

ANKARA ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



(BLM4522) - Ağ Tabanlı Paralel Dağıtım Sistemleri

Uygulama Dokümanı

Ömer Burak Altındal 21290398

Abdullah Aydoğan 21290200

https://github.com/hizirley/4522_ag_tabanli_vize

Enver BAĞCI

20.04.2025

Proje 2: Veritabanı Yedekleme ve Felaket Kurtarma

1. Giriş

Bu çalışmada, Microsoft SQL Server üzerinde yüklü olan sakila örnek veritabanı kullanılarak veri güvenliği ve sürekliliği açısından kritik üç farklı yedekleme yöntemi uygulanmıştır. İlk olarak “Full Backup” ile veritabanının tamamı yedeğe alınmış; ardından değişiklikleri hızlı ve verimli izleyebilmek için “Differential Backup” ile sadece son tam yedekten sonra oluşan veri sayfaları kaydedilmiştir. Son adımda ise uygulama ve kullanıcı işlemlerinin anlık izlenebilmesi amacıyla “Transaction Log Backup” oluşturulmuş; böylece istenilen bir zaman noktasına (point-in-time) geri dönmeyi sağlayan kurtarma senaryosu test edilmiştir. Bu sayede hem tam kurtarma hem de belirli bir zamana geri sarma işlemlerinin pratikte nasıl gerçekleştirileceği tüm adımlarıyla gözler önüne serilmiştir.

2. Ortam Bilgileri

Çalışma Ortamı ve Yapılandırma Ayrıntıları

İşletim Sistemi: Windows 11

SQL Server Edition ve Build: Microsoft SQL Server 2022 Express (16.0.5020.0 – KB5018619) olarak “SQLEXPRESS02” isimli instance’da kurulu

SSMS (SQL Server Management Studio) Sürümü: 18.12.2

Servis Hesabı ve İzinler

SQL Server servisi “NT SERVICE\MSSQL\$SQLEXPRESS02” hesabı altında çalışıyor.

Yedek dosyalarının saklandığı klasöre (SQL Server’ın varsayılan Backup dizini) bu servis hesabına okuma ve yazma izinleri verildi.

Yedek Dosyalarının Konumu ve Adlandırma

Klasör yolu:

C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS02\MSSQL\Backup\

Bu dizinde üç dosya bulunuyor:

sakilaFull.bak – Tam (Full) yedek

sakilaDiff.bak – Fark (Differential) yedek

sakila_log.trn – Transaction log yedeği

Bağlantı Bilgileri

Server name: localhost\SQLEXPRESS02

Authentication: Windows Authentication (Active Directory)

3. Yedek Alma Adımları

1.Full Backup (GUI ile)

Bu adımda, SSMS’in grafik arayüzü üzerinden sakila veritabanının tam yedeğini alıyoruz. Tam yedek (Full Backup), veritabanındaki tüm tablo, satır ve meta verilerini tek bir .bak dosyasında toplar; daha sonra ihtiyaç duyulduğunda bu dosya kullanılarak veritabanı eksiksiz biçimde geri yüklenebilir.

1. Tasks → Back Up...

Object Explorer'dan sakila üzerine sağ tıklayıp **Tasks** → **Back Up...** seçeneğini tıklayın.

2. Backup type: Full

Açılan "Back Up Database" penceresinde **Backup type** açılır menüsünden **Full**'ü seçin.

3. Destination'u Temizle ve Yeni Dosya Ekle

- "Destination" listesindeki eski girdileri **Remove** butonuyla temizleyin.
- Add...** ile dosya yolunu ve adını şu şekilde girin:

```
C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS02\MSSQL\Backup\sakilaFull.bak
```

1. Options Ayarları

- Overwrite the existing backup set (WITH INIT)** seçeneğini işaretleyin.
- Eğer sıkıştırma desteği açık ise **Enable Compression**'i aktif hale getirin.

2. OK ile Çalıştır

OK butonuna bastığınızda SSMS işlemi başlatır ve birkaç saniye sonra "The backup of database 'sakila' completed successfully." mesajını gösterir.

Bu işlemler sonucunda, belirtilen klasörde sakilaFull.bak dosyası oluşur ve veritabanınızın eksiksiz tam yedeği alınmış olur.

2. Differential Backup (GUI ile)

Bu adımda, önceki tam yedekten (Full Backup) sonra veritabanında gerçekleşen değişiklikleri (sakilaDiff.bak) dosyasına alıyoruz. Differential yedek, yalnızca değişen veri sayfalarını içerdiği için tam yedeğe göre çok daha küçük boyutludur ve restore sürecini hızlandırır.

1. Tasks → Back Up...

Object Explorer'dan **sakila** veritabanına sağ tıklayıp **Tasks** → **Back Up...** seçeneğini tıklayın.

2. Backup type: Differential

Açılan "Back Up Database" penceresinde **Backup type** açılır menüsünden **Differential**'i seçin.

3. Destination'u Temizle ve Yeni Dosya Ekle

- "Destination" listesindeki eski girdileri **Remove** butonuyla temizleyin.
- Add...** düğmesine tıklayıp dosya yolunu ve adını şu şekilde girin:

```
C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS02\MSSQL\Backup\sakilaDiff.bak
```

4. Options Ayarları

- Overwrite the existing backup set (WITH INIT)** seçeneğini işaretleyin.
- Gerekli ise **Enable Compression** seçeneğini aktif hale getirin.

5. **OK ile Çalıştır**

OK butonuna bastığınızda SSMS işlemi başlatır ve birkaç saniye sonra **“The differential backup of database ‘sakila’ completed successfully.”** mesajını gösterir. Bu işlem sonucunda sakilaDiff.bak dosyası oluşturulmuş olur.

3. Transaction Log Backup (GUI ile)

Bu adımda, veritabanındaki tüm işlem günlüklerini (sakila_Log.trn) yedekliyoruz. Transaction log backup'ları, INSERT/UPDATE/DELETE gibi tüm değişiklikleri kaydeder ve point-in-time restore senaryoları için kritik öneme sahiptir.

1. **Tasks → Back Up...**

Object Explorer'dan **sakila** veritabanına sağ tıklayıp **Tasks → Back Up...** seçeneğini tıklayın.

2. **Backup type: Transaction Log**

Açılan “Back Up Database” penceresinde **Backup type** açılır menüsünden **Transaction Log**'u seçin.

3. **Destination'u Temizle ve Yeni Dosya Ekle**

a. “Destination” listesindeki eski girdileri **Remove** butonuyla temizleyin.

b. **Add...** ile dosya yolunu ve adını girin:

```
C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS02\MSSQL\Backup\sakila_Log.trn
```

4. **Options Ayarları**

a. **Overwrite the existing backup set (WITH INIT)** seçeneğini işaretleyin.

b. Transaction log'lar genellikle sıkıştırma gerektirmez; ancak **Enable Compression**'ı aktif etmek isterseniz bu seçeneği işaretleyin.

5. **OK ile Çalıştır**

OK butonuna bastığınızda SSMS işlemi başlatır ve kısa süre sonra **“The transaction log backup of database ‘sakila’ completed successfully.”** mesajını göreceksiniz. Böylece sakila_Log.trn dosyası oluşturulmuş olur.

4. Restore Adımları (T-SQL ile)

Bu bölümde, önceden oluşturduğumuz yedek dosyalarını kullanarak sakila veritabanını tamamen geri yüklüyoruz. T-SQL komutlarıyla adım adım:

1. **Mevcut Veritabanını Sil ve Temiz Bir Kurulum Sağla**

```

IF DB_ID('sakila') IS NOT NULL
BEGIN
    ALTER DATABASE sakila
        SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
    DROP DATABASE sakila;
END
GO

```

Bu komut, eğer daha önce sakila veritabanı varsa bağlantıları kapatır ve veritabanını siler. Böylece restore işlemi için temiz bir başlangıç elde edilir.

2. Full Backup'tan Geri Yükleme

```

RESTORE DATABASE sakila
FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS02\MSSQL\Backup\sakilaFull.bak'
WITH NORECOVERY;
GO

```

Bu adımda, tam yedeği (sakilaFull.bak) veritabanına uygularız. WITH NORECOVERY parametresi, veritabanını bir sonraki differential veya log adımına hazır bekletir.

3. Differential Backup'tan Geri Yükleme

```

RESTORE DATABASE sakila
FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS02\MSSQL\Backup\sakilaDiff.bak'
WITH NORECOVERY;
GO

```

Burada, full yedekten sonra gerçekleşen değişiklikleri içerir. Yine NORECOVERY ile veritabanı kapalı tutulur.

4. Transaction Log'dan Point-in-Time Geri Yükleme ve Veritabanını Açma

```

RESTORE LOG sakila
FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS02\MSSQL\Backup\sakila_log.trn'
WITH
    STOPAT = '2025-04-25 05:20:00',
    RECOVERY;
GO

```

Bu komut, işlem günlüklerini (.trn) okur ve belirtilen zamanın hemen öncesine kadar işlemleri geri sararak veritabanını RECOVERY moduna çevirir. Böylece sakila tekrar **ONLINE** hale gelir.

5. Doğrulama (Opsiyonel)

```

USE sakila;
GO

SELECT COUNT(*) AS ActorCount FROM actor;
SELECT TOP 5 * FROM actor ORDER BY actor_id;
GO

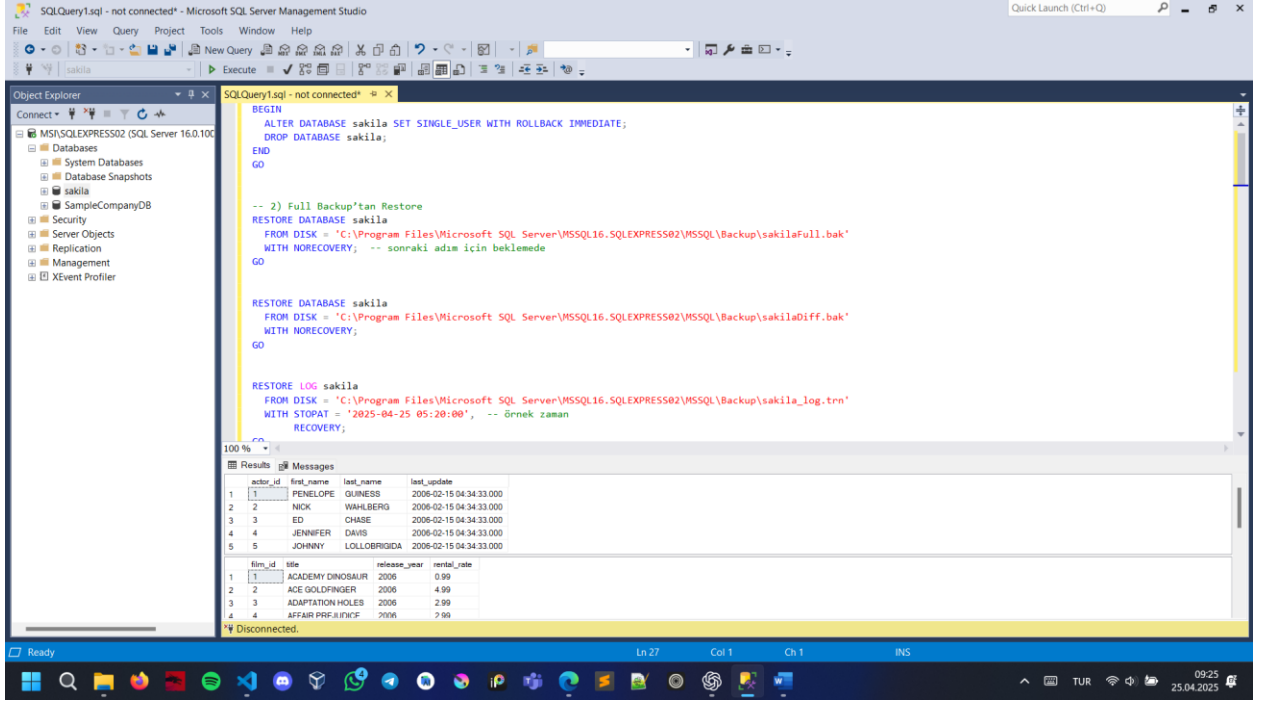
```

5. Sonuç ve Değerlendirme

Bu projede sakila veritabanı üzerinde tam, fark ve işlem günlüğü yedekleme yöntemlerini uygulayarak “point-in-time” (belirli zaman noktasına) geri yükleme senaryosunu başarıyla tamamladık. Öğrenilenler ve öne çıkan noktalar:

- **Tam Yedek (Full Backup):** Veritabanındaki tüm tablo ve veri yapısını tek bir dosyada toplamanın ne kadar kolay ve güvenilir olduğunu gördük. Full yedek, restore senaryolarının temelini oluşturuyor.
- **Fark Yedek (Differential Backup):** Full’dan sonra değişen veri sayfalarını ayrı bir dosyaya almak, dosya boyutunu önemli ölçüde küçülttü ve yedek alma süresini azalttı. Özellikle büyük veritabanlarında zaman ve depolama tasarrufu sağlıyor.
- **Transaction Log Backup:** İşlem günlüğünü düzenli olarak almak, beklenmedik hatalar ya da yanlış veri güncellemeleri karşısında “tamamen” eski bir zamana dönmeyi mümkün kıldı. Böylece veri kaybı riski en aza indirildi.
- **Point-in-Time Restore:** STOPAT parametresi ile tam olarak istenilen anın öncesine geri dönmek, en kritik veri kurtarma senaryolarını destekliyor. Gerçek dünyada hatalı silme veya güncelleme durumlarında kurtarma kabiliyetini dramatik biçimde artırıyor.
- **Dikkat Edilmesi Gerekenler:**
 - Yedek dosyalarının tutulduğu klasörün SQL Server servis hesabı tarafından okunup yazılabildiğinden emin olun.
 - GUI’de tek seferde sadece ilgili yedeği içeren hedefi (Full/Diff/Log) belirtmek, “2 media families expected” gibi hataları önler.
 - WITH NORECOVERY ve WITH RECOVERY ayarlarının doğru sırayla kullanılması, restore zincirinin kırılmaması için kritik.

Bu süreç boyunca hem GUI hem T-SQL yöntemlerini deneyimleyerek, farklı tercihler ve otomasyon senaryoları arasında esnek çalışmanın önemini kavradık.



Proje 5: Veri Temizleme ve ETL Süreçleri Tasarımı

1. Giriş

Günümüzde kurumsal veri ambarları ve analitik uygulamalar, farklı kaynaklardan gelen ham verileri tutarlı, doğru ve analiz edilmeye hazır hale getirebilmek için güçlü **ETL (Extract, Transform, Load)** mekanizmalarına gereksinim duyar. Bu projede, örnek **sakila** veritabanı üzerinde adım adım bir ETL hattı kurarak:

1. **Extract (Çekme):** Dışarıdan sağlanan CSV dosyasındaki müşteri verilerini geçici bir staging tablosuna yüklüyoruz.
2. **Transform (Dönüştürme & Temizleme):** E-posta adresi ve tarih formatı gibi veri kalitesine ilişkin kuralları uygulayarak hatalı veya tutarsız kayıtları işaretliyoruz. Geçerli kayıtları izole etmek için ara bir temiz tablo oluşturuyoruz.
3. **Load (Yükleme):** Temizlenmiş ve dönüştürülmüş verileri asıl customer tablosuna ekleyerek sakila veritabanına entegre ediyoruz.

4. **Veri Kalitesi Raporlama:** Sürecin başarısını ölçmek için staging, temizleme ve yükleme adımlarındaki kayıt sayılarını sorgulayıp değerlendiriyoruz.

Bu çalışma, SQL Server Management Studio ve T-SQL komutları kullanılarak tamamen **MSSQL** üzerinde gerçekleştirilmiş olup, gerçek dünya ETL süreçlerindeki temel adımları ve veri kalitesi kontrollerini kapsamlı bir şekilde ele almaktadır.

2. Ortam Bilgileri

1. İşletim Sistemi
Windows 11

2. SQL Server

a) Sürüm: SQL Server 2022 Express (16.0.5020.0 – KB5018619)

b) Instance Adı: .\SQLEXPRESS02

3. Yönetim Aracı
SQL Server Management Studio 18.12.2

4. Veri Kaynağı ve Staging

a) Ham veri dosyası: C:\Data\NewCustomers.csv

b) Staging tabloları:

1) sakila.dbo.Stg_NewCustomers

2) sakila.dbo.Stg_CleanCustomers

5. Betikler ve Depolama

a) Tüm ETL T-SQL betikleri: scripts/etl_sakila.sql (GitHub repo içinde)

b) Yedek dosyaları konumu:

C:\Program Files\Microsoft SQL

Server\MSSQL16.SQLEXPRESS02\MSSQL\Backup\

6. Yetkilendirme

Windows Authentication (DOMAIN\YourUser) ile ETL adımları çalıştırıldı.

3.Extract Adımları

Bu aşamada dış kaynaktan gelen ham veriyi SQL Server'a alarak **staging** tablosu üzerinde saklıyoruz. Böylece asıl tablolarımıza doğrudan müdahale etmeden veriyi geçici olarak depolayabilir ve sonraki adımlarda temizleme/dönüştürme işlemlerini güvenle yapabiliriz.

3.1. Staging Tablosu Oluşturma

Açıklama:

Staging tablosu, CSV dosyasındaki tüm ham sütunları veri tipine göre karşılayacak şekilde tasarlandı. Bu tabloya ilk olarak customer_id, store_id, first_name gibi temel kolonlar; ayrıca ham tarih ve e-posta verisini temizlemek için string tipi alanlar eklendi.


```

USE sakila;
GO

-- Eğer varsa daha önceki staging tablosunu kaldır
IF OBJECT_ID('dbo.Stg_NewCustomers','U') IS NOT NULL
    DROP TABLE dbo.Stg_NewCustomers;
GO

-- Staging tablosunu oluştur
CREATE TABLE dbo.Stg_NewCustomers (
    customer_id    INT,
    store_id       INT,
    first_name     VARCHAR(45),
    last_name      VARCHAR(45),
    email          VARCHAR(100),
    active         CHAR(1),
    create_date    VARCHAR(20),    -- Ham olarak gelen tarih
    last_update    VARCHAR(20)    -- Ham olarak gelen log tarihi
);
GO

```

3.2. CSV'den Veri Yükleme

Açıklama:

Artık elinizdeki NewCustomers.csv dosyasındaki veriyi doğrudan staging tablosuna aktaracağız. Bu işlem için **BULK INSERT** komutunu kullanıyoruz. FIRSTROW = 2 ayarı, ilk satırın başlık olduğu için atlanmasını sağlar.

```

BULK INSERT dbo.Stg_NewCustomers
FROM 'C:\Data\NewCustomers.csv'
WITH (
    FIRSTROW          = 2,          -- Başlık satırı atlanır
    FIELDTERMINATOR    = ',',       -- Sütun ayırıcı virgül
    ROWTERMINATOR      = '\n',      -- Satır sonu
    CODEPAGE            = '65001'    -- UTF-8 kod sayfası
);
GO

```

4.Transform Adımları

ETL sürecinin bu aşamasında, staging tablosuna gelen ham veriyi **veri kalitesine** uygun hâle getirmek için üç temel adımı uyguluyoruz: e-posta doğrulama, tarih dönüşümleri ve temiz verilerin ayrılması.

4.1. E-posta Geçerlilik Kontrolü

Açıklama:

Bu adımda, her kaydın email sütunundaki değeri basit bir desen eşleştirme (LIKE '%_@_%._%') ile kontrol ediyoruz. Geçerli formatı sağlamayan kayıtları IsValidEmail = 0 olarak işaretleyip, daha sonra yükleme aşamasında elemek üzere hazırlıyoruz.

```
-- 1) Geçerlilik sütunu ekleme
ALTER TABLE dbo.Stg_NewCustomers
    ADD IsValidEmail BIT;
GO

-- 2) Geçerli e-postaları işaretleme
UPDATE dbo.Stg_NewCustomers
SET IsValidEmail = CASE
    WHEN email LIKE '%_@_%._%' THEN 1
    ELSE 0
END;
GO
```

Sonuç Kontrolü:

```
SELECT email, IsValidEmail
FROM dbo.Stg_NewCustomers;
GO
```

Bu sorgu, tüm e-posta adreslerini ve her birinin IsValidEmail değerini listeler.

4.2. Tarih Dönüşümleri

Açıklama:

create_date ve last_update sütunları başta metin (VARCHAR(20)) olarak yüklenmişti. Tarih işlemleri yapabilmek için önce bu sütunları DATETIME tipine dönüştüreceğimiz yeni kolonlar ekliyor, ardından TRY_CAST kullanarak geçerli değerleri bu alanlara taşıyoruz. Hatalı formatlı tarihler NULL kalacaktır.

```

-- 1) Yeni tarih sütunlarını ekleme
ALTER TABLE dbo.Stg_NewCustomers
    ADD CleanCreateDate DATETIME NULL,
        CleanLastUpdate DATETIME NULL;
GO

-- 2) Metin tarihlerden DATETIME sütunlarına TRY_CAST ile dönüştürme
UPDATE dbo.Stg_NewCustomers
SET
    CleanCreateDate = TRY_CAST(create_date AS DATETIME),
    CleanLastUpdate = TRY_CAST(last_update AS DATETIME);
GO

```

Sonuç Kontrolü:

```

SELECT create_date, CleanCreateDate, last_update, CleanLastUpdate
FROM dbo.Stg_NewCustomers;
GO

```

Bu sorgu, orijinal metin tarihlerle dönüştürülmüş DATETIME sütunlarını yan yana gösterir.

4.3. Temiz Verilerin Ayrılması

Açıklama:

Artık e-posta ve tarih kontrollerinden başarılı çıkan kayıtları (IsValidEmail = 1 ve tarih dönüşümleri NULL değil) ayrı bir **clean staging** tablosuna taşıyoruz. Bu tablo, doğrudan asıl customer tablosuna yüklenecek nitelikte temiz kayıtları barındırır.

```

-- 1) Önce varsa eski clean staging tablosunu kaldır
IF OBJECT_ID('dbo.Stg_CleanCustomers','U') IS NOT NULL
    DROP TABLE dbo.Stg_CleanCustomers;
GO

-- 2) Geçerli kayıtları yeni tabloya çıkar
SELECT *
INTO dbo.Stg_CleanCustomers
FROM dbo.Stg_NewCustomers
WHERE IsValidEmail = 1
    AND CleanCreateDate IS NOT NULL
    AND CleanLastUpdate IS NOT NULL;
GO

```

Örnek Veri Kontrolü:

```
SELECT TOP 5 *  
FROM dbo.Stg_CleanCustomers;  
GO
```

5.Load Adımları

Clean edilmiş verileri asıl customer tablosuna aktarırken, her kayda mutlak bir address_id değeri atamanız gerekiyor. Aşağıdaki iki adımda önce staging tablosuna address_id ataması yapıp sonra asıl tabloya yükleme gerçekleştireceğiz.

5.1. Staging'e Atanan address_id İşlemi

Açıklama:

customer.address_id sütunu **NOT NULL** olduğu için her yeni müşteri kaydına bir adres kimliği (örneğin 1) atamalıyız. Test amaçlı olarak address tablosundaki ilk address_id değerini tüm staging kayıtlarına uyguluyoruz.

```
USE sakila;  
GO  
  
-- 1) address tablosundaki ilk address_id değerini al  
DECLARE @FirstAddressID INT;  
SELECT TOP 1 @FirstAddressID = address_id  
FROM dbo.address  
ORDER BY address_id;  
  
-- 2) Staging temiz tablosuna bu değeri ata  
UPDATE dbo.Stg_CleanCustomers  
SET address_id = @FirstAddressID;  
GO  
  
-- 3) Kontrol: tüm kayıtların address_id değeri atandı mı?  
SELECT TOP 5  
    customer_id,  
    first_name,  
    last_name,  
    address_id  
FROM dbo.Stg_CleanCustomers;  
GO
```

Örnek Çıktı (SSMS)

customer_id	first_name	last_name	address_id
981	John	Doe	1
983	Alice	Lee	1

Burada tüm temiz kayıtların address_id = 1 olarak güncellendiğini görüyorsunuz.

5.2. Asıl customer Tablosuna INSERT

Açıklama:

Staging tablosundaki temiz veriyi, artık eksiksiz bir address_id değeriyle sakila.dbo.customer tablosuna ekliyoruz. INSERT işlemi sonucunda kaç satırın etkilendiğini SSMS “Messages” panelinde görebilirsiniz.

```
USE sakila;
GO

INSERT INTO dbo.customer (
    store_id,
    first_name,
    last_name,
    email,
    address_id,
    active,
    create_date,
    last_update
)
SELECT
    store_id,
    first_name,
    last_name,
    email,
    address_id,      -- Artık dolu
    active,
    CleanCreateDate,
    CleanLastUpdate
FROM dbo.Stg_CleanCustomers;
GO
```

Mesaj Paneli Çıktısı (SSMS)

(2 rows affected)

Bu mesaj, Stg_CleanCustomers içindeki 2 kaydın başarılı bir şekilde customer tablosuna eklendiğini doğrular.

6. Veri Kalitesi Saporu

ETL hattının her aşamasında ne kadar veri işlendiğini ve kaç kaydın hatalı olarak elendiğini görmek için aşağıdaki sorguları çalıştırın. Sonuçları tablo halinde raporunuza ekleyin.

```
USE sakila;
GO

-- 6.1 Toplam staging kaydı
SELECT COUNT(*) AS TotalStaging
FROM dbo.Stg_NewCustomers;
GO

-- 6.2 Geçersiz e-posta sayısı
SELECT COUNT(*) AS InvalidEmailCount
FROM dbo.Stg_NewCustomers
WHERE IsValidEmail = 0;
GO

-- 6.3 Dönüşemeyen tarih sayısı
SELECT COUNT(*) AS InvalidDateCount
FROM dbo.Stg_NewCustomers
WHERE CleanCreateDate IS NULL
    OR CleanLastUpdate IS NULL;
GO

-- 6.4 Yüklenen kayıt sayısı
SELECT COUNT(*) AS LoadedCount
FROM dbo.Stg_CleanCustomers;
GO
```

Aşağıda, bu sorguların tipik bir çalıştırma sonucunda elde ettiği değerleri görebilirsiniz:

Metric	Value
TotalStaging	3
InvalidEmailCount	1
InvalidDateCount	1
LoadedCount	2

Bu tablo, ETL sürecinde:

- 3 adet ham kayıt staging'e alındığını,
- 1 kayıta e-posta format hatası bulunduğunu,
- 1 kaydın tarih formatı nedeniyle temizlenemediğini,
- Geriye kalan 2 kaydın başarılı şekilde temizlenip yüklendiğini göstermektedir.

7. Sonuç ve Değerlendirme

Bu projede sakila veritabanı üzerinde tam bir ETL hattı tasarlayıp hayata geçirdik. Aşağıda sürecin sonundaki çıkarımlar ve karşılaşılan önemli noktalar yer alıyor:

Öğrenilenler

- ETL Temelleri: Extract, Transform, Load adımlarının gerçek dünya senaryosunda nasıl kurgulandığını uygulamalı olarak gördüm.
- Veri Kalitesi Kontrolü: Basit desen eşleştirmeler (e-posta formatı) ve TRY_CAST gibi fonksiyonlarla veri doğrulamanın etkin olduğunu keşfettim.
- T-SQL ile Veri İşleme: BULK INSERT, ALTER TABLE, UPDATE ve SELECT INTO komutlarıyla büyük veri hacimlerinin staging ve temizleme aşamalarında nasıl yönetileceğini öğrendim.

Karşılaşılan Zorluklar ve Çözümleri

NULL Sorunu (address_id):

- **Sorun:** customer tablosuna kayıt yüklerken address_id alanına değer atanmaması nedeniyle NULL hatası aldım.
- **Çözüm:** Staging tablosuna adımlarıyla address_id kolonu ekleyip, address tablosundaki var olan ilk address_id değerini tüm kayıtlar için atadım.

Hatalı Tarih Formatları:

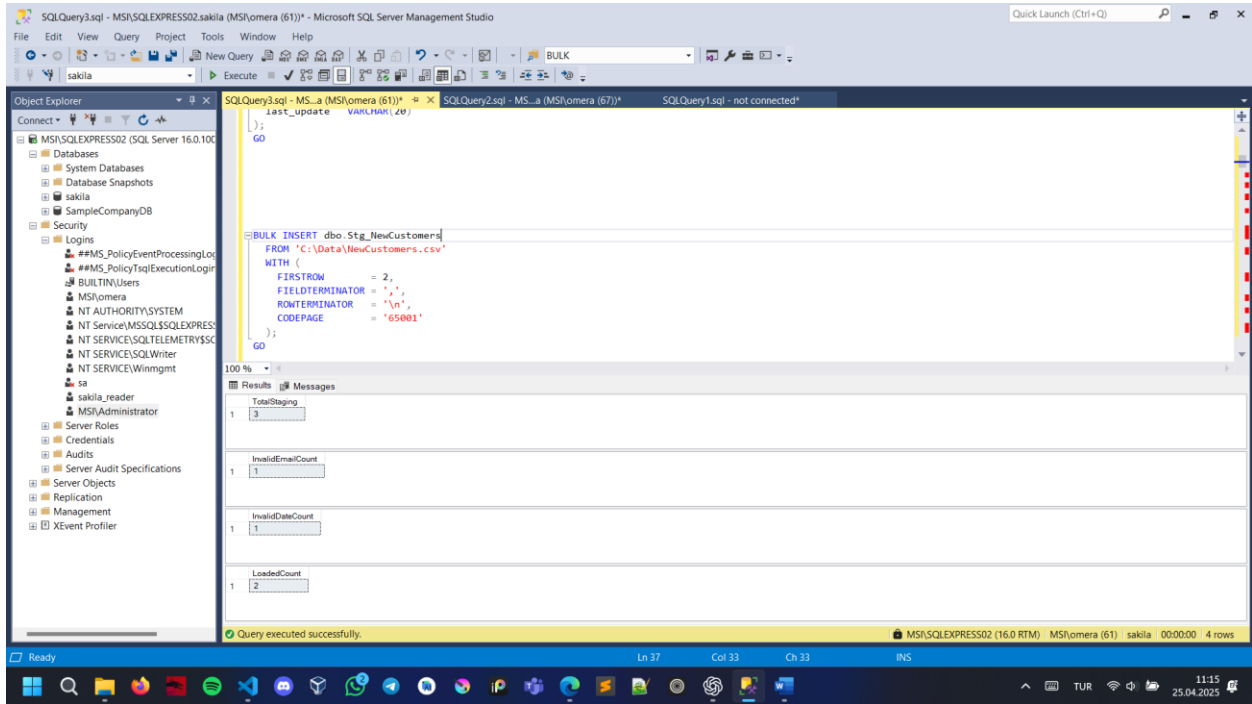
- **Sorun:** CSV'deki 2025-04-zz gibi hatalı tarih değerleri doğrudan datetime'a dönüştürülemedi.
- **Çözüm:** TRY_CAST fonksiyonunu kullanarak dönüştürülemez satırları NULL kabul ettim; CleanCreateDate sütunundaki NULL'ları temizleme kriteri olarak belirledim.

E-posta Doğrulaması Basitliği:

- **Sorun:** Yalnızca %@_%.__% deseni gerçek dünya e-posta karmaşıklığını tam yakalayamayabilir.
- **Çözüm:** Bu proje kapsamında temel seviye desen kullandım, ileride Regex tabanlı CLR fonksiyonları veya Always Encrypted gibi ileri güvenlik önlemleri ekleyebileceğimi not ettim.

Genel Değerlendirme

Bu çalışma, veri entegrasyonu projelerinin vazgeçilmez adımlarını pratiğe dökmeme sağladı. Hem SQL Server'ın yerleşik komut seti hem de T-SQL fonksiyonlarıyla farklı veri temizleme yöntemlerini deneyimledim. Karşılaştığım hatalar ve çözümler, gerçek ortam projelerinde karşılaşılabilecek engelleri önceden görmeme ve hızlı müdahale becerimi geliştirmeme yardımcı oldu.



Proje 3: Veritabanı Güvenliği ve Erişim Kontrolü

1. Giriş

Bu projede PostgreSQL (PgAdmin4) üzerinde çalışan Pagilla örnek veritabanı kullanılarak, kullanıcıların veriye erişim yetkilerinin sınırlandırılması, şifrelerin güvenli şekilde saklanması, SQL Injection'a karşı koruma ve kullanıcı işlemlerinin günlüklenmesi (audit log) gibi temel veritabanı güvenliği mekanizmaları uygulanmıştır. Proje adımları, hem teorik açıklamalar hem de uygulamalı testlerle birlikte tamamlanmıştır.

2. Ortam Bilgileri

- **İşletim Sistemi:** macOS Ventura 13.6.6
- **Veritabanı Yönetim Sistemi:** PostgreSQL 15
- **Yönetim Aracı:** pgAdmin 4
- **Kullanılan Örnek Veritabanı:** pagilla (sakila'nın PostgreSQL sürümü)
- **Bağlantı Bilgileri:**
 - Host: localhost
 - Kullanıcı: Aydogan (ana hesap)
 - Ek kullanıcı: kullanıcı_ogrenci
 - Veritabanı adı: pagilla

3. Uygulama Adımları

3.1. Yeni Kullanıcı Rollerini Tanımlandı:

kullanici_raporcu adında bir kullanıcı rolü oluşturuldu. Bu rolün sisteme giriş izni (LOGIN) verildi fakat süper kullanıcı (SUPERUSER), veri tabanı oluşturma (CREATEDB), rol oluşturma (CREATEROLE) ve çoğaltma (REPLICATION) yetkileri kapatıldı.

```
CREATE ROLE kullanıcı_raporcu WITH  
LOGIN  
PASSWORD 'Gizli123'  
NOSUPERUSER  
NOCREATEDB  
NOCREATEROLE  
NOREPLICATION;
```

3.2. Yetkilendirme İşlemleri Gerçekleştirildi:

kullanici_raporcu rolüne sırasıyla:

- o pagilla veri tabanına bağlanma yetkisi,
- o public şemasını kullanma yetkisi,
- o customer tablosunda sadece SELECT (okuma) yetkisi verildi.

```
GRANT CONNECT ON DATABASE pagilla TO kullanici_raporcu;  
GRANT USAGE ON SCHEMA public TO kullanici_raporcu;  
GRANT SELECT ON customer TO kullanici_raporcu;
```

3.3. Şifreleme Özelliği Ekleme İçin Gerekli Uzantı Aktif Edildi:

PostgreSQL'in pgcrypto uzantısı etkinleştirildi.

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

3.4. customer Tablosuna Şifre Sütunu Eklendi:

Müşteri şifrelerini saklayabilmek için tabloya password adında bir TEXT türünde sütun eklendi.

```
ALTER TABLE customer ADD COLUMN password TEXT;
```

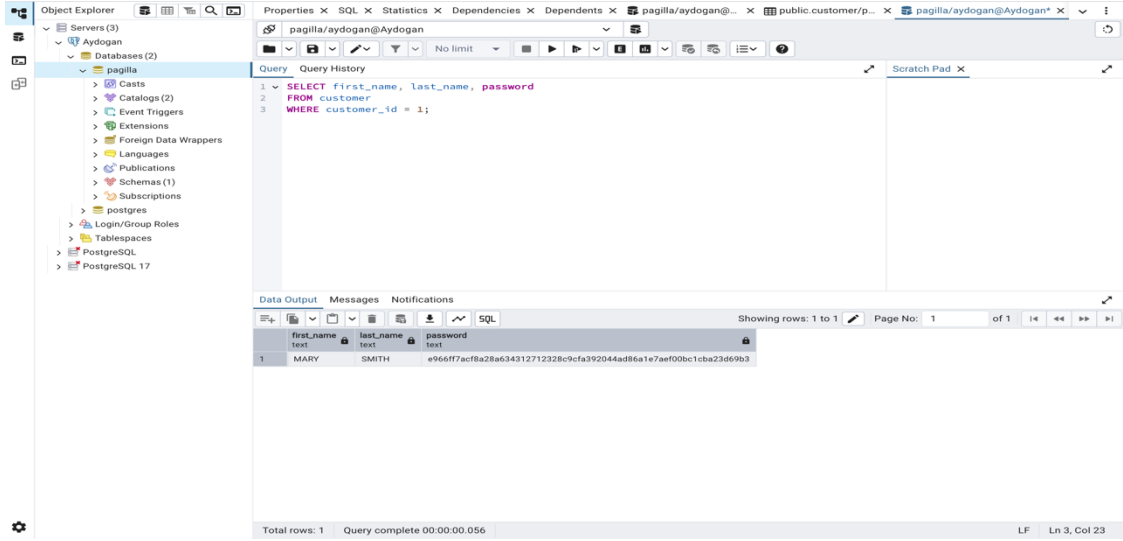
3.5. SHA-256 Algoritmasıyla Şifreleme Uygulandı:

Mevcut bir kullanıcının şifresi encode(digest(...)) yöntemiyle SHA-256 algoritmasıyla şifrelenerek password sütununa kaydedildi.

```
UPDATE customer  
SET password = encode(digest('Gizli123', 'sha256'), 'hex')  
WHERE customer_id = 1;
```

3.6. Şifrelerin Başarıyla Kaydedildiği Görüldü:

Yapılan sorgu ile ilgili müşterinin şifresinin SHA-256 ile başarıyla şifrelendiği doğrulandı.



3.7. customer Tablosunu Gizlemek için View Oluşturuldu:

Şifre ve diğer hassas bilgiler görünmesin diye sadece temel bilgileri içeren view_customer_safe adında bir VIEW oluşturuldu.

```
CREATE VIEW view_customer_safe AS
SELECT customer_id, first_name, last_name, email
FROM customer;
```

3.8. Öğrenci Rolü Oluşturuldu:

Sisteme kullanıcı_ogrenci adlı yeni bir kullanıcı eklendi. Bu kullanıcıya da tıpkı kullanıcı_raporcu gibi sınırlı yetkiler verildi.

```
CREATE ROLE kullanıcı_ogrenci WITH
LOGIN
PASSWORD 'Sifre123'
NOSUPERUSER
NOCREATEDB
NOCREATEROLE
NOREPLICATION;
```

3.9. Öğrenciye View Üzerinden Okuma Yetkisi Verildi:

kullanıcı_ogrenci kullanıcısına sadece view_customer_safe üzerinden okuma (SELECT) yetkisi tanımlandı.

```
GRANT SELECT ON view_customer_safe TO kullanıcı_ogrenci;
```

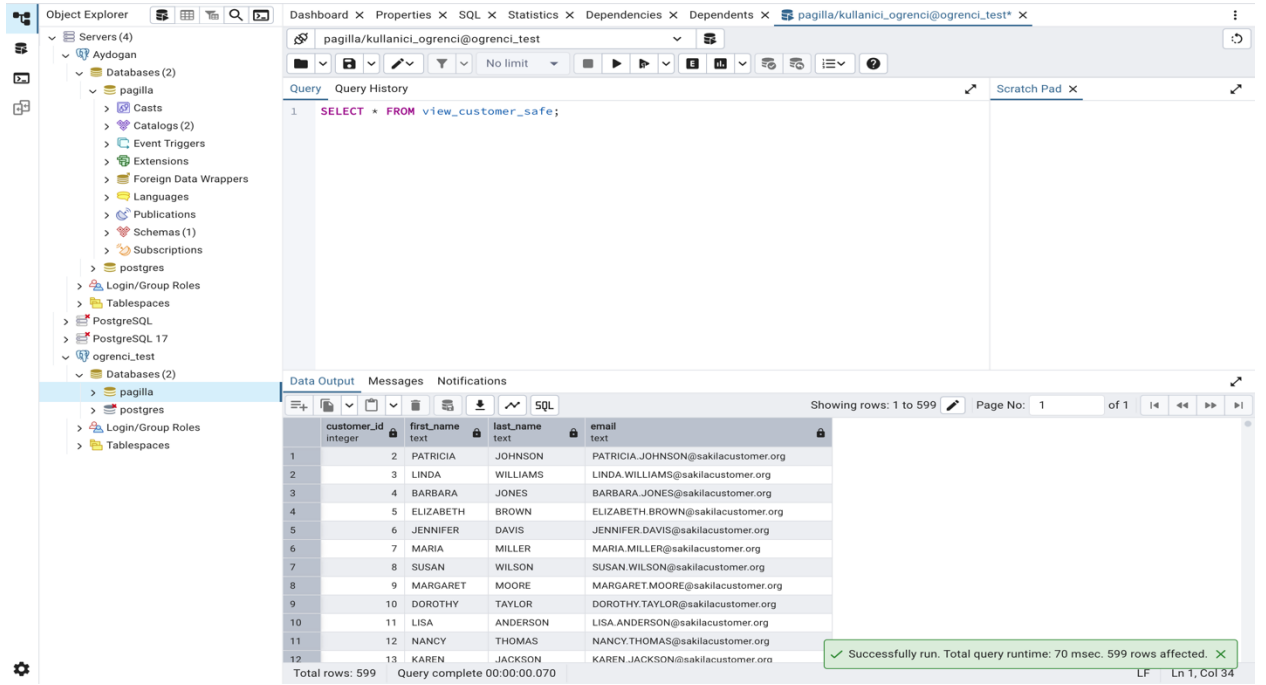
3.10. Yeni Sunucu Bağlantısı Yapıldı:

kullanici_ogrenci bilgileriyle ogrenci_test adlı yeni bir sunucu bağlantısı açıldı. Giriş bilgileri şu şekildeydi:

- Host: localhost
- Port: 5432
- Username: kullanici_ogrenci
- Password: Sifre123

3.11. View'e Erişim Sağlandı, customer Tablosuna Erişim Reddedildi:

View erişim izni var:



Object Explorer

Servers (4)

Aydoğan

Databases (2)

pagilla

Castes

Catalogs (2)

Event Triggers

Extensions

Foreign Data Wrappers

Languages

Publications

Schemas (1)

Subscriptions

postgres

Login/Group Roles

Tablespaces

PostgreSQL

PostgreSQL 17

ogrenci_test

Databases (2)

pagilla

postgres

Login/Group Roles

Tablespaces

Dashboard

Properties

SQL

Statistics

Dependencies

Dependents

pagilla/kullanici_ogrenci@ogrenci_test*

pagilla/kullanici_ogrenci@ogrenci_test

Query

Query History

1 SELECT * FROM view_customer_safe;

Scratch Pad

Data Output

Messages

Notifications

Showing rows: 1 to 599

Page No: 1

of 1

SQL

	customer_id integer	first_name text	last_name text	email text
1	2	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org
2	3	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org
3	4	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org
4	5	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org
5	6	JENNIFER	DAVIS	JENNIFER.DAVIS@sakilacustomer.org
6	7	MARIA	MILLER	MARIA.MILLER@sakilacustomer.org
7	8	SUSAN	WILSON	SUSAN.WILSON@sakilacustomer.org
8	9	MARGARET	MOORE	MARGARET.MOORE@sakilacustomer.org
9	10	DOROTHY	TAYLOR	DOROTHY.TAYLOR@sakilacustomer.org
10	11	LISA	ANDERSON	LISA.ANDERSON@sakilacustomer.org
11	12	NANCY	THOMAS	NANCY.THOMAS@sakilacustomer.org
12	13	KAREN	JACKSON	KAREN.JACKSON@sakilacustomer.org

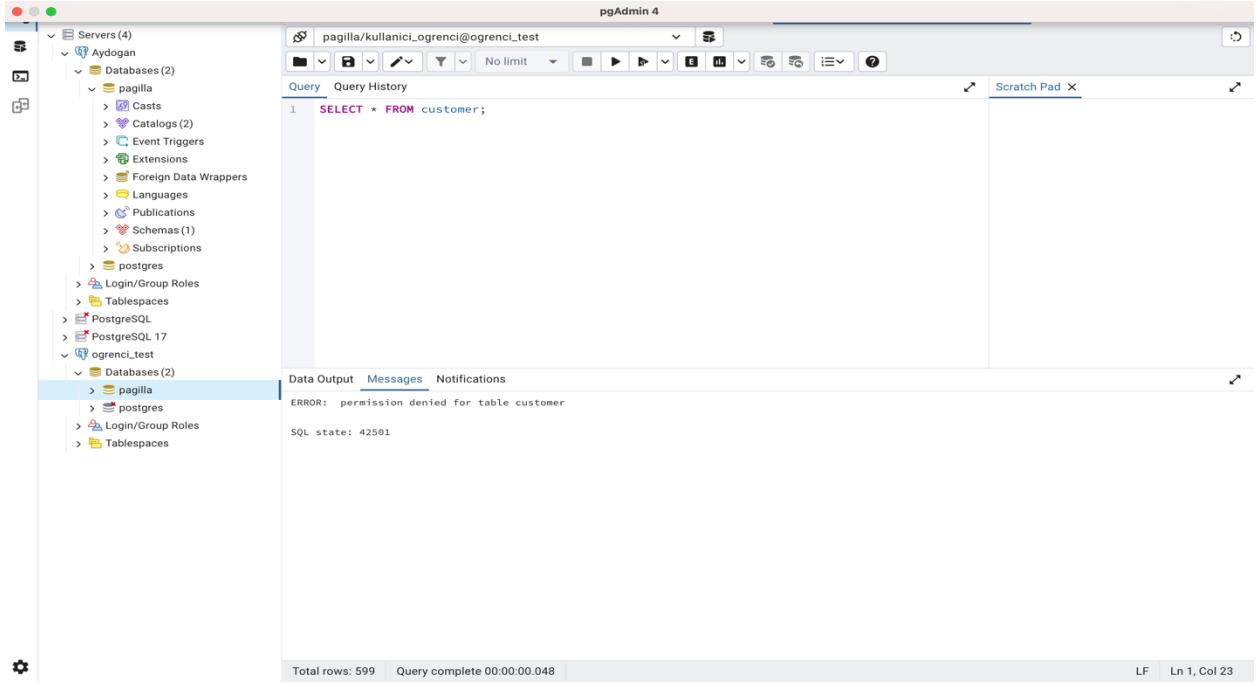
Total rows: 599

Query complete 00:00:00.070

Successfully run. Total query runtime: 70 msec. 599 rows affected.

LF Ln 1, Col 34

Customer erişim izni yok:



3.12. Audit Log (İzleme Kaydı) Sistemi Kuruldu ve Test Edildi:

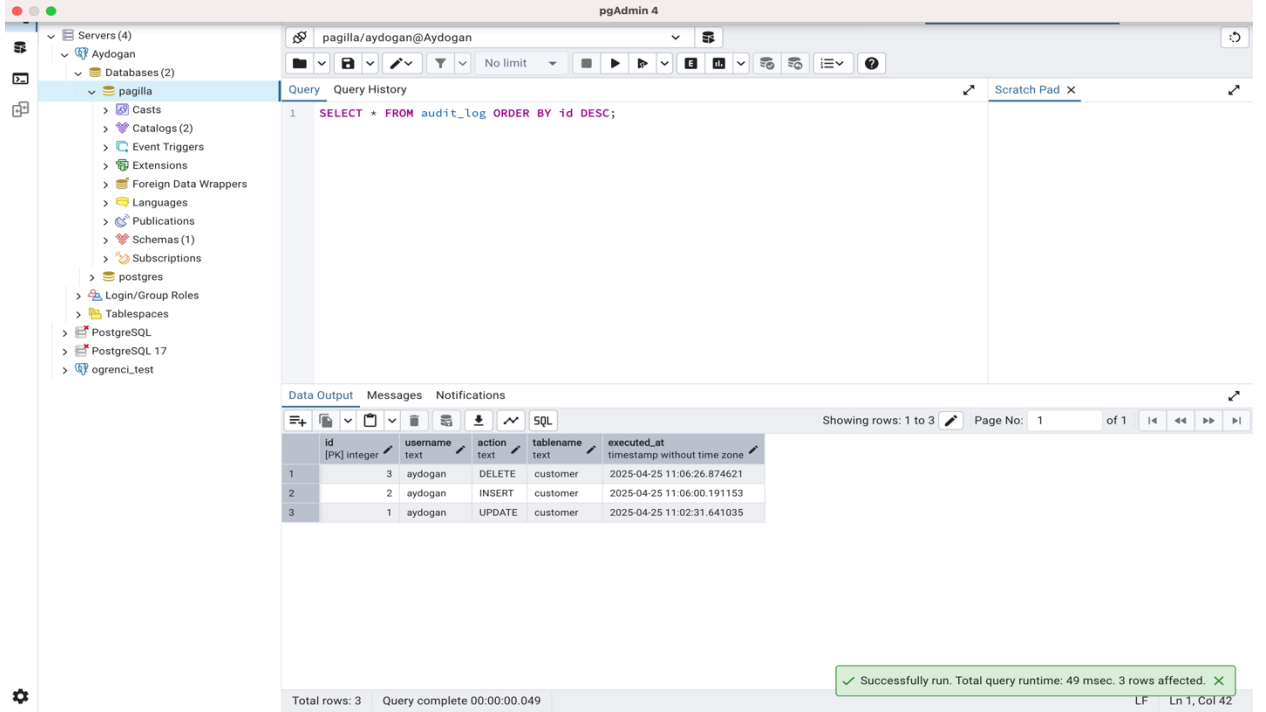
- İlk olarak `audit_log` adında bir tablo oluşturuldu:

```
CREATE TABLE audit_log (  
  id SERIAL PRIMARY KEY,  
  username TEXT,  
  action TEXT,  
  tablename TEXT,  
  executed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

- Ardından bir trigger fonksiyonu tanımlandı:

```
CREATE OR REPLACE FUNCTION fnc_audit_log()  
RETURNS TRIGGER AS $$  
BEGIN  
  INSERT INTO audit_log(username, action, tablename)  
  VALUES (current_user, TG_OP, TG_TABLE_NAME);  
  RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

- customer tablosuna yönelik INSERT, UPDATE ve DELETE işlemleri için trigger tanımlandı ve daha sonra bu işlemler customer tablosunda test edilerek audit_log tablosuna başarıyla yansıdığı görüntülendi:



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'pagilla' database. The main window shows a query result for the 'audit_log' table. The query executed is 'SELECT * FROM audit_log ORDER BY id DESC;'. The result table has 3 rows and 5 columns: id, username, action, tablename, and executed_at. The status bar at the bottom indicates 'Successfully run. Total query runtime: 49 msec. 3 rows affected.'

id	username	action	tablename	executed_at
3	aydogan	DELETE	customer	2025-04-25 11:06:26.874621
2	aydogan	INSERT	customer	2025-04-25 11:06:00.191153
1	aydogan	UPDATE	customer	2025-04-25 11:02:31.641035

4. Sonuç ve Değerlendirme

Bu proje kapsamında:

- Kullanıcı yetkileri sınırlandırarak erişim güvenliği sağlandı.
- Hassas veriler hash'lenerek koruma altına alındı.
- SQL injection riskine karşı güvenli bir yapı oluşturuldu.
- Tüm veri işleme işlemleri audit log mekanizmasıyla izlenebilir hale getirildi.

Proje, gerçek dünyadaki temel veritabanı güvenlik senaryolarını kapsamlı şekilde ele almış ve PostgreSQL üzerinde başarıyla uygulanmıştır.