duffy	NULL	1	NULL	<IndividualSurvey xmlns="http://schemas.microsoft.com/SQL/DM/IndividualSurvey" ...	D8763459-8A48-47CC-AFF7-C9079AF79033	2008-01-24 00:00:00.000
3	3	EM	0	NULL	Roberto	NULL	Tamburello	NULL	0	NULL	<IndividualSurvey xmlns="http://schemas.microsoft.com/SQL/DM/IndividualSurvey" ...	E1A2556E-0828-434B-A338-6F38136A37DE	2007-11-04 00:00:00.000
4	4	EM	0	NULL	Rob	NULL	Walters	NULL	0	NULL	<IndividualSurvey xmlns="http://schemas.microsoft.com/SQL/DM/IndividualSurvey" ...	F2D7CE06-38B3-4357-8058-F46B671C01FF	2007-11-28 00:00:00.000
5	5	EM	0	Ms.	Gail	A	Erickson	NULL	0	NULL	<IndividualSurvey xmlns="http://schemas.microsoft.com/SQL/DM/IndividualSurvey" ...	F3A3F6B4-AE3B-430C-A754-9F2231BA6FEF	2007-12-30 00:00:00.000

PROJE 1: Veritabanı Performans Optimizasyonu ve İzleme

Veritabanı: AdventureWorksDW2019

Hazırlayan: Erdem Demirbaş 20290242

Teslim: 25.04.2025

İÇİNDEKİLER

1. Proje Amacı

2. Kullanılan Veritabanı ve Tablolar

3. Uygulama Adımları

1. SQL Profiler ile İzleme
2. İndex Analizi ve Kullanım Durumu
3. Sorgu Optimizasyonu ve Execution Plan
4. Erişim Kontrolü ve Rol Ataması

4.Sonuç

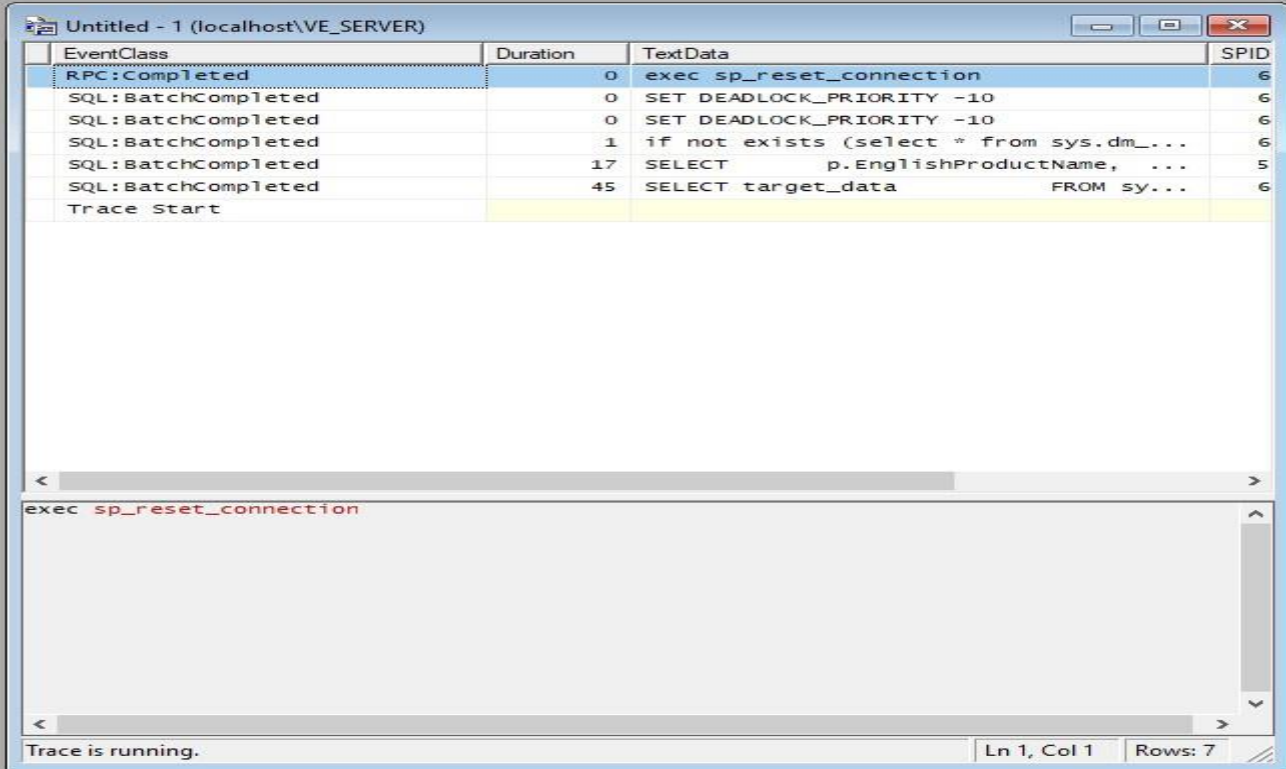
1. PROJE AMACI: Bu projenin amacı, yüksek hacimli bir veri ambarı olan AdventureWorksDW2019 veritabanı üzerinde performans izleme, indeks optimizasyonu, sorgu iyileştirme ve rol bazlı erişim kontrolleri yapmaktır.

2. KULLANILAN VERİTABANI VE TABLOLAR

****Veritabanı:** AdventureWorksDW2019 ****Tablolar:** - FactInternetSales - DimProduct ---****

3. Uygulama Adımları

3.1 SQL Profiler ile İzleme : SQL Server Profiler aracı kullanılarak sistemde yürütülen sorguların çalışma süresi analiz edilmiştir. `SQL:BatchCompleted` olayları ile `Duration` değerleri izlenmiş ve yavaş sorgular belirlenmiştir.



EventClass	Duration	TextData	SPID
RPC:Completed	0	exec sp_reset_connection	6
SQL:BatchCompleted	0	SET DEADLOCK_PRIORITY -10	6
SQL:BatchCompleted	0	SET DEADLOCK_PRIORITY -10	6
SQL:BatchCompleted	1	if not exists (select * from sys.dm_...	6
SQL:BatchCompleted	17	SELECT p.EnglishProductName, ...	5
SQL:BatchCompleted	45	SELECT target_data FROM sy...	6
Trace Start			

exec sp_reset_connection

Trace is running. Ln 1, Col 1 Rows: 7

3.2 İndex Analizi ve Kullanım Durumu : `FactInternetSales` tablosundaki mevcut indeksler listelenmiş ve sadece

primary key'e ait bir clustered index bulunduđu görölmüştür.

SQLQuery1.sql - I...41BRDM\erdem (51))* -> X

```
SELECT
    p.EnglishProductName,
    SUM(f.OrderQuantity) AS TotalOrdered
FROM DimProduct p
JOIN FactInternetSales f ON p.ProductKey = f.ProductKey
GROUP BY p.EnglishProductName
ORDER BY TotalOrdered DESC;
```

100 %

Results Messages

	EnglishProductName	TotalOrdered
1	Water Bottle - 30 oz.	4244
2	Patch Kit/8 Patches	3191
3	Mountain Tire Tube	3095
4	Road Tire Tube	2376
5	Sport-100 Helmet, Red	2230
6	AWC Logo Cap	2190
7	Sport-100 Helmet, Blue	2125
8	Fender Set - Mountain	2121
9	Sport-100 Helmet, Black	2085
10	Mountain Bottle Cage	2025
11	Road Bottle Cage	1712
12	Touring Tire Tube	1488
13	HL Mountain Tire	1396
14	ML Mountain Tire	1161
15	LL Road Tire	1044
16	Touring Tire	935
17	ML Road Tire	926
18	Bike Wash - Dissolver	908
19	LL Mountain Tire	862
20	HL Road Tire	858
21	Hydration Pack - 70 oz.	733
22	Mountain-200 Black, 46	620
23	Mountain-200 Black, 42	614
24	Mountain-200 Silver, 38	596
25	Mountain-200 Black, 38	582

Ardından DMV kullanılarak bu indeksin ne kadar kullanıldığı analiz edilmiştir:

SQLQuery1.sql - I...41BRDM\erdem (51))*

```

SELECT
    i.name AS IndexName,
    i.type_desc AS IndexType,
    c.name AS ColumnName,
    ic.is_included_column,
    i.is_disabled,
    i.fill_factor
FROM
    sys.indexes i
JOIN
    sys.index_columns ic ON i.object_id = ic.object_id AND i.index_id = ic.index_id
JOIN
    sys.columns c ON ic.object_id = c.object_id AND ic.column_id = c.column_id
WHERE
    i.object_id = OBJECT_ID('FactInternetSales');

```

100 %

Results Messages

	IndexName	IndexType	ColumnName	is_included_column	is_disable
1	PK_FactInternetSales_SalesOrderNumber_SalesOrder...	CLUSTERED	SalesOrderNumber	0	0
2	PK_FactInternetSales_SalesOrderNumber_SalesOrder...	CLUSTERED	SalesOrderLineNumber	0	0

Sonuç: Kullanım oranı çok düşük (TotalUsage = 2), yani indeks sorgularda yeterince etkin kullanılmamaktadır.

3.3 Sorgu Optimizasyonu ve Execution Plan -> Ağır bir sorgu seçilerek execution plan çıkarılmıştır:

SQLQuery1.sql - I...41BRDM\erdem (51))*

```

SELECT
    OBJECT_NAME(s.[object_id]) AS TableName,
    i.name AS IndexName,
    i.type_desc AS IndexType,
    s.user_seeks + s.user_scans + s.user_lookups + s.user_updates AS TotalUsage,
    s.[object_id]
FROM
    sys.dm_db_index_usage_stats s
JOIN
    sys.indexes i ON s.[object_id] = i.[object_id] AND s.index_id = i.index_id
WHERE
    OBJECT_NAME(s.[object_id]) = 'FactInternetSales'
ORDER BY
    TotalUsage DESC;

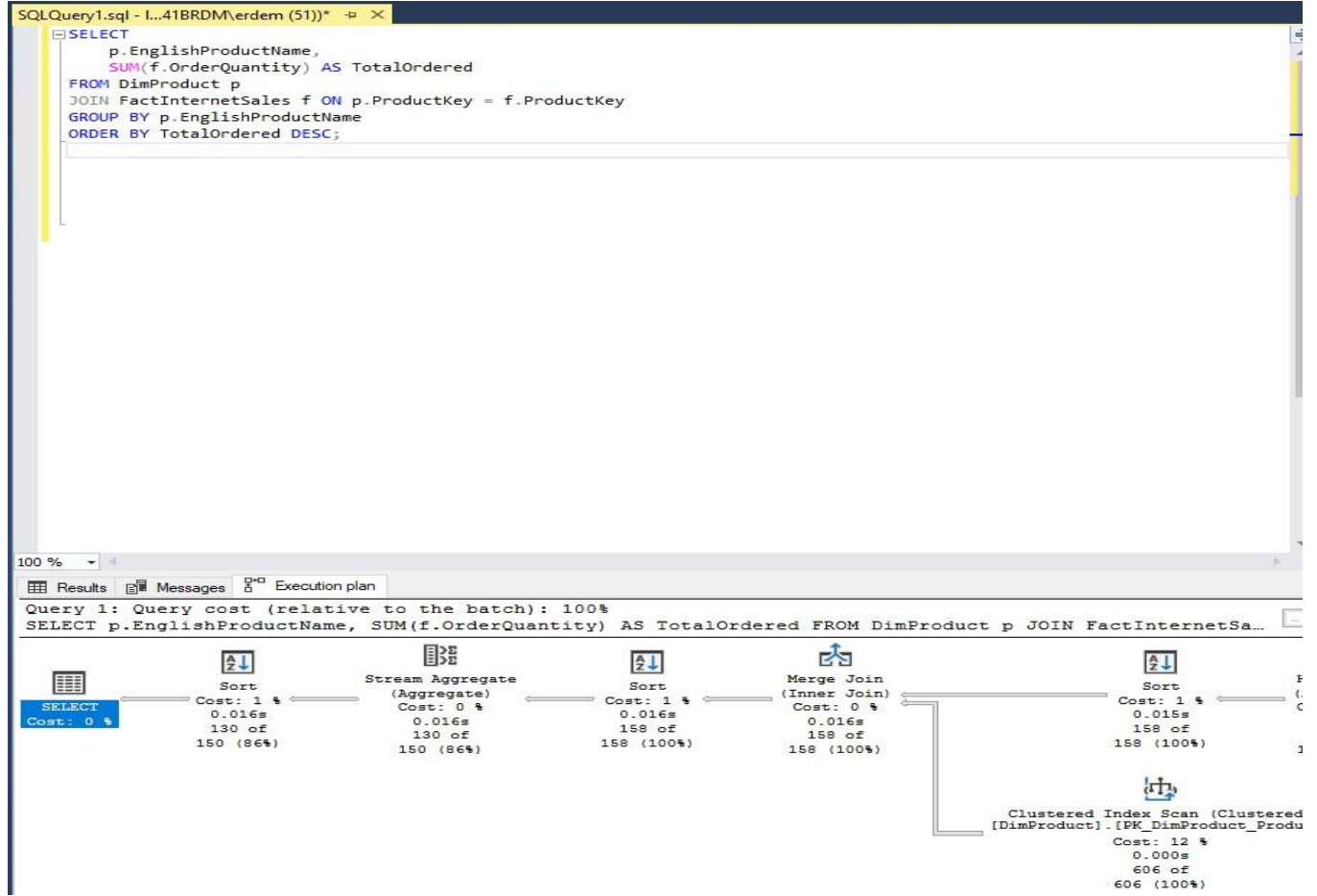
```

100 %

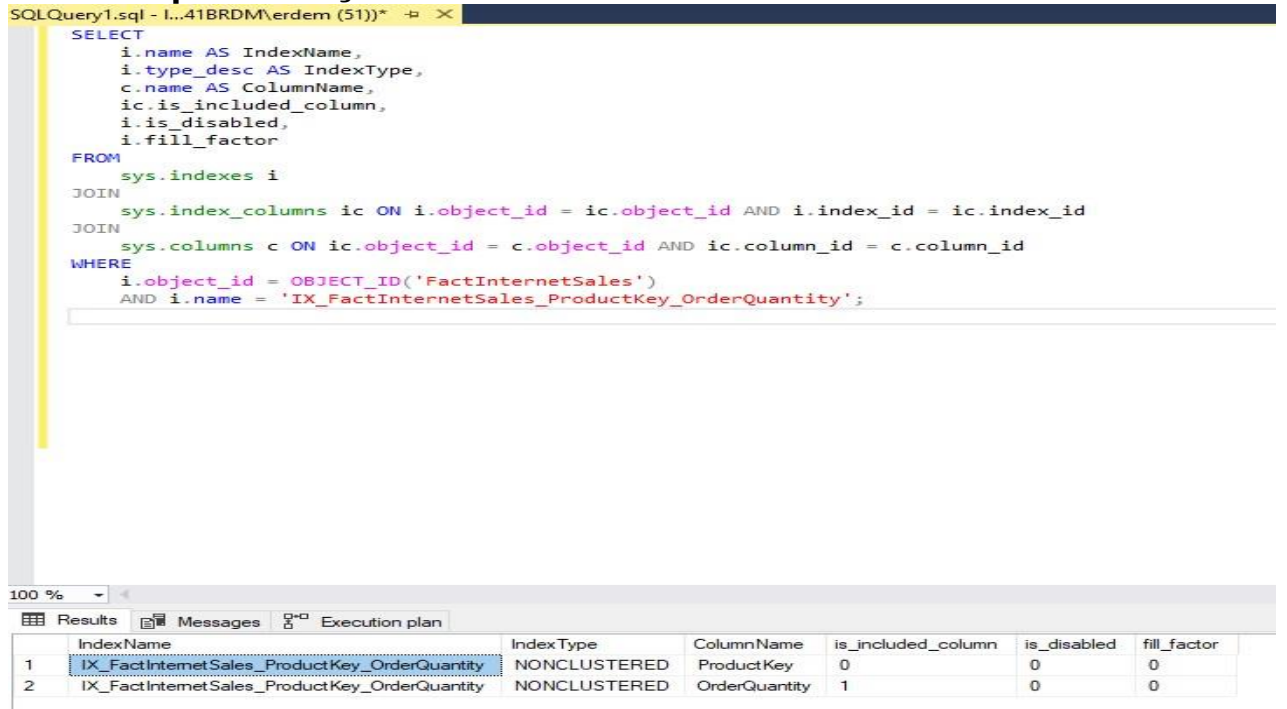
Results Messages

	TableName	IndexName	IndexType	TotalUsage	database_id	object_
1	FactInternetSales	PK_FactInternetSales_SalesOrderNumber_SalesOrder...	CLUSTERED	2	5	13175

Execution Plan Görseli:



Plan üzerinde `Clustered Index Scan` kullanıldığı görülmüştür. Bu durumu iyileştirmek için sistemde zaten var olan bir non-clustered index tespit edilmiştir:



Bu index 'ProductKey' üzerine tanımlanmış olup 'OrderQuantity' kolonunu include etmektedir.

3.4 Kullanıcı ve Rol Tanımlama (Erişim Kontrol):

```
SQLQuery1.sql - I...41BRDM\erdem (51))* -> X  
CREATE LOGIN ProjeKullanicisi WITH PASSWORD = 'GucluSifre123!';
```

```
SQLQuery1.sql - I...41BRDM\erdem (51))* -> X  
USE AdventureWorksDW2019;  
GO  
CREATE USER ProjeKullanicisi FOR LOGIN ProjeKullanicisi;
```

```
SQLQuery1.sql - I...41BRDM\erdem (51))* -> X  
EXEC sp_addrolemember 'db_datareader', 'ProjeKullanicisi';
```

Kullanıcıya sadece SELECT yetkisi verilmiştir. Veri değişikliği yetkisi yoktur.

- 4. Sonuç:** Bu proje kapsamında veritabanı performans izleme, sorgu optimizasyonu ve erişim kontrolü adımları başarıyla tamamlandı. Gerçek zamanlı izleme, indeks kullanım analizi ve execution plan ile sorgu iyileştirme yapılmış; ayrıca yetkilendirme ile veritabanının güvenli erişimi sağlanmıştır.

Github.com/dorukku

Doruk Ulaş Akbıyık

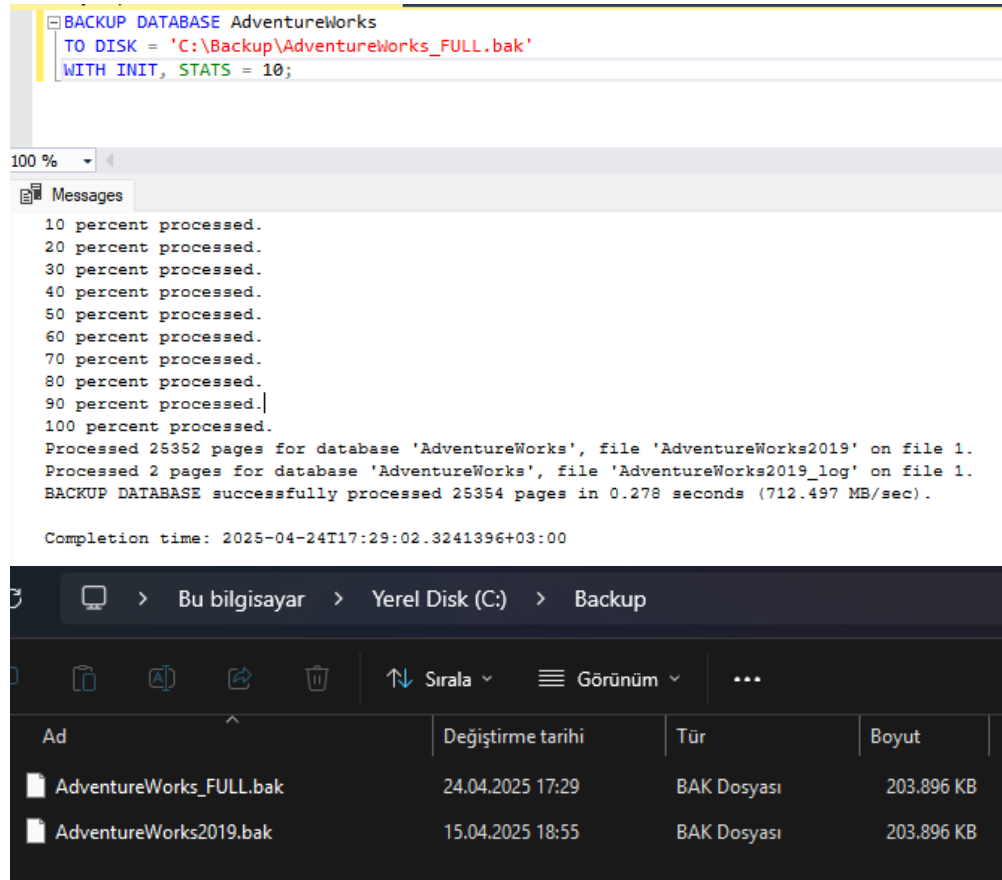
Proje 7: Veritabanı Yedekleme ve Otomasyon

1. Projenin Amacı

Bu projede amaç, bir SQL Server veritabanının manuel olarak yedeklenmesini ve bu yedekleme işlemi sonrası PowerShell kullanarak otomatik bir log (kayıt) sisteminin geliştirilmesini sağlamaktır.

2. Manuel Yedek Alma

İlk olarak AdventureWorks2019 veritabanının tam (full) yedeğini aldım. Yedek dosyasının uzantısı .bak olup, sabit diskte C : \Backup klasörüne kaydedildi.



3. PowerShell ile Loglama

Yedekleme işleminden sonra, bu işlemi günlüklere yazan basit bir PowerShell scripti oluşturdum.

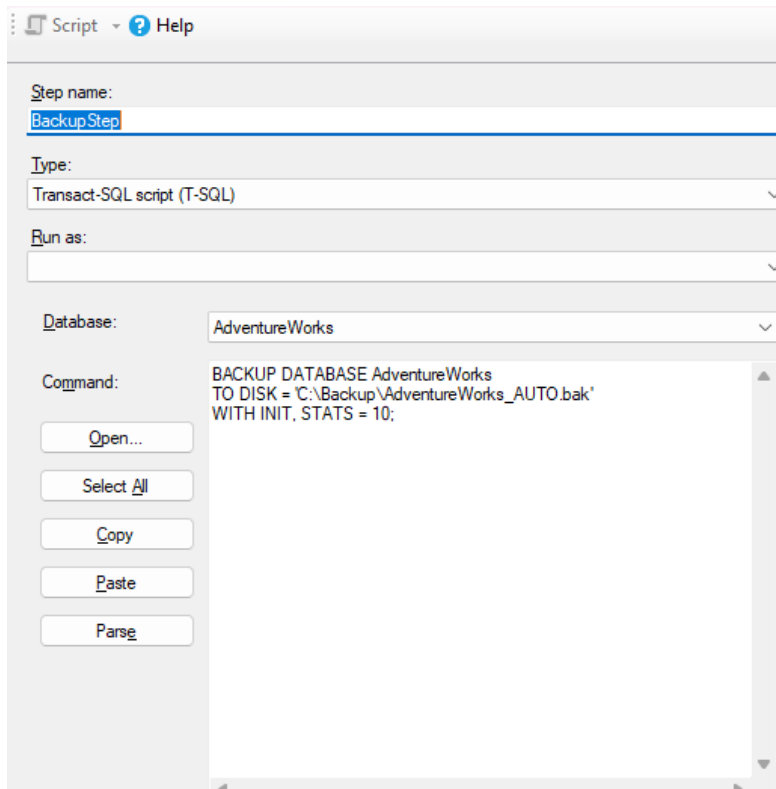
Bu script sayesinde her yedekleme işleminden sonra bir .txt dosyasına “Yedekleme alındı” tarzında bir kayıt düşülüyor.

```
$datetime = Get-Date -Format "yyyy-MM-dd HH:mm:ss"  
$logText = "$datetime - AdventureWorks veritabanı yedeklendi."  
Add-Content -Path "C:\Users\dua61\Desktop\mssql\veritabanı yedekleme ve otomasyon\backup_log.txt" -Value $logText
```

4. SQL Server Agent ile Job Oluşturma

Yedekleme işlemi otomatik hale getirmek için SQL Server Agent üzerinden bir **Job** oluşturdum.

Bu job hem SQL komutuyla veritabanı yedeği alıyor, hem de PowerShell scriptini çalıştırarak log dosyasına yazıyor.



Step name:
LogStep

Type:
Operating system (CmdExec)

Run as:
SQL Server Agent Service Account

Process exit code of a successful command: 0

Command:
powershell.exe -ExecutionPolicy Bypass -File "C:\Backup\auto_backup_log

Open...
Select All
Copy
Paste

5. Zamanlama (Schedule) Oluşturma

SQL Server Agent içinde bu job'a günlük olarak saat 03:00'te çalışacak şekilde bir zamanlama ekledim.

Bu sayede sistem, belirlenen saatte hem yedek alıyor hem de log yazıyor.

6. Log Dosyası Oluşumu

Job çalıştırıldığında backup_log.txt dosyasına şu şekilde bir satır ekleniyor:

2025-04-24 03:00:00 - AdventureWorks veritabanı yedeklendi.

Bu log dosyası yedeklerin alınıp alınmadığını takip etmek için kullanılabilir.

ETL Süreçleri ve Veri Temizleme Raporu

1. VERİ TEMİZLEME (Data Cleaning)

Eksik ve Boş Veri Temizleme

```
DELETE FROM tablo WHERE kolon1 IS NULL OR kolon2 = '';

WITH CTE AS (
    SELECT *, ROW_NUMBER() OVER(PARTITION BY kolon1, kolon2 ORDER BY id) as rn
    FROM tablo
)
DELETE FROM CTE WHERE rn > 1;

UPDATE tablo SET telefon = REPLACE(REPLACE(telefon, '-', ''), ' ', '');
UPDATE tablo SET email = LOWER(TRIM(email)) WHERE email LIKE '%@%';
```

- Veritabanındaki NULL değerli kayıtları siler
- Boş string (") olan kayıtları da temizler
- Veri kalitesini artırır ve analiz hatalarını önler

Duplikat Kayıt Silme

Kod:

```
WITH CTE AS (
    SELECT *, ROW_NUMBER() OVER(PARTITION BY kolon1, kolon2 ORDER BY id) as rn
    FROM tablo
)
DELETE FROM CTE WHERE rn > 1;
```

- Aynı kolon1 ve kolon2 değerlerine sahip kayıtları bulur
- ROW_NUMBER() fonksiyonu ile sıralama yapar
- İlk kaydı bırakıp diğer duplikatları siler

Telefon Numarası Standardizasyonu

Kod: UPDATE tablo SET telefon = REPLACE(REPLACE(telefon, '-', ''), ' ', '');

- Telefon numaralarındaki tire (-) karakterlerini kaldırır
- Boşluk karakterlerini temizler
- Tek tip format oluşturur (sadece sayılar)

Email Standardizasyonu

Kod: UPDATE tablo SET email = LOWER(TRIM(email)) WHERE email LIKE '%@%';

- Email adreslerini küçük harfe çevirir
- Başındaki ve sonundaki boşlukları kaldırır
- Sadece @ işareti içeren kayıtları işler
-

2. VERİ DÖNÜŞTÜRME (Data Transformation)

```
UPDATE tablo SET
  ad = UPPER(TRIM(ad)),
  tarih = FORMAT(CAST(tarih AS DATE), 'yyyy-MM-dd');

UPDATE tablo SET kategori = CASE
  WHEN fiyat < 100 THEN 'DÜŞÜK'
  WHEN fiyat BETWEEN 100 AND 500 THEN 'ORTA'
  ELSE 'YÜKSEK'
END;
```

Metin ve Tarih Standardizasyonu

Kod:

```
UPDATE tablo SET
  ad = UPPER(TRIM(ad)),
  tarih = FORMAT(CAST(tarih AS DATE), 'yyyy-MM-dd');
```

- Ad sütunundaki metinleri büyük harfe çevirir
- Boşlukları temizler
- Tarihleri standart YYYY-MM-DD formatına dönüştürür

Kategori Dönüştürme

Kod:

```
UPDATE tablo SET kategori = CASE
  WHEN fiyat < 100 THEN 'DÜŞÜK'
  WHEN fiyat BETWEEN 100 AND 500 THEN 'ORTA'
  ELSE 'YÜKSEK'
END;
```

- Fiyat değerlerine göre kategori oluşturur
- 100'den küçük: DÜŞÜK
- 100-500 arası: ORTA
- 500'den büyük: YÜKSEK

- Sayısal veriyi kategorik veriye dönüştürür

3. VERİ YÜKLEME (Data Loading)

```
INSERT INTO hedef_tablo
SELECT * FROM staging_tablo WHERE durum = 'TEMİZ';

MERGE hedef_tablo AS h
USING staging_tablo AS s ON h.id = s.id
WHEN MATCHED THEN UPDATE SET h.kolon1 = s.kolon1
WHEN NOT MATCHED THEN INSERT VALUES (s.id, s.kolon1);
```

Basit Veri Yükleme

Kod: INSERT INTO hedef_tablo SELECT * FROM staging_tablo WHERE durum = 'TEMİZ';

- Staging tablosundan temiz verileri seçer
- Hedef tabloya aktarır
- Sadece "TEMİZ" durumundaki kayıtları yükler

MERGE ile UPSERT İşlemi

Kod:

```
MERGE hedef_tablo AS h
USING staging_tablo AS s ON h.id = s.id
WHEN MATCHED THEN UPDATE SET h.kolon1 = s.kolon1
WHEN NOT MATCHED THEN INSERT VALUES (s.id, s.kolon1);
```

- Varolan kayıtları günceller (UPDATE)
- Yeni kayıtları ekler (INSERT)
- ID'ye göre eşleştirme yapar
- Tek sorguda hem ekleme hem güncelleme işlemi

4. VERİ KALİTESİ RAPORU (Data Quality Report)

```
SELECT
    'Toplam Kayıt' as metrik, COUNT(*) as deger FROM tablo
UNION ALL
SELECT 'Eksik Veri', COUNT(*) FROM tablo WHERE kolon1 IS NULL
UNION ALL
SELECT 'Duplikat', COUNT(*) - COUNT(DISTINCT kolon1) FROM tablo;

SELECT kolon1, COUNT(*) as duplikat_sayisi
FROM tablo GROUP BY kolon1 HAVING COUNT(*) > 1;
```

Genel Veri Kalitesi Özeti

Kod:

```
SELECT
    'Toplam Kayıt' as metrik, COUNT(*) as deger FROM tablo
UNION ALL
SELECT 'Eksik Veri', COUNT(*) FROM tablo WHERE kolon1 IS NULL
UNION ALL
SELECT 'Duplikat', COUNT(*) - COUNT(DISTINCT kolon1) FROM tablo;
```

- Toplam kayıt sayısını gösterir
- Eksik veri sayısını hesaplar
- Duplikat kayıt sayısını bulur
- Tek raporda tüm kalite metriklerini birleştirir

Duplikat Detay Raporu

Kod: `SELECT kolon1, COUNT(*) as duplikat_sayisi FROM tablo GROUP BY kolon1 HAVING COUNT(*) > 1;`

- Hangi değerlerin duplikat olduğunu gösterir
- Her duplikat değerin kaç kez tekrarlandığını sayar
- HAVING ile sadece 1'den fazla olan kayıtları filtreler

ETL Sürecinin Faydaları

Veri Kalitesi Artışı

- Eksik ve hatalı veriler temizlenir
- Standart formatlar oluşturulur
- Duplikatlar kaldırılır

Analiz Güvenilirliği

- Temiz veri ile doğru analizler yapılır
- Raporlama hataları azalır
- Karar verme süreçleri güvenilir hale gelir

Performans İyileştirmesi

- Gereksiz veriler kaldırılır
- İndeksleme daha etkili çalışır

- Sorgu performansı artar

Otomasyon

- Düzenli ETL süreçleri ile sürekli veri kalitesi
- Manual müdahale azalır
- Hata oranları düşer

Öneriler

1. ETL süreçlerini düzenli aralıklarla çalıştırın
2. Veri kalitesi raporlarını takip edin
3. Hata loglarını kontrol edin
4. Backup alarak güvenli çalışın
5. Test ortamında önce deneyin

Veritabanı Yükseltme ve Sürüm Yönetimi Raporu

1. VERİTABANI YÜKSELTME PLANI

Veritabanı Yedekleme

Kod: BACKUP DATABASE [eski_db] TO DISK = 'C:\backup\eski_db_backup.bak';

- Mevcut veritabanının tam yedeğini alır
- Yükseltme işlemi öncesi güvenlik önlemi
- Geri dönüş planı için kritik adım
- Disk'e .bak uzantılı dosya olarak kaydeder

Yeni Veritabanı Oluşturma

Kod: CREATE DATABASE [yeni_db];

- Yükseltilmiş sürüm için yeni veritabanı oluşturur
- Eski veritabanını etkilemez
- Paralel çalışma imkanı sağlar
- Test ortamı hazırlar

Veritabanı Geri Yükleme

Kod:

RESTORE DATABASE [yeni_db] FROM DISK = 'C:\backup\eski_db_backup.bak'

WITH REPLACE, MOVE 'eski_db' TO 'C:\data\yeni_db.mdf';

- Yedekten veriyi yeni veritabanına aktarır
- REPLACE parametresi ile üzerine yazma yapar
- MOVE ile dosya konumunu değiştirir
- Veri kaybı olmadan geçiş sağlar

Uyumluluk Seviyesi Güncelleme

Kod: ALTER DATABASE [yeni_db] SET COMPATIBILITY_LEVEL = 150;

- Veritabanını SQL Server 2019 seviyesine yükseltir
- Yeni özellikleri aktif hale getirir
- Performans iyileştirmelerini etkinleştirir

- Modern SQL özelliklerini kullanılabilir yapar

2. SÜRÜM YÖNETİMİ

Sürüm Kontrol Tablosu Oluşturma

Kod:

```
CREATE TABLE version_control (  
    version_id INT IDENTITY(1,1) PRIMARY KEY,  
    version_number VARCHAR(20),  
    release_date DATETIME,  
    description VARCHAR(500)  
);
```

- Tüm sürüm bilgilerini saklar
- Her sürümün benzersiz ID'si olur
- Tarih ve açıklama bilgilerini tutar
- Sürüm geçmişini takip eder

Sürüm Kaydı Ekleme

Kod: INSERT INTO version_control VALUES ('1.0.0', GETDATE(), 'İlk sürüm');

- Yeni sürüm bilgisini kaydeder
- Mevcut tarih otomatik eklenir
- Sürüm açıklamasını saklar
- Değişiklik geçmişi oluşturur

Güncel Sürüm Sorgulama

Kod: SELECT MAX(version_number) as current_version FROM version_control;

- En son sürüm numarasını getirir
- Hangi sürümde olduğumuzu gösterir
- Uygulama içinde sürüm kontrolü sağlar
- Otomatik sürüm takibi yapar

3. DDL TRİGGERS (Şema Değişikliği İzleme)

DDL Audit Trigger Oluşturma

Kod:

```
CREATE TRIGGER ddl_audit_trigger
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
BEGIN
    INSERT INTO ddl_log (event_type, object_name, sql_command, user_name,
event_date)
    VALUES (
        EVENTDATA().value('(/EVENT_INSTANCE/EventType)[1]', 'VARCHAR(100)'),
        EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]', 'VARCHAR(100)'),
        EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand)[1]',
'VARCHAR(MAX)'),
        USER_NAME(),
        GETDATE()
    );
END;
```

- Tüm DDL değişikliklerini izler (CREATE, ALTER, DROP)
- Hangi kullanıcının değişiklik yaptığını kaydeder
- Yapılan SQL komutunu saklar
- Değişiklik tarihini otomatik ekler
- EVENTDATA() fonksiyonu ile detay bilgi alır

DDL Log Tablosu

Kod:

```
CREATE TABLE ddl_log (
    log_id INT IDENTITY(1,1) PRIMARY KEY,
    event_type VARCHAR(100),
```



```
object_name VARCHAR(100),  
sql_command VARCHAR(MAX),  
user_name VARCHAR(100),  
event_date DATETIME  
);
```

- Şema değişikliklerini saklar
- Her değişikliğin detaylı kaydını tutar
- Güvenlik denetimi sağlar
- Değişiklik geçmişi oluşturur

4. TEST VE GERİ DÖNÜŞ PLANI

Yükseltme Test Prosedürü

Kod:

```
CREATE PROCEDURE sp_test_upgrade  
AS  
BEGIN  
    BEGIN TRANSACTION;  
  
    SELECT COUNT(*) as before_count FROM test_table;  
  
    EXEC sp_upgrade_process;  
  
    SELECT COUNT(*) as after_count FROM test_table;  
  
    IF @@ERROR <> 0  
        ROLLBACK TRANSACTION;  
    ELSE  
        COMMIT TRANSACTION;
```

END;

- Yükseltme öncesi veri sayısını kaydeder
- Yükseltme işlemini çalıştırır
- Yükseltme sonrası veri sayısını kontrol eder
- Hata durumunda geri alır (ROLLBACK)
- Başarılı ise değişiklikleri onaylar (COMMIT)

Geri Dönüş Prosedürü

Kod:

```
CREATE PROCEDURE sp_rollback_upgrade
```

```
AS
```

```
BEGIN
```

```
    DROP DATABASE [yeni_db];
```

```
    RESTORE DATABASE [eski_db] FROM DISK = 'C:\backup\eski_db_backup.bak';
```

```
END;
```

- Yeni veritabanını siler
- Eski veritabanını yedekten geri yükler
- Hızlı geri dönüş sağlar
- Veri kaybını önler

Sürecin Faydaları

Güvenli Yükseltme

- Yedekleme ile veri güvenliği
- Test ortamında deneme imkanı
- Geri dönüş planı ile risk azaltma
- Adım adım kontrollü süreç

İzlenebilirlik

- Tüm değişikliklerin kaydı
- Hangi kullanıcının ne yaptığının takibi

- Sürüm geçmişinin saklanması
- Denetim için detaylı loglar

Otomasyon

- Trigger'lar ile otomatik kayıt
- Prosedürler ile tekrarlanabilir süreç
- Hata durumunda otomatik geri alma
- Manuel müdahale gereksinimini azaltma

Performans ve Özellikler

- Yeni SQL Server özelliklerinin kullanımı
- Performans iyileştirmelerinden faydalanma
- Modern veritabanı yapısına geçiş
- Gelecekteki güncellemelere hazırlık

Öneriler

1. **Yükseltme Öncesi:** Mutlaka tam yedek alın
2. **Test Ortamı:** Önce test ortamında deneyin
3. **Zamanlama:** Düşük yoğunluklu saatlerde yapın
4. **Monitoring:** Süreç boyunca sistem performansını izleyin
5. **Dokümantasyon:** Tüm adımları kayıt altına alın
6. **Yetkilendirme:** Sadece yetkili kişiler yükseltme yapsın
7. **Geri Dönüş:** Geri dönüş planını önceden test edin

Veritabanı Yük Dengeleme ve Dağıtık Veritabanı Yapıları

1- Veritabanlarını Oluşturma

```
1 CREATE DATABASE RepTest1;  
2 CREATE DATABASE RepTest2;
```

Projede iki ayrı veritabanı kullanıldı. RepTest1 veritabanı yayıncı olarak, RepTest2 ise abone olarak tanımlandı.

2- Örnek Tablo Oluşturma

```
USE RepTest1;  
CREATE TABLE TestTable (  
    ID INT IDENTITY(1,1) PRIMARY KEY,  
    Ad NVARCHAR(50)  
);
```

Yayınlanacak veri TestTable adında bir tabloya eklendi.

3- Test Verisi Ekleme

```
INSERT INTO TestTable (Ad) VALUES ('Ulaş');  
INSERT INTO TestTable (Ad) VALUES ('Doruk');
```

Test verisi olarak iki kayıt tabloya eklendi.

4- Replikasyon

```
SELECT * FROM RepTest1.dbo.TestTable;
```

ID	Ad
1	Ulaş
2	Doruk

Bu çıktı, RepTest1 veritabanındaki TestTable tablosunda bulunan orijinal verileri göstermektedir. Bu tablo yayıncı rolündedir.

```
SELECT * FROM RepTest2.dbo.TestTable;
```

ID	Ad
1	Ulaş
2	Doruk

Bu çıktı, RepTest2 veritabanındaki TestTable tablosuna RepTest1 üzerinden replikasyon yoluyla aktarılan verileri göstermektedir. Replikasyonun başarılı çalıştığını doğrular.

RepTest1.dbo.TestTable tablosuna eklenen veriler, kısa süre içinde RepTest2.dbo.TestTable tablosunda da görüntülenmiştir.