

ANKARA ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ
BÖLÜMÜ



(BLM4522) Ağ Tabanlı Paralel
Dağıtım Sistemleri

Mertcan Özdemir

Ömer Faruk Karagöz

21290194

21290585

Github: mertcan-ozdemir

Github repo link:

<https://github.com/mertcan-ozdemir/BLM4522-21290194>

veritabanı güvenliği ve erişimi <https://youtu.be/B4G72MCuPuM>

veritabanı otomasyon ve bildirim https://youtu.be/_M4R-XZu6eM

veritabanı yedekleme ve felaketten kurtarma senaryosu

<https://youtu.be/xM3Ql4Mqlm4>

veritabanı performans optimizasyonu ve izleme

https://youtu.be/rLBt1m3E_UY

Veritabanı Yük dengeleme ve dağıtık veritabanı yapıları

<https://youtu.be/znoigYaJsPA>

Veri temizleme ve ETL süreçleri tasarımı <https://youtu.be/jt45tOfMzml>

Veri tabanı yükseltme ve sürüm yönetimi <https://youtu.be/OYtfAQXz3n0>

1. Veri tabanı Güvenliği ve Erişim Kontrolü

Bu bölümde MovieLensDB üzerinde gerçekleştirilen veritabanı güvenliği ve erişim kontrolü uygulamalarını kapsamaktadır. Temel olarak kullanıcı kimlik doğrulama yöntemleri, erişim izinlerinin yönetimi, SQL injection saldırılarına karşı alınan önlemler ve kullanıcı aktivitelerinin izlenmesi (audit loglama) konularına odaklanılmıştır.

1.1. Erişim Yönetimi

Veritabanına erişimi olan kullanıcıların kimlik doğrulama işlemleri SQL Server Authentication ve Windows Authentication ile sağlanabilir. Çalışmamızda SQL Authentication yöntemini tercih ettik.

```
-- SQL Auth ile yeni bir kullanıcı oluşturma
USE [master];
GO

CREATE LOGIN [mertcan_omer] WITH PASSWORD = 'MovieLens';
GO

USE MovieLensDB;
GO

CREATE USER [mertcan_omer] FOR LOGIN [mertcan_omer];
GO
```

İlk olarak master komutu ile öncelikle sistem veritabanına geçilmiştir. Çünkü SQL Server üzerinde yeni bir oturum tanımlama yalnızca master veritabanı üzerinden yapılabilir.

Ardından CREATE LOGIN komutu ile SQL Server genelinde geçerli olacak yeni bir oturum tanımlanır. Ardından, kullanıcıya yetki verilmek istenen veritabanı olan MovieLensDB seçilmiştir.

Ve son olarak daha önce master veritabanında oluşturulan login'e karşılık gelen bir veritabanı kullanıcısı oluşturulur. Böylece mertcan_omer, MovieLensDB veritabanı içinde işlem yapabilir hale gelir. Bu adımları takip ederek SQL Server Authentication yöntemi ile yeni bir kullanıcı oluşturulmuştur.

```
-- Kullanıcıya sadece 'kullanici' tablosuna erişim izni ver
GRANT SELECT ON dbo.kullanici TO [mertcan_omer];
GO

-- Kullanıcıya 'rating' tablosuna erişim izni verilmesin
DENY SELECT ON dbo.rating TO [mertcan_omer];
GO
```

Oluşturduğumuz yeni kullanıcının erişebileceği ve erişemeyeceği tabloları burada GRANT (Erişim izni vermek için) ve DENY (Erişim izni reddetmek için) komutları ile belirtildi.

Yukarıda adımlarda SQL Server Authentication ile kullanıcı oluşturma ve o kullanıcıya erişim izinleri verme veya reddetme işlemlerinin nasıl yapıldığı gösterilmiştir.

1.2. Veri Şifreleme

Bu bölümde, MovieLensDB adlı veritabanının yetkisiz erişimlere karşı korunması amacıyla Transparent Data Encryption (TDE) yöntemi uygulanmıştır. TDE, veritabanındaki tüm verilerin disk düzeyinde şifrlenmesini sağlar.

1.2.1. Master Key Oluşturma

Master veritabanında şifreli işlemler için bir ana anahtar (master key) oluşturulmuştur

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'SifreliParola123!';
```

1.2.2. Sertifika Oluşturulması

Master key ile ilişkili bir sertifika oluşturulmuştur:

```
CREATE CERTIFICATE MovieLensCert  
WITH SUBJECT = 'TDE Certificate';
```

1.2.3. Şifreleme Anahtarının Tanımlanması

Şifreleme işlemlerini gerçekleştirecek olan anahtar, hedef veritabanı olan MovieLensDB içinde tanımlanmıştır

```
CREATE DATABASE ENCRYPTION KEY  
WITH ALGORITHM = AES_256  
ENCRYPTION BY SERVER CERTIFICATE MovieLensCert;
```

1.2.4. TDE'nin Aktif Edilmesi

Veritabanı şifrelemesi aşağıdaki komut ile aktif edilmiştir

```
ALTER DATABASE MovieLensDB  
SET ENCRYPTION ON;
```

1.2.5. Kontrol Etme

Aşağıdaki komutla şifrlenip şifrlenmediği kontrol edilmiştir.

```

SELECT
    db.name,
    db.is_encrypted,
    dm.encryption_state,
    dm.percent_complete,
    dm.key_algorithm,
    dm.key_length
FROM sys.databases db
LEFT JOIN sys.dm_database_encryption_keys dm
ON db.database_id = dm.database_id;

```

100 %

Results Messages

	name	is_encrypted	encryption_state	percent_complete	key_algorithm	key_length
1	master	0	NULL	NULL	NULL	NULL
2	tempdb	1	3	0	AES	256
3	model	0	NULL	NULL	NULL	NULL
4	msdb	0	NULL	NULL	NULL	NULL
5	proje	0	NULL	NULL	NULL	NULL
6	deneme	0	NULL	NULL	NULL	NULL
7	MovieLensDB	1	3	0	AES	256

1.3. SQL Injection Testleri

SQL Injection, kötü niyetli kullanıcıların web uygulamalarındaki veri giriş alanları aracılığıyla SQL sorgularına müdahale ederek veritabanına yetkisiz erişim sağlamaya çalıştığı bir saldırı türüdür.

```

-- Bu, sorguyu değiştirebilir ve tüm veritabanındaki verileri çekebilir.
SELECT * FROM dbo.user WHERE username = '' OR 1=1; --' AND password = 'password';

```

Yukarıdaki sorguda OR 1=1 ifadesi her zaman doğru döneceği için, kimlik doğrulaması atlanarak

sistemdeki tüm kullanıcılar listelenebilir. SQL Injection'ı engellemenin en etkili yollarından biri parametrelili sorgular kullanmaktır. Bu yöntemle kullanıcıdan alınan veriler doğrudan sorguya gömülmez; veri ve sorgu mantıksal olarak ayrılır. Örneğin, C# ile ADO.NET kullanarak parametrelili bir sorgu örneği:

```
using (SqlCommand cmd = new SqlCommand("SELECT * FROM dbo.user WHERE username = @username AND password = @password", connection)){  
    cmd.Parameters.AddWithValue("@username", username);  
    cmd.Parameters.AddWithValue("@password", password);}
```

Yukarıdaki sorgu ile kullanıcının girdiği değerler @username ve @password parametrelerine atanır. Parametreler SQL sorgusundan **bağımsız** şekilde veritabanına iletilir. SQL komutuna dışardan müdahale edilemez hale gelir ve SQL Injection engellenmiş olur.

1.4. Audit Logları

SQL Server Audit özelliği, veri tabanı üzerindeki kullanıcı aktivitelerini detaylı şekilde izlemek ve kayıt altına almak için geliştirilmiştir. Burada, kullanıcı aktivitelerini izlemek için SQL Server Audit yapılandırmasının nasıl gerçekleştirileceğini adım adım açıklamaktadır. Örnek uygulama olarak MovieLens_Audit adlı bir denetim (audit) tanımlanmıştır.

```
-- Audit özelliğini açma  
CREATE SERVER AUDIT MovieLens_Audit  
TO FILE (FILEPATH = 'C:\AuditLogs\');  
GO
```

Yukarıda Audit loglarının yazılacağı dosya konumunu belirten bir audit nesnesi oluşturulmuştur. MovieLens_Audit adlı bir denetim tanımlanmıştır. Log dosyaları C:\AuditLogs\ dizinine yazılacaktır.

```
-- Audit spesifik veritabanı işlemleri ekleme  
CREATE SERVER AUDIT SPECIFICATION MovieLens_Audit_Specification  
FOR SERVER AUDIT MovieLens_Audit  
ADD (FAILED_LOGIN_GROUP),  
ADD (DATABASE_OBJECT_PERMISSION_CHANGE_GROUP),  
ADD (SCHEMA_OBJECT_ACCESS_GROUP)  
WITH (STATE = ON);  
GO
```

Yukarıda sunucu düzeyinde hangi aktivitelerin izleneceği belirlenmiştir.

FAILED_LOGIN_GROUP: Başarısız giriş denemelerini izlenmesini sağlar.

DATABASE_OBJECT_PERMISSION_CHANGE_GROUP: Veritabanı nesnelerinin izinlerinde yapılan değişikliklerin izlenmesini sağlar.

SCHEMA_OBJECT_ACCESS_GROUP: Şema nesnelere yapılan erişim işlemlerin izlenmesini sağlar.

```
-- Audit özelliğini aktif hale getirme
ALTER SERVER AUDIT MovieLens_Audit
WITH (STATE = ON);
GO
```

Yukarıda Audit nesnesi çalışır hale getirilmiştir. Artık kullanıcı aktiviteleri izlenmeye başlanmıştır.

```
SELECT *
FROM fn_get_audit_file('C:\AuditLogs\*', NULL, NULL);
GO
```

Yukarıda Audit kayıtlarını incelemek için bir sorgu oluşturulmuştur. fn_get_audit_file fonksiyonu, belirttiğiniz dizindeki tüm audit loglarını okur ve görüntüler. Bu kayıtlar, kullanıcı hareketlerini detaylı bir şekilde sunar (tarih, kullanıcı adı, işlem türü vb.).

	event_time	sequence_number	action_id	succeeded	permission_bitmask	is_column_permission	session_id	server_principal_id	database_principal_id	target_server_principal_id	target_database_principal_id
1	2025-04-22 18:28:08.2966549	1	AUSC	1	0x00000000000000000000000000000000	0	66	259	0	0	0
2	2025-04-22 18:28:25.0764294	1	LGIF	0	0x00000000000000000000000000000000	0	0	0	0	0	0
3	2025-04-22 18:28:29.1885353	1	LGIF	0	0x00000000000000000000000000000000	0	0	0	0	0	0
4	2025-04-22 18:28:31.9007933	1	LGIF	0	0x00000000000000000000000000000000	0	0	0	0	0	0
5	2025-04-22 18:28:35.5810652	1	SL	1	0x00000000000000000000000000000001	1	59	267	2	0	0
6	2025-04-22 18:28:35.5951253	1	SL	1	0x00000000000000000000000000000001	1	59	267	2	0	0
7	2025-04-22 18:28:35.5951253	1	SL	1	0x00000000000000000000000000000001	1	59	267	2	0	0
8	2025-04-22 18:28:35.6041257	1	SL	1	0x00000000000000000000000000000001	1	59	267	2	0	0
9	2025-04-22 18:28:35.6061275	1	SL	1	0x00000000000000000000000000000001	1	59	267	2	0	0
10	2025-04-22 18:28:35.6564253	1	SL	1	0x00000000000000000000000000000001	1	65	267	2	0	0
11	2025-04-22 18:28:35.6634149	1	SL	1	0x00000000000000000000000000000001	1	59	267	2	0	0
12	2025-04-22 18:28:35.7724606	1	EX	1	0x00000000000000000000000000000020	0	59	267	2	0	0
13	2025-04-22 18:28:35.7823969	1	SL	1	0x00000000000000000000000000000001	1	59	267	2	0	0
14	2025-04-22 18:28:35.8434301	1	SL	1	0x00000000000000000000000000000001	1	70	267	2	0	0

Verilen sorgu sonucu yukarıdaki gibi bir tablo çıktısı verilir.

2. Veritabanı Yedekleme ve Felaketten Kurtarma Planı

Bu bölümde, SQL Server ortamında veritabanı yedekleme ve felaketten kurtarma planlarının nasıl oluşturulacağına bahsedilmektedir. Yedekleme stratejileri, otomatikleştirme, felaket senaryoları ve test süreçleri ele alınmıştır. Çalışmalarda MovieLens 100K veri seti kullanılmıştır.

2.1. Yedekleme Stratejileri

2.1.1. Tam Yedekleme

Veritabanının tamamı yedeklenir. Komutu şu şekildedir.

```
BACKUP DATABASE MovieLensDB  
TO DISK = 'C:\Backups\movielens_full.bak';
```

2.1.2. Fark Yedekleme

Son tam yedekten sonra değişen veriler yedeklenir. Komutu şu şekildedir.

```
BACKUP DATABASE MovieLensDB  
TO DISK = 'C:\Backups\movielens_diff.bak'  
WITH DIFFERENTIAL;
```

2.1.3. Artımlı Yedekleme

Veritabanında yapılan her değişikliği içeren log dosyaları yedeklenir.

```
BACKUP LOG MovieLensDB  
TO DISK = 'C:\Backups\movielens_log.trn';
```

2.2. Zamanlayıcılarla Yedekleme

Zamanlayıcılar ile yedekleme yapmak için SQL Server Agent üzerinden Job oluşturulabilir. Job oluşturmak için SQL Server Agent > New > Job seçilir. Açılan pencerenin sol üst kısmından steps seçilir. New butonuna basılıp tekrar açılan pencereden step için isim verilir, Database ve type olarak Transact-SQL script seçilir. Son olarak command girilir ve OK butonuna basılır.

New Job Step

Select a page

- General
- Advanced

Script Help

Step name: fullbackup

Type: Transact-SQL script (T-SQL)

Run as:

Database: MovieLensDB

Command:

```
BACKUP DATABASE MovieLensDB  
TO DISK = 'C:\Backups\movielens_auto_full.bak'  
WITH INIT;
```

Open... Select All Copy Paste Parse

Previous Next

OK Cancel

Connection

Server: LAPTOP-ILM2R7E6

Connection: LAPTOP-ILM2R7E6\Mertcan

[View connection properties](#)

Progress

Ready

Step tamamlandıktan sonra zamana bağlamak için schedule seçilir. Burada kaç günde veya hangi saat aralığında step'i çalıştıracağı seçilir.

New Job Schedule

Name: Jobs in Schedule

Schedule type: Recurring ☒ Enabled

One-time occurrence

Date: 24.04.2025 Time: 01:04:10

Frequency

Occurs: Weekly

Recurs every: 1 week(s) on

☐ Monday ☐ Wednesday ☐ Friday ☐ Saturday
☐ Tuesday ☐ Thursday ☒ Sunday

Daily frequency

☒ Occurs once at: 00:00:00
☐ Occurs every: 1 hour(s) Starting at: 00:00:00 Ending at: 23:59:59

Duration

Start date: 24.04.2025 ☐ End date: 24.04.2025 ☒ No end date

Summary

Description: Occurs every week on Sunday at 00:00:00. Schedule will be used starting on 24.04.2025.

OK Cancel Help

2.3. Felaketten Kurtarma Senaryoları

Veritabanında bir tablo veya kendisi silinirse yedekler ile geri döndürülebilir.

```
USE master;  
ALTER DATABASE MovieLensDB SET SINGLE_USER WITH ROLLBACK IMMEDIATE;  
  
RESTORE DATABASE MovieLensDB  
FROM DISK = 'C:\Backups\movielens_full.bak'  
WITH REPLACE;  
  
ALTER DATABASE MovieLensDB SET MULTI_USER;
```

Veritabanı yedeğinin geri yüklenmesi sürecinde öncelikle sistem veritabanı olan master veritabanına geçilmiştir, çünkü geri yükleme işlemleri sırasında hedef veritabanı kullanımda olamaz. Bu işlemten sonra, ALTER DATABASE MovieLensDB SET SINGLE_USER WITH ROLLBACK IMMEDIATE komutu ile MovieLensDB veritabanı

tek kullanıcı moduna alınmış ve o an bağlı olan tüm kullanıcıların işlemleri iptal edilerek bağlantıları sonlandırılmıştır. Ardından, RESTORE DATABASE MovieLensDB FROM DISK = 'C:\Backups\movielens_full.bak' WITH REPLACE komutu ile belirtilen .bak dosyasından veritabanı geri yüklenmiştir. WITH REPLACE ifadesi, mevcut veri tabanının üzerine yazılmasına olanak tanımaktadır. Son olarak, ALTER DATABASE MovieLensDB SET MULTI_USER komutu ile veritabanı tekrar çok kullanıcı moduna geçirilmiş ve normal erişime açılmıştır. Bu işlemler sayesinde veri tabanı eski haline geri döndürülmüştür.

2.4. Test Yedekleme Senaryoları

Yedeklemelerin doğruluğunu kontrol etmek için sorgu ile bir veri sildik ve yedekleri geri yükledikten sonra silinen veri geri geldi mi kontrol edildi.

Veri tabanımızda select * FROM kullanıcı WHERE user_id = 196 AND item_id = 242; sorgusunu çalıştırdığımızda user_id = 196 ve item_id = 242 denk olduğu bilgileri siler.

```
select * from dbo.kullanici WHERE user_id = 196 AND item_id = 242;
```

	user_id	item_id	rating	timestamp
1	196	242	3	881250949

Sorgusu bize böyle bir çıktı veriyor. DELETE FROM kullanıcı WHERE user_id = 196 AND item_id = 242; sorgusunu çalıştırsak bunu silecektir. Bu satırı geri getirmek için yukarıda anlatıldığı gibi veri tabanı yedeği yüklenir ve aynı sorgu tekrar çalıştırıldığında user_id = 196 ve item_id = 242 çıktısının sonucu geri geldiği görülür.

3. Veritabanı Yedekleme ve Otomasyon Çalışması

3.1. SQL Server Agent ile yedekleme süreçlerini otomatikleştirme

SQL Server Agent kullanarak bir job ataması ile zamana dayalı otomatik yedekleme işlemi yapılmıştır. Bölüm 2.2.'de daha ayrıntılı olarak anlatılmıştır. Özet olarak SQL Server Agent'a yeni bir job atanır, çalışma sıklığı belirlenir (projemizde günlük olarak çalışacak şekilde atanmıştır) ve aşağıdaki şekildeki gibi yapacağı iş belirlenir (yedekleme).

```
BACKUP DATABASE MovieLensDB  
TO DISK = 'C:\Backups\movielens_auto_full.bak'  
WITH INIT;
```

3.2. T-SQL Scripting ile yedekleme raporları oluşturma

```
SELECT
    database_name,
    backup_start_date,
    backup_finish_date,
    type AS backup_type,
    physical_device_name
FROM msdb.dbo.backupset b
JOIN msdb.dbo.backupmediafamily m ON b.media_set_id = m.media_set_id
WHERE database_name = 'MovieLensDB'
ORDER BY backup_finish_date DESC;
```

Yukarıdaki sorgu sayesinde geçmiş yedekleme kayıtlarını listelenir. Bu sorgu ile hangi veritabanının, ne zaman yedeklendiği ve nereye kaydedildiği aşağıdaki şekildeki gibi detaylı olarak listelenir.

	database_name	backup_start_date	backup_finish_date	backup_type	physical_device_name
1	MovieLensDB	2025-04-23 17:26:12.000	2025-04-23 17:26:12.000	D	C:\Backups\movielens_auto_full.bak
2	MovieLensDB	2025-04-23 17:18:17.000	2025-04-23 17:18:17.000	L	C:\Backups\movielens_log.trn
3	MovieLensDB	2025-04-23 17:17:07.000	2025-04-23 17:17:07.000	I	C:\Backups\movielens_diff.bak
4	MovieLensDB	2025-04-23 17:16:17.000	2025-04-23 17:16:17.000	D	C:\Backups\movielens_full.bak

3.3. Otomatik Yedekleme Uyarıları

Yedekleme sırasında meydana gelen hataları kullanıcıya mail ile bildirmek otomasyon için çok önemlidir. Aşağıda bu işlemin nasıl yapıldığı açıklanmıştır.

İlk olarak bir mail profili oluşturulur ve SMTP sunucusu, port, kullanıcı adı/şifre gibi ayarları yapılır ve bir test maili göndererek doğruluğu kontrol edilir.

İlk aşamadan sonra SQL Server Agent'ın, Database Mail profilini kullanabilmesi için bunu tanıması gerekir. Tanımlama işlemini SQL Server Agent Alert System'dan aşağıdaki şekilde yapılır.

Mail session

☒ Enable mail profile

Mail system: Database Mail

Mail profile: enyeni

☒ Save copies of the sent messages in the Sent Items folder

Test...

Pager e-mails

Address formatting for pager e-mails:

	Prefix:	Pager:	Suffix:
To line:		<input type="checkbox"/>	
Cc line:		<input type="checkbox"/>	
Subject:			

To:
Cc:
Subject: <Subject>

☒ Include body of e-mail in notification message

Ardından bildirim gönderilecek kişi ya da sistem (e-posta) adresi aşağıdaki gibi operatör olarak tanımlanır.

Name: operatoryedek ☒ Enabled

Notification options

E-mail name: ghostroachtr@gmail.com

Pager e-mail name: ghostroachtr@gmail.com

En son olarak yedekleme job'u başarısız olursa hangi operatöre, ne zaman bildirim gideceğini Job Notifications ile aşağıdaki gibi tanımlanır.

Actions to perform when the job completes:

<input checked="" type="checkbox"/> E-mail:	operatoryedek	When the job fails
<input type="checkbox"/> Page:		When the job fails
<input type="checkbox"/> Write to the Windows Application event log:		When the job fails
<input type="checkbox"/> Automatically delete job:		When the job succeeds

Yaptığımız işlemler sonucu job'un yedekleme hatası alması durumunda kullanıcıya mail gönderiyor mu test etmek için job'u kasıtlı olarak hata alacak şekilde çalıştırdık. Yaptıklarımız sonucunda mail adresimize Yedekleme başarısız maili aşağıdaki gibi gelmiş oldu.

[The job failed.] SQL Server Job System: 'FullBackupStep' completed on \\LAPTOP-ILM2R7E6. Gelen Kutusu x

ghostroachtr <ghostroachtr@gmail.com>
Alıcı: ben

JOB RUN: 'FullBackupStep' was run on 23.04.2025 at 18:46:21
DURATION: 0 hours, 0 minutes, 1 seconds
STATUS: Failed
MESSAGES: The job failed. The Job was invoked by User LAPTOP-ILM2R7E6\\Mertcan. The last step to run was step 1 (FullBackupStep).

4. VERİ TABANI PERFORMANS OPTİMİZASYONU VE İZLEME

Bu bölümde MovieLensDB üzerinde gerçekleştirilen performans optimizasyonu ve izleme uygulamalarını kapsamaktadır. İzleme altyapısının etkinleştirilmesi, dinamik yönetim görünümleri ile anlık performans analizi, indeks yönetimi ve sorgu iyileştirmesi konularına odaklanılmıştır.

4.1. İzleme Altyapısını Etkinleştirme

4.1.1. Query Store

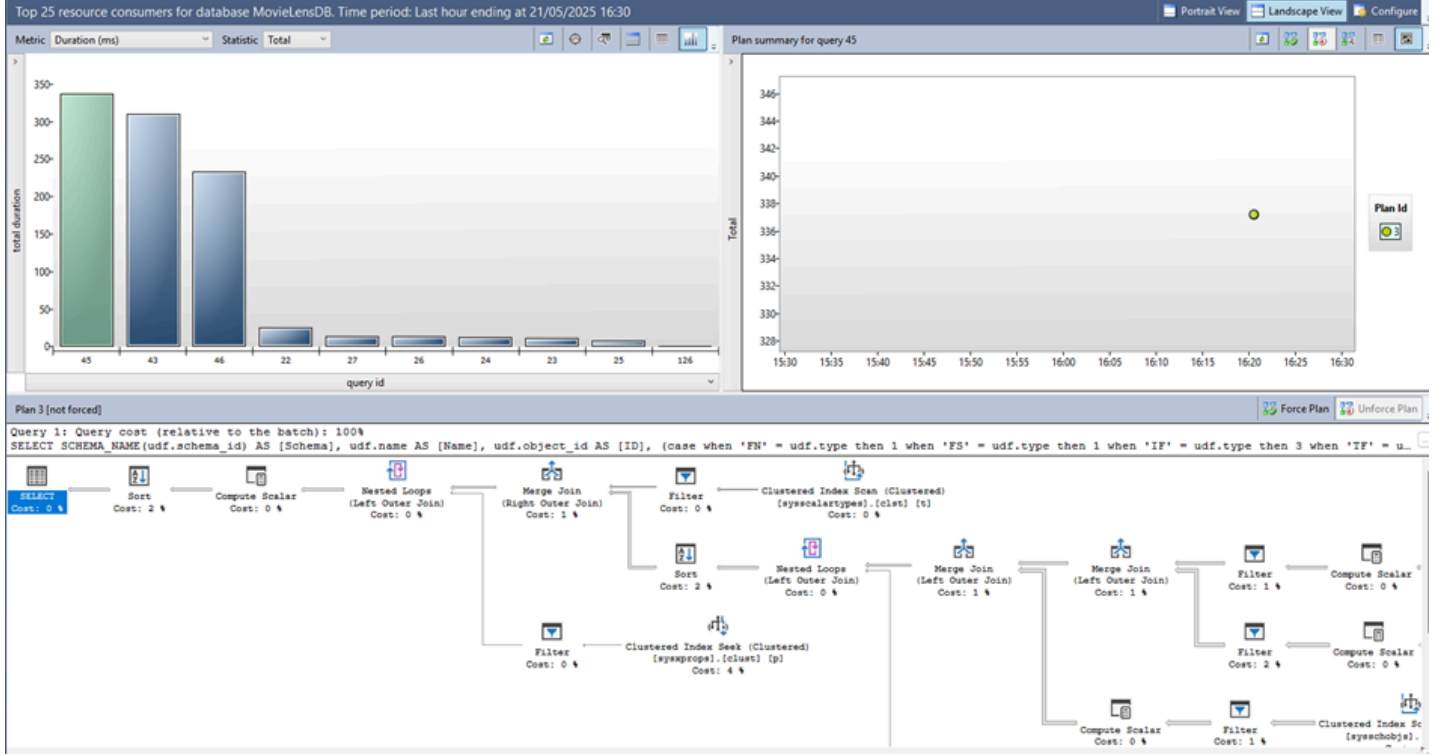
SSMS üzerinde “Query Store – Top Resource Consuming Queries” raporu ile CPU, I/O veya süre bazında en pahalı 25 sorgu listelenir.

```
ALTER DATABASE MovieLensDB SET QUERY_STORE = ON;
```

Messages

Commands completed successfully.

Completion time: 2025-05-21T16:20:30.3471914+03:00



4.1.2. SQL Profiler

Gerçek zamanlı olarak SQL Server'a gönderilen tüm sorgu yürütme olaylarını yakalar. Özellikle uzun süren ve yüksek kaynak tüketimli sorguları tespit ederek performans darboğazlarını ortaya koyar. Ortaya çıkan veriler, hangi sorguların ne kadar CPU, I/O ve süre harcadığını gösterir.

Sırası ile Profiler başlatılır, şablon seçimi gerçekleştirilir(TSQL_Duration), filtre uygulanır (Duration > 1000 ms), trace çalıştırılır ve son olarak sonuç kaydedilir.

Trace Properties

General | Events Selection

Trace name: Untitled - 6

Trace provider name: LAPTOP-ILM2R7E6

Trace provider type: Microsoft SQL Server "2022" version: 16.0.1000

Use the template: TSQL_Duration

☐ Save to file: [] Set maximum file size (MB): 5 ☒ Enable file rollover ☐ Server processes trace data

☐ Save to table: [] Set maximum rows (in thousands): 1

☐ Enable trace stop time: 21/05/2025 17:41:30 ☒ Set trace duration (in minutes): 60

Run | İptal | Yardım

EventClass	Duration	TextData	SPID	BinaryData
RPC:Completed	1	exec sp_trace_setstatus 3,1	53	0x00000...
RPC:Completed	2	exec sp_executesql N'SELECT c1mns.na...	55	0x00000...
RPC:Completed	2	exec sp_executesql N'SELECT c1mns.na...	55	0x00000...
RPC:Completed	3	exec sp_executesql N'SELECT c1mns.na...	55	0x00000...
RPC:Completed	218	exec sp_executesql N'SELECT c1mns.co...	55	0x00000...
RPC:Completed	492	exec sp_executesql N'SELECT c1mns.na...	55	0x00000...
SQL:BatchCompleted	0	SET DEADLOCK_PRIORITY -10	69	
SQL:BatchCompleted	0	SET DEADLOCK_PRIORITY -10	69	
SQL:BatchCompleted	0	use [MovieLensDB]	55	
SQL:BatchCompleted	0	use [MovieLensDB]	55	
SQL:BatchCompleted	0	use [MovieLensDB]	55	
SQL:BatchCompleted	0	use [MovieLensDB]	55	
SQL:BatchCompleted	0	use [MovieLensDB]	55	
SQL:BatchCompleted	2	SELECT dtb.name AS [Name], CAST(0 AS...	55	
SQL:BatchCompleted	3	if not exists (select * from sys.dm...	69	
SQL:BatchCompleted	87	SELECT target_data FROM sy...	69	
SQL:BatchCompleted	301	select * from dbo.ratings	65	
Trace Start				

Profiler arayüzünde CPU-, I/O- veya süreye göre en maliyetli ilk 25 sorguyu tablolar halinde görüntülenir. Hangi sorguların darboğaza yol açtığını hızlıca belirlemek ve yoğun sorgu anlarında sistem kaynaklarının nasıl tükendiğini görmek için kullanılır.

4.2. DMV'lerle Anlık Performans Analizi

Veritabanında eksik indekslerin hangi tablolar veya sorgular için önerildiğini tespit eder. Her bir önerinin kullanıcı üzerindeki ortalama maliyet (avg_total_user_cost) ve getirdiği performans kazancını (avg_user_impact) sayısal olarak sunar. Böylece hangi indeksi eklemenin en faydalı olacağı kolayca anlaşılır.

```
SELECT
    d.*,
    s.avg_total_user_cost,
    s.avg_user_impact
FROM sys.dm_db_missing_index_details d
JOIN sys.dm_db_missing_index_groups g
    ON d.index_handle = g.index_handle
JOIN sys.dm_db_missing_index_group_stats s
    ON g.index_group_handle = s.group_handle
ORDER BY s.avg_user_impact DESC;
```

index_handle	database_id	object_id	equality_columns	inequality_columns	included_columns	statement	avg_total_user_cost	avg_user_impact
--------------	-------------	-----------	------------------	--------------------	------------------	-----------	---------------------	-----------------

Veri tabanımızda eksik indeks olmadığı için bu sorgunun bize verdiği tablo boştur.

sys.dm_db_missing_index_details (d): Önerilen eksik indeksin sütun bilgilerini ve hedef tabloyu döner.

sys.dm_db_missing_index_groups (g): İlgili eksik indeks önerilerini gruplar.

sys.dm_db_missing_index_group_stats (s): Her grup için ortalama maliyet ve kazanç metriklerini içerir. avg_user_impact'e göre azalan sırada çalıştırılır; böylece en yüksek potansiyel kazanç önce listelenir.

4.2.1. Index Kullanım Analizi

```
SELECT
    OBJECT_NAME(i.object_id) AS TableName,
    i.name AS IndexName,
    u.*
FROM sys.indexes i
LEFT JOIN sys.dm_db_index_usage_stats u
    ON u.index_id = i.index_id
    AND u.object_id = i.object_id
WHERE OBJECTPROPERTY(i.object_id, 'IsUserTable') = 1
ORDER BY
    COALESCE(u.user_seeks, 0)
    + COALESCE(u.user_scans, 0)
    + COALESCE(u.user_lookups, 0);
```

	TableName	IndexName	database_id	object_id	index_id	user_seeks	user_scans	user_lookups	user_updates	last_user_seek	last_user_scan	last_user_lookup	last_user_update	system_seeks	system_scans	system_lookups	system_updates
1	users	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	DatabaseVersion	PK_Database__16C6402F13EF2FBD	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	ratings	NULL	7	901579250	0	0	4	0	0	NULL	2025-05-21 16:43:24.657	NULL	NULL	0	0	0	0

Yukarıdaki sorgu ile hiç kullanılmayan indeks'leri tespit edilebilir, az kullanılan ama yazma işlemlerini yavaşlatan indeks'leri tespit edilebilir, sık kullanılan indeks'lerin önemini anlaşılabılır. Gereksiz indeks'leri kaldırarak disk alanı ve yazma performansı kazancı sağlanabilir.

sys.indexes tablosundan (i) tüm kullanıcı tanımlı tabloların (IsUserTable = 1) index bilgilerini alır. sys.dm_db_index_usage_stats görünümünden (u), o index'in ne kadar "arama (seek)", "tarama (scan)" veya "arama/look-up" işlemine maruz kaldığını getirir.

LEFT JOIN ile, hiç kullanılmamış (usage_stats kaydı olmayan) index'ler de sonuçta gözükür.

Böylece "kullanılmayan" index'leri tespit etmek mümkün olur.

Son olarak, ORDER BY ile en az kullanılan index'ten en çok kullanılan index'e doğru sıralama yapar.

4.2.2. Veritabanındaki İndexlerin Fiziksel Fragmentasyon Oranları

```

SELECT
    OBJECT_NAME(object_id) AS TableName,
    index_type_desc,
    avg_fragmentation_in_percent,
    page_count
FROM sys.dm_db_index_physical_stats(
    DB_ID(),
    NULL,
    NULL,
    NULL,
    'LIMITED'
)
WHERE page_count > 1000;

```

TableName	index_type_desc	avg_fragmentation_in_percent	page_count
-----------	-----------------	------------------------------	------------

sys.dm_db_index_physical_stats dinamik yönetim fonksiyonunu kullanarak veritabanındaki indexlerin fiziksel fragmentasyon oranlarını ve sayfa sayılarını analiz etmektedir.

Performans iyileştirme için kritik seviyedeki indexleri belirlemeyi, disk I/O maliyetini düşürmeyi, gereksiz bakım işlemlerini önleyerek kaynak tasarrufu sağlamayı amaçlar.

Sorgu Bileşenleri:

sys.dm_db_index_physical_stats: Veritabanındaki indexlerin fiziksel fragmentasyonunu ve sayfa bilgilerini getirir.

DB_ID(): Mevcut veritabanı.

'LIMITED': Üst seviye index sayfaları analiz edilir.

Filtre: page_count > 1000

Çıktı Kolonları

TableName: Indexin ait olduğu tablo.

index_type_desc: Index tipi (Clustered, Nonclustered vb.).

avg_fragmentation_in_percent: Fragmentasyon oranı (%).

page_count: Indexin kapladığı sayfa sayısı.

4.3. İndeks Yönetimi

4.3.1. Yeni İndeks Örneği

```

CREATE NONCLUSTERED INDEX IX_ratings_movie_user
ON dbo.ratings(item_id, user_id)
INCLUDE (rating, timestamp);

```

Commands completed successfully.

Completion time: 2025-05-21T16:55:30.7394961+03:00

Bu sorgu, CREATE NONCLUSTERED INDEX komutu ile, belirtilen sütunlar üzerinde ayrı bir index yapısı oluşturur. Ek sütunlar INCLUDE ifadesiyle index'e eklenerek, sorgu sırasında bu verilerin doğrudan index'ten okunması sağlanır. Bu sayede ratings tablosunda item_id (film) ve user_id (kullanıcı) sütunlarına göre aramaları hızlandırmak amaçlanır. Tabloya tam tarama (table scan) yerine yalnızca index okunması sayesinde, özellikle “belirli bir film için ortalama puan” gibi sorgular çok daha hızlı çalışır. Büyük veri tabanlarında sorgu performansını artırmak, I/O maliyetini düşürmek ve raporlama sürelerini kısaltmak için kullanılır.

4.3.2. Parçalanmış İndekslerin Otomatik Bakımı

```
ALTER INDEX ALL ON dbo.ratings  
REBUILD WITH (ONLINE = ON, SORT_IN_TEMPDB = ON);
```

Bu sorgu, ALTER INDEX ... REBUILD kullanılarak; ONLINE = ON ile işlem sırasında tabloya erişim kesilmez, SORT_IN_TEMPDB = ON ile sıralama yükü geçici veritabanına (tempdb) aktararak ratings tablosundaki tüm index'leri yeniden oluşturur (rebuild), böylece parçalanmış (fragmented) yapıyı temizler. Sürekli veri ekleme/güncelleme sonucu bozulan index yapısını düzeltir, sorgu performansının stabil kalmasını sağlar ve bakım süresini kısaltır. Üretim ortamlarında, bakım penceresi kısıtlıyken veritabanını çevrimdışı yapmadan indeksleri optimize etmek için kullanılır.

4.3.3. Gereksiz İndeks Kaldırma

```
DROP INDEX dbo.ratings.IX_ratings_movie_user;
```

Bu sorgu, DROP INDEX komutu ile gereksiz veya artık kullanılmayan index yapısı veritabanından uzaklaştırarak belirtilen index'i siler. Disk alanı kazanımı, index bakım (rebuild/reorganize) maliyetinin düşmesi ve yazma işlemlerinin (INSERT/UPDATE/DELETE) daha hızlı gerçekleşmesini sağlar. Kullanılmayan index'ler hem depolama hem de performans üzerinde olumsuz etki yapar; düzenli temizlikle kaynaklar daha verimli kullanılması amaçlanır.

4.3.4. Sorgu İyileştirme Örneği

Sorguya WITH (INDEX = ...) ipucu eklenerek, SQL Server'ın önceden oluşturulmuş index'i (IX_ratings_movie_user) kullanması sağlayarak belirli bir film (movieId) için ortalama puanı hesaplar. SQL optimizasyon motorunun yanlış index veya tablo taraması seçme riskini ortadan kaldırır ve sorgu süresini düşürür. Kritik raporlama veya ad-hoc sorgularda, en iyi performansı

kesin olarak elde etmek istendiğinde kullanılır.

```
SELECT AVG(rating)
FROM dbo.ratings
WHERE movieId = 1;
```

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2025-05-21T17:11:19.7801702+03:00

Öncesi

```
SELECT AVG(rating)
FROM dbo.ratings
WITH (INDEX = IX_ratings_movie_user)
WHERE item_id = 1;
```

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

Sonrası

4.3.5. Rol ve Erişim Yönetimi

Oturum açma ve Kullanıcı Oluşturma:

```
CREATE LOGIN aaaa WITH PASSWORD = 'password';
USE MovieLensDB;
GO
CREATE USER aaaa FOR LOGIN aaaa;
EXEC sp_addrolemember 'db_datareader', 'aaaa';
```

Bu sorgu, CREATE LOGIN ile SQL Server seviyesinde bir giriş hesabı oluşturur. CREATE USER ile Belirli bir veritabanı için bu login'i kullanıcıya eşler. sp_addrolemember ile Kullanıcıyı db_datareader rolüne ekleyerek okuma izni verir. Önce sunucu seviyesinde kimlik doğrulaması ayarlanır, sonra veritabanı bağlamında ilgili roller atanır. Ayrıcalık yönetimi, yetkisiz erişimlerin önlenmesi ve en iyi güvenlik uygulamalarının sağlar. Çok kullanıcıli ortamlarda, her kullanıcıya ihtiyaç duyduğu minimum izinlerin verilmesi için kullanılır.

Kullanıcı Bilgisi Sorgulama:

```

SELECT
    name AS UserName,
    type_desc AS UserType,
    authentication_type_desc AS AuthType
FROM sys.database_principals
WHERE name = 'aaaa';

```

	UserName	UserType	AuthType
1	aaaa	SQL_USER	INSTANCE

Bu sorgu, veritabanı içindeki belirli bir kullanıcı hakkında ad, tip ve kimlik doğrulama yöntemi bilgilerini döndürür. Kullanıcı yapılandırmasının doğrulanması, yetki denetimi ve denetim süreçlerinde şeffaflık kazandırır. Güvenlik incelemeleri, denetim kayıtları ve sorun gidermede bilgi toplamak için kullanılır.

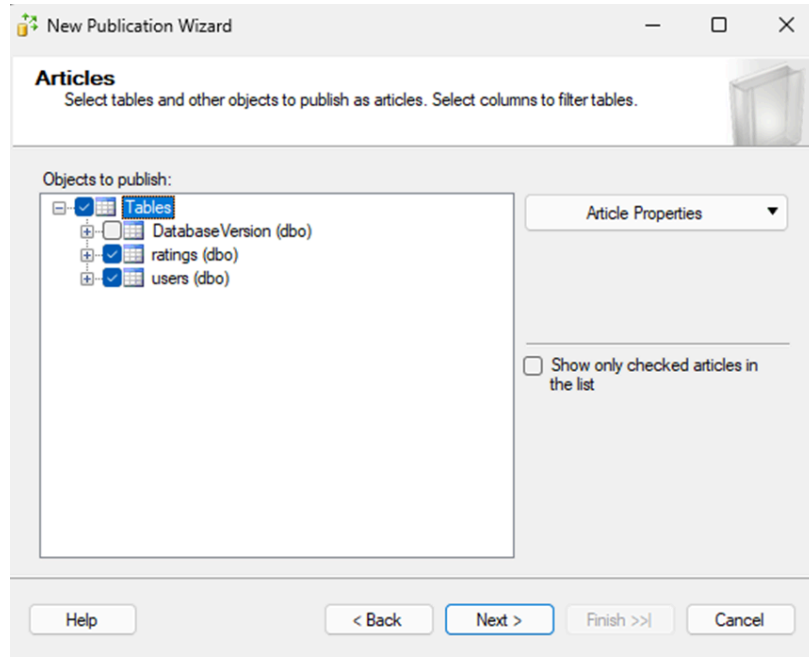
5. VERİTABANI YÜK DENGELEME VE DAĞITIK VERİTABANI YAPILARI

Bu bölüm, iki temel yüksek erişilebilirlik ve yük dengeleme yöntemi olan **Replication (Çoğaltma)** ve **Database Mirroring (Veritabanı Aynalama/Fırlatma)** yöntemlerini açıklamaktadır.

5.1. Snapshot / Transactional Replication Kurulumu

Replication, bir veritabanındaki değişikliklerin belirli periyotlarla veya anlık olarak başka bir/başka sunucudaki veritabanına çoğaltılmasını sağlar. **Snapshot Replication:** Belirlenen zamanlamada tüm tablo verisinin kopyasını alır ve abonelere gönderir. **Transactional Replication:** Kaynak veritabanında yapılan her değişikliği (INSERT/UPDATE/DELETE) yakalayarak anlık ya da çok kısa süreli gecikmeyle abonelere aktarır.

Sırası ile Distributor'ü Yapılandırma, Publication Oluşturma ve Subscription Oluşturma adımlarından oluşur.



Distributor'ü Yapılandırma: SSMS'da Replication ► Configure Distribution seçilerek, kendi makinemiz dağıtıcı (distributor) olarak ayarlanır. Snapshot dosyalarının saklanacağı klasör belirtilir (ör. C:\Replication\Snapshot).

Publication Oluşturma: SSMS'da Local Publications ► New Publication seçilir, Kaynak veritabanı: *MovieLensDB* seçilir. Publication Type: *Snapshot/Transactional*. Articles ekranında çoğaltılacak tabloları seçilir. Snapshot Agent'i "Run immediately" modunda çalıştıracak şekilde zamanlanır. Güvenlik ayarı olarak SQL Server Agent servisi hesabı kullanılabilir. Publication'a örnek isim: ML_SnapshotPub.

Subscription Oluşturma: SSMS'da Local Subscriptions ► New Subscription seçilir. Publisher ve Publication seçildikten sonra Subscriber olarak aynı instance ve yeni bir veritabanı (MovieLensReplica) belirlenir. Snapshot için tek seferlik, transactional replication için ise sürekli zamanlanmış (örneğin dakikada bir) çalıştırma seçilebilir.

5.1.1.1. Replikasyon Testi

Aşağıdaki sorgu ile Publisher'da örnek veri eklenir.

```
INSERT INTO dbo.users (user_id, age, gender, occupation, zip code)
VALUES (1001, 27, 'M', 'engineer', '12345');
```

Aşağıdaki sorgu ile Subscriber'da(MovieLensReplica) eklenen kaydı sorgulayarak çoğaltmanın başarılı olduğunu doğrulanır.

```
-- Publisher'da (MovieLensDB)
INSERT dbo.users(user_id, age, gender, occupation, zip_code)
VALUES (1001, 27, 'M', 'engineer', '12345');

-- Snapshot ise Distribution Agent'ı elle başlat
-- Transactional ise birkaç saniye içinde aktarılır

-- Subscriber'da (MovieLensReplica)
SELECT * FROM dbo.users WHERE user_id = 1001;
```

%

Results Messages

user_id	age	gender	occupation	zip_code
1001	27	M	engineer	12345

Okuma işlemlerini replikalar üzerinden yönlendirerek kaynak veritabanı üzerindeki baskıyı azaltmak, farklı coğrafi konumlardaki sunucular arasında veri tutarlılığı sağlamak, veri kaybı riskini minimize etmek ve felaket kurtarma senaryoları oluşturmak amaçlanır.

5.2. Database Mirroring ile Yük Devri (Failover)

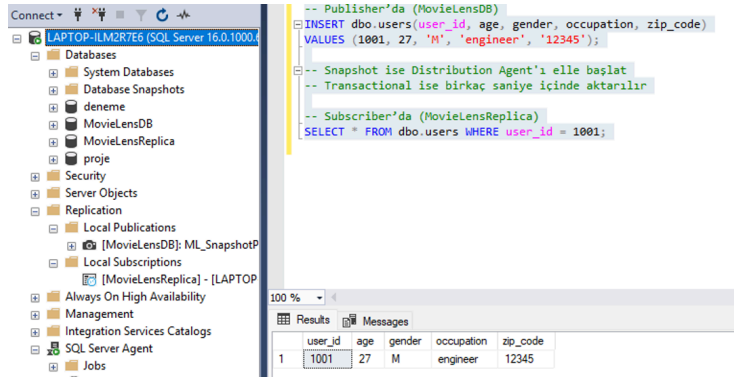
Database Mirroring, Microsoft SQL Server’da yüksek erişilebilirlik ve felaket kurtarma (disaster recovery) senaryoları için kullanılan bir özelliktir. Temelde bir veritabanının “eşlenmiş” (mirrored) bir kopyasını iki ayrı sunucu arasında tutarak; birincil (principal) sunucuya erişim kesildiğinde, aynadaki (mirror) sunucunun kısa sürede devreye girmesini sağlar.

Principal (Birincil) Sunucu: Üretim trafiğini yöneten, üzerinde okuma/yazma işlemlerinin yapıldığı veritabanının asıl kopyası.

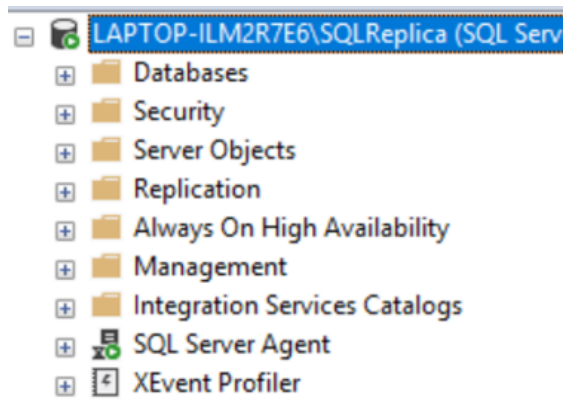
Mirror (Ayna) Sunucu: Principal’daki işlemleri eş zamanlı veya gecikmeli olarak replay eden, normalde kullanıcı trafiğini almayıp “beklemede” duran kopya.

Donanım arızası, bakım zamanı veya plan dışı kesintilerde uygulamanın erişilebilirliğini artırmak ve coğrafi olarak farklı lokasyonlardaki veri merkezleri arasında kritik verinin kopyalanması sağlamak amaçları ile kullanılır.

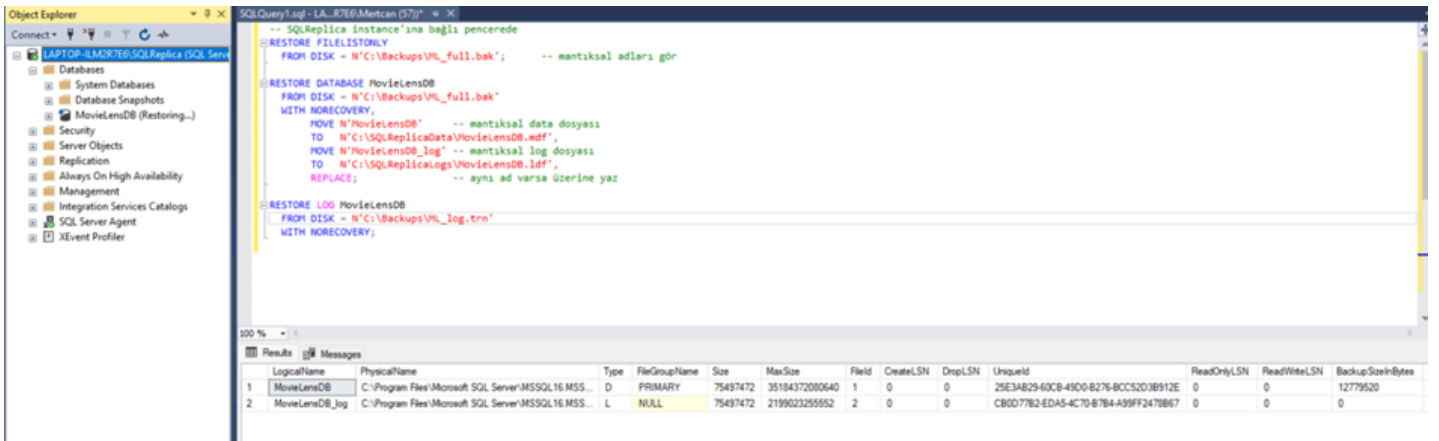
Aşağıdaki görselde oluşturduğumuz principal database yapısı bulunmakta.



Aşağıdaki görselde oluşturduğumuz mirror database yapısı bulunmaktadır.



Aşağıdaki sorgu ile principal database'den mirror database'e yedek alınıp aktarılmıştır.



Her iki sunucuda da birbirlerinin bağlantı noktalarını (7022/7023) tanımlayan birer endpoint oluşturulur. Aşağıdaki görselde sırası ile principal ve mirror database'ler için endpointler oluşturulmuştur.


```
CREATE ENDPOINT Endpoint_Mirroring  
STATE=STARTED AS TCP (LISTENER_PORT = 7022)  
FOR DATABASE_MIRRORING (ROLE=ALL);
```

```
CREATE ENDPOINT Endpoint_Mirroring  
STATE=STARTED AS TCP (LISTENER_PORT = 7023)  
FOR DATABASE_MIRRORING (ROLE=ALL);
```

Bu adımlardan sonra sırası ile hem principal hem de mirror databasesi için aşağıdaki sorgulardaki gibi oturumlar açılır. Her iki taraf da karşısındakini “partner” olarak tanımlıyor ve aralarındaki bağlantı kuruluyor.

```
ALTER DATABASE MovieLensDB  
SET PARTNER = 'TCP://LAPTOP-ILM2R7E6:7023';  
GO
```

```
ALTER DATABASE MovieLensDB  
SET PARTNER = 'TCP://LAPTOP-ILM2R7E6:7022';  
GO
```

Son olarak principal ve mirror rolleri değiştirildikten sonra rollerin gerçekten değişip değişmediği aşağıdaki sorgu ile kontrol edilir. Aşağıdaki sorgu ile eski principal (yeni mirror) databasemizin yeni principal (eski mirror) database’i ile rolleri değiştirdiğini gözlemledik.

```
ALTER DATABASE MovieLensDB SET PARTNER FAILOVER;
```

Messages

Msg 1470, Level 16, State 1, Line 1

The alter database for this partner config values may only be initiated on the current principal server for database "MovieLensDB".

6. VERİ TEMİZLEME VE ETL SÜREÇLERİ TASARIMI

Bu projede, büyük ve heterojen veri setlerinin temizlenmesi ve hedef veritabanına yüklenmesi amacıyla bir ETL (Extract, Transform, Load) süreci tasarlanıp uygulanmıştır. Amaç, veri hatalarını tespit etmek, veri uyumsuzluklarını gidermek ve güvenilir analizlere uygun temiz bir veri tabanı elde etmektir.

6.1. Staging (Geçici) Tabloların Oluşturulması

Öncelikle temizleme ve dönüştürme adımları için geçici bir şema ve bu şemadaki tablolar

oluşturulur:

```
-- 1. Staging şeması oluştur
CREATE SCHEMA Stg;
```

```
-- 2. Staging tabloları
SELECT *
INTO Stg.Users_Stg
FROM dbo.Users;

SELECT *
INTO Stg.Ratings_Stg
FROM dbo.Rating;
```

6.2. Veri Temizleme

Staging tablolarına aktarılan ham veride yer alan eksik, tutarsız veya hatalı kayıtlar işleme alınmadan önce tespit edilip ortamdan temizlenir. Bu adım, sonraki dönüştürme ve analiz süreçlerinde güvenilir sonuçlar elde edilmesini sağlar.

Staging üzerindeki ham veride basit kurallarla hatalı kayıtlar ayıklanır:

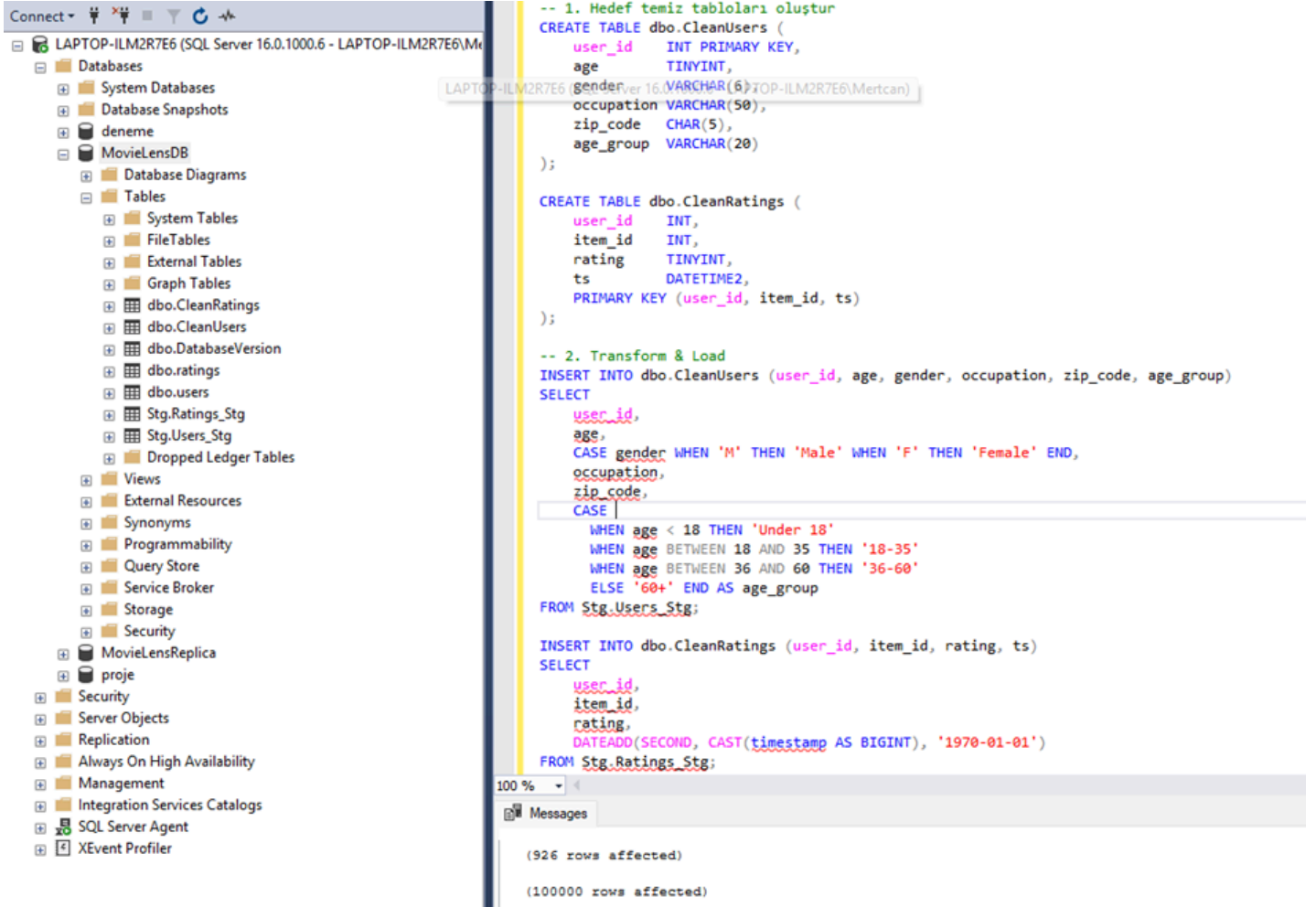
```
-- Örnek hata kontrolleri
-- Users: yaş eksik/olumsuz, cinsiyet kodu geçersiz, zip_code formatı hatalı
DELETE FROM Stg.Users_Stg
WHERE age NOT BETWEEN 1 AND 120
OR gender NOT IN ('M', 'F')
OR zip_code NOT LIKE '[0-9][0-9][0-9][0-9][0-9]';

-- Ratings: rating değeri 1-5 arası değilse veya timestamp NULL
DELETE FROM Stg.Ratings_Stg
WHERE rating NOT BETWEEN 1 AND 5
OR timestamp IS NULL;
```

6.3. Dönüştürme ve Yükleme İşlemleri

Bu aşamada, temizlenmiş veriler istenen formata dönüştürülerek hedef tablolara yüklenir. Cinsiyet kodları ve zaman damgaları (timestamp) açıklayıcı değerlere çevrilir, yaş grupları eklenir ve verinin

analize hazır hale gelmesi sağlanır.



The screenshot displays the SQL Server Enterprise Manager interface on the left, showing the database structure of 'MovieLensDB'. The right pane shows the SQL Query Editor with the following scripts:

```
-- 1. Hedef temiz tabloları oluştur
CREATE TABLE dbo.CleanUsers (
    user_id INT PRIMARY KEY,
    age TINYINT,
    gender VARCHAR(6),
    occupation VARCHAR(50),
    zip_code CHAR(5),
    age_group VARCHAR(20)
);

CREATE TABLE dbo.CleanRatings (
    user_id INT,
    item_id INT,
    rating TINYINT,
    ts DATETIME2,
    PRIMARY KEY (user_id, item_id, ts)
);

-- 2. Transform & Load
INSERT INTO dbo.CleanUsers (user_id, age, gender, occupation, zip_code, age_group)
SELECT
    user_id,
    age,
    CASE gender WHEN 'M' THEN 'Male' WHEN 'F' THEN 'Female' END,
    occupation,
    zip_code,
    CASE
        WHEN age < 18 THEN 'Under 18'
        WHEN age BETWEEN 18 AND 35 THEN '18-35'
        WHEN age BETWEEN 36 AND 60 THEN '36-60'
        ELSE '60+' END AS age_group
FROM Stg.Users_Stg;

INSERT INTO dbo.CleanRatings (user_id, item_id, rating, ts)
SELECT
    user_id,
    item_id,
    rating,
    DATEADD(SECOND, CAST(timestamp AS BIGINT), '1970-01-01')
FROM Stg.Ratings_Stg;
```

The Messages pane at the bottom indicates the execution results: (926 rows affected) and (100000 rows affected).

6.4. Veri Kalitesi Raporları

ETL sürecinin her bir aşamasında işlenen kayıt sayıları kaydedilir ve temizleme öncesi/sonrası değerler karşılaştırılır. Bu sayede veri kaybı takip edilir, kalite kontrolü yapılır ve süreç performansı ölçümlenir.

ETL süreci sırasında kayıt öncesi/sonrası satır sayıları gibi metrikler **ETL_DataQualityLog** tablosuna eklenir.

```

CREATE TABLE dbo.ETL_DataQualityLog (
    RunDate          DATETIME2  DEFAULT SYSUTCDATETIME(),
    Stage            VARCHAR(20),
    RecordsBefore    INT,
    RecordsAfter     INT
);

-- Örnek: Users temizleme öncesi / sonrası
DECLARE @before INT = (SELECT COUNT(*) FROM Stg.Users_Stg);
-- (temizleme adımları yapıldıktan sonra)
DECLARE @after  INT = (SELECT COUNT(*) FROM dbo.CleanUsers);

INSERT INTO dbo.ETL_DataQualityLog (Stage, RecordsBefore, RecordsAfter)
VALUES ('Users Cleaning', @before, @after);

```

Oluşturduğu tablo ve çıktısı aşağıdadır.

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the Object Explorer displays the database structure for 'MovieLensDB', including tables like 'dbo.CleanRatings', 'dbo.CleanUsers', 'dbo.ETL_DataQualityLog', 'dbo.ratings', 'dbo.users', 'Stg.Ratings_Stg', and 'Stg.Users_Stg'. The 'dbo.ETL_DataQualityLog' table is selected. On the right, the SQL Query window shows a query that selects the top 1000 records from the 'dbo.ETL_DataQualityLog' table, displaying columns: RunDate, Stage, RecordsBefore, and RecordsAfter. The Results pane at the bottom shows the output of the query, which is a single row with the following values:

RunDate	Stage	RecordsBefore	RecordsAfter
2025-05-24 15:04:19.0992305	Users Cleaning	926	926

7. Veritabanı Yükseltme ve Sürüm Yönetimi

Bu bölümde veritabanının yükseltilmesi ve sürüm yönetimi hakkında oluşturulan senaryolar ve yapılması gerekenler açıklanmıştır.

7.1. Veritabanı Yükseltme Planı

Senaryo: SQL Server 2019 → SQL Server 2022

Adım-Adım Geçiş Adımları:

1. FULL yedek alınır ve test instance'a restore edilir.
2. Data Migration Assistant (DMA) ile analiz yapılır; uyumluluk raporu alınır.
3. Deprecated/behavior-change uyarıları gözden geçirilir.
4. Cut-over: Son FULL yedek + transaction log replay (standby).
5. Uygulama bağlantıları yeni instance'a yönlendirilir.

Rollback Senaryosu:

Restore Job: Süreç sonunda health-check hatası alınırsa, otomatik olarak standby yedeğe geri dönülür.

Log Shipping Geri Dönüşü: Orijinal primari tekrar aktif edilerek servis devam ettirilir.

7.2. Sürüm Yönetimi

Bu bölümde, yükseltmeler ve rollback senaryoları için yapılması gereken sürüm yönetimleri açıklanmıştır.

7.2.1. DDL'nun İzlenmesi

Her DDL işlemini schema_audit tablosuna kaydeden trigger aşağıdaki gibidir:

```

IF NOT EXISTS (SELECT 1 FROM sys.tables WHERE name = 'schema_audit')
BEGIN
    CREATE TABLE dbo.schema_audit (
        audit_id INT IDENTITY PRIMARY KEY,
        event_time DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME(),
        login_name SYSNAME NOT NULL,
        event_type NVARCHAR(100),
        object_type NVARCHAR(100),
        object_name NVARCHAR(100),
        sql_text NVARCHAR(MAX)
    );
END
GO

IF OBJECT_ID('dbo.trg_DDL_Audit','TR') IS NULL
BEGIN
    CREATE TRIGGER dbo.trg_DDL_Audit
    ON DATABASE
    AFTER CREATE_TABLE, ALTER_TABLE, DROP_TABLE,
        CREATE_PROCEDURE, ALTER_PROCEDURE, DROP_PROCEDURE
    AS
    BEGIN
        INSERT INTO dbo.schema_audit(login_name, event_type, object_type, object_name, sql_text)
        SELECT
            ORIGINAL_LOGIN(),
            EVENTDATA().value('/EVENT_INSTANCE/EventType[1]','NVARCHAR(100)'),
            EVENTDATA().value('/EVENT_INSTANCE/ObjectTypeId[1]','NVARCHAR(100)'),
            EVENTDATA().value('/EVENT_INSTANCE/ObjectName[1]','NVARCHAR(100)'),
            EVENTDATA().value('/EVENT_INSTANCE/TSQLCommand[1]','NVARCHAR(MAX)');
    END;
END
GO

```

7.2.2. Migration Script Örneği

Aşağıda iki adımlı örnek migration script'i yer alıyor. Rapor ekinde “MovieLensDB_Migrations.sql” adıyla verilecek.

Migration V1 – Index Ekleme:

```

IF NOT EXISTS (
    SELECT 1 FROM sys.indexes
    WHERE name = 'IX_ratings_UserId_MovieId'
    AND object_id = OBJECT_ID('dbo.ratings')
)
BEGIN
    CREATE NONCLUSTERED INDEX IX_ratings_UserId_MovieId
    ON dbo.ratings(UserId, MovieId);
END
GO

-- Migration V2 - CreatedDate Kolonu Ekleme
IF COL_LENGTH('dbo.ratings','CreatedDate') IS NULL
BEGIN
    ALTER TABLE dbo.ratings
    ADD CreatedDate DATETIME2 NOT NULL
    CONSTRAINT DF_ratings_CreatedDate DEFAULT SYSUTCDATETIME();
END
GO

```

7.3. Test ve Doğrulama

Bu bölümde yapılan yükseltmeler ve sürüm kontrolleri için kullanılan test ve doğrulama

yöntemleri açıklanmıştır.

7.3.1. Test Senaryoları

Veri Bütünlüğü: Kayıt sayısının korunması (COUNT(*) ve checksum karşılaştırma)

Performans: Kritik sorguların tepki sürelerinin kontrolü (Query Store Before/After istatistikleri)

Bağlantı Testi: Uygulama–DB entegrasyonunun doğrulanması (Smoke test API çağrıları, UAT)