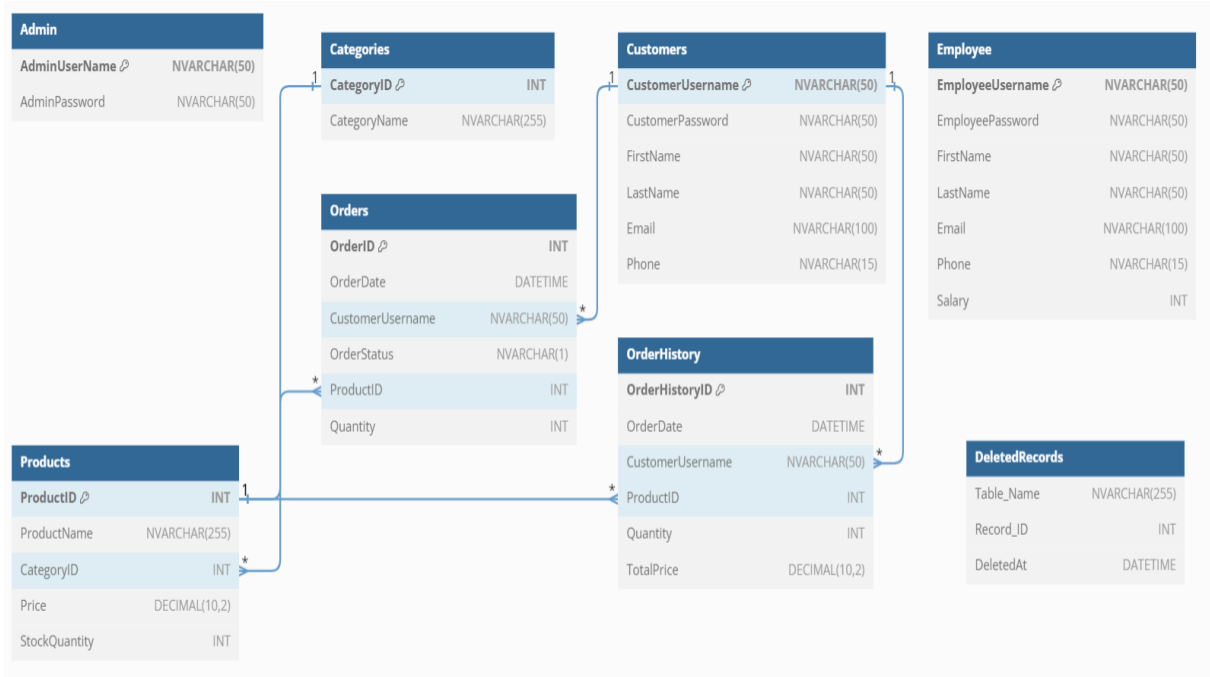


**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ****2023-2024 AKADEMİK YILI**  
**GÜZ DÖNEMİ****Veritabanı Yönetim Sistemleri****Online Alışveriş Sistemi Projesi Raporu****N.Ö. Grup 32****Teslim Eden:**

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

<b>1- ER DİYAGRAMLARININ OLUŞTURULMASI.....</b>	<b>2</b>
<b>2- TABLOLAR.....</b>	<b>3</b>
<b>3- ROL ATAMALARI.....</b>	<b>12</b>
<b>4-TRİGGERLAR.....</b>	<b>15</b>
<b>5- PROSEDÜRLER.....</b>	<b>24</b>
<b>6- UYGULAMA GİRİŞ EKRANI.....</b>	<b>27</b>
<b>7- KAYIT OLMA EKRANI.....</b>	<b>28</b>
<b>8- ADMIN GİRİŞ EKRANI.....</b>	<b>30</b>
<b>9- ÜRÜNLER FORMU.....</b>	<b>31</b>
<b>10- Çalışan Bilgi Formu.....</b>	<b>33</b>
<b>11- MÜŞTERİ BİLGİ FORMU.....</b>	<b>35</b>
<b>12- Kar Zarar Formu.....</b>	<b>36</b>
<b>13- SİPARİŞ FORMLARI.....</b>	<b>37</b>
<b>14- YEDEKLEME FORMU.....</b>	<b>43</b>
<b>15- SİLİNİMİŞ KAYITLAR FORMU.....</b>	<b>46</b>
<b>16- PERSONEL GİRİŞ FORMU.....</b>	<b>47</b>
<b>17- PERSONEL MÜŞTERİ BİLGİ FORMU.....</b>	<b>49</b>
<b>18- PERSONEL BİLGİLERİM FORMU.....</b>	<b>50</b>
<b>19- MÜŞTERİ GİRİŞ FORMU.....</b>	<b>52</b>
<b>20- MÜŞTERİ BİLGİLERİM FORMU.....</b>	<b>54</b>
<b>21- MÜŞTERİ SİPARİŞ FORMU.....</b>	<b>56</b>
<b>22- SİPARİŞ OLUŞTURMA FORMU.....</b>	<b>57</b>
<b>23- SEPET FORMU.....</b>	<b>59</b>
<b>24- MÜŞTERİNİN SİPARİŞLERİNİ GÖRME FORMU.....</b>	<b>62</b>

## **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

```

CREATE TABLE Admin (
    AdminUserName NVARCHAR(50) PRIMARY KEY,
    AdminPassword NVARCHAR(50),
);

```

*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 değer 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.

```

CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY IDENTITY(1,1),
    CategoryName NVARCHAR(255)
);

```

*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.

```
CREATE TABLE Customers (  
    CustomerUsername NVARCHAR(50) PRIMARY KEY,  
    CustomerPassword NVARCHAR(50),  
    FirstName NVARCHAR(50),  
    LastName NVARCHAR(50),  
    Email NVARCHAR(100),  
    Phone NVARCHAR(15)  
);
```

*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.

```
CREATE TABLE Employee (  
    EmployeeUsername NVARCHAR(50) PRIMARY KEY,  
    EmployeePassword NVARCHAR(50),  
    FirstName NVARCHAR(50),  
    LastName NVARCHAR(50),  
    Email NVARCHAR(100),  
    Phone NVARCHAR(15),  
    Salary INT  
);
```

*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.

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY IDENTITY(1,1),  
    ProductName NVARCHAR(255),  
    CategoryID INT,  
    Price DECIMAL(10, 2),  
    StockQuantity INT,  
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)  
);
```

*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.

```
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY IDENTITY(1,1),  
    OrderDate DATETIME,  
    CustomerUsername NVARCHAR(50),  
    ProductID INT,  
    Quantity INT,  
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID),  
    FOREIGN KEY (CustomerUsername) REFERENCES Customers(CustomerUsername)  
);
```

*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.

```
CREATE TABLE OrderHistory (  
    OrderHistoryID INT PRIMARY KEY IDENTITY(1,1),  
    OrderDate DATETIME,  
    CustomerUsername NVARCHAR(50),  
    ProductID INT,  
    Quantity INT,  
    TotalPrice DECIMAL(10, 2),  
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID),  
    FOREIGN KEY (CustomerUsername) REFERENCES Customers(CustomerUsername),  
);
```

*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 geçmiş 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 olarak ilişkilendirmektir.

```
CREATE TABLE DeletedRecords (  
    Table_Name NVARCHAR(255),  
    Record_ID INT,  
    DeletedAt DATETIME  
);
```

*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

```
- Create Role Admin;  
  
Grant Select, Insert, Update, Delete on Categories to Admin;  
  
Grant Select, Insert, Update, Delete on Customers to Admin;  
  
Grant Select on DeletedRecords to Admin;  
  
Grant Select, Insert, Update, Delete on Employee to Admin;  
  
Grant Select, Insert, Update, Delete on OrderHistory to Admin;  
  
Grant Select, Insert, Update, Delete on Orders to Admin;  
  
Grant Select, Insert, Update, Delete on Products to Admin;
```

*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.

```
-- Çalışan rolünü oluşturun
CREATE ROLE Calisan;

-- Categories tablosuna erişim izni verin
GRANT SELECT, INSERT ON Categories TO Calisan;

-- Customers tablosuna erişim izni verin (CustomerPassword sütunu hariç)
GRANT SELECT (CustomerUsername, FirstName, LastName, Email, Phone) ON Customers TO Calisan;

-- Employee tablosuna erişim izni verin (Tablo düzeyinde)
GRANT SELECT, INSERT, UPDATE ON Employee TO Calisan;

-- Employee tablosundaki belirli sütunlara erişim izni verin (Sütun düzeyinde)
GRANT SELECT (EmployeeUsername, EmployeePassword, FirstName, LastName, Email, Phone) ON Employee TO Calisan;

-- Products tablosuna erişim izni verin
GRANT SELECT, INSERT, UPDATE, DELETE ON Products TO Calisan;

-- Orders tablosuna erişim izni verin
GRANT SELECT, UPDATE, DELETE ON Orders TO Calisan;

-- OrderHistory tablosuna erişim izni verin
GRANT SELECT ON OrderHistory TO Calisan;
```

### 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.

```
Create Role Müsteri;  
  
Grant Select,INSERT, Update on Customers to Müsteri;  
  
Grant Select on Categories to Müsteri;  
  
Grant Select on Products to Müsteri;  
  
Grant Select, Insert, Delete on Orders to Müsteri;  
  
Grant Select on OrderHistory to Müsteri;
```

### *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 işlemlerde 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

```
CREATE TRIGGER EmployeeDeletions
ON Employee
AFTER DELETE
AS
BEGIN
    INSERT INTO DeletedRecords (Table_Name, Record_ID, DeletedAt)
    SELECT 'Employee', deleted.EmployeeUsername, GETDATE()
    FROM deleted;
END;
```

*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.

```
CREATE TRIGGER CustomerDeletions
ON Customers
AFTER DELETE
AS
BEGIN
    INSERT INTO DeletedRecords (Table_Name, Record_ID, DeletedAt)
    SELECT 'Customers', deleted.CustomerUsername, GETDATE()
    FROM deleted;
END;
```

*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.

```
CREATE TRIGGER RolettoCustomer
ON Customers
AFTER INSERT
AS
BEGIN
    DECLARE @CustomerUsername NVARCHAR(50)

    -- Yeni eklenen Employee'yi al
    SELECT @CustomerUsername = CustomerUsername
    FROM inserted;

    IF @CustomerUsername IS NOT NULL AND @CustomerUsername <> ''
    BEGIN
        EXEC sp_addrolemember 'Müşteri', @CustomerUsername;
    END
END;
```

*Şekil 4.3*

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

**CREATE TRIGGER RolettoCustomer:** Bu ifade, "RolettoCustomer" 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 'Müşteri', @CustomerUsername;** Bu ifade, **sp\_addrolemember** prosedürünü çağırarak "Müşteri" rolüne ('Müşteri') 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 "Müşteri" rolüne ekleyecektir. Bu tür bir tetikleyici, kullanıcı rollerini otomatik olarak yönetmek ve güncellemek için kullanışlı olabilir.

```
CREATE TRIGGER RoleToEmployee
ON Employee
FOR INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted
    )
    BEGIN
        DECLARE @EmployeeUsername NVARCHAR(50)

        -- Yeni eklenen Employee'yi al
        SELECT TOP 1 @EmployeeUsername = EmployeeUsername
        FROM inserted;

        -- Yeni Employee'ye "calisan" rolünü ata
        IF @EmployeeUsername IS NOT NULL
        BEGIN
            EXEC sp_addrolemember 'Calisan', @EmployeeUsername;
        END
    END
END;
```

*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.

```

CREATE TRIGGER MoveOrderToHistory
ON Orders
AFTER UPDATE
AS
BEGIN
    DECLARE @OrderID INT;
    DECLARE @OrderStatus NVARCHAR(1);

    -- Güncellenen siparişin kimliğini ve durumunu al
    SELECT @OrderID = OrderID, @OrderStatus = OrderStatus
    FROM inserted;

    -- Eğer sipariş tamamlandı ise (örneğin "OrderStatus" sütunu G ise)
    IF @OrderStatus = 'G'
    BEGIN
        -- Silinen siparişi "OrderHistory" tablosuna ekle
        INSERT INTO OrderHistory (CustomerUsername, OrderDate, ProductID, Quantity, TotalPrice)
        SELECT od.CustomerUsername, od.OrderDate, od.ProductID, od.Quantity, od.TotalPrice
        FROM Orders od
        INNER JOIN Products p ON od.ProductID = p.ProductID
        WHERE od.OrderID = @OrderID;

        -- Silinen siparişi "Orders" tablosundan kaldır
        DELETE FROM Orders
        WHERE OrderID = @OrderID;
    END;
END;

```

**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.

```
CREATE TRIGGER UpdateStockOnOrderCompletion
ON OrderHistory
AFTER INSERT
AS
BEGIN
    DECLARE @ProductID INT;
    DECLARE @Quantity INT;

    -- Geçmiş siparişler tablosundan yeni eklenen kayıtları al
    SELECT @ProductID = ProductID, @Quantity = Quantity
    FROM inserted;

    -- Ürün stok miktarını güncelle
    UPDATE Products
    SET StockQuantity = StockQuantity - @Quantity
    WHERE ProductID = @ProductID;
END;
```

*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 taşıırken, **@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.

```
CREATE TRIGGER trg_Orders_Insert ON Orders
FOR INSERT
AS
BEGIN
    UPDATE Orders
    SET TotalPrice = Quantity * (SELECT Price FROM Products WHERE ProductID = Orders.ProductID)
    WHERE ProductID IN (SELECT ProductID FROM inserted)
END;
```

*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.

```
CREATE TRIGGER TrackProductsDeletions
ON Products
AFTER DELETE
AS
BEGIN
    INSERT INTO DeletedRecords (Table_Name, Record_ID, DeletedAt)
    SELECT 'Products', deleted.ProductID, GETDATE()
    FROM deleted;
END;
```

*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

```
CREATE PROCEDURE GetCustomerInfo
    @CustomerUsername varchar(50)
AS
BEGIN
    SELECT * FROM Customers WHERE CustomerUsername = @CustomerUsername;
END;
```

*Şekil 5.1*

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

```
CREATE PROCEDURE GetOrderHistoryByCustomer
    @kullaniciAdi NVARCHAR(50)
AS
BEGIN
    SELECT * FROM OrderHistory WHERE CustomerUsername = @kullaniciAdi;
END;
```

*Şekil 5.2*



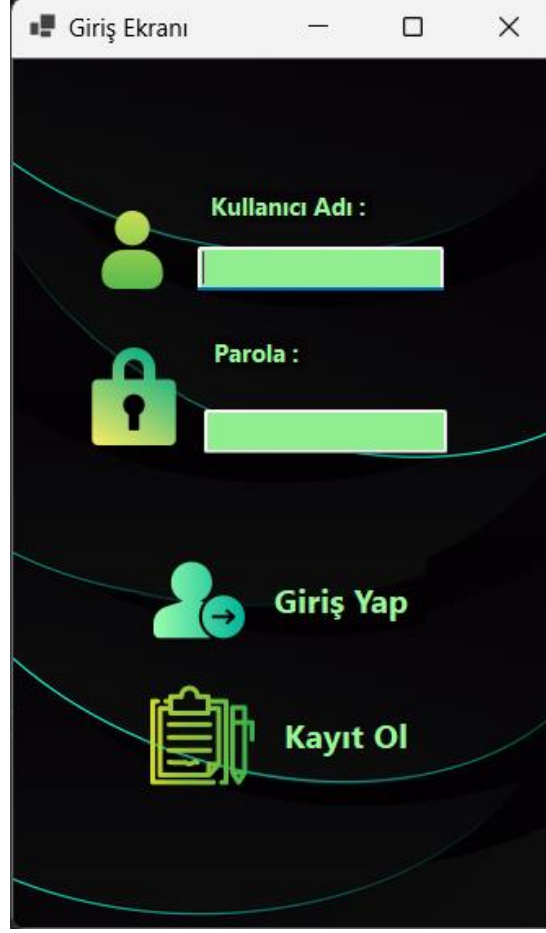
Şekil 5.2’de görünen GetOrderHistoryByCustomer Prosedürü, OrderHistory Tablosundaki verileri CustomerUsername’i @kullaniciAdi olan veriye göre filtreleme sorgusu işlemi görmektedir.

```
CREATE PROCEDURE UpdateProduct
    @ürün_adi NVARCHAR(255),
    @categoryID INT,
    @fiyat DECIMAL(10, 2),
    @stok INT,
    @productID INT
AS
BEGIN
    UPDATE Products
    SET ProductName = @ürün_adi,
        CategoryID = @categoryID,
        Price = @fiyat,
        StockQuantity = @stok
    WHERE ProductID = @productID;
END;
```

**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=OnlineAlisveris;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();
```

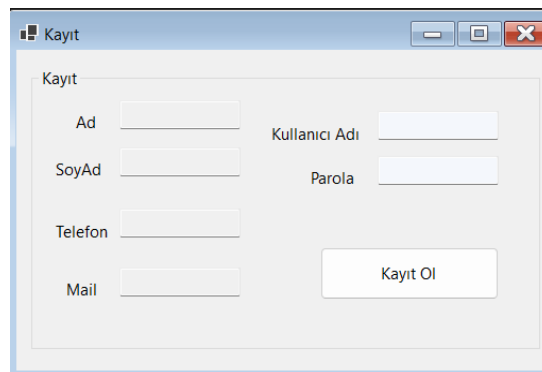
Bu bölüm, kullanıcı türüne bağlı olarak ilgili pencereyi açar. Örneğin, müşteri ise **MusteriGiris** penceresini, çalışan ise **personelgiris** penceresini açar. **this.Hide()** ile ise giriş yapan kullanıcı giriş penceresinin kapatılmasını sağlar.

#### 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=OnlineAlisveris;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 Parametrelili SQL Sorgusu Çalıştırma:

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

Bu blok, parametrelili 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)
```

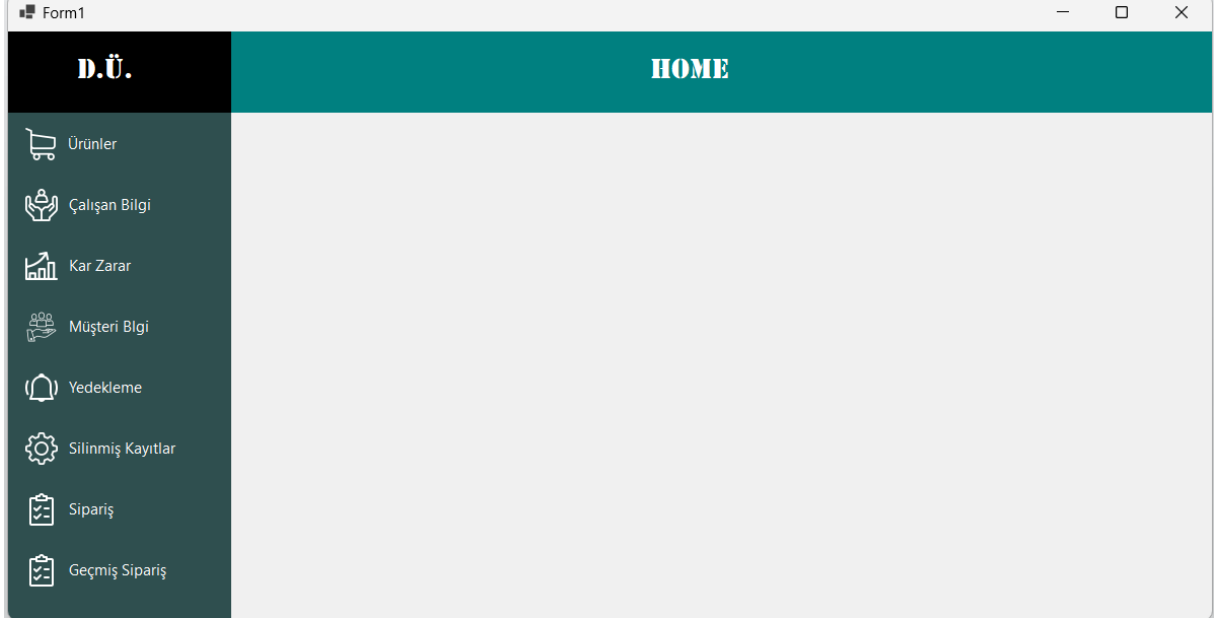
**ExecuteNonQuery** metodu, sorguyu çalıştırarak etkilenen satır sayısını döndürür. Eğer en az bir satır etkilenmişse, yani yeni bir kullanıcı başarıyla eklenmişse, kullanıcıya bir mesaj gösterilir. Aksi takdirde, eklenme işlemi başarısız olmuştur.

### 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. ADMIN 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

ProductID	ProductName	CategoryID	Price	StockQuantity
1	Akıllı Telefon	1	599,99	98
2	Klavye	1	49,99	46
3	Tişört	2	19,99	189
4	Çanta	3	39,99	146
5	Kitap 3	3	35,00	250
2007	Televizyon	1	400,99	100
2008	Bilgisayar	1	309,99	50
2009	Hırka	2	25,99	200
2010	Kalem	3	9,99	150
2011	Kazak	2	30,00	250
2012	Eldiven	3	15,99	100
2013	Mont	2	50,99	100

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



EmployeeUserName	EmployeePassword	FirstName	LastName	Email	Phone	Salary
ab	ab	aa	bb	ab	555	200
ömerbk	ab	ömer	kaya	ab	555	50000

**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 metod, **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 metod, 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 metod, **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

CustomerUsername	CustomerPassword	FirstName	LastName	Email	Phone
Emre_Kaan	emre123	Emre	Kaan	Emre.Kaan@email...	05316117472
janedoe	securepass	Jane	Doe	janedoe@email...	555-987-6543
johndoe	password123	John	Doe	johndoe@email...	555-123-4567
user3	userpass	User	Lastname	555-555-5555	user3@email.co...

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 metod, **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

Sipariş Tarihi	Toplam Fiyat	Sipariş ID
27.10.2023 12:45:00	49,99	1
24.12.2023 13:13:40	149,97	1002
24.12.2023 13:13:40	79,96	1003
14.12.2023 18:32:40	1199,98	1004
24.12.2023 13:43:46	159,96	1006
24.12.2023 13:43:45	59,97	1007

Brüt Gelir : 1699,83  
Kar : 509,949

Ay : Bütün Aylar  
Yıl : 2023  
SEÇ

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.MonthGenitiveNa
mes; 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.

OrderHistoryID	OrderDate	CustomerUsernam	ProductID	Quantity	Total
001	27.10.2023 12:45	user3	2	1	49,99
002	24.12.2023 13:13	user3	2	3	149,97
003	24.12.2023 13:13	user3	3	4	79,96
004	14.12.2023 18:32	johndoe	1	2	1199,9
006	24.12.2023 13:43	user3	4	4	159,96
007	24.12.2023 13:43	user3	3	3	59,97

**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.
12. namespace WinFormsApp1
13. {
14.     public partial class Admin_Gecmis_Siparis : Form
15.     {
16.         string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial
Catalog=OnlineAlisveris;Integrated Security=True";
17.
18.         private DataTable originalDataTable;
19.         public Admin_Gecmis_Siparis()
20.         {
21.             InitializeComponent();
22.         }
23.
24.         private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
25.         {
26.             using (SqlConnection connection = new SqlConnection(connectionString))
27.             {
28.                 // SQL sorgusu

```

```

29.         string query = "SELECT * FROM OrderHistory";
30.
31.         // SQL sorgusunu çalıştırmak için bir SqlDataAdapter ve bir DataTable kullanılır.
32.         SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
33.         originalDataTable = new DataTable(); // Orijinal DataTable'ı sınıf seviyesinde tanımlanan değişkenle eşleştiriyoruz.
34.
35.         // Verileri çek ve DataGridView'i
36.         // doldur
37.         adapter.Fill(originalDataTable);
38.         dataGridView1.DataSource = originalDataTable;
39.     }
40. }
41.
42. private void textBox1_TextChanged(object sender, EventArgs e)
43. {
44.     if (originalDataTable != null)
45.     {
46.         string aramaMetni = textBox1.Text.Trim();
47.
48.         // Orijinal DataTable'ın bir kopyasını al
49.         DataTable filteredDataTable = originalDataTable.Clone();
50.
51.         // DataView oluşturup orijinal DataTable'ı filtreliyoruz.
52.         DataView dv = originalDataTable.DefaultView;
53.
54.         // Eğer aramaMetni sayısal bir değerse
55.         if (int.TryParse(aramaMetni, out _))
56.         {
57.             dv.RowFilter = $"OrderHistoryID = {aramaMetni}";
58.         }
59.         else
60.         {
61.             // Eğer aramaMetni metin bir değerse
62.             dv.RowFilter = $"CustomerUsername LIKE '%{aramaMetni}%';";
63.         }
64.
65.         // Filtrelenmiş verileri kopyalanmış DataTable'a ekliyoruz.
66.         foreach (DataRowView drv in dv)
67.         {
68.             filteredDataTable.ImportRow(drv.Row);
69.         }
70.
71.         // Kopyalanmış DataTable'ı DataGridView'e yüklüyoruz.
72.         dataGridView1.DataSource = filteredDataTable;
73.     }
74. }
75.
76. private void Admin_Personel_Gecmis_Siparis_Load(object sender, EventArgs e)
77. {
78.     dataGridView1_CellContentClick(sender, new DataGridViewCellEventArgs(0, 0));
79. }
80.
81. private void button1_Click(object sender, EventArgs e)
82. {
83.     // DataGridView'de seçili satırın index'ini al
84.     int selectedIndex = dataGridView1.CurrentCell.RowIndex;
85.
86.     // Seçili satırın OrderID değerini al
87.     int orderID = Convert.ToInt32(dataGridView1.Rows[selectedIndex].Cells["OrderHistoryID"].Value);
88.
89.     // OrderStatus'u "G" olarak güncelle
90.     DeleteOrderStatus(orderID);

```

```

91.
92.     // Güncellenmiş verileri DataGridView'e yükle
93.     dataGridView1_CellContentClick(sender, new DataGridViewCellEventArgs(0, 0));
94. }
95.
96. private void DeleteOrderStatus(int orderID)
97. {
98.     using (SqlConnection connection = new SqlConnection(connectionString))
99.     {
100.         connection.Open();
101.
102.         // SQL sorgusu
103.         string query = $"DELETE From OrderHistory WHERE OrderHistoryID = {orderID}";
104.
105.         using (SqlCommand command = new SqlCommand(query, connection))
106.         {
107.             // Sorguyu çalıştır
108.             command.ExecuteNonQuery();
109.         }
110.     }
111. }
112. }
113. }

```

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=OnlineAlisveris;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 selectedIndex = dataGridView1.CurrentCell.RowIndex;

    // Seçili satırın OrderID değerini al
    int orderID = Convert.ToInt32(dataGridView1.Rows[selectedIndex].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 selectedIndex = dataGridView1.CurrentCell.RowIndex;

    // Seçili satırın OrderID değerini al
    int orderID = Convert.ToInt32(dataGridView1.Rows[selectedIndex].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();

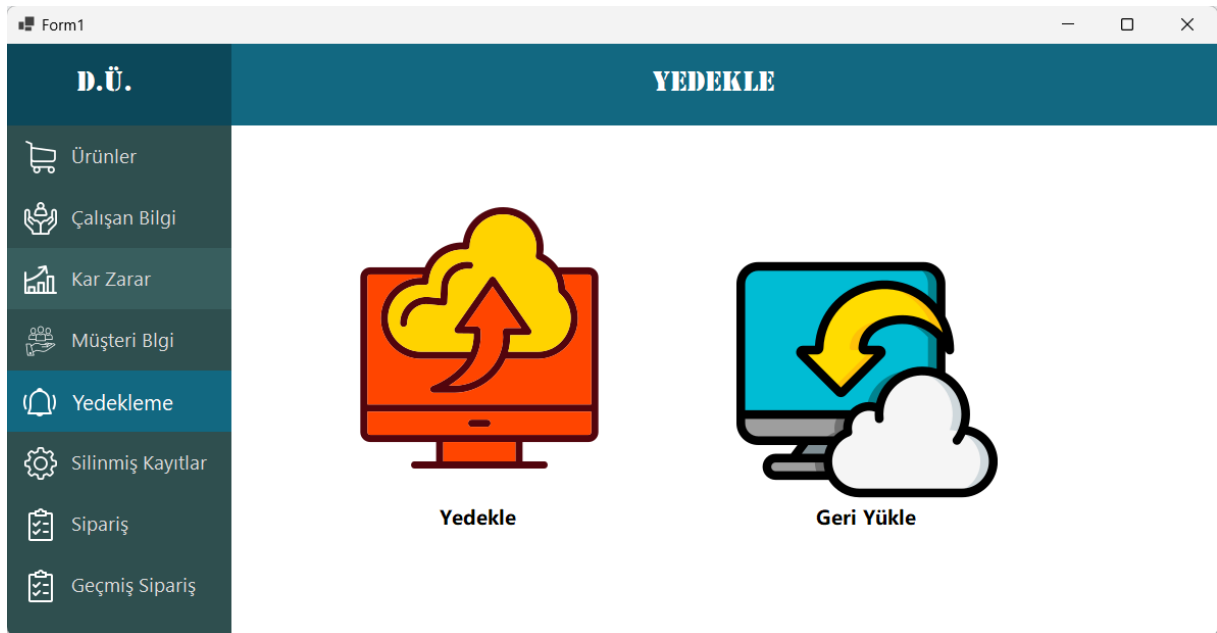
    // SQL sorgusu
    string query = $"DELETE From Orders WHERE OrderID = {orderId}";

    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=OnlineAlisveris;Integrated Security=True";

                // Veritabanı bağlantısı oluştur
                using (SqlConnection connection = new SqlConnection(connectionString))
                {
                    connection.Open();

                    // Backup sorgusu
                    string backupQuery = $"BACKUP DATABASE [OnlineAlisveris] 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 [OnlineAlisveris] 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 kullanimdanKaldirQuery = "USE master ALTER DATABASE [OnlineAlisveris] SET SINGLE_USER
WITH ROLLBACK IMMEDIATE";
    string geriAlQuery = "USE master ALTER DATABASE [OnlineAlisveris] 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(kullanimdanKaldirQuery, 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. **KullanımdanKaldır** metodu, veritabanını kullanımdan kaldırmak için kullanılır. Bu işlem, veritabanının kilitlenmesini ve ardından yedekleme veya geri yükleme işlemlerini mümkün kılar.

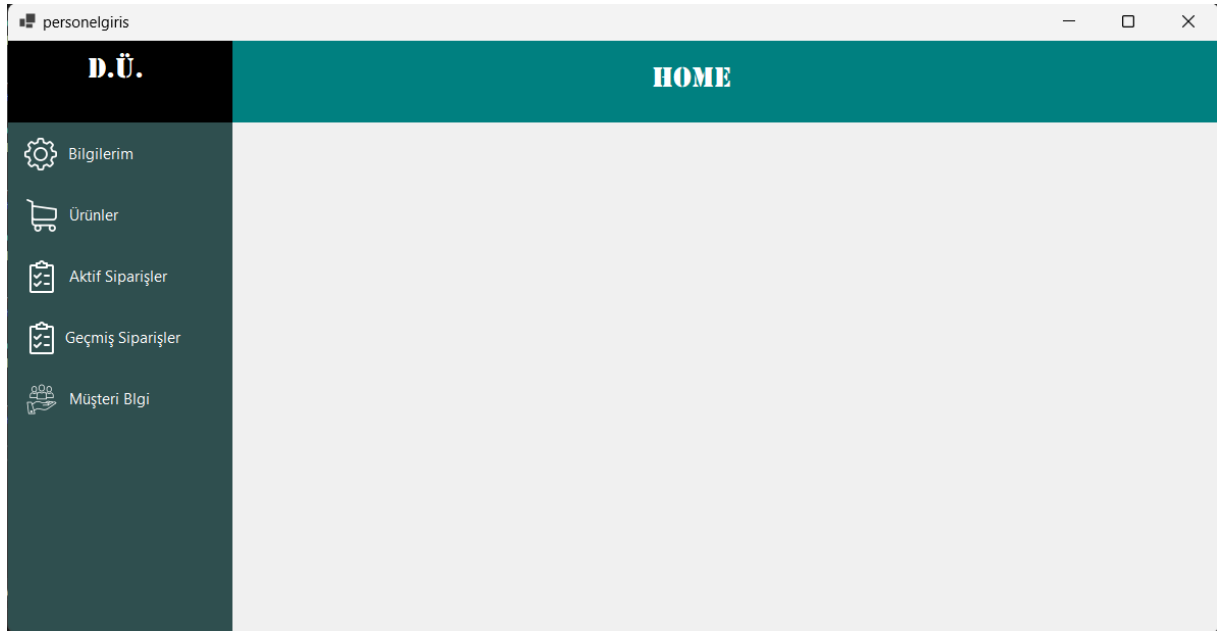
## 15. SİLİNİMİŞ KAYITLAR FORMU

	Table_Name	Record_ID	DeletedAt
▶	Customers	Emre_Kaan	27.10.2023 05:49
	Customers	Emre_Kaan	27.10.2023 05:50
	Customers	Emre_Kaan	28.10.2023 17:59
	Employee	emre	6.11.2023 19:14
	Employee	melih	7.11.2023 19:14
	Employee	aaa	8.12.2023 12:14
	Employee	asdads	9.12.2023 18:59
	Products	1002	12.12.2023 17:05
	Products	2006	12.12.2023 17:39
	Products	2005	13.12.2023 14:06
*			

*Şekil 15.1*

Şekil 15.1’de görünen C# programı, bir Windows Forms uygulaması aracılığıyla SQL Server veritabanında silinen kayıtların tutulduğu 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.

#### 2. 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.

#### 3. 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.

#### 4. 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.

#### 5. 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.

#### 6. 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.

#### 7. 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



CustomerUsername	FirstName	LastName	Email	Phone
Emre_Kaan	Emre	Kaan	Emre.Kaan@email.c...	05316117472
janedoe	Jane	Doe	janedoe@email.com	555-987-6543
johndoe	John	Doe	johndoe@email.com	555-123-4567
user3	User	Lastname	555-555-5555	user3@email.com

**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.

### 2. MusteriGetir Metodu:

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

Bu metod, müşteri verilerini veritabanından çekip DataGridView kontrolüne yükler.

### 3. 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.

### 4. Düzenleme Butonu:

```
private void btnDuzenle_Click(object sender, EventArgs e) { // ... (Müşteri bilgilerini  
güncelleme işlemleri) }
```

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

#### 5. 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.

#### 6. 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

The screenshot shows a Windows application window titled 'personelgiris'. The main form is titled 'D.Ü. BİLGİLERİM'. On the left, there is a sidebar with a dark blue background and white text. The sidebar contains five items: 'Bilgilerim' (with a gear icon), 'Ürünler' (with a shopping cart icon), 'Aktif Siparişler' (with a list icon), 'Geçmiş Siparişler' (with a list icon), and 'Müşteri Bİgi' (with a person icon). The main area of the form is light gray and contains several input fields and a button. The input fields are arranged in two columns. The first column contains 'Kullanıcı Adı' (User Name) with the value 'ab', 'Şifre' (Password) with the value 'ab', 'İsim' (Name) with the value 'aa', and 'Soyisim' (Surname) with the value 'bb'. The second column contains 'Mail' with the value 'ab', 'Telefon' (Phone) with the value '555', and 'Maaş' (Salary) with the value '200'. At the bottom right of the main area is a 'Kaydet' (Save) button.

*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 çalışanın temel bilgilerini içerir. Bu sınıf, veritabanından çekilen personel bilgilerini temsil etmek için kullanılır.

## 2. 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.

## 3. 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.

## 4. 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.

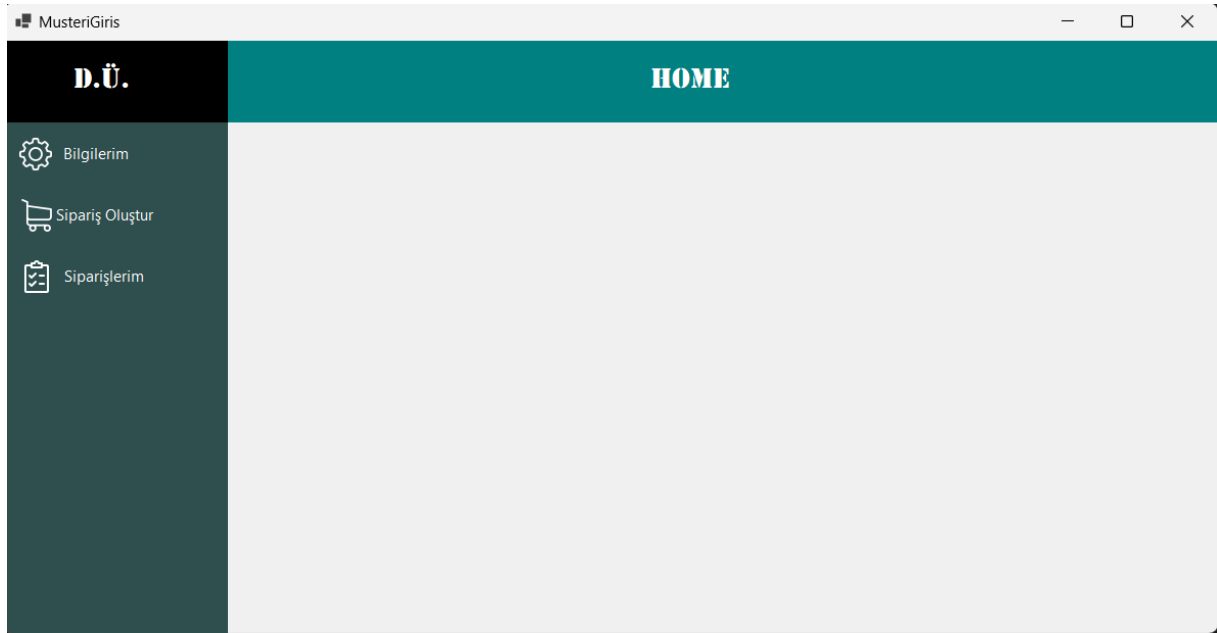
## 5. Veri Güncelleme İşlemi:

```
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.

### 2. 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.

### 3. 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.

### 4. 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

The screenshot shows a Windows Forms application titled 'MusteriGiris'. The main window has a dark blue header with 'D.Ü.' on the left and 'MUSTERIBILGILERIM' on the right. Below the header is a sidebar with three items: 'Bilgilerim' (with a gear icon), 'Sipariş Oluştur' (with a shopping cart icon), and 'Siparişlerim' (with a clipboard icon). The main content area is light gray and contains a form with the following fields and labels: 'Kullanıcı Adı' (user3), 'Sifre' (userpass), 'Ad' (User), 'Soyad' (Lastname), 'Mail' (555-555-5555), and 'Telefon' (user3@email.com). A 'Kaydet' button is located at the bottom right of the form.

**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.

### 2. 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.

### 3. 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.

#### 4. 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=OnlineAlisveris;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.

#### 5. 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=OnlineAlisveris;Integrated Security=True")) { connection.Open(); using
(SqlCommand command = new SqlCommand(updateQuery, connection)) { //
Parametreleri ekleyin command.Parameters.AddWithValue("@NewPassword",
```

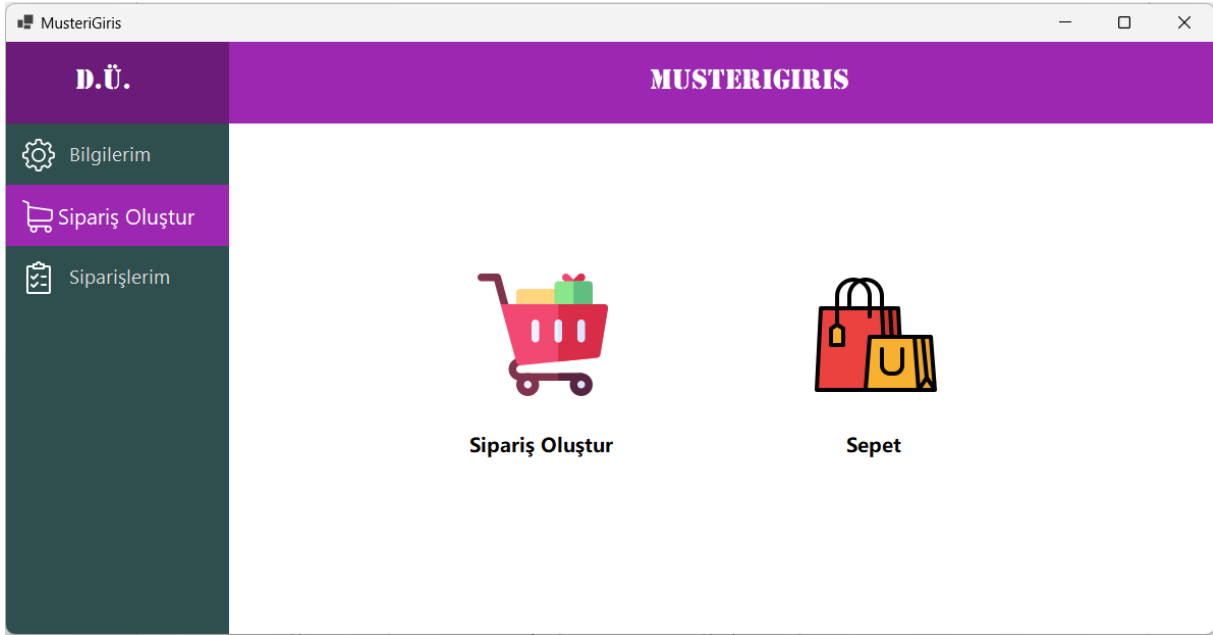
```
newPassword); command.Parameters.AddWithValue("@NewEmail", newEmail);
command.Parameters.AddWithValue("@NewPhone", newPhone);
command.Parameters.AddWithValue("@CustomerUsername", KName); // Sorguyu
çalıştırın int affectedRows = command.ExecuteNonQuery(); if (affectedRows > 0) {
MessageBox.Show("Veriler 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.

## 6. Customer Sınıfı:

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

## 21. MÜŞTERİ SİPARİŞ FORMU

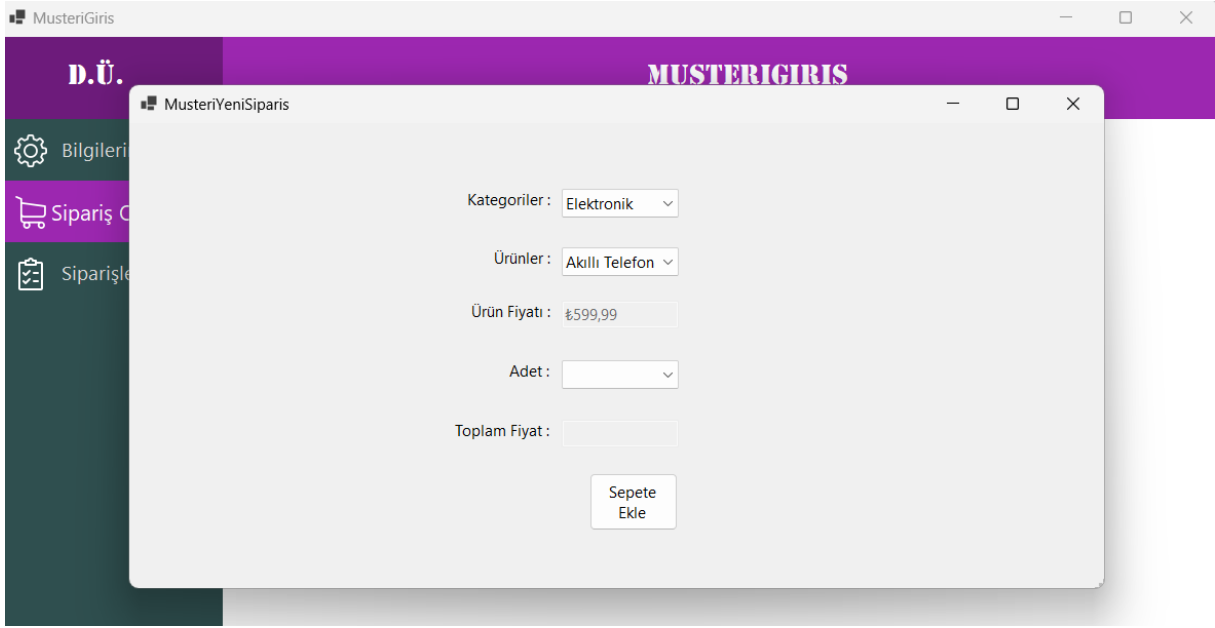


*Şekil 21.1*

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.

**2. 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.

### 3. **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.

### 4. **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.

### 5. **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); } }

```

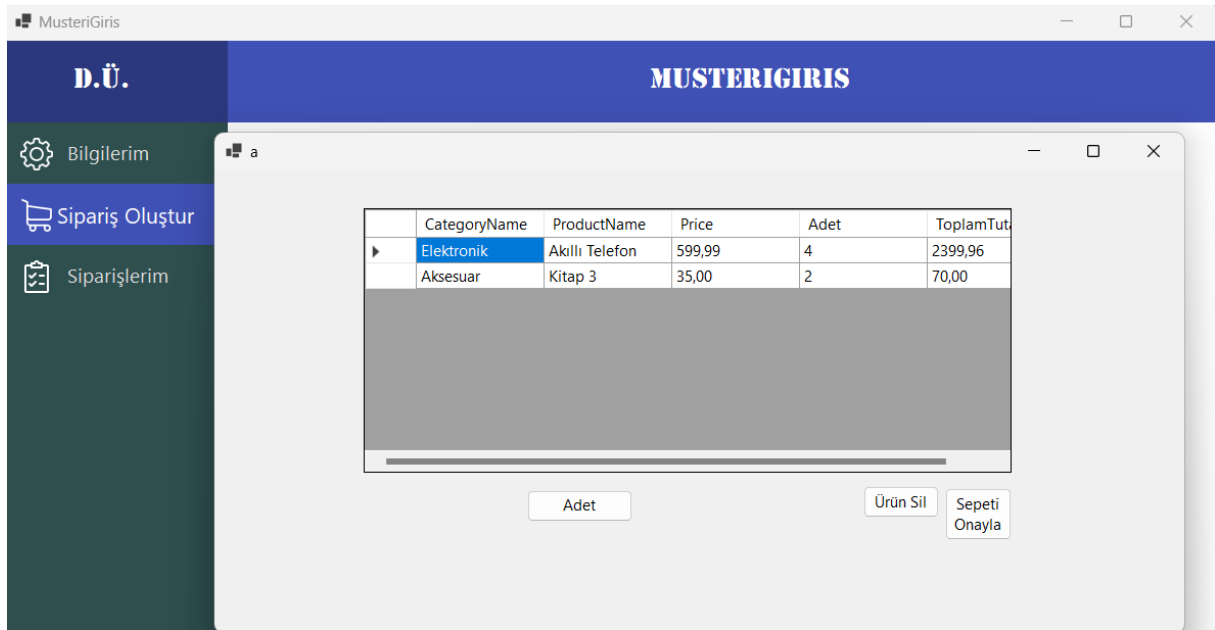
Bu metod, müşterinin seçtiği ürünü ve adeti sepete ekler. Eklenen ürün bilgilerini **sepetVerileri** adlı DataTable'a ekler.

### 6. **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.

### 2. 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.

### 3. 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.

### 4. 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.

### 5. 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.

## 6. 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 muster =
Application.OpenForms["MusteriGiris"] as MusteriGiris; muster.dt.Clear();
muster.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.

## 7. 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 deęerini döndürecektir. object result = command.ExecuteScalar();
// Eęer result null deęilse ve bir int'e dönüştürülebiliyorsa, bu deęeri döndür. if (result
!= null && int.TryParse(result.ToString(), out int productID)) { return productID; } } }
} catch (Exception ex) { MessageBox.Show("Veritabanından veri getirme sırasında
hata oluřtu: " + ex.Message, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);
} // Hata durumlarında veya ProductID bulunamadığında -1 deęerini döndürebilirsiniz.
return -1; }

```

Bu metod, bir ürünün ismi verildiğinde bu ürünün ID'sini veritabanından alır. Eęer ürün bulunamazsa veya bir hata oluřursa -1 deęerini döndürür.

Bu açıklamalar, **MusteriSepet** formundaki önemli metodları ve kontrol akışını kapsamaktadır. Bu formun işlevsellięi, müşterinin sepetine ürün eklemesi, çıkarması ve sipariř oluřturması üzerinedir.

## 24. MÜŞTERİNİN SİPARİřLERİNİ GÖRME FORMU

OrderID	OrderDate	CustomerUserman	ProductID	Quantity	OrderStatus
1005	24.12.2023 13:14	user3	4	5	A
1006	24.12.2023 13:18	user3	1	2	A
3007	25.12.2023 20:47	user3	1	1	A

řekil 24.1

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

### 1. Constructor (Yapıcı Metod):

```

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

```

Bu metod, form oluřturulduğunda çalışır. ComboBox'ın varsayılan seçeneęini (onay bekleyen sipariřler) belirler.

## 2. Onaylanan Siparişleri Getiren Metod:

```
public void Onaylanan() { string kullanıcıAdi = 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", kullanıcıAdi); // 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.

## 3. Onay Bekleyen Siparişleri Getiren Metod:

```
public void OnayBekleyen() { string kullanıcıAdi = 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", kullanıcıAdi); // 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.

## 4. 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