



**BLM4522 - Ağ Tabanlı Paralel Dağıtım Sistemleri [A]**

Mustafa Girgin

21290649

Merhaba hocam,  
Vize için 2, 3 ve 5. projeleri yapmıştım. Final için geri kalan projelerimi de tamamladım.

GitHub hesabım: <https://github.com/antelcha/sqlprojeler>

Videolar drive linkleri şu şekilde:

## **Vize kısmı**

### **3. Veritabanı Güvenliği ve Erişim Kontrolü**

[https://drive.google.com/file/d/1wl7qa\\_8-u1VH2Rt4eXEBDBbGj2VSLiSB/view?usp=sharing](https://drive.google.com/file/d/1wl7qa_8-u1VH2Rt4eXEBDBbGj2VSLiSB/view?usp=sharing)

### **2. Veritabanı Yedekleme ve Felaketten Kurtarma Planı**

[https://drive.google.com/file/d/1\\_MeIRiXIXufiji1CM9dEr2TPAuqzpvS1A/view?usp=sharing](https://drive.google.com/file/d/1_MeIRiXIXufiji1CM9dEr2TPAuqzpvS1A/view?usp=sharing)

### **5. Veri Temizleme ve ETL Süreçleri Tasarımı**

<https://drive.google.com/file/d/1tAcQGo5LVVqvB-b5IUOxLIJdD2Hiosrr/view?usp=sharing>

## **Final kısmı**

### **1. Proje**

[https://drive.google.com/file/d/15vIBKFa9\\_rQXWr1490T7J3gRmU\\_N7C6v/view?usp=sharing](https://drive.google.com/file/d/15vIBKFa9_rQXWr1490T7J3gRmU_N7C6v/view?usp=sharing)

### **4. Proje**

[https://drive.google.com/file/d/1E4Bc1\\_xnkqmluMtP6QGoDrjoQrQEvlS/vi ew?usp=sharing](https://drive.google.com/file/d/1E4Bc1_xnkqmluMtP6QGoDrjoQrQEvlS/vi ew?usp=sharing)

### **6. Proje**

[https://drive.google.com/file/d/1SsOsVjIVGrFV06tjk\\_PEqVaUQZzki94e/view?usp=sharing](https://drive.google.com/file/d/1SsOsVjIVGrFV06tjk_PEqVaUQZzki94e/view?usp=sharing)

### **7. Proje**

[https://drive.google.com/file/d/1QpO65cbZj7xeVV5zBNGcUrX-KE71\\_p1C/view?usp=sharing](https://drive.google.com/file/d/1QpO65cbZj7xeVV5zBNGcUrX-KE71_p1C/view?usp=sharing)

Aşağıdaki kısımda da 7 raporu birleştirip kojuyorum.

# Veritabanı Performans Optimizasyonu ve İzleme

AdventureWorks Veritabanında Sorgu Performans Optimizasyonu AdventureWorks veritabanı üzerinde yavaş çalışan bir sorguyu tespit ederek, index optimizasyonu ile performansını artırmayı hedefledim. Bu süreçte SQL Server'in performans izleme araçlarını kullanarak, optimizasyon öncesi ve sonrası değerleri karşılaştırdım.

Performans Problemi Tespiti İlk olarak, müşteri siparişlerini ve ürün detaylarını getiren kompleks bir sorgu oluşturdum. Bu sorgu birden fazla tablo birleştirmesi (JOIN), tarih filtrelemesi ve LIKE operatörü içeriyordu. STATISTICS TIME ve IO'yu aktif ederek sorgunun kaynak tüketimini ölçtüm.

[photo]

Sorgu çalıştırıldığında şu değerler elde edildi:

Your text here 2

CPU Time: 36 ms

Total Logical Reads: 1,810

SalesOrderDetail: 890 reads

SalesOrderHeader: 686 reads

Performans Analizi ve Missing Index Kontrolü SQL Server'in Dynamic Management Views (DMV) kullanarak, veritabanının önerdiği eksik indeksleri kontrol ettim. Sistem, sorgu performansını artırabilecek potansiyel indeksleri belirledi.

İndeks Oluşturma Stratejisi Analiz sonuçlarına dayanarak üç adet nonclustered index oluştururdum:

IX\_SalesOrderHeader\_OrderDate: OrderDate kolonuna index ekleyerek tarih filtrelemesini hızlandırdım. CustomerID ve SalesOrderID kolonlarını include ederek covering index oluştururdum.

IX\_Product\_Name: Product tablosundaki Name kolonuna index ekleyerek LIKE aramasını optimize ettim.

IX\_SalesOrderDetail\_SalesOrderID\_ProductID: JOIN işlemlerinde kullanılan kolonlara composite index oluşturdum. OrderQty, UnitPrice ve LineTotal kolonlarını include ederek key lookup'ları önledim.

Optimizasyon Sonrası Test İndeksler oluşturulduktan sonra aynı soruyu tekrar çalıştırıldım ve performans metriklerini karşılaştırdım:

Optimizasyon sonrası değerler:

CPU Time: 33 ms (%8 azalma)

Total Logical Reads: 964 (%47 azalma)

SalesOrderDetail: 689 reads (%23 azalma)

SalesOrderHeader: 41 reads (%94 azalma!)

Performans Karşılaştırma Raporu Kazanımları net bir şekilde görebilmek için detaylı bir karşılaştırma raporu hazırladım:

Ana Kazanımlar:

Toplam logical read sayısı %47 azaldı (1,810'dan 964'e)

En büyük iyileşme SalesOrderHeader tablosunda görüldü (%94 azalma)

Table scan işlemleri index seek'e dönüştü

Covering index sayesinde key lookup ihtiyacı ortadan kalktı

Execution Plan Analizi Optimizasyon öncesi ve sonrası execution plan'ları karşılaştırdığında:

Öncesi:

Clustered Index Scan (yüksek maliyet)

Key Lookup işlemleri

Sort operasyonları

Sonrası:

Index Seek (düşük maliyet)

Covering index kullanımı

Daha verimli join stratejileri

Bu çalışma, doğru indeks stratejisinin soru performansı üzerindeki dramatik etkisini açıkça göstermiştir. Veritabanı optimizasyonu sürekli bir süreç olduğundan, düzenli performans analizleri ve iyileştirmeler yapılması kritik öneme sahiptir.

## ÖZET RAPOR

- ? TOPLAM LOGICAL READ: %47 AZALDI (1810 → 964)
- ? EN BÜYÜK İYİLESME: SalesOrderHeader - %94 AZALDI
- ? CPU KULLANIMI: %8 AZALDI
- ? 3 ADET OPTIMIZE INDEX OLUSTURULDU

SONUÇ: Soru performansi BASARIYLA optimize edildi!

[2025-05-30 00:50:30] completed in 5 ms

Performans Metrikleri	Optimizasyon Öncesi	Optimizasyon Sonrası	İyileşme Oranı
CPU Time (ms)	36	33	%8 azaldı
Total Logical Reads	1810	964	%47 azaldı
SalesOrderDetail Reads	890	689	%23 azaldı
SalesOrderHeader Reads	686	41	%94 azaldı
Customer Reads	123	123	Degismedi
Person Reads	105	105	Degismedi
Product Reads	6	4	Degismedi

```
USE AdventureWorks;
GO

-- PERFORMANS KARŞILAŞTIRMA RAPORU
PRINT '====='
PRINT '      PERFORMANS OPTİMİZASYON RAPORU'
PRINT '====='
PRINT ''

-- Geçici tablo oluştur
CREATE TABLE #PerformansKarsilastirma
(
    Metrik NVARCHAR(50),
    Oncesi NVARCHAR(50),
```

```

        Sonrasi NVARCHAR(50),
        Iyilesme NVARCHAR(50)
    )

-- Verileri ekle
INSERT INTO #PerformansKarsilastirma VALUES
('CPU Time (ms)', '36', '33', '%8 azaldı'),
('Total Logical Reads', '1810', '964', '%47 azaldı'),
('SalesOrderDetail Reads', '890', '689', '%23 azaldı'),
('SalesOrderHeader Reads', '686', '41', '%94 azaldı'),
('Customer Reads', '123', '123', 'Değişmedi'),
('Person Reads', '105', '105', 'Değişmedi'),
('Product Reads', '6', '6', 'Değişmedi')

-- Sonuçları göster
SELECT
    Metrik AS 'Performans Metrikleri',
    Oncesi AS 'Optimizasyon Öncesi',
    Sonrasi AS 'Optimizasyon Sonrası',
    Iyilesme AS 'İyileşme Oranı'
FROM #PerformansKarsilastirma

DROP TABLE #PerformansKarsilastirma
GO

PRINT ''
PRINT '===== OLUŞTURULAN İNDEKSLER ====='
PRINT ''
PRINT ''
PRINT ''

-- Index bilgileri
SELECT
    i.name AS 'Index Adı',
    OBJECT_NAME(i.object_id) AS 'Tablo',
    i.type_desc AS 'Index Tipi',
    STUFF(
        SELECT ', ' + c.name
        FROM sys.index_columns ic
        JOIN sys.columns c ON ic.object_id = c.object_id AND ic.column_id =
c.column_id
        WHERE ic.object_id = i.object_id AND ic.index_id = i.index_id AND
ic.is_included_column = 0
        ORDER BY ic.key_ordinal
        FOR XML PATH('')
    ), 1, 2, '') AS 'Index Kolonları',
    STUFF(
        SELECT ', ' + c.name
        FROM sys.index_columns ic

```

```
        JOIN sys.columns c ON ic.object_id = c.object_id AND ic.column_id =
c.column_id
        WHERE ic.object_id = i.object_id AND ic.index_id = i.index_id AND
ic.is_included_column = 1
        FOR XML PATH('')
), 1, 2, '') AS 'Include Kolonları'
FROM sys.indexes i
WHERE i.name IN (
    'IX_SalesOrderHeader_OrderDate',
    'IX_Product_Name',
    'IX_SalesOrderDetail_SalesOrderID_ProductID'
)
ORDER BY i.name
GO

PRINT ''
PRINT '====='
PRINT '      ÖZET RAPOR'
PRINT '====='
PRINT ''
PRINT '  ✓ TOPLAM LOGICAL READ: %47 AZALDI (1810 → 964)'
PRINT '  ✓ EN BÜYÜK İYİLEŞME: SalesOrderHeader - %94 AZALDI'
PRINT '  ✓ CPU KULLANIMI: %8 AZALDI'
PRINT '  ✓ 3 ADET OPTİMİZE INDEX OLUŞTURULDU'
PRINT ''
PRINT 'SONUÇ: Sorgu performansı BAŞARIYLA optimize edildi!'
PRINT '=====
```

Mustafa Girgin  
21290649

<https://github.com/antelcha/sqlprojeler>

# Veritabanı Yedekleme ve Felaketten Kurtarma Planı

AdventureWorks veritabanı üzerinde temel yedekleme stratejilerini uygulayarak, olası veri kayıplarına karşı geri dönüşü mümkün kılan bir yapı tasarladım. Bu süreçte, sırasıyla tam yedek (full backup), fark yedeği (differential backup) ve işlem günlüklerinin (log backup) alınması ve daha sonra bu yedeklerin doğru sırayla geri yüklenmesi işlemlerini gerçekleştirdim.

## 1. Recovery Model Ayarlanması

SQL Server'da işlem günlüklerini yedekleyebilmek için veritabanının FULL recovery modeline geçirilmesi gereklidir. Bu sayede sistem, son log yedeğinden itibaren yapılan her işlemi izleyebilir ve olası felaket durumlarında nokta atışı geri yüklemeye olanak tanır.

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

## 2. Tam Yedek (Full Backup)

İlk olarak veritabanının o anki tam yedeğini aldım. Bu işlem, veritabanının tüm sayfalarını ve yapılarını içerir. Yedekleme dosyasını, Docker container içinde `/var/opt/mssql/backup/` klasörüğe kaydettim.

```
1 ✓ BACKUP DATABASE AdventureWorks
2 TO DISK = '/var/opt/mssql/backup/AdventureWorks_Full.bak'
3 WITH INIT, FORMAT;
4 |
```

```
[2025-04-21 20:26:44] [S0001][4035] Processed 25376 pages for database 'AdventureWorks', file 'AdventureWorks2022' on file 1.  
[2025-04-21 20:26:44] [S0001][4035] Processed 2 pages for database 'AdventureWorks', file 'AdventureWorks2022_log' on file 1.  
[2025-04-21 20:26:44] [S0001][3014] BACKUP DATABASE successfully processed 25378 pages in 0.250 seconds (793.046 MB/sec).  
[2025-04-21 20:26:44] completed in 371 ms
```

### 3. Fark Yedeği (Differential Backup)

Tam yedekten sonra veritabanında yapılan değişikliklerin yedeğini almak için differential backup kullandım. Bu işlem yalnızca **tam yedekten sonra değişen sayfaları** içерdiği için daha hızlı ve hafiftir.

```
1 ✓ BACKUP DATABASE AdventureWorks  
2 TO DISK = '/var/opt/mssql/backup/AdventureWorks_Diff.bak'  
3 WITH DIFFERENTIAL, INIT;  
4
```

### 4. Artık Yedek (Transaction Log Backup)

Transaction log yedeği, en son alınan yedekten sonra yapılan işlemleri içerir. Bu sayede, veri kaybı olmadan tam zamanında geri dönüş sağlanabilir. Özellikle kurumsal ortamlarda, sistemin her an kurtarılabilir olması için log yedeklemesi zorunludur.

```
1 ✓ BACKUP LOG AdventureWorks  
2 TO DISK = '/var/opt/mssql/backup/AdventureWorks_Log.trn'  
3 WITH INIT;  
4  
5
```

### 5. Yedeklerin Listelenmesi

Alınan yedeklerin kayıt altına alındığını kontrol etmek için `msdb.dbo.backupset` tablosunu sorguladım. Böylece hangi türde yedeklerin hangi tarihte alındığını görebildim.

```
1 ✓ SELECT
2     database_name,
3     backup_start_date,
4     backup_finish_date,
5     CASE
6         WHEN type = 'D' THEN 'FULL'
7         WHEN type = 'I' THEN 'DIFFERENTIAL'
8         WHEN type = 'L' THEN 'LOG'
9     END AS backup_type,
10    physical_device_name
11   FROM msdb.dbo.backupset bs
12  JOIN msdb.dbo.backupmediafamily bmf
13    ON bs.media_set_id = bmf.media_set_id
14 WHERE database_name = 'AdventureWorks'
15 ORDER BY backup_finish_date DESC;|
```

Output | Plan | master.sys.databases | Result 355 ×

	database_name	backup_start_date	backup_finish_date	backup_type	physical_device_name
1	AdventureWorks	2025-04-21 17:28:02.000	2025-04-21 17:28:02.000	LOG	/var/opt/mssql/backup/AdventureWorks_LOG.trn
2	AdventureWorks	2025-04-21 17:27:32.000	2025-04-21 17:27:33.000	DIFFERENTIAL	/var/opt/mssql/backup/AdventureWorks_DIFF.bak
3	AdventureWorks	2025-04-21 17:26:44.000	2025-04-21 17:26:44.000	FULL	/var/opt/mssql/backup/AdventureWorks_Full.bak

## 6. Veritabanını Kasten Silip Geri Yükleme (Felaket Senaryosu)

Bu adımda, AdventureWorks veritabanını bilinçli olarak sileceğim ve daha önce aldığım yedeklerle sıfırdan geri yükleyeceğim

```
WHERE name = 'AdventureWorks'
[2025-04-21 20:32:30] 1 row retrieved starting from 1 in 355 ms (execution: 11 ms, fetching: 344 ms)
master> -- Önce aktif bağlantıları kapatmak gerekebilir
        ALTER DATABASE AdventureWorks SET SINGLE_USER WITH ROLLBACK IMMEDIATE;

-- Sonra veritabanını silebiliriz
        DROP DATABASE AdventureWorks;
[2025-04-21 20:32:55] completed in 105 ms
```

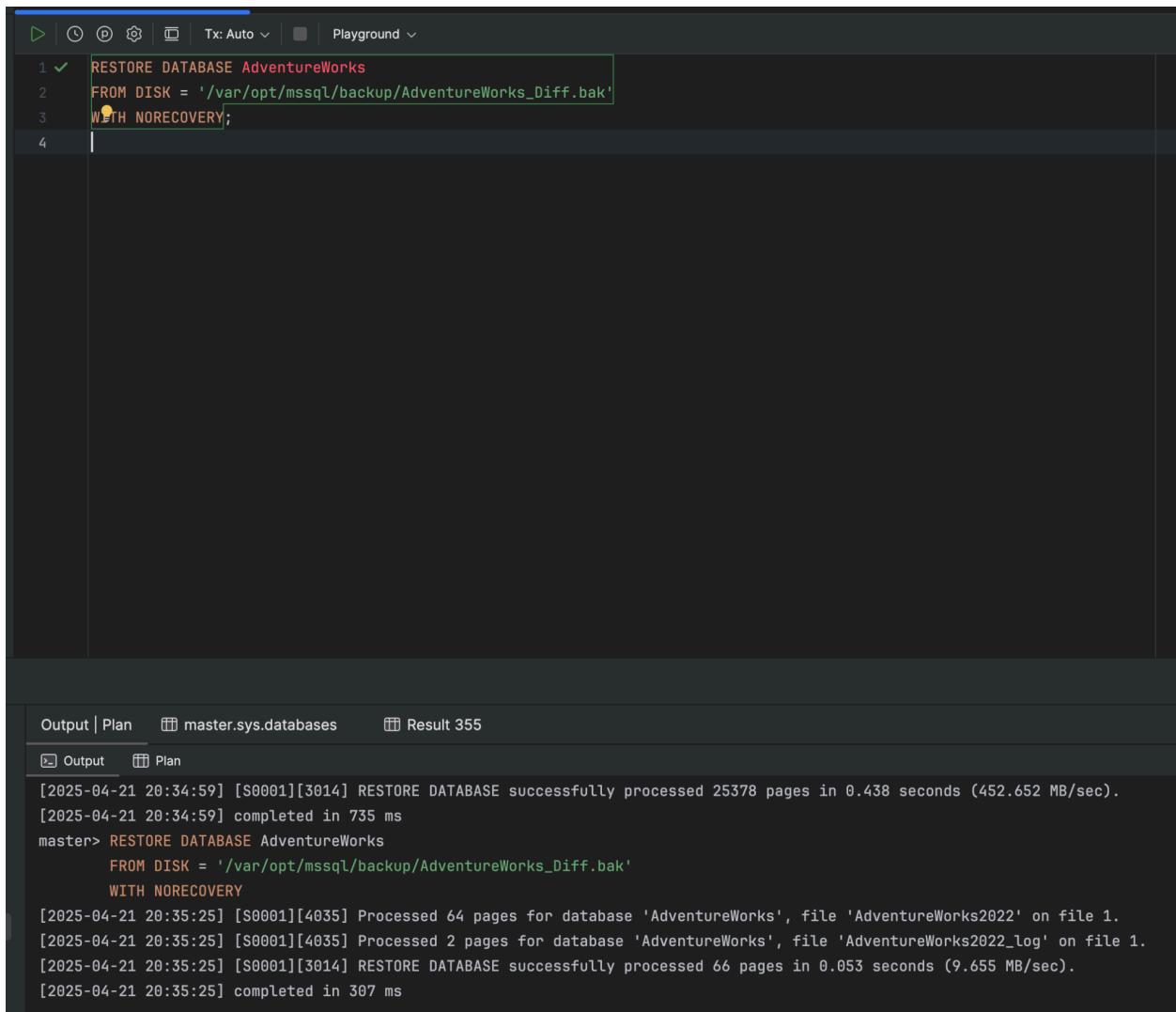
## 7. Full Backup'tan Geri Yükleme

```
1 ✓ RESTORE DATABASE AdventureWorks
2   FROM DISK = '/var/opt/mssql/backup/AdventureWorks_Full.bak'
3   WITH REPLACE, NORECOVERY;
4
5

Output | Plan   master.sys.databases   Result 355
Output  Plan
master>-- Once aktif baglantilari kapatmak gerekebilir
        ALTER DATABASE AdventureWorks SET SINGLE_USER WITH ROLLBACK IMMEDIATE;

        -- Sonra veritabanini silebiliriz
        DROP DATABASE AdventureWorks;
[2025-04-21 20:32:55] completed in 105 ms
master> RESTORE DATABASE AdventureWorks
        FROM DISK = '/var/opt/mssql/backup/AdventureWorks_Full.bak'
        WITH REPLACE, NORECOVERY
[2025-04-21 20:34:58] [S0001][4035] Processed 25376 pages for database 'AdventureWorks', file 'AdventureWorks2022' on file 1.
[2025-04-21 20:34:58] [S0001][4035] Processed 2 pages for database 'AdventureWorks', file 'AdventureWorks2022_log' on file 1.
[2025-04-21 20:34:59] [S0001][3014] RESTORE DATABASE successfully processed 25378 pages in 0.438 seconds (452.652 MB/sec).
[2025-04-21 20:34:59] completed in 735 ms
```

## 8. Differential Backup'tan Geri Yükleme



The screenshot shows a SQL playground interface with a dark theme. In the top-left corner, there are several icons: a green play button, a clock, a circular arrow, a refresh symbol, and a square. To the right of these are dropdown menus labeled "Tx: Auto" and "Playground". Below these are four numbered lines of T-SQL code:

```
1 ✓ RESTORE DATABASE AdventureWorks  
2 FROM DISK = '/var/opt/mssql/backup/AdventureWorks_Diff.bak'  
3 WITH NORECOVERY;  
4
```

Below the code editor, there are tabs for "Output" and "Plan". The "Output" tab is selected, showing the results of the restore command. The output log includes the following entries:

```
[2025-04-21 20:34:59] [S0001][3014] RESTORE DATABASE successfully processed 25378 pages in 0.438 seconds (452.652 MB/sec).  
[2025-04-21 20:34:59] completed in 735 ms  
master> RESTORE DATABASE AdventureWorks  
    FROM DISK = '/var/opt/mssql/backup/AdventureWorks_Diff.bak'  
    WITH NORECOVERY  
[2025-04-21 20:35:25] [S0001][4035] Processed 64 pages for database 'AdventureWorks', file 'AdventureWorks2022' on file 1.  
[2025-04-21 20:35:25] [S0001][4035] Processed 2 pages for database 'AdventureWorks', file 'AdventureWorks2022_log' on file 1.  
[2025-04-21 20:35:25] [S0001][3014] RESTORE DATABASE successfully processed 66 pages in 0.053 seconds (9.655 MB/sec).  
[2025-04-21 20:35:25] completed in 307 ms
```

## 9. Log Backup'tan Geri Yükleme

```
1 ✓ RESTORE LOG AdventureWorks
2   FROM DISK = '/var/opt/mssql/backup/AdventureWorks_Log.trn'
3   WITH RECOVERY;
```

Output | Plan master.sys.databases Result 355

Output Plan

```
[2025-04-21 20:35:25] [S0001][3014] RESTORE DATABASE successfully processed 66 pages in 0.053 seconds (9.655 MB/sec).
[2025-04-21 20:35:25] completed in 307 ms
master> RESTORE LOG AdventureWorks
      FROM DISK = '/var/opt/mssql/backup/AdventureWorks_Log.trn'
      WITH RECOVERY
[2025-04-21 20:36:27] [S0001][4035] Processed 0 pages for database 'AdventureWorks', file 'AdventureWorks2022' on file 1.
[2025-04-21 20:36:27] [S0001][4035] Processed 8 pages for database 'AdventureWorks', file 'AdventureWorks2022_log' on file 1.
[2025-04-21 20:36:28] [S0001][3014] RESTORE LOG successfully processed 8 pages in 0.007 seconds (8.928 MB/sec).
[2025-04-21 20:36:28] completed in 423 ms
```

## 10. Veritabanı Durumunu Kontrol Etme

The screenshot shows a database management interface with a query editor and a results grid.

**Query Editor:**

```
1 ✓ SELECT name, state_desc  
2      FROM sys.databases  
3      WHERE name = 'AdventureWorks';  
4
```

**Results Grid:**

	name	state_desc
1	AdventureWorks	ONLINE

Mustafa Girgin  
21290649

<https://github.com/antelcha/sqlprojeler>

# Veritabanı Güvenliği ve Erişim Kontrolü

## HumanResources Şeması İçin Güvenli Kullanıcı Tanımlama

AdventureWorks veritabanı üzerinde temel güvenlik önlemlerini adım adım uygulayarak, HumanResources şemasına özel bir kullanıcı tanımladım. Buradaki hedefim, yalnızca ihtiyaç duyulan en temel yetkileri vererek, sistemin hem işlevsellliğini hem de güvenliğini korumaktı.

### 1. Sunucu Seviyesinde Login Oluşturulması

İlk olarak, sisteme kimlerin bağlanabileceğini kontrol altına almak gerekiyor. Bu nedenle, SQL Server Authentication kullanarak HR\_Login isimli bir kimlik tanımladım. Bu login henüz herhangi bir veritabanında işlem yapma yetkisine sahip değil; sadece sunucuya erişim izni var.

```
1 CREATE LOGIN HR_Login
2   WITH PASSWORD = 'HrLogin1!';
3
4
5
```

### 2. Veritabanı Seviyesinde User Tanımlanması

Sunucuya bağlanma yetkisi olan HR\_Login'in, AdventureWorks veritabanı içerisinde işlem yapabilmesi için burada bir kullanıcıya (user) dönüştürülmesi gerekiyordu. Bu yüzden önce veritabanını seçtim, ardından HR\_User adında bir kullanıcı oluşturdum. Henüz herhangi bir

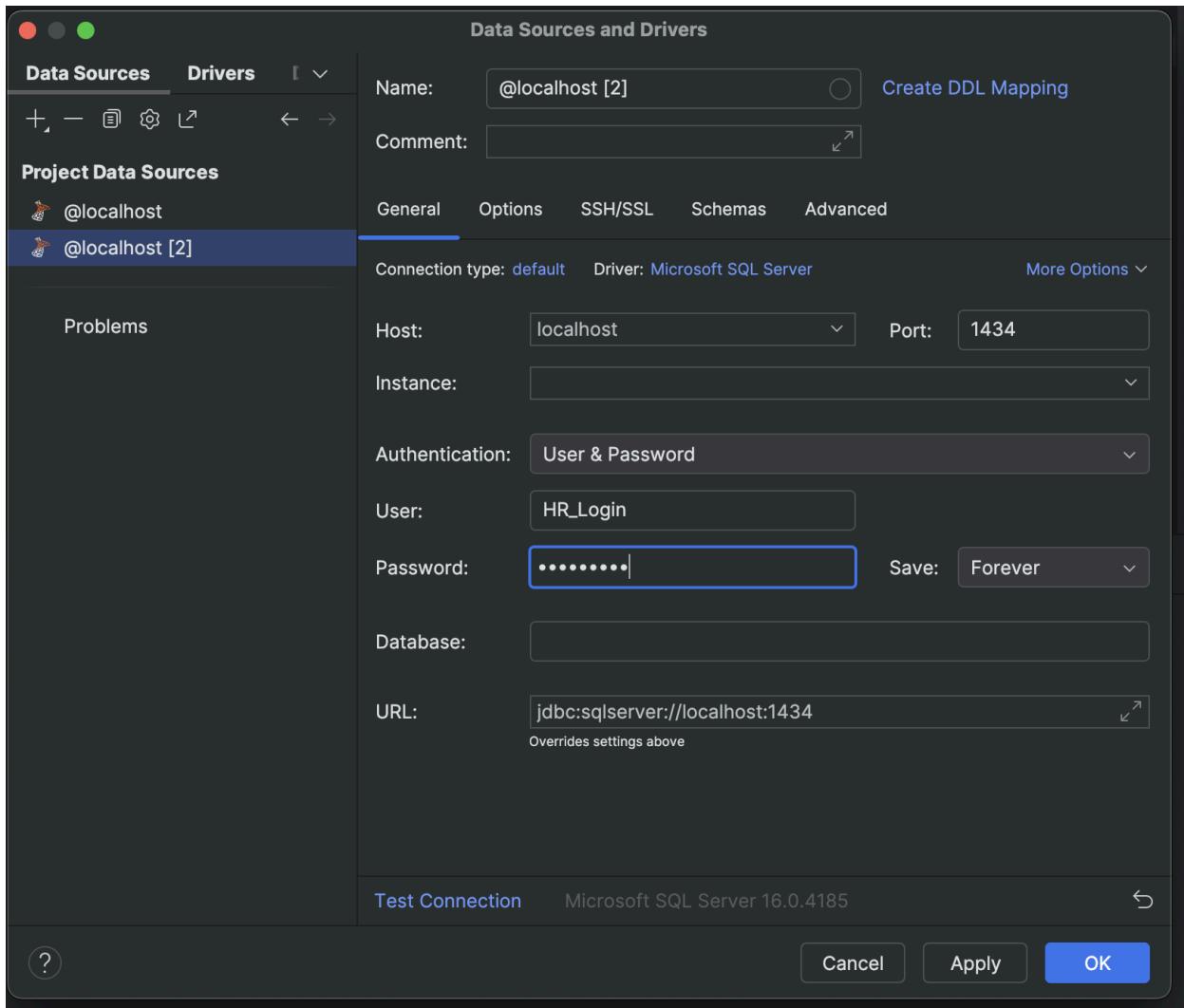
tabloya erişimi yok.

```
1 USE AdventureWorks;
2 CREATE USER HR_User FOR LOGIN HR_Login;
3 GO
4
5 |
```

### 3. Yetkilendirme (GRANT)

HR\_User'ın asıl amacı HumanResources şemasındaki verilere erişim sağlamak olduğu için, bu kullanıcıya yalnızca SELECT (okuma) yetkisi verdim. Böylece kullanıcı verileri görüntüleyebilecek, ancak değiştiremeyecek. Yetkilendirme sürecini **GRANT SELECT ON SCHEMA :: HumanResources TO HR\_User** şeklinde tamamladım.

```
1 GRANT SELECT ON SCHEMA :: HumanResources TO HR_User;
2 GO
3
4 |
```



#### 4. Yapılandırmanın Test Edilmesi

Tanımladığım yapıların beklentiği gibi çalıştığından emin olmak için, DataGrip üzerinden HR\_Login ile bağlantı kurdum ve bazı testler yaptım:

1. **Yetkili erişim testi:** HumanResources şemasındaki bir tabloya (örneğin Employee) SELECT sorgusu gönderdim, veri başarılı şekilde geldi.

The screenshot shows the DataGrip interface with a query editor and an output viewer.

**Query Editor:**

```
1 ✓ SELECT * FROM HumanResources.Employee
```

**Output Viewer:**

	BusinessEntityID	NationalIDNumber	LoginID	OrganizationNode	OrganizationLevel	JobTitle
1	1	295847284	adventure-works\ken0	<null>	<null>	Chief Executive Officer
2	2	245797967	adventure-works\terri0	0x58	1	Vice President of Engineering
3	3	509647174	adventure-works\roberto0	0x5AC0	2	Engineering Manager
4	4	112457891	adventure-works\rob0	0x5AD6	3	Senior Tool Designer
5	5	695256908	adventure-works\gail0	0x5ADA	3	Design Engineer
6	6	99832692	adventure-works\jossef0	0x5ADE	3	Design Engineer
7	7	134969118	adventure-works\dylan0	0x5AE1	3	Research and Development Manager
8	8	811994146	adventure-works\diane1	0x5AE158	4	Research and Development Engineer
9	9	658797903	adventure-works\gigi0	0x5AE168	4	Research and Development Engineer
10	10	879342154	adventure-works\michael6	0x5AE178	4	Research and Development Manager
11	11	974026903	adventure-works\ovidiu0	0x5AE3	3	Senior Tool Designer
12	12	480168528	adventure-works\thierry0	0x5AE358	4	Tool Designer
13	13	486228782	adventure-works\janice0	0x5AE368	4	Tool Designer
14	14	42487730	adventure-works\michael8	0x5AE5	3	Senior Design Engineer
15	15	5690285	adventure-works\sharon0	0x5AE7	3	Design Engineer
16	16	24756624	adventure-works\david0	0x68	1	Marketing Manager
17	17	253022876	adventure-works\kevin0	0x6AC0	2	Marketing Assistant
18	18	222969461	adventure-works\john5	0x6B40	2	Marketing Specialist
19	19	52541318	adventure-works\mary2	0x6BC0	2	Marketing Assistant
20	20	323403273	adventure-works\wanida0	0x6C20	2	Marketing Assistant
21	21	243522160	adventure-works\ter	290 rows	2	Marketing Specialist
22		95958330	adventure-works\sarayeu	0x6CD0	2	Marketing Specialist

2. **Yetkisiz erişim denemesi (okuma):** Sales şemasındaki bir tabloya sorgu gönderdim. “SELECT permission was denied...” hatasıyla karşılaştım.

The screenshot shows the DataGrip interface with a query editor and an error message.

**Query Editor:**

```
1 ❌ SELECT * FROM Sales.SalesOrderHeader
```

**Error Message:**

[S0005][229] Line 1: The SELECT permission was denied on the object 'SalesOrderHeader', database 'AdventureWorks', schema 'Sales'.

3. **Yetkisiz erişim denemesi (yazma):** HumanResources.Employee tablosuna INSERT ya da UPDATE sorgusu göndermemeyi denedim. Beklendiği gibi işlem reddedildi.

The screenshot shows a dark-themed SSMS interface. In the top-left corner, there's a status bar with '1 1' and a warning icon. The main area contains a single line of SQL code: 'UPDATE HumanResources.Department SET Name = 'Test' WHERE DepartmentID = 1;'. In the bottom-right corner of the main window, there are three small icons: a red exclamation mark, a yellow triangle, and a green checkmark. Below the main window, a dark red status bar displays the error message: '[S0005][229] Line 1: The UPDATE permission was denied on the object 'Department', database 'AdventureWorks', schema 'HumanResources'. To the right of the error message are three buttons: 'Explain with AI', 'Fix with AI', and a close button ('X').

Bu testler, kullanıcıya verdığım yetkilerin yalnızca gereken işlemlerle sınırlı olduğunu gösterdi.

## SQL Injection Zayıflığı ve Korunma Yöntemleri

### 1. Giriş

Bu bölümde, SQL Injection zayıflığının ne olduğunu ve bu tür saldırırlara karşı nasıl önlem alınabileceğini örneklerle ele alıyorum. SQL Injection, kullanıcıdan alınan verilerin yeterince kontrol edilmemesi durumunda kötü niyetli kişilerin sisteme yetkisiz SQL komutları çalıştırmasına olanak tanıyan ciddi bir güvenlik açığıdır.

### 2. Zayıflı Senaryo Oluşturulması

AdventureWorks veritabanında HumanResources.Employee tablosu üzerinde, JobTitle'a göre filtreleme yapan zayıflı bir stored procedure yazdım. Bu prosedürde kullanıcı girdisi doğrudan sorguya eklentiği için injection'a açık hale geliyor. Kodda string birleştirme yöntemiyle SQL komutu oluşturuluyor.

```
1 ✓ USE AdventureWorks;
2 GO
3
4 -- Eğer prosedür zaten varsa, önce sil (tekrar tekrar çalıştırabilmek için)
5 ✓ IF OBJECT_ID('dbo.usp_GetEmployeeByJobTitle_Vulnerable', 'P') IS NOT NULL
6     DROP PROCEDURE dbo.usp_GetEmployeeByJobTitle_Vulnerable;
7 GO
8
9 ✓ CREATE PROCEDURE dbo.usp_GetEmployeeByJobTitle_Vulnerable
10    @JobTitle NVARCHAR(50)
11 AS
12 BEGIN
13     DECLARE @SQL NVARCHAR(MAX);
14
15     -- Kullanıcı girdisi (@JobTitle) doğrudan sorguya ekleniyor - ZAFİYET BURADA
16     SET @SQL = N'SELECT BusinessEntityID, NationalIDNumber, LoginID, JobTitle
17         FROM HumanResources.Employee
18         WHERE JobTitle LIKE ''%'' + @JobTitle + ''%''';
19
20     PRINT N'Çalıştırılacak Güvensiz Sorgu: ' + @SQL; -- Çalıştırılacak sorguyu görmek için
21
22     EXEC sp_executesql @SQL; -- Dinamik SQL'i çalıştır
23 END
24 GO
25 |
```

```
AdventureWorks> USE AdventureWorks
[2025-04-21 14:47:40] [S0001][5701] Changed database context to 'AdventureWorks'.
[2025-04-21 14:47:40] completed in 6 ms
AdventureWorks> IF OBJECT_ID('dbo.usp_GetEmployeeByJobTitle_Vulnerable', 'P') IS NOT NULL
    DROP PROCEDURE dbo.usp_GetEmployeeByJobTitle_Vulnerable;
[2025-04-21 14:47:40] completed in 6 ms
AdventureWorks> CREATE PROCEDURE dbo.usp_GetEmployeeByJobTitle_Vulnerable
    @JobTitle NVARCHAR(50)
    AS
    BEGIN
        DECLARE @SQL NVARCHAR(MAX);

        -- Kullanıcı girdisi (@JobTitle) doğrudan sorguya ekleniyor - ZAFİYET BURADA
        SET @SQL = N'SELECT BusinessEntityID, NationalIDNumber, LoginID, JobTitle
            FROM HumanResources.Employee
            WHERE JobTitle LIKE ''%'' + @JobTitle + ''%''';

        PRINT N'Çalıştırılacak Güvensiz Sorgu: ' + @SQL; -- Çalıştırılacak sorguyu görmek için

        EXEC sp_executesql @SQL; -- Dinamik SQL'i çalıştır
    END
[2025-04-21 14:47:40] completed in 33 ms
```

### 3. Zafiyetli Senaryonun Test Edilmesi (Normal Kullanım)

İlk olarak prosedürü ‘Manager’ gibi geçerli bir girdiye göre çalışırdım. Beklendiği gibi sadece bu unvana sahip çalışanlar listelendi. Aynı zamanda PRINT komutu ile oluşan SQL sorgusunu da gözlemedim.

The screenshot shows a SQL query window and its results. The query is:

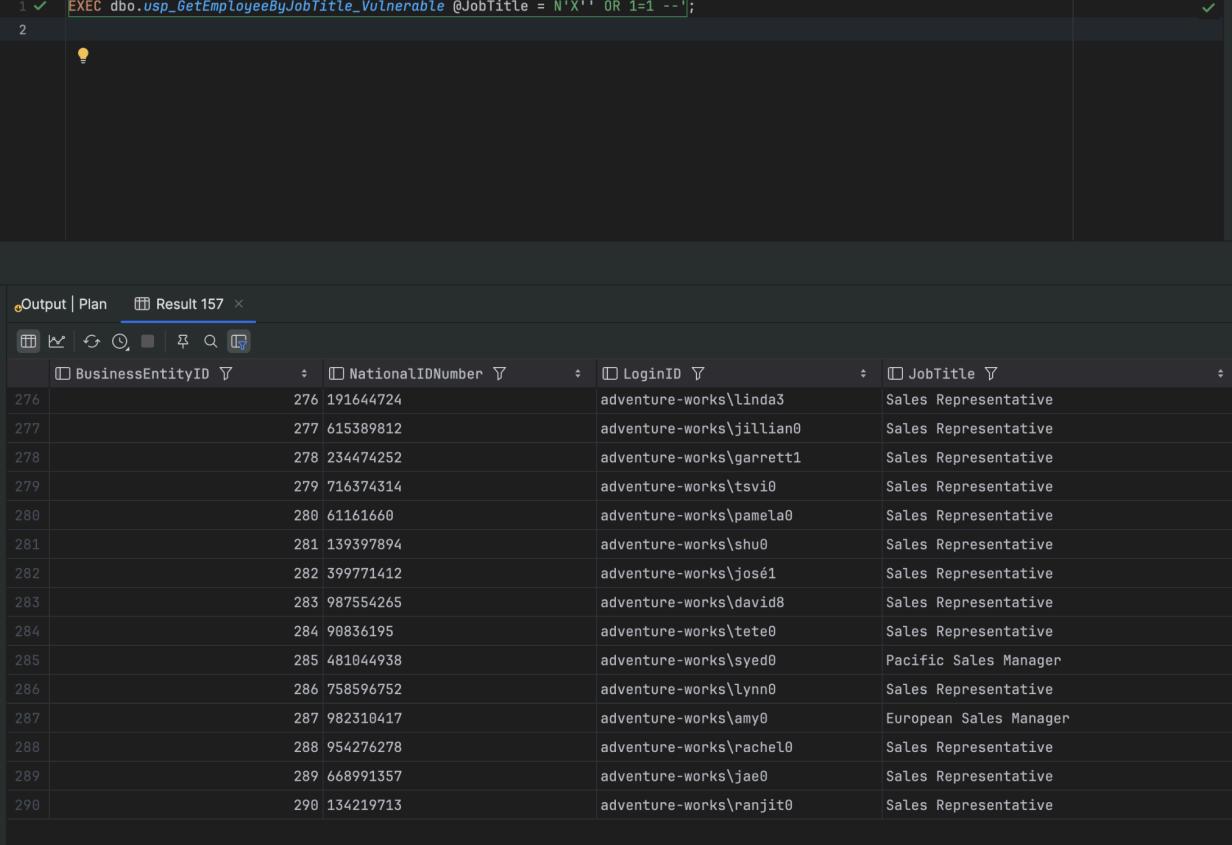
```
1 ✓ EXEC dbo.usp_GetEmployeeByJobTitle_Vulnerable @JobTitle = N'Manager';
2
```

The results tab is selected, showing 156 rows of data. The columns are:

BusinessEntityID	NationalIDNumber	LoginID	JobTitle
1	3 509647174	adventure-works\roberto0	Engineering Manager
2	7 134969118	adventure-works\dylan0	Research and Development Manager
3	10 879342154	adventure-works\michael0	Research and Development Manager
4	16 24756624	adventure-works\david0	Marketing Manager
5	26 277173473	adventure-works\peter0	Production Control Manager
6	211 398223854	adventure-works\hazem0	Quality Assurance Manager
7	217 345106466	adventure-works\zainal0	Document Control Manager
8	227 141165819	adventure-works\gary1	Facilities Manager
9	235 535145551	adventure-works\paula0	Human Resources Manager
10	241 30845	adventure-works\david0	Accounts Manager
11	249 121491555	adventure-works\wendy0	Finance Manager
12	250 895209680	adventure-works\sheela0	Purchasing Manager
13	263 441044382	adventure-works\jean0	Information Services Manager
14	264 858323870	adventure-works\stephanie0	Network Manager
15	274 502097814	adventure-works\stephen0	North American Sales Manager
16	285 481044938	adventure-works\syed0	Pacific Sales Manager
17	287 982310417	adventure-works\amy0	European Sales Manager

#### 4. SQL Injection Denemesi

Ardından, 'X' OR 1=1 -- şeklinde bilindik bir injection girdisi ile prosedürü tekrar çalışırdım. Bu sefer WHERE koşulu geçersiz hale geldi ve tüm kayıtlar listelendi. PRINT çıktısındaki sorgu da bu yapıyı açıkça gösteriyordu. OR 1=1 ifadesi koşulu sürekli doğruya çevirdi, -- ise sorgunun kalanını yorum satırına aldı.

```
1 ✓ EXEC dbo.usp_GetEmployeeByJobTitle_Vulnerable @JobTitle = N'X'' OR 1=1 --';  
2  
  
eOutput | Plan | Result 157 ×  


|     | BusinessEntityID | NationalIDNumber | LoginID                  | JobTitle               |
|-----|------------------|------------------|--------------------------|------------------------|
| 276 | 276              | 191644724        | adventure-works\linda3   | Sales Representative   |
| 277 | 277              | 615389812        | adventure-works\jillian0 | Sales Representative   |
| 278 | 278              | 234474252        | adventure-works\garrett1 | Sales Representative   |
| 279 | 279              | 716374314        | adventure-works\tsvi0    | Sales Representative   |
| 280 | 280              | 61161660         | adventure-works\pamela0  | Sales Representative   |
| 281 | 281              | 139397894        | adventure-works\shu0     | Sales Representative   |
| 282 | 282              | 399771412        | adventure-works\josé1    | Sales Representative   |
| 283 | 283              | 987554265        | adventure-works\david8   | Sales Representative   |
| 284 | 284              | 90836195         | adventure-works\tete0    | Sales Representative   |
| 285 | 285              | 481044938        | adventure-works\syed0    | Pacific Sales Manager  |
| 286 | 286              | 758596752        | adventure-works\lynn0    | Sales Representative   |
| 287 | 287              | 982310417        | adventure-works\amy0     | European Sales Manager |
| 288 | 288              | 954276278        | adventure-works\rachel0  | Sales Representative   |
| 289 | 289              | 668991357        | adventure-works\jae0     | Sales Representative   |
| 290 | 290              | 134219713        | adventure-works\ranjit0  | Sales Representative   |


```

## 5. Güvenli Yöntem: Parametreli Soru

Aynı işlevi gören ancak injection'a karşı güvenli bir stored procedure yazdım. Bu versiyonda kullanıcı girdisi SQL metnine doğrudan dahil edilmek yerine, bir parametre olarak işlendi. Böylece SQL Server bu veriyi yalnızca veri olarak ele aldı, kod olarak değil.

```
1 ✓ USE AdventureWorks;
2 GO
3
4 ✓ IF OBJECT_ID('dbo.usp_GetEmployeeByJobTitle_Secure', 'P') IS NOT NULL
5     DROP PROCEDURE dbo.usp_GetEmployeeByJobTitle_Secure;
6 GO
7
8 ✓ CREATE PROCEDURE dbo.usp_GetEmployeeByJobTitle_Secure
9     @JobTitle NVARCHAR(50)
10    AS
11    BEGIN
12        -- Parametre doğrudan WHERE içinde kullanılıyor, SQL Injection riski yok!
13        -- SQL Server, @JobTitle içindeki özel karakterleri SQL kodu olarak yorumlamaz.
14        SELECT BusinessEntityID, NationalIDNumber, LoginID, JobTitle
15        FROM HumanResources.Employee
16        WHERE JobTitle LIKE '%' + @JobTitle + '%';
17    END
18 GO
19
```

Output | Plan | Result 157

Output Plan Result 157

```
DROP PROCEDURE dbo.usp_GetEmployeeByJobTitle_Secure;
[2025-04-21 14:51:11] completed in 6 ms
AdventureWorks> CREATE PROCEDURE dbo.usp_GetEmployeeByJobTitle_Secure
    @JobTitle NVARCHAR(50)
AS
BEGIN
    -- Parametre doğrudan WHERE içinde kullanılıyor, SQL Injection riski yok!
    -- SQL Server, @JobTitle içindeki özel karakterleri SQL kodu olarak yorumlamaz.
    SELECT BusinessEntityID, NationalIDNumber, LoginID, JobTitle
    FROM HumanResources.Employee
    WHERE JobTitle LIKE '%' + @JobTitle + '%';
END
[2025-04-21 14:51:11] completed in 30 ms
```

## 6. Güvenli Prosedürle Test

Bu güvenli prosedürü de yine aynı zararlı girdiyle test ettim. Sonuç olarak sistem bu girdiyi kod gibi değil, doğrudan arama kriteri olarak değerlendirdi ve doğal olarak eşleşen kayıt bulunamadı. Beklendiği gibi, veri sizıntısı yaşanmadı.

The screenshot shows a SQL playground interface with the following details:

- Top Bar:** Includes icons for file, save, undo, redo, and a dropdown menu labeled "Playground".
- Query Editor:** Displays the following SQL code:

```
1 ✓ EXEC dbo.usp_GetEmployeeByJobTitle_Secure @JobTitle = N'X'' OR 1=1 --';
```
- Output Tab:** Labeled "Result 159" and shows the following schema:

BusinessEntityID	NationalIDNumber	LoginID	JobTitle
------------------	------------------	---------	----------

Bu bölümde, SQL Injection'ın ne kadar tehlikeli olabileceğini ve bu riskin nasıl önlenebileceğini net şekilde göstermiş oldum. Uygulama geliştirirken kullanıcıdan alınan her veriye kuşkuyla yaklaşmak gerekiyor. Parametreli sorgular ya da benzer güvenlik önlemleri bu tür saldırılara karşı en temel savunma hattını oluşturuyor.

Mustafa Girgin 21290649

<https://github.com/antelcha/sqlprojeler>

# Veritabanı Replikasyonu ve Dağıtık Yapı Kurulumu

SQL Server üzerinde iki farklı instance arasında veritabanı replikasyonu kurulumunu adım adım gerçekleştirdim. Buradaki amacım, bir Publisher (Yayınçı) ve bir Subscriber (Abone) arasında veri senkronizasyonunu sağlamaktı.

Docker Container'larının Hazırlanması İlk olarak, iki ayrı SQL Server instance'ını Docker üzerinde çalıştırmak için gerekli yapılandırmayı yaptım. Docker Compose dosyasında her iki sunucu için SQL Server Agent'ı aktif ettim ve gerekli port yönlendirmelerini tanımladım.  
[photo]

Publisher Sunucusunun Yapılandırılması Publisher rolündeki ilk SQL Server instance'ında (sql\_server\_1) sırasıyla şu adımları uyguladım:

a) Öncelikle distributor kurulumunu gerçekleştirdim ve distribution database'ini oluşturdum:

Subscriber Sunucusunun Yapılandırılması İkinci SQL Server instance'ında (sql\_server\_2) subscription ayarlarını yaptım:

a) Önce subscriber veritabanını oluşturdum:

b) Push subscription tanımlamasını gerçekleştirdim:

Subscriber'da replikasyon kontrolü: Eklenen verilerin subscriber'a başarıyla replike olduğunu gözlemledim.

Bu testler, replikasyon yapılandırmasının beklentiği gibi çalıştığını gösterdi. Veriler publisher'dan subscriber'a sorunsuz şekilde aktarıldı.

Tüm bu adımları otomatize etmek için üç ayrı SQL script hazırladım:

1-publisher-setup.sql: Publisher kurulum adımları

2-subscriber-setup.sql: Subscriber kurulum adımları

3-test-replication.sql: Test senaryoları

Bu scriptler sayesinde replikasyon kurulumunu hızlı ve hatasız şekilde tekrarlayabiliyorum.

Herhangi bir sorun yaşandığında SQL Server error log'larını kontrol ederek hatanın kaynağını kolayca tespit edebiliyorum.

```
-- 1. PUBLISHER SETUP (sql_server_1'de çalıştırılacak)
USE master;
GO
```

```
-- Mevcut distributor yapılandırmasını temizle
EXEC sp_dropdistributor @no_checks = 1, @ignore_distributor = 1;
GO

-- Replikasyon dağıticısını yapılandırır
EXEC sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO

-- Distribution database'ini oluştur
USE master;
GO
IF NOT EXISTS (SELECT * FROM sys.databases WHERE name = 'distribution')
BEGIN
    -- Local distributor olarak ayarla
    EXEC sp_adddistributor
        @distributor = @@SERVERNAME,
        @password = 'Distr!bution2024';

    -- Distribution database'ini oluştur
    EXEC sp_adddistributiondb
        @database = 'distribution';
END
GO

-- Test veritabanını oluştur
USE master;
GO
IF EXISTS (SELECT * FROM sys.databases WHERE name = 'TestDB')
    DROP DATABASE TestDB;
GO
CREATE DATABASE TestDB;
GO

-- Veritabanını replikasyon için hazırla
USE TestDB;
GO
EXEC sp_replicationdboption
    @dbname = 'TestDB',
    @optname = 'publish',
    @value = 'true';
GO
```

```
-- Test tablosunu oluştur
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY IDENTITY(1,1),
    CustomerName NVARCHAR(100),
    CreatedDate DATETIME DEFAULT GETDATE()
);
GO

-- Yayıncı sunucuyu yapılandırır
USE [TestDB]
GO
EXEC sp_addpublishation
    @publication = 'CustomerPublication',
    @description = 'Publication for Customers table',
    @sync_method = 'concurrent',
    @retention = 0,
    @allow_push = 'true',
    @allow_pull = 'true',
    @allow_anonymous = 'true',
    @enabled_for_internet = 'false',
    @snapshot_in_defaultfolder = 'true';
GO

-- Yayın için makale ekle
USE [TestDB]
GO
EXEC sp_addarticle
    @publication = 'CustomerPublication',
    @article = 'Customers',
    @source_owner = 'dbo',
    @source_object = 'Customers',
    @type = 'logbased',
    @description = 'Customer table article',
    @creation_script = NULL,
    @pre_creation_cmd = 'drop',
    @schema_option = 0x000000000803509D,
    @identityrangemanagementoption = 'manual',
    @destination_table = 'Customers',
    @destination_owner = 'dbo';
GO

-- Snapshot Agent'ı başlat
EXEC sp_startpublication_snapshot @publication = 'CustomerPublication';
GO
```

```

-- 2. SUBSCRIBER SETUP (sql_server_2'de çalıştırılacak)
USE master;
GO

-- Eğer varsa subscriber database'i sil
IF EXISTS (SELECT * FROM sys.databases WHERE name = 'TestDB')
    DROP DATABASE TestDB;
GO

-- Subscriber database'i oluştur
CREATE DATABASE TestDB;
GO

-- Subscription oluştur
USE TestDB;
GO

-- Push subscription oluştur
EXEC sp_addsubscription
    @publication = 'CustomerPublication',
    @subscriber = 'sql_server_2',
    @destination_db = 'TestDB',
    @subscription_type = 'Push',
    @sync_type = 'automatic',
    @article = 'all',
    @update_mode = 'read only';
GO

-- Push subscription agent'i oluştur
EXEC sp_addpushsubscription_agent
    @publication = 'CustomerPublication',
    @subscriber = 'sql_server_2',
    @subscriber_db = 'TestDB',
    @subscriber_security_mode = 0,
    @subscriber_login = 'sa',
    @subscriber_password = 'yourStrong(!)Password';
GO

```

```
-- 3. TEST REPLICATION
```

```
-- Publisher'da çalıştırılacak (sql_server_1):
USE TestDB;
GO

-- Test verisi ekle
INSERT INTO Customers (CustomerName)
VALUES
    ('Test Customer 1'),
    ('Test Customer 2'),
    ('Test Customer 3');
GO

-- Eklenen verileri kontrol et
SELECT * FROM Customers;
GO

-- Subscriber'da çalıştırılacak (sql_server_2):
USE TestDB;
GO

-- Replike olan verileri kontrol et
SELECT * FROM Customers;
GO

-- Not: Veriler subscriber'a replike olduysa,
-- publisher'da eklenen verilerin aynısını subscriber'da görmelisiniz.
```

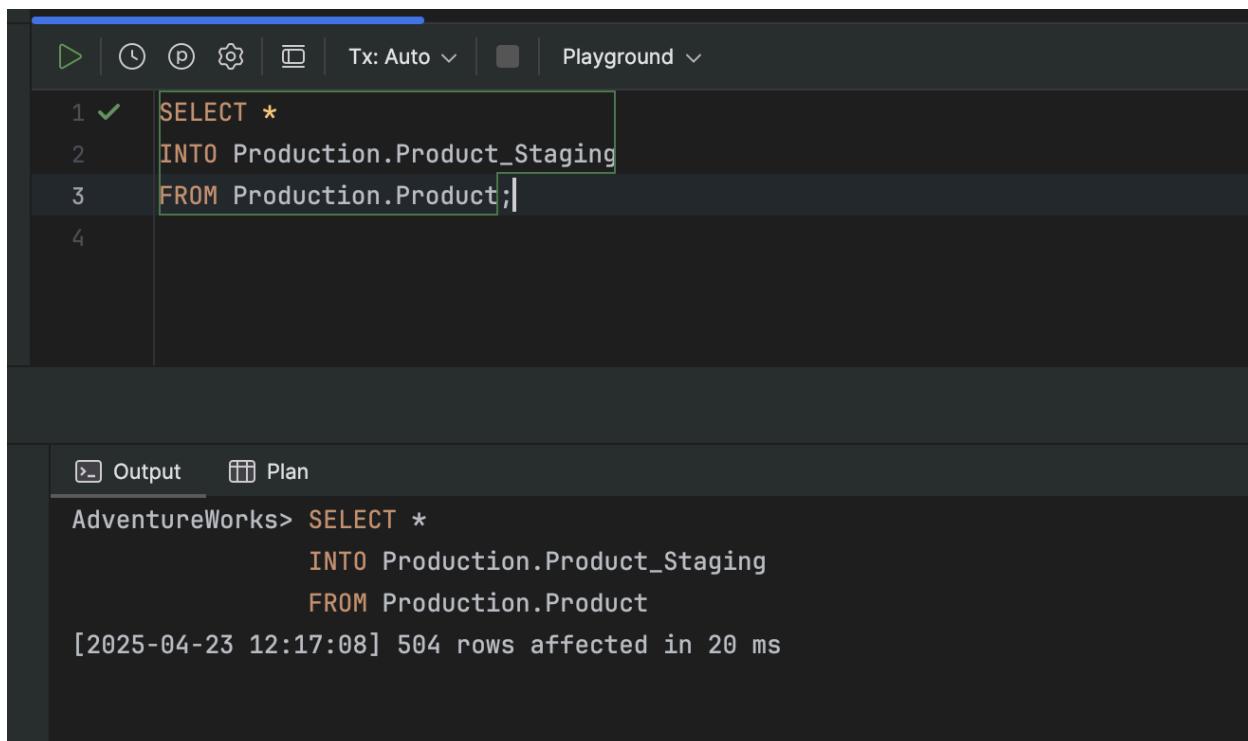
Mustafa Girgin  
21290649

# Veri Temizleme ve ETL Süreci Tasarımı

AdventureWorks veritabanındaki `Production.Product` tablosu üzerinde veri temizleme süreci tasarlandı. Bu süreçte, bazı veriler kasıtlı olarak bozulduktan sonra, yalnızca anlamlı değerlere sahip kayıtlar seçilerek yeni bir tabloya aktarıldı. Böylece hatalı veriler devre dışı bırakılmış oldu.

## 1. Staging Tablosunun Oluşturulması

İlk adım olarak, orijinal `Production.Product` tablosunun bir kopyası alındı. Böylece işlemler doğrudan ana tablo üzerinde değil, bu kopya üzerinde yapıldı.



The screenshot shows a database interface with a query editor and a results pane. The query editor contains the following SQL code:

```
1 ✓ SELECT *
2 INTO Production.Product_Staging
3 FROM Production.Product;
```

The results pane shows the execution of the query:

```
AdventureWorks> SELECT *
    INTO Production.Product_Staging
    FROM Production.Product
[2025-04-23 12:17:08] 504 rows affected in 20 ms
```

## 2. Veri Bozulması (Simülasyon Amaçlı)

Temizleme işlemlerinin uygulanabilmesi için, bazı veriler kasıtlı olarak geçersiz hale getirildi.

```
7 UPDATE Production.Product_Staging
8 SET SellStartDate = '1900-01-01'
9 WHERE ProductID % 4 = 0;
10

Output Plan
AdventureWorks> SELECT *
INTO Production.Product_Staging
FROM Production.Product
[2025-04-23 12:17:08] 504 rows affected in 20 ms
AdventureWorks> -- Fiyat alanı negatif değerlere çekildi
UPDATE Production.Product_Staging
SET ListPrice = -99.99
WHERE ProductID % 10 = 0;

-- Satış başlangıç tarihi anlamını yitirecek şekilde geriye alındı
UPDATE Production.Product_Staging
SET SellStartDate = '1900-01-01'
WHERE ProductID % 4 = 0;
[2025-04-23 12:17:49] 177 rows affected in 43 ms
```

Bu işlemler sonucunda, belirli satırlardaki fiyatlar negatif hale geldi ve bazı tarihler anlamsız derecede eskiye çekildi.

### 3. Verilerin Temizlenmesi (Bozuk Verilerin Dışlanması)

Temiz bir veri kümesi elde etmek amacıyla, yalnızca anlamlı fiyat ve tarih bilgilerine sahip kayıtlar seçildi.

```
1 ✓ SELECT *
2   INTO Production.Product_Cleaned
3   FROM Production.Product_Staging
4   WHERE ListPrice >= 0
5     AND SellStartDate >= '2000-01-01';
6
```

Output Plan

AdventureWorks> SELECT \*

```
    INTO Production.Product_Cleaned
    FROM Production.Product_Staging
    WHERE ListPrice >= 0
          AND SellStartDate >= '2000-01-01'
```

[2025-04-23 12:19:20] 351 rows affected in 41 ms

#### 4. Temizlik İşleminin Değerlendirilmesi

Temizleme işleminden sonra, kaç kaydın devre dışı bırakıldığını görmek için bir özet sorgu çalıştırıldı:

```
1 ✓ SELECT
2     COUNT(*) AS TotalRecords,
3     SUM(CASE WHEN ListPrice < 0 THEN 1 ELSE 0 END) AS BozukFiyatSayisi,
4     SUM(CASE WHEN SellStartDate < '2000-01-01' THEN 1 ELSE 0 END) AS GecersizTarihSayisi
5 FROM Production.Product_Staging;
6
```

Output | Plan Result 453 ×

	TotalRecords	BozukFiyatSayisi	GecersizTarihSayisi
1	504	50	127

```
1 ✓ SELECT
2     COUNT(*) AS TotalRecords,
3     SUM(CASE WHEN ListPrice < 0 THEN 1 ELSE 0 END) AS BozukFiyatSayisi,
4     SUM(CASE WHEN SellStartDate < '2000-01-01' THEN 1 ELSE 0 END) AS GecersizTarihSayisi
5 FROM Production.Product_Cleaned;
6
```

Output | Plan Result 454 ×

	TotalRecords	BozukFiyatSayisi	GecersizTarihSayisi
1	351	0	0

Bu sonuçlar, veri temizliğinin etkisini ve ne kadar kayıt dışlandığını sayısal olarak göstermektedir.

## Farklı Kaynaklardan Gelen Verilerin Dönüşürtlmesi ve Birleştirilmesi

Bu aşamada, iki farklı veri kaynağından gelen personel bilgileri üzerinde veri dönüştürme işlemleri gerçekleştirilmiş ve veriler ortak bir formata getirilerek tek bir yapıda birleştirilmiştir. Bu

İşlem, sistemler arası veri entegrasyonu örneği olarak tasarlanmış ve ETL sürecinin uygulamalı bir adımı olarak gerçekleştirılmıştır.

Auto ▾ Playground ▾

### Import

from: personel\_A.csv

to: personel\_A master.dbo [@localhost]

Schema: dbo master [localhost]

Table: personel\_A

New: auto >

#### Mapping

to: Column	to: Type	from: Column
ad_soyad	VARCHAR(MAX)	ad_soyad
yas	int	<Auto> (yas)
cinsiyet	VARCHAR(MAX)	cinsiyet
maas	int	<Auto> (maas)

Write errors to file /private/var/folders/q5/y3y5nv8j4j7gvdq74j8qv

Insert convertible values as null

Disable indexes and triggers, lock table (may be faster)

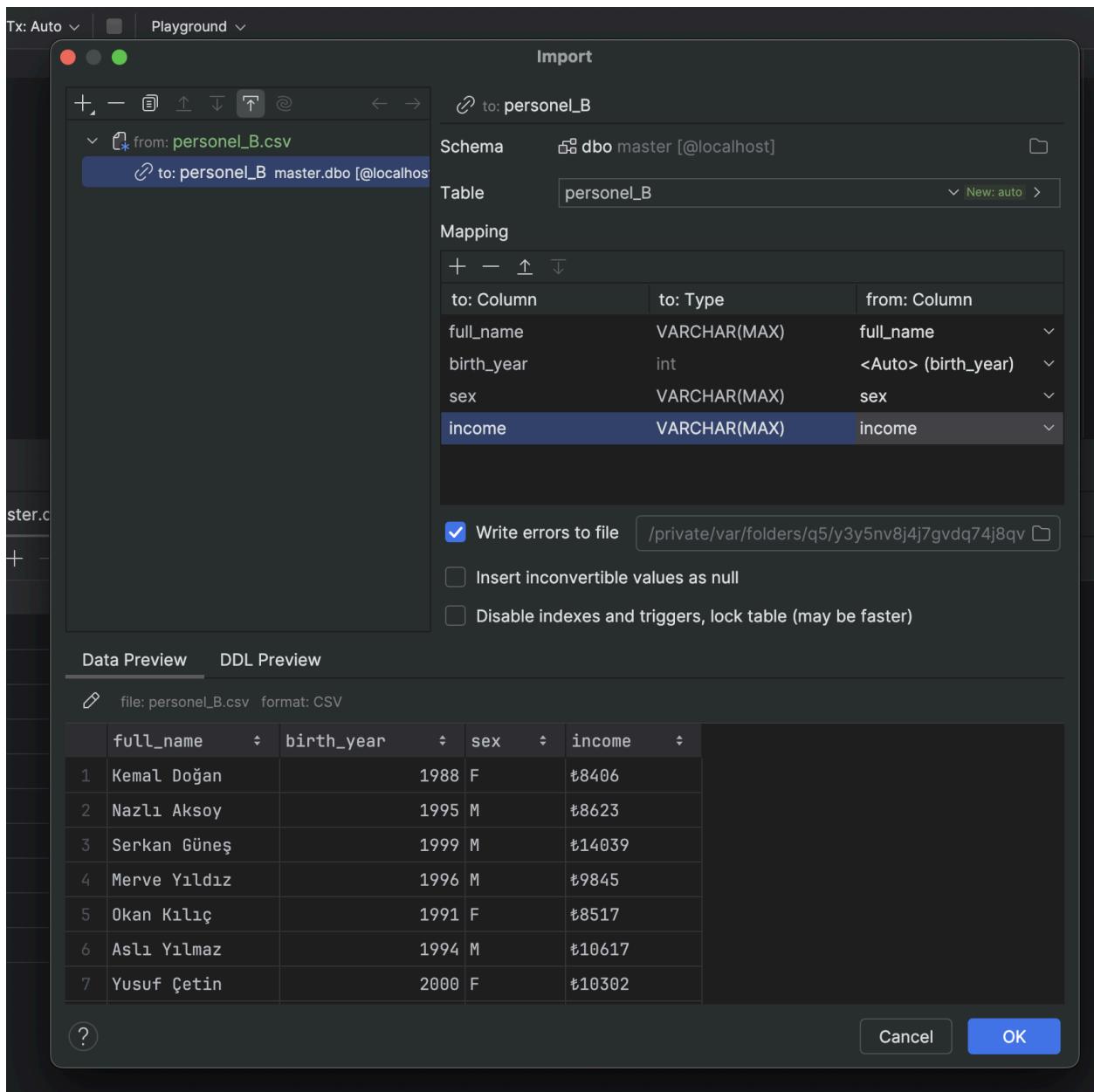
Data Preview DDL Preview

file: personel\_A.csv format: CSV

	ad_soyad	yas	cinsiyet	maas
1	Ali Yılmaz	29	erkek	11528
2	Ayşe Demir	39	erkek	14012
3	Mehmet Koç	31	kadın	11792
4	Elif Yalçın	37	kadın	8919
5	Can Kaya	35	kadın	9947
6	Zeynep Şahin	33	erkek	10763
7	Burak Arslan	26	kadın	10844

?

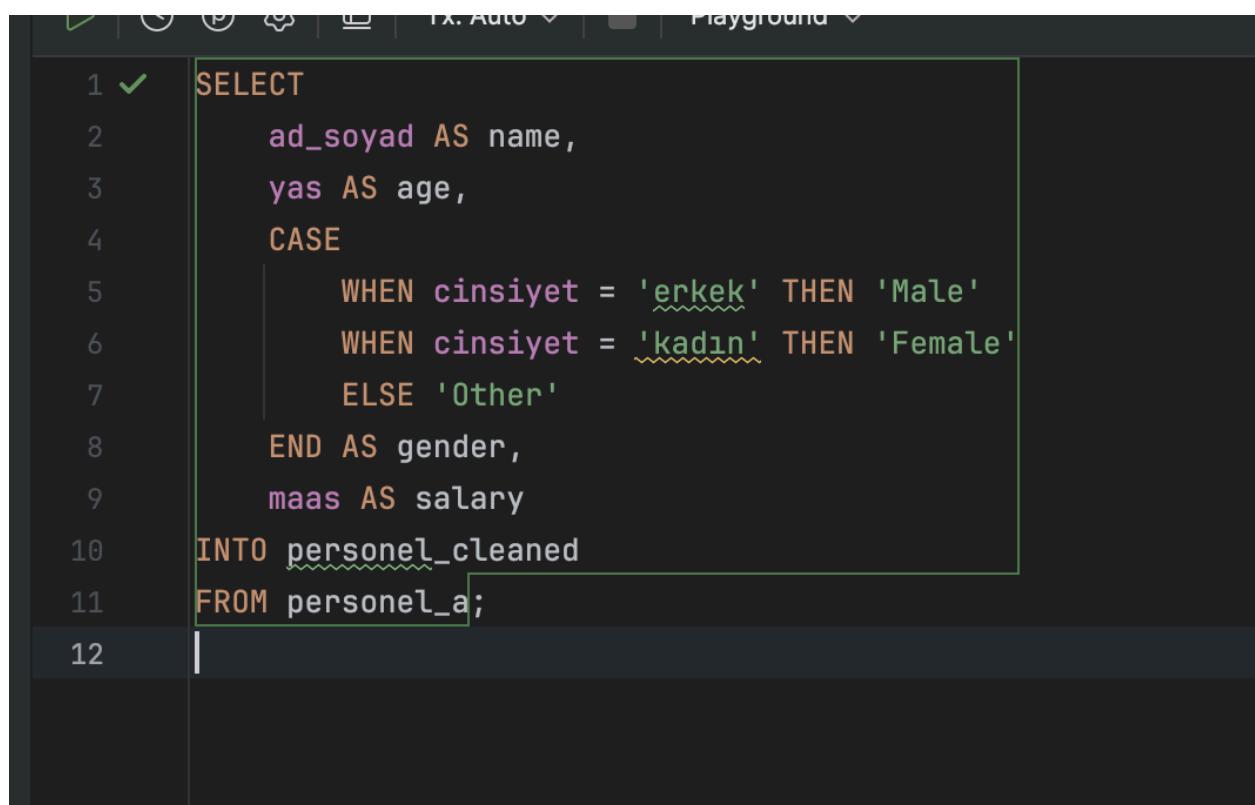
Cancel OK



İki farklı sistemden gelen veriler personnel\_A ve personnel\_B tablolarına aktarıldı.

## Verilerin dönüştürülmesi

İki tablo arasındaki kolon isimleri ve veri tipleri farklı olduğundan, veriler ortak bir yapıya getirilmeden önce dönüştürme işlemleri uygulanmıştır. Amaç, iki kaynaktan gelen bilgilerin tek bir birleşik tablo altında birleştirilmesini mümkün kılmaktır.



The screenshot shows a PostgreSQL playground interface with a dark theme. The code editor contains the following SQL query:

```
1 ✓ SELECT
2     ad_soyad AS name,
3     yas AS age,
4     CASE
5         WHEN cinsiyet = 'erkek' THEN 'Male'
6         WHEN cinsiyet = 'kadın' THEN 'Female'
7         ELSE 'Other'
8     END AS gender,
9     maas AS salary
10    INTO personel_cleaned
11   FROM personel_a;
```

The code is highlighted with syntax coloring. Line 1 has a green checkmark icon. The entire code block is enclosed in a light green rectangular box. The top bar of the interface includes icons for file operations, a transaction status (Tx. Auto), and a dropdown labeled "Playground".

```
1 ✓ INSERT INTO personel_cleaned (name, age, gender, salary)
2 SELECT
3     full_name AS name,
4     YEAR(GETDATE()) - birth_year AS age,
5     CASE
6         WHEN sex = 'M' THEN 'Male'
7         WHEN sex = 'F' THEN 'Female'
8         ELSE 'Other'
9     END AS gender,
10    REPLACE(REPLACE(income, '₺', ''), '.', '') AS salary
11 FROM personel_b;
12
```

Bu adım sonucunda, farklı formatlara ve veri yapılarına sahip iki kaynaktan alınan veriler tek bir biçimde dönüştürülmüş ve analiz için uygun olan `personel_cleaned` adlı tabloya aktarılmıştır.

```
1 ✓ SELECT * FROM personnel_cleaned;
```

	name	age	gender	salary
1	Ali Yilmaz	29	Male	11528
2	Ayse Demir	39	Male	14012
3	Mehmet Koç	31	Female	11792
4	Elif Yalçın	37	Female	8919
5	Can Kaya	35	Female	9947
6	Zeynep Sahin	33	Male	10763
7	Burak Arslan	26	Female	10844
8	Sena Kurt	41	Male	8077
9	Ahmet Tunç	40	Female	9585
10	Fatma Aydin	37	Female	13806
11	Kemal Dogan	37	Female	8406
12	Nazli Aksoy	30	Male	8623
13	Serkan Günes	26	Male	14039
14	Merve Yildiz	29	Male	9845
15	Okan Kılıç	34	Female	8517
16	Asli Yilmaz	31	Male	10617
17	Yusuf Çetin	25	Female	10302
18	Buse Korkmaz	41	Male	10877
19	Cemre Öztürk	41	Female	9816
20	Eren Dursun	42	Male	11283

Mustafa Girgin  
21290649

<https://github.com/antelcha/sqlprojeler>

# Veritabanı Versiyon Yönetimi ve Sürüm Kontrolü

## 1. İlk Kurulum ve Temel Yapılandırma

VersionControlDB veritabanı üzerinde temel versiyon yönetimi yapısını adım adım kurdum. Buradaki hedefim, veritabanı değişikliklerini kontrollü ve güvenli bir şekilde yönetmektı.

İlk olarak, `1\_initial\_setup.sql` dosyasında veritabanımızı ve temel tablolarımızı oluşturuyoruz.

Bu aşamada:

- VersionControlDB adında yeni bir veritabanı oluşturuyoruz
- DatabaseVersion tablosunu oluşturuyoruz
- İlk versiyon kaydını (1.0.0) ekliyoruz
- Örnek bir Customers tablosu oluşturuyoruz

```

1  -- Veritabanı oluşturma
2  IF NOT EXISTS (SELECT * FROM sys.databases WHERE name = 'VersionControlDB')
3  BEGIN
4      CREATE DATABASE VersionControlDB;
5  END
6  GO
7
8  USE VersionControlDB;
9  GO
10
11 -- Versiyon kontrol tablosu
12 BEGIN TRANSACTION;
13     IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'DatabaseVersion')
14     BEGIN
15         CREATE TABLE DatabaseVersion (
16             VersionID INT PRIMARY KEY IDENTITY(1,1),
17             VersionNumber VARCHAR(20) NOT NULL,
18             AppliedDate DATETIME DEFAULT GETDATE(),
19             Description NVARCHAR(500),
20             Status VARCHAR(50)
21     );
22
23     -- İlk versiyon kaydı
24     INSERT INTO DatabaseVersion (VersionNumber, Description, Status)
25     VALUES ('1.0.0', 'Initial database version', 'Completed');
26
27
28 -- Örnek tablo oluşturma
29 IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'Customers')
30     BEGIN
31         CREATE TABLE Customers (
32             CustomerID INT PRIMARY KEY IDENTITY(1,1),
33             FirstName NVARCHAR(50),
34             LastName NVARCHAR(50),
35             Email NVARCHAR(100)
36     );
37
38     END
39 COMMIT TRANSACTION; |

```

## 2. Şema Değişikliklerini İzleme Sistemi

`2\_schema\_tracking.sql` dosyasında, veritabanında yapılan tüm değişiklikleri otomatik olarak izleyen bir sistem kuruyoruz. Bu sistem:

- SchemaChanges tablosunu oluşturuyor
- DDL Trigger ekliyor
- Tüm şema değişikliklerini kaydediyor

```
USE VersionControlDB;
GO

-- Şema değişikliklerini izlemek için tablo oluşturma
IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'SchemaChanges')
BEGIN
    CREATE TABLE SchemaChanges (
        ChangeID INT PRIMARY KEY IDENTITY(1,1),
        EventType NVARCHAR(100),
        ObjectName NVARCHAR(256),
        ObjectType NVARCHAR(100),
        SQLCommand NVARCHAR(MAX),
        LoginName NVARCHAR(256),
        ChangeDate DATETIME DEFAULT GETDATE()
    );
END
GO

-- DDL Trigger oluşturma
IF NOT EXISTS (SELECT * FROM sys.triggers WHERE name = 'TR_TrackSchemaChanges')
BEGIN
    EXEC('CREATE TRIGGER TR_TrackSchemaChanges
    ON DATABASE
    FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE,
        CREATE PROCEDURE, ALTER_PROCEDURE, DROP_PROCEDURE,
        CREATE FUNCTION, ALTER_FUNCTION, DROP_FUNCTION,
        CREATE VIEW, ALTER_VIEW, DROP_VIEW
    AS
    BEGIN
        SET NOCOUNT ON;

        DECLARE @EventData XML = EVENTDATA();

        INSERT INTO SchemaChanges (
            EventType,
            ObjectName,
            ObjectType,
            SQLCommand,
            LoginName
        )
        VALUES (
            @EventData.value(''(/EVENT_INSTANCE/EventType)[1]'', ''NVARCHAR(100)''),
            @EventData.value(''(/EVENT_INSTANCE/ObjectName)[1]'', ''NVARCHAR(256)''),
            @EventData.value(''(/EVENT_INSTANCE/ObjectType)[1]'', ''NVARCHAR(100)''),
            @EventData.value(''(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]'', ''NVARCHAR(MAX)''),
            @EventData.value(''(/EVENT_INSTANCE/LoginName)[1]'', ''NVARCHAR(256)'')
        );
    END;')
END
GO
```

### 3. Versiyon Yönetim Prosedürleri

`3\_version\_management.sql` dosyasında, versiyon yönetimi için gerekli stored procedure'leri oluşturuyoruz. Bu prosedürler:

- sp\_UpgradeDatabase: Yeni versiyona geçiş
- sp\_RollbackDatabase: Önceki versiyona dönüş
- sp\_GetVersionHistory: Versiyon geçmişi görüntüleme

```
USE VersionControlDB;
GO

-- Veritabanı versiyonunu yükseltme prosedürü
IF NOT EXISTS (SELECT * FROM sys.procedures WHERE name = 'sp_UpgradeDatabase')
BEGIN
    EXEC('CREATE PROCEDURE sp_UpgradeDatabase
        @TargetVersion VARCHAR(20),
        @Description NVARCHAR(500)
    AS
    BEGIN
        SET NOCOUNT ON;

        BEGIN TRY
            BEGIN TRANSACTION;

            -- Mevcut versiyonu kontrol et
            DECLARE @CurrentVersion VARCHAR(20);
            SELECT TOP 1 @CurrentVersion = VersionNumber
            FROM DatabaseVersion
            ORDER BY VersionID DESC;

            IF @CurrentVersion >= @TargetVersion
            BEGIN
                RAISERROR (N'''Hedef versiyon mevcut versiyondan küçük veya eşit
olamaz.''', 16, 1);
                RETURN;
            END

            -- Yeni versiyon kaydı ekle
            INSERT INTO DatabaseVersion (VersionNumber, Description, Status)
            VALUES (@TargetVersion, @Description, ''In Progress'');
        END TRY
        BEGIN CATCH
            IF @@TRANCOUNT > 0
                ROLLBACK TRANSACTION;
            RAISERROR (N'''Hedef versiyon mevcut versiyondan küçük veya eşit
olamaz.''', 16, 1);
        END CATCH
    END
END
```

```

-- Burada yükseltme işlemleri gerçekleştirilir
-- (Örnek: ALTER TABLE, CREATE TABLE vb.)

-- Versiyon durumunu güncelle
UPDATE DatabaseVersion
SET Status = ''Completed''
WHERE VersionNumber = @TargetVersion;

COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

    -- Hata durumunu kaydet
    UPDATE DatabaseVersion
    SET Status = ''Failed - '' + ERROR_MESSAGE()
    WHERE VersionNumber = @TargetVersion;

    RAISERROR (N'''Versiyon yükseltme işlemi başarısız oldu.'', 16, 1);
END CATCH
END')
END
GO

-- Veritabanı geri alma prosedürü
IF NOT EXISTS (SELECT * FROM sys.procedures WHERE name = 'sp_RollbackDatabase')
BEGIN
    EXEC ('CREATE PROCEDURE sp_RollbackDatabase
        @TargetVersion VARCHAR(20)
    AS
    BEGIN
        SET NOCOUNT ON;

        BEGIN TRY
            BEGIN TRANSACTION;

            -- Mevcut versiyonu kontrol et
            DECLARE @CurrentVersion VARCHAR(20);
            SELECT TOP 1 @CurrentVersion = VersionNumber
            FROM DatabaseVersion

```

```

        ORDER BY VersionID DESC;

        IF @CurrentVersion <= @TargetVersion
        BEGIN
            RAISERROR (N'''Geri alma versiyonu mevcut versiyondan buyuk veya esit
olamaz.'', 16, 1);
            RETURN;
        END

        -- Geri alma islemi kaydi
        INSERT INTO DatabaseVersion (VersionNumber, Description, Status)
        VALUES (@TargetVersion, ''Rollback from '' + @CurrentVersion, ''In
Progress'');

        -- Burada geri alma islemleri gereklestirilir
        -- (Ornek: DROP TABLE, ALTER TABLE vb.)

        -- Versiyon durumunu güncelle
        UPDATE DatabaseVersion
        SET Status = ''Completed''
        WHERE VersionNumber = @TargetVersion;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        -- Hata durumunu kaydet
        UPDATE DatabaseVersion
        SET Status = ''Rollback Failed: '' + ERROR_MESSAGE()
        WHERE VersionNumber = @TargetVersion;

        RAISERROR (N'''Geri alma islemi basarisiz oldu.'', 16, 1);
    END CATCH
END')
END
GO

-- Versiyon geçmisini görüntüleme prosedürü
IF NOT EXISTS (SELECT * FROM sys.procedures WHERE name = 'sp_GetVersionHistory')
BEGIN

```

```

EXEC('CREATE PROCEDURE sp_GetVersionHistory
AS
BEGIN
    SELECT
        VersionNumber,
        AppliedDate,
        Description,
        Status
    FROM DatabaseVersion
    ORDER BY VersionID DESC;
END')
END
GO

```

#### 4. Versiyon Yükseltme Örneği

'4\_upgrade\_v2.sql' dosyasında 2.0.0 versiyonuna geçiş yapıyoruz. Bu aşamada:

- Customers tablosuna yeni kolonlar ekliyoruz
- Orders tablosunu oluşturuyoruz
- Versiyon bilgisini güncelliyoruz

```

USE VersionControlDB;
GO

-- Versiyon 2.0.0'a yükseltme örneği
BEGIN TRY
    BEGIN TRANSACTION;

    -- Yeni sürüm için değişiklikleri uygula
    -- 1. Customers tablosuna yeni kolonlar ekle
    IF NOT EXISTS (SELECT * FROM sys.columns WHERE object_id = OBJECT_ID('Customers')
    AND name = 'PhoneNumber')
        BEGIN
            ALTER TABLE Customers
            ADD PhoneNumber NVARCHAR(20);
        END

    IF NOT EXISTS (SELECT * FROM sys.columns WHERE object_id = OBJECT_ID('Customers')
    AND name = 'CreatedDate')
        BEGIN
            ALTER TABLE Customers
            ADD CreatedDate DATETIME DEFAULT GETDATE();
        END

```

```

END

-- 2. Orders tablosunu oluştur (eğer yoksa)
IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'Orders')
BEGIN
    CREATE TABLE Orders (
        OrderID INT PRIMARY KEY IDENTITY(1,1),
        CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID),
        OrderDate DATETIME DEFAULT GETDATE(),
        TotalAmount DECIMAL(18,2)
    );
END

-- Versiyon bilgisini güncelle
EXEC sp_UpgradeDatabase '2.0.0', 'Added phone number to Customers and created
Orders table';

COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

    -- Hata bilgisini kaydet
    INSERT INTO DatabaseVersion (VersionNumber, Description, Status)
    VALUES ('2.0.0', 'Upgrade failed: ' + ERROR_MESSAGE(), 'Failed');

    RAISERROR (N'Versiyon yükseltme işlemi basarisiz oldu.', 16, 1);
END CATCH;

```

## 5. Test ve Doğrulama

Son olarak `5\_test\_commands.sql` dosyası ile sistemin düzgün çalıştığını test ediyoruz. Bu testler:

- Örnek veri ekleme
- Versiyon geçmişi kontrolü
- Şema değişikliklerini görüntüleme
- Örnek raporlar oluşturma

```

USE VersionControlDB;
GO

```

```
-- 1. Versiyon geçmişini kontrol et
EXEC sp_GetVersionHistory;

-- 2. Şema değişikliklerini kontrol et
SELECT TOP 10 *
FROM SchemaChanges
ORDER BY ChangeDate DESC;

-- 3. Test verisi ekle
INSERT INTO Customers (FirstName, LastName, Email, PhoneNumber)
VALUES
    ('Ahmet', 'Yılmaz', 'ahmet@email.com', '5551234567'),
    ('Ayşe', 'Demir', 'ayse@email.com', '5557654321');

-- 4. Test siparişleri ekle
INSERT INTO Orders (CustomerID, TotalAmount)
VALUES
    (1, 150.50),
    (1, 75.25),
    (2, 225.00);

-- 5. Test sorguları
-- 5.1. Müşteri ve sipariş bilgilerini birleştirerek göster
SELECT
    c.CustomerID,
    c.FirstName + ' ' + c.LastName AS CustomerName,
    c.Email,
    c.PhoneNumber,
    COUNT(o.OrderID) AS TotalOrders,
    SUM(o.TotalAmount) AS TotalSpent
FROM Customers c
LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY
    c.CustomerID,
    c.FirstName,
    c.LastName,
    c.Email,
    c.PhoneNumber;

-- 5.2. Son 5 siparişi göster
SELECT TOP 5
    o.OrderID,
    c.FirstName + ' ' + c.LastName AS CustomerName,
    o.OrderDate,
    o.TotalAmount
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID
ORDER BY o.OrderDate DESC;
```

```

-- 6. Geri alma testi (DİKKAT: Bu komutu sadece test ortamında çalıştırın!)
-- EXEC sp_RollbackDatabase '1.0.0';

-- 7. Hata durumu testi
BEGIN TRY
    -- Kasıtlı hata oluştur (duplicate primary key)
    INSERT INTO Customers (CustomerID, FirstName, LastName, Email)
    VALUES (1, 'Test', 'User', 'test@email.com');
END TRY
BEGIN CATCH
    PRINT 'Beklenen hata oluştu: ' + ERROR_MESSAGE();
END CATCH;

-- 8. Versiyon kontrolü
DECLARE @CurrentVersion VARCHAR(20);
SELECT TOP 1 @CurrentVersion = VersionNumber
FROM DatabaseVersion
ORDER BY VersionID DESC;

IF @CurrentVersion = '2.0.0'
    PRINT 'Versiyon yükseltme başarılı: ' + @CurrentVersion;
ELSE
    PRINT 'Versiyon yükseltme başarısız. Mevcut versiyon: ' + @CurrentVersion;

```

	VersionNumber	AppliedDate	Description	Status
1	4.0.0	2025-05-29 17:16:10.093	Added phone number to Customers and created Orders table	Completed
2	2.0.0	2025-05-29 17:15:53.533	Upgrade failed: String or binary data would be truncated in t...	Failed
3	2.0.0	2025-05-29 16:54:35.027	Upgrade failed: String or binary data would be truncated in t...	Failed
4	1.0.0	2025-05-29 16:38:16.317	Rollback from 2.0.0	Completed
5	2.0.0	2025-05-29 16:37:44.527	Added phone number to Customers and created Orders table	Completed
6	1.0.0	2025-05-29 16:36:20.610	Initial database version	Completed

	EventType	ObjectName	ObjectType	SQLCommand	LoginName	ChangeDate
1	CREATE_TABLE	Orders	TABLE	CREATE TABLE Orders (OrderID INT PRIMARY KEY IDENTIT...	sa	2025-05-29 16:37:44
2	ALTER_TABLE	Customers	TABLE	ALTER TABLE Customers ADD PhoneNumber NVARCHAR(20...	sa	2025-05-29 16:37:44
3	CREATE_PROCEDURE	sp_GetVersionHistory	PROCEDURE	-- Versiyon geçmişini görüntüleme prosedürü: CREATE PROCEDURE ...	sa	2025-05-29 16:37:37
4	CREATE_PROCEDURE	sp_RollbackDatabase	PROCEDURE	-- Veritabanı geri alma prosedürü: CREATE PROCEDURE sp_Rollbac...	sa	2025-05-29 16:37:37
5	CREATE_PROCEDURE	sp_UpgradeDatabase	PROCEDURE	-- Veritabanı versiyonunu yükseltme prosedürü: CREATE PROCEDUR...	sa	2025-05-29 16:37:37

CustomerID	CustomerName	Email	PhoneNumber	TotalOrders	TotalSpent
1	Ahmet Yılmaz	ahmet@email.com	5551234567	4	451.50
2	Ayse Demir	ayse@email.com	5557654321	2	450.00
3	Ahmet Yılmaz	ahmet@email.com	5551234567	0	<null>
4	Ayse Demir	ayse@email.com	5557654321	0	<null>

OrderID	CustomerName	OrderDate	TotalAmount
4	Ahmet Yılmaz	2025-05-29 17:16:40.350	150.50
5	Ahmet Yılmaz	2025-05-29 17:16:40.350	75.25
6	Ayse Demir	2025-05-29 17:16:40.350	225.00
1	Ahmet Yılmaz	2025-05-29 16:54:43.840	150.50
2	Ahmet Yılmaz	2025-05-29 16:54:43.840	75.25

Bu proje sayesinde:

- Veritabanı değişikliklerini kontrollü yapabiliyoruz
- Her değişikliği kayıt altına alıyoruz
- Hata durumunda geri dönebiliyoruz
- Kim, ne zaman, ne değiştirmiş, hepsini görebiliyoruz

Özellikle dikkat ettiğimiz noktalar:

1. Her değişikliği transaction içinde yapıyoruz
2. Hata kontrolü yapıyoruz
3. Geri alma planı hazırlıyoruz
4. Tüm değişiklikleri dokümanete ediyoruz

[photo]

Son olarak, tüm versiyon geçmişini ve şema değişikliklerini gösteren bir özet ekran görüntüsü ekleyin. Bu görüntüde DatabaseVersion ve SchemaChanges tablolarının son durumları görmeli.

Bu şekilde veritabanı versiyon yönetimini güvenli ve kontrollü bir şekilde yapabiliyoruz.

Mustafa Girgin 21290649

<https://github.com/antelcha/sqlprojeler>

# Otomatik Yedekleme Sistemi Kurulumu ve Testi

Bu projede, SQL Server veritabanları için kapsamlı bir otomatik yedekleme sistemi geliştirdim. Sistem, düzenli yedeklemeleri otomatik olarak alıp, yedekleme geçmişini takip ediyor ve olası hataları raporluyor.

## 1. Veritabanı ve Tablo Yapılarının Oluşturulması

İlk adım olarak, yedekleme sisteminin ihtiyaç duyduğu veritabanı ve tabloları oluşturdum.

TestDB adında bir test veritabanı ve içerisinde örnek veriler için Customers tablosu ile yedekleme kayıtları için BackupHistory tablosu yer alıyor.

```
-- Veritabanı kontrolü ve oluşturma
IF NOT EXISTS (SELECT name FROM sys.databases WHERE name = 'TestDB')
BEGIN
    CREATE DATABASE TestDB;
    PRINT 'TestDB veritabanı oluşturuldu.';
END
ELSE
BEGIN
    PRINT 'TestDB veritabanı zaten mevcut.';
END
GO
```

```
USE TestDB;
GO

-- Customers tablosu kontrolü ve oluşturma
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Customers]') AND type in (N'U'))
BEGIN
    CREATE TABLE Customers (
        CustomerID INT PRIMARY KEY IDENTITY(1,1),
        FirstName NVARCHAR(50),
        LastName NVARCHAR(50),
        Email NVARCHAR(100),
        CreatedDate DATETIME DEFAULT GETDATE()
    );
    PRINT 'Customers tablosu oluşturuldu.';
END
```

```

ELSE
BEGIN
    PRINT 'Customers tablosu zaten mevcut.';
END
GO

-- BackupHistory tablosu kontrolü ve oluşturma
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[BackupHistory]') AND type in (N'U'))
BEGIN
    CREATE TABLE BackupHistory (
        BackupID INT PRIMARY KEY IDENTITY(1,1),
        DatabaseName NVARCHAR(100),
        BackupType NVARCHAR(50),
        BackupStartTime DATETIME,
        BackupEndTime DATETIME,
        BackupStatus NVARCHAR(50),
        BackupSizeKB BIGINT,
        BackupLocation NVARCHAR(500)
    );
    PRINT 'BackupHistory tablosu oluşturuldu.';
END
ELSE
BEGIN
    PRINT 'BackupHistory tablosu zaten mevcut.';
END
GO

-- Örnek veri ekleme
IF NOT EXISTS (SELECT TOP 1 * FROM Customers)
BEGIN
    INSERT INTO Customers (FirstName, LastName, Email)
    VALUES
        ('John', 'Doe', 'john@example.com'),
        ('Jane', 'Smith', 'jane@example.com'),
        ('Mike', 'Johnson', 'mike@example.com');
    PRINT 'Örnek müşteri verileri eklendi.';
END
ELSE
BEGIN
    PRINT 'Customers tablosunda zaten veri mevcut.';
END
GO

```

## 2. Yedekleme Prosedürünün Oluşturulması

Yedekleme işlemlerini yönetmek için `sp_CreateBackup` adında bir stored procedure oluştururdum. Bu prosedür:

Yedekleme başlangıç kaydını oluşturuyor  
Yedekleme işlemini gerçekleştiriyor  
İşlem sonucunu kaydediyor  
Hata durumunda uygun şekilde raporluyor

```
USE TestDB;
GO

-- Yedekleme prosedürü kontrolü ve oluşturma
IF EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[sp_CreateBackup]') AND type in (N'P'))
BEGIN
    DROP PROCEDURE [dbo].[sp_CreateBackup];
    PRINT 'Eski sp_CreateBackup prosedürü silindi.';
END
GO

CREATE PROCEDURE sp_CreateBackup
    @DatabaseName NVARCHAR(100),
    @BackupPath NVARCHAR(500)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @BackupFileName NVARCHAR(1000)
    DECLARE @StartTime DATETIME
    DECLARE @SQL NVARCHAR(MAX)

    SET @StartTime = GETDATE()
    SET @BackupFileName = @BackupPath + @DatabaseName + '_' +
        REPLACE(CONVERT(VARCHAR, @StartTime, 112) + REPLACE(CONVERT(VARCHAR,
        @StartTime, 108), ':', ''), ' ', '_') + '.bak'

    BEGIN TRY
        -- Yedekleme başlangıç kaydı
        INSERT INTO BackupHistory (DatabaseName, BackupType, BackupStartTime,
        BackupStatus)
        VALUES (@DatabaseName, 'FULL', @StartTime, 'IN_PROGRESS')

        -- Yedekleme komutunu oluştur
        SET @SQL = 'BACKUP DATABASE ' + @DatabaseName +
            ' TO DISK = ''' + @BackupFileName + ''''

        -- Yedekleme işlemini gerçekleştir
        EXEC (@SQL)

        -- Yedekleme kaydını güncelle
```

```

        UPDATE BackupHistory
        SET BackupEndTime = GETDATE(),
            BackupStatus = 'COMPLETED',
            BackupLocation = @BackupFileName,
            BackupSizeKB = (SELECT size * 8 FROM sys.master_files WHERE
database_id = DB_ID(@DatabaseName) AND type_desc = 'ROWS')
        WHERE DatabaseName = @DatabaseName
        AND BackupStartTime = @StartTime

        PRINT 'Yedekleme işlemi başarıyla tamamlandı.';

END TRY
BEGIN CATCH
    -- Hata durumunda kaydı güncelle
    UPDATE BackupHistory
    SET BackupEndTime = GETDATE(),
        BackupStatus = 'FAILED'
    WHERE DatabaseName = @DatabaseName
    AND BackupStartTime = @StartTime

    PRINT 'Hata: ' + ERROR_MESSAGE();
    THROW;
END CATCH
END;
GO
PRINT 'sp_CreateBackup prosedürü oluşturuldu.';

-- Yedekleme rapor view kontrolü ve oluşturma
IF EXISTS (SELECT * FROM sys.views WHERE object_id =
OBJECT_ID(N'[dbo].[vw_BackupReport]'))
BEGIN
    DROP VIEW [dbo].[vw_BackupReport];
    PRINT 'Eski vw_BackupReport view silindi.';
END
GO

CREATE VIEW vw_BackupReport
AS
SELECT
    DatabaseName,
    BackupType,
    BackupStartTime,
    BackupEndTime,
    DATEDIFF(MINUTE, BackupStartTime, BackupEndTime) as DurationMinutes,
    BackupStatus,
    BackupSizeKB / 1024.0 as BackupSizeMB,
    BackupLocation
FROM BackupHistory;
GO
PRINT 'vw_BackupReport view oluşturuldu.';
```

### 3. Otomatik Yedekleme Job'ının Oluşturulması

SQL Server Agent kullanarak Daily\_Database\_Backup adında bir job oluşturduk. Bu job her gece yarısı otomatik olarak çalışacak şekilde ayarlandı.

```
USE msdb;
GO

-- Job kontrolü ve silme
IF EXISTS (SELECT job_id FROM msdb.dbo.sysjobs WHERE name =
N'Daily_Database_Backup')
BEGIN
    EXEC dbo.sp_delete_job @job_name = N'Daily_Database_Backup';
    PRINT 'Eski Daily_Database_Backup job silindi.';
END
GO

-- Schedule kontrolü ve silme
IF EXISTS (SELECT schedule_id FROM msdb.dbo.sysschedules WHERE name =
N'DailyBackupSchedule')
BEGIN
    EXEC dbo.sp_delete_schedule @schedule_name = N'DailyBackupSchedule';
    PRINT 'Eski DailyBackupSchedule schedule silindi.';
END
GO

BEGIN TRY
    -- Job oluştur
    EXEC dbo.sp_add_job
        @job_name = N'Daily_Database_Backup',
        @description = N'Her gün otomatik yedek alma işlemi',
        @enabled = 1;

    -- Job'a adım ekle
    EXEC sp_add_jobstep
        @job_name = N'Daily_Database_Backup',
        @step_name = N'Perform Backup',
        @subsystem = N'TSQL',
        @command = N'EXEC TestDB.dbo.sp_CreateBackup
                        @DatabaseName = ''TestDB'',
                        @BackupPath = ''/var/opt/mssql/backup'''';

    -- Günlük çalışma planı oluştur
    EXEC dbo.sp_add_schedule
        @schedule_name = N'DailyBackupSchedule',
        @freq_type = 4, -- Günlük
        @freq_interval = 1, -- Her gün
```

```

@active_start_time = 000000; -- Gece yarısı (00:00)

-- Schedule'ı job'a bağla
EXEC sp_attach_schedule
    @job_name = N'Daily_Database_Backup',
    @schedule_name = N'DailyBackupSchedule';

-- Job'ı aktif et
EXEC dbo.sp_add_jobserver
    @job_name = N'Daily_Database_Backup';

PRINT 'Daily_Database_Backup job başarıyla oluşturuldu.';
END TRY
BEGIN CATCH
    PRINT 'Hata: ' + ERROR_MESSAGE();
    THROW;
END CATCH
GO

```

[2025-05-29 21:01:56] [S0001][22022] SQLServerAgent
Daily\_Database\_Backup job başarıyla oluşturuldu.
[2025-05-29 21:01:56] completed in 41 ms

#### 4. Test ve Doğrulama

Sistemin düzgün çalıştığından emin olmak için çeşitli testler gerçekleştirdim:

##### Manuel Yedekleme Testi

İlk olarak manuel bir yedekleme işlemi başlattım:

```

USE TestDB;
GO

-- Manuel yedek alma testi
PRINT 'Manuel yedekleme testi başlatılıyor...';
EXEC sp_CreateBackup
    @DatabaseName = 'TestDB',
    @BackupPath = '/var/opt/mssql/backup/';
GO

-- Yeni test verisi ekleme
PRINT 'Test verileri ekleniyor...';
INSERT INTO Customers (FirstName, LastName, Email)

```

```

VALUES
    ('Sarah', 'Wilson', 'sarah@example.com'),
    ('Robert', 'Brown', 'robert@example.com');
GO

-- Job durumunu kontrol et
USE msdb;
GO

PRINT 'Job durumu kontrol ediliyor...';
SELECT
    j.name AS 'Job Name',
    CASE j.enabled
        WHEN 1 THEN 'Enabled'
        ELSE 'Disabled'
    END AS 'Job Status',
    CASE
        WHEN jh.run_date IS NOT NULL AND jh.run_time IS NOT NULL
        THEN CONVERT(DATETIME,
            CAST(jh.run_date AS CHAR(8)) + ' ' +
            STUFF(STUFF(RIGHT('000000' + CAST(jh.run_time AS VARCHAR(6)), 6), 5,
0, ':'), 3, 0, ':'))
        END AS 'Last Run Time',
    CASE jh.run_status
        WHEN 0 THEN 'Failed'
        WHEN 1 THEN 'Succeeded'
        WHEN 2 THEN 'Retry'
        WHEN 3 THEN 'Canceled'
        WHEN 4 THEN 'In Progress'
    END AS 'Last Run Status'
FROM
    msdb.dbo.sysjobs j
LEFT JOIN (
    SELECT
        job_id,
        run_date,
        run_time,
        run_status
    FROM msdb.dbo.sysjobhistory
    WHERE step_id = 0
) jh ON j.job_id = jh.job_id
WHERE
    j.name = 'Daily_Database_Backup';
GO

-- Yedekleme geçmişini kontrol et
USE TestDB;
GO

```

```
PRINT 'Yedekleme geçmişi kontrol ediliyor...';
SELECT
    DatabaseName,
    BackupType,
    FORMAT(BackupStartTime, 'yyyy-MM-dd HH:mm:ss') as StartTime,
    FORMAT(BackupEndTime, 'yyyy-MM-dd HH:mm:ss') as EndTime,
    DurationMinutes,
    BackupStatus,
    CAST(BackupSizeMB AS DECIMAL(10,2)) as BackupSizeMB,
    BackupLocation
FROM vw_BackupReport
ORDER BY BackupStartTime DESC;
GO

-- Job'ı manuel olarak çalıştır
USE msdb;
GO

PRINT 'Job manuel olarak çalıştırılıyor...';
EXEC msdb.dbo.sp_start_job N'Daily_Database_Backup';
GO

-- 10 saniye bekle
WAITFOR DELAY '00:00:10';
GO

-- Son durumu tekrar kontrol et
PRINT 'Son durum kontrol ediliyor...';
USE TestDB;
GO

SELECT TOP 5 * FROM vw_BackupReport
ORDER BY BackupStartTime DESC;
GO
```

Bu proje sayesinde, veritabanı yedeklemelerinin güvenli ve otomatik bir şekilde alınmasını, takip edilmesini ve raporlanması sağlanan kapsamlı bir sistem geliştirmiş oldum. Sistem, olası hata durumlarına karşı dayanıklı ve kolayca izlenebilir bir yapıda tasarlandı.