ANKARA ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM4522 – Ağ Tabanlı Paralel Dağıtım Sistemleri

Abdullah Gedik - 20290251

Ahmet Buğra Özer - 20290281

18/06/2025

Giriş

Bu doküman üzerinde gerçekleştirilen projeler Microsoft'un örnek veri tabanlarından "**pubs**" veritabanı üzerinde gerçekleştirilmiştir.

Veri Tabanının Linki: https://github.com/microsoft/sql-server-samples/blob/master/samples/databases/northwind-pubs/instpubs.sql

Proje GitHub Linkleri:

Abdullah GEDİK

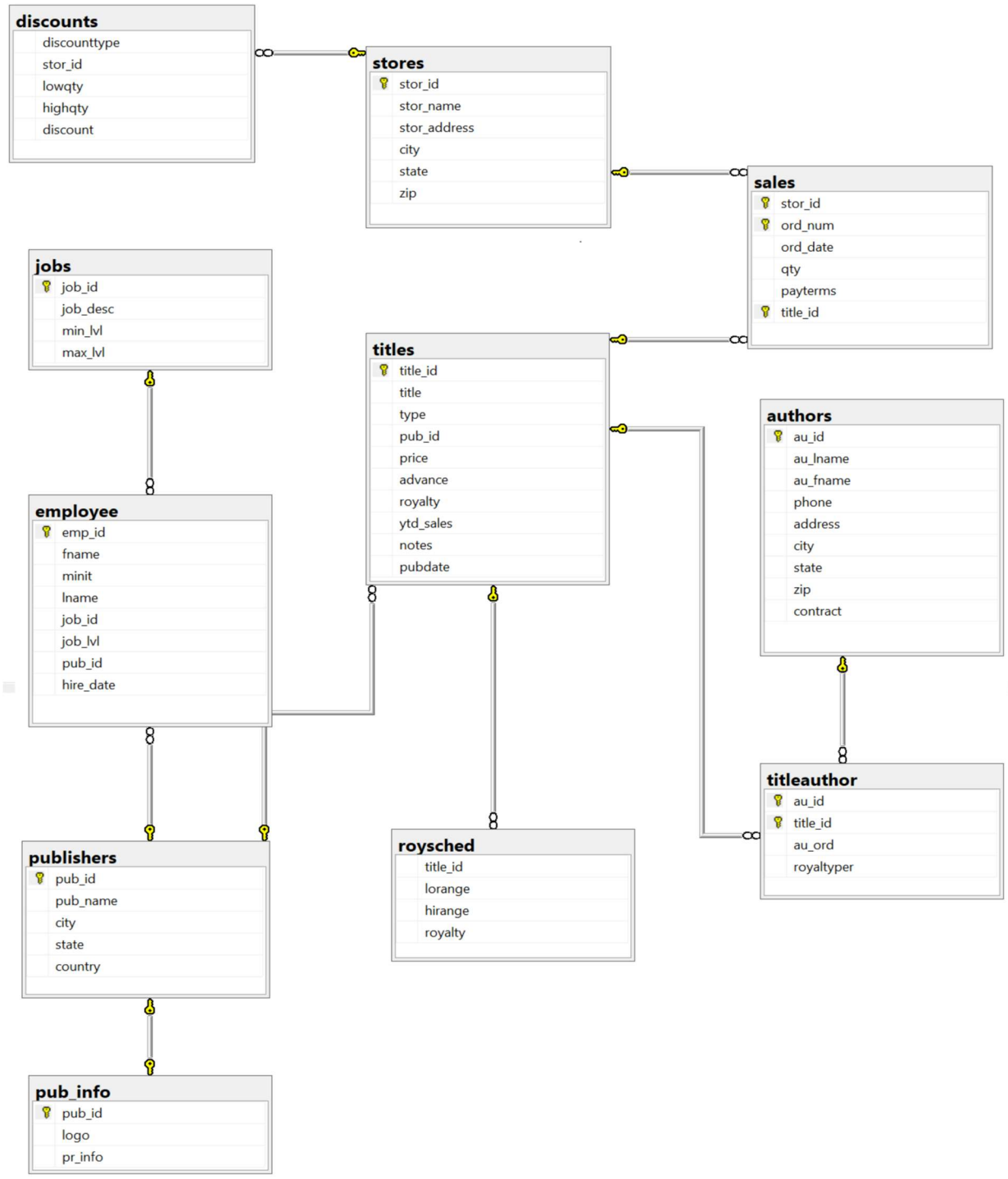
- https://github.com/abdullahgedik/MSSQL_Project

Ahmet Buğra Özer

- https://github.com/ahmetbugraozer/MSSQL_Project

Proje Adımlarının Videoları:

https://drive.google.com/drive/folders/1F96I1jNVQnzVSdowzRYIpNcsvvuy3lOz



Proje 1: Veri Tabanı Performans Optimizasyonu ve İzleme

1. Veritabanı İzleme (Monitoring & Analiz)

Amaç: Performansı etkileyen sorguları ve sistem kaynaklarını izlemek.

1.1 SQL Server Profiler ile İzleme

- SSMS > Tools > **SQL Server Profiler** açılır.
- Yeni bir Trace başlatılır:
    - Server'a bağlanılır.
    - Trace Properties penceresinde:
        - **Trace template:** "TSQL_Replay" ya da "TSQL_Duration"
        - **Events Selection** sekmesinde: `Duration`, `Reads`, `Writes`, `CPU`, `TextData`, `LoginName`, `DatabaseName` gibi alanlar seçilir.
    - Start

1.2 DMV (Dynamic Management Views) Kullanımı

Örnek: En çok kaynak tüketen ilk 5 sorguyu bulmak:

```
SELECT TOP 5
    qs.total_elapsed_time / qs.execution_count AS [Average Time],
    qs.execution_count,
    qs.total_logical_reads,
    qs.total_logical_writes,
    st.text AS [SQL Text]
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS st
ORDER BY [Average Time] DESC;
```

2. İndeks Yönetimi

Amaç: Sorguları hızlandırmak için doğru indeks kullanımı.

2.1 Mevcut İndekslerin Görülmesi

```
SELECT * FROM sys.indexes WHERE object_id = OBJECT_ID('authors');
```

2.2 Önerilen İndeksler

```
SELECT
    migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks +
migs.user_scans) AS impact,
    mid.statement AS TableName,
    mid.equality_columns,
    mid.inequality_columns,
    mid.included_columns
FROM sys.dm_db_missing_index_groups mig
JOIN sys.dm_db_missing_index_group_stats migs ON migs.group_handle =
mig.index_group_handle
JOIN sys.dm_db_missing_index_details mid ON mig.index_handle =
mid.index_handle
ORDER BY impact DESC;
```

2.3 Gereksiz İndeksleri Bulma

```
SELECT
    OBJECT_NAME(i.object_id) AS TableName, i.name AS IndexName,
i.index_id,
    user_seeks, user_scans, user_lookups, user_updates
FROM sys.dm_db_index_usage_stats s
INNER JOIN sys.indexes i ON s.object_id = i.object_id AND s.index_id =
i.index_id
WHERE OBJECTPROPERTY(s.object_id,'IsUserTable') = 1
ORDER BY user_seeks + user_scans DESC;
```

3. Sorgu İyileştirme

Amaç: Yavaş sorguları tespit edip, execution plan a göre optimize etmek. Adım adım:

3.1 Sorgu Sürelerinin Ölçülmesi

```
SET STATISTICS TIME ON;
-- Sorgu buraya girilir
SET STATISTICS TIME OFF;
```

3.2 Execution Plan Kullanımı

- Sorgu girilir> SSMS'te "**Include Actual Execution Plan**" tuşuna basılr > Sorgu çalıştırılır.
- Execution plan'da "Table Scan" ya da "Key Lookup" varsa, indeks eksikliği olabilir.

---

4. Erişim ve Rol Yönetimi

Amaç: Farklı kullanıcı rolleri oluşturarak güvenli erişim sağlamak. Adım adım:

4.1 Rol Oluşturma

```
CREATE ROLE DatabaseAdmin
GRANT SELECT ON sales TO DatabaseAdmin --sales tablosu için
GRANT SELECT ON titles TO DatabaseAdmin --titles tablosu için

GRANT INSERT, UPDATE ON sales TO DatabaseAdmin
GRANT INSERT, UPDATE ON titles TO DatabaseAdmin
```

4.2 Yeni Kullanıcı Oluşturulur

```
CREATE LOGIN DatabaseAdmin WITH PASSWORD = 'password';
CREATE USER new_admin FOR LOGIN DatabaseAdmin;
EXEC sp_addrolemember DatabaseAdmin, new_admin;
```

4.3 Rol Atama

```
EXEC sp_addrolemember 'db_datareader', 'ornekKullanici';
```

Proje 2: Veri Tabanı Yedekleme ve Felaketten Kurtarma Planı

1. Yedekleme Stratejileri

Amaç: Veri kaybını en aza indirecek şekilde farklı yedekleme türlerini kullanmak.

1.1 Tam (Full) Yedekleme

Veritabanının tüm verilerini yedekler.

```
BACKUP DATABASE Pubs TO DISK = 'C:\Backup\Pubs_FULL.bak' WITH FORMAT,
INIT;
```

1.2 Fark (Differential) Yedekleme

Son tam yedekten bu yana değişen verileri yedekler. Daha hızlıdır.

```
BACKUP DATABASE Pubs TO DISK = 'C:\Backup\Pubs_DIFF.bak' WITH
DIFFERENTIAL;
```

1.3 Artık (Transaction Log) Yedekleme

Almış olduğunuz yedeğin (differential veya full) log kayıtlarını almak için sorgunun devamına alttaki satırı ekleyin:

```
BACKUP LOG Pubs TO DISK = 'C:\Backup\Pubs_LOG.trn';
```

Not: Transaction log yedekleme yapabilmek için veri tabanı `FULL` recovery modelde olmalı:

```
ALTER DATABASE Pubs SET RECOVERY FULL;
```

2. Zamanlayıcıyla Otomatik Yedekleme

Amaç: Manuel işlem yapmadan düzenli olarak yedeklemek.

2.1 SQL Server Agent ile Backup Job Oluşturma

- SSMS > Object Explorer > **SQL Server Agent > Jobs > New Job**
- **Steps** sekmesinde:
    - Yeni Step → Ad: `FullBackup`

- Type: `Transact-SQL script (T-SQL)`

```
BACKUP DATABASE Pubs TO DISK = 'C:\Backup\Pubs_AUTO_FULL.bak';
```

- **Schedules** sekmesinde:
  - Yeni zamanlama ekle (örneğin: her gece saat 02:00)
- Job kaydedilir ve test edilir.
- Aynı şekilde diff ve log yedekleme işleri de zamanlanabilir.

---

3. Felaketten Kurtarma (Disaster Recovery)

Amaç: Veritabanı silindiğinde veya bozulduğunda, geri yükleyebilmek.

3.1 Tam + Fark + Log ile Point-in-Time Restore

Senaryo: Bugün saat 14:00'te yanlışlıkla veri silindi, saat 13:45'e geri dönmek istiyoruz.

1. Önce tam yedek restore edilir (ama veri tabanı hemen kullanıma açılmaz):

```
RESTORE DATABASE Pubs FROM DISK = 'C:\Backup\Pubs_FULL.bak' WITH
NORECOVERY;
```

2. Ardından fark yedeği yüklenir:

```
RESTORE DATABASE Pubs FROM DISK = 'C:\Backup\Pubs_DIFF.bak' WITH
NORECOVERY;
```

3. Son olarak log yedeğiyle belirli zamana dönülür:

```
RESTORE LOG Pubs FROM DISK = 'C:\Backup\Pubs_LOG.trn'
WITH STOPAT = '2025-04-23 13:45:00', RECOVERY;
```

4. Test Yedekleri

Amaç: Yedeklerin gerçekten işe yarayıp yaramadığını kontrol etmek.

4.1 Yedeği Farklı Bir İsimle Restore Etmek

```
RESTORE DATABASE Pubs_Test FROM DISK = 'C:\Backup\Pubs_FULL.bak'
WITH MOVE 'pubs' TO 'C:\Backup\Pubs_Test_Data.mdf',
```

```
    MOVE 'pubs_log' TO 'C:\Backup\Pubs_Test_Log.ldf',
    REPLACE;
```

Bu işlem, var olan bir yedeğin başka bir isimle başarıyla yüklenip yüklenmeyeceğini test eder.

Proje 3: Veri Tabanı Güvenliği ve Erişim Kontrolü

1. Erişim Yönetimi

Amaç: Veritabanına kimlerin eriştiğini belirlemek ve yetki sınırları koymak.

1.1 SQL Server Authentication ile Kullanıcı Oluşturma

```
CREATE LOGIN newLogin WITH PASSWORD = 'password';
CREATE USER newUser FOR LOGIN newLogin;
EXEC sp_addrolemember 'db_datareader', 'newUser';
```

1.2 Windows Authentication (AD hesabı ile erişim)

Bu yöntem domain ortamlarında kullanılır:

```
CREATE LOGIN [DOMAIN\username] FROM WINDOWS;
CREATE USER [DOMAIN\username] FOR LOGIN [DOMAIN\username];
EXEC sp_addrolemember 'db_owner', [DOMAIN\username];
```

Fark: SQL Auth şifreye bağlıdır, Windows Auth işletim sistemi kullanıcılarına bağlıdır.

2. Veri Şifreleme (TDE - Transparent Data Encryption)

Amaç: Veritabanı dosyalarının çalınması durumunda bile verilerin gizli kalmasını sağlamak.

2.1 TDE için anahtar ve sertifika oluşturma

```
USE master;
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'password';

CREATE CERTIFICATE TDESertifikasi WITH SUBJECT = 'TDE Sertifikasi';

USE Pubs;
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE TDESertifikasi;

ALTER DATABASE Pubs SET ENCRYPTION ON;
```

Artık `.mdf` ve `.ldf` dosyaları şifrelenmiş olacak.

3. SQL Injection Testleri ve Önlemleri

Amaç: SQL Injection saldırılarını anlamak, test etmek ve bu saldırılara karşı etkili koruma yöntemleri uygulamak.

SQL Injection: Kullanıcıdan alınan giriş verisinin doğrudan SQL sorgularına yerleştirilmesiyle kötü amaçlı SQL komutlarının çalıştırılmasına neden olur.

3.1 SQL Injection Test Senaryoları

Veri tabanında `authors` tablosu üzerinden test yapabiliriz.

Test 1: Normal Sorgu (Beklenen)

```
SELECT * FROM authors WHERE au_lname = 'White';
```

Test 2: Injection Sorgusu (Saldırı)

```
SELECT * FROM authors WHERE au_lname = '' OR 1=1 --';
```

Bu sorgu, tüm `authors` tablosunu getirir. Kötü niyetli kullanıcı bu şekilde verileri yetkisiz biçimde görüntüleyebilir.

3.2 SQL Injection'a Karşı Korunma Yöntemleri

1. Parametreli Sorgular (Stored Procedure veya Prepared Statements)

Stored Procedure Örneği:

```
CREATE PROCEDURE GetAuthorByLastName
```

```
    @lastname NVARCHAR(50)
AS
BEGIN
    SELECT * FROM authors WHERE au_lname = @lastname;
END
```

Kullanımı:

```
EXEC GetAuthorByLastName @lastname = 'White';
```

Bu yöntem dışarıdan gelen verileri doğrudan SQL'e yapıştırmaz.

2. ORM Kullanımı (Entity Framework, Dapper, vs.)

Modern uygulamalarda ORM'ler injection'a karşı doğası gereği korumalıdır:

```
-- Örneğin sadece harf içeren girişleri kabul et
DECLARE @lname NVARCHAR(100)
SET @lname = 'White'

IF @lname NOT LIKE '%[^a-zA-Z ]%'
BEGIN
    EXEC GetAuthorByLastName @lname
END
ELSE
BEGIN
    PRINT 'Geçersiz karakter içeriyor'
END
```

3. Giriş Doğrulama ve Filtreleme

- Girişlerde `'`, `--`, `;` gibi karakterleri engelle.
- Regex kullanarak girişleri sadece harf ve sayı ile sınırla.

4. Audit Logları Oluşturma

Amaç: Kim ne zaman ne yaptığını izleyebilmek.

SQL Server Audit ile injection'a neden olan kullanıcı etkinliklerini logla:

```
-- Audit log’u almak için master database’ine geçiş yapılması gerekiyor
USE master;
GO

-- Basit audit örneği
CREATE SERVER AUDIT Audit_SQLInjection
TO FILE (FILEPATH = 'C:\AuditLogs\');

CREATE SERVER AUDIT SPECIFICATION AuditSelectStatements
FOR SERVER AUDIT Audit_SQLInjection
ADD (SCHEMA_OBJECT_ACCESS_GROUP);

ALTER SERVER AUDIT Audit_SQLInjection WITH (STATE = ON);
```

Özet:

Teknik	Açıklama
Parametreli sorgular	Kullanıcı girdileri SQL'e gömülmeden çalışır
Stored Procedure kullanımı	SQL kodu ile veri ayrışır
ORM’ler	Girişler otomatik filtrelenir
Giriş doğrulama / whitelist	Zararlı karakterler engellenir
WAF ve SQL Audit	Saldırı tespiti ve izleme

Proje 4: Veri Tabanı Yük Dengeleme ve Dağıtık Veritabanı Yapıları

1. Veritabanı Replikasyonu Kurulumu

1.1. Distribütör’ü (Distributor) Yapılandırma

1. **SSMS** → `Replication` → sağ tık → `Configure Distribution…`

2. Wizard'ı kullanarak "publisher" olarak `SQLPRINCIPAL`'ı seçip, yeni bir distribution veritabanı (`distribution`) oluşturulur.

   Bu işlem, arka planda `sp_adddistributor`, `sp_adddistributiondb`, `sp_adddistpublisher` prosedürlerini çağırır.

1.2. Yayıncı (Publisher) ve Yayın (Publication) Oluşturma

1. SSMS'te `Replication` → `Local Publications` → sağ tık → `New Publication…`

2. Sunucu olarak `SQLPRINCIPAL`'ı seçilir.

3. Database olarak da `pubs` seçilir.

4. Devamında Replikasyon türü olarak **Transactional** (anlık senkron) seçilir.

5. Publication'a uygun bir isim verilir.

6. İçeriğe dahil etmek için gerekli tablolar işaretlenir.

1.3. Abone (Subscriber) ve Abonelik (Subscription) Oluşturma

1. `Replication` → `Local Subscriptions` → sağ tık → `New Subscriptions…`

2. Publication olarak az önce oluşturulan seçilir.

3. Subscriber makine olarak `SQLMIRROR`, hedef database olarak yeni bir `pubs_sub` veritabanı seçilir.

4. "Pull Subscription" ile sunucu aboneye push eder ya da "Push Subscription" ile yayıncı gönderir.

5. Son olarak Schedule'lar ayarlanır.

---

2. Database Mirroring ile Yük Dengeleme

Mirroring, daha hafif bir senkron/osenkron yedek tutma ve otomatik geçiş (failover) sunar.

3.1. Endpoint'leri Oluşturma

Her iki instansta da T-SQL ile:

```
-- Principal (SQLPRINCIPAL)
CREATE ENDPOINT [Mirroring]
    STATE=STARTED
    AS TCP (LISTENER_PORT=5022)
    FOR DATABASE_MIRRORING (ROLE=ALL);
```

```
-- Mirror (SQLMIRROR)
CREATE ENDPOINT [Mirroring]
    STATE=STARTED
    AS TCP (LISTENER_PORT=5022)
    FOR DATABASE_MIRRORING (ROLE=ALL);
```

3.2. Güvenlik (Certificates/Logins) Ayarı

Basit ortam için Windows Authentication kullanabilirsiniz (aynı Domain üzerindeyse):

```
-- Principal
ALTER DATABASE pubs SET PARTNER = 'TCP://SQLMIRROR.domain.local:5022';

-- Mirror
ALTER DATABASE pubs SET PARTNER =
'TCP://SQLPRINCIPAL.domain.local:5022';
```

Eğer otomatik failover isterseniz bir **witness** instansı da kurun ve:

```
-- Principal
ALTER DATABASE pubs SET WITNESS = 'TCP://SQLWITNESS.domain.local:5022';
```

3.3. Mirroring'i Başlatma

```
-- Principal
ALTER DATABASE pubs SET SAFETY FULL;           -- SYNCHRONOUS
ALTER DATABASE pubs SET PARTNER SAFETY FULL;   -- Güvenli failever

-- Mirror
-- (zaten yukarıda partner tanımlandı)
```

Test:

- `SQLPRINCIPAL`'i durdurun.
- SSMS'te `pubs` veritabanının `Mirror State`'inin `Failover` ile `Principal` olarak atandığını görün.

---

3. Always On Availability Groups (AG) ile Yük Dengeleme

Always On AG, mirroring'e göre daha güçlüdür; read-only yedek sunucular, dağıtık grup yapısı, otomatik failover ve bakım kolaylığı sunar.

4.1. Önkoşullar

1. **Windows Failover Cluster** kurulumu ve konfigürasyonu yapılmış olmalı.
2. Her bir SQL instansı "Always On Availability Groups" özelliği etkinleştirilmeli (SQL Server Configuration Manager'dan).

4.2. AG'yi Oluşturma

1. SSMS → `Always On High Availability` → `New Availability Group Wizard…`
2. AG'ye bir isim verin (`AG_Pubs`).
3. Primary replica olarak `SQLPRINCIPAL`, secondary olarak `SQLMIRROR` ekleyin.
4. Database olarak `pubs`'ı seçin (önce FULL yedek alınmış ve RESTORE WITH NORECOVERY yapılmış olmalı).

5. Listener oluşturun (ör. `pubs-listener` IP ve port tanımlayın).

6. Senkronizasyon modu: Synchronous with automatic failover seçin.

4.3. Okuma ve Yazma Yönlendirmesi

- Uygulamalar yazma işlemleri için listener'ı (`pubs-listener`) kullanır.
- Read-only istekler için doğrudan sekonderlere yönlendirme yapılabilir (Connection string'de `ApplicationIntent=ReadOnly`).

Test:

- Primary üzerinde tabloya satır ekleyin.
- Sekonder üzerinde satırın göründüğünü kontrol edin.
- Primary'i kapatıp failover'ın otomatik gerçekleştiğini ve listener üzerinden yazma-okuma işlemlerinin sürdüğünü test edin.

---

4. Failover Senaryoları ve İzleme

1. **Manuel Failover**
    - SSMS'te AG → sağ tık → `Failover…` wizard'ı kullanarak sekonder'i primary yapın.
2. **Otomatik Failover**
    - Eğer mirroring'de witness veya AG'de automatic failover seçiliyse, primary düştüğünde birkaç saniye içinde geçiş gerçekleşir.
3. **Monitoring**
    - **Replikasyon**:
        - `Replication Monitor` ile Latency, Delivery latency, undelivered commands izleyin.
    - **Mirroring**:

`sys.database_mirroring_states` DMV'si:

```
 SELECT DB_NAME(database_id) AS DBName, mirroring_state_desc,
mirroring_role_desc
FROM sys.database_mirroring_states;
```

- ■
- **AG**:
  - ■ `Always On Dashboard`
  - ■ DMV'ler:

```
SELECT * FROM sys.dm_hadr_availability_replica_states;
SELECT * FROM sys.dm_hadr_database_replica_states;
```

Özet

1. **Replikasyon** ile tablo düzeyinde veri çoğaltma ve senkronizasyon.
2. **Database Mirroring** ile tam veritabanı seviyesinde yedek tutma ve failover.
3. **Always On AG** ile ölçeklenebilir, read-only replika desteği ve güçlü otomatik failover.
4. **Failover Testleri** ve izleme DMV'leri ile sürekliliği garanti altına alma.

Bu adımları "pubs" veritabanı üzerinde uygulamış olduk.

Proje 5: Veri Temizleme ve ETL Süreçleri Tasarımı

Amaç:

Verilerin **doğruluğunu, tutarlılığını ve analiz edilebilirliğini** sağlamak için SQL Server ortamında **ETL (Extract, Transform, Load)** süreci uygulanır. Bu süreçle veri hataları düzeltilir, farklı kaynaklardan gelen veriler birleştirilir ve hedef tablolara yüklenir.

---

1. Veri Temizleme (Data Cleaning)

Örnek Senaryo: `authors` tablosundaki eksik ve hatalı verilerin tespiti ve düzeltilmesi.

1. Eksik Değerlerin Tespiti:

```
-- Telefon veya adres bilgisi olmayan yazarlar
SELECT * FROM authors
WHERE phone IS NULL OR address IS NULL;
```

2. Hatalı Formatların Tespiti:

```
-- Hatalı telefon formatı: XXX XXX-XXXX olmalı (örneğin 415 555-1212)
SELECT * FROM authors
WHERE phone NOT LIKE '[0-9][0-9][0-9] [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]';
```

3. Hatalı Verilerin Güncellenmesi:

```
-- Eksik telefon numaralarına varsayılan değer atanması
UPDATE authors
SET phone = '000 000-0000'
WHERE phone IS NULL;
```

---

2. Veri Dönüştürme (Data Transformation)

Örnek Senaryo: `titles` tablosundaki fiyatların döviz kuru dönüşümü ile normalize edilmesi.

1. USD → TRY dönüşümü (örneğin kur: 32.5)

```
SELECT title_id, title, price, price * 32.5 AS price_in_try
FROM titles
WHERE price IS NOT NULL;
```

2. Yeni sütunla güncelleme:

```
ALTER TABLE titles ADD price_try MONEY;
```

```sql
UPDATE titles
SET price_try = price * 32.5
WHERE price IS NOT NULL;
```

---

3. Veri Yükleme (Data Loading)

Örnek Senaryo: Temizlenen verilerin `authors_cleaned` adlı hedef tabloya aktarılması.

1. Temiz tabloyu oluştur:

```sql
SELECT * INTO authors_cleaned FROM authors WHERE 1 = 0;
```

2. Temizlenmiş veriyi yükle:

```sql
INSERT INTO authors_cleaned
SELECT * FROM authors
WHERE phone IS NOT NULL AND address IS NOT NULL;
```

> Alternatif olarak SSIS gibi araçlarla CSV, Excel veya dış veri kaynaklarından veri yükleme de mümkündür.

---

4. Veri Kalitesi Raporları

Örnek: Temizleme sonrası veri kalitesine dair temel rapor

1. Eksik veri oranı:

```sql
SELECT
  COUNT(*) AS total_records,
  SUM(CASE WHEN phone IS NULL THEN 1 ELSE 0 END) AS missing_phone,
  SUM(CASE WHEN address IS NULL THEN 1 ELSE 0 END) AS missing_address
FROM authors;
```

2. Uyumlu format oranı:

```sql
SELECT
```

```
  COUNT(*) AS total_authors,
  SUM(CASE
    WHEN phone LIKE '[0-9][0-9][0-9] [0-9][0-9][0-9]-[0-9][0-9][0-9][0-
9]'
    THEN 1 ELSE 0 END) AS valid_phone_count
FROM authors;
```

Sonuç:

Süreç	Açıklama
Veri Temizleme	Eksik ve hatalı kayıtların tespiti ve düzeltilmesi
Dönüştürme	Farklı formatların normalize edilmesi
Yükleme	Hedef tablolara aktarım işlemi
Raporlama	Kalite kontrollerine dair çıktıların üretilmesi

Bu süreçler manuel SQL komutları ile olduğu kadar, **SSIS** gibi görsel ETL araçlarıyla da otomatikleştirilebilir.

Proje 6: Veri Tabanı Yükseltme ve Sürüm Yönetimi

Amaç:

SQL Server ortamında mevcut bir veritabanını daha yeni bir sürüme sorunsuz şekilde yükseltmek ve bu süreçte yapılan tüm değişiklikleri kontrol altına almak. Ayrıca, yükseltme sonrası test ve geri dönüş stratejilerinin oluşturulması.

---

1. Veritabanı Yükseltme Planı

1. Ön Analiz:

- Mevcut SQL Server sürümünü belirle (örneğin SQL Server 2012)
- Hedef sürümü netleştir (örneğin SQL Server 2019)
- Kullanılan özel işlevler (CLR, Service Broker, vb.) kontrol edilir
- In-place veya side-by-side migration tercihi yapılır

2. Yedekleme:

Yükseltme öncesi tüm veritabanlarının **tam yedeği** alınmalıdır.

```
BACKUP DATABASE pubs TO DISK = 'C:\Backup\pubs_pre_upgrade.bak';
```

3. Uyumluluk Kontrolleri (Microsoft DMA aracı ile):

Microsoft'un **Data Migration Assistant (DMA)** aracı ile aşağıdakiler tespit edilir:

- Eski fonksiyonlar (deprecated features)
- Uyumluluk sorunları
- Performans önerileri

---

2. Sürüm Yönetimi (Schema Version Control)

Veritabanı yapısındaki değişiklikleri takip etmek için manuel veya otomatik yöntemler kullanılabilir.

1. DDL Triggers ile Şema Değişikliklerini İzleme

Örnek DDL Trigger:

```
CREATE TRIGGER ddl_audit_trigger
```

```
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
BEGIN
    INSERT INTO ddl_log (event_time, user_name, event_type, object_name,
t_sql)
    SELECT
        GETDATE(),
        SYSTEM_USER,
        EVENTDATA().value('(/EVENT_INSTANCE/EventType)[1]',
'nvarchar(100)'),
        EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]',
'nvarchar(100)'),

EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]',
'nvarchar(max)');
END;
```

Gerekli Kayıt Tablosu:

```
CREATE TABLE ddl_log (
    id INT IDENTITY(1,1) PRIMARY KEY,
    event_time DATETIME,
    user_name NVARCHAR(100),
    event_type NVARCHAR(100),
    object_name NVARCHAR(100),
    t_sql NVARCHAR(MAX)
);
```

Bu sayede kim, ne zaman, hangi yapısal değişikliği yaptı detaylıca izlenebilir.

---

3. Test ve Geri Dönüş Planı

1. Test Senaryoları:

- CRUD işlemleri yapılır

- Saklı yordamlar (stored procedures) ve tetikleyiciler (triggers) çalıştırılır
- Performans karşılaştırmaları yapılır
- Uygulama ile entegrasyon testleri

2. Rollback Stratejisi:

- Geri dönülebilirlik için yükseltmeden **önce alınan yedek** dosyası saklanır
- **Side-by-side** geçiş yapıldıysa eski sunucu çevrimdışı alınmaz
- Olağanüstü durumda hızlıca restore yapılabilir:

```
RESTORE DATABASE pubs FROM DISK = 'C:\Backup\pubs_pre_upgrade.bak'
WITH REPLACE;
```

Özet:

Süreç	Açıklama
Yükseltme Planı	Uyumluluk analizleri, yedekleme, geçiş tipi seçimi
Sürüm Takibi	DDL trigger ile otomatik yapısal değişiklik izleme
Test Planı	Yükseltme sonrası fonksiyon, performans ve uygulama testleri
Geri Dönüş	Backup restore ya da eski sürümde kalma opsiyonu

Sürüm yönetimi süreci, **SQL Source Control**, **Git**, veya **Redgate SQL Change Automation** gibi gelişmiş araçlarla da otomatikleştirilebilir.

Proje 7: Veri Tabanı Yedekleme ve Otomasyon Çalışması

Amaç:

SQL Server'da veritabanı yedekleme işlemlerini otomatikleştirerek, veri kaybı riskini azaltmak ve yönetim süreçlerini optimize etmek. Ayrıca yedekleme işlemlerinin düzenli, hatasız gerçekleşmesini sağlamak için raporlama ve uyarı sistemleri oluşturulmuştur.

---

1. SQL Server Agent ile Otomatik Yedekleme

1. SQL Server Agent İş Tanımı:

SQL Server Agent aracılığıyla günlük olarak çalışan bir **backup job** oluşturulur.

T-SQL Komutu (Full Backup):

```
DECLARE @BackupPath NVARCHAR(500)
SET @BackupPath = 'C:\Backup\pubs_full_' + CONVERT(VARCHAR(8),
GETDATE(), 112) + '.bak'

BACKUP DATABASE pubs
TO DISK = @BackupPath
WITH INIT, COMPRESSION, STATS = 10;
```

> `GETDATE(), 112` ifadesi ile yedeğin adında tarih bilgisi yer alır (`20250528` formatında).

2. SQL Server Agent Job Oluşturma Adımları:

1. SQL Server Management Studio (SSMS) → SQL Server Agent → Jobs → New Job
2. Ad: `Full_Backup_Pubs`
3. Steps sekmesinde yukarıdaki T-SQL komutu eklenir
4. Schedules sekmesinde **günlük saat 02:00** olarak zamanlanır

---

2. PowerShell/T-SQL ile Yedekleme Raporları

1. Son Yedeklemeyi Kontrol Eden T-SQL Sorgusu:

```
SELECT
    database_name,
    MAX(backup_finish_date) AS last_backup_date,
```

```
    DATEDIFF(HOUR, MAX(backup_finish_date), GETDATE()) AS
hours_since_last_backup
FROM msdb.dbo.backupset
WHERE type = 'D'
GROUP BY database_name;
```

Bu sorgu, tam yedeklerin ne kadar süre önce alındığını gösterir. 24 saatten eskiyse müdahale gerekir.

---

3. Otomatik Uyarı Sistemi

1. SQL Server Database Mail Ayarı (tek seferlik yapılandırma):

```
-- Database Mail'i aktif et
EXEC sp_configure 'show advanced options', 1; RECONFIGURE;
EXEC sp_configure 'Database Mail XPs', 1; RECONFIGURE;
```

2. Operatör Tanımı:

```
USE msdb;
EXEC msdb.dbo.sp_add_operator
    @name = N'AdminOperator',
    @email_address = N'dba@example.com';
```

3. Alert Tanımı:

```
EXEC msdb.dbo.sp_add_alert
    @name = N'Backup Failure Alert',
    @message_id = 18210,  -- Backup failure error code
    @severity = 016,
    @enabled = 1,
    @notification_message = N'Backup job failed!',
    @include_event_description_in = 1;
```

4. Operatöre Bildirim:

```
EXEC msdb.dbo.sp_add_notification
    @alert_name = N'Backup Failure Alert',
    @operator_name = N'AdminOperator',
    @notification_method = 1; -- 1 = Email
```

Özet:

Süreç	Açıklama
Otomatik Yedekleme	SQL Server Agent ile günlük yedekleme job'u
Raporlama	Son yedekleme tarihleri ve eksik yedeklerin tespiti
Uyarı Sistemi	Mail ile bilgilendirme, hata takibi ve müdahale mekanizması

Yedekleme otomasyonu, **disaster recovery** senaryolarında kritik rol oynar. Ayrıca, yedeklerin **uzak sunucuya kopyalanması**, **şifrelenmesi** ve **versiyonlaması** gibi ileri düzey adımlar da güvenlik için önerilir.