

EĞİTİM :

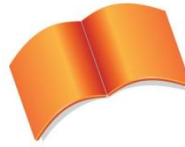
**VERİYE ERİŞİM VE
NAVİGASYON**

Bölüm :

Veriye Erişim

Konu :

Veriye Erişim



Microsoft Türkiye

Açık Akademi

Bildiğiniz gibi, geçmişte web siteleri tamamen durağan yapılardan oluşmaktaydı, zamanla bu durağanlık kullanıcılara yetmemeye başladı ve yeni bir ihtiyaç hissedildi. Sadece HTML'den oluşan durağan (static) siteler yerini yavaş yavaş Javascript ile desteklenen sitelere bırakmaya başladı ancak bu da kullanıcılara yetmedi. Javascript kullanılarak kullanıcı ile etkileşime giren siteler geliştirilmesi mümkündü ama kullanıcılar hala veriye erişemiyorlardı. Kullanıcılar geliştirilen siteler üzerinden online olarak veriye erişip, veri üzerinde işlem yapmak istiyorlardı bu sebeple ASP ve PHP gibi verinin yönetilebildiği web teknolojileri oldukça üne kavuştu ve halen bu iki teknolojinin kullanımına devam edilmektedir.

Microsoft .NET Platformu piyasaya sürüldükten sonra pek çok teknolojiye olduğu gibi veriye erişim teknolojilerinde de birçok yenilik kullanıcıları karşıladı. Bu yeniliklerin en büyüğü; tamamen XML'i baz alan ADO.NET teknolojisidir. ADO.NET ile birlikte XML'e kısmen destek vermek yerine teknoloji tamamen XML mimarisi üzerine kurulmuş ve en alt düzeyde bile XML'e destek verilmiştir. ADO.NET yazılım geliştiricilere veriye erişimde pek çok kolaylıklar sunmaktadır, bu sebeple ADO.NET yazılım geliştiricilerin veriye erişme tercihi haline gelmiştir.

ASP.NET ile uygulama geliştirilirken de veriye erişimde kullanılacak olan teknoloji ADO.NET'tir, dolayısıyla ADO.NET ile veriye erişmeyi bilen bir yazılım geliştirici aynı kodları ASP.NET ile yazılım geliştirirken de kullanabilmektedir. Microsoft .NET teknolojilerinin birbiri ile bu kadar entegre çalışabiliyor olması her teknolojinin ayrı ayrı kullanımını dolayısıyla Microsoft .NET'in kullanımını oldukça yaygın hale getirmiştir. ADO.NET teknolojisi kullanılarak ASP.NET üzerinden veriye erişilebilmesi ve ASP.NET'in veri kontrolleri, ASP.NET'i web programcılığının bir numaralı teknolojisi haline getirmiştir ve günümüzde halen ASP.NET kullanımı artmaya devam etmektedir.

Veri Kaynağı Kontrolleri (Data Sources)

Veri kaynağı kontrolleri veritabanı ile uygulama arasında veri taşıyan kontrollerdir. Yazılım geliştiricilerin hayatına .NET 2.0 ile birlikte giren bu kontroller, veri işlemlerinde oldukça kolaylıklar sağlamaktadır. Bağlı olan veri tabanı yönetim sistemine göre değişiklik gösteren bu kontroller ile birlikte kod yazmadan kolaylıkla INSERT, UPDATE, DELETE ve SELECT sorguları oluşturulabilmektedir.

Veri kaynağı kontrolleri, veri kontrolleri ile birlikte bütünleşik olarak çalışacak şekilde tasarlanmıştır. İlerleyen ekranlarda da örneklerden görülebileceği gibi bir SqlDataSource kontrolü FormView kontrolüne bağlanarak hiç kod yazmadan, FormView kontrolünün tüm şablonları oluşturulup şablonların içerisine de TextBox gibi kontroller ilave edilerek veri tabanına veri girilmesi sağlanabilmektedir.

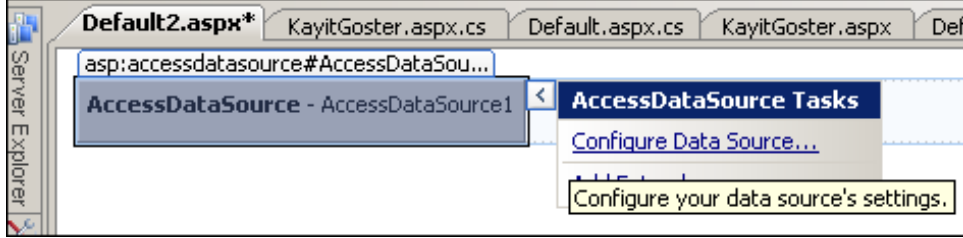


Veri kaynağı ve Veri Kontrolleri Visual Studio içerisinde yer alan ToolBox'ta, Data tabında yer almaktadır.

AccessDataSource

Veritabanı yönetim sistemi olarak Microsoft Access tercih edildiyse veritabanında bulunan veriler ile işlem yapmak için AccessDataSource kontrolü kullanılabilir. AccessDataSource kullanmak için veri tabanı dosyası App_Data klasörü içerisine koyulup buradan da AccessDataSource kontrolü aracılığı ile seçilerek uygulama içerisinde kolaylıkla kullanılabilir. Access dosyasını App_Data içerisine koymak mecburi değildir ama ASP.NET 2.0 ile birlikte gelen App_Data klasörü, veri tabanı dosyalarını barındırmak için özel bir klasör olduğundan Access dosyalarını bu alanda tutmak tercih edilmez.

AccessDataSource kontrolünü kullanmak için ilk olarak bir tane Access veritabanı dosyası App_Data klasörünün içerisine koyulur bu işlemin ardından da sayfaya bir tane AccessDataSource kontrolü eklenerek aşağıdaki resimden de görüldüğü gibi AccessDataSource kontrolünün ayarlarını gerçekleştirmek için kontrolün SmartTag'inden **"Configure DataSource..."** seçeneği tıklanır. Daha sonra açılacak olan pencereden veritabanı dosyasının yerini göstermek için **"Browse"** butonuna tıklanarak veritabanı dosyası seçilir.



Yukarıdaki işlemlerin ardından AccessDataSource kontrolü kullanılacak olan veritabanı dosyasını tanıyor olacaktır. Ayarların yapıldığı bölümde Next tuşu ile bir sonraki adıma geçildiğinde veritabanı dosyasında bulunan tablolar listelenecek ve bu tablolardan kullanılacak olan tablo ve bu tablonun kullanılacak olan sütun ya da sütunları seçilebiliyor olacaktır. Bu adımdan sonraki adımlar SqlDataSource kontrolü ile hemen hemen aynı olduğu için bu başlık altında incelenmeyecektir. Tablo ve sütun seçimi hakkındaki daha geniş bilgi SqlDataSource başlığı altında yer almaktadır.

XmlDataSource

Veri kaynağı olarak bir XML dosyası kullanılacağı zaman XmlDataSource tercih edilebilir. Sayfaya bir tane XmlDataSource kontrolü ekleyip kontrolün SmartTag'inden **"Configure Data Source..."** seçeneği seçilerek açılan ekrandan kullanılacak olan **XML** dosyası, XML dosyasındaki verilerin kurallarını belirleyen **XSL** (XML Transform Files) ve XML dosyasındaki verileri filtrelemek için **Xpath** ifadesi belirlenebilir.

SiteMapDataSource

ASP.NET 2.0 ile birlikte web sitelerinde menü ve ağaç yapısı şeklinde menülerin oluşturulabileceği kontroller yazılım geliştiricilerin hayatına girerek üçüncü parti yazılım kullanma ihtiyacını azaltmıştır. Bu kontrollerin elemanları kendi özellikleri ile direkt belirtilebileceği gibi sitenin hiyerarşik yapısını tutan .sitemap uzantılı dosyalardan da sağlanabilmektedir. Yönlendirme Kontrollerinin (**Navigation Controls**) içeriği .sitemap uzantılı XML tabanlı dosyadan sağlanıyor ise bu dosyadaki verileri uygulamaya taşımak için SiteMapDataSource kontrolü kullanılmalıdır. Menü kontrolleri ve **SiteMapDataSource** hakkında daha geniş bilgiye **"Master Page ve Navigasyon"** adlı bölümden ulaşılabilirsin.

LinqDataSource

ContextDataSource sınıfından kalıtılmış olan LinqDataSource kontrolü, web geliştiricilere, farklı tiplerdeki veri kaynaklarından veri sorgulama ve güncelleme amaçlı kullanılacak ortak bir programlama modeli sunan Language-Integrated Query (LINQ)'nin nimetlerinden faydalanma imkanı veriyor. LinqDataSource kontrolü de, diğer kontrollere benzer bir şekilde, kontrolün SmartTag'inden **"Configure Data Source..."** seçeneği seçilerek ayarlanabilir.

EntityDataSource

Web uygulamaları ile ADO.NET Entity Framework arasında veri bağlamak için kullanılacak olan EntityDataSource kontrolü, .NET Framework 3.5 Servis Paketi 1'den itibaren Framework içerisinde yer almaya başladı. SqlDataSource, LinqDataSource, XmlDataSource, ve ObjectDataSource kontrolleri ile benzer bir kullanıma sahip olan EntityDataSource, yine aynı şekilde SmartTag'inden **"Configure Data Source..."** seçeneği seçilerek ayarlanabilir.

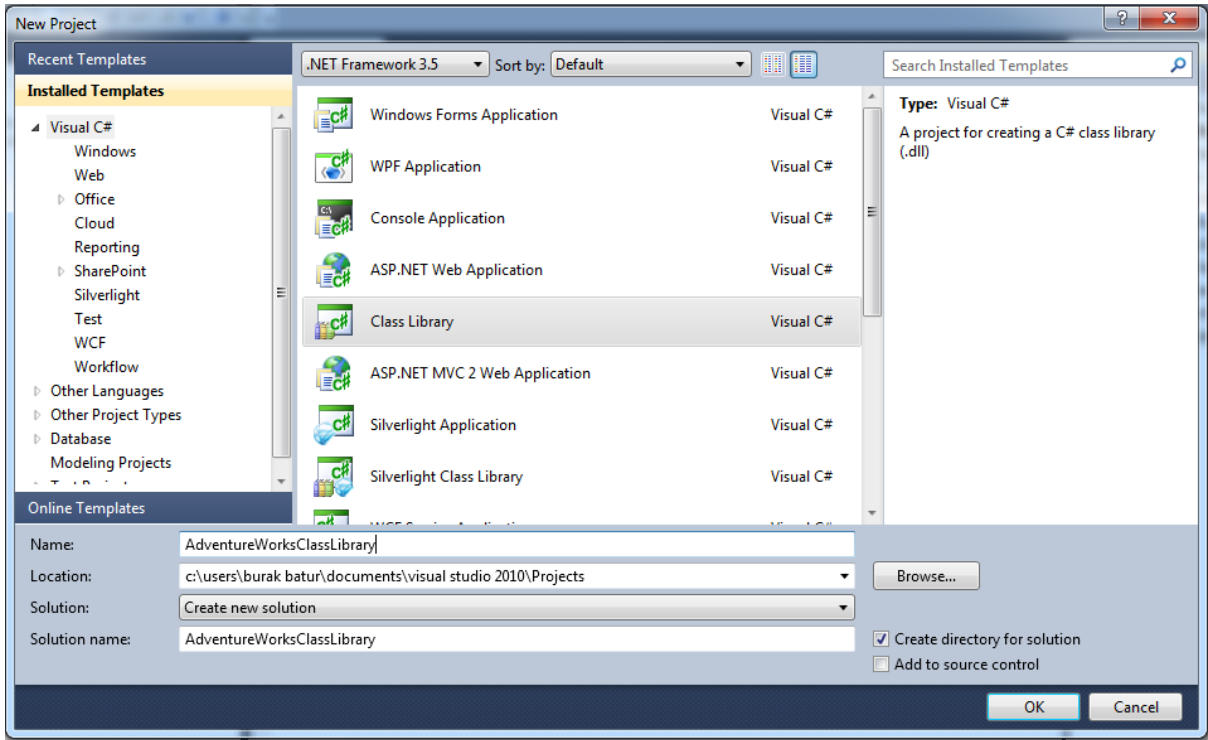
ObjectDataSource

Büyük projelerde genellikle sunum katmanından direkt veriye erişilmez, veriye erişmek ayrı bir iş katmanı yazılır ve veriye geliştirilen bu iş katmanı üzerinden erişilir. Projede bir iş katmanı kullanılacağı zaman bu iş katmanı uygulama içerisinde ObjectDataSource ile kullanılabilir. ObjectDataSource kontrolü kendine belirtilen iş katmanında bulunan sınıfın nesnesini üretir ve ardından bu nesnenin metodlarını kullanarak veri işlemlerini gerçekleştirir.

ObjectDataSource'un kullanımı da diğer veri kaynağı kontrolleri gibi düşünülebilir. Kullanılacak olan sayfaya bir tane ObjectDataSource kontrolü sürüklenip bırakıldıktan sonra ardından kontrolün SmartTag'inden "Configure Data Source..." linkine tıklanılarak ayarların yapılacağı olduğu Sihirbaz açılır ve yine kod yazmadan tüm veri işlemleri gerçekleştirilebilir. ObjectDataSource kontrolünün kullanımının örneklenmesi için bir tane iş katmanına ihtiyaç vardır. Şimdi birlikte bu iş katmanını tasarlayalım;

ADO.NET Bilgisi İle İş Katmanı

Veriye erişim bölümünde hem kod yazarak hem de kod yazmadan Visual Studio'nun araçları ile veriye nasıl erişileceğini öğreneceğiz. İlk olarak ADO.NET bilgimizi de kullanarak bir ClassLibrary projesi oluşturup yazdığımız kodlar aracılığı ile veriye erişeceğiz. Bu işlem için AdventureWorks veritabanındaki Production şeması altında yer alan ProductCategory tablosundaki temel veri işlemlerini gerçekleştirecek bir sınıf tasarlayacağız. Sınıfta veri güncelleme, veri silme, veri ekleme ve veri seçme görevlerini üstlenecek olan toplam dört adet metod yer alacak. Yazılacak olan sınıf bir **Class Library Projesi** içerisinde yer alacağı için dll haline getirilip daha sonra da farklı projelerde kullanılabilir. Visual Studio ortamında bir ClassLibrary projesi **File→New→Project** yolu ile açılan ekrandan ClassLibrary proje tipi seçilerek açılabilir. Aşağıdaki resimde, oluşturulan ClassLibrary projesinin özelliklerinin belirtildiği ekran yer almaktadır.



Açılan projeye **ProductCategory** adında bir sınıf ekleyelim. Sınıf içerisinde toplam dört adet metod olacaktır. Bunlar, verileri getirecek olan **VeriCek()**, yeni veri ekleyecek olan **VeriEkle()**, veri güncelleyecek olan **VeriGuncelle()** ve veri silme görevini üstlenecek olan **VeriSil()** isimli metodlardır. Aşağıdaki kod bloğu oluşturulan ProductCategory sınıfının kodlarını göstermektedir.

```

public class ProductCategory
{
    //Bağlantı nesnesi için değişken tanımlanıyor
    SqlConnection baglanti;

    //Komut nesnesi için değişken tanımlanıyor
    SqlCommand komut;

    public ProductCategory()
    {
        //Bağlantı nesnesi örnekleniyor
        baglanti = new SqlConnection("Data Source=.; Initial Catalog=AdventureWorks; Integrated Security=true");

        //Komut nesnesi örnekleniyor
        komut = new SqlCommand();
        komut.Connection = baglanti;
    }

    public DataTable VeriCek()
    {
        komut.CommandText = "select * from Production.ProductCategory";

        DataTable dt = new DataTable();

        SqlDataAdapter adapter = new SqlDataAdapter(komut);
        adapter.Fill(dt);

        return dt;
    }

    public void VeriEkle(string name)
    {
        komut.CommandText = "INSERT INTO Production.ProductCategory (Name) VALUES (@Name)";
        komut.Parameters.AddWithValue("@Name", name);

        baglanti.Open();
        komut.ExecuteNonQuery();
        baglanti.Close();
    }

    public void VeriGuncelle(int productCategoryID, string name)
    {
        komut.CommandText = "UPDATE Production.ProductCategory SET Name=@Name WHERE ProductCategoryID=@ProductCategoryID ";
        komut.Parameters.AddWithValue("@Name", name);
        komut.Parameters.AddWithValue("@ProductCategoryID", productCategoryID);

        baglanti.Open();
        komut.ExecuteNonQuery();
        baglanti.Close();
    }

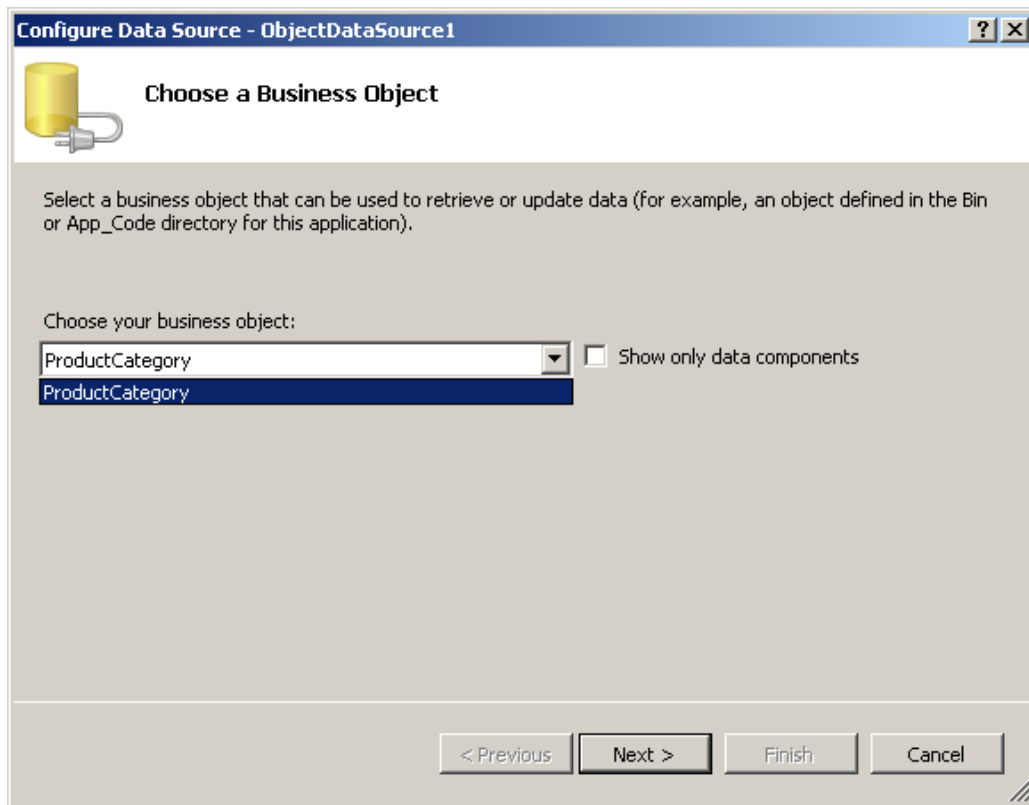
    public void VeriSil(int productCategoryID)
    {

```

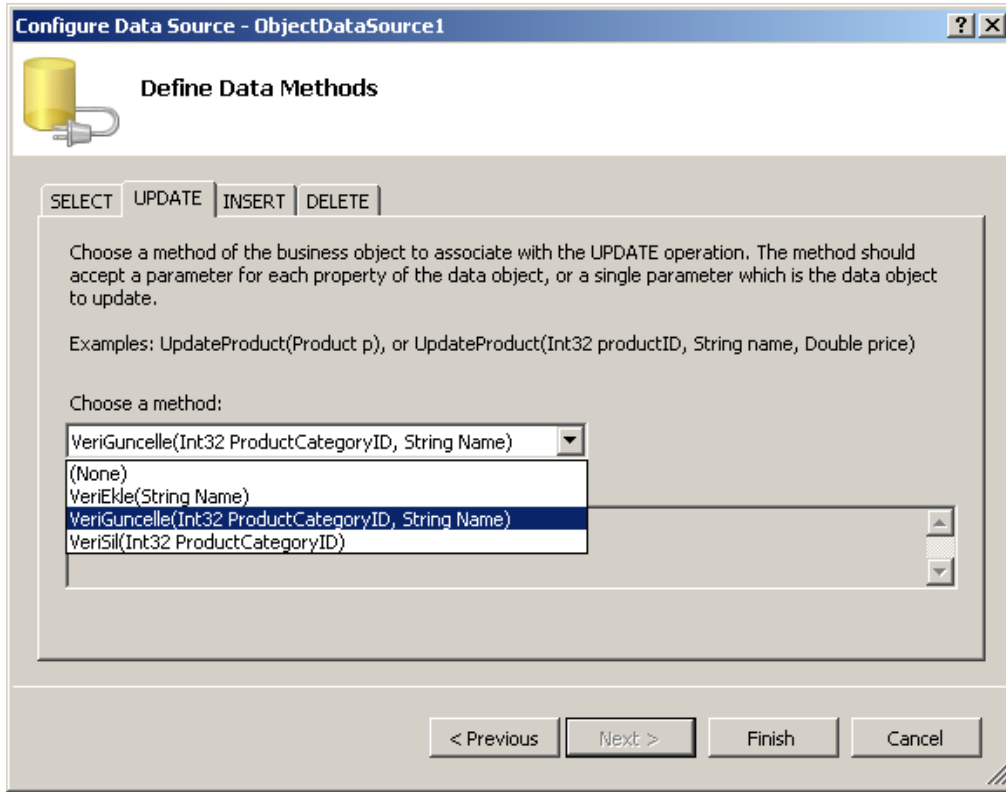
```
komut.CommandText = "DELETE From Production.ProductCategory WHERE ProductCategoryID=@ProductCategoryID ";
komut.Parameters.AddWithValue("@ProductCategoryID", productCategoryID);

baglanti.Open();
komut.ExecuteNonQuery();
baglanti.Close();
}
}
```

Tasarladığımız iş katmanı hatırlanacağı üzere bir **ClassLibrary** projesi olarak oluşturulmuştu. ClassLibrary projesi derlendiğinde bir tane dll üretecektir. Üretilen bu dll, ASP.NET projesinde **Bin** klasörü altına eklenerek kullanılabilir. Oluşturulan dll, projeye eklendikten sonra yeni bir sayfa açıp sayfaya bir tane **ObjectDataSource** kontrolü sürüklenip bırakıldıktan sonra kontrolün SmartTag'inde "Configure Data Source..." linkine tıklayarak gerekli özellikleri ayarlamaya başlayalım. Aşağıda karşılaşılan ilk ekran yer almaktadır. Bu ekrandan kullanılacak olan iş nesnesi seçilmelidir.



Yukarıdaki resimde görülen alandan istenilen sınıf seçilip **Next** ile bir sonraki adıma geçildikten sonra her işlem için seçilen sınıf içerisinde bulunan bir metot seçilecektir. Bu alanda Select kısmında veri döndüren metodlar listelenirken diğer bölümlerde de amaçlarına uygun şekilde, diğer metodlar listeleniyor olacaktır.



Yukarıda görülen ekran aracılığı ile gerekli metodlar seçildikten sonra **Finish** tuşuna basılarak gerekli ayarlamalar tamamlanabilir. Bu işlemten sonra **ObjectDataSource** kontrolü herhangi bir veri kontrolüne bağlanarak kod yazmadan burada tanımlanan işlemler gerçekleştirilebilir.



ObjectDataSource tarafından oluşturulan nesneye ObjectDataSource'un **ObjectCreated**, **ObjectCreating** ve **ObjectDisposing** olayları tetiklendiği anda ulaşılabilir ve oluşturulan nesne elde edilip üzerinde işlem yapılabilir.

EĞİTİM :

**VERİYE ERİŞİM VE
NAVİGASYON**

Bölüm :

Veriye Erişim

Konu :

SqlDataSource



Microsoft Türkiye

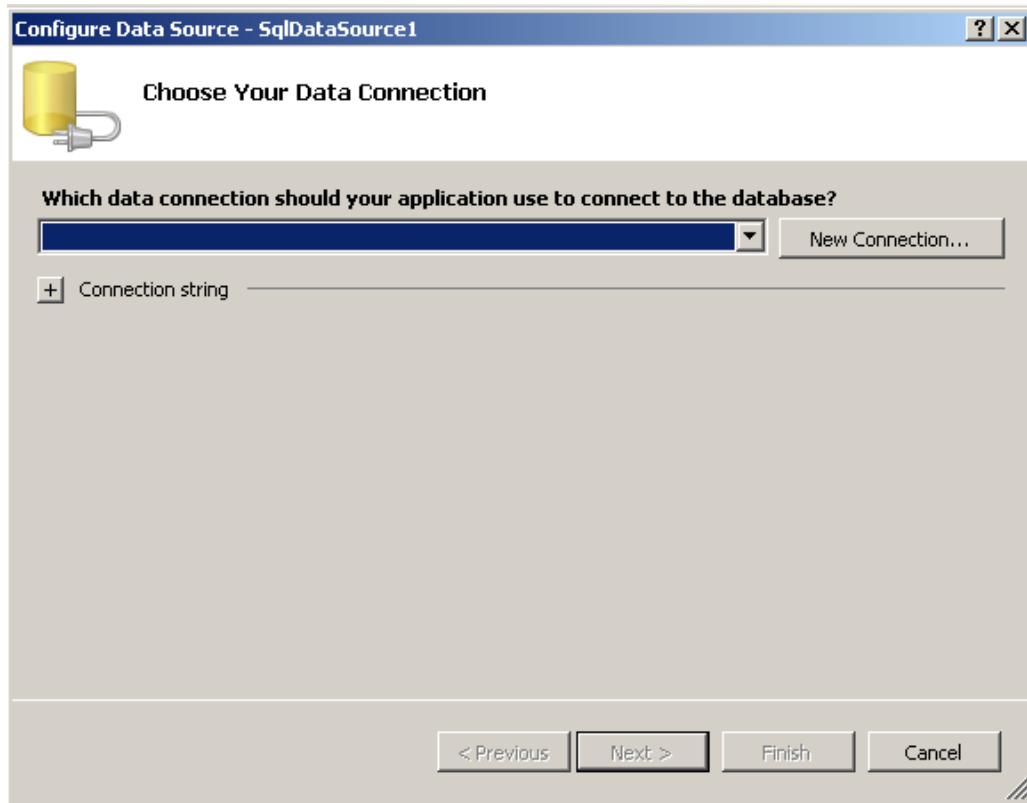
Açık Akademi

SqlDataSource

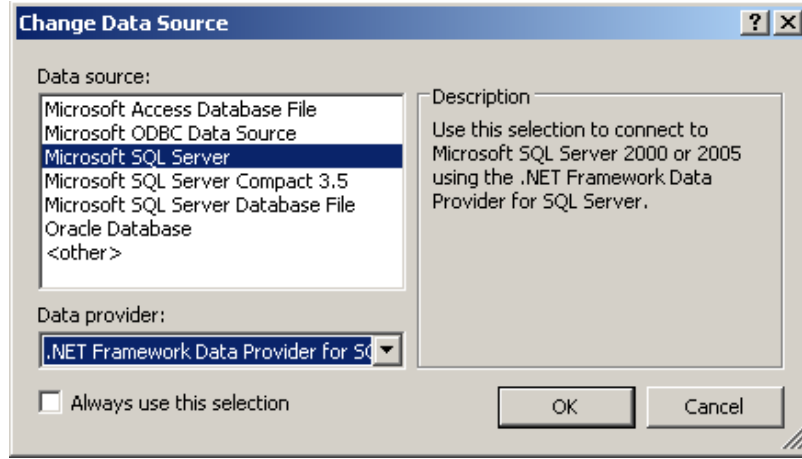
Kullanılacak olan veri tabanı yönetim sistemi SQL Server ailesinden ise veriler üzerinde çalışmak için **SqlDataSource** kullanılabilir. SqlDataSource ile birlikte tek satır kod yazmadan veritabanında bulunan talolara erişilebileceği gibi **StoredProcedure**'lere de erişip işlem yapılabilir. Kullanımından daha önce bahsedilen diğer dört veri kaynağı kontrolüne benzer olan SqlDataSource kontrolü, bu bölümde daha detaylı bir şekilde ele alınacaktır.

Diğer dört veri kaynağı kontrolüne göre kullanım amacı daha geniş ve daha zahmetsiz olan SqlDataSource kontrolü ile SQL Server üzerindeki işlemler oldukça kolay bir şekilde gerçekleştirilebilmektedir. SqlDataSource kontrolü de diğer kontroller gibi UPDATE, SELECT, DELETE ve INSERT sorgularını yazabilecek yeteneğe sahiptir hatta sorgu yazarken çakışmaları engelleyecek olan bir takım önlemleri de veri güncelleme ve silme sorgularına dahil edebilecek yetenekleri de vardır. INSERT, UPDATE ve DELETE sorguları oluşturulurken dışarıdan alınan veriler parametre aracılığı ile alındığı için uygulama SQL Injection ataklarından da korunmuş olmaktadır.

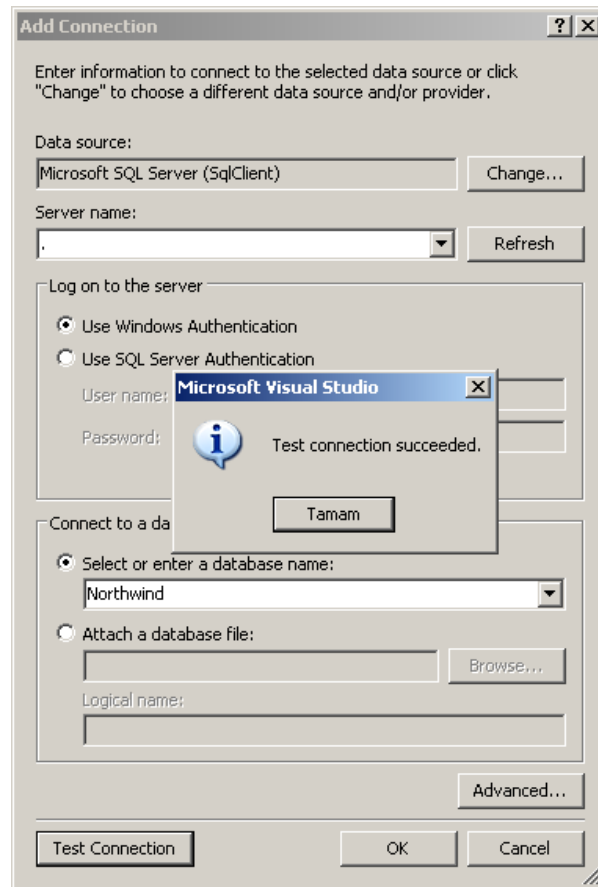
SqlDataSource kullanımını örneklemek için SQL Server ile birlikte gelen Northwind veritabanı kullanılacaktır. Sıradaki uygulamayı adım adım birlikte gerçekleştirmeden önce bilgisayarına kurulu olan SQL Server'a, **Northwind** veri tabanını eklemen gerekiyor. <http://enocetaninbelirleyecegiadres.com/dosya.rar> adresinden indirebileceğin veri tabanını, SQL eğitimlerinde Attach Database başlığı altında anlatıldığı gibi ekleyebilirsin. Yeni bir proje açıp projeye sayfa ekledikten sonra, adım adım birlikte ilerleyelim. Daha önce diğer veri kaynağı kontrolleri anlatılırken olduğu gibi, SqlDataSource kullanımında da, sayfaya bir tane SqlDataSource kontrolü sürükleyip, kontrolün SmartTag'inde Configure Data Source... linkine tıklayarak başlayalım. SqlDataSource'un ayarlarını gerçekleştirecek olan sihirbaz, aşağıdaki ekran ile başlayacaktır. Bu ekran aracılığı ile uygulamada daha önce oluşturulmuş olan bağlantı cümleleri (Connection String) listelenir ve bunlardan birinin seçilmesi sağlanabilir.



Eğer daha önceden oluşturulan bir bağlantı cümlesi yoksa ya da var olan bağlantı cümleleri bağlanılmak istenilen veritabanını işaret etmiyorsa “**New Connection**” butonu ile yeni bir bağlantı cümlesi oluşturulabilir. Geliştirilen uygulamada bağlantı cümlesi olmadığını varsayıp, **New Connection** butonuna tıklanıldığında eğer bu işlem ilk sefer yapılıyorsa, sağlayıcı seçimi için aşağıdaki ekranla karşılaşılıyor olacaktır.

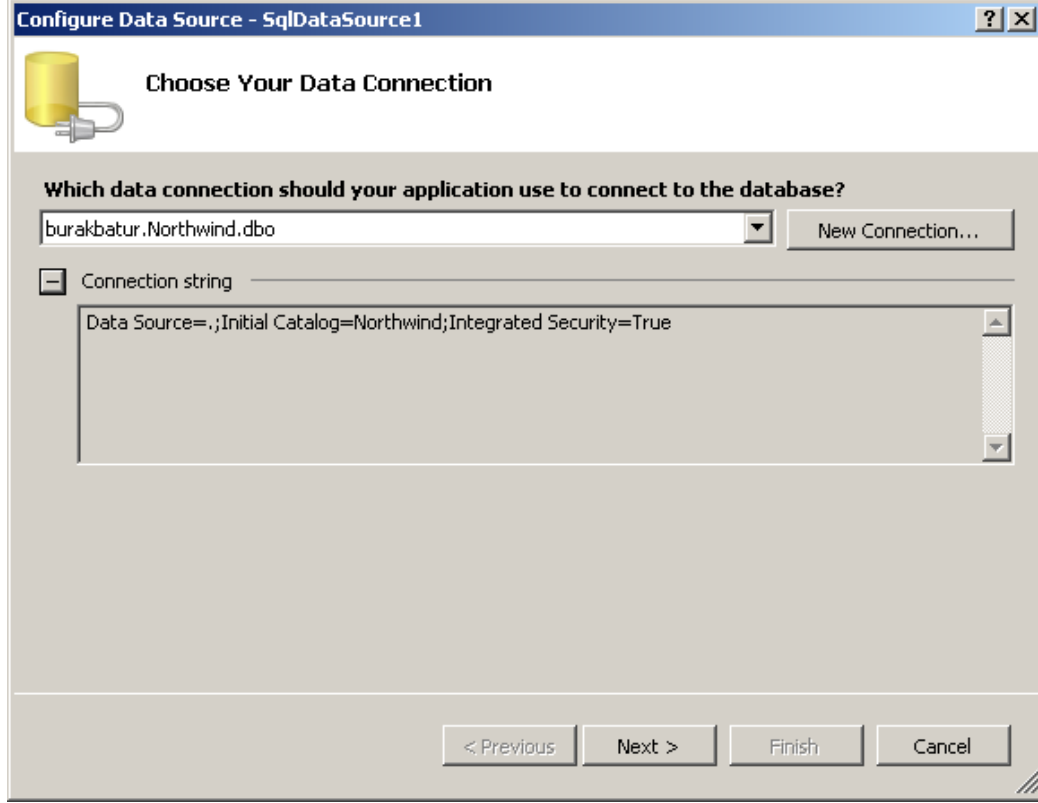


Veri kaynağı olarak “**Microsoft SQL Server**” ve sağlayıcı olarak da “**.NET Framework Data Provider for SQL Server**” seçtikten sonra, **OK** tuşu ile bu ekrandan SQL Server’ın bulunduğu makinenin, veritabanının ve bağlantı şeklinin seçilebileceği aşağıdaki ekrana geçebilirsiniz.



SQL Server’ın bulunduğu sunucu olarak yerel makine seçiliyor. SQL Server’a bağlanırken kullanılacak olan güvenlik tipi olarak da **Windows Authentication** seçiliyor. Eğer bu iki ayar da herhangi bir problem yoksa en

alttaki bölümde belirtilen veri kaynağında bulunan tüm veritabanları listeleniyor olmalıdır. Bu alandan da **Northwind** veritabanı seçilip bağlantıda bir problem olup olmadığını test etmek için Test Connection butonuna tıklanıyor. Eğer herhangi bir problem yoksa bağlantının başarılı bir şekilde kurulabildiğini belirten “**Test Connection succeeded**” mesajı görüntüleniyor olacaktır. Mesaj görüntüledikten sonra **OK** tıklanılarak aşağıdaki ekrana dönülür. Bu ekran, oluşturulan bağlantı cümlesini görüntülüyor olacaktır.



Bir önceki adımda belirtilen değerlere göre oluşturulan bağlantı cümlesi bu aşamada görüntülenmektedir. Artık **Next** butonu ile bir sonraki adıma geçilebilir. Bir sonraki adım oluşturulan bağlantı cümlesinin uygulama ayarlama dosyasına (**web.config**) kaydedilip kaydedilmeyeceğini soracaktır. “**Yes**” kutucuğu işaretlenip bir sonraki adıma geçilir.



Bağlantı cümlesi bir ayarlama dosyası yerine, veri kaynağının bulunduğu sayfaya da kaydedilebilir ancak bağlantı cümlesini ayarlama dosyasına kaydetmek tüm sayfalardan kaydedilen bağlantı cümlesine erişimi sağlayacağı için veri tabanına ihtiyaç duyulan her yerde yeni bir tane bağlantı cümlesi oluşturulmasına gerek bırakmamaktadır. Ayrıca bağlantı cümlesi ayarlama dosyasında saklanılarak yayınlama ve bakım işlemleride daha kolay bir şekilde yapılabilir.

Bir sonraki adım veritabanında bulunan tabloları listeleyecektir. Bu tablolardan istenilen herhangi bir tanesi seçilip, seçili olan tablonun da istenilen sütunları seçilebilir. Listelenen tablolardan “**Categories**” tablosu ve bu tablonunda tüm sütunlarını aşağıda görüldüğü gibi seçelim. Seçilen tablo ve sütunlara göre gekeli SQL sorgusu Visual Studio tarafından oluşturulacaktır.

Configure Data Source - SqlDataSource1

Configure the Select Statement

How would you like to retrieve data from your database?

☐ Specify a custom SQL statement or stored procedure

☒ Specify columns from a table or view

Name: Categories

Columns:

- ☒ *
- ☐ CategoryID
- ☐ CategoryName
- ☐ Description
- ☐ Picture

☐ Return only unique rows

WHERE...

ORDER BY...

Advanced...

SELECT statement:

SELECT * FROM [Categories]

< Previous Next > Finish Cancel

Yukarıda görülen ekrandan SELECT sorgusu ile birlikte INSERT, UPDATE, DELETE sorguları da oluşturulabilir. Hatta, WHERE butonuna tıklanarak, oluşturulan sorguya WHERE ifadesinin de eklenmesi sağlanıp, kayıtlar filtrelenerek getirilebilir. ORDER BY butonu ile getirilen kayıtların belirtilen sıraya göre dizilmesi sağlanabilir. “Return only unique rows” kutusu işaretlenerek sorguya DISTINCT sözcüğü eklenir ve tekrar eden kayıtların sonuç kümesine eklenmemesi sağlanabilir. Bütün bunların yetmediği ya da Stored Procedure kullanımı gerektiren bir durumda “Specify a custom SQL statement or stored procedure” seçeneği ile özel bir SQL sorgusu yazılabilir ya da var olan bir Stored Procedure seçilebilir.

WHERE butonu tıklandığında yeni bir ekran açılacak ve bu ekrandan istenilen filtreleme uygulanabilecektir. Oluşturulan WHERE tümcesinde bir parametre kullanılarak, SQL Injection ataklarından korunma sağlanmış olur. Parametrenin değerinin nereden geleceği de, bu ekrandan seçilebilmektedir.

Add WHERE Clause

Add one or more conditions to the WHERE clause for the statement. For each condition you can specify either a literal value or a parameterized value. Parameterized values get their values at runtime based on their properties.

Column:

Operator:

Source:

Parameter properties

Value:

Value:

Add

Remove

OK Cancel

WHERE tümcesi oluşturulurken, ilk olarak hangi sütun üzerinde tanımlanacağı, ardından kullanılacak olan karşılaştırma operatörü ve sonrasında da parametrenin değerinin nereden geleceği seçilip Add tuşu ile tümce eklenir. Bu şekilde direkt QueryString'den gelen değerlere göre bile sorgulama yapılabilmesi mümkün olmaktadır.

Yukarıdaki erkarandan sorgunun hazırlandığı ana ekrana döndükten sonra bu ekrandan ORDER BY butonu ile verilerin belirtilen kritere göre sıralanması sağlanabilir.

Add ORDER BY Clause

Specify the columns you would like to order by.

Sort by ☒ Ascending ☐ Descending

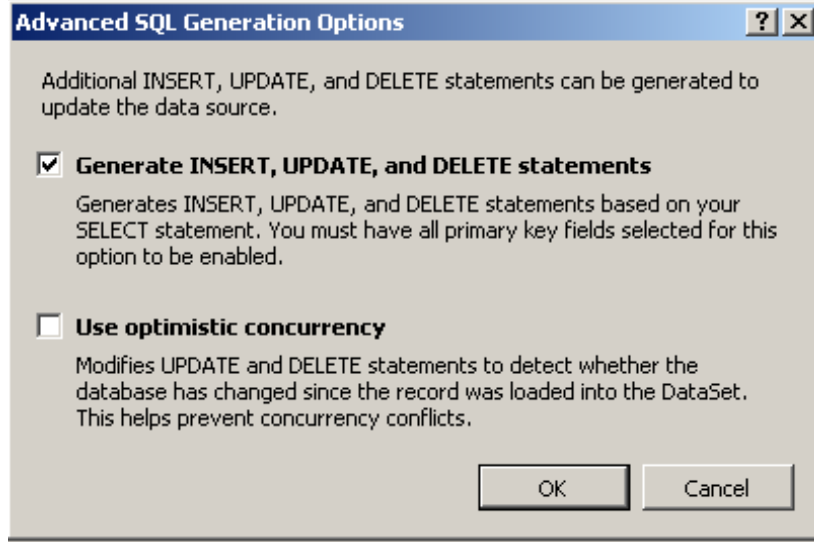
Then by ☒ Ascending ☐ Descending

Then by ☒ Ascending ☐ Descending

SELECT statement:

OK Cancel

ORDER BY tümcesi oluştururken hangi sütuna göre sıralama yapılacağını belirtmesi yeterli olacaktır. Bu işlemin ardından da sıralama artalan mı yoksa azalan mı olacak belirleniyor. Sonrasında da, OK tuşuna basıp oluşturulan tümceyi sorguya ekliyoruz. Bu işlemin ardından sorgunun hazırlandığı ana ekrana dönecektir. Bu ekrandan INSERT, UPDATE ve DELETE cümlelerinin oluşturulması için ADVANCED butonuna basıp, aşağıda görülen ekrana ulaşılabilir.



ADVANCED bölümünde “**Generate INSERT, UPDATE and DELETE statements**” kutucuğu işaretlenerek veri güncellemek için gerekli olan sorguların hazırlanması sağlanır. Kutucuk seçildikten sonra OK tuşuna basıldığında INSERT, UPDATE ve DELETE cümleleri oluşturulacaktır.



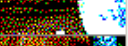
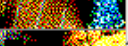
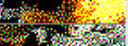
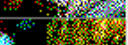
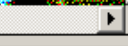
Use optimistic concurrency seçeneği seçilerek, UPDATE ve DELETE işlemlerinde, işlem görecektir olan satırın tüm sütunları WHERE koşulunda AND komutu ile sorguya dahil edilerek, kesin olarak işlem görecektir olan satırda işlem yapılması sağlanır ve çakışmaların ele alınması sağlanmış olur.

Tüm sorgular oluşturulduktan sonra Next tuşu ile son adıma geçilir. Son adım Select cümlesinin test edilecek olduğu alandır. Bu ekrandan “**Test Query**” tuşu ile oluşturulan SELECT cümlesi test edilebilir.

Configure Data Source - SqlDataSource1

Test Query

To preview the data returned by this data source, click Test Query. To complete this wizard, click Finish.

CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffees, teas, beers, and ales	
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	
3	Confections	Desserts, candies, and sweet breads	
4	Dairy Products	Cheeses	
5	Grains/Cereals	Breads, crackers, pasta, and cereal	

Test Query

SELECT statement:

SELECT * FROM [Categories] ORDER BY [CategoryName]

< Previous Next > Finish Cancel

Finish tuşuna basıldıktan tüm sorgular oluşturup **SqlDataSource**'un bulunduğu sayfanın aspx tarafına eklenecektir. SqlDataSource'un oluşturduğu kodlar aşağıda yer almaktadır. Kodlara dikkat edilecek olursa dışarıdan veri alınması gereken tüm yerlerde parametre kullanılmıştır.

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%"$ ConnectionStrings:NorthwindConnectionString %>"
    DeleteCommand="DELETE FROM [Categories] WHERE [CategoryID] = @CategoryID"
    InsertCommand="INSERT INTO [Categories] ([CategoryName], [Description],
        [Picture]) VALUES (@CategoryName, @Description, @Picture)"
    SelectCommand="SELECT * FROM [Categories] ORDER BY [CategoryName]"
    UpdateCommand="UPDATE [Categories] SET [CategoryName] = @CategoryName,
        [Description] = @Description,
        [Picture] = @Picture WHERE [CategoryID] = @CategoryID">
    <DeleteParameters>
        <asp:Parameter Name="CategoryID" Type="Int32" />
    </DeleteParameters>
    <UpdateParameters>
        <asp:Parameter Name="CategoryName" Type="String" />
        <asp:Parameter Name="Description" Type="String" />
        <asp:Parameter Name="Picture" Type="Object" />
        <asp:Parameter Name="CategoryID" Type="Int32" />
    </UpdateParameters>
    <InsertParameters>
        <asp:Parameter Name="CategoryName" Type="String" />
        <asp:Parameter Name="Description" Type="String" />
        <asp:Parameter Name="Picture" Type="Object" />
    </InsertParameters>
</asp:SqlDataSource>
```

SqlDataSource tarafından oluşturulan kodlar dikkatle incelendiğinde uygulama ayarlama dosyasında bulunan bağlantı cümlesinin "**ConnectionString**="<%"\$ ConnectionStrings:NorthwindConnectionString %>" " satırı ile SqlDataSource'a bildirildiği görülmektedir. Kodlarda her sorguda tablo isimlerinin köşeli parantez içerisinde alınarak tablo isimlerinde boşluk ve desteklenmeyen karakter olması durumu ele alınmıştır ayrıca daha

önce de belirtildiği gibi dışarıdan veri alınmasını gerektiren durumlarda parametre kullanılmıştır. Parametreler tanımlanırken tipleri de ayrıca belirtilerek tanımlanmıştır. Görüldüğü gibi tek satır kod yazmadan bir çok işlem yaptık. Bu kodlar kullanmak istenildiğinde, veri kontrollerinden birine bağlanarak yine kod yazmadan kullanılabilir.

SqlDataSource'ta Hata Yakalanması

SqlDataSource kullanılan bir uygulamada eğer varsayılan ayarlar kullanılıyorsa, hataya sebep olacak bir durumda, kontrol dışı ve istenmeyen bir şekilde, varsayılan hata mesajı görüntülenecek ve uygulama sonlandırılacaktır. SqlDataSource'un olayları ele alınarak veri ile ilgili işlemler gerçekleştirilmeden ve gerçekleştirildikten sonra istenilen işlemler yapılabilir. Örneğin veri getirilen bir durumda, Select sorgusu çalışmadan önce **Selecting** olayı tetiklenir ve bu alanda eğer istenilirse Select sorgusuna müdahale edilip sorgu yeniden düzenlenebilir ya da iptal edilebilir. Select sorgusu çalıştıktan sonra ise **Selected** olayı tetiklenecektir. Selected olayında ise sorgunun çalıştırılmasından kaç satır etkilendiği gibi bilgilere ulaşılabilir. Aynı mekanizma benzer şekilde INSERT, UPDATE ve DELETE cümlelerinin çalıştırılmasından önce ve çalıştırıldıktan sonra da işlemektir. Bu alanda sorgu çalıştırıldıktan sonra tetiklenen olaylar ele alındığında çalıştırılacak metodun **e** parametresi üzerinden erişilen **Exception** ve **ExceptionHandled** adında iki tane özellik dikkat çekiyor. Exception özelliği oluşan hatayı döndürür ve bu alanda eğer hata oluştuysa oluşan hata hakkında işlem yapılmasına olanak tanır. ExceptionHandled özelliği ise boolean tipinden değer alır ve hata kod ile yakalandıysa **"true"** değeri atanarak varsayılan hata mesajının görüntülenmesini engeller. Aşağıdaki kod bloklarında Select sorgusu çalıştırıldıktan sonra bir hata oluştu ise bu hatayı ele alacak olan kodlar yer almaktadır.

```
protected void SqlDataSource1_Selected(object sender, SqlDataSourceStatusEventArgs e)
{
    if (e.Exception!=null)
    {
        Response.Write("Hata: "+e.Exception.Message);
        e.ExceptionHandled = true;
    }
}
```

Kodlar metod adından da anlaşılacağı üzere SqlDataSource'un selected olayı tetiklendiğinde çalışacaktır ve eğer hata var ise oluşturulan hatanın mesajı ekrana yazdırılıp hatanın yakalandığı belirtilecektir.

EĞİTİM :

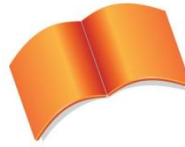
**VERİYE ERIŞİM VE
NAVİGASYON**

Bölüm :

Veriye Erişim

Konu :

Veri Kontrolleri



Microsoft Türkiye

Açık Akademi

Veri Kontrolleri

Veri kontrolleri son kullanıcıya veri göstermek ve son kullanıcının veriyle etkileşime girmesini sağlamak için kullanılan kontrollerdir. ASP.NET'in sunmuş olduğu veri kontrolleri ile birlikte daha önceki script tabanlı dillerde döngüler ve HTML kodları ile yapılan pek çok işlem kolaylıkla gerçekleştirilebilmektedir. Visual Studio'nun araç kutusunda **Data** sekmesinde yer alan veri kontrolleri Web Form'a sürükleyip bırakılarak kolaylıkla kullanılabilir. Veri kontrolleri, bir önceki başlık altında incelenen Veri Kaynağı Kontrolleri ile birlikte bütünleşik olarak çalışıp tek satır kod yazmadan veriye erişim sağlayabilecek yeteneklere sahiptir.

Veri kontrolleri, veri kaynağı kontrollerine bağlı kalmaksızın arka planda yazılım geliştiriciler tarafından yazılan kodlar ile de kolaylıkla kullanılabilir. Bu alanda veri kontrollerini tanıyıp veri görüntüleyen öğelerinin detaylarını biliyor olmak gerekmektedir. Bu bölümde veri kontrollerinin her iki kullanım şeklini de örnekleyeceğiz.

ASP.NET'in ilk sürümünde, sadece **Repeater**, **DataList** ve **DataGrid** kontrolleri varken, sonraki sürümler ile birlikte **DataGrid**, **GridView** adını aldı ve yeni veri kontrolleri eklendi. Bunlar kullanıcıların bir kayıt üzerinde tüm işlemleri yapmalarını sağlayacak olan **FormView** ve var olan kontrollerle birlikte bütünleşik olarak çalışıp bir kaydın detaylarını göstermek için kullanılacak **DetailsView** kontrolleridir. İlerleyen ekranlarda kontrollerin kullanımı detaylıca anlatılmaktadır. İlk olarak bu kontrollerin içerisinde en ilkeli olan **Repeater** ile kontroller hakkında konuşmaya başlayalım.

Repeater

Veri görüntülemek için kullanılacak olan Repeater kontrolü veri kontrolleri içerisindeki en ilkel ve dolayısıyla en az yeteneği olan veri kontrolüdür. Kontrolde veri görüntülemek için kullanılacak toplam beş tane şablon (Template) vardır, bu şablonlarda yazılım geliştirici tarafından gerçekleştirilen tasarıma göre, veriler kullanıcılara sunulur. Bu şablonlar ile ilgili bilgi, aşağıda listelenmektedir.

ItemTemplate: Verinin görüntülenecek olduğu şablondur. Bu alanda gerçekleştirilen tasarıma göre veriler ardışık olarak gösterilir. Yazılım geliştiriciler, bu alanda tasarım yapmak ve diğer ASP.NET sunucu kontrolleri ve HTML kontrollerini bu alana eklemekte özgürdürler.

ItemTemplate içerisinde veri `<%# Eval("GoruntulenecekSutunAdi ") %>` kod bloğu ile görüntülenebilir. Hatta bu kod bloğu, Label gibi veri görüntüleyen kontrollerin Text özelliğine bağlanarak da kolaylıkla kullanılabilir. **Eval** ile birlikte belirtilen sütun kullanıcıya sadece okunabilir şekilde görüntülenebilir yani veri kaynağından uygulamaya tek yönlü veri taşıma işlemi gerçekleştirilmiş olur. ItemTemplate içerisinde belirtilen sütunun görüntülenmesi için Repeater kontrolüne herhangi bir veri kaynağı kontrolü bağlanmış olması ya da arka planda kod yazarak kullanılacak olan veri kaynağı belirtilmiş olmalıdır. Aksi takdirde, doğal olarak veri görüntülenemeyecektir.

AlternatingItemTemplate: Web sitelerinde veriler görüntülenirken, genellikle ardışık verilerin görünüşleri birbirlerinden farklı tutularak kullanıcıların verileri daha kolay takip etmesi sağlanmış olur. Bu farklılık AlternatingItemTemplate ile sağlanabilmektedir. Örneğin ItemTemplate'te gösterilecek olan verinin font rengi mavi arka plan rengi beyaz olarak tasarlanıp, AlternatingItemTemplate'te de bunun tam tersi yapılarak oldukça dikkat çekici ve hoş bir görünüm elde edilebilir.

SeparatorTemplate: Veriler görüntülenirken ardışık veriler arasına bir ayraç eklenmek isteniliyorsa SeparatorTemplate kullanılmalıdır. SeparatorTemplate görüntülenen her veriden sonra otomatik olarak görüntülenip araya belirtilen tasarımı ekleyerek verilerin birbirlerinden kesin bir şekilde ayrılmasını sağlar.

HeaderTemplate: Repeater'da görüntülenen verilere bir başlık bilgisi eklemek için HeaderTemplate kullanılır. HeaderTemplate veriler gösterilmeden önce en üstte görüntülenip gerekli bilgileri kullanıcılara gösterecektir. Bu alanda görüntülenen içerik hakkında kullanıcılara bilgi vermek mantıklı bir yaklaşım olacaktır.

FooterTemplate: Verilerin görüntülenme işlemi bittikten sonra kullanıcılara en altta bir bilgi sunmak için FooterTemplate kullanılır. FooterTemplate veriler görüntüledikten hemen sonra verilerin altına eklenerek görüntülenir ve istenilen içerik kullanıcılara sunulur. FooterTemplate ile kullanıcılara görüntülenen verilerin sayısı, sayfa sonu toplamı vb. değerler hakkında bilgi vermek iyi bir yaklaşım olacaktır.

Bu şablonların tamamının kullanımını içerecek şekilde SQL Server'ın örnek veri tabanlarından biri olan Northwind'den Products tablosunda bulunan birkaç sütun görüntülenmesini içerecek küçük bir örnek gerçekleştirildiğinde Repeater kontrolünün kullanımı daha iyi bir şekilde anlaşılabilir. Bu işlemler için yeni bir Web Site projesi açıp, Web Form'a Repeater kontrolü sürükleyip bıraktıktan sonra, Repeater kontrolünün şablonları doldurulmaya başlanabilir. İlk olarak ItemTemplate'tin ayarlanması ile işlemlere başlayalım. Bu alanda ProductID, ProductName ve UnitPrice alanlarını göstermek yeterli olacaktır. ProductID ve ProductName yan yana ve hemen altlarında da UnitPrice alanı görüntülenecek şekilde bir tasarım aşağıda gördüğün gibi gerçekleştirilebilir.

```
<asp:Repeater ID="Repeater1" runat="server">
  <ItemTemplate>
    <asp:Label ID="Label1" runat="server" Text='<%# Eval("ProductID")
%>' Font-Size="Small"></asp:Label>
    &nbsp;
    <asp:Label ID="Label2" runat="server" Text='<%# Eval("ProductName")
%>' Font-Size="Small"></asp:Label>
    <br />
    <asp:Label ID="Label3" runat="server" Text='<%# Eval("UnitPrice")
%>'></asp:Label>
    <br />
  </ItemTemplate>
</asp:Repeater>
```

Kodlar incelendiğinde toplam üç tane Label kontrolü kullanıldığı, ilk iki tanesinin arasında bir boşluk olacak şekilde yan yana eklendiği ve diğerinde bu ikisinin hemen altına gelecek şekilde tasarlandığı görülecektir. Label'ların Text özelliğine de Repeater'a bağlanılan veri kaynağından veriler **Text='<%# Eval("ProductID") %>'** kod bloğu ile getirilmektedir. Tasarıma dikkat edildiğinde şu ana kadar yazı rengi ile ilgili herhangi bir şey yapılmadığı fark edilecektir.

Şimdi bir tane **AlternatingItemTemplate** ekleyip, ardışık kayıtların birbirinden farklı görüntülenmesini sağlayalım. Bu işlemten sonra Repeater'ın kodları aşağıdaki hali alacaktır.

```
<asp:Repeater ID="Repeater1" runat="server">
  <ItemTemplate>
    <asp:Label ID="Label1" runat="server" Text='<%# Eval("ProductID")
%>' Font-Size="Small"></asp:Label>
    &nbsp;
    <asp:Label ID="Label2" runat="server" Text='<%# Eval("ProductName")
%>' Font-Size="Small"></asp:Label>
    <br />
    <asp:Label ID="Label3" runat="server" Text='<%# Eval("UnitPrice")
%>'></asp:Label>
    <br />
  </ItemTemplate>
  <AlternatingItemTemplate>
    <asp:Label ID="Label4" runat="server" Text='<%# Eval("ProductID")
%>' Font-Size="Small"
    ForeColor="Blue"></asp:Label>
```

```

        &nbsp;
        <asp:Label ID="Label5" runat="server" Text='<%# Eval("ProductName")
%>' Font-Size="Small"
        ForeColor="Blue"></asp:Label>
        <br />
        <asp:Label ID="Label6" runat="server" Text='<%# Eval("UnitPrice")
%>' ForeColor="Blue"></asp:Label>
        <br />
    </AlternatingItemTemplate>
</asp:Repeater>

```

Kodlardan görüldüğü gibi ItemTemplate'ten farklı olarak sadece Label'ların ForeColor özellikleri Blue olarak ayarlandı. Diğer özelliklerde ise herhangi bir değişiklik yapılmadan aynen korundu.

Sırada, listelenen ürünlerin arasına birer ayraç ekleme işlemi var. Bu işlem için var olan kodlara bir tane **SeparatorTemplate** eklenip gerekli işlemler gerçekleştirilebilir.

```

<SeparatorTemplate>
    <hr />
</SeparatorTemplate>

```

Yukarıda görülen kodlar ile listenen her veriden sonra araya bir tane çizgi çektilmesi sağlanmış oluyor.

Listenen verilerin ne olduğunu kullanıcılara belirtmek için **HeaderTemplate** içinde gerekli tasarım yapıp ardından FooterTemplate ile kullanıcılara bu alanda gösterilen veriler hakkında bilgi verilebilir. En altta toplam ürün sayısı listeleniyor olsun. Bu işlem için kod tarafında bir tane özellik (Property) ekleyip listelenen ürün sayısı bu özellik aracılığı ile bu alana yazdırılıyor olsun. Tanımlanan özellik aşağıda yer almaktadır.

```

private int _toplamUrunSayisi;

protected int ToplamUrunSayisi
{
    get { return _toplamUrunSayisi; }
    set { _toplamUrunSayisi = value; }
}

```

Ürünler listelendikten sonra toplam ürün sayısı bu özelliğe atanarak sayfaya taşınmış olacaktır. Topam ürün sayısının sayfaya taşınması için FooterTemplate'te yazılması gereken kodlar ve başlık görüntülemek için yazılan kodlar aşağıda yer almaktadır.

```

<HeaderTemplate>
    <asp:Label ID="Label7" runat="server" Text="Ürünler" BackColor="Blue"
    ForeColor="white" width="200px"></asp:Label>
    <br />
</HeaderTemplate>

<FooterTemplate>
    <asp:Label ID="Label8" runat="server" BackColor="Blue" ForeColor="white"
    width="200px">Toplam Ürün Sayısı: <%Response.Write(ToplamUrunSayisi); %></asp:Label>
</FooterTemplate>

```

Kodlara dikkat edildiğinde arka planların mavi ve yazıların beyaz olduğu görülecektir. Kod tarafında yazılan özelliğin de kontrole Response.Write() ile bağlandığı dikkatli gözlerden kaçmamıştır. Yukarıdaki iki şablon ve bir önceki adımda tasarlanan SeperatorTemplate Repeater'ın şablonları arasına eklendikten sonra sayfa çalıştırılmadan dizayn bölümünde Repeater'ın görünümü aşağıdaki gibi olacaktır.

Ürünler
Databound Databound Databound
Databound Databound Databound
Databound Databound Databound
Databound Databound Databound
Databound Databound Databound

Bu işlemlerin sonunda sıra geldi veritabanından verileri uygulamaya getirecek işlemleri yapmaya bu alanda veriler bir veri kaynağı kontrolünden getirilebileceği gibi eğer istenirse kod yazarak da getirilebilir. Bu örnekte ikinci yöntem tercih edilip daha sonraki örneklere de kullanmak için geriye DataTable döndüren bir metod yazılacaktır. Metod, Northwind veritabanındaki Products tablosundaki istenilen verileri getiriyor olacaktır. Yazılan metod aşağıda yer almaktadır. Tabi bu metodu kullanmak için System.Data.SqlClient isimalanının sayfaya eklenmiş olması gerekmektedir.

```
protected DataTable VeriGetir()
{
    SqlConnection baglanti = new SqlConnection();
    baglanti.ConnectionString = "Data Source=.; Initial Catalog=Northwind;
Integrated Security=True";

    SqlCommand komut = new SqlCommand();
    komut.Connection = baglanti;
    komut.CommandText = "Select TOP 5 ProductID,ProductName,UnitPrice FROM
Products";

    SqlDataAdapter adapter = new SqlDataAdapter(komut);
    DataTable dt = new DataTable();

    adapter.Fill(dt);

    return dt;
}
```

Gerekli metod da yazıldıktan sonra sıra geldi verileri bağlamaya. Sayfa ilk defa çalıştırıldığında gerekli verileri bağlamak için PageLoad'a aşağıdaki kodlar yazılmalıdır.

```
protected void Page_Load(object sender, EventArgs e)
{
    //Bir DataTable tanımlanıp metotdan dönen değerler DataTable'a atılıyor.
    DataTable dt = VeriGetir();

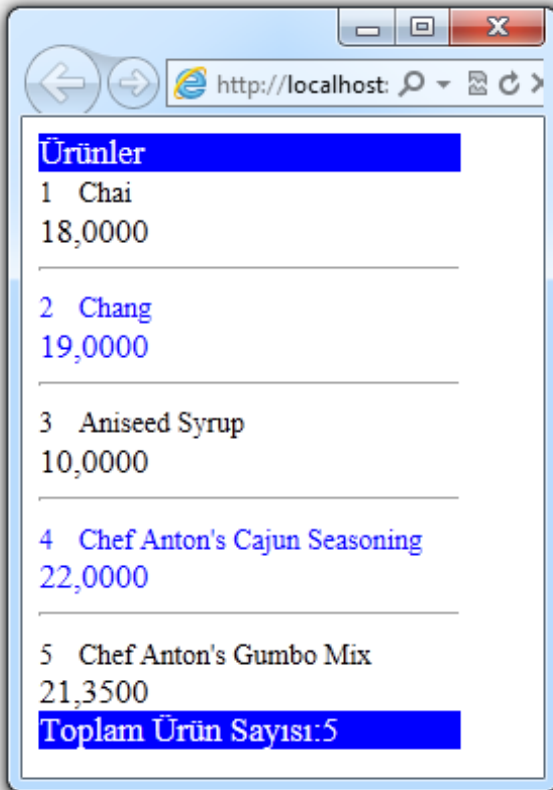
    //Toplam ürün sayısı ayarlanıyor.
    ToplamUrunSayisi = dt.Rows.Count;

    //Repeater'ın kullanacak olduğu DataSource belirtiliyor.
    Repeater1.DataSource = dt;

    //Repeater'a veri bağlanıyor.
}
```

```
} Repeater1.DataBind();
```

Kodlar incelendiğinde ilk olarak bir tane DataTable nesnesi tanımlanıp metotdan dönen değer bu DataTable'a atanıyor. Metotdan dönen değer, direkt Repeater'a DataSource (veri kaynağı) olarak da gösterilebilirdi ama buradaki amaç ilk olarak toplam eleman sayısını hesaplıyor olmak. DataTable'daki değerlere göre ToplamElemanSayisi isimli özelliğe ilgili değer atanıyor. Ardından, ilgili DataTable, Repeater'a DataSource olarak gösteriliyor. ASP.NET tarafında sayfaya veri getirmek için bu kadar işlem yeterli değildir. Veri kaynağı gösterildikten sonra verilerin bağlanması için **DataBind()** metodunun çağırılması gerekir. Bu işlemlerin ardından sayfa çalıştırıldığında aşağıdaki görünüm elde edilecektir.



DataList

DataList'te tıpkı Repeater gibi verilerin liste halinde görüntülenmesine olanak tanır ancak Repeater'a göre daha gelişmiş bir kontroldür ve yazılım geliştiricilere, Repeater'a göre daha fazla şablon ve özellik sunar. Repeater'da veri güncelleme problem oluştururken, DataList'tin veri güncellemek için özel bir şablonu vardır ve DataList kontrolü üzerinden kolaylıkla veri güncelleme işlemleri gerçekleştirilebilir.

DataList'te de veri görüntüleme işlemleri tıpkı Repeater'da olduğu gibi şablonlar aracılığı ile yapılmaktadır. DataList, Repeater'da da bulunan toplam beş adet şablon ile birlikte seçilen veriyi görüntülemek ve veri güncellemek için iki adet şablona daha sahiptir. Bu şablonlar açıklamaları ile birlikte aşağıda yer almaktadır.

EditItemTemplate: DataList üzerinden veri güncelleneceği zaman, veri güncellemek için, verileri güncellenebilir bir şekilde listeleyecek olan kontroller, bu şablon içerisinde yer almalıdır. Bu şablon varsayılan olarak gizlenmiş durumdadır, bu şablonu görünür kılmak için DataList'in **EditItemIndex** özelliğine güncellemek istenilen kaydın sıra numarası belirtilmelidir.

SelectedItemTemplate: Seçilen verinin ne şekilde görüntüleneceği bu şablon aracılığı ile belirlenmektedir. Bu şablon da, tıpkı EditItemTemplate'te olduğu gibi varsayılan olarak görünmez, ancak SelectedIndex özelliğine seçili olan kaydın sıra numarası atandığında görünür olur.

Bir önceki Repeater örneğini DataList ile bir kez daha gerçekleştirelim ve verilerin güncellenmesi sağlayalım. Verileri DataList'te görüntülemek için yeni bir WebForm'a bir DataList ilave edilip verileri görüntüleyecek olan şablonları tıpkı bir önceki Repeater örneğinde olduğu gibi ayarlayalım. Bu işlemten sonra EditItemTemplate ayarlayıp, verileri, kullanıcılara üzerinde değişiklik yapılabilecek şekilde sunacağız. DataList'in görünümünü değiştirmek için, DataList'in şablonlarında gerekli yerlere, Button kontrolü ilave edip, eklenen Button kontrollerinin CommandName özellikleri, gerekli işlemleri yapacak şekilde ayarlanmalıdır. DataList'te belirli olayları tetikleyen, belirli CommandName'ler yer almaktadır. DataList'te kullanılacak CommandName'ler ve açıklamaları şunlardır;

Edit: CommandName özelliği Edit olarak ayarlanan Button'a tıklanıldığında EditCommand olayı tetiklenir ve tıklanılan alan güncellenebilir bir şekilde görüntülenir.

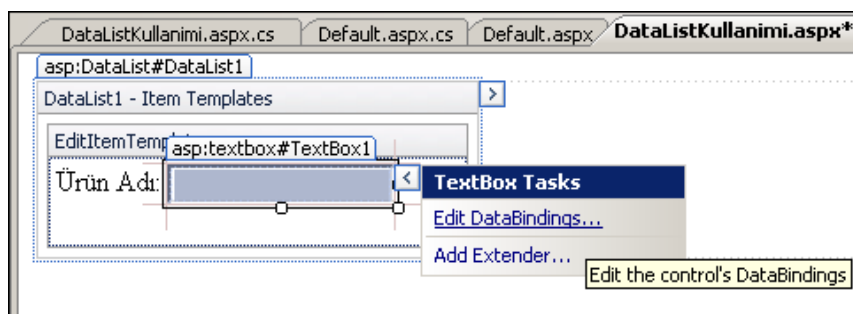
Update: EditItemTemplate içindeki bir Button'un CommandName özelliğine atanması mantıklı olan Update ile UpdateCommand olayı tetiklenir ve üzerinde çalışılan verinin güncellenmesi sağlanabilir.

Delete: DeleteCommand olayının tetiklenmesi için üzerine tıklanıldığında verinin silinmesi için gerekli işlemleri gerçekleştirecek olan buttonun CommandName özelliği Delete olarak atanmalıdır.

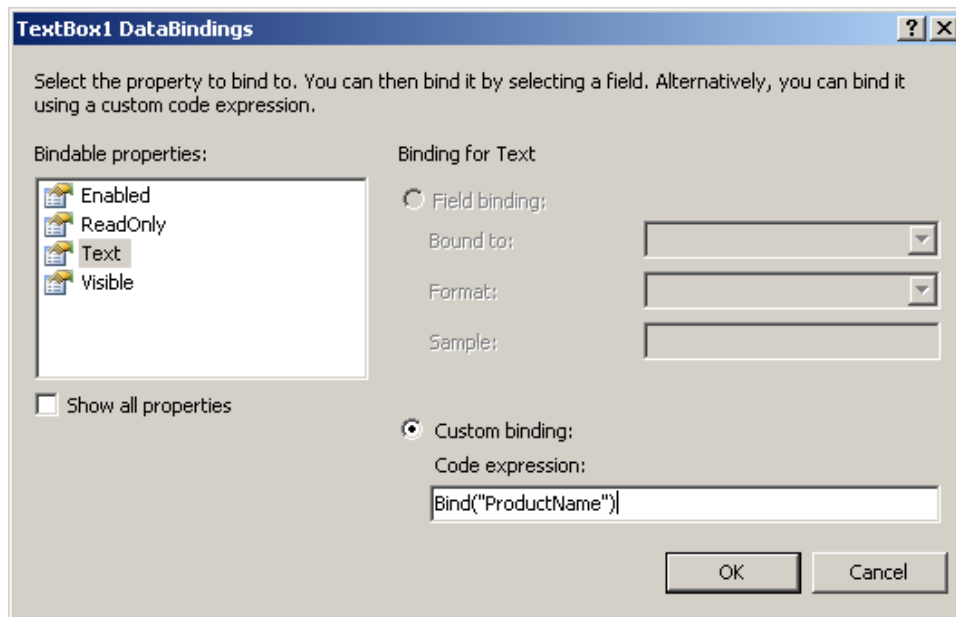
Cancel: CancelCommand olayını tetikleyecek olan CommandName. DataList'tte üzerinde çalışılan verinin güncellenmesinden vazgeçilmek istenildiğinde bir Button'un CommandName özelliği Cancel olarak ayarlanıp EditItemTemplate içerisine eklenebilir. Böyle bir durumda kullanıcılar Button'a tıkladıklarında CancelCommand olayı tetiklenecektir, tetiklenen olay yakalanıp gerekli işlemler yaptırılabilir.

DataList'in veri görüntüleyecek olan şablonları bir önceki gibi aynen ayarlandıktan sonra dizayn bölümüne geçildiğinde Visual Studio ile görsel olarak da şablonların tasarlanabildiği farkedilmelidir. Datalist, Repeater kontrolüne göre daha gelişmiş bir kontrol olduğu için Visual Studio ortamında da, daha kolay bir şekilde tasarım yapılabilir. Visual Studio ortamında şablonları tasarlamak için SmartTag üzerindeki **Edit Templates** bağlantısı ile şablonlar ayrı ayrı görüntülenir ve istenilen şablon sürükleyip bırak tekniği ile kolayca yeniden tasarlanabilir. DataList'in şu andaki görünümü bir önceki örnektekinden farksız olmalıdır. İlk olarak verilerin güncellenebilmesi için EditItemTemplate içerisinde gerekli işlemler gerçekleştirilmelidir. Bu işlem için SmartTag'den Edit Templates bağlantısına tıklandıktan sonra karşılaşılan görünümünden EditItemTemplate seçilip gerekli işlemler gerçekleştirilebilir.

EditItemTemplate içerisine bir tane TextBox sürükleyip bırakıldıktan sonra, görüntüleyecek olduğu veriyi bağlamak için TextBox'ın SmartTag'i üzerinde yer alan Edit DataBindings bağlantısına tıklanır ve TextBox'ın Text özelliğine getirilecek olan veri belirlenir.



Edit DataBindings bağlantısına tıklandıktan sonra aşağıdaki ekran ile karşılaşılacaktır. Bu ekran yardımı ile EditItemTemplate içerisine eklenen kontrollerin özellikleri ayarlanabilir.

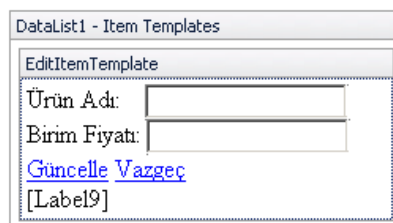


Yukarıdaki resimden de görüldüğü gibi TextBox kontrolünün Text özelliğine veri bağlanmaktadır. Bu alanda verinin **Bind** metodu ile bağlandığına dikkat edilmelidir. Daha önceki bölümlerde veriler sadece veritabanından uygulamaya taşınıyorsa Eval metodunun kullanımının uygun olduğundan bahsedilmişti. Veriler çift yönlü olarak gelip gidiyor ise de Bind metodunun kullanımı uygundur.



Eval veri kaynağından uygulamaya doğru tek yönlü veri akışı sağlarken **Bind** çift yönlü veri akışı sağlar. Veri güncelleme, silme ve ekleme işlemlerinde Bind metodunu tercih etmek gerekmektedir.

EditItemTemplate içerisine, birim fiyatı güncellemek için, bir tane TextBox daha ekleyip benzer şekilde veri bağlandıktan sonra iki adet LinkButton ekleyip CommandName özelliklerini **Update** ve **Cancel** olarak ayarlayalım. Bu işlemin ardından son olarak üzerinde çalışılan verinin belirlenebilmesi için visible özelliği False olarak ayarlanan bir tane de Label kontrolü ekleyerek EditItemTemplate için gerekli işlemleri tamamlayalım. Bu işlemin ardından EditItemTemplate'ın görünümü aşağıdaki gibi olacaktır.



EditItemTemplate üzerinde design tarafında yapılan bu değişikliklerden sonra Visual Studio'nun oluşturmuş olduğu kodlar aşağıda yer almaktadır.

[illegible]


```

        <br />
        Birim Fiyatı:
        <asp:TextBox ID="TbFiyat" runat="server" Text="<%#
Bind("UnitPrice") %>'></asp:TextBox>
        <br />
        <asp:LinkButton ID="LinkButton1" runat="server"
CommandName="Update">Güncelle</asp:LinkButton>
        &nbsp;<asp:LinkButton ID="LinkButton2" runat="server"
CommandName="Cancel">Vazgeç</asp:LinkButton>
        <br />
        <asp:Label ID="LabelProductID" runat="server" Text="<%#
Eval("ProductID") %>'
            visible="False"></asp:Label>
        &nbsp;<br />
    </EditItemTemplate>

```

EditItemTemplate'in ayarlanmasından sonra sıra geldi seçilen kaydın güncellenebilir olarak görüntülenmesini sağlayacak olan ve belirtilen kaydın silinmesini sağlayacak olan işlemlere. Bu işlemler için DataList'in ItemTemplate ve AlternatingItemTemplate şablonlarına ikişer tane LinkButton eklenip CommandName özellikleri **Edit** ve **Delete** olarak ayarlanacaktır. Şablonlara ilave edilecek olan kodlar aşağıda yer almaktadır. Bu kodlar gerekli şablonlara eklendikten sonra, sıra, tetiklenen olayları yakalayıp işlem yapmaya geldi.

```

<asp:LinkButton ID="LinkButton3" runat="server"
CommandName="Edit">Düzenle</asp:LinkButton>
&nbsp;<br />
<asp:LinkButton ID="LinkButton4" runat="server"
CommandName="Delete">Sil</asp:LinkButton>

```



DataList'te veri görüntülerken kullanılacak olan **VeriGetir()** isimli metod bir önceki örnekte yazılmıştır ve bu örneğin pek çok yerinde kullanılmaktadır. VeriGetir() isimli metoda daha önceki sayfalardan ulaşılabilir. ToplamUrunSayisi isimli özellik de daha önceki örnekte tanımlanmıştır. Bu örnekte yer alan VeriGetir() metodu ve ToplamUrunSayisi özelliği diğer örnekten kopyalanmıştır.

Düzenle butonuna tıklanıldığı zaman EditCommand olayı tetiklenecektir, dolayısıyla bu olay ele alınıp gerekli işlemler yapılmalıdır. DataList'in özellikleri penceresinden olaylar bölümüne geçilip EditCommand olayına çift tıklanılarak, EditCommand olayı tetiklendiğinde devreye girecek olan metod oluşturulur. Oluşturulan metodun içerisinde, DataList'te hangi kayıt üzerine tıklanıldığının belirlenmesi gerekmektedir. Üzerine tıklanan kayıttan index numarasına metodun parametrelerinden ikincisi olan **DataListCommandEventArgs** tipindeki e parametresi üzerinden erişilebilir.

```

protected void DataList1_EditCommand(object source, DataListCommandEventArgs e)
{
    DataList1.EditItemIndex = e.Item.ItemIndex;
    DataTable dt = VeriGetir();
    ToplamUrunSayisi = dt.Rows.Count;
    DataList1.DataSource = dt;
    DataList1.DataBind();
}

```

Kodlar incelendiğinde DataList'in **EditItemIndex** özelliğine, seçili olan öğenin indexinin atandığı ve seçili olan öğenin güncellenebilir olarak görüntülenmesinin sağlandığı görülebilir. Bu özelliğin varsayılan değeri -1 olarak ayarlanmış durumdadır. Yukarıdaki kodlar yazıldıktan sonra DataList üzerindeki herhangi bir kayda tıklanıldığında, tıklanan kaydın güncellenebilir bir şekilde görüntülediği fark edilmelidir. Güncelleme işlemi yapmadan geri dönmek istenildiğinde, Vazgeç butonuna basılabilir. Fakat, bu butona basıldığında, hiçbir şey olmadığı görülecektir. Güncelleme işminden vazgeçmek için EditItemIndex özelliğinin yeniden -1 olarak

ayarılanması gerekmektedir. Bu işlem için yazılması gereken kodlar ise aşağıda yer almaktadır. Kodlar, Vazgeç butonu tıklanıldıktan sonra tetiklenecek olan CancelCommand olayının ele alındığı metod içerisine yazılmalıdır.

```
protected void DataList1_CancelCommand(object source, DataListCommandEventArgs e)
{
    DataList1.EditItemIndex = -1;
    DataTable dt = VeriGetir();
    ToplamUrunSayisi = dt.Rows.Count;
    DataList1.DataSource = dt;
    DataList1.DataBind();
}
```

İstenilen kaydın silinmesi için DeleteCommand olayı yakalanmalıdır. Bu işlem için Visual Studio ortamında DataList'in özelliklerinden olaylar bölümünde DeleteCommand'a çift tıklanır ve aşağıda yer alan belirtilen kaydı silecek olan kodlar yazılır.

```
protected void DataList1_DeleteCommand(object source, DataListCommandEventArgs e)
{
    string urunID = ((Label)e.Item.FindControl("LabelID")).Text;
    SqlConnection baglanti = new SqlConnection();
    baglanti.ConnectionString = "Data Source=.; Initial Catalog=Northwind; Integrated Security=True";

    SqlCommand komut = new SqlCommand();
    komut.Connection = baglanti;
    komut.CommandText = "Delete FROM Products WHERE ProductID=@ProductID";
    komut.Parameters.AddWithValue("@ProductID", urunID);

    baglanti.Open();
    komut.ExecuteNonQuery();
    baglanti.Close();
    DataTable dt = VeriGetir();
    ToplamUrunSayisi = dt.Rows.Count;
    DataList1.DataSource = dt;
    DataList1.DataBind();
}
```

Kodlar incelendiğinde, hangi ürünün silineceğinin belirlenmesi için, DataList'in ItemTemplate şablonu içerisinde bulunan **LabelID** isimli Label'ın Text özelliğindeki değerin alındığı görülmektedir. Kullanıcının silinmesi için tıklamış olduğu kayıt ise **DataListCommandEventArgs** tipindeki e parametresinin **Item** özelliğinden elde edilmektedir. Item özelliğinin FindControl metodu ile, ürünün ID'sini taşıyan Label, bulunup ardından Label kontrolüne dönüştürülüp, Text özelliğindeki değer, urunID isimli değişkene aktarılmaktadır. Kodların geriye kalan bölümünde standart veri silme işlemi gerçekleştirilmektedir.

Veri silme işleminin ardından sıra geldi veri güncelleme işlemine bu işlem için de DataList'in UpdateCommand olayı ele alınmalıdır. Hatırlanacağı üzere bu olayı EditItemTemplate içerisine eklenen, CommandName özelliği Update olarak ayarlanmış olan LinkButton tetikleyici olacaktır.

```
protected void DataList1_UpdateCommand(object source, DataListCommandEventArgs e)
{
    string urunID = ((Label)e.Item.FindControl("LabelProductID")).Text;
    string urunAdi = ((TextBox)e.Item.FindControl("TbAd")).Text;
    decimal urunFiyati = decimal.Parse(((TextBox)e.Item.FindControl("TbFiyat")).Text);

    SqlConnection baglanti = new SqlConnection();
}
```

```

        baglanti.ConnectionString = "Data Source=.; Initial Catalog=Northwind;
Integrated Security=True";

        SqlCommand komut = new SqlCommand();
        komut.Connection = baglanti;
        komut.CommandText = @"UPDATE Products SET
                                ProductName=@ProductName,
                                UnitPrice=@UnitPrice
                                WHERE ProductID=@ProductID";
        komut.Parameters.AddWithValue("@ProductID", urunID);
        komut.Parameters.AddWithValue("@ProductName", urunAdi);
        komut.Parameters.Add("@UnitPrice", SqlDbType.Money).Value = urunFiyati;

        baglanti.Open();
        komut.ExecuteNonQuery();
        baglanti.Close();
        DataTable dt = VeriGetir();
        DataList1.EditItemIndex = -1;
        ToplamUrunSayisi = dt.Rows.Count;
        DataList1.DataSource = dt;
        DataList1.DataBind();
    }

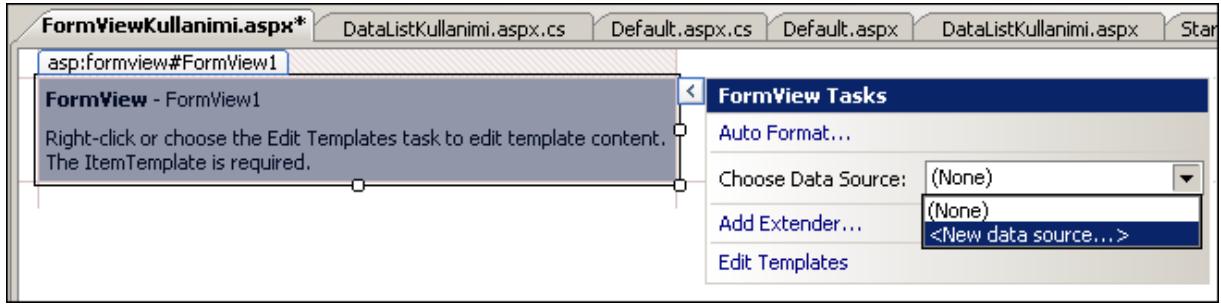
```

Yukarıdaki kodlarla birlikte güncelleme işlemi gerçekleştirilecektir. Bu alanda da, silme işleminde olduğu gibi listelenen kontrollerin Text özelliklerindeki değerler alındı ve Sql sorgusuna parametre olarak gönderildi. Bu alanda ürünün fiyatının gönderildiği parametre, tip güvenliği nedeni ile tipi belirtilerek daha önceki bölümlerden farklı bir şekilde gönderildi.

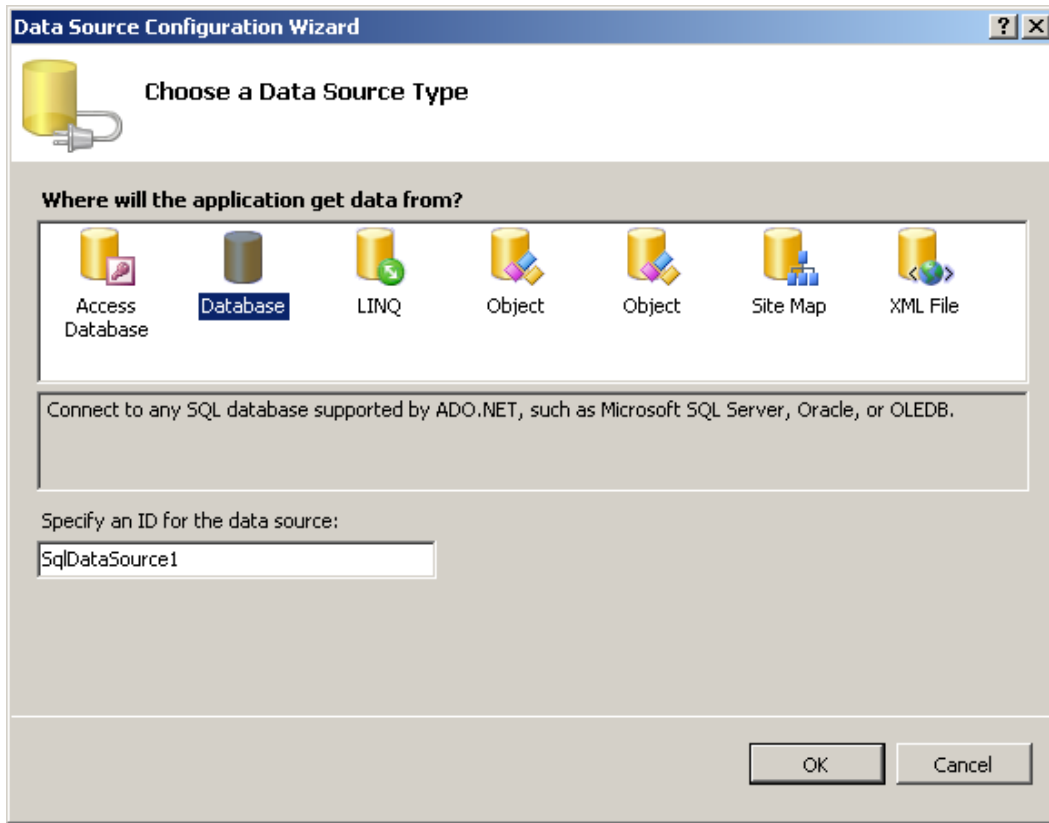
Bu örnek ile birlikte DataList üzerinde veri görüntüleme, veri silme ve veri güncelleme işlemlerinin tamamı gerçekleştirilmiş oldu. Kodlar yazıldıktan sonra, uygulama çalıştırıldığında, üzerine tıklanılan kaydın güncelleştirilebilir bir şekilde görüntülediği farkedilecektir. Şu andaki örnekte sadece beş kayıt listelendiği için bu bir problem oluşturmamakta ancak daha fazla kayıt listelendiği durumlarda kullanıcılar düzenlemek istedikleri kayda tıkladıklarında sayfada PostBack işlemi gerçekleştirilecek ve ardından sayfanın en üstü kullanıcılara gösteriliyor olacaktır, bu durum kullanıcı deneyimi açısından oldukça kötü bir durumdur ve önlem almak gerekir. Page direktifinde bulunan **MaintainScrollPositionOnPostBack** özelliğine true değeri atanarak PostBack sonrasında kaydırma çubuklarının eski yerlerinde olması sağlanır ve kullanıcı tekrardan güncellemek istediği kaydı bulmak zorunda kalmaz. DataList ile bu şekilde güncelleme işlemi yapılacaksa MaintainScrollPositionOnPostBack özelliğinin true olarak ayarlanması önerilir.

FormView

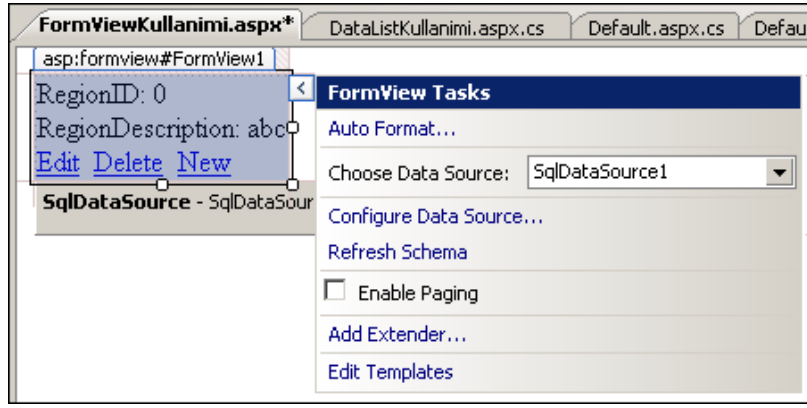
FormView kontrolü verileri teker teker kullanıcıya gösteren kontroldür. FormView kontrolü kullanılarak veriler kullanıcıya teker teker gösterilip her veri üzerinde işlem yapılmasına olanak tanınabilir. Veriler görüntülenirken ve veriler üzerinde çalışırken, kontrolün adından da anlaşılacağı üzere, veriler, alışılan Form görünümü ile kullanıcılara gösterilir. FormView kontrolünün veriler üzerinde güncelleme yapabilmesinin yanında, veri ekleme yeteneği de vardır. Herhangi bir veri kaynağı kontrolü bağlanarak tüm şablonları otomatik olarak oluşturulabilir. FormView kullanmak için bir WebForm'a bir tane FormView kontrolü sürükleyip bırakalım. Daha önceki örneklerden farklı olarak, FormView kontrolü ile birlikte SqlDataSource kontrolünü kullanacağız. FormView ile birlikte bir veri kaynağı kontrolü kullanmak için, kontrolün SmartTag'inden sayfada var olan bir veri kaynağı seçilebilir ya da yeni bir tane oluşturulabilir.



Geliştirilen örnekte sayfa içerisinde herhangi bir veri kaynağı olmadığı için yeni bir tane eklemek amacı ile “**New data source**” seçeneği seçilerek sayfaya bir tane veri kaynağı kontrolü eklenmesini sağlayalım. Bu alandan “New data source” seçeneği seçildikten sonra sayfaya eklenebilecek kontrollerin listelendiği aşağıdaki ekranla karşılaşılacaktır.



Veri kaynaklarının listelendiği ekrandan **Database** seçilip Ok tuşuna bastıktan sonra daha önce SqlDataSource bölümünde açıklanan bağlantı cümlesi seçme ekranı ile karşı karşıya kalınacaktır. Bu bölümden istenilen veri tabanı seçilip devam edilir. Bu örnek için, bu alanda Northwind veritabanı seçilecektir. Daha önce bahsettiğimiz gibi tablo seçme bölümünden **Region** tablosu seçilip, Advanced bölümünden de “Generate INSERT, UPDATE, and DELETE statements” seçeneği seçilerek tüm sorgu cümlelerinin oluşturulmasını sağlayalım. Bu işlemden sonra Next tuşu ile bir sonraki adıma geçip Test Query tuşu ile oluşturulan sorguyu test edelim. Sorgunun test edilmesi ile birlikte FormView kontrolünün kullanacak olduğu tablo belirlenmiş olacak ve bu şemaya göre FormView’in şablonları oluşturulacaktır.



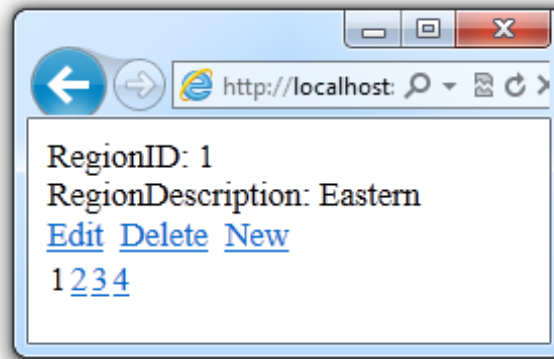
Veri kaynağı kontrolünün ayarlarından sonra Finish tuşu ile sihirbazı sonlandırılır. Ardından, Visual Studio ekranına döndüğünde FormView'in görünümünün tamamen değiştiği görülecektir. Çünkü; verileri sadece okunabilir, güncellenebilir şekilde görüntüleyecek olan şablonların içeriği doldurulmuştur ve hatta yeni veri eklemek için gerekli olan şablonun içeriği de doldurulmuş durumdadır. Edit Templates bağlantısı ile şablonlara göz atılacak olursa bu alanda daha önce açıklanmayan üç adet şablon dikkat çekecektir. Bunlar;

InsertItemTemplate: FormView üzerinden veritabanına yeni veri eklemek için gerekli kontrollerin yer alacak olduğu şablondur. Bu şablonda bulunan kontrollere veri getirilmez ve kullanıcı bu şablonu görüntülediğinde doğal olarak boş şekilde görüntülenir. Kullanıcı gerekli alanları doldurup veri ekledikten sonra tüm kontroller otomatik olarak yeniden boşaltılır.

EmptyDataTemplate: Kullanıcıya sunulacak veri olmadığı zaman, gösterilecek olan şablondur. Bu alanda kullanıcıya verilerin neden boş geldiği hakkında bilgi verilip kullanıcının yapması gereken işlemler belirtilebilir.

PagerTemplate: FormView'de sayfalama kullanılıyorsa sayfalayıcının görünümü bu alanda özelleştirilebilir.

Veriler arasında dolaşılması amacı ile sayfalamanın aktif hale getirilmesi için kontrolün SmartTag'inden Enable Paging kutucuğu seçildikten sonra sayfa çalıştırıldığında aşağıdaki görünümü alacaktır.



Edit butonu ile görüntülenen kayıt güncellenebilir bir şekilde görüntülenebilir ve güncelleme işlemi gerçekleştirilebilir. **Delete** butonu ile kayıt silinebilir ve **New** butonu ile yeni bir kayıt eklenebilir. Bu butonların yaptığı iş, aslında sadece görüntülenen şablonu değiştirmekten ibarettir. Edit butonuna tıklanıldığında EditItemTemplate görünür olmakta, New butonu ile de InsertItemTemplate görünür olmaktadır. Bu butonların çalışma prensibi de DataList kontrolündeki butonlarda olduğu gibidir. Yani, her butonun CommandName özelliğine atanan değer o butonun yapacağı işi belirlemektedir. FormView'de kullanılan CommandName'ler incelendiğinde iki yeni değer dikkat çekmektedir. Bunlar;

New: InsertItemTemplate'in görüntülenmesini sağlayacak olan komuttur.

Insert: InsertItemTemplate içerisinde kullanılması uygun olan bu komut ile üzerinde çalışılan tabloya yeni bir kayıt eklenir ve bir kayıt daha eklemek için tüm TextBox'lar otomatik olarak boşaltılır.

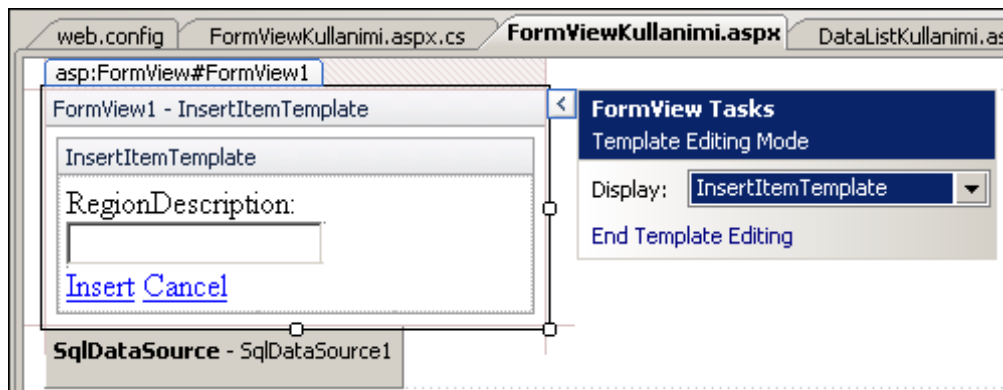
FormView'in toplam üç adet görünüm modu vardır bunlar **ReadOnly**, **Edit** ve **Insert** modlarıdır. Bu modlar **DefaultMode** özelliği ile belirtilir. ReadOnly modda adından da anlaşılacağı üzere veriler sadece okunabilir şekilde gösterilir. ReadOnly mod, FormView'ün varsayılan modudur. Edit modda gösterilen veri üzerinde değişiklik yapılabilirken, Insert modda ise yeni bir kayıt ekleme olanağı sunulmaktadır. Görünüm modlarından kasıt da aslında görüntülenen şablonu değiştirmektir. ReadOnly modda ItemTemplate, Edit modda EditItemTemplate ve Insert modda da InsertItemTemplate görüntülenir. Bu şekilde görüntülenecek olan görünüm modunun seçilebiliyor olması FormView'in daha etkin kullanımı için bir avantajdır. Web dünyasında genellikle kullanıcıların veri güncellediği ve veri eklediği sayfalar farklı olmaktadır. Bu durumda bir tablo için iki ayrı sayfa tasarlanıp birinde DefaultMode değeri Insert, diğerinde de Edit olarak ayarlanarak sayfaları amaçlarına göre ikiye ayırmak kolaylıkla sağlanabilir.

FormView'in Olayları

FormView'in olayları incelendiğinde daha önce bahsedilen SqlDataSource'un olayları ile benzerlik gösterdiği görülmektedir. SqlDataSource'da hatırlanacağı üzere veri eklenmeden hemen önce tetiklenecek olan Inserting ve veri eklendikten hemen sonra tetiklenecek olan Inserted olayı yer almaktaydı. Diğer durumlar için de benzer olaylar söz konusuydu ve bu olaylar yakalandığında pek çok önemli işlem gerçekleştirilebilmekteydi.

FormView'de de veri eklemekten hemen önce **ItemInserting** olayı tetiklenir ItemInserting olayı yakalandığında eklenen veri değiştirilebilir veya yeni veriler eklenebilir ya da veri ekleme işlemi iptal edilebilir. Veri eklendikten sonra da **ItemInserted** olayı tetiklenir ve olay yakalandığında eklenen kayıt hakkında bilgi elde edilebilir. Hata oluşması durumunda da bu alanda işlem yapıp, kullanıcıya anlamlı bir hata mesajı gösterilebilir. Diğer durumlar için de benzer isimde olaylar söz konusudur ve benzer işlemler gerçekleştirilebilir.

NorthWind veritabanında bulunan Region tablosu incelendiğinde RegionID değerinin otomatik olarak artmadığı görülecektir. FormView aracılığı ile bu tabloya veri eklemek istendiğinde FormView'in InsertItemTemplate'inde RegionID içinde bir TextBox olduğu görülecektir. Eğer RegionID değerinin otomatik olarak artması isteniyorsa bunu uygulama tarafından yapmak için ilk olarak RegionID değerinin kullanıcıya sorulması engellenmelidir. Bu işlem için FormView'in InsertItemTemplate'ine geçilir ve RegionID TextBox'ı kaldırılarak InsertItemTemplate aşağıdaki hale getirilir.



InsertItemTemplate içerisinden RegionID TextBox'ı çıkartıldıktan sonra SqlDataSource hala bir adet RegionID parametresi bekliyor olacaktır. Insert sorgusu çalıştırılmadan hemen önce bu parametre

SqlDataSource' a belirtilmelidir. Bu durumda FormView'in **ItemInserting** olayı ele alınıp insert sorgusu çalışmadan hemen önce RegionID parametresinin değeri belirtilebilir. ItemInserting olayı tetiklendiğinde çalışacak olan kodlar aşağıda yer almaktadır.

```
protected void FormView1_ItemInserting(object sender, FormViewInsertEventArgs e)
{
    SqlConnection baglanti = new SqlConnection();
    baglanti.ConnectionString =
    WebConfigurationManager.ConnectionStrings["NorthwindConnectionString"].ConnectionString;

    SqlCommand komut = new SqlCommand();
    komut.Connection = baglanti;
    komut.CommandText = @" SELECT TOP 1 RegionID
                           FROM Region
                           ORDER BY RegionID DESC";

    baglanti.Open();
    int regionID = Convert.ToInt32(komut.ExecuteScalar());
    baglanti.Close();

    regionID++;
    SqlDataSource1.InsertParameters["RegionID"].DefaultValue =
    regionID.ToString();
}
```

Yukarıdaki kodlarda ilk olarak en büyük RegionID değeri bulunuyor ve bu değer bir arttırılıp SqlDataSource'un Insert parametrelerinden RegionID'ye atanıyor. Dolayısı ile RegionID değerinin otomatik olarak artması sağlanmış oluyor. Bu işlemi veritabanı düzeyinde Identity özelliğini Yes olarak ayarlayarak yapmak çok daha mantıklı bir çözüm olacaktır ancak veritabanı üzerinde değişiklik yapılamadığı durumlarda bu gibi sorunları FormView'in olaylarını yakalayıp çözmek mecburi olacaktır.



Bağlantı nesnesi oluşturulurken bağlantı cümlesinin değeri web.config'de bulunan NorthwindCS isimli bağlantı cümlesinden elde edilmektedir. web.config'deki bağlantı cümlesinin elde edilmesi için **WebConfigurationManager** sınıfı kullanılmalıdır. WebConfigurationManager sınıfı System.Web.Configuration isimalanı altında yer almaktadır.

Veri eklerken RegionID değerinin uygulama düzeyinde belirlenmesi çakışmalara yol açabilir ve ortama istisna fırlatılabilir. Fırlatılan istisna, ele alınıp kullanıcıya anlamlı bir hata mesajı gösterilmeli ve uygulamanın anlamsız bir şekilde sonlanması önüne geçilmelidir. İstisna, veri eklendikten sonra fırlatılacağı için **ItemInserted** olayı ele alındığında hedeflenen tasarım gerçekleştirilmiş olur.

```
protected void FormView1_ItemInserted(object sender, FormViewInsertedEventArgs e)
{
    if (e.Exception != null)
    {
        Response.Write("<b>Hata: </b>" + e.Exception);
        e.ExceptionHandled = true;
    }
}
```

ItemInserted olayı tetiklendiğinde çalışacak olan kodlarda ilk olarak istisna fırlatılıp fırlatılmadığı kontrol ediliyor. Eğer bir istisna fırlatıldı ise hata mesajı kullanıcıya gösteriliyor ve uygulamanın anlamsız bir şekilde sonlanmaması için istisnanın yakalandığı belirtiliyor.

GridView

Veriler SqlServer'da olduğu gibi tablo halinde gösterilmek istendiğinde, GridView kontrolü tercih edilmelidir. GridView kullanımı açısından diğer kontrollerle kıyaslandığında en kolay kullanılabilen kontroldür. GridView kullanmak için kontrole bir veri kaynağı belirtmek yeterli olmaktadır. **AutoGenerateColumns** özelliği true olarak ayarlı ise bir veri kaynağı belirtildiği durumda GridView tüm sütunları otomatik olarak oluşturup verileri görüntüleyecektir. Sütunlar otomatik olarak oluşturulabildiği gibi eğer istenilirse yazılım geliştiriciler tarafından da oluşturulabilir. Kontrole herhangi bir veri kaynağı kontrolü eklendiğinde tüm sütunlar ayrı ayrı eklenip görüntüleniyor olacaktır, veri kaynağı kontrolü ile birlikte kullanıldığında GridView üzerinden güncelleme, sıralama, sayfalama ve veri silme işlemleri de oldukça kolay bir şekilde gerçekleştirilebilmektedir.

SqlServer ile birlikte gelen AdventureWorks veritabanındaki Production şeması altında yer alan Product tablosundaki kayıtlar GridView aracılığı ile kullanıcıya gösterilmek istendiğinde, aşağıdaki metod yazılıp kullanılabilir. Bu metod sayfanınPostBack olmaması durumunda PageLoad'da çağrılarak GridView'de veriler kolaylıkla görüntülenebilir.

```
protected void veriGetir()
{
    SqlConnection baglanti = new SqlConnection();
    baglanti.ConnectionString =
    ConfigurationManager.ConnectionStrings["AdventureworksCS"].ConnectionString;

    SqlCommand komut = new SqlCommand();
    komut.Connection = baglanti;
    komut.CommandText = "Select ProductID,Name,ListPrice FROM
    Production.Product";

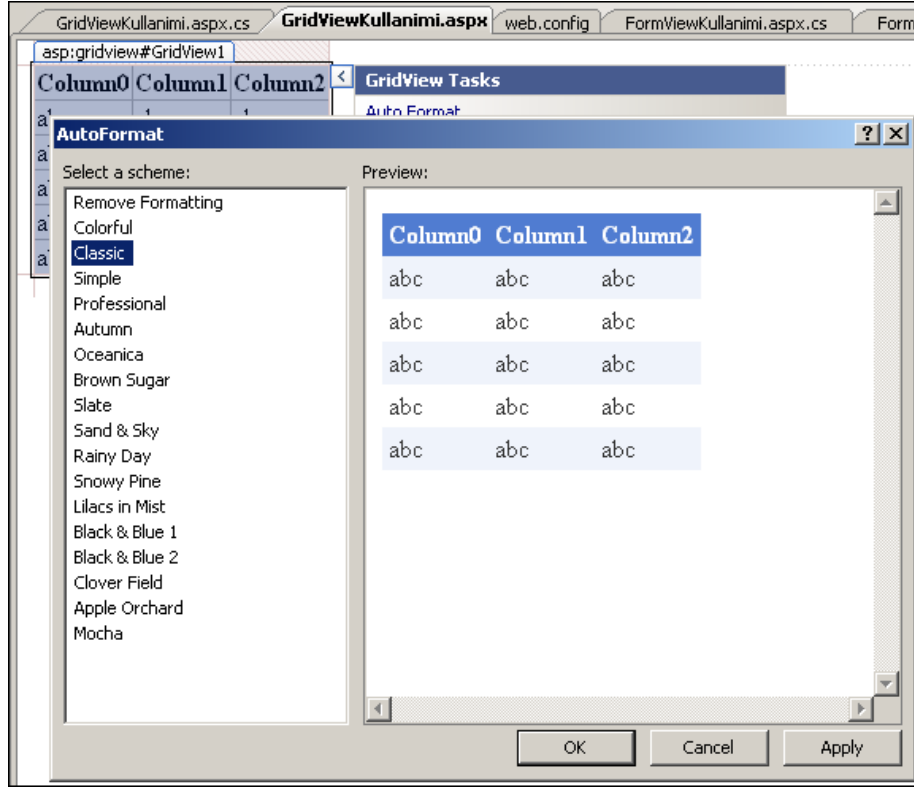
    DataTable dt = new DataTable();
    SqlDataAdapter adapter = new SqlDataAdapter(komut);
    adapter.Fill(dt);

    GridView1.DataSource = dt;
    GridView1.DataBind();
}
```

Sayfa çalıştırıldığında ekran görüntüsü aşağıdaki gibi olacaktır. Veriler otomatik olarak tablo halinde görüntülendi. Dikkat edilmesi gereken bir durum da herhangi bir şablon ayarlamasının yapılmamış olmasıdır. Tüm sütunlar başlık bilgileri ile birlikte otomatik olarak görüntülendi ve kullanıcıya gösterildi.

ProductID	Name	ListPrice
1	Adjustable Race	152,0875
2	Bearing Ball	152,0875
3	BB Ball Bearing	152,0875
4	Headset Ball Bearings	152,0875
316	Blade	152,0875
317	LL Crankarm	152,0875
318	ML Crankarm	152,0875
319	HL Crankarm	152,0875
320	Chainring Bolts	152,0875
321	Chainring Nut	152,0875

Veriler kullanıcıya gösterilirken herhangi bir stillendirme kullanılmadı. Veriler, direkt olarak varsayılan sitilde görüntülendi. Oysa ki, bu şekilde ardışık veriler kullanıcıya gösterilerken, verilerin daha iyi okunması adına farklı sitillerde kullanıcıya sunulur. Daha iyi bir görünüm için GridView'in sitilleri ile teker teker oynanabileceği gibi Visual Studio tarafından sunulan görünümlerden biri de tercih edilebilir. Visual Studio tarafından pek çok kontrolde hazır sitiller sunulmaktadır ve bu sitiller kullanılarak göze hoş gelen oldukça güzel tasarımlar oluşturulabilir. Visual Studio tarafından sunulan sitiller, kontrollerin SmartTag'lerinde yer almaktadır.

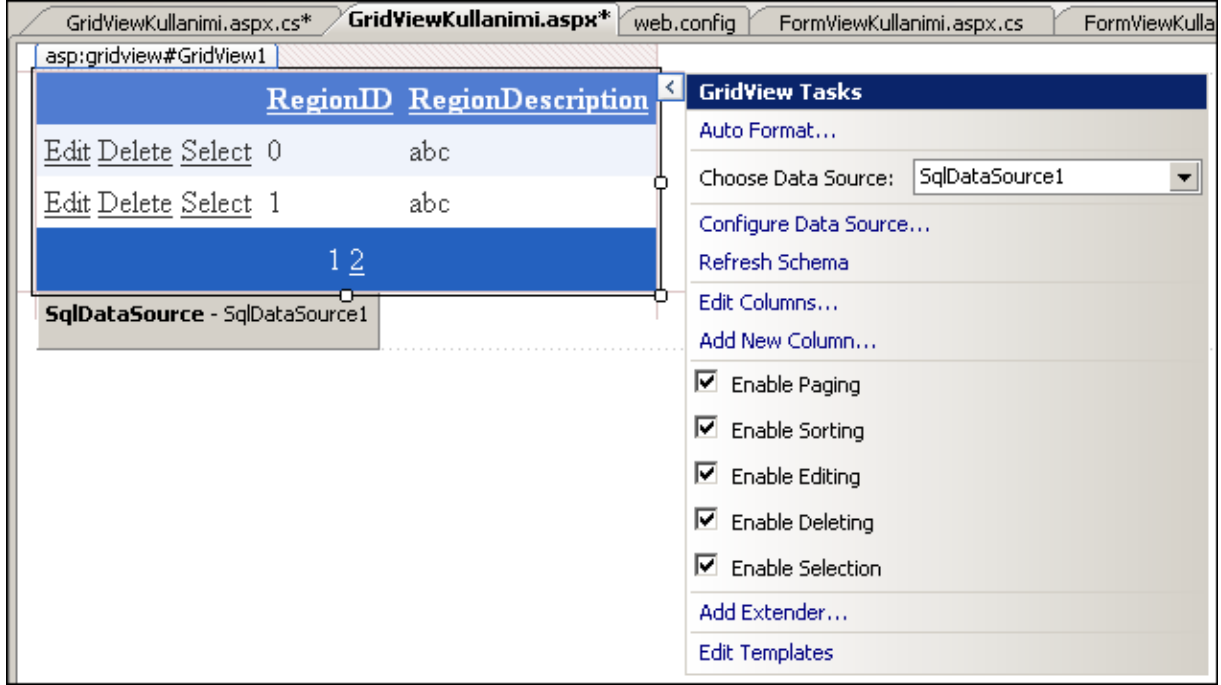


GridView kontrolünün SmartTag'inden Auto Format bağlantısı ile GridView için sunulan tasarımlara ulaşılabilir. Sitenin genel görünümüne göre uygun olan herhangi bir tasarım seçip Ok tuşu ile onaylandıktan sonra GridView seçilen görünüme bürünecektir ve çalıştırdıktan sonra da kullanıcılara daha hoş gelen bir görünüm elde edilecektir. Aşağıdaki resimde Classic görünümü seçildikten sonra GridView'in çalışma anındaki görüntüsü yer almaktadır.

ProductID	Name	ListPrice
1	Adjustable Race	152,0875
2	Bearing Ball	152,0875
3	BB Ball Bearing	152,0875
4	Headset Ball Bearings	152,0875
316	Blade	152,0875
317	LL Crankarm	152,0875
318	ML Crankarm	152,0875
319	HL Crankarm	152,0875
320	Chainring Bolts	152,0875

GridView önceden yazılan kodlar ile birlikte kullanılabileceği gibi herhangi bir veri kaynağı kontrolü ile de kullanılabilir. Hatta bir veri kaynağı kontrolü ile pek çok özelliği daha kolay kullanılabilir. GridView'in veri kaynağı kontrolü ile kullanımını örneklemek için bir WebForm üzerine yeni bir GridView sürükleyip bıraktıktan

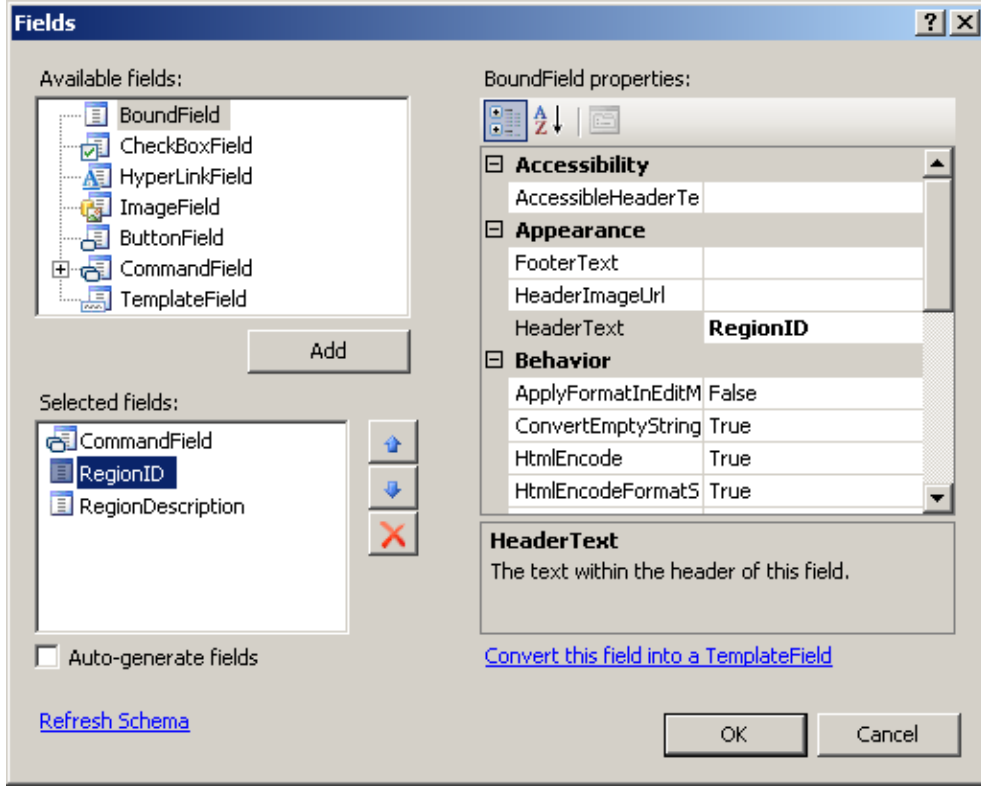
sonra GridView'in SmartTag'inden "New data source" seçeneği ile sayfaya bir tane SqlDataSource ekleyelim. FormView örneğinde kullanılan Nortwind veritabanında yer alan Region tablosu seçiliyor olsun ve tüm ayarları tıpkı FormView örneğindeki gibi gerçekleştirelim. Gerekli ayarlamaların ardından GridView'in SmartTag'inin görünümü değişecektir ve pek çok yeni seçenek sunulacaktır.



SmartTag'de artık verilerin sayfanmasına, sıralanmasına, düzenlenmesine, silinmesine ve seçilmesine olanak tanıyacak olan seçeneklerin yer aldığı görülmektedir. Bu seçeneklerin tamamı seçilip **PageSize** özelliği 2 olarak ayarlandıktan sonra uygulamayı çalıştırın ve göz atın. Görüleceği üzere her sayfada iki kayıt listelenmektedir ve başlık bilgilerine tıklanarak kayıtlar sıralanabilmektedir. Edit seçeneği ile satır güncelleme moduna alınıp güncelleme işlemi yapılabilir ve Delete seçeneği ile de kayıt silinebilir. Select seçeneği kullanılarak da satırın seçilmesi sağlanmaktadır.

GridView Sütunları

GridView kontrolünün sütunları otomatik oluşturulabileceği gibi yazılım geliştiriciler tarafından da tanımlanabilir. Bir önceki örnek göz önüne getirildiğinde, bu alanda RegionID değerinin görüntülenmesine gerek yoktur. Çünkü; son kullanıcının ilgilenecek olduğu alan RegionDescription sütunudur. RegionID'yi GridView'in sütunları arasından çıkartmak için GridView sütunlarının, otomatik olarak oluşturulması iptal edilip, sütunlar geliştirici tarafından eklenmelidir. Bu işlem için GridView'in SmartTag'indeki "Edit Columns.." bağlantısı kullanılabilir. Edit Columns bağlantısı tıklanıldıktan sonra GridView'de kullanılabilecek olan alanların listelendiği bir ekran ile karşılaşılacaktır. Bu ekran yardımı ile var olan alanlar kaldırılıp yenileri eklenebilir.



GridView'in alanlarının listelenmiş olduğu yukarıdaki resimde görülen ekranın, sol altında bulunan **"Selected fields"** bölümü, görüntülenen alanları listeler. Bu bölümde bulunan alanların özellikleri de sağ tarafta bulunan özellikler bölümünden ayarlanabilir. Selected fields bölümünün hemen sağ tarafında bulunan butonlar yardımı ile bu bölümdeki alanlar kaldırılabilir ya da sıralaması değiştirilebilir. **"Available fields"** bölümünde kullanılabilecek olan mevcut sütunlar listelenmektedir. İstenilen sütun GridView'e eklenip kullanılabilir. Bu alanda kullanılabilecek olan alanlar şunlardır;

BoundField: BoundField ile kontrole bağlanan veri kaynağından gelen veri metin olarak görüntülenebilir. BoundField GridView'e eklendikten sonra Data bölümünden DataField ile görüntülenmek istenilen sütun belirtilerek görüntülenebilir.

CheckBoxField: İstenilen veriyi CheckBox olarak görüntüleyecek olan alandır.

HyperLinkField: Görüntülenecek olan veriyi link olarak gösterebilecek olan alandır. Bu alanda veritabanından direkt gelen linkler kullanılabileceği gibi, eğer istenilirse dinamik olarak linkler de oluşturulabilir.

ImageField: URL'i veritabanında depolanan bir imajı görüntülemek için ImageField kullanılabilir.

ButtonField: Görüntülenen veriler arasında Button gösterilip üzerinde tıklanıldığında bir işlem yaptırılmak isteniyorsa, ButtonField tercih edilebilir. Button'un Text özelliği veritabanındaki bilgilerden getirilip görüntülenebilir.

CommandField: Görüntülenen veriler üzerinde güncelleme, silme, seçme gibi işlemler yapılacağı zaman CommandField kullanılmalıdır. CommandField'ler her satırın yanında görüntülenerek o satır üzerinde işlem yapılmasına olanak tanır.

TemplateField: GridView'de veriler genellikle tek satırdan oluşur ve her hücrede tek bir veri gösterilir. GridView'in görünümü özelleştirilmek isteniyorsa, TemplateField kullanılarak istenilen özelleştirme

gerçekleştirilebilir. Veri görüntülerken, BoundField'da verinin güncellenebileceği alanlar otomatik olarak oluşturulurken, TemplateField'da veri güncellemek için gerekli olan alanlar, DataList'te olduğu gibi oluşturulmalıdır.

Örneğe geri dönecek olursak, bu alanda gerçekleştirilmek istenen işlem, hatırlanacağı üzere RegionID değerinin listelenmesini engellemektir. İstenen işlemin gerçekleştirilmesi için "Selected fields" bölümünden RegionID üzerinde gelinir ve alan silinip OK tuşuna basılarak listelenen sütunlar arasından, RegionID kaldırılır. Listelenen alanlara yeni bir alan eklemek istendiğinde de yine Edit Columns bağlantısı ile tüm alanların listelenmiş olduğu alana geçilmelidir ve eklenmek istenen sütun seçildikten sonra gerekli ayarlar gerçekleştirilip OK tuşuna basılmalıdır.

GridView üzerindeki bir satırın güncellenme işlemi yapılmak istenildiğinde şöyle bir senaryo düşünelim; GridView üzerindeki satırda RegionID'yi de barındıran bir link bulunsun ve güncelleme yapılmak istenen sayfaya kullanıcıyı yönlendiriyor olsun. Güncelleme sayfasında da, güncelleme işlemi gerçekleştirilip, kayıtların listelendiği sayfaya yönlendiriliyor olsun. Böyle bir senaryoyu gerçekleştirmek için bir adet sayfaya daha ihtiyaç vardır. Projeye **RegionGuncelle.aspx** adında yeni bir WebForm ekleyelim ve içine de bir tane FormView kontrolü ilave edip, FormView kontrolüne bir adet SqlDataSource kontrolü bağlayarak, üzerinde çalışılan Region tablosunun güncellenmesini sağlayalım. FormView kontrolüne, SqlDataSource daha önceki örneklerdeki gibi bağlanmalıdır. Ancak, bu alanda ilave olarak, bir de WHERE bölümü yer almalıdır. Sorguya WHERE bölümünü eklemek için veri kaynağının ayarları yapılırken, tablonun seçildiği bölümden WHERE butonu tıklanmalıdır. WHERE butonu tıklandıktan sonra aşağıdaki ekran ile karşılaşılacaktır.

Add WHERE Clause

Add one or more conditions to the WHERE clause for the statement. For each condition you can specify either a literal value or a parameterized value. Parameterized values get their values at runtime based on their properties.

Column: RegionID

Operator: =

Source: QueryString

SQL Expression: [RegionID] = @RegionID

Parameter properties

QueryString field: RegionID

Default value:

Value: Request.QueryString("RegionID")

Add

WHERE clause:

SQL Expression	Value

Remove

OK Cancel

WHERE tümcesi oluşturulurken sütun olarak "RegionID", Operatör olarak "=" ve RegionID'nin gelecek olduğu kaynak da QueryString olarak belirtiliyor. QueryString'in alanı da, RegionID olarak ayarlanıp Add butonu ile bir tane WHERE tümcesi oluşturuluyor. Ardından, OK tuşuna basılarak oluşturulan tümce sorguya ekleniyor. SQL sorgusu hazırlandıktan sonra sorgu test edilip FormView'in şablonları oluşturuluyor. RegionGuncelle.aspx sayfası, adından da anlaşılacağı üzere sadece güncelleme yapılabilecek bir sayfadır. Dolayısı ile bu sayfada yer alan FormView'de ItemTemplate ve InsertItemTemplate olmasına gerek yoktur. Sadece EditItemTemplate

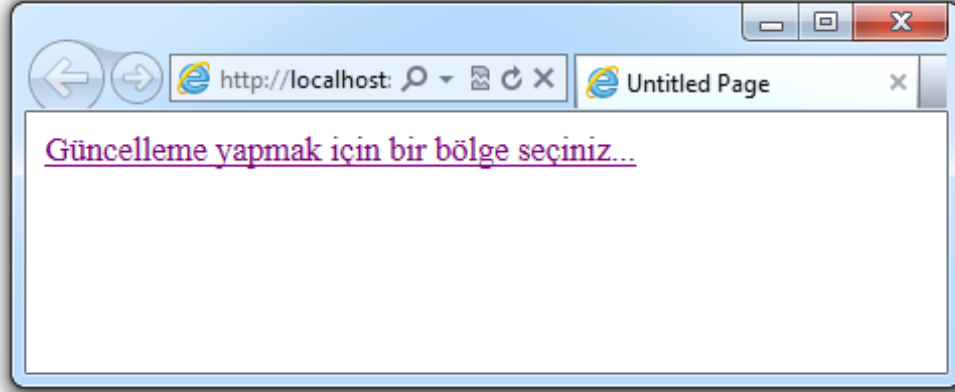
yeterli olacaktır. Kullanıcıların bu sayfaya direkt veya yanlış parametre ile gelebilecekleri de düşünülerek EmptyDataTemplate'te doldurulmalıdır. EmptyDataTemplate içerisine bir adet HyperLink kontrolü eklenip listelenecek kayıt olamadığını belirtilerek, Region'ların listelendiği diğer sayfaya bir link verilmesi uygun olacaktır. EditItemTemplate içerisinde de RegionID değerinin güncellenmesi engellenecek şekilde, verinin Label içerisinde görüntülenmesini sağlayalım. Hemen ardından, güncelleme sayfasında, üzerinde çalışılan verinin silinmesini de gerçekleştirebilecek şekilde bir adet CommandName özelliği Delete olarak ayarlanmış LinkButton eklenmelidir. FormView'in sadece EditItemTemplate'i bulunduğu için **DefaultMode** özelliği **Edit** olarak ayarlanmalıdır. Gerekli ayarlamalar yapıldıktan sonra FormView'in kodları aşağıdaki gibi olmalıdır.



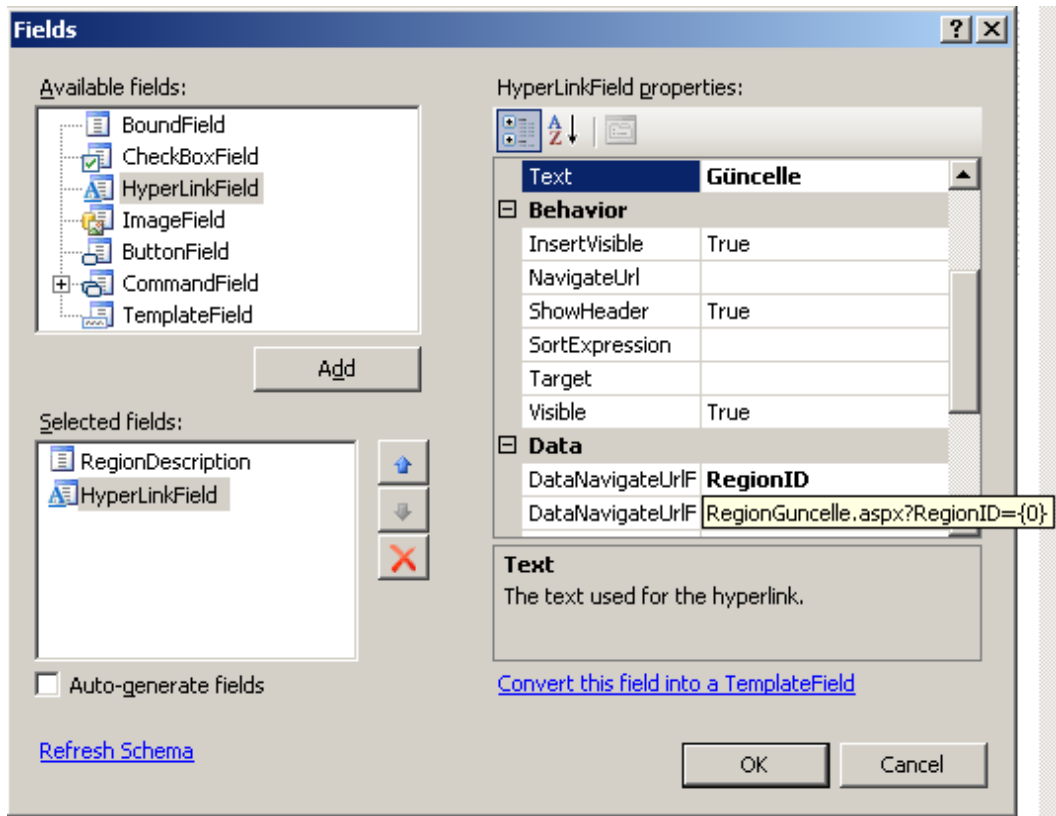
QueryString, adres çubuğu üzerinden, sayfalar arası veri taşımaya olanak tanıyan durum yönetimi tekniklerinden birisidir. QueryString tekniği ve kullanımı, Durum Yönetimi bölümünde detaylı bir şekilde ele alınmaktadır.

```
<asp:FormView ID="FormView1" runat="server" DataKeyNames="RegionID"
DataSourceID="SqlDataSource1" DefaultMode="Edit">
  <EditItemTemplate>
    RegionID:
    <asp:Label ID="RegionIDLabel1" runat="server" Text='<%#
Eval("RegionID") %>' />
    <br />
    RegionDescription:
    <asp:TextBox ID="RegionDescriptionTextBox" runat="server"
      Text='<%# Bind("RegionDescription") %>' />
    <br />
    <asp:LinkButton ID="UpdateButton" runat="server"
CausesValidation="True"
      CommandName="Update" Text="Güncelle" />
    &nbsp;<asp:LinkButton ID="LinkButton1" runat="server"
CommandName="Delete">Sil</asp:LinkButton>
    &nbsp;<asp:HyperLink ID="HyperLink1" runat="server"
      NavigateUrl="~/GridViewKullanimi.aspx">Tüm
Bölgeler</asp:HyperLink>
  </EditItemTemplate>
  <EmptyDataTemplate>
    <asp:HyperLink ID="HyperLink2" runat="server"
      NavigateUrl="~/GridViewKullanimi.aspx">Güncelleme yapmak için
    bir bölge
    seçiniz...</asp:HyperLink>
  </EmptyDataTemplate>
</asp:FormView>
```

FormView'in ayarları gerçekleştirildikten sonra RegionGuncelle.aspx çalıştırıldığında aşağıdaki görüntü ile karşılaşılacaktır. Dolayısı ile bu sayfaya mutlaka geçerli bir parametre ile gelinmelidir. Sayfaya geçerli bir parametre ile gelmek için bölgelerin listelendiği GridView'e bir adet HyperLink sütunu ekleyip gerekli ayarları gerçekleştireceğiz.

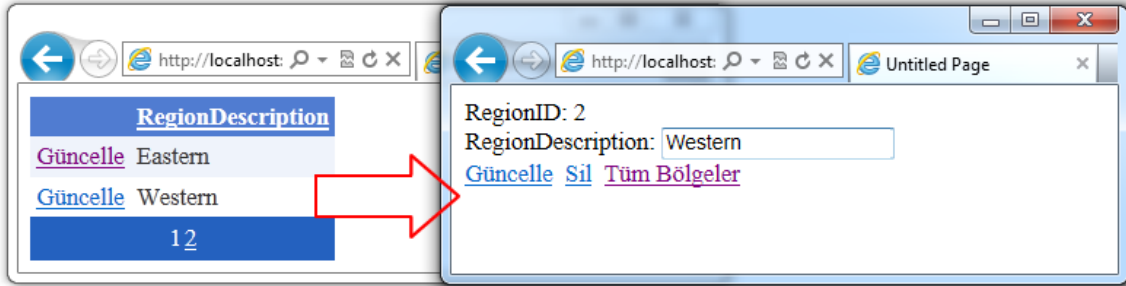


GridView'e yeni bir sütun eklemek için SmartTag'inden "Edit columns.." bağlantısı ile GridView'de kullanılabilir alanların listelendiği ekrana geçiş yapalım. Eklenebilecek alanlar arasında bir adet HyperLink alanı ekleyelim ve ayarlarını aşağıdaki resimde görüldüğü gibi yapalım. Veri güncellemek için başka bir sayfaya geçileceğinden dolayı, GridView'in alanları arasında olan CommandField'e artık gerek yoktur. Bu alanı da listeden çıkaralım.



HyperLinkField'in Text özelliği Güncelle olarak ayarlandı ve DataNavigateUrlFields alanı "**RegionID**" olarak ve DataNavigateUrlFormatString alanı da "**RegionGuncelle.aspx=RegionID={0}**" olarak güncelleme sayfasına RegionID parametresini gönderecek şekilde ayarlandı ve ardından da sıralaması ilk sütunda görüntülenecek şekilde değiştirildi. Sonrasında, OK tuşuna basılarak, yapılan bu ayarlar kaydedildi. Gerçekleştirilen son ayarların ardından uygulama çalıştırıldığında güncelleme işlemi farklı bir sayfa üzerinden gerçekleştiriliyor olacaktır. Şu andaki sadece bir sütunun güncellendiği örnekte, ikinci bir sayfaya geçmek yerine güncelleme işlemi GridView

üzerinden yapmak daha mantıklı olacaktır ama çok sütünlü sayfalarda ikinci bir sayfa ekliyor olmak verilerin daha hoş görüntülenmesini sağlayacaktır ve kullanıcı memnuniyetini yükseltecektir.



GridView'in Olayları

GridView'de de diğer kontrollerde olduğu gibi GridView'e özel olaylar vardır. Bunlar, tetiklendikten sonra yakalanıp ele alındığında, oldukça kullanışlı bir takım işlemler gerçekleştirilebilir. GridView'in olaylarına da her kontrolde olduğu gibi özelliklerinden erişilebilir. Herhangi bir GridView'in olaylarına göz atıldığında FormView'deki metodlara benzer şekilde RowUpdated ve RowUpdating gibi olaylar dikkat çekmektedir. GridView içerisinde ayrıca her satıra veri bağlandıkran sonra tetiklenen ve satırlar üzerinde işlem yapılmasını sağlayan olaylar da yer almaktadır. Bu olaylardan RowDataBound kullanılarak güzel bir örnek tasarlanabilir. GridView kullanımını örnekleyen ilk örnek hatırlandığında AdventureWorks veritabanındaki Product tablosunda bulunan veriler görüntülenmişti. Hatırlanacağı üzere ürünün ID'si, adı ve fiyatı listelenmekteydi. Fiyat alanındaki değere bakılıp, değer 200'den küçükse arka plan rengi kırmızı, 1000'den büyükse de arka plan renginin turkuaz olması sağlanabilir. Düşünülen bu senaryo için RowDataBound olayı ele alınmalıdır. RowDataBound olayı tetiklendiğinde çalışacak olan olay aşağıda yer almaktadır.

```
protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        if (double.Parse(e.Row.Cells[2].Text) < 200)
        {
            e.Row.BackColor = System.Drawing.Color.Red;
            e.Row.ForeColor = System.Drawing.Color.White;
        }
        else if (double.Parse(e.Row.Cells[2].Text) > 1000)
        {
            e.Row.BackColor = System.Drawing.Color.Turquoise;
            e.Row.ForeColor = System.Drawing.Color.Black;
        }
    }
}
```

Yukarıdaki kodlar incelendiğinde ilk olarak üzerinde çalışılan kaydın tipi kontrol ediliyor. Satırın tipi DataControlRowType tipinde bir numaralandırıcı ile belirtilmektedir ve dolayısı ile karşılaştırılırken de bahsedilen numaralandırıcı üzerinden karşılaştırma yapılıyor. Satır tipi başlık gibi veri taşımayan bir satır da olabilir bu şekilde veri taşımayan satırlarda Parse işlemi hata üreteceği için üzerinde çalışılacak olan satırın mutlaka veri taşıyan bir satır olması sağlanmalıdır. İkinci olarak olayın yakalanmasındaki asıl amaç olan ListPrice alanındaki değerin double tipine dönüştürülüp belirtilen değerler ile karşılaştırılması ve satırların arka plan renklerinin değişmesi sağlanmaktadır.

DetailsView

ASP.NET 2.0 ile birlikte gelen DetailsView kontrolü, adından da anlaşılacağı üzere listelenen verinin detayını göstermek amacı ile kullanılmaktadır. DetailsView kontrolü uygulamalarda genellikle GridView ile kullanılıp GridView'de görüntülenen verilerin detaylarını göstermek amacı ile kullanılmaktadır. Kontrolün veri görüntülemek dışında veri güncellemek, silmek ve veri eklemek gibi yetenekleri de söz konusudur. GridView ile entegre olarak çalışabilecek şekilde bir örnek yaparak DetailsView kontrolünün kullanımını örnekleyelim. Bahsedilen işlemi gerçekleştirmek için yeni bir tane WebForm'a bir adet GridView ekleyelim ve GridView'e bir adet SqlDataSource kontrolü bağlanarak Northwind veritabanındaki Products tablosunda yer alan ProductID ve ProductName sütunlarını seçelim. GridView kontrolünü sayfalama yapacak ve seçim yapılacak şekilde ayarlayalım. Ardından, Auto Format bölümünden Classic seçerek, daha hoş bir görünüm oluşturmayı ihmal etmeyelim. ProductID sütununu kullanıcılara göstermenin de bir anlamı olmayacaktır. Bu sebeple, SmartTag'den "Edit columns.." bağlantısı aracılığı ile ProductID alanını kaldıralım. Sayfa çalışırıldığında verilerin onar onar listelendiği farkedilecektir. Şimdi sıra geldi seçilen verinin detayını göstermeye.

Seçilen verinin detayı DetailsView kontrolünde görüntülenecektir. Sayfaya bir tane DetailsView kontrolü ekleyip SmartTag'inden yeni bir tane veri kaynağı kontrolü oluşturalım. Oluşturulan yeni veri kaynağı kontrolü de SqlDataSource olacaktır. Bu kontrol, Northwind'deki Products tablosundaki verilerin tamamını getirecektir ancak sadece GridView'de seçilen satırı getirmesi beklenmektedir. Dolayısıyla, bir adet WHERE tümcesine ihtiyaç vardır. Bu sebeple, tablo seçim ekranında WHERE butonuna basılıp aşağıdaki resimde görülen ayarlar ile GridView'de seçilen satırdaki verilerin getirilmesi sağlanır.

Resimden de görüleceği gibi koşul olarak CategoryID'si sayfada yer alan GridView1 isimli kontrolün seçili olan değerine eşit olan veri DetailsView kontrolünde listenecek şekilde bir WHERE tümcesi ayarlandı. Resimde görülen ekranda, ilk olarak Add butonu ile WHERE tümcesi oluşturulur. Ardından, OK ile SQL sorgusuna eklenir. Sorgu oluşturulduktan sonra test edilerek, DetailsView kontrolünün şablonları da oluşturulur. Daha sonra, uygulama çalıştırılabilir. Ancak, daha güzel bir görünüm için, ilk olarak DetailsView kontrolünün SmartTag'inden

Auto Format bağlantısı aracılığı ile Classic seçilir. Ardından da görüntülenecek veri olmaması durumunda kullanıcıya bilgi vermek amacı ile EmptyDataTemplate içerisine “Detaylarını görüntülemek istediğiniz ürünü seçiniz...” metni eklenir. Bu ayarlamalar sonrasında, uygulama çalıştırıldığında aşağıdaki görünüm elde edilebilecektir.

The screenshot shows a web browser window with the address bar displaying 'http://localhost:'. The page is titled 'Untitled Page'. The main content area is divided into two sections. On the left, there is a list of products, each preceded by a 'Select' link. The products are: Alice Mutton, Aniseed Syrup, Boston Crab Meat, Camembert Pierrot, Carnarvon Tigers, Chai, Chang, Chartreuse verte, Chef Anton's Cajun Seasoning, and Chef Anton's Gumbo Mix. Below the list is a blue bar with the numbers 1 through 8. On the right, there is a details section for the selected product, Alice Mutton. The details are as follows:

ProductID	17
ProductName	Alice Mutton
SupplierID	7
CategoryID	6
QuantityPerUnit	20 - 1 kg tins
UnitPrice	39,0000
UnitsInStock	0
UnitsOnOrder	0
ReorderLevel	0
Discontinued	<input checked="" type="checkbox"/>

EĞİTİM :

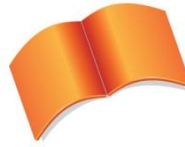
**VERİYE ERIŞİM VE
NAVİGASYON**

Bölüm :

MasterPage ve Navigasyon

Konu :

MasterPage



Microsoft Türkiye

Açık Akademi

MasterPage

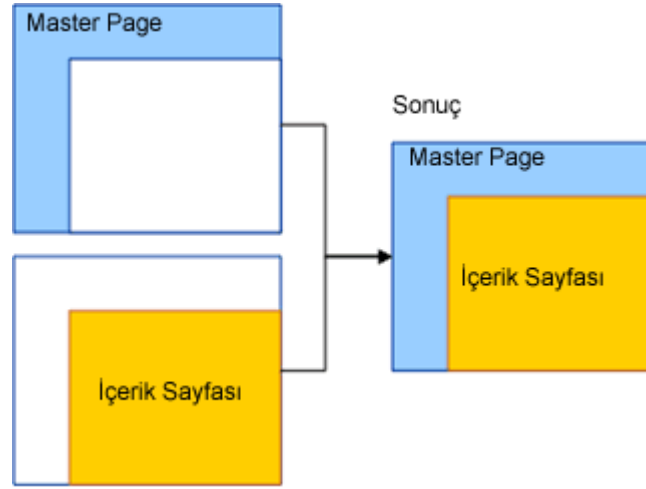
Bir web sitesindeki sayfalara dikkat edildiğinde sayfaların tasarımlarının birbirlerine benzediği hatta pek çok yerinin aynı olduğu dikkat çekmektedir. Aynı olan alanlar da genellikle sitenin logosu ve menü gibi alanlar olmaktadır. Menü ve logo gibi alanların dışında sadece içerik değiştirilerek yeni sayfalar oluşturulmaktadır. Beş on sayfalık küçük siteler için böyle bir tasarımı gerçekleştirmek çok zor olmamaktadır fakat daha profesyonel ve daha fazla sayfadan oluşan siteler için tasarım yapmak problem haline gelecektir. Bir tane boş şablon oluşturup şablonun HTML bilgisini kopyalayıp yeni sayfa oluşturmak, sayfa tasarım problemini çözebilen oldukça kullanışlı bir yöntemdir ki yıllar önce sayfa tasarımı bu şekilde yapılmaktaydı. Şablonu kopyalayıp yapıştırmak, oldukça hızlı ve kullanışlı bir çözüm olmakla beraber, aslında pek çok problemi de yanında getirmektedir. Bu şekilde tasarım yapmak oldukça kolaydır ancak bu sefer de güncelleme yapmak çok zor bir hale gelir. Yaklaşık yüz sayfadan oluşan bir şirket sitesinin arka plan rengi değiştirilmek istendiğinde, her sayfa teker teker açılıp güncellenecek mi? Eğer CSS (Cascading Style Sheets) kullanılmadıysa bu sorunun cevabı ne yazık ki Evet! Ancak CSS kullanarak sitenin görünümlerini belirleyecek olan tasarım kodları, .css uzantılı ayrı bir dosyaya koyulup sitede bulunan tüm sayfalara CSS dosyası uygulandıysa sitede bulunan ve o CSS dosyasını kullanan tüm sayfaların stil ayarları tek bir yerden yönetilebilir dolayısıyla sitenin sitili oldukça kolay bir şekilde güncellenebilir.

Sitelerin sitelerinin güncellenmesi CSS dosyaları ile sağlanabiliyor. Peki; menü ve resim gibi tüm sayfalarda yer alan içerikler değişirse ne olacak? Bu soruna da ASP ve PHP’de include dosyaları ile çözüm bulunmuştur. Sitenin belli bölümleri ayrı sayfalarda tutularak ve bu sayfalar, diğer sayfaların içine gömülerek ortak içeriğin de tek bir yerden yönetilmesi sağlanmıştır. Include dosyaları oldukça kullanışlı gibi görünmesine rağmen büyük sitelerde dosya sayısının artması, ve include dosyaları içerisine de başka sayfaların gömülmesi ile birlikte yönetim oldukça zorlaşmış ve içinden çıkılmaz bir hal almıştır.

ASP.NET yazılım geliştiricilere pek çok yenilik sunarken içerik yönetimini de ihmal etmemiştir. ASP.NET’in 1.0 ve 1.1 versiyonlarında ortak içerik problemi **User Controls** (Kullanıcı Kontrolleri) adı verilen yapılar ile çözülmüştür. User Control’ler kullanım itibarı ile include dosyalarına benzemekle beraber sunmuş olduğu programatik erişim imkanları ile birlikte include dosyalarından çok daha esnek ve kullanışlıdır. ASP.NET’in eski versiyonlarında ortak içerik problemi için tek çözüm User Control’lerdi ancak ASP.NET 2.0 ve sonrasında, yazılım geliştiricilerin hayatına **MasterPage** adı verilen tasarımın **kalıtımla** diğer sayfalara aktarılabilirdiği yapılar girdi. MasterPage’ler ile birlikte ilk olarak sitenin tasarımı MasterPage’de gerçekleştiriliyor. Tasarım belirlenirken tüm sayfalarda sabit kalması gereken ve her sayfada değişmesi gereken alanlar, ilerleyen sayfalarda açıklanan teknikler ile belirleniyor ve siteye yeni eklenen sayfa MasterPage’den kalıtılarak tasarım yeni sayfaya uygulanmış oluyor. Tasarımın kalıtılabilir olması ile birlikte yukarıda bahsedilen tüm sorunlar tek çırpıda çözülmüş oluyor. Tüm sitede ortak olarak görüntülenen bir alanı MasterPage’den değiştirerek o MasterPage’den türeyen tüm sayfalarda da aynı değişikliğin gerçekleştirilmesi sağlanmış oluyor.

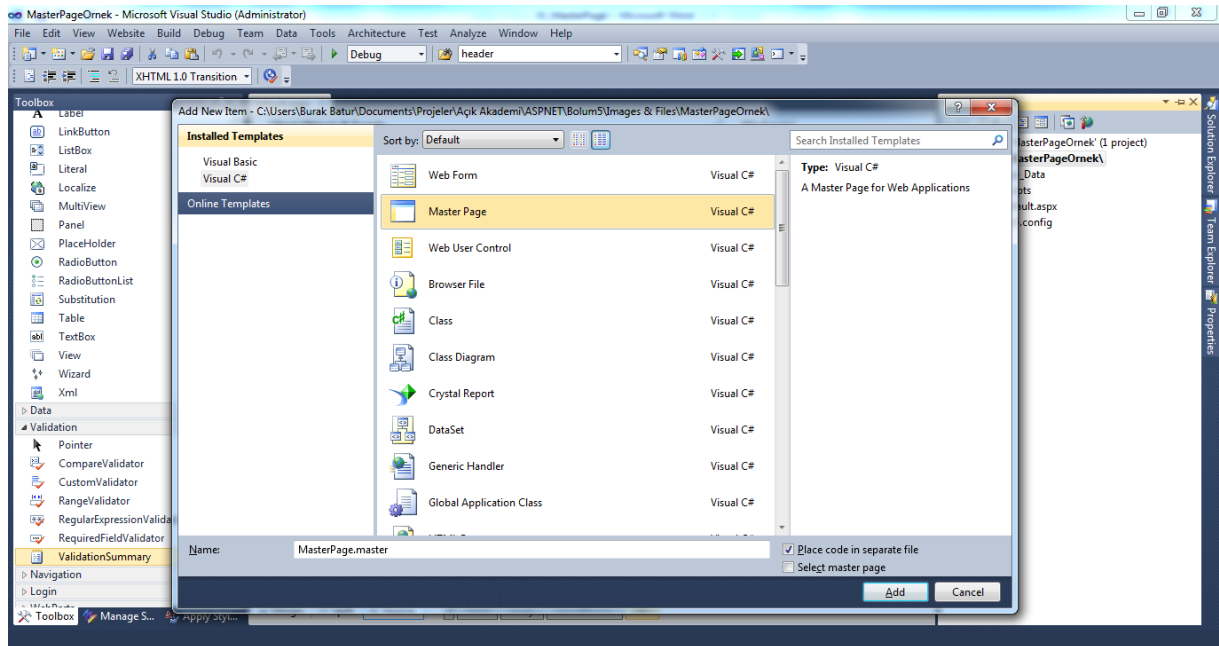
MasterPage’ler standart ASP.NET sayfalarına çok benzemektedir aslında aynıdır da denilebilir. Standart bir ASP.NET sayfasının uzantısı .aspx iken MasterPage’lerin uzantısı **.master**’dır, ASP.NET sayfaları @Page direktifi ile başlarken MasterPage’ler **@Master** direktifi ile başlar ve sayfanın genel ayarları @Master direktifinden belirlenir. Görüldüğü gibi standart bir ASP.NET sayfası ile MasterPage’ler arasında iki temel fark vardır. Dolayısıyla, MasterPage’ler de diğer sayfalar gibi kolaylıkla tasarlanabilir.

MasterPage’ler tarayıcıdan talep edildiğinde son kullanıcıya gösterilmezler. Bir MasterPage’in içeriği son kullanıcıya gösterilmek isteniliyorsa bu MasterPage’den bir adet sayfa türetilmeli ve tarayıcıdan türeyen sayfa talep edilmelidir. MasterPage’den türeyen sayfa kullanıcı tarafından talep edildiğinde aşağıdaki resimde görüldüğü gibi MasterPage’in ve türeyen sayfanın içerikleri birleştirilerek son kullanıcıya gösterilir.



MasterPage'lerden türetilen .aspx uzantılı sayfalara **ContentPage** ismi verilmektedir.

Visual Studio ortamında bir web sitesi projesine MasterPage eklemek için, yeni öğe ekleme menüsünden MasterPage seçilmelidir. MasterPage eklenirken de tıpkı Web Form'da olduğu gibi programla dili ve kodun ayrı sayfaya koyulması seçimleri de yapılabilmektedir.



ContentPlaceHolder Kontrolü

ContentPlaceHolder kontrolü MasterPage'in içerisine eklenerek MasterPage'lerden türeyen sayfalarda içeriği değiştirilebilecek alanları belirlemek amacı ile kullanılan kontroldür. Siteye yeni bir MasterPage eklendiğinde Visual Studio ortamında iki adet ContentPlaceHolder kontrolü varsayılan olarak MasterPage'e eklenmiş olarak gelir. Bunlardan ilki sayfanın HEAD bölümünde yer alırken ContentPage'lerde sayfaya özgü tanımlamaların yapılmasına olanak tanır. İkincisi ise BODY bölümünde yer alarak sayfaya özgü içeriğin eklenebilmesine olanak tanır. Siteye eklenen yeni bir MasterPage'in yapısı aşağıdaki kod bloğunda görüldüğü gibi olmaktadır.

```

<%@ Master Language="C#"
AutoEventWireup="true" CodeFile="Ana.master.cs" Inherits="Ana" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Ana</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
            </asp:ContentPlaceHolder>
        </div>
    </form>
</body>
</html>

```

Kodları incelediğinde sayfanın @Master drektifi ile başladığı fark etmiş olmalısın. Biraz daha alta göz atıldığında HEAD bölümünde yer alan ContentPlaceHolder kontrolü dikkat çekecektir. BODY bölümüne geçildiğinde ise ilk dikkat çeken şey form etiketidir. Hatırlayacağın üzere, ASP.NET sayfalarında sadece bir tane sunucu tarafında çalışan form bulunabilmekteydi. MasterPage içerisinde bir adet sunucu tarafında çalışan form bulunduğuna göre, bu; MasterPage'lerden türeyen sayfalarda form olmayacağı anlamına gelmektedir. Çünkü; buradaki form kalımla alt sayfalara aktarılıyor olacaktır. Form etiketinin içerisinde de diğer ContentPlaceHolder kontrolü yer alıyor.

MasterPage tasarımı yapılırken sabit kalacak olan alanlar, direkt olarak sayfaya eklenirken ContentPage'lerde (Yani; MasterPage'den türeyen sayfalarda) değiştirilebilecek olan alanlar ContentPlaceHolder kontrolleri ile belirlenmelidir. Bir logo, bir menü ve bir içerik bölümünden oluşan bir sayfa düşünüldüğünde böyle bir tasarımı gerçeklemek için, logo ve menü direkt olarak sayfaya eklenirken, içerik için belirlenen alan ContentPlaceHolder kontrolü ile tanımlanacaktır. Bir örnekle konunun daha iyi anlaşılmasını sağlayalım. Yukarıdaki cümlede bahsettiğimiz sayfa tasarımını, Visual Studio üzerinde oluşturduğumuz MasterPage'e uygulayalım. MasterPage'in en üstünde ve sol tarafında tüm sayfalarda sabit olarak kalacak bir metin alanı belirleyelim. Ardından, sağ bölümün büyükçe bir alanının ContentPage ile de düzenlenebilmesini sağlayalım. Bahsettiğimiz örneğin Visual Studio ortamındaki tasarımı aşağıdaki şekilde olacaktır.



Resimden görüleceği üzere MasterPage'in içerisinde bir adet tablo vardır. Tablonun en üst satırında, iki satıra yayılacak şekilde logo ekleyebilirsiniz. Ya da bu alanı sık sık değişecek bir banner alanı olarak kullanabilirsiniz. Alttaki satırın sol tarafında, menü için bir alan ve menünün altına da bir tane ContentPlaceHolder kontrolü yer almaktadır. Alttaki satırın sağ tarafında ise, bir tane ContentPlaceHolder kontrolü yer almaktadır. Bu kontrolün amacı ContentPage'lerin içeriklerini bu alanda görüntülemektir. Yukarıdaki resimde dikkat çeken bir durum daha vardır. Sağ taraftaki ContentPlaceHolder kontrolünün içinde herhangi bir şey yer almazken, sol tarafta bir adet resim yer almaktadır. ContentPlaceHolder'ın bu şekilde varsayılan olarak içinde yer alan içeriğe, varsayılan içerik denir. ContentPage'lerde eğer istenilirse ContentPlaceHolder'ların içerikleri görüntülenebilir. Ancak, tabi ki buradaki içerik yok sayılıp sayfaya özgü içerik de eklenebilir. Oluşturduğumuz MasterPage'in HTML kodları aşağıdaki gibi olmalıdır.

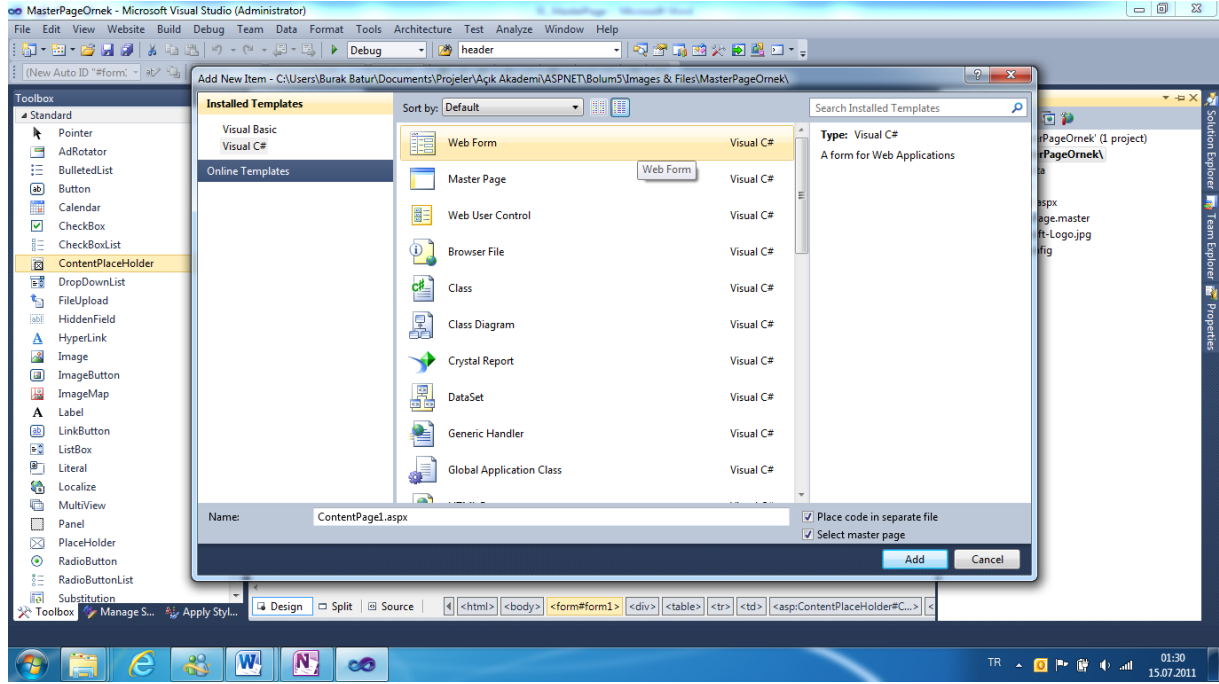
```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

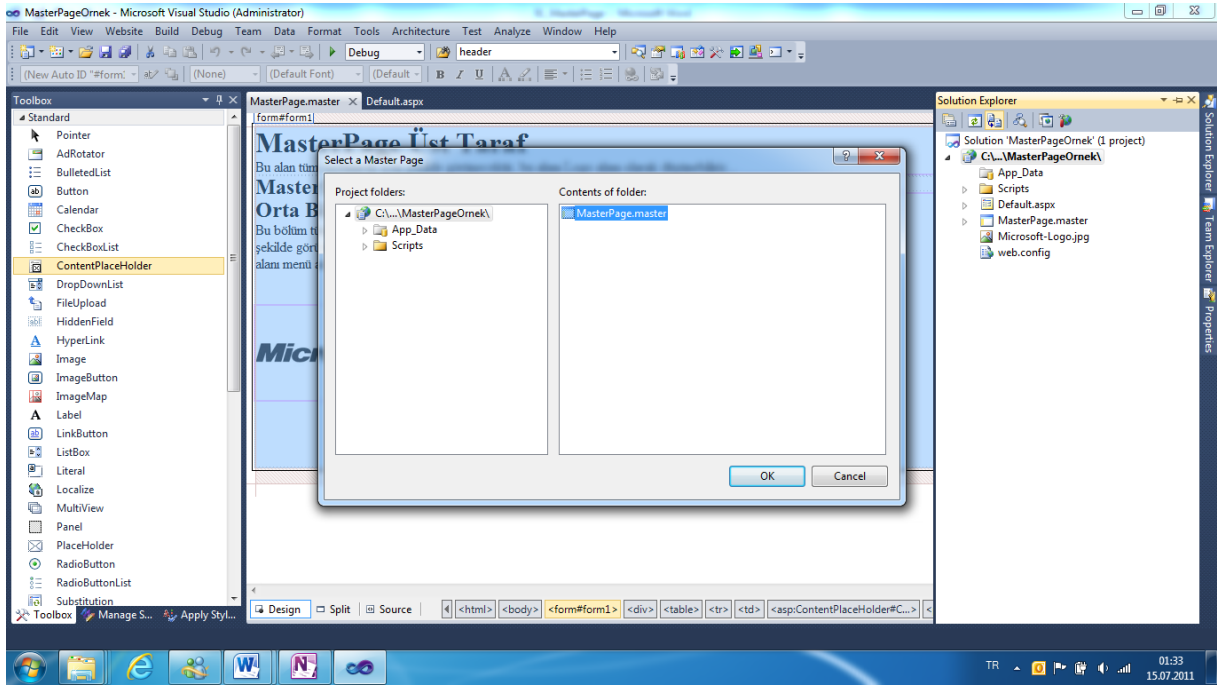
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <table cellpadding="0" cellspacing="0" style="width:800px">
                <tr>
                    <td colspan="2">
                        <strong><span style="font-size: xx-large">MasterPage Üst
Taraf</span></strong><br />
                        Bu alan tüm sayfalarda aynı şekilde görünecektir, bu alanı Logo
alanı olarak
                        düşünebiliriz...</td>
                    </tr>
                    <tr>
                        <td style="width:200px; vertical-align:top;">
                            <strong><span style="font-size: x-large">MasterPage Sol Orta
Bölüm</span></strong><br />
                            Bu bölüm tüm sayfalarda aynı şekilde görüntülenecektir. Bu
alanı menü alanı gibi
                            düşünebiliriz.<br />
                            <asp:ContentPlaceHolder ID="ContentPlaceHolderSol"
runat="server">
                                <p>
                                    </p>
                                </asp:ContentPlaceHolder>
                                <br />
                                <br />
                            </td>
                            <td style="width:600px; vertical-align:top;">
                                <asp:ContentPlaceHolder id="ContentPlaceHolderSag" runat="server">
                                </asp:ContentPlaceHolder>
                            </td>
                        </tr>
                    </table>
                </div>
            </form>
        </body>
    </html>
```

MasterPage'lerden Sayfa Türetmek

Bir MasterPage'den başka bir sayfa türetilmek istenildiğinde Visual Studio'nun nimetlerinden faydalanılabilir. Visual Studio ortamında, bir web sitesi projesine yeni bir sayfa eklemek istenildiğinde, projeye eklenilebilecek öğelerin listelendiği ekrandan bir web sayfası seçildiğinde, **“Select Master Page”** kutucuğu aktif hale gelecektir. Şimdi bu işlemi birlikte gerçekleştirelim. Solution Explorer üzerinde projeye sağ tıklayıp **“Add New Item”** seçeneğini seçelim. Yeni öğe ekleme ekranından WebPage seçeneğini seçip alt tarafta da **“Select Master Page”** kutucuğunu işaretleyelim. Gerçekleştirmemiz gereken işlemi aşağıdaki resimde görüyorsun.



“Select Master Page” işaretlenip, sayfaya bir isim de verildikten sonra, Add butonuna basalım. Bu adımdan sonra Visual Studio bizi MasterPage'lerin listelendiği bir ekrana yönlendirip sayfanın MasterPage'ini belirtmemizi sağlayacaktır. Aşağıdaki ekran görüntüsünde de gördüğün gibi gerekli Masterpage'i seçip, OK tuşuna bastıktan sonra, yeni sayfamız projeye eklenecektir. Aşağıdaki erkanda bir tane MasterPage var ancak bir projede istediğin kadar MasterPage ekleyebileceğini hatırlatmakta fayda var.



MasterPage’den sayfa türetmenin bir diğer yolu da MasterPage üzerindeyken sağ tıklayıp, menüden “**Add Content Page**” seçeneğini seçmektir. MasterPage’in üzerindeyken “Add Content Page” seçeneği seçildiğinde projeye, üzerinde sağ tıklanan MasterPage’ten türeyen yeni bir sayfa eklenecektir.

MasterPage’ten türeyen sayfanın içeriği aşağıdaki kod bloğunda görüldüğü gibi olacaktır. Visual Studio, varsayılan olarak tüm ContentPlaceHolder sayısı kadar **Content** kontrolü oluşturacak ve oluşturulan bu Content’leri MasterPage’deki ContentPlaceHolder kontrollerine bağlayacaktır. Content kontrolü ContentPlaceHolder kontrolüne bağlanıp, ContentPlaceHolder’ın yerine sayfaya içerik eklemek için kullanılacak olan kontroldür.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.master"
AutoEventWireup="true" CodeFile="ContentPage1.aspx.cs" Inherits="ContentPage1" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolderSol"
Runat="Server">
</asp:Content>

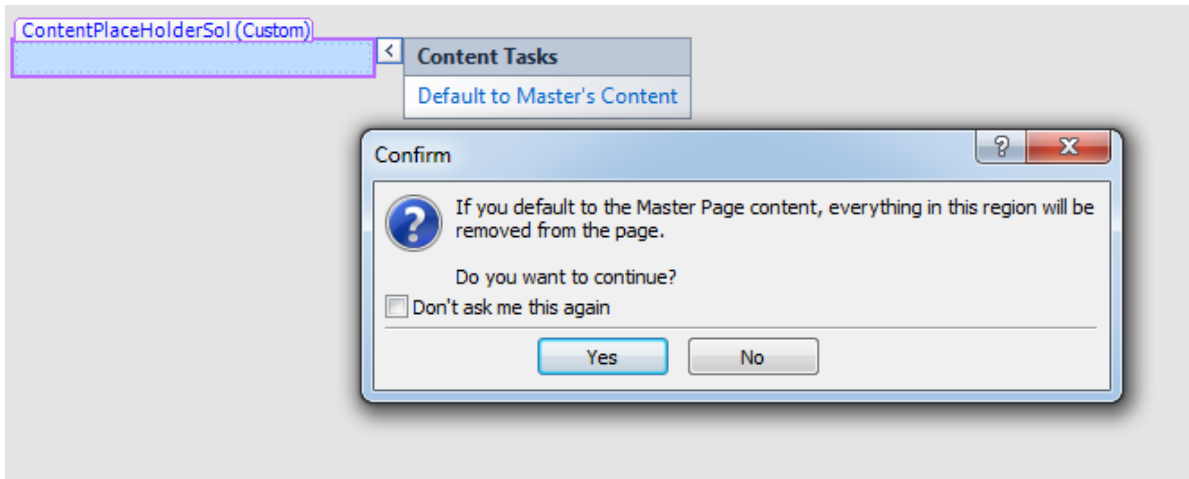
<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolderSag"
Runat="Server">
</asp:Content>
```

ContentPage’deki kodlar incelendiğinde ilk olarak Page direktifinde sayfanın MasterPage’inin belirtildiği fark edilecektir. Dikkat çeken bir diğer nokta da daha önce bahsedildiği gibi sayfada form etiketinin olmamasıdır. Çünkü; MasterPage’deki form, kalımla bu sayfaya da geçmiştir. Sayfada toplam üç adet **Content** kontrolü yer almaktadır. Bunların bağlı oldukları ContentPlaceHolder’lar da, **ContentPlaceHolderID** özelliği ile kontrollere bildirilmektedir. ContentPage’in Visual Studio ortamında dizayn bölümündeki görünümü aşağıdaki

resimde yer almaktadır. Gördüğün üzere ContentPlaceholder ile belirtilen alanlar dışındaki alanlar MasterPage'den aynen ve sadece okunabilir şekilde görüntülenmiştir.

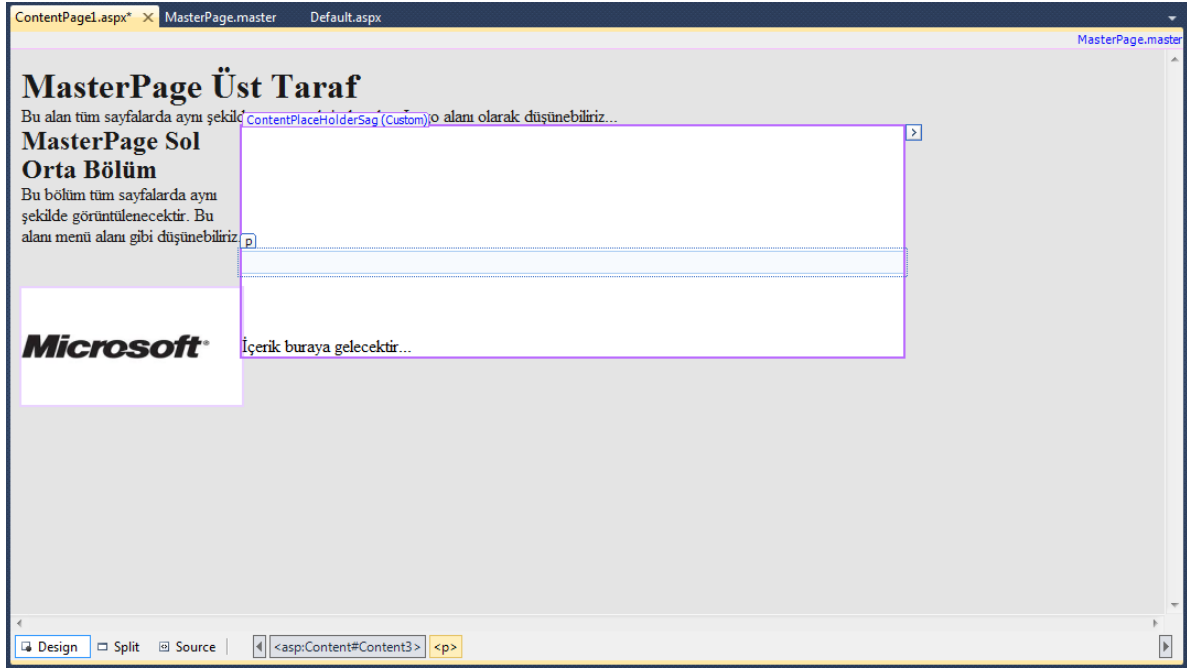


Visual Studio ortamında MasterPage'ten türeyen sayfaların tasarımına, dizayn tarafında da destek verilmektedir. ContentPage'in türemiş olduğu MasterPage sağ üstte gösterilmektedir. Eğer istenirse, MasterPage'in gösterilmiş olduğu linke tıklanıp, MasterPage'in tasarım görünümüne geçilebilir. Bu alanda dikkat çeken bir diğer nokta da HEAD bölümünde yer alan ContentPlaceholder'a bağlanan Content kontrolünün görüntülenmiyor olmasıdır. Bu durum gayet normaldir, çünkü; HEAD bölümü sayfaya özel tanımlamaların ve ayarların yapıldığı bölümdür ve görsel çıktısı yoktur. Oluşturulan Content kontrollerinin SmartTag'lerine tıklandığında, sayfada MasterPage'de bulunan içeriği görüntülemek amacı ile **“Default to Master's Content”** seçeneğinin sunulduğu görülmektedir. Buraya tıklandığında aşağıdaki mesajla karşılaşılacaktır.



Mesajda, MasterPage'de yer alan varsayılan içeriğe dönüldüğünde, ContentPage'de oluşturulan içeriğin silineceğini belirtip bu işlem için onay istenmektedir. Yes tuşuna basıldığında, mevcut sayfada, MasterPage'de ContentPlaceholder içerisinde bulunan resim görüntülenecektir. Daha önceki bölümlerde de belirttiği gibi MasterPage'de bulunan her ContentPlaceholder'a Content bağlamak mecburi değildir. ContentPlaceholder'a

Content bağlanmadığı zaman ContentPlaceholder'ın varsayılan içeriği görüntülenir. Aslında resimde görüntülenen mesaj bunu söylemektedir. Çünkü; Yes tuşuna basıldığında ContentPlaceholderSol'a bağlanan Content kontrolü tüm içeriği ile birlikte silinecektir.



Yukarıdaki ekran görüntüsünde bulunan soldaki resim, MasterPage'den gelmektedir. Sağ tarafta da sayfaya özgü içeriğin eklendiği görülebilir. Sayfa çalıştırıldığında Logo ve menünün ekleneceği bölüm ile, menünün altında bulunan logo MasterPage'den, sağ taraftaki bilgiler ise ContentPage'den geliyor olacaktır. MasterPage'den gelen varsayılan içeriği iptal etmek için ContentPlaceholder'ın SmartTag'inde **"Create Custom Content"** bağlantısına tıklayabilirsiniz. Create Custom Content bağlantısına tıklanıldıktan sonra bir adet Content kontrolü oluşturulup, MasterPage'deki ilgili ContentPlaceholder'a bağlanacaktır. Sayfanın çalışma zamanındaki görüntüsü aşağıda yer almaktadır.



MasterPage'den ContentPage türetilebileceği gibi başka bir MasterPage'de türeyebilir. Yani MasterPage'in de MasterPage'i olabilir. Çok fazla bölümü olan web siteleri geliştirilirken bu modelin tercih edildiği durumlar olabilmektedir. Sitenin genel görünümünü sabitleyecek olan bir MasterPage tanımlanır ve alt bölümlerinde kendi içlerinde tasarım bütünlüğü olması için her alt bölüme özel bir MasterPage tanımlanır ve sitenin genel MasterPage'inden türetilerek tasarım bütünlüğü sağlanmış olur. Bu durum da genellikle en üst düzey MasterPage'de, üstte bir menü yer alır ve tüm sayfalarda aynı menü görüntülenir, alt bölüm MasterPage'lerinde ise sol tarafa bir tane menü eklenip her alt bölümün kendine özel bir sol menüsü olması sağlanır.

EĞİTİM :

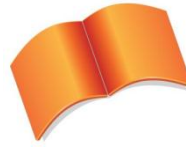
**VERİYE ERIŞİM VE
NAVİGASYON**

Bölüm :

MasterPage ve Navigasyon

Konu :

Navigasyon



Microsoft Türkiye

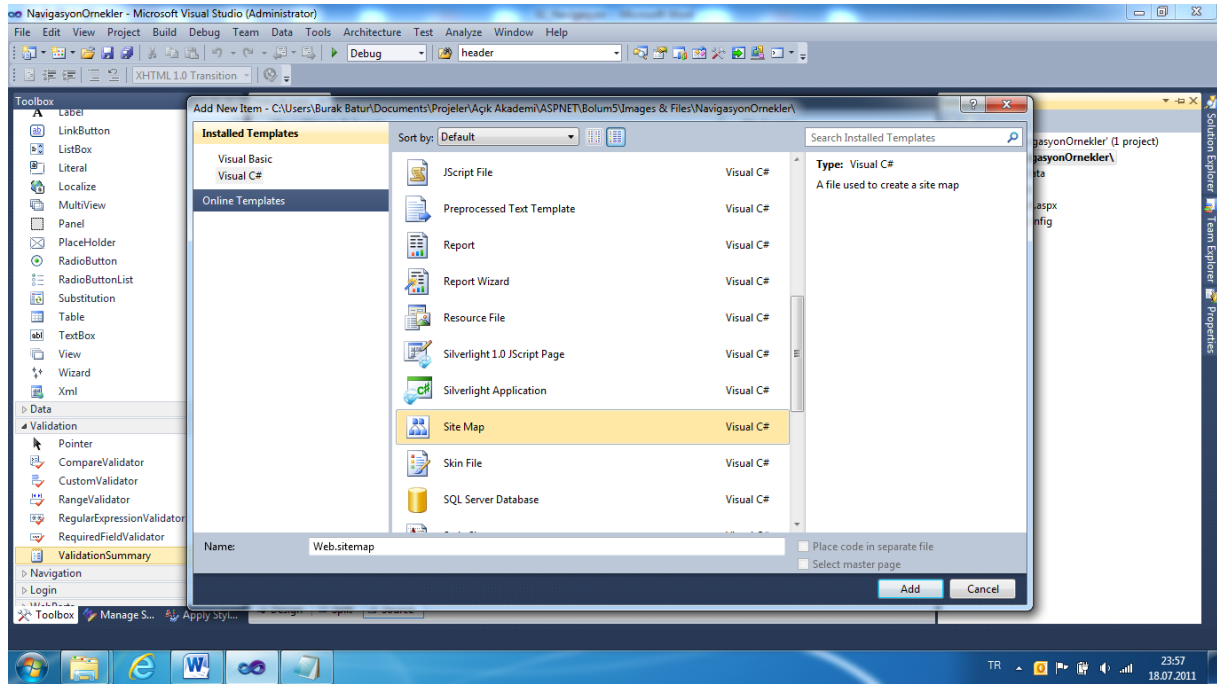
Açıık Akademi

Navigasyon

Web siteleri genellikle çok sayıda sayfadan oluşur. Bu sebeple, web sayfalarında yönlendirme oldukça önemli bir yer tutmaktadır. ASP.NET'in ilk sürümlerinde, siteye menü eklemek için ekstra yazılımlar kullanmak gerekirdi. Ancak, ASP.NET 2.0 ve sonrasında, yazılım geliştiricilerin hayatına giren yeni yönlendirme kontrolleri ile birlikte, oldukça güzel menüler ekstra bir yazılım kullanmadan kolaylıkla hazırlanabilmektedir. ASP.NET'in sağlamış olduğu yönlendirme kontrolleri ve yapılar ile birlikte siteye menü eklemekten daha ileriye gidilip sitenin hiyerarşik yapısı da XML tabanlı bir dosyada tutulabilmekte ve bu dosyada yer alan bilgiler ile yönlendirme kontrolleri dinamik olarak oluşturulabilmektedir. Yönlendirme kontrolleri ile birlikte sitenin hiyerarşik yapısının tutulabileceği **SiteMap** adı verilen dosyalar, **TreeView** ve **Menü** kontrolleri ile birlikte kullanıcının o anda nerede bulunduğunu gösteren **SiteMapPath** isimli kontrol, yazılım geliştiricilerin hayatına girmiştir.

SiteMap

XML tabanlı SiteMap dosyaları sitenin hiyerarşik yapısının tutulabileceği **.sitemap** uzantısı ile biten ASP.NET'in özel dosya tiplerinden biridir. SiteMap dosyası diğer kontroller tarafından veri kaynağı olarak kullanılarak, menü gibi kontrollerin içeriğinin dinamik olarak oluşturulmasını sağlayan dosyalardır. Visual Studio ortamında projeye bir tane SiteMap dosyası eklemek için Proje üzerinde sağ tıklanarak **"Add New Item"** seçeneği ile ASP.NET projelerinde kullanılabilecek dosya tipleri listelenir. Ardından bu ekrandan SiteMap'e tıklanarak projeye **web.sitemap** isimli bir dosya eklenir.



Web.sitemap dosyası özel bir dosyadır ve ilerleyen ekranlarda açıklanan SiteMapDataSource kontrolü, direkt olarak sitenin kök dizininde bu dosyayı arar ve bu dosyada bulunan bilgileri okur. SiteMap dosyası eklendikten sonra Visual Studio ortamında aşağıdaki görünüm ile karşılaşılacaktır. Visual Studio dosyayı oluştururken örnek olarak birkaç tane de düğüm ekler ve kolayca veri eklenmesine olanak tanır. SiteMap dosyasında mutlaka bir adet kök düğüm olmak zorundadır bu sebeple en üstte bir tane kayıt eklenmiştir ve onun da altına yeni kayıtlar eklenmiş vaziyettedir.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="" title="" description="">
```

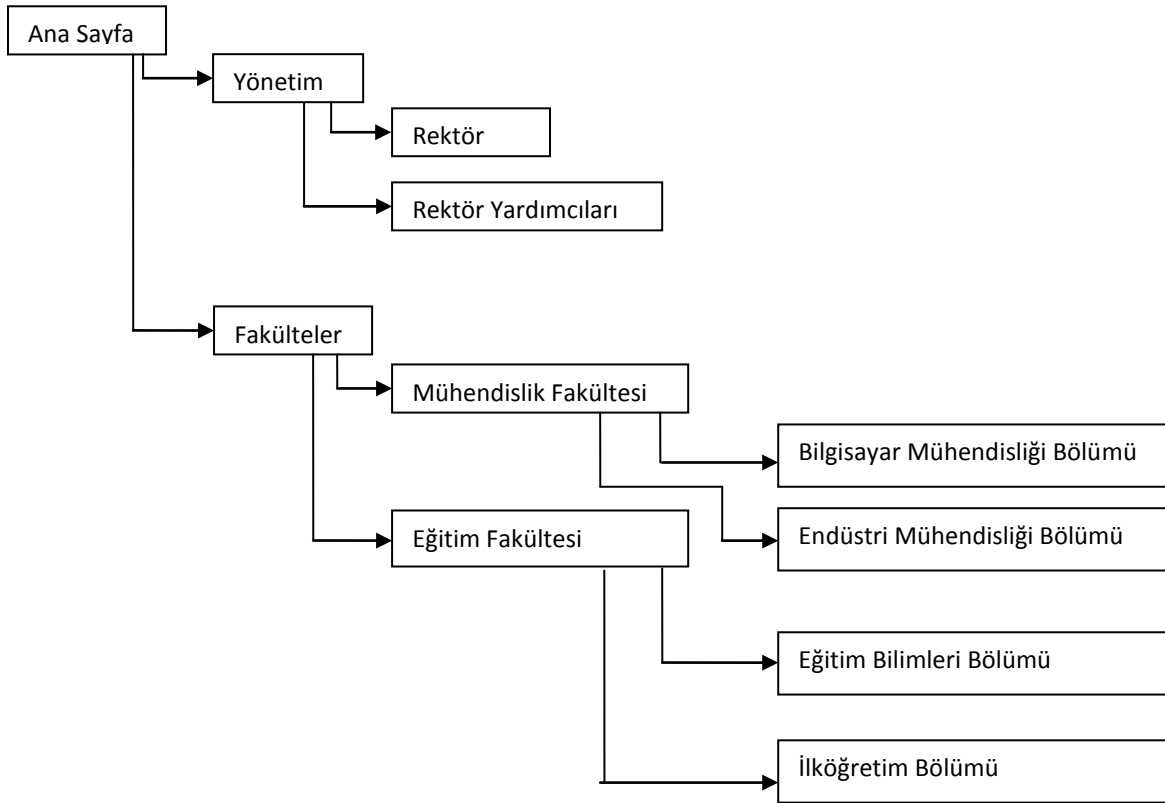
```

<siteMapNode url="" title="" description="" />
<siteMapNode url="" title="" description="" />
</siteMapNode>
</siteMap>

```

SiteMap'a bir sayfa SiteMapNode düğümü olarak eklenir. Düğüm altına alt düğümler alacaksa <SiteMapNode>.....</SiteMapNode> şeklinde eklenir ancak eklenen düğüm bir alt düğüm içermiyorsa <SiteMapNode /> şeklinde sonlandırılır ve özellikleri belirtilir. Yukarıda listelenen özelliklerden **url**, sitemap içerisinde listenecek olan sayfanın, adres bilgisini tutacak olan özelliktir. **title** sayfanın menü gibi kontrollerde görüntülenecek olan başlık bilgisini tutarken, **description** özelliği ise sayfanın açıklamasını tutuyor olacaktır.

SiteMap kullanımını bir örnek ile açıklamadan önce ilk olarak gerçek bir senaryodan yola çıkılarak bir üniversite sitesini gözümüzün önüne getirelim. Pek çok üniversite sitesinin hiyerarşik yapısı aşağıdaki şekildeki görüntülenen yapı ile benzerlik göstermektedir örnek olarak da SiteMap dosyasında aşağıdaki yapıyı hayata geçireceğiz.



Yukarıdaki yapıyı gerçeklemek için biraz önce eklediğimiz SiteMap dosyasına verileri ekleyeceğiz. Ancak öncelikle bir adet MasterPage oluşturalım ve bu MasterPage'den de her birim için bir adet sayfa türetelim. Oluşturmuş olduğumuz bu sayfaların adreslerini URL özelliğinde belirteceğiz. Sayfalar eklendikten sonra oluşturulan SiteMap dosyası aşağıda yer almaktadır. Unutmayın, ilk olarak ekranda gördüğünüz URL bölümünde belirtilen sayfaları oluşturuyoruz ve ardından SiteMap dosyamızı bu hale getiriyoruz.

```

<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="default.aspx" title="Ana Sayfa" description="Ana Sayfa">
    <siteMapNode url="yonetim.aspx" title="Yönetim" description="Yönetim
ile ilgili Sayfalar">

```

```

        <siteMapNode url="rektor.aspx" title="Rektör"
        description="Rektör" />

        <siteMapNode url="rektory.aspx" title="Rektör Yardımcıları"
        description="Rektör Yardımcıları" />

    </siteMapNode>

    <siteMapNode url="fakulte.aspx" title="Fakülteler"
    description="Fakülteler Listesi">
        <siteMapNode url="muhendislik.aspx"
        title="Mühendislik Fakültesi">
            <siteMapNode url="bilgisayar.aspx"
            title="Bilgisayar Mühendisliği" />

            <siteMapNode url="endustri.aspx"
            title="Endüstri Mühendisliği" />
        </siteMapNode>

        <siteMapNode url="egitim.aspx" title="Eğitim Fakültesi">
            <siteMapNode url="egitimbil.aspx"
            title="Eğitim Bilimleri"/>

            <siteMapNode url="ilkogretim.aspx" title="İlköğretim" />
        </siteMapNode>
    </siteMapNode>
</siteMap>

```

SiteMap Dosyalarını parçalamak

Siteler büyüdükçe SiteMap dosyaları da büyür ve içerik iyice karmaşık hale gelir bu gibi durumların önüne geçmek için SiteMap dosyaları içindeki veriler hiyerarşiye göre farklı dosyalara koyulup bu dosyalar web.sitemap içerisinde birleştirilebilir, böylece her grup veri, farklı bir fiziksel dosyada olacağı için web.sitemap dosyası içinde yaşanması olası bir karmaşıklığın önüne geçilmiş olur.

Üniversite örneği yeniden göz önüne alındığında bu durum daha iyi anlaşılabilecektir. Şu anda gerçekleştirilen SiteMap tasarımı, üniversitenin sadece çok küçük bir parçası için gerçekleştirilmiştir. Oysa ki üniversitelerde daha fazla bölüm ve birim vardır. Bu bölüm ve birimlerin hepsini aynı dosyaya koymak hem doyanın tasarımı hem de bakımı açısından problem teşkil etmektedir. Bu durumun önüne geçmek için yukarıdaki örnek yeniden ele alındığında iki ana bölüm dikkat çekiyor. Bunlar, Yönetim ve Fakülteler bölümleridir. Her iki bölüm için de ayrı ayrı birer tane SiteMap dosyası oluşturulup bu dosyalar web.sitemap içerisine eklenerek web.sitemap'in karmaşıklığı oldukça azaltılabilir. Bu işlemleri gerçeklemek için projeye **yonetim.sitemap** ve **fakulte.sitemap** adında iki adet daha dosya ekleyip içeriklerini aşağıdaki gibi dolduralım.

yonetim.sitemap

```

<?xml version="1.0" encoding="utf-8" ?>
<sitemap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="yonetim.aspx" title="Yönetim" description="Yönetim ile
    ilgili Sayfalar">
        <siteMapNode url="rektor.aspx" title="Rektör" description="Rektör" />
        <siteMapNode url="rektory.aspx" title="Rektör Yardımcıları"
        description="Rektör Yardımcıları" />
    </siteMapNode>
</sitemap>

```

fakulte.sitemap

```

<?xml version="1.0" encoding="utf-8" ?>
<sitemap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="fakulte.aspx" title="Fakülteler"
    description="Fakülteler Listesi">

```

```

<siteMapNode url="muhendislik.aspx" title="Mühendislik Fakültesi">
    <siteMapNode url="bilgisayar.aspx"
        title="Bilgisayar Mühendisliği" />
    <siteMapNode url="endustri.aspx"
        title="Endüstri Mühendisliği" />
</siteMapNode>

<siteMapNode url="egitim.aspx" title="Eğitim Fakültesi">
    <siteMapNode url="egitimbil.aspx" title="Eğitim Bilimleri"/>
    <siteMapNode url="ilkogretim.aspx" title="İlköğretim" />
</siteMapNode>
</siteMapNode>
</siteMap>

```

SiteMap dosyaları oluşturulduktan sonra sıra geldi bu dosyaları web.sitemap isimli dosyaya dahil etmeye. Bir SiteMap dosyası SiteMapNode'un **SiteMapFile** özelliği ile bir SiteMap dosyası içine dahil edilebilir. Aşağıda web.sitemap dosyasının yeni ve sade hali yer almaktadır. Bu şekilde diğer SiteMap dosyaları da kendi içlerinde hiyerarşiye göre gruplandırılarak oldukça sade ve bakımı kolay SiteMap dosyaları oluşturulabilir.

```

<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="default.aspx" title="Ana Sayfa" description="Ana Sayfa">
        <siteMapNode siteMapFile="yonetim.sitemap" />
        <siteMapNode siteMapFile="fakulte.sitemap" />
    </siteMapNode>
</siteMap>

```

SiteMapDataSource Kontrolü

SiteMapDataSouce kontrolü web.sitemap dosyasından verileri okuyup TreeView ve Menü kontrollerine taşıyacak olan veri kaynağı kontrolüdür. SiteMapDataSouce kontrolü XmlSiteMapProvider sınıfı aracılığı ile verileri kök dizinde bulunan web.sitemap isimli dosyadan okuyarak kontrollere taşır. Kontroller de SiteMapDataSource kontrolünden gelen veriler ile dinamik olarak menüler oluşturur ve kolaylıkla kullanılabilir.



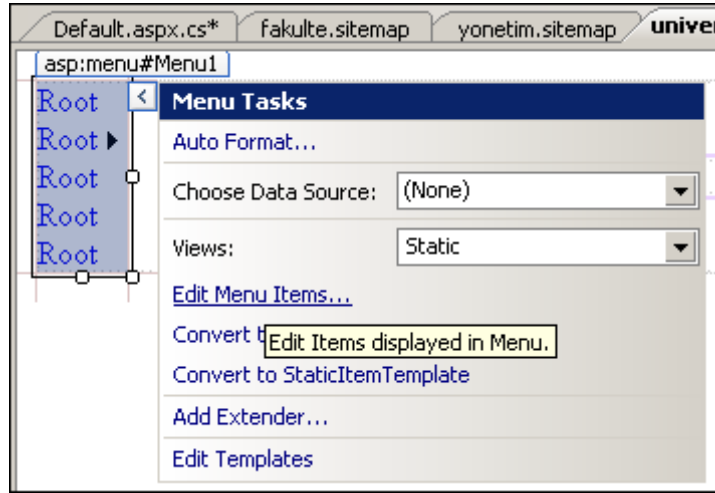
SiteMapDataSource kontrolünü sayfaya ekledikten sonra web.sitemap dosyasının yerinin gösterilmesine gerek yoktur. Çünkü; SiteMapDataSource kontrolü web.sitemap dosyasını uygulamanın kök klasöründe arayacaktır.

Navigasyon Kontrolleri

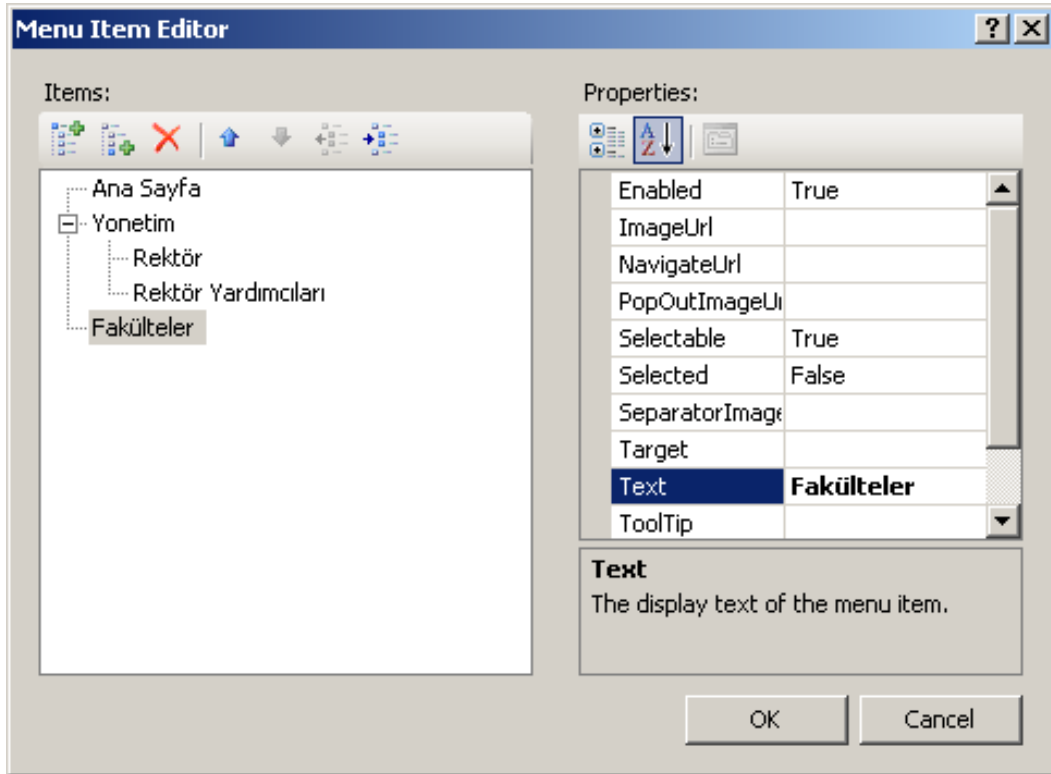
Navigasyon kontrolleri Visual Studio'nun araç kutusunda "**Navigation**" sekmesinde yer alan ve sayfalara menü özelliği ekleyebilecek ASP.NET 2.0 ile birlikte gelen kontrollerdir. Navigasyon kontrolleri SiteMapDataSource aracılığı ile SiteMap'ta bulunan verileri görüntüleyebileceği gibi, statik olarak veri eklemeye veya dinamik olarak çalışma zamanında verilerin eklenmesine de izin verir.

Menu

Menu kontrolü sayfaya eklenerek yatay ya da dikey olarak açılıp kapanan menüler oluşturulmasına olanak tanıyan yönlendirme kontrolüdür. Visual Studio ile sunulan şablonlar ile oldukça hoş görüntüler de elde edilebilen Menu kontrolü kullanılarak, başka yazılımlara ihtiyaç duymadan, JavaScript ile gerçekleştirilebilen Menüler, JavaScript bilmeye gerek kalmadan kolayca oluşturulabilir.

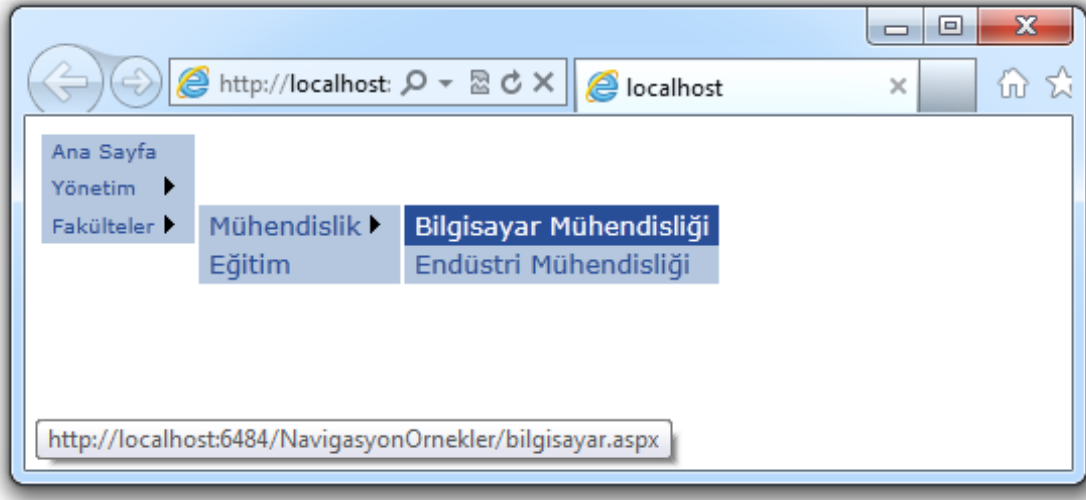


Menu kontrolüne öğe eklemek için SmartTag'indeki **"Edit Menu Items"** bağlantısı kullanılabilir. Bu bağlantı Menu kontrolünün öğelerinin düzenlenmesini sağlayacak olan dialog penceresini açacaktır. Dialog penceresi üzerinden hem parent hem de child öğeler eklenebilir.



Yukarıda görüntülenen ekran yardımı ile Menu kontrolüne öğe eklenebilir. En soldaki buton aracılığı ile parent, yani içerisine öğe alan bir öğe eklenirken, soldan ikinci buton yardımı ile de parent bir üyenin altına child bir öğe eklenebilir. Öğeler eklendikten sonra özellikleri sağdaki özellikler penceresinden kolaylıkla ayarlanabilir. Özellikler penceresi ile link verilmek istenilen sayfa **NavigateUrl** özelliği aracılığı ile projede bulunan sayfalardan herhangi birisi seçilerek belirtilebilir. **Text** özelliği menü de görüntülenecek olan özelliktir. **Target** özelliği ile linkin nerede açılacağı belirlenebilir.

Menu'nun öğeleri eklenip sunulan sitillerden Classic seçildikten sonra çalışma zamanındaki görünümü aşağıdaki gibi olacaktır. Görüldüğü üzere tek satır JavaScript kodu yazmadan, oldukça kullanışlı bir açılıp kapanan Menu oluşturulabildi.



Menu'nun iki farklı görünümü vardır bunlardan biri **Static** diğeri **Dynamic**'tir. Static görünüm adı üzerinde sayfaya ilk olarak gelindikten sonra sabit olarak kullanıcıya gösterilen görünümdür. Dynamic görünüm ise alt öğelerin listeleyecek olan kullanıcı kök elemanın üzerine geldiğinde açılan menü olarak tanımlanabilir. Kontrolün özellikler pencersinden bu iki görünüm ve genel görünüm için pek çok ayar yapılabilir bunlardan bazıları aşağıda listelenmektedir;

DynamicMenuItemStyle: Açılan menülerin stillerinin belirlendiği özellik.

DynamicMenuStyle: Dinamik olarak görüntülenecek olan menünün stil ayarlarının yapıldığı bölüm.

DynamicPopOutImageUrl: Dinamik menü içerisindeki açılan menüleri belirtmek için görüntülenecek olan resmin yolunun belirlendiği bölüm.

ItemWrap: True değeri atandığında menü içerisinde listelenen öğelerin Text özelliğinin, menünün genişliğini geçmesi durumunda, Text özelliğinin bölünebildiği yerden bölünüp kalanının alt satıra geçirilmesi sağlanmış olur.

LevelMenuItemStyles: Bu özellik ile birlikte her seviyedeki öğelere farklı farklı stil belirlemesi yapılabilir. Özelliği kullanmak için özelliğin sağ tarafındaki elips butona tıklanılarak bir diyalog penceresi açılır ve bu pencereden her seviye için stil ayarı belirlenebilir.

MaximumDynamicDisplayLevels: Dinamik olarak kaçınıcı seviyeye kadar öğe görüntüleneceği bu özellik aracılığı ile belirlenir. Öğeleri teker teker eklerken pek anlamı olmayan bu özellik, SiteMap'ten getirilen verilere göre menü oluşturulması durumunda oldukça önem kazanıyor. SiteMap dosyasında sitenin tüm hiyerarşisi bulunabilir ancak bunlardan istenilen kadarı menüde görüntülenebilir.

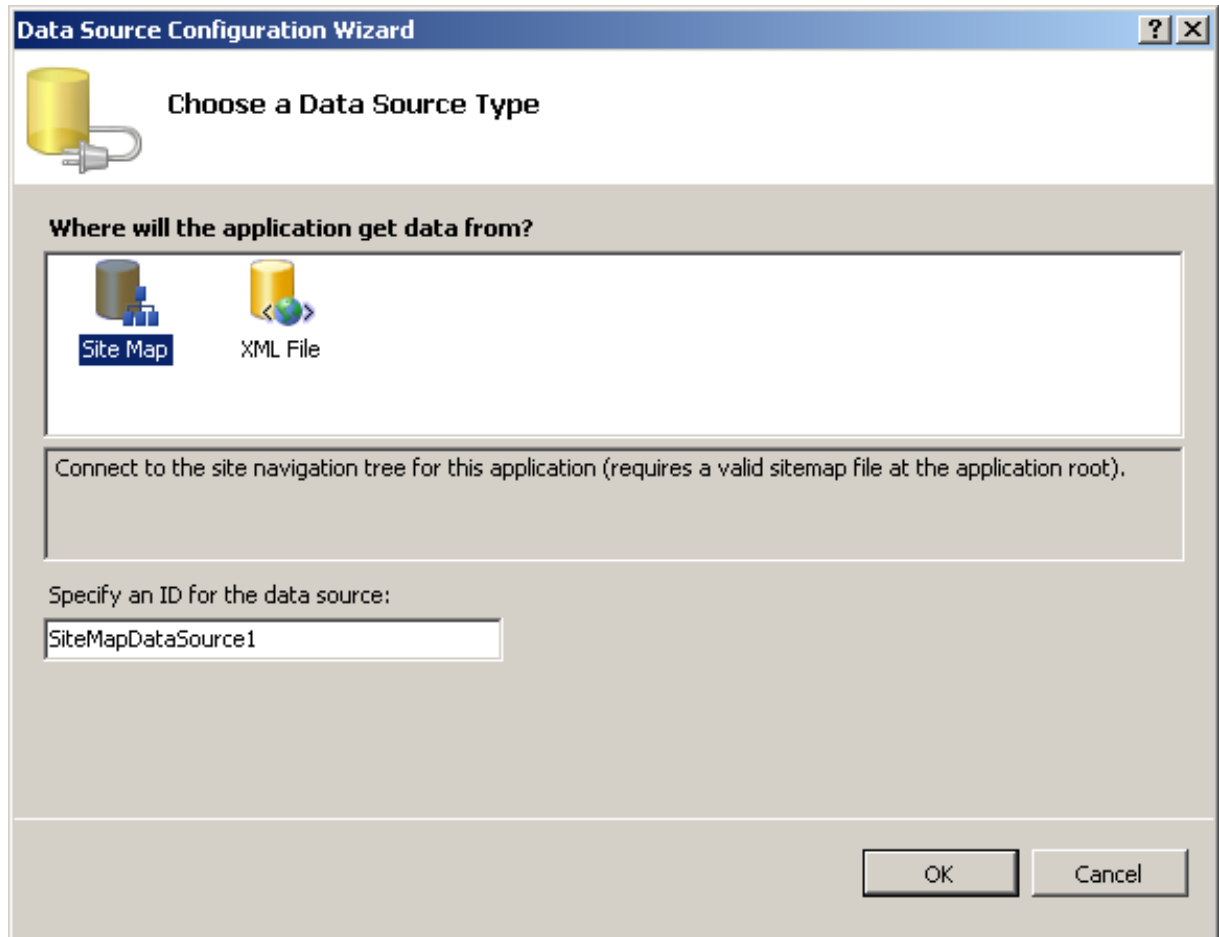
Orientation: Menünün yatay'mı (Horizontal) yok sa dikey'mi (Vertical) görüntüleneceği belirlenir.

StaticDisplayLevels: Statik olarak kaçınıcı seviyeye kadar öğe görüntüleneceği bu özellik aracılığıyla belirtilir. Bu özellikte belirtilen seviyeye kadar olan öğeler statik olarak görüntülenir ve ardından kalanlar MaximumDynamicDisplayLevels özelliğinde belirtilen seviyeye kadar dinamik olarak görüntülenir.

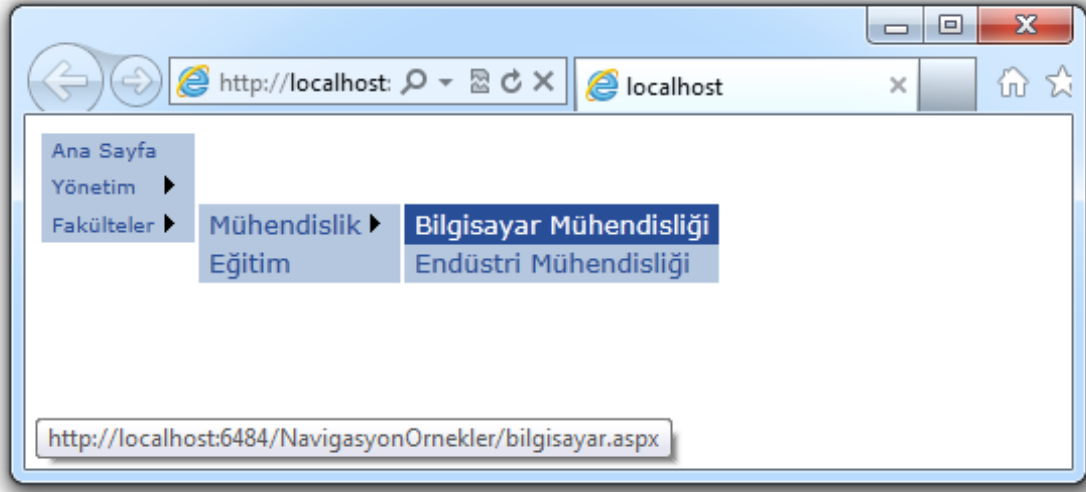
StaticPopOutImageUrl: Statik olarak görüntülenen öğelerin altında listenecek olan alt öğeler olduğunda, bu öğelerin olduğunu kullanıcıya bildirmek için gösterilecek olan resmin yolu belirtilebilir. Eğer herhangi bir resim belirtilmedi ise otomatik olarak bir ok işareti görüntüleniyor olacaktır.

SiteMap'teki Verilere Göre Menü Oluşturmak

Menü elemanları direkt belirtilebileceği gibi SiteMap dosyasındaki verilere göre de oluşturulabilir. Aslında mantıklı olan yöntem de menüyü SiteMap'te bulunan verilere göre oluşturmaktır çünkü bu alandaki veriler sitenin farklı yerlerinde de kullanılabilir ve menu tarzı bir oluşumun gerektiği bir durumda yeniden aynı işlemlerin yapılmasının önüne geçilmiş olur. Menü kontrolündeki öğeleri SiteMap'ten gelen bilgiler ile doldurmak için SiteMapDataSource kontrolüne ihtiyaç vardır. Sayfaya SiteMapDataSource eklemek için Menü'nün SmartTag'ı kullanılabilir. Menü'nün SmartTag'inde Choose Data Source bölümünden New Data Source seçeneği ile Menü kontrolüne yeni bir veri kaynağı eklenebilir. New Data Source seçeneği seçildikten sonra veri kontrollerinde olduğu gibi Menü ile kullanılacak veri kaynaklarının listelendiği bir diyalog penceresi açılacaktır. Dialog penceresi iki adet veri kaynağı listeliyor olacaktır bunlar SiteMapDataSource ve XmlDataSource kontrolleridir. Menü kontrolünün öğeleri herhangi bir Xml dosyasındaki verilerden oluşturulabileceği gibi, özel bir Xml dosyası olan SiteMap'ten de oluşturulabilir, bu bölümde amaca uygun olarak SiteMapDataSource kontrolü seçilip Ok tuşuna basılarak sayfaya bir adet SiteMapDataSource kontrolü eklenir ve Menü'ye bağlanır.



SiteMapDataSource'un menüye bağlanmasının ardından sayfa çalıştırıldığında menünün sadece ana sayfayı görüntülediği fark edilecektir. Bu durumun nedeni menünün StaticDisplayLevels özelliğinin 1 olmasıdır. StaticDisplayLevels özelliğine 2 değeri atandığında aşağıdaki görünüm elde edilebilecektir.



Kod Yazarak Menü Oluşturmak

Menüye öğe eklemenin bir diğer yolunda kod yazmaktır. Menu nesnesinin **Items** koleksiyonu kullanılarak var olan öğelere ulaşılabilir ya da yeni öğe eklenebilir. Items koleksiyonunun Add metodu MenuItem tipinde bir nesne kabul eder ve aldığı nesneyi menüye ekler. Menüye eklenen öğelerin altına, alt öğeler eklenecekse bunlar da MenuItem sınıfının **ChildItems** isimli koleksiyonu ile menüye dahil edilir. Veritabanından gelen veriler ile dinamik bir menü oluşturulmak isteniyorsa kod yazarak Menu kontrolünü oluşturmak mantıklı bir çözüm olacaktır. Aşağıda bulunan kodlarda SiteMap örneğinde hayal edilen üniversitenin hiyerarşisi Menu kontrolü için kurulmaktadır.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        //İlk olarak Yönetim bölümü oluşturuluyor
        MenuItem yonetim = new MenuItem();
        yonetim.Text = "Yönetim"; //Menu'de görüntülenecek metin
        //üzerine tıklanıldığında gidilecek olan adres
        yonetim.NavigateUrl = "yonetim.aspx";

        //Yönetim Menüsünün alt menüleri oluşturuluyor
        MenuItem rektor = new MenuItem();
        rektor.Text = "Rektör";
        rektor.NavigateUrl = "rektor.aspx";

        MenuItem rektory = new MenuItem();
        rektory.Text = "Rektör Yardımcıları";
        rektory.NavigateUrl = "rektory.aspx";

        /*Oluşturulan MenuItem nesneleri yonetim'in
        ChildItems koleksiyonuna ekleniyor*/

        yonetim.ChildItems.Add(rektor);
        yonetim.ChildItems.Add(rektory);

        //yonetim nesnesi Menu'ye ekleniyor
        Menu1.Items.Add(yonetim);

        //Fakülteler oluşturuluyor
        MenuItem fakulteler = new MenuItem();
        fakulteler.Text = "Fakülteler";
        fakulteler.NavigateUrl = "fakulte.aspx";

        MenuItem muhendeslik = new MenuItem();
```

```

muhendeslik.Text = "Mühendislik";
muhendeslik.NavigateUrl = "muhendislik.aspx";

MenuItem egitim = new MenuItem();
egitim.Text = "Eğitim Fakültesi";
egitim.NavigateUrl = "egitim.aspx";

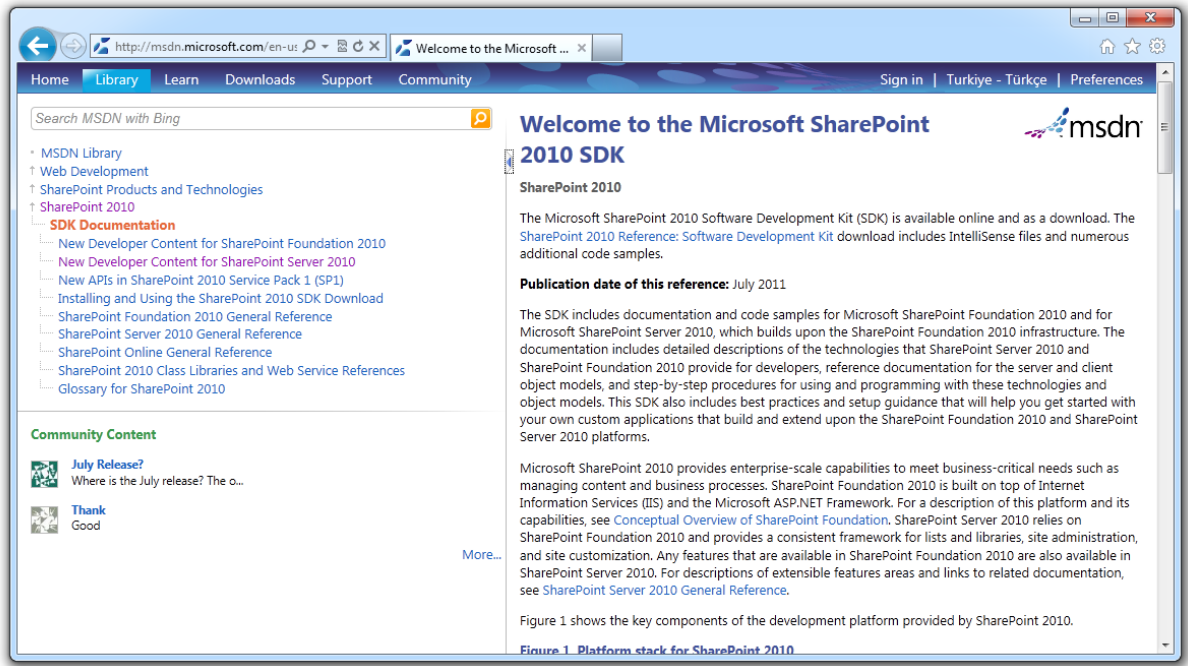
fakulteler.ChildItems.Add(muhendeslik);
fakulteler.ChildItems.Add(egitim);

Menu1.Items.Add(fakulteler);
}

```

TreeView

TreeView kontrolü ağaç şekline menüler oluşturulmasına olanak tanıyan yapılardır. Site haritası gibi haritalar yapmak için kullanılabilecek TreeView kontrolü de tek satır JavaScript kodu yazmadan oluşturulabilmektedir. TreeView kontrolüne gerçek hayattan örnek vermek gerekirse en somut örnek MSDN olarak gösterilebilir. Microsoft yıllardır MSDN'de TreeView tarzı bir kontrol kullanmaktadır. Aşağıda MSDN'in ekran görüntüsünü bulabilirsiniz.



TreeView kontrolüne de tıpkı Menu kontrolünde olduğu gibi öğe eklenebilir. Ancak burada Menu'den farklı olarak Items yerine Nodes koleksiyonu vardır. TreeView'e yeni düğüm eklemek için SmartTag'den **Edit Nodes** seçilebilir ya da özellikler penceresinden **Nodes** özelliği kullanılabilir. Düğüm eklerken gidilecek yol, Menu'de olduğu gibidir. Yani, parent ve child öğeler burada da belirtiliyor ve Nodes koleksiyonuna ekleniyor.

TreeView'e kod yazarak da düğüm eklenebilir. Mantık olarak, Menu ile aynı olmasına rağmen burada biraz önce de açıklandığı gibi **Nodes** ve **ChildNodes** koleksiyonları söz konusudur. Bu koleksiyonlar içlerine **TreeNode** tipinde bir nesne kabul ederler. Aşağıdaki örnek kısaca TreeView'e kod yazarak düğüm eklemeyi örneklemektedir. Menu örneğinin daha kısası olan bu örnek ile sadece yönetim bölümü ve fakülteler düğümü eklenmektedir.

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)

```

```

{
    //İlk olarak Yönetim bölümü oluşturuluyor
    TreeNode yönetim = new TreeNode();
    yönetim.Text = "Yönetim";
    yönetim.NavigateUrl = "yonetim.aspx";

    //Yönetimin alt düğümleri oluşturuluyor
    TreeNode rektor = new TreeNode();
    rektor.Text = "Rektör";
    rektor.NavigateUrl = "rektor.aspx";

    TreeNode rektory = new TreeNode();
    rektory.Text = "Rektör Yardımcıları";
    rektory.NavigateUrl = "rektory.aspx";

    //Yönetim bölümünün alt düğümleri kendisine ekleniyor
    yönetim.ChildNodes.Add(rektor);
    yönetim.ChildNodes.Add(rektory);

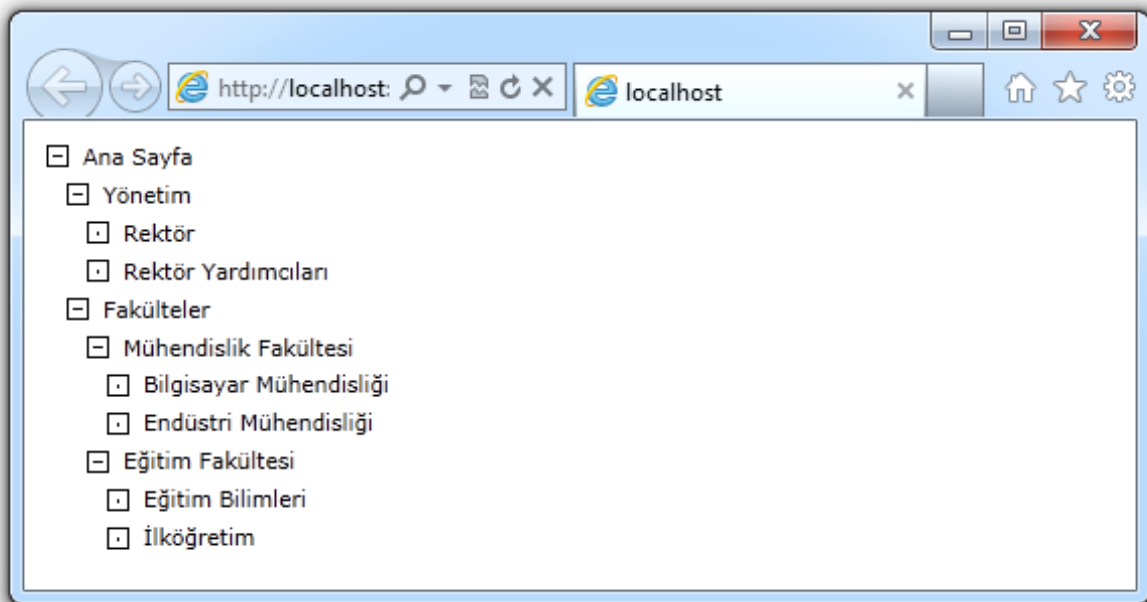
    //Yönetim bölümü TreeView'e ekleniyor
    TreeView1.Nodes.Add(yonetim);

    //Fakülteler alt düğümü oluşturuluyor
    TreeNode fakulteler = new TreeNode();
    fakulteler.Text = "Fakülteler";
    fakulteler.NavigateUrl = "fakulteler.aspx";

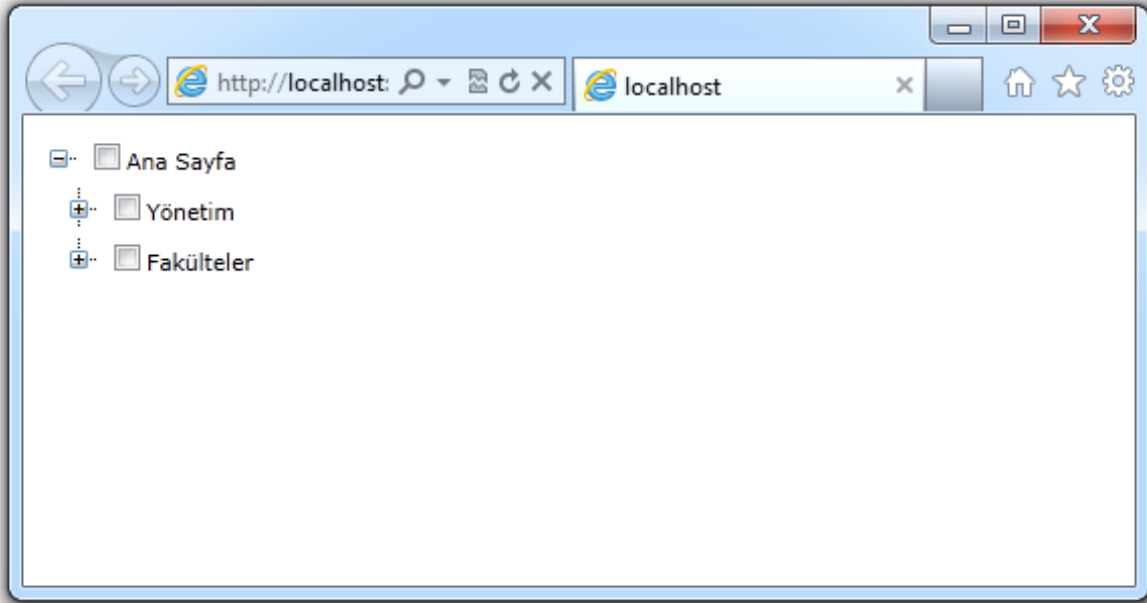
    //Fakülteler TreeView'e ekleniyor
    TreeView1.Nodes.Add(fakulteler);
}
}

```

TreeView'e de tıpkı Menu'de olduğu gibi SiteMap'ten veri getirebilir. Bu işlem için de Menu'de olduğu gibi SmartTag'den Choose Data Source bölünden yeni bir veri SiteMapDataSource eklemek ve bunu TreeView'e bağlamak yeterli olacaktır. Aşağıdaki resimde SiteMap'ten gelen veriler doğrultusunda oluşturulan bir TreeView yer almaktadır. SiteMap'ın içeriği de hatırlanacağı üzere SiteMap'ın anlatıldığı bölümde doldurulmuştu. TreeView, içerisinde pek çok kontrolle birlikte sunulduğu gibi, hazır stiller de sunulmaktadır. Bunlardan biri de MSDN'dir. Aşağıdaki resimde görüntülenen TreeView'e MSND stili uygulanmıştır.

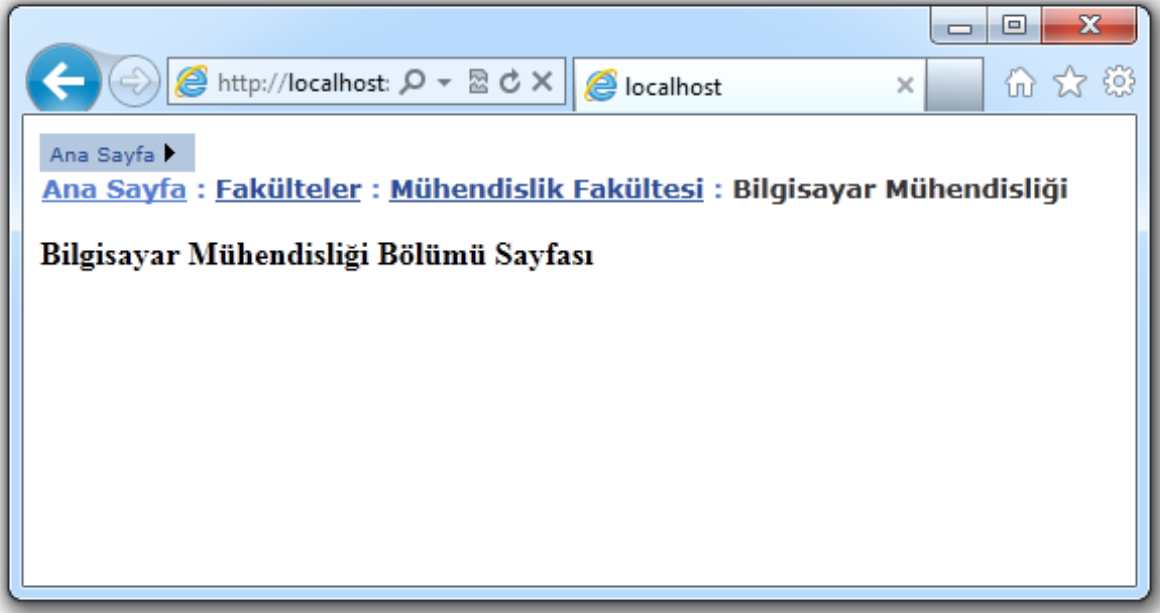


TreeView kontrolünde de hayatı kolaylaştıran pek çok özellik vardır, bu özellikler yardımı ile birlikte düşünülen pek çok işlem kolaylıkla yapılabilir. TreeView içeren bir sayfa talep edildiğinde varsayılan olarak tüm düğümler açılır. Ancak, bazen böyle bir durum istenmeyebilir. Örneğin sadece kök düğümlerin görüntülenmesi gibi bir durum söz konusu olabilir. Bu işlem için **ExpandDepth** özelliği kullanılır. Varsayılan olarak **FullExpand** değerinde olan bu özellik ile TreeView'in düğümlerinin ilk yüklendiğinde kaçınıcı seviyeye kadar açılacağı belirtilmiş oluyor. Bu özellik yukarıdaki örnek için 1 yapıldığında TreeView'in sadece kök seviyesi görüntüleniyor olacaktır. Bir diğer özellik de düğümler arasında kaybolmayı engelleyecek olan **ShowLines** özelliğidir. Varsayılan değeri False olan ShowLines özelliği, True olarak ayarlandığında düğümler arasında çizgiler görüntülenir. TreeView'in düğümlerinin yanında CheckBox görüntülenmesi isteniyorsa **ShowCheckBoxes** özelliği All olarak ayarlanabilir. ShowCheckBoxes özelliği ile sadece kök düğümlerde veya sadece alt düğümlerde CheckBox görünmesi de sağlanabilir. Bu paragrafta anlatılan özellikler çerçevesinde gerekli ayarlamalar yapıldıktan sonra TreeView'in yeni görünümü aşağıdaki gibi olacaktır.



SiteMapPath

SiteMapPath kontrolü, kullanıcı sitede dolaşırken, kullanıcıya hiyerarşik olarak nerede olduğunu gösteren ve daha üst seviyelere geri dönmeyi sağlayacak olan kontroldür. SiteMapPath kontrolü sayfaya sürüklenip bırakıldıktan sonra, direkt olarak SiteMap içerisindeki bilgileri okur ve kullanıcıya buradaki bilgiler doğrultusunda bilgiler sunar. Aşağıda, daha önce hazırlanan SiteMap'teki bilgilere göre oluşturulan SiteMapPath kontrolünün, çalışma zamanındaki görüntüsünü görüyorsunuz. Tahmin edeceğin gibi Menü ve SiteMapPath kontrolleri MasterPage üzerinde yer alıyor.



Bu ders notu, **Açık Akademi** projesi çerçevesinde **TCM** tarafından **Microsoft Türkiye** için hazırlanmıştır.
Tüm hakları **Microsoft Türkiye**'ye aittir. İzinsiz çoğaltılamaz, para ile satılamaz.