AĞ TABANLI PARALEL DAĞITIM SİSTEMLERİ PROJE TESLİMİ

AHMET EREN ÖZCAN

21290207

GİTHUB:ahmeterenozcan

https://github.com/ahmeterenozcan

1-) SQL Server Veritabanı Yedekleme ve Otomasyon Projesi

https://github.com/ahmeterenozcan/veritabaniyedeklemeveotomasyon

https://www.youtube.com/watch?v=QEIEpCWdMwU

2-)Veritabanı Güvenliği ve Erişim Kontrolü

https://github.com/ahmeterenozcan/veritabaniguvenligiveerisimkontrolu

https://youtu.be/z9y_caFj00g

3) Veri Temizleme ve ETL Süreci Uygulaması

https://github.com/ahmeterenozcan/veritemizlemeveETL

https://youtu.be/iMeKqdyHYCU

4) Veritabanı Performans Optimizasyonu ve İzleme

https://github.com/ahmeterenozcan/VeritabaniPerformansiOptimizasyonuveIzleme

https://youtu.be/LS57KEE1zCE

5) Veritabanı Yedekleme ve Felaketten Kurtarma Planı

https://github.com/ahmeterenozcan/Veritabaniyedeklemevefelakettenkurtarma

https://youtu.be/X4DjyKH7RpY

6) Veritabanı Yük Dengeleme ve Dağıtık Veritabanı Yapıları

https://github.com/ahmeterenozcan/veritabaniyukdengelemevedagitikveritabaniyapilari

https://youtu.be/oldOSC1kykI

7)Veritabanı Yükseltme ve Sürüm Kontrolü

https://github.com/ahmeterenozcan/veritabaniyukseltmevesurumkontrolu

https://youtu.be/fiu5DMfeGS8

1-) SQL Server Veritabanı Yedekleme ve Otomasyon Projesi

Proje Adı: KitaplikDB Günlük Yedekleme

Adım 1: Veritabanı Oluşturma

Yapılan İşlem:
SSMS üzerinde KitaplikDB adlı bir veritabanı oluşturuldu. Daha sonra örnek bir Kitaplar tablosu eklendi ve örnek veriler girildi.

Adım 2: Yedekleme Klasörü Oluşturma

Yapılan İşlem:
D:\SQL_Backups\ klasörü oluşturuldu. SQL Server servis hesabının bu klasöre erişim izni verildi.

Adım 3: SQL Server Agent'ı Başlatma

Yapılan İşlem:
SQL Server Agent servisi başlatıldı. Bu sayede zamanlanmış görevler (Job'lar) çalıştırılabilir hale geldi.

Adım 4: SQL Server Agent Job Oluşturma

Yapılan İşlem:
Bir Job oluşturularak, KitaplikDB veritabanı her gün otomatik olarak yedeklenecek şekilde ayarlandı.

New Job
New Job Schedule
Name: GunlukYedekZamani
Jobs in Schedule
Schedule type: Recurring
Enabled
One-time occurrence
Date: 7.04.2025
Time: 14:09:14
Frequency
Occurs: Daily
Recurs every: 1 day(s)
Daily frequency
Occurs once at: 02:00:00
Occurs every: 1 hour(s)
Starting at: 00:00:00
Ending at: 23:59:59
Duration
Start date: 7.04.2025
End date: 7.04.2025
No end date:
Summary
Description: Occurs every day at 02:00:00. Schedule will be used starting on 7.04.2025.
OK
Cancel
Help
Solution Explorer
Search Solution Explorer (Ctrl+ş)
Solution 'Solution1' (0 projects)
Miscellaneous Files
SQLQuery1.sql
Properties
[Qry] Query1.dtq
(Identity)
(Name) Query1.dtq
Database Name KitaplikDB
Server Name desktop-g2tfbjv
Query Designer
Destination Table
Distinct Values No
GROUP BY Extensi <None>
Output All Colum No
Query Parameter L No parameters have
SQL Comment ***** Script for Selec
Top Specification Yes
(Identity)
Cell is Read Only.

New Job Step
Select a page
General
Advanced
Script
Help
Step name: GunlukYedek
Type: Transact-SQL script (T-SQL)
Run as:
Database: KitaplikDB
Command:
BACKUP DATABASE KitaplikDB
TO DISK = 'D:\SQL_Backups\KitaplikDB_Gunluk.bak'
WITH INIT, FORMAT;
Open...
Select All
Copy
Paste
Parse
Connection
Server: DESKTOP-G2TFBJV
Connection: DESKTOP-G2TFBJV\33
View connection properties
Progress
Ready
Previous
Next
OK
Cancel
Solution Explor
Search Solutio
Properties
[Qry] Query1.
(Identity)
(Name)
Database Na
Server Nam
Query Desi
Destination
Distinct Valu
GROUP BY E
Output All C
Query Paran
SQL Comm
Top Specific
(Identity)

Adım 6: Yedekleme Raporu Oluşturma

Yapılan İşlem:
Job geçmişi sorgulandı ve yedeklemenin başarılı olup olmadığı izlendi.

2-)Veritabanı Güvenliği ve Erişim Kontrolü

Adım 1 – Veritabanı ve Tablo Oluşturma

KullaniciDB isimli bir veritabanı ve içerisinde Kullanicilar adlı tablo oluşturulmuştur.
Tabloya örnek kullanıcı verileri girilmiştir.

Adım 2 – Kullanıcı Oluşturma ve Yetki Verme

Açıklama:

ornek_kullanici adlı bir SQL Server Login oluşturuldu ve sadece SELECT izni verildi.

Burada ise yetkisiz sorgulama yapılıyor

Adım 3 – SQL Injection Testi ve Korunma

Açıklama:

SQL Injection saldırısı simüle edilerek, zararlı girişler test edilmiştir. Ardından güvenli sorgulama için Stored Procedure kullanılmıştır.

Burada injectiona karşı koruma gösterilmiştir.

Adım 4 – Verilerin Maskelemesi

Açıklama:

Sifre kolonuna dinamik maskeleme uygulanarak yetkisiz kullanıcıların görememesi sağlanmıştır.

ornek_kullanici ile bağlanıp Sifre kolonunun maskelenmiş hali

Yönetici ile aynı sorguda Sifre kolonunun görünmesi

SQLQuery2.sql - DE...P-G2TFBJV\33 (61))* | SQLQuery1.sql - DE...P-G2TFBJV\33 (53))*

```sql
SELECT name, is_masked, masking_function
FROM sys.masked_columns
WHERE object_id = OBJECT_ID('Kullanicilar');
```

121 %

Results | Messages

	name	is_masked	masking_function
1	Sifre	1	default()

sys.masked_columns ile kontrol ekranı

Adım 5 – Audit: Kullanıcı İşlemlerini İzleme

Açıklama:

ornek_kullanici tarafından yapılan SELECT sorguları SQL Server Audit sistemi ile izlenmiştir.

Audit_KullaniciDB adlı audit tanımlandı

Database Audit Specification oluşturuldu

Kullanıcı sorgusundan sonra oluşan loglar

3) Veri Temizleme ve ETL Süreci Uygulaması

Adım 1: Hatalı Veri İçeren Kaynak Tabloyu Oluşturma

ETL_Projesi adında yeni bir veritabanı oluşturulmuş ve Musteriler adında bir tablo tanımlanmıştır. Bu tabloya bilinçli olarak hatalı ve eksik veriler eklenmiştir.

```sql
CREATE DATABASE ETL_Projesi;
GO

USE ETL_Projesi;
GO

CREATE TABLE Musteriler (
    MusteriID INT,
    Ad NVARCHAR(50),
    Soyad NVARCHAR(50),
    Email NVARCHAR(100),
    Telefon VARCHAR(15),
    DogumTarihi DATE
);
GO

INSERT INTO Musteriler (MusteriID, Ad, Soyad, Email, Telefon, DogumTarihi) VALUES
(1, 'Ahmet', 'Yılmaz', 'ahmetyilmaz[at]mail.com', '05431234567', '1990-05-12'), -- yanlış email
(2, NULL, 'Kaya', 'kaya@gmail.com', '0532xxxxxxx', '1985-10-30'),               -- eksik ad
(3, 'Mehmet', 'Demir', 'mehmet@gmail.com', NULL, NULL),                         -- eksik telefon
(4, 'Ayşe', 'Öztürk', 'ayse.ozturk@gmail.com', 'invalid_number', '1979-13-40'), -- bozuk tarih
(5, 'Fatma', NULL, 'fatma@gmail.com', '05001234567', '2000-01-15');             -- eksik soyad
```

```sql
SELECT * FROM Musteriler;
```

	MusteriID	Ad	Soyad	Email	Telefon	DogumTarihi
1	1	Ahmet	Yılmaz	ahmetyilmaz[at]mail.com	05431234567	1990-05-12
2	2	NULL	Kaya	kaya@gmail.com	0532xxxxxxx	1985-10-30
3	3	Mehmet	Demir	mehmet@gmail.com	NULL	NULL
4	4	Ayşe	Öztürk	ayse.ozturk@gmail.com	invalid_number	NULL
5	5	Fatma	NULL	fatma@gmail.com	05001234567	2000-01-15

Adım 2: Veri Temizleme İşlemleri

Eksik veya hatalı veriler tespit edilip düzeltildi. Örneğin:

- **NULL alanlar Bilinmiyor, 00000000000, 2000-01-01 gibi değerlerle güncellendi**
- **E-posta formatları düzeltildi ([at] → @)**
- **Bozuk telefon numaraları geçerli hale getirildi**

SQLQuery1.sql - DE...P-G2TFBJV\33 (57))*

```sql
UPDATE Musteriler SET Ad = 'Bilinmiyor' WHERE Ad IS NULL;
UPDATE Musteriler SET Soyad = 'Bilinmiyor' WHERE Soyad IS NULL;
UPDATE Musteriler SET Telefon = '00000000000' WHERE Telefon IS NULL;
UPDATE Musteriler SET DogumTarihi = '2000-01-01' WHERE DogumTarihi IS NULL;


UPDATE Musteriler
SET Email = REPLACE(Email, '[at]', '@')
WHERE Email LIKE '%[at]%';

UPDATE Musteriler
SET Telefon = '05555555555'
WHERE ISNUMERIC(Telefon) = 0 OR LEN(Telefon) < 11;
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(2 rows affected)

(5 rows affected)

(2 rows affected)

Completion time: 2025-04-18T20:37:24.3576563+03:00
```

121 %

Solution Explorer

Search Solution Explorer (Ctrl+ş)

Solution 'Solution1' (0 projects)

Miscellaneous Files

SQLQuery1.sql

Properties

Current connection parameters

Aggregate Status	
Connection failures	
Elapsed time	00:00:00.0292158
Finish time	18.04.2025 20:37:24
Name	DESKTOP-G2TFBJV
Rows returned	0
Start time	18.04.2025 20:37:24
State	Open
Connection	
Connection name	DESKTOP-G2TFBJV (
Connection Details	
Connection elapsed	00:00:00.0292158
Connection encrypt	Not encrypted

Name

SQLQuery1.sql - DE...P-G2TFBJV\33 (57))*

```sql
SELECT * FROM Musteriler;
```

121 %

Results | Messages

	MusteriID	Ad	Soyad	Email	Telefon	DogumTarihi
1	1	Ahmet	Yılmaz	ahmetyilmaz@mail.com	05431234567	1990-05-12
2	2	Bilinmiyor	Kaya	kaya@gmail.com	05555555555	1985-10-30
3	3	Mehmet	Demir	mehmet@gmail.com	00000000000	2000-01-01
4	4	Ayşe	Öztürk	ayse.ozturk@gmail.com	05555555555	2000-01-01
5	5	Fatma	Bilinmiyor	fatma@gmail.com	05001234567	2000-01-15

Adım 3: ETL - Temiz Veriyi Yeni Tabloya Aktarma (Load)

Temizlenmiş veriler Musteriler_Temiz adlı yeni tabloya aktarılmıştır. Bu işlem ETL sürecinin "Load" adımını temsil eder.

SQLQuery1.sql - DE...P-G2TFBJV\33 (57))*

```sql
CREATE TABLE Musteriler_Temiz (
    MusteriID INT,
    Ad NVARCHAR(50),
    Soyad NVARCHAR(50),
    Email NVARCHAR(100),
    Telefon VARCHAR(15),
    DogumTarihi DATE
);
```

121 %

Messages

```
Commands completed successfully.

Completion time: 2025-04-18T20:39:04.0196032+03:00
```

Burada yeni bir tablo oluşturuldu.

QLQuery1.sql - DE...P-G2TFBJV\33 (57))*

```sql
INSERT INTO Musteriler_Temiz (MusteriID, Ad, Soyad, Email, Telefon, DogumTarihi)
SELECT MusteriID, Ad, Soyad, Email, Telefon, DogumTarihi
FROM Musteriler;
```

21 %

Messages

```
(5 rows affected)

Completion time: 2025-04-18T20:39:29.6348608+03:00
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (57) | ETL_Projesi | 00:00:00 | 0 r

Temiz veriyi bu yeni tablomuza aktardık.

	MusterilD	Ad	Soyad	Email	Telefon	DogumTarihi
1	1	Ahmet	Yılmaz	ahmetyilmaz@mail.com	05431234567	1990-05-12
2	2	Bilinmiyor	Kaya	kaya@gmail.com	05555555555	1985-10-30
3	3	Mehmet	Demir	mehmet@gmail.com	00000000000	2000-01-01
4	4	Ayşe	Öztürk	ayse.ozturk@gmail.com	05555555555	2000-01-01
5	5	Fatma	Bilinmiyor	fatma@gmail.com	05001234567	2000-01-15

Bu da yeni tablomuz.

4) Veritabanı Performans Optimizasyonu ve İzleme

Adım 1 – Veritabanı ve Örnek Veri Oluşturma

SQLQuery1.sql - DE...P-G2TFBJV\33 (58))*

```sql
USE PerformansDB;
GO

CREATE TABLE Siparisler (
    SiparisID INT IDENTITY(1,1) PRIMARY KEY,
    MusteriAdi NVARCHAR(100),
    UrunAdi NVARCHAR(100),
    SiparisTarihi DATE,
    Miktar INT,
    Fiyat MONEY
);
```

121 %

Messages

```
Commands completed successfully.

Completion time: 2025-04-25T21:00:01.9532127+03:00
```

Performans testleri için kullanılacak Siparisler adında örnek bir tablo oluşturulmuştur. Bu tabloda müşteri adı, ürün adı, sipariş tarihi, miktar ve fiyat gibi alanlar yer almakta olup, her siparişin benzersiz bir kimliği olması için SiparisID alanı IDENTITY ile tanımlanmıştır.

SQLQuery1.sql - DE...P-G2TFBJV\33 (58))*

```sql
SET NOCOUNT ON;
DECLARE @i INT = 0;

WHILE @i < 1000000
BEGIN
    INSERT INTO Siparisler (MusteriAdi, UrunAdi, SiparisTarihi, Miktar, Fiyat)
    VALUES (
        'Müşteri_' + CAST(@i AS NVARCHAR),
        'Ürün_' + CAST(@i % 100 AS NVARCHAR),
        DATEADD(DAY, -(@i % 365), GETDATE()),
        ABS(CHECKSUM(NEWID())) % 10 + 1,
        ABS(CHECKSUM(NEWID())) % 500 + 50
    );
    SET @i += 1;
END;
```

121 %

Messages

```
Commands completed successfully.

Completion time: 2025-04-25T21:02:45.1658291+03:00
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (58) | PerformansDB | 00:01:33 | 0 rows

Performans testlerinin gerçekçi olması adına Siparisler tablosuna 1 milyon sahte veri girilmiştir. WHILE döngüsü ile rastgele müşteri ve ürün isimleri, geçmişe dönük rastgele tarihler ve miktar/fiyat kombinasyonları üretilmiştir. Bu işlem SQL Server üzerinde yüksek veri yoğunluğu oluşturarak ileride yapılacak performans analizleri için temel hazırlamıştır.

Adım 2: Ağır Sorgu ile Performans Yükü Oluşturma

	MusteriAdi	ToplamSiparis	ToplamMiktar	ToplamTutar
1	Müşteri_129350	1	7	549,00
2	Müşteri_102111	1	1	549,00
3	Müşteri_670371	1	10	549,00
4	Müşteri_169837	1	5	549,00
5	Müşteri_864611	1	9	549,00
6	Müşteri_85004	1	10	549,00
7	Müşteri_979962	1	1	549,00
8	Müşteri_137550	1	5	549,00
9	Müşteri_236452	1	3	549,00
10	Müşteri_839599	1	6	549,00
11	Müşteri_528808	1	10	549,00
12	Müşteri_907737	1	5	549,00
13	Müşteri_267325	1	6	549,00

Veritabanı performansını analiz edebilmek ve SQL Profiler, Activity Monitor gibi araçlarda gözlem yapabilmek için sistem üzerinde yük oluşturacak kasıtlı bir ağır sorgu çalıştırılmıştır. Bu sorgu 1 milyon satırlık Siparisler tablosu üzerinde GROUP BY ve ORDER BY işlemleri ile kaynak kullanımını artırmıştır.

SQL Server'ın yerleşik aracı olan Activity Monitor kullanılarak, çalıştırılan ağır sorgunun sistem kaynakları üzerindeki etkisi gözlemlenmiştir.

Öne çıkan metrikler:

- **CPU Süresi: 2809 ms**
- **Elapsed Time (Geçen Süre): 2036 ms**
- **Memory Used (KB): 66560**
- **Allocated Memory (KB): 568712**

Bu ekran, sistem üzerinde en çok kaynak tüketen sorguların görüntülenmesini sağlar ve performans darboğazlarını tespit etmekte önemli bir rol oynar.

SQLQuery1.sql - DE...P-G2TFBJV\33 (58))* | DESKTOP-G2TFBJV - Activity Monitor

```sql
SELECT TOP 10
    qs.total_elapsed_time / qs.execution_count AS AvgElapsedTime,
    qs.execution_count,
    qs.total_elapsed_time,
    SUBSTRING(qt.text, qs.statement_start_offset / 2,
              (CASE WHEN qs.statement_end_offset = -1
                    THEN LEN(CONVERT(NVARCHAR(MAX), qt.text)) * 2
                    ELSE qs.statement_end_offset END - qs.statement_start_offset) / 2) AS QueryT
    qt.dbid,
    qt.objectid
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
ORDER BY AvgElapsedTime DESC;
```

121 %

Results | Messages

	AvgElapsedTime	execution_count	total_elapsed_time	QueryText	dbid	objectid
1	32207158	5	161035792	select * from OpenRowset(TABLE TRCDATA, @traceid, @...	32767	-508658945
2	3980517	3	11941551	SELECT MusteriAdi, COUNT(*) AS ToplamSiparis, ...	NULL	NULL
3	3894587	1	3894587	SELECT MusteriAdi, COUNT(*) AS ToplamSiparis, ...	NULL	NULL
4	3816577	3	11449732	SELECT MusteriAdi, COUNT(*) AS SiparisSayisi, ...	NULL	NULL
5	204271	1	204271	SELECT db_id() as database_id, sm.[is_inlineable]...	NULL	NULL
6	165963	3	497890	SELECT TOP 10 MusteriAdi, COUNT(*) AS SiparisSa...	NULL	NULL
7	157568	1	157568	SELECT TOP 10 MusteriAdi, COUNT(*) AS SiparisSa...	NULL	NULL
8	101213	1	101213	)SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], s...	NULL	NULL
9	63421	1	63421	)SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], s...	NULL	NULL
10	62569	1	62569	)SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], s...	NULL	NULL

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (58) | PerformansDB | 00:00:00 | 10 rows

Ln 14 Col 1 Ch 1 INS

SQL Server'daki sys.dm_exec_query_stats DMV'si kullanılarak, geçmişte çalıştırılan sorgular arasından en uzun sürede tamamlananlar (AvgElapsedTime) belirlenmiştir. Bu analiz, hangi sorguların optimize edilmesi gerektiğine karar vermede önemli rol oynar.

SQL Server Profiler kullanılarak sistem üzerindeki sorguların gerçek zamanlı takibi yapılmıştır. Profiler sayesinde her bir sorgunun:

- **TextData (komut içeriği)**
- **CPU kullanımı**
- **Reads (veri okuma miktarı)**
- **Süre (Duration)**

gibi detaylı bilgileri toplanmıştır.

Gözlemler:

- **CPU: 2000+**
- **Reads: 9576**
- **Duration: 3811 ms**

Adım 3: Sorgu Performansını Artırmak İçin İndeks Kullanımı

Veri tablosunda en çok filtrelenen ve gruplanan sütunlara uygun bir şekilde indeks tanımlanmıştır.
MusteriAdi ve SiparisTarihi alanlarına oluşturulan bileşik indeks ile sorguların erişim ve filtreleme süresi azaltılmıştır.

İndeks oluşturulduktan sonra aynı sorgu tekrar çalıştırılmış ve Activity Monitor aracılığıyla sistem üzerindeki etkisi gözlemlenmiştir.
Aşağıdaki karşılaştırmada indeks öncesi ve sonrası durum kıyaslanmıştır:

Metrik	İndeks Öncesi	İndeks Sonrası
CPU (ms)	2809	2811
Elapsed Time (ms)	2036	3286
Used Memory (KB)	66560	49408
Logical Reads	0	48

- **CPU süresi benzer kalmış olsa da,**
- **Bellek kullanımı düşmüş ve**
- **Okuma işlemleri (logical reads) artmıştır → bu da indeksin devreye girdiğini gösterir.**

Adım 4: Optimize Edilmiş Sorgu ile Veri Çekme

Veritabanı performansını artırmak amacıyla optimize edilmiş sorgular kullanılmıştır. Bu örnekte:

- **TOP 10 ifadesiyle sadece en çok tutar yapan ilk 10 müşteri çekilmiştir.**
- **WHERE filtresi doğrudan tarih karşılaştırmasıyla yazılmıştır (fonksiyon kullanılmadan).**
- **GROUP BY ve ORDER BY kombinasyonu ile analiz yapılmıştır.**

Adım 5: Veri Yöneticisi Rolleri

Veritabanı güvenliğini sağlamak amacıyla iki farklı kullanıcı tanımlanmıştır:

- **Kullanici_Oku: Sadece sorgu (SELECT) yetkisine sahip olacak kullanıcı**
- **Kullanici_Yonetici: Sorgu çalıştırma yanında INSERT, UPDATE, DELETE gibi işlemleri de yapabilecek tam yetkili kullanıcı**

İlk adım olarak SQL Server Authentication ile giriş bilgileri oluşturulmuş, ardından bu log-in'ler PerformansDB veritabanında kullanıcı olarak tanımlanmıştır.

Veritabanı güvenliğini güçlendirmek ve kullanıcıların sadece ihtiyaç duyduğu verilere erişmesini sağlamak için rol tabanlı erişim kontrolü uygulanmıştır.

➤ Oluşturulan Roller:

- **RaporGoruntuleyici: Sadece SELECT yetkisi olan kullanıcılar (okuma yetkisi)**
- **Yonetici: SELECT, INSERT, UPDATE, DELETE gibi tüm yetkilere sahip olan yöneticiler**

Bu görselde hem Users, hem Roles, hem de Logins bölümleri açıkça görülüyor, bu da rol bazlı kullanıcı yönetiminin eksiksiz yapıldığını gösteriyor.

5) Veritabanı Yedekleme ve Felaketten Kurtarma Planı

Adım 0 – Test Veritabanı ve Örnek Veri Oluşturma

- **CREATE DATABASE DisasterDB; ile felaket senaryosunu simüle edeceğimiz “DisasterDB” adlı boş veritabanını oluşturduk.**
- **USE DisasterDB; ile komutların bu veritabanı üzerinde çalışmasını sağladık.**
- **CREATE TABLE Urunler (...) ile içinde UrunID, UrunAdi ve Fiyat sütunları olan bir tablo tanımladık.**
- **INSERT INTO Urunler (...) VALUES (...) sorgusuyla tablomuza üç satır (“Kalem”, “Defter”, “Silgi”) ekleyerek test verisini yerleştirdik.**

Adım 1 – Full Yedek Oluşturma

BACKUP DATABASE DisasterDB TO DISK = N'D:\SQL_Backups\DisasterDB_Full.bak' WITH FORMAT, INIT, NAME = N'DisasterDB-FullBackup';
komutuyla "DisasterDB" veritabanının tam yedeğini aldık.

- **FORMAT, INIT: Varolan yedek dosyasını silip yenisini oluşturur.**
- **NAME: Yedeğe açıklayıcı bir etiket atar.**

Adım 2 – Transaction Log Yedeği İçin Veri Değişikliği

Transaction-log yedeğini almadan hemen önce veritabanında değişiklik yapıyoruz ki log backup'ı aldığımızda bu eklemenin de log'a işlenmiş olduğunu görebilelim. Felaket kurtarma senaryosunda log yedeğini en güncel değişiklikleri içerecek şekilde almamız gerekiyor.

Adım 3 – Differential Yedek Oluşturma

Bu komutla, son full yedeğin alındığı andan sonra yapılan değişiklikleri içeren "differential" yedeği aldık. Böylece her seferinde tüm veriyi tekrar yedeklemek yerine, sadece değişen sayfalar yedeklenir ve kurtarma süreci hızlanır.

Adım 4 – Transaction Log (Log) Yedeği Oluşturma

- **USE DisasterDB; ile çalıştığımız veritabanını seçtik.**
- **BACKUP LOG komutuyla, tam ve differential yedekler arasındaki tüm işlem kayıtlarını (transaction log) bir .trn dosyasına kaydettik.**
- **NAME = N'DisasterDB-LogBackup' parametresiyle yedeğe bir etiket verdik.**

Adım 5 – Point-in-Time (Zamana Özel) Kurtarma Senaryosu İçin Ek Veri Üretme

Bu aşamada, daha önce tam ve fark yedekleri alındıktan ve transaction log yedeği alındıktan sonra veritabanına yeni bir kayıt ekleyerek "zamanına özel" kurtarmayı test etmek için bir değişiklik üretiyoruz.

- **Tam ve fark yedeklerini elde ettikten sonra sistem üzerinde yeni bir işlem (INSERT) yaptığımızı göstermek, transaction log'daki bu değişikliğin "up to a point" (belirli bir noktaya kadar) kurtarma senaryosunda nasıl atlanabileceğini test etmemize yarar.**
- **Bir sonraki adımda, RESTORE LOG ... WITH STOPAT = '<süre>'; komutuyla "Silgi" satırının eklenmediği bir zamana geri dönebileceğimizi kanıtlamış olacağız.**

Silgi satırının eklendiğini gösteren sorgu

Adım 6 – Point-in-Time Kurtarma (Silgi Satırını Geri Atlama)

SQLQuery1.sql - DE...P-G2TFBJV\33 (60))*

```sql
-- 1. Tam yedeği restore et (NORECOVERY)
RESTORE DATABASE DisasterDB_Restore
FROM DISK = N'D:\SQL_Backups\DisasterDB_Full.bak'
WITH
    NORECOVERY,
    MOVE 'DisasterDB' TO 'C:\SQLData\DisasterDB_Restore.mdf',
    MOVE 'DisasterDB_Log' TO 'C:\SQLData\DisasterDB_Restore.ldf';
GO

-- 2. Differential yedeği restore et (NORECOVERY)
RESTORE DATABASE DisasterDB_Restore
FROM DISK = N'D:\SQL_Backups\DisasterDB_Differential.bak'
WITH
    NORECOVERY;
GO

-- 3. Log yedeğini point-in-time ile restore et (RECOVERY)
RESTORE LOG DisasterDB_Restore
FROM DISK = N'D:\SQL_Backups\DisasterDB_Log.trn'
WITH
    STOPAT = '2025-05-02 15:30:00',
    RECOVERY;
```

121 %

Messages

```
Processed 400 pages for database 'DisasterDB_Restore', file 'DisasterDB' on file 1.
Processed 2 pages for database 'DisasterDB_Restore', file 'DisasterDB_log' on file 1.
RESTORE DATABASE successfully processed 402 pages in 0.019 seconds (165.090 MB/sec).
Processed 104 pages for database 'DisasterDB_Restore', file 'DisasterDB' on file 1.
Processed 2 pages for database 'DisasterDB_Restore', file 'DisasterDB_log' on file 1.
RESTORE DATABASE successfully processed 106 pages in 0.008 seconds (103.027 MB/sec).
Processed 0 pages for database 'DisasterDB_Restore', file 'DisasterDB' on file 1.
Processed 9 pages for database 'DisasterDB_Restore', file 'DisasterDB_log' on file 1.
Msg 4335, Level 16, State 1, Line 18
The specified STOPAT time is too early. All or part of the database is already rolled forward beyond that point.
Msg 3013, Level 16, State 1, Line 18
RESTORE LOG is terminating abnormally.
```

121 %

Query completed with errors. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (60) | DisasterDB | 00:00:00 | 0 rows

RESTORE LOG komutunun STOPAT = '2025-05-02 15:30:00' zamanını işleyemediğini ve “specified STOPAT time is too early” hatası verdiğini görüyoruz.

15:30:00 zamanından sonraki işlemler zaten işlenmiş (yani veritabanı log’u o noktaya kadar geri sarılamıyor, çünkü o noktadan önceki tüm işlem kayıtları çoktan uygulanmış).

SQLQuery1.sql - DE...P-G2TFBJV\33 (60))*

```sql
RESTORE LOG DisasterDB_Restore
FROM DISK = N'D:\SQL_Backups\DisasterDB_Log.trn'
WITH
  STOPAT = '2025-05-02 21:25:42.000',  -- HEADERONLY'dan aldığın zaman
  RECOVERY;
GO
```

121 %

Messages

```
Processed 0 pages for database 'DisasterDB_Restore', file 'DisasterDB' on file 1.
Processed 9 pages for database 'DisasterDB_Restore', file 'DisasterDB_log' on file 1.
This backup set contains records that were logged before the designated point in time. The database is being left in the restoring
RESTORE LOG successfully processed 9 pages in 0.006 seconds (11.718 MB/sec).

Completion time: 2025-05-02T21:34:48.5012676+03:00
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (60) | DisasterDB | 00:00:00 | 0 rows

Ln 7 Col 1 Ch 1 INS

- **Son eklediğim “Silgi” kaydı (UrunID = 5, fiyat 1.75) ise kurtarmayı durdurduğunuz zamandan sonra eklendiği için geri yükleme kapsamına alınmadı ve tabloda silinmiş olarak görünüyor.**

Yani point-in-time restore’ı log’daki STOPAT zamanıyla eşleştirdiğinizde, o andan sonra yapılmış ekleme/güncelleme işlemleri geri alınır. Bu sayede istenmeyen bir değişikliği “zaman içinde geri sararak” ortadan kaldırabilirsiniz.

Adım-7 Felaket Senaryosu Testi (DB Kaybından Geri Dönüş)

7.1 Orijinal Veritabanını Kaldır / Offline Yap

```sql
-- (İsteğe bağlı) Offline yap
ALTER DATABASE DisasterDB SET OFFLINE WITH ROLLBACK IMMEDIATE;
GO

-- Veya tam sil
DROP DATABASE DisasterDB;
GO
```

121 %

Messages

```
Failed to restart the current database. The current database is switched to master.

Completion time: 2025-05-02T21:38:03.9433251+03:00
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (60) | DisasterDB | 00:00:00 | 0 rows

Ln 8 Col 1 Ch 1 INS

7.2 Tam + Differential + Log Yedek Zinciriyle Geri Yükleme

SQLQuery1.sql - DE...P-G2TFBJV\33 (60))*

```sql
-- 1. Tam Yedekten Restore (NORECOVERY)
RESTORE DATABASE DisasterDB
FROM DISK = N'D:\SQL_Backups\DisasterDB_Full.bak'
WITH
    NORECOVERY,
    MOVE 'DisasterDB' TO 'C:\SQLData\DisasterDB.mdf',
    MOVE 'DisasterDB_Log' TO 'C:\SQLData\DisasterDB.ldf';
GO

-- 2. Differential Yedekten Restore (NORECOVERY)
RESTORE DATABASE DisasterDB
FROM DISK = N'D:\SQL_Backups\DisasterDB_Differential.bak'
WITH
    NORECOVERY;
GO

-- 3. Log Yedeğini Son Hal ile Restore (RECOVERY)
RESTORE LOG DisasterDB
FROM DISK = N'D:\SQL_Backups\DisasterDB_Log.trn'
WITH
    RECOVERY;
GO
```

121 %

Messages

```
Processed 400 pages for database 'DisasterDB', file 'DisasterDB' on file 1.
Processed 2 pages for database 'DisasterDB', file 'DisasterDB_log' on file 1.
RESTORE DATABASE successfully processed 402 pages in 0.019 seconds (165.090 MB/sec).
Processed 104 pages for database 'DisasterDB', file 'DisasterDB' on file 1.
Processed 2 pages for database 'DisasterDB', file 'DisasterDB_log' on file 1.
RESTORE DATABASE successfully processed 106 pages in 0.008 seconds (103.027 MB/sec).
Processed 0 pages for database 'DisasterDB', file 'DisasterDB' on file 1.
Processed 9 pages for database 'DisasterDB', file 'DisasterDB_log' on file 1.
RESTORE LOG successfully processed 9 pages in 0.004 seconds (17.578 MB/sec).
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (60) | DisasterDB | 00:00:00 | 0 rows

Ln 23 Col 1 Ch 1 INS

7.3 Tüm Veritabanı Geri Geldi

6) Veritabanı Yük Dengeleme ve Dağıtık Veritabanı Yapıları

A) Transactional Replication

1. Distributor'ı Etkinleştirme

```
-- 1.1 Distributor olarak sunucuyu kaydet ve parola ata
EXEC sp_adddistributor
  @distributor = N'(local)',
  @password    = N'DistribPassw0rd!';
-- Expected Message: "Changed Distributor to '(local)'"

-- 1.2 Distribution DB oluştur
EXEC sp_adddistributiondb
  @database    = N'distribution',
  @data_folder = N'C:\SQLData',
  @log_folder  = N'C:\SQLData',
  @security_mode = 1;
"Created distribution database 'distribution'"

-- 1.3 Publisher kaydı
EXEC sp_adddistpublisher
  @publisher       = N'(local)',
  @distribution_db = N'distribution',
  @security_mode   = 1;
"registered publisher '(local)' to distribution database 'distribution'"
```

2. Publication Oluşturma

```
USE LoadBalanceDB;
GO

-- 2.1 DB'yi yayınlamaya hazırla
EXEC sp_replicationdboption
  @dbname = N'LoadBalanceDB',
  @optname = N'publish',
  @value   = N'true';
```

"Changed database option publish on database 'LoadBalanceDB' to true"

```
-- 2.2 Publication ekle
EXEC sp_addpublication
  @publication      = N'SiparisPublication',
  @description      = N'Siparisler tablosu transactional publication',
  @sync_method      = N'native',
  @retention        = 0,
  @allow_push       = N'true',
  @allow_pull       = N'true',
  @publication_type = N'transactional';
```

"Created publication 'SiparisPublication' on database 'LoadBalanceDB'"

```
-- 2.3 Article (tablo) ekle
EXEC sp_addarticle
  @publication     = N'SiparisPublication',
  @article         = N'Siparisler',
  @source_owner    = N'dbo',
  @source_object   = N'Siparisler',
  @type            = N'logbased',
  @schemab_option  = 0x000000000803509F;
"Added article 'Siparisler' to publication 'SiparisPublication'"
```

3. Subscription Oluşturma

```
-- 3.1 Subscriber DB'yi oluştur
CREATE DATABASE LoadBalance_Sub;
GO

-- 3.2 Push abonelik ekle
EXEC sp_addsubscription
  @publication      = N'SiparisPublication',
  @subscriber       = N'(local)',
  @destination_db   = N'LoadBalance_Sub',
  @subscription_type= N'push',
  @sync_type        = N'automatic',
  @article          = N'all';
"Created subscription for publication 'SiparisPublication'"
```

```
-- 3.3 Snapshot’ı başlat
EXEC sp_startpublication_snapshot
  @publication = N'SiparisPublication';
"Snapshot agent started for publication 'SiparisPublication'"
```

B) Database Mirroring

1. Endpoint’leri Oluşturma

```
-- Ana (Principal) sunucu
CREATE ENDPOINT [Mirroring]
  STATE=STARTED
  AS TCP (LISTENER_PORT=5022)
  FOR DATABASE_MIRRORING (ROLE=PARTNER,
ENCRYPTION=DISABLED);
GO

-- Yedek (Mirror) sunucu, aynı komutlar burada da çalıştırılmalı
-- (örneğin uzak sunucuda)
CREATE ENDPOINT [Mirroring]
  STATE=STARTED
  AS TCP (LISTENER_PORT=5022)
  FOR DATABASE_MIRRORING (ROLE=PARTNER,
ENCRYPTION=DISABLED);
GO
"Command(s) completed successfully."
```

2. Veritabanını Full Yedekten Hazırlama

```
-- Principal’da full backup
BACKUP DATABASE LoadBalanceDB
  TO DISK = N'D:\SQL_Backups\LB_MirrorFull.bak'
  WITH FORMAT, INIT;
"BACKUP DATABASE successfully processed ... pages"
GO

-- Mirror’da restore (NORECOVERY)
RESTORE DATABASE LoadBalanceDB
  FROM DISK = N'D:\SQL_Backups\LB_MirrorFull.bak'
  WITH NORECOVERY,
      MOVE 'LoadBalanceDB'     TO 'C:\SQLData\LB_DB.mdf',
      MOVE 'LoadBalanceDB_log' TO 'C:\SQLData\LB_DB_log.ldf';
"RESTORE DATABASE successfully processed ... pages"
GO
```

3. Log Backup & Restore

```
-- Principal’da log backup
BACKUP LOG LoadBalanceDB
  TO DISK = N'D:\SQL_Backups\LB_MirrorLog.trn'
  WITH INIT;
"BACKUP LOG successfully processed ... pages"
GO

-- Mirror’da log restore (NORECOVERY)
RESTORE LOG LoadBalanceDB
  FROM DISK = N'D:\SQL_Backups\LB_MirrorLog.trn'
  WITH NORECOVERY;
"RESTORE LOG successfully processed ... pages"
GO
```

4. Mirroring’i Başlatma

```
-- Principal
ALTER DATABASE LoadBalanceDB
  SET PARTNER = N'TCP://MirrorServerName:5022';
"Database 'LoadBalanceDB' is now in a mirroring session"
GO

-- Mirror
ALTER DATABASE LoadBalanceDB
  SET PARTNER = N'TCP://PrincipalServerName:5022';
"Database 'LoadBalanceDB' is now in a mirroring session"
GO
```

5. Failover Testi

```
-- Principal’dan Mirror’a geçiş
ALTER DATABASE LoadBalanceDB
  SET PARTNER FAILOVER;
"The database is now the principal database"
GO


-- Yeni kontrol
SELECT
  mirroring_state_desc,
  mirroring_role_desc
FROM sys.database_mirroring
WHERE DB_NAME(database_id) = 'LoadBalanceDB';
-- Results after failover: mirroring_role_desc = PRINCIPAL on former Mirror
GO
```

7)VERİ TABANI YÜKSELTME VE SÜRÜM KONTROLÜ

Adım 1 – Pre-Upgrade Tam Yedeği Alma

- **Geri Dönüş Noktası: Veritabanımız üzerinde yapacağınız şema ve veri değişikliklerinden önce tam bir yedek alarak, eğer yükseltme veya değişiklikler sırasında bir sorun çıkarsa doğrudan bu noktaya geri dönebiliriz.**

- **Sürüm Kontrolü: Yükseltme öncesi sistemin tam olarak hangi durumda olduğunu belgelemek, sonradan yapılacak karşılaştırma ve raporlamalar için kritik.**

Adım 2 – Sürüm Geçmişi (Version History) Tablosunu Oluşturma

SQLQuery1.sql - DE...P-G2TFBJV\33 (52))*

```sql
USE KullaniciDB;
GO

CREATE TABLE dbo.VersionHistory (
  ChangeID      INT             IDENTITY(1,1) PRIMARY KEY,
  EventTime     DATETIME2       NOT NULL  DEFAULT SYSUTCDATETIME(),
  LoginName     SYSNAME         NOT NULL,
  EventType     NVARCHAR(100)   NOT NULL,
  ObjectType    NVARCHAR(100)   NOT NULL,
  ObjectName    NVARCHAR(256)   NOT NULL,
  CommandText   NVARCHAR(MAX)   NOT NULL
);
GO
```

121 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-04T17:47:30.3190559+03:00
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (52) | KullaniciDB | 00:00:00 | 0 rows

Ln 14 Col 1 Ch 1 INS

- **Değişiklik Kaydı: Veritabanı yükseltme veya sürüm yönetimi sürecinde yapılan her şema değişikliğini (yeni tablo ekleme, kolon değiştirme, trigger oluşturma vb.) bu tabloda tutarak bir "audit trail" oluşturuyoruz.**

- **Otomatik Zaman ve Kullanıcı: EventTime kolonu otomatik olarak işlemin tarih-zamanını, LoginName ise işlemi yapan SQL oturumunu kaydediyor. Böylece kimin, ne zaman, hangi komutu çalıştırdığı tek bir merkezi tabloda toplanmış oluyor.**

Adım 3 – DDL Trigger ile Şema Değişikliklerini Otomatik Olarak Kaydetme

SQLQuery1.sql - DE...P-G2TFBJV\33 (52))*

```sql
CREATE TRIGGER trg_SchemaChanges
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
BEGIN
  SET NOCOUNT ON;

  INSERT INTO dbo.VersionHistory (
    LoginName,
    EventType,
    ObjectType,
    ObjectName,
    CommandText
  )
  SELECT
    ORIGINAL_LOGIN(),
    EVENTDATA().value('(/EVENT_INSTANCE/EventType)[1]', 'NVARCHAR(100)'),
    EVENTDATA().value('(/EVENT_INSTANCE/ObjectType)[1]', 'NVARCHAR(100)'),
    EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]', 'NVARCHAR(256)'),
    EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand)[1]', 'NVARCHAR(MAX)')
  ;
END;
GO
```

121 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-04T17:48:25.1842660+03:00
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (52) | KullaniciDB | 00:00:00 | 0 rows

Ln 24 | Col 1 | Ch 1 | INS

- **Trigger Kapsamı:**
  - **ON DATABASE FOR DDL_DATABASE_LEVEL_EVENTS ile tüm veritabanı düzeyindeki DDL (CREATE, ALTER, DROP vb.) olayları dinliyoruz.**
- **EVENTDATA() Fonksiyonu:**
  - **XML formatındaki olay bilgisinden (EventType, ObjectType, ObjectName, TSQLCommand) ilgili alanları çıkarıyoruz.**
- **VersionHistory Tablosuna Ekleme:**
  - **ORIGINAL_LOGIN() ile kimlik doğrulama yapan kullanıcı adını alıyoruz.**
  - **Diğer alanları EVENTDATA()'dan çekip, VersionHistory tablosuna INSERT ediyoruz.**

Adım 4 – Şema Değişikliğini Test Etme ve Kayıtların Kontrolü

- **Şema Değişikliği:**

- **Kullanicilar tablosuna Telefon2 adında, NULL kabul eden yeni bir sütun ekledik.**

- **Trigger'ın Otomatik Tetiklenmesi:**

- **Önceki adımda yarattığımız trg_SchemaChanges tetikleyicisi (DDL_DATABASE_LEVEL_EVENTS) tam da böyle bir DDL olayı (ALTER TABLE) yakalamak üzere ayarlanmıştı.**

- **VersionHistory Kaydı:**

- **ALTER TABLE komutu yürütülür yürütülmez, tetikleyici devreye girdi ve XML içindeki:**
  - **Olay türü (EventType = "ALTER_TABLE")**
  - **Nesne türü (ObjectType = "TABLE")**

- **Nesne adı (ObjectName = “dbo.Kullanicilar”)**
- **Çalışan komut metni**
- **Kullanıcı adı (ORIGINAL_LOGIN())**

- **bunları VersionHistory tablosuna ekledi.**

Yukarıda gördüğümüz gibi , en son yaptığımız ALTER TABLE işlemi VersionHistory tablosuna başarıyla kaydedilmiş:

- **FORMAT, INIT seçenekleriyle önceki yedek setlerini temizledik ve dosyanın başından başlayarak yeni bir backup set oluşturduk.**

- **Çıktıda “464 pages” ve “2 pages” ile gösterilen miktarlar veri ve log dosyaları için işlenen sayfa sayısını; MB/sec ise yedekleme hızını veriyor.**

- **Bu işlemden sonra KullaniciDB_PostUpgrade.bak dosyamız, veritabanımızın güncel (yükseltilmiş) halinin tam bir kopyasını içeriyor.**

Adım 5: Test & Geri Dönüş Planı – Veritabanını Offline'a Alıp Silmek

Yükseltme sonrası aldığımız KullaniciDB_PostUpgrade.bak yedeğini geri yükleyebileceğimiz temiz bir ortam yaratmak — önce veritabanını tamamen ortadan kaldırıyor, sonra (bir sonraki adımda) yedeği RESTORE ile geri getiriyoruz.

Adım 6: Ön Yükseltme Yedeğini Geri Yükleme

Önceki adımda silip temizlediğimiz 'KullaniciDB' veritabanını, KullaniciDB_PreUpgrade.bak dosyasından RESTORE DATABASE … WITH MOVE komutu ile C:\SQLData altında orijinal mantıksal dosya isimleri korunarak geri yükledik ve RECOVERY aşamasıyla veritabanını aktif hale getirdik.

SQLQuery1.sql - DE...P-G2TFBJV\33 (52))*

```sql
USE KullaniciDB;
GO

-- Örneğin, upgrade sırasında eklediğimiz DDL trigger ve sütunların olmadığını doğrulayın
SELECT *
FROM sys.triggers
WHERE name = N'DDLChangeTracker';  -- bu trigger'ın artık yok olması gerekiyor

-- Tablo yapısını kontrol edin
EXEC sys.sp_help 'Kullanicilar';  -- "YeniSütun" gibi upgrade öğelerinin listede olmaması gereki
```

121 %

Results | Messages

name	object_id	parent_class	parent_class_desc	parent_id	type	type_desc	create_date	modify_date	is_ms_shipped	is_disabled	is_not_for_replication	is_instead_of

	Name	Owner	Type	Created_datetime
1	Kullanicilar	dbo	user table	2025-04-07 14:37:06.810

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	KullaniciID	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	KullaniciAdi	nvarchar	no	100			yes	(n/a)	(n/a)	Turkish_CI_AS
3	Sifre	nvarchar	no	200			yes	(n/a)	(n/a)	Turkish_CI_AS
4	Email	nvarchar	no	200			yes	(n/a)	(n/a)	Turkish_CI_AS

	Identity	Seed	Increment	Not For Replication
1	KullaniciID	1	1	0

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

	index_name	index_description	index_keys
1	PK__Kullanic__E011F09B82C7BA9B	clustered, unique, primary key located on PRIMARY	KullaniciID

	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1	PRIMARY KEY (clustered)	PK__Kullanic__E011F09B82C7BA9B	(n/a)	(n/a)	(n/a)	(n/a)	KullaniciID

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (52) | KullaniciDB | 00:00:00 | 10 rows

Ln 11 | Col 1 | Ch 1 | INS

Bu adımda ön-upgrade yedeğini geri yükledikten sonra veritabanının gerçekten eski hâline döndüğünü doğruladık.

Adım 7: SchemaChangeLog Tablosunu Oluşturma

SQLQuery1.sql - DE...P-G2TFBJV\33 (52))*

```sql
USE KullaniciDB;
GO

DROP TABLE IF EXISTS dbo.SchemaChangeLog;
GO

-- Şimdi tekrar oluşturun:
CREATE TABLE dbo.SchemaChangeLog
(
  LogID       INT IDENTITY(1,1) PRIMARY KEY,
  EventDate   DATETIME2      NOT NULL DEFAULT SYSUTCDATETIME(),
  EventType   NVARCHAR(100)  NOT NULL,
  ObjectName  NVARCHAR(256)  NOT NULL,
  TsqlCommand NVARCHAR(MAX)  NOT NULL,
  LoginName   NVARCHAR(128)  NOT NULL DEFAULT ORIGINAL_LOGIN()
);
GO
```

121 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-04T17:58:18.2412315+03:00
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (52) | KullaniciDB | 00:00:00 | 0 rows

Ln 18 Col 1 Ch 1 INS

```sql
CREATE TRIGGER trg_SchemaChange
ON DATABASE
FOR
    CREATE_TABLE, ALTER_TABLE, DROP_TABLE,
    CREATE_PROCEDURE, ALTER_PROCEDURE, DROP_PROCEDURE,
    CREATE_VIEW, ALTER_VIEW, DROP_VIEW
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO dbo.SchemaChangeLog(EventType, ObjectName, TsqlCommand)
    SELECT
        EVENTDATA().value('(/EVENT_INSTANCE/EventType)[1]', 'NVARCHAR(100)'),
        EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]',  'NVARCHAR(256)'),
        EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand)[1]','NVARCHAR(MAX)');
END;
GO
```

```
Commands completed successfully.

Completion time: 2025-05-04T17:58:59.5406935+03:00
```

Bu iki adım, “Sürüm Yönetimi” projesinin DDL değişikliklerini izleme gereksinimini karşılıyor. İlk olarak log tablosunu tanımladık, ardından da veritabanı seviyesinde DDL tetikleyicisi oluşturarak tüm şema değişikliklerini merkezi bir tabloda toplamış olduk.

SQLQuery1.sql - DE...P-G2TFBJV\33 (52))*

```sql
CREATE TABLE Test_Dummy
(
  DummyID   INT PRIMARY KEY,
  DummyName NVARCHAR(50)
);
GO
```

121 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-04T17:59:24.0928694+03:00
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (52) | KullaniciDB | 00:00:00 | 0 rows

Ln 7 Col 1 Ch 1 INS

SQLQuery1.sql - DE...P-G2TFBJV\33 (52))*

```sql
SELECT TOP 5 *
FROM dbo.SchemaChangeLog
ORDER BY LogID DESC;
GO
```

121 %

Results | Messages

	LogID	EventDate	EventType	ObjectName	TsqlCommand	LoginName
1	1	2025-05-04 14:59:24.0928694	CREATE_TABLE	Test_Dummy	CREATE TABLE Test_Dummy (DummyID INT PRIMA...	DESKTOP-G2TFBJV\33

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (52) | KullaniciDB | 00:00:00 | 1 rows

Ln 5 Col 1 Ch 1 INS

SQLQuery1.sql - DE...P-G2TFBJV\33 (52))*

```sql
USE KullaniciDB;
GO

-- Simülasyon: önemli bir tabloyu sil
DROP TABLE dbo.Kullanicilar;
GO
```

121 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-04T18:13:25.1296231+03:00
```

121 %

Query executed successfully. | DESKTOP-G2TFBJV (15.0 RTM) | DESKTOP-G2TFBJV\33 (52) | KullaniciDB | 00:00:00 | 0 rows

Ln 7 Col 1 Ch 1 INS

1. **Test_Dummy tablosu CREATE edildi → tetikleyici CREATE_TABLE olayını logladı.**
2. **SchemaChangeLog sorgusuyla bu olayı görüntüledik.**
3. **Kullanicilar tablosunu DROP ettik → tetikleyici DROP_TABLE olayını logladı.**
4. **Yine SchemaChangeLog sorgusuyla bu DROP olayını doğruladık.**

Bu adımlar "Sürüm Yönetimi" sürecinde DDL değişiklik izleme ve geri dönüş planı için gerekli. Tetikleyici sayesinde tüm CREATE/ALTER/DROP işlemleri otomatik olarak kaydediliyor; gerektiğinde, tam olarak ne zaman, kim tarafından ve hangi komutlarla yapıldığı SchemaChangeLog tablosundan geri alınabiliyor.

Adım 8 Geri Dönüş

- **Veritabanını tek kullanıcı moduna geçiriyoruz.**
- **Halen açık oturumlar varsa hemen kesiliyor (ROLLBACK IMMEDIATE), böylece RESTORE işlemi öncesi hiçbir bağlantı engelleme yapmaz.**

- **Daha önce aldığımız “pre-upgrade” tam yedeğini geri yüklüyoruz.**
- **REPLACE seçeneği varolan veritabanı dosyalarının üzerine yazar.**
- **RECOVERY ile veritabanını normal (kullanılabilir) duruma getirir.**

Restore işleminden sonra veritabanını tekrar çok kullanıcılı moda alıyoruz, böylece birden fazla oturum yeniden bağlanabilir.