ADO.Net

1. **ADO.Net Giriş**
2. **Connection String**
3. **ADO.Net Basic Objects (SqlConnection, SqlCommand)**
4. **ExecuteScalar**
5. **ExecuteNonQuery**
6. **ExecuteReader**
7. **DataSet ve DataTable**
8. **DataAdapter kullanarak DataSet'i doldurma**
9. **DataReader kullanarak Data almak**

ADO.Net Giriş

Ado.net, veritabanlarındaki bilgileri elde etmek veya veritabanında bulunan verileri değiştirmek için geliştirilmiş bir veri erişim aracıdır.Ado.net, .net sınıf kütüphanesindeki System.Data isim alanındaki sınıfların tamamını temsil eder.

ADO.NET (ActiveX Data Objects.NET) , Aktivex tabanlı çalışan, Microsoft'un veri erişim teknolojisidir.

Bu veri erişim, sadece SQL Database bağlantısı değildir, Oracle, mySql, Access ve tüm office araçları olmak üzere tüm database tiplerine erişim imkânı sunmaktadır.

ADO.NET, geliştireceğimiz uygulamamız ile Database arasında tam bir Köprü görevi görür. Verileri okuyabilir, update edebiliriz. SQL sorguları ve procedure leri çalıştırtabiliriz. Bu sebeptendir ki, veri tabanı bağlantılı geliştirdiğimiz tüm uygulamamızda kullanmamız gereken en önemli teknolojidir ADO.NET.

ADO.NET ile ilgili kullanacağımız tüm classlar, Microsoft'un "**using System.Data**" adlı kütüphanesinde mevcuttur. Ve bizde tüm işlemlerimizde bu kütüphanemizden faydalanacağız.

ADO.NET 'in veri tabanına 2 tür bağlantı şekli vardır;

Connected, Disconnected..

Connected Bağlantı Şekli

Bu bağlantı şeklimizde, Uygulamamız, database ile sürekli bağlantılı haldedir ve anlık olarak database de okuma ve update işlemlerimiz yapılmaktadır.

Bağlantı sürekli olduğu için, veri okuma işlemi son satırı okuyana kadar açık kalması gerekmektedir, bu da uygulamamızda biraz yavaşlığa sebep olabilir.

Disconnected Bağlantı Şekli

Bu bağlantı şeklinde uygulamamız, veri tabanını kullanmak için sürekli ona müracaat etmek yerine, veri tabanının bir kopyasını RAM üzerine alır ve bundan sonraki tüm işlemlerini RAM deki veri üzerinde yapar.

RAM'e alınan veri üzerinde, insert, update ve delete işlemleri kolaylıkla yapılır ve bu işlem için database'e kadar gitmemize gerek yoktur. Fakat anlaşılacağı gibi yaptığımız değişiklikler RAM'deki veri üzerinde olmaktadır. Gerçek database deki değişiklik için ayrıca bir işlem gerektirir.

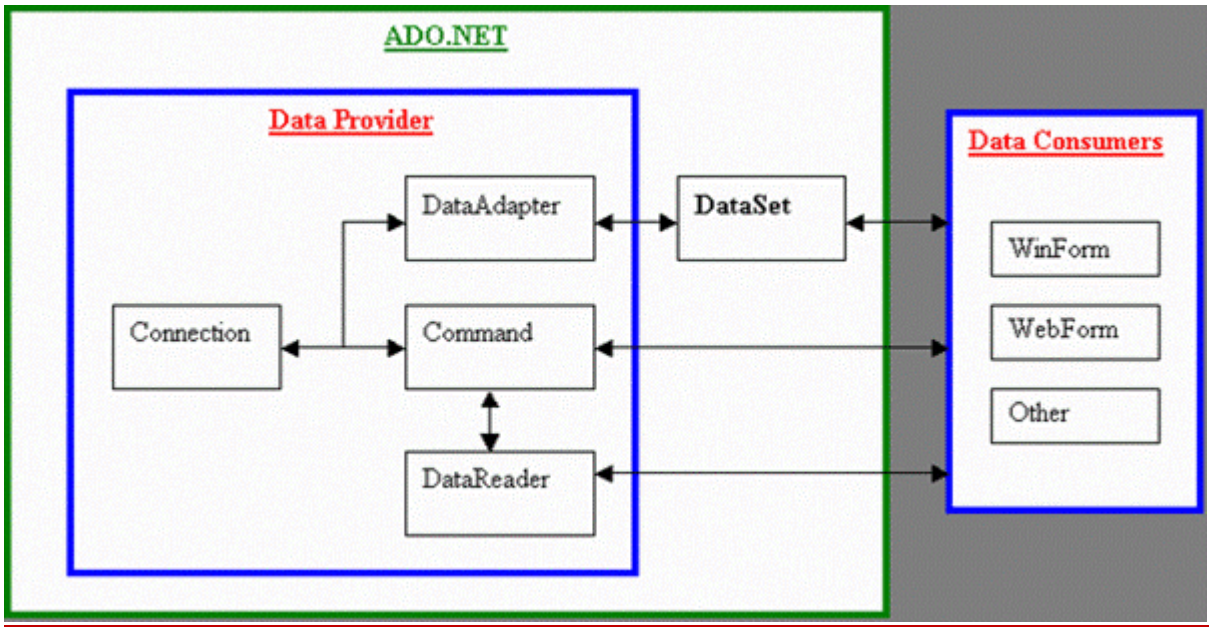
Bu bağlantı şeklinin en önemli avantajı performanstır. Çünkü bilmemiz gereken en önemli şeylerden biri, veri tabanına bağlantı framework açısından her zaman zahmetli bir iştir ve bize yavaşlık getirir. Bu modelde, veri tabanına 1 kere bağlanılmakta ve istediğimiz veriyi RAM'e aldıktan sonra bir daha database'e gitmemize gerek kalmamaktadır.

Dezavantajı ise, Gerçek database deki verilerin güncellenmesi geç yapıldığı için zaman zaman veri uyuşmazlığı ve tutarsızlığı yaşanmaktadır.

Bu bağlantı şekline örnek olarak: Sahada kullanılan bir çok el terminalleri bu mantıkla çalışmaktadır. El terminallerine sabah veriler yüklenir (RAM e yükleme aşaması) ve gün boyunca cihazlar ile bilgi girişleri yapılmaktadır. Bunların hepsi cihazdaki RAM de tutulur, ve gün sonunda cihazlar internete bağlandıktan sonra gerçek database ile Senkronizasyon işlemi yapılır. Yani database'e sadece 2 defa gidilir. Buda bize performans sağlamaktadır.
(Günümüzde artık 3G teknolojisi olduğu için, bir çok terminallerde internet bağlantısı olduğu için Connected bağlantı şeklinin kullanıldığı durumlarda vardır)

DataAdapter : Gönderdiğimiz sorgu sonucu gelen veriyi geçici hafızaya alan sınıfımızdır. Database çeşidine göre farklılık gösterir.
(SqlDataAdapter, OleDbDataAdapter, OracleDataAdapter)

*DataAdapter ile database den verimizi çekeriz, fakat alınan veriyi kullanabilmemiz için **DataTable** ve **DataSet** gibi veri türüne dönüştürmemiz gerekmektedir.*
Connected ve Disconnected bağlantı çeşitlerini ve çalışma prensibini daha iyi anlayabilmek için aşağıdaki tablo bize yardımcı olacaktır.



Yukarıda görüldüğü gibi ADO.NET teknolojisinde bir connection için 3 farklı yolumuz vardır.
DataAdapter: *Disconnected bağlantılar*
Command, DataReader: *Connected Bağlantılar*

ADO.NET teknolojisini kullanacak uygulama tarafımızda bir kısıtlama olmadığını görmekteyiz. Yani Windows Form uygulamalarında, Web Uygulamalarında ve diğer tüm Database bağlantısı gerektirecek uygulamalarda bu teknolojiden faydalanabilmekteyiz.

Burada önemli bi nokta da, aslında Disconnected Bağlantı modelinde bahsetmiştik, ***DataAdapter*** sınıfımız ile uygulamamız arasında ***DataSet*** ve ***DataTable*** gibi aracı bir sınıfımız daha vardır. Bu aracı sınıf ile beraber, disconnected bağlantılarımızı rahatlıkla uygulamalarımızda kullanabilmekteyiz.

Connection String

SqlConnection : Bu sınıf ile SQL'de yer alan veritabanlarımıza bağlanabiliriz. System.Data.SqlClient namespace' i içerisinde yer almaktadır.

```
using System.Data.SqlClient; --using'e eklenir.
```

```
-- Connection nesnesi yaratılır. (SQL auth.)
```

```
SqlConnection conn = new
SqlConnection("Server=.;database=Northwind;UID=kullaniciAdi;PWD=şifre");
```

```
--(Windows auth.)

SqlConnection conn = new SqlConnection("server=.;Database=Northwind;Integrated
Security=true;");
```

Server (Data Source): Bağlanacağımız bilgisayarın adı yada ip numarasını belirtiriz. Localhost kullanıldığında ismi **localhost** ya da **'.'** (nokta) yazarak belirtilebilir.

Database: Bağlanacağımız database'in adını belirtiriz.

User ID(uid): Bağlanacağımız veritabanına hangi kullanıcı adı ile gireceğimizi belirtiriz.

Password (pwd): User id'mize ait şifremizi belirtiriz.

SQL server iki tip oturum açma şekli sunar. Bunlar **SQL Server Authentication** ve **Windows Authentication** olarak adlandırılır. Windows Authentication herhangi bir kullanıcı adı ve şifre girilmesine ihtiyaç duymaz. Çünkü ilgili makinede SQL kuruludur. Tabi istenirse girilebilir o ayrı.. Connection String yazarken eğer Windows Authentication ile bağlantı kuracaksam **Integrated Security = True** deyimini kullanmam gerek.

Her sağlayıcının kendisine özel connection string'i (bağlantı cümleciği) vardır. www.connectionstrings.com adresinden bulabilirsiniz.

ConnectionString'i 4 farklı şekilde yaratabiliriz;

1. Class içinde her defasında yeni bir connection nesnesi oluşturarak.

```
SqlConnection conn = new SqlConnection("Server=.; Database=northwind; UID=kullaniciAdi;
PWD=şifre");
```

2. Global Scope'da connection nesnesi oluşturarak. (Constructor metodunun hemen altında)

```
SqlConnection conn = new SqlConnection("Server=.; Database=northwind; UID=kullaniciAdi;
PWD=şifre");
```

3. Class olarak tanımlayarak. (sadece "get" olarak)

```
public static string ConnectionString
        {
                get {
                        return "Server=.; database=northwind; UID=kullaniciAdi; PWD=şifre";
                        }
        }
```

```
// “Tools” isminde oluşturduğumuz ConnectionString class’ını Form’a çağırma;

SqlConnection conn = new SqlConnection(Tools.ConnectionString);
```

4. App.config veya web.config içinde oluşturarak.

```
<configuration>
        <connectionStrings>
                <add name="erman"
                        connectionString="Server=.; database=northwind; UID=kullaniciAdi;
                        PWD=şifre" providerName="System.Data.SqlClient"/>
        </connectionStrings>
</configuration>
```

```
// App.config’de oluşturduğumuz ConnectionString’i Form’a çağırma;      SqlConnection
conn = new SqlConnection
(ConfigurationManager.ConnectionStrings["erman"].ConnectionString);

// veya

SqlConnection conn = new SqlConnection();

conn.ConnectionString =
ConfigurationManager.ConnectionStrings["erman"].ConnectionString;
```

SqlCommand

Bir bağlantı nesnesi ve bir SQL cümleciği ile oluşturulabilir. SQL cümleciği veritabanından sorgulama yapmamızı sağlar. Database çeşidine göre farklılık gösterir.
(**SqlCommand**, OleDbCommand, OracleCommand) gibi..

```
string sorgu = "select firstname from employees";
SqlCommand cmd = new SqlCommand(sorgu, conn);

-- veya

SqlCommand cmd = new SqlCommand("select firstname from employees", conn);
```

DataReader : SqlCommand nesnesinin çalıştırılması ile elde edilen sonuçları okumak için kullanılır. Sorgu sonucumuzda elde ettiğimiz veride , satır satır dönmemize yarayan sınıftır. Database çeşidine göre farklılık gösterir.

(SqlDataReader OleDbDataReader, OracleDataReader) gibi..

SqlCommand'ı 2 farklı şekilde yaratabiliriz;

```
// Genel bir SqlConnection tanımlıyoruz.
SqlConnection conn = new SqlConnection("Server=.; Database=northwind; UID=kullaniciAdi;
PWD=şifre");
```

1. Bir bağlantı nesnesi ve bir SQL cümleciği ile oluşturulabilir. *(Sürekli kullandığımız yol)*

```
SqlCommand cmd = new SqlCommand("Select FirstName From Employees", conn);
```

2. CommandType özelliğini kullanarak. Bu özellik ile komutun ne şekilde oluşturulacağı bildirilir. Text ve StoredProcedure olacak şekilde ayarlanabilir. Varsayılan olarak Text'tir. Bu özelliğin text olması, komutun bir SQL cümleciği olduğunu göstermektir.

```
SqlCommand cmd = new SqlCommand();
cmd.CommandType = CommandType.Text; // CommandType.StoredProcedure olabilir.
cmd.CommandText = " Select FirstName From Employees";
cmd.Connection = conn;
```

ExecuteScalar

Bu metot ile veritabanından tek bir değer elde edilir. Örneğin bir tablodaki kayıt sayısını elde etmek için kullanılan COUNT komutunun kullanıldığı SQL cümleciklerinin çalıştırılması ExecuteScalar() metodu ile yapılabilir. Aşağıdaki programda Employees tablosundaki kayıt sayısının nasıl elde edildiği görülmektedir.

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace Program
{
  public class AccessSql
  {
    public static void Main()
    {
      string kaynak = "Server=.;Database=Northwind;Integrated Security=SSPI;";

      SqlConnection baglanti = new SqlConnection(kaynak);

      baglanti.Open();
```

```
            string sorgu = "select count(*) from employees";

            SqlCommand komut = new SqlCommand(sorgu, baglanti);

            int kayit = (int)komut.ExecuteScalar();

            Console.WriteLine("Toplam Kayıt Sayısı: " + kayit);

            baglanti.Close();

            Console.ReadKey();
        }
    }
}
```

ExecuteScalar metodunun geri dönüş değeri object türünden olduğu için kayıt sayısını elde etmek için int türüne dönüştürme işlemi yapıldı. Programı çalıştırdığınızda Employees tablosundaki toplam kayıt sayının ekrana yazıldığını göreceksiniz.

ExecuteNonQuery

Bu metodu genellikle bir veritabanındaki kayıtları değiştirmek. Silmek ya da yeni bir kayıt eklemek istediğimizde kullanırız. Bu metodun geri dönüş değeri komutun çalıştırıldıktan sonra etkilediği kayıt sayısıdır.

Bir tabloya yeni bir kayıt eklemek için aşağıdaki SQL cümleciği kullanılabilir;

```
INSERT INTO Üyeler (Sütunlar) VALUES (Sütun Değerleri)
```

Örnek(Insert) : Veritabanında ki bir tabloya yeni bir kaydın nasıl eklendiğini gösterelim;

```
static void Main(string[] args)
    {
        string kaynak = "Data Source=.;Initial Catalog=Deneme;Integrated Security=SSPI;";
        SqlConnection con = new SqlConnection(kaynak);
        con.Open();

        string sorgu = @"Insert into kisi (ad,soyad) values ('Erman','Zöhre')";

        SqlCommand cmd = new SqlCommand(sorgu, con);
```

```
        Console.WriteLine("Etkilenen satır sayısı: " + cmd.ExecuteNonQuery()); //consolde etkilenen satır sayısını görebiliriz bu şekilde.

        Console.ReadKey();
        con.Close();
         }
```

Örnek(Update) : Veritabanında bazı kriterlere uyan kayıtları değiştirmek (güncelleştirmek) için de Update metodu kullanılır. Mesela şifresi 1234 olan kullanıcıyı 5678 yapalım.

```
UPDATE Kisi SET sifre=5678 where sifre=1234
```

**ExecuteNonQuery() metodunun geri dönüş değeri komutun çalıştırılması sonucu elde eldilen kayıt sayısı, silinen ya da değiştirilen kayıt sayısı olabilir.

ExecuteReader

Bu metodun SQL cümleciğinin SELECT komutu içerdiği durumlarda kullanıldığı sıklıkla görülür. SELECT komutu ile bir veritabanından bir sorgu yapılarak bir kayıt kümesi elde edilir. Elde edilen bu kayıt kümesi SqlDataReader nesnesine aktarılır. SqlDataReader ile bu kayıt kümesinden çeşitli yollardan veriler elde edilebilir. ExecuteReader() metoduna bir örnek vermeden önce SqlDataReader sınıfının özelliklerini inceleyelim.

SqlDataReader nesnesi ile bir kayıttan sadece ileriye doğru ilerleme yapılır. Yani bir kayıt kümesinden bir kayıt okuduktan sonra tekrar geriye dönüp eski kayıtları okumak mümkün değildir. SqlDataReader nesnelerini *new operatörü* ile doğrudan oluşturamayız. Bunun yerine SqlCommand nesnesinin ExecuteReader() metodunun geri dönüş değeri kullanılır.

Örnek : ExecuteReader() metodu kullanılarak SqlDataReader nesnesi ile Kisi tablosundaki kişilerin ad,soyad ve şifrelerini ekrana yazdırınız.

```
class Program
  {
    static void Main(string[] args)
    {
      string kaynak = "Data Source=.;Initial Catalog=Northwind;Integrated Security=SSPI;";
      SqlConnection con = new SqlConnection(kaynak);
      con.Open();
```

```
            string sorgu = "select * from Employees";

            SqlCommand cmd = new SqlCommand(sorgu, con);

            SqlDataReader dr = cmd.ExecuteReader();

            Console.WriteLine("{0,-10}{1,-10}{2,-10}", "ad", "soyad", "ülke"); //0 yerine ad gelecek sonra 10 birim boşluk bırakacak, 1 yerine soyad.
            Console.WriteLine("--------------------------");

            while (dr.Read())
            {
                Console.WriteLine("{0,-10}{1,-10}{2,-10}", dr["firstname"], dr[1], dr["country"]); // index sırasına göre istediğimiz sütunu çağırabiliriz. Yani dr[1] yerine dr["lastname"] yazabiliriz.
            }

            Console.ReadKey();
            con.Close();
        }
    }
```

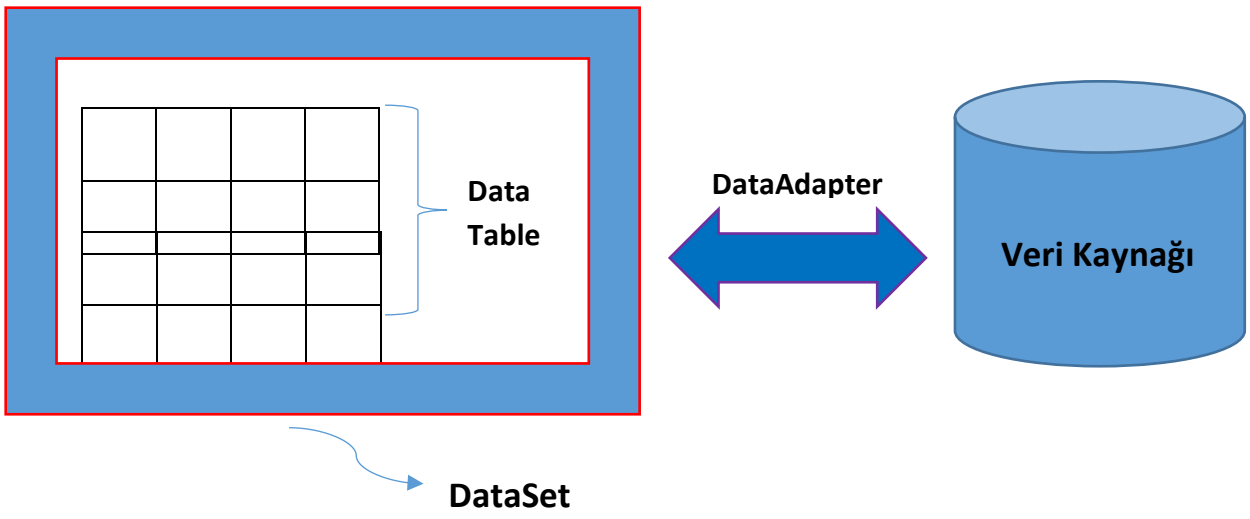
DataSet ve DataTable

Veri kaynağından bağımsız olarak veriler üzerinde işlem yapmamızı ve gerektiğinde DataAdapter yardımıyla DataSet üzerindeki verileri tekrar veri kaynağına aktarmamızı sağlar.

Bir DataSet ile, veri kaynağından bağlantı kesilse bile DataSet içindeki verilerle çalışabiliriz(Disconnected mimari). DataSet bir veri kaynağının şablonu gibidir. Bir DataSet içinde bir ya da daha fazla tablo olabilir. Bu tablolar System.Data isim alanındaki DataTable sınıfı ile temsil edilir. Tabloların yapısında ise kayıtları ve sütunları temsil eden DataRow ve DataColumn sınıfları mevcuttur.

DataSet nesnelerine veri kaynağından bilgi alıp doldurmak için DataAdapter sınıfı kullanılır.

Bahsedilen bu sınıfların birbirleriyle olan ilişkisi aşağıdaki şekilde gösterilmiştir ;



Yukarıdaki şekilde de görüldüğü üzere veri kaynağı ile DataSet nesneleri arasındaki bağlantı DataAdapter nesnesi ile olmaktadır.

DataTable sınıfı bir veri kaynağındaki tablolara benzer. Yani bir DataTable nesnesinde sütunlar ve verileri içeren kayıt satırları bulunmaktadır. Bir DataSet nesnesi içinde birden fazla DataTable olabileceği için farklı DataTable nesneleri arasında çeşitli ilişkiler kurulabilir. DataTable nesnelerinin Columns özelliği ilgili tablodaki sütunları bir DataColumn koleksiyonu şeklinde verir. Aynı şekilde Rows özelliği de tablodaki satırları bir DataRow koleksiyonu şeklinde verir.

DataColumn sınıfı, bir tablodaki sütunları temsil eder. DataColumn ile temsil edilen bir sütunun veri türü C#'taki herhangi bir temel veri türü olabilir. DataColumn nesneleri ile bir sütunun çeşitli özellikleri değiştirilebilir.

DataRow nesnesi ise bir tablodaki satırları temsil eder.

Örnek : DataAdapter nesnesi ile DataSet'i dolduralım.

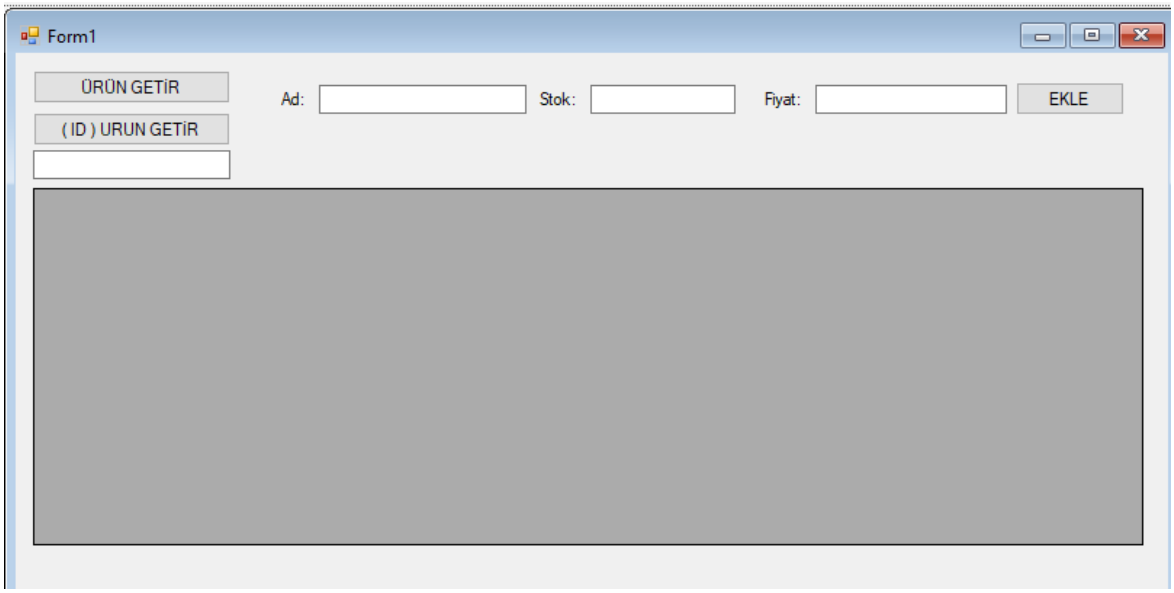
```
class Program
  {
    static void Main(string[] args)
    {
      //bağlantımızı oluşturuyoruz.
      string kaynak = "Data Source=.;Initial Catalog=Northwind;Integrated Security=SSPI;";
      SqlConnection con = new SqlConnection(kaynak);
      con.Open();

      //sorgumuzu yazıp dataAdapter üzerineden dataset e yollayacagız.
      string sorgu = "select * from employees";
```

```
        SqlDataAdapter adapter = new SqlDataAdapter(sorgu,con);

        //datasetimizi oluşturup, bilgileri üzerine basıyoruz DataAdapter'in fill metoduyla.
        DataSet ds = new DataSet();

        adapter.Fill(ds,"Erman");

        //bağlantıyı kapatıyoruz, artık dataset üzerine aldık verileri
        con.Close();

        //dataset'in üzerinde dönüyoruz bütün verilerimizi almak için,bunun için.
        foreach (DataRow satir in ds.Tables["Erman"].Rows)
        {
           Console.WriteLine(satir[2].ToString() + " " + satir[1].ToString());
        }
        Console.ReadKey();
    }
  }
```

Örnek 2: DataReader kullanarak databaseden veri alalım, Windows form örneği olsun, 1'er adet button ve listbox ekleyelim formumuza.

```
private void button1_Click(object sender, EventArgs e)
    {
        string kaynak = "Data Source=.;Initial Catalog=Northwind; Integrated Security=SSPI;";

        SqlConnection con = new SqlConnection(kaynak);


        string sorgu = "select * from employees";

        SqlCommand cmd = new SqlCommand(sorgu, con);

        con.Open();

        SqlDataReader dr = cmd.ExecuteReader();

        while (dr.Read())
        {
           listBox1.Items.Add(dr["firstname"].ToString()+" "+ dr["lastname"].ToString());
        }

        con.Close();
    }
```

Derste Yapılan Örnekler - 1 :

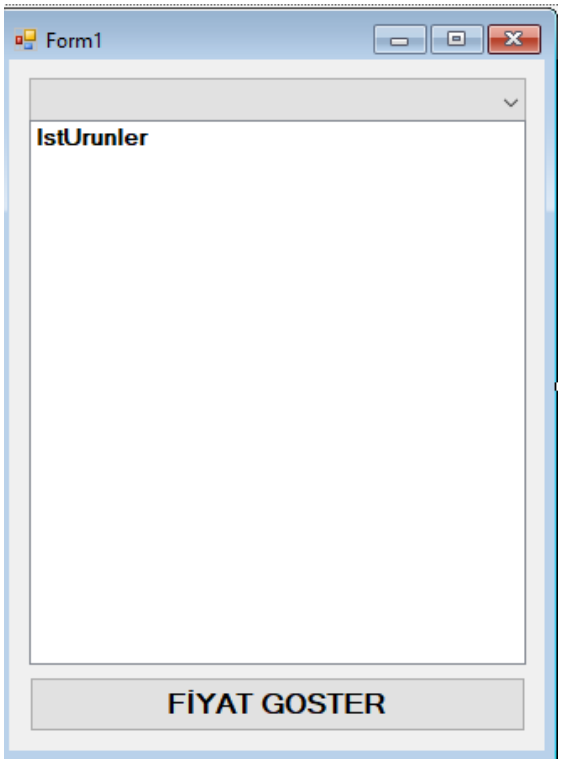



```
SqlConnection con = new SqlConnection
  ("Server=MD;Database=Northwind;Trusted_Connection=True;");
```

```
private void btnEkle_Click(object sender, EventArgs e)
{
    try
    {
        if (txtAd.Text != "")
        {
            SqlCommand cmd = new SqlCommand("Insert Into Products
              (ProductName,UnitsInStock,UnitPrice) Values(@pName,@uStock,@uPrice)", con);
            cmd.Parameters.AddWithValue("@pName", txtAd.Text);
            cmd.Parameters.AddWithValue("@uStock", txtStok.Text);
            cmd.Parameters.AddWithValue("@uPrice", txtFiyat.Text);

            SqlDataAdapter da = new SqlDataAdapter(cmd);

            DataTable dt = new DataTable();
            da.Fill(dt);
            dataGridView1.DataSource = dt;

            MessageBox.Show("KAYDETME İŞLEMİ BAŞARI İLE TAMAMLANDI");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
```

```
private void btnUrunGetir_Click(object sender, EventArgs e)
{
    SqlDataAdapter da = new SqlDataAdapter("Select * from Products", con);
    DataTable dt = new DataTable();
    da.Fill(dt);

    dataGridView1.DataSource = dt;
}

private void btnIdUrunGetir_Click(object sender, EventArgs e)
{
    if (txtId.Text != "")
    {
        SqlCommand cmd = new SqlCommand("Select * from Products where
          ProductID=@pID", con);
        cmd.Parameters.AddWithValue("@pID", txtId.Text);

        SqlDataAdapter da = new SqlDataAdapter(cmd);
        DataTable dt = new DataTable();
        da.Fill(dt);

        dataGridView1.DataSource = dt;
    }
}
```

Derste Yapılan Örnekler - 2 :



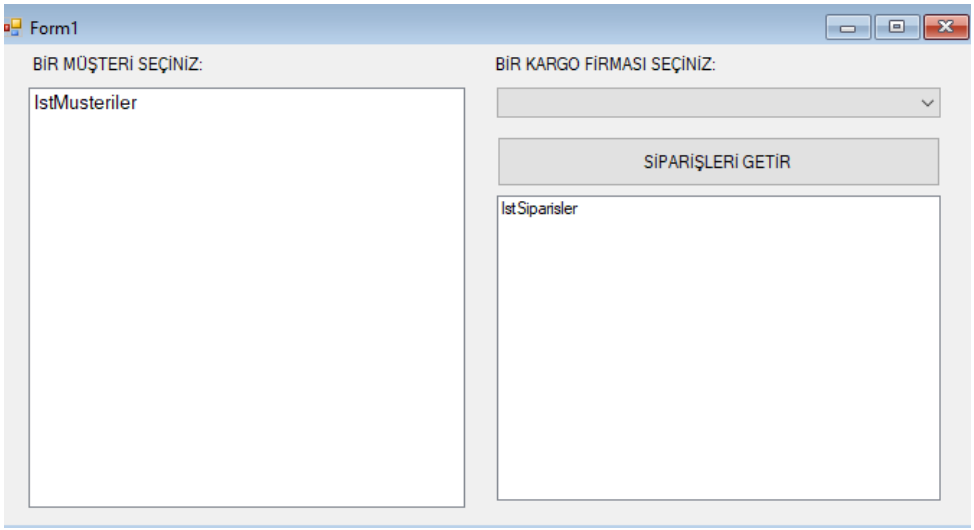
```
class Tools
{
    public static string ConnectionString
    {
        get
        {
            return "Server=MD;Database=Northwind;Trusted_Connection=True;";
        }
    }
}
```

```
private void Form1_Load(object sender, EventArgs e)
{
    SqlDataAdapter dap = new SqlDataAdapter("Select CategoryName,CategoryId
      FROM Categories", Tools.ConnectionString);
    DataTable dt = new DataTable();
    dap.Fill(dt);
    cmbKategoriler.DataSource = dt;
    cmbKategoriler.DisplayMember = "CategoryName";
    cmbKategoriler.ValueMember = "CategoryId";
}
```

```
private void cmbKategoriler_SelectedIndexChanged(object sender, EventArgs e)
{
   try
   {
        // data set
        SqlDataAdapter dap = new SqlDataAdapter("Select ProductName,UnitPrice
          from Products Where CategoryId=@id", Tools.ConnectionString);
        dap.SelectCommand.Parameters.AddWithValue("@id",
          cmbKategoriler.SelectedValue);
        DataSet ds = new DataSet();
        dap.Fill(ds);

        lstUrunler.DataSource = ds.Tables[0];
        lstUrunler.DisplayMember = "ProductName";
        lstUrunler.ValueMember = "UnitPrice";
   }
   catch (Exception ex)
   {
       //
   }
}
```

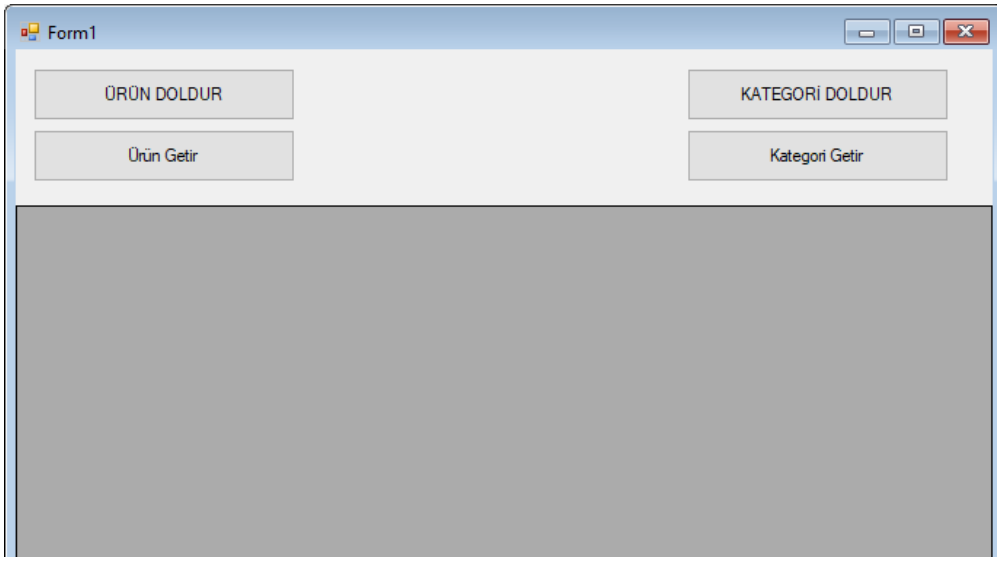
```
private void btnFiyat_Click(object sender, EventArgs e)
{
    if (lstUrunler.SelectedIndex != -1)
    {
        MessageBox.Show(lstUrunler.SelectedValue.ToString());
    }
}
```

Derste Yapılan Örnekler - 3:




```
private void Form1_Load(object sender, EventArgs e)
{
    //İKİ FARKLI SORGU VERİLDİ VE 2 FARKLI TABLO ÇEKİLDİ
    SqlDataAdapter dap = new SqlDataAdapter("Select CustomerID,CompanyName From
      Customers;Select ShipperId,CompanyName From Shippers",
      Tools.ConnectionString);
    DataSet ds = new DataSet();
    dap.Fill(ds);
    //LİSTEYE TÜM MÜŞTERİLER YÜKLENİYOR
    lstMusteriler.DataSource = ds.Tables[0];
    lstMusteriler.DisplayMember = "CompanyName";
    lstMusteriler.ValueMember = "CustomerId";

    //COMBOBOX'A TÜM KARGO ŞİRKETLERİ YÜKLENİYOR
    cmbKargocular.DataSource = ds.Tables[1];
    cmbKargocular.DisplayMember = "CompanyName";
    cmbKargocular.ValueMember = "ShipperId";
}


private void btnSiparisleriGetir_Click(object sender, EventArgs e)
{
   SqlDataAdapter dap = new SqlDataAdapter("Select CONVERT(nvarchar(10),OrderID)
     + '     ' + Convert(nvarchar(20),OrderDate,10) as 'DisplayColumn' From
     Orders Where CustomerID=@cId and Shipvia= @sid", Tools.ConnectionString);

    dap.SelectCommand.Parameters.AddWithValue("@cId",
      lstMusteriler.SelectedValue);
    dap.SelectCommand.Parameters.AddWithValue("@sid",
      cmbKargocular.SelectedValue);
    DataTable dt = new DataTable();
    dap.Fill(dt);

    lstSiparisler.DataSource = dt;
    lstSiparisler.DisplayMember = "DisplayColumn";
}
```

Derste Yapılan Örnekler - 4:




```
SqlConnection baglanti = new SqlConnection
  ("Server=MD;Database=Northwind;Trusted_Connection=True;");
DataSet ds = new DataSet();
DataTable dt;
SqlDataAdapter adp;
private void btnKategoriDoldur_Click(object sender, EventArgs e)
{
    adp = new SqlDataAdapter("select * from Categories", baglanti);
    dt = new DataTable("Kategoriler");
    adp.Fill(dt);
    ds.Tables.Add(dt);
    MessageBox.Show("Alınan veriler, kategoriler tablosuna eklendi");
}
private void btnKategori_Click(object sender, EventArgs e)
{
    dataGridView1.DataSource = ds.Tables["Kategoriler"];
}
private void btnUrunDoldur_Click(object sender, EventArgs e)
{
    adp = new SqlDataAdapter("Select * from Products", baglanti);
    dt = new DataTable("Urunler");
    adp.Fill(dt);
    ds.Tables.Add(dt);
    MessageBox.Show("Alınan Veriler, Urunler Tablosuna Eklendi");
}

private void btnUrun_Click(object sender, EventArgs e)
{
    dataGridView1.DataSource = ds.Tables["Urunler"];
}
```