**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**](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’ta 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.

1. **Destination’u Temizle ve Yeni Dosya Ekle**

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

b) **Add…** ile dosya yolunu ve adını şu şekilde girin:

 **1. Options Ayarları**

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

b)Eğer sıkıştırma desteği açık ise **Enable Compression**’ı 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**’ı seçin.
3. **Destination’u Temizle ve Yeni Dosya Ekle**  
   a. “Destination” listesindeki eski girdileri **Remove** butonuyla temizleyin.  
   b. **Add…** düğmesine tıklayıp dosya yolunu ve adını şu şekilde girin:



1. **Options Ayarları**  
   a. **Overwrite the existing backup set (WITH INIT)** seçeneğini işaretleyin.  
   b. Gerekli ise **Enable Compression** seçeneğini aktif hale getirin.
2. **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:



1. **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.
2. **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.

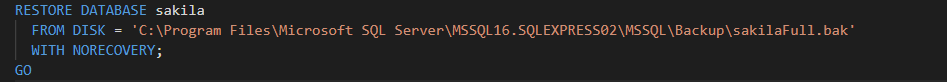
## Restore Adımları (T-SQL ile)

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

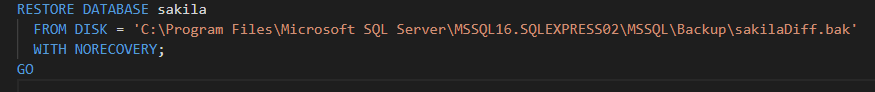
1. **Mevcut Veritabanını Sil ve Temiz Bir Kurulum Sağla**

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

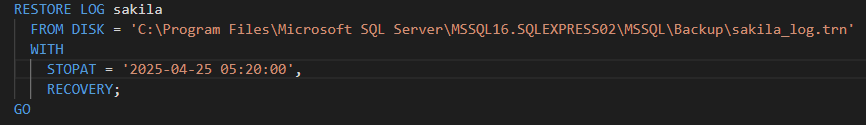
1. **Full Backup’tan Geri Yükleme**

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

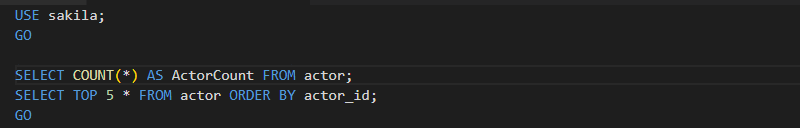
1. **Differential Backup’tan Geri Yükleme**

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

1. **Transaction Log’dan Point-in-Time Geri Yükleme ve Veritabanını Açma**

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.

1. **Doğrulama (Opsiyonel)**

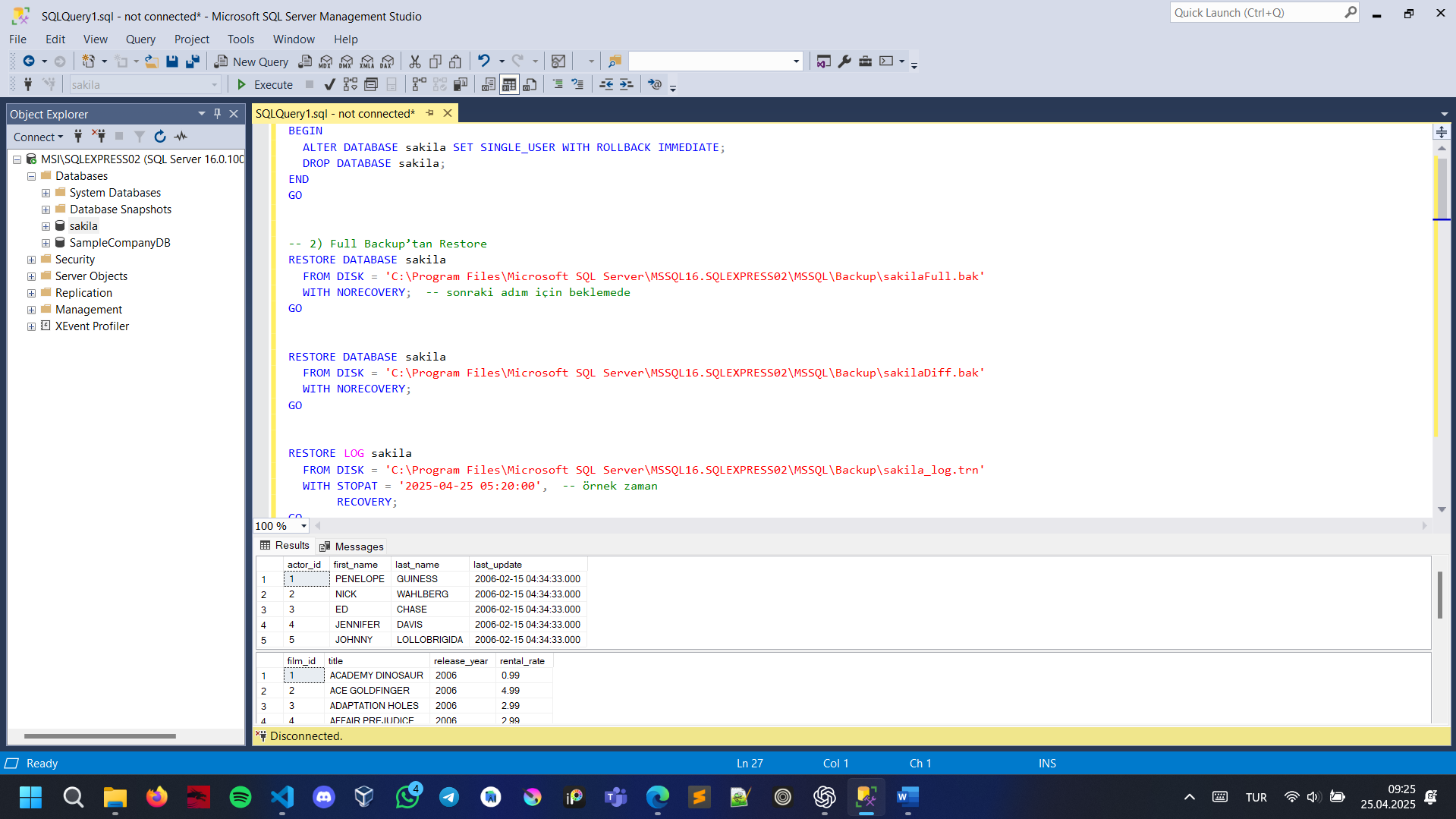


## Sonuç ve Değerlendirme

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’den 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

## 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\

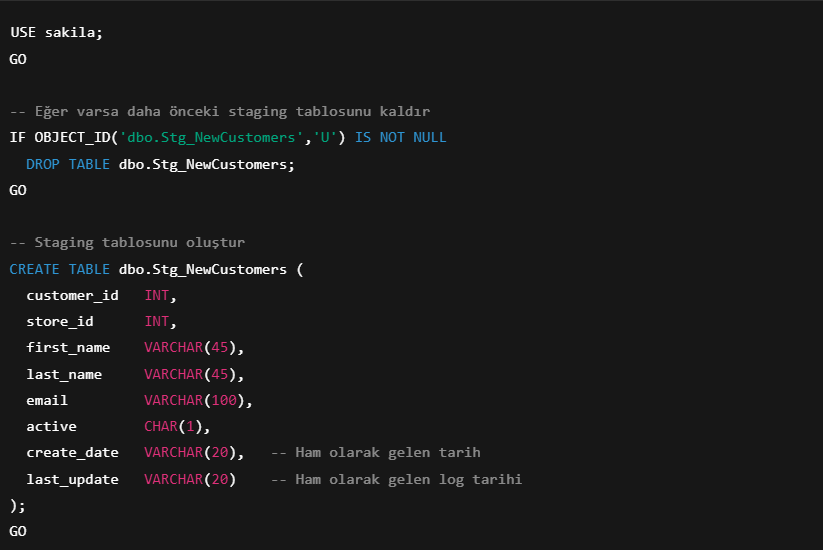
6. Yetkilendirme  
 Windows Authentication (DOMAIN\YourUser) ile ETL adımları çalıştırıldı.

## 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.

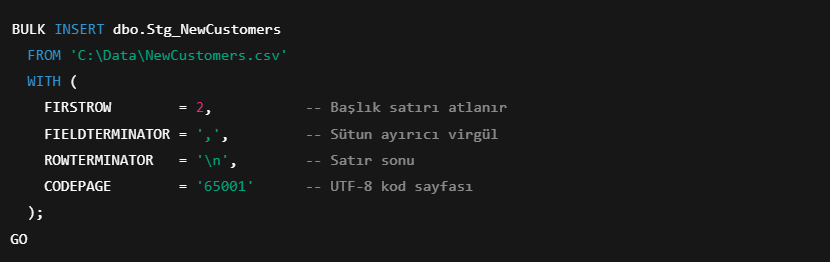
**3.1. Staging Tablosu Oluşturma**

**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.

****

**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.

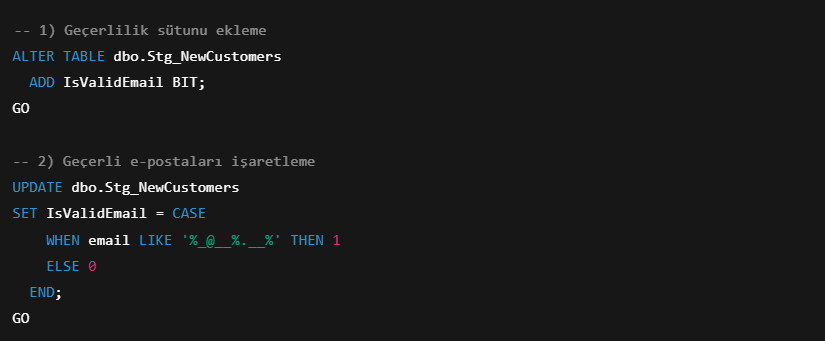
****

## 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**

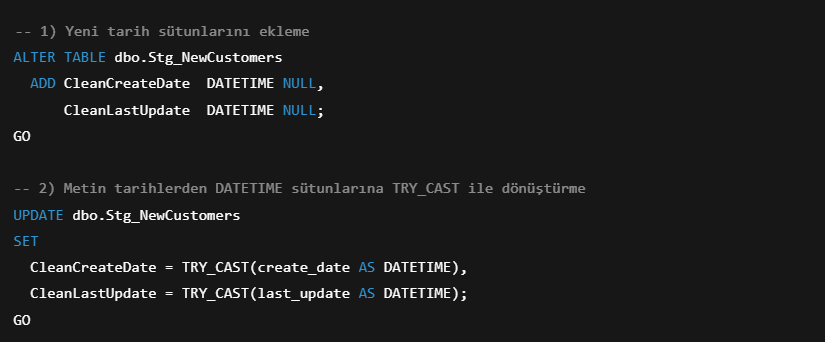
**Açıklama:**  
Bu adımda, her kaydın email sütunundaki değeri basit bir desen eşleştirme (LIKE '%\_@\_\_%.\_\_%') ile kontrol ediyoruz. Geçerli formatı sağlamayan kayıtları IsValidEmail = 0 olarak işaretleyip, daha sonra yükleme aşamasında elemek üzere hazırlıyoruz.

**Sonuç Kontrolü:**

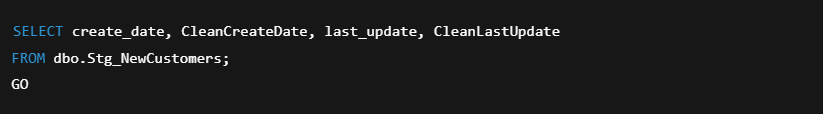
Bu sorgu, tüm e-posta adreslerini ve her birinin IsValidEmail değerini listeler.

**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.

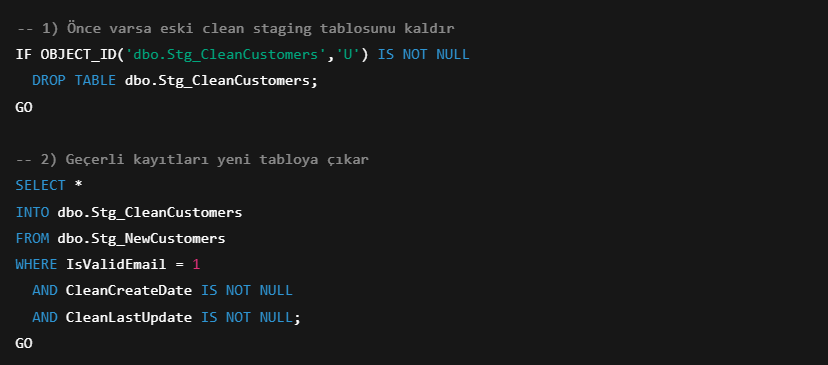
****

**Sonuç Kontrolü:**

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.

**Örnek Veri Kontrolü:**

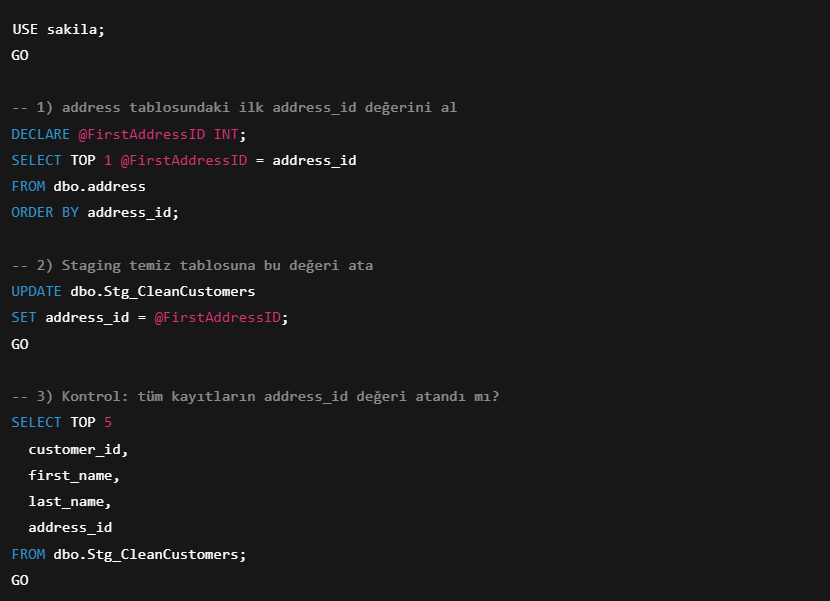


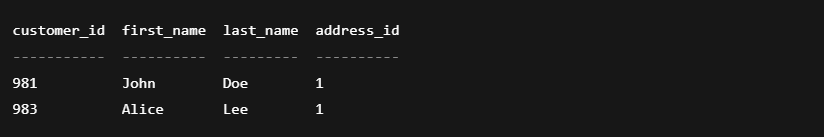
## 5.Load Adımları

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.

**Örnek Çıktı (SSMS)**

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.

**Mesaj Paneli Çıktısı (SSMS)**

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.



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 kayıtta 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:**

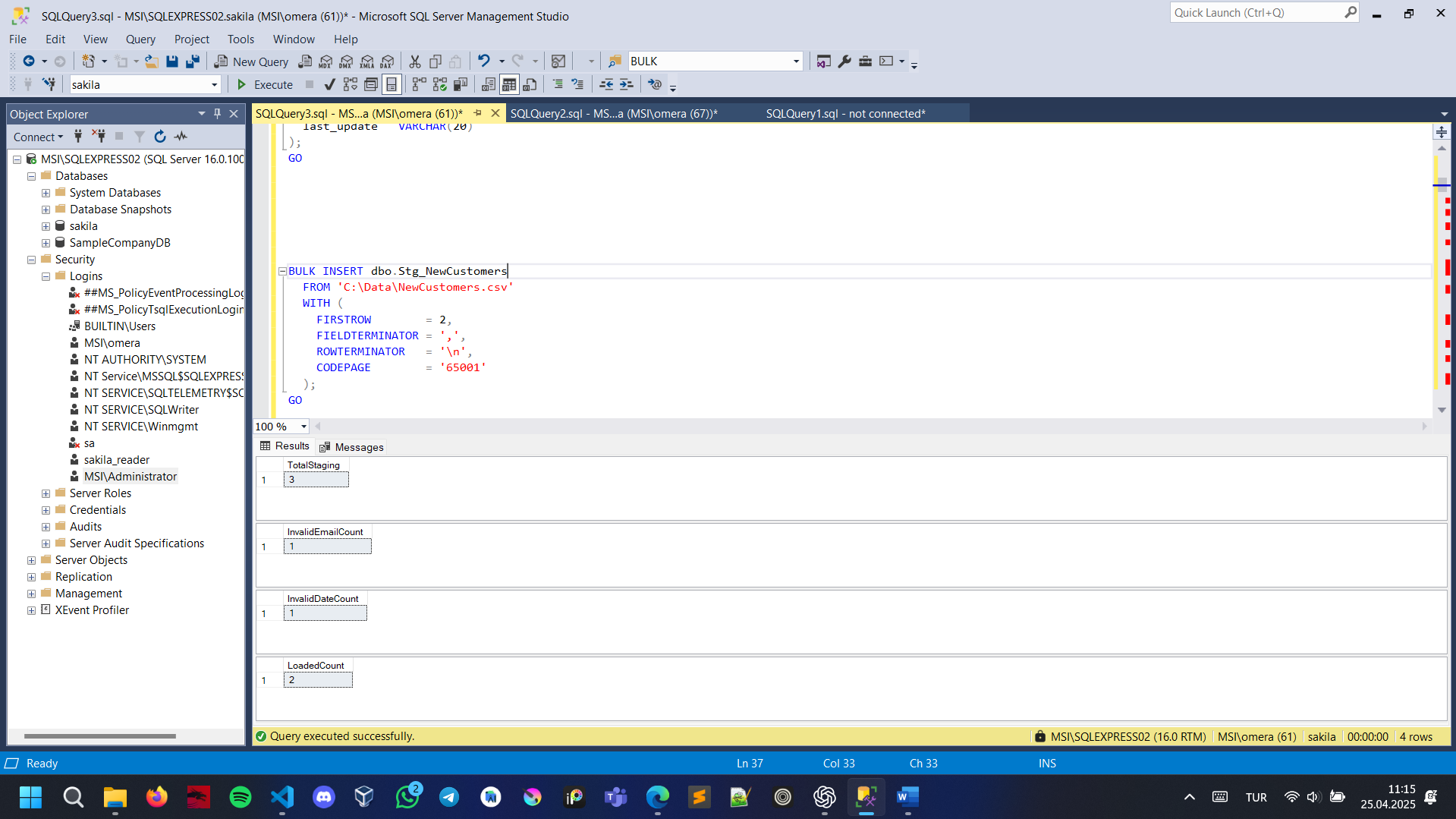
* **Sorun:** CSV’deki 2025-04-zz gibi hatalı tarih değerleri doğrudan datetime’a dönüştürülemedi.
* **Çözüm:** TRY\_CAST fonksiyonunu kullanarak dönüştürülemez satırları NULL kabul ettim; CleanCreateDate sütunundaki NULL’ları temizleme kriteri olarak belirledim.

**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 dökmemi 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

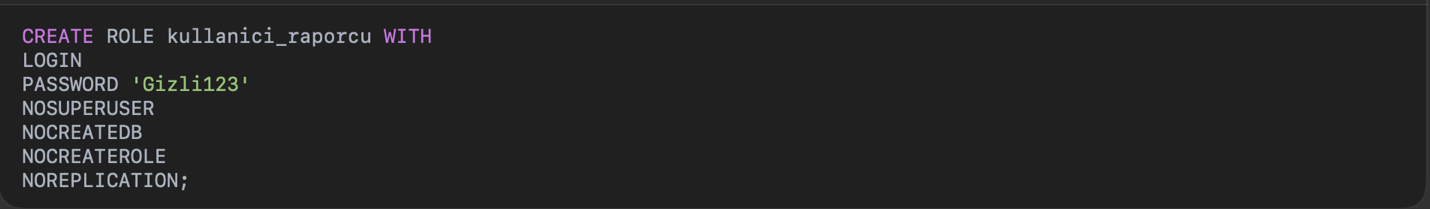
Bu projede PostgreSQL (PgAdmin4) üzerinde çalışan Pagilla örnek veritabanı kullanılarak, kullanıcıların veriye erişim yetkilerinin sınırlandırılması, şifrelerin güvenli şekilde saklanması, SQL Injection’a karşı koruma ve kullanıcı işlemlerinin günlüklenmesi (audit log) gibi temel veritabanı güvenliği 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: kullanici\_ogrenci
  + Veritabanı adı: pagilla

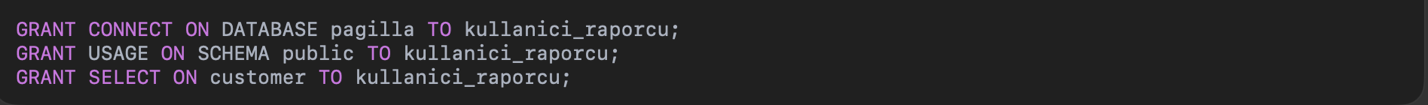
## 3. Uygulama Adımları

**3.1. Yeni Kullanıcı Rolleri 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.

**3.2. Yetkilendirme İşlemleri Gerçekleştirildi:**

kullanici\_raporcu rolüne sırasıyla:

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

**3.3. Şifreleme Özelliği Eklemek İçin Gerekli Uzantı Aktif Edildi:**

PostgreSQL’in pgcrypto uzantısı etkinleştirildi.



**3.4. customer Tablosuna Şifre Sütunu Eklendi:**

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

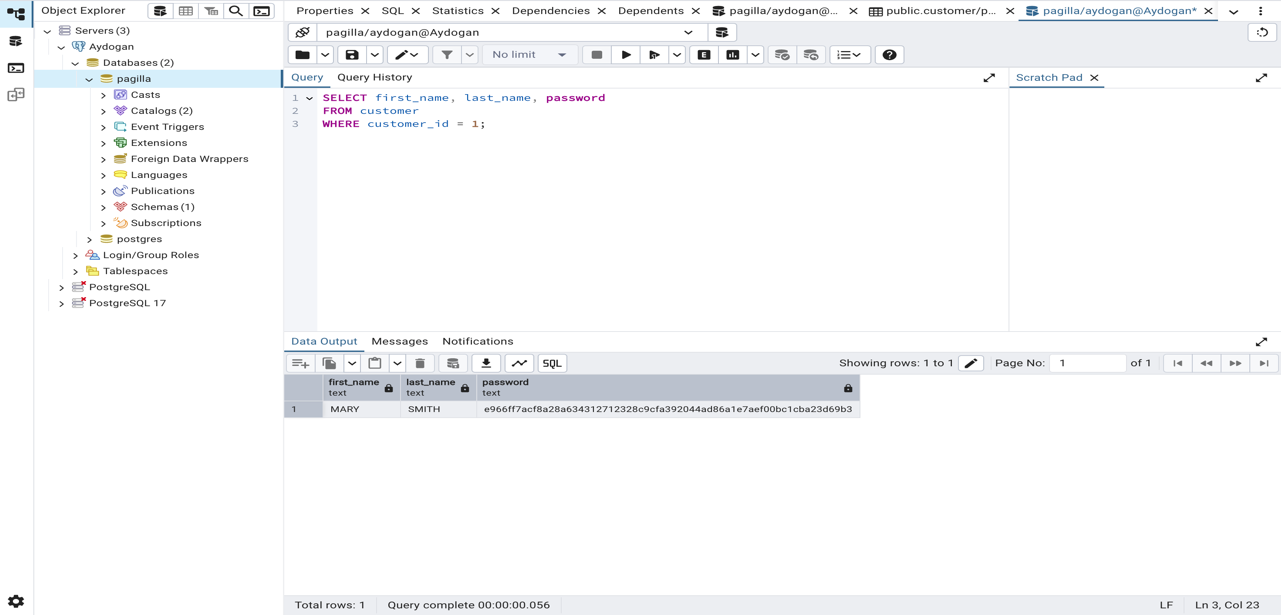


**3.5. SHA-256 Algoritmasıyla Şifreleme Uygulandı:**

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

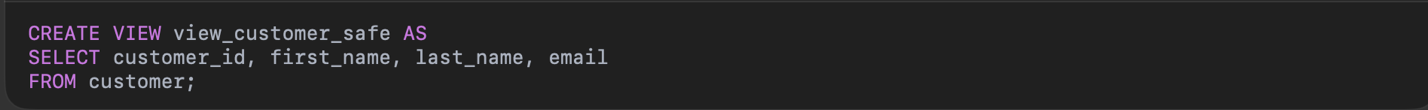
**3.6. Şifrelerin Başarıyla Kaydedildiği Görüldü:**

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.



**3.8. Öğrenci Rolü Oluşturuldu:**

Sisteme kullanici\_ogrenci adlı yeni bir kullanıcı eklendi. Bu kullanıcıya da tıpkı kullanici\_raporcu gibi sınırlı yetkiler verildi.



**3.9. Öğrenciye View Üzerinden Okuma Yetkisi Verildi:**

kullanici\_ogrenci kullanıcısına sadece view\_customer\_safe üzerinden okuma (SELECT) yetkisi tanımlandı.



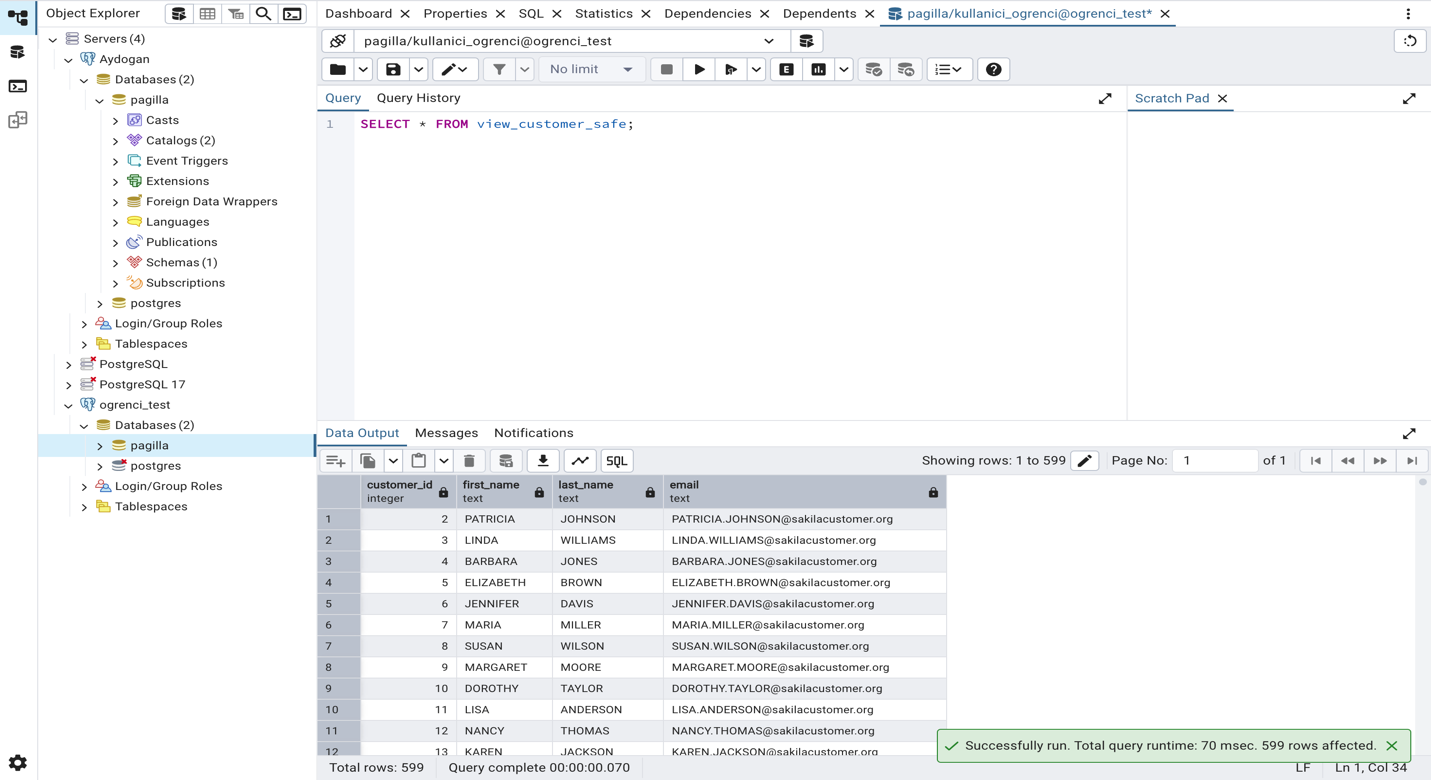
**3.10. Yeni Sunucu Bağlantısı Yapıldı:**

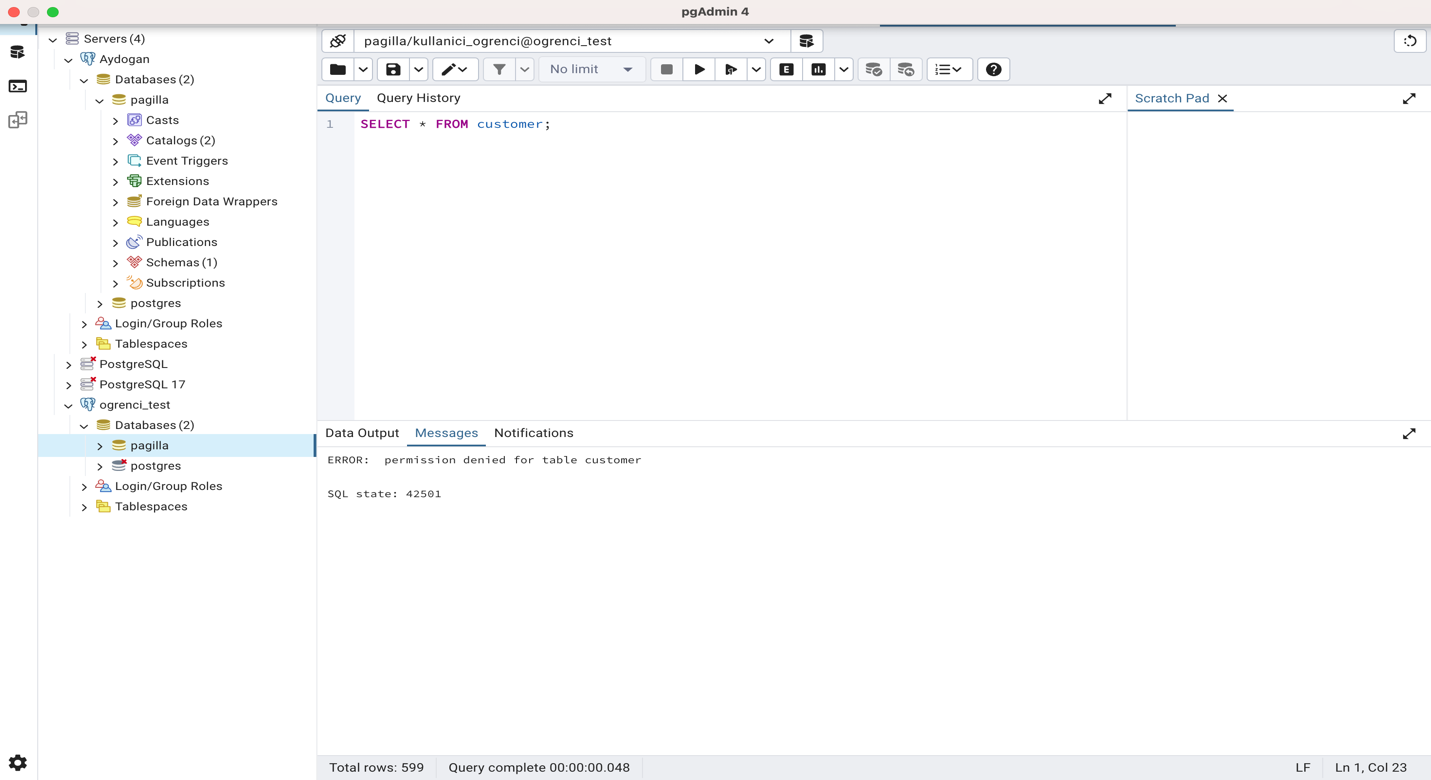
kullanici\_ogrenci bilgileriyle ogrenci\_test adlı yeni bir sunucu bağlantısı açıldı. Giriş bilgileri şu şekildeydi:

* + Host: localhost
  + Port: 5432
  + Username: kullanici\_ogrenci
  + Password: Sifre123

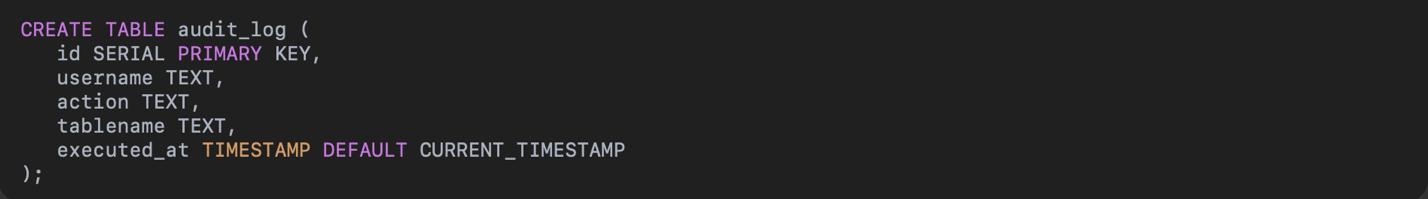
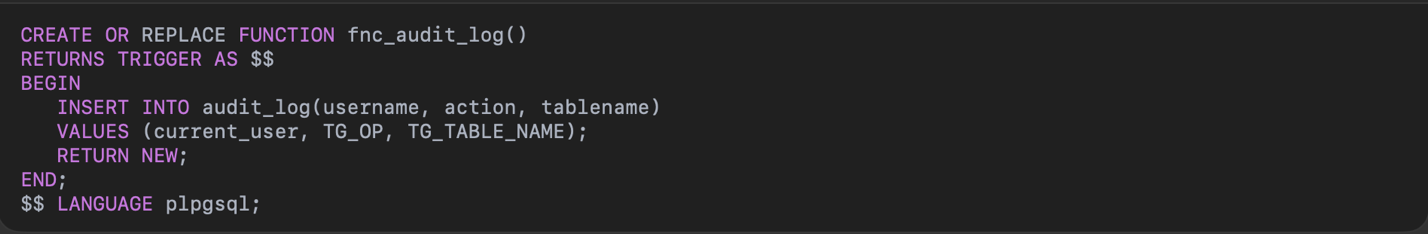
**3.11. View’e Erişim Sağlandı, customer Tablosuna Erişim Reddedildi:**

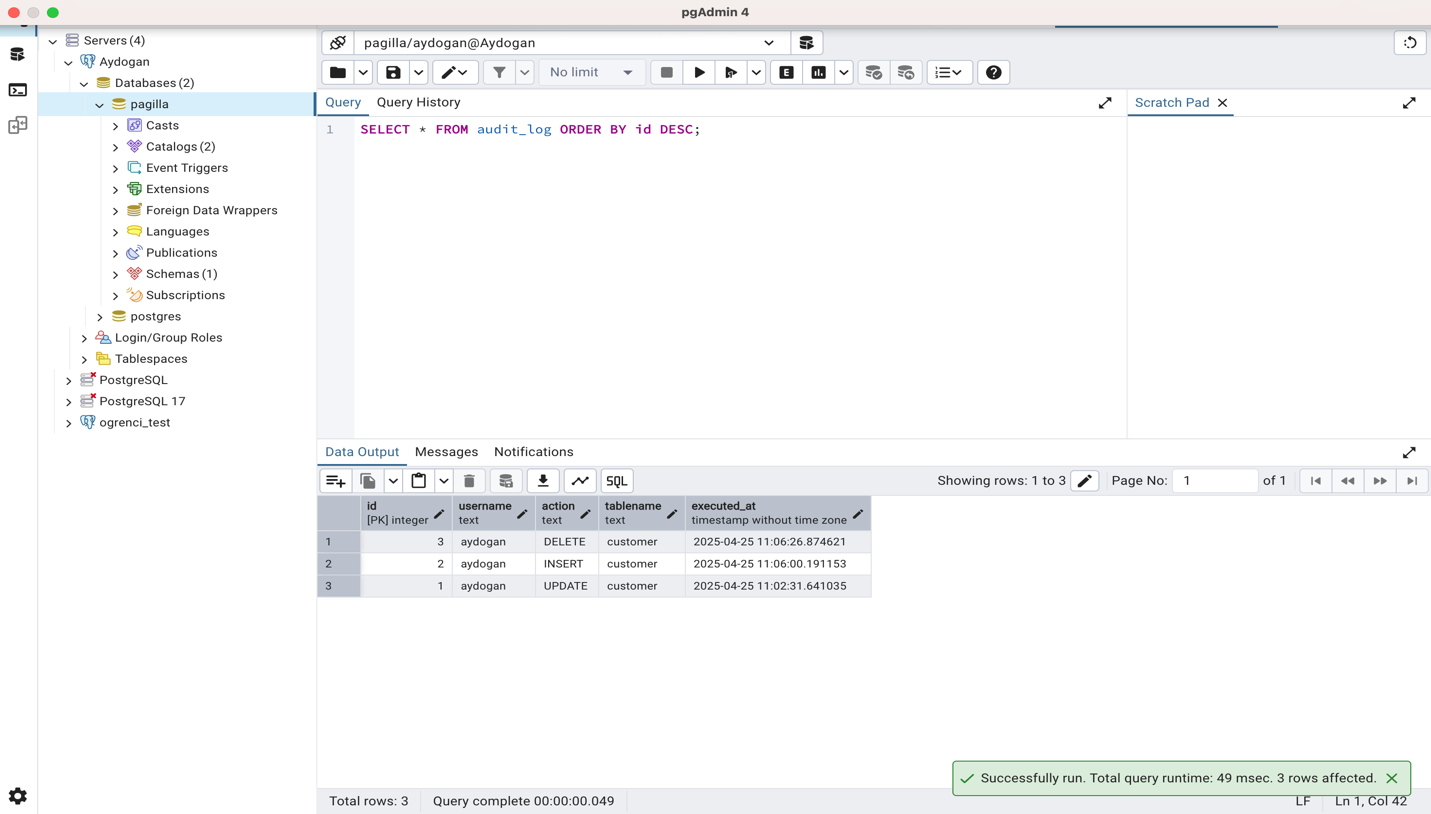
**View erişim izni var:**



**Customer erişim izni yok:**

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

* + **İlk olarak audit\_log adında bir tablo oluşturuldu:**
  + **Ardından bir trigger fonksiyonu tanımlandı:**
  + **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:**

****

## 4. Sonuç ve Değerlendirme

Bu proje kapsamında:

* Kullanıcı yetkileri sınırlanarak 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.