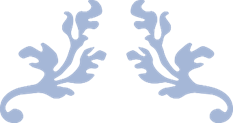


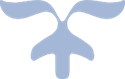
**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

2023-2024 AKADEMİK YILI

GÜZ DÖNEMİ



Veritabanı Yönetim Sistemleri

Online Alışveriş Sistemi Projesi Raporu

N.Ö. Grup 32

# Teslim Eden:

|  |  |
| --- | --- |
| **Numarası** | **Adı Soyadı** |
| 191001061 | Ömer Barış KAYAALP |
| 201001036 | Muharrem GEDİK |
| 211001036 | Umut BALCI |
| 191001017 | Emre Kaan ARSLAN |

**1- ER DİYAGRAMLARININ OLUŞTURULMASI............................................................................................................................. 2**

**2- TABLOLAR......................................................................................................................................................................................... 3**

**3- ROL ATAMALARI............................................................................................................................................................................12**

**4-TRİGGERLAR....................................................................................................................................................................................15**

**5- PROSEDÜRLER................................................................................................................................................................................24**

**6- UYGULAMA GİRİŞ EKRANI........................................................................................................................................................ 27**

**7- KAYIT OLMA EKRANI.................................................................................................................................................................. 28**

**8- ADMİN GİRİŞ EKRANI.................................................................................................................................................................. 30**

**9- ÜRÜNLER FORMU......................................................................................................................................................................... 31**

**10- Çalışan Bilgi Formu........................................................................................................................................................................ 33**

**11- MÜŞTERİ BİLGİ FORMU........................................................................................................................................................... 35**

**12- Kar Zarar Formu........................................................................................................................................................................... 36**

**13- SİPARİŞ FORMLARI.................................................................................................................................................................... 37**

**14- YEDEKLEME FORMU................................................................................................................................................................ 43**

**15- SİLİNMİŞ KAYITLAR FORMU.................................................................................................................................................. 46**

**16- PERSONEL GİRİŞ FORMU........................................................................................................................................................ 47**

**17- PERSONEL MÜŞTERİ BİLGİ FORMU.................................................................................................................................... 49**

**18- PERSONEL BİLGİLERİM FORMU........................................................................................................................................... 50**

**19- MÜŞTERİ GİRİŞ FORMU........................................................................................................................................................... 52**

**20- MÜŞTERİ BİLGİLERİM FORMU............................................................................................................................................. 54**

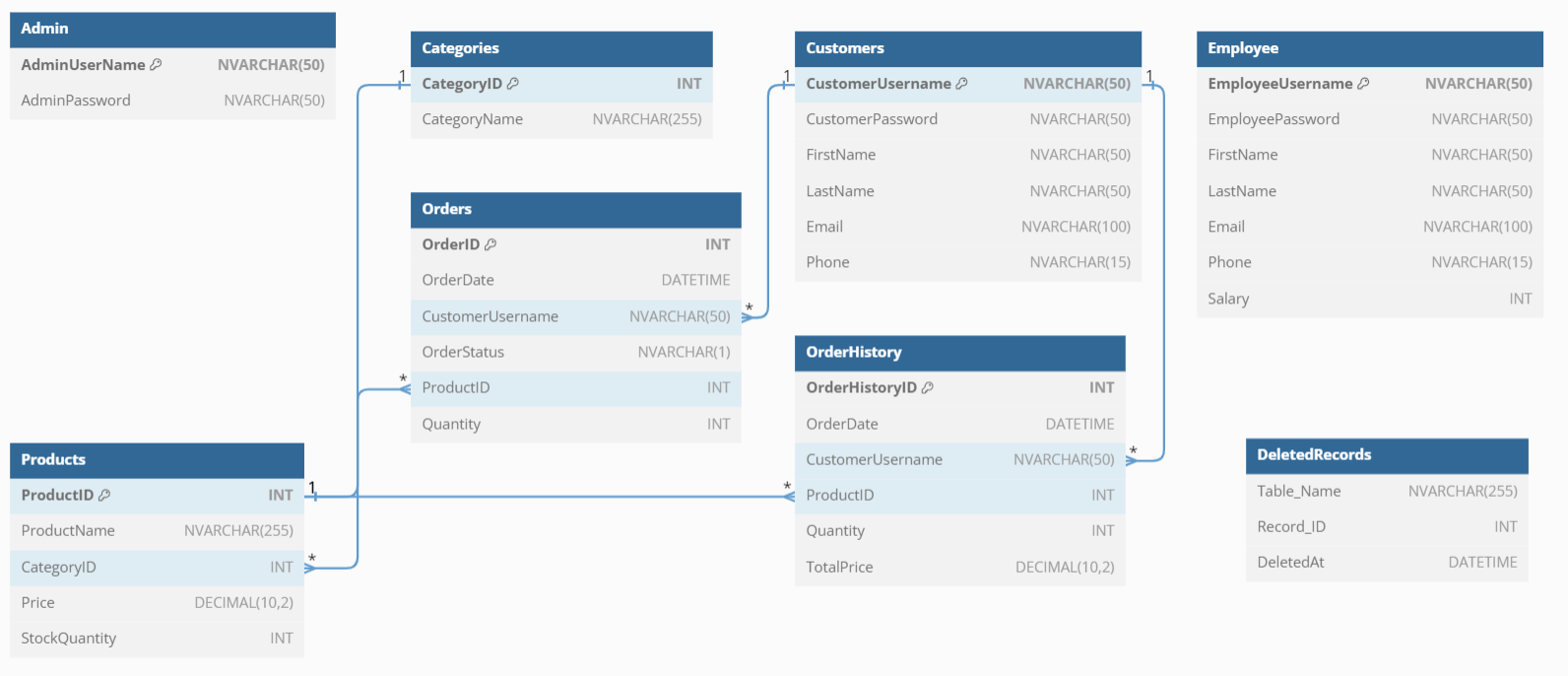
**21- MÜŞTERİ SİPARİŞ FORMU....................................................................................................................................................... 56**

**22- SİPARİŞ OLUŞTURMA FORMU................................................................................................................................................ 57**

**23- SEPET FORMU.............................................................................................................................................................................. 59**

**24- MÜŞTERİNİN SİPARİŞLERİNİ GÖRME FORMU................................................................................................................. 62**

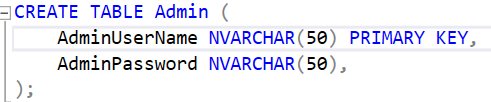
**1. ER DİYAGRAMLARININ OLUŞTURULMASI**



***Şekil 1.1***

Öğrenci Bilgi Sistemine uygun bir veritabanı oluşturulmak için Şekil 1’de görüldüğü üzere ER diyagramları çizildi ve aralarındaki bire – bir, bire – çok ve çoka çok ilişkiler gösterildi.

**2.TABLOLAR**



***Şekil 2.1***

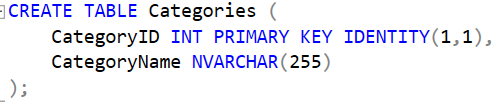
Bu SQL kodu, bir "Admin" tablosunu oluşturmak için kullanılır. İşte kodun kısa bir açıklaması:

**CREATE TABLE Admin**: Bu ifade, yeni bir tablo oluşturulacağını belirtir ve tablonun adının "Admin" olduğunu söyler.

**AdminUserName NVARCHAR(50) PRIMARY KEY,**: Bu kısım, tablonun ilk sütununu belirtir. **AdminUserName** adında bir sütun, veri tipi olarak **NVARCHAR(50)** (en fazla 50 karakter uzunluğunda Unicode metin) ile tanımlanır. Aynı zamanda bu sütun, tablonun birincil anahtarıdır (**PRIMARY KEY**), yani her bir değerin benzersiz olduğu ve bu sütun üzerinden kayıtlara erişim sağlandığı bir anahtar sütundur.

**AdminPassword NVARCHAR(50),**: Bu kısım, tablonun ikinci sütununu belirtir. **AdminPassword** adında bir sütun, yine **NVARCHAR(50)** veri tipi ile tanımlanır. Bu sütun, kullanıcıların şifrelerini depolamak için kullanılabilir.

Bu kodun amacı, bir yönetici (admin) tablosu oluşturarak, bu tabloya yönetici kullanıcı adları ve şifreleri gibi bilgileri saklama imkanı sağlamaktır. Ancak, şifreleri düz metin olarak depolamak güvenlik riski oluşturabilir. Genellikle şifrelerin güvenliği için hash fonksiyonları ve diğer güvenlik önlemleri kullanılması tavsiye edilir.



***Şekil 2.2***

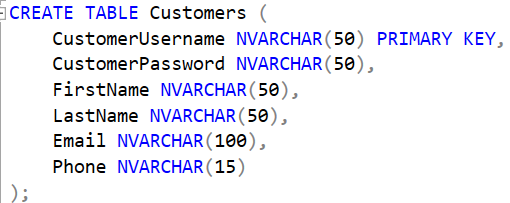
Bu SQL kodu, "Categories" adında bir tablonun oluşturulmasını sağlar. İşte kodun kısa bir açıklaması:

**CREATE TABLE Categories**: Bu ifade, yeni bir tablo oluşturulacağını belirtir ve tablonun adının "Categories" olduğunu söyler.

**(CategoryID INT PRIMARY KEY IDENTITY(1,1),**: Bu kısım, tablonun ilk sütununu belirtir. **CategoryID** adında bir sütun, veri tipi olarak **INT** (tam sayı) ile tanımlanır. Ayrıca, bu sütun tablonun birincil anahtarıdır (**PRIMARY KEY**). **IDENTITY(1,1)** ifadesi, bu sütunun otomatik olarak artan bir şekilde (identity) değer alacağını ve 1'den başlayarak 1'er 1'er artacağını belirtir. Yani, her yeni kayıt eklediğinizde, **CategoryID** otomatik olarak bir öncekinden 1 daha fazla olacaktır.

**CategoryName NVARCHAR(255)**: Bu kısım, tablonun ikinci sütununu belirtir. **CategoryName** adında bir sütun, **NVARCHAR(255)** veri tipi ile tanımlanır. Bu sütun, kategorilerin isimlerini depolamak için kullanılır.

Bu kodun amacı, kategorileri temsil eden bir tablo oluşturarak, her kategoriye benzersiz bir kimlik (CategoryID) atamak ve kategorilerin isimlerini (CategoryName) depolamak için bir yapı sağlamaktır. Otomatik artan bir kimlik sütunu, her kategori eklediğinizde benzersiz bir kimlik sağlar.



***Şekil 2.3***

Bu SQL kodu, "Customers" adında bir tablonun oluşturulmasını sağlar. İşte kodun kısa bir açıklaması:

**CREATE TABLE Customers**: Bu ifade, yeni bir tablo oluşturulacağını belirtir ve tablonun adının "Customers" olduğunu söyler.

**CustomerUsername NVARCHAR(50) PRIMARY KEY,**: Bu kısım, tablonun ilk sütununu belirtir. **CustomerUsername** adında bir sütun, veri tipi olarak **NVARCHAR(50)** (en fazla 50 karakter uzunluğunda Unicode metin) ile tanımlanır. Aynı zamanda bu sütun, tablonun birincil anahtarıdır (**PRIMARY KEY**), yani her bir müşteri kullanıcı adının benzersiz olduğu ve bu sütun üzerinden kayıtlara erişim sağlandığı bir anahtar sütundur.

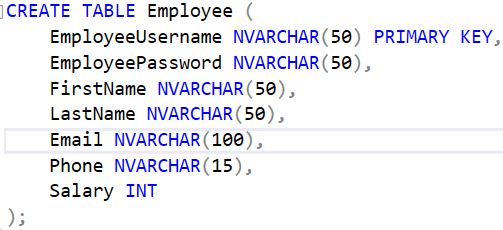
**CustomerPassword NVARCHAR(50),**: Bu kısım, tablonun ikinci sütununu belirtir. **CustomerPassword** adında bir sütun, yine **NVARCHAR(50)** veri tipi ile tanımlanır. Bu sütun, müşteri şifrelerini depolamak için kullanılır.

**FirstName NVARCHAR(50), LastName NVARCHAR(50),**: Bu kısımlar, müşterilerin adını ve soyadını depolamak için iki ayrı sütunu tanımlar.

**Email NVARCHAR(100),**: Müşterilerin e-posta adresini depolamak için bir sütun.

**Phone NVARCHAR(15)**: Müşterilerin telefon numarasını depolamak için bir sütun.

Bu kodun amacı, müşteri bilgilerini depolamak için bir tablo oluşturarak, her müşteriye benzersiz bir kullanıcı adı atamak ve müşteri ile ilgili diğer bilgileri (şifre, ad, soyad, e-posta, telefon) depolamak için bir yapı sağlamaktır. Bu tasarım, müşteri verilerini düzenli ve güvenli bir şekilde saklamayı amaçlar. Ancak, gerçek uygulamalarda şifrelerin güvenliği için daha karmaşık önlemler alınması önerilir.



***Şekil 2.4***

Bu SQL kodu, "Employee" adında bir tablonun oluşturulmasını sağlar. İşte kodun kısa bir açıklaması:

**CREATE TABLE Employee**: Bu ifade, yeni bir tablo oluşturulacağını belirtir ve tablonun adının "Employee" olduğunu söyler.

**(EmployeeUsername NVARCHAR(50) PRIMARY KEY,**: Bu kısım, tablonun ilk sütununu belirtir. **EmployeeUsername** adında bir sütun, veri tipi olarak **NVARCHAR(50)** (en fazla 50 karakter uzunluğunda Unicode metin) ile tanımlanır. Aynı zamanda bu sütun, tablonun birincil anahtarıdır (**PRIMARY KEY**), yani her bir çalışan kullanıcı adının benzersiz olduğu ve bu sütun üzerinden kayıtlara erişim sağlandığı bir anahtar sütundur.

**EmployeePassword NVARCHAR(50),**: Bu kısım, tablonun ikinci sütununu belirtir. **EmployeePassword** adında bir sütun, yine **NVARCHAR(50)** veri tipi ile tanımlanır. Bu sütun, çalışan şifrelerini depolamak için kullanılır.

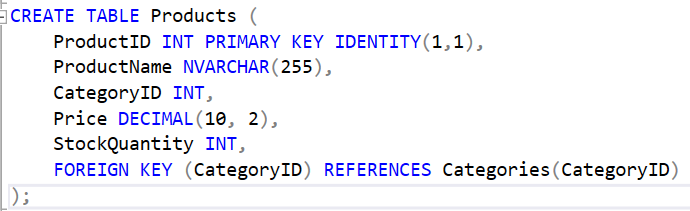
**FirstName NVARCHAR(50), LastName NVARCHAR(50),**: Bu kısımlar, çalışanların adını ve soyadını depolamak için iki ayrı sütunu tanımlar.

**Email NVARCHAR(100),**: Çalışanların e-posta adresini depolamak için bir sütun.

**Phone NVARCHAR(15),**: Çalışanların telefon numarasını depolamak için bir sütun.

**Salary INT**: Çalışanların maaşını depolamak için bir sütun. Bu sütun, tam sayı veri tipinde olduğu için (**INT**), çalışanların maaşlarını tam sayı olarak saklar.

Bu kodun amacı, işçi (employee) bilgilerini depolamak için bir tablo oluşturarak, her çalışana benzersiz bir kullanıcı adı atamak ve çalışan ile ilgili diğer bilgileri (şifre, ad, soyad, e-posta, telefon, maaş) depolamak için bir yapı sağlamaktır.



***Şekil 2.5***

Bu SQL kodu, "Products" adında bir tablonun oluşturulmasını sağlar ve bu tablonun tasarımında çeşitli özellikleri içerir. İşte kodun kısa bir açıklaması:

**CREATE TABLE Products**: Bu ifade, yeni bir tablo oluşturulacağını belirtir ve tablonun adının "Products" olduğunu söyler.

**(ProductID INT PRIMARY KEY IDENTITY(1,1),**: Bu kısım, tablonun ilk sütununu belirtir. **ProductID** adında bir sütun, veri tipi olarak **INT** (tam sayı) ile tanımlanır. Aynı zamanda bu sütun, tablonun birincil anahtarıdır (**PRIMARY KEY**). **IDENTITY(1,1)** ifadesi, bu sütunun otomatik olarak artan bir şekilde (identity) değer alacağını ve 1'den başlayarak 1'er 1'er artacağını belirtir. Yani, her yeni kayıt eklediğinizde, **ProductID** otomatik olarak bir öncekinden 1 daha fazla olacaktır.

**ProductName NVARCHAR(255),**: Bu kısım, ürünlerin adını depolamak için bir sütunu belirtir. **NVARCHAR(255)** veri tipi, en fazla 255 karakter uzunluğunda Unicode metinleri destekler.

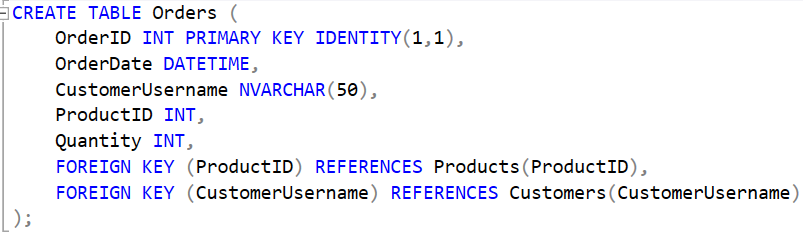
**CategoryID INT,**: Bu kısım, ürünün hangi kategoriye ait olduğunu belirtmek için bir sütunu tanımlar. Bu sütun, başka bir tablo olan "Categories" tablosundan referans alınan bir dış anahtar (**FOREIGN KEY**) içerir. Bu şekilde, her ürün bir kategori ile ilişkilendirilebilir.

**Price DECIMAL(10, 2),**: Bu kısım, ürünün fiyatını depolamak için bir sütunu belirtir. **DECIMAL(10, 2)** veri tipi, 10 basamaklı sayıları destekler ve bunların 2 basamağını ondalık kısmında saklar. Yani, ürün fiyatları ondalık sayı olarak depolanır.

**StockQuantity INT,**: Bu kısım, ürünün stok miktarını depolamak için bir sütunu belirtir. **INT** veri tipi, tam sayı değerlerini destekler.

**FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)**: Bu ifade, "CategoryID" sütununu dış anahtar olarak belirtir ve bu sütunun "Categories" tablosundaki "CategoryID" sütunu ile ilişkilendirildiğini gösterir. Bu, "Products" tablosundaki her bir ürünün bir kategori ile ilişkilendirilmesini sağlar.

Bu kodun amacı, ürün bilgilerini depolamak için bir tablo oluşturarak, her ürüne benzersiz bir kimlik (ProductID) atamak, ürünün adını, kategori bilgisini, fiyatını ve stok miktarını saklamak ve kategori bilgisini "Categories" tablosundan alarak ilişkilendirmektir.



***Şekil 2.6***

Bu SQL kodu, "Orders" adında bir tablonun oluşturulmasını sağlar ve bu tablonun tasarımında çeşitli özellikleri içerir. İşte kodun kısa bir açıklaması:

**CREATE TABLE Orders**: Bu ifade, yeni bir tablo oluşturulacağını belirtir ve tablonun adının "Orders" olduğunu söyler.

**OrderID INT PRIMARY KEY IDENTITY(1,1),**: Bu kısım, tablonun ilk sütununu belirtir. **OrderID** adında bir sütun, veri tipi olarak **INT** (tam sayı) ile tanımlanır. Aynı zamanda bu sütun, tablonun birincil anahtarıdır (**PRIMARY KEY**). **IDENTITY(1,1)** ifadesi, bu sütunun otomatik olarak artan bir şekilde (identity) değer alacağını ve 1'den başlayarak 1'er 1'er artacağını belirtir. Yani, her yeni sipariş eklediğinizde, **OrderID** otomatik olarak bir öncekinden 1 daha fazla olacaktır.

**OrderDate DATETIME,**: Bu kısım, siparişin oluşturulduğu tarihi depolamak için bir sütunu belirtir. **DATETIME** veri tipi, tarih ve saat bilgisini içerir.

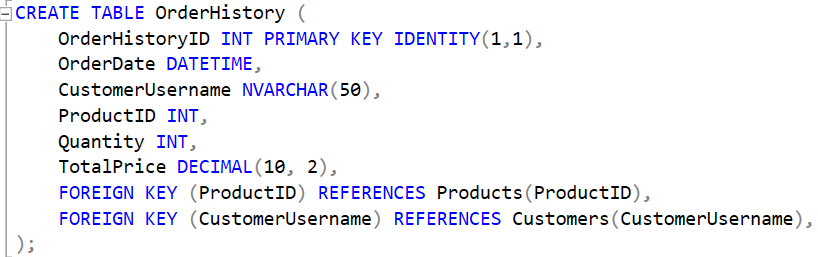
**CustomerUsername NVARCHAR(50),**: Bu kısım, siparişi veren müşterinin kullanıcı adını depolamak için bir sütunu belirtir. Bu sütun, başka bir tablo olan "Customers" tablosundan referans alınan bir dış anahtar (**FOREIGN KEY**) içerir.

**ProductID INT,**: Bu kısım, sipariş edilen ürünün kimliğini depolamak için bir sütunu belirtir. Bu sütun, başka bir tablo olan "Products" tablosundan referans alınan bir dış anahtar içerir.

**Quantity INT,**: Bu kısım, sipariş edilen ürünün miktarını depolamak için bir sütunu belirtir. **INT** veri tipi, tam sayı değerlerini destekler.

**FOREIGN KEY (ProductID) REFERENCES Products(ProductID),**: Bu ifade, "ProductID" sütununu dış anahtar olarak belirtir ve bu sütunun "Products" tablosundaki "ProductID" sütunu ile ilişkilendirildiğini gösterir. Bu, "Orders" tablosundaki her bir siparişin bir ürün ile ilişkilendirilmesini sağlar.

**FOREIGN KEY (CustomerUsername) REFERENCES Customers(CustomerUsername)**: Bu ifade, "CustomerUsername" sütununu dış anahtar olarak belirtir ve bu sütunun "Customers" tablosundaki "CustomerUsername" sütunu ile ilişkilendirildiğini gösterir. Bu, "Orders" tablosundaki her bir siparişin bir müşteri ile ilişkilendirilmesini sağlar.Bu kodun amacı, sipariş bilgilerini depolamak için bir tablo oluşturarak, her siparişe benzersiz bir kimlik (OrderID) atamak, siparişin tarihini, müşteri bilgisini, sipariş edilen ürün bilgisini ve miktarını saklamak, ayrıca ürün ve müşteri bilgilerini ilgili tablolardan referans alarak ilişkilendirmektir.



***Şekil 2.7***

Bu SQL kodu, "OrderHistory" adında bir tablonun oluşturulmasını sağlar ve bu tablonun tasarımında çeşitli özellikleri içerir. İşte kodun kısa bir açıklaması:

**CREATE TABLE OrderHistory**: Bu ifade, yeni bir tablo oluşturulacağını belirtir ve tablonun adının "OrderHistory" olduğunu söyler.

**(OrderHistoryID INT PRIMARY KEY IDENTITY(1,1),**: Bu kısım, tablonun ilk sütununu belirtir. **OrderHistoryID** adında bir sütun, veri tipi olarak **INT** (tam sayı) ile tanımlanır. Aynı zamanda bu sütun, tablonun birincil anahtarıdır (**PRIMARY KEY**). **IDENTITY(1,1)** ifadesi, bu sütunun otomatik olarak artan bir şekilde (identity) değer alacağını ve 1'den başlayarak 1'er 1'er artacağını belirtir. Yani, her yeni sipariş geçmişi kaydı eklediğinizde, **OrderHistoryID** otomatik olarak bir öncekinden 1 daha fazla olacaktır.

**OrderDate DATETIME,**: Bu kısım, siparişin geçmişe kaydedildiği tarihi depolamak için bir sütunu belirtir. **DATETIME** veri tipi, tarih ve saat bilgisini içerir.

**CustomerUsername NVARCHAR(50),**: Bu kısım, siparişi veren müşterinin kullanıcı adını depolamak için bir sütunu belirtir. Bu sütun, başka bir tablo olan "Customers" tablosundan referans alınan bir dış anahtar (**FOREIGN KEY**) içerir.

**ProductID INT,**: Bu kısım, sipariş edilen ürünün kimliğini depolamak için bir sütunu belirtir. Bu sütun, başka bir tablo olan "Products" tablosundan referans alınan bir dış anahtar içerir.

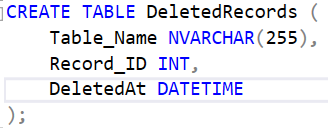
**Quantity INT,**: Bu kısım, sipariş edilen ürünün miktarını depolamak için bir sütunu belirtir. **INT** veri tipi, tam sayı değerlerini destekler.

**TotalPrice DECIMAL(10, 2),**: Bu kısım, siparişin toplam fiyatını depolamak için bir sütunu belirtir. **DECIMAL(10, 2)** veri tipi, 10 basamaklı sayıları destekler ve bunların 2 basamağını ondalık kısmında saklar. Yani, toplam fiyat ondalık sayı olarak depolanır.

**FOREIGN KEY (ProductID) REFERENCES Products(ProductID),**: Bu ifade, "ProductID" sütununu dış anahtar olarak belirtir ve bu sütunun "Products" tablosundaki "ProductID" sütunu ile ilişkilendirildiğini gösterir. Bu, "OrderHistory" tablosundaki her bir siparişin bir ürün ile ilişkilendirilmesini sağlar.

**FOREIGN KEY (CustomerUsername) REFERENCES Customers(CustomerUsername)**: Bu ifade, "CustomerUsername" sütununu dış anahtar olarak belirtir ve bu sütunun "Customers" tablosundaki "CustomerUsername" sütunu ile ilişkilendirildiğini gösterir. Bu, "OrderHistory" tablosundaki her bir siparişin bir müşteri ile ilişkilendirilmesini sağlar.

Bu kodun amacı, geçmişe yönelik sipariş bilgilerini depolamak için bir tablo oluşturarak, her geçmiş siparişe benzersiz bir kimlik (OrderHistoryID) atamak, siparişin geçmişe kaydedildiği tarihini, müşteri bilgisini, sipariş edilen ürün bilgisini, miktarını ve toplam fiyatını saklamak, ayrıca ürün ve müşteri bilgilerini ilgili tablolardan referans alarak ilişkilendirmektir.



***Şekil 2.8***

Bu SQL kodu, "DeletedRecords" adında bir tablonun oluşturulmasını sağlar. İşte kodun kısa bir açıklaması:

**CREATE TABLE DeletedRecords**: Bu ifade, yeni bir tablo oluşturulacağını belirtir ve tablonun adının "DeletedRecords" olduğunu söyler.

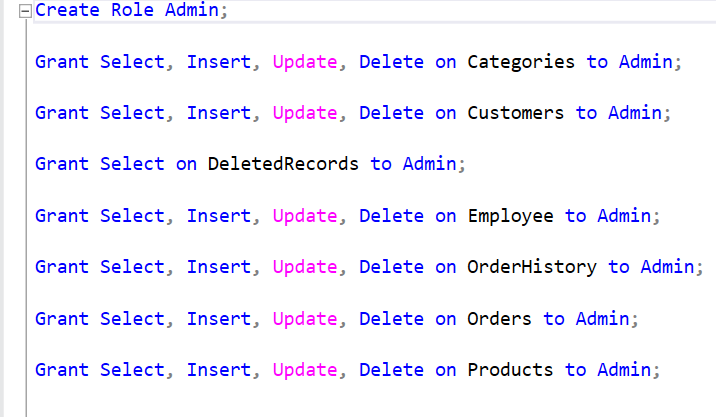
**(Table\_Name NVARCHAR(255),**: Bu kısım, tablonun ilk sütununu belirtir. **Table\_Name** adında bir sütun, veri tipi olarak **NVARCHAR(255)** (en fazla 255 karakter uzunluğunda Unicode metin) ile tanımlanır. Bu sütun, silinen kaydın bulunduğu ana tablonun adını depolar.

**Record\_ID INT,**: Bu kısım, tablonun ikinci sütununu belirtir. **Record\_ID** adında bir sütun, veri tipi olarak **INT** (tam sayı) ile tanımlanır. Bu sütun, silinen kaydın benzersiz kimliğini (ID) depolar.

**DeletedAt DATETIME**: Bu kısım, tablonun üçüncü sütununu belirtir. **DeletedAt** adında bir sütun, veri tipi olarak **DATETIME** ile tanımlanır ve bu sütun, kaydın ne zaman silindiğini belirtir.

Bu kodun amacı, veritabanında silinen kayıtların geçmişini takip etmek için bir tablo oluşturarak, hangi tablodan hangi kaydın silindiğini, hangi ID'ye sahip olduğunu ve ne zaman silindiğini kaydetmektir. Bu tür bir tablo, veri kaybını önlemek ve geri dönüş yapabilmek için faydalı olabilir.

**3. ROL ATAMALARI**

  
  
 ***Şekil 3.1***

Bu SQL kodları, bir rol oluşturulması (**Admin**) ve bu role belirli tablolara erişim izinlerinin verilmesi işlemlerini gerçekleştirir. İşte kodların kısa açıklamaları:

**Create Role Admin;**: Bu ifade, "Admin" adında bir rol oluşturur. Bir rol, belirli kullanıcıların veya kullanıcı gruplarının belli yetkilere sahip olmalarını sağlar.

**Grant Select, Insert, Update, Delete on Categories to Admin;**: Bu ifade, "Admin" rolüne "Categories" tablosunda **SELECT**, **INSERT**, **UPDATE**, ve **DELETE** yetkilerini verir. Yani, "Admin" rolü bu tablo üzerinde okuma, ekleme, güncelleme ve silme işlemleri gerçekleştirebilir.

**Grant Select, Insert, Update, Delete on Customers to Admin;**: Bu ifade, "Admin" rolüne "Customers" tablosunda aynı yetkileri verir.

**Grant Select on DeletedRecords to Admin;**: Bu ifade, "Admin" rolüne "DeletedRecords" tablosunda sadece **SELECT** yetkisini verir. Yani, "Admin" rolü bu tablo üzerinde sadece okuma işlemi yapabilir.

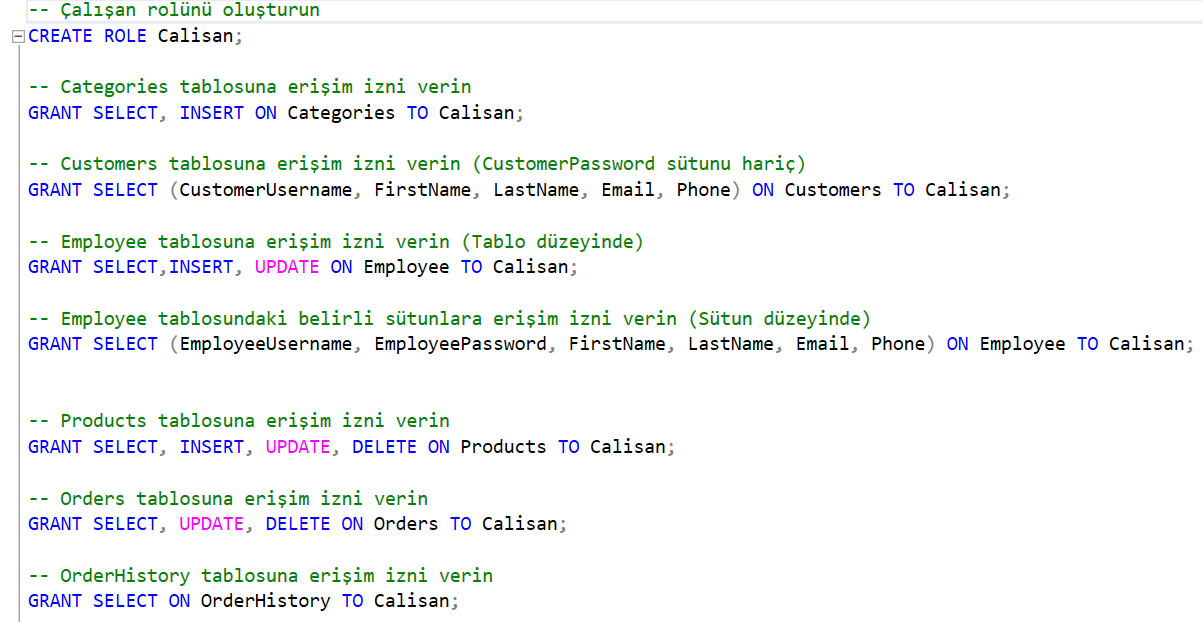
**Grant Select, Insert, Update, Delete on Employee to Admin;**: Bu ifade, "Admin" rolüne "Employee" tablosunda aynı yetkileri verir.

**Grant Select, Insert, Update, Delete on OrderHistory to Admin;**: Bu ifade, "Admin" rolüne "OrderHistory" tablosunda aynı yetkileri verir.

**Grant Select, Insert, Update, Delete on Orders to Admin;**: Bu ifade, "Admin" rolüne "Orders" tablosunda aynı yetkileri verir.

**Grant Select, Insert, Update, Delete on Products to Admin;**: Bu ifade, "Admin" rolüne "Products" tablosunda aynı yetkileri verir.

Bu kodlar, veritabanı yöneticilerine, "Admin" rolüne sahip kullanıcılara belirli tablolarda okuma, ekleme, güncelleme ve silme işlemleri gerçekleştirme yetkisi verir. Her tablo için ayrı ayrı yetkiler belirlenmiş ve bu yetkiler "Admin" rolüne atanmıştır.



***Şekil 3.2***

Bu SQL kodları, "Calisan" adında bir rol oluşturulması ve bu role belirli tablolara ve sütunlara erişim izinlerinin verilmesi işlemlerini gerçekleştirir. İşte kodların kısa açıklamaları:

**CREATE ROLE Calisan;**: Bu ifade, "Calisan" adında bir rol oluşturur. Bu rol, belirli yetkilere sahip kullanıcıları temsil eder.

**GRANT SELECT, INSERT ON Categories TO Calisan;**: Bu ifade, "Calisan" rolüne "Categories" tablosunda **SELECT** ve **INSERT** yetkilerini verir. Yani, "Calisan" rolü bu tablo üzerinde okuma ve ekleme işlemleri gerçekleştirebilir.

**GRANT SELECT (CustomerUsername, FirstName, LastName, Email, Phone) ON Customers TO Calisan;**: Bu ifade, "Calisan" rolüne "Customers" tablosunda sadece belirli sütunlara (**CustomerUsername**, **FirstName**, **LastName**, **Email**, **Phone**) **SELECT** yetkisi verir. Diğer sütunlara erişim izni verilmez.

**GRANT SELECT,INSERT, UPDATE ON Employee TO Calisan;**: Bu ifade, "Calisan" rolüne "Employee" tablosunda **SELECT**, **INSERT** ve **UPDATE** yetkilerini verir. Yani, "Calisan" rolü bu tablo üzerinde okuma, ekleme ve güncelleme işlemleri gerçekleştirebilir.

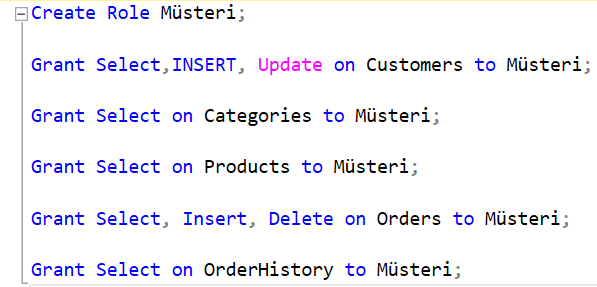
**GRANT SELECT (EmployeeUsername, EmployeePassword, FirstName, LastName, Email, Phone) ON Employee TO Calisan;**: Bu ifade, "Calisan" rolüne "Employee" tablosundaki belirli sütunlara sadece **SELECT** yetkisi verir.

**GRANT SELECT, INSERT, UPDATE, DELETE ON Products TO Calisan;**: Bu ifade, "Calisan" rolüne "Products" tablosunda **SELECT**, **INSERT**, **UPDATE** ve **DELETE** yetkilerini verir.

**GRANT SELECT, UPDATE, DELETE ON Orders TO Calisan;**: Bu ifade, "Calisan" rolüne "Orders" tablosunda **SELECT**, **UPDATE** ve **DELETE** yetkilerini verir.

**GRANT SELECT ON OrderHistory TO Calisan;**: Bu ifade, "Calisan" rolüne "OrderHistory" tablosunda sadece **SELECT** yetkisi verir.

Bu kodlar, "Calisan" rolüne belirli tablo ve sütunlarda hangi türde yetkilere sahip olacağını belirten SQL GRANT ifadelerini içerir. Bu şekilde, "Calisan" rolündeki kullanıcılar belirtilen yetkilere sahip olacak ve bu yetkiler üzerinden veritabanındaki işlemleri gerçekleştirebileceklerdir. Bu izinler, belirli tablo ve sütunlara erişim kontrolü sağlamak için kullanışlıdır.



***Şekil 3.3***

Bu SQL kodları, "Müsteri" adında bir rol oluşturulması ve bu role belirli tablolara ve işlemlere erişim izinlerinin verilmesi işlemlerini gerçekleştirir. İşte kodların kısa açıklamaları:

**Create Role Müsteri;**: Bu ifade, "Müsteri" adında bir rol oluşturur. Bu rol, belirli yetkilere sahip kullanıcıları temsil eder.

**Grant Select, INSERT, UPDATE ON Customers TO Müsteri;**: Bu ifade, "Müsteri" rolüne "Customers" tablosunda **SELECT**, **INSERT** ve **UPDATE** yetkilerini verir. Yani, "Müsteri" rolü bu tablo üzerinde okuma, ekleme ve güncelleme işlemleri gerçekleştirebilir.

**Grant Select ON Categories TO Müsteri;**: Bu ifade, "Müsteri" rolüne "Categories" tablosunda sadece **SELECT** yetkisini verir. Diğer işlemlere erişim izni verilmez.

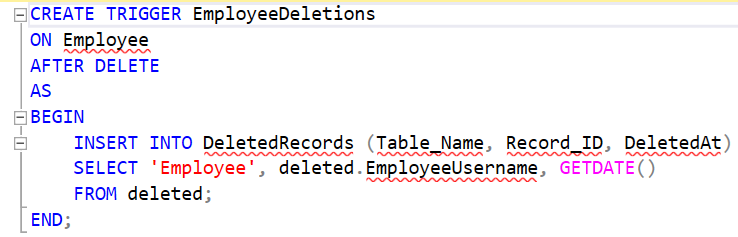
**Grant Select ON Products TO Müsteri;**: Bu ifade, "Müsteri" rolüne "Products" tablosunda sadece **SELECT** yetkisini verir.

**Grant Select, INSERT, DELETE ON Orders TO Müsteri;**: Bu ifade, "Müsteri" rolüne "Orders" tablosunda **SELECT**, **INSERT** ve **DELETE** yetkilerini verir. Güncelleme yetkisi verilmediği için **UPDATE** işlemi yapılamaz.

**Grant Select ON OrderHistory TO Müsteri;**: Bu ifade, "Müsteri" rolüne "OrderHistory" tablosunda sadece **SELECT** yetkisini verir.

Bu kodlar, "Müsteri" rolüne belirli tablo ve işlemlarda hangi türde yetkilere sahip olacağını belirten SQL GRANT ifadelerini içerir. Bu şekilde, "Müsteri" rolündeki kullanıcılar belirtilen yetkilere sahip olacak ve bu yetkiler üzerinden veritabanındaki işlemleri gerçekleştirebileceklerdir. Bu izinler, belirli tablo ve işlemlerde erişim kontrolü sağlamak için kullanışlıdır.

**4. TRİGGERLAR**



***Şekil 4.1***

Bu SQL kodu, "Employee" tablosunda gerçekleştirilen silme işlemlerini takip eden bir tetikleyici (trigger) oluşturur. İşte kodun açıklaması:

**CREATE TRIGGER EmployeeDeletions**: Bu ifade, "EmployeeDeletions" adında bir tetikleyici oluşturulacağını belirtir.

**ON Employee**: Bu kısım, tetikleyicinin hangi tablo üzerinde çalışacağını belirtir. Bu durumda, "Employee" tablosu üzerinde çalışacak.

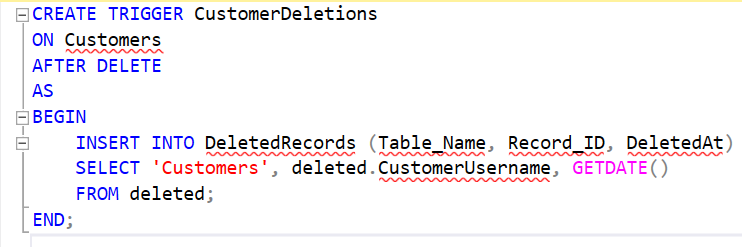
**AFTER DELETE**: Bu kısım, tetikleyicinin hangi olaydan sonra çalışacağını belirtir. Bu tetikleyici, bir kayıt silindikten sonra çalışacak.

**AS BEGIN**: Bu kısım, tetikleyici kodunun başlangıcını belirtir. Tetikleyici kodu, bu anahtar kelimenin ardından gelir.

**INSERT INTO DeletedRecords (Table\_Name, Record\_ID, DeletedAt)**: Bu ifade, "DeletedRecords" adlı tabloya yeni bir kayıt eklemek için kullanılır. Bu tablo, silinen kayıtların geçmişini tutmak amacıyla kullanılıyor.

**SELECT 'Employee', deleted.EmployeeUsername, GETDATE() FROM deleted;**: Bu kısım, "Employee" tablosundan silinen kayıtların bilgilerini seçer ve bu bilgileri "DeletedRecords" tablosuna ekler. **'Employee'** ifadesi, silinen kayıtların hangi tablodan olduğunu belirtir. **deleted.EmployeeUsername** ifadesi, silinen kaydın "Employee" tablosundaki kullanıcı adını belirtir. **GETDATE()** fonksiyonu ise silme işleminin gerçekleştiği tarih ve saat bilgisini sağlar.

Bu tetikleyici, "Employee" tablosundan bir kayıt silindiğinde çalışacak ve silinen kaydın bilgilerini "DeletedRecords" tablosuna ekleyecektir. Bu tür tetikleyiciler, veritabanında gerçekleşen belirli olayları izlemek ve bu olaylara tepki vermek amacıyla kullanılır. Bu örnekte, silinen çalışan kayıtlarının geçmişini tutmak için bir tetikleyici oluşturulmuştur.



***Şekil 4.2***

Bu SQL kodu, "Customers" tablosundaki silme işlemlerini takip eden bir tetikleyici (trigger) olan "CustomerDeletions" adında bir tetikleyici oluşturur. İşte kodun açıklaması:

**CREATE TRIGGER CustomerDeletions**: Bu ifade, "CustomerDeletions" adında bir tetikleyici oluşturulacağını belirtir.

**ON Customers**: Bu kısım, tetikleyicinin hangi tablo üzerinde çalışacağını belirtir. Bu durumda, "Customers" tablosu üzerinde çalışacak.

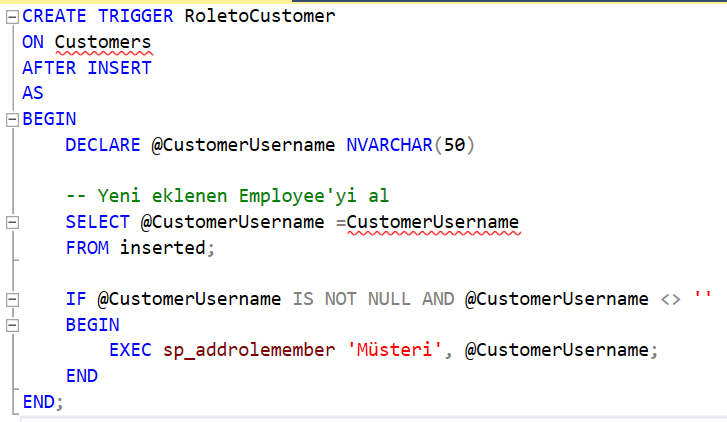
**AFTER DELETE**: Bu kısım, tetikleyicinin hangi olaydan sonra çalışacağını belirtir. Bu tetikleyici, bir kayıt silindikten sonra çalışacak.

**AS BEGIN**: Bu kısım, tetikleyici kodunun başlangıcını belirtir. Tetikleyici kodu, bu anahtar kelimenin ardından gelir.

**INSERT INTO DeletedRecords (Table\_Name, Record\_ID, DeletedAt)**: Bu ifade, "DeletedRecords" adlı tabloya yeni bir kayıt eklemek için kullanılır. Bu tablo, silinen kayıtların geçmişini tutmak amacıyla kullanılıyor.

**SELECT 'Customers', deleted.CustomerUsername, GETDATE() FROM deleted;**: Bu kısım, "Customers" tablosundan silinen kayıtların bilgilerini seçer ve bu bilgileri "DeletedRecords" tablosuna ekler. **'Customers'** ifadesi, silinen kayıtların hangi tablodan olduğunu belirtir. **deleted.CustomerUsername** ifadesi, silinen kaydın "Customers" tablosundaki müşteri kullanıcı adını belirtir. **GETDATE()** fonksiyonu ise silme işleminin gerçekleştiği tarih ve saat bilgisini sağlar.

Bu tetikleyici, "Customers" tablosundan bir müşteri kaydı silindiğinde çalışacak ve silinen kaydın bilgilerini "DeletedRecords" tablosuna ekleyecektir. Bu tür tetikleyiciler, veritabanında gerçekleşen belirli olayları izlemek ve bu olaylara tepki vermek amacıyla kullanılır. Bu örnekte, silinen müşteri kayıtlarının geçmişini tutmak için bir tetikleyici oluşturulmuştur.



***Şekil 4.3***

Bu SQL kodu, "Customers" tablosuna yeni bir müşteri eklenmesi durumunda çalışacak bir tetikleyici (trigger) olan "RoletoCustomer" adında bir tetikleyici oluşturur. İşte kodun açıklamaları:

**CREATE TRIGGER RoletoCustomer**: Bu ifade, "RoletoCustomer" adında bir tetikleyici oluşturulacağını belirtir.

**ON Customers**: Bu kısım, tetikleyicinin hangi tablo üzerinde çalışacağını belirtir. Bu durumda, "Customers" tablosu üzerinde çalışacak.

**AFTER INSERT**: Bu kısım, tetikleyicinin hangi olaydan sonra çalışacağını belirtir. Bu tetikleyici, bir kayıt eklendikten sonra çalışacak.

**AS BEGIN**: Bu kısım, tetikleyici kodunun başlangıcını belirtir. Tetikleyici kodu, bu anahtar kelimenin ardından gelir.

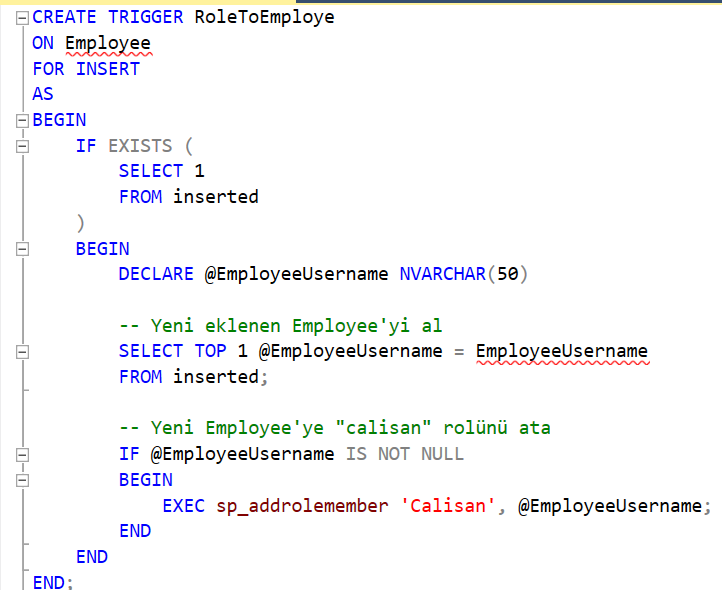
**DECLARE @CustomerUsername NVARCHAR(50)**: Bu ifade, tetikleyici içinde kullanılacak olan bir değişkenin tanımlandığını belirtir. **@CustomerUsername** adında bir NVARCHAR türünde değişken tanımlanır.

**SELECT @CustomerUsername = CustomerUsername FROM inserted;**: Bu ifade, **inserted** adlı geçici tablodan yeni eklenen müşterinin kullanıcı adını alır ve **@CustomerUsername** değişkenine atar. **inserted** tablosu, tetikleyici tarafından otomatik olarak sağlanan ve tetikleyici olayının gerçekleştiği anki durumu içeren geçici bir tablodur.

**IF @CustomerUsername IS NOT NULL AND @CustomerUsername <> ''**: Bu ifade, **@CustomerUsername** değişkeninin boş olmadığını ve null olmadığını kontrol eder.

**EXEC sp\_addrolemember 'Müsteri', @CustomerUsername;**: Bu ifade, **sp\_addrolemember** prosedürünü çağırarak "Müsteri" rolüne (**'Müsteri'**) yeni eklenen müşteriyi (**@CustomerUsername**) ekler. Bu prosedür, belirtilen rolü belirtilen kullanıcıya eklemek için kullanılır.

Bu tetikleyici, "Customers" tablosuna yeni bir müşteri eklenmesi durumunda çalışacak ve eklenen müşteriyi "Müsteri" rolüne ekleyecektir. Bu tür bir tetikleyici, kullanıcı rollerini otomatik olarak yönetmek ve güncellemek için kullanışlı olabilir.



***Şekil 4.4***

Bu SQL kodu, "Employee" tablosuna yeni bir çalışan eklenmesi durumunda çalışacak bir tetikleyici (trigger) olan "RoleToEmployee" adında bir tetikleyici oluşturur. İşte kodun açıklamaları:

**CREATE TRIGGER RoleToEmployee**: Bu ifade, "RoleToEmployee" adında bir tetikleyici oluşturulacağını belirtir.

**ON Employee**: Bu kısım, tetikleyicinin hangi tablo üzerinde çalışacağını belirtir. Bu durumda, "Employee" tablosu üzerinde çalışacak.

**FOR INSERT**: Bu kısım, tetikleyicinin hangi olaydan sonra çalışacağını belirtir. Bu tetikleyici, bir kayıt eklendikten sonra çalışacak.

**AS BEGIN**: Bu kısım, tetikleyici kodunun başlangıcını belirtir. Tetikleyici kodu, bu anahtar kelimenin ardından gelir.

**IF EXISTS (SELECT 1 FROM inserted)**: Bu ifade, **inserted** adlı geçici tabloda en az bir satırın var olup olmadığını kontrol eder. **inserted** tablosu, tetikleyici tarafından otomatik olarak sağlanan ve tetikleyici olayının gerçekleştiği anki durumu içeren geçici bir tablodur.

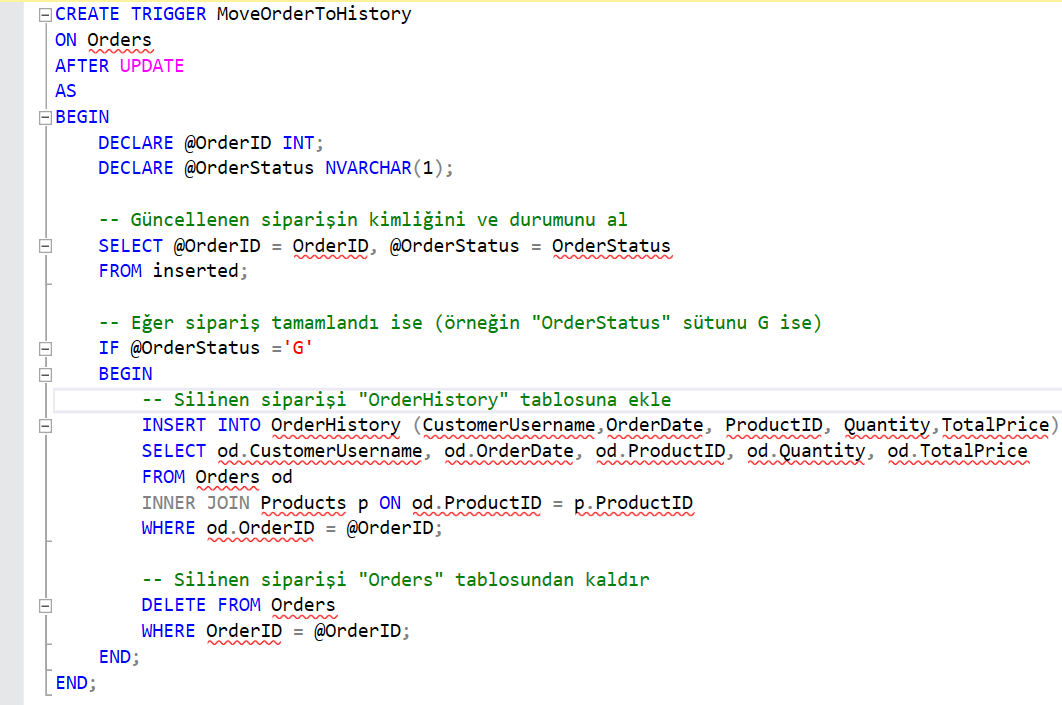
**DECLARE @EmployeeUsername NVARCHAR(50)**: Bu ifade, tetikleyici içinde kullanılacak olan bir değişkenin tanımlandığını belirtir. **@EmployeeUsername** adında bir NVARCHAR türünde değişken tanımlanır.

**SELECT TOP 1 @EmployeeUsername = EmployeeUsername FROM inserted;**: Bu ifade, **inserted** tablosundan yeni eklenen çalışanın kullanıcı adını alır ve **@EmployeeUsername** değişkenine atar. **TOP 1** ifadesi, sadece bir satır alındığını ve birden fazla sonuç bulunsa bile sadece birinin seçildiğini belirtir.

**IF @EmployeeUsername IS NOT NULL**: Bu ifade, **@EmployeeUsername** değişkeninin boş olmadığını kontrol eder.

**EXEC sp\_addrolemember 'Calisan', @EmployeeUsername;**: Bu ifade, **sp\_addrolemember** prosedürünü çağırarak "Calisan" rolüne (**'Calisan'**) yeni eklenen çalışanı (**@EmployeeUsername**) ekler. Bu prosedür, belirtilen rolü belirtilen kullanıcıya eklemek için kullanılır.

Bu tetikleyici, "Employee" tablosuna yeni bir çalışan eklenmesi durumunda çalışacak ve eklenen çalışana "Calisan" rolünü ekleyecektir. Bu tür bir tetikleyici, çalışanların otomatik olarak belirli rollerle ilişkilendirilmesini sağlamak için kullanışlı olabilir.



***Şekil 4.5***

Bu SQL kodu, "Orders" tablosunda bir siparişin durumu güncellendikten sonra çalışacak olan "MoveOrderToHistory" adlı bir tetikleyici (trigger) oluşturur. İşte kodun açıklamaları:

**CREATE TRIGGER MoveOrderToHistory**: Bu ifade, "MoveOrderToHistory" adında bir tetikleyici oluşturulacağını belirtir.

**ON Orders**: Bu kısım, tetikleyicinin hangi tablo üzerinde çalışacağını belirtir. Bu durumda, "Orders" tablosu üzerinde çalışacak.

**AFTER UPDATE**: Bu kısım, tetikleyicinin hangi olaydan sonra çalışacağını belirtir. Bu tetikleyici, bir kayıt güncellendikten sonra çalışacak.

**AS BEGIN**: Bu kısım, tetikleyici kodunun başlangıcını belirtir. Tetikleyici kodu, bu anahtar kelimenin ardından gelir.

**DECLARE @OrderID INT;** ve **DECLARE @OrderStatus NVARCHAR(1);**: Bu ifadeler, tetikleyici içinde kullanılacak olan iki değişkenin tanımlandığını belirtir. **@OrderID**, güncellenen siparişin kimliğini (ID) taşırken, **@OrderStatus**, güncellenen siparişin yeni durumunu taşır.

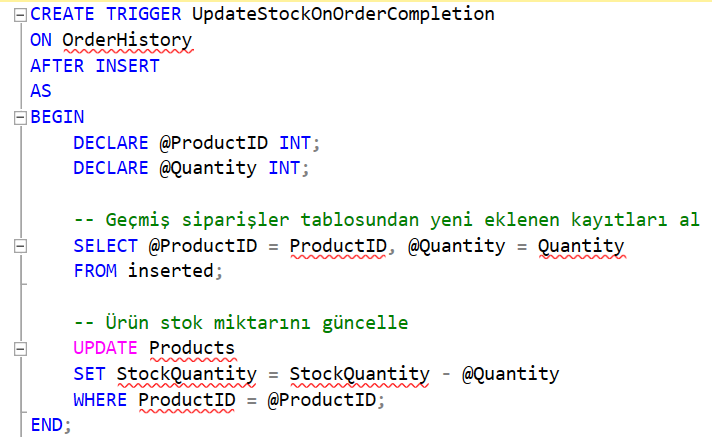
**SELECT @OrderID = OrderID, @OrderStatus = OrderStatus FROM inserted;**: Bu ifade, **inserted** adlı geçici tablodan güncellenen siparişin ID'sini ve durumunu alır. **inserted** tablosu, tetikleyici tarafından otomatik olarak sağlanan ve tetikleyici olayının gerçekleştiği anki durumu içeren geçici bir tablodur.

**IF @OrderStatus ='G'**: Bu ifade, siparişin yeni durumunun 'G' (tamamlandı) olup olmadığını kontrol eder.

**INSERT INTO OrderHistory (...) SELECT ... FROM Orders ...**: Bu ifade, eğer sipariş tamamlandıysa, bu siparişi "OrderHistory" tablosuna ekler. Bu işlem, tamamlanan siparişin detaylarını "OrderHistory" tablosuna taşır.

**DELETE FROM Orders WHERE OrderID = @OrderID;**: Bu ifade, tamamlanan siparişi "Orders" tablosundan kaldırır.

Bu tetikleyici, bir siparişin durumu 'G' (tamamlandı) olarak güncellendiğinde çalışacak ve tamamlanan siparişi "OrderHistory" tablosuna ekleyip "Orders" tablosundan kaldıracaktır. Bu tür bir tetikleyici, tamamlanan siparişleri geçmişe taşımak ve sipariş geçmişi oluşturmak için kullanışlıdır.



***Şekil 4.6***

Bu SQL kodu, "OrderHistory" tablosuna yeni bir sipariş eklendiğinde çalışacak bir tetikleyici olan "UpdateStockOnOrderCompletion" adında bir tetikleyici oluşturur. İşte kodun açıklamaları:

**CREATE TRIGGER UpdateStockOnOrderCompletion**: Bu ifade, "UpdateStockOnOrderCompletion" adında bir tetikleyici oluşturulacağını belirtir.

**ON OrderHistory**: Bu kısım, tetikleyicinin hangi tablo üzerinde çalışacağını belirtir. Bu durumda, "OrderHistory" tablosu üzerinde çalışacak.

**AFTER INSERT**: Bu kısım, tetikleyicinin hangi olaydan sonra çalışacağını belirtir. Bu tetikleyici, bir kayıt eklendikten sonra çalışacak.

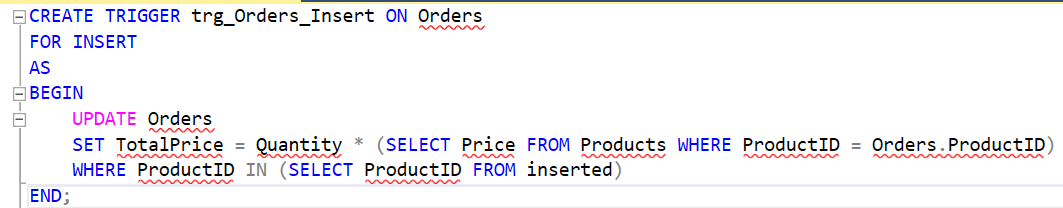
**AS BEGIN**: Bu kısım, tetikleyici kodunun başlangıcını belirtir. Tetikleyici kodu, bu anahtar kelimenin ardından gelir.

**DECLARE @ProductID INT;** ve **DECLARE @Quantity INT;**: Bu ifadeler, tetikleyici içinde kullanılacak olan iki değişkenin tanımlandığını belirtir. **@ProductID**, yeni eklenen siparişin ürün ID'sini taşırken, **@Quantity**, yeni eklenen siparişin miktarını taşır.

**SELECT @ProductID = ProductID, @Quantity = Quantity FROM inserted;**: Bu ifade, **inserted** adlı geçici tablodan yeni eklenen siparişin ürün ID'sini ve miktarını alır. **inserted** tablosu, tetikleyici tarafından otomatik olarak sağlanan ve tetikleyici olayının gerçekleştiği anki durumu içeren geçici bir tablodur.

**UPDATE Products SET StockQuantity = StockQuantity - @Quantity WHERE ProductID = @ProductID;**: Bu ifade, yeni eklenen siparişin ürününün stok miktarını günceller. Yeni eklenen siparişin miktarı kadar, ilgili ürünün stok miktarından çıkartılır.

Bu tetikleyici, "OrderHistory" tablosuna yeni bir sipariş eklendiğinde çalışacak ve ilgili ürünün stok miktarını güncelleyecektir. Bu tür bir tetikleyici, stok takibi yapmak ve sipariş geçmişini güncellemek için kullanışlı olabilir.



***Şekil 4.7***

Bu SQL kodu, "Orders" tablosuna yeni bir sipariş eklenmesi durumunda çalışacak bir tetikleyici olan "trg\_Orders\_Insert" adlı bir tetikleyici oluşturur. İşte kodun açıklamaları:

**CREATE TRIGGER trg\_Orders\_Insert**: Bu ifade, "trg\_Orders\_Insert" adında bir tetikleyici oluşturulacağını belirtir.

**ON Orders**: Bu kısım, tetikleyicinin hangi tablo üzerinde çalışacağını belirtir. Bu durumda, "Orders" tablosu üzerinde çalışacak.

**FOR INSERT**: Bu kısım, tetikleyicinin hangi olaydan sonra çalışacağını belirtir. Bu tetikleyici, bir kayıt eklendikten sonra çalışacak.

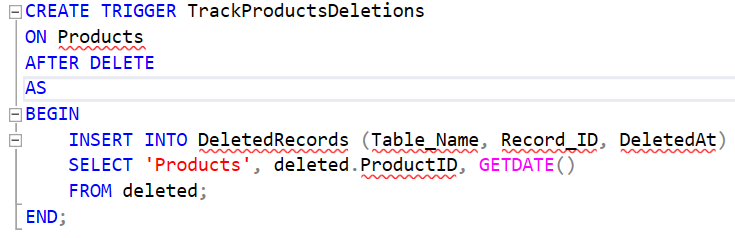
**AS BEGIN**: Bu kısım, tetikleyici kodunun başlangıcını belirtir. Tetikleyici kodu, bu anahtar kelimenin ardından gelir.

**UPDATE Orders SET TotalPrice = ...**: Bu ifade, yeni eklenen siparişlerin toplam fiyatlarını günceller. Güncelleme işlemi, "Orders" tablosundaki ilgili siparişlerin "TotalPrice" sütununu, o siparişin miktarı ile ilgili ürünün birim fiyatının çarpımıyla hesaplanan değere günceller.

**Quantity \* (SELECT Price FROM Products WHERE ProductID = Orders.ProductID)**: Bu kısım, her bir siparişin miktarını, o siparişin bağlı olduğu ürünün birim fiyatıyla çarpar. Bu, siparişin toplam fiyatını temsil eder.

**WHERE ProductID IN (SELECT ProductID FROM inserted)**: Bu kısım, güncelleme işleminin, "inserted" adlı geçici tablo içindeki ürünleri içermesi için bir filtre ekler. "inserted" tablosu, tetikleyici tarafından sağlanan ve tetikleyici olayının gerçekleştiği anki durumu içeren geçici bir tablodur.

Bu tetikleyici, "Orders" tablosuna yeni bir sipariş eklenmesi durumunda çalışacak ve eklenen siparişlere ait toplam fiyatları güncelleyecektir.



***Şekil 4.8***

Bu SQL kodu, "Products" tablosundan bir ürün silindiğinde çalışacak bir tetikleyici olan "TrackProductsDeletions" adlı bir tetikleyici oluşturur. İşte kodun açıklamaları:

**CREATE TRIGGER TrackProductsDeletions**: Bu ifade, "TrackProductsDeletions" adında bir tetikleyici oluşturulacağını belirtir.

**ON Products**: Bu kısım, tetikleyicinin hangi tablo üzerinde çalışacağını belirtir. Bu durumda, "Products" tablosu üzerinde çalışacak.

**AFTER DELETE**: Bu kısım, tetikleyicinin hangi olaydan sonra çalışacağını belirtir. Bu tetikleyici, bir kayıt silindikten sonra çalışacak.

**AS BEGIN**: Bu kısım, tetikleyici kodunun başlangıcını belirtir. Tetikleyici kodu, bu anahtar kelimenin ardından gelir.

**INSERT INTO DeletedRecords ...**: Bu ifade, "DeletedRecords" adlı tabloya silinen ürünün bilgilerini ekler.

**SELECT 'Products', deleted.ProductID, GETDATE() FROM deleted;**: Bu kısım, "Products" tablosundan silinen bir ürünün bilgilerini, "DeletedRecords" tablosuna ekler.

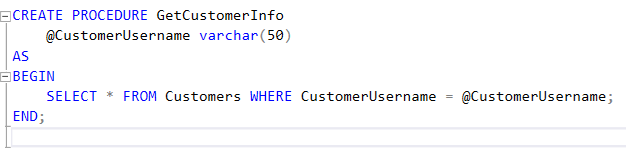
**'Products'**: Bu ifade, silinen kaydın hangi tabloya ait olduğunu belirtir. Bu durumda, "Products" tablosu.

**deleted.ProductID**: Bu ifade, silinen ürünün ID'sini alır. **deleted** adlı geçici tablo, tetikleyici tarafından sağlanan ve tetikleyici olayının gerçekleştiği anki durumu içeren geçici bir tablodur.

**GETDATE()**: Bu ifade, silme işleminin gerçekleştiği tarih ve saati alır ve bu bilgiyi "DeletedRecords" tablosuna ekler.

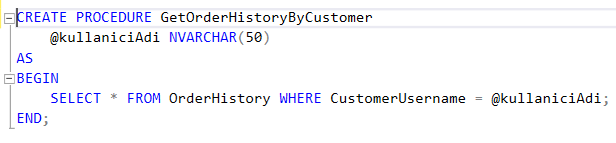
Bu tetikleyici, "Products" tablosundan bir ürün silindiğinde çalışacak ve silinen ürünün bilgilerini "DeletedRecords" tablosuna ekleyecektir. Bu tür bir tetikleyici, silinen kayıtları takip etmek ve geçmiş kayıtları saklamak için kullanışlı olabilir.

**5 PROSEDÜRLER**

****

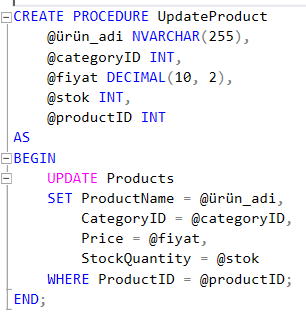
***Şekil 5.1***

Şekil 5.1’de görünen GetCustomerInfo Prosedürü, Customer Tablosundaki verileri @customerUsername e göre filtreleme sorgusu işlemi görmektedir.



***Şekil 5.2***

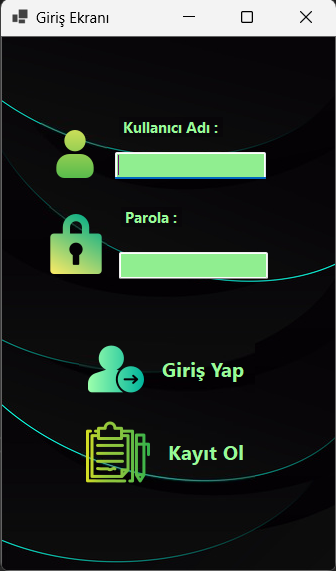
Şekil 5.2’de görünen GetOrderHistoryByCustomer Prosedürü, OrderHistory Tablosundaki verileri CustomerUsername’i @kullaniciAdi olan veriye göre filtreleme sorgusu işlemi görmektedir.

****

***Şekil 5.3***

Şekil 5.3’de görünen UpdateProduct Prosedürü, Product Tablosundaki verileri girilen verilere göre güncelleme işlemi görmektedir.

**6. UYGULAMA GİRİŞ EKRANI**



***Şekil 6.1***

Şekil 6.1’de görünen C# programı, bir kullanıcının giriş yapmasını sağlayan bir Windows Forms uygulamasını temsil eder. Aşağıda kodun belirli bölümlerini açıklıyorum:

**6.1 Bağlantı Dizesi:**

string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial Catalog=OnlineAlısveris;Integrated Security=True";

Bu satır, SQL Server veritabanına bağlanmak için kullanılan bağlantı dizesini tanımlar. **Data Source** ile sunucu adı belirtilir, **Initial Catalog** ile veritabanı adı belirtilir ve **Integrated Security=True** ile Windows kimlik doğrulamasının kullanılacağını belirtir.

**6.2 Giriş Butonu İşlevi:**

private void button1\_Click(object sender, EventArgs e)

Bu metod, giriş butonuna tıklandığında çalışır. Kullanıcının girdiği kullanıcı adı ve şifreyi alır, SQL sorgularını kullanarak veritabanında bu bilgilerle eşleşen kayıtları kontrol eder ve kullanıcı türüne göre ilgili pencereyi açar.

**6.3 SQL Sorguları:**

string query = "SELECT CustomerUsername, CustomerPassword FROM Customers WHERE CustomerUsername= @kullanici\_adi AND CustomerPassword= @sifre"; string query2 = "SELECT EmployeeUsername, EmployeePassword FROM Employee WHERE EmployeeUsername= @kullanici\_adi2 AND EmployeePassword= @sifre2"; string query3 = "SELECT AdminUserName, AdminPassword FROM Admin WHERE AdminUserName= @kullanici\_adi3 AND AdminPassword= @sifre3";

Bu sorgular, sırasıyla müşteri, çalışan ve yönetici tablolarında kullanıcı adı ve şifre eşleşmelerini kontrol eder. **@kullanici\_adi**, **@sifre** gibi parametreler, SQL sorgularına kullanıcı giriş bilgilerini eklemek için kullanılır.

**6.4 Bağlantı Açma ve SQL Sorguları Çalıştırma:**

using (SqlConnection connection = new SqlConnection(connectionString))

Bu blok, **SqlConnection** sınıfını kullanarak veritabanına bağlanmayı sağlar. **using** bloğu kullanıldığı için, bağlantı işlemi bittiğinde otomatik olarak kapatılır.

**6.5 Kullanıcı Türüne Göre Pencere Açma:**

MusteriGiris pCustomer = new MusteriGiris(); pCustomer.KAd(kullanici\_adi); pCustomer.Show(); this.Hide();

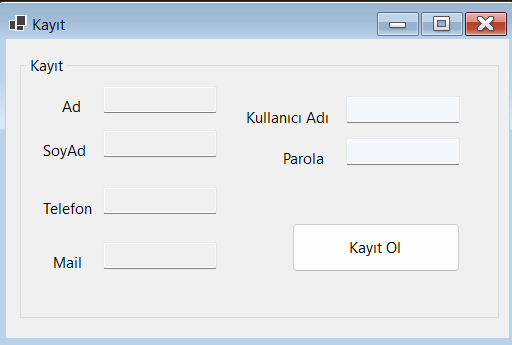
Bu bölüm, kullanıcı türüne bağlı olarak ilgili pencereyi açar. Örneğin, müşteri ise **MusteriGiris** penceresini, çalışan ise **personelgiris** penceresini açar. **this.Hide()** ile ise giriş yapan kullanıcı giriş penceresinin kapatılmasını sağlar.

**6.6 Kullanıcı Kaydı Butonu İşlevi:**

private void button2\_Click(object sender, EventArgs e)

Bu metod, "Kayıt" butonuna tıklanıldığında çalışır ve kullanıcıyı kayıt penceresine yönlendirir.

**7. KAYIT OLMA EKRANI**



***Şekil 7.1***

Şekil 7.1’de görünen C# programı, kullanıcı kaydı eklemek için kullanılan bir Windows Forms uygulamasını temsil eder. İşte bu kodun önemli bölümlerini açıklamalarıyla birlikte:

**7.1 Bağlantı Dizesi:**

string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial Catalog=OnlineAlısveris;Integrated Security=True";

Bu satır, SQL Server veritabanına bağlanmak için kullanılan bağlantı dizesini tanımlar. **Data Source** ile sunucu adı belirtilir, **Initial Catalog** ile veritabanı adı belirtilir ve **Integrated Security=True** ile Windows kimlik doğrulamasının kullanılacağını belirtir.

**7.2 Bağlantı Açma Metodu:**

public void Baglanti()

Bu metod, veritabanına bağlantı açmayı sağlar. Ancak bu metod, kod içinde çağrılmamış gibi görünüyor.

**7.3 Kayıt Ekleme Butonu İşlevi:**

private void button1\_Click(object sender, EventArgs e)

Bu metod, kullanıcı kaydı eklemek için kullanılan butonun tıklama olayını temsil eder. Kullanıcının girdiği bilgileri alır ve bu bilgilerle SQL sorgusunu çalıştırarak veritabanına yeni bir kullanıcı ekler.

**7.4 SQL Sorgusu:**

string query = "INSERT INTO Customers(CustomerUsername, CustomerPassword, FirstName, LastName, Phone, Email)VALUES(@kullanici\_adi, @sifre, @ad, @soyAd, @telefon, @mail)";

Bu sorgu, **Customers** tablosuna yeni bir kayıt eklemek için kullanılır. **VALUES** kısmında **@kullanici\_adi**, **@sifre**, **@ad**, **@soyAd**, **@telefon**, ve **@mail** parametreleri, kullanıcının girdiği bilgilerle değiştirilecek yer tutuculardır.

**7.5 Parametreli SQL Sorgusu Çalıştırma:**

using (SqlCommand command = new SqlCommand(query, connection))

Bu blok, parametreli SQL sorgusunu çalıştırmak için kullanılır. **@kullanici\_adi**, **@sifre**, vb. gibi parametreler, kullanıcının girdiği bilgilerle doldurulur.

**7.6 Kullanıcı Ekleme ve Sonuç Kontrolü:**

int affectedRows = command.ExecuteNonQuery(); if (affectedRows > 0)

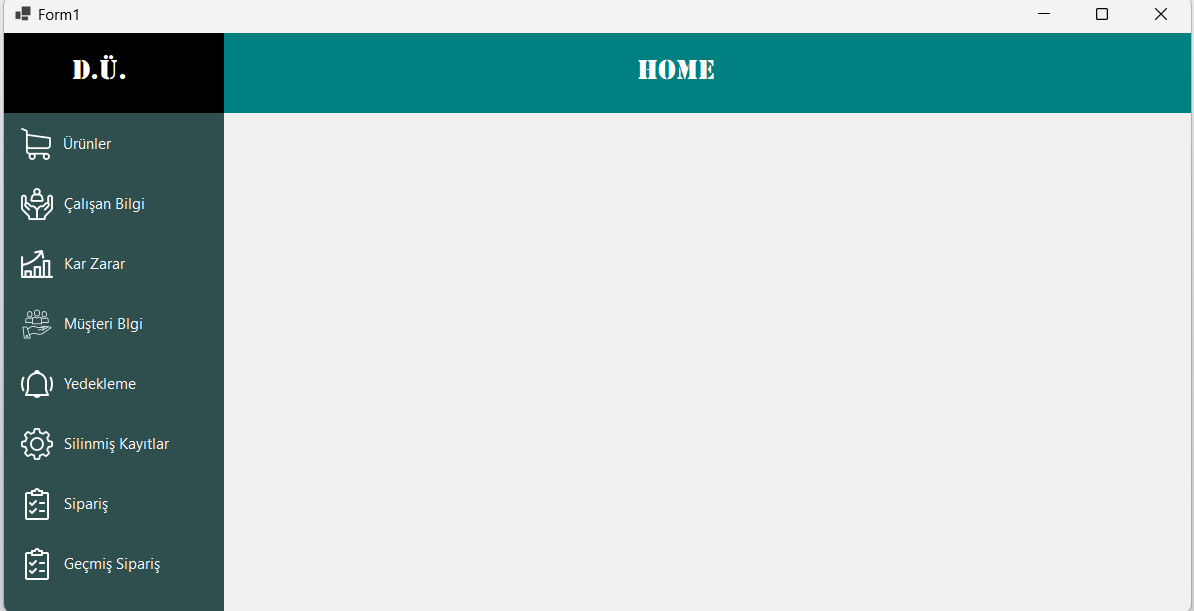
**ExecuteNonQuery** metodu, sorguyu çalıştırarak etkilenen satır sayısını döndürür. Eğer en az bir satır etkilenmişse, yani yeni bir kullanıcı başarıyla eklenmişse, kullanıcıya bir mesaj gösterilir. Aksi takdirde, eklenme işlemi başarısız olmuştur.

**7.7 Form Geçişi:**

Form pgiris = new Form1(); pgiris.Show(); this.Hide();

Eğer yeni kullanıcı başarıyla eklenmişse, kullanıcıyı giriş formuna yönlendirmek için bu kod kullanılır. **this.Hide()** ile ise kayıt formu kapatılır.

**8. ADMİN GİRİŞ EKRANI**



***Şekil 8.1***

Şekil 8.1’de görünen programı, bir Windows Forms uygulamasını temsil eder ve bir admin panelini içerir. Aşağıda, Admin sınıfındaki önemli yöntem ve olayları açıklıyorum:

**8.1 Admin Sınıfı:**

public partial class Admin : Form { // Constructor (Yapıcı Metot) public Admin() { InitializeComponent(); } }

Bu kısım, **Admin** sınıfını tanımlar. **partial** anahtar kelimesi, sınıfın aynı isimde başka bir dosyada da tanımlı olabileceğini belirtir.

**8.2 Butonlara Tıklama İşlemi:**

private void button1\_Click(object sender, EventArgs e) { Form pPersonel = new Personel(); pPersonel.ShowDialog(); } private void button2\_Click(object sender, EventArgs e) { Form pMusteri = new AdminMusteri(); pMusteri.ShowDialog(); } // Diğer buton olayları benzer şekilde tanımlanmıştır.

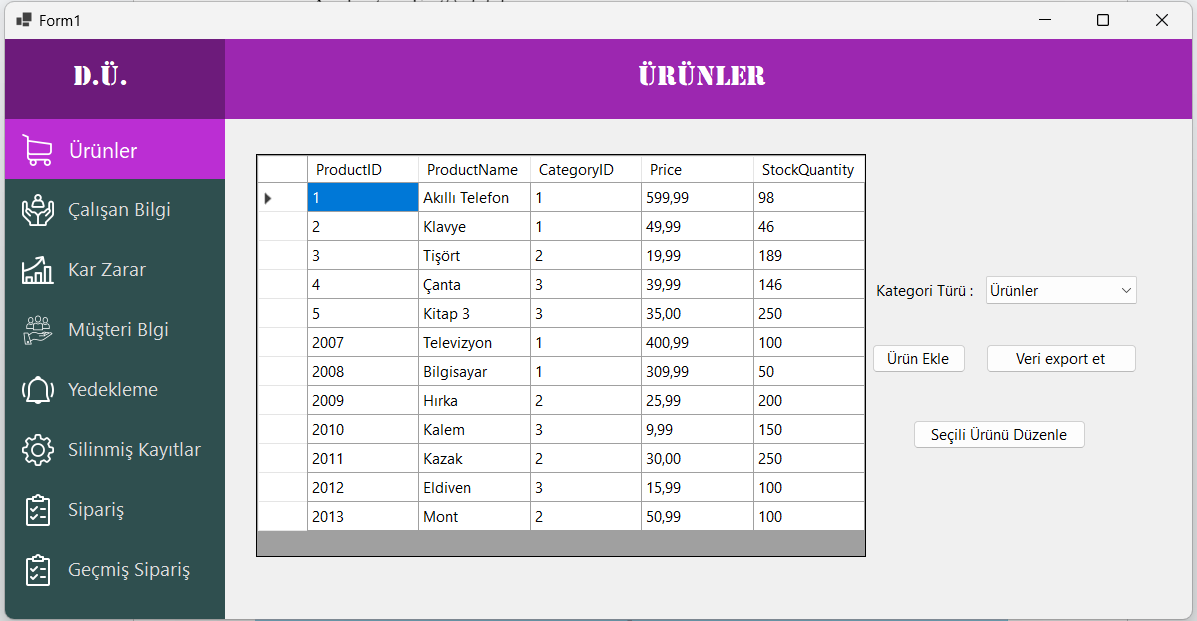
Bu metodlar, her bir butona tıklandığında hangi formun açılacağını belirler. Örneğin, **button1\_Click** metodunda **Personel** formu açılır.

**8.3 Form Kapatma İşlemi:**

private void Admin\_FormClosing(object sender, FormClosingEventArgs e) { // Form kapatılma olayı kontrol edilir. if (e.CloseReason == CloseReason.UserClosing) { DialogResult result = MessageBox.Show("Uygulamayı kapatmak istediğinizden emin misiniz?", "Uyarı", MessageBoxButtons.YesNo, MessageBoxIcon.Question); if (result == DialogResult.No) { e.Cancel = true; // Kapatmayı iptal et } else { Application.Exit(); } } }

Bu metod, admin panelinin X tuşuna basılarak kapatılmak istendiğinde bir uyarı penceresi gösterir ve kullanıcının kararına göre uygulamayı kapatıp kapatmama işlemi gerçekleştirir.

**9. ÜRÜNLER FORMU**



***Şekil 9.1***

Şekil 9.1’de görünen C# programı, bir ürün formunu temsil eder. Bu formda, bir kategori seçimiyle ilişkilendirilmiş ürünleri gösteren bir DataGridView, ürünleri düzenlemek veya eklemek için düğmeler ve verileri Excel'e dışa aktarmak için bir düğme bulunmaktadır. İşte bu formun bazı önemli bölümleri:

**9.1 ComboBox ve DataGridView Doldurma:**

private void FillComboBox() { // Kategorileri ComboBox'a dolduran metod // ... } public void FillDataGridView() { // Seçilen kategoriye göre ürünleri DataGridView'e dolduran metod // ... } private void Urun\_Load(object sender, EventArgs e) { // Form yüklendiğinde çalışacak metodlar FillComboBox(); FillDataGridView(); }

Bu metodlar, ComboBox'ı kategorilerle doldurmak ve DataGridView'i seçilen kategoriye göre doldurmak için kullanılır.

**9.2 ComboBox Seçim Değişikliği Olayı:**

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e) { // ComboBox'taki seçim değiştiğinde çalışan olay // ... }

Bu olay, ComboBox'taki seçim değiştiğinde çağrılır ve seçilen kategoriye göre DataGridView'i günceller.

**9.3 Ürün Ekle ve Düzenle Düğmeleri:**

private void button2\_Click(object sender, EventArgs e) { // Yeni ürün eklemek için kullanılan metod // ... } private void button3\_Click(object sender, EventArgs e) { // Seçilen ürünü düzenlemek için kullanılan metod // ... }

Bu metodlar, sırasıyla yeni bir ürün eklemek ve seçilen bir ürünü düzenlemek için kullanılır.

**9.4 Excel'e Veri Aktarma:**

private void buttonExp\_Click(object sender, EventArgs e) { // DataGridView'deki verileri Excel'e aktaran metod // ... }

Bu metod, DataGridView'deki verileri bir Excel dosyasına aktarır.

**9.5 GetYourData Metodu:**

private DataTable GetYourData() { // Products tablosundan veri çeken ve DataTable'a dolduran metod // ... }

Bu metod, Products tablosundan verileri çeker ve bir DataTable'a doldurur. Excel'e aktarma işleminde kullanılır.

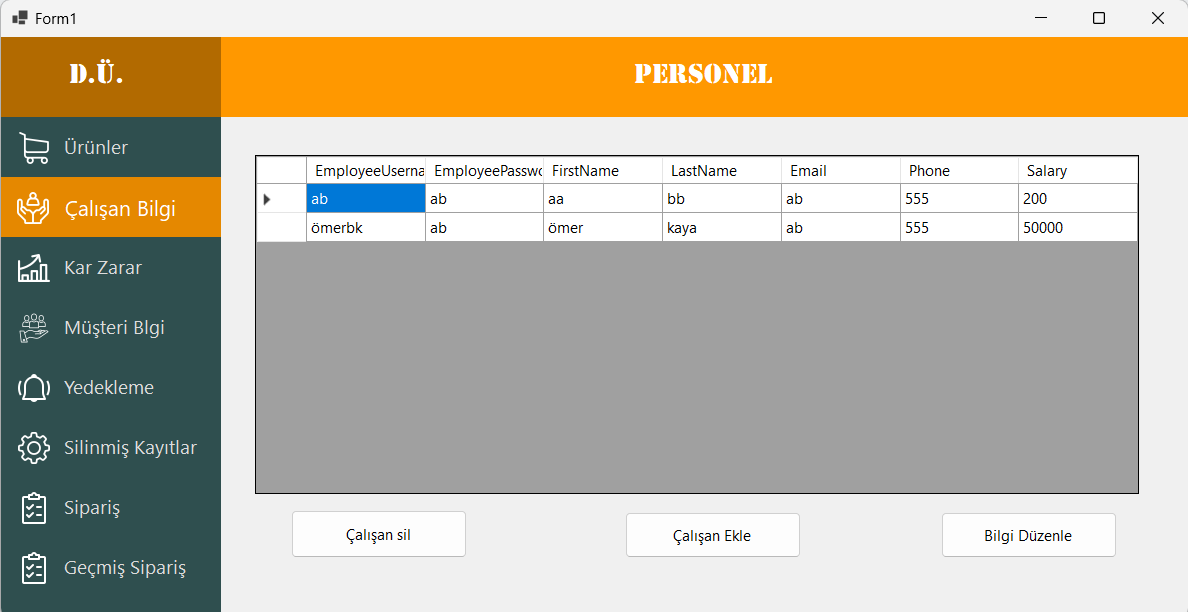
**9.6 Excel Export Düğmesi:**

using (SaveFileDialog sfd = new SaveFileDialog() { Filter = "Excel Workbook|\*.xlsx" }) { // Excel'e veri aktarmak için SaveFileDialog ve XLWorkbook kullanılır // ... }

Bu düğme, Excel'e veri aktarma işlemi için bir SaveFileDialog kullanarak bir Excel dosyasını seçme imkanı sağlar.

Bu program, kullanıcıların kategorilere göre ürünleri görüntülemelerine, düzenlemelerine, yeni ürünler eklemelerine ve DataGridView'deki verileri Excel dosyasına aktarmalarına olanak tanır.

**10. Çalışan Bilgi Formu**



***Şekil 10.1***

Şekil 10.1’de görünen C# programı, bir Personel formunu temsil eder. İşte **Personel** sınıfında bulunan önemli kısımlar:

**10.1 Veritabanı Bağlantı Dizesi:**

Veritabanı bağlantı dizesi string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial Catalog=OnlineAlısveris;Integrated Security=True";

Bu kısım, SQL Server veritabanına bağlanmak için kullanılan bağlantı dizesini içerir.

**10.2 Constructor (Yapıcı Metot):**

public Personel() { InitializeComponent(); }

Bu metot, **Personel** sınıfının bir örneğini oluşturur ve formun bileşenlerini başlatır.

**10.3 Veri Gösterme Metodu (goster):**

public void goster() { dataGridView1.DataSource = null; using (SqlConnection connection = new SqlConnection(connectionString)) { string query = "SELECT \* FROM Employee"; SqlDataAdapter adapter = new SqlDataAdapter(query, connection); DataTable dataTable = new DataTable(); adapter.Fill(dataTable); dataGridView1.DataSource = dataTable; } }

Bu metot, **Employee** tablosundaki verileri çeker ve bunları **dataGridView1** adlı DataGridView bileşenine yükler.

**10.4 Silme Butonu Olayı (button1\_Click):**

private void button1\_Click(object sender, EventArgs e) { if (dataGridView1.SelectedRows.Count > 0) { // Seçilen satırdaki veriyi silme işlemi // ... dataGridView1.Rows.RemoveAt(selectedIndex); } }

Bu metot, **dataGridView1** üzerinde seçili olan bir satırı siler ve aynı zamanda bu satırı SQL Server veritabanından da siler.

**10.5 Form Açılışında Veri Gösterme Olayı (Personel\_Load):**

private void Personel\_Load(object sender, EventArgs e) { dataGridView1\_CellContentClick(sender, new DataGridViewCellEventArgs(0, 0)); }

Bu metot, form yüklendiğinde **dataGridView1\_CellContentClick** metodunu çağırarak verileri gösterir.

**10.6 Diğer Buton Olayları:**

* + **button2\_Click**: Yeni bir personel kaydı eklemek için başka bir formu açar.
  + **button3\_Click**: Seçilen personelin bilgilerini düzenlemek için başka bir formu açar.
  + **button4\_Click**: Bu butonun herhangi bir işlevi bulunmuyor.

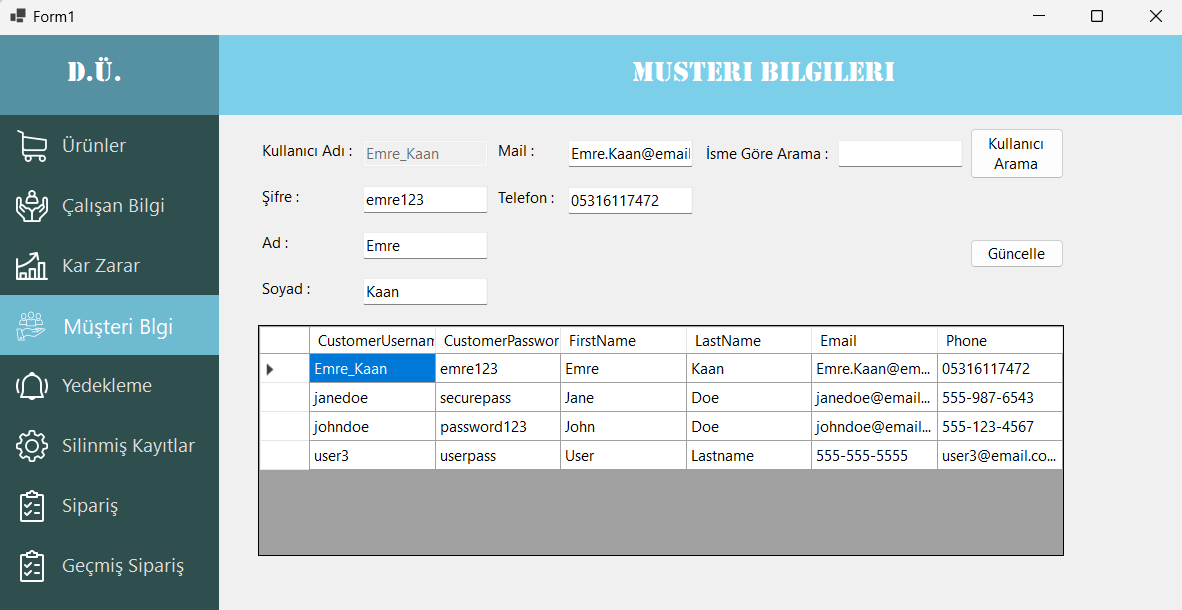
**10.7 DataGridView Hücre İçeriği Olayı (dataGridView1\_CellContentClick):**

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e) { using (SqlConnection connection = new SqlConnection(connectionString)) { string query = "SELECT \* FROM Employee"; SqlDataAdapter adapter = new SqlDataAdapter(query, connection); DataTable dataTable = new DataTable(); adapter.Fill(dataTable); dataGridView1.DataSource = dataTable; } }

Bu metot, **dataGridView1** içeriği değiştiğinde (**CellContentClick** olayı tetiklendiğinde) verileri tekrar yükler.

Bu kodlar, bir personel yönetim formu tasarlamak ve işlevselliğini sağlamak için kullanılır.

**11. MÜŞTERİ BİLGİ FORMU**



***Şekil 11.1***

Şekil 11.1’de görünen C# programı, bir yönetici tarafından müşteri bilgilerini düzenlemek ve müşteri aramak için kullanılan bir formu temsil eder. İşte **AdminMusteri** sınıfındaki önemli bölümler:

**11.1 Bağlantı ve Komut Nesneleri:**

SqlConnection connection; SqlCommand komut; SqlDataAdapter da;

Bu değişkenler, SQL Server veritabanına bağlanmak ve sorguları çalıştırmak için gerekli nesneleri içerir.

**11.2 MusteriGetir Metodu:**

void MusteriGetir() { // Veritabanından müşteri bilgilerini çeken ve DataGridView'e yükleyen metod // ... }

Bu metot, **Customers** tablosundaki müşteri bilgilerini çeker ve bunları **dataGridView1** adlı DataGridView bileşenine yükler.

**11.3 Musteri\_Load Olayı:**

private void Musteri\_Load(object sender, EventArgs e) { MusteriGetir(); }

Bu olay, form yüklendiğinde **MusteriGetir** metodunu çağırarak müşteri bilgilerini gösterir.

**11.4 dataGridView1\_CellEnter Olayı:**

private void dataGridView1\_CellEnter(object sender, DataGridViewCellEventArgs e) { // Seçilen hücreye girildiğinde ilgili müşteri bilgilerini TextBox'lara yükleyen olay // ... }

Bu olay, **dataGridView1** üzerinde bir hücreye girildiğinde ilgili müşteri bilgilerini TextBox'lara yükler.

**11.5 btnDuzenle\_Click Olayı:**

private void btnDuzenle\_Click(object sender, EventArgs e) { // TextBox'lardan alınan bilgilerle müşteri bilgilerini güncelleyen olay // ... }

Bu olay, TextBox'lardan alınan güncellenmiş müşteri bilgilerini kullanarak **Customers** tablosundaki veriyi günceller.

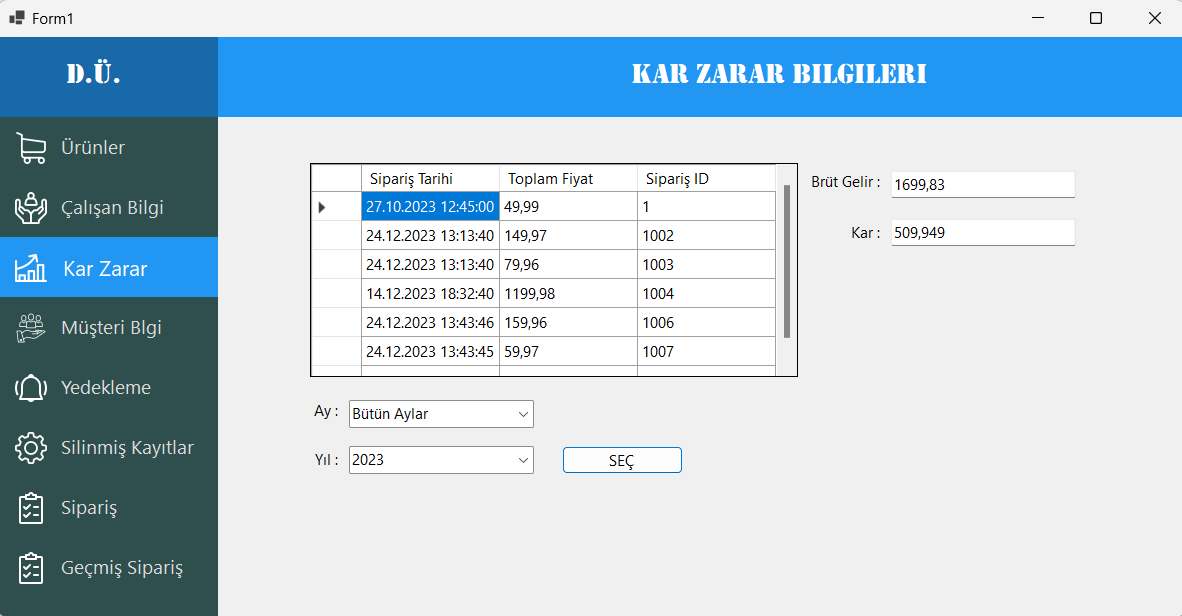
**11.6 btnAra\_Click Olayı:**

private void btnAra\_Click(object sender, EventArgs e) { // TextBox'tan alınan isme göre müşteri arayan ve sonuçları gösteren olay // ... }

Bu olay, **txtArama** TextBox'ındaki isme göre müşteri arar ve sonuçları **dataGridView1**'de gösterir.

Bu kodlar, yönetici tarafından müşteri bilgilerini düzenlemek ve aramak için kullanılan bir formu uygular.

**12. Kar Zarar Formu**



***Şekil 12.1***

Şekil 12.1’de görünen C# programı, bir kar ve zarar hesaplama formunu temsil eder. Form, bir ComboBox ile yıl ve ayları seçme, bir DataGridView ile seçilen yıl ve ayın siparişlerini görüntüleme ve birkaç hesaplama yapma yeteneğine sahiptir. İşte programın önemli bölümleri:

**12.1 Form Load Olayı:**

private void KarZarar\_Load(object sender, EventArgs e) { // Form yüklendiğinde çalışacak metodlar // ... }

Bu metod, form yüklendiğinde ComboBox'ı yıllarla doldurur ve DataGridView için sütunları ayarlar.

**12.2 Yılları ve Ayları ComboBox'a Ekleme:**

for (int year = DateTime.Now.Year; year >= 2000; year--) { comboBox2.Items.Add(year.ToString()); } comboBox1.Items.Add("Bütün Aylar"); string[] monthNames = System.Globalization.CultureInfo.CurrentCulture.DateTimeFormat.MonthGenitiveNames; for (int month = 1; month <= 12; month++) { comboBox1.Items.Add(monthNames[month - 1]); }

Bu kod, ComboBox'a yılları ve ayları ekler.

**12.3 Veri Yükleme ve Hesaplama Metodları:**

private void btnSelect\_Click(object sender, EventArgs e) { // Seçilen yılın verilerini DataGridView'e yükleyen ve ikinci TextBox'ı hesaplayan metod LoadDataForSelectedYear(); CalculateAndDisplaySecondTextBox(); } private void LoadDataForSelectedYear() { // Seçilen yılın verilerini DataGridView'e yükleyen metod // ... } private void CalculateAndDisplaySecondTextBox() { // İkinci TextBox'ı hesaplayan ve görüntüleyen metod // ... }

**LoadDataForSelectedYear** metodu, seçilen yılda sipariş verilerini DataGridView'e yükler. **CalculateAndDisplaySecondTextBox** metodu, toplam fiyat üzerinden yüzde 30'u hesaplar ve ikinci TextBox'a gösterir.

**12.4 Toplam Fiyat Hesaplama Metodları:**

private void CalculateTotalPriceForSelectedMounth() { // Belirli bir ay seçildiğinde toplam fiyatı hesaplayan metod // ... } private void CalculateTotalPriceForSelectedYear() { // Belirli bir yıl seçildiğinde toplam fiyatı hesaplayan metod // ... }

Bu metotlar, belirli bir ay veya yıldaki toplam fiyatı hesaplar.

**13 SİPARİŞ FORMLARI**

**13.1 Admin\_Gecmis\_Siparis Formu:**

public partial class Admin\_Gecmis\_Siparis : Form { // ... }

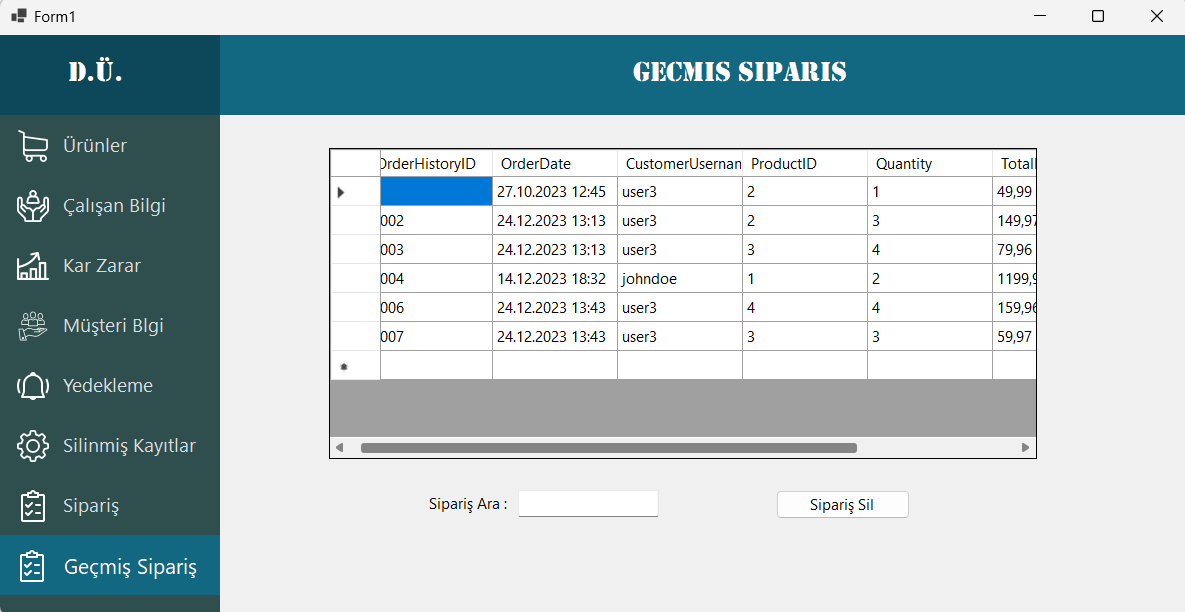
Bu kod, geçmiş siparişleri görüntülemek için kullanılan **Admin\_Gecmis\_Siparis** adlı bir form sınıfını tanımlar. Formun içeriği burada yer almıyor.

**13.2 PersonelSipariş Formu:**

public partial class PersonelSipariş : Form { // ... }

Bu kod, personel tarafından yeni sipariş almak için kullanılan **PersonelSipariş** adlı bir form sınıfını tanımlar. Formun içeriği burada yer almıyor.

Bu program, sipariş yönetimi uygulamasının ana formunu ve bu form üzerinden diğer formlara geçiş yapabilen butonları içerir. Her bir buton, farklı bir işlevi yerine getiren bir alt form açar.



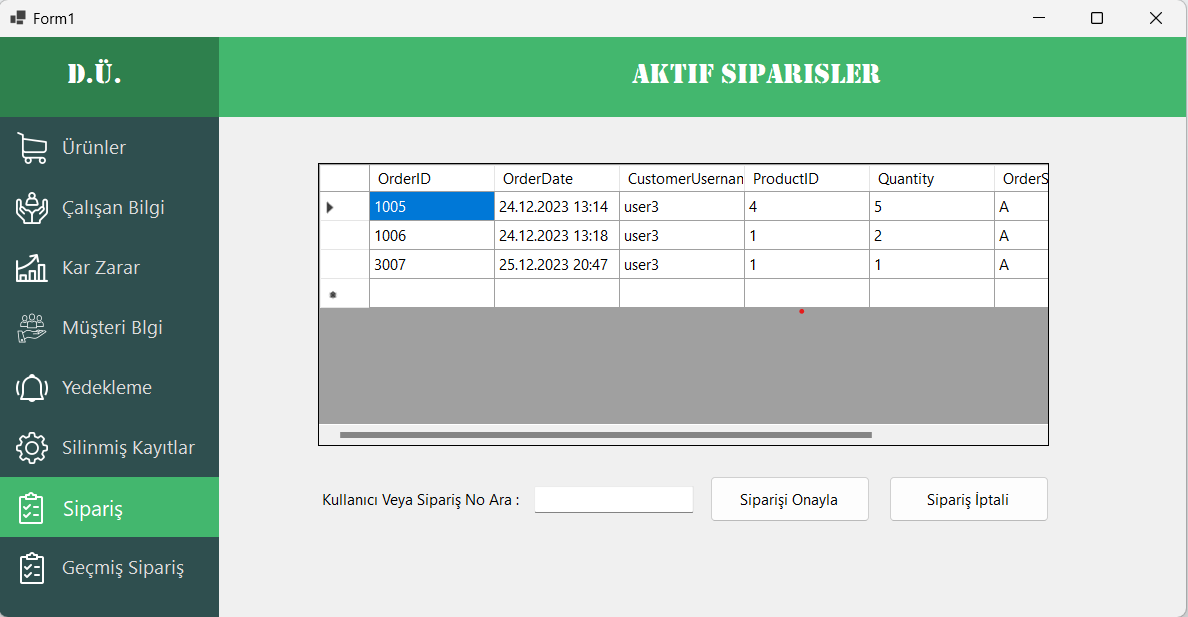
***Şekil 13.1***

Şekil 13.1’de görünen C# programı, geçmiş siparişleri filtrelemek ve belirli bir siparişi silmek için kullanılan bir Windows Forms uygulamasını temsil eder. İşte önemli bölümler:

**Gecmis\_Siparis Formu:**

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Data.SqlClient;
6. using System.Drawing;
7. using System.Linq;
8. using System.Text;
9. using System.Threading.Tasks;
10. using System.Windows.Forms;
11. namespace WinFormsApp1
12. {
13. public partial class Admin\_Gecmis\_Siparis : Form
14. {
15. string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial Catalog=OnlineAlısveris;Integrated Security=True";
16. private DataTable originalDataTable;
17. public Admin\_Gecmis\_Siparis()
18. {
19. InitializeComponent();
20. }
21. private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)
22. {
23. using (SqlConnection connection = new SqlConnection(connectionString))
24. {
25. // SQL sorgusu
26. string query = "SELECT \* FROM OrderHistory";
27. // SQL sorgusunu çalıştırmak için bir SqlDataAdapter ve bir DataTable kullanılır.
28. SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
29. originalDataTable = new DataTable(); // Orijinal DataTable'ı sınıf seviyesinde tanımlanan değişkenle eşleştiriyoruz.
30. // Verileri çek ve DataGridView'i
31. // doldur
32. adapter.Fill(originalDataTable);
33. dataGridView1.DataSource = originalDataTable;
34. }
35. }
36. private void textBox1\_TextChanged(object sender, EventArgs e)
37. {
38. if (originalDataTable != null)
39. {
40. string aramaMetni = textBox1.Text.Trim();
41. // Orijinal DataTable'ın bir kopyasını al
42. DataTable filteredDataTable = originalDataTable.Clone();
43. // DataView oluşturup orijinal DataTable'ı filtreliyoruz.
44. DataView dv = originalDataTable.DefaultView;
45. // Eğer aramaMetni sayısal bir değerse
46. if (int.TryParse(aramaMetni, out \_))
47. {
48. dv.RowFilter = $"OrderHistoryID = {aramaMetni}";
49. }
50. else
51. {
52. // Eğer aramaMetni metin bir değerse
53. dv.RowFilter = $"CustomerUsername LIKE '%{aramaMetni}%'";
54. }
55. // Filtrelenmiş verileri kopyalanmış DataTable'a ekliyoruz.
56. foreach (DataRowView drv in dv)
57. {
58. filteredDataTable.ImportRow(drv.Row);
59. }
60. // Kopyalanmış DataTable'ı DataGridView'e yüklüyoruz.
61. dataGridView1.DataSource = filteredDataTable;
62. }
63. }
64. private void Admin\_Personel\_Gecmis\_Siparis\_Load(object sender, EventArgs e)
65. {
66. dataGridView1\_CellContentClick(sender, new DataGridViewCellEventArgs(0, 0));
67. }
68. private void button1\_Click(object sender, EventArgs e)
69. {
70. // DataGridView'de seçili satırın index'ini al
71. int selectedRowIndex = dataGridView1.CurrentCell.RowIndex;
72. // Seçili satırın OrderID değerini al
73. int orderID = Convert.ToInt32(dataGridView1.Rows[selectedRowIndex].Cells["OrderHistoryID"].Value);
74. // OrderStatus'u "G" olarak güncelle
75. DeleteOrderStatus(orderID);
76. // Güncellenmiş verileri DataGridView'e yükle
77. dataGridView1\_CellContentClick(sender, new DataGridViewCellEventArgs(0, 0));
78. }
79. private void DeleteOrderStatus(int orderID)
80. {
81. using (SqlConnection connection = new SqlConnection(connectionString))
82. {
83. connection.Open();
84. // SQL sorgusu
85. string query = $"DELETE From OrderHistory WHERE OrderHistoryID = {orderID}";
86. using (SqlCommand command = new SqlCommand(query, connection))
87. {
88. // Sorguyu çalıştır
89. command.ExecuteNonQuery();
90. }
91. }
92. }
93. }
94. }

Bu kod, geçmiş siparişleri görüntülemek, siparişleri filtrelemek ve belirli bir siparişi iptal etmek için kullanılan bir formu içerir. **dataGridView1** adlı DataGridView kontrolü, veritabanındaki sipariş verilerini görüntülemek için kullanılır. **textBox1** ise siparişleri filtrelemek için kullanılır. **button1**, seçilen bir siparişi iptal etmek için kullanılır ve bu işlem veritabanındaki ilgili kaydı siler.



***Şekil 13.2***

Şekil 13.2’de görünen C# programı, personel tarafından siparişlerin görüntülenmesi, sipariş durumlarının güncellenmesi ve siparişlerin silinmesi için kullanılan bir Windows Forms uygulamasını temsil eder. İşte önemli bölümler:

**Aktif\_Sipariş Formu:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WinFormsApp1

{

public partial class PersonelSipariş : Form

{

string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial Catalog=OnlineAlısveris;Integrated Security=True";

private DataTable originalDataTable;

public PersonelSipariş()

{

InitializeComponent();

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

// SQL sorgusu

string query = "SELECT \* FROM Orders";

// SQL sorgusunu çalıştırmak için bir SqlDataAdapter ve bir DataTable kullanılır.

SqlDataAdapter adapter = new SqlDataAdapter(query, connection);

originalDataTable = new DataTable(); // Orijinal DataTable'ı sınıf seviyesinde tanımlanan değişkenle eşleştiriyoruz.

// Verileri çek ve DataGridView'i doldur

adapter.Fill(originalDataTable);

dataGridView1.DataSource = originalDataTable;

}

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

if (originalDataTable != null)

{

string aramaMetni = textBox1.Text.Trim();

// Orijinal DataTable'ın bir kopyasını al

DataTable filteredDataTable = originalDataTable.Clone();

// DataView oluşturup orijinal DataTable'ı filtreliyoruz.

DataView dv = originalDataTable.DefaultView;

// Eğer aramaMetni sayısal bir değerse

if (int.TryParse(aramaMetni, out \_))

{

dv.RowFilter = $"OrderID = {aramaMetni}";

}

else

{

// Eğer aramaMetni metin bir değerse

dv.RowFilter = $"CustomerUsername LIKE '%{aramaMetni}%'";

}

// Filtrelenmiş verileri kopyalanmış DataTable'a ekliyoruz.

foreach (DataRowView drv in dv)

{

filteredDataTable.ImportRow(drv.Row);

}

// Kopyalanmış DataTable'ı DataGridView'e yüklüyoruz.

dataGridView1.DataSource = filteredDataTable;

}

}

private void PersonelSipariş\_Load(object sender, EventArgs e)

{

dataGridView1\_CellContentClick(sender, new DataGridViewCellEventArgs(0, 0));

}

private void button1\_Click(object sender, EventArgs e)

{

// DataGridView'de seçili satırın index'ini al

int selectedRowIndex = dataGridView1.CurrentCell.RowIndex;

// Seçili satırın OrderID değerini al

int orderID = Convert.ToInt32(dataGridView1.Rows[selectedRowIndex].Cells["OrderID"].Value);

// OrderStatus'u "G" olarak güncelle

UpdateOrderStatus(orderID, "G");

// Güncellenmiş verileri DataGridView'e yükle

dataGridView1\_CellContentClick(sender, new DataGridViewCellEventArgs(0, 0));

}

private void UpdateOrderStatus(int orderID, string newStatus)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// SQL sorgusu

string query = $"UPDATE Orders SET OrderStatus = '{newStatus}' WHERE OrderID = {orderID}";

using (SqlCommand command = new SqlCommand(query, connection))

{

// Sorguyu çalıştır

command.ExecuteNonQuery();

}

}

}

private void button2\_Click(object sender, EventArgs e)

{

// DataGridView'de seçili satırın index'ini al

int selectedRowIndex = dataGridView1.CurrentCell.RowIndex;

// Seçili satırın OrderID değerini al

int orderID = Convert.ToInt32(dataGridView1.Rows[selectedRowIndex].Cells["OrderID"].Value);

// OrderStatus'u "G" olarak güncelle

DeleteOrderStatus(orderID);

// Güncellenmiş verileri DataGridView'e yükle

dataGridView1\_CellContentClick(sender, new DataGridViewCellEventArgs(0, 0));

}

private void DeleteOrderStatus(int orderID)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// SQL sorgusu

string query = $"DELETE From Orders WHERE OrderID = {orderID}";

using (SqlCommand command = new SqlCommand(query, connection))

{

// Sorguyu çalıştır

command.ExecuteNonQuery();

}

}

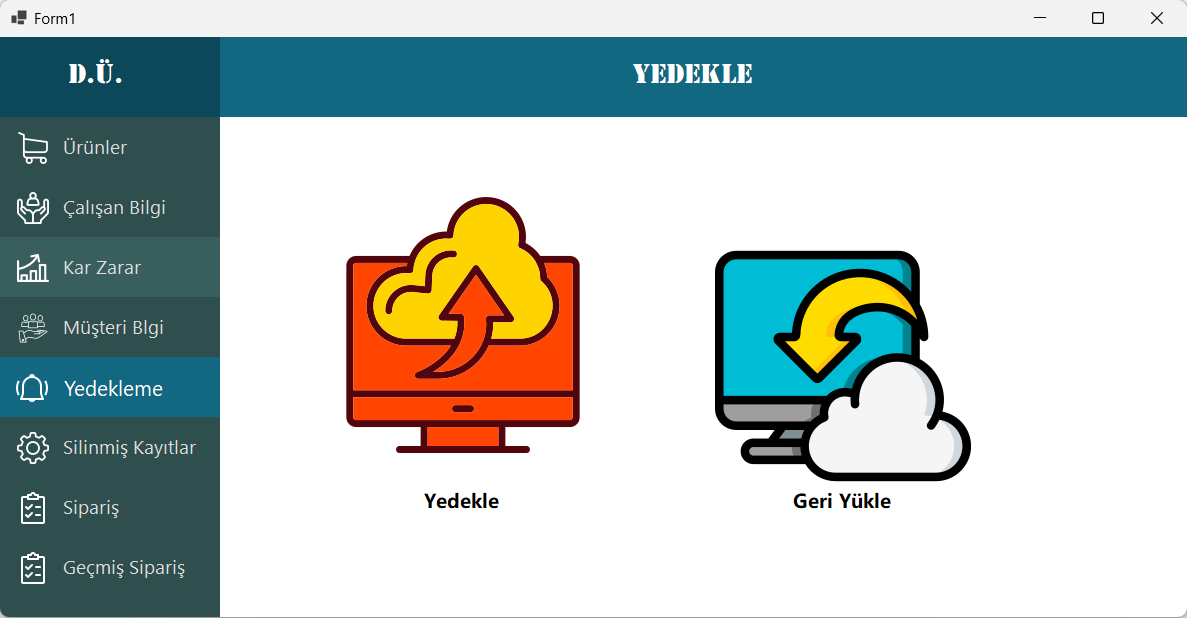
}

}

}

Bu kod, personelin siparişleri görüntülemesini, sipariş durumlarını güncellemesini ve siparişleri silmesini sağlayan bir formu içerir. **dataGridView1** adlı DataGridView kontrolü, veritabanındaki sipariş verilerini görüntülemek için kullanılır. **textBox1**, siparişleri filtrelemek için kullanılır. **button1**, seçilen bir siparişi "Gönderildi" durumuna getirmek için kullanılır. **button2**, seçilen bir siparişi silmek için kullanılır.

**14 YEDEKLEME FORMU**



***Şekil 14.1***

Şekil 14.1’de görünen C# programı, bir Windows Forms uygulaması aracılığıyla SQL Server veritabanının yedeklenmesini ve geri yüklenmesini sağlar. İşte önemli bölümler:

using System.Data.SqlClient;

namespace WinFormsApp1

{

public partial class Yedekle : Form

{

public Yedekle()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

// Kullanıcıya kaydedilecek dosyanın konumunu seçtirme

SaveFileDialog saveFileDialog = new SaveFileDialog

{

Title = "Yedekleme Dosyasını Kaydet",

Filter = "Yedek Dosyaları (\*.bak)|\*.bak",

FileName = "VeritabaniBackup.bak",

InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Desktop),

};

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

// Seçilen dosya yolu

string yedekYolu = saveFileDialog.FileName;

// Veritabanını kullanımdan kaldır

KullanimdanKaldir();

// Veritabanı bağlantısı için connection string

string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial Catalog=OnlineAlısveris;Integrated Security=True";

// Veritabanı bağlantısı oluştur

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// Backup sorgusu

string backupQuery = $"BACKUP DATABASE [OnlineAlısveris] TO DISK='{yedekYolu}' WITH FORMAT";

// Backup işlemi için SqlCommand oluştur

using (SqlCommand command = new SqlCommand(backupQuery, connection))

{

// Backup işlemini gerçekleştir

command.ExecuteNonQuery();

}

}

MessageBox.Show("Veritabanı başarıyla yedeklendi.", "Başarılı", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show("Hata oluştu: " + ex.Message, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void button2\_Click(object sender, EventArgs e)

{

try

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

Title = "Yedek Dosyasını Seç",

Filter = "Yedek Dosyaları (\*.bak)|\*.bak",

InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Desktop),

RestoreDirectory = true

};

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

// Seçilen dosya yolu

string yedekDosyaYolu = openFileDialog.FileName;

// Veritabanını kullanımdan kaldır

KullanimdanKaldir();

// Veritabanı bağlantısı için connection string

string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial Catalog=master;Integrated Security=True";

// Veritabanı bağlantısı oluştur

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// Restore sorgusu

string restoreQuery = $"USE master RESTORE DATABASE [OnlineAlısveris] FROM DISK='{yedekDosyaYolu}' WITH REPLACE";

// Restore işlemi için SqlCommand oluştur

using (SqlCommand command = new SqlCommand(restoreQuery, connection))

{

// Restore işlemini gerçekleştir

command.ExecuteNonQuery();

}

}

MessageBox.Show("Veritabanı başarıyla restore edildi.", "Başarılı", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show("Hata oluştu: " + ex.Message, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void KullanimdanKaldir()

{

// Veritabanını kullanımdan kaldır

string kullanımdanKaldirQuery = "USE master ALTER DATABASE [OnlineAlısveris] SET SINGLE\_USER WITH ROLLBACK IMMEDIATE";

string geriAlQuery = "USE master ALTER DATABASE [OnlineAlısveris] SET MULTI\_USER";

string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial Catalog=master;Integrated Security=True";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand command1 = new SqlCommand(kullanımdanKaldirQuery, connection))

{

command1.ExecuteNonQuery();

}

using (SqlCommand command2 = new SqlCommand(geriAlQuery, connection))

{

command2.ExecuteNonQuery();

}

}

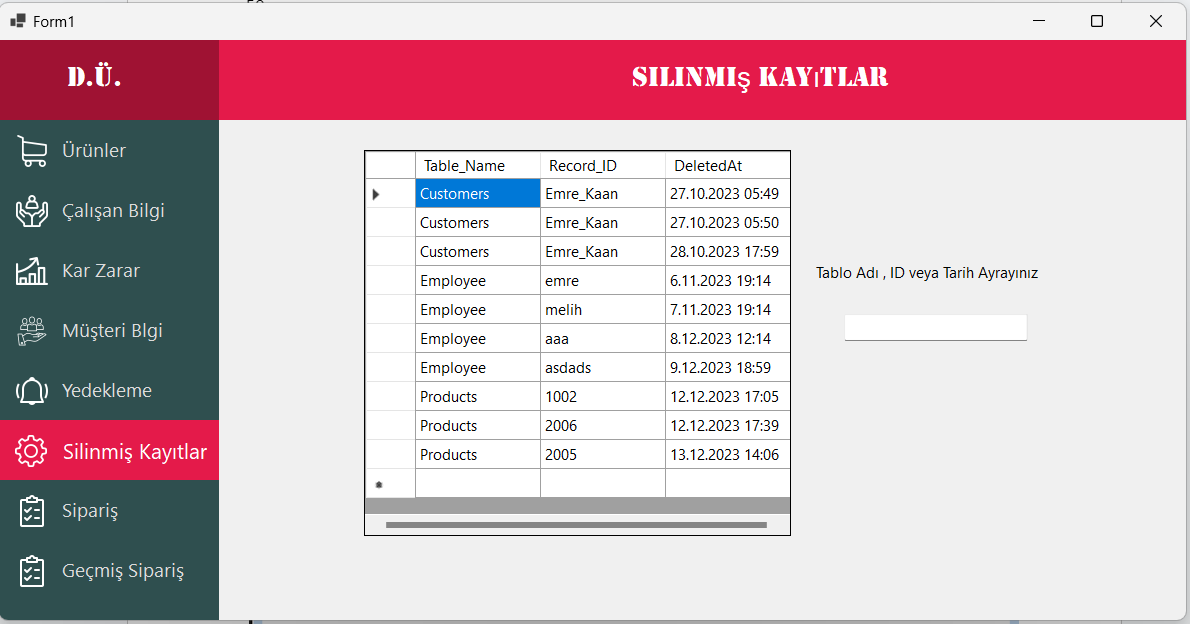
}

}

}

Bu form, "Yedekle" butonuyla veritabanını yedekleme ve "Geri Yükle" butonuyla veritabanını geri yükleme yeteneklerini sağlar. **KullanimdanKaldir** metodu, veritabanını kullanımdan kaldırmak için kullanılır. Bu işlem, veritabanının kilitlenmesini ve ardından yedekleme veya geri yükleme işlemlerini mümkün kılar.

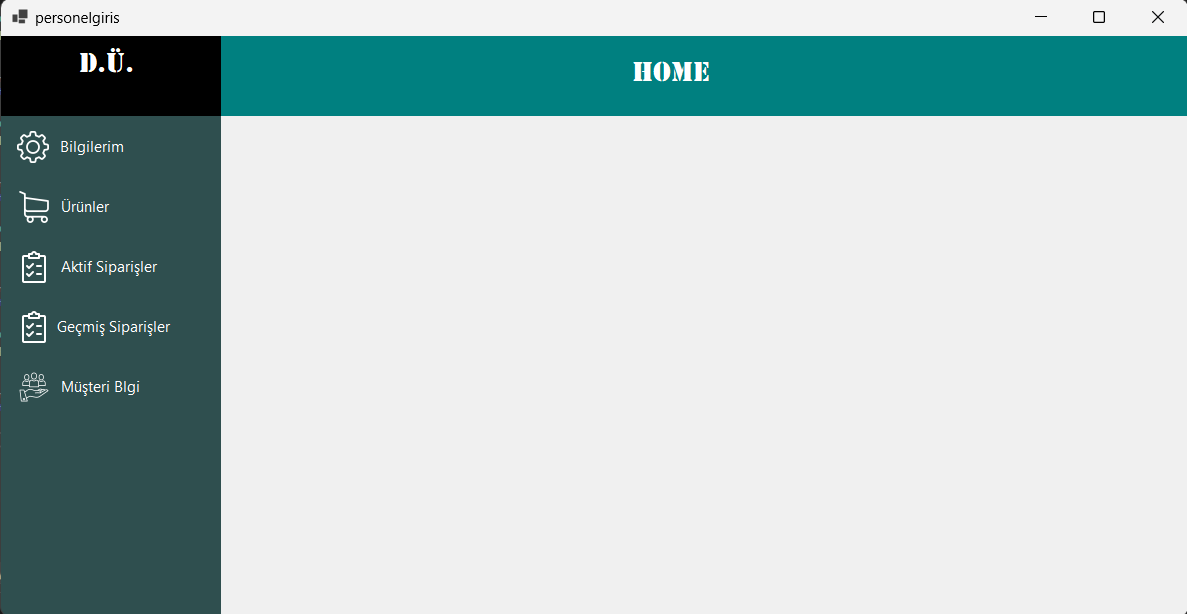
**15. SİLİNMİŞ KAYITLAR FORMU**



***Şekil 15.1***

Şekil 15.1’de görünen C# programı, bir Windows Forms uygulaması aracılığıyla SQL Server veritabanında silinen kayıtların tutuldugu tabloyu gösterir ve tablo içinde arama imkanı verir.

**16. PERSONEL GİRİŞ FORMU**



***Şekil 16.1***

Bu C# Windows Forms uygulaması, bir personelin kullanıcı arayüzünü sağlayan "personelgiris" adlı bir form içerir. Aşağıda, formdaki önemli noktaların açıklamalarını bulabilirsiniz:

1. **Constructor ve Kullanıcı Adı Alanı:**

public string Kname; public personelgiris() { InitializeComponent(); }

Bu kısımda, **personelgiris** formunun constructor'ı ve **Kname** adlı bir kamu (public) string alanı bulunmaktadır. Bu alan, personelin kullanıcı adını depolamak için kullanılır.

1. **Sipariş Formunu Açma:**

private void button1\_Click(object sender, EventArgs e) { Form personel = new PersonelSipariş(); personel.ShowDialog(); }

Bu kod parçası, "Sipariş" butonuna tıklandığında, **PersonelSipariş** formunu açar.

1. **Ürün Formunu Açma:**

private void button2\_Click(object sender, EventArgs e) { Form Purun = new Urun(); Purun.ShowDialog(); }

Bu kod parçası, "Ürün" butonuna tıklandığında, **Urun** formunu açar.

1. **Müşteri Formunu Açma:**

private void button4\_Click(object sender, EventArgs e) { Form Pmüsteri = new PersonelMusteri(); Pmüsteri.ShowDialog(); }

Bu kod parçası, "Müşteri" butonuna tıklandığında, **PersonelMusteri** formunu açar.

1. **Personel Bilgileri Formunu Açma:**

private void button3\_Click(object sender, EventArgs e) { personelBilgilerim Pbilgi = new personelBilgilerim(); Pbilgi.KName = Kname; Pbilgi.ShowDialog(); }

Bu kod parçası, "Personel Bilgilerim" butonuna tıklandığında, **personelBilgilerim** formunu açar ve bu formun **KName** özelliğine mevcut personelin kullanıcı adını atar.

1. **Geçmiş Sipariş Formunu Açma:**

private void button5\_Click(object sender, EventArgs e) { Form pSiparisgecmis1 = new Personel\_Gecmis\_Siparis(); pSiparisgecmis1.ShowDialog(); }

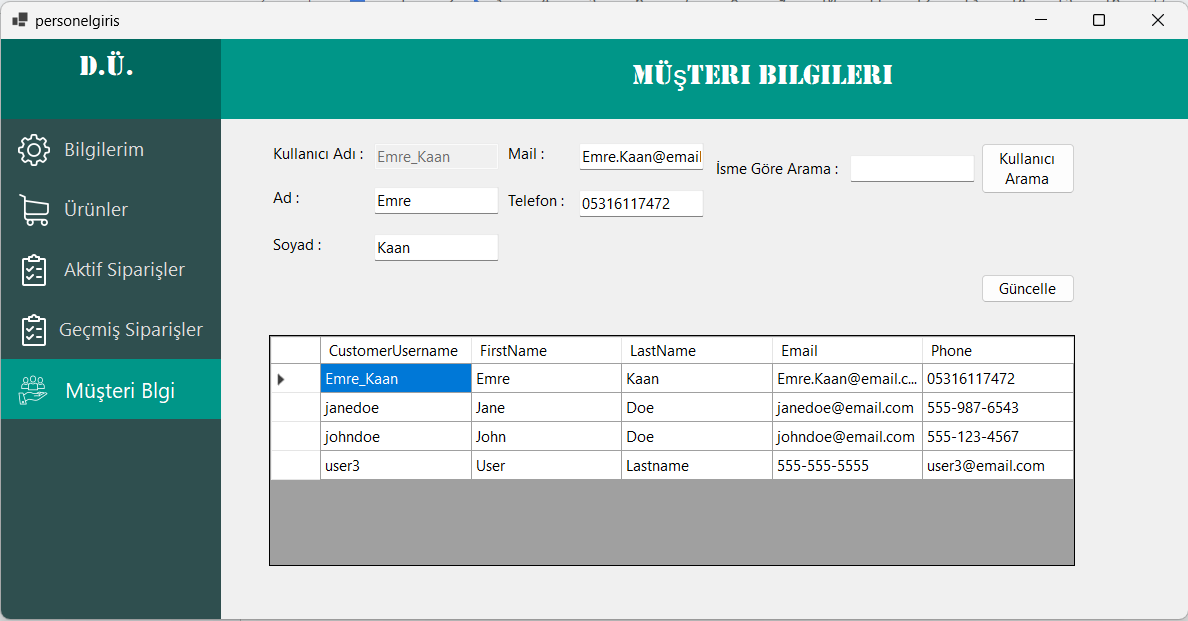
Bu kod parçası, "Geçmiş Sipariş" butonuna tıklandığında, **Personel\_Gecmis\_Siparis** formunu açar.

1. **Form Kapatma Olayı:**

private void personelgiris\_FormClosing(object sender, FormClosingEventArgs e) { if (e.CloseReason == CloseReason.UserClosing) { DialogResult result = MessageBox.Show("Uygulamayı kapatmak istediğinizden emin misiniz?", "Uyarı", MessageBoxButtons.YesNo, MessageBoxIcon.Question); if (result == DialogResult.No) { e.Cancel = true; // Kapatmayı iptal et } else { Application.Exit(); } } }

Bu kod parçası, form kapatılmak istendiğinde bir onay penceresi gösterir ve kullanıcının "Hayır" seçeneğini seçmesi durumunda kapatma işlemini iptal eder, "Evet" seçeneğini seçmesi durumunda uygulamayı kapatır.

**17. PERSONEL MÜŞTERİ BİLGİ FORMU**

****

***Şekil 17.1***

Bu C# Windows Forms uygulaması, personelin müşteri bilgilerini görüntüleyebileceği, düzenleyebileceği ve arama yapabileceği "PersonelMusteri" adlı bir form içerir. İşlevselliğin temelini, müşteri bilgilerini görüntüleme, düzenleme ve arama yapma oluşturur. İşte bu formdaki önemli kısımların açıklamaları:

1. **Bağlantı ve Komutlar:**

SqlConnection connection; SqlCommand komut; SqlDataAdapter da; public PersonelMusteri() { InitializeComponent(); }

Bu kısımda, SqlConnection, SqlCommand ve SqlDataAdapter nesneleri tanımlanmıştır. Bu nesneler, veritabanına bağlanmak ve sorguları çalıştırmak için kullanılır.

1. **MusteriGetir Metodu:**

void MusteriGetir() { // ... (Müşteri verilerini çekme ve DataGridView'e yükleme) }

Bu metod, müşteri verilerini veritabanından çekip DataGridView kontrolüne yükler.

1. **Arama Butonu:**

private void btnAra\_Click(object sender, EventArgs e) { // ... (Arama işlemleri) }

Bu metod, "Ara" butonuna tıklandığında müşteri adına göre arama yapar ve sonuçları DataGridView'e yükler.

1. **Düzenleme Butonu:**

private void btnDuzenle\_Click(object sender, EventArgs e) { // ... (Müşteri bilgilerini güncelleme işlemleri) }

Bu metod, "Düzenle" butonuna tıklandığında seçilen müşterinin bilgilerini günceller.

1. **Form Yüklendiğinde Müşteri Bilgilerini Getirme:**

private void PersonelMusteri\_Load(object sender, EventArgs e) { MusteriGetir(); }

Bu metod, form yüklendiğinde müşteri bilgilerini getirir ve DataGridView'e yükler.

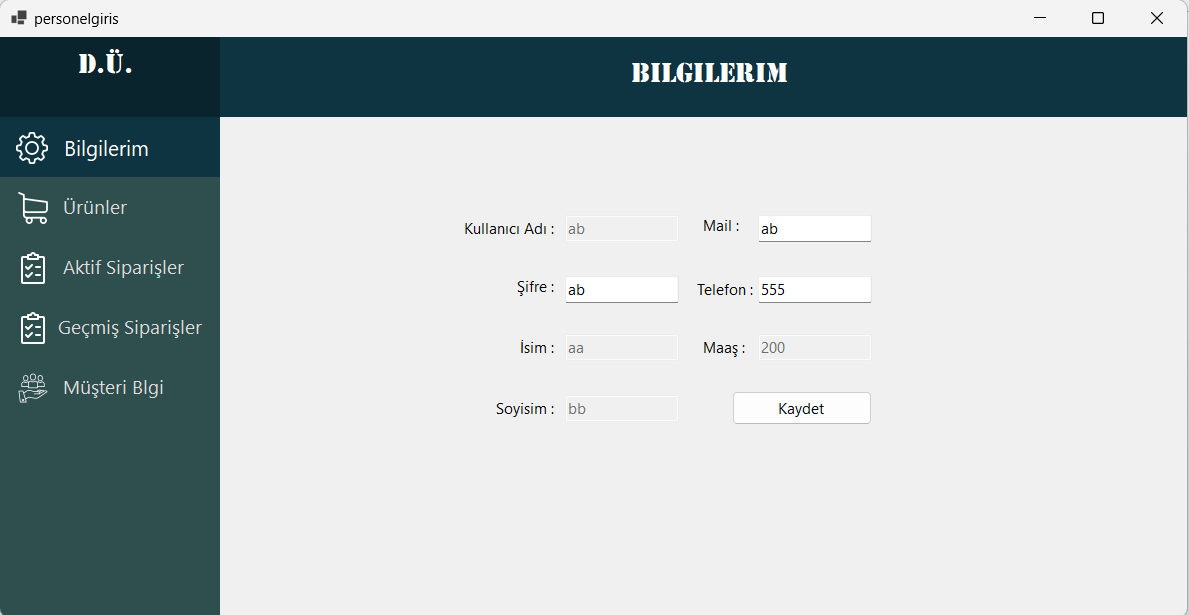
1. **DataGridView'da Satır Seçildiğinde Bilgileri Gösterme:**

private void dataGridView1\_CellEnter(object sender, DataGridViewCellEventArgs e) { // ... (DataGridView'da seçilen satırın bilgilerini TextBox'lara yükleme) }

Bu metod, DataGridView'da bir satıra tıklandığında seçilen müşterinin bilgilerini ilgili TextBox'lara yükler.

Bu form, personelin müşteri bilgilerini düzenleyebilmesi ve arama yapabilmesi için gerekli temel işlevselliği içermektedir.

**18. PERSONEL BİLGİLERİM FORMU**



***Şekil 18.1***

Bu C# Windows Forms uygulaması, personelin kendi bilgilerini görüntüleyebileceği ve güncelleyebileceği "personelBilgilerim" adlı bir form içerir. İşlevselliği, personelin kendi bilgilerini veritabanından çekme, bu bilgileri TextBox'lara yerleştirme ve ardından güncelleme işlemlerini içerir. İşte bu formdaki önemli kısımların açıklamaları:

1. **Employee Sınıfı:**

public class Employee { public string EmployeeUsername { get; set; } public string EmployeePassword { get; set; } public string FirstName { get; set; } public string LastName { get; set; } public string Email { get; set; } public string Phone { get; set; } public decimal Salary { get; set; } }

**Employee** sınıfı, bir personel çalışanının temel bilgilerini içerir. Bu sınıf, veritabanından çekilen personel bilgilerini temsil etmek için kullanılır.

1. **Form Yüklendiğinde Verilerin Alınması ve TextBox'lara Yerleştirilmesi:**

private void personelBilgilerim\_Load(object sender, EventArgs e) { // Form yüklendiğinde veritabanından verileri al employeeList = GetEmployeeDataFromDatabase(); // Verileri TextBox'lara yerleştir if (employeeList.Count > 0) { FillTextBoxes(employeeList[0]); } }

Bu metod, form yüklendiğinde veritabanından personel bilgilerini alır ve bu bilgileri TextBox'lara yerleştirir.

1. **Veritabanından Veri Çekme:**

private List<Employee> GetEmployeeDataFromDatabase() { // ... (Veritabanından personel bilgilerini çekme işlemleri) }

Bu metod, veritabanından personel bilgilerini çeker ve bu bilgileri **Employee** nesneleri olarak içeren bir liste döndürür.

1. **TextBox'lara Veri Yerleştirme:**

private void FillTextBoxes(Employee employee) { // ... (Verileri TextBox'lara yerleştirme işlemleri) }

Bu metod, bir **Employee** nesnesini alır ve bu nesnedeki bilgileri ilgili TextBox'lara yerleştirir.

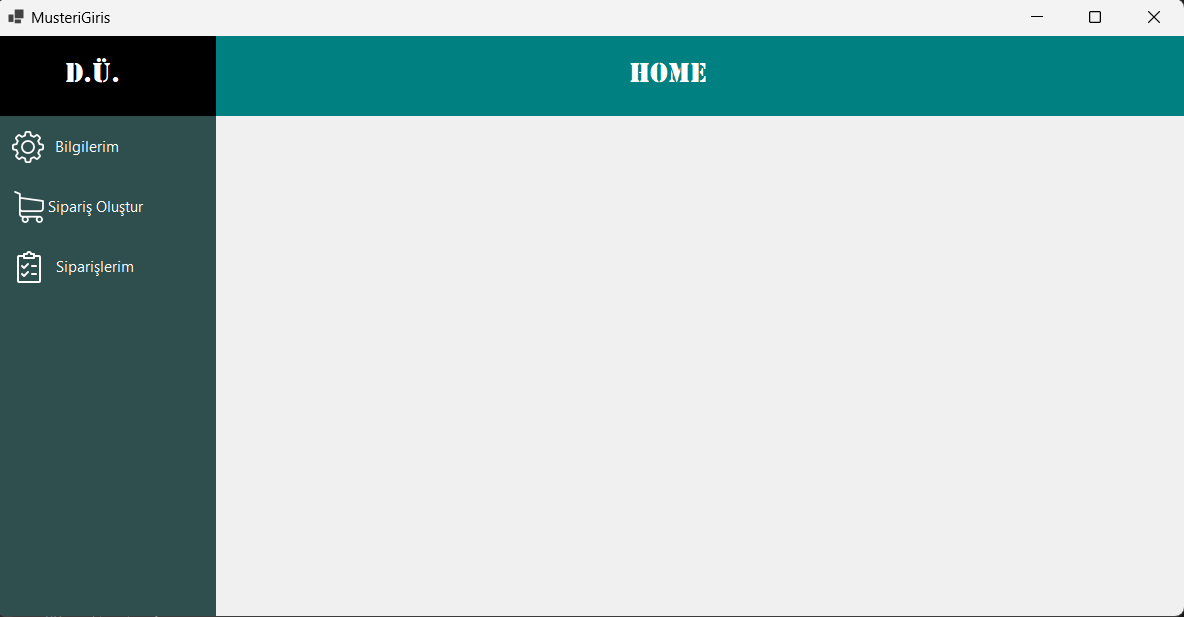
1. **Veri Güncelleme İşlemi:**

private void button1\_Click(object sender, EventArgs e) { // ... (Veri güncelleme işlemleri) }

Bu metod, "Güncelle" butonuna tıklandığında TextBox'lardaki verileri alır ve bu verileri kullanarak veritabanındaki personel bilgilerini günceller.

Bu form, personelin kendi bilgilerini görüntüleyebilmesi ve güncelleyebilmesi için gerekli temel işlevselliği içermektedir.

**19. MÜŞTERİ GİRİŞ FORMU**



***Şekil 19.1***

Bu C# Windows Forms uygulaması, müşteri girişi için kullanılan "MusteriGiris" adlı bir form içerir. Form, müşterinin bilgilerini görüntüleyebilmesi, yeni sipariş verebilmesi, geçmiş siparişlerini inceleyebilmesi ve sepetini kontrol edebilmesi için bazı temel işlevselliği içerir. Aşağıda bu formun önemli kısımlarının açıklamalarını bulabilirsiniz:

1. **DataTable Tanımları ve Başlangıç Ayarları:**

public partial class MusteriGiris : Form { MusteriSepet Sepet; public DataTable dt = new DataTable(); public DataTable dt2 = new DataTable(); public string KullaniciAd; public MusteriGiris() { InitializeComponent(); // DataTable'ların sütunlarını tanımlama dt2.Columns.Add("CategoryName", typeof(string)); dt2.Columns.Add("ProductName", typeof(string)); dt2.Columns.Add("Price", typeof(decimal)); dt2.Columns.Add("Adet", typeof(int)); dt2.Columns.Add("ToplamTutar", typeof(decimal)); dt.Columns.Add("CategoryName", typeof(string)); dt.Columns.Add("ProductName", typeof(string)); dt.Columns.Add("Price", typeof(decimal)); dt.Columns.Add("Adet", typeof(int)); dt.Columns.Add("ToplamTutar", typeof(decimal)); }

Bu kısımda, iki adet **DataTable** tanımlanmıştır: birincisi **dt** (temel veri tablosu), ikincisi ise **dt2** (sepet veri tablosu). Bu tablolar, müşterinin sepetini ve temel bilgilerini tutmak için kullanılır.

1. **Kullanıcı Adını Alma ve Diğer Formlarla İletme:**

public void KAd(string KAd) { KullaniciAd = KAd; }

Bu metod, müşteri giriş formundan alınan kullanıcı adını (**KAd**) sınıf seviyesindeki **KullaniciAd** değişkenine atar.

1. **Diğer Formlara Geçiş ve Bilgi İletme:**

private void button1\_Click(object sender, EventArgs e) { MusteriBilgilerim mstrBlg = new MusteriBilgilerim(); mstrBlg.KName = KullaniciAd; mstrBlg.ShowDialog(); }

Bu metod, "MusteriBilgilerim" formunu çağırarak müşteri bilgilerini görüntüleme işlemini başlatır.

private void button2\_Click(object sender, EventArgs e) { MusteriYeniSiparis yniSprs = new MusteriYeniSiparis(); yniSprs.ShowDialog(); }

Bu metod, "MusteriYeniSiparis" formunu çağırarak yeni sipariş verme işlemini başlatır.

private void button3\_Click(object sender, EventArgs e) { MergeDataTable(dt, dt2); if (Sepet == null || Sepet.IsDisposed) { Sepet = new MusteriSepet(); Sepet.BilgiEkle(dt); Sepet.KAd(KullaniciAd); Sepet.Show(); } if (Sepet != null) { Sepet.BilgiEkle(dt); Sepet.KAd(KullaniciAd); Sepet.Show(); } }

Bu metod, müşterinin sepetini görüntülemek için "MusteriSepet" formunu çağırır ve sepet bilgilerini ileterek formu gösterir.

private void button4\_Click(object sender, EventArgs e) { MusteriGecmisSiparis gcmsSprs = new MusteriGecmisSiparis(); gcmsSprs.ShowDialog(); }

Bu metod, "MusteriGecmisSiparis" formunu çağırarak geçmiş siparişleri görüntüleme işlemini başlatır.

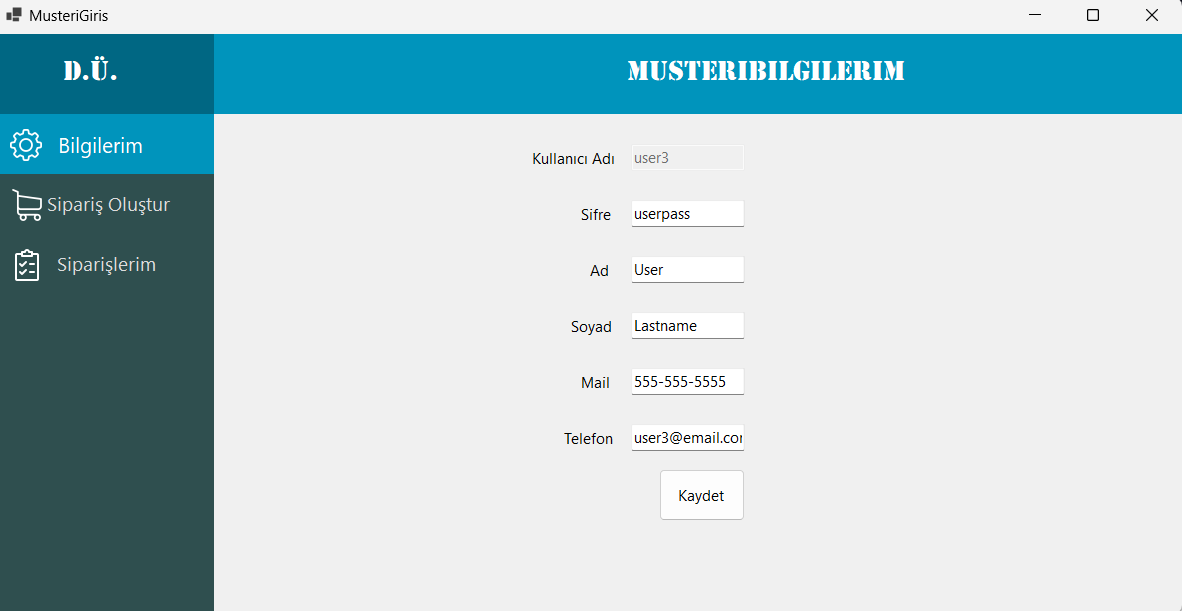
1. **DataTable Birleştirme ve Karşılaştırma Metodları:**

public void MergeDataTableWithoutClear(DataTable destinationTable, DataTable sourceTable) { // DataTable'ları birleştirme (ekleme) destinationTable.Clear(); foreach (DataRow row in sourceTable.Rows) { destinationTable.ImportRow(row); } }

Bu metod, iki **DataTable**'ı birleştirir ve hedef tabloyu temizlemez.

public void MergeDataTable(DataTable destinationTable, DataTable sourceTable) { // DataTable'ları birleştirme (ekleme) foreach (DataRow sourceRow in sourceTable.Rows) { bool rowExists = false; // Hedef DataTable'da her bir satırı kontrol et foreach (DataRow destinationRow in destinationTable.Rows) { // Satırları karşılaştır if (DataRowEquals(sourceRow, destinationRow)) { rowExists = true; break; }

**20. MÜŞTERİ BİLGİLERİM FORMU**

  
***Şekil 20.1***

Bu C# Windows Forms uygulaması, müşteri bilgilerini görüntülemek ve güncellemek için kullanılan "MusteriBilgilerim" adlı bir form içerir. Aşağıda bu formun önemli kısımlarının açıklamalarını bulabilirsiniz:

1. **DataTable Tanımları ve Başlangıç Ayarları:**

public partial class MusteriBilgilerim : Form { private List<Customer> customerList; public string KName; public MusteriBilgilerim() { InitializeComponent(); }

Bu kısımda, **MusteriBilgilerim** sınıfı tanımlanmıştır. Bu sınıf müşteri bilgilerini içerir. **customerList** adında bir liste, müşteri bilgilerini içermek için kullanılmaktadır. **KName** adında bir değişken, müşteri adını tutar.

1. **Form Yüklendiğinde Çalışacak Metod:**

private void MusteriBilgilerim\_Load(object sender, EventArgs e) { customerList = GetEmployeeDataFromDatabase(); if (customerList.Count > 0) { FillTextBoxes(customerList[0]); } }

Bu metod, form yüklendiğinde çalışır. Müşteri bilgilerini veritabanından alır ve eğer müşteri varsa ilgili TextBox'lara yerleştirir.

1. **TextBox'lara Veri Dolduran Metod:**

private void FillTextBoxes(Customer customerList) { // Verileri TextBox'lara yerleştir textBox1.Text = customerList.CustomerUsername; textBox2.Text = customerList.CustomerPassword; textBox3.Text = customerList.FirstName; textBox4.Text = customerList.LastName; textBox5.Text = customerList.Email; textBox6.Text = customerList.Phone; }

Bu metod, parametre olarak aldığı **Customer** nesnesinin bilgilerini ilgili TextBox'lara doldurur.

1. **Veritabanından Müşteri Bilgilerini Alacak Metod:**

private List<Customer> GetEmployeeDataFromDatabase() { List<Customer> customers = new List<Customer>(); try { using (SqlConnection connection = new SqlConnection("Data Source=localhost\\SQLEXPRESS;Initial Catalog=OnlineAlısveris;Integrated Security=True")) { connection.Open(); string query = "EXEC GetCustomerInfo @CustomerUsername =@CustomerUsername"; using (SqlCommand command = new SqlCommand(query, connection)) { // @EmployeeUsername parametresine değeri atayın command.Parameters.AddWithValue("@CustomerUsername", KName); using (SqlDataReader reader = command.ExecuteReader()) { while (reader.Read()) { Customer customer = new Customer { CustomerUsername = reader["CustomerUsername"].ToString(), CustomerPassword = reader["CustomerPassword"].ToString(), FirstName = reader["FirstName"].ToString(), LastName = reader["LastName"].ToString(), Email = reader["Email"].ToString(), Phone = reader["Phone"].ToString() }; customers.Add(customer); } } } } } catch (Exception ex) { MessageBox.Show("Veritabanından veri alınırken bir hata oluştu: " + ex.Message); } return customers; }

Bu metod, veritabanından müşteri bilgilerini almak için kullanılır. **GetCustomerInfo** adlı bir saklı yordam (stored procedure) çağrılır ve sonuçları **Customer** nesneleri olarak listeye eklenir.

1. **Bilgileri Güncelleyen Metod:**

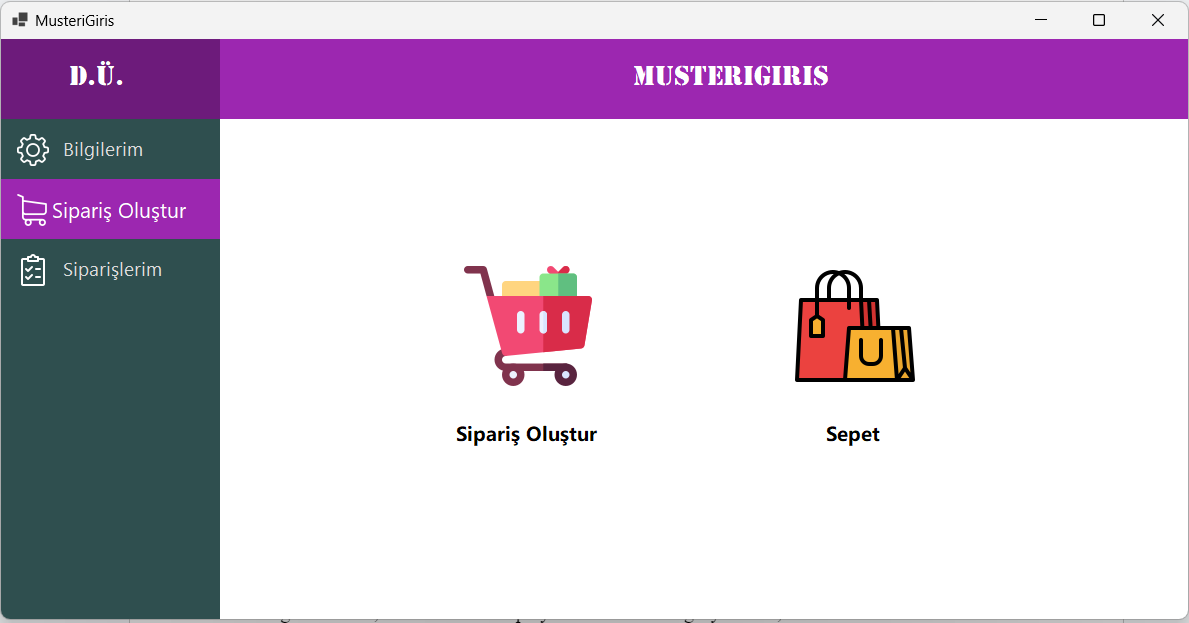
private void button1\_Click(object sender, EventArgs e) { // Güncellenecek verileri al string newPassword = textBox2.Text; string newEmail = textBox5.Text; string newPhone = textBox6.Text; // Güncelleme sorgusunu oluştur string updateQuery = "UPDATE Customers SET CustomerPassword = @NewPassword, Email = @NewEmail, Phone = @NewPhone WHERE CustomerUsername = @CustomerUsername"; try { using (SqlConnection connection = new SqlConnection("Data Source=localhost\\SQLEXPRESS;Initial Catalog=OnlineAlısveris;Integrated Security=True")) { connection.Open(); using (SqlCommand command = new SqlCommand(updateQuery, connection)) { // Parametreleri ekleyin command.Parameters.AddWithValue("@NewPassword", newPassword); command.Parameters.AddWithValue("@NewEmail", newEmail); command.Parameters.AddWithValue("@NewPhone", newPhone); command.Parameters.AddWithValue("@CustomerUsername", KName); // Sorguyu çalıştırın int affectedRows = command.ExecuteNonQuery(); if (affectedRows > 0) { MessageBox.Show("Veriler başarıyla güncellendi."); } else { MessageBox.Show("Güncelleme işlemi başarısız oldu."); } } } } catch (Exception ex) { MessageBox.Show("Veriler güncellenirken bir hata oluştu: " + ex.Message); } }

Bu metod, müşteri bilgilerini güncellemek için kullanılır. Veritabanındaki **Customers** tablosunda ilgili müşteriyi günceller. Güncelleme işlemi başarılı ise bir mesaj penceresi gösterir, aksi takdirde bir hata mesajı gösterir.

1. **Customer Sınıfı:**

public class Customer { public string CustomerUsername { get; set; } public string CustomerPassword { get; set; }

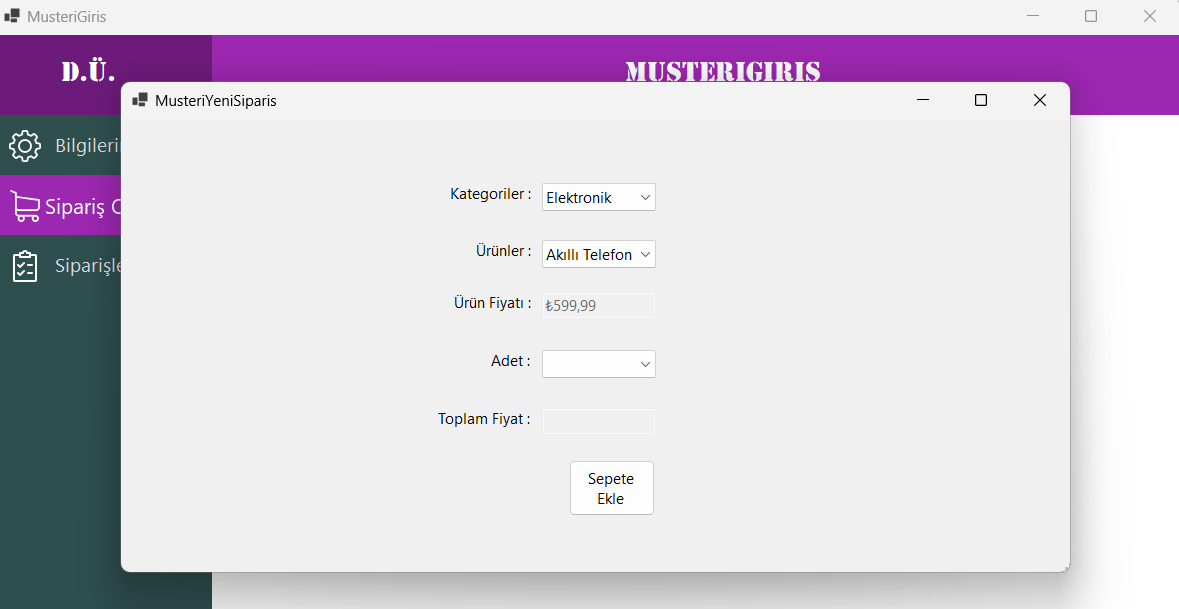
**21. MÜŞTERİ SİPARİŞ FORMU**



***Şekil 21.1***

Şekil 21.1’de görünen bu form Müşterinin sipariş oluşturması ve sepetini görüntülemesi için iki butondandan oluşmaktadır.

**22. SİPARİŞ OLUŞTURMA FORMU**

****

***Şekil 22.1***

Bu C# Windows Forms uygulaması, müşterinin yeni sipariş vermesi için kullanılan "MusteriYeniSiparis" adlı bir form içerir. Aşağıda bu formun önemli kısımlarının açıklamalarını bulabilirsiniz:

1. **ComboBox'ları Dolduran Metod:**

private void FillComboBox() { comboBox1.DropDownStyle = ComboBoxStyle.DropDownList; // Categories tablosundan verileri çek categoriesTable = new DataTable(); using (SqlConnection connection = new SqlConnection(connectionString)) { connection.Open(); string query = "SELECT CategoryID, CategoryName FROM Categories"; SqlDataAdapter adapter = new SqlDataAdapter(query, connection); adapter.Fill(categoriesTable); } // ComboBox'a verileri ata comboBox1.DataSource = categoriesTable; comboBox1.DisplayMember = "CategoryName"; comboBox1.ValueMember = "CategoryID"; }

Bu metod, **Categories** tablosundan kategorileri çeker ve bunları **comboBox1** adlı ComboBox'a ekler.

1. **ComboBox1 Seçim Değişikliğinde Ürünleri Dolduran Metod:**

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e) { if (comboBox1.SelectedValue != null) { // Seçilen öğeyi DataRowView olarak al DataRowView selectedCategoryRow = (DataRowView)comboBox1.SelectedItem; // DataRowView'dan CategoryID değerini al int selectedCategoryId = Convert.ToInt32(selectedCategoryRow["CategoryID"]); string query = "SELECT ProductID, ProductName, Price FROM Products WHERE CategoryID = @SelectedCategoryID"; productsTable = new DataTable(); using (SqlConnection connection = new SqlConnection(connectionString)) using (SqlCommand command = new SqlCommand(query, connection)) { // Parametre ekleyerek SQL sorgusunu güvenli hale getir command.Parameters.AddWithValue("@SelectedCategoryID", selectedCategoryId); connection.Open(); SqlDataAdapter adapter = new SqlDataAdapter(command); adapter.Fill(productsTable); } // ComboBox'a verileri ata comboBox2.DataSource = productsTable; comboBox2.DisplayMember = "ProductName"; comboBox2.ValueMember = "ProductID"; } }

Bu metod, **comboBox1**'de seçilen kategoriye ait ürünleri **comboBox2**'ye ekler.

1. **ComboBox2 Seçim Değişikliğinde Ürün Fiyatını ve TextBox'ları Dolduran Metod:**

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e) { if (comboBox2.SelectedValue != null) { DataRowView selectedProductRow = (DataRowView)comboBox2.SelectedItem; productPrice = Convert.ToDecimal(selectedProductRow["Price"]); textBox1.Text = productPrice.ToString("C"); } }

Bu metod, **comboBox2**'de seçilen ürünün fiyatını alır ve **textBox1**'e gösterir.

1. **ComboBox3 Seçim Değişikliğinde Toplam Tutarı Hesaplayan Metod:**

private void comboBox3\_SelectedIndexChanged(object sender, EventArgs e) { decimal price = (comboBox3.SelectedIndex + 1) \* productPrice; textBox2.Text = price.ToString(); }

Bu metod, **comboBox3**'de seçilen adet bilgisine göre toplam tutarı hesaplar ve **textBox2**'ye gösterir.

1. **Ürünü Sepete Ekleyen Metod:**

private void button1\_Click(object sender, EventArgs e) { if (!isButtonClicked) { DataRowView selectedRow = (DataRowView)comboBox1.SelectedItem; DataRowView selected2Row = (DataRowView)comboBox2.SelectedItem; int CategoryID = Convert.ToInt32(selectedRow["CategoryID"]); string CategoryName = selectedRow["CategoryName"].ToString(); int ProductID = Convert.ToInt32(selected2Row["ProductID"]); string ProductName = selected2Row["ProductName"].ToString(); decimal Price = productPrice; int Adet = (comboBox3.SelectedIndex + 1); decimal ToplamTutar = Convert.ToDecimal(textBox2.Text); DataRow newRow = sepetVerileri.NewRow(); newRow["CategoryName"] = CategoryName; newRow["ProductName"] = ProductName; newRow["Price"] = Price; newRow["Adet"] = Adet; newRow["ToplamTutar"] = ToplamTutar; sepetVerileri.Rows.Add(newRow); MessageBox.Show("Ürün sepete eklendi!", "Başarılı", MessageBoxButtons.OK, MessageBoxIcon.Information); } }

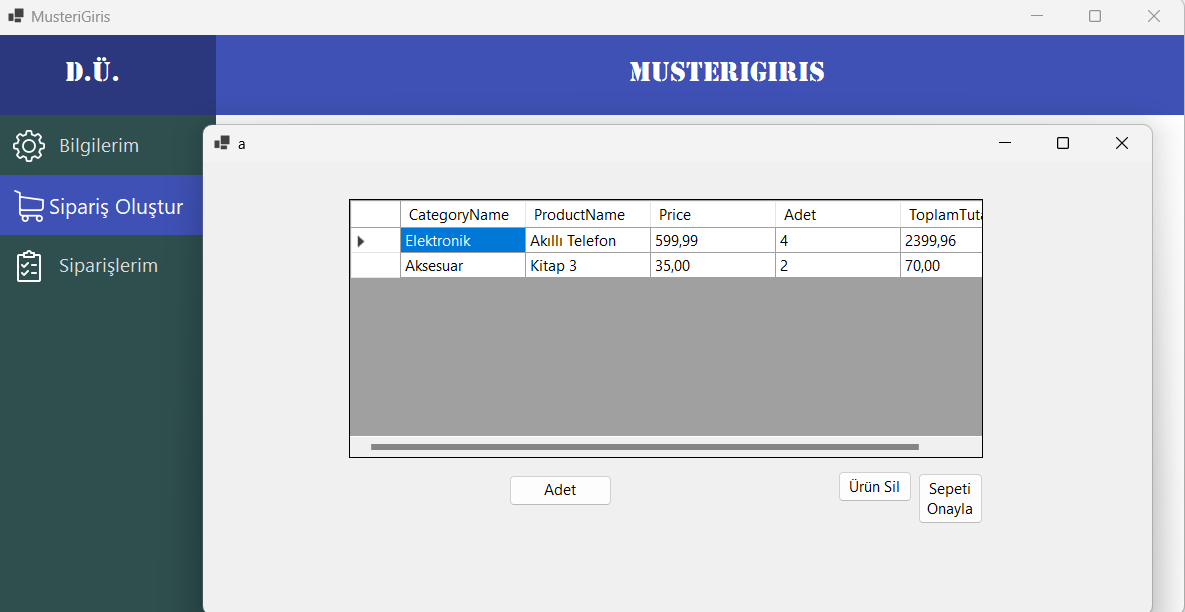
Bu metod, müşterinin seçtiği ürünü ve adeti sepete ekler. Eklenen ürün bilgilerini **sepetVerileri** adlı DataTable'a ekler.

1. **Form Kapatıldığında, Sepet Bilgilerini Ana Forma Aktaran Metod:**

private void MusteriYeniSiparis\_FormClosed(object sender, FormClosedEventArgs e) { MusteriGiris mstrGrs = Application.OpenForms["MusteriGiris"] as MusteriGiris; if (mstrGrs != null) { mstrGrs.MergeDataTableWithoutClear(mstrGrs.dt2, sepetVerileri); } }

Bu metod, müşteri yeni sipariş formu kapatıldığında, müşteri giriş formunda bulunan **dt2** adlı DataTable'a, **sepetVerileri** adlı DataTable'daki bilgileri ekler. **MergeDataTableWithoutClear** metodunu kullanarak birleştirme işlemi gerçekleştirilir.

**23. SEPET FORMU**



***Şekil 23.1***

Bu C# Windows Forms uygulaması içindeki "MusteriSepet" formu, müşterinin sepetindeki ürünleri gösteren bir arayüz sağlar. İşlevselliğin bir kısmı **MusteriGiris** formu ile etkileşim içindedir. İşte bu formun önemli kısımlarının açıklamaları:

1. **DataGridView Oluşturan Metod:**

private void InitializeDataGridView() { // DataGridView sütunlarını oluştur dataGridView1.Columns.Add("CategoryNameColumn", "CategoryName"); dataGridView1.Columns.Add("ProductNameColumn", "ProductName"); dataGridView1.Columns.Add("PriceColumn", "Price"); dataGridView1.Columns.Add("AdetColumn", "Adet"); dataGridView1.Columns.Add("ToplamTutarColumn", "ToplamTutar"); }

Bu metod, **dataGridView1** adlı DataGridView kontrolü için sütunları oluşturur.

1. **DataTable Oluşturan Metod:**

private void InitializeDt() { dt = new DataTable(); dt.Columns.Add ("CategoryNameColumn", typeof(string));dt.Columns.Add("ProductNameColumn", typeof(string)); dt.Columns.Add("PriceColumn", typeof(decimal)); dt.Columns.Add("AdetColumn", typeof(int)); dt.Columns.Add("ToplamTutarColumn", typeof(decimal)); }

Bu metod, **dt** adlı DataTable'ı oluşturur ve gerekli sütunları ekler.

1. **DataGridView'a Bilgi Ekleyen Metod:**

public void BilgiEkle(DataTable dt) { // DataGridView'ı temizle dataGridView1.Rows.Clear(); // DataTable'daki her bir satırı kontrol ederek DataGridView'a ekle foreach (DataRow row in dt.Rows) { // Eğer aynı bilgiler zaten varsa ekleme if (!IsDataExists(row)) { dataGridView1.Rows.Add(row.ItemArray); } } }

Bu metod, DataTable'daki her bir satırı kontrol eder ve eğer aynı bilgiler zaten DataGridView'da varsa ekleme işlemi gerçekleştirmez.

1. **DataGridView'da Varlığı Kontrol Eden Metod:**

private bool IsDataExists(DataRow newRow) { // DataGridView'da her bir satırı kontrol et foreach (DataGridViewRow dataGridViewRow in dataGridView1.Rows) { bool isMatch = true; // Her bir hücreyi kontrol et ve eğer değerler eşleşmiyorsa isMatch'i false yap for (int i = 0; i < newRow.ItemArray.Length; i++) { if (!dataGridViewRow.Cells[i].Value.Equals(newRow.ItemArray[i])) { isMatch = false; break; } } // Eğer tüm hücreler eşleşiyorsa bu satır zaten var demektir if (isMatch) { return true; } } // Eğer eşleşen bir satır bulunamazsa, bu veri yok demektir return false; }

Bu metod, yeni bir satırın, DataGridView'da zaten bulunan bir satıra eşit olup olmadığını kontrol eder.

1. **Ürün Adedi Güncelleme İşlemini Gerçekleştiren Metod:**

private void button3\_Click(object sender, EventArgs e) { if (comboBox1.Visible == false) { comboBox1.Visible = true; } // ComboBox'ta seçili değeri al string yeniAdetValue = comboBox1.SelectedItem?.ToString(); if (!string.IsNullOrEmpty(yeniAdetValue) && yeniAdetValue != adetValue) { // Seçilen satırın indeksini al int selectedRowIndex = dataGridView1.CurrentCell.RowIndex; // Eğer geçerli bir satır tıklandıysa devam et if (selectedRowIndex >= 0 && selectedRowIndex < dataGridView1.Rows.Count) { // Adet değerini güncelle dataGridView1.Rows[selectedRowIndex].Cells["AdetColumn"].Value = yeniAdetValue; mstrGrs.dt.Rows[selectedRowIndex]["Adet"] = yeniAdetValue; // Adet değerini int olarak al if (int.TryParse(yeniAdetValue, out int adet)) { // Fiyat değerini decimal olarak al decimal fiyat = Convert.ToDecimal(dataGridView1.Rows[selectedRowIndex].Cells["PriceColumn"].Value); // ToplamTutar'ı hesapla ve güncelle decimal toplamTutar = adet \* fiyat; dataGridView1.Rows[selectedRowIndex].Cells["ToplamTutarColumn"].Value = toplamTutar.ToString(); mstrGrs.dt.Rows[selectedRowIndex]["ToplamTutar"] = toplamTutar.ToString(); // Güncelleme işlemi tamamlandıktan sonra adetValue'i de güncelle adetValue = yeniAdetValue; } else { // Hata durumunda kullanıcıya bilgi ver MessageBox.Show("Geçerli bir adet değeri giriniz.", "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error); } } } }

Bu metod, seçilen satırın ürün adedini günceller ve bu güncellemeyi DataTable'a ve **MusteriGiris** formundaki DataTable'lara da yansıtır.

1. **Sipariş Oluşturma İşlemini Gerçekleştiren Metod:**

private void button1\_Click(object sender, EventArgs e) { tabloekle(); try { using (SqlConnection connection = new SqlConnection(connectionString)) { connection.Open(); foreach (DataRow row in dt.Rows) { // Her bir satır için ProductName'i al ve IDGetir fonksiyonunu çağır string productName = row["ProductNameColumn"].ToString(); int productID = IDGetir(productName); using (SqlCommand command = new SqlCommand("INSERT INTO Orders (CustomerUserName, ProductID, Quantity, OrderStatus, TotalPrice) VALUES (@CustomerUserName, @ProductID, @Quantity, @OrderStatus, @TotalPrice)", connection)) { command.Parameters.AddWithValue("@CustomerUserName", KullaniciAdi); command.Parameters.AddWithValue("@ProductID", productID); command.Parameters.AddWithValue("@Quantity", row["AdetColumn"]); command.Parameters.AddWithValue("@OrderStatus", "A"); command.Parameters.AddWithValue("@TotalPrice", row["ToplamTutarColumn"]); command.ExecuteNonQuery(); } } MessageBox.Show("Siparişiniz Başarı ile oluşturuldu.", "Başarılı", MessageBoxButtons.OK, MessageBoxIcon.Information); dataGridView1.Rows.Clear(); dt.Rows.Clear(); MusteriGiris musteri = Application.OpenForms["MusteriGiris"] as MusteriGiris; musteri.dt.Clear(); musteri.dt2.Clear(); } } catch (Exception ex) { MessageBox.Show("Veritabanına ekleme sırasında hata oluştu: " + ex.Message, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error); } }

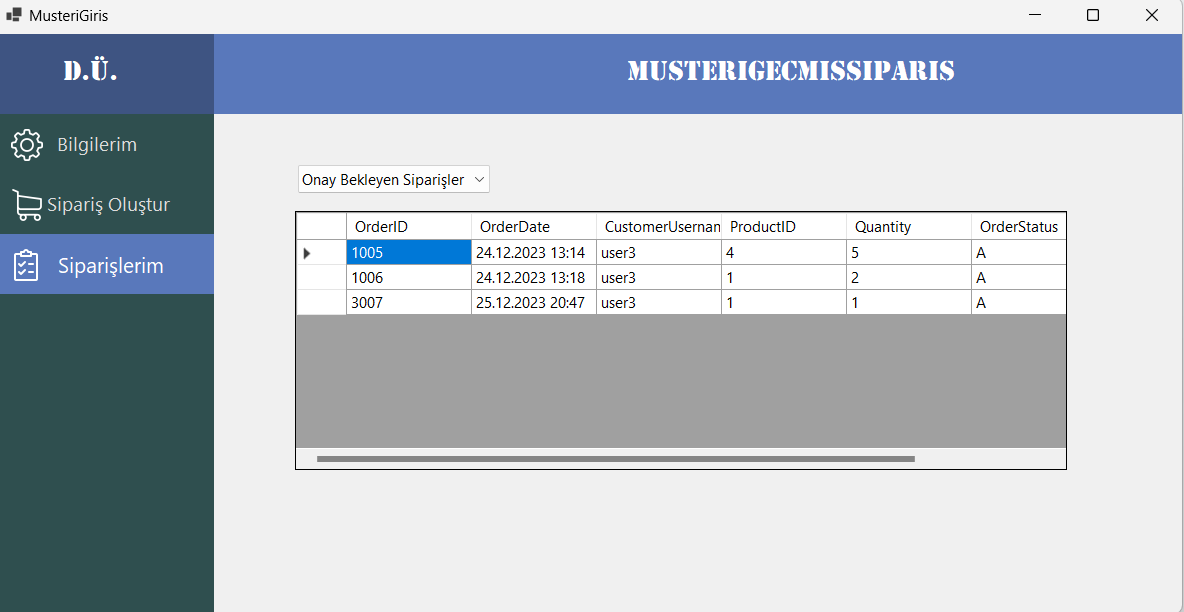
Bu metod, sepet içindeki ürünlerle ilgili siparişi veritabanına ekler. **Orders** tablosuna müşteri bilgileri, ürün ID'si, adet, sipariş durumu ve toplam tutar bilgilerini ekler.

1. **Ürün ID'sini Getiren Metod:**

private int IDGetir(string productName) { try { using (SqlConnection connection = new SqlConnection(connectionString)) { connection.Open(); using (SqlCommand command = new SqlCommand("SELECT ProductID FROM Products WHERE ProductName = @ProductName", connection)) { command.Parameters.AddWithValue("@ProductName", productName); // ExecuteScalar, yalnızca bir tek bir değer döndüren sorgular için kullanılır. // Bu durumda, ProductID değerini döndürecektir. object result = command.ExecuteScalar(); // Eğer result null değilse ve bir int'e dönüştürülebiliyorsa, bu değeri döndür. if (result != null && int.TryParse(result.ToString(), out int productID)) { return productID; } } } } catch (Exception ex) { MessageBox.Show("Veritabanından veri getirme sırasında hata oluştu: " + ex.Message, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error); } // Hata durumlarında veya ProductID bulunamadığında -1 değerini döndürebilirsiniz. return -1; }

Bu metod, bir ürünün ismi verildiğinde bu ürünün ID'sini veritabanından alır. Eğer ürün bulunamazsa veya bir hata oluşursa -1 değerini döndürür.

Bu açıklamalar, **MusteriSepet** formundaki önemli metodları ve kontrol akışını kapsamaktadır. Bu formun işlevselliği, müşterinin sepetine ürün eklemesi, çıkarması ve sipariş oluşturması üzerinedir.

**24. MÜŞTERİNİN SİPARİŞLERİNİ GÖRME FORMU**  
  


**Şekil 24.1**

Bu C# Windows Forms uygulaması içindeki "MusteriGecmisSiparis" formu, müşterinin geçmiş siparişlerini görüntülemesini sağlar. İşlevselliği iki ayrı durumu ele alır: onay bekleyen siparişler ve onaylanmış siparişler. İşte bu formun önemli kısımlarının açıklamaları:

1. **Constructor (Yapıcı Metod):**

public MusteriGecmisSiparis() { InitializeComponent(); comboBox1.SelectedIndex = 0; }

Bu metod, form oluşturulduğunda çalışır. ComboBox'ın varsayılan seçeneğini (onay bekleyen siparişler) belirler.

1. **Onaylanan Siparişleri Getiren Metod:**

public void Onaylanan() { string kullaniciAdi = Form1.kullanici\_adi; using (SqlConnection connection = new SqlConnection(connectionString)) { // SQL sorgusu string query = "EXEC GetOrderHistoryByCustomer @kullaniciAdi = @kullaniciAdi"; using (SqlCommand command = new SqlCommand(query, connection)) { // Parametre ekleyin command.Parameters.AddWithValue("@kullaniciAdi", kullaniciAdi); // SQL sorgusunu çalıştırmak için bir SqlDataAdapter ve bir DataTable kullanılır. SqlDataAdapter adapter = new SqlDataAdapter(command); dataTable = new DataTable(); // Verileri çek ve DataGridView'i doldur adapter.Fill(dataTable); dataGridView1.DataSource = dataTable; } } }

Bu metod, müşterinin daha önce onaylanmış siparişlerini alır ve bunları bir DataTable içine doldurur. DataTable, DataGridView kontrolüne bağlanarak müşteriye gösterilir.

1. **Onay Bekleyen Siparişleri Getiren Metod:**

public void OnayBekleyen() { string kullaniciAdi = Form1.kullanici\_adi; using (SqlConnection connection = new SqlConnection(connectionString)) { // SQL sorgusu string query = "SELECT \* FROM Orders WHERE CustomerUsername = @kullaniciAdi"; using (SqlCommand command = new SqlCommand(query, connection)) { // Parametre ekleyin command.Parameters.AddWithValue("@kullaniciAdi", kullaniciAdi); // SQL sorgusunu çalıştırmak için bir SqlDataAdapter ve bir DataTable kullanılır. SqlDataAdapter adapter = new SqlDataAdapter(command); dataTable2 = new DataTable(); // Verileri çek ve DataGridView'i doldur adapter.Fill(dataTable2); dataGridView1.DataSource = dataTable2; } } }

Bu metod, müşterinin onay bekleyen siparişlerini alır ve bunları bir DataTable içine doldurur. DataTable, DataGridView kontrolüne bağlanarak müşteriye gösterilir.

1. **ComboBox Seçim Değiştirdiğinde Olay:**

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e) { if (comboBox1.SelectedIndex == 0) { OnayBekleyen(); } if (comboBox1.SelectedIndex == 1) { Onaylanan(); } }

Bu olay, ComboBox'taki seçenek değiştiğinde tetiklenir. Eğer seçilen seçenek "Onay Bekleyen Siparişler" ise **OnayBekleyen** metodu çağrılır. Eğer seçilen seçenek "Onaylanan Siparişler" ise **Onaylanan** metodu çağrılır. Bu, müşterinin geçmiş siparişlerini farklı durumlarla filtrelemesine olanak tanır.

SON