

ANKARA ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



BLM4522 – Ağ Tabanlı Paralel Dağıtım Sistemleri

Proje Dökümantasyonu

25/04/2025

Melih ÜZEL 20290299

Pınar ERSÜREN 20290249

Github Profil:

Github Profil:

<https://github.com/virtuososlove>

<https://github.com/persuren>

Proje Linki: <https://github.com/virtuososlove/MSSQL-Project-Repo>

Proje 1: Veritabanı Performans Optimizasyonu ve İzleme

Kullanılan veritabanı: 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>

Video Linki: <https://drive.google.com/file/d/1-jKb38KlbptWWVQpCRb9F1URVMYDjd5D/view?usp=sharing>

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.

1- Veritabanı İzleme

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

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.

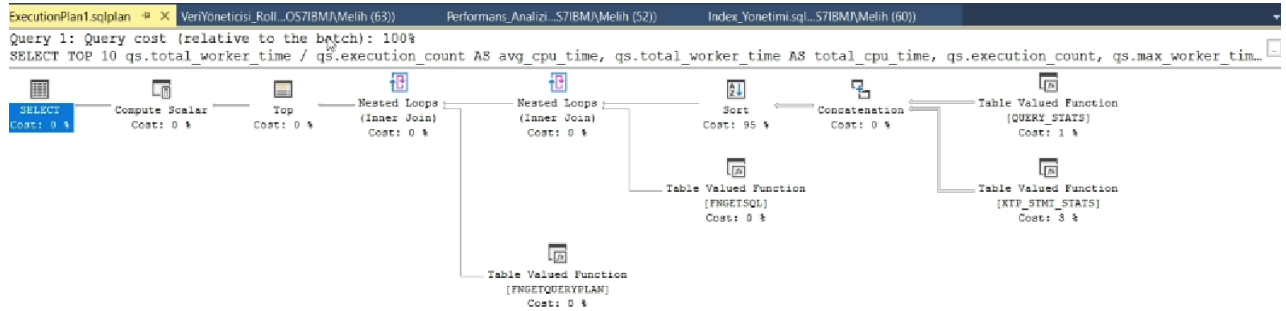
	avg_cpu_time	total_cpu_time	execution_count	max_cpu_time	avg_elapsed_time	total_elapsed_time	max_elapsed_time	query_text	query_plan
1	53834	484507	9	57176	74284	668556	78478	SELECT TOP 10 qs.total_worker_time / qs.execution_co...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	16385	147486	9	27571	21856	196706	33929	SELECT large_data FROM sys.dirty_session_large...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	49249	49249	1	49249	62895	62895	62895	SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], sp...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
4	21389	21389	1	21389	25029	25029	25029	SELECT SCHEMA_NAME(uf.schema_id) AS [Schema], u...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
5	17754	17754	1	17754	17795	17795	17795	WITH Quantiles AS (SELECT DISTINCT ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
6	1376	9537	7	1501	1835	12851	1971	insert #filetmpin select file_or_directory_name, 1 - is_dir...	NULL
7	670	9391	14	927	809	11338	1188	insert #filetmpin select file_or_directory_name, 1 - is_dir...	NULL
8	7938	7938	1	7938	7939	7939	7939	SELECT tbl.name AS [Name], tbl.object_id AS [ID], tbl.creat...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
9	958	7670	8	1054	1248	9991	1349	insert #filetmpin select file_or_directory_name, 1 - is_dir...	NULL
10	5309	5309	1	5309	9622	9622	9622	SELECT is_member(N'db_accessadmin') AS [aDbAccess...	<ShowPlanXML xmlns="http://schemas.microsoft.com...

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

Sorgu tipini ve query_plan başlığı altındaki Execution Planı incelediğimizde performansı hakkında bir takım bilgiler elde edebiliyoruz. Execution Plandaki 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 uyan daha verimli bir indeks bulunamamıştır.

Query_text kısmındaki sorgu tipine bakıldığında 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.

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

Sorgu çalıştırıldığında boş bir dönüş alındığı görülüyor. Sorgunun herhangi bir sonuç döndürmemesi, SQL Server'ın 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 çalıştırdığımızda 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	TotalRead
1	dbo	FactInternetSales	IX_FactInternetSales_OrderDateKey_Includes	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 bulunduğunu 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:

- Parçalanma: %0.56. Bu çok düşük bir oran.
- Sayfa Sayısı (page_count): 1238
- İstatistik Güncelleme (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 Güncelleme (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 gerekir. Eski istatistikler, Sorgu İyileştirici'nin sorgular için verimsiz yürütme planları seçmesine neden olabilir.

4- Veri Yöneticisi Roller

Son adımımız olan veri yöneticisi 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 sorunun 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 çalıştırıldıktan 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.

Results		Messages			
	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 veritabanı: 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>

VideoLinki: <https://drive.google.com/file/d/1lhhs955h3OutuYg9Rwi5b8uTrkyY-vNY/view?usp=sharing>

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.

1- Tam, Artık, Fark Yedeklemeleri

Öncelikle işe Docker üzerinden gerekli container'ı oluşturarak başladık.

Terminal üzerinden aşağıdaki kodu çalıştırdıktan 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;
```

SQL

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

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:

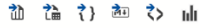
Öncelikle Agent’ımızın durumunu kontrol ederek başladık. Aşağıda bulunan komutu çalıştırdığımızda sonucun ‘running’ durumunda olduğunu gördük. Bu da bizim istediğimiz sonuçtu. Sonrasında da kendimiz için bir Job tanımlı oluşturduk.

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

SQL

(1 row affected)

Total execution time: 00:00:00.053



	servicename	status_desc
1	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_QUOTE(DATE)).bak'' WITH COMPRESSION, STATS =
13    @database_name = N'master';
14 GO
```

Oluşturduğumuz bu Job ile günlük her sabah 08:00 saatinde backup almasını 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/1FT7GK8yYlzd8SG7d3MNLk-71x0EzUuSy?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/1FT7GK8yYlzd8SG7d3MNLk-71x0EzUuSy?usp=sharing>” linkinde yer alıyor.

Son olarak şuna 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.

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).

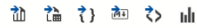
Total execution time: 00:00:00.386

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.

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 çalıştırdığımızda aşağıda gözüken containerlarımız oluşmuş ve çalışmaya başlamış oldu.

<input type="checkbox"/>	 sql_prima fdbcb296	mcr.micros	Running	1433:1433	3 hours ago	2.34	■	⋮	🗑
<input type="checkbox"/>	 sql_mirror 186f5e2fcc	mcr.micros	Running	1434:1433	3 hours ago	4.16	■	⋮	🗑

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

```
[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 çalıştırdık 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>

Video Linki:

https://drive.google.com/file/d/1KR9HGROutJxSyWJwpe9nDIB4c5EV_a6/view?usp=sharing

Bu projede istenen adımlar 4 temel başlık altında değerlendirilmiştir:

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

Biz de yaptığımız adımları bu başlıklar üzerinde değerlendirecek olursak.

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 VarsayilanVeritabani
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 bulunduğunu gösteriyor.

100 %

Results

Messages

	LoginAdi	LoginTipi	is_disabled	OlusturmaTarihi	VarsayilanVeritabani
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	SampleETLLLogin	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-OS7IBM\JMelih	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 girişi (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 VarsayilanVeritabani
FROM
    sys.server_principals
WHERE
    name = 'OrnekSqlKullanici';
GO

```

Bu sorgu sonucunda aşağıdaki görüldüğü gibi işlemin başarılı olduğunu görüyoruz.

	LoginAdi	LoginTipi	VarsayilanVeritabani
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 belirlemiş oluruz.

```

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 çalıştırdığımızda aşağıdaki sonuca ulaşırız. Bu da istenen yetkilerin kullanıcıya sağladığını gösterir.

Results				Messages				Results				Messages			
	VeritabanıKullaniciAdi	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 çalıştıralım. Aşağıdaki sorguları çalıştırdığımızda ilk sorgunun çalıştığını fakat ikinci sorgunun hata verdiğini görürüz

--Login ve Yetkiyi Test Etme

```
--İzin verilen sorgu
USE WideWorldImporters;
SELECT TOP 5 CustomerID, CustomerName FROM Website.Customers;
GO
```

--İzin verilmeyen sorgu

```
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 VeritabanıKullaniciAdi,
    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 çalıştırdığımız 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 kullanıcıymış gibi davranarak test etmeliyiz. Aşağıdaki sorgu bunu gerçekleştirir. Tam anlamıyla kullanıcıymış gibi davranarak 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
```

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 anahtarı oluşturduk. Şifreleme algoritması olarak AES_256'yı 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 OrjinalKategoriAdi,
    EncryptedCategoryName AS SifreliVeri,
    CONVERT(NVARCHAR(50), DECRYPTBYKEY(EncryptedCategoryName)) AS CozulmusKategoriAdi
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	OrjinalKategoriAdi	SifreliVeri	CozulmusKategoriAdi
1	1	Agent	0x00FBA550CCC63C4DB06B161093C6C0F60200000048BE68...	Agent
2	2	Wholesaler	0x00FBA550CCC63C4DB06B161093C6C0F6020000005C016E...	Wholesaler
3	3	Novelty Shop	0x00FBA550CCC63C4DB06B161093C6C0F602000000C24A30...	Novelty Shop

3- SQL Injection Testleri

İlk olarak bilerek güvenlik açıkları içeren bir soru yazarak çalıştırıyoruz.


```

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

```

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

```

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

	CustomerID	CustomerName	PhoneNumber
653	1051	Sylvie Laramée	(787) 555-0100
654	1052	Ian Olofsson	(339) 555-0100
655	1053	Luis Saucedo	(210) 555-0100
656	1054	Emma Salpa	(205) 555-0100
657	1055	Adriana Pena	(252) 555-0100
658	1056	Kalyani Benjaree	(212) 555-0100
659	1057	Ganesh Majumdar	(217) 555-0100
660	1058	Jaroslav Fisar	(215) 555-0100
661	1059	Jibek Juniskyz	(217) 555-0100
662	1060	Anand Mudaliyar	(206) 555-0100
663	1061	Agrita Abele	(206) 555-0100

Bu durumu çözmek için güvenli prosedür yani parametrelili 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	

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 çalıştıran 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 vardı.';
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 çalıştırdıktan sonra log kaydımız oluşmuş. Son olarak içeriğini görüntüleyerek sağlamasını yapalım.

Local Disk (C:) > test				Search 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 çalıştırdığımızda log sonuçlarımızı ve nelerin kayıt edildiğini gözlemleyebiliriz.

```
--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
```

event_time	action_id	succeeded	LoginName	DatabaseUserName	ServerName	DatabaseName	SchemaName	ObjectName	SqlStatement
2025-04-23 19:54:23.1000375	AUSC	1	DESKTOP...		DESKTOP...				
2025-04-23 20:03:10.0073207	SL	1	DESKTOP...	OrnekSqlKullanici_User	DESKTOP...	WideWorldImporters	Website	Customers	SELECT TOP 1 CustomerID, CustomerName FROM Websit...
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 gözlemlleme işlemini de başarıyla tamamladık.