# ASP.NET Veri Erişim İşlemleri

**İki tip veri erişim şekli vardır:**

**1.** Direct Data Access
**2.** Disconnected Data Access

## 1.Direct Data Access

Kodlarla database'e bağlanıp mümkün olan en kısa sürede bağlantıyı sonlandırma işlemine direct data access denir. Bir **DataSet** nesnesine database'den çekilen verilerin aktarılmasından hemen sonra bağlantının sonlandırılması işlemine ise Disconnected Data Access denir. Direct Data Access yöntemi memory'de database'den çekilen verilerin tutulmasına gerek duymadığı için daha elverişli bir yöntemdir. Bazı durumlarda verilerin memory'e yüklenmesi mantıklı olur. Mesela, product catolog'u veritabanından alırız, bunu bir **DataSet** içinde tutarız o anda bir başka kullanıcı product catalog'unu görmek istediğinde database'e bağlanmaya ihtiyaç duymadan bunu sağlayabiliriz. Sonuç olarak, yerine göre direct data access, yerine göre disconnected data access kullanılmalıdır.

### Direct Data Access Steps (SqlServer)

**1. SqlConnection, SqlCommand, SqlDataReader** nesneleri yaratılır. **SqlConnection** ve **SqlCommand** nesneleri database'e bağlanmayı ; **SqlDataReader** nesnesi ise bağlantı kurulduktan sonra veri okumayı sağlar. Eğer database'e bilgi eklemek veya var olan bir bilgiyi değiştirmek istiyorsak **SqlDataReader** nesnesi yaratılmaz, çünkü veri okunmuyor veri kayıt işlemi yapılıyor.

**2.** Bağlantı kapatılır.

### Creating a Connection

Genelde connection sayısı sınırlıdır. Bu sınır aşıldığında yeni bir connection sağlanamaz. Bu nedenle mümkün olduğu kadar connection sayısını az tutmak gerekir. Düşünsenize bir sayfada bir tane connection açılmış olsa ve o sayfaya aynı anda yüzbin kişi ulaşmak istese yüzbin tane connection açılmış olacaktır!
Connection kodlarını **try catch** bloğuna yazmak gerekir ; çünkü bir hata olduğunda kullanıcıya bilgi vermemiz gerekir. Ayrıca connection'un kapandığından da emin olmalıyız.
Connection nesnesi yaratılırken **connectionString** property'e database türüne göre bir **string** değer atamalıyız. Bu **string** bilgisayarın data kaynağını bulabilmesi için ihtiyacı olan tüm bilgileri içerir.

```
SqlConnection myConnection=new SqlConnection();
myConnection.ConnectionString="Data Source=localhost ; Initial Catalog= Pubs ; Integrated Security=SSPI";
```

### ConnectionString Property

**ConnectionString** birbirinden bağımsız bilgilerin noktalı virgülle birleşmesiyle oluşur.

**DataSource** : Data **Source(SqlServer)'un** yüklü olduğu server'ın ismini gösterir. localhost olunca database ile server aynı bilgisayardadır anlamına gelir.

**Initial Catolog** : Data Source'taki database'in ismini ifade eder. Yukarıdaki örneğimizde biz Pubs isimli database' e bağlanıyoruz.

**Integrated** Security : **SQL** Server'a kullanıcı hesabıyla bağlanmak istediğimizi ifade eder. Ayrıca **SQL** Server'a **kullanıcıID** ve password'le de bağlanabiliriz. Lakin **kullanıcıID** ve password'le data source'a bağlanmak web tasarımlarında pek uygulanmaz.

**ConnectionTimeOut** : Data source'a bağlantı kurulamadığında hata üretmeden önce kodlarımızın saniye cinsinden ne kadar süre bekletileceğini belirler. Kullanılmadığı zaman **default** değer **15** saniyedir.

**Not:** Web uygulamasında **App_Data** isimli bir klasör vardır. Bu klasöre **.mdf** database dosyaları kayıt edilir.

### Storing the Connection String

Tipik olarak, uygulamamızda tüm database bağlantıları aynı connection string'i kullanacaklardır. Bu nedenden dolayı connection string'lerimizi web.config dosyasına kaydedip buradan çağırabiliriz.

```
<configuration>
    <connectionStrings>
        <add name="ApplicationServices" connectionString="data source=.\SQLEXPRESS;
               Integrated Security=SSPI; AttachDBFilename=|DataDirectory|\aspnetdb.mdf;User Instance=true"
                  providerName="System.Data.SqlClient"/>
        <add name="DilekceConnectionString" connectionString="Data Source=localhost;Initial
               Catalog=Dilekce;Integrated Security=True" providerName="System.Data.SqlClient"/>
        <add name="dbSchool" connectionString="Data Source=localhost; Initial Catalog=schoolinformationsystem;
               IntegratedSecurity=True;AttachDBFileName=|DataDirectory|\schoolinformationsystem_log.ldf;"/>
        </connectionStrings>
</configuration>
```

<add> elementi ile connection string'ler eklenir. Şimdi eklediğimiz bu string'lerden birini **Page_Load event** handler'de çağıralım:

```
using System.Web.Configuration;
protected void Page_Load(object sender, EventArgs e)
{
    string connectionString = WebConfigurationManager.ConnectionStrings["dbSchool "].ConnectionString;
}
```

Görüldüğü gibi **WebConfigurationManager** class'ının çalışabilmesi **System.Web.Configuration** namespace'i **import** edilmiştir.

**Not:** Visual Studio'da **Server Explorer** kullanılarak database eklendiğinde otomatik olarak **web.config** dosyasına **connectionString** kaydedilir. Connection nesnesi yaratıldıktan sonra bağlantının kurulabilmesi için **Open()** metodu çağrılır.

**Örnek:**

```
using System.Data.SqlClient; //SqlConnection class'i için
using System.Web.Configuration; //WebConfigurationManager
protected void Page_Load(object sender, EventArgs e)
{
    string connectionString = WebConfigurationManager.ConnectionStrings["Dilekce"].ConnectionString;
    SqlConnection myConnection = new SqlConnection();
    myConnection.ConnectionString = connectionString;
    try
    {
        myConnection.Open();
        myConnection.Close();
    }
    catch (Exception er)
    {
        throw;//Hata meydana geldiğinde Page_Load metodunu çağıran metoda sorun gönderilir.
    }
}
```

**Select İşlemi**

**SqlCommand'ın** iki property'si **"select"** işlemlerini yapmak için kullanılır :

**1. Connection** : Açılan connection'un nesnesini atarız.
**2. CommandText** : Query string'i atarız.

```
SqlCommand myCommand = new SqlCommand();
myCommand.Connection=myConnection;
myCommand.CommandText="Select * from tblDilekce";
```

**DataReader:**

Connection açılır, **DataReader** ile veriler okunur, connection kapatılır.

```
SqlDataReader reader;
reader = myCommand.ExecuteReader();//myCommand.ExecuteReader() metodu SqlDataReader nesnesi //dönderir.
while (reader.Read())// Read metodu her çağrıldığında bir row/satır okur. Okunacak satir kalmayınca false dönderir.
//while döngüsü içerisinde kullanırsak tablo'daki tüm row'ları okuruz.
{
    string konu = (string)reader["Konu"];//reader.Read() satır bilgisi dönderir. "Konu" sütun ismidir. yani
    //"Konu" sütunundaki bilgiyi alır. reader["Konu"] bir object dönderir. bu object'e string'e dönüştürülmüş.
}
```

Buraya kadar kullanılan tüm kodları birleştirelim:

```
protected void Page_Load(object sender, EventArgs e)
{
    string connectionString = WebConfigurationManager.ConnectionStrings["DilekceConnectionString"].ConnectionString;
    SqlConnection myConnection = new SqlConnection();
    myConnection.ConnectionString = connectionString;
    SqlCommand myCommand = new SqlCommand();
    SqlDataReader myReader;
    try
    {
        myConnection.Open();
        myCommand.Connection = myConnection;
        myCommand.CommandText = "Select * from tblDilekce";
        myReader = myCommand.ExecuteReader();
        while (myReader.Read())
        {
            string konu = (string)myReader["Konu"];
        }
        myReader.Close();
        myConnection.Close();
    }
    catch (Exception er)
    {
        throw;//Hata meydana geldiğinde Page_Load metodunu çağıran metoda sorun gönderilir.
    }
}
```

**Insert,Update,Delete İşlemleri**

Veri okumayacağımız için **DataReader** nesnesi yaratmamıza gerek yoktur. insert,update,delete işlemlerini **SqlCommand** nesnesinin **executeNonQuery()** metodu sağlar. Bu metod etkilenen row sayısını dönderir. Eğer etkilenen row sayısı sıfır ise işlem başarısız olmuştur anlamına gelir; çünkü hiçbir satır değişmemiştir.

```
protected void cmdInsert_Click(Object sender, EventArgs e)
{
    string insertSQL;
    insertSQL = "INSERT INTO Authors (";
    insertSQL += "au_id, au_fname, au_lname, ";
    insertSQL += "phone, address, city, state, zip, contract) ";
    insertSQL += "VALUES (";
```

```
    insertSQL += "@au_id, @au_fname, @au_lname, ";
    insertSQL += "@phone, @address, @city, @state, @zip, @contract)";
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand(insertSQL, con);
    // Add the parameters.
    cmd.Parameters.AddWithValue("@au_id", txtID.Text);
    cmd.Parameters.AddWithValue("@au_fname", txtFirstName.Text);
    cmd.Parameters.AddWithValue("@au_lname", txtLastName.Text);
    cmd.Parameters.AddWithValue("@phone", txtPhone.Text);
    cmd.Parameters.AddWithValue("@address", txtAddress.Text);
    cmd.Parameters.AddWithValue("@city", txtCity.Text);
    cmd.Parameters.AddWithValue("@state", txtState.Text);
    cmd.Parameters.AddWithValue("@zip", txtZip.Text);
    cmd.Parameters.AddWithValue("@contract", Convert.ToInt16(chkContract.Checked));
    // Try to open the database and execute the update.
    int added = 0;
    try
    {
        con.Open();
        added = cmd.ExecuteNonQuery();
        lblStatus.Text = added.ToString() + " record inserted.";
    }
    catch (Exception err)
    {
        lblStatus.Text = "Error inserting record. ";
        lblStatus.Text += err.Message;
    }
    finally
    {
        con.Close();
    }
}
protected void cmdUpdate_Click(Object sender, EventArgs e)
{
    // Define ADO.NET objects.
    string updateSQL;
    updateSQL = "UPDATE Authors SET ";
    updateSQL += "au_fname=@au_fname, au_lname=@au_lname, ";
    updateSQL += "phone=@phone, address=@address, city=@city, state=@state, ";
    updateSQL += "zip=@zip, contract=@contract ";
    updateSQL += "WHERE au_id=@au_id_original";
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand(updateSQL, con);
    // Add the parameters.
    cmd.Parameters.AddWithValue("@au_fname", txtFirstName.Text);
    cmd.Parameters.AddWithValue("@au_lname", txtLastName.Text);
    cmd.Parameters.AddWithValue("@phone", txtPhone.Text);
    cmd.Parameters.AddWithValue("@address", txtAddress.Text);
    cmd.Parameters.AddWithValue("@city", txtCity.Text);
    cmd.Parameters.AddWithValue("@state", txtState.Text);
    cmd.Parameters.AddWithValue("@zip", txtZip.Text);
    cmd.Parameters.AddWithValue("@contract",
    Convert.ToInt16(chkContract.Checked));
    cmd.Parameters.AddWithValue("@au_id_original",
    lstAuthor.SelectedItem.Value);
    // Try to open database and execute the update.
    int updated = 0;
    try
    {
        con.Open();
        updated = cmd.ExecuteNonQuery();
        lblStatus.Text = updated.ToString() + " record updated.";
    }
    catch (Exception err)
    {
        lblStatus.Text = "Error updating author. ";
        lblStatus.Text += err.Message;
    }
    finally
    {
        con.Close();
    }
}
```

## 2.Disconnect Data Access

**DataSet** nesnesi kullanılarak yapılır.

**Kullanımı şöyledir:** Database'e bağlanılır veriler getirilir, **DataSet** nesnesine kaydedilir, database bağlantısı kapatılır.

**Kullanım nedenleri :**

**1.** Direct Data Access'te veriler üzerinde sadece ileri hareket edebiliyorduk. Örneğin **Read()** metod ile satır satır okuma yaparken önceki okunan satıra geri dönemiyorduk. Disconnect Data Access'te ise veriler üzerinde ileri geri gidebiliriz.

**2.** Daha sonra kullanılmak amacıyla verileri bir dosyaya kaydetmek için kullanılabilir. **DataSet** nesnesi'ne kaydedilen verileri **XML** formatına dönüştürebiliriz.

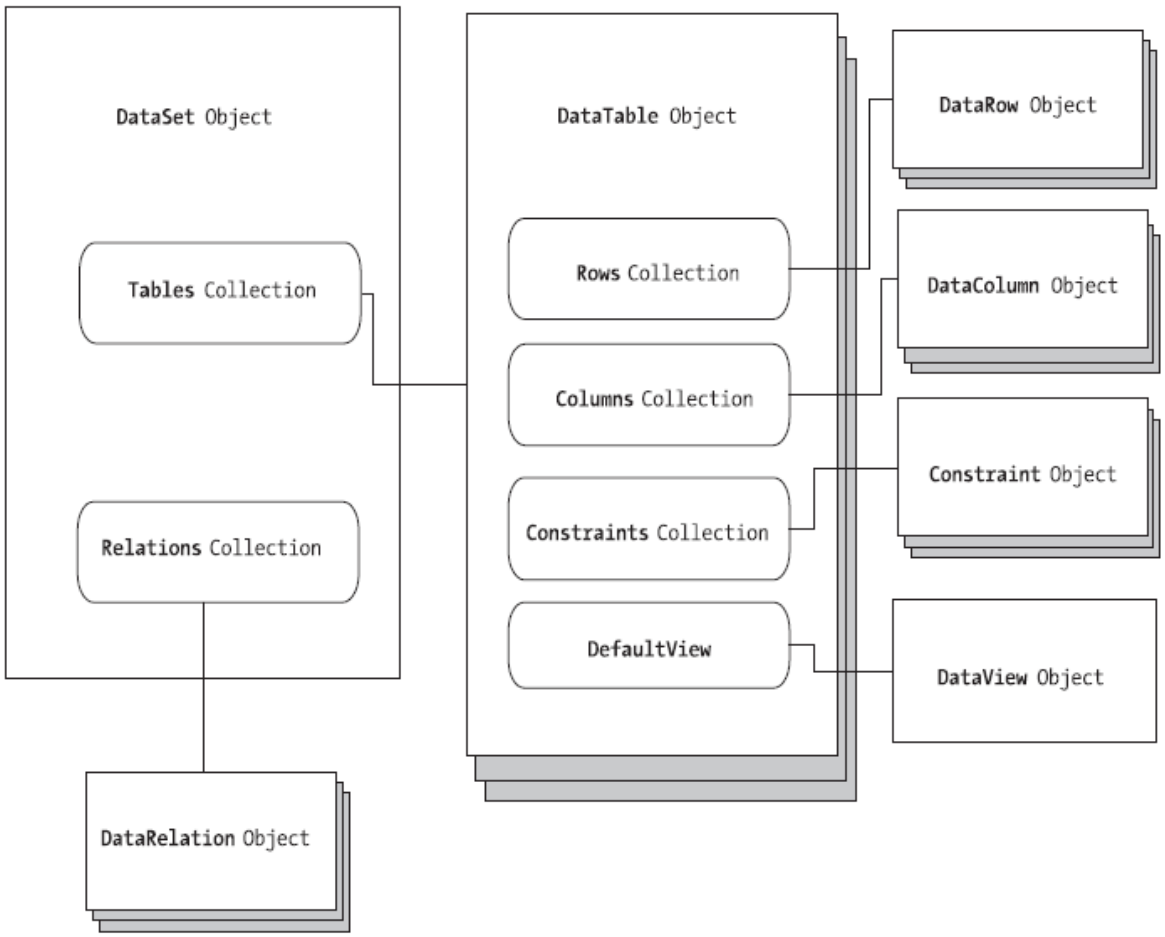


**Figure 15-16.** *The DataSet family of objects*

**DataSet'e Kayıt İşleminin Yapılışı :**

Hatırlarsak **SQL** Server için dört tane data provider sınıf vardı:

1. **SqlConnection** : Bağlantı açıp kapamaktan sorumlu
2. **SqlCommand** : Query'nin execute edilmesinden sorumlu
3. **SqlDataReader** : Bağlantı yapıldıktan sonra verilerin database'den okunmasından sorumlu
4. **SqlDataAdapter** : **DataSet'e** database'den kendi okuduğu verilerin kaydedilmesinden sorumlu. **SqlDataAdapter SqlDataReader'ın** yaptığı işi de yapıyor.

**Örnek:**

```
protected void Page_Load(object sender, EventArgs e)
{
    string connectionString = WebConfigurationManager.ConnectionStrings["DilekceConnectionString"].ConnectionString;
    SqlConnection myConnection = new SqlConnection();
    SqlCommand myCommand = new SqlCommand();
    SqlDataAdapter myAdapter = new SqlDataAdapter();
    DataSet myDataSet = new DataSet();
    myConnection.ConnectionString = connectionString;
    myCommand.Connection = myConnection;
    myCommand.CommandText = "Select * from tblDilekce";
    myAdapter.SelectCommand = myCommand;
    try
    {
        myConnection.Open();
        myAdapter.Fill(myDataSet,"tblDilekce");
        myConnection.Close();
    }
    catch (Exception er)
    {
        throw;//Hata meydana geldiğinde Page_Load metodunu çağıran metoda sorun gönderilir.
    }
    //Database'e bağlanmadan DataSet içindeki satırları okuyoruz.
    foreach (DataRow row in myDataSet.Tables["tblDilekce"].Rows)
    {
        ListItem newItem = new ListItem();
        newItem.Text = row["Konu"] + ", ";
    }
}
```