

**ANKARA ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**



BLM4522 – Ağ Tabanlı Paralel Dağıtım Sistemleri

Proje Dökümantasyonu

06/2025

Pınar ERSÜREN 20290249

Melih ÜZEL 202990299

PROJE LINKLERİ

Github Linki: <https://github.com/persuren/Mssql-Project>

Proje 1: Veritabanı Performans Optimizasyonu ve İzleme

Video: <https://drive.google.com/file/d/19tOP-Ktaobz16DwlzPfJMD4feATggw-h/view?usp=sharing>

Proje 2: Veritabanı Yedekleme ve Felaketten Kurtarma Planı

Video: <https://drive.google.com/file/d/1iBjalv254yR5cywnDWGsxDVKCJTL61tU/view?usp=sharing>

Proje 3: Veritabanı Güvenliği ve Erişim Kontrolü

Video: <https://drive.google.com/file/d/1YpHcaSIZi33J7adqRlrH--79o2xq2vw/view?usp=sharing>

Proje 4: Veritabanı Yük Dengeleme ve Dağıtık Veritabanı

Yapıları

Video: <https://drive.google.com/file/d/1ctu89VQFYwexQ7y3b-eSL83nBAzPUYFK/view?usp=sharing>

Proje 5: Veri Temizleme ve ETL Süreçleri Tasarımı

Video: <https://drive.google.com/file/d/1PneTRK7dtxeArYcgP5alxeP0qJenYpri/view?usp=sharing>

Proje 6: Veritabanı Yükseltme ve Sürüm Yönetimi

Video: <https://drive.google.com/file/d/1cWli9xM0BbtSooA6xPC2vYPWyBebvZA/view?usp=sharing>

Proje 7: Veritabanı Yedekleme ve Otomasyon Çalışması

Video: <https://drive.google.com/file/d/1AfAzn64qgLIHtV6sp5Zgg-1aZlv5uJpl/view?usp=sharing>

İÇİNDEKİLER

| | |
|---|-----------|
| Proje 1: Veritabanı Performans Optimizasyonu ve İzleme..... | 4 |
| Bölüm 1: Veritabanı İzleme | 4 |
| Bölüm 2: Soru İyileştirme ve Performans Analizleri | 4 |
| Bölüm 3: İndeks Yönetimi | 9 |
| Proje 2: Veritabanı Yedekleme ve Felaketten Kurtarma Planı..... | 15 |
| Bölüm 1: Tam, Artık, Fark Yedeklemeleri..... | 15 |
| Bölüm 2: Zamanlayıcı ile Yedekleme Otomasyonu | 17 |
| Bölüm 3: Felaketten Kurtarma Senaryoları..... | 19 |
| Bölüm 4: Test Yedekleme Senaryoları | 21 |
| Proje 3: Veritabanı Güvenliği ve Erişim Kontrolü..... | 23 |
| Bölüm 1: Erişim Yönetimi..... | 23 |
| Bölüm 2: Veri Şifreleme | 27 |
| Bölüm 3: SQL Injection Testleri..... | 29 |
| Bölüm 4: Audit Logları | 31 |
| Proje 4: Veritabanı Yük Dengeleme ve Dağıtık Veritabanı Yapıları..... | 34 |
| Bölüm 1: SQL Server Replikasyonu ile Veritabanı Çoğaltma..... | 34 |
| Bölüm 2: Yük Dengeleme | 37 |
| Bölüm 3: Failover Senaryoları | 41 |
| Proje 5: Veri Temizleme ve ETL Süreçleri Tasarımı | 46 |
| Bölüm 1: Veri Temizleme | 46 |
| Bölüm 2: Veri Dönüşürme | 50 |
| Bölüm 3: Veri Yükleme..... | 52 |
| Bölüm 4: Veri Kalitesi Raporları | 55 |
| Proje 6: Veritabanı Yükseltme ve Sürüm Yönetimi | 58 |
| Bölüm 1: Veritabanı Yükseltme Planı..... | 58 |
| Bölüm 2: Sürüm Yönetimi | 62 |
| Bölüm 3: Test ve Geri Dönüş Planı..... | 68 |
| Proje 7 - Veritabanı Yedekleme ve Otomasyon Çalışması | 73 |
| Link: https://www.kaggle.com/datasets/khushikyad001/covid-19-global-dataset | 73 |
| Bölüm 1: Yedekleme Süreçlerini Otomatikleştirme | 73 |
| Bölüm 2: T-SQL ile Rapor Oluşturma..... | 77 |
| Bölüm 3: Otomatik Yedekleme Uyarıları..... | 79 |

Proje 1: Veritabanı Performans Optimizasyonu ve İzleme

Kullanılan veri tabanı: Microsoft AdventureWorks Data Warehouse 2022

Link:<https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms#download-backup-files>

Bu projede istenen adımlar 4 temel başlık altında değerlendirilmiştir:

- 1- Veritabanı İzleme
- 2- İndeks Yönetimi
- 3- Sorgu İyileştirme
- 4- Veri Yöneticisi Roller: Farklı roller için erişim yönetimi.

Biz de yaptığımız adımları bu başlıklar üzerinde değerlendirecek olursak.

Bölüm 1: Veritabanı İzleme

Bu başlık genel olarak birçok sorgu içerisinde Dynamic Management Views (DMV) kullanılarak performansın ölçülmesi, izlenmesi ve hataların tespit edilmesi için gerçekleştirılmıştır.

Bölüm 2: Sorgu İyileştirme ve Performans Analizleri

İlk olarak, veritabanında en fazla CPU zamanı harcayan sorguları tespit ederek başlamak mantıklı olacaktır. Bu, optimizasyona nereden başlayacağımızı belirlememize yardımcı olacak.

```
USE AdventureWorksDW2022;
GO

SELECT TOP 10
    qs.total_worker_time / qs.execution_count AS avg_cpu_time,
    qs.total_worker_time AS total_cpu_time,
    qs.execution_count,
    qs.max_worker_time AS max_cpu_time,
    qs.total_elapsed_time / qs.execution_count AS avg_elapsed_time,
    qs.total_elapsed_time,
    qs.max_elapsed_time,
    SUBSTRING(st.text, (qs.statement_start_offset/2) + 1,
        ((CASE qs.statement_end_offset
            WHEN -1 THEN DATALENGTH(st.text)
            ELSE qs.statement_end_offset
            END - qs.statement_start_offset)/2) + 1) AS query_text,
    qp.query_plan
FROM
    sys.dm_exec_query_stats AS qs
CROSS APPLY
    sys.dm_exec_sql_text(qs.sql_handle) AS st
CROSS APPLY
    sys.dm_exec_query_plan(qs.plan_handle) AS qp
ORDER BY
    qs.total_worker_time DESC;
GO
```

Bu sorgu, sys.dm_exec_query_stats DMV'sini kullanarak sorgu istatistiklerini çeker ve en yüksek toplam CPU süresine sahip ilk 10 sorguyu listelenmesini sağlar.

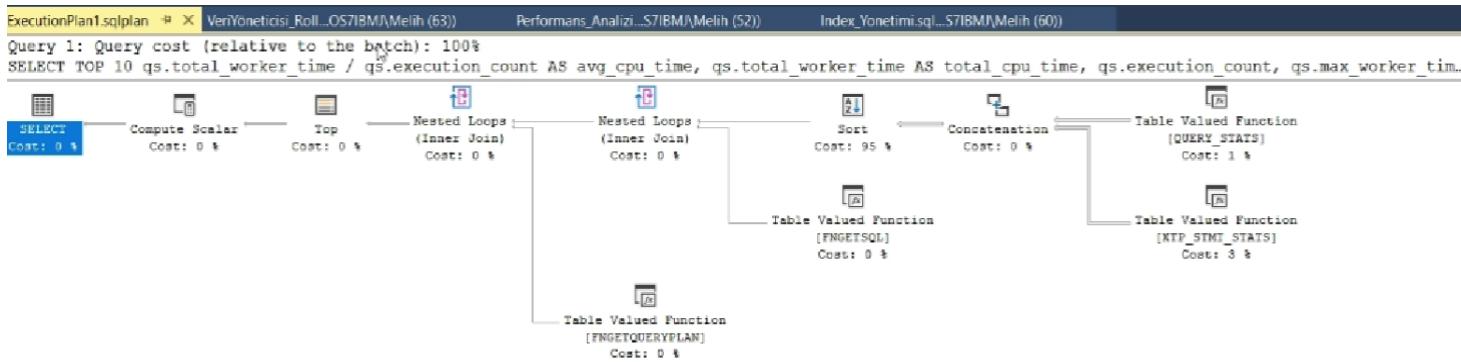
| | avg_cpu_time | total_cpu_time | execution_count | max_cpu_time | avg_elapsed_time | total_elapsed_time | max_elapsed_time | query_text | query_plan |
|----|--------------|----------------|-----------------|--------------|------------------|--------------------|------------------|--|---|
| 1 | 53834 | 484507 | 9 | 57176 | 74284 | 668556 | 78478 | SELECT TOP 10 qs.total_worker_time / qs.execution_c... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |
| 2 | 16385 | 147466 | 9 | 27571 | 21856 | 196706 | 33929 | SELECT target_data FROM sys.dm_xe_session_targets... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |
| 3 | 49249 | 49249 | 1 | 49249 | 62095 | 62095 | 62095 | SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], sp... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |
| 4 | 21389 | 21389 | 1 | 21389 | 25029 | 25029 | 25029 | SELECT SCHEMA_NAME(udt.schema_id) AS [Schema], u... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |
| 5 | 17754 | 17754 | 1 | 17754 | 17795 | 17795 | 17795 | WITH Quotiles AS (SELECT DISTINCT ... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |
| 6 | 1376 | 9637 | 7 | 1501 | 1835 | 12851 | 1971 | insert #filempfn selectfile_or_directory_name, 1 - is_d... ... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |
| 7 | 670 | 9391 | 14 | 927 | 809 | 11338 | 1188 | insert #filempfn selectfile_or_directory_name, 1 - is_d... ... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |
| 8 | 7938 | 7938 | 1 | 7938 | 7938 | 7938 | 7938 | SELECT ltbl.name AS [Name], ltbl.object_id AS [ID], ltbl.create... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |
| 9 | 958 | 7670 | 8 | 1064 | 1248 | 9991 | 1349 | insert #filempfn selectfile_or_directory_name, 1 - is_d... ... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |
| 10 | 5309 | 5309 | 1 | 5309 | 9622 | 9622 | 9622 | SELECT ia_member(N'db_acceaaadmin') AS [iaDbAccess... | <ShowPlanXML xmlnames='http://schemas.microsoft.com/... |

Mevcut sorguyu çalıştırıldığımızda yukarıdaki gibi bir sonuç elde ediyoruz. İlk sıradaki sorgu, mevcut değerlere bakıldığından diğerlerine göre belirgin şekilde daha fazla kaynak (CPU ve süre) harcadığı anlaşılıyor.

Sorgu tipini ve query_plan başlığı altındaki Execution Planı incelediğimizde performansı hakkında bir takım bilgiler elde edebiliyoruz. Execution Plan'daki en yüksek maliyetli (%50) işlem "Clustered Index Scan". Bu genellikle, sorgunun ilgili tabloyu büyük ölçüde taraması gerektiği anlamına gelir çünkü WHERE koşuluna veya JOIN'lere uygun daha verimli bir indeks bulunamamıştır.

Query_text kısmındaki sorgu tipine bakıldığına ise, SQL Server Management Studio'nun (SSMS) veritabanındaki fonksiyonlar gibi nesneleri listelemek için kullandığı bir metadata sorgusu olduğu görülüyor. sys.all_objects, sys.sql_modules gibi sistem tablolarını sorgulaya yarıyor.

Dolayısıyla bu tarz yapıdaki sistem sorgularını optimize etmek pek mümkün bir işlem değildir.



Bahsi geçen Execution Plan

Performans analizi için farklı işlemler deneyelim. FactInternetSales tablosunu ve ilişkili boyut tablolarını kullanan tipik bir analitik sorgu çalıştırarak performans ölçümü sağlamayı deneyebiliriz.

```

USE AdventureWorksDW2022;
GO

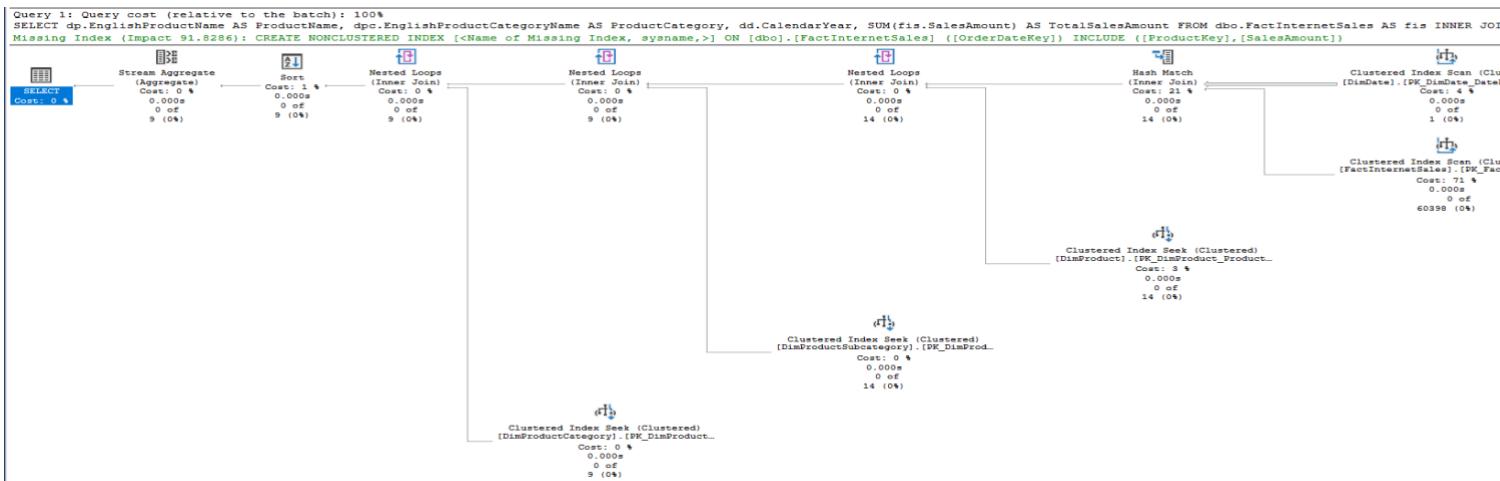
]SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO

]SELECT
    dp.EnglishProductName          AS ProductName,
    dpc.EnglishProductCategoryName AS ProductCategory,
    dd.CalendarYear,
    SUM(fis.SalesAmount)          AS TotalSalesAmount
FROM
    dbo.FactInternetSales AS fis
INNER JOIN
    dbo.DimProduct AS dp ON fis.ProductKey = dp.ProductKey
INNER JOIN
    dbo.DimProductSubcategory AS dps ON dp.ProductSubcategoryKey = dps.ProductSubcategoryKey
INNER JOIN
    dbo.DimProductCategory AS dpc ON dps.ProductCategoryKey = dpc.ProductCategoryKey
INNER JOIN
    dbo.DimDate AS dd ON fis.OrderDateKey = dd.DateKey
WHERE
    dd.CalendarYear >= 2020
GROUP BY
    dp.EnglishProductName,
    dpc.EnglishProductCategoryName,
    dd.CalendarYear
ORDER BY
    ProductCategory,
    ProductName,
    CalendarYear;
GO

]SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
GO

```

Bu sorgu çalıştırıldıktan sonra Messages kısmındaki sonuçlar ve Execution Plan incelenerek performans hakkında çeşitli bilgiler elde edilebilir.



Sorgunun çalıştırılmasıyla elde edilen Execution Plan

Messages kısmı incelediğinde logical reads değerinin 1013 olduğu görülmüyor. Bu, tablonun sorgu için bir kez tarandığını ve 1013 veri sayfasının okunduğunu gösteriyor. Bu, genellikle tablonun tamamının veya büyük bir kısmının okunduğu anlamına gelir ve genellikle performans için istenmeyen bir durumdur.

Execution Plan incelendiğinde, plandaki en yüksek maliyetli operatör, FactInternetSales tablosu üzerindeki Clustered Index Scan. Mevcut bir indeks olan PK_FactInternetSales sorgunun ihtiyaçlarını karşılamak için tamamen taranmak zorunda kalmış.

Planın en üstünde yeşil renkle yazan mesaj: Missing Index (Impact 80.8396): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[FactInternetSales] ([OrderDateKey]) INCLUDE ([ProductKey],[SalesAmount]) SQL Server Query Optimizer

Bu sorgunun performansını %80.8 oranında iyileştirebileceğini düşündüğü bir indeks öneriyor. Öneri, FactInternetSales tablosunda OrderDateKey sütunu üzerine bir NONCLUSTERED indeks oluşturulması ve bu indekse ProductKey ile SalesAmount sütunlarının da dahil edilmesi yönünde.

```
-- Önerilen Nonclustered Indexini oluşturma
USE AdventureWorksDW2022;
GO

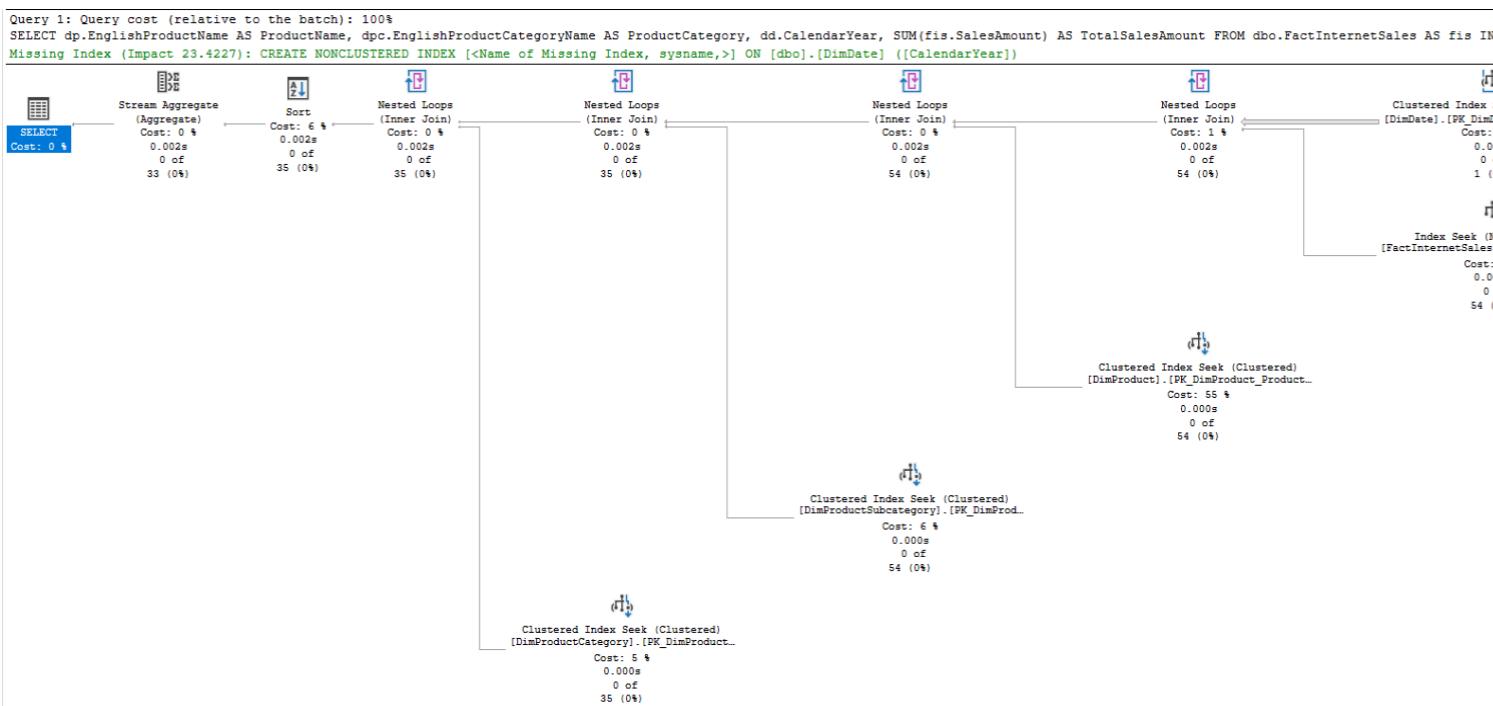
CREATE NONCLUSTERED INDEX [IX_FactInternetSales_OrderDateKey_Includes]
ON [dbo].[FactInternetSales] ([OrderDateKey])
INCLUDE ([ProductKey], [SalesAmount]);
GO
```

Bu sorguyu çalıştırarak bize öneri olarak verilmiş indexi oluşturmuş oluyoruz. Önceki sorguyu tekrar çalıştırarak yine Execution Plan ve Messages kısmını inceleyerek performans artışını doğrulayabiliyoruz.

Önceki ve Sonraki Performansların Karşılaştırılması

| Metrik | İndeks Öncesi | İndeks Sonrası | Değişim | Sonuç |
|------------|---------------|----------------|---------|-----------------|
| Zaman: | | | | |
| CPU Süresi | 0 ms | 16 ms | +16 ms | (Önemsiz Artış) |
| Geçen Süre | 20 ms | 19 ms | -1 ms | (Benzer) |

| Mantıksal Okumalar (IO): | | | | |
|---------------------------|----------------------------|--------------------------------------|------------------------------|----------|
| FactInternetSales | 1013 sayfa | 301 sayfa | -712 sayfa (%70.3 Azalma) | İYİLEŞME |
| DimProduct | 188 sayfa | 2 sayfa | -186 sayfa (%98.9 Azalma) | İYİLEŞME |
| Worktable (tempdb) | 1942 sayfa | 0 sayfa | -1942 sayfa (%100 Azalma) | İYİLEŞME |
| Toplam Mantıksal Okumalar | ~3236 sayfa | ~396 sayfa | ~ -2840 sayfa (%87.8 Azalma) | İYİLEŞME |
| Execution Plan: | | | | |
| FactInternetSales Erişimi | Clustered Index Scan (%72) | NonClustered Index Seek (%35) | Scan -> Seek | İYİLEŞME |
| En Maliyetli Operatör | Clustered Index Scan (%72) | Clustered Index Seek (DimDate) (%35) | Maliyet Azaldı ve Kaydı | İYİLEŞME |
| Eksik İndeks Önerisi | Var (Etki ~%80) | Yok | Öneri Giderildi | İYİLEŞME |
| Worktable Kullanımı | Var (Hash Join/Agg) | Yok (Stream Agg) | Kullanım Kalktı | İYİLEŞME |



İndeks eklenmesi sonrası Execution Plan

Bölüm 3: İndeks Yönetimi

Proje hedeflerimizden biri olan İndeks Yönetimi kapsamında, veritabanındaki önemli ve büyük tablolardan biri olan FactInternetSales için SQL Server'in önerdiği eksik indeksler olup olmadığını kontrol edelim. Aşağıdaki sorgu çalıştırılarak eksik indeks olup olmadığı kontrol edilebilir.

```
USE AdventureWorksDW2022;
GO

SELECT
    db_name(mid.database_id) AS DatabaseName,
    object_schema_name(mid.object_id, mid.database_id) + '.' + object_name(mid.object_id, mid.database_id) AS TableName,
    migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks + migs.user_scans) AS ImprovementMeasure,
    'CREATE NONCLUSTERED INDEX [IX_' + object_name(mid.object_id, mid.database_id) + '_'
        + REPLACE(REPLACE(REPLACE(ISNULL(mid.equality_columns,''),',','_'),'[',''),']',''))
        + CASE WHEN mid.equality_columns IS NOT NULL AND mid.inequality_columns IS NOT NULL THEN '_' ELSE '' END
        + REPLACE(REPLACE(ISNULL(mid.inequality_columns,''),',','_'),'[',''),']','')
        + CASE WHEN mid.included_columns IS NOT NULL THEN '_Includes' ELSE '' END + ']' -- Added _Includes for clarity
    + ' ON ' + mid.statement -- schema.table name
    + '(' + ISNULL (mid.equality_columns,'')
    + CASE WHEN mid.equality_columns IS NOT NULL AND mid.inequality_columns IS NOT NULL THEN ',' ELSE '' END
    + ISNULL (mid.inequality_columns, '') + ')'
    + ISNULL (' INCLUDE (' + mid.included_columns + ')', '') AS CreateIndexStatement,
    migs.user_seeks,
    migs.user_scans,
    migs.last_user_seek,
    migs.last_user_scan,
    mid.equality_columns,
    mid.inequality_columns,
    mid.included_columns
FROM sys.dm_db_missing_index_groups mig
INNER JOIN sys.dm_db_missing_index_group_stats migs ON migs.group_handle = mig.index_group_handle
INNER JOIN sys.dm_db_missing_index_details mid ON mig.index_handle = mid.index_handle
WHERE mid.database_id = DB_ID('AdventureWorksDW2022')
    AND mid.object_id = OBJECT_ID('dbo.FactInternetSales')
ORDER BY ImprovementMeasure DESC;
GO
```

| DatabaseName | TableName | ImprovementMeasure | CreateIndexStatement | user_seeks | user_scans | last_user_seek | last_user_scan | equality_columns | inequality_columns | included_columns |
|--------------|-----------|--------------------|----------------------|------------|------------|----------------|----------------|------------------|--------------------|------------------|
| | | | | | | | | | | |

Sorgu çalıştırıldığında boş bir dönüş alındığı görülüyor. Sorgunun herhangi bir sonuç döndürmemesi, SQL Server'in FactInternetSales tablosu için şu anda eksik olarak değerlendirdiği ve önemli performans artışı sağlayacağına inandığı bir indeks önerisi olmadığı anlamına geliyor.

Kullanılmayan indeksleri belirlemek için aşağıdaki sorguyu çalıştırabiliriz.

```

USE AdventureWorksDW2022;
GO

SELECT
    OBJECT_SCHEMA_NAME(i.object_id) AS SchemaName,
    OBJECT_NAME(i.object_id) AS TableName,
    i.name AS IndexName,
    i.type_desc AS IndexType,
    ius.user_seeks,
    ius.user_scans,
    ius.user_lookups,
    ius.user_updates,
    ius.last_user_seek,
    ius.last_user_scan,
    ius.last_user_lookup,
    (ISNULL(ius.user_seeks, 0) + ISNULL(ius.user_scans, 0) + ISNULL(ius.user_lookups, 0)) AS TotalReads
FROM sys.indexes i
INNER JOIN sys.objects o ON i.object_id = o.object_id
LEFT JOIN sys.dm_db_index_usage_stats ius ON i.object_id = ius.object_id AND i.index_id = ius.index_id AND ius.database_id = DB_ID('AdventureWorksDW2022')
WHERE o.is_ms_shipped = 0
    AND i.type_desc != 'HEAP'
    AND i.is_primary_key = 0
    AND i.is_unique_constraint = 0
    AND OBJECT_NAME(i.object_id) = 'FactInternetSales'
ORDER BY
    TotalReads ASC,
    ius.user_updates DESC;
GO

```

Bu sorguyu çalıştığımızda primary key olmayan sadece bir indeks olduğunu görüyoruz.

| SchemaName | TableName | IndexName | IndexType | user_seeks | user_scans | user_lookups | user_updates | last_user_seek | last_user_scan | last_user_lookup | TotalReads |
|------------|-------------------|---|--------------|------------|------------|--------------|--------------|-------------------------|-------------------------|------------------|------------|
| dbo | FactInternetSales | IX_FactInternetSales_OrderDateKey_Include | NONCLUSTERED | 5 | 1 | 0 | 0 | 2025-04-13 18:40:22.050 | 2025-04-13 20:15:42.950 | NULL | 6 |

Sorgu sonucunda FactInternetSales sistemde kullanım primary olmayan indeksin sistem kaydının olduğunu doğruladı. Dolayısıyla, kullanım eksikliğine dayanarak kaldırılacak bir indeks bulunmuyor.

Son olarak tüm indekslerin kullanım istatistiklerini görüntülemek mantıklı olabilir. Bu sayede kullanılmayan indeksleri silerebiliriz.

Daha önceki adımda kullanılmayan indeksleri ararken Primary Key olanları hariç tutmuştuk, çünkü PK'ler genellikle tablonun temel erişimi için kritik öneme sahiptir. Ancak, tablodaki tüm indekslerin kullanım istatistiklerine bakmak, bize genel tablo erişim aktivitesi hakkında bir fikir verebilir ve belki gözden kaçan bir durumu ortaya çıkarabilir. Aşağıdaki sorgu bize tüm indeksleri veriyor.

```

SELECT
    OBJECT_SCHEMA_NAME(i.object_id) AS SchemaName,
    OBJECT_NAME(i.object_id) AS TableName,
    i.name AS IndexName,
    i.type_desc AS IndexType,
    i.is_primary_key,
    ius.user_seeks,
    ius.user_scans,
    ius.user_lookups,
    ius.user_updates,
    (ISNULL(ius.user_seeks, 0) + ISNULL(ius.user_scans, 0) + ISNULL(ius.user_lookups, 0)) AS TotalReads,
    ius.last_user_seek,
    ius.last_user_scan,
    ius.last_user_lookup,
    ius.last_user_update
FROM sys.indexes i
INNER JOIN sys.objects o ON i.object_id = o.object_id
LEFT JOIN sys.dm_db_index_usage_stats ius ON i.object_id = ius.object_id AND i.index_id = ius.index_id AND ius.database_id = DB_ID('AdventureWorksDW2022')
WHERE o.is_ms_shipped = 0
    AND i.type_desc != 'HEAP'
    AND OBJECT_NAME(i.object_id) = 'FactInternetSales'
ORDER BY
    IndexName;
GO

```

Sonuca bakıldığında her iki indeksin de kullanıldığını yani sistem kaydının bulduğunu görüntüleyebiliyoruz. Bu da hiçbir indeksin silinemeyeceğini gösteriyor.

| | SchemaName | TableName | IndexName | IndexType | is_primary_key | user_seeks | user_scans | user_lookups | user_updates | TotalReads | last_user_seek | last_user_scan | last_user_lookup | last_user_update |
|---|------------|-------------------|---|--------------|----------------|------------|------------|--------------|--------------|------------|-------------------------|-------------------------|------------------|------------------|
| 1 | dbo | FactInternetSales | IX_FactInternetSales_OrderDateKey_Includes | NONCLUSTERED | 0 | 2 | 0 | 0 | 0 | 2 | 2025-04-13 18:22:05.553 | NULL | NULL | NULL |
| 2 | dbo | FactInternetSales | PK_FactInternetSales_SalesOrderNumber_SalesOrder... | CLUSTERED | 1 | 0 | 3 | 0 | 0 | 3 | NULL | 2025-04-13 17:46:01.193 | NULL | NULL |

Bu konuda başka yapabileceğimiz bir şey var mı diye düşünürsek. FactInternetSales tablosundaki indekslerin parçalanma durumunu ve istatistiklerin en son ne zaman güncellendiğini kontrol edebiliriz.

```

USE AdventureWorksDW2022;
GO

|SELECT
    OBJECT_SCHEMA_NAME(ips.object_id) AS SchemaName,
    OBJECT_NAME(ips.object_id) AS TableName,
    i.name AS IndexName,
    ips.index_type_desc,
    ips.avg_fragmentation_in_percent,
    ips.page_count,
    st.name AS StatisticsName,
    STATS_DATE(st.object_id, st.stats_id) AS LastStatsUpdate
FROM sys.dm_db_index_physical_stats(DB_ID(), OBJECT_ID('dbo.FactInternetSales'), NULL, NULL, 'SAMPLED') AS ips
INNER JOIN sys.indexes AS i ON ips.object_id = i.object_id AND ips.index_id = i.index_id
LEFT JOIN sys.stats st ON i.object_id = st.object_id AND i.name = st.name
WHERE ips.page_count > 100
ORDER BY
    ips.avg_fragmentation_in_percent DESC;
GO

```

Bu sorgu FactInternetSales tablosundaki indekslerin ortalama parçalanma yüzdesini ve sayfa sayısını gösterir.

| | SchemaName | TableName | IndexName | index_type_desc | avg_fragmentation_in_percent | page_count | StatisticsName | LastStatsUpdate |
|---|------------|-------------------|---|--------------------|------------------------------|------------|---|-------------------------|
| 1 | dbo | FactInternetSales | PK_FactInternetSales_SalesOrderNumber_SalesOrder... | CLUSTERED INDEX | 0.564971751412429 | 1239 | PK_FactInternetSales_SalesOrderNumber_SalesOrder... | 2017-10-27 14:36:32.340 |
| 2 | dbo | FactInternetSales | IX_FactInternetSales_OrderDateKey_Includes | NONCLUSTERED INDEX | 0 | 307 | IX_FactInternetSales_OrderDateKey_Includes | 2025-04-13 18:03:20.880 |

| | SchemaName | ObjectName | StatisticsName | LastStatsUpdate | auto_created | user_created | no_recompute |
|----|------------|-------------------|--|-------------------------|--------------|--------------|--------------|
| 1 | dbo | FactInternetSales | IX_FactInternetSales_OrderDateKey_Includes | 2025-04-13 18:03:20.880 | 0 | 0 | 0 |
| 2 | dbo | FactInternetSales | _WA_Sys_0000000A_4E88ABD4 | 2017-10-27 14:36:33.610 | 1 | 0 | 0 |
| 3 | dbo | FactInternetSales | _WA_Sys_00000008_4E88ABD4 | 2017-10-27 14:36:33.490 | 1 | 0 | 0 |
| 4 | dbo | FactInternetSales | _WA_Sys_00000006_4E88ABD4 | 2017-10-27 14:36:33.470 | 1 | 0 | 0 |
| 5 | dbo | FactInternetSales | _WA_Sys_00000001_4E88ABD4 | 2017-10-27 14:36:33.450 | 1 | 0 | 0 |
| 6 | dbo | FactInternetSales | _WA_Sys_00000004_4E88ABD4 | 2017-10-27 14:36:33.427 | 1 | 0 | 0 |
| 7 | dbo | FactInternetSales | _WA_Sys_00000003_4E88ABD4 | 2017-10-27 14:36:33.407 | 1 | 0 | 0 |
| 8 | dbo | FactInternetSales | _WA_Sys_00000002_4E88ABD4 | 2017-10-27 14:36:33.387 | 1 | 0 | 0 |
| 9 | dbo | FactInternetSales | _WA_Sys_00000005_4E88ABD4 | 2017-10-27 14:36:33.367 | 1 | 0 | 0 |
| 10 | dbo | FactInternetSales | _WA_Sys_00000007_4E88ABD4 | 2017-10-27 14:36:33.343 | 1 | 0 | 0 |
| 11 | dbo | FactInternetSales | PK_FactInternetSales_SalesOrderNumber_... | 2017-10-27 14:36:32.340 | 0 | 0 | 0 |

PK_FactInternetSales için p arçalanma %0.56. Bu çok düşük bir oran.

Sayfa Sayısı (page_count): 1238

İstatistik Güncellemeye (LastStatsUpdate): 2017-10-27 14:30.

IX_FactInternetSales

Parçalanma: %0. İndeks yeni oluşturulduğu için parçalanma yok, bu beklenen bir durum.

Sayfa Sayısı (page_count): 387.

İstatistik Güncellemeye (LastStatsUpdate): 2025-04-13 18:08.

Genellikle %5'in altındaki parçalanma oranları iyi kabul edilir ve müdahale etmeye gerek olmaz.

```
-- İstatistik güncellemesi
USE AdventureWorksDW2022;
GO

UPDATE STATISTICS dbo.FactInternetSales WITH FULLSCAN;
GO

-- Tüm İndekslerin Kullanım İstatistiklerini Görüntüleme
USE AdventureWorksDW2022;
GO
```

İki indeks arasındaki yıl farkının çok olması sebebiyle eski istatistikleri güncellemek için yukarıdaki sorgu çalıştırılması gereklidir. Eski istatistikler, Sorgu İyileştirici'nin sorgular için verimsiz yürütme planları seçmesine neden olabilir.

Veri Yöneticisi Rollerı

Son adımıımız olan veri yönetici rollerini gerçekleştirmek için de birtakım sorgular çalıştırabiliriz.

```
USE AdventureWorksDW2022;
GO

CREATE ROLE AnalystReadOnly;
GO

CREATE ROLE ETLOperator;
GO
```

Bu sorgu veritabanımızda AnalystReadOnly ve ETLOperator isimli iki boş rol oluşturur.

```
USE AdventureWorksDW2022;
GO

GRANT SELECT ON SCHEMA::dbo TO AnalystReadOnly;
GO

GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::dbo TO ETLOperator;
GO
```

Bu sorgu, AnalystReadOnly rolüne dbo şemasındaki her şey üzerinde sadece SELECT yapma hakkı, ETLOperator rolüne ise SELECT, INSERT, UPDATE, DELETE yapma hakkı tanıyacaktır.

```
CREATE LOGIN SampleAnalystLogin WITH PASSWORD = 'YourStrongPasswordHere';
GO
```

```
CREATE LOGIN SampleETLLogin WITH PASSWORD = 'YourStrongPasswordHere';
GO
```

```
USE AdventureWorksDW2022;
GO

CREATE USER SampleAnalystUser FOR LOGIN SampleAnalystLogin;
GO

CREATE USER SampleETLUser FOR LOGIN SampleETLLogin;
GO

ALTER ROLE AnalystReadOnly ADD MEMBER SampleAnalystUser;
GO

ALTER ROLE ETLOperator ADD MEMBER SampleETLUser;
GO
```

Yukarıdaki sorgunun çalıştırılmasından sonra

SampleAnalystLogin ile sunucuya bağlanan bir kişi, AdventureWorksDW2022 veritabanında SampleAnalystUser olarak tanınacak ve AnalystReadOnly rolünün izinlerine (yani dbo şeması üzerinde sadece SELECT) sahip olacaktır. Bu kullanıcı INSERT, UPDATE, DELETE yapmaya çalıştığında hata alacaktır.

SampleETLLogin ile sunucuya bağlanan bir kişi/uygulama, SampleETLUser olarak tanınacak ve ETLOperator rolünün izinlerine (yani dbo şeması üzerinde SELECT, INSERT, UPDATE, DELETE) sahip olacaktır.

```
USE AdventureWorksDW2022;
GO

DENY SELECT ON dbo.DimEmployee TO AnalystReadOnly;
GO
```

Bu sorgu çalıştırıldıkten sonra, AnalystReadOnly rolüne atanmış bir kullanıcı dbo.DimEmployee tablosundan veri çekmeye çalıştığında artık izin hatası alacaktır, ancak dbo şemasındaki diğer tablolardan veri çekmeye devam edebilecektir.

```
USE AdventureWorksDW2022;
GO

SELECT
    princ.name AS RoleName,
    perm.permission_name,
    perm.state_desc,
    perm.class_desc,
    CASE
        WHEN perm.major_id = 0 THEN 'Database'
        WHEN perm.class_desc = 'SCHEMA' THEN SCHEMA_NAME(perm.major_id)
        WHEN perm.class_desc = 'OBJECT_OR_COLUMN' THEN OBJECT_SCHEMA_NAME(perm.major_id) + '.' + OBJECT_NAME(perm.major_id)
        ELSE CAST(perm.major_id AS VARCHAR)
    END AS ObjectOrSchemaName
FROM sys.database_permissions AS perm
JOIN sys.database_principals AS princ
    ON perm.grantee_principal_id = princ.principal_id
WHERE
    princ.name = 'AnalystReadOnly'
ORDER BY
    perm.class_desc,
    ObjectOrSchemaName,
    perm.permission_name;
GO
```

AnalystReadOnly rolünde hangi izinlerin atandığını görmek için yukarıdaki sorgu çalıştırılabilir.

| | RoleName | permission_name | state_desc | class_desc | ObjectOrSchemaName |
|---|-----------------|-----------------|------------|------------------|--------------------|
| 1 | AnalystReadOnly | SELECT | DENY | OBJECT_OR_COLUMN | dbo.DimEmployee |
| 2 | AnalystReadOnly | SELECT | GRANT | SCHEMA | dbo |

Yukarıdaki sonuçta da görüldüğü gibi mevcut izinler atadığımız gibi.

Proje 2: Veritabanı Yedekleme ve Felaketten Kurtarma Planı

Kullanılan veri tabanı: Top 1000 Most Played Spotify Songs of All Time

Link: <https://www.kaggle.com/datasets/kunalgp/top-1000-most-played-spotify-songs-of-all-time?resource=download>

Bu projede istenen adımlar 4 temel başlık altında değerlendirilmiş:

- 1- Tam, Artık, Fark Yedeklemeleri
- 2- Zamanlayıcılarla Yedekleme
- 3- Felaketten Kurtarma Senaryoları
- 4- Test Yedekleme Senaryoları

Biz de yaptığımız adımları bu başlıklar üzerinde değerlendirecek olursak.

Bölüm 1: Tam, Artık, Fark Yedeklemeleri

Öncelikle işe Docker üzerinden gerekli container’ı oluşturarak başladık.

Terminal üzerinden aşağıdaki kodu çalıştırıldıkten sonra uygulamamıza bağlantı sağlamış olduk.

```
docker run -e "MSSQL_AGENT_ENABLED=true" -e "ACCEPT_EULA=1" -e "MSSQL_SA_PASSWORD=MyStrongPass123" -e "MSSQL_PID=Developer" -e "MSSQL_USER=SA" -p 1433:1433 -d --name=sql mcr.microsoft.com/azure-sql-edge
```

Kendimize çalışmak için ‘spotify’ adında boş bir veritabanı oluşturduk.

Sonrasında ise indirdiğimiz veri setini ‘Import Wizard’ komutuyla ekleyerek ‘.csv’ formatındaki dosyamızı tablo şeklinde sisteme aktardık.

Tüm Veritabanını (Full Backup) Yedekleme

```
[1] 1 BACKUP DATABASE spotify
 2 TO DISK = '/var/opt/mssql/backup/spotify_full.bak'
 3 WITH INIT,
 4     NAME = 'Full Backup of spotify',
 5     STATS = 10;
```

Bu adımdan sonra başarılı bir şekilde **Tam Yedeklememizi** kaydetmiş olduk.

Fark Yedekleme (Differential Backup)

```
[11] 1 BACKUP DATABASE spotify
2 TO DISK = '/var/opt/mssql/backup/spotify_diff.bak'
3 WITH DIFFERENTIAL,
4      INIT,
5      NAME = 'Differential Backup of spotify',
6      STATS = 10;
7
```

SQL

Burada da **Fark Yedeklememizi** alıp istediğimiz konuma kaydettik.

Artık Yedeklemeler (Transaction Log Backup)

```
[6] 1 ALTER DATABASE spotify SET RECOVERY FULL;
```

SQL

Commands completed successfully.

Total execution time: 00:00:00.037

```
[8] 1 SELECT name, recovery_model_desc
2 FROM sys.databases
3 WHERE name = 'spotify';
```

SQL

(1 row affected)

Total execution time: 00:00:00.010

| | name | recovery_model_desc |
|---|---------|---------------------|
| 1 | spotify | FULL |

Artık Yedekleme yaparken ‘recovery’ durumumuzu öncelikle ‘FULL’ durumuna getirdik ve kontrol ettik. Bunu yapmadığımız zamanda ise hata aldığımızı gözlemledik.

```
[13] 1 BACKUP LOG spotify
2 TO DISK = '/var/opt/mssql/backup/spotify_log.trn'
3 WITH INIT,
4      NAME = 'First Transaction Log Backup of spotify',
5      STATS = 10;
```

SQL

100 percent processed.

Processed 2 pages for database 'spotify', file 'spotify_log' on file 1.

BACKUP LOG successfully processed 2 pages in 0.005 seconds (2.343 MB/sec).

Total execution time: 00:00:00.125

Bölüm 2: Zamanlayıcı ile Yedekleme Otomasyonu

Bu adıma geldiğimizde elimizde iki seçenek vardı. İlk olarak SQL Server Agent kullanarak bir Job tanımladık, sonrasında ise bu işlemleri aynı zamanda Cron Job ile de yapabildiğimiz gösterdik.

İlk adımdan başlarsak:

Öncelikle Agent'ımızın durumunu kontrol ederek başladık. Aşağıda bulunan komutu çalıştırduğumızda sonucun ‘running’ durumunda olduğunu gördük. Bu da bizim istediğimiz sonuçtu. Sonrasında da kendimiz için bir Job tanımı oluşturduk.

```
[15] 1  SELECT servicename, status_desc
2  FROM sys.dm_server_services
3  WHERE servicename LIKE '%SQL Server Agent%';

(1 row affected)

Total execution time: 00:00:00.053

+-----+-----+
| servicename | status_desc |
+-----+-----+
| SQL Server Agent (MSSQLSERVER) | Running |
+-----+-----+
```

SQL Server Agent ile devam etmek için Server Agent Extension ile yeni bir Job adımı oluşturacağız.

```
[16] 1  USE msdb;
2  GO
3
4  EXEC dbo.sp_add_job
5    @job_name = N'SpotifyDB_Backup_Job';
6  GO
7
8  EXEC sp_add_jobstep
9    @job_name = N'SpotifyDB_Backup_Job',
10   @step_name = N'Backup Spotify Database',
11   @subsystem = N'TSQL',
12   @command = N'BACKUP DATABASE [spotify] TO DISK = ''/var/opt/mssql/backups/spotify_${ESCAPE_SQUOTE(DATE)}.bak'' WITH COMPRESSION, STATS =
13   @database_name = N'master';
14  GO
```

Oluşturduğumuz bu Job ile günlük her sabah 08:00 saatinde backup olmasını istedik

```
[17] 1 EXEC dbo.sp_add_schedule
2      @schedule_name = N'Daily_8AM',
3      @freq_type = 4, -- Daily
4      @freq_interval = 1, -- Every day
5      @active_start_time = 080000; -- 8:00 AM
6 GO
7
8 EXEC sp_attach_schedule
9      @job_name = N'SpotifyDB_Backup_Job',
10     @schedule_name = N'Daily_8AM';
11 GO
```

^

Commands completed successfully.

Commands completed successfully.

Total execution time: 00:00:00.142

```
[18] 1 EXEC dbo.sp_add_jobserver
2      @job_name = N'SpotifyDB_Backup_Job';
3 GO
```

^

Commands completed successfully.

Total execution time: 00:00:00.322

Docker Volume ile yedeklerin kalıcı olması sağlanıyor.

```
docker run -e "MSSQL_AGENT_ENABLED=true" -d -p 1433:1433 --name sql_server_container -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=MyStrongPass123" -v
sql_backups:/var/opt/mssql/backups mcr.microsoft.com/mssql/server:2019-latest
```

Alternatif olarak aynı işlemi Cron Job ile de yapabiliriz. Aşağıdaki kodu "/usr/local/bin/backup_spotify.sh" dosyası olarak kaydedeceğiz.

```
CONTAINER_NAME="sql_server_container"
DB_USER="sa"
DB_PASS="MyStrongPass123"
DB_NAME="spotify"
BACKUP_DIR="/var/opt/mssql/backups/"
BACKUP_FILE="${BACKUP_DIR}/spotify_$(date +%Y%m%d_%H%M%S).bak"
docker exec $CONTAINER_NAME /opt/mssql-tools/bin/sqlcmd -S localhost -U $DB_USER -P $DB_PASS -Q "BACKUP DATABASE [$DB_NAME] TO DISK =
N'/var/opt/mssql/backups/spotify.bak' WITH COMPRESSION, STATS = 10"
docker cp $CONTAINER_NAME:/var/opt/mssql/backups/spotify.bak $BACKUP_FILE
find $BACKUP_DIR -name "spotify_*bak" -type f -mtime +30 -exec rm {} \;
echo "Backup completed: $BACKUP_FILE"
```

Terminalimize de sırasıyla şu adımları yazacağız.

- chmod +x /usr/local/bin/backup_spotify.sh
- crontab -e
- 0 8 * * * /usr/local/bin/backup_spotify.sh >> /var/log/spotify_backup.log 2>&1

Gerçekleştirilen Cron Job işlemlerine ait Google Drive Dosyasının linki : <https://drive.google.com/drive/folders/1FT7GK8yYIzd8SG7d3MNlk-71xEzUuSy?usp=sharing>

Cron Job ile yaptığımız işlemler burada yer alıyor. Kodların başarılı bir şekilde çalıştığını gösteren kodların ekran fotoğrafları da

[“<https://drive.google.com/drive/folders/1FT7GK8yYIzd8SG7d3MNLk-71x0EzUuSy?usp=sharing>” linkinde yer alıyor.](https://drive.google.com/drive/folders/1FT7GK8yYIzd8SG7d3MNLk-71x0EzUuSy?usp=sharing)

Son olarak şuana kadar oluşturduğumu Job ve kendimiz çalıştırarak aldığımız backup dosyalarını kontrol edelim.

```
[26] 1  SELECT
2    database_name,
3    backup_start_date,
4    backup_finish_date,
5    type,
6    physical_device_name,
7    backup_size/1024/1024 AS backup_size_mb
8  FROM msdb.dbo.backupset bs
9  JOIN msdb.dbo.backupmediafamily bmf ON bs.media_set_id = bmf.media_set_id
10 WHERE database_name = 'spotify'
11 ORDER BY backup_start_date DESC;
```

(4 rows affected)

Total execution time: 00:00:00.029

| database_name | backup_start_date | backup_finish_date | type | physical_device_name | backup_size_mb |
|---------------|-------------------------|-------------------------|------|---|----------------|
| 1 spotify | 2025-04-23 12:01:43.000 | 2025-04-23 12:01:43.000 | D | /var/opt/mssql/backups/spotify_20250423.bak | 3.39843750000 |
| 2 spotify | 2025-04-23 11:49:23.000 | 2025-04-23 11:49:23.000 | L | /var/opt/mssql/backup/spotify_log.trn | 0.07812500000 |
| 3 spotify | 2025-04-23 11:49:04.000 | 2025-04-23 11:49:04.000 | I | /var/opt/mssql/backup/spotify_diff.bak | 1.33593750000 |
| 4 spotify | 2025-04-23 11:40:44.000 | 2025-04-23 11:40:44.000 | D | /var/opt/mssql/backup/spotify_full.bak | 3.33593750000 |

Başarılı bir şekilde hepsinin listelendiğini görebiliyoruz.

Bölüm 3: Felaketten Kurtarma Senaryoları

İlk olarak elimizde bulunan veritabanının aniden silindiği durumu baz alalım. Veritabanının tüm bağlantılarını kopararak silme işlemi gerçekleştiriyoruz.

```
[28] 1  -- Veritabanındaki tüm bağlantıları sonlandır
2 USE master;
3 GO
4 ALTER DATABASE spotify SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
5 GO
6
7 -- Şimdi veritabanını silebilirsiniz
8 DROP DATABASE spotify;
9 GO
```

Commands completed successfully.

Sonrasında sırasıyla aşağıdaki komutları çalıştırıyoruz.

```
[33] 1  RESTORE DATABASE spotify
2    FROM DISK = '/var/opt/mssql/backup/spotify_full.bak'
3    WITH REPLACE, NORECOVERY;
```

```
[34] 1 RESTORE DATABASE spotify
2 FROM DISK = '/var/opt/mssql/backup/spotify_diff.bak'
3 WITH NORECOVERY;
```

^

```
[35] 1 RESTORE LOG spotify
2 FROM DISK = '/var/opt/mssql/backup/spotify_log.trn'
3 WITH NORECOVERY;
```

^

```
[36] 1 RESTORE DATABASE spotify
```

▽

RESTORE DATABASE successfully processed 0 pages in 0.366 seconds (0.000 MB/sec).

Total execution time: 00:00:00.386

Son komuttan sonra başarılı bir şekilde ‘restore’ işlemimizi gerçekleştirdiğimizi görmüş olduk.

```
[37] 1 EXEC msdb.dbo.sp_help_job;
```

Commands completed successfully.

Total execution time: 00:00:00.217

| job_id | originating_server | name | enabled | description | start_step_id |
|--|--------------------|----------------------|---------|---------------------------|---------------|
| 1 0933f684-b0f2-4976-9a27-e6a3bfb5ebcb | D2AC8703F7AC | SpotifyDB_Backup_Job | 1 | No description available. | 1 |

Son adımlara geldiğimizde eskiden tanımlamış olduğumuz Job’ları listeliyoruz. Bu sayede çalışması duran Job olup olmadığını kontrol edebiliyoruz.

```
[38] 1 EXEC msdb.dbo.sp_update_job
2 @job_name = N'SpotifyDB_Backup_Job',
3 @enabled = 1;
```

^

Commands completed successfully.

Bu kod sayesinde de duran Job’ları tekrardan aktif hale getirebiliyoruz.

Point-in-time Restore

```
[40] 1 RESTORE DATABASE spotify
2 FROM DISK = '/var/opt/mssql/backup/spotify_full.bak'
3 WITH STOPAT = '2025-04-23 12:00:00', RECOVERY;
```

^

Son olarak da Point-in-time Restore kullanarak istediğimiz tarihteki bir kayıt zamanına ulaşarak o tarihteki verileri tekrardan elde etmiş oluyoruz.

Bölüm 4: Test Yedekleme Senaryoları

Bu adımda da ‘Data Mirroring’ üzerinde durduk.

Mirroring işlemi için bize 2 adet birbiriyle iletişim kurabilen server gerekiyordu. Bunu terminalimizde yazdığımız kodlar aracılığıyla Docker üzerinden oluşturduk. Oluşturduğumuz bu serverları ‘Primary’ ve ‘Mirroring’ olarak adlandırdık. Bu adıma dair kod aşağıda bulunmaktadır.

```
docker network create sqlnet

docker run --platform linux/amd64 -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=YourStrong!Passw0rd' \
-p 1433:1433 --name sql_primary --network sqlnet -d mcr.microsoft.com/mssql/server:2019-latest

docker run --platform linux/amd64 -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=YourStrong!Passw0rd' \
-p 1434:1433 --name sql_mirror --network sqlnet -d mcr.microsoft.com/mssql/server:2019-latest
```

Bu kodu çalıştırıldığımızda aşağıda gözüken containerlarımız oluşmuş ve çalışmaya başlamış oldu.



Sonrasında Primary'e bağlı olan serverda aşağıda bulunan kodları çalıştırındık.

```
[2] 1 ALTER DATABASE spotify_mirroring SET RECOVERY FULL;
2 GO
```

Commands completed successfully.

Total execution time: 00:00:00.020

```
[3] 1 BACKUP DATABASE spotify_mirroring
2 TO DISK = '/var/opt/mssql/backup/MirrorTestDB.bak'
3 WITH FORMAT;
4 GO
```

Processed 408 pages for database 'spotify_mirroring', file 'spotify_mirroring' on file 1.

Processed 2 pages for database 'spotify_mirroring', file 'spotify_mirroring_log' on file 1.

BACKUP DATABASE successfully processed 410 pages in 0.042 seconds (76.171 MB/sec).

Total execution time: 00:00:00.242

Başarılı bir şekilde tamamlandıktan sonra Mirroring server'ına geçtik.

```
[2] 1 RESTORE DATABASE spotify_mirroring  
2 FROM DISK = '/var/opt/mssql/backup/MirrorTestDB.bak'  
3 WITH NORECOVERY;  
4 GO  
5
```

^

Processed 408 pages for database 'spotify_mirroring', file 'spotify_mirroring' on file 1.

Processed 2 pages for database 'spotify_mirroring', file 'spotify_mirroring_log' on file 1.

RESTORE DATABASE successfully processed 410 pages in 0.018 seconds (177.734 MB/sec).

Total execution time: 00:00:00.397

Orada da gerçekleştirdiğimiz bu işlemle başarılı bir sonuç elde ettik.

Tekrardan Primary server'ına dönüp aşağıdaki kodu çalıştırıldık ve aynı adımı Mirror'da tekrarladık.

```
[4] 1 CREATE ENDPOINT [Mirroring]  
2 STATE = STARTED  
3 AS TCP (LISTENER_PORT = 5022)  
4 FOR DATABASE_MIRRORING (ROLE=PARTNER);  
5 GO  
6
```

^

Commands completed successfully.

Total execution time: 00:00:02.080

```
1 ALTER DATABASE spotify_mirroring  
2 SET PARTNER = 'TCP://sql_primary:5022';  
3 GO
```

^

Bu adımlar da başarılı bir şekilde tamamlandıktan sonra Data Mirroring işlemimizi başarıyla tamamlamış olduk.

Proje 3: Veritabanı Güvenliği ve Erişim Kontrolü

Kullanılan veritabanı: Wide World Importers sample database v1.0

Link:<https://github.com/Microsoft/sql-server-samples/releases/tag/wide-world-importers-v1.0>

Bu projede istenen adımlar 4 temel başlık altında değerlendirilmiş:

- 1- Erişim Yönetimi
- 2- Veri Şifreleme
- 3- SQL Injection Testleri
- 4- Audit Logları

Biz de yaptığımız adımları bu başlıklar üzerinde değerlendirecek olursak.

Bölüm 1: Erişim Yönetimi

Mevcut Sunucu Girişlerini İnceleme işlemi ile başlayabiliriz. Aşağıdaki sorgu, sunucu seviyesindeki girişleri ve türlerini listeleyecektir.

```
SELECT
    name AS LoginAdi,
    type_desc AS LoginTipi,
    is_disabled,
    create_date AS OlusturmaTarihi,
    default_database_name AS VarsayılanVeritabani
FROM
    sys.server_principals
WHERE
    type IN ('U', 'G', 'S')
    AND name NOT LIKE '##%'
ORDER BY
    LoginTipi,
    LoginAdi;
```

Bu sorgunun sonucunu aşağıda görüntüleyebiliriz. Beklendiği gibi hem SQL_LOGIN hem de WINDOWS_LOGIN, kullanıcının kendi hesabı ve çeşitli NT servis hesabı tiplerinde girişlerin bulunduğu gösteriyor.

100 %

Results Messages

| | LoginAdi | LoginTipi | is_disabled | OlusturmaTarihi | VarsayılanVeritabani |
|----|--------------------------------------|---------------|-------------|-------------------------|----------------------|
| 1 | sa | SQL_LOGIN | 1 | 2003-04-08 09:10:35.460 | master |
| 2 | SampleAnalystLogin | SQL_LOGIN | 0 | 2025-04-13 20:47:18.837 | master |
| 3 | SampleETLLogin | SQL_LOGIN | 0 | 2025-04-13 20:47:18.840 | master |
| 4 | BUILTIN\Users | WINDOWS_GROUP | 0 | 2025-03-13 22:13:00.850 | master |
| 5 | DESKTOP-OS7IBMJ\Melih | WINDOWS_LOGIN | 0 | 2025-03-13 22:13:00.827 | master |
| 6 | NT AUTHORITY\SYSTEM | WINDOWS_LOGIN | 0 | 2025-03-13 22:13:00.857 | master |
| 7 | NT SERVICE\SQLTELEMETRY\\$SQLEXPRESS | WINDOWS_LOGIN | 0 | 2025-03-13 22:13:01.670 | master |
| 8 | NT SERVICE\SQLWriter | WINDOWS_LOGIN | 0 | 2025-03-13 22:13:00.833 | master |
| 9 | NT SERVICE\Winmgmt | WINDOWS_LOGIN | 0 | 2025-03-13 22:13:00.840 | master |
| 10 | NT Service\MSSQL\$SQLEXPRESS | WINDOWS_LOGIN | 0 | 2025-03-13 22:13:00.847 | master |

Şimdi SQL Server Authentication kullanarak yeni bir sunucu girişü (login) oluşturalım. Aşağıdaki sorguyu çalıştırarak bu işlemi sağlayabiliriz.

```
CREATE LOGIN OrnekSqlKullanici
WITH PASSWORD = 'Sifre123!',
DEFAULT_DATABASE = WideWorldImporters,
CHECK_POLICY = ON,
CHECK_EXPIRATION = ON;
GO
```

Aşağıdaki sorguyu çalıştırarak login oluşumu başarılı olup olmadığını kontrol edelim.

```
SELECT
    name AS LoginAdi,
    type_desc AS LoginTipi,
    default_database_name AS VarsayılanVeritabani
FROM
    sys.server_principals
WHERE
    name = 'OrnekSqlKullanici';
GO
```

Bu sorgu sonucunda aşağıdaki görüldüğü gibi işlemin başarılı olduğunu görüyoruz.

Results Messages

| | LoginAdi | LoginTipi | VarsayılanVeritabani |
|---|-------------------|-----------|----------------------|
| 1 | OrnekSqlKullanici | SQL_LOGIN | WideWorldImporters |

Aşağıdaki sorguyu çalıştırarak bir veritabanı kullanıcısı oluşturabiliriz.

```
--Veritabanı Kullanıcısı Oluşturma
CREATE USER OrnekSqlKullanici_User FOR LOGIN OrnekSqlKullanici;
GO
SELECT
    name AS VeritabaniKullaniciAdi,
    type_desc AS KullaniciTipi,
    authentication_type_desc AS KimlikDogrulamaTipi
FROM
    sys.database_principals
WHERE
    name = 'OrnekSqlKullanici_User';
GO
```

Aşağıdaki sorguyu çalıştırarak veritabanı kullanıcımıza yetki verebiliriz. Bu sayede hangi kısımlara erişip erişemeyeceğini belirlemiştir.

```
GRANT SELECT ON Website.Customers TO OrnekSqlKullanici_User;
GO

SELECT
    princ.name AS KullaniciAdi,
    perm.permission_name AS Yetki,
    perm.state_desc AS YetkiDurumu, -- GRANT (izin verildi), DENY (yasaklandı)
    obj.name AS NesneAdi,
    sch.name AS SemaAdi
FROM
    sys.database_permissions AS perm
INNER JOIN
    sys.database_principals AS princ ON perm.grantee_principal_id = princ.principal_id
INNER JOIN
    sys.objects AS obj ON perm.major_id = obj.object_id
INNER JOIN
    sys.schemas AS sch ON obj.schema_id = sch.schema_id
WHERE
    princ.name = 'OrnekSqlKullanici_User' AND obj.name = 'Customers' AND sch.name = 'Website';
GO
```

Bu sorguları çalıştırıldığımızda aşağıdaki sonuca ulaşırız. Bu da istenen yetkilerin kullanıcıya sağladığını gösterir.

| | Results | Messages | | Results | Messages | | | | |
|---|------------------------|---------------|-------------------|--------------|------------------------|-------------|----------|-----------|---------|
| | VeritabaniKullaniciAdi | KullaniciTipi | KimlikDogrulamaTi | KullaniciAdi | Yetki | YetkiDurumu | NesneAdi | SemaAdi | |
| 1 | OrnekSqlKullanici_User | SQL_USER | INSTANCE | 1 | OrnekSqlKullanici_User | SELECT | GRANT | Customers | Website |

Verdiğimiz yetkileri doğru şekilde kullanabiliyor muyuz bunu denemek için sorgu çalıştırıralım. Aşağıdaki sorguları çalıştırıldığımızda ilk sorgunun çalıştığını fakat ikinci sorgunun hata verdiğiini görürüz

```
--Login ve Yetkiyi Test Etme

--izin verilen soru
USE WideWorldImporters;
SELECT TOP 5 CustomerID, CustomerName FROM Website.Customers;
GO

--izin verilmeyen soru

USE WideWorldImporters;
SELECT TOP 5 FullName FROM Application.People;
GO
```

Bu adımları gerçekleştirdikten sonra Windows login için de bir kullanıcı oluşturalım ve erişimlerini test edelim.

DESKTOP kullanıcısı için login'i için WideWorldImporters veritabanı içinde bir kullanıcı oluşturalım.

Aşağıdaki sorgu ile kullanıcımıza yetki verelim.

```
CREATE USER Melih_User FOR LOGIN [DESKTOP\Melih];
GO

SELECT
    name AS VeritabaniKullaniciAdi,
    type_desc AS KullaniciTipi,
    authentication_type_desc AS KimlikDogrulamaTipi
FROM
    sys.database_principals
WHERE
    name = 'Melih_User';
GO

GRANT SELECT ON Application.People TO Melih_User;
GO

SELECT
    princ.name AS KullaniciAdi,
    perm.permission_name AS Yetki,
    perm.state_desc AS YetkiDurumu,
    obj.name AS NesneAdi,
    sch.name AS SemaAdi
FROM
    sys.database_permissions AS perm
INNER JOIN
    sys.database_principals AS princ ON perm.grantee_principal_id = princ.principal_id
INNER JOIN
    sys.objects AS obj ON perm.major_id = obj.object_id
INNER JOIN
    sys.schemas AS sch ON obj.schema_id = sch.schema_id
WHERE
    princ.name = 'Melih_User' AND obj.name = 'People';
GO
```

Aşağıdaki sorguyu çalıştırarak izinleri test edebiliriz. Fakat bu sorgular doğru çalışmayacaktır. Çünkü biz admin pc üzerinden çalıştığımız için mevcut kısıtlamalar işe yaramaz ve istediğimiz yere erişebiliriz.

```

-- yetkinin testi için deneme sorguları
-- izin verilen sorgu
SELECT TOP 5 PersonID, FullName FROM Application.People;
GO
-- izin verilmeyen sorgu
SELECT TOP 5 CustomerID, CustomerName FROM Website.Customers;
GO
-- izin verilmeyen sorgu
SELECT TOP 5 OrderID FROM Sales.Orders;
GO

```

Bu yüzden başka bir kullanıcımış gibi davranışarak test etmeliyiz. Aşağıdaki sorgu bunu gerçekleştirir. Tam anlamıyla kullanıcımış gibi davranışarak sorguları çalıştırır. Bu sayede kısıtlamalara tabii oluruz.

```

EXECUTE AS USER = 'Melih_User';

-- Bu sorgu çalışmalı
PRINT 'Application.People sorgusu deneniyor (Başarılı olmalı):';
SELECT TOP 5 PersonID, FullName FROM Application.People;

-- Bu sorgu HATA VERMELİ
PRINT 'Website.Customers sorgusu deneniyor (Hata vermeli):';
BEGIN TRY
    SELECT TOP 5 CustomerID, CustomerName FROM Website.Customers;
    PRINT '>> BEKLENMEYEN BAŞARI: Website.Customers sorgusu çalıştı!';
END TRY
BEGIN CATCH
    PRINT '>> BEKLENEN HATA ALINDI: ' + ERROR_MESSAGE();
END CATCH
REVERT;
GO

```

Bölüm 2: Veri Şifreleme

Veri şifrelemenin temel amacı, veritabanında saklanan hassas bilgilerin yetkisiz erişime karşı korunmasıdır. Özellikle veritabanı dosyalarının (.mdf, .ldf) veya yedeklerin fiziksel olarak ele geçirilmesi durumunda, verilerin okunamaz halde olmasını sağlamayı hedefler (at-rest encryption).

Aşağıdaki sorguyu çalıştırarak Sales.CustomerCategories tablosuna, şifrelenmiş kategori adlarını binary formatta saklamak üzere EncryptedCategoryName adında yeni bir sütun ekledik.

```

]IF COL_LENGTH('Sales.CustomerCategories', 'EncryptedCategoryName') IS NULL
]BEGIN
]    ALTER TABLE Sales.CustomerCategories
]        ADD EncryptedCategoryName VARBINARY(MAX) NULL;
]        PRINT 'EncryptedCategoryName sütunu eklendi.';
]END
ELSE
]BEGIN
]    PRINT 'EncryptedCategoryName sütunu zaten var.';
]END
GO

```

Aşağıdaki sorgu ile CategoryName_SMKey adında yeni bir simetrik şifreleme anahtarını oluşturduk. Şifreleme algoritması olarak AES_256'yi seçtik. Bu yeni anahtarın kendisini de güvenli bir şekilde saklamak için, veritabanında daha önceden var olan TDE_Cert isimli sertifikayı kullanarak şifreledik.

```

]IF NOT EXISTS (SELECT * FROM sys.symmetric_keys WHERE name = 'CategoryName_SMKey'
]BEGIN
]    CREATE SYMMETRIC KEY CategoryName_SMKey
]        WITH ALGORITHM = AES_256
]        ENCRYPTION BY CERTIFICATE TDE_Cert;
]        PRINT 'CategoryName_SMKey simetrik anahtarı oluşturuldu.';
]END
ELSE
]BEGIN
]    PRINT 'CategoryName_SMKey simetrik anahtarı zaten var.';
]END
GO

```

Aşağıdaki sorgu ile. İlk 3 kategoriye ait EncryptedCategoryName sütunları, orijinal kategori adlarının şifrelenmiş binary halleriyle güncellendi. Bu sayede örnek bir şifreleme gerçekleştirmiş olduk.

```

]OPEN SYMMETRIC KEY CategoryName_SMKey
]    DECRYPTION BY CERTIFICATE TDE_Cert;

]UPDATE Sales.CustomerCategories
SET EncryptedCategoryName = ENCRYPTBYKEY(KEY_GUID('CategoryName_SMKey'), CAST(CustomerCategoryName AS VARBINARY(8000)))
WHERE CustomerCategoryID <= 3 AND CustomerCategoryName IS NOT NULL;

CLOSE SYMMETRIC KEY CategoryName_SMKey;
GO

```

Bu sayede şifreleme işlemlerimizi tamamlamış olduk ve son bir sorgu ile istediğimiz veriler gerçekten şifrelenmiş mi diye kontrol edebiliriz. Aşağıdaki sorgunun sonucunu inceleyelim.

```

OPEN SYMMETRIC KEY CategoryName_SMKey
DECRYPTION BY CERTIFICATE TDE_Cert;

SELECT
    CustomerCategoryID,
    CustomerCategoryName AS OrjinalKategoriAdı,
    EncryptedCategoryName AS SifreliVeri,
    CONVERT(NVARCHAR(50), DECRYPTBYKEY(EncryptedCategoryName)) AS CozulmusKategoriAdı
FROM Sales.CustomerCategories
WHERE CustomerCategoryID <= 3;

CLOSE SYMMETRIC KEY CategoryName_SMKey;
GO

```

Aşağıdaki görselde de görüldüğü gibi şifrelemiş olduğumuz sütunumuzun hem şifreli verisi hem de çözülmüş kategorisi doğru şekilde görünmekte. Bu da şifreleme işlemimizin başarıyla gerçekleştiğini gösteriyor.

| | CustomerCategoryID | OrjinalKategoriAdı | SifreliVeri | CozulmusKategoriAdı |
|---|--------------------|--------------------|---|---------------------|
| 1 | 1 | Agent | 0x00FBA550CCC63C4DB06B161093C6C0F60200000048BE68... | Agent |
| 2 | 2 | Wholesaler | 0x00FBA550CCC63C4DB06B161093C6C0F6020000005C016E... | Wholesaler |
| 3 | 3 | Novelty Shop | 0x00FBA550CCC63C4DB06B161093C6C0F602000000C24A30... | Novelty Shop |

Bölüm 3: SQL Injection Testleri

İlk olarak bilerek güvenlik açıkları içeren bir soru yazarak çalıştırıyoruz.

```

EXECUTE AS USER = 'Melih_User';

-- Bu soru çalışmalı
PRINT 'Application.People sorgusu deneniyor (Başarılı olmalı)';
SELECT TOP 5 PersonID, FullName FROM Application.People;

-- Bu soru HATA VERMELİ
PRINT 'Website.Customers sorgusu deneniyor (Hata vermelii)';
BEGIN TRY
    SELECT TOP 5 CustomerID, CustomerName FROM Website.Customers;
    PRINT '>> BEKLENMEYEN BAŞARI: Website.Customers sorgusu çalıştı!';
END TRY
BEGIN CATCH
    PRINT '>> BEKLENEN HATA ALINDI: ' + ERROR_MESSAGE();
END CATCH
REVERT;
GO

```

Şimdi bunun üzerine bir saldırı denemesi yapıyoruz. İlk olarak normal şekilde bir sorgulama işlemi yaparak saldırı denemesiyle olan farkını karşılaşacağız. İkinci sorguda daha fazla satır olmasını bekliyoruz çünkü bir güvenlik zafiyeti verdik ve bu zafiyetten faydalananarak erişmememiz gereken verilere erişebileceklerini test edeceğiz.

```
--normal çalışma denemesi
EXEC dbo.FindCustomers_Vulnerable @CustomerNameFragment = N'Tailspin';

--saldırı simülasyonu
EXEC dbo.FindCustomers_Vulnerable @CustomerNameFragment = N''' OR 1=1 --';
```

110 %

Results Messages

| CustomerID | CustomerName | PhoneNumber |
|------------|------------------------------|----------------|
| 197 | Tailspin Toys (Magalia, CA) | (209) 555-0100 |
| 198 | Tailspin Toys (Buell, MO) | (314) 555-0100 |
| 199 | Tailspin Toys (Antonito, CO) | (303) 555-0100 |
| 200 | Tailspin Toys (Tooele, UT) | (385) 555-0100 |
| 201 | Tailspin Toys (Skyway, WA) | (206) 555-0100 |

| CustomerID | CustomerName | PhoneNumber |
|------------|------------------|----------------|
| 653 | Sylvie Laramée | (787) 555-0100 |
| 654 | Ian Olofsson | (339) 555-0100 |
| 655 | Luis Saucedo | (210) 555-0100 |
| 656 | Emma Salpa | (205) 555-0100 |
| 657 | Adriana Pena | (252) 555-0100 |
| 658 | Kalyani Benjaree | (212) 555-0100 |
| 659 | Ganesh Majumdar | (217) 555-0100 |
| 660 | Jaroslav Fisar | (215) 555-0100 |
| 661 | Jibek Juniskzy | (217) 555-0100 |
| 662 | Anand Mudaliyar | (206) 555-0100 |
| 663 | Agrita Abele | (206) 555-0100 |

Bu durumu çözmek için güvenli prosedür yani parametreli sorgu oluşturulabilir. Aşağıdaki sorgu ile bu işlemi gerçekleştiriyoruz.

```
CREATE OR ALTER PROCEDURE dbo.FindCustomers_Safe
    @CustomerNameFragment NVARCHAR(100)
AS
BEGIN
    SELECT CustomerID, CustomerName, PhoneNumber
    FROM Website.Customers
    WHERE CustomerName LIKE '%' + @CustomerNameFragment + '%';
END
GO
```

Tekrardan güvenli ve güvensiz şekilde erişme testlerini yaptığımızda artık güvensiz olan sorgunun bir sonuç döndürmediğini görüyoruz. Bir önceki sorguda yaptığımız işlem başarıyla gerçekleşmiş.

Results Messages

| CustomerID | CustomerName | PhoneNumber |
|------------|--------------|-------------|
|------------|--------------|-------------|

Bölüm 4: Audit Logları

SQL Server Audit, sunucu ve veritabanı seviyesinde gerçekleşen belirli olayları (eylemleri) izlemenizi ve kaydetmenizi sağlayan güçlü bir özelliktir. Bu özellik sayesinde “kim, ne zaman, ne yaptı?” sorularının cevaplarını bulabilirsiniz.

İlk adım, denetim kayıtlarımızın nereye yazılacağını belirleyen ana Audit nesnesini oluşturmaktır. Kayıtları genellikle bir dosyaya yazmak en esnek yöntemdir.

Aşağıdaki iki sorguyu çalıştırarak bir audit nesnesi oluşturmak mümkün. Bunun için ilk önce, arka planda çalışan SQL Server Veritabanı Altyapısı Hizmetini çalıştırın Windows kullanıcı hesabına audit işlemleri için yetki vermek gerekiyor. Bizim yetkili olmamız yeterli olmuyor. Kaydedeceği klasörü oluşturup okuma yazma ve hatta değiştirme yetkisi vermek gerekiyor.

Bu işlemleri yaptıktan sonra aşağıdaki iki sorguyu çalıştırarak audit nesnesi oluşturulabilir.

```
--Audit nesnesi oluşturma
USE master;
GO
IF NOT EXISTS (SELECT * FROM sys.server_audits WHERE name = 'OrnekDB_Audit')
BEGIN
    CREATE SERVER AUDIT OrnekDB_Audit
    TO FILE (FILEPATH = 'C:\test\'')
    WITH (QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE);
    PRINT 'Audit nesnesi oluşturuldu.';
END
ELSE
BEGIN
    PRINT 'Audit nesnesi "OrnekDB_Audit" zaten var.';
END
GO
--Audit nesnesini aktif etme
USE master;
GO
IF EXISTS (SELECT * FROM sys.server_audits WHERE name = 'OrnekDB_Audit' AND is_state_enabled = 0)
BEGIN
    ALTER SERVER AUDIT OrnekDB_Audit WITH (STATE = ON);
    PRINT 'Audit nesnesi etkinleştirildi.';
END
ELSE IF EXISTS (SELECT * FROM sys.server_audits WHERE name = 'OrnekDB_Audit' AND is_state_enabled = 1)
BEGIN
    PRINT 'Audit nesnesi zaten etkin durumda.';
END
ELSE
BEGIN
    PRINT 'HATA: OrnekDB_Audit isimli Audit nesnesi bulunamadı!';
END
GO
```

Daha sonra mevcut audit üzerinde bir spesifikasyon oluşturarak aktif edebiliriz. Aşağıdaki iki sorgu bunu sağlar.

```

USE WideWorldImporters;
GO
-- Yeni spesifikasyonu oluşturalım
IF EXISTS (SELECT * FROM sys.database_audit_specifications WHERE name = 'OrnekDB_Audit_Spec')
BEGIN
    ALTER DATABASE AUDIT SPECIFICATION OrnekDB_Audit_Spec WITH (STATE = OFF);
    DROP DATABASE AUDIT SPECIFICATION OrnekDB_Audit_Spec;
    PRINT 'Mevcut "OrnekDB_Audit_Spec" spesifikasyonu silindi.';
END

CREATE DATABASE AUDIT SPECIFICATION OrnekDB_Audit_Spec
FOR SERVER AUDIT OrnekDB_Audit

ADD (SELECT ON OBJECT::Website.Customers BY OrnekSqlKullanici_User),
ADD (DELETE ON OBJECT::Sales.CustomerCategories BY public)
WITH (STATE = OFF);
GO

-- Spesifikasyonun etkin olup olmadığını kontrol edip değilse etkinleştirelim
IF EXISTS (SELECT * FROM sys.database_audit_specifications WHERE name = 'OrnekDB_Audit_Spec' AND is_state_enabled = 0)
BEGIN
    ALTER DATABASE AUDIT SPECIFICATION OrnekDB_Audit_Spec WITH (STATE = ON);
    PRINT 'Veritabanı Audit Spesifikasyonu etkinleştirildi.';
END
ELSE IF EXISTS (SELECT * FROM sys.database_audit_specifications WHERE name = 'OrnekDB_Audit_Spec' AND is_state_enabled = 1)
BEGIN
    PRINT 'Veritabanı Audit Spesifikasyonu zaten etkin durumda.';
END
ELSE
BEGIN
    PRINT 'HATA: OrnekDB_Audit_Spec isimli spesifikasyon bulunamadı!';
END

```

Bu sorgunun sonucunda aktifleştirildi bildirimini alarak işlemleri başarıyla tamamladığımızı anlayabiliriz. Son olarak Log kaydı oluşması için örnek bir SELECT ve DELETE sorgusu çalıştıralım. Bu işlemi yaptığımızda audit nesnesinin bağlı olduğu ilgili dosyaya log kaydedilecektir.

```

EXECUTE AS USER = 'OrnekSqlKullanici_User';

SELECT TOP 1 CustomerID, CustomerName FROM Website.Customers;

REVERT;
GO

DECLARE @TestCategoryName NVARCHAR(50) = 'Silinecek Denetim Kategorisi';

IF NOT EXISTS (SELECT 1 FROM Sales.CustomerCategories WHERE CustomerCategoryName = @TestCategoryName)
BEGIN
    INSERT INTO Sales.CustomerCategories (CustomerCategoryName, LastEditedBy) VALUES (@TestCategoryName, 1);
    PRINT @TestCategoryName + ' eklendi.';
END
ELSE
BEGIN
    PRINT @TestCategoryName + ' zaten vardi.';
END

PRINT @TestCategoryName + ' siliniyor...';
DELETE FROM Sales.CustomerCategories
WHERE CustomerCategoryName = @TestCategoryName;

IF @@ROWCOUNT > 0
    PRINT 'Test kategorisi silindi (Bu DELETE loglanmış olmalı).';
ELSE
    PRINT 'Silinecek test kategorisi bulunamadı.';
GO

```

Görüldüğü üzere deneme sorgularımızı çalıştırıldıkten sonra log kaydımız oluşmuş. Son olarak içeriğini görüntüleyerek sağlamasını yapalım.

| Local Disk (C:) > test | | | |
|--|--------------------|---------------|------|
| Name | Date modified | Type | Size |
| OrnekDB_Audit_38AE34F1-6389-436F-BD... | 4/23/2025 10:54 PM | SQLAUDIT File | 0 KB |

Aşağıda görüldüğü gibi sorguyu çalıştırduğumızda log sonuçlarını ve nelerin kayıt edildiğini gözlemliyoruz.

```
--audit loglarının kontrolü
SELECT
    event_time,
    action_id,
    succeeded,
    session_server_principal_name AS LoginName,
    database_principal_name AS DatabaseUserName,
    server_instance_name AS ServerName,
    database_name AS DatabaseName,
    schema_name AS SchemaName,
    object_name AS ObjectName,
    statement AS SqlStatement -- Çalıştırılan sorgu
FROM
    sys.fn_get_audit_file ('C:\test\OrnekDB_Audit_* .sqlaudit', default, default);
GO
```

Results Messages

| | event_time | action_id | succeeded | LoginName | DatabaseUserName | ServerName | DatabaseName | SchemaName | ObjectName | SqlStatement |
|---|-----------------------------|-----------|-----------|-------------|------------------------|-------------|--------------------|------------|--------------------|---|
| 1 | 2025-04-23 19:54:23.1000375 | AUSC | 1 | DESKTOP-... | DESKTOP-... | | | | | |
| 2 | 2025-04-23 20:03:10.0073207 | SL | 1 | DESKTOP-... | OrnekSqlKullanicı_User | DESKTOP-... | WideWorldImporters | Website | Customers | SELECT TOP 1 CustomerID, CustomerName FROM Website... |
| 3 | 2025-04-23 20:03:20.4645984 | DL | 1 | DESKTOP-... | dbo | DESKTOP-... | WideWorldImporters | Sales | CustomerCategories | DELETE FROM Sales.CustomerCategories WHERE Custo... |

Bu sayede audit log oluşturma ve gözleme işlemini de başarıyla tamamladık.

Proje 4: Veritabanı Yük Dengeleme ve Dağıtık Veritabanı Yapıları

Kullanılan veri tabanı: Powerlifting Database

Link: <https://www.kaggle.com/datasets/open-powerlifting/powerlifting-database?resource=download>

Bu projede istenen adımlar 3 temel başlık altında değerlendirilmiştir:

- 1- Veritabanı Replikasyonu
- 2- Yük Dengeleme
- 3- Failover Senaryoları

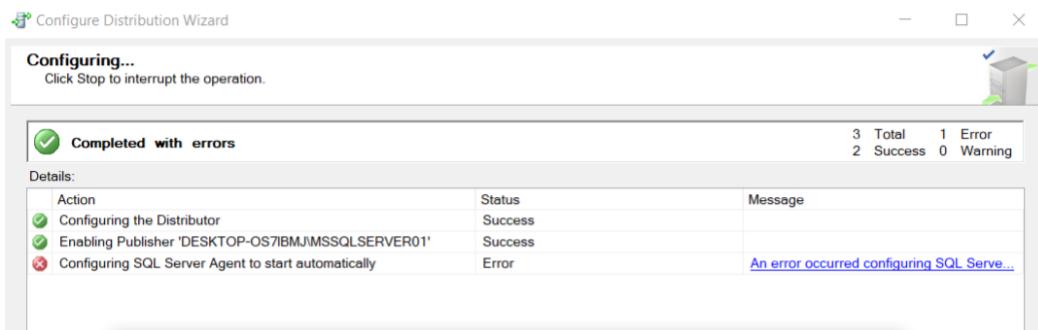
Biz de yaptığımız adımları bu başlıklar üzerinde değerlendirecek olursak.

Bölüm 1: SQL Server Replikasyonu ile Veritabanı Çoğaltma

Bu bölüm, SQL Server'ın yerel Replikasyon özelliklerini kullanarak veritabanı çoğaltma sürecinin kurulmasına odaklanmaktadır. Veritabanı replikasyonu, veri ve veritabanı nesnelerini bir kaynak veritabanından bir veya daha fazla hedef veritabanına kopyalamak ve ardından tutarlılığı korumak için bu veritabanları arasında senkronizasyon sağlamak amacıyla kullanılan bir teknolojidir. Bu bölüm, sağlam bir replikasyon topolojisi uygulamak için gerekli olan temel planlama, yapılandırma ve doğrulama aşamalarını detaylandıracaktır. Belirtilen prosedürler, veri yedekliliği sağlamak, ölçülebilir raporlama yeteneklerini kolaylaştmayı veya verileri uzak ofislere dağıtmayı amaçlayacaktır.

Adım 1.1: Adım 1: Ortam Hazırlığı ve Distributor Yapılandırması

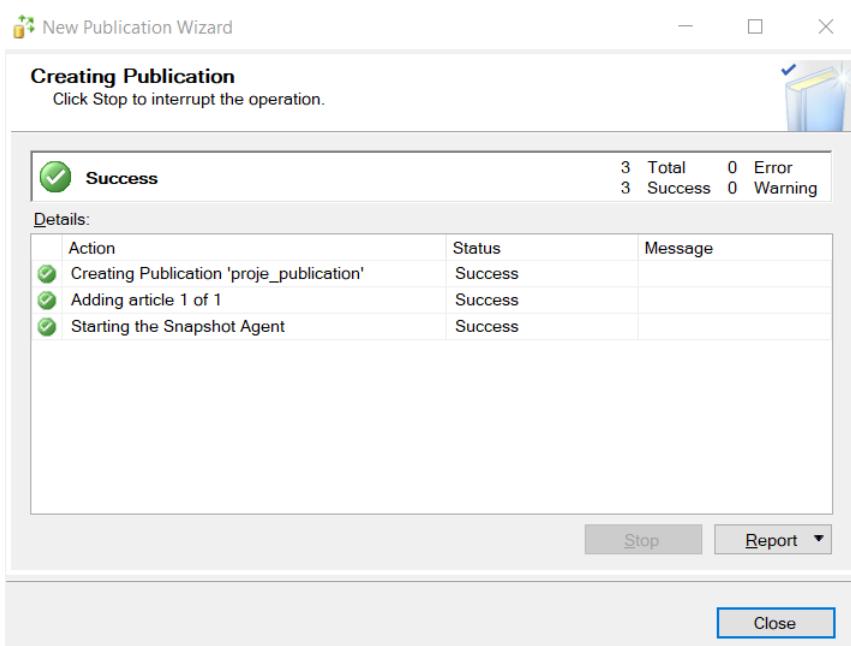
Projenin ilk etabında, Microsoft SQL Server üzerinde veri replikasyon altyapısının tesis edilmesi amacıyla Distributor rolünün konfigürasyonu gerçekleştirılmıştır. Bu kapsamında, Publisher olarak atanan sunucu üzerinde SQL Server Management Studio kullanılarak Configure Distribution Wizard sihirbazı çalıştırılmış; bu işlem neticesinde replikasyon metaverilerinin ve işlem kayıtlarının tutulacağı distribution veritabanı tesis edilmiş ve ilgili sunucunun Yayıncı olarak yetkilendirilmesi sağlanmıştır.



Buradaki söz konusu error durumu son aşama olarak MSSQLSERVER01 için SQL Server Agent'in otomatik olarak başlatılamadığını söylüyor. SQL Server Configuration Manager üzerinden manuel olarak kendi elimizle başlattığımızda bu sorunu ortadan kaldırmamız oldukça basit.

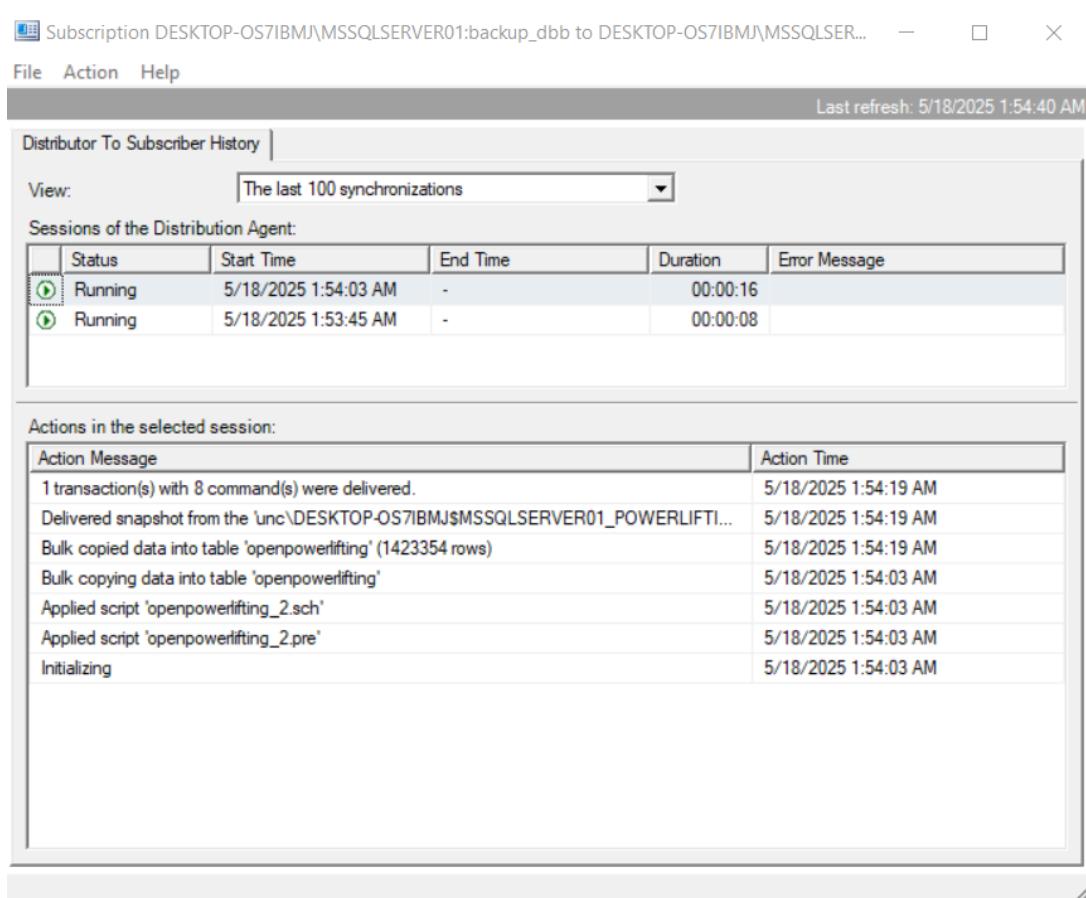
Adım 1.2: Publication Oluşturma

Projenin ikinci adımında, veri replikasyonuna kaynak teşkil edecek olan powerlifting_db isimli veritabanı üzerinde bir yayın tanımlama süreci icra edilmiştir. Bu kapsamda, yayın türü olarak Snapshot Publication seçilmiş ve kaynak veritabanında bulunan tüm tablolar, replike edilecek articles olarak yayına dahil edilmiştir. Snapshot Agent'ın güvenlik yapılandırması, SQL Server Agent servis hesabının yetkileri altında çalışacak şekilde belirlenmiştir. Bu işlemlerin başarıyla tamamlanması neticesinde, powerlifting_db veritabanındaki veriler, Subscriber sunuculara dağıtılmaya hazır hale getirilmiş ve böylece replikasyon mimarisinin temel bir bileşeni olan yayın başarıyla oluşturulmuştur.



Adım 1.3: Subscription Oluşturma

Projenin ilk bölümünün son aşamasında, powerlifting_db veritabanı için oluşturulan Anlık Görüntü Yayınmasına (Snapshot Publication) bir abonelik (Subscription) tanımlanmıştır. Bu süreçte, başlangıçta farklı bir sunucu (MSSQLSERVER02) üzerinde hedeflenen abonelik yapılandırmasında anlık görüntü klasörüne erişimle ilgili bir sorunla karşılaşılmış; bu durum, Abone'nin Yayıncı ve Dağıtıcı ile aynı sunucu olan MSSQLSERVER01 üzerine alınması ve backup_dbb adında yeni bir abonelik veritabanı oluşturulması stratejisiyle çözümlenmiştir. "Push" dağıtım yöntemi ve "sürekli çalışma" (run continuously) senkronizasyon zamanlaması ile yapılandırılan abonelik neticesinde, Dağıtım Agent'i (Distribution Agent) tarafından kaynak veritabanındaki şema ve yaklaşık 1.4 milyon satırlık veri, backup_dbb hedef veritabanına başarıyla aktarılmış, bu durum hem Replication Monitor üzerinden hem de kullanıcı tarafından hedef veritabanı içeriğinin doğrudan incelenmesiyle teyit edilmiştir. Böylelikle, SQL Server Replication kullanılarak veri çoğaltma ve temel senkronizasyonun sağlanması hedefi başarıyla yerine getirilmiş ve projenin ilk ana bölümünü tamamlanmıştır.



Bölüm 2: Yük Dengeleme

Projenin ikinci bölümü, Microsoft SQL Server ortamında öncelikli olarak yüksek erişilebilirlik) sağlamak amacıyla, öğrenme ve karşılaştırma hedefleri doğrultusunda Database Mirroring teknolojisinin incelenmesi ve yapılandırılmasına odaklanacaktır. Bu kapsamda, Principal ve Mirror SQL Server örnekleri üzerinde özel yansıtma uç noktalarının Endpoints tanımlanması, yansıtılacak veritabanının Full Recovery Model'ine getirilip ilk tam yedeğin alınarak ayna sunucusuna NORECOVERY seçeneğiyle geri yüklenmesi ve ardından senkron veya asenkron çalışma modlarından biri seçilerek yansıtma oturumunun başlatılması adımları uygulamalı olarak gerçekleştirilecektir. Ayrıca, automatic failover senaryoları için istege bağlı bir Witness sunucu yapılandırması da değerlendirilecektir. Yük dengeleme hedefi için ise, doğrudan sorgulanamayan ayna veritabanı üzerinde Database Snapshots oluşturularak bu statik kopyalar üzerinden okuma işlemleri yapılması ve bu yaklaşımın getirdiği avantaj ve kısıtlamalar incelenecektir. Bu bölümün temel amacı, veritabanı servis sürekliliğini artırmak için alternatif bir yüksek erişilebilirlik çözümünü deneyimlemek ve yük dengeleme kavramını bu teknolojinin sunduğu dolaylı yöntemler çerçevesinde ele almaktır.

Adım 2.1: Ön Koşullar ve Ortam Hazırlığı

Database Mirroring oturumunu başlatmadan önce, Principal sunucu olarak görev yapacak MSSQLSERVER01 SQL Server örneği üzerindeki powerlifting_db veritabanının yansıtılmaya uygun hale getirilmesi amacıyla gerekli yedekleme işlemleri gerçekleştirilmiştir. Bu kapsamda, veritabanının kurtarma modelinin FULL olduğu teyit edildikten sonra, T-SQL komutları kullanılarak öncelikle veritabanının güncel bir full backup alınmıştır. Tam yedekleme işleminin başarıyla tamamlanmasının hemen ardından, yansıtma için kesintisiz bir günlük zinciri oluşturmak amacıyla aynı veritabanının bir transaction log backup alınmıştır.

```
SELECT name, recovery_model_desc
FROM sys.databases
WHERE name = 'powerlifting_db';
GO

BACKUP DATABASE powerlifting_db
TO DISK = N'C:\Backups\powerlifting_db_FULL_YENI_DENEME.bak'
WITH
    FORMAT,
    INIT,
    NAME = N'powerlifting_db - Yeni Deneme Full Backup',
    STATS = 10,
    CHECKSUM;
GO

BACKUP LOG powerlifting_db
TO DISK = N'C:\Backups\powerlifting_db_LOG_YENI_DENEME.trn'
WITH
    FORMAT,
    INIT,
    NAME = N'powerlifting_db - Yeni Deneme Log Backup',
    STATS = 10,
    CHECKSUM;
GO
```

Birincil sunucuda powerlifting_db veritabanı için gerekli yedekleme işlemleri tamamlandıktan sonra, Mirror sunucu olarak görev yapacak MSSQLSERVER02 SQL Server örneği üzerinde bu veritabanı yansıtma için özenle hazırlanmıştır. Bu amaçla, MSSQLSERVER01 üzerinden alınan tam yedek dosyası, MSSQLSERVER02'nin ilgili veri ve günlük dosyası yolları belirtilerek ve veritabanını daha sonraki işlem günlüğü yedeklerini kabul edebilir durumda bırakmak için T-SQL komutları aracılığıyla WITH NORECOVERY seçeneğiyle geri yüklenmiştir. Tam yedek geri yüklemesinin başarıyla tamamlanmasını takiben, yine MSSQLSERVER01'den alınan işlem günlüğü yedeği de aynı şekilde WITH NORECOVERY seçeneğiyle MSSQLSERVER02'deki powerlifting_db veritabanı üzerine geri yüklenmiştir. Bu işlemler neticesinde, powerlifting_db veritabanı MSSQLSERVER02 üzerinde Restoring durumuna getirilerek yansıtma oturumunun başlatılması için temel zemin oluşturulmuştur.

```

:RESTORE DATABASE powerlifting_db
FROM DISK = N'C:\Backups\powerlifting_db_FULL_YENI_DENEME.bak'
WITH
    MOVE N'powerlifting_db' TO N'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER02\MSSQL\DATA\powerlifting_db.mdf',
    MOVE N'powerlifting_db_log' TO N'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER02\MSSQL\DATA\powerlifting_db_log.ldf',
    NORECOVERY,
    NOUNLOAD,
    REPLACE,
    STATS = 10;
GO

:RESTORE LOG powerlifting_db
FROM DISK = N'C:\Backups\powerlifting_db_LOG_YENI_DENEME.trn'
WITH
    NORECOVERY,
    NOUNLOAD,
    STATS = 10;
GO

```

Birincil powerlifting_db veritabanının yedeklenmesi ve bu yedeklerin Ayna sunucudaki powerlifting_db veritabanına Restoring durumunda başarıyla geri yüklenmesiyle her iki veritabanı da yansıtma için hazır hale getirilmiştir. Bu hazırlıkların ardından, Principal sunucu olarak görev yapacak MSSQLSERVER01 ile Mirror sunucu MSSQLSERVER02 arasında yansıtma trafigi için güvenli ve özel bir iletişim kanalı tesis etmek amacıyla Endpoints yapılandırılmıştır. Her iki SQL Server örneği üzerinde de T-SQL komutları aracılığıyla, varsayılan olarak TCP port 5022 üzerinden dinleme yapacak şekilde, PARTNER rolüyle ve AES algoritması kullanılarak şifrelenmiş DATABASE_MIRRORING için özelleşmiş endpoint'ler oluşturulmuş ve STARTED durumuna getirilmiştir. Yapılan kontrol sorguları, her iki endpoint'in de aktif olduğunu ve yansıtma oturumuna katılmak üzere doğru rol ve port ayarlarıyla çalıştığını teyit etmiştir. Bu yapılandırma, Birincil ve Ayna sunucuların yansıtma verilerini güvenli bir şekilde senkronize edebilmesi için gerekli olan temel iletişim altyapısını tamamlamıştır.

```

:IF NOT EXISTS (SELECT 1 FROM sys.database_mirroring_endpoints WHERE name = 'Mirroring_Endpoint_MSSQLSERVER01')
BEGIN
    CREATE ENDPOINT Mirroring_Endpoint_MSSQLSERVER01
    STATE = STARTED
    AS TCP (LISTENER_PORT = 5022, LISTENER_IP = ALL)
    FOR DATABASE_MIRRORING (
        ROLE = PARTNER,
        ENCRYPTION = REQUIRED ALGORITHM AES
    );
    PRINT 'Mirroring_Endpoint_MSSQLSERVER01 başarıyla oluşturuldu veya zaten vardı.';
END
ELSE
BEGIN
    PRINT 'Mirroring_Endpoint_MSSQLSERVER01 zaten mevcut.';
END
GO

```

```

IF NOT EXISTS (SELECT 1 FROM sys.database_mirroring_endpoints WHERE name = 'Mirroring_Endpoint_MSSQLSERVER02')
BEGIN
    CREATE ENDPOINT Mirroring_Endpoint_MSSQLSERVER02
        STATE = STARTED
        AS TCP (LISTENER_PORT = 5023, LISTENER_IP = ALL)
        FOR DATABASE_MIRRORING (
            ROLE = PARTNER,
            ENCRYPTION = REQUIRED ALGORITHM AES
        );
        PRINT 'Mirroring_Endpoint_MSSQLSERVER02 başarıyla oluşturuldu veya zaten vardi.';
END
ELSE
BEGIN
    PRINT 'Mirroring_Endpoint_MSSQLSERVER02 zaten mevcut.';
END
GO

```

Adım 2.2: Veritabanı Yansıtma Oturumunu Başlatma

Birincil ve Ayna sunucular arasında güvenli endpoint iletişimini tesis edildikten sonra, her bir SQL Server örneğinin altında çalışan servis hesaplarının, karşısındaki sunucunun yansıtma endpoint'ine bağlanabilmesi için gerekli kimlik doğrulama ve yetkilendirme adımları atılmıştır. Bu doğrultuda, MSSQLSERVER01 SQL Server örneği üzerinde, MSSQLSERVER02 örneğinin servis hesabı için bir Windows kimlik doğrulamalı SQL Server oturum açma hesabı oluşturulmuştur. Benzer şekilde, MSSQLSERVER02 SQL Server örneği üzerinde de MSSQLSERVER01 örneğinin servis için bir login tanımlanmıştır. Bu login'ler oluşturulduktan sonra, T-SQL komutları aracılığıyla her bir sunucudaki yansıtma endpoint'i üzerinde, karşı sunucunun ilgili servis hesabı login'ine CONNECT izni verilmiştir. Bu işlemler neticesinde, her iki SQL Server örneğinin servis hesapları, yansıtma oturumunu kurmak ve sürdürmek için birbirlerinin endpoint'lerine güvenli bir şekilde erişim yetkisine sahip olmuştur.

```

CREATE LOGIN [NT SERVICE\SQL$MSSQLSERVER01] FROM WINDOWS;
PRINT '[NT SERVICE\SQL$MSSQLSERVER01] login''i MSSQLSERVER02 üzerinde oluşturuldu.';
GO

GRANT CONNECT ON ENDPOINT::Mirroring_Endpoint_MSSQLSERVER02 TO [NT SERVICE\SQL$MSSQLSERVER01];
PRINT '[NT SERVICE\SQL$MSSQLSERVER01] login''ine Mirroring_Endpoint_MSSQLSERVER02 için CONNECT izni verildi.';
GO

CREATE LOGIN [NT SERVICE\MSSQL$MSSQLSERVER02] FROM WINDOWS;
PRINT '[NT SERVICE\MSSQL$MSSQLSERVER02] login''i MSSQLSERVER01 üzerinde oluşturuldu.';
GO

GRANT CONNECT ON ENDPOINT::Mirroring_Endpoint_MSSQLSERVER01 TO [NT SERVICE\MSSQL$MSSQLSERVER02];
PRINT '[NT SERVICE\MSSQL$MSSQLSERVER02] login''ine Mirroring_Endpoint_MSSQLSERVER01 için CONNECT izni verildi.';
GO

```

Tüm ön hazırlıklar tamamlandıktan sonra, powerlifting_db veritabanı için yansıtma oturumu Principal sunucusu olan MSSQLSERVER01 üzerinden başlatılmıştır. Bu amaçla, MSSQLSERVER01'e, Mirror sunucusunun MSSQLSERVER02 örneğindeki

5022 numaralı portta dinleyen endpoint olduğu bildirilmiştir. Bu komutun başarıyla icra edilmesiyle birlikte, MSSQLSERVER01 ve MSSQLSERVER02 arasında powerlifting_db veritabanı için bir yansıtma ortaklıği kurulmuş ve Yüksek Güvenlik Modunda senkronizasyon başlamıştır. Sonuç olarak, MSSQLSERVER01 üzerindeki powerlifting_db veritabanı Principal, Synchronized durumuna, MSSQLSERVER02 üzerindeki powerlifting_db veritabanı ise Mirror, Synchronized / Restoring durumuna geçerek, veritabanı için yüksek erişilebilirlik altyapısının ilk adımı başarıyla tesis edilmiştir. Bu aşamada, Birincil veritabanında yapılan tüm değişiklikler anlık olarak Ayna veritabanına yansıtılmaya başlanmıştır.

```
ALTER DATABASE powerlifting_db
SET PARTNER = N'TCP://DESKTOP-0S7IBMJ:5022';
GO
```

Bölüm 3: Failover Senaryoları

Projenin üçüncü ve son bölümü, önceki aşamada powerlifting_db veritabanı için Microsoft SQL Server üzerinde MSSQLSERVER01 Principal ve MSSQLSERVER02 Mirror sunucuları arasında başarıyla tesis edilen Database Mirroring yapılandırması temel alınarak, sistemin sürekliliğini ve veri bütünlüğünü güvence altına almak amacıyla çeşitli failover stratejilerinin uygulanmasına ve yönetimsel işlemlerin gerçekleştirilmesine odaklanacaktır. Bu bölümde, planlı bakım veya kontrollü rol değişimi senaryoları için T-SQL komutları kullanılarak Manual Failover işleminin nasıl yapılacağı ve sunucu rollerinin nasıl değiştirileceği adım adım ele alınacaktır. Ayrıca, Principal sunucunun beklenmedik bir şekilde hizmet dışı kalması durumunda, Mirror sunucusunun Forced Service yöntemiyle, potansiyel veri kaybı riskleri de dikkate alınarak, servisi nasıl devralacağı ve veritabanının nasıl erişilebilir hale getirileceği incelenecaktır. Ek olarak, sistemin kesintilere karşı otonom bir yanıt verme kapasitesini artırmak amacıyla bir Witness sunucunun yapılandırılması ve Database Mirroring oturumunun sonlandırılması gibi kritik yönetimsel işlemler de T-SQL düzeyinde detaylandırılacaktır.

Adım 3.1: Mevcut Veritabanı Yansıtma Durumunun Teyit Edilmesi

Database Mirroring oturumunun sağlıklı bir şekilde kurulduğunu ve powerlifting_db veritabanı için rollerin bekendiği gibi yapılandırıldığını doğrulamak amacıyla, hem Principal hem de Mirror sunucu örnekleri üzerinde sys.database_mirroring sistem

kataloğu görünümüne yönelik bir T-SQL sorgusu çalıştırılmıştır. Bu sorgu, yansıtma yapılandırmalarındaki her bir veritabanının rolünü, durumunu ve ortak bilgilerini detaylı bir şekilde raporlamaktadır.

```
SELECT
    DB_NAME(database_id) AS DatabaseName,
    mirroring_role_desc AS Role,
    mirroring_state_desc AS State,
    mirroring_partner_name AS PartnerName,
    mirroring_partner_instance AS PartnerInstance,
    mirroring_witness_name AS WitnessName,
    mirroring_witness_state_desc AS WitnessState
FROM sys.database_mirroring
WHERE DB_NAME(database_id) = 'powerlifting_db';
```

Sorgunun çıktısı aşağıda gibidir. Doğru sonuçlar alındığı görülmektedir.

| | DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|---|-----------------|-----------|--------------|----------------------------|------------------------------|-------------|--------------|
| 1 | powerlifting_db | PRINCIPAL | SYNCHRONIZED | TCP://DESKTOP-OS7IBMJ:5024 | DESKTOP-OS7IBMJMSSQLSERVER02 | NULL | NULL |

| | DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|---|-----------------|--------|--------------|----------------------------|------------------------------|-------------|--------------|
| 1 | powerlifting_db | MIRROR | SYNCHRONIZED | TCP://DESKTOP-OS7IBMJ:5023 | DESKTOP-OS7IBMJMSSQLSERVER01 | NULL | NULL |

Adım 3.2: Manual Failover İşleminin Gerçekleştirilmesi

Database Mirroring oturumunun sağlıklı ve SYNCHRONIZED olduğu Adım 3.1'de teyit edildikten sonra, planlı bir rol değişimi senaryosu uygulamak amacıyla Manuel Failover işlemi gerçekleştirilmiştir. Bu işlem, mevcut Principal sunucu olan MSSQLSERVER01 üzerinde, powerlifting_db veritabanı için ALTER DATABASE powerlifting_db SET PARTNER FAILOVER; T-SQL komutunun yürütülmesiyle başlatılmıştır. Komutun başarıyla tamamlanması neticesinde, MSSQLSERVER01 sunucusu Mirror rolünü üstlenirken, MSSQLSERVER02 sunucusu Principal rolünü alarak powerlifting_db veritabanına hizmet vermeye başlamıştır. Bu rol değişimi sırasında herhangi bir veri kaybı yaşanmamış ve veritabanı erişilebilirliği kesintisiz bir şekilde devam etmiştir. İşlem sonrasında sys.database_mirroring üzerinden yapılan durum kontrolleri, rollerin beklediği gibi değiştiğini ve oturumun yeni rollerle SYNCHRONIZED durumda sağlıklı bir şekilde devam ettiğini doğrulamıştır.

```
USE master;
GO
ALTER DATABASE powerlifting_db
    SET PARTNER FAILOVER;
GO
```

| DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|-----------------|--------|--------------|----------------------------|-----------------------------|-------------|--------------|
| powerlifting_db | MIRROR | SYNCHRONIZED | TCP://DESKTOP-OS7IBMJ:5024 | DESKTOP-OS7IBMJ\SQLSERVER02 | NULL | NULL |

| DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|-----------------|-----------|--------------|----------------------------|-----------------------------|-------------|--------------|
| powerlifting_db | PRINCIPAL | SYNCHRONIZED | TCP://DESKTOP-OS7IBMJ:5023 | DESKTOP-OS7IBMJ\SQLSERVER01 | NULL | NULL |

Adım 3.3: Rollerı Geri Değiştirme

Önceki Manuel Failover işlemi neticesinde MSSQLSERVER02 sunucusu Principal, MSSQLSERVER01 sunucusu ise Mirror rolünü üstlenmişti. Bu adımda, sistemin esnekliğini ve rollerin sorunsuz bir şekilde geri çevrilebildiğini göstermek amacıyla, rollerin tekrar eski haline getirilmesi için ikinci bir Manuel Failover işlemi gerçekleştirılmıştır. Bu işlem, mevcut Principal sunucu olan MSSQLSERVER02 üzerinde, powerlifting_db veritabanı için ALTER DATABASE powerlifting_db SET PARTNER FAILOVER; T-SQL komutunun yürütülmesiyle başlatılmıştır. Komutun başarıyla tamamlanması sonucunda, MSSQLSERVER02 sunucusu tekrar Mirror rolünü üstlenirken, MSSQLSERVER01 sunucusu orijinal Principal rolüne geri dönmüştür. Bu rol değişimi sırasında da herhangi bir veri kaybı yaşanmamış ve veritabanı erişilebilirliği kesintisiz devam etmiştir. Yapılan durum kontrolleri, rollerin beklentiği gibi değiştiğini ve oturumun yeni rollerle SYNCHRONIZED durumda sağlıklı bir şekilde devam ettiğini doğrulamıştır.

```
USE master;
GO
ALTER DATABASE powerlifting_db
    SET PARTNER FAILOVER;
GO
```

Sorgunun çıktısı aşağıda gibidir. Doğru sonuçlar alındığı görülmektedir.

| 1 | DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|---|-----------------|-----------|--------------|----------------------------|-----------------------------|-------------|--------------|
| 1 | powerlifting_db | PRINCIPAL | SYNCHRONIZED | TCP://DESKTOP-OS7IBMJ:5024 | DESKTOP-OS7IBMJ\SQLSERVER02 | NULL | NULL |

| 1 | DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|---|-----------------|--------|--------------|----------------------------|-----------------------------|-------------|--------------|
| 1 | powerlifting_db | MIRROR | SYNCHRONIZED | TCP://DESKTOP-OS7IBMJ:5023 | DESKTOP-OS7IBMJ\SQLSERVER01 | NULL | NULL |

Adım 3.4: Zorlanmış Servis ile Yük Devretme

Veritabanı yansıtma yapılandırmasının kritik bir testi olarak, Birincil sunucu MSSQLSERVER01'in beklenmedik bir şekilde hizmet dışı kaldığı bir felaket senaryosu canlandırılmıştır. Bu durumda, powerlifting_db veritabanına erişimin hızla yeniden sağlanması amacıyla, Ayna sunucu olan MSSQLSERVER02 üzerinde ALTER DATABASE powerlifting_db SET PARTNER

FORCE_SERVICE_ALLOW_DATA_LOSS; T-SQL komutu yürütüülerek Zorlanmış Servis başlatılmıştır. Bu komutun icrası neticesinde, MSSQLSERVER02 sunucusu Birincil rolünü üstlenmiş ve powerlifting_db veritabanı, potansiyel veri kaybı riski kabul edilerek istemci bağlantılarına açılmıştır. Bu işlem, Birincil sunucunun tamamen kullanılamaz hale geldiği durumlarda servis kesintisini en aza indirmek için kritik bir müdahale yöntemi olarak değerlendirilmiştir. Eski Birincil sunucu tekrar çevrimiçi olduğunda, yansıtma oturumunun durumu ve veritabanlarının senkronizasyonu için ek adımlar gerekeceği gözlemlenmiştir.

```
USE master;
GO
ALTER DATABASE powerlifting_db
    SET PARTNER FORCE_SERVICE_ALLOW_DATA LOSS;
GO
```

| DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|-----------------|-----------|--------------|-------------|-----------------|-------------|--------------|
| powerlifting_db | PRINCIPAL | SYNCHRONIZED | NULL | NULL | NULL | NULL |

Adım 3.4: Zorlanmış Servis ile Yük Devretme

Veritabanı yansıtma yapılandırmasının süreklilik ve dayanıklılık testleri kapsamında, Birincil sunucu MSSQLSERVER01'in beklenmedik bir şekilde hizmet dışı kaldığı ve ulaşılamaz olduğu kritik bir felaket senaryosu canlandırılmıştır. Bu durumda, powerlifting_db veritabanına erişimin en kısa sürede ve minimum kesintiyle yeniden sağlanması amacıyla, Ayna sunucu olan MSSQLSERVER02 üzerinde Zorlanmış Servis başlatma prosedürü uygulanmıştır. Bu işlem, ALTER DATABASE powerlifting_db SET PARTNER FORCE_SERVICE_ALLOW_DATA LOSS; T-SQL komutunun MSSQLSERVER02 üzerinde yürütülmesiyle gerçekleştirilmiştir. Komutun başarıyla icra edilmesi neticesinde, MSSQLSERVER02 sunucusu Birincil rolünü üstlenmiş ve powerlifting_db veritabanı, potansiyel veri kaybı riski kabul edilerek istemci bağlantılarına açılmıştır. Bu müdahale, Birincil sunucunun tamamen kullanılamaz hale geldiği durumlarda servis devamlılığını sağlamak için hayatı bir strateji olarak değerlendirilmiştir. Eski Birincil sunucu olan MSSQLSERVER01'in tekrar çevrimiçi olması durumunda, yansıtma oturumunun durumu ve veritabanlarının yeniden senkronizasyonu için ek manuel adımlar gerekeceği tespit edilmiştir.

```

USE master;
GO
ALTER DATABASE powerlifting__db
    SET PARTNER FORCE_SERVICE_ALLOW_DATA_LOSS;
GO

```

Kontrol sorgusu çalıştırıldığında

| DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|------------------|-----------|--------------|-------------|-----------------|-------------|--------------|
| powerlifting__db | PRINCIPAL | SYNCHRONIZED | NULL | NULL | NULL | NULL |

Adım 3.5: Witness Sunucu Eklemek ve Otomatik Yük Devretmeyi Yapılandırmak

Mevcut powerlifting__db Database Mirroring oturumunun High Safety with Automatic Failover yeteneği kazanması amacıyla, sisteme bir Witness sunucu dahil edilmiştir. Tanık sunucu olarak ana bilgisayar üzerinde çalışan SQLEXPRESS SQL Server örneği kullanılmıştır. Bu işlem öncesinde, SQLEXPRESS örneği üzerinde DATABASE_MIRRORING için Endpoint TCP port 5025 üzerinden WITNESS rolüyle oluşturulmuş ve SQL Server servis hesaplarına gerekli bağlantı izinleri verilmiştir. Ardından, mevcut Principal sunucu olan MSSQLSERVER01 üzerinden, powerlifting__db veritabanı için aşağıdaki T-SQL komutu yürütülerek Tanık sunucu yansıtma oturumuna başarıyla atanmıştır:

```

ALTER DATABASE powerlifting__db
    SET WITNESS = 'TCP://DESKTOP-OS7IBMJ\SQLEXPRESS:5025';
GO

```

Bu soru sonrası kontrol işlemi yapıldığında aşağıdaki sonuçları alırız.

| DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|------------------|-----------|--------------|----------------------------|--------------------------------|---------------------------------------|--------------|
| powerlifting__db | PRINCIPAL | SYNCHRONIZED | TCP://DESKTOP-OS7IBMJ:5023 | DESKTOP-OS7IBMJ\MISSQLSERVER01 | TCP://DESKTOP-OS7IBMJ\SQLEXPRESS:5025 | CONNECTED |

| DatabaseName | Role | State | PartnerName | PartnerInstance | WitnessName | WitnessState |
|------------------|--------|--------------|----------------------------|--------------------------------|---------------------------------------|--------------|
| powerlifting__db | MIRROR | SYNCHRONIZED | TCP://DESKTOP-OS7IBMJ:5024 | DESKTOP-OS7IBMJ\MISSQLSERVER02 | TCP://DESKTOP-OS7IBMJ\SQLEXPRESS:5025 | CONNECTED |

Proje 5: Veri Temizleme ve ETL Süreçleri Tasarımı

Kullanılan veri tabanı: Wide World Importers sample database v1.0

Link: <https://github.com/Microsoft/sql-server-samples/releases/tag/wide-world-importers-v1.0>

Bu projede istenen adımlar 4 temel başlık altında değerlendirilmiştir:

- 1- Veri temizleme
- 2- Veri Dönüştürme
- 3- Veri Yükleme
- 4- Veri Kalitesi Raporları

Biz de yaptığımız adımları bu başlıklar üzerinde değerlendirecek olursak.

Bölüm 1: Veri Temizleme

Bu bölümde, Wide World Importers veritabanındaki verilerin kalitesini artırmak amacıyla hatalı, eksik veya tutarsız verileri tespit edip SQL kullanarak temizleme işlemleri gerçekleştireceğiz.

Adım 1.1: Eksik Birincil Kontak E-posta Adresine Sahip Müşterilerin Tespiti

Bu adımda, Sales.Customers tablosu ile Application.People tablosu birleştirilerek, müşterilerin birincil iletişim kişilerine ait e-posta adreslerinin eksik (NULL veya boş) olup olmadığı kontrol edilmiştir.

```
USE WideWorldImporters;
GO

SELECT
    C.CustomerID,
    C.CustomerName,
    P.PersonID AS PrimaryContactPersonID,
    P.FullName AS PrimaryContactName,
    P.EmailAddress AS PrimaryContactEmail
FROM
    Sales.Customers AS C
INNER JOIN
    Application.People AS P ON C.PrimaryContactPersonID = P.PersonID
WHERE
    P.EmailAddress IS NULL OR P.EmailAddress = '';
GO
```

Yukarıdaki sorgunun sonucunun boş olması Wide World Importers veritabanındaki müşterilerin birincil iletişim kişilerine ait e-posta adresi alanında eksik veri (NULL veya boş) bulunmadığını gösteriyor.

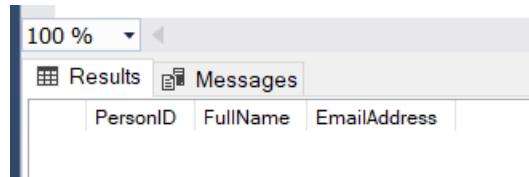
| CustomerID | CustomerName | PrimaryContactPersonID | PrimaryContactName | PrimaryContactEmail |
|------------|--------------|------------------------|--------------------|---------------------|
| | | | | |

Adım 1.2: Geçersiz E-posta Formatı Kontrolü

Bu adımda, Application.People tablosundaki EmailAddress sütununda '@' karakteri içermeyen, yani geçersiz formatta olabilecek e-posta adresleri kontrol edilecektir.

```
SELECT
    PersonID,
    FullName,
    EmailAddress
FROM
    Application.People
WHERE
    EmailAddress IS NOT NULL
    AND EmailAddress <> ''
    AND EmailAddress NOT LIKE '%@%';
GO
```

Yukarıdaki sorgunun sonucunda, '@' karakteri içermeyen ve potansiyel olarak geçersiz formattaki e-posta adresleri listelenir. Eğer sorgu sonuç döndürürse, bu kayıtların incelenmesi ve düzeltilmesi gereklidir. Eğer sorgu sonuç döndürmezse, **Wide World Importers** veritabanındaki tüm e-posta adreslerinin en azından '@' karakteri içerdığı anlaşılır. Aşağıdaki durumda da bu olay mevcuttur.

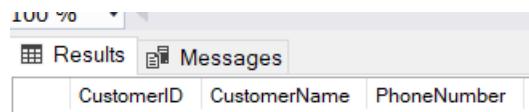


Adım 1.3: Eksik Telefon Numarası Kontrolü

Bu adımda, Sales.Customers tablosundaki PhoneNumber sütununda eksik (NULL veya boş) değer olup olmadığı kontrol edilecektir.

```
SELECT
    CustomerID,
    CustomerName,
    PhoneNumber
FROM
    Sales.Customers
WHERE
    PhoneNumber IS NULL OR PhoneNumber = '';
GO
```

Yukarıdaki sorgunun sonucunda, telefon numarası bilgisi eksik olan müşteriler listelenir. Eğer sorgu sonuç döndürürse, bu kayıtların iletişim bilgileri eksiktir ve güncellenmesi gerekebilir. Eğer sorgu sonuç döndürmezse, **Wide World Importers** veritabanındaki tüm müşterilerin PhoneNumber alanının dolu olduğu anlaşılır. Aşağıda bunu doğrulayan sorgu sonucunu görüntüleyebiliriz.

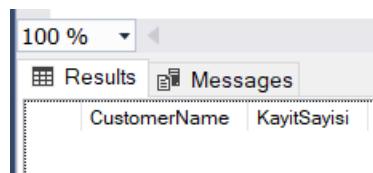


Şu ana kadar yaptığımız kontrollerde (eksik e-posta, geçersiz e-posta formatı, eksik telefon numarası) herhangi bir veri kalitesi sorununa rastlamadık. Bu, **Wide World Importers** örnek veritabanının bu alanlar açısından oldukça temiz olduğunu gösteriyor.

Adım 1.4: Potansiyel Tekrarlı Müşteri Kayıtları Kontrolü

Bu adımda, **Sales.Customers** tablosunda aynı CustomerName değerine sahip, yani potansiyel olarak tekrarlı olabilecek müşteri kayıtları sorgulanacaktır.

```
SELECT
    CustomerName,
    COUNT(*) AS KayitSayisi
FROM
    Sales.Customers
GROUP BY
    CustomerName
HAVING
    COUNT(*) > 1;
GO
```



Sonuç kümesinin boş olması (0 satır), **Wide World Importers** veritabanındaki **Sales.Customers** tablosunda aynı CustomerName değerine sahip birden fazla müşteri kaydı bulunmadığını göstermektedir. Bu kontrol sonucunda potansiyel tekrarlı kayıtlara rastlanmamıştır.

Adım 1.5: Kişi İsimlerindeki Baş/Son Boşlukları Kontrol Etme ve Temizleme

Adım 1.5.1: Başında veya Sonunda Boşluk Olan İsimleri Tespit Etme

Bu adımda, **Application.People** tablosundaki FullName sütununda başında veya sonunda gereksiz boşluk karakteri (whitespace) içeren kayıtlar olup olmadığı kontrol edilecektir. LEN() fonksiyonu sondaki boşlukları göz ardı edebildiğinden, kontrol için daha güvenilir olan DATALENGTH() fonksiyonu kullanılarak orijinal değerin byte uzunluğu ile LTRIM(RTRIM()) sonrası byte uzunluğu karşılaştırılacaktır.

```
SELECT
    PersonID,
    FullName,
    DATALENGTH(FullName) AS OrjinalDataUzunluk,
    DATALENGTH(LTRIM(RTRIM(FullName))) AS TemizDataUzunluk
FROM
    Application.People
WHERE
    DATALENGTH(FullName) <> DATALENGTH(LTRIM(RTRIM(FullName)));
GO
```

Yukarıdaki sorgu çalıştırıldığında sonuçları aşağıda görüldüğü gibidir. Sorgu, PersonID 3217 için OrjinalDataUzunluk (28) ve TemizDataUzunluk (26) değerlerinin farklı olduğunu göstermiştir. Bu, ilgili kaydın FullName alanında başında veya sonunda (bu durumda muhtemelen sonda) temizlenmesi gereken boşluk karakterleri bulunduğu kesin olarak teyit etmektedir.

| | PersonID | FullName | OrjinalDataUzunluk | TemizDataUzunluk |
|---|----------|---------------|--------------------|------------------|
| 1 | 3217 | David Novacek | 28 | 26 |

Adım 1.5.2: Tespit Edilen İsimlerdeki Baş/Son Boşlukları Temizleme

Bir önceki adımda DATALENGTH karşılaştırması ile tespit edilen, başında veya sonunda boşluk bulunan FullName değerleri temizlenecektir. UPDATE işlemi, DATALENGTH koşulunu sağlayan tüm satırları hedefleyecektir.

```
UPDATE Application.People
SET FullName = LTRIM(RTRIM(FullName))
WHERE
    DATALENGTH(FullName) <> DATALENGTH(LTRIM(RTRIM(FullName)));
GO

PRINT CAST(@@ROWCOUNT AS VARCHAR) + ' satırın FullName alanı baş/son boşluklardan temizlendi.';
GO
```

Yukarıdaki UPDATE sorgusu çalıştırıldığında, başında veya sonunda boşluk bulunan kayıtların FullName alanları temizlenir ve etkilenen satır sayısı mesaj olarak gösterilir.

Temizleme işleminin başarılı olduğunu ve artık başında/sonunda boşluk olan kayıt kalmadığını doğrulamak için, Bir önceki adımdaki DATALENGTH kontrol sorgusu tekrar çalıştırılacaktır. Bu sorgunun artık hiçbir kayıt döndürmemesi beklenir.

| | PersonID | FullName | OrjinalDataUzunluk | TemizDataUzunluk |
|--|----------|----------|--------------------|------------------|
| | | | | |

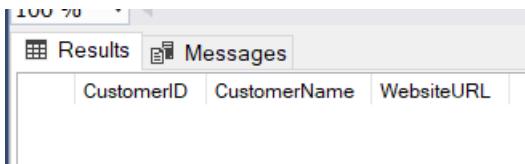
Yukarıdaki sonuçtan da görüldüğü gibi bir önceki adımdaki DATALENGTH kontrol sorgusu çalıştırıldığında bir sonuç dönmediğini görüyoruz. Bu da işlemimizin başarılı olduğu anlamına gelir.

Adım 1.6 - WebsiteURL Protokol Kontrolü ve Düzeltme

Bu adımda, **Sales.Customers** tablosundaki WebsiteURL sütununda 'http://' veya 'https://' protokolü ile başlamayan, yani eksik protokole sahip URL'ler kontrol edilecektir

```
SELECT
    CustomerID,
    CustomerName,
    WebsiteURL
FROM
    Sales.Customers
WHERE
    WebsiteURL IS NOT NULL
    AND WebsiteURL <> ''
    AND WebsiteURL NOT LIKE 'http://%'
    AND WebsiteURL NOT LIKE 'https://%';
GO
```

Yukarıdaki sorgu çalıştırıldığında sonuçları aşağıda görüldüğü gibidir. Sorgunun hiçbir kayıt döndürmemesi, Sales.Customers tablosundaki tüm geçerli WebsiteURL değerlerinin zaten bir protokol (http:// veya https://) içerdigini göstermektedir. Bu kontrol sonucunda düzeltilmesi gereken bir URL bulunmamıştır.



Bölüm 2: Veri Dönüştürme

Veri temizleme adımlarının ardından, bu bölümde verilerin analiz ve raporlama için daha kullanışlı ve standart bir formata getirilmesi amacıyla veri dönüştürme işlemleri yapılacaktır. Dönüşüm, mevcut verilerin formatını, yapısını veya değerlerini değiştirmeyi içerir.

Adım 2.1: Müşteri İsimlerini Büyük Harfe Dönüştürme

Bu ilk dönüşüm adımında, **Sales.Customers** tablosundaki CustomerName sütunundaki değerlerin tutarlılık amacıyla tümüyle büyük harfe dönüştürülmüş hali gösterilecektir.

```
USE WideWorldImporters;
GO

SELECT TOP 10
    CustomerID,
    CustomerName AS OrjinalIsim,
    UPPER(CustomerName) AS BuyukHarfIsim
FROM
    Sales.Customers;
GO
```

Yukarıdaki sorgu çalıştırıldı ve sonuçları ekran görüntüsünde görüldüğü gibidir. Soru, ilk 10 müşteri isminin orijinal halini (OrjinalIsim) ve UPPER() fonksiyonu kullanılarak tamamen büyük harfe çevrilmiş halini (BuyukHarfIsim) yan yana listeler. Bu çıktı, dönüşümün bekleniği gibi çalıştığını görsel olarak doğrulamaktadır ve tüm isimlerin büyük harfe çevrilmesinin veri setine nasıl yansıyacağını göstermektedir.

| | CustomerID | OrjinalIsim | BuyukHarfIsim |
|----|------------|------------------------------------|------------------------------------|
| 1 | 1 | Tailspin Toys (Head Office) | TAILSPIN TOYS (HEAD OFFICE) |
| 2 | 2 | Tailspin Toys (Sylvanite, MT) | TAILSPIN TOYS (SYLVANITE, MT) |
| 3 | 3 | Tailspin Toys (Peeples Valley, AZ) | TAILSPIN TOYS (PEEPLES VALLEY, AZ) |
| 4 | 4 | Tailspin Toys (Medicine Lodge, KS) | TAILSPIN TOYS (MEDICINE LODGE, KS) |
| 5 | 5 | Tailspin Toys (Gasport, NY) | TAILSPIN TOYS (GASPORT, NY) |
| 6 | 6 | Tailspin Toys (Jessie, ND) | TAILSPIN TOYS (JESSIE, ND) |
| 7 | 7 | Tailspin Toys (Frankewing, TN) | TAILSPIN TOYS (FRANKEWING, TN) |
| 8 | 8 | Tailspin Toys (Bow Mar, CO) | TAILSPIN TOYS (BOW MAR, CO) |
| 9 | 9 | Tailspin Toys (Netcong, NJ) | TAILSPIN TOYS (NETCONG, NJ) |
| 10 | 10 | Tailspin Toys (Wimbledon, ND) | TAILSPIN TOYS (WIMBLEDON, ND) |

Adım 2.1.1: Müşteri İsimlerini Kalıcı Olarak Büyük Harfe Güncelleme

Bu adımda, bir önceki adımda gösterilen dönüşüm kalıcı hale getirilecektir. Sales.Customers tablosundaki tüm CustomerName değerleri UPPER() fonksiyonu kullanılarak büyük harfli halleriyle güncellenecektir.

```
-- Sales.Customers tablosundaki CustomerName sütununu büyük harfe güncellenmesi
UPDATE Sales.Customers
SET CustomerName = UPPER(CustomerName);
GO

-- Güncellemenin etkisini görmek için ilk 10 kaydı tekrar seçilmesi
SELECT TOP 10
    CustomerID,
    CustomerName
FROM
    Sales.Customers;
GO
```

Doğrulama sorgusunun sonucunda, CustomerName sütunundaki değerlerin artık tamamen büyük harf olduğu görülmektedir. Bu, müşteri isimlerini büyük harfe dönüştürme işleminin kalıcı olarak başarıyla uygulandığını teyit eder.

Adım 2.2: Tam Teslimat Adresi Oluşturma

Bu adımda, müşterilerin teslimat adres bilgileri (Sales.Customers, Application.Cities, Application.StateProvinces tablolarından alınarak) tek bir TamTeslimatAdresi sütununda birleştirilecektir.

```
SELECT TOP 10
    C.CustomerID,
    C.CustomerName,
    CONCAT(
        C.DeliveryAddressLine1,
        CASE
            WHEN C.DeliveryAddressLine2 IS NOT NULL AND C.DeliveryAddressLine2 <> ''
            THEN N', ' + C.DeliveryAddressLine2
            ELSE N''
        END,
        N', ', CT.CityName,
        N', ', SP.StateProvinceName,
        N' ', C.DeliveryPostalCode
    ) AS TamTeslimatAdresi
FROM
    Sales.Customers AS C
INNER JOIN
    Application.Cities AS CT ON C.DeliveryCityID = CT.CityID
INNER JOIN
    Application.StateProvinces AS SP ON CT.StateProvinceID = SP.StateProvinceID;
GO
```

Yukarıdaki sorgu çalıştırıldı ve sonuçları aşağıda görüldüğü gibidir. Sorgu, ilgili tablolardan alınan adres bileşenlerini (DeliveryAddressLine1, DeliveryAddressLine2, CityName, StateProvinceName, DeliveryPostalCode) başarılı bir şekilde birleştirerek ilk 10 müşteri için formatlanmış TamTeslimatAdresi'ni oluşturmuştur.

| | CustomerID | CustomerName | TamTeslimatAdresi |
|----|------------|------------------------------------|--|
| 1 | 11 | TAILSPIN TOYS (DEVAULT, PA) | Unit 250, 1432 Pullela Street, Devault, Pennsylvania ... |
| 2 | 10 | TAILSPIN TOYS (WIMBLEDON, ND) | Unit 67, 372 Joo Lane, Wimbledon, North Dakota 90... |
| 3 | 9 | TAILSPIN TOYS (NETCONG, NJ) | Shop 33, 25 Kasesalu Street, Netcong, New Jersey 9... |
| 4 | 8 | TAILSPIN TOYS (BOW MAR, CO) | Shop 282, 752 Shaker Doush Boulevard, Bow Mar, C... |
| 5 | 7 | TAILSPIN TOYS (FRANKIEWING, TN) | Shop 27, 904 Kellnerova Street, Frankiewing, Tennessee... |
| 6 | 6 | TAILSPIN TOYS (JESSIE, ND) | Shop 196, 483 Raut Lane, Jessie, North Dakota 902... |
| 7 | 5 | TAILSPIN TOYS (GASPORT, NY) | Unit 176, 1674 Skujins Boulevard, Gasport, New York... |
| 8 | 4 | TAILSPIN TOYS (MEDICINE LODGE, KS) | Suite 164, 967 Riutta Boulevard, Medicine Lodge, Kansas... |
| 9 | 3 | TAILSPIN TOYS (PEEPLES VALLEY, AZ) | Unit 217, 1970 Khandke Road, Peeples Valley, Arizona... |
| 10 | 2 | TAILSPIN TOYS (SYLVANITE, MT) | Shop 245, 705 Dita Lane, Sylvanite, Montana 90216 |

Adım 2.3: Hesap Açılış Yılını Çıkarma

Bu adımda, Sales.Customers tablosundaki AccountOpenedDate sütunundan yıl bilgisi YEAR() fonksiyonu kullanılarak ayrı bir sütun olarak alınacaktır.

```
|SELECT TOP 10
    CustomerID,
    CustomerName,
    AccountOpenedDate,
    YEAR(AccountOpenedDate) AS HesapAcilisYili
FROM
    Sales.Customers
ORDER BY
    AccountOpenedDate;
|go|
```

Yukarıdaki soru çalıştırıldı ve sonuçları aşağıda görüldüğü gibidir. Soru, ilk 10 müşterinin hesap açılış tarihini (AccountOpenedDate) ve bu tarihten YEAR() fonksiyonu ile başarılı bir şekilde çıkarılan yıl bilgisini (HesapAcilisYili) gösterir.

| | CustomerID | CustomerName | AccountOpenedDate | HesapAcilisYili |
|----|------------|------------------------------------|-------------------|-----------------|
| 1 | 1 | TAILSPIN TOYS (HEAD OFFICE) | 2013-01-01 | 2013 |
| 2 | 2 | TAILSPIN TOYS (SYLVANITE, MT) | 2013-01-01 | 2013 |
| 3 | 3 | TAILSPIN TOYS (PEEPLES VALLEY, AZ) | 2013-01-01 | 2013 |
| 4 | 4 | TAILSPIN TOYS (MEDICINE LODGE, KS) | 2013-01-01 | 2013 |
| 5 | 5 | TAILSPIN TOYS (GASPORT, NY) | 2013-01-01 | 2013 |
| 6 | 6 | TAILSPIN TOYS (JESSIE, ND) | 2013-01-01 | 2013 |
| 7 | 7 | TAILSPIN TOYS (FRANKIEWING, TN) | 2013-01-01 | 2013 |
| 8 | 8 | TAILSPIN TOYS (BOW MAR, CO) | 2013-01-01 | 2013 |
| 9 | 9 | TAILSPIN TOYS (NETCONG, NJ) | 2013-01-01 | 2013 |
| 10 | 10 | TAILSPIN TOYS (WIMBLEDON, ND) | 2013-01-01 | 2013 |

Bölüm 3: Veri Yükleme

Adım 3.1: Yıllık Müşteri Özeti Tablosunu Oluşturma ve Yükleme

Bu adım 3 alt adımdan oluşmaktadır: Hedef tablonun oluşturulması, hesaplanan verinin yüklenmesi ve yüklenen verinin doğrulanması.

Adım 3.1.1: Hedef Tabloyu Oluşturma

Veri yükleme işleminin ilk adımı olarak, yıllık müşteri sayısı özetini saklayacak olan Sales.CustomerSummaryByYear adında yeni bir tablo oluşturulacaktır. Tablonun zaten var olup olmadığını kontrol eden bir mekanizma eklenecektir.

```

IF OBJECT_ID('Sales.CustomerSummaryByYear', 'U') IS NULL
BEGIN
    CREATE TABLE Sales.CustomerSummaryByYear (
        AcilisYili INT PRIMARY KEY,
        MusteriSayisi INT
    );
    PRINT 'Sales.CustomerSummaryByYear tablosu oluşturuldu.';
END
ELSE
BEGIN
    PRINT 'Sales.CustomerSummaryByYear tablosu zaten mevcut. Tekrar oluşturulmayacak.' ;
END
GO

```

Yukarıdaki sorgu çalıştırıldığında tablo ya oluşturulur ya da zaten var olduğu mesajı alınır.

Adım 3.1.2: Hesaplanan Özeti Hedef Tabloya Yükleme

Şimdi, Sales.Customers tablosundan hesaplanan yıllık müşteri sayısı özeti, bir önceki adımda oluşturulan veya var olan Sales.CustomerSummaryByYear tablosuna INSERT INTO ... SELECT ifadesi kullanılarak yüklenecektir. Tekrarlı yüklemelerde sorun olmaması için yükleme öncesinde tablo içeriği TRUNCATE TABLE komutu ile temizlenmektedir.

```

-- Yükleme öncesi tabloyu temizleme işlemi
IF OBJECT_ID('Sales.CustomerSummaryByYear', 'U') IS NOT NULL
BEGIN
    TRUNCATE TABLE Sales.CustomerSummaryByYear;
END
GO

-- Hesaplanan özet veriyi Sales.CustomerSummaryByYear tablosuna yükleme işlemi
INSERT INTO Sales.CustomerSummaryByYear (AcilisYili, MusteriSayisi)
SELECT
    YEAR(AccountOpenedDate) AS HesapAcilisYili,
    COUNT(*) AS MusteriSayisi
FROM
    Sales.Customers
GROUP BY
    YEAR(AccountOpenedDate);
GO

```

Yukarıdaki sorgu çalıştırıldığında, hesaplanan özet veriler hedef tabloya yüklenir ve kaç satırın yüklediği mesaj olarak gösterilir.

Adım 3.1.3: Yüklenen Veriyi Doğrulama

Verilerin Sales.CustomerSummaryByYear tablosuna doğru şekilde yüklediğini kontrol etmek için yeni oluşturulan ve yüklenen bu tablonun içeriği sorgulanacaktır.

```

SELECT
    AcilisYili,
    MusteriSayisi
FROM
    Sales.CustomerSummaryByYear
ORDER BY
    AcilisYili;
GO

```

Yukarıdaki sorgu çalıştırıldığında sonuçları aşağıda görüldüğü gibidir. Bu sonuçlar, yıllık müşteri sayısı özetinin başarıyla oluşturulup kalıcı bir tabloya yüklediğini teyit eder.

| | AcilisYili | MusteriSayisi |
|---|------------|---------------|
| 1 | 2013 | 626 |
| 2 | 2014 | 14 |
| 3 | 2015 | 18 |
| 4 | 2016 | 5 |

Adım 3.2: Tam Teslimat Adreslerini Yeni Tabloya Yükleme

Bu adım da 3 alt adımdan oluşmaktadır: Hedef tablonun oluşturulması, birleştirilmiş adres verisinin yüklenmesi ve yüklenen verinin doğrulanması.

Adım 3.2.1: Hedef Tabloyu Oluşturma

Bu adımda, Bölüm 2'de oluşturulan tam teslimat adreslerini kalıcı olarak saklamak üzere Sales.CustomerFullDeliveryAddress adında yeni bir tablo oluşturulacaktır.

```

IF OBJECT_ID('Sales.CustomerFullDeliveryAddress', 'U') IS NULL
BEGIN
    CREATE TABLE Sales.CustomerFullDeliveryAddress (
        CustomerID INT PRIMARY KEY,
        TamTeslimatAdresi NVARCHAR(500) NULL
    );
    PRINT 'Sales.CustomerFullDeliveryAddress tablosu oluşturuldu.';
END
ELSE
BEGIN
    PRINT 'Sales.CustomerFullDeliveryAddress tablosu zaten mevcut. Tekrar oluşturulmayacak.' ;
END
GO

```

Yukarıdaki sorgu çalıştırıldığında tablo ya oluşturulur ya da zaten var olduğu mesajı alınır.

Adım 3.2.2: Birleştirilmiş Adresleri Hedef Tabloya Yükleme

Oluşturulan (veya zaten var olan) Sales.CustomerFullDeliveryAddress tablosuna, ilgili tablolardan birleştirilerek elde edilen tam teslimat adresleri yüklenecektir. Yükleme öncesinde tablo içeriği TRUNCATE TABLE komutu ile temizlenmektedir.

```

INSERT INTO Sales.CustomerFullDeliveryAddress (CustomerID, TamTeslimatAdresi)
SELECT
    C.CustomerID,
    CONCAT(
        C.DeliveryAddressLine1,
        CASE WHEN C.DeliveryAddressLine2 IS NOT NULL AND C.DeliveryAddressLine2 <> '' THEN N', ' + C.DeliveryAddressLine2 ELSE N'' END,
        N', ', CT.CityName,
        N', ', SP.StateProvinceName,
        N', ', C.DeliveryPostalCode
    )
FROM
    Sales.Customers AS C
INNER JOIN
    Application.Cities AS CT ON C.DeliveryCityID = CT.CityID
INNER JOIN
    Application.StateProvinces AS SP ON CT.StateProvinceID = SP.StateProvinceID;
GO

PRINT CAST(@@ROWCOUNT AS VARCHAR) + ' satır Sales.CustomerFullDeliveryAddress tablosuna yüklandı.';
GO

```

Yukarıdaki sorgu çalıştırıldığında, birleştirilmiş adres verileri hedef tabloya yüklenir ve kaç satırın yüklediği mesaj olarak gösterilir.

Adım 3.2.3: Yüklenen Veriyi Doğrulama

Tam teslimat adreslerinin Sales.CustomerFullDeliveryAddress tablosuna doğru bir şekilde yüklediğini kontrol etmek için tablodan ilk 10 kayıt sorgulanacaktır.

```

SELECT TOP 10
    CustomerID,
    TamTeslimatAdresi
FROM
    Sales.CustomerFullDeliveryAddress
ORDER BY
    CustomerID;
GO

```

Yukarıdaki sorgu çalıştırıldığında sonuçları aşağıda görüldüğü gibidir. Bu, Bölüm 2'de dönüştürülerek elde edilen birleştirilmiş adres verisinin yeni tabloya başarıyla yüklediğini gösterir.

| | CustomerID | TamTeslimatAdresi |
|----|------------|---|
| 1 | 1 | Shop 38, 1877 Mittal Road, Lisco, Nebraska 90410 |
| 2 | 2 | Shop 245, 705 Dita Lane, Sylvanite, Montana 90216 |
| 3 | 3 | Unit 217, 1970 Khandke Road, Peeples Valley, Arizona 90205 |
| 4 | 4 | Suite 164, 967 Riutta Boulevard, Medicine Lodge, Kansas 90152 |
| 5 | 5 | Unit 176, 1674 Skujins Boulevard, Gasport, New York 90261 |
| 6 | 6 | Shop 196, 483 Raut Lane, Jessie, North Dakota 90298 |
| 7 | 7 | Shop 27, 904 Kellnerova Street, Frankewing, Tennessee 90761 |
| 8 | 8 | Shop 282, 752 Shaker Doust Boulevard, Bow Mar, Colorado 90484 |
| 9 | 9 | Shop 33, 25 Kasesalu Street, Netcong, New Jersey 90129 |
| 10 | 10 | Unit 67, 372 Joo Lane, Wimbledon, North Dakota 90061 |

Bölüm 4: Veri Kalitesi Raporları

Bu bölümde, projenin ilk aşamasında gerçekleştirilen veri kalitesi kontrollerinin sonuçları özetlenmektedir. Veri kalitesi raporlaması, veritabanındaki olası sorunları belgelemek, veri setinin güvenilirliği hakkında bilgi vermek ve gelecekteki veri yönetimi stratejileri için temel oluşturmak açısından önemlidir.

Adım 4.1: Veri Temizleme Kontrolleri Bulgularının Özeti

Geçerleştirilen proje kapsamında Wide World Importers veritabanında aşağıdaki temel veri kalitesi kontrolleri gerçekleştirilmiş ve şu sonuçlar elde edilmiştir:

1. Eksik Birincil Kontak E-posta Adresi Kontrolü (Adım 1.1):

- Amaç: Sales.Customers tablosundaki müşterilerin Application.People tablosunda kayıtlı birincil iletişim kişilerine ait e-posta adreslerinin eksik (NULL veya boş) olup olmadığını kontrol etmek.
- Bulgu: Yapılan sorgulama sonucunda, birincil iletişim kişisine ait e-posta adresi eksik olan herhangi bir müşteri kaydına rastlanmamıştır.

2. Geçersiz E-posta Formatı Kontrolü (Adım 1.2):

- Amaç: Application.People tablosundaki e-posta adreslerinin temel formatını (içinde '@' karakteri bulunup bulunmadığını) kontrol etmek.
- Bulgu: Yapılan sorgulama sonucunda, '@' karakteri içermeyen ve potansiyel olarak geçersiz formattaki bir e-posta adresine rastlanmamıştır.

3. Eksik Telefon Numarası Kontrolü (Adım 1.3):

- Amaç: Sales.Customers tablosundaki PhoneNumber sütununda eksik (NULL veya boş) değer olup olmadığını kontrol etmek.
- Bulgu: Yapılan sorgulama sonucunda, telefon numarası bilgisi eksik olan herhangi bir müşteri kaydına rastlanmamıştır.

4. Potansiyel Tekrarlı Müşteri Kayıtları Kontrolü (Adım 1.4):

- Amaç: Sales.Customers tablosunda aynı CustomerName değerine sahip birden fazla kayıt olup olmadığını kontrol etmek.
- Bulgu: Yapılan sorgulama sonucunda, aynı müşteri ismine sahip birden fazla kayda rastlanmamıştır.

Genel Değerlendirme:

Geçerleştirilen bu temel veri kalitesi kontrolleri sonucunda, Wide World Importers örnek veritabanının müşteri iletişim bilgileri (e-posta, telefon) ve müşteri isimlerinin tekilliği açısından herhangi bir sorun içermediği görülmüştür. İncelenen alanlarda eksik veya temel formatı bozuk verilere ve tekrarlı kayıtlara rastlanmamıştır. Bu durum, kullanılan örnek veri setinin bu spesifik kontroller açısından temiz ve tutarlı olduğunu göstermektedir. Daha kapsamlı veri kalitesi analizleri için farklı kontroller (adres tutarlılığı, geçersiz karakterler, aykırı değer analizi vb.) de yapılabilir, ancak projenin bu aşamasında yapılan temel kontroller olumlu sonuçlanmıştır.

Adım 4.2: SQL ile Süreç Etkisi Raporlama

SQL sorguları kullanılarak süreç hakkında raporlama yapmak için, adres birleştirme ve yükleme adımlının (Adım 3.2) bir etkisi incelenecektir. Aşağıdaki ilk sorgu, **Sales.Customers** tablosundaki orijinal teslimat adresi satırlarının (DeliveryAddressLine1 ve DeliveryAddressLine2) toplam ortalama karakter uzunluğunu hesaplar. İkinci sorgu ise Adım 3.2'de oluşturulan ve yüklenen **Sales.CustomerFullDeliveryAddress** tablosundaki birleştirilmiş tam adreslerin (TamTeslimatAdresi) ortalama karakter uzunluğunu hesaplar.

```
-- Orijinal adres satırlarının (Line1 + Line2) ortalama toplam uzunluğunu veren sorgu
SELECT
    AVG(
        LEN(ISNULL(DeliveryAddressLine1, N''))
        + LEN(ISNULL(DeliveryAddressLine2, N''))
    ) AS OrtalamaOrijinalAdresUzunlugu
FROM
    Sales.Customers;
GO

-- Birleştirilip yüklenen tam adreslerin ortalama uzunluğunu veren sorgu
SELECT
    AVG(LEN(TamTeslimatAdresi)) AS OrtalamaBirlesikAdresUzunlugu
FROM
    Sales.CustomerFullDeliveryAddress;
GO
```

Yukarıdaki sorgular çalıştırıldığında sonuçları aşağıda görüldüğü gibidir. İlk sonuç, orijinal adres satırlarının (DeliveryAddressLine1 + DeliveryAddressLine2) ortalama toplam uzunluğunun **25 karakter** olduğunu göstermektedir. İkinci sonuç ise, birleştirilip **Sales.CustomerFullDeliveryAddress** tablosuna yüklenen tam adreslerin ortalama uzunluğunun **55 karakter** olduğunu göstermektedir. Bu metrikler, adres birleştirme ve yükleme işleminin sonucunda adres bilgisinin ortalama olarak daha uzun (daha fazla detay içerir şekilde) hale geldiğini sayısal olarak koymaktadır.

| | Results | Messages |
|-------|-------------------------------|----------|
| <hr/> | | |
| 1 | OrtalamaOrijinalAdresUzunlugu | |
| <hr/> | | |
| 1 | 55 | |

Proje 6: Veritabanı Yükseltme ve Sürüm Yönetimi

Kullanılan veri tabanı: Powerlifting Database

Link: <https://www.kaggle.com/datasets/open-powerlifting/powerlifting-database?resource=download>

Bu projede istenen adımlar 3 temel başlık altında değerlendirilmiş:

- 1- Veritabanı Yükseltme Planı
- 2- Sürüm Yönetimi
- 3- Test ve Geri Dönüş Planı,

Biz de yaptığımız adımları bu başlıklar üzerinde değerlendirecek olursak:

Bölüm 1: Veritabanı Yükseltme Planı

Bu bölüm, mevcut powerlifting_db veritabanının daha yeni bir SQL Server sürümüne veya aynı sürümün farklı bir yapısına yükseltilmesi sürecini planlamaya odaklanmaktadır. Bu bölümde, yükseltme stratejisinin temel adımları ve dikkat edilmesi gereken noktalar ele alınacaktır. Bu bölümdeki adımlar genellikle doğrudan SQL çalıştırımaktan çok, planlama ve belgeleme faaliyetlerini içerir ancak mevcut durumu anlamak için bazı sorgular kullanılacaktır.

Adım 1.1: Mevcut Veritabanı Durumunun Analizi ve Belgelenmesi

Yükseltme planının ilk ve en kritik adımı, mevcut SQL Server ve powerlifting_db ortamının detaylı bir şekilde analiz edilmesi ve belgelenmesidir. Bu, mevcut sürüm, yapılandırma, boyut, içerik ve bağımlılıklar hakkında net bir fikir edinmemizi sağlar. Aşağıdaki sorgular, bu temel bilgileri toplamak için kullanılabilir.

```
-- Mevcut SQL Server örneğinin sürüm bilgilerini al
SELECT @@VERSION AS SQLServerVersionInfo;
GO

-- 'powerlifting_db' veritabanının uyumluluk seviyesini kontrol et
USE master;
GO
SELECT compatibility_level
FROM sys.databases
WHERE name = N'powerlifting_db';
GO

-- 'powerlifting_db' veritabanının boyutunu göster
USE powerlifting_db;
GO
EXEC sp_spaceused;
GO
```

```
USE powerlifting_db;
GO
-- Kullanıcı tablolarını listele
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_SCHEMA <> 'sys';

-- View'ları listele
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'VIEW' AND TABLE_SCHEMA <> 'sys';

-- Stored Procedure'ları listele
SELECT SPECIFIC_SCHEMA, SPECIFIC_NAME
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_TYPE = 'PROCEDURE' AND SPECIFIC_SCHEMA <> 'sys';

-- Fonksiyonları listele
SELECT SPECIFIC_SCHEMA, SPECIFIC_NAME, ROUTINE_TYPE
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_TYPE IN ('FUNCTION', 'TABLE_FUNCTION') AND SPECIFIC_SCHEMA <> 'sys';
GO
```

Yukarıdaki sorgular çalıştırıldığında sonuçları aşağıda görüldüğü gibidir:

SQL Server Sürümü: Mevcut sunucu Microsoft SQL Server 2022 (RTM-GDR) sürümünde çalışmaktadır.

Veritabanı Uyumluluk Seviyesi: powerlifting_db veritabanının uyumluluk seviyesi 160'tır (SQL Server 2022).

Veritabanı Boyutu: Veritabanının toplam boyutu yaklaşık 400 MB'tır ve bunun ~355 MB'ı veri ve indeksler için ayrılmıştır.

Veritabanı Nesneleri: Veritabanı yapısı oldukça basittir; tek bir kullanıcı tablosu (dbo.openpowerlifting) içermekte ve şu anda özel view, stored procedure veya fonksiyon barındırmamaktadır.

Bu bilgiler, yükseltme stratejisi oluşturmak için kritik bir temel sağlar. Mevcut sistemin güncel bir SQL Server sürümünde ve uyumluluk seviyesinde olması, veritabanı boyutunun yönetilebilir olması ve özel kod nesnelerinin bulunmaması, olası bir yükseltme veya güncelleme işlemini teknik açıdan basitleştirebilir. Ancak, yine de sunucu yapılandırmaları, güvenlik ayarları, bakım planları, yedekleme stratejileri ve özellikle dbo.openpowerlifting tablosuna bağlı uygulamaların detaylı analizi ve belgelenmesi gereklidir.

| SQLServerVersionInfo | | | |
|----------------------|---------------------|-------------------|---------------|
| | compatibility_level | database_name | database_size |
| 1 | 160 | powerlifting_db | 400.00 MB |
| | | unallocated space | 37.34 MB |

| | reserved | data | index_size | unused |
|---|-----------|-----------|------------|----------|
| 1 | 363176 KB | 330368 KB | 1312 KB | 31496 KB |

| TABLE_SCHEMA TABLE_NAME | | | |
|-------------------------------|----------------------|--------------|--|
| 1 | dbo openpowerlifting | | |
| SPECIFIC_SCHEMA SPECIFIC_NAME | | | |
| | | ROUTINE_TYPE | |

Adım 1.2: Yükseltme Hedefleri ve Kapsamının Belirlenmesi

powerlifting_db veritabanı ve barındığı SQL Server örneği için planlanan yükseltmenin temel hedefleri, kapsamı ve kısıtları aşağıdaki gibi belirlenmiştir:

Gerekçe: En güncel SQL Server güvenlik yamalarından, potansiyel performans iyileştirmelerinden faydalanan ve platformu güncel tutmak.

Hedef Sürüm: Mevcut SQL Server 2022 örneğini, yayınlanmış **en son Kararlı Kümulatif Güncelleme (CU)** paketine yükseltmek.

Kapsam:

Yalnızca mevcut SQL Server veritabanı motoru örneği güncellenecektir.

İşletim sistemi veya donanımda değişiklik yapılmayacaktır.

powerlifting_db veritabanı şemasında veya uyumluluk seviyesinde değişiklik yapılmayacaktır.

Veritabanına bağlanan uygulamaların uyumluluğu ayrı bir test sürecinde doğrulanacaktır.

Temel Kısıtlar/Beklentiler:

Yükseltme işlemi için planlanan maksimum kesinti süresi 2 saatir.

Yükseltme sonrası veritabanı performansının en az mevcut seviyede kalması veya iyileşmesi beklenmektedir.

Adım 1.3: Yükseltme Yönteminin Seçilmesi ve Risk Analizi

Önceki adımlarda yapılan analizler (mevcut SQL Server 2022 sürümü, veritabanının basit yapısı) ve belirlenen hedefler (en son Kümülatif Güncelleme (CU) paketini uygulama) doğrultusunda, bu plan için Yerinde Yükseltme yöntemi tercih edilmiştir. Bu yöntemde, mevcut SQL Server örneği üzerine doğrudan güncelleme yapılır.

Bu yöntemle ilişkili temel riskler şunlardır:

- Geri Dönüş Zorluğu: Yükseltme sırasında beklenmedik bir sorun olursa, önceki duruma geri dönmek (rollback) daha karmaşıktır. Tam yedeklemeler esastır.
- Beklenmedik Sorunlar: Kurulum hataları veya yapılandırma sorunları yaşanabilir.
- Uygulama Uyumluluğu: Veritabanına bağlanan uygulamalarda uyumluluk sorunları çıkabilir.
- Performans Değişiklikleri: Yükseltme sonrası performansta düşüşler olabilir.
- Yetersiz Test: Yükseltme öncesi yetersiz test, canlı ortamda sorun riskini artırır.

Bu tanımlanan riskler için alınacak önlemler ve test stratejileri Bölüm 3 (Test ve Geri Dönüş Planı) altında detaylandırılacaktır.

Adım 1.4: Önemli Yapılandırma Ayarlarının Belgelenmesi

Yükseltme planlamasının bir parçası olarak, mevcut SQL Server örneğinin önemli yapılandırma ayarlarının belgelenmesi, yükseltme sonrası tutarlılığı sağlamak ve olası performans değişikliklerini analiz etmek açısından önemlidir. `sp_configure` sistem saklı yordamı bu ayarları görüntülemek için kullanılır. Tüm ayarları görebilmek için öncelikle 'show advanced options' seçeneğinin etkinleştirilmesi gerekebilir.

```
-- Gelişmiş seçenekleri göstermeyi etkinleştir
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
GO

-- Mevcut tüm yapılandırma ayarlarını ve değerlerini liste
EXEC sp_configure;
GO
```

Yukarıdaki sorgular çalıştırıldığında sonuçları aşağıda görüldüğü gibidir. İlk sorgu, gelişmiş yapılandırma seçeneklerinin görünürüğünü sağlar. İkinci sorgu (`sp_configure`) ise, SQL Server örneğinin çok sayıda yapılandırma ayarını ve bu ayarların mevcut çalışan değerlerini (`run_value`) listeler. Aşağıdaki görselde görülen listeden, örneğin cost threshold for parallelism ayarının çalışan değerinin 5 olduğu gibi bilgiler elde edilir. Yükseltme planı için, bu çıktıdan özellikle max server memory, max degree of parallelism gibi performansla ilgili kritik ayarların mevcut çalışan değerleri

dikkatlice not edilmelidir. Bu belgeleme, yükseltme sonrasında bu önemli ayarların değişip değişmediğini kontrol etmek ve gerekirse eski haline getirmek veya bilinçli olarak yeni değerlerde bırakmak için temel oluşturur.

| | Results | Messages | | | |
|----|-------------------------------------|-------------|------------|--------------|------------|
| | name | minimum | maximum | config_value | run_value |
| 1 | access check cache bucket count | 0 | 65536 | 0 | 0 |
| 2 | access check cache quota | 0 | 2147483647 | 0 | 0 |
| 3 | Ad Hoc Distributed Queries | 0 | 1 | 0 | 0 |
| 4 | ADR cleaner retry timeout (min) | 0 | 32767 | 15 | 15 |
| 5 | ADR Cleaner Thread Count | 1 | 32767 | 1 | 1 |
| 6 | ADR Preallocation Factor | 0 | 32767 | 4 | 4 |
| 7 | affinity I/O mask | -2147483648 | 2147483647 | 0 | 0 |
| 8 | affinity mask | -2147483648 | 2147483647 | 0 | 0 |
| 9 | affinity64 I/O mask | -2147483648 | 2147483647 | 0 | 0 |
| 10 | affinity64 mask | -2147483648 | 2147483647 | 0 | 0 |
| 11 | Agent XPs | 0 | 1 | 1 | 1 |
| 12 | allow filesystem enumeration | 0 | 1 | 1 | 1 |
| 13 | allow polybase export | 0 | 1 | 0 | 0 |
| 14 | allow updates | 0 | 1 | 0 | 0 |
| 15 | automatic soft-NUMA disabled | 0 | 1 | 0 | 0 |
| 16 | backup checksum default | 0 | 1 | 0 | 0 |
| 17 | backup compression algorithm | 0 | 2 | 0 | 0 |
| 18 | blocked process threshold (s) | 0 | 86400 | 0 | 0 |
| 19 | c2 audit mode | 0 | 1 | 0 | 0 |
| 20 | clr enabled | 0 | 1 | 0 | 0 |
| 21 | clr strict security | 0 | 1 | 1 | 1 |
| 22 | column encryption enclave type | 0 | 2 | 0 | 0 |
| 23 | contained database authenticati... | 0 | 1 | 0 | 0 |
| 24 | cost threshold for parallelism | 0 | 32767 | 5 | 5 |
| 25 | cross db ownership chaining | 0 | 1 | 0 | 0 |
| 26 | cursor threshold | -1 | 2147483647 | -1 | -1 |
| 27 | Data processed daily limit in TB | 0 | 2147483647 | 2147483647 | 2147483647 |
| 28 | Data processed monthly limit in ... | 0 | 2147483647 | 2147483647 | 2147483647 |
| 29 | Data processed weekly limit in TB | 0 | 2147483647 | 2147483647 | 2147483647 |
| 30 | Database Mail XPs | 0 | 1 | 0 | 0 |
| 31 | default full-text language | 0 | 2147483647 | 1033 | 1033 |
| 32 | default language | 0 | 9999 | 0 | 0 |
| 33 | default trace enabled | 0 | 1 | 1 | 1 |
| 34 | disallow results from triggers | 0 | 1 | 0 | 0 |
| 35 | external scripts enabled | 0 | 1 | 0 | 0 |
| 36 | filestream access level | 0 | 2 | 0 | 0 |

Bölüm 2: Sürüm Yönetimi

Bu bölüm, powerlifting_db veritabanının yapısında zaman içinde meydana gelen değişiklikleri yönetmeye ve izlemeye odaklanmaktadır. Etkili sürüm yönetimi, veritabanı şemasının farklı versiyonları arasındaki farkları anlamayı, planlı değişiklikleri kontrollü bir şekilde uygulamayı ve beklenmedik veya yetkisiz değişiklikleri tespit etmeyi içerir. Bu bölümün ilk adımlarında, veritabanının belirli sürümleri arasındaki farkları analiz etme ve planlama yöntemlerine değinilecek, ardından ise DDL tetikleyicileri kullanarak anlık şema değişiklerini izleme mekanizması kurulacaktır.

Adım 2.1: Mevcut Şemanın Dışa Aktarılması

Veritabanı sürüm yönetiminin temel adımlarından biri, mevcut (baseline) veritabanı şemasının tam bir anlık görüntüsünü (snapshot) oluşturmaktır. Bu anlık görüntü, gelecekteki değişiklikleri karşılaştırmak veya gerektiğinde eski bir sürüm'e dönmemek için

referans noktası görevi görür. Bu projede tercih edilen yöntem DACPAC (Data-Tier Application Component Package) Çıkarma'dır.

Veritabanının tüm şema nesnelerini (tablolar, view'lar, prosedürler, fonksiyonlar, indeksler, kullanıcılar vb.) içeren bir .dacpac dosyası oluşturulur.

Bu DACPAC dosyası, şemanın taşınabilir, model tabanlı bir temsilidir ve "v1" olarak kabul edilen temel şema versyonunu kayıt altına alır. Bu dosya, sürüm kontrol sistemlerinde saklanabilir ve sonraki adımlarda şema karşılaştırma işlemleri için kaynak olarak kullanılır.

Bu adımda doğrudan çalıştırılacak bir T-SQL sorgusu yoktur; işlem, belirtilen yönetim araçları üzerinden gerçekleştirilir ve sonuç olarak bir .dacpac dosyası elde edilir.

Adım 2.2: Hedef Sürüm (v2) Değişikliklerinin Tanımlanması

Mevcut veritabanı şemasının anlık görüntüsü (v1.dacpac) alındıktan ve dbo.openpowerlifting tablosunun yapısı incelendikten sonra, "v2" sürümü için planlanan yapısal değişiklikler ve iyileştirmeler bir fark tablosunda değerlendirilmek için veri tabanının yapısını öğrenmemiz gereklidir. Bunun için aşağıdaki sorgu çalıştırılarak sonucu incelenmesi uygundur.

```
USE powerlifting_db;
GO
EXEC sp_help 'dbo.openpowerlifting';
GO
```

| | Name | Owner | Type | Created_datetime | | | | | | |
|----|-------------------------------|---------|------------|-------------------------|------|-------|----------|--------------------|----------------------|---------------|
| 1 | openpowerlifting | dbo | user table | 2025-05-03 20:58:05.800 | | | | | | |
| 1 | Column_name | Type | Computed | Length | Prec | Scale | Nullable | TrimTrailingBlanks | FixedLenNullInSource | Collation |
| 1 | Name | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 2 | Sex | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 3 | Event | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 4 | Equipment | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 5 | Age | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 6 | AgeClass | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 7 | Division | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 8 | BodyweightKg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 9 | WeightClassKg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 10 | Squat1Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 11 | Squat2Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 12 | Squat3Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 13 | Squat4Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 14 | Best3SquatKg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 15 | Bench1Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 16 | Bench2Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 17 | Bench3Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 18 | Bench4Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 19 | Best3BenchKg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 20 | Deadlift1Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 21 | Deadlift2Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 22 | Deadlift3Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 23 | Deadlift4Kg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 24 | Best3Deadlift... | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 25 | TotalKg | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 26 | Place | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| 27 | Wilks | varchar | no | -1 | | | yes | no | yes | Turkish_CI_AS |
| | Identity | Seed | Increment | Not For Replication | | | | | | |
| 1 | No identity column defined. | NULL | NULL | NULL | | | | | | |
| | RowGuidCol | | | | | | | | | |
| 1 | No rowguidcol column defined. | | | | | | | | | |
| | Data_located_on_filegroup | | | | | | | | | |
| 1 | PRIMARY | | | | | | | | | |

Yukarıdaki sorgu sonucunun analiz edilmesine dayanarak, powerlifting_db veritabanının "v2" sürümü için aşağıdaki gibi iyileştirmeler ve değişiklikler planlayabiliriz.

| A | B | C | D | E |
|------------|---|--------------------|---------------------------------------|---|
| Objet Türü | Ad (Schema.Obj) | v1 Durumu | v2 İhtiyaç | Notlar |
| Column | dbo.openpowerlifting.LiftID | Yok | Eklenecek (INT IDENTITY(1,1) PK) | Benzersiz Kayıt ID'si, Birincil Anahtar |
| Column | dbo.openpowerlifting.Age | varchar (Nullable) | INT (Nullable) | Daha uygun veri tipi |
| Column | dbo.openpowerlifting.BodyweightKg | varchar (Nullable) | DECIMAL(5, 2) (Nullable) | Daha uygun veri tipi (örn. 120.50 kg) |
| Column | dbo.openpowerlifting.TotalKg | varchar (Nullable) | DECIMAL(7, 2) (Nullable) | Daha uygun veri tipi (örn. 1100.50 kg) |
| Column | dbo.openpowerlifting.Wilks | varchar (Nullable) | DECIMAL(7, 4) (Nullable) | Daha uygun veri tipi (örn. 650.1234) |
| Column | dbo.openpowerlifting.Place | varchar (Nullable) | INT (Nullable) | Daha uygun veri tipi (Sayısal sıra) |
| Column | dbo.openpowerlifting.Name | varchar (Nullable) | varchar (NOT NULL) | İsim boş olmamalı |
| Column | dbo.openpowerlifting.Sex | varchar (Nullable) | varchar (NOT NULL) | Cinsiyet boş olmamalı |
| Column | dbo.openpowerlifting.Event | varchar (Nullable) | varchar (NOT NULL) | Event (SBD, BD vb.) boş olmamalı |
| Column | dbo.openpowerlifting.MeetName | Yok | Eklenecek (NVARCHAR(200) Nullable) | Yarışma adını ekle |
| Column | dbo.openpowerlifting.MeetDate | Yok | Eklenecek (DATE Nullable) | Yarışma tarihini ekle |
| Column | dbo.openpowerlifting.Federation | Yok | Eklenecek (NVARCHAR(50) Nullable) | Federasyon bilgisini ekle |
| Index | IX_openpowerlifting_Name | Yok | Eklenecek (Non-Clustered on Name) | İsim aramalarını hızlandır |
| Index | IX_openpowerlifting_MeetDate | Yok | Eklenecek (Non-Clustered on MeetDate) | Tarihe göre filtrelemeyi hızlandır |

Bu değişiklik listesi, v2 sürümünün hedeflerini yansıtır ve ilgili paydaşlar tarafından onaylandıktan sonra geliştirme ve uygulama için temel oluşturur.

Adım 2.3: Şema Karşılaştırma

SSMS'te doğrudan bir şema karşılaştırma aracı bulunmadığından, iki veritabanı sürümleri (örn. v1 ve v2) arasındaki farkları bulmak için manuel bir yöntem izlenebilir. Bu

yöntem, her iki sürümün şema script'lerini oluşturup bir metin karşılaştırma aracıyla incelemeyi içerir.

Adım 2.3.1: Kaynak Şema Script'ini Oluşturma (v1)

SSMS Object Explorer'da powerlifting_db veritabanı üzerinden generate script aracılığıyla karşılaştırmanın temeli olacak mevcut (v1) veritabanı şemasının tam bir SQL script'ini oluşturulur.

Adım 2.3.2: Hedef Şema Script'ini Oluşturma (v2)

Adım 2.2'de Fark Tablosu'nda tanımlanan tüm v2 değişikliklerinin uygulandığı bir veritabanı (örn. geliştirme veya test ortamındaki powerlifting_db_v2) üzerinde Adım 2.3.1'deki aynı adımlar ve aynı gelişmiş script seçenekleri tekrarlanır.

Eğer v2 değişikliklerinin uygulandığı ayrı bir veritabanı yoksa, bu adım atlanır ve karşılaştırma doğrudan Fark Tablosu'na göre yapılır veya sadece v1 script'i referans olarak kullanılır.

Adım 2.3.3: Script Dosyalarını Karşılaştırma

Oluşturulan iki script dosyası arasındaki farkları bularak v1'den v2'ye geçiş için gereken DDL değişikliklerini tespit edilir.

Adım 2.4: Şema Değişiklikleri Log Tablosunun Oluşturulması

Veritabanı şemasında yapılacak değişiklikleri DDL tetikleyicisi ile otomatik olarak izleyebilmek için, bu değişikliklere ait bilgilerin kaydedileceği bir log tablosuna ihtiyaç vardır. Bu adımda, dbo.SchemaChangesLog adında, DDL olayının tipi, zamanı, ilgili nesne, işlemi yapan kullanıcı ve çalıştırılan T-SQL komutu gibi bilgileri içerecek olan log tablosu oluşturulacaktır.

```
IF OBJECT_ID('dbo.SchemaChangesLog', 'U') IS NULL
BEGIN
    CREATE TABLE dbo.SchemaChangesLog (
        LogID INT IDENTITY(1,1) PRIMARY KEY,
        EventTime DATETIME DEFAULT GETDATE(),
        EventType NVARCHAR(100),
        ObjectSchema NVARCHAR(100) NULL,
        ObjectName NVARCHAR(100) NULL,
        TSQLCommand NVARCHAR(MAX),
        LoginName NVARCHAR(100)
    );
    PRINT 'dbo.SchemaChangesLog tablosu oluşturuldu.';
END
ELSE
BEGIN
    PRINT 'dbo.SchemaChangesLog tablosu zaten mevcut.';
END
GO
```

Yukarıdaki sorgu çalıştırıldığında, şema değişikliklerini loglamak için kullanılacak hedef tablo (dbo.SchemaChangesLog) veritabanında oluşturulur veya zaten var olduğu belirtilir. Bu tablo, bir sonraki adımda oluşturulacak DDL tetikleyicisi için bir önkoşuldur.

Adım 2.5: Şema Değişikliklerini İzlemek İçin DDL Tetikleyicisi Oluşturma

Veritabanı şemasında kimin, ne zaman, ne değişiklik yaptığı takip etmek için otomatik bir mekanizma kurmak önemlidir. Bu adımda, powerlifting_db veritabanı üzerinde gerçekleşen belirli DDL (Veri Tanımlama Dili - örn. CREATE, ALTER, DROP) olaylarını yakalayan bir DDL Tetikleyicisi (DDL Trigger) oluşturulacaktır. Bu tetikleyici, yakaladığı olayın detaylarını (olay tipi, nesne adı, T-SQL komutu, yapan kullanıcı vb.) EVENTDATA() fonksiyonu aracılığıyla alacak ve Adım 2.4'te oluşturduğumuz dbo.SchemaChangesLog tablosuna kaydedecektir. Bu örnekte, tetikleyici şimdilik sadece Tablo ve İndeks olaylarını (DDL_TABLE_EVENTS, DDL_INDEX_EVENTS) izleyecektir.

```

IF EXISTS (SELECT * FROM sys.triggers WHERE name = 'trg_LogSchemaChanges' AND parent_class_desc = 'DATABASE')
BEGIN
    DROP TRIGGER trg_LogSchemaChanges ON DATABASE;
    PRINT 'Mevcut trg_LogSchemaChanges DDL tetikleyicisi silindi.';
END
GO

-- Yeni DDL tetikleyicisini oluştur
CREATE TRIGGER trg_LogSchemaChanges
ON DATABASE
FOR DDL_TABLE_EVENTS, DDL_INDEX_EVENTS

AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @EventData XML = EVENTDATA();
    DECLARE @EventType NVARCHAR(100);
    DECLARE @ObjectSchema NVARCHAR(100);
    DECLARE @ObjectName NVARCHAR(100);
    DECLARE @TSQLCommand NVARCHAR(MAX);
    DECLARE @LoginName NVARCHAR(100);

    SET @EventType = @EventData.value('/EVENT_INSTANCE/EventType[1]', 'NVARCHAR(100)');
    SET @ObjectSchema = @EventData.value('/EVENT_INSTANCE/SchemaName[1]', 'NVARCHAR(100)');
    SET @ObjectName = @EventData.value('/EVENT_INSTANCE/ObjectName[1]', 'NVARCHAR(100)');
    SET @TSQLCommand = @EventData.value('/EVENT_INSTANCE/TSQLCommand/CommandText[1]', 'NVARCHAR(MAX)');
    SET @LoginName = @EventData.value('/EVENT_INSTANCE/LoginName[1]', 'NVARCHAR(100)');

    INSERT INTO dbo.SchemaChangesLog
        (EventType, ObjectSchema, ObjectName, TSQLCommand, LoginName)
    VALUES
        (@EventType, @ObjectSchema, @ObjectName, @TSQLCommand, @LoginName);
END;
GO

```

Yukarıdaki soru çalıştırıldığında, trg_LogSchemaChanges adında veritabanı seviyesinde bir DDL tetikleyicisi oluşturulur (veya varsa güncellenir). Bu tetikleyici artık aktiftir ve powerlifting_db veritabanında bir tablo veya indeks üzerinde CREATE, ALTER veya DROP işlemi yapıldığı anda otomatik olarak çalışacak ve işlem detaylarını dbo.SchemaChangesLog tablosuna kaydedecektir.

Adım 2.6: DDL Tetikleyicisinin Test Edilmesi

Bu adım, tetikleyicinin beklentiği gibi çalışıp çalışmadığını görmek için kontrollü bir DDL işlemi yapmayı ve ardından log tablosunu incelemeyi içerir.

Adım 2.6.1: Test Amaçlı Tablo Oluşturma

DDL tetikleyicisini (trg_LogSchemaChanges) test etmek amacıyla, kapsamı dahilinde olan (DDL_TABLE_EVENTS) basit bir DDL komutu çalıştırılacaktır. Bunun için dbo.TestTableForDDL adında geçici bir test tablosu oluşturulacaktır. Bu işlemin tetikleyicimizi çalıştırması ve log tablosuna bir kayıt eklemesi beklenir.

```

USE powerlifting_db;
GO

-- Tetikleyiciyi test etmek için basit bir tablo oluştur
CREATE TABLE dbo.TestTableForDDL (
    TestID INT PRIMARY KEY,
    TestData VARCHAR(50)
);
GO
PRINT 'dbo.TestTableForDDL tablosu test amacıyla oluşturuldu.';
GO

```

Yukarıdaki CREATE TABLE komutu çalıştırıldığında, test tablosu oluşturulur. Bu işlemin trg_LogSchemaChanges DDL tetikleyicisini çalıştırılmış olması gereklidir.

Adım 2.6.2: Log Tablosunun Kontrol Edilmesi

CREATE TABLE işleminin loglanıp loglanmadığını görmek için dbo.SchemaChangesLog tablosu sorgulanacaktır. En son olayın en üstte görünmesi için EventTime'a göre tersten sıralama yapılması faydalıdır. CREATE_TABLE tipinde yeni bir kayıt görmeyi bekliyoruz.

```

-- ----- Son 5 kaydı gösterelim
SELECT TOP 5 -- Son 5 kaydı gösterelim
    LogID,
    EventTime,
    EventType,
    ObjectSchema,
    ObjectName,
    LoginName,
    TSQLCommand
FROM
    dbo.SchemaChangesLog
ORDER BY
    EventTime DESC;
GO

```

Yukarıdaki sorgu çalıştırıldığında sonuçları aşağıda görüldüğü gibidir. Sonuçlarda, en güncel kaydın (LogID 2) EventType değerinin CREATE_TABLE ve ObjectName değerinin TestTableForDDL olduğu görülmektedir. Ayrıca çalıştırılan T-SQL komutu da loglanmıştır. Bu, DDL tetikleyicisinin test tablosunun oluşturulma olayını başarıyla yakaladığını doğrular.

| LogID | EventTime | EventType | ObjectSchema | ObjectName | LoginName | TSQLCommand |
|-------|-------------------------|--------------|--------------|------------------|--------------------|---|
| 1 | 2025-05-04 01:45:55.130 | CREATE_TABLE | dbo | TestTableForDDL | DESKTOP-OS7BMJMeih | CREATE TABLE dbo.TestTableForDDL (TestID INT PRIMARY KEY, TestData VARCHAR(50)) |
| 2 | 2025-05-04 01:39:52.623 | CREATE_TABLE | dbo | SchemaChangesLog | DESKTOP-OS7BMJMeih | CREATE TABLE dbo.SchemaChangesLog (LogID INT IDENTITY(1,1) PRIMARY KEY, EventTime DATETIME DEFAULT GETDATE()) |

Adım 2.6.3: Test Amaçlı Tablonun Silinmesi (DDL İşlemi)

Test işlemini tamamlamak ve ortamı temizlemek için oluşturulan test tablosu şimdilik silinecektir. Bu DROP TABLE işlemi de tetikleyici tarafından yakalanmalı ve loglanmalıdır.

```

IF OBJECT_ID('dbo.TestTableForDDL', 'U') IS NOT NULL
BEGIN
    DROP TABLE dbo.TestTableForDDL;
    PRINT 'dbo.TestTableForDDL tablosu test amacıyla silindi.';
END
ELSE
BEGIN
    PRINT 'dbo.TestTableForDDL tablosu bulunamadı.';
END
GO

```

Yukarıdaki DROP TABLE komutu çalıştırıldığında, test tablosu silinir. Bu işlemin de trg_LogSchemaChanges DDL tetikleyicisini çalıştırılmış olması gereklidir.

Adım 2.6.4: Log Tablosunun Kontrol Edilmesi (İkinci Kontrol)

DROP TABLE işleminin loglanıp loglanmadığını görmek için dbo.SchemaChangesLog tablosu tekrar sorgulanacaktır. Şimdi en üstte DROP_TABLE tipinde yeni bir kayıt görmeyi bekliyoruz.

```

SELECT TOP 5
    LogID,
    EventTime,
    EventType,
    ObjectSchema,
    ObjectName,
    LoginName,
    TSQLCommand
FROM
    dbo.SchemaChangesLog
ORDER BY
    EventTime DESC;
GO

```

Yukarıdaki sorgu çalıştırıldığında sonuçları aşağıda görüldüğü gibidir. Sonuçlarda, en güncel kaydın (LogID 3) EventType değerinin DROP_TABLE ve ObjectName değerinin TestTableForDDL olduğu görülmektedir. Bu, DDL tetikleyicisinin test tablosunun silinme olayını da başarıyla yakalayıp logladığını teyit eder. Tetikleyici mekanizması bekleniği gibi çalışmaktadır.

| | LogID | EventTime | EventType | ObjectSchema | ObjectName | LoginName | TSQLCommand |
|---|-------|-------------------------|--------------|--------------|------------------|-----------------------|---|
| 1 | 3 | 2025-05-04 01:46:26.103 | CREATE_TABLE | dbo | TestTableForDDL | DESKTOP-OS7IBMJ\Melih | CREATE TABLE dbo.TestTableForDDL (TestID INT PRIMARY KEY, TestData NVARCHAR(50)) |
| 2 | 2 | 2025-05-04 01:45:55.130 | DROP_TABLE | dbo | TestTableForDDL | DESKTOP-OS7IBMJ\Melih | DROP TABLE dbo.TestTableForDDL |
| 3 | 1 | 2025-05-04 01:39:52.623 | CREATE_TABLE | dbo | SchemaChangesLog | DESKTOP-OS7IBMJ\Melih | CREATE TABLE dbo.SchemaChangesLog (LogID INT IDENTITY(1,1) PRIMARY KEY, LogTime DATETIME, EventType NVARCHAR(50), ObjectSchema NVARCHAR(100), ObjectName NVARCHAR(100), LoginName NVARCHAR(100), TSQLCommand NVARCHAR(MAX)) |

Bölüm 3: Test ve Geri Dönüş Planı

Bu bölüm, Bölüm 1'de planlanan veritabanı yükseltme veya Bölüm 2 kapsamında uygulanan önemli şema değişiklikleri sonrasında gerçekleştirilecek kritik adımları ele alır: Test ve Geri Dönüş (Rollback). Test süreci, yapılan değişiklıkların bekleniği gibi çalıştığını, sistemin kararlılığını ve performansını doğrulamayı amaçlar. Geri Dönüş Planı ise, testler sırasında veya değişiklik sonrası operasyonda ciddi sorunlar yaşanması durumunda, sistemi bilinen son kararlı duruma güvenli bir şekilde döndürmek için izlenecek adımları tanımlar. Bu bölümdeki adımlar da büyük ölçüde planlama ve prosedür tanımlama odaklıdır.

Adım 3.1: Yükseltme Sonrası Testlerin Gerçekleştirilmesi

Yükseltme işlemi (veya önemli değişiklikler) tamamlandıktan sonra, sistemin sağlığını ve bekleniği gibi çalıştığını doğrulamak için bir dizi test gerçekleştirilir. Bu adımda, temel bazı testler SQL sorguları aracılığıyla uygulanacak ve sonuçları belgeleneciktir.

Adım 3.1.1: Temel Bağlantı ve Kullanıcı Doğrulama

İlk kontrol, veritabanına başarılı bir şekilde bağlanıldığını ve doğru kullanıcı bağlamında olunduğunu teyit etmektir.

```
SELECT DB_NAME() AS CurrentDatabase;
SELECT SUSER_SNAME() AS CurrentLogin;
SELECT USER_NAME() AS CurrentUser;
GO
```

Yukarıdaki sorgu çalıştırıldığında sonuçları aşağıda görüldüğü gibidir. Sonuçlar, bağlantının powerlifting_db veritabanına yapıldığını ve işlemi gerçekleştiren login/kullanıcı bilgilerini doğrulamaktadır. Temel erişimde bir sorun olmadığı teyit edilmiştir.

| Results | |
|---------|-----------------------|
| | CurrentDatabase |
| 1 | powerlifting_db |
| | CurrentLogin |
| 1 | DESKTOP-OS7IBMJ\Melih |
| | CurrentUser |
| 1 | dbo |

Adım 3.1.2: Fonksiyonel Test

Veritabanının temel veri işleme (CRUD - Create, Read, Update, Delete) fonksiyonlarının çalıştığını test etmek için geçici bir tablo üzerinde basit işlemler yapılacaktır.

```

PRINT 'Adım 3.1.2: Fonksiyonel Test Başlıyor...';
IF OBJECT_ID('dbo.TestCRUD', 'U') IS NOT NULL DROP TABLE dbo.TestCRUD; -- Varsa önce sil
CREATE TABLE dbo.TestCRUD (ID INT PRIMARY KEY, TestValue VARCHAR(10));
PRINT 'dbo.TestCRUD tablosu oluşturuldu.';
GO

-- 2. Veri Ekleme (INSERT)
INSERT INTO dbo.TestCRUD (ID, TestValue) VALUES (1, 'Test A');
PRINT '1 satır dbo.TestCRUD tablosuna eklendi.';
GO

-- 3. Veri Okuma (SELECT)
PRINT 'Eklenen veri okunuyor:';
SELECT ID, TestValue FROM dbo.TestCRUD WHERE ID = 1;
GO

-- 4. Veri Güncelleme (UPDATE)
UPDATE dbo.TestCRUD SET TestValue = 'Test B' WHERE ID = 1;
PRINT '1 satır dbo.TestCRUD tablosunda güncellendi.';
GO

-- 5. Güncellenmiş Veriyi Okuma (SELECT)
PRINT 'Güncellenen veri okunuyor:';
SELECT ID, TestValue FROM dbo.TestCRUD WHERE ID = 1;
GO

-- 6. Veri Silme (DELETE)
DELETE FROM dbo.TestCRUD WHERE ID = 1;
PRINT '1 satır dbo.TestCRUD tablosundan silindi.';
GO

-- 7. Tabloyu Silme (DROP)
DROP TABLE dbo.TestCRUD;
PRINT 'dbo.TestCRUD tablosu silindi.';
PRINT 'Adım 3.1.2: Fonksiyonel Test Tamamlandı.';
GO

```

Yukarıdaki SQL script bloğu hatasız bir şekilde çalıştırılmıştır. PRINT mesajları işlemlerin sırayla tamamlandığını göstermiş, SELECT sorgularının sonuçları ise verinin bekleniği gibi eklendiğini ('Test A') ve güncellendiğini ('Test B') doğrulamıştır. Bu, temel CRUD işlemlerinin veritabanı üzerinde sorunsuz çalıştığını teyit eder.

| | ID | TestValue |
|---|----|-----------|
| 1 | 1 | Test A |

| | ID | TestValue |
|---|----|-----------|
| 1 | 1 | Test B |

Adım 3.1.3: Veri Bütünlüğü Kontrolü

Veritabanının fiziksel ve mantıksal bütünlüğünde bir sorun olup olmadığını kontrol etmek için DBCC CHECKDB komutu çalıştırılacaktır. Bu komut, yükseltme sonrası rutin kontrollerin önemli bir parçasıdır. Sadece hataları göstermesi için WITH NO_INFOMSGS parametresi kullanılacaktır.

```

DBCC CHECKDB ('powerlifting_db') WITH NO_INFOMSGS;
GO

```

Yukarıdaki sorgu çalıştırıldığında herhangi bir hata mesajı döndürmemiştir. WITH NO_INFOMSGS kullanıldığı için sonuç döndürmemesi, powerlifting_db veritabanında herhangi bir tutarlılık veya bozulma hatası bulunmadığını gösterir. Veritabanı bütünlüğü testi başarıyla tamamlanmıştır.

Adım 3.2: Geri Dönüş (Rollback) Stratejisinin Belirlenmesi

Herhangi bir veritabanı yükseltme veya önemli değişiklik işleminde, işlerin beklenmedik bir şekilde ters gitmesi ihtimaline karşı net bir Geri Dönüş Stratejisi bulunmalıdır. Bu strateji, sistemi değişikliğin yapılmasından önceki, bilinen son kararlı duruma döndürmeyi amaçlar.

Bölüm 1, Adım 1.3'te bu proje için Yerinde Yükseltme (In-place Upgrade) yöntemi seçilmiştir. Bu yöntem için en güvenilir ve yaygın geri dönüş stratejisi şudur:

Ana Strateji: Yedekten Geri Dönme

Ön Koşul: Yükseltme işlemine başlamadan hemen önce, powerlifting_db veritabanının ve kritik sistem veritabanlarının tam yedeklerinin alınmış ve bu yedeklerin doğrulanmış olması zorunludur. (Yedek doğrulaması Adım 3.3'te ele alınacaktır). Sunucu seviyesinde önemli değişiklikler yapılıyorsa (bu senaryoda olmasa da), işletim sistemi veya sanal makine seviyesinde anlık görüntü almak da değerlendirilebilir.

Uygulama: Eğer yükseltme sonrası yapılan testler (Adım 3.1) kritik hatalar ortaya çıkarırsa veya sistem kararsız hale gelirse ve sorun kısa sürede çözülemese, Geri Dönüş Planı devreye alınır. Bu genellikle aşağıdaki adımları içerir:

- 1- Veritabanına gelen tüm bağlantıları durdurma.
- 2- SQL Server servislerini durdurma.
- 3- Yükseltme öncesi alınan tam yedekten powerlifting_db veritabanını geri yükleme (RESTORE DATABASE ... WITH REPLACE).
- 4- Sistem veritabanları da etkilendiye (genellikle büyük sürüm yükseltmelerinde daha olasıdır), onları da yedekten geri yükleme (bu daha karmaşık bir işlemidir).
- 5- SQL Server servislerini başlatma.
- 6- Temel bağlantı ve fonksiyon testleri yaparak sistemin eski kararlı duruma döndüğünü doğrulama.

Karar Verme: Geri dönüş kararının hangi koşullarda ve kim tarafından verileceği önceden belirlenmelidir.

Adım 3.3: Yedekleme ve Doğrulama Prosedürü

Adım 3.2'de belirlenen "Yedekten Geri Dönme" stratejisinin başarılı olabilmesi için, yükseltme veya değişiklik işlemine başlamadan hemen önce güvenilir bir tam veritabanı yedeğinin alınması ve bu yedeğin kullanılabilir olduğunun doğrulanması hayatı öneme sahiptir. Bu adım, bu prosedürleri tanımlar.

Adım 3.3.1: Yükseltme Öncesi Tam Yedek Alma

Yükseltme işlemine başlamadan önceki son adımlardan biri, powerlifting_db veritabanının tam yedeğini almaktır. Bu yedek, geri dönüş senaryosunda kullanılacak ana referanstır. Aşağıdaki komut, powerlifting_db için tam yedek alır.

```

USE master;
GO
-- powerlifting_db veritabanının tam yedegini al
]BACKUP DATABASE [powerlifting_db]
TO DISK = N'C:\SQLBackups\powerlifting_db_PreUpgrade_20250405_0246.bak'
WITH
    NAME = N'PowerliftingDB Pre-Upgrade Full Backup',
    DESCRIPTION = N'SQL Server yükseltmesi/güncellemesi öncesi alınan tam yedek.',
    COMPRESSION,
    STATS = 10;
GO

```

Yukarıdaki BACKUP DATABASE komutu çalıştırıldığında, powerlifting_db veritabanının tam yedeği belirtilen dosya yoluna kaydedilir. Komutun başarıyla tamamlandığına dair mesaj alınmalıdır. Bu yedek dosyası güvenli bir yerde saklanmalıdır. Aşağıda mevcut sonucu gösteren çıktı görülmektedir.

```

Messages
10 percent processed.
20 percent processed.
30 percent processed.
40 percent processed.
50 percent processed.
60 percent processed.
70 percent processed.
80 percent processed.
90 percent processed.
100 percent processed.
Processed 45536 pages for database 'powerlifting_db', file 'powerlifting_db' on file 1.
Processed 2 pages for database 'powerlifting_db', file 'powerlifting_db_log' on file 1.
BACKUP DATABASE successfully processed 45538 pages in 8.044 seconds (44.226 MB/sec).

Completion time: 2025-05-04T02:57:42.0566215+03:00

```

Adım 3.3.2: Yedek Dosyasını Doğrulama

Yedek dosyasının fiziksel olarak oluşturulmuş olması, onun geçerli ve geri yüklenebilir olduğu anlamına gelmez. Yedeğin bütünlüğünü ve okunabilirliğini kontrol etmek için RESTORE VERIFYONLY komutu kullanılır. Bu komut, yedeği gerçekten geri yüklemez, sadece başlıklarını okur ve temel bütünlük kontrollerini yapar.

```

USE master;
GO
-- Bir önceki adımda alınan yedek dosyasının bütünlüğünü doğrula
]RESTORE VERIFYONLY
FROM DISK = N'C:\SQLBackups\powerlifting_db_PreUpgrade_20250405_0246.bak'
-- WITH FILE = 1;
GO

```

Yukarıdaki RESTORE VERIFYONLY komutu çalıştırıldığında, "The backup set on file 1 is valid." mesajı başarıyla alınmıştır. Bu, alınan yedeğin temel bütünlük kontrollerini geçtiğini ve geri dönüş (restore) senaryosunda kullanılabilir olduğunu doğrular. Geri dönüş planı için gerekli olan geçerli yedek artık mevcuttur ve doğrulanmıştır. Aşağıda ilgili sorgunun çıktısı çıktı görülmektedir.

```

Messages
The backup set on file 1 is valid.

Completion time: 2025-05-04T02:57:56.2128148+03:00

```

Proje 7 - Veritabanı Yedekleme ve Otomasyon Çalışması

Kullanılan veri tabanı: Covid-19

Link: <https://www.kaggle.com/datasets/khushikyad001/covid-19-global-dataset>

Bu projede istenen adımlar 3 temel başlık altında değerlendirilmiş:

- 5- Yedekleme Süreçlerini Otomatikleştirme: SQL Server Agent
- 6- Yedekleme Raporları Oluşturma: Powershell veya T-SQL Scripting
- 7- Otomatik Yedekleme Uyarıları

Biz de yaptığımız adımları bu başlıklar üzerinde değerlendirecek olursak.

Bölüm 1: Yedekleme Süreçlerini Otomatikleştirme

Öncelikle log'ları saklayabilmemiz için bir Log Tablosu oluşturduk.

Veritabanı Log Tablosu Oluşturma

```
[1] 1 USE [covid];
2 GO
3
4 CREATE TABLE BackupLog (
5     id INT IDENTITY(1,1) PRIMARY KEY,
6     database_name NVARCHAR(100),
7     backup_start DATETIME,
8     backup_end DATETIME,
9     backup_size_MB FLOAT,
10    backup_path NVARCHAR(500),
11    error_message NVARCHAR(MAX),
12    log_date DATETIME DEFAULT GETDATE()
13 );
```

Commands completed successfully.

Commands completed successfully.

Total execution time: 00:00:00.159

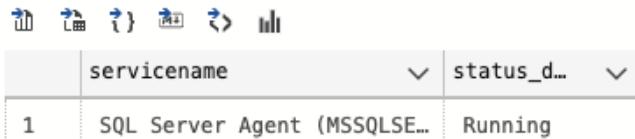
Sonrasında ise SQL Agent durumumuza baktık. İstediğimiz sonuç "Running" durumunda olmasıdır. "Stopped" durumunda ise tekrardan başlatmamız gereklidir. Aşağıda verilen kodu çalıştırıldığımızda çıktısının "Running" şeklinde olduğunu gözlemleyebiliriz.

SQL Server Agent Kontrolü

```
[2] 1  SELECT servicename, status_desc  
2    FROM sys.dm_server_services  
3   WHERE servicename LIKE '%SQL Server Agent%';
```

(1 row affected)

Total execution time: 00:00:00.079



| | servicename | status_d... |
|---|------------------------------|-------------|
| 1 | SQL Server Agent (MSSQLSE... | Running |

Sonrasında operatör ve bildirim ayarlarına geçtik. İlk olarak e posta bildirimi için bir operatör tanımladık.

Operatör ve Bildirim Ayarları

```
[9] 1  USE msdb;  
2  GO  
3  
4  EXEC dbo.sp_add_operator  
5      @name = N'DBA_Team_2',  
6      @email_address = N'persuren@gmail.com';  
7  GO
```

Commands completed successfully.

Commands completed successfully.

Total execution time: 00:00:00.020

Bu adımdan sonra yedekleme job'ını oluşturduk ve ona bildirim ekledik.

```
[10] 1 USE msdb;
2 GO
3
4 EXEC dbo.sp_add_job @job_name = N'CovidDB_Backup_Job_New';
5 GO
```

Commands completed successfully.

Commands completed successfully.

Total execution time: 00:00:00.052

```
[11] 1 USE msdb;
2 GO
3
4 EXEC dbo.sp_update_job
5     @job_name = N'CovidDB_Backup_Job_New',
6     @notify_level_email = 2,
7     @notify_email_operator_name = N'DBA_Team_2';
8 GO
```

Commands completed successfully.

Commands completed successfully.

Total execution time: 00:00:00.029

Bir sonraki adıma geldiğimizde yedekleme job'ını ve adımlarını oluşturduk. İlk adımda veritabanımızın yedeğini aldık.

```
[12] 1 USE msdb;
2 GO
3
4 EXEC sp_add_jobstep
5     @job_name = N'CovidDB_Backup_Job_New',
6     @step_name = N'Full_Backup',
7     @subsystem = N'TSQL',
8     @command = N'
9         DECLARE @path NVARCHAR(500);
10        SET @path = N''/var/opt/mssql/backups/covid_''
11        + CONVERT(VARCHAR, GETDATE(), 112) + ''_'''
12        + REPLACE(CONVERT(VARCHAR, GETDATE(), 108), '':'', '') + ''.bak'';
13        BACKUP DATABASE [covid] TO DISK = @path WITH COMPRESSION, STATS = 10;
14        ,
15        @database_name = N'master';
16 GO
17'
```

İkinci adıma geldiğimizde ise log temizleme işlemi yaptık. Bunun için de 7 günden eski olan yedekleri sildik.

```
18
19 EXEC sp_add_jobstep
20     @job_name = N'CovidDB_Backup_Job_New',
21     @step_name = N'Cleanup_Old_Backups',
22     @command = N'
23         EXEC xp_delete_file 0, N''/var/opt/mssql/backups/'', N'bak'', DATEADD(DAY, -7, GETDATE());
24     ',
25     @database_name = N'master';
26 GO
27
28
```

Üçüncü adımda ise raporlamayı gerçekleştirdik. Bu adımda log'larımızı Log Tablosuna kaydettik.

```
29
30 EXEC sp_add_jobstep
31     @job_name = N'CovidDB_Backup_Job_New',
32     @step_name = N'Log_Backup_Info',
33     @command = N'
34         INSERT INTO covid.dbo.BackupLog (database_name, backup_start, backup_end, backup_size_MB, backup_path)
35             SELECT
36                 database_name,
37                 backup_start_date,
38                 backup_finish_date,
39                 backup_size/1024/1024,
40                 physical_device_name
41             FROM msdb.dbo.backupset bs
42             JOIN msdb.dbo.backupmediafamily bmf ON bs.media_set_id = bmf.media_set_id
43             WHERE bs.database_name = ''covid''
44                 AND bs.type = ''D''
45                 AND backup_start_date >= DATEADD(HOUR, -1, GETDATE());
46         ',
47     @database_name = N'master';
48 GO
```

Oluşturduğumuz job'a son olarak zamanlama ve aktivasyon ekledik. Öncelikle zamanlama oluşturduk.

```
1
2 EXEC dbo.sp_add_schedule
3     @schedule_name = N'Daily_1AM',
4     @freq_type = 4,
5     @freq_interval = 1,
6     @active_start_time = 010000;
7 GO
8
```

Job'a oluşturduğumuz bu zamanlamayı ekledik.

```
9  
10 EXEC sp_attach_schedule  
11     @job_name = N'CovidDB_Backup_Job_New',  
12     @schedule_name = N'Daily_1AM';  
13 GO  
14
```

Son olarak job'ı sunucuya atadık. Bu şekilde projenin ilk adımını başarılı bir şekilde çalıştırılmış olduk.

```
14  
15  
16 EXEC dbo.sp_add_jobserver  
17     @job_name = N'CovidDB_Backup_Job_New';  
18 GO
```

Bölüm 2: T-SQL ile Rapor Oluşturma

İlk olarak hata almayı önlemek için Backup Log Tablosu'nu kontrol ettik ve olmadığı durumda sıfırdan oluşturmasını istedik.

```
[2] 1 USE covid;  
2 GO  
3  
4 IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'BackupLog')  
5 BEGIN  
6     CREATE TABLE BackupLog (  
7         log_id INT IDENTITY(1,1) PRIMARY KEY,  
8         database_name NVARCHAR(255),  
9         error_message NVARCHAR(MAX),  
10        backup_date DATETIME DEFAULT GETDATE()  
11    );  
12 END;  
13 GO
```

Commands completed successfully.

Commands completed successfully.

Total execution time: 00:00:00.039

Sonrasında hata yönetimi için bir Stored Procedure oluşturduk.

```

[3] 1 USE covid;
2 GO
3
4 CREATE OR ALTER PROCEDURE usp_Backup_CovidDB
5 AS
6 BEGIN
7     BEGIN TRY
8         DECLARE @path NVARCHAR(500);
9         SET @path = N'/var/opt/mssql/backups/covid_'
10        + CONVERT(VARCHAR, GETDATE(), 112) + '_'
11        + REPLACE(CONVERT(VARCHAR, GETDATE(), 108), ':', '') + '.bak';
12
13        BACKUP DATABASE [covid] TO DISK = @path WITH COMPRESSION, STATS = 10;
14    END TRY
15    BEGIN CATCH
16        INSERT INTO BackupLog (database_name, error_message)
17        VALUES ('covid', ERROR_MESSAGE());
18    END CATCH
19 END;

```

^

Commands completed successfully.

Commands completed successfully.

Total execution time: 00:00:00.030

Bu kodu çalıştırıldıktan sonra Backup tablomuzu kontrol ettik. Beklentimiz tablonun sütunlarının dönmesi, satırların ise boş olması. Bunun sebebi şuan için aldığımız bir hata olmamasıdır. İleriki adımlarda kasıtlı olarak hata yaratıp bu tabloya kaydedilip edilmeyeceğini test edeceğiz.

```

[4] 1 USE covid;
2 GO
3
4 SELECT * FROM BackupLog;
5 GO

```

^

Commands completed successfully.

(0 rows affected)

Total execution time: 00:00:00.019

Göründüğü üzere tam olarak istediğimiz sonuca ulaştık.

Bölüm 3: Otomatik Yedekleme Uyarıları

Veritabanı yedekleme işleminde bir hata ile karşılaşılması durumunda yöneticilere bildirim göndermenin en temel yolu mail sistemini kullanmaktadır. Bu sebeple öncelikle mail sistemini aktif duruma getirdik. Kullandığımız “msdb” sistem veritabanı, SQL Server Agent tarafından kullanılan görevlerin ve mail profillerinin saklandığı veritabanıdır.

```
[1] 1 USE msdb;
2 GO
3 EXEC sp_configure 'show advanced options', 1;
4 RECONFIGURE;
5 EXEC sp_configure 'Database Mail XPs', 1;
6 RECONFIGURE;
7 GO
```

Commands completed successfully.

- `sp_configure`: SQL Server yapılandırma ayarlarını görmek veya değiştirmek için kullanılır.
- `'show advanced options', 1`: Gelişmiş seçeneklerin görünür olmasını sağlar.
- `RECONFIGURE`: Yapılan ayarın sunucuya uygulanmasını sağlar.
- `'Database Mail XPs', 1`: Database Mail özelliğini etkinleştirir (1 aktif, 0 pasif).
- `Database Mail XPs (Extended Procedures)`: SQL Server'ın e-posta gönderme fonksiyonlarını sağlayan bileşendir.

Sonrasında E-posta hesabının tanımlanması aşamasına geçtik.

```
[1] 1 EXEC msdb.dbo.sysmail_add_account_sp
2     @account_name = 'DBA_Notifications',
3     @email_address = 'persuren@gmail.com',
4     @display_name = 'SQL Server DBA Alerts',
5     @mailserver_name = 'smtp.gmail.com',
6     @port = 587,
7     @username = 'persuren',
8     @password = 'A93dp3!',
9     @enable_ssl = 1;
10    GO
```

```

11
12 EXEC msdb.dbo.sysmail_add_profile_sp
13      @profile_name = 'DBA_Profile';
14 GO
15
16 EXEC msdb.dbo.sysmail_add_profileaccount_sp
17      @profile_name = 'DBA_Profile',
18      @account_name = 'DBA_Notifications',
19      @sequence_number = 1;
20 GO

```

Bu işlemleri gerçekleştirebilmek için biz "gmail" hesabımızı kullandık. Orda yer alan SMTP ayarlarından mail adresimiz için gerekli bilgileri topladık ve yeni bir şifre oluşturduk. Bu sayede bir hesap oluşturduk ve sonrasında profilimizi oluşturduk. Son adımımda hesabımızı profile eklemiş olduk.

```

[3] 1  CREATE OR ALTER PROCEDURE usp_Backup_CovidDB
2  AS
3  BEGIN
4      BEGIN TRY
5          DECLARE @path NVARCHAR(500) = N'C:\Backup\covid.bak';
6          BACKUP DATABASE [covid] TO DISK = @path WITH COMPRESSION;
7      END TRY
8      BEGIN CATCH
9          DECLARE @errorMessage NVARCHAR(MAX) = ERROR_MESSAGE();
10
11         INSERT INTO BackupLog (database_name, error_message)
12         VALUES ('covid', @errorMessage);
13
14         EXEC msdb.dbo.sp_send_dbmail
15             @profile_name = 'DBA_Profile',
16             @recipients = 'persuren@gmail.com',
17             @subject = 'COVID Veritabanı Yedekleme Hatası',
18             @body = 'Yedekleme işlemi sırasında şu hata oluştu: ' + @errorMessage;
19     END CATCH
20 END;
21 GO

```

^

Commands completed successfully.

Total execution time: 00:00:00.088

Bu kod sayesinde hatalı işlem durumunda tablomuza loglar kaydedilecek ve hata mesajı mail olarak gönderilecek şekilde ayarlandı.

```
[10] 1 USE covid;
2 GO
3
4 CREATE OR ALTER PROCEDURE usp_Backup_CovidDB
5 AS
6 BEGIN
7     BEGIN TRY
8         DECLARE @path NVARCHAR(500) = N'/invalid_directory/covid.bak';
9         BACKUP DATABASE [covid] TO DISK = @path WITH COMPRESSION, STATS = 10;
10    END TRY
11    BEGIN CATCH
12        INSERT INTO BackupLog (database_name, error_message)
13        VALUES ('covid', ERROR_MESSAGE());
14    END CATCH
15 END;
16 GO
```

Commands completed successfully.

Commands completed successfully.

Total execution time: 00:00:00.039

Test etmek için hatalı bir adres eklenerek backup alınmaya çalışıldı.

```
[11] 1 USE covid;
2 GO
3 EXEC usp_Backup_CovidDB;
```

Commands completed successfully.

(1 row affected)

Total execution time: 00:00:00.120

Bu şekilde procedure'ü çalıştırılmış olduk. Tablomuzda bunu kontrol etmek istediğimiz ise;

```
[12] 1 USE covid;
2 GO
3 SELECT * FROM BackupLog;
```

Commands completed successfully.

(1 row affected)

Total execution time: 00:00:00.060

| | i. | database_na... | backup_st... | backup_e... | backup_size_... | backup_pa... | error_message | log_date |
|---|----|----------------|--------------|-------------|-----------------|--------------|--------------------------------------|------------|
| 1 | 1 | covid | NULL | NULL | NULL | NULL | BACKUP DATABASE is terminating ab... | 2025-05-04 |

Satırımızın başarılı bir şekilde eklendiğini gördük.

Bu yapılan işlemleri alternatif olarak Python scripti ile de yapabiliyoruz.

```
1 import smtplib
2 from email.mime.text import MIMEText
3
4 def send_email(error_message):
5     sender = "persuren@gmail.com"
6     receiver = "ofyeter60@gmail.com"
7     password = "akl 34 abcyd38!"
8
9     msg = MIMEText(f"Hata Mesajı: {error_message}")
10    msg['Subject'] = "SQL Server Backup Hatası"
11    msg['From'] = sender
12    msg['To'] = receiver
13
14    with smtplib.SMTP("smtp.gmail.com", 587) as server:
15        server.starttls()
16        server.login(sender, password)
17        server.sendmail(sender, receiver, msg.as_string())
18
```

Python script dosyamızın içeriği bu şekildedir. Buna uygun olacak formatta prosedürü düzenlememiz gereklidir.

```
[14] 1 USE covid;
2 GO
3 CREATE OR ALTER PROCEDURE usp_Backup_CovidDB
4 AS
5 BEGIN
6     BEGIN TRY
7         DECLARE @path NVARCHAR(500) = N'/invalid_path/covid.bak';
8         BACKUP DATABASE [covid] TO DISK = @path WITH COMPRESSION;
9     END TRY
10    BEGIN CATCH
11        DECLARE @errorMessage NVARCHAR(MAX) = ERROR_MESSAGE();
12        INSERT INTO BackupLog (database_name, error_message) VALUES ('covid', @errorMessage);
13
14        DECLARE @escapedErrorMessage NVARCHAR(MAX) = REPLACE(@errorMessage, '''', '\''');
15
16        DECLARE @cmd NVARCHAR(MAX) = N'python3 /scripts/send_email.py "' + @escapedErrorMessage + '"';
17        EXEC xp_cmdshell @cmd;
18    END CATCH
19 END;
20 GO
```

Düzenlediğimizde son hali bu şekilde olmaktadır.